```c
// Analog LCD Example
// Jason Losh

//-----------------------------------------------------------------------------
// Hardware Target
//-----------------------------------------------------------------------------

// Target Platform: EK-TM4C123GXL with LCD/Temperature Sensor
// Target uC:       TM4C123GH6PM
// System Clock:    40 MHz

// Hardware configuration:
// Red Backlight LED:
//   PB5 drives an NPN transistor that powers the red LED
// Green Backlight LED:
//   PE5 drives an NPN transistor that powers the green LED
// Blue Backlight LED:
//   PE4 drives an NPN transistor that powers the blue LED
// LM60 Temperature Sensor:
//   AN0/PE3 is driven by the sensor (Vout = 424mV + 6.25mV / degC with +/-2degC uncalibrated error)
// ST7565R Graphics LCD Display Interface:
//   MOSI (SSI2Tx) on PB7
//   MISO (SSI2Rx) is not used by the LCD display but the pin is used for GPIO for A0
//   SCLK (SSI2Clk) on PB4
//   A0 connected to PB6
//   ~CS connected to PB1

//-----------------------------------------------------------------------------
// Device includes, defines, and assembler directives
//-----------------------------------------------------------------------------

#include <stdint.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include "tm4c123gh6pm.h"
#include "graphics_lcd.h"
#include "wait.h"
#define RED_BL_LED   (*((volatile uint32_t *)(0x42000000 + (0x400053FC-0x40000000)*32 + 5*4)))
#define GREEN_BL_LED (*((volatile uint32_t *)(0x42000000 + (0x400243FC-0x40000000)*32 + 5*4)))
#define BLUE_BL_LED  (*((volatile uint32_t *)(0x42000000 + (0x400243FC-0x40000000)*32 + 4*4)))

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Subroutines
//-----------------------------------------------------------------------------

// Initialize Hardware
void initHw()
{
    // Configure HW to work with 16 MHz XTAL, PLL enabled, system clock of 40 MHz
    SYSCTL_RCC_R = SYSCTL_RCC_XTAL_16MHZ | SYSCTL_RCC_OSCSRC_MAIN | SYSCTL_RCC_USESYSDIV | (4
```

```
  << SYSCTL_RCC_SYSDIV_S);
56
57    // Set GPIO ports to use APB (not needed since default configuration -- for clarity)
58    // Note UART on port A must use APB
59    SYSCTL_GPIOHBCTL_R = 0;
60
61    // Enable GPIO port B and E peripherals
62    SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOB | SYSCTL_RCGC2_GPIOE;
63
64    // Configure three backlight LEDs
65    GPIO_PORTB_DIR_R |= 0x20;  // make bit5 an output
66    GPIO_PORTB_DR2R_R |= 0x20; // set drive strength to 2mA
67    GPIO_PORTB_DEN_R |= 0x20;  // enable bit5 for digital
68    GPIO_PORTE_DIR_R |= 0x30;  // make bits 4 and 5 outputs
69    GPIO_PORTE_DR2R_R |= 0x30; // set drive strength to 2mA
70    GPIO_PORTE_DEN_R |= 0x30;  // enable bits 4 and 5 for digital
71
72    // Configure A0 and ~CS for graphics LCD
73    GPIO_PORTB_DIR_R |= 0x42;  // make bits 1 and 6 outputs
74    GPIO_PORTB_DR2R_R |= 0x42; // set drive strength to 2mA
75    GPIO_PORTB_DEN_R |= 0x42;  // enable bits 1 and 6 for digital
76
77    // Configure SSI2 pins for SPI configuration
78    SYSCTL_RCGCSSI_R |= SYSCTL_RCGCSSI_R2;           // turn-on SSI2 clocking
79    GPIO_PORTB_DIR_R |= 0x90;                        // make bits 4 and 7 outputs
80    GPIO_PORTB_DR2R_R |= 0x90;                       // set drive strength to 2mA
81    GPIO_PORTB_AFSEL_R |= 0x90;                      // select alternative functions for MOSI,
  SCLK pins
82    GPIO_PORTB_PCTL_R = GPIO_PCTL_PB7_SSI2TX | GPIO_PCTL_PB4_SSI2CLK; // map alt fns to SSI2
83    GPIO_PORTB_DEN_R |= 0x90;                        // enable digital operation on TX, CLK
  pins
84
85    // Configure the SSI2 as a SPI master, mode 3, 8bit operation, 1 MHz bit rate
86    SSI2_CR1_R &= ~SSI_CR1_SSE;                      // turn off SSI2 to allow
  re-configuration
87    SSI2_CR1_R = 0;                                  // select master mode
88    SSI2_CC_R = 0;                                   // select system clock as the clock
  source
89    SSI2_CPSR_R = 40;                                // set bit rate to 1 MHz (if SR=0 in CR0)
90    SSI2_CR0_R = SSI_CR0_SPH | SSI_CR0_SPO | SSI_CR0_FRF_MOTO | SSI_CR0_DSS_8; // set SR=0,
  mode 3 (SPH=1, SPO=1), 8-bit
91    SSI2_CR1_R |= SSI_CR1_SSE;                       // turn on SSI2
92
93    // Configure AN0 as an analog input
94    SYSCTL_RCGCADC_R |= 1;                           // turn on ADC module 0 clocking
95    GPIO_PORTE_AFSEL_R |= 0x08;                      // select alternative functions for AN0
  (PE3)
96    GPIO_PORTE_DEN_R &= ~0x08;                       // turn off digital operation on pin PE3
97    GPIO_PORTE_AMSEL_R |= 0x08;                      // turn on analog operation on pin PE3
98    ADC0_CC_R = ADC_CC_CS_SYSPLL;                    // select PLL as the time base (not
  needed, since default value)
99    ADC0_ACTSS_R &= ~ADC_ACTSS_ASEN3;                // disable sample sequencer 3 (SS3) for
  programming
100   ADC0_EMUX_R = ADC_EMUX_EM3_PROCESSOR;            // select SS3 bit in ADCPSSI as trigger
101   ADC0_SSMUX3_R = 0;                               // set first sample to AN0
102   ADC0_SSCTL3_R = ADC_SSCTL3_END0;                 // mark first sample as the end
```

```
103     ADC0_ACTSS_R |= ADC_ACTSS_ASEN3;                // enable SS3 for operation
104 }
105
106 int16_t readAdc0Ss3()
107 {
108     ADC0_PSSI_R |= ADC_PSSI_SS3;                    // set start bit
109     while (ADC0_ACTSS_R & ADC_ACTSS_BUSY);          // wait until SS3 is not busy
110     return ADC0_SSFIFO3_R;                          // get single result from the FIFO
111 }
112
113 //-----------------------------------------------------------------------------
114 // Main
115 //-----------------------------------------------------------------------------
116
117 int main(void)
118 {
119     // Initialize hardware
120     initHw();
121
122     // Turn-on all LEDs to create white backlight
123     RED_BL_LED = 1;
124     GREEN_BL_LED = 1;
125     BLUE_BL_LED = 1;
126
127     // Initialize graphics LCD
128     initGraphicsLcd();
129
130     // Draw legend
131     setGraphicsLcdTextPosition(0, 0);
132     putsGraphicsLcd("Raw ADC");
133     setGraphicsLcdTextPosition(0, 1);
134     putsGraphicsLcd("Unfiltered (C)");
135     setGraphicsLcdTextPosition(0, 2);
136     putsGraphicsLcd("Filtered (C)");
137
138     // Display raw ADC value and temperatures
139     uint16_t raw;
140     float instantTemp, iirTemp;
141     char str[10];
142     float alpha = 0.99;
143     int firstUpdate = true;
144     while(1)
145     {
146         // Read sensor
147         raw = readAdc0Ss3();
148         // Calculate temperature in degC as follows:
149         //   For the 12-bit SAR ADC with Vref+ = 3.3V and Vref- = 0V, outputing a result R:
150         //   Resolution is approx 0.81mV / LSb or 0.13 degC / LSb
151         //   R(Vin) = floor(Vin/3.3V * 4096) -> Vin(R) ~= 3.3V * ((R+0.5) / 4096)
152         //   (~ and 0.5LSb offset in Vin(R) equation are introduced for mid-tread value of the
    SAR transfer function)
153         //   T(Vin) = (Vin - 0.424V) / 0.00625V
154         //   T(R) ~= ([3.3V * ((R+0.5) / 4096)] - 0.424V) / 0.00625V
155         //   T(R) ~= (0.12890625 * R) - 67.775546875 (simplified floating point equation to
    save cycles)
156         instantTemp = ((raw / 4096.0 * 3.3) - 0.424) / 0.00625;
```

```
157         // First order IIR filter
158         //   In the z-domain:
159         //     H(z) = sum(j = 0..M) {bj * z^(-j)
160         //            -------------------------
161         //            sum(i = 0..N) {ai * z^(-i)
162         //   Setting a0 = 1, yields:
163         //     H(z) = sum(j = 0..M) {bj * z^(-j)
164         //            -----------------------------
165         //            1 + sum(i = 1..N) {ai * z^(-i)
166         //   for N = 1, M = 0:
167         //     H(z) = b0 / [1 + a1 * z^(-1)]
168         //   Given IIR difference equation:
169         //     sum(i = 0..N) {ai * y(n-i)} = sum(j = 0..M) {bj * x(n-j)}
170         //   Separating y(n), rearranging and inverting signs of a(1-N), yields
171         //     a0 * y(n) = sum(i = 1..N) {ai * y(n-i)} + sum(j = 0..M) {bj * x(n-j)}
172         //     for N = 1, M = 0, and a0 = 1,
173         //       y(n) = b0 * x(n) + a1 * y(n-1)
174         //   Setting b0 = (1-a1), yields
175         //     y(n) = alpha * y(n-1) + (1-alpha) * x(n)
176         //   Adding an exception for the first sample, yields:
177         //     y(n) = x(n); n = 0
178         //     y(n) = alpha * y(n-1) + (1-alpha) * x(n); n > 0
179         if (firstUpdate)
180         {
181             iirTemp = instantTemp;
182             firstUpdate = false;
183         }
184         else
185             iirTemp = iirTemp * alpha + instantTemp * (1-alpha);
186 
187         // display raw ADC value and temperatures
188         sprintf(str, "%u", raw);
189         setGraphicsLcdTextPosition(100, 0);
190         putsGraphicsLcd(str);
191         sprintf(str, "%3.1f", instantTemp);
192         setGraphicsLcdTextPosition(100, 1);
193         putsGraphicsLcd(str);
194         sprintf(str, "%3.1f", iirTemp);
195         setGraphicsLcdTextPosition(100, 2);
196         putsGraphicsLcd(str);
197         waitMicrosecond(500000);
198     }
199 }
200 
201 
```