Main.cpp

```
#include "tasks.hpp"
#include "examples/examples.hpp"
#include "periodic_scheduler/periodic_callback.h"
#include "uart2.hpp"
#include "uart3.hpp"
#include "utilities.h"
#include "uart0_min.h"
#include "handlers.hpp"
// #include "io.hpp"
#include "printf_lib.h"
#include "stdio.h"
#include "LabGPIO.hpp"
#include "interrupts.hpp"
#include "adcDriver.hpp"
#include <stdint.h>
#include "pwmDriver.hpp"



PWMDriver B;
ADCDriver A;
//ADC_PIN Pin25 = ADC_PIN_0_25;

int main(void)
{




    int test;



    B.pwmSelectPin(B.PWM_PIN_2_0);
    B.pwmInitSingleEdgeMode(0xF4240);
    A.adcInitBurstMode();
    //A.adcSelectPin(Pin25);
    A.adcSelectPin(A.ADC_PIN_0_26);

    //test = (int)(A.readADCVoltageByChannel(3));
    // xTaskCreate(LabGPIO, (const char*)"task_one_code",
STACK_BYTES(2048), 0, 1, 0);
    // vTaskStartScheduler();
```

```
    //LPC_PINCON->PINSEL1 |= (1<<18);
    while(1){

        test = (int)(A.readADCVoltageByChannel(3));
        // value = (uint16_t)LPC_ADC->ADGDR;
        // value = (value>>4);
        // test = (int)(value);
        // for (int i = 0; i < 256; ++i)
        // {
        //     /* code */
        // }
        printf("Value is: %i\n", test);
        //vTaskDelay(1000);
    }




        return 0;

  }
```

adcDriver.hpp

```
#ifndef ADC_DRIVER_H_
#define ADC_DRIVER_H_

#include <stdio.h>
#include <stdint.h>
#include "io.hpp"


class ADCDriver
{
public:
    enum ADC_PIN
    {
        ADC_PIN_0_25,       // AD0.2 <-- Light Sensor -->
        ADC_PIN_0_26,       // AD0.3
        ADC_PIN_1_30,       // AD0.4
        ADC_PIN_1_31,       // AD0.5

        /* These ADC channels are compromised on the SJ-One,
hence you do not need to support them
        ADC_PIN_0_23 = 0,   // AD0.0
```

```
        ADC_PIN_0_24,       // AD0.1
        ADC_PIN_0_3,        // AD0.6
        ADC_PIN_0_2         // AD0.7
        */
    };

    // Nothing needs to be done within the default constructor
    ADCDriver();

    /**
    * 1) Powers up ADC peripheral
    * 2) Set peripheral clock
    * 2) Enable ADC
    * 3) Select ADC channels
    * 4) Enable burst mode
    */
    void adcInitBurstMode();

    /**
    * 1) Selects ADC functionality of any of the ADC pins that
are ADC capable
    *
    * @param adc_pin_arg is the ADC_PIN enumeration of the
desired pin.
    *
    * WARNING: For proper operation of the SJOne board, do NOT
configure any pins
    *           as ADC except for 0.26, 1.31, 1.30
    */
    void adcSelectPin(ADC_PIN adc_pin_arg);

    /**
    * 1) Returns the voltage reading of the 12bit register of a
given ADC channel
    *
    * @param adc_channel_arg is the number (0 through 7) of the
desired ADC channel.
    */
    float readADCVoltageByChannel(uint8_t adc_channel_arg);
};

#endif
```

adcDriver.cpp

```
#include "adcDriver.hpp"
```

```
#include "io.hpp"
#include "LPC17xx.h"
#include "lpc_isr.h"
#include <iostream>



// Nothing needs to be done within the default constructor
ADCDriver::ADCDriver(){}

/**
* 1) Powers up ADC peripheral
* 2) Set peripheral clock
* 2) Enable ADC
* 3) Select ADC channels
* 4) Enable burst mode
*/
void ADCDriver::adcInitBurstMode(){
    LPC_SC->PCONP |= (1<<12);
    LPC_ADC->ADCR |= (1<<21);
    LPC_SC->PCLKSEL0 |= (1<<24);


    //Clear bits first
    LPC_PINCON->PINSEL1 &=~ (3<<18);
    LPC_PINCON->PINSEL1 &=~ (3<<20);
    LPC_PINCON->PINSEL3 &=~ (3<<28);
    LPC_PINCON->PINSEL3 &=~ (3<<30);

    LPC_PINCON->PINMODE1 |= (3<<20);

    LPC_ADC->ADINTEN &= ~(1<<8);            //Clear ADGINTEN
    LPC_ADC->ADCR |= (1<<16);               //enable burst

}

/**
* 1) Selects ADC functionality of any of the ADC pins that are
ADC capable
*
* @param adc_pin_arg is the ADC_PIN enumeration of the desired
pin.
*
* WARNING: For proper operation of the SJOne board, do NOT
configure any pins
*           as ADC except for 0.26, 1.31, 1.30
*/
```

```
void ADCDriver::adcSelectPin(ADC_PIN adc_pin_arg){
    switch(adc_pin_arg){
        case ADC_PIN_0_25:
            LPC_PINCON->PINSEL1 |= (1<<18);
            break;
        case ADC_PIN_0_26:
            LPC_PINCON->PINSEL1 |= (1<<20);
            break;
        case ADC_PIN_1_30:
            LPC_PINCON->PINSEL3 |= (3<<28);
            break;
        case ADC_PIN_1_31:
            LPC_PINCON->PINSEL3 |= (3<<30);
            break;
    }
}

/**
* 1) Returns the voltage reading of the 12bit register of a
given ADC channel
*
* @param adc_channel_arg is the number (0 through 7) of the
desired ADC channel.
*/
float ADCDriver::readADCVoltageByChannel(uint8_t
adc_channel_arg){
    uint16_t value = 0x0000;
    //  LPC_ADC->ADGDR |= (adc_channel_arg<<24);
    switch(adc_channel_arg){
        case 0:
            value = (uint16_t)LPC_ADC->ADDR0;
            break;
        case 1:
            value = (uint16_t)LPC_ADC->ADDR1;
            break;
        case 2:
            value = (uint16_t)LPC_ADC->ADDR2;
            break;
        case 3:
            value = (uint16_t)LPC_ADC->ADDR3;
            break;
        case 4:
            value = (uint16_t)LPC_ADC->ADDR4;
            break;
        case 5:
            value = (uint16_t)LPC_ADC->ADDR5;
            break;
```

```
        case 6:
            value = (uint16_t)LPC_ADC->ADDR6;
            break;
        case 7:
            value = (uint16_t)LPC_ADC->ADDR7;
            break;
    }
        value = (value>>4);

        // for (int i = 0; i < 256; ++i)
        // {
        //     /* code */
        // }
        //printf("Value is: %i\n", test);
        return value;
}
```

pwmDriver.hpp

```
#ifndef PWM_DRIVER_H_
#define PWM_DRIVER_H_

#include <stdint.h>

class PWMDriver
{
public:
    enum PWM_PIN
    {
        PWM_PIN_2_0,    // PWM1.1
        PWM_PIN_2_1,    // PWM1.2
        PWM_PIN_2_2,    // PWM1.3
        PWM_PIN_2_3,    // PWM1.4
        PWM_PIN_2_4,    // PWM1.5
        PWM_PIN_2_5,    // PWM1.6
    };

    /// Nothing needs to be done within the default constructor
    PWMDriver();

    /**
    * 1) Select PWM functionality on all PWM-able pins.
    */
    void pwmSelectAllPins();

     /**
    * 1) Select PWM functionality of pwm_pin_arg
```

```
    *
    * @param pwm_pin_arg is the PWM_PIN enumeration of the
desired pin.
    */
    void pwmSelectPin(PWM_PIN pwm_pin_arg);

    /**
    * Initialize your PWM peripherals.  See the notes here:
    * http://books.socialledge.com/books/embedded-drivers-real-
time-operating-systems/page/pwm-%28pulse-width-modulation%29
    *
    * In general, you init the PWM peripheral, its frequency,
and initialize your PWM channels and set them to 0% duty cycle
    *
    * @param frequency_Hz is the initial frequency in Hz.
    */
    void pwmInitSingleEdgeMode(uint32_t frequency_Hz);

     /**
    * 1) Convert duty_cycle_percentage to the appropriate match
register value (depends on current frequency)
    * 2) Assign the above value to the appropriate MRn register
(depends on pwm_pin_arg)
    *
    * @param pwm_pin_arg is the PWM_PIN enumeration of the
desired pin.
    * @param duty_cycle_percentage is the desired duty cycle
percentage.
    */
     void setDutyCycle(PWM_PIN pwm_pin_arg, float
duty_cycle_percentage);

     /**
    * Optional:
    * 1) Convert frequency_Hz to the appropriate match register
value
    * 2) Assign the above value to MR0
    *
    * @param frequency_hz is the desired frequency of all pwm
pins
    */
     void setFrequency(uint32_t frequency_Hz);
};

#endif
```

pwmDriver.cpp

```
#include <stdint.h>
#include "pwmDriver.hpp"
#include "io.hpp"
#include "LPC17xx.h"
#include <iostream>


    /// Nothing needs to be done within the default constructor
PWMDriver::PWMDriver() {}

/**
* 1) Select PWM functionality on all PWM-able pins.
*/
void PWMDriver::pwmSelectAllPins(){
    LPC_PINCON->PINSEL4 &=~ (0xFFF);
    LPC_PINCON->PINSEL4 |= (0x555);
}

/**
* 1) Select PWM functionality of pwm_pin_arg
*
* @param pwm_pin_arg is the PWM_PIN enumeration of the desired
pin.
*/
void PWMDriver::pwmSelectPin(PWM_PIN pwm_pin_arg){
    LPC_PINCON->PINSEL4 &=~ (0xFFF);
    LPC_PINCON->PINSEL4 |= (1<<pwm_pin_arg * 2);
}

/**
* Initialize your PWM peripherals.  See the notes here:
* http://books.socialledge.com/books/embedded-drivers-real-time-
operating-systems/page/pwm-%28pulse-width-modulation%29
*
* In general, you init the PWM peripheral, its frequency, and
initialize your PWM channels and set them to 0% duty cycle
*
* @param frequency_Hz is the initial frequency in Hz.
*/
void PWMDriver::pwmInitSingleEdgeMode(uint32_t frequency_Hz){
    LPC_SC->PCONP |= (1<<6);
    LPC_SC->PCLKSEL0 |= (1<<12);
    LPC_PWM1->MCR |= (1<< 1);
    LPC_PWM1->TCR |= (0xA<<0);
```

```
    LPC_PWM1->CTCR &=~ (0xF<<0);
    LPC_PWM1->PCR |= (0x3F<<9);

    LPC_PWM1->PR |= (0xFFFFFFFF<<0);

    uint32_t MR0Value = 0xF4240/frequency_Hz;
    //LPC_PWM1->MR0 |= (MR0Value<<0);
    LPC_PWM1->MR0 &=~ (0xFFFFFFFF<<0);
    LPC_PWM1->MR0 |= (0xF4200<<0);
    LPC_PWM1->MR1 &=~ (0xFFFFFFFF<<0);
    LPC_PWM1->MR1 |= (0xF4000<<0);

}

/**
* 1) Convert duty_cycle_percentage to the appropriate match
register value (depends on current frequency)
* 2) Assign the above value to the appropriate MRn register
(depends on pwm_pin_arg)
*
* @param pwm_pin_arg is the PWM_PIN enumeration of the desired
pin.
* @param duty_cycle_percentage is the desired duty cycle
percentage.
*/
void PWMDriver::setDutyCycle(PWM_PIN pwm_pin_arg, float
duty_cycle_percentage){

}

/**
* Optional:
* 1) Convert frequency_Hz to the appropriate match register
value
* 2) Assign the above value to MR0
*
* @param frequency_hz is the desired frequency of all pwm pins
*/
void PWMDriver::setFrequency(uint32_t frequency_Hz){

}
```