Nickolas Schiffer

CMPE 146 S19

Lab: ADC_PWM

```
#include <stdlib.h>
#include <LPC17xx.h>
#include <tasks.hpp>
#include <stdio.h>
#include "ADC/adcDriver.hpp"
#include "PWM/pwmDriver.hpp"
#include "GPIO/GPIOInterrupt.hpp"
#include <math.h>

typedef bool Mode;

#define VREF 3.3
#define MODE   bool
#define NORMAL false
#define EC     true

LabAdc::ADC_Channel pot_channel        = LabAdc::channel_3;
LabAdc::ADC_Channel light_sens_channel = LabAdc::channel_2;
LabAdc::Pin pot_pin        = LabAdc::k0_26;
LabAdc::Pin light_sens_pin = LabAdc::k0_25;

LabPwm::PWM_Pin red_pin   = LabPwm::k2_0;
LabPwm::PWM_Pin green_pin = LabPwm::k2_1;
LabPwm::PWM_Pin blue_pin  = LabPwm::k2_2;

struct sw{
    uint8_t port = 2;
    uint8_t pin  = 7;
}sw1;

typedef enum state{
    Normal,
    RGBPulse,
    LightSense,
    KnobRGB
};

state mode = Normal;





float duty_cycle_red, duty_cycle_green, duty_cycle_blue;


inline float map(float x, float in_min, float in_max, float out_min, float
out_max){
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

/*
 * Upon Interrupt, this method changes the operation mode
 */
void vSwitchMode(){
    switch (mode){
        case Normal:
            mode = RGBPulse;
            break;
        case RGBPulse:
            mode = LightSense;
            break;
        case LightSense:
            mode = KnobRGB;
            break;
        case KnobRGB:
            mode = Normal;
            break;
        default:
            break;
    }
    return;
}

/*
 *  **Extra Credit**
 * Maps Light Sensor or Potentiometer into point in rainbow.
 * Rainbow is calculated via 3 offset sine waves.
 * Voltage value is mapped from 0-3.3v to an angle 0-360 degrees to be used in sine function.
 */
void vLightRGB(void *pvParameters){
    auto pwm = LabPwm();
    auto adc = LabAdc();
    adc.AdcSelectPin(pot_pin);
    adc.AdcSelectPin(light_sens_pin);

    float voltage;
    double angle = 0;

    while(1){
        while((mode !=LightSense) && (mode != KnobRGB)){
            vTaskDelay(100);
        }
        if (mode == LightSense)
            voltage = adc.ReadAdcVoltageByChannel(light_sens_channel);
        else
            voltage = adc.ReadAdcVoltageByChannel(pot_channel);
        angle = (double)map(voltage, 0, 3.3, 0, 360);
        duty_cycle_red   = (float)((1*(sin((double)(angle/180*M_PI))+1))/2);
        duty_cycle_green = (float)((1*(sin((double)(angle/180*M_PI+((double)(2.f/3.f)*M_PI)))+1))/2);
        duty_cycle_blue  = (float)((1*(sin((double)(angle/180*M_PI+((double)(4.f/3.f)*M_PI)))+1))/2);
        pwm.SetDutyCycle(red_pin,duty_cycle_red);
        pwm.SetDutyCycle(green_pin,duty_cycle_green);
        pwm.SetDutyCycle(blue_pin,duty_cycle_blue);
        vTaskDelay(10);


        }

}

/*
 *  **Extra Credit**
 * Rainbow is calculated via 3 offset sine waves.
 * for-loop loops through angle 0-360 degrees which covers the rainbow.
 * Potentiometer adjusts the step size for iterating through the circle, therefore increasing the
 * rainbow speed.
 */
void vRGBTEST(void *pvParamters){
    auto pwm = LabPwm();
    auto adc = LabAdc();
    adc.AdcSelectPin(pot_pin);

    float voltage;
    uint16_t delay = 1;

    while(1){
        while(mode != RGBPulse){
            vTaskDelay(100);
        }
        for (double x = 0; x < 360; x+=delay){
            voltage = adc.ReadAdcVoltageByChannel(pot_channel);
            delay = (uint16_t)map(voltage, 0, 3.3, 1, 50);
            duty_cycle_red   = (float)((1*(sin((double)(x/180*M_PI))+1))/2);
            duty_cycle_green = (float)((1*(sin((double)(x/180*M_PI+((double)(1.5)*M_PI)))+1))/2);
            duty_cycle_blue  = (float)((1*(sin((double)(x/180*M_PI+((double)
(0.5)*M_PI)))+1))/2);
      pwm.SetDutyCycle(red_pin,duty_cycle_red);
      pwm.SetDutyCycle(green_pin,duty_cycle_green);
      pwm.SetDutyCycle(blue_pin,duty_cycle_blue);
      vTaskDelay(10);
    }


    }


}

/*
 * Prints operation mode,
 * Voltage,
 * and duty cycles
 */
void vPrintTask(void *pvParamters){
  auto adc = LabAdc();
  adc.AdcSelectPin(pot_pin);
  float voltage;

  while (1){
    voltage = adc.ReadAdcVoltageByChannel(pot_channel);
    printf("Mode: %d\nvoltage: %f\nr_ds: %f\ng_ds: %f\nb_ds: %f\n\n", mode, voltage, duty_cycle_red, duty_cycle_green, duty_cycle_blue);
    vTaskDelay(1000);
  }
}

/*
 * Sets duty cycle of voltage to VREF.
 */
void vPWMADCTEST(void *pvParameters){
  auto pwm = LabPwm();
  auto adc = LabAdc();

  adc.AdcSelectPin(pot_pin);
  adc.AdcInitBurstMode();

  pwm.PwmSelectPin(red_pin);
  pwm.PwmSelectPin(green_pin);
  pwm.PwmSelectPin(blue_pin);
  pwm.PwmInitSingleEdgeMode(100);
  float voltage;

```

```
    while (1){
        while(mode != Normal){
            vTaskDelay(100);
        }

        voltage = adc.ReadAdcVoltageByChannel(pot_channel);
        duty_cycle_red = duty_cycle_green = duty_cycle_blue = (float)(voltage / (float)VREF);
        pwm.SetDutyCycle(red_pin, duty_cycle_red);
        pwm.SetDutyCycle(green_pin, duty_cycle_green);
        pwm.SetDutyCycle(blue_pin, duty_cycle_blue);
        vTaskDelay(10);
    }


}
/*
 * Detects switch button interrupt. Used to change operation mode.
 */
void Eint3Handler(){
    GPIOInterrupt *interruptHandler = GPIOInterrupt::getInstance();
    interruptHandler->HandleInterrupt();
}

int main(){
    scheduler_add_task(new terminalTask(PRIORITY_HIGH));


    GPIOInterrupt *gpio_interrupts = GPIOInterrupt::getInstance();
    gpio_interrupts->Initialize();
    gpio_interrupts->AttachInterruptHandler(sw1.port,sw1.pin, (IsrPointer)vSwitchMode, kRisingEdge);
    isr_register(EINT3_IRQn, Eint3Handler);


    xTaskCreate(vPWMADCTEST, "PWMADCTest", 1000, NULL, PRIORITY_LOW, NULL);
    xTaskCreate(vRGBTEST, "RGBTest", 1000, NULL, PRIORITY_LOW, NULL);
    xTaskCreate(vPrintTask, "Print", 1000, NULL, PRIORITY_LOW, NULL);
    xTaskCreate(vLightRGB, "LightSens", 1000, NULL, PRIORITY_LOW, NULL);

    scheduler_start();
    return EXIT_FAILURE;
}

```

```
 * pwmDriver.hpp

#ifndef PWMDRIVER_HPP_
#define PWMDRIVER_HPP_

#include <sys/_stdint.h>
#include <LPC17xx.h>
#include "printf_lib.h"

//#define PCLK_RATE    1500000
#define PCLK_RATE    48000000
#define RESOLUTION   1000
#define DEFAULT_FREQ 1000

class LabPwm
{
 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
    };

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

    /**
    * 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 = DEFAULT_FREQ);

    /**
    * 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);
 private:
    static uint64_t pr;
    static uint64_t mr0;
};



#endif /* PWMDRIVER_HPP_ */

```

```
 * pwmDriver.cpp

#include <PWM/pwmDriver.hpp>

uint64_t LabPwm::pr = 0;
uint64_t LabPwm::mr0 = 0;

LabPwm::LabPwm(){
    pr = 0;
    mr0 = 0;
}

void LabPwm::PwmSelectAllPins()
{
   //Select pins 2.0 - 2.5 as PWM: 01
   LPC_PINCON->PINSEL4 |= (1 << 0);
   LPC_PINCON->PINSEL4 |= (1 << 2);
   LPC_PINCON->PINSEL4 |= (1 << 4);
   LPC_PINCON->PINSEL4 |= (1 << 6);
   LPC_PINCON->PINSEL4 |= (1 << 8);
   LPC_PINCON->PINSEL4 |= (1 << 10);

   LPC_PINCON->PINSEL4 &= ~(1 << 1);
   LPC_PINCON->PINSEL4 &= ~(1 << 3);
   LPC_PINCON->PINSEL4 &= ~(1 << 5);
   LPC_PINCON->PINSEL4 &= ~(1 << 7);
   LPC_PINCON->PINSEL4 &= ~(1 << 9);
   LPC_PINCON->PINSEL4 &= ~(1 << 11);

   LPC_PINCON->PINMODE4 |= (0xFFF);


   /*
    * Initialize PWM Channels
    */
   LPC_PWM1->PCR &= ~(1 << 2);
   LPC_PWM1->PCR &= ~(1 << 3);
   LPC_PWM1->PCR &= ~(1 << 4);
   LPC_PWM1->PCR &= ~(1 << 5);
   LPC_PWM1->PCR &= ~(1 << 6);

   /*
    * Enable PWM Output on all channels
    */
   LPC_PWM1->PCR |= (1 << 9);
   LPC_PWM1->PCR |= (1 << 10);
   LPC_PWM1->PCR |= (1 << 11);
   LPC_PWM1->PCR |= (1 << 12);
   LPC_PWM1->PCR |= (1 << 13);
   LPC_PWM1->PCR |= (1 << 14);
}

void LabPwm::PwmSelectPin(PWM_Pin pwm_pin_arg)
{
    /*
     * Select pin as PWM
     */
    LPC_PINCON->PINSEL4 |= (1 << (2*pwm_pin_arg));
    /*
     * Set Single Edge Controlled Mode for requested pin
     */
    if (pwm_pin_arg > 0)
        LPC_PWM1->PCR &= ~(1 << (pwm_pin_arg + 1));
    /*
     * Enable PWM output on Requested pin
     */
    LPC_PWM1->PCR |= (1 << (pwm_pin_arg + 9));
}

void LabPwm::PwmInitSingleEdgeMode(uint32_t frequency_Hz)
{
    /*
     * Enable PWM peripheral power and clock
     */
    LPC_SC->PCONP    |=  (1 << pconp_pwm1);
    //LPC_SC->PCLKSEL0 |=  (2 << (2*pclk_pwm1)); // /8
    LPC_SC->PCLKSEL0 |=  (1 << (2*pclk_pwm1));

    /*
     * PR: Prescaler Register Controls Count Rate
     * Want 1Khz default
     * PCLK = 48MHz/4 => 12Mhz
     * 12MHz / (PC + 1) = 1Khz => PC = 11999
     */
    pr = (uint64_t)((((uint32_t)PCLK_RATE / frequency_Hz)/RESOLUTION) - 1);
    u0_dbg_printf("pr: %u\n\n", pr);
    mr0 = RESOLUTION;
    LPC_PWM1->MR0 = (uint32_t)mr0;
    LPC_PWM1->PR  = (uint32_t)pr;
    /*
     * Set to single edge
     */
    LPC_PWM1->PCR &= ~(0x1F << 2);
    /*
```

```
     * Set all MR (match counters) to 0 for 0% duty cycle
     */
    LPC_PWM1->MR0 = (uint32_t)mr0;
    LPC_PWM1->MR1 = 0;
    LPC_PWM1->MR2 = 0;
    LPC_PWM1->MR3 = 0;
    LPC_PWM1->MR4 = 0;
    LPC_PWM1->MR5 = 0;
    LPC_PWM1->MR6 = 0;
    /*
     * Reset when TC reaches MR0
     */
    LPC_PWM1->MCR |= (1 << 1);
    /*
     * PWMLER
     */
    LPC_PWM1->LER |= (1 << 0);
    LPC_PWM1->LER |= (1 << 1);
    LPC_PWM1->LER |= (1 << 2);
    LPC_PWM1->LER |= (1 << 3);
    LPC_PWM1->LER |= (1 << 4);
    LPC_PWM1->LER |= (1 << 5);
    LPC_PWM1->LER |= (1 << 6);





    /*
     * Enable Counter
     */

    LPC_PWM1->TCR |= 1;
    LPC_PWM1->TCR |=  (1 << 1);
    LPC_PWM1->TCR &= ~(1 << 1);
    LPC_PWM1->TCR |= (1 << 3);


}

void LabPwm::SetDutyCycle(PWM_Pin pwm_pin_arg, float duty_cycle_percentage)
{
    if ((duty_cycle_percentage < 0) || (duty_cycle_percentage > 1))
        return;
    uint32_t mr = (uint32_t)(duty_cycle_percentage * (float)mr0);

    switch(pwm_pin_arg){
        case k2_0:
            LPC_PWM1->MR1 = mr;
            LPC_PWM1->LER |= (1 << 1);
            break;
        case k2_1:
            LPC_PWM1->MR2 = mr;
            LPC_PWM1->LER |= (1 << 2);
            //u0_dbg_printf("mr0: %u, pr: %u, tc: %u\n\n", LPC_PWM1->MR0, LPC_PWM1->PR, LPC_PWM1->TC);
            break;
        case k2_2:
            LPC_PWM1->MR3 = mr;
            LPC_PWM1->LER |= (1 << 3);
            break;
        case k2_3:
            LPC_PWM1->MR4 = mr;
            LPC_PWM1->LER |= (1 << 4);
            break;
        case k2_4:
            LPC_PWM1->MR5 = mr;
            LPC_PWM1->LER |= (1 << 5);
            break;
        case k2_5:
            LPC_PWM1->MR6 = mr;
            LPC_PWM1->LER |= (1 << 6);
            break;
        default:
            return;
    }
}

void LabPwm::SetFrequency(uint32_t frequency_Hz)
{
    if (frequency_Hz <= 0)
        return;
    pr = (uint64_t)((((uint32_t)PCLK_RATE / frequency_Hz)/RESOLUTION) - 1);
    mr0 = RESOLUTION;
    LPC_PWM1->PR  = (uint32_t)pr;
    LPC_PWM1->MR0 = (uint32_t)mr0;
    LPC_PWM1->LER |= (1 << 0);

}

```

```
/*
 * adcDriver.hpp
 *
 *  Created on: Mar 2, 2019
 *      Author: Nick Schiffer
 */

#ifndef ADCDRIVER_HPP_
#define ADCDRIVER_HPP_


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

#define CLOCK_DIV       4
#define ADC_PIN_NUMBER 18
#define VREF            3.3

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

        /* These ADC channels are compromised on the SJ-One,
         * hence you do not need to support them
         */
        // k0_23 = 0,   // AD0.0
        // k0_24,       // AD0.1
        // k0_3,        // AD0.6
        // k0_2         // AD0.7
    };

    enum ADC_Channel {
        channel_0 = 0,
        channel_1 = 1,
        channel_2 = 2,
        channel_3 = 3,
        channel_4 = 4,
        channel_5 = 5,
        channel_6 = 6,
        channel_7 = 7,
    };


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

    /**
    * 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 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 AdcSelectPin(Pin pin);

    /**
    * 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 ReadAdcVoltageByChannel(ADC_Channel channel);
private:
    enum pinsel {
        p0_25 = 18,       // AD0.2 <-- Light Sensor -->
        p0_26 = 20,       // AD0.3
        p1_30 = 28,       // AD0.4
        p1_31 = 30,       // AD0.5
    };

};


#endif /* ADCDRIVER_HPP_ */

```

```cpp
 * adcDriver.cpp

#include "adcDriver.hpp"

LabAdc::LabAdc()
{
}

void LabAdc::AdcInitBurstMode()
{
    /*
     * Set pin 0.25 to ADC0.2
     */
    //LPC_PINCON->PINSEL1 |= (1 << 18);

    /*
     * Initialize ADC Power
     */
    LPC_SC->PCONP |= (1 << pconp_adc);
    /*
     * Set clock divider (should be <= 13MHz) -> 48Mhz / 4 = 12Mhz: bits 15:8
     */
    LPC_ADC->ADCR |= (4 << CLOCK_DIV);
    /*
     * Set START bits to 000
     */
    LPC_ADC->ADCR &= ~(7 << 24);
    /*
     * Enable Burst Mode: bit 16
     */
    LPC_ADC->ADCR |= (1 << 16);
    /*
     * Enable ADC Operational State: pin 21
     */
    LPC_ADC->ADCR |= (1 << 21);
}

void LabAdc::AdcSelectPin(Pin pin)
{
    switch(pin){
        case k0_25:
            LPC_PINCON->PINSEL1 |= (1 << p0_25);
            LPC_ADC->ADCR       |= (1 << k0_25);
            break;
        case k0_26:
            LPC_PINCON->PINSEL1 |= (1 << p0_26);
            LPC_ADC->ADCR       |= (1 << k0_26);
            break;
        case k1_30:
            LPC_PINCON->PINSEL3 |= (1 << p1_30);
            LPC_ADC->ADCR       |= (1 << k1_30);
            break;
        case k1_31:
            LPC_PINCON->PINSEL3 |= (1 << p1_31);
            LPC_ADC->ADCR       |= (1 << k1_31);
            break;
        default:
            break;
    }
}

float LabAdc::ReadAdcVoltageByChannel(ADC_Channel channel)
{
    float voltage = 0;
    switch (channel){
        case channel_0:
            voltage = (uint16_t)LPC_ADC->ADDR0;
            break;
        case channel_1:
            voltage = (uint16_t)LPC_ADC->ADDR1;
            break;
        case channel_2:
            voltage = (uint16_t)LPC_ADC->ADDR2;
            break;
        case channel_3:
            voltage = (uint16_t)LPC_ADC->ADDR3;
            break;
        case channel_4:
            voltage = (uint16_t)LPC_ADC->ADDR4;
            break;
        case channel_5:
            voltage = (uint16_t)LPC_ADC->ADDR5;
            break;
        case channel_6:
            voltage = (uint16_t)LPC_ADC->ADDR6;
            break;
        case channel_7:
            voltage = (uint16_t)LPC_ADC->ADDR7;
            break;
        default:
            return -1;
    }

    voltage /= (float)0xFFFF;
    voltage *= (float)VREF;
    return voltage;
}

```