Lab 1

CCS AND MORE

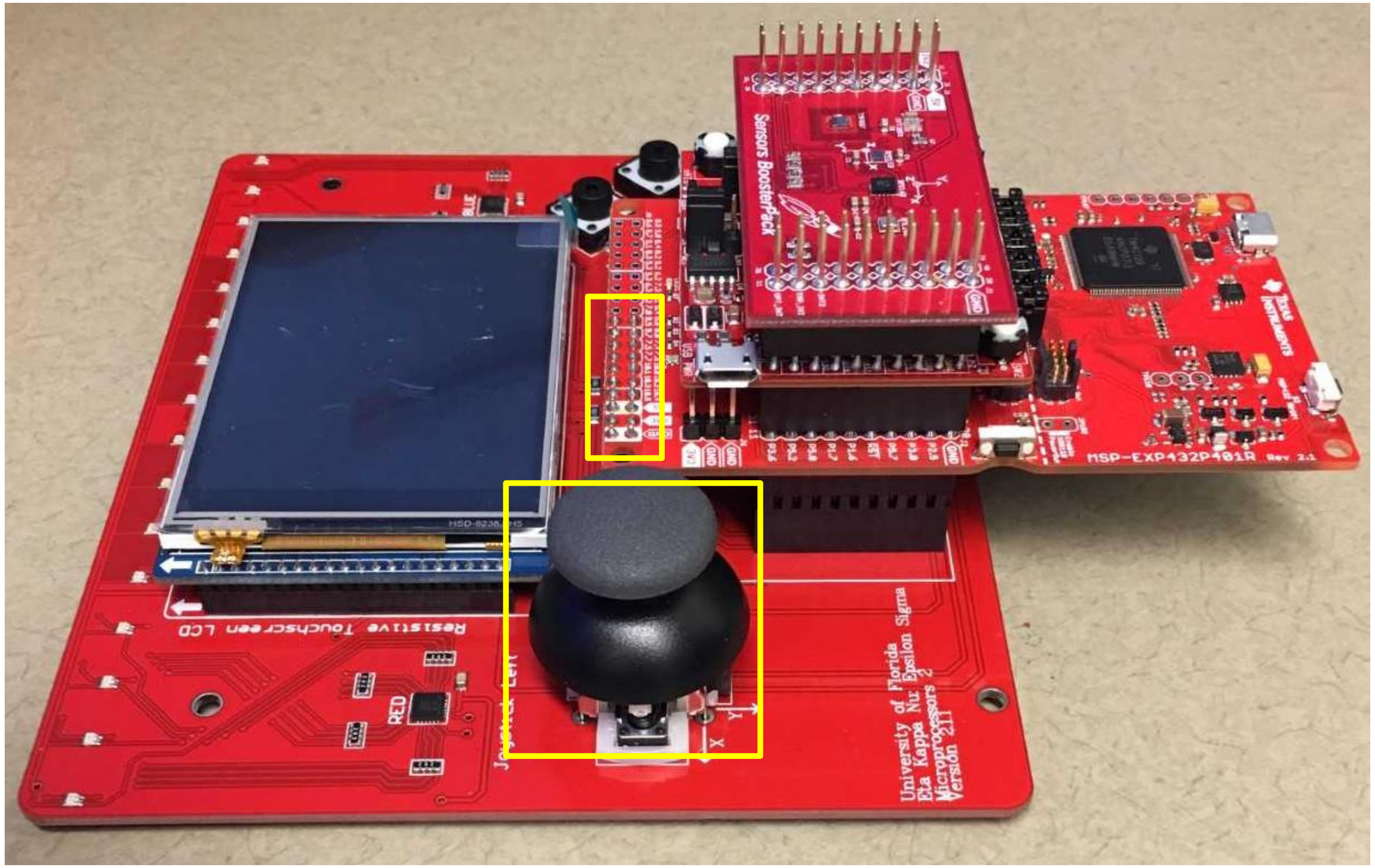
Overview

- Each subgroup of 10 students has a 2 hour lab session each week.
- This lab session is for demonstrating your assignment to the TAs and asking for help from the TAs.
- 4 labs in total constitute 60% of your grade.
- Each lab will span 2-3 weeks.
- Within this time it is up to you to demonstrate to the lab TAs that your code works on the board.
- The TA will go through your code and ask you questions. Code quality can effect your final grade for that lab.
- The 5th lab is the final project which is 20% of the course grade
- you can earn up to 10% extra credit if you build a new project on top of Lab 5

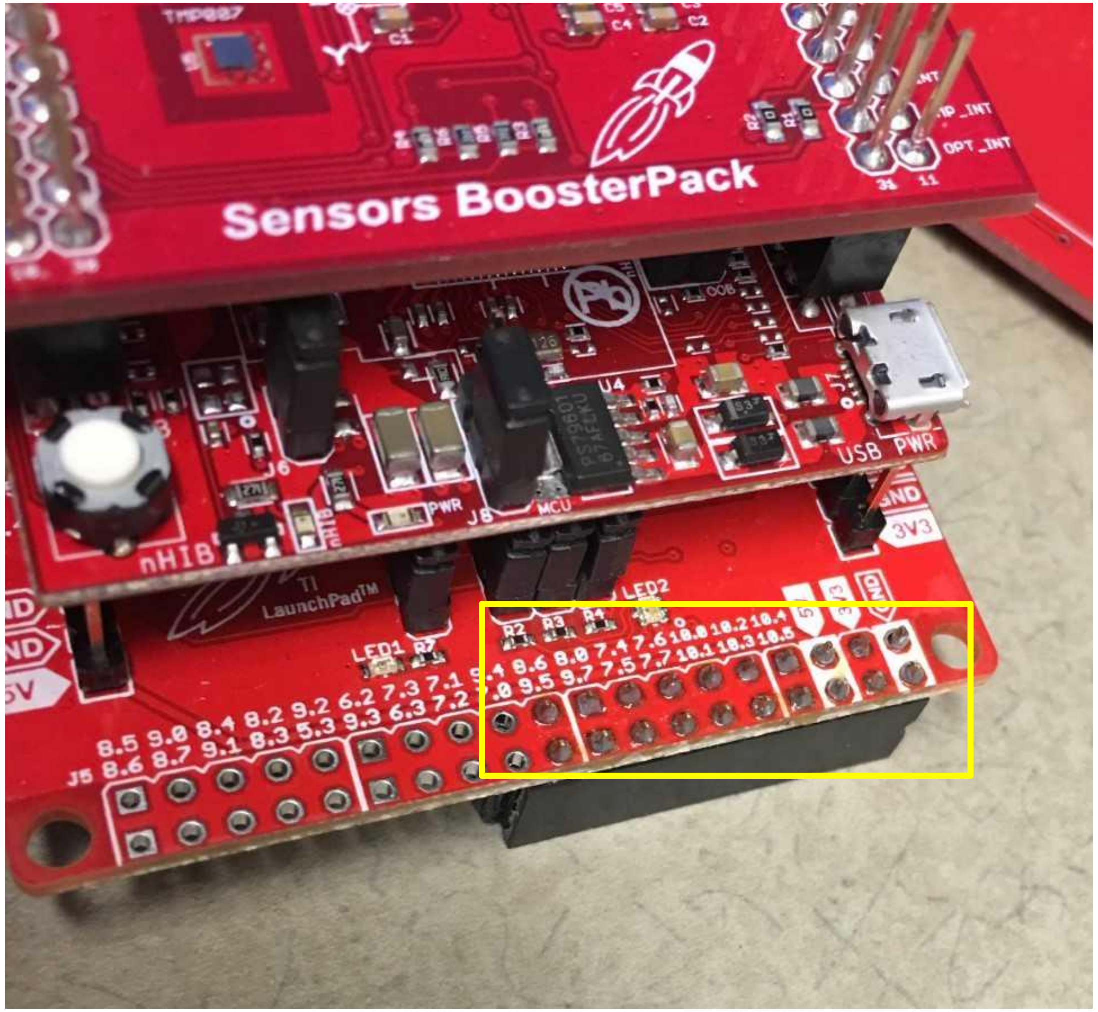
Board Preparation

- MSP432P401R Board and extensions

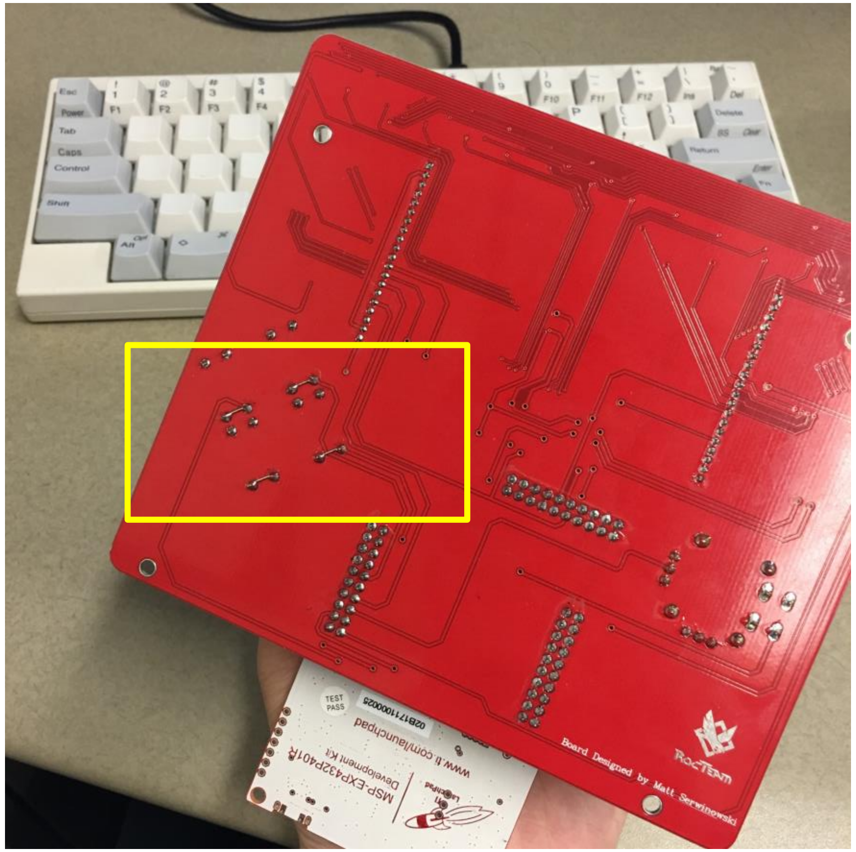
- You need to solder the joystick and the female headers

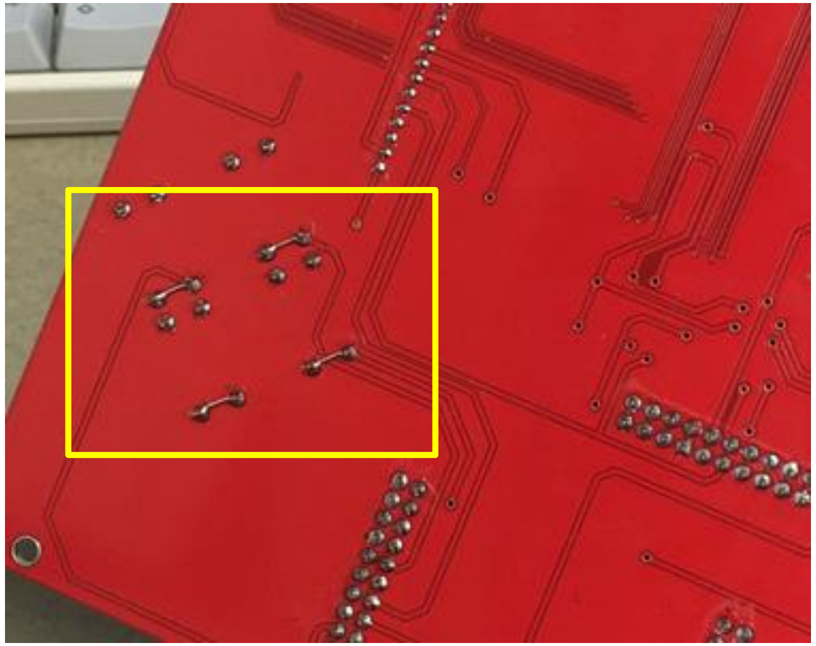


Board Preparation



Board Preparation



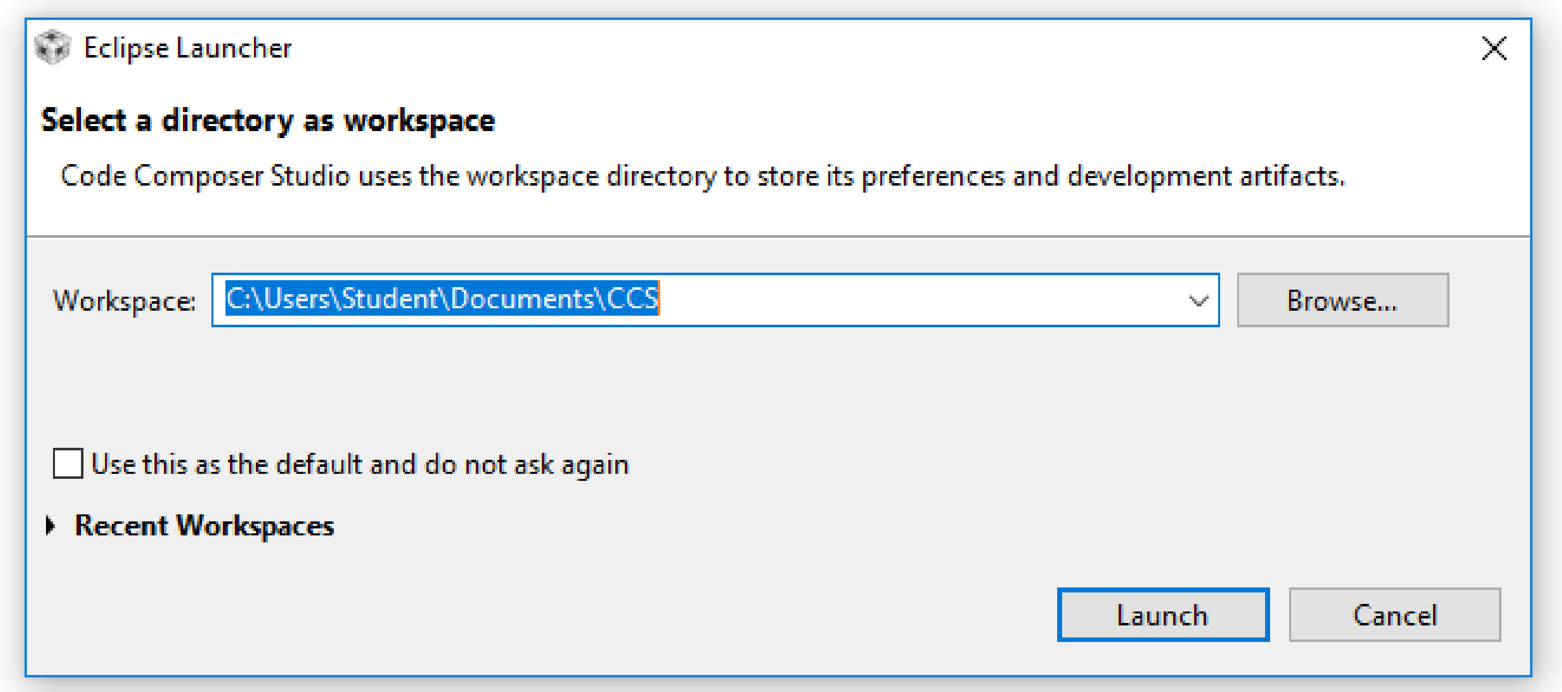


Lab 1 : Part A

- Implement a Fletcher-16 checksum. The algorithm is discussed in the lab manual and on Wikipedia.
    - You can implement the simpler "straightforward" fletcher-16 implementation rather than the optimized one.
    - You have to write your main function in C and have it call an assembly subroutine.
    - Within the assembly code you have to call a C function.
    - You have to write the fletcher code in C too and compare the output with the assembly version.
    - If the check passes you have to send a message through the UART

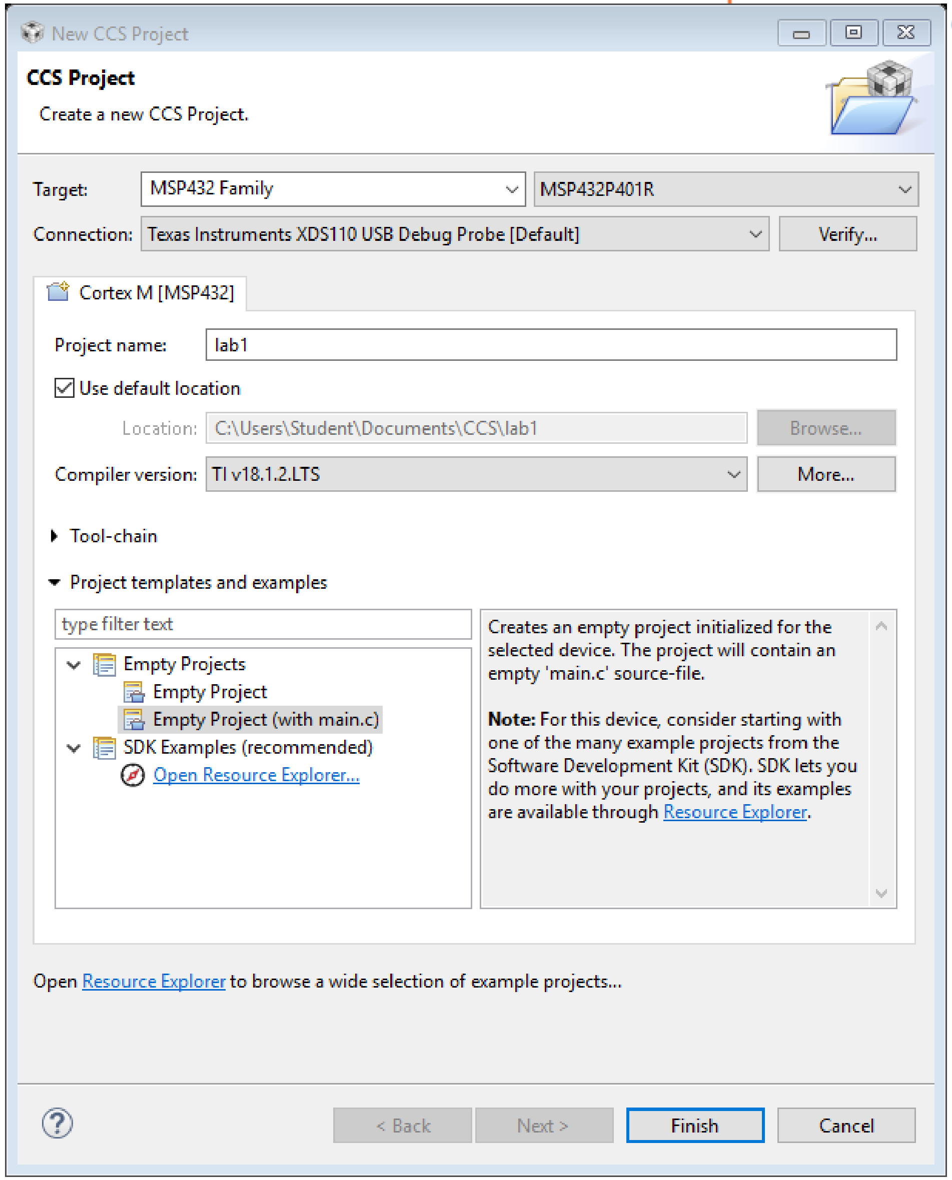
CCS

- Is based on eclipse
- Select a workspace

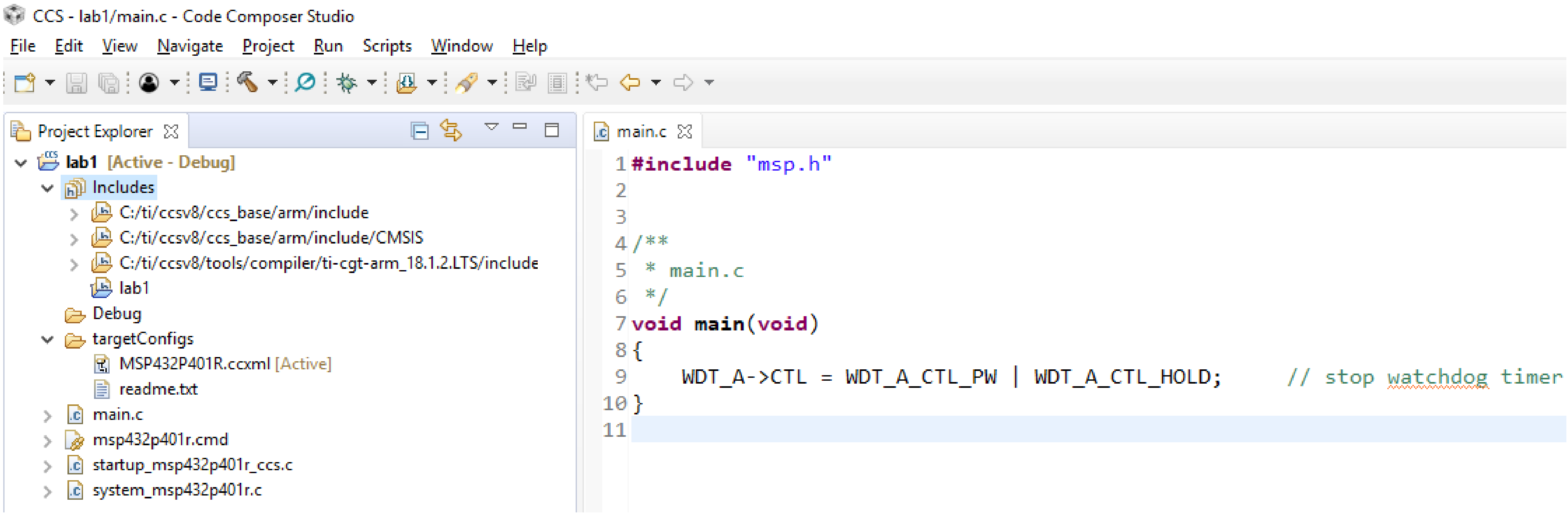


CCS

- Create new project: **File->New->CCS Project**
- Have the board connected and then select the MSP432 family with the MSP432P401R device.
- The debugger should be configured automatically

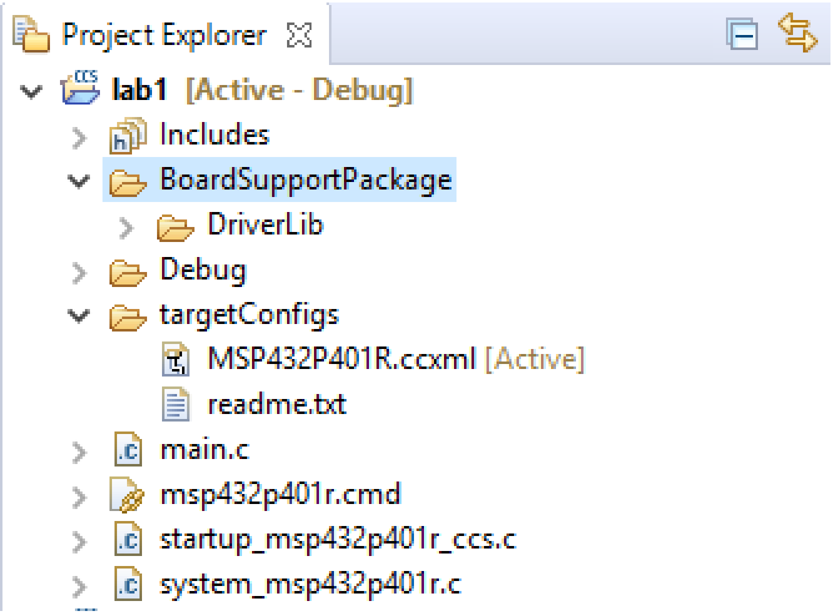


CCS


CCS - lab1/main.c - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
Project Explorer
lab1 [Active - Debug]
Includes
C:/ti/ccsv8/ccs_base/arm/include
C:/ti/ccsv8/ccs_base/arm/include/CMSIS
C:/ti/ccsv8/tools/compiler/ti-cgt-arm_18.1.2.LTS/include
lab1
Debug
targetConfigs
MSP432P401R.ccxml [Active]
readme.txt
main.c
msp432p401r.cmd
startup_msp432p401r_ccs.c
system_msp432p401r.c
main.c
#include "msp.h"
/**
 * main.c
 */
void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;   // stop watchdog timer
}

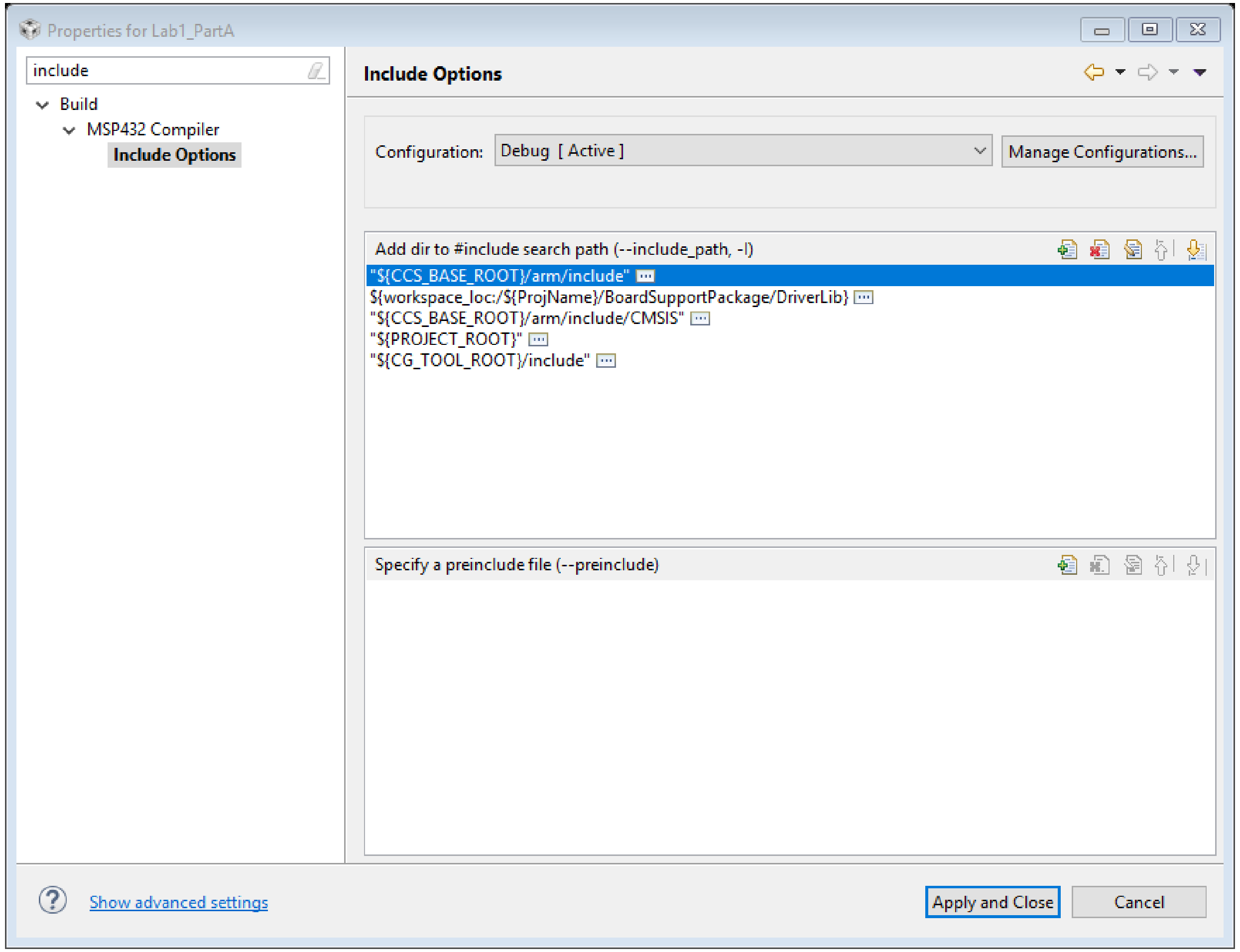
Board Support Package (BSP)

- A library for high level functions for hassle-free operations on a specific board. This is in addition to the CMSIS interface that is included by default.
- TI provides the `DriverLib` which you can download from canvas
- Download it and include in a new folder called `BoardSupportPackage`



UART from driverLib

- Right click on project->**properties->include Options** (search for this using the search bar).



UART from DriverLib

Nelms Institute for the Connected World
UNIVERSITY of FLORIDA

```
/* Configuratin for UART */
static const eUSCI_UART_Config Uart115200Config =
{
    EUSCI_A_UART_CLOCKSOURCE_SMCLK, // SMCLK Clock Source
    6, // BRDIV
    8, // UCxBRF
    0, // UCxBRS
    EUSCI_A_UART_NO_PARITY, // No Parity
    EUSCI_A_UART_LSB_FIRST, // LSB First
    EUSCI_A_UART_ONE_STOP_BIT, // One stop bit
    EUSCI_A_UART_MODE, // UART mode
    EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION // Oversampling
};
```

UART from driverLib


```c
#include <driverlib.h>

static inline void uartTransmitString(char * s)
{
    /* Loop while not null */
    while(*s)
    {
        MAP_UART_transmitData(EUSCI_A0_BASE, *s++);
    }
}

void uartInit()
{
    /* select the GPIO functionality */
    MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1, GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);

    /* configure the digital oscillator */
    CS_setDCOCenteredFrequency(CS_DCO_FREQUENCY_12);

    /* configure the UART with baud rate 115200 */
    MAP_UART_initModule(EUSCI_A0_BASE, &Uart115200Config);

    /* enable the UART */
    MAP_UART_enableModule(EUSCI_A0_BASE);
}
```

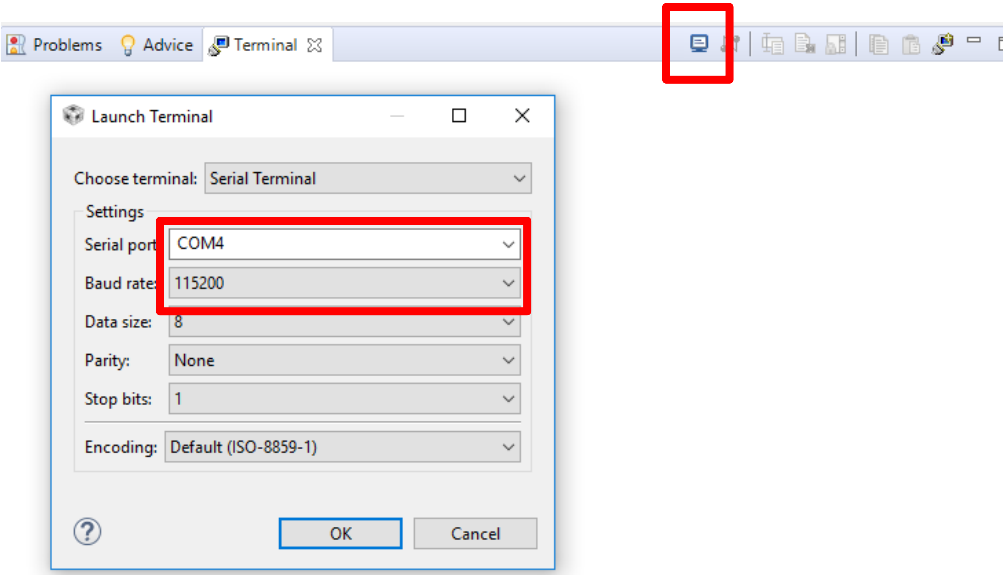
UART from driverLib

```
/**
 * main.c
 */
void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

    uartInit();

    char str[255];
    int checksum = 14;

    snprintf(str, 255, "Hello world. checksum is %d", checksum);

    uartTransmitString(str);

    while(1);
}
```

UART from driverLib

- Use the build icon to compile and the debug icon program the board.

- Use **View->Terminal** to open the terminal (serial port).
- Configure to the COM port that is connected to the port with the baud-rate (115200)



Fletcher16 in C

```c
uint16_t Fletcher16( uint8_t *data, int count )
{
    uint16_t sum1 = 0;
    uint16_t sum2 = 0;
    int index;

    for( index = 0; index < count; ++index )
    {
        sum1 = (sum1 + data[index]) % 255;
        sum2 = (sum2 + sum1) % 255;
    }

    return (sum2 << 8) | sum1;
}
```

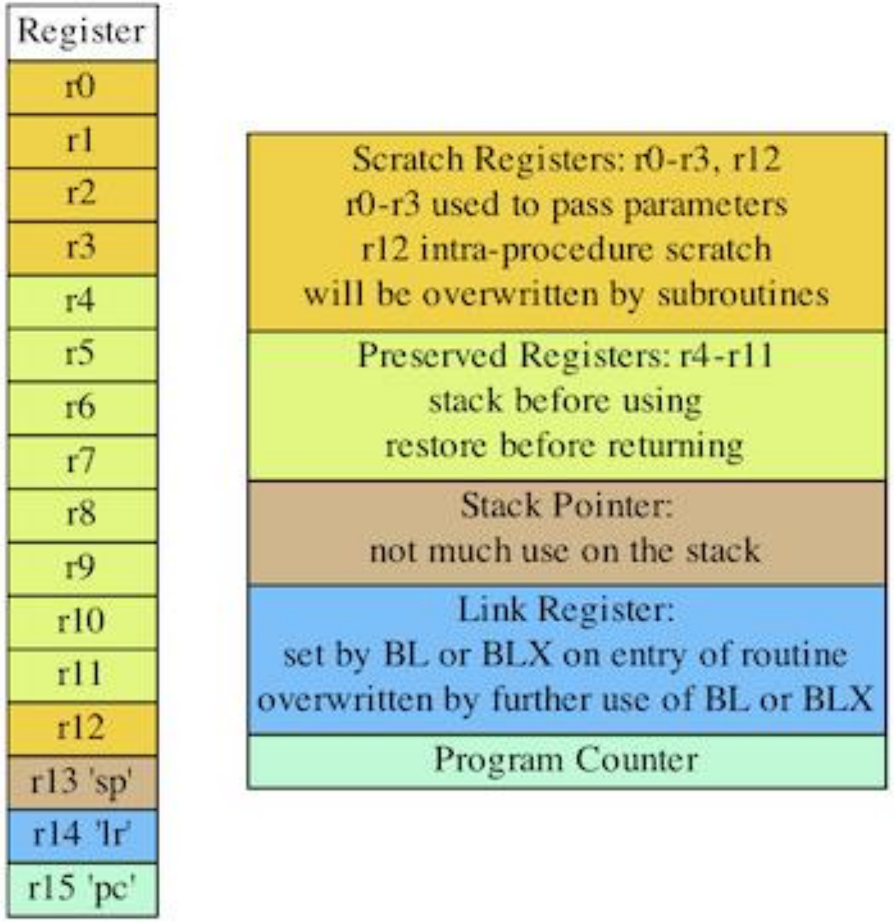
Linking C and Assembly (ASM)

- `fletcher16` is an ASM function
- This takes in a pointer to `uint8_t` and an `int` count and returns a `uint16_t`
- The output of the ASM function is assigned to the returned variable
- `fletcher16` needs to be defined in a .s file that is linked to the program.
- If the file is included in the source directory CCS will hopefully pick it up.

```
extern uint16_t fletcher16(uint8_t *data, int count);

void main(){

    uint16_t retval = 0;
    uint8_t data[] = {1, 2, 3};
    int count = 3;

    retval = fletcher16(data, count);
```

Linking C and Assembly (ASM)

- To make the function C callable, we must pretend we are the compiler and use the registers in the same way
- R0-R3 are the initial registers used to pass parameters into and out of a function. R12 is a special register for intra-procedure communication. These registers must be saved before calling the function **(save-on-call)**
- If more the 4 registers are needed, the stack is utilized
- R4-R11 (and R14) must be saved by the called function **(save-on-entry)**
- If the function returns a value it places it in R0.
- R13-R15 are SP, LR, and PC



http://www.ti.com/lit/ug/spnu151r/spnu151r.pdf

Register Use in the ARM Procedure Call Standard

Linking C and Assembly (ASM)

- `.def` : Functions or variables created that can be accessed from other functions
- `.ref` : If this ASM function needed to access another function/variable it would be specified here
- `.thumb` says that we are using thumb mode
- `.align 2` is needed because when in thumb mode, instructions are 16 bit rather than 32 bits
- `.text` signals start of code section
- `fletcher` is the name of the function and works like a normal lable
- `.asmfunc` : Specify that we are starting a function rather than a label
- BL : call subroutine (store PC in LR)
- BX : jump to register address
- End the function, align again, and end the file

http://downloads.ti.com/docs/esd/SLAU131K/Content/SLAU131K_HTML/assembler_directives.html

```
    .def fletcher16

    .ref Modulus255

    .thumb
    .align 2
    .text

fletcher16:

    .asmfunc

    PUSH {R2 - R7} ; not using R7-R11
    PUSH {LR}

    MOV R7, R0   ; R0 contains first parameter
    MOV R4, R1   ; R1 contains second parameter

    ; ... body of function

    BL Modulus255    ; for calling C function
    MOV R5, R0       ; grabing result

    MOV R0, R5 ; move return value to R0
    POP {LR} ; restore registers
    POP {R2 - R7}

    BX LR ; branch back using LR

    .endasmfunc

    .align
    .end
```

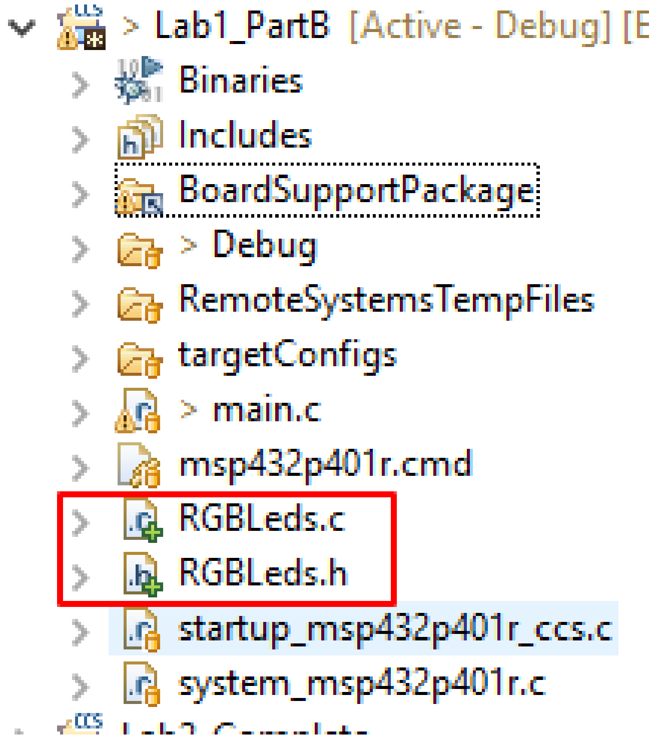
Lab 1 Part B

- Part B: Implement a RGB LED Driver based on LP3943 and I2C communication.
  - Initialize the I2C peripheral(eUSCI_B_2 port).
  - Turn off all LEDs.
  - Implement a function to change the color of 16 LEDs.
  - Demonstration: Display the result of Lab1 Part A with LEDs in hex. You must use all three colors(red, green, blue) to show that you can properly use the LED Driver.
  - DO NOT use the function from DriverLib to initialize the I2C module.
  - READ the LP3943 datasheet and MSP432 I2C tech detail before implementation!
  - READ carefully about the EUSCI_B2 Registers usage.

http://www.ti.com/lit/ds/symlink/lp3943.pdf http://www.ti.com/lit/ug/slau356h/slau356h.pdf

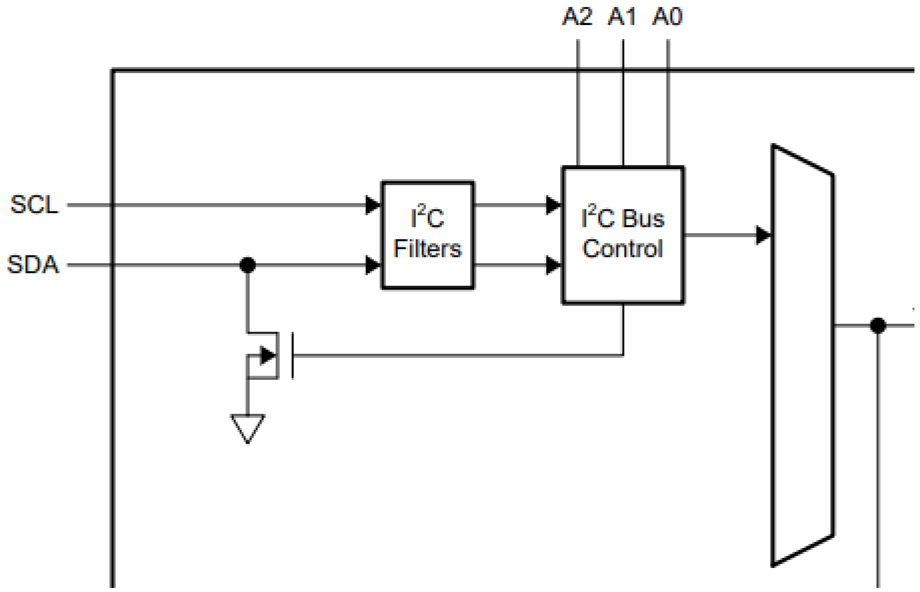
Driver files

- Add new files to implement the LED driver.
- Separate your LED driver with definition(.c file) and implementation(.h file)



Address of I2C Slave

- Base address: 0b1100000(0x60)
- Hard wired ADR2, ADR1, ADR0(A2, A1 A0)
- LEFT LP3943: GND, GND, GND
- MIDDLE LP3943: GND, GND, VCC
- RIGHT LP3943: GND, VCC, VCC
- Address(Left to right): 0x60, 0x61, 0x62



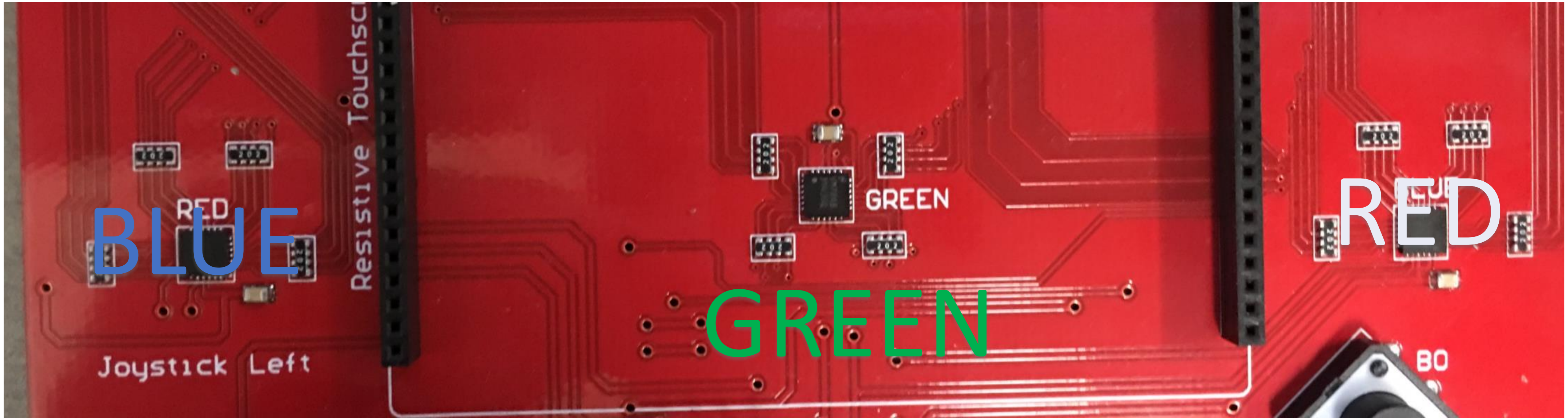
MSB							LSB
ADR6 bit7	ADR5 bit6	ADR4 bit5	ADR3 bit4	ADR2 bit3	ADR1 bit2	ADR0 bit1	R/W bit0
1	1	0	0	X	X	X	0

←— I²C SLAVE address (chip address) —→

Figure 4. Chip Address Byte

Modification of board

- Due to the wire connection issue.
- The colors LP3943 control should be: (Left to right) BLUE, GREEN, RED



Header file

- Use enum to specify the color you are using.
- Put all the initialization code in one init function.
- Implement a function to setup the color of LED. (You can also implement a function to change the PWM mode of LED if you want.)

```c
#ifndef RGBLEDS_H_
#define RGBLEDS_H_

/* Enums for RGB LEDs */
typedef enum device
{
    BLUE = 0,
    GREEN = 1,
    RED = 2
} unit_desig;
1

/*
 * LP3943_ColorSet
 * This function will set the frequencies and PWM duty cycle
 * for each register of the specified unit.
 */
static void LP3943_ColorSet(uint32_t unit, uint32_t PWM_DATA);


/*
 * LP3943_LedModeSet
 * This function will set each of the LEDs to the desired operating
 * mode. The operating modes are on, off, PWM1 and PWM2.
 */
void LP3943_LedModeSet(uint32_t unit, uint16_t LED_DATA);


/*
 * Performs necessary initializations for RGB LEDs
 */
void init_RGBLEDS();

```

Initialization

```c
/*
 * Performs necessary initializations for RGB LEDs
 */
void init_RGBLEDS()
{
    uint16_t UNIT_OFF = 0x0000;

    // Software reset enable
    UCB2CTLW0 = UCSWRST;

    // Initialize I2C master
    // Set as master, I2C mode, Clock sync, SMCLK source, Transmitter
    UCB2CTLW0 |= /* Put your code here */;

    // Set the Fclk as 400khz.
    // Presumes that the SMCLK is selected as source and Fsmclk is 12MHz..
    UCB2BRW = 30;

    // In conjunction with the next line, this sets the pins as I2C mode.
    // (Table found on p160 of SLAS826E)
    // Set P3.6 as UCB2_SDA and 3.7 as UCB2_SLC
    P3SEL0 |= /* Put your code here */
    P3SEL1 &= /* Put your code here */

    // Bitwise anding of all bits except UCSWRST.
    UCB2CTLW0 &= ~UCSWRST;

    LP3943_LedModeSet(RED, UNIT_OFF);
    LP3943_LedModeSet(GREEN, UNIT_OFF);
    LP3943_LedModeSet(BLUE, UNIT_OFF);
}
```

Change the color

- Set each of the LEDs to the desire operating mode.
- You only need to implement ON or OFF in this part.

```
void LP3943_LedModeSet(uint32_t unit, uint16_t LED_DATA)
{
    /*
     * LP3943_LedModeSet
     * This function will set each of the LEDs to the desired operating
     * mode. The operating modes are on, off, PWM1 and PWM2.
     *
     * The units that can be written to are:
     *   UNIT | 0 | Red
     *   UNIT | 1 | Blue
     *   UNIT | 2 | Green
     *
     *   The registers to be written to are:
     *  --------
     *  | LS0 | LED0-3 Selector      |
     *  | LS1 | LED4-7 Selector      |
     *  | LS2 | LED8-11 Selector     |
     *  | LS3 | LED12-16 Selector    |
     *  --------
     *  */
```

Change the color

- Generate data you want send via I2C.
- Set initial slave address since we are master.
- Generate START condition.
- Wait for buffer availability.
- LOOP: Fill TXBUF with the data for the LP3943.
- Wait for buffer availability. B LOOP
- Generate STOP condition.

Change the color

- Registers you will be using in this part:
  - eUSCI_Bx I2C Slave Address Register
  - eUSCI_Bx Control Word Register 0.
  - eUSCI_Bx Transmit Buffer Register.
  - eUSCI_Bx Interrupt Flag Register.

Test with your driver

- A simple example to test your driver.

```
    while(1) {
        LED = 0x0001;
        for (int i = 0; i < 16; i++)
        {
            LP3943_LedModeSet(RED, LED);
            Delay(DELAY_TIME);
            LED <<= 1;
        }

        LP3943_LedModeSet(RED, 0x0000);
        LED = 0x0001;
        for (int i = 0; i < 16; i++)
        {
            LP3943_LedModeSet(BLUE, LED);
            Delay(DELAY_TIME);
            LED <<= 1;
        }
        LP3943_LedModeSet(BLUE, 0x0000);

        LED = 0x0001;
        for (int i = 0; i < 16; i++)
        {
            LP3943_LedModeSet(GREEN, LED);
            Delay(DELAY_TIME);
            LED <<= 1;
        }
        LP3943_LedModeSet(GREEN, 0x0000);
    }

```

Lab 1 : Part C

- Create LED animated color patterns using a timer interrupt (SYSTICK) and a button interrupt
  - Initialize the timer and the button interrupts
  - Write interrupt handlers that will use the LED library to change the state of the LEDs.
  - Enter **sleep mode** instead of software loop when waiting for interrupts

MSP432x Interrupts

- Exception states: (Inactive) -> (Pending) -> (Active) -> (Active & Pending)
- A higher priority Interrupt can preempt a lower priority one.
- A priority equal to the current active ISR is set to active & pending.
- A priority lower than the current active ISR sets the state to pending.
- On entry: The vector is fetched from the vector-table and the context (?) is saved prior to entry to the first level of interrupts.
- If the value of the PC when jumping to the ISR is loaded back into the program-counter the system detects a return from interrupt and sets the exception state back to inactive.

http://www.ti.com/lit/ds/symlink/msp432p401r.pdf http://www.ti.com/lit/ug/slau356h/slau356h.pdf

MSP432x NVIC

- MSP432x NVIC supports 64 external interrupt lines at 8 levels of priority (highest 0).
- Interrupts must be enabled through the NVIC before they can be serviced.
- `ICERx` for disabling and `ISERx` for enabling.
- `ICPRx` for disabling and `ISPRx` for setting and clearing the pending status. (`STIR?`)
- `IABRx` for reading which interrupt is active
- `IPRx` register for setting priorities
- A higher priority Interrupt can preempt a lower priority one.
- The context (?) is saved prior to entry to the first level of interrupts.

http://www.ti.com/lit/ds/symlink/msp432p401r.pdf http://www.ti.com/lit/ug/slau356h/slau356h.pdf

MSP432x SysTick Interrupt

- SysTick can create system interrupts handled by system handlers.
- Its priority is configurable and defaults at 0.
- Its frequency on the MSP432P401R board by default is 3MHz. You can read this value with `CS_getMCLK` from the DriverLib (search for similar functions in CMSIS)

```
#include "msp.h"
#include "driverlib.h"

void SysTick_Handler() {
// called every second
}

void main(void)
{

    SysTick_Config(3000000);
    SysTick_enableInterrupt();

    while(1);
}
```

http://www.ti.com/lit/ds/symlink/msp432p401r.pdf http://www.ti.com/lit/ug/slau356h/slau356h.pdf

MSP432x SysTick Interrupt

- Always read inside of the library functions. If things fail you will know what is going on.

```
__STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks)
{
  if ((ticks - 1UL) > SysTick_LOAD_RELOAD_Msk)
  {
    return (1UL);                                                  /* Reload value impossible */
  }

  SysTick->LOAD  = (uint32_t)(ticks - 1UL);                        /* set reload register */
  NVIC_SetPriority (SysTick_IRQn, (1UL << __NVIC_PRIO_BITS) - 1UL); /* set Priority for Systick Interrupt */
  SysTick->VAL   = 0UL;                                            /* Load the SysTick Counter Value */
  SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
                   SysTick_CTRL_TICKINT_Msk   |
                   SysTick_CTRL_ENABLE_Msk;                        /* Enable SysTick IRQ and SysTick Timer */
  return (0UL);                                                    /* Function successful */
}
```

http://www.ti.com/lit/ds/symlink/msp432p401r.pdf http://www.ti.com/lit/ug/slau356h/slau356h.pdf

MSP432x Port Interrupts

Nelms Institute for the Connected World
UNIVERSITY of FLORIDA

- Four GPIO Ports: Multiple functionalities on each pin.
- `PxIN, PxOUT, PxDIR, PxREN, PxDS, PxSEL0, PxSEL1, PxIES, PxIE, PxIFG` registers related to the port.
- Ports can cause interrupts and events.
- Buttons are connected as follows:
  - B0 : P4.4
  - B1 : P5.5
  - B2 : P5.4
  - B3 : P4.5

```
P4->DIR &= ~BIT4;
P4->IFG &= ~BIT4;// P4.4 IFG cleared
P4->IE |= BIT4;  // Enable interrupt on P4.4
P4->IES |= BIT4; // high-to-low transition
P4->REN |= BIT4; // Pull-up resister
P4->OUT |= BIT4; // Sets res to pull-up
```

http://www.ti.com/lit/ds/symlink/msp432p401r.pdf

http://www.ti.com/lit/ug/slau356h/slau356h.pdf

MSP432x Port Interrupts

```
void main(void)
{
    P4->DIR &= ~BIT4;
    P4->IFG &= ~BIT4;// P4.4 IFG cleared
    P4->IE |= BIT4;  // Enable interrupt on P4.4
    P4->IES |= BIT4; // high-to-low transition
    P4->REN |= BIT4; // Pull-up resister
    P4->OUT |= BIT4; // Sets res to pull-up

    NVIC_EnableIRQ(PORT4_IRQn);

    while(1) {}
}
```

```
void PORT4_IRQHandler(void){
    P4->IFG &= ~BIT4; // must
    clear IFG flag
    // reading PxIV will
        automatically clear IFG

    // rest of ISR
}
```

http://www.ti.com/lit/ds/symlink/msp432p401r.pdf

http://www.ti.com/lit/ug/slau356h/slau356h.pdf

MSP432x Low Power Modes (LPMs)

Nelms Institute for the Connected World
UNIVERSITY of FLORIDA

- Interrupt Driven Programming Embedded System design paradigm:
  - All code is in interrupts.
  - Main simply does the initialization
  - Power down CPU when waiting for interrupts (can drastically improve battery life)
- MSP432 modes :
  - LMP0 : shallowest sleep. CPU clock stops. Peripherals timers and ports still running (400 ~ 500 uA at 3Mhz)
  - LMP3,4: All high frequency clock consumers are disabled. Only RTC and WDT running. longer wake-up time. (0.5 ~ 2 uA at 3Mhz)
- On interrupts the CPU wakes up and goes back to sleep once the ISR is done

http://www.ti.com/lit/ds/symlink/msp432p401r.pdf http://www.ti.com/lit/ug/slau356h/slau356h.pdf

MSP432x LPM Mode


```
void main(void)
{
    P4->DIR &= ~BIT4;
    P4->IFG &= ~BIT4;// P4.4 IFG cleared
    P4->IE |= BIT4;  // Enable interrupt on P4.4
    P4->IES |= BIT4; // high-to-low transition
    P4->REN |= BIT4; // Pull-up resister
    P4->OUT |= BIT4; // Sets res to pull-up

    NVIC_EnableIRQ(PORT4_IRQn);

    while(1) {}
}
```

```
void main(void)
{
    P4->DIR &= ~BIT4;
    P4->IFG &= ~BIT4;// P4.4 IFG cleared
    P4->IE |= BIT4;  // Enable interrupt on P4.4
    P4->IES |= BIT4; // high-to-low transition
    P4->REN |= BIT4; // Pull-up resister
    P4->OUT |= BIT4; // Sets res to pull-up

    NVIC_EnableIRQ(PORT4_IRQn);

    PCM_gotoLPM0(); // enter LPM mode
}
```

http://www.ti.com/lit/ds/symlink/msp432p401r.pdf

http://www.ti.com/lit/ug/slau356h/slau356h.pdf