Experiment #1 – Flashing LEDs

# Youssef Samwel

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

# EEL4742C Embedded Systems

Prof. Dr. Zakhia Abichar - Section 00419

# 1/19/2024 – 1/25/2024



# **Project Description**

In this experiment we explored control LEDs through setting port 1 and 9 values. In addition, to simple controls, we also experimented with animated patterns using LEDs such as the firefighter patterns.

# **2.0 Experiment Code**

A screenshot of a computer

Description automatically generated

LED Port Connection

## 2.2 Part 1 – Flashing One LED

void FlashLED() {

    volatile uint32\_t i;

    for(;;) {

           // Delay loop

           for (i=0; i<20000; i++) {}

           P1OUT ^= redLED; // Toggle the LED

       }

}

Part 1 of the lab involved the flashing of the red LED on the MSP430 board. The code works by entering an infinite loop where it toggles red LED port bit using the XOR logical operation. The empty for-loop is used to cause a delay to allow the user to see the flashing otherwise the LED would appear to have a “dimmer” output to the human eye. The keyboard, volatile, is used to prevent the compiler from optimizing the empty for-loop since it does nothing.

## 2.4 Part 2 – Flashing Two LEDs

void FlashingTwoLeds() {

    volatile uint32\_t i;

    for(;;) {

           // Delay loop

           \_delay\_cycles(500000);

           P1OUT ^= redLED; // Toggle the LED

           P9OUT ^= greenLED; // Toggle the LED

       }

}

Part 2 have the same operation as Part 1 with the addition of toggling the green LED along with the red LED. In this part, however, I switched from using the Spin wait for-loop to the compiler intrinsic function “\_delay\_cycles(500000);” this is a slightly better choice because it is easier to use and shows the exact amount of clock cycles would be delayed.

## 2.6 Part 3 – Setting Long Delay

Part 3 was completed in Part 2. Part 3 required the usage of 32-bit unsigned integer (defined in stdint.h header) to create a longer delay, which was not possible in the default “int” data type because it was 16-bit signed integer which would have overflowed with larger delays. Part 3 also showed an alternative to SW for-loop by using the compiler intrinsic SW loop called “\_delay\_cycles([clock cycles])”.

## 2.8 Part 4 – Firefighter Truck LED Pattern

Pattern 1

void FirefighterTruckLedPattern1() {

    volatile uint32\_t i;

    int8\_t counter = 8;

    for(;;) {

           // Delay loop

           \_delay\_cycles(45000);

           if(counter >= 0) {

               P1OUT ^= redLED; // Toggle the LED

               P9OUT &= ~greenLED;

           } else {

               P1OUT &= ~redLED;

               P9OUT ^= greenLED; // Toggle the LED

               if(counter <= -8)

                   counter = 8;

           }

           counter--;

       }

}

The first pattern attempted to match the pattern shown in the provided video, which was quick flashing of the red LED followed by the green LED (this was an exclusive operation). The code worked by using a directional counter, initially the counter would decrement the counter (while flashing the red LED) until the counter is negative. Once the counter is negative it will flash the green LED until the counter gets to a value of “-8” at the point the cycle is restarted.

Pattern 2

void FirefighterTruckLedPattern2() {

    uint16\_t j = 0;

    uint16\_t k = 0;

    for(;;) {

           // Delay loop

        P1OUT |= redLED;

        P9OUT &= ~greenLED;

        for(k = 1; k <= 4; k++) {

            for(j = 0; j < 5; j++) {

                switch(k) {

                case 1: \_delay\_cycles(300000); break;

                case 2: \_delay\_cycles(200000); break;

                case 3: \_delay\_cycles(100000); break;

                case 4: \_delay\_cycles(30000); break;

                }

                P1OUT ^= redLED; // Toggle the LED

                P9OUT ^= greenLED; // Toggle the LED

            }

        }

        P1OUT &= ~redLED;

        P9OUT &= ~greenLED;

        for(k = 1; k <= 6; k++) {

            \_delay\_cycles(200000);

            P1OUT ^= redLED; // Toggle the LED

            P9OUT ^= greenLED; // Toggle the LED

        }

     }

}

The second pattern starts with flashing one LED exclusive and then increases the flashing frequency and finishes by flashing both LEDs flashing. The function utilizes nested loops to create a specific LED pattern. Inside the outer loop, the code alternates between turning on and off red and green LEDs with specified delay intervals. The first set of nested loops creates a sequence of flashes with varying delays, simulating a specific lighting pattern. After that sequence, there's another set of loops for a different pattern. The function runs indefinitely, continuously generating the defined LED patterns, possibly serving as a visual signal or indicator in the firefighter truck system.

# 3.0 Complete Code

#include <msp430fr6989.h>

#include <stdint.h>

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

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

void FlashLED() {

    volatile uint32\_t i;

    for(;;) {

           // Delay loop

           for (i=0; i<20000; i++) {}

           P1OUT ^= redLED; // Toggle the LED

       }

}

void FlashingTwoLeds() {

    volatile uint32\_t i;

    for(;;) {

           // Delay loop

           \_delay\_cycles(500000);

           P1OUT ^= redLED; // Toggle the LED

           P9OUT ^= greenLED; // Toggle the LED

       }

}

void FirefighterTruckLedPattern1() {

    volatile uint32\_t i;

    int8\_t counter = 6;

    for(;;) {

           // Delay loop

           \_delay\_cycles(45000);

           if(counter >= 0) {

               P1OUT ^= redLED; // Toggle the LED

               P9OUT &= ~greenLED;

           } else {

               P1OUT &= ~redLED;

               P9OUT ^= greenLED; // Toggle the LED

               if(counter <= -8)

                   counter = 8;

           }

           counter--;

       }

}

void FirefighterTruckLedPattern2() {

    uint16\_t j = 0;

    uint16\_t k = 0;

    for(;;) {

           // Delay loop

        P1OUT |= redLED;

        P9OUT = 0;

        for(k = 1; k <= 4; k++) {

            for(j = 0; j < 5; j++) {

                switch(k) {

                case 1: \_delay\_cycles(300000); break;

                case 2: \_delay\_cycles(200000); break;

                case 3: \_delay\_cycles(100000); break;

                case 4: \_delay\_cycles(30000); break;

                }

                P1OUT ^= redLED; // Toggle the LED

                P9OUT ^= greenLED; // Toggle the LED

            }

        }

        P1OUT &= ~redLED;

        P9OUT &= ~greenLED;

        for(k = 1; k <= 6; k++) {

            \_delay\_cycles(200000);

            P1OUT ^= redLED; // Toggle the LED

            P9OUT ^= greenLED; // Toggle the LED

        }

     }

}

void main(void) {

    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer

    PM5CTL0 &= ~LOCKLPM5; // Disable GPIO power-on default highimpedance mode

    P1DIR |= redLED; // Direct pin as output

    P9DIR |= greenLED; // Direct pin as output

    P1OUT = P9OUT = 0; // Reset port outputs

    FirefighterTruckLedPattern2();

}

# **4.0 Student Q&A**

1. In this lab, we used a delay loop to create a small delay; what is its effect on the battery life if the device is battery operated? Is it a good way of generating delays?   
Spin-wait loops increase power usage leading to reduced battery life. It is also not a good way of generating delays because it relies on the CPU clock frequency and does not allow the developer to easily use time periods delays (such as 10 milliseconds).

2. The MSP430 CPU runs on a clock that can be slowed down or sped up; what happens to the delay generated by the delay loop if the clock driving the CPU speeds up or slows down?  
As stated in the previous question, the delay will either get shorter or longer based on the change in clock frequency.

3. How does the code run in the debug mode? Is the microcontroller running as an independent computer?  
The code communicates with the computer to stop/continue execution of the program through the JTAG hardware. The microcontroller is not independent from the computer.

4. How does the code run in the normal mode? Is the microcontroller running as an independent computer?  
The code does not communicate with the host computer and is independent of the computer.

5. In which mode does the reset button work?  
Normal mode.

6. What is the data type uint16\_t ? What about int16\_t ? Are these standard C syntax? These are not standard C syntax, however, they’re included in most standard C compilers (typically inside stdint.h header file). The typedef use the compiler’s datatype to ensure that the developer’s intended data-width is used. The type uint16\_t is unsigned 16-bit integer whereas int16\_t is signed 16-bit integer.

# **5.0 Conclusion**

In Conclusion, the lab experiment involved controlling LEDs on an MSP430 board, progressing from basic flashing patterns to more complex firefighter truck LED sequences. The code demonstrated efficient delay mechanisms and addressed questions about power consumption and clock frequency impact. The experiment provided hands-on experience in embedded systems programming and coding considerations for system efficiency.