**ADC.hpp**

```cpp
#ifndef LPC_ADC_H__
#define LPC_ADC_H__

#include "LPC17xx.h"
#include <stdint.h>
#include "io.hpp"
#include "printf_lib.h"

class Lab_ADC
{
public:
    enum Pin
    {
        k0_25,        // AD0.2 <-- Light Sensor -->
        k0_26,        // AD0.3
        k1_30,        // AD0.4
        k1_31,        // AD0.5
    };

    Lab_ADC();
    void AdcInitBurstMode();
    void AdcSelectPin(Pin pin);
    float ReadAdcVoltageByChannel(uint8_t channel);
};

#endif
```

**ADC.cpp**

```
#include "ADC.hpp"

 /**
    * 1) Powers up ADC peripheral
    * 2) Set peripheral clock
    * 2) Enable ADC
    * 3) Select ADC channels
    * 4) Enable burst mode
    */
Lab_ADC :: Lab_ADC(){}
void Lab_ADC :: AdcInitBurstMode()
{
    LPC_ADC->ADCR = ~(0xffffffff);
    LPC_SC->PCONP |= (1 << 12);                          // Enable ADC
power PCADC
    LPC_ADC->ADCR |= (1 << 21);                          // Enable
power to ADC through PDN bit
    LPC_SC->PCLKSEL0 |= ((1 << 24) | (1 << 25));         // Peripheral
Clock div
    LPC_ADC->ADCR |= (4 << 8) | (1 << 16);               // Divide by 4
to get clock less than 13Mhz and enable burst mode

}

    /**
    * 1) Selects ADC functionality of any of the ADC pins that are ADC
capable
    *
    * @param pin is the LabAdc::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 Lab_ADC :: AdcSelectPin(Pin pin)
{
    if(pin == k0_25)
    {
        LPC_PINCON->PINSEL1 |= (1 << 18);
        LPC_PINCON->PINMODE1 |= (1 << 19);
        LPC_GPIO0->FIODIR &= ~(1 << 25);
    }
    if(pin == k0_26)
    {
        LPC_PINCON->PINSEL1 |= (1 << 20);
```

```
        LPC_PINCON->PINMODE1 |= (1 << 21);
        LPC_GPIO0->FIODIR &= ~(1 << 26);
    }
    if(pin == k1_30)
    {
        LPC_PINCON->PINSEL3 |= (3 << 28);
        LPC_PINCON->PINMODE3 |= (1 << 29);
        LPC_GPIO1->FIODIR &= ~(1 << 30);
    }
    if(pin == k1_31)
    {
        LPC_PINCON->PINSEL3 |= (3 << 30);
        LPC_PINCON->PINMODE3 |= (1 << 31);
        LPC_GPIO1->FIODIR &= ~(1 << 31);
    }

}

    /**
    * 1) Returns the voltage reading of the 12bit register of a given
ADC channel
    * You have to convert the ADC raw value to the voltage value
    * @param channel is the number (0 through 7) of the desired ADC
channel.
    */
float Lab_ADC ::  ReadAdcVoltageByChannel(uint8_t channel)
{
    LPC_ADC->ADCR |= (1 << channel);
    uint16_t conv_val = 0;

    switch(channel) {
        case 2:
            conv_val = ((LPC_ADC->ADDR2 >> 4) & 0x0fff);
            break;
        case 3:
            conv_val = ((LPC_ADC->ADDR3 >> 4) & 0x0fff);
            break;
        case 4:
            conv_val = ((LPC_ADC->ADDR4 >> 4) & 0x0fff);
            break;
        case 5:
            conv_val = ((LPC_ADC->ADDR5 >> 4) & 0x0fff);
            break;
        default:
            u0_dbg_printf("Wrong ADC pin selection");
    }
```

```
    float op_volt = ((conv_val * 3.3)/4096);

    return op_volt;
}
```

**PWM.hpp**

```
#ifndef LPC_PWM_H__
#define LPC_PWM_H__

#include "LPC17xx.h"
#include <stdint.h>
#include "io.hpp"
#include "printf_lib.h"


class Lab_PWM
{
 public:
    enum PWM_PIN
    {
        k2_0,    // PWM1.1
        k2_1,    // PWM1.2
        k2_2,    // PWM1.3
        k2_3,    // PWM1.4
        k2_4,    // PWM1.5
        k2_5,    // PWM1.6
    };

    Lab_PWM();
    void PwmSelectAllPins();
    void PwmSelectPin(PWM_PIN pwm_pin_arg);
    void PwmInitSingleEdgeMode(uint32_t frequency_Hz);
    void SetDutyCycle(PWM_PIN pwm_pin_arg, float
duty_cycle_percentage);
    void SetFrequency(uint32_t frequency_Hz);
};

#endif
```

**PWM.cpp**

```
#include "PWM.hpp"

Lab_PWM :: Lab_PWM() {
}

void Lab_PWM :: PwmSelectAllPins()
{
    LPC_PINCON->PINSEL4 |= (0x0555 << 0);
    LPC_PINCON->PINMODE4 |= (0x0aaa << 0);
}

void Lab_PWM :: PwmSelectPin(PWM_PIN pwm_pin_arg)
{
    if(pwm_pin_arg <=6)
    {
        if (pwm_pin_arg <=6)
        LPC_PINCON->PINSEL4 |= (1 << (pwm_pin_arg * 2));
        LPC_PINCON->PINMODE4 |= (2 << (pwm_pin_arg * 2));
    }
    else u0_dbg_printf("Error0: Pin not found\n");
}

void Lab_PWM :: PwmInitSingleEdgeMode(uint32_t frequency_Hz)
{
    LPC_SC->PCONP |= (1 << 6);
    LPC_SC->PCLKSEL0 |= (3 << 12);               //Clock for PWM
peripheral = 6MHz
    LPC_PWM1->PR |= ((6000000/frequency_Hz) - 1);
}

void Lab_PWM :: SetDutyCycle(PWM_PIN pwm_pin_arg, float
duty_cycle_percentage)
{
    if(duty_cycle_percentage <1)
    {
        duty_cycle_percentage = 1;
    }
    LPC_PWM1->PCR |= (1 << (pwm_pin_arg + 9));  //Configure channel 2
to single edge & enable the channel 2 to be output

    if(pwm_pin_arg == k2_0)
    {
        LPC_PWM1->MR1 = duty_cycle_percentage;
    }
```

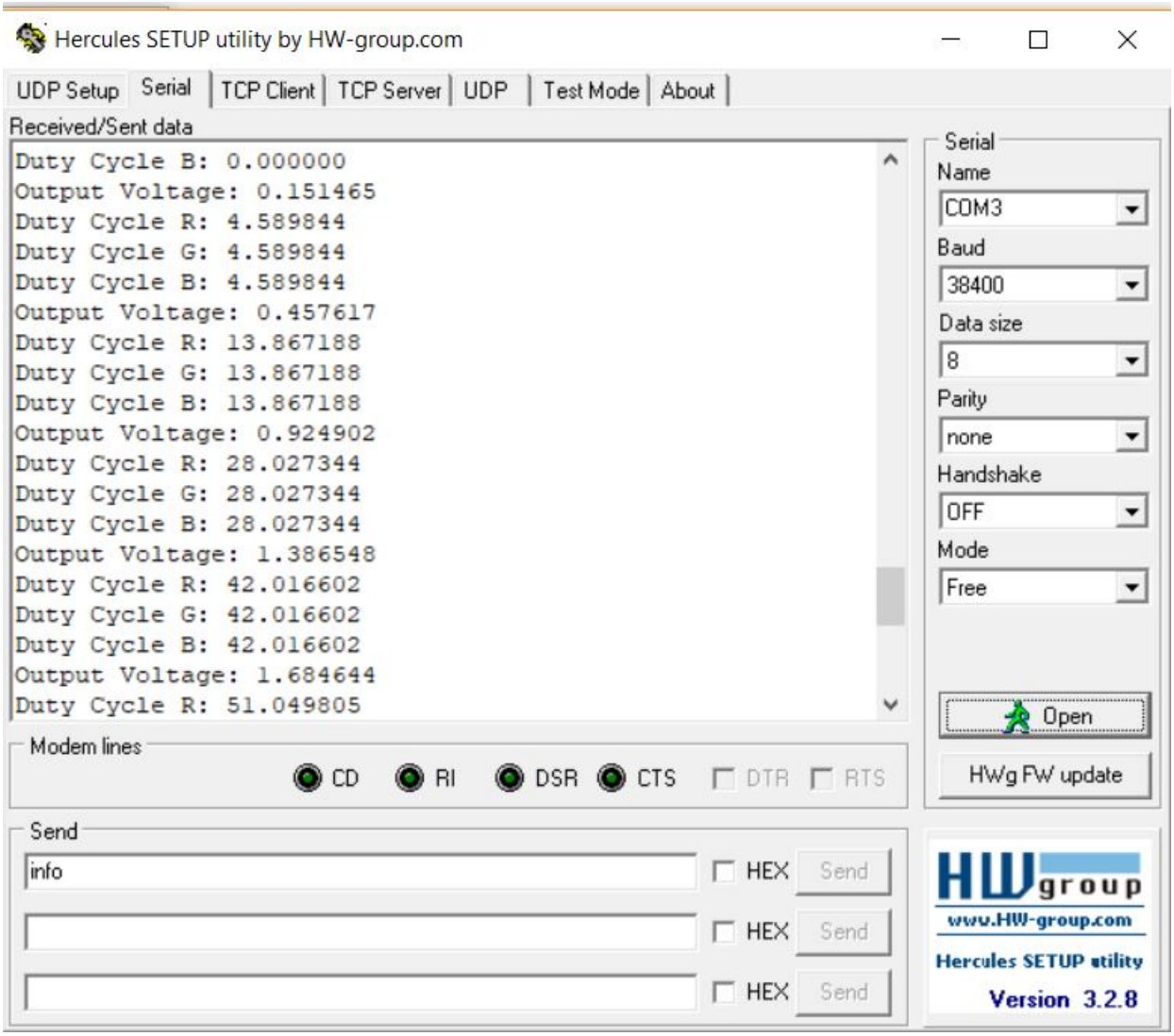
```
    else if(pwm_pin_arg == k2_1)
    {
        LPC_PWM1->MR2 = duty_cycle_percentage;
    }
    else if(pwm_pin_arg == k2_2)
    {
        LPC_PWM1->MR3 = duty_cycle_percentage;
    }
    else if(pwm_pin_arg == k2_3)
    {
        LPC_PWM1->MR4 = duty_cycle_percentage;
    }
    else if(pwm_pin_arg == k2_4)
    {
        LPC_PWM1->MR5 = duty_cycle_percentage;
    }
    else if(pwm_pin_arg == k2_5)
    {
        LPC_PWM1->MR6 = duty_cycle_percentage;
    }
    else u0_dbg_printf("Error in PWM_pin selection");

    LPC_PWM1->TCR = (1 << 1);
    LPC_PWM1->TCR = (1 << 0) | (1 << 3);      //Enable PWM mode(use
after setting up the PWM Match register)
}

void Lab_PWM :: SetFrequency(uint32_t frequency_Hz)
{
    LPC_PWM1->MCR = (1 << 1);
//Enable: Reset TC when TC = MR0
    LPC_PWM1->MR0 = ((6000000/(LPC_PWM1->PR + 1))/frequency_Hz);
}
```

**Main.cpp**

```
#include <stdint.h>
#include "LPC17xx.h"
#include "utilities.h"
#include "ADC.hpp"
#include "PWM.hpp"
#include "FreeRTOS.h"
#include "Task.h"

Lab_ADC ADC;
Lab_PWM PWM1;
Lab_PWM PWM2;
Lab_PWM PWM3;

float value_MR = 0, convv = 0,duty_cycle = 0;

void vADC_Task(void *pvParameter)
{
    while(1)
    {
        convv = ADC.ReadAdcVoltageByChannel(3);
        vTaskDelay(2);
        duty_cycle = (convv*100)/3.3;
        value_MR = (duty_cycle*LPC_PWM1->MR0)/100;
        PWM1.SetDutyCycle(PWM1.k2_1, value_MR);
        PWM2.SetDutyCycle(PWM1.k2_2, value_MR);
        PWM3.SetDutyCycle(PWM1.k2_3, value_MR);

        vTaskDelay(200);
    }
}

void vDisplay_Task(void *pvParameter)
{
    while(1)
    {
        u0_dbg_printf("Output Voltage: %f\n", convv);
        u0_dbg_printf("Duty Cycle R: %f\n", duty_cycle);
        u0_dbg_printf("Duty Cycle G: %f\n", duty_cycle);
        u0_dbg_printf("Duty Cycle B: %f\n", duty_cycle);

        vTaskDelay(1000);
    }
}
```

```
int main(void)
{
   // PWM1.PwmSelectAllPins();
    PWM1.PwmInitSingleEdgeMode(2000000);
    PWM1.SetFrequency(500);
    PWM2.PwmInitSingleEdgeMode(2000000);
    PWM2.SetFrequency(2000);
    PWM3.PwmInitSingleEdgeMode(2000000);
    PWM3.SetFrequency(10000);
    PWM1.PwmSelectPin(PWM1.k2_1);
    PWM2.PwmSelectPin(PWM1.k2_2);
    PWM3.PwmSelectPin(PWM1.k2_3);

    ADC.AdcInitBurstMode();
    ADC.AdcSelectPin(ADC.k0_26);

    xTaskCreate(vADC_Task, (const char*) "Read_data", 512, NULL, 2,
NULL);
    xTaskCreate(vDisplay_Task, (const char*) "Read_data", 512, NULL,
1, NULL);
    vTaskStartScheduler();
    return 0;
}
```

OUTPUT:


Hercules SETUP utility by HW-group.com
UDP Setup
Serial
TCP Client
TCP Server
UDP
Test Mode
About
Received/Sent data
Duty Cycle B: 0.000000
Output Voltage: 0.151465
Duty Cycle R: 4.589844
Duty Cycle G: 4.589844
Duty Cycle B: 4.589844
Output Voltage: 0.457617
Duty Cycle R: 13.867188
Duty Cycle G: 13.867188
Duty Cycle B: 13.867188
Output Voltage: 0.924902
Duty Cycle R: 28.027344
Duty Cycle G: 28.027344
Duty Cycle B: 28.027344
Output Voltage: 1.386548
Duty Cycle R: 42.016602
Duty Cycle G: 42.016602
Duty Cycle B: 42.016602
Output Voltage: 1.684644
Duty Cycle R: 51.049805
Serial
Name
COM3
Baud
38400
Data size
8
Parity
none
Handshake
OFF
Mode
Free
Open
HWg FW update
Modem lines
CD
RI
DSR
CTS
DTR
RTS
Send
info
HEX
Send
HEX
Send
HEX
Send
HW group
www.HW-group.com
Hercules SETUP utility
Version 3.2.8