Lab 1 Report
EEL4742C - 00446

Yousef Awad

September 2025

Contents

1.0 Introduction

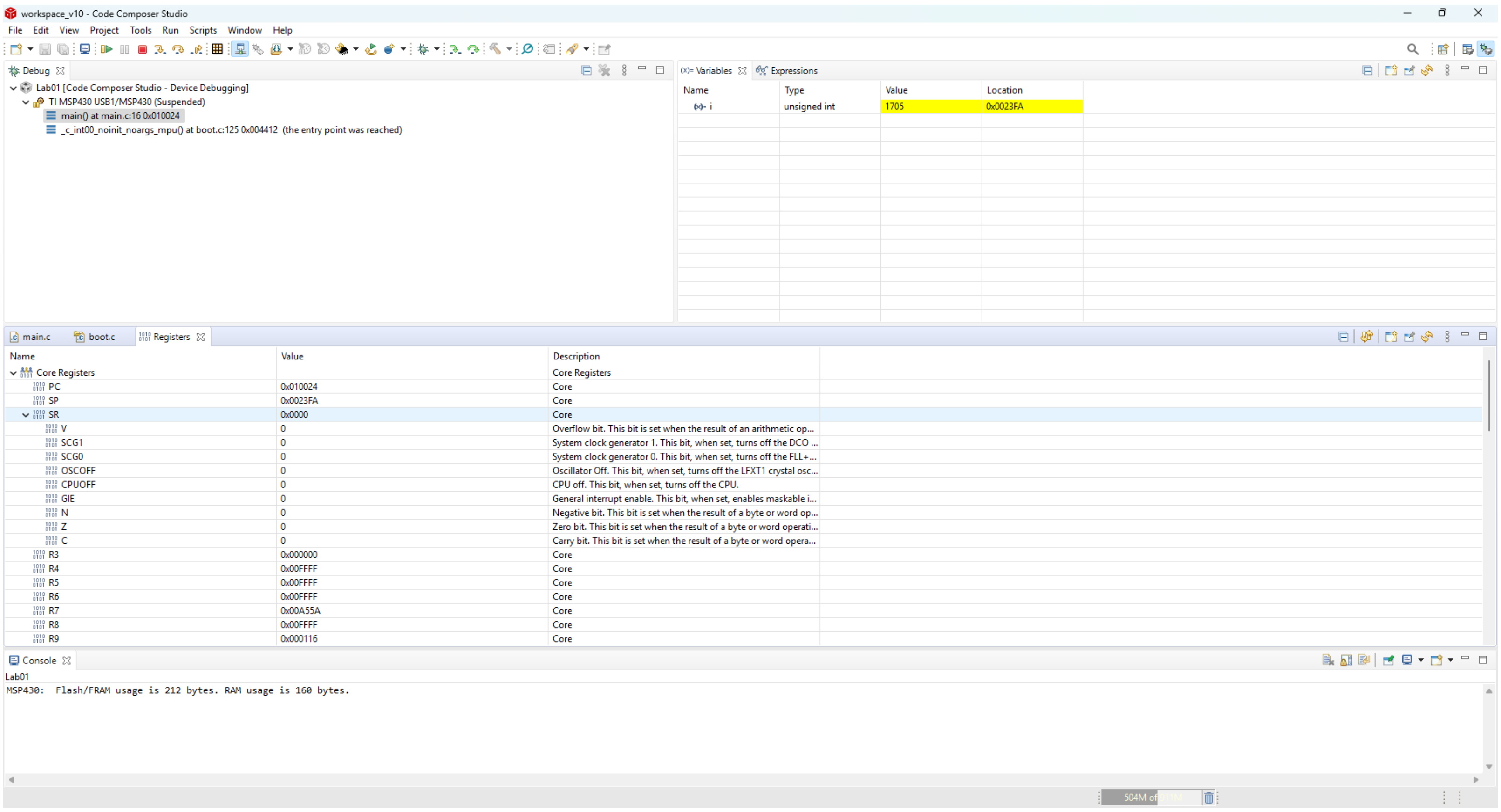
This lab has us exploring a totally new (very new, never been done before ever) method of pushing to the MSP430 via Code Composer Studio as well as to try and control the LEDs it has (this also never has been done until this lab, I swear).
This experiment helped us build the confidence and comprehension of the topics covered in the lecture while also letting us build familiarity with the design chocies/logic behind developing embedded systems.

1.1 Documentation

The first resources that were provided to us within the experiment was the MSP430-FR6989 development board documentation via the Lab Manual.
Alongside the board documentation we were also given the Family User's Guide which specifically explained and showed the diagrams of the components in the MSP430. Alongside this we were also given the Chip's Data Sheet which gives out the pinout configuration of all the GPIO's and LEDs and other things connected to the MSP430 that we can use via the already written by the TI dev team.

1.2 Flashing the LED

For flashing the LEDs we first had to figure out the Code Composer Studio (CCS) works. First, we have to create a new project via File → New Project. After this we then use a blank project so that CCS doesn't do any thingy-majigs that break the code (As I had to debug this entire thing when I accidentally did that on my laptop...). To code this thing up we first have to disable the watchdog timer so that the system doesn't hang (aka it will think the program is constantly crashing and keep on reestarting it). After this we declare a variable for the loop counters, as well as set up the GPIO pins via the P1DIR and P1OUT, with P1DIR being set to output for the red LED as well as to turn it off by default. After this we then do an infinite loop followed by the internal loop that cycles through 200,000 additions (or approximaately 0.312 seconds) before toggling the red LED on and off.
Now go to the next page for the graphic of the registers...


workspace_v10 - Code Composer Studio

Debug

```
Lab01 [Code Composer Studio - Device Debugging]
  TI MSP430 USB1/MSP430 (Suspended)
    main() at main.c:16 0x010024
    _c_int00_noinit_noargs_mpu() at boot.c:125 0x004412  (the entry point was reached)
```

Variables

Name	Type	Value	Location
i	unsigned int	1705	0x0023FA

Registers

Name	Value	Description
Core Registers		Core Registers
PC	0x010024	Core
SP	0x0023FA	Core
SR	0x0000	Core
V	0	Overflow bit. This bit is set when the result of an arithmetic op...
SCG1	0	System clock generator 1. This bit, when set, turns off the DCO ...
SCG0	0	System clock generator 0. This bit, when set, turns off the FLL+...
OSCOFF	0	Oscillator Off. This bit, when set, turns off the LFXT1 crystal osc...
CPUOFF	0	CPU off. This bit, when set, turns off the CPU.
GIE	0	General interrupt enable. This bit, when set, enables maskable i...
N	0	Negative bit. This bit is set when the result of a byte or word op...
Z	0	Zero bit. This bit is set when the result of a byte or word operati...
C	0	Carry bit. This bit is set when the result of a byte or word opera...
R3	0x000000	Core
R4	0x00FFFF	Core
R5	0x00FFFF	Core
R6	0x00FFFF	Core
R7	0x00A55A	Core
R8	0x00FFFF	Core
R9	0x000116	Core

Console

```
Lab01
MSP430:  Flash/FRAM usage is 212 bytes. RAM usage is 160 bytes.
```

As well as here's the code!!

```c
// Code that flashes the red LED
#include <msp430fr6989.h>
#include <stdint.h> // for uint32_t

#define redLED BIT0 // P1.0
#define greenLED BIT7 // P9.7

void main(void)
{
    volatile uint i;
    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Disables GPIO power on by default to high impedance

    P1DIR |= redLED;
    P1OUT &= ~redLED;

    for(;;)
    {
        for(i=0; i<20000; i++) // Generates a delay of ~0.312 seconds
        {
            P1OUT ^= redLED; // Toggle the led via xor
        }
    }
}
```

1.3 Flashing Two LEDs

Now we were told by the lab manual to code up 2 flashing leds!!!!!!! WOWWWWWWW. To do this I simply updated the code above to add in the Green LED in P9 as well as found the schematic for the LED Controller proper!

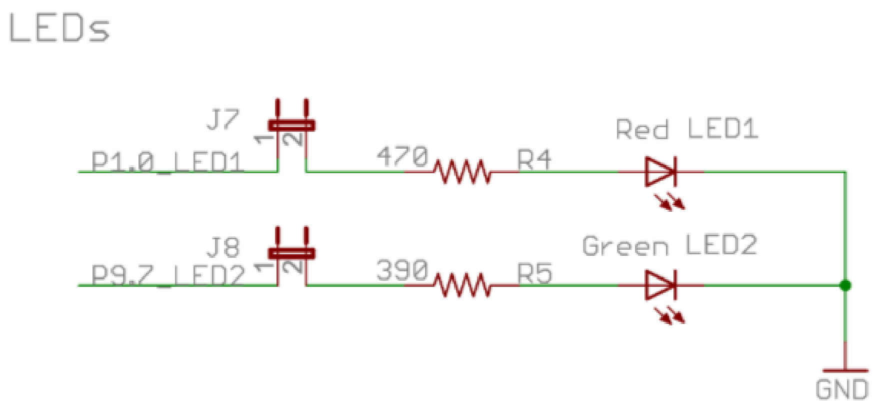
```
// Code that flashes the red LED
#include <msp430fr6989.h>
#include <stdint.h> // for uint32_t

#define redLED BIT0 // P1.0
#define greenLED BIT7 // P9.7

void main(void)
{
    volatile uint i;
    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Disables GPIO power on by default to
    high impedance

    P1DIR |= redLED;
    P1OUT &= ~redLED;

    for(;;)
    {
        for(i=0; i<20000; i++) // Generates a delay of ~0.312
    seconds
        {
            P1OUT ^= redLED; // Toggle the led via xor
        }
    }
}
```



1.4 Setting a Long Delay

Now we were told to make the delay longer! BUT, we have glaring problem. That problem????? It's the fact that we accidentally used a 16 bit unsigned integer for our loop counter... BUT the lab manual wants me to make a VERY VERY VERY long delay. To fix this, we can simply either... use a 32 bit unsigned integer ORRRR simply use the _delay_cycles function that just tells the CPU "YOOOOO, YOU GOTTA STOP NOW", and it stops for that long... Below, I shall show both answers...

```c
// Code that flashes the red LED
#include <msp430fr6989.h>
#include <stdint.h> // for uint32_t

#define redLED BIT0 // P1.0
#define greenLED BIT7 // P9.7

void main(void)
{
    volatile uint32_t i; // solution 2
    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Disables GPIO power on by default to high impedance

    P1DIR |= redLED;
    P1OUT &= ~redLED;

    P9DIR |= greenLED; // Direct pin as output
    P9OUT &= ~greenLED; // Turns led off

    for(;;)
    {
        P9OUT ^= greenLED; // This makes it out of sync
        // Solution 2
        // for(i=0; i<200000; i++) // Generates a delay of ~3.12 seconds
        // {
        //     // Nothing, for delay
        // }
        // solution 3
        _delay_cycles(3000000); // 3*10^6 cycle stop or 3 seconds worth
        P1OUT ^= redLED; // Toggle the led via xor
        P9OUT ^= greenLED; // In-sync
    }
}
```

1.5 Designing the Firetruck LED Patterns

For this one, we were told to make two fire truck patterns (because we like fire trucks and not ambulances. OK!!!!!!) and are only told to just give you the code.

```c
// Code that is Fire Truck Pattern 1
#include <msp430fr6989.h>
#include <stdint.h> // for uint32_t

#define redLED BIT0 // P1.0
#define greenLED BIT7 // P9.7

void main(void)
{
    volatile uint32_t i; // solution 2
    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Disables GPIO power on by default to
    high impedance

    P1DIR |= redLED;
    P1OUT &= ~redLED;

    P9DIR |= greenLED; // Direct pin as output
    P9OUT &= ~greenLED; // Turns led off

    for (;;)
    {
        int j;
        int k;
        for (k = 0; k < 2; k++)
        {
            for (j = 0; j < 8; j++)
            {
                P1OUT ^= redLED;
                _delay_cycles(31250);
            }
            for (j = 0; j < 8; j++)
            {
                P9OUT ^= greenLED;
                _delay_cycles(31250);
            }
        }
        for (k = 0; k < 2; k++)
        {
            // half
            for (j = 0; j < 8; j++)
            {
                P1OUT ^= redLED;
                _delay_cycles(62500);
            }
            for (j = 0; j < 8; j++)
            {
                P9OUT ^= greenLED;
                _delay_cycles(62500);
            }
        }
    }
}
```

```c
// Code that shows Fire Truck Pattern 2
#include <msp430fr6989.h>
#include <stdint.h> // for uint32_t

#define redLED BIT0 // P1.0
#define greenLED BIT7 // P9.7

void main(void)
{
    volatile uint32_t i; // solution 2
    WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Disables GPIO power on by default to high impedance

    P1DIR |= redLED;
    P1OUT &= ~redLED;

    P9DIR |= greenLED; // Direct pin as output
    P9OUT &= ~greenLED; // Turns led off

    for (;;)
    {
        int j;
        for (j = 0; j < 8; j++)
        {
            P9OUT ^= greenLED; // This makes it out of sync
            _delay_cycles(500000); // 3*10^6 cycle stop or 3 seconds worth
            P1OUT ^= redLED; // Toggle the led via xor
        }
        for (j = 0; j < 8; j++)
        {
            P9OUT ^= greenLED; // This makes it out of sync
            _delay_cycles(250000); // 3*10^6 cycle stop or 3 seconds worth
            P1OUT ^= redLED; // Toggle the led via xor
        }
        for (j = 0; j < 8; j++)
        {
            P9OUT ^= greenLED; // This makes it out of sync
            _delay_cycles(125000); // 3*10^6 cycle stop or 3 seconds worth
            P1OUT ^= redLED; // Toggle the led via xor
        }

        P9OUT &= ~greenLED; // Turns green off
        P1OUT &= ~redLED; // Turns red off

        // On then off 3 times
        for (j = 0; j < 8; j++)
        {
            P9OUT ^= greenLED;
            P1OUT ^= redLED;
            _delay_cycles(250000);
        }
    }
}
```

Student Q&A

Q1

Given: 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 setting delays?

Using a delay/for loop for the small delay is honestly a terrible idea, due to the fact that it is having to compute the additions of the amount of times that it loops and therefore wastes energy doing addition cycles that are unneeded when you can simply use the _delay_cycles function instead.

Q2

Given: The MSP430 CPU clock can be configured to multiple frequencies via software. What happens to the delay generated by the delay loop if the CPU clock frequency is changed?

If the frequency of the clock changes, the delay loop will also change due to the fact that the delay is reliant on the amount of cycles, OR, the amount of additions PER cycle that it computes.

Q3

Given: How does the code run in the debug mode? Is the microcontroller running as an independent computer?

In debug mode, the code runs in tandem to the computer you are on, and therefore cannot be run as an independent computer.

Q4

Given: How does the code run in the normal mode? Is the microcontroller running as an independent computer?

In normal mode, the code runs seperate to the computer you used to push the code, and therefore means it is running as an independent computer.

Q5

Given: In which mode does the reset button work?

The Reset button only works when you are in normal mode. It *does not* work in debug mode.

Q6

Given: What is the data type uint16 t ? What about int16 t ? Are these standard C syntax?

The data type of uint16_t is an unsigned integer of bit length 16. int16_t is the signed integer variant again with 16 bit length. These are not in the standard C *syntax*, BUT can be included as a module as it is part of the C standard *library* just like stdio or stdstring.

Conclusion

To sum it all up, this lab was a perfect introduction to using the MSP430-FR6989 as well as the Code Composer Studio software stack. It provided me with experience on how to push code to the board, as well as finely control the LEDs using logical operators to set individual bits on and off on a given register. My only complaint for this lab is that the videos for the Fire Truck portion were hard to watch and it was also frustrating to try and guesstimate what they were doing without being able to slow down the video. I do recommend for next time uploading them as a video proper so that future student have the ability to slow down the video and accurately be able to see the changes to the LEDs.