

Nickolas Schiffer

CMPE 146 S19

Lab: ADC\_PWM

main.cpp

```
1#include <stdlib.h>
2#include <LPC17xx.h>
3#include <tasks.hpp>
4#include <stdio.h>
5#include "ADC/adcDriver.hpp"
6#include "PWM/pwmDriver.hpp"
7#include "GPIO/GPIOInterrupt.hpp"
8#include <math.h>
9
10typedef bool Mode;
11
12#define VREF 3.3
13#define MODE    bool
14#define NORMAL false
15#define EC      true
16
17LabAdc::ADC_Channel pot_channel      = LabAdc::channel_3;
18LabAdc::ADC_Channel light_sens_channel = LabAdc::channel_2;
19LabAdc::Pin pot_pin      = LabAdc::k0_26;
20LabAdc::Pin light_sens_pin = LabAdc::k0_25;
21
22LabPwm::PWM_Pin red_pin  = LabPwm::k2_0;
23LabPwm::PWM_Pin green_pin = LabPwm::k2_1;
24LabPwm::PWM_Pin blue_pin  = LabPwm::k2_2;
25
26struct sw{
27    uint8_t port = 2;
28    uint8_t pin  = 7;
29}sw1;
30
31typedef enum state{
32    Normal,
33    RGBPulse,
34    LightSense,
35    KnobRGB
36};
37
38state mode = Normal;
39
40
41
42
43
44float duty_cycle_red, duty_cycle_green, duty_cycle_blue;
45
46
47inline float map(float x, float in_min, float in_max, float out_min, float
```

```

    out_max){
48     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
49 }
50
51 /*
52  * Upon Interrupt, this method changes the operation mode
53  */
54 void vSwitchMode(){
55     switch (mode){
56         case Normal:
57             mode = RGBPulse;
58             break;
59         case RGBPulse:
60             mode = LightSense;
61             break;
62         case LightSense:
63             mode = KnobRGB;
64             break;
65         case KnobRGB:
66             mode = Normal;
67             break;
68         default:
69             break;
70     }
71     return;
72 }
73
74 /*
75  * **Extra Credit**
76  * Maps Light Sensor or Potentiometer into point in rainbow.
77  * Rainbow is calculated via 3 offset sine waves.
78  * Voltage value is mapped from 0-3.3v to an angle 0-360 degrees to be used in
    sine function.
79  */
80 void vLightRGB(void *pvParameters){
81     auto pwm = LabPwm();
82     auto adc = LabAdc();
83     adc.AdcSelectPin(pot_pin);
84     adc.AdcSelectPin(light_sens_pin);
85
86     float voltage;
87     double angle = 0;
88
89     while(1){
90         while((mode != LightSense) && (mode != KnobRGB)){
91             vTaskDelay(100);
92         }

```

main.cpp

```
93     if (mode == LightSense)
94         voltage = adc.ReadAdcVoltageByChannel(light_sens_channel);
95     else
96         voltage = adc.ReadAdcVoltageByChannel(pot_channel);
97     angle = (double)map(voltage, 0, 3.3, 0, 360);
98     duty_cycle_red    = (float)((1*(sin((double)(angle/180*M_PI))+1))/2);
99     duty_cycle_green  = (float)((1*(sin((double)(angle/180*M_PI+((double)
100 (2.f/3.f)*M_PI))))+1))/2);
101     duty_cycle_blue   = (float)((1*(sin((double)(angle/180*M_PI+((double)
102 (4.f/3.f)*M_PI))))+1))/2);
103     pwm.SetDutyCycle(red_pin,duty_cycle_red);
104     pwm.SetDutyCycle(green_pin,duty_cycle_green);
105     pwm.SetDutyCycle(blue_pin,duty_cycle_blue);
106     vTaskDelay(10);
107 }
108
109 }
110
111 /*
112  *  **Extra Credit**
113  *  Rainbow is calculated via 3 offset sine waves.
114  *  for-loop loops through angle 0-360 degrees which covers the rainbow.
115  *  Potentiometer adjusts the step size for iterating through the circle,
116  *  therefore increasing the
117  *  rainbow speed.
118  */
119 void vRGBTEST(void *pvParamters){
120     auto pwm = LabPwm();
121     auto adc = LabAdc();
122     adc.AdcSelectPin(pot_pin);
123
124     float voltage;
125     uint16_t delay = 1;
126
127     while(1){
128         while(mode != RGBPulse){
129             vTaskDelay(100);
130         }
131         for (double x = 0; x < 360; x+=delay){
132             voltage = adc.ReadAdcVoltageByChannel(pot_channel);
133             delay = (uint16_t)map(voltage, 0, 3.3, 1, 50);
134             duty_cycle_red    = (float)((1*(sin((double)(x/180*M_PI))+1))/2);
135             duty_cycle_green  = (float)((1*(sin((double)(x/180*M_PI+((double)
136 (1.5)*M_PI))))+1))/2);
137             duty_cycle_blue   = (float)((1*(sin((double)(x/180*M_PI+((double)
```

```

        (0.5)*M_PI)))+1))/2);
136         pwm.SetDutyCycle(red_pin,duty_cycle_red);
137         pwm.SetDutyCycle(green_pin,duty_cycle_green);
138         pwm.SetDutyCycle(blue_pin,duty_cycle_blue);
139         vTaskDelay(10);
140     }
141
142
143     }
144
145
146 }
147
148 /*
149  * Prints operation mode,
150  * Voltage,
151  * and duty cycles
152  */
153 void vPrintTask(void *pvParameters){
154     auto adc = LabAdc();
155     adc.AdcSelectPin(pot_pin);
156     float voltage;
157
158     while (1){
159         voltage = adc.ReadAdcVoltageByChannel(pot_channel);
160         printf("Mode: %d\nvoltage: %f\nr_ds: %f\nng_ds: %f\nb_ds: %f\n\n",
mode, voltage, duty_cycle_red, duty_cycle_green, duty_cycle_blue);
161         vTaskDelay(1000);
162     }
163 }
164
165 /*
166  * Sets duty cycle of voltage to VREF.
167  */
168 void vPWMADCTEST(void *pvParameters){
169     auto pwm = LabPwm();
170     auto adc = LabAdc();
171
172     adc.AdcSelectPin(pot_pin);
173     adc.AdcInitBurstMode();
174
175     pwm.PwmSelectPin(red_pin);
176     pwm.PwmSelectPin(green_pin);
177     pwm.PwmSelectPin(blue_pin);
178     pwm.PwmInitSingleEdgeMode(100);
179     float voltage;
180

```

main.cpp

```
181     while (1){
182         while(mode != Normal){
183             vTaskDelay(100);
184         }
185
186         voltage = adc.ReadAdcVoltageByChannel(pot_channel);
187         duty_cycle_red = duty_cycle_green = duty_cycle_blue = (float)(voltage
/ (float)VREF);
188         pwm.SetDutyCycle(red_pin, duty_cycle_red);
189         pwm.SetDutyCycle(green_pin, duty_cycle_green);
190         pwm.SetDutyCycle(blue_pin, duty_cycle_blue);
191         vTaskDelay(10);
192     }
193
194
195 }
196 /*
197  * Detects switch button interrupt. Used to change operation mode.
198  */
199 void Eint3Handler(){
200     GPIOInterrupt *interruptHandler = GPIOInterrupt::getInstance();
201     interruptHandler->HandleInterrupt();
202 }
203
204 int main(){
205     scheduler_add_task(new terminalTask(PRIORITY_HIGH));
206
207
208     GPIOInterrupt *gpio_interrupts = GPIOInterrupt::getInstance();
209     gpio_interrupts->Initialize();
210     gpio_interrupts->AttachInterruptHandler(sw1.port, sw1.pin,
(IsrPointer)vSwitchMode, kRisingEdge);
211     isr_register(EINT3_IRQn, Eint3Handler);
212
213
214     xTaskCreate(vPWMADCTEST, "PWMADCTest", 1000, NULL, PRIORITY_LOW, NULL);
215     xTaskCreate(vRGBTEST, "RGBTest", 1000, NULL, PRIORITY_LOW, NULL);
216     xTaskCreate(vPrintTask, "Print", 1000, NULL, PRIORITY_LOW, NULL);
217     xTaskCreate(vLightRGB, "LightSens", 1000, NULL, PRIORITY_LOW, NULL);
218
219     scheduler_start();
220     return EXIT_FAILURE;
221 }
222
```

## pwmDriver.hpp

```
2 * pwmDriver.hpp
7
8 #ifndef PWMDRIVER_HPP_
9 #define PWMDRIVER_HPP_
10
11 #include <sys/_stdint.h>
12 #include <LPC17xx.h>
13 #include "printf_lib.h"
14
15 // #define PCLK_RATE 1500000
16 #define PCLK_RATE 48000000
17 #define RESOLUTION 1000
18 #define DEFAULT_FREQ 1000
19
20 class LabPwm
21 {
22 public:
23     enum PWM_Pin
24     {
25         k2_0,    // PWM1.1
26         k2_1,    // PWM1.2
27         k2_2,    // PWM1.3
28         k2_3,    // PWM1.4
29         k2_4,    // PWM1.5
30         k2_5,    // PWM1.6
31     };
32
33     /// Nothing needs to be done within the default constructor
34     LabPwm();
35
36     /**
37      * 1) Select PWM functionality on all PWM-able pins.
38      */
39     void PwmSelectAllPins();
40
41     /**
42      * 1) Select PWM functionality of pwm_pin_arg
43      *
44      * @param pwm_pin_arg is the PWM_PIN enumeration of the desired pin.
45      */
46     void PwmSelectPin(PWM_Pin pwm_pin_arg);
47
48     /**
49      * Initialize your PWM peripherals. See the notes here:
50      *
51      http://books.socialledge.com/books/embedded-drivers-real-time-operating-systems/page/pwm-%28pulse-width-modulation%29
52      */
53 }
```

## pwmDriver.hpp

```
51  *
52  * In general, you init the PWM peripheral, its frequency, and initialize
  your PWM channels and set them to 0% duty cycle
53  *
54  * @param frequency_Hz is the initial frequency in Hz.
55  */
56  void PwmInitSingleEdgeMode(uint32_t frequency_Hz = DEFAULT_FREQ);
57
58  /**
59  * 1) Convert duty_cycle_percentage to the appropriate match register value
  (depends on current frequency)
60  * 2) Assign the above value to the appropriate MRn register (depends on
  pwm_pin_arg)
61  *
62  * @param pwm_pin_arg is the PWM_PIN enumeration of the desired pin.
63  * @param duty_cycle_percentage is the desired duty cycle percentage.
64  */
65  void SetDutyCycle(PWM_Pin pwm_pin_arg, float duty_cycle_percentage);
66
67  /**
68  * Optional:
69  * 1) Convert frequency_Hz to the appropriate match register value
70  * 2) Assign the above value to MR0
71  *
72  * @param frequency_hz is the desired frequency of all pwm pins
73  */
74  void SetFrequency(uint32_t frequency_Hz);
75 private:
76  static uint64_t pr;
77  static uint64_t mr0;
78 };
79
80
81
82 #endif /* PWMDRIVER_HPP_ */
83
```



pwmDriver.cpp

```
2 * pwmDriver.cpp
7
8#include <PWM/pwmDriver.hpp>
9
10uint64_t LabPwm::pr = 0;
11uint64_t LabPwm::mr0 = 0;
12
13LabPwm::LabPwm(){
14    pr = 0;
15    mr0 = 0;
16}
17
18void LabPwm::PwmSelectAllPins()
19{
20    //Select pins 2.0 - 2.5 as PWM: 01
21    LPC_PINCON->PINSEL4 |= (1 << 0);
22    LPC_PINCON->PINSEL4 |= (1 << 2);
23    LPC_PINCON->PINSEL4 |= (1 << 4);
24    LPC_PINCON->PINSEL4 |= (1 << 6);
25    LPC_PINCON->PINSEL4 |= (1 << 8);
26    LPC_PINCON->PINSEL4 |= (1 << 10);
27
28    LPC_PINCON->PINSEL4 &= ~(1 << 1);
29    LPC_PINCON->PINSEL4 &= ~(1 << 3);
30    LPC_PINCON->PINSEL4 &= ~(1 << 5);
31    LPC_PINCON->PINSEL4 &= ~(1 << 7);
32    LPC_PINCON->PINSEL4 &= ~(1 << 9);
33    LPC_PINCON->PINSEL4 &= ~(1 << 11);
34
35    LPC_PINCON->PINMODE4 |= (0xFFF);
36
37
38    /*
39     * Initialize PWM Channels
40     */
41    LPC_PWM1->PCR &= ~(1 << 2);
42    LPC_PWM1->PCR &= ~(1 << 3);
43    LPC_PWM1->PCR &= ~(1 << 4);
44    LPC_PWM1->PCR &= ~(1 << 5);
45    LPC_PWM1->PCR &= ~(1 << 6);
46
47    /*
48     * Enable PWM Output on all channels
49     */
50    LPC_PWM1->PCR |= (1 << 9);
51    LPC_PWM1->PCR |= (1 << 10);
52    LPC_PWM1->PCR |= (1 << 11);
```

pwmDriver.cpp

```
53     LPC_PWM1->PCR |= (1 << 12);
54     LPC_PWM1->PCR |= (1 << 13);
55     LPC_PWM1->PCR |= (1 << 14);
56 }
57
58 void LabPwm::PwmSelectPin(PWM_Pin pwm_pin_arg)
59 {
60     /*
61      * Select pin as PWM
62      */
63     LPC_PINCON->PINSEL4 |= (1 << (2*pwm_pin_arg));
64     /*
65      * Set Single Edge Controlled Mode for requested pin
66      */
67     if (pwm_pin_arg > 0)
68         LPC_PWM1->PCR &= ~(1 << (pwm_pin_arg + 1));
69     /*
70      * Enable PWM output on Requested pin
71      */
72     LPC_PWM1->PCR |= (1 << (pwm_pin_arg + 9));
73 }
74
75 void LabPwm::PwmInitSingleEdgeMode(uint32_t frequency_Hz)
76 {
77     /*
78      * Enable PWM peripheral power and clock
79      */
80     LPC_SC->PCONP |= (1 << pconp_pwm1);
81     //LPC_SC->PCLKSEL0 |= (2 << (2*pclk_pwm1)); // /8
82     LPC_SC->PCLKSEL0 |= (1 << (2*pclk_pwm1));
83
84     /*
85      * PR: Prescaler Register Controls Count Rate
86      * Want 1Khz default
87      * PCLK = 48MHz/4 => 12Mhz
88      * 12MHz / (PC + 1) = 1Khz => PC = 11999
89      */
90     pr = (uint64_t)((((uint32_t)PCLK_RATE / frequency_Hz)/RESOLUTION) - 1);
91     u0_dbg_printf("pr: %u\n\n", pr);
92     mr0 = RESOLUTION;
93     LPC_PWM1->MR0 = (uint32_t)mr0;
94     LPC_PWM1->PR = (uint32_t)pr;
95     /*
96      * Set to single edge
97      */
98     LPC_PWM1->PCR &= ~(0x1F << 2);
99     /*
```

pwmDriver.cpp

```
100     * Set all MR (match counters) to 0 for 0% duty cycle
101     */
102     LPC_PWM1->MR0 = (uint32_t)mr0;
103     LPC_PWM1->MR1 = 0;
104     LPC_PWM1->MR2 = 0;
105     LPC_PWM1->MR3 = 0;
106     LPC_PWM1->MR4 = 0;
107     LPC_PWM1->MR5 = 0;
108     LPC_PWM1->MR6 = 0;
109     /*
110     * Reset when TC reaches MR0
111     */
112     LPC_PWM1->MCR |= (1 << 1);
113     /*
114     * PWMLER
115     */
116     LPC_PWM1->LER |= (1 << 0);
117     LPC_PWM1->LER |= (1 << 1);
118     LPC_PWM1->LER |= (1 << 2);
119     LPC_PWM1->LER |= (1 << 3);
120     LPC_PWM1->LER |= (1 << 4);
121     LPC_PWM1->LER |= (1 << 5);
122     LPC_PWM1->LER |= (1 << 6);
123
124
125
126
127
128     /*
129     * Enable Counter
130     */
131
132     LPC_PWM1->TCR |= 1;
133     LPC_PWM1->TCR |= (1 << 1);
134     LPC_PWM1->TCR &= ~(1 << 1);
135     LPC_PWM1->TCR |= (1 << 3);
136
137
138 }
139
140 void LabPwm::SetDutyCycle(PWM_Pin pwm_pin_arg, float duty_cycle_percentage)
141 {
142     if ((duty_cycle_percentage < 0) || (duty_cycle_percentage > 1))
143         return;
144     uint32_t mr = (uint32_t)(duty_cycle_percentage * (float)mr0);
145
146     switch(pwm_pin_arg){
```

pwmDriver.cpp

```
147     case k2_0:
148         LPC_PWM1->MR1 = mr;
149         LPC_PWM1->LER |= (1 << 1);
150         break;
151     case k2_1:
152         LPC_PWM1->MR2 = mr;
153         LPC_PWM1->LER |= (1 << 2);
154         //u0_dbg_printf("mr0: %u, pr: %u, tc: %u\n\n", LPC_PWM1->MR0,
LPC_PWM1->PR, LPC_PWM1->TC);
155         break;
156     case k2_2:
157         LPC_PWM1->MR3 = mr;
158         LPC_PWM1->LER |= (1 << 3);
159         break;
160     case k2_3:
161         LPC_PWM1->MR4 = mr;
162         LPC_PWM1->LER |= (1 << 4);
163         break;
164     case k2_4:
165         LPC_PWM1->MR5 = mr;
166         LPC_PWM1->LER |= (1 << 5);
167         break;
168     case k2_5:
169         LPC_PWM1->MR6 = mr;
170         LPC_PWM1->LER |= (1 << 6);
171         break;
172     default:
173         return;
174 }
175 }
176
177 void LabPwm::SetFrequency(uint32_t frequency_Hz)
178 {
179     if (frequency_Hz <= 0)
180         return;
181     pr = (uint64_t)((((uint32_t)PCLK_RATE / frequency_Hz)/RESOLUTION) - 1);
182     mr0 = RESOLUTION;
183     LPC_PWM1->PR = (uint32_t)pr;
184     LPC_PWM1->MR0 = (uint32_t)mr0;
185     LPC_PWM1->LER |= (1 << 0);
186
187 }
188
```

## adcDriver.hpp

```
1 /*
2  * adcDriver.hpp
3  *
4  * Created on: Mar 2, 2019
5  * Author: Nick Schiffer
6  */
7
8 #ifndef ADCDRIVER_HPP_
9 #define ADCDRIVER_HPP_
10
11
12 #include <LPC17xx.h>
13 #include "io.hpp"
14
15 #define CLOCK_DIV 4
16 #define ADC_PIN_NUMBER 18
17 #define VREF 3.3
18
19 class LabAdc
20 {
21 public:
22     enum Pin
23     {
24         k0_25 = 2,      // AD0.2 <-- Light Sensor -->
25         k0_26 = 3,      // AD0.3
26         k1_30 = 4,      // AD0.4
27         k1_31 = 5,      // AD0.5
28
29         /* These ADC channels are compromised on the SJ-One,
30          * hence you do not need to support them
31          */
32         // k0_23 = 0,    // AD0.0
33         // k0_24,        // AD0.1
34         // k0_3,         // AD0.6
35         // k0_2          // AD0.7
36     };
37
38     enum ADC_Channel {
39         channel_0 = 0,
40         channel_1 = 1,
41         channel_2 = 2,
42         channel_3 = 3,
43         channel_4 = 4,
44         channel_5 = 5,
45         channel_6 = 6,
46         channel_7 = 7,
47     };
48 }
```

## adcDriver.hpp

```
48
49
50 // Nothing needs to be done within the default constructor
51 LabAdc();
52
53 /**
54  * 1) Powers up ADC peripheral
55  * 2) Set peripheral clock
56  * 2) Enable ADC
57  * 3) Select ADC channels
58  * 4) Enable burst mode
59  */
60 void AdcInitBurstMode();
61
62 /**
63  * 1) Selects ADC functionality of any of the ADC pins that are ADC capable
64  *
65  * @param pin is the LabAdc::Pin enumeration of the desired pin.
66  *
67  * WARNING: For proper operation of the SJOne board, do NOT configure any
68  pins
69  * as ADC except for 0.26, 1.31, 1.30
70  */
71 void AdcSelectPin(Pin pin);
72
73 /**
74  * 1) Returns the voltage reading of the 12bit register of a given ADC
75  channel
76  * You have to convert the ADC raw value to the voltage value
77  * @param channel is the number (0 through 7) of the desired ADC channel.
78  */
79 float ReadAdcVoltageByChannel(ADC_Channel channel);
80 private:
81 enum pinsel {
82     p0_25 = 18, // AD0.2 <-- Light Sensor -->
83     p0_26 = 20, // AD0.3
84     p1_30 = 28, // AD0.4
85     p1_31 = 30, // AD0.5
86 };
87
88 };
89 #endif /* ADCDRIVER_HPP_ */
90
```

## adcDriver.cpp

```
2 * adcDriver.cpp
7
8#include "adcDriver.hpp"
9
10LabAdc::LabAdc()
11{
12}
13
14void LabAdc::AdcInitBurstMode()
15{
16    /*
17     * Set pin 0.25 to ADC0.2
18     */
19    //LPC_PINCON->PINSEL1 |= (1 << 18);
20
21    /*
22     * Initialize ADC Power
23     */
24    LPC_SC->PCONP |= (1 << pconp_adc);
25    /*
26     * Set clock divider (should be <= 13MHz) -> 48Mhz / 4 = 12Mhz: bits 15:8
27     */
28    LPC_ADC->ADCR |= (4 << CLOCK_DIV);
29    /*
30     * Set START bits to 000
31     */
32    LPC_ADC->ADCR &= ~(7 << 24);
33    /*
34     * Enable Burst Mode: bit 16
35     */
36    LPC_ADC->ADCR |= (1 << 16);
37    /*
38     * Enable ADC Operational State: pin 21
39     */
40    LPC_ADC->ADCR |= (1 << 21);
41}
42
43void LabAdc::AdcSelectPin(Pin pin)
44{
45    switch(pin){
46        case k0_25:
47            LPC_PINCON->PINSEL1 |= (1 << p0_25);
48            LPC_ADC->ADCR      |= (1 << k0_25);
49            break;
50        case k0_26:
51            LPC_PINCON->PINSEL1 |= (1 << p0_26);
52            LPC_ADC->ADCR      |= (1 << k0_26);
```

adcDriver.cpp

```
53         break;
54     case k1_30:
55         LPC_PINCON->PINSEL3 |= (1 << p1_30);
56         LPC_ADC->ADCR      |= (1 << k1_30);
57         break;
58     case k1_31:
59         LPC_PINCON->PINSEL3 |= (1 << p1_31);
60         LPC_ADC->ADCR      |= (1 << k1_31);
61         break;
62     default:
63         break;
64 }
65 }
66
67 float LabAdc::ReadAdcVoltageByChannel(ADC_Channel channel)
68 {
69     float voltage = 0;
70     switch (channel){
71         case channel_0:
72             voltage = (uint16_t)LPC_ADC->ADDR0;
73             break;
74         case channel_1:
75             voltage = (uint16_t)LPC_ADC->ADDR1;
76             break;
77         case channel_2:
78             voltage = (uint16_t)LPC_ADC->ADDR2;
79             break;
80         case channel_3:
81             voltage = (uint16_t)LPC_ADC->ADDR3;
82             break;
83         case channel_4:
84             voltage = (uint16_t)LPC_ADC->ADDR4;
85             break;
86         case channel_5:
87             voltage = (uint16_t)LPC_ADC->ADDR5;
88             break;
89         case channel_6:
90             voltage = (uint16_t)LPC_ADC->ADDR6;
91             break;
92         case channel_7:
93             voltage = (uint16_t)LPC_ADC->ADDR7;
94             break;
95         default:
96             return -1;
97     }
98
99     voltage /= (float)0xFFFF;
```



adcDriver.cpp

```
100     voltage *= (float)VREF;  
101     return voltage;  
102 }  
103
```