Polling a Switch

Overview:

In this lab, you will use GPIO signals for both inputs and outputs. As in the previous lab, you will use output signals to control the RGB LED and the red LED on P1.0. In addition, you will use polling to monitor an input signal from a pushbutton switch in order to control the LEDs based on user input.

Part 1: Simple Traffic Light Controller

In this part, you will write a program based on the flowchart shown in Figure 1 to simulate a very simple traffic light controller that responds to a pedestrian pushbutton.

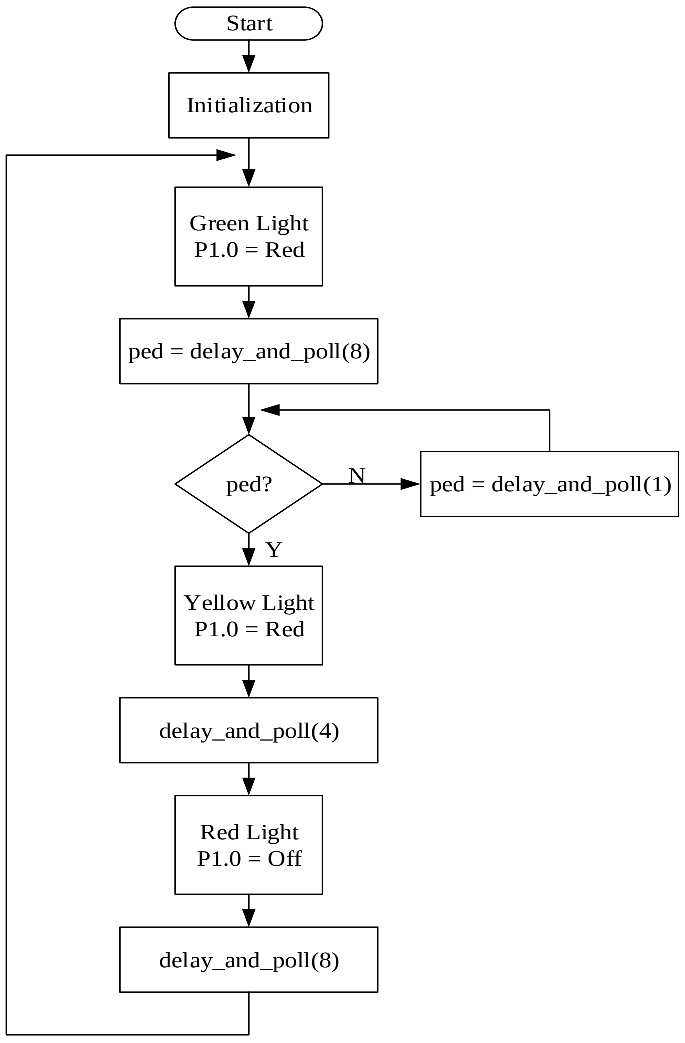
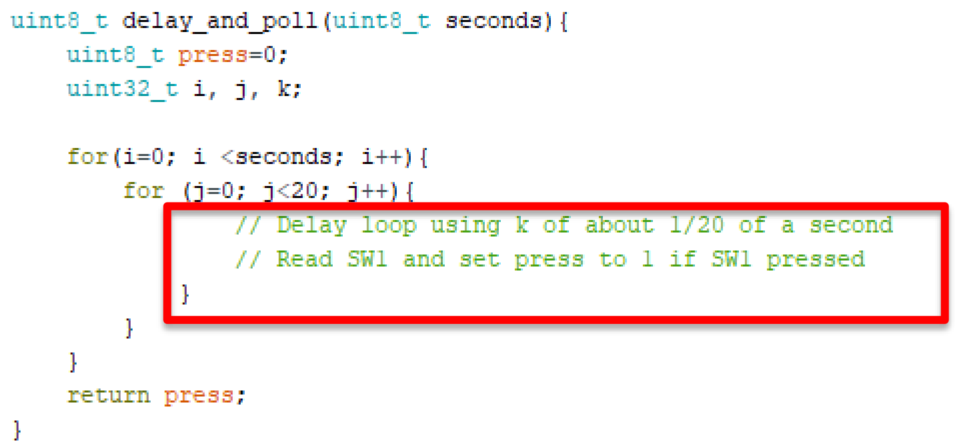


Figure 1: Traffic Light Controller Flowchart

Design Guidelines

The initialization code should be very straightforward. As usual, you will need to stop the watchdog timer and configure the input and output pins, as needed. You will also need a variable, *ped*, to hold the return value from the delay_and_poll() function, which is described below.

The key function in the traffic light controller program is the function delay_and_poll(seconds). The basic outline of this function is given below in Example 2. This function delays for a certain number of seconds given by the input parameter *seconds* while checking the SW1 button about 20 times per second to see if it has been pressed. If SW1 is pressed, a flag variable, *press*, should be set to 1. The function returns the *press* variable. Thus, if SW1 is pressed at any time during the delay_and_poll() function, it should return a 1; otherwise, the function will return a 0. (Note that the delay timing should be accurate to within about 20%. In other words, it doesn't need to be precise!)



```
uint8_t delay_and_poll(uint8_t seconds){
    uint8_t press=0;
    uint32_t i, j, k;

    for(i=0; i <seconds; i++){
        for (j=0; j<20; j++){
                // Delay loop using k of about 1/20 of a second
                // Read SW1 and set press to 1 if SW1 pressed
        }
        }
    }
    return press;
}
```

Example 1: Basic outline for delay_and_poll function

Notice in the flowhart that sometimes the program ignores the return value from the delay_and_poll() function. For this program, we only care about the pedestrian button when the Green light is on for cars. For simplicity, we ignore if the pedestrian button is pressed when the Yellow or Red light is on for cars.

The Red LED (P1.0) is for the pedestrian. When the RGB LED is Red, indicating that cars must stop, the P1.0 LED will go off, indicating that the pedestrian may cross the street. At other times, the P1.0 LED will be Red, indicating that the pedestrian may not cross.

If you want, you can add print statements to show how your traffic light controller is working. For example, you could output phrases such as "Green light", "Yellow light", "Red light", "Walk", "Don't walk", etc.

You could use your RGB_output() function to control the RGB LED, if you want. For example, you could turn on the Green, Yellow and Red lights with calls to RGB_output(2), RGB_output(3), RGB_output(1), respectively.

Part 2: An Implementation of Morse Code

In this part of the lab, you will implement a simplified Morse code translator. In order to prevent the design from being too cumbersome, you will only need to design the input loop to check for numbers 0-9.

Design Guidelines

The code should be able to detect the duration that SW1 is pressed and discern whether the button has been pressed longer than some threshold. It is up to you to fine tune the value of this input threshold to a reasonable value. Button inputs that are longer than the defined threshold constitute a dash, while shorter button presses constitute a dot.

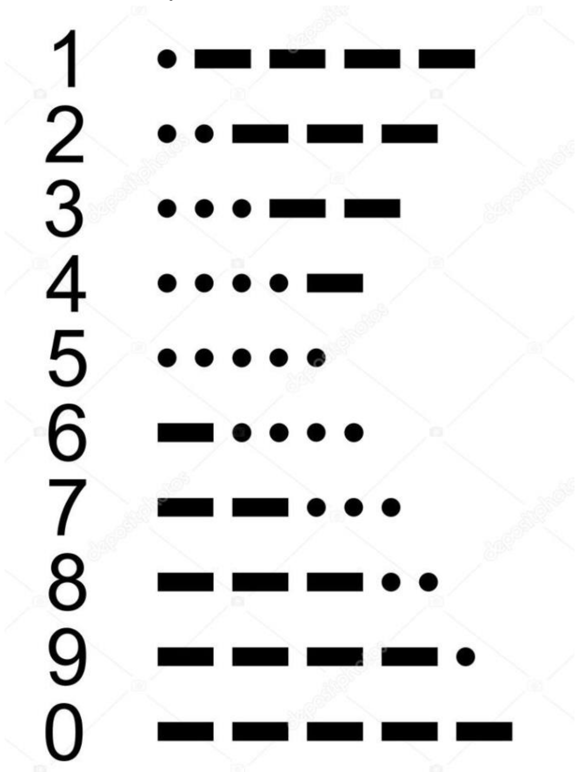


Figure 2: Morse Code Number Representation

Notice that in figure 2 each valid input is exactly five button inputs. Consequently, your code should have a counter variable used to indicate that a valid number has been passed by the user. Additionally, you should create a function to compare the user input to the actual representations of Morse code. This will help in making the code more concise.

Finally, once a valid input is passed, you must print the result to the serial terminal. If the user provides an invalid input, make sure to output some error notifying the user that the input was invalid. A useful hit is to also print whether the code detects a dot or dash upon each button input. This is useful for fine tuning the detection threshold.

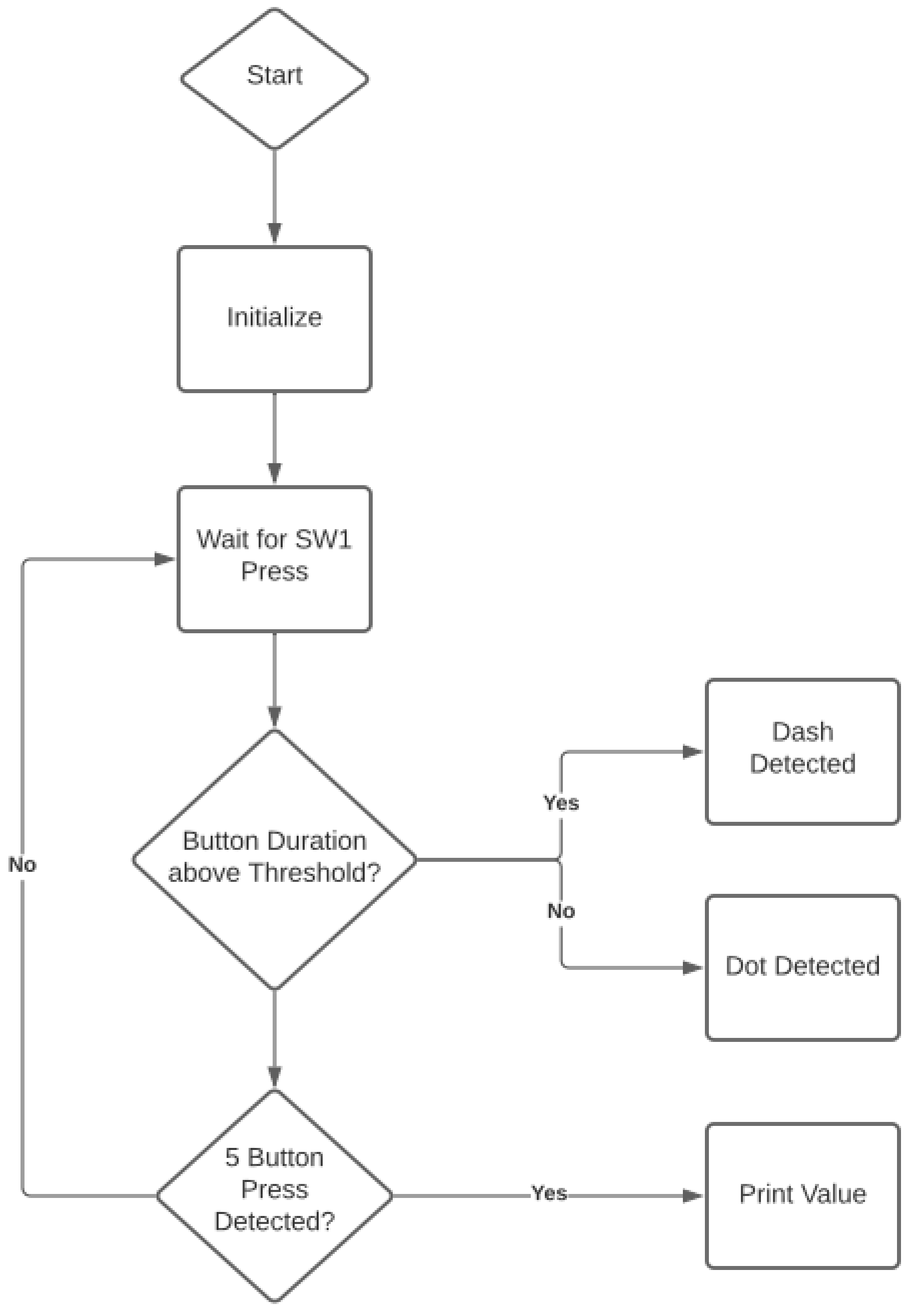


Figure 3: Morse Code Detection Flow Chart