```
/**-----------------------------------------------------------------------------

   \file main.cpp
--                                                                            --
--              ECEN 5803 Mastering Embedded System Architecture              --
--                  Project 1 Module 4                                        --
--                Microcontroller Firmware                                    --
--                      main.cpp                                              --
--                                                                            --
-------------------------------------------------------------------------------
--
--  Designed for:  University of Colorado at Boulder
--
--
--  Designed by:  Tim Scherr
--  Revised by:  David James and Ismail Yesildirek
--
-- Version: 2.0.2
-- Date of current revision:  2018-10-04
-- Target Microcontroller: Freescale MKL25ZVMT4
-- Tools used:  ARM mbed compiler
--              ARM mbed SDK
--              Freescale FRDM-KL25Z Freedom Board
--
--
-- Functional Description:  Main code file generated by mbed, and then
--                           modified to implement a super loop bare metal OS.
--
--      Copyright (c) 2015, 2016 Tim Scherr  All rights reserved.
--
*/

#define MAIN
#include "shared.h"
#include "math.h"
#include "TestData.h"
#include "MKL25Z4.h"

#undef MAIN

#define ADC_0                   (0U)
#define CHANNEL_0               (0U)
#define CHANNEL_1               (1U)
#define CHANNEL_2               (2U)
#define LED_ON                  (0U)
#define LED_OFF                 (1U)
#define ADCR_VDD                (65535U)    /*! Maximum value when use 16b resolution */
#define V_BG                    (1000U)     /*! BANDGAP voltage in mV (trim to 1.0V) */
#define V_TEMP25                (716U)      /*! Typical VTEMP25 in mV */
#define M                       (1620U)     /*! Typical slope: (mV x 1000)/oC */
#define STANDARD_TEMP           (25)

/*************************************************************
 * Define constants bluff body width (d - inches)
 * and pipe inner diameter (PID - inches)
 *************************************************************/
#define d_width 0.5 //inches
#define PID 2.900 //inches
#define PIDm 0.07366 //meters
#define sample_period 0.0001 // 100us


extern volatile uint16_t SwTimerIsrCounter;

Ticker tick;            //  Creates a timer interrupt using mbed methods
 /***************      ECEN 5803 add code as indicated   ***************/
                // Add code to control red, green and blue LEDs here

 uint32_t frequency = 0.0f; //for the frequency calculation
 uint32_t temperature = 2300; //room temperature, Celsius (x100)
 uint32_t St_int = 0; //St integration for averaging
```

```
 72    uint32_t Re = 1500000; //initialize Re between 10,000 and 10,000,000
 73    uint16_t iters = 0;   //keep track of iterations for St average
 74    uint32_t Flow = 0; //<----the purpose of this whole program
 75
 76       unsigned char c_spi;
 77    //These variables can be made available to other files
 78    //uint32_t viscosity = 0;
 79    //uint32_t rho_density = 0;
 80    //uint32_t St_const = 0;
 81    //uint32_t velocity = 0;
 82
 83    /*************************************************************
 84     * ADC/SPI tutorial in book: Freescale ARM Cortex-M
 85     * Embedded Programming: Using C Language (ARM books Book 3)
 86     *************************************************************/
 87     void ADC0_init(void);
 88     void ADC1_init(void);
 89     void ADC2_init(void);
 90     void SPI0_init(void);
 91     void SPI0_write(unsigned char * data, int size);
 92   /*************************************************************
 93    * Read raw analog data (frequency and temperature)
 94    * from the flowmeter.
 95    *************************************************************/
 96   void read_internal_temp()
 97   {
 98   uint16_t internal_temp =0;
 99   ADC0->SC1[0] = 26; /* start conversion on channel 26 temperature */
100   while(!(ADC0->SC1[0] & 0x80)) { }
101   /* wait for COCO to be set to 1 at bit7 of register SC1*/
102   internal_temp = ADC0->R[0]; /* read conversion result and clear COCO flag */
103   //printf("Internal temp is: %d", internal_temp);
104   /*The sample number is internal_temp @16bit resolution*/
105   }
106   void readADC()
107   {
108
109
110
111   }
112   void read_vrefl()
113   {
114   uint16_t ptb0_vrefl = 0;
115   ADC0->SC1[0] = 30; /* start conversion on channel 30 VREFL */
116   while(!(ADC0->SC1[0] & 0x80)) { }
117   /* wait for COCO to be set to 1 at bit7 of register SC1*/
118   ptb0_vrefl = ADC0->R[0]; /* read conversion result and clear COCO flag */
119   /*ptb0_vrefl value is from 0 to 255*/
120   //printf("Internal VREFL is: %d", ptb0_vrefl);
121   }
122   void read_FREQ()
123   {
124
125       //This algorithm mirrors the Simulink diagram in the report
126       uint16_t i = 0; //index
127       uint16_t prev_edge = 0; //for T calculation
128       uint16_t this_edge = 0; //for T calculation
129       uint16_t max = 0; //for the sample max
130       uint16_t edges = 0; //edge count
131       uint16_t period = 0; //current period
132       uint32_t period_int = 0;//running total, discrete integration
133
134       bool high = false;
135     // we've got a 1000 sample block to work with:
136       for(i = 0; i<1000; i++)
137       {//look for a value that hits 0.9*max
138           if(ADCbuffer[i]>0.9*max)
139           {
140             if(!high) //rising edge
141             {
142               high = !high; //easier to find the period of a square wave...
```

```
143                 prev_edge = this_edge; //update prev_edge
144                 this_edge = i;        //then update this_edge
145                 edges++; //keep count for the average
146                 period = this_edge - prev_edge; //period in samples
147                 period_int+=period; //running total for the average
148               } //if the current value is the largest we've seen, set it as the max
149               if(ADCbuffer[1] > max) max=ADCbuffer[i];
150             }//if we drop below 90%, we aren't near a maximum anymore
151             else high = false;
152         }//convert from Z to t
153         float period_avg = sample_period*(float)period_int/(edges-1);
154         // f=1/T, 100 is a scalar for lossless integer math
155         frequency = 100/period_avg; //
156         //frequency = 39948; // uncomment for a constant frequency
157 
158         // set it up so that the temperature doesn't change any more than plus/minus 1 degree
159         temperature += (powf(-1.0,(rand()%2)))*(rand()%2); // T in Celsius (x100)
160         //temperature = 2300; // uncomment for constant room temperature
161     }
162 
163     /*****************************************************************
164      *
165      *
166      *****************************************************************/
167     void calculate_flow()
168     {
169         iters += 1;
170        uint32_t temperatureK = temperature + 27315; //Kelvin (x100)
171        //uint32_t temperatureD = (temperature * 9.0f/5.0f) + 3200; //Fahrenheit (x100)
172       //Calculate values per equations provided.
173         uint32_t viscosity = 24*powf(10,24780.0f/((float)temperatureK - 14000.0f)); // (x1,000,000)
174         //viscosity = 24*powf(10,24780.0f/((float)temperatureK - 14000.0f)); // (x1,000,000)
175         uint32_t rho_density = 1000*( 1- (((float)temperature+28894.14)/
176                                           (508929.2*((float) temperature+6812.963 )))
177                                             *powf((((float)temperature*0.01)-3.9863),2)) ; // 1:1
178         //rho_density = 1000*( 1- (((float)temperature+28894.14)/
179         //                                   (508929.2*((float) temperature+6812.963 )))
180         //                                     *powf((((float)temperature*0.01)-3.9863),2)) ; // 1:1
181         uint32_t St = 2684-10356/powf(Re,0.5); // (x10,000)
182         St_int += St;
183         uint32_t St_const = St_int/iters;
184         St_const = St_int/iters;
185         if(iters>1000){ St_int=St_const; iters=1;}
186         //velocity = 10000*frequency*d_width/St_const; // (x100)
187         uint32_t velocity = 10000*frequency*d_width/St_const; // (x100)
188         //Re*1000000 to adjust for scaling of viscosity (x1,000,000)
189         Re = 1000000*((float)rho_density*((float)velocity/3937)*(PIDm))/(float)viscosity;
190         Flow = 2.45*PID*PID*velocity/12;
191     }
192 
193     /*****************************************************************
194      *
195      *
196      *****************************************************************/
197     void adc_out()
198     {
199 
200     }
201     /*****************************************************************
202      *
203      *
204      *****************************************************************/
205     void Pulse_output()
206     {
207 
208     }
209     /*****************************************************************
210      *
211      *
212      *****************************************************************/
213     void LCD_Display()
```

```
214   {
215
216   }
217
218   int main()
219   {
220   /****************      ECEN 5803 add code as indicated   ***************/
221                       //  Add code to call timer0 function every 100 uS
222       tick.attach(&timer0, 0.0001); // setup ticker to call flip every 100 microseconds
223       uint32_t  count = 0;
224
225   // initialize serial buffer pointers
226      rx_in_ptr =  rx_buf; /* pointer to the receive in data */
227      rx_out_ptr = rx_buf; /* pointer to the receive out data*/
228      tx_in_ptr =  tx_buf; /* pointer to the transmit in data*/
229      tx_out_ptr = tx_buf; /*pointer to the transmit out */
230
231
232       /****************      ECEN 5803 add code as indicated   ***************/
233       // uncomment this section after adding monitor code.
234
235      UART_direct_msg_put("\r\nCode ver. ");
236      UART_direct_msg_put( CODE_VERSION );
237      UART_direct_msg_put("\r\n");
238      UART_direct_msg_put( COPYRIGHT );
239      UART_direct_msg_put("\r\n");
240
241      set_display_mode();
242       ADC0_init();
243       ADC1_init();
244       ADC2_init();
245       SPI0_init(); /* enable SPI0 */
246       unsigned char display_flow[14] = {'F','l','o','w','(','G','P','M',')',' ',
247       '1','2','8','3'};
248       unsigned char display_frequency[15] = {'F','r','e','q','(','H','z',')',' ',
249       '3','9','9','.','5','9'};
250       unsigned char display_temp[13] = {'T','e','m','p','(','C',')',' ',
251       '2','3','.','4','0'};
252     //  unsigned char flow_char [7];
253     //  unsigned char freq_char [6];
254     //  unsigned char temp_char[5];
255       while(1)       // Cyclical Executive Loop
256       {
257         //  read_FREQ();        //reads ADC buffer and calculates the frequency
258           readADC();
259           read_vrefl(); //reads ADC ch0
260           read_internal_temp(); //reads ADC ch2
261           calculate_flow();   //calculates volumentric flow in Gallons per minute
262
263           count++;                  // counts the number of times through the loop
264           serial();            // Polls the serial port
265           chk_UART_msg();     // checks for a serial port message received
266           monitor();           // Sends serial port output messages depending
267                             //  on commands received and display mode
268
269      /****************      ECEN 5803 add code as indicated   ***************/
270
271   //  4-20 output ()    // use TMP0 channel 3  proporional rate to flow
272
273   //  Pulse output()   // use TMP0 channel 4  propotional rate to frequency
274
275   //  LCD_Display()   // use the SPI port to send flow number
276
277         /*Send flow data to LCD*/
278         for(int i =0;i<13;i++)
279           {
280             c_spi = display_flow[i];
281             //printf("Flow is: %c", display_flow[i]);
282           }
283           SPI0_write(display_flow,11); //send data through SPI to LCD
284
```

```
285           /*Send frequency data to LCD*/
286           for (int i =0;i<14;i++)
287           {
288             c_spi = display_frequency[i];
289         //    SPI0_write(c_spi);
290           }
291           SPI0_write(display_frequency,15); //send data through SPI to LCD
292           /*Send temp data to LCD*/
293           for (int i =0;i<12;i++)
294           {
295             c_spi = display_temp[i];
296           }
297           SPI0_write(display_temp,13); //send data through SPI to LCD
298   }
299   //  End ECEN 5803 code addition
300
301       }
302
303     void ADC0_init(void)
304   {
305   SIM->SCGC5 |= 0x0400; /* clock to PORTB */
306   PORTB->PCR[0] = 0; /* PTB0 analog input */
307   SIM->SCGC6 |= 0x8000000; /* clock to ADC0 */
308   ADC0->SC2 &= ~0x40; /* software trigger */
309   /* no low power, clock div by 1, long sample time, single ended 8 bit, bus clock */
310   ADC0->CFG1 = 0x01 | 0x10 | 0x00 | 0x00;
311   }
312   /*Re use ADC0 to set up other pins*/
313     void ADC1_init(void)
314   {
315   SIM->SCGC5 |= 0x0400; /* clock to PORTB */
316   PORTB->PCR[1] = 0; /* PTB1 analog input */
317   SIM->SCGC6 |= 0x8000000; /* clock to ADC1 */
318   ADC0->SC2 &= ~0x40; /* software trigger */
319   /* no low power, clock div by 1, long sample time, single ended 16 bit, bus clock */
320   ADC0->CFG1 = 0x01 | 0x10 | 0x03 | 0x00;
321   }
322     void ADC2_init(void)
323   {
324   SIM->SCGC5 |= 0x0400; /* clock to PORTB */
325   PORTB->PCR[2] = 0; /* PTB2 analog input */
326   SIM->SCGC6 |= 0x8000000; /* clock to ADC2 */
327   ADC0->SC2 &= ~0x40; /* software trigger */
328   /* no low power, clock div by 1, long sample time, single ended 16 bit, bus clock */
329   ADC0->CFG1 = 0x01 | 0x10 | 0x03 | 0x00;
330   }
331
332   void SPI0_init(void) {
333   SIM->SCGC5 |= 0x0800; /* enable clock to Port C */
334     /*Set ports to alternative 2 for SPI*/
335   PORTC->PCR[4] = 0x200; /* make PTC4 pin as SPI0 PCS0 */
336   PORTC->PCR[5] = 0x200; /* make PTC5 pin as SPI0 SCK */
337   PORTC->PCR[6] = 0x200; /* make PTC6 pin as SPI0 MOSI */
338   PORTC->PCR[7] = 0x200; /* make PTC7 pin as SPI0 MISO */
339   PTC->PDDR |= 0x01; /* make PTC0 as output pin for /SS */
340   PTC->PSOR = 0x01; /* make PTC0 idle high */
341   SIM->SCGC4 |= 0x400000; /* enable clock to SPI0 */
342   SPI0->C1 = 0x10; /* disable SPI and make SPI0 master */
343   SPI0->BR = 0x60; /* set Baud rate to 1 MHz */
344   SPI0->C1 |= 0x40; /* Enable SPI module */
345   }
346   void SPI0_write(unsigned char * data,int size) {
347   volatile char dummy;
348   PTC->PCOR = 1; /* assert /SS */
349   while(!(SPI0->S & 0x20)) { } /* wait until tx ready */
350   //send all data then clear and exit.
351   for (int i=0;i<size;i++){
352   SPI0->D = data[i]; /* send data byte */
353   while(!(SPI0->S & 0x80)) { } /* wait until tx complete */
354   }
355   dummy = SPI0->D; /* clear SPRF */
```

```
356   PTD->PSOR = 1; /* deassert /SS */
357   }
358
```