Experiment #2 – Using the Push Buttons

# Youssef Samwel

# [yo800238@ucf.edu](mailto:yo800238@ucf.edu)

# EEL4742C Embedded Systems

Prof. Dr. Zakhia Abichar - Section 00419

# 2/1/2024



# **Project Description**

# This laboratory focuses on exploring the implementation of various functionalities using two push buttons and LEDs on the MSP430FR6989 microcontroller. The experiment begins by establishing the default configuration of the buttons, where they are active low and connected to 3.3 V through a 470 Ω resistor, ensuring a default high state. To mitigate false triggers due to noise or static electric fields, pull-up resistors are employed. The first experiment, labeled "2.1 Turning on the LED with the Button," demonstrates basic button functionality by turning on the red and green LEDs individually when the corresponding buttons are pressed. Subsequent experiments, labeled "2.2 Two Push Buttons: Occupancy Monitor" and "2.3/2.4 Two Push Buttons: Electrical Generator Load Control (v1/v2)," showcase more complex applications. In the latter experiments, the code introduces safety features, debouncing mechanisms, and efficient LED control based on button inputs. The provided code snippets and explanations illustrate the progression of functionality and safety measures implemented in each experiment. The overall goal is to provide a comprehensive understanding of button and LED interfacing on the MSP430FR6989 microcontroller.

# **2.0 Experiment Code**

A screenshot of a computer

Description automatically generated

Button default configuration

## 2.1 Turning on the LED with the Button

void TurningOnTheLEDWithTheButton\_2\_1() {

    for(;;) {

        if(IsButton1Pressed()) {

            LightRedLED(true);

        } else {

            LightRedLED(false);

        }

        if(IsButton2Pressed()) {

            LightGreenLED(true);

        } else {

            LightGreenLED(false);

        }

    }

}

Explain why the buttons are active low and why the pull-up resistor is used.The buttons are connected to 3.3 V by a 470 resistor, this will cause a default high state. Therefore, the button will be in the “unpressed” state if the signal is high. To ensure that we do not have a false trigger due to noise or static electric fields, we put the internal pull-up resistor to the high state.

The code works by entering an infinite loop where we poll button 1 and 2. If button 1 is pressed, we light the red led and same with button 2 (green led instead).

## 2.2 Two Push Buttons: Occupancy Monitor

void TwoPushButtonsOccupancyMonitor\_2\_2() {

    uint8\_t selectedBtn = 0;

    for(;;) {

        bool btn1State = IsButton1Pressed();

        bool btn2State = IsButton2Pressed();

        LightRedLED(btn1State);

        LightGreenLED(btn2State);

        \_delay\_cycles(20000);

    }

}

The code works by entering an infinite loop where we constantly set the state of the red and green LEDs to the clicked state of button 1 and button 2, respectively. The “\_delay\_cycles(20000);” statements is used for debouncing.

## 2.3 Two Push Buttons: Electrical Generator Load Control (v1)

void TwoPushButtonsElectricGeneratorLoadControl\_2\_3() {

    uint8\_t selectedBtn = 0;

    for(;;) {

        bool btn1State = IsButton1Pressed();

        bool btn2State = IsButton2Pressed();

        if(selectedBtn == 0) {

            if(btn1State) {

                selectedBtn = 1;

            } else if(btn2State) {

                selectedBtn = 2;

            }

        } else {

            selectedBtn &= btn1State ? 3 : ~1;

            selectedBtn &= btn2State ? 3 : ~2;

        }

        LightRedLED(selectedBtn == 1);

        LightGreenLED(selectedBtn == 2);

    }

}

This C code defines a function void TwoPushButtonsElectricGeneratorLoadControl\_2\_3() that continuously monitors the states of two buttons, representing potential electrical generator load control inputs. The variable selectedBtn is used to track which button is currently selected or pressed. Initially set to 0, it checks if either button 1 or button 2 is pressed, assigning the corresponding value to selectedBtn. Once a button is selected, subsequent iterations update the selection based on the current state of the buttons, using bitwise operations to clear the respective bits if the buttons are not pressed. The code also controls two LEDs: the red LED is activated when button 1 is selected, and the green LED is activated when button 2 is selected. The loop continues indefinitely, ensuring continuous monitoring and control of the system based on button inputs.

## 2.4 Two Push Buttons: Electrical Generator Load Control (v2)

void TwoPushButtonsElectricGeneratorLoadControl\_2\_4() {

    uint8\_t selectedBtn = 0;

    bool safetyFlag = false;

    for(;;) {

        bool btn1State = IsButton1Pressed();

        bool btn2State = IsButton2Pressed();

        if(!btn1State && !btn2State)

            safetyFlag = false;

        if(selectedBtn == 0) {

            if(btn1State) {

                selectedBtn = 1;

            } else if(btn2State) {

                selectedBtn = 2;

            }

        } else {

            if(btn1State && btn2State) {

                safetyFlag = true;

            }

            selectedBtn &= btn1State ? 3 : ~1;

            selectedBtn &= btn2State ? 3 : ~2;

        }

        LightRedLED(selectedBtn == 1 && !safetyFlag);

        LightGreenLED(selectedBtn == 2 && !safetyFlag);

    }

}

This C code defines a function void TwoPushButtonsElectricGeneratorLoadControl\_2\_4() that extends the functionality of the previous code by introducing a safety mechanism. The variable safetyFlag is introduced and set to false initially. The loop monitors the states of two buttons, and if both buttons are not pressed, it resets the safety flag. The button selection logic remains similar to the previous function, where initially, a button is selected if none is currently chosen. However, if both buttons are pressed simultaneously, the safetyFlag is set to true. The button selection is then updated based on the individual button states, taking into account the safety condition. The red and green LEDs are controlled accordingly, with the safety flag preventing the activation of the LEDs if both buttons are pressed simultaneously, ensuring a safety feature in the electric generator load control system. The loop continues indefinitely, providing continuous monitoring and control of the system with the added safety check.

# 3.0 Complete Code

#include <msp430fr6989.h>

#include <stdint.h>

#include <stdbool.h>

#define BUTTON1 BIT1

#define BUTTON2 BIT2

#define redLED BIT0 // Red LED at P1.0

#define greenLED BIT7 // Green LED at P9.7

/\*\*

 \* main.c

 \*/

bool IsButton1Pressed() {

    return (~P1IN & BUTTON1) ? true : false;

}

bool IsButton2Pressed() {

    return (~P1IN & BUTTON2) ? true : false;

}

void LightRedLED(bool state) {

    if(state)

        P1OUT |= redLED;

    else

        P1OUT &= ~redLED;

}

void LightGreenLED(bool state) {

    if(state)

        P9OUT |= greenLED;

    else

        P9OUT &= ~greenLED;

}

void ToggleRedLED() {P1OUT ^= redLED;}

void ToggleGreenLED() {P9OUT ^= greenLED;}

void TurningOnTheLEDWithTheButton() {

    for(;;) {

        if(IsButton1Pressed()) {

            LightRedLED(true);

        } else {

            LightRedLED(false);

        }

        if(IsButton2Pressed()) {

            LightGreenLED(true);

        } else {

            LightGreenLED(false);

        }

    }

}

void TwoPushButtonsOccupancyMonitor\_2\_2() {

    uint8\_t selectedBtn = 0;

    for(;;) {

        bool btn1State = IsButton1Pressed();

        bool btn2State = IsButton2Pressed();

        LightRedLED(btn1State);

        LightGreenLED(btn2State);

        \_delay\_cycles(20000);

    }

}

void TwoPushButtonsElectricGeneratorLoadControl\_2\_3() {

    uint8\_t selectedBtn = 0;

    for(;;) {

        bool btn1State = IsButton1Pressed();

        bool btn2State = IsButton2Pressed();

        if(selectedBtn == 0) {

            if(btn1State) {

                selectedBtn = 1;

            } else if(btn2State) {

                selectedBtn = 2;

            }

        } else {

            selectedBtn &= btn1State ? 3 : ~1;

            selectedBtn &= btn2State ? 3 : ~2;

        }

        LightRedLED(selectedBtn == 1);

        LightGreenLED(selectedBtn == 2);

    }

}

void TwoPushButtonsElectricGeneratorLoadControl\_2\_4() {

    uint8\_t selectedBtn = 0;

    bool safetyFlag = false;

    for(;;) {

        bool btn1State = IsButton1Pressed();

        bool btn2State = IsButton2Pressed();

        if(!btn1State && !btn2State)

            safetyFlag = false;

        if(selectedBtn == 0) {

            if(btn1State) {

                selectedBtn = 1;

            } else if(btn2State) {

                selectedBtn = 2;

            }

        } else {

            if(btn1State && btn2State) {

                safetyFlag = true;

            }

            selectedBtn &= btn1State ? 3 : ~1;

            selectedBtn &= btn2State ? 3 : ~2;

        }

        LightRedLED(selectedBtn == 1 && !safetyFlag);

        LightGreenLED(selectedBtn == 2 && !safetyFlag);

    }

}

int main(void)

{

    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer

    PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins

    P1DIR |= redLED;

    P9DIR |= greenLED;

    P1DIR &= ~BUTTON1;

    P1DIR &= ~BUTTON2;

    P1REN |= BUTTON1 | BUTTON2;

    P1OUT |= BUTTON1 | BUTTON2;

    LightRedLED(false);

    LightGreenLED(false);

    TwoPushButtonsElectricGeneratorLoadControl\_2\_4();

}

# **4.0 Student Q&A**

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

When a pin is in input mode, the P1OUT is used for pull-up or pull-down resistor. Otherwise, P1OUT set’s the value of the pin to either high or low.

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.

This condition will always be false because only the third bit can be set in the expression. To correct this condition, either solutions will work: if((Data & BIT3)==BIT3) or if(Data & BIT3).

3. Comment on the codes’ power-efficiency if the device is battery operated. Is reading the button via polling power efficient?Polling the button would induce a constant current flow to the MCU, increasing the average power consumption; this will have a negative impact on battery timespan.

# **5.0 Conclusion**

In conclusion, this laboratory experiment on MSP430FR6989 microcontroller explored fundamental button configurations and progressively introduced advanced functionalities, including an occupancy monitor and an electrical generator load control system with safety features. The provided code snippets and detailed explanations illustrated the evolution of functionality, emphasizing real-time responsiveness and safety considerations. Addressing pull-up resistors, correct bit comparisons, and power efficiency in the Q&A section added valuable insights. This hands-on experience effectively integrated theoretical concepts with practical implementation, enhancing understanding of embedded systems development.