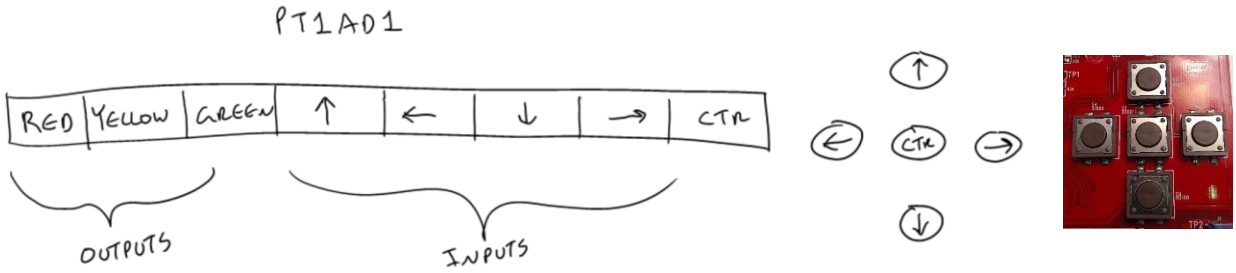
CMPE1250 – LED and Switch Library

It is time to start building up a library to house the useful and reusable code you are creating for the LEDs. You will additionally begin to operate the switches in this assignment as well.

Any code you write that may be reused in a subsequent activity should be placed in a library. This ensures that only one copy of the code exists, and simple reuse is possible.

Before you begin this exercise, make sure you attend the lecture on creating compilation units and/or watch the supporting video on Moodle. Templates for the necessary files are available on Moodle.

You have already seen that your indicator LEDs may be turned on or off by setting or clearing the corresponding bits in **PT1AD1**. Recall from the previous notes that the switches occupy the bottom five bits of **PT1AD1** and are configured by the port initialization code to be inputs:



To turn on a LED, you set the bit that corresponds to that LED in **PT1AD1**.

To read a switch, you read the bit that corresponds to that switch in **PT1AD1**.

For example, if you want to know if the down switch *is currently being pushed*, you would check bit two in **PT1AD1** to see if it is a **1**. If so, the switch is currently being pushed, if not, it is not:

```
// assume port already initialized
if (PT1AD1 & 0b00000100)
{
  // down switch is being pushed
}
```

Using a suitable mask and the bitwise **AND** operator, you may check for any individual, or combination of switch states. Keep in mind that the state of the switch is instantaneously checked at that line of code, and only reflects the state of the switch at that moment in time.

A sample header for your switch/LED library may be found on Moodle. You may model your own library after this one or go your own way. Throughout the rest of the course, the notes, demos, and other course materials will use the names found in the sample library.

Your instructor will discuss the contents of this header and will explain the usefulness of enumerations when used to self-document code, and facilitate operator choices.

You will need to complete your implementation file to provide the actual behaviors of these functions. This is not provided, but a generic template is available on Moodle.

To complete this assignment, you will need to have a fully formed and documented compilation unit (library) for the switches and LEDs, and suitable test code to ensure that all functions are exercised, with all pattern of arguments. This is important, as the check-off code may not be comprehensive in this regard.

Application Notes:

Because you are always reading from the port to detect button states, you will invariably use the bitwise AND operation for this task, as shown above. Any button that is pressed will return a 1 in the corresponding mask position, if found in the mask. A 1, regardless of position, will be considered true in an expression that tests for true or false in C.

Turning a LED on requires that you set the corresponding bit in the port register. This will typically be done with a bitwise OR operation:

```
// turn RED red LED on
PT1AD1 |= 0b10000000;
```

To turn off a LED, the corresponding bit in the port register must be cleared. This will typically be done with a bitwise AND operation:

```
// turn RED red LED off
PT1AD1 &= 0b01111111;

// or, easier to read?
PT1AD1 &= ~0b10000000;
```

Toggling a LED is typically achieved with the bitwise XOR operator. This is a neat trick, as conceptually the LEDs are write-only devices. The port is buffered, so you can read the state of the port. The last value written may be read, even if the pin direction is set to output.

Keep in mind that masks do not need to be literal values, and may be represented with variables, including enumerations.

You may achieve check-off for this assignment by completing any one of the following difficulty tiers:

Tier 1

Create a program that includes your new switch/LED library.

In the main program loop, check the left switch, and if it is pushed, turn on the red LED. If it is not pushed, turn off the red LED. Test your code. When run, the red LED should be on while the left switch is being pushed, and off when not.

Add the necessary code to have the center and right switches operate the same way on the yellow and green LEDs respectively.

Tier 2

Similar to Tier 1, but pressing L, C, or R will turn on the corresponding LED. The LED(s) will stay on until the top or bottom switch is pressed, turning off all LEDs.

Tier 3

Similar to Tier 2, but you will only permit a maximum of two LEDs to be on at any time. *Any* combination of two LEDs is permitted.

If you can, you should modify the code so that pressing a switch *toggles* the associated LED, as long as you never (even for a fraction of a second) have more than two LEDs on at a time. Getting the LEDs to not blink at very high speed is the problem here. You may still use the same method as tier 2 for setting/clearing the LEDs if that is easier, with the added restriction of a two LED maximum.

Note: Don't write code that arbitrarily clears then sets LED values on a condition. This can lead to flickering of the LED that won't be perceptible to a human, but could be disastrous to other electronic devices in other applications. Your code should be structured so that definitive assignment is performed, without redundant or arbitrary assignment.

Don't forget to call the initialization function in the 'one-time inits' section of your program, or the switches and LEDs won't function!