EEL 4742C: Embedded Systems

Christopher Badolato
9/23/2019
EEL4742-0011

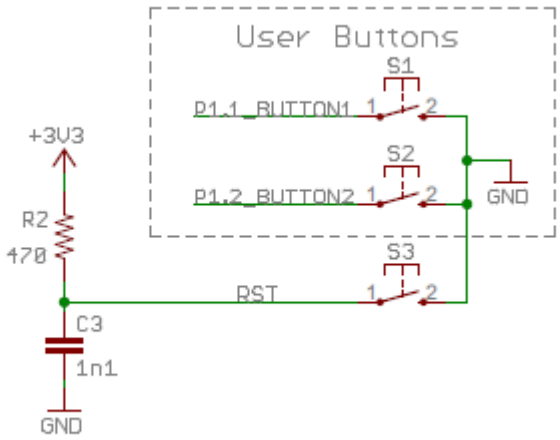
Introduction

In this lab we will set up the control of the s1 and s2 buttons located and Port1.1 and Port1.2 respectively. First, we set up button s1 to turn on the red LED located at Port 1.0. We then, set up control of button s2 to control the green LED located at Port 9.7. Next we want to only be able to turn on one LED at a time (the one that is held down first), even if both buttons are pressed. And finally, I created a custom version of the previous that will toggle the LED that corresponds to the button that is currently pressed.

Part 2.1: ...

My approach to this part of the lab was to follow the code given to set control of the LED located at P1.0 to the button S1 located at P1.1.
We first direct the P1.1 BIT (corresponds to button s1) to input (BIT1). We then enable the built in resistor which will allow us to read the volage of the circuit, we then must configure the button P1OUT as pull-up (1) or pull-down (0). We set the resistor to pull-up so that the port P1IN at the corresponding button bit, in this case BIT1 (Port 1.1) will read 1 when NOT active.

Once these values are set we can use the P1IN at the corresponding bit to the button to check if the button is currently being pressed, if so the LED bit (P1IN, BIT0) will be turn on. If P1IN BIT1 is 0, the button is being pressed (active low), otherwise the bit is high and will read 1.



Explain why the buttons are active low and why the pull-up resistor is used.

The buttons are active low because when are pressed the Vcc of the circuit goes to GND and are read as zero.
The pull-up resistor is used to configure either pull-up to Vcc or pull-down to ground. We pull-up so that when the button is pressed it will read 0 but, initially (or on release) will be read as 1.

```
//Christopher Badolato
//9/16/2019
//LAB 2.1
//EEL 4742L-0011

// Turning on the red LED while button S1 is pushed
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7
#define BUT1 BIT1 // Button S1 at P1.1

void main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins

        // Configure and initialize LED
        // Direct pin as output, confirm the LED is off
    P1DIR |= redLED;
    P1OUT &= ~redLED;
        // Configure buttons
    P1DIR &= ~BUT1; // Direct pin as input
    P1REN |= BIT1; // Enable built-in resistor
    P1OUT |= BIT1; // Set resistor as pull-up


        // Polling the button in an infinite loop

    for(;;) {
            //if s1 is pressed P1.1
            // Turn red LED on
            // Otherwise turn red LED off
        if ((P1IN & BIT1) == 0)
            P1OUT |= redLED;
        else P1OUT &= ~redLED;
    }
}
```

Part 2.2: ...

My approach to this part of the lab is similar to part 1 above, but we configure button s2 to the LED located at P1.2 to the LED located at P9.7

We will need to enable the resistor bit, set the resistor to pull-up, then check if the P1IN at BIT2 is equal to 0. If it is then we will turn on the LED, otherwise the LED will be shutoff.

```
//Christopher Badolato
//9/16/2019
//LAB 2.2
//EEL 4742L-0011
    // Turning on the red LED while button S1 is pushed
    //Turning on green LED while s2 button is pushed as well.
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7
#define BUT1 BIT1 // Button S1 at P1.1

void main(void) {
    WDTCTL = WDTPW | WDTHOLD;
    PM5CTL0 &= ~LOCKLPM5;
        // Configure and initialize LEDs
    P1DIR |= redLED; // Direct pin as output
    P9DIR |= greenLED; // Direct pin as output

    P1OUT &= ~redLED; // Turn LED Off
    P9OUT &= ~greenLED; // Turn LED Off

        // Configure buttons s1
    P1DIR &= ~BUT1; // Direct pin as input
    P1REN |= BIT1; // Enable built-in resistor
    P1OUT |= BIT1; // Set resistor as pull-up

        //Config for button s2
    P1DIR &= ~BIT2; // Direct pin as input
    P1REN |= BIT2; // Enable built-in resistor
    P1OUT |= BIT2; // Set resistor as pull-up

        // Polling the button in an infinite loop
    for(;;) {
            //configure button s1 to RED LED
            // Turn red LED on if pressed
            // otherwise Turn red LED off
        if ((P1IN & BIT1) == 0)
            P1OUT |= redLED;
        else P1OUT &= ~redLED;
            //configure button s2 to green LED if pressed
            //Turn green LED on
            //otherwise Turn green LED off
        if ((P1IN & BIT2) == 0)
            P9OUT |= greenLED;
        else P9OUT &= ~greenLED;
    }
}
```

Part 2.3: ...

My approach to this part of the lab was to instead of checking IF the button value is set to 0, we use while loops to continously check if the bits are 0.

While button s1 (P1 BIT1) is pressed (equals 0) turn on the red LED but, make sure that the green LED is shut off, that way even if button s2 is pressed the green LED will remain off.

We will as well do, while button s2 is pressed (P1 BIT2) turn on the green LED but shut off red LED. This will ensure that whichever button is pressed first, will only control the LED it corresponds with, and if the other button is pressed only the first pressed buttons corresponding LED will remain lit.

Stress test the code; what happens if both buttons are pushed simultaneously?
When both buttons are pressed simultaneously the one that is pressed first will turn on only.

```
//Christopher Badolato
//9/16/2019
//LAB 2.3
//EEL 4742L-0011

// Turning on the red LED while button S1 is pushed
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7
#define BUT1 BIT1 // Button S1 at P1.1
#define BUT2 BIT2 // Button S2 at P1.2

void main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins

        // Configure and initialize LEDs
    P1DIR |= redLED; // Direct pin as output
    P9DIR |= greenLED; // Direct pin as output
    P1OUT &= ~redLED; // Turn LED Off
    P9OUT &= ~greenLED; // Turn LED Off

        // Configure button s1
    P1DIR &= ~BUT1; // Direct pin as input
    P1REN |= BIT1; // Enable built-in resistor
    P1OUT |= BIT1; // Set resistor as pull-up

        // Configure button s2
    P1DIR &= ~BIT2; // Direct pin as input
    P1REN |= BIT2; // Enable built-in resistor
    P1OUT |= BIT2; // Set resistor as pull-up

        // Polling the button in an infinite loop
    for(;;) {
            //While our s1 button is pushed down
```

```
            // Turn red LED on
            //turn off green;
            //turn off red on release
        while((P1IN & BIT1) == 0){
                P1OUT |= redLED;
                P9OUT &= ~greenLED;
                P1OUT &= ~redLED;
        }
            //While our s2 button is pushed
            //Turn green LED on
            //turn off red
            //turn off green on release
        while((P1IN & BIT2) == 0){
            P9OUT |= greenLED;
            P1OUT &= ~redLED;
            P9OUT &= ~greenLED;
        }
    }
}
```

Part 2.4: ...

In the part, I kept it like the code above, as well as combining the if statements of the code for part 2.2. Within our while loops I check to see if the other button is pressed as well, if it is, the current LED that is lit will toggle with a slight delay.

While button s1 is press the red LED will turn on, if button s2 is pressed the red LED will toggle until s2 is released. If S1 remains pressed the red LED will no longer toggle if s2 is released.

While button s2 is pressed the green LED will turn on, if button s1 is pressed the green LED will toggle until s1 is released. If s2 remains pressed the green LED will no longer toggle if s1 is released.

```
//Christopher Badolato
//9/16/2019
//LAB 2.4
//EEL 4742L-0011


#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7
#define BUT1 BIT1 // Button S1 at P1.1
#define BUT2 BIT2 // Button S2 at P1.2

void main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins
```

```
    volatile int i, j;

        // Configure and initialize LEDs
    P1DIR |= redLED; // Direct pin as output
    P9DIR |= greenLED; // Direct pin as output
    P1OUT &= ~redLED; // Turn LED Off
    P9OUT &= ~greenLED; // Turn LED Off

        // Configure button s1
    P1DIR &= ~BUT1; // Direct pin as input
    P1REN |= BIT1; // Enable built-in resistor
    P1OUT |= BIT1; // Set resistor as pull-up

        // Configure button s2
    P1DIR &= ~BIT2; // Direct pin as input
    P1REN |= BIT2; // Enable built-in resistor
    P1OUT |= BIT2; // Set resistor as pull-up

        // Polling the button in an infinite loop
    for(;;) {
            //While our s1 button is pushed down
            //Turn red LED on
            //If we then click button s2.
            //we enter a for loop that will toggle the same (red) LED on off on (0
1 0)
            //The loop inside will ensure appropriate delay
        while((P1IN & BIT1) == 0){
                P1OUT |= redLED;
                if((P1IN & BIT2) == 0){
                    for(i = 0; i < 2; i++){
                        for(j = 0; j< 10000; j++){}
                        P1OUT ^= redLED;
                    }
                }
                    //turn off green (Make sure it is never on if s1 is pushed)
                    //turn off red on release
                P9OUT &= ~greenLED;
                P1OUT &= ~redLED;
        }
            //While our s2 button is pushed down
            //Turn green LED on
            //If we then click button s1.
            //we enter a for loop that will toggle the same (green) LED on off on
(0 1 0)
            //The loop inside will ensure appropriate delay
        while((P1IN & BIT2) == 0){
            P9OUT |= greenLED;
            if((P1IN & BIT1) == 0){
                for(i = 0; i < 2; i++){
                    for(j = 0; j< 10000; j++){}
                    P9OUT ^= greenLED;
                }
            }
                //turn off red (Make sure it is never on if s2 is pushed)
                //turn off green on release
```

```
            P1OUT &= ~redLED;
            P9OUT &= ~greenLED;
        }
    }
}
```

Student Q&A

1. When a pin is configured as input, P1IN is used for data, which leaves P1OUT available for other use? In such case, what is P1OUT used for?

Since we are using P1IN for data, P1OUT can be used to set the resistor to pull-down (0) or pull-up (1). This will pull-up to Vcc or pull-down to ground. In this case we use pull-up to Vcc so that the resistors will read high on release.

2. A programmer wrote this line of code to check if bit 3 is equal to 1: if((Data & BIT3)==1). Explain why this if-statement is incorrect.

Because we are checking BIT3 this is the value 0000 1000, this is equal to 8 in decimal, but we are seeing If the bit equals 1 which it never will. We must use if((Data & BIT3) == BIT3) to check if the bit is turned on.

3. Comment on the codes' power-efficiency if the device is battery operated. Is reading the button via polling power efficient?

No reading the button via poll is not power efficient, we are running constantly, which eats a lot of CPU cycles unnecessarily.

Conclusion

In this lab experiment we configured the buttons located at port 1.1 and 1.2 to turn on the red and green LEDs on button press and turn off on button release. We then wrote code so that each button is independent of the other, meaning the LED that stays on corresponds to the button that is pressed first. If we press S1 first red LED remains on if button s1 is also pressed and vice versa. Finally, I created my own design, which is similar to the previous design but, on opposite button press, the current LED that is lit will toggle.