

EEL 4742C: Embedded Systems

Christopher Badolato

9/15/2019

EEL4742-0011

Introduction

In this lab experiment we will be programming the MSP430 to toggle different light effects with the LEDs located at P1.0 and P9.7.

Part 1.2:

Flash the LED located at P1.0 with a delay loop.

Part 1.3:

Set a longer period with an unsigned 32-bit int that takes two register spaces

Part 1.4:

Flash two LEDs P1.0 and P9.7 out of sync

Part 1.5:

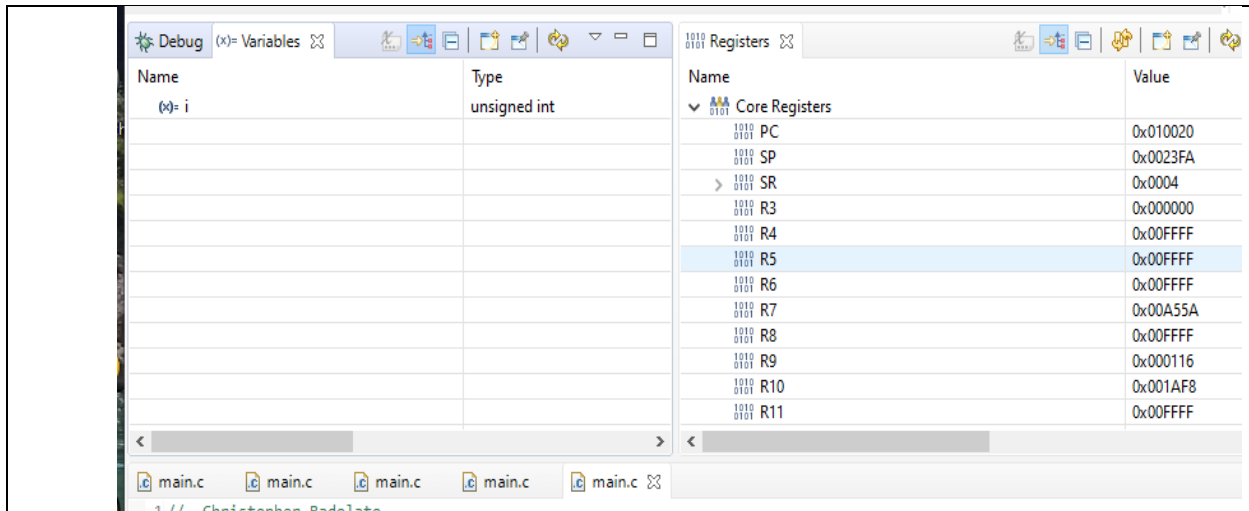
Flash two LEDs P1.0 and P9.7 at different rates

Part 1.2: ...

My approach to solving this part of the lab was to follow the code given to understand the pin layout of the MSP430. Our LED is located at P1.0, this corresponds to the variable P1. Then, the .0 corresponds to the BIT (BIT0 on = 0000 0000 0000 00001) value of the individual Pin (P1.1, P1.2, P2.0, P2.1 this varies by board). Though before we declare the variables, we must set the pin with direction as input (0) or output (1) with P1DIR (Direction) that corresponds to the bit value of P1.0. We must then initialize the pin. If input, P1OUT can be set to on (1) or off (0) at the .0 bit that corresponds to the pin we are selecting. If input, we use P1IN. In our case we use output and set the LED as off by ANDING our P1OUT with the inverse of redLED which will clear the value to 0.

Once everything is initialized, we can start our running loop, this will keep the controller running our code continuously. Inside the loop we have a delay loop which will essentially counts from 0 to 20000 to create a slight delay in the toggling of the LED (That way we can notice the LED's toggling). The next line will XOR our P1OUT toggling the voltage to the pin each time by resetting the bit (between 0000 0000 or 0000 00001) each time.

Each pin labeled P1 is represented by the same 8 bits. So, we can keep the LED located at BIT0 and use the other pins by editing that same P1OUT value. so P1.1 is BIT1 (0000 0010).



The screenshot shows a debugger interface with two main panels. The left panel, titled 'Debug (x)= Variables', displays a table with columns 'Name' and 'Type'. It shows a single variable '(x)= i' of type 'unsigned int'. The right panel, titled 'Registers', displays a table with columns 'Name' and 'Value'. It shows 'Core Registers' including PC, SP, SR, R3, R4, R5 (highlighted), R6, R7, R8, R9, R10, and R11, each with a hexadecimal value. The bottom of the image shows the source code for 'main.c'.

```
// Christopher Badolato
// 9/9/20
// EEL 4742L-0011

#include <msp430.h>
#include <stdint.h>
#define redLED BIT0
/**
 * main.c
 */
int main(void){
    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
    volatile unsigned int i;

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

    P1DIR |= redLED; // Direct pin as output
    P1OUT &= ~redLED; // Turn LED Off

    for(;;) {
        //Delay Loop
        for(i = 0; i<20000; i++) {}
        //Toggle the red
        P1OUT ^= redLED;
    }
}
```

Part 1.3: ...

My approach to solving this part of the lab was to simply follow the code given. Since our previous example has a regular integer, the value is a 16-bit integer (unsigned) therefore we can only represent ~65000 values, if our int i becomes greater than this value, the code will not run. So, we can change the integer to a 32-bit value using 2 spaces in memory to represent the value increasing our loop maximum to 4 billion (Which might take a while). In this case I just increased the value to 120,000 which exceeds the 16-bit integer maximum.

```
// Christopher Badolato
// 9/9/20
// EEL 4742L-0011

#include <msp430.h>
#include <stdint.h>
#define redLED BIT0

/**
 * main.c
 */
int main(void){
    WDTCTL = WDTPW | WDTHOLD;    // stop watchdog timer
    //volatile i;
    volatile uint32_t i;

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

    P1DIR |= redLED; // Direct pin as output
    P1OUT &= ~redLED; // Turn LED Off

    //Toggle Red LED with a Longer delay
    //with a 32-bit int i to loop through.
    for(;;) {
        for(i = 0; i<120000; i++) {

        }

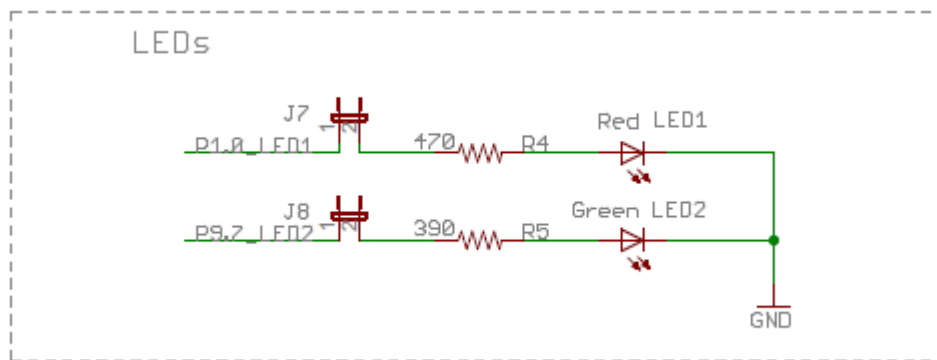
        P1OUT ^= redLED;

        //With 2 delay loops and 16-bit integer
        /*
        for(i = 0; i<60000; i++) {
        }
        for(i = 0; i<60000; i++) {
        }
        P1OUT ^= redLED;
        */
    }
}
```

Part 1.4: ...

My approach to solving this part of the lab was to create a new variable and definition for the green LED located at Pin 9.7. We set the P9DIR at pin 9.7 on by turning the bit at P9DIR to 1 by OR operation with BIT7 (1000 0000). We must also initially turn the LED off by AND operation with the inverse of BIT7 (0000 0000).

Now we are set to start the continuous loop. First, the red led will be toggled with the XOR operation turning it on. Then, our delay loop will occur, and then the red led will toggle with the XOR operation adding the delay between the toggles we can alternate the LED.



```
// Christopher Badolato
// 9/9/20
// EEL 4742L-0011

#include <msp430.h>
#include <stdint.h>
#define redLED BIT0
#define greenLED BIT7

/**
 * main.c
 */
int main(void){
    WDTCTL = WDTPW | WDTHOLD;    // stop watchdog timer
    volatile unsigned int i;

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

    P1DIR |= redLED; // Direct pin as output
    P1OUT &= ~redLED; // Turn LED Off
```

```

P9DIR |= greenLED; // Direct pin as output
P9OUT &= ~greenLED; // turn green LED off
//Toggle Red LED
//Have the green LED toggle twice before the RED toggles.
for(;;) {
    P1OUT ^= redLED;
    //P9out ^= greenLED;
    for(i = 0; i<20000; i++) {}
    //Toggle the green LED
    P9OUT ^= greenLED;
}
}

```

Part 1.5: ...

My approach to solving this part of the lab was to follow the approach above to flash the LEDs at different rates, but this time we will allow the green LED to toggle twice before the red LED will toggle. By adding a nested for loop, we will be able to run the green LED toggle and delay twice before the continuous loop resets, toggling the red LED.

```

// Christopher Badolato
// 9/9/20
// EEL 4742L-0011

#include <msp430.h>
#include <stdint.h>
#define redLED BIT0
#define greenLED BIT7

/**
 * main.c
 */
int main(void){
    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
    volatile unsigned int i;
    volatile unsigned int j;

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

    P1DIR |= redLED; // Direct pin as output
    P1OUT &= ~redLED; // Turn LED Off

    P9DIR |= greenLED; // Direct pin as output
    P9OUT &= ~greenLED; // turn green LED off
    //Toggle Red LED
    //Have the green LED toggle twice before the RED toggles.
    for(;;) {
        P1OUT ^= redLED;

```

```
for(j = 0; j < 2; j++){  
    for(i = 0; i<20000; i++) {}  
        //Toggle the green LED  
        P9OUT ^= greenLED;  
    }  
}
```

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?

No, the processor will still have to be running to allow the loop to run during the delay. Therefore, the CPU is still processing.

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?

Yes, a microcontroller is just sequential logic circuit therefore it needs a clock to run. The edges of the clock are needed to drive all logic within the CPU. Each instruction takes a certain number of clock cycles. The MSP430 has variable clock sources which can drive different clock signals to the CPU therefore changing the time it takes to execute the loop.

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

No, The computer is running the debugger to test the code before it is loaded onto the board.

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

Yes, the code is stored in the flash memory and can be used apart from the desktop.

5. In which mode does the reset button work?

Normal Mode, we can reset the code on the board on click, the lights will reset their flashing. If held the code will stay off until release.

6. What is the data type `uint16_t` ? What about `int16_t` ? Are these standard C syntax?

`uint16_t` - unsigned 16-bit integer
`int16_t` - Signed 16-bit integer.

Syntax in standard C:
Short does not guarantee a 16-bit int.
`int16_t` and `uint16_t`
CAN be used in standard C.

Conclusion

In conclusion, we have learned how to first, set pins up to a variable value, and how these pins correspond to a certain BIT of the PX variable. We must, give pins direction with the variable `PXDIR` (input or output). Then, we can turn the pin on or off with `P1OUT` and `P1IN`. We also studied, creating a delay loop to toggle our pins to notice a change (on or off), creating a larger loop with a 32-bit integer, as well as toggling multiple pins (in this case the LEDS) at different rates.