```
// 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);

  // Set GPIO ports to use APB (not needed since default configuration -- for clarity)
  // Note UART on port A must use APB
  SYSCTL_GPIOHBCTL_R = 0;

  // Enable GPIO port B and E peripherals
  SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOB | SYSCTL_RCGC2_GPIOE;

  // Configure three backlight LEDs
  GPIO_PORTB_DIR_R |= 0x20;  // make bit5 an output
  GPIO_PORTB_DR2R_R |= 0x20; // set drive strength to 2mA
  GPIO_PORTB_DEN_R |= 0x20;  // enable bit5 for digital
  GPIO_PORTE_DIR_R |= 0x30;  // make bits 4 and 5 outputs
  GPIO_PORTE_DR2R_R |= 0x30; // set drive strength to 2mA
  GPIO_PORTE_DEN_R |= 0x30;  // enable bits 4 and 5 for digital

  // Configure A0 and ~CS for graphics LCD
  GPIO_PORTB_DIR_R |= 0x42;  // make bits 1 and 6 outputs
  GPIO_PORTB_DR2R_R |= 0x42; // set drive strength to 2mA
  GPIO_PORTB_DEN_R |= 0x42;  // enable bits 1 and 6 for digital

  // Configure SSI2 pins for SPI configuration
  SYSCTL_RCGCSSI_R |= SYSCTL_RCGCSSI_R2;        // turn-on SSI2 clocking
  GPIO_PORTB_DIR_R |= 0x90;                     // make bits 4 and 7 outputs
  GPIO_PORTB_DR2R_R |= 0x90;                    // set drive strength to 2mA
        GPIO_PORTB_AFSEL_R |= 0x90;             // select alternative functions for MOSI, SCLK pins
  GPIO_PORTB_PCTL_R = GPIO_PCTL_PB7_SSI2TX | GPIO_PCTL_PB4_SSI2CLK; // map alt fns to SSI2
  GPIO_PORTB_DEN_R |= 0x90;                     // enable digital operation on TX, CLK pins

  // Configure the SSI2 as a SPI master, mode 3, 8bit operation, 1 MHz bit rate
  SSI2_CR1_R &= ~SSI_CR1_SSE;                   // turn off SSI2 to allow re-configuration
  SSI2_CR1_R = 0;                               // select master mode
  SSI2_CC_R = 0;                                // select system clock as the clock source
  SSI2_CPSR_R = 40;                             // set bit rate to 1 MHz (if SR=0 in CR0)
  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
  SSI2_CR1_R |= SSI_CR1_SSE;                    // turn on SSI2

  // Configure AN0 as an analog input
  SYSCTL_RCGCADC_R |= 1;                        // turn on ADC module 0 clocking
        GPIO_PORTE_AFSEL_R |= 0x08;             // select alternative functions for AN0 (PE3)
  GPIO_PORTE_DEN_R &= ~0x08;                    // turn off digital operation on pin PE3
  GPIO_PORTE_AMSEL_R |= 0x08;                   // turn on analog operation on pin PE3
  ADC0_CC_R = ADC_CC_CS_SYSPLL;                 // select PLL as the time base (not needed, since default value)
  ADC0_ACTSS_R &= ~ADC_ACTSS_ASEN3;             // disable sample sequencer 3 (SS3) for programming
  ADC0_EMUX_R = ADC_EMUX_EM3_PROCESSOR;         // select SS3 bit in ADCPSSI as trigger
  ADC0_SSMUX3_R = 0;                            // set first sample to AN0
  ADC0_SSCTL3_R = ADC_SSCTL3_END0;              // mark first sample as the end
  ADC0_ACTSS_R |= ADC_ACTSS_ASEN3;              // enable SS3 for operation
}

int16_t readAdc0Ss3()
{
  ADC0_PSSI_R |= ADC_PSSI_SS3;                  // set start bit
  while (ADC0_ACTSS_R & ADC_ACTSS_BUSY);        // wait until SS3 is not busy
  return ADC0_SSFIFO3_R;                        // get single result from the FIFO
}

//-----------------------------------------------------------------------------
// Main
//-----------------------------------------------------------------------------

int main(void)
{
  // Initialize hardware
  initHw();

  // Turn-on all LEDs to create white backlight
  RED_BL_LED = 1;
  GREEN_BL_LED = 1;
  BLUE_BL_LED = 1;

  // Initialize graphics LCD
  initGraphicsLcd();

  // Draw legend
  setGraphicsLcdTextPosition(0, 0);
  putsGraphicsLcd("Raw ADC");
  setGraphicsLcdTextPosition(0, 1);
  putsGraphicsLcd("Unfiltered (C)");
  setGraphicsLcdTextPosition(0, 2);
  putsGraphicsLcd("Filtered (C)");

  // Display raw ADC value and temperatures
  uint16_t raw;
  float instantTemp, iirTemp;
  char str[10];
  float alpha = 0.99;
  int firstUpdate = true;
  while(1)
  {
          // Read sensor
    raw = readAdc0Ss3();
    // Calculate temperature in degC as follows:
    //   For the 12-bit SAR ADC with Vref+ = 3.3V and Vref- = 0V, outputing a result R:
    //   Resolution is approx 0.81mV / LSb or 0.13 degC / LSb
    //   R(Vin) = floor(Vin/3.3V * 4096) -> Vin(R) ~= 3.3V * ((R+0.5) / 4096)
    //   (~ and 0.5LSb offset in Vin(R) equation are introduced for mid-tread value of the SAR transfer function)
    //   T(Vin) = (Vin - 0.424V) / 0.00625V
    //   T(R) ~= ([3.3V * ((R+0.5) / 4096)] - 0.424V) / 0.00625V
    //   T(R) ~= (0.12890625 * R) - 67.775546875 (simplified floating point equation to save cycles)
    instantTemp = ((raw / 4096.0 * 3.3) - 0.424) / 0.00625;
    // First order IIR filter
    //   In the z-domain:
    //     H(z) = sum(j = 0..M) {bj * z^(-j)
    //            ----------------------------
    //            sum(i = 0..N) {ai * z^(-i)
    //   Setting a0 = 1, yields:
    //     H(z) = sum(j = 0..M) {bj * z^(-j)
    //            ----------------------------
    //             1 + sum(i = 1..N) {ai * z^(-i)
    //   for N = 1, M = 0:
    //     H(z) = b0 / [1 + a1 * z^(-1)]
    //   Given IIR difference equation:
    //     sum(i = 0..N) {ai * y(n-i)} = sum(j = 0..M) {bj * x(n-j)}
    //   Separating y(n), rearranging and inverting signs of a(1-N), yields
    //     a0 * y(n) = sum(i = 1..N) {ai * y(n-i)} + sum(j = 0..M) {bj * x(n-j)}
    //     for N = 1, M = 0, and a0 = 1,
    //       y(n) = b0 * x(n) + a1 * y(n-1)
    //   Setting b0 = (1-a1), yields
    //     y(n) = alpha * y(n-1) + (1-alpha) * x(n)
    //   Adding an exception for the first sample, yields:
    //     y(n) = x(n); n = 0
    //     y(n) = alpha * y(n-1) + (1-alpha) * x(n); n > 0
    if (firstUpdate)
    {
            iirTemp = instantTemp;
      firstUpdate = false;
    }
    else
            iirTemp = iirTemp * alpha + instantTemp * (1-alpha);

    // display raw ADC value and temperatures
    sprintf(str, "%u", raw);
    setGraphicsLcdTextPosition(100, 0);
    putsGraphicsLcd(str);
    sprintf(str, "%3.1f", instantTemp);
    setGraphicsLcdTextPosition(100, 1);
    putsGraphicsLcd(str);
    sprintf(str, "%3.1f", iirTemp);
    setGraphicsLcdTextPosition(100, 2);
    putsGraphicsLcd(str);
    waitMicrosecond(500000);
  }
}
```