Nickolas Schiffer CMPE 146 S19

Lab: ADC_PWM

```
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>
10 typedef bool Mode;
11
12#define VREF 3.3
13#define MODE
                 bool
14#define NORMAL false
15#define EC
             true
16
17 LabAdc::ADC Channel pot channel
                                   = LabAdc::channel 3;
18 LabAdc::ADC_Channel light_sens_channel = LabAdc::channel_2;
19 LabAdc::Pin pot pin = LabAdc::k0 26;
20 LabAdc::Pin light sens pin = LabAdc::k0 25;
21
22 LabPwm::PWM Pin red pin = LabPwm::k2 0;
23 LabPwm::PWM Pin green pin = LabPwm::k2 1;
24 LabPwm::PWM Pin blue pin = LabPwm::k2 2;
25
26 struct sw{
      uint8_t port = 2;
      uint8_t pin = 7;
28
29 }sw1;
30
31typedef enum state{
32
      Normal,
33
      RGBPulse,
34
      LightSense,
35
      KnobRGB
36 };
38 state mode = Normal;
39
40
41
42
44 float duty cycle red, duty cycle green, duty cycle blue;
45
46
47 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;
48
49 }
50
51/*
52 * Upon Interrupt, this method changes the operation mode
53 */
54 void vSwitchMode(){
      switch (mode){
56
          case Normal:
57
               mode = RGBPulse;
58
               break;
59
           case RGBPulse:
60
              mode = LightSense;
61
              break;
62
          case LightSense:
              mode = KnobRGB;
63
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 */
80void 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
           }
```

```
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
                                                                 = (float)((1*(sin((double)(angle/180*M PI))+1))/2);
                          duty cycle red
  99
                          duty_cycle_green = (float)((1*(sin((double))(angle/180*M_PI+((double))
       (2.f/3.f)*MPI))+1))/2);
100
                          duty cycle blue = (float)((1*(sin((double)(angle/180*M PI+((double)(angle/180*M PI+((double)(a
       (4.f/3.f)*MPI))+1))/2);
101
                          pwm.SetDutyCycle(red_pin,duty_cycle_red);
102
                          pwm.SetDutyCycle(green_pin,duty_cycle_green);
103
                          pwm.SetDutyCycle(blue pin,duty cycle blue);
104
                          vTaskDelay(10);
105
106
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,
       therefore increasing the
116 * rainbow speed.
117 */
118 void vRGBTEST(void *pvParamters){
119
                 auto pwm = LabPwm();
120
                 auto adc = LabAdc();
121
                 adc.AdcSelectPin(pot_pin);
122
123
                 float voltage;
124
                 uint16_t delay = 1;
125
126
                 while(1){
127
                          while(mode != RGBPulse){
128
                                   vTaskDelay(100);
129
                          }
130
                          for (double x = 0; x < 360; x+=delay){
131
                                   voltage = adc.ReadAdcVoltageByChannel(pot_channel);
132
                                   delay = (uint16_t)map(voltage, 0, 3.3, 1, 50);
133
                                   duty cycle red
                                                                        = (float)((1*(sin((double)(x/180*M PI))+1))/2);
134
                                   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)
135
```

```
(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 *pvParamters){
154
       auto adc = LabAdc();
155
       adc.AdcSelectPin(pot pin);
156
       float voltage;
157
158
       while (1){
159
           voltage = adc.ReadAdcVoltageByChannel(pot_channel);
           printf("Mode: %d\nvoltage: %f\nr ds: %f\ng ds: %f\nb ds: %f\n\n",
160
   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);
       pwm.PwmSelectPin(green_pin);
176
177
       pwm.PwmSelectPin(blue pin);
178
       pwm.PwmInitSingleEdgeMode(100);
179
       float voltage;
180
```

```
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);
           vTaskDelay(10);
191
192
       }
193
194
195 }
196/*
197 * Detects switch button interrupt. Used to change operation mode.
198 */
199 void Eint3Handler(){
       GPIOInterrupt *interruptHandler = GPIOInterrupt::qetInstance();
200
       interruptHandler->HandleInterrupt();
201
202 }
203
204 int main(){
       scheduler add task(new terminalTask(PRIORITY HIGH));
205
206
207
208
       GPIOInterrupt *gpio interrupts = GPIOInterrupt::qetInstance();
       gpio interrupts->Initialize();
209
210
       gpio interrupts->AttachInterruptHandler(sw1.port,sw1.pin,
   (IsrPointer)vSwitchMode, kRisingEdge);
211
       isr register(EINT3 IROn, Eint3Handler);
212
213
       xTaskCreate(vPWMADCTEST, "PWMADCTest", 1000, NULL, PRIORITY_LOW, NULL);
214
       xTaskCreate(vRGBTEST, "RGBTest", 1000, NULL, PRIORITY_LOW, NULL);
215
216
       xTaskCreate(vPrintTask, "Print", 1000, NULL, PRIORITY LOW, NULL);
       xTaskCreate(vLightRGB, "LightSens", 1000, NULL, PRIORITY_LOW, NULL);
217
218
219
       scheduler start();
220
       return EXIT FAILURE;
221 }
222
```

pwmDriver.hpp

```
2 * pwmDriver.hpp
8#ifndef PWMDRIVER_HPP_
 9#define PWMDRIVER HPP
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
          k2_{1},
26
                   // PWM1.2
27
          k2 2,
                   // PWM1.3
28
          k2 3,
                   // PWM1.4
29
          k2 4,
                   // PWM1.5
          k2_{5}
30
                   // 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
      * 1) Select PWM functionality of pwm pin arg
42
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
  http://books.socialledge.com/books/embedded-drivers-real-time-operating-systems
  /page/pwm-%28pulse-width-modulation%29
```

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
      * 1) Convert duty cycle percentage to the appropriate match register value
59
  (depends on current frequency)
      * 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.
      * @param duty cycle percentage is the desired duty cycle percentage.
63
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;
      static uint64 t mr0;
77
78 };
79
80
81
82#endif /* PWMDRIVER HPP */
83
```

```
2 * pwmDriver.cpp
 8#include <PWM/pwmDriver.hpp>
10 \text{ uint} 64\_t \text{ } LabPwm::pr = 0;
11 uint64 t LabPwm::mr0 = 0;
12
13 LabPwm::LabPwm(){
      pr = 0;
14
15
      mr\theta = 0;
16 }
17
18 void LabPwm::PwmSelectAllPins()
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 &= \sim(1 << 7);
32
     LPC_PINCON->PINSEL4 \&= \sim(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 \&= \sim (1 << 2);
42
     LPC PWM1->PCR &= \sim(1 << 3);
43
     LPC PWM1->PCR &= \sim(1 << 4);
44
     LPC PWM1->PCR &= \sim(1 << 5);
45
     LPC PWM1->PCR &= \sim(1 << 6);
46
47
     /*
48
      * Enable PWM Output on all channels
49
50
     LPC PWM1->PCR = (1 << 9);
51
     LPC_PWM1->PCR = (1 << 10);
     LPC PWM1->PCR \mid= (1 << 11);
52
```

pwmDriver.cpp

```
LPC PWM1->PCR \mid= (1 << 12);
53
     LPC PWM1->PCR |= (1 << 13);
54
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 &= \sim(1 << (pwm pin arg + 1));
69
70
       * Enable PWM output on Requested pin
71
72
      LPC PWM1->PCR \mid= (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
       * Want 1Khz default
86
87
       * PCLK = 48MHz/4 \Rightarrow 12Mhz
       * 12MHz / (PC + 1) = 1Khz => PC = 11999
88
89
90
      pr = (uint64_t)((((uint32_t)PCLK_RATE / frequency Hz)/RESOLUTION) - 1);
      u0 dbg_printf("pr: %u\n\n", pr);
91
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 &= \sim(0x1F << 2);
99
      /*
```

pwmDriver.cpp

```
* Set all MR (match counters) to 0 for 0% duty cycle
100
101
102
       LPC_PWM1->MR0 = (uint32_t)mr0;
103
       LPC PWM1->MR1 = 0;
       LPC PWM1->MR2 = 0;
104
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 \mid = (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 \mid = (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 \mid= (1 << 1);
134
       LPC PWM1->TCR &= \sim(1 << 1);
       LPC_PWM1->TCR \mid = (1 << 3);
135
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;
       uint32_t mr = (uint32_t)(duty_cycle_percentage * (float)mr0);
144
145
146
       switch(pwm_pin_arg){
```

pwmDriver.cpp

```
147
           case k2 0:
                LPC PWM1->MR1 = mr;
148
149
                LPC_PWM1->LER = (1 << 1);
150
                break:
151
           case k2 1:
152
                LPC PWM1->MR2 = mr;
153
                LPC PWM1->LER = (1 << 2);
                //u0_dbg_printf("mr0: %u, pr: %u, tc: %u\n\n", LPC_PWM1->MR0,
154
   LPC PWM1->PR, LPC PWM1->TC);
155
               break;
156
           case k2 2:
157
                LPC PWM1->MR3 = mr;
158
                LPC PWM1->LER = (1 \ll 3);
159
                break;
160
           case k2 3:
161
                LPC PWM1->MR4 = mr;
                LPC PWM1->LER = (1 \ll 4);
162
163
                break;
164
           case k2 4:
                LPC PWM1->MR5 = mr;
165
                LPC PWM1->LER = (1 << 5);
166
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)</pre>
180
           return:
       pr = (uint64_t)((((uint32_t)PCLK_RATE / frequency_Hz)/RESOLUTION) - 1);
181
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
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
16#define ADC PIN NUMBER 18
17#define VREF
                           3.3
18
19 class LabAdc
20 {
21 public:
      enum Pin
22
23
          k0 \ 25 = 2
24
                           // 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,
          * hence you do not need to support them
30
          */
31
32
          // k0_23 = 0,
                           // AD0.0
33
          // k0 24,
                          // AD0.1
          // k0 3,
34
                          // AD0.6
35
          // k0 2
                           // AD0.7
36
      };
37
38
      enum ADC Channel {
          channel_0 = 0,
39
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
      };
```

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
      * 3) Select ADC channels
57
58
      * 4) Enable burst mode
59
      */
60
      void AdcInitBurstMode();
61
      /**
62
      * 1) Selects ADC functionality of any of the ADC pins that are ADC capable
63
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
  pins
                  as ADC except for 0.26, 1.31, 1.30
68
69
      */
70
      void AdcSelectPin(Pin pin);
71
72
      /**
73
      * 1) Returns the voltage reading of the 12bit register of a given ADC
  channel
74
      * 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.
75
76
77
      float ReadAdcVoltageByChannel(ADC Channel channel);
78 private:
79
      enum pinsel {
          p0_25 = 18,
                           // AD0.2 <-- Light Sensor -->
80
81
          p0_26 = 20,
                           // AD0.3
                           // AD0.4
82
          p1 30 = 28,
83
          p1 \ 31 = 30,
                           // AD0.5
84
      };
85
86 };
87
89 #endif /* ADCDRIVER HPP */
90
```

adcDriver.cpp

```
2 * adcDriver.cpp
 8#include "adcDriver.hpp"
10 LabAdc::LabAdc()
11 {
12 }
13
14 void LabAdc::AdcInitBurstMode()
15 {
16
       * Set pin 0.25 to ADC0.2
17
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 &= \sim(7 << 24);
33
34
       * Enable Burst Mode: bit 16
35
36
      LPC ADC->ADCR = (1 \ll 16);
37
38
       * Enable ADC Operational State: pin 21
39
      LPC ADC->ADCR = (1 \ll 21);
40
41 }
42
43 void 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);
               LPC ADC->ADCR
60
                                   |= (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

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