HØGSKOLEN I BUSKERUD
OG VESTFOLD

THE SENSETION PROJECT
SOURCE CODE

EN-2015-01

Torgrim Rudland Næss
Sindre Røstad Følke
Mette Varegg

20.05.2015

HØGSKOLEN I BUSKERUD
OG VESTFOLD

HØGSKOLEN I BUSKERUD OG VESTFOLD

Table of Contents

HØGSKOLEN I BUSKERUD OG VESTFOLD

1 ZP2015 Source Files

1.1 LMP91000.c

```
1.  /*
2.  * ZP2015 - LMP91000.c
3.  * Author: Torgrim Rudland Næss
4.  */
5.
6.  #include <msp430.h>
7.  #include "macros.h"
8.  #include "timer.h"
9.  #include "LMP91000.h"
10. #include "i2c.h"
11.
12. /* Control the demux to enable I2C communication for LMP91000 */
13. void enable_LMP91000_i2c(unsigned char device_number)
14. {
15.     // P2.0-2 controls the demux's address selection.
16.     switch (device_number)
17.     {
18.     case 0:
19.         // Set 3-bit address sequence on P2.0-2.
20.         bit_clear(P2OUT, BIT0);
21.         bit_clear(P2OUT, BIT1);
22.         bit_clear(P2OUT, BIT2);
23.
24.         // Activate enable pin on demux
25.         DECODER_ENABLE;
26.
27.         // 1 ms delay to make sure MENB have time to be set low before proceeding
28.         wait(1);
29.         break;
30.     case 1:
31.         bit_set(P2OUT, BIT0);
32.         bit_clear(P2OUT, BIT1);
33.         bit_clear(P2OUT, BIT2);
34.         DECODER_ENABLE;
35.         wait(1);
36.         break;
37.     case 2:
38.         bit_clear(P2OUT, BIT0);
39.         bit_set(P2OUT, BIT1);
40.         bit_clear(P2OUT, BIT2);
41.         DECODER_ENABLE;
42.         wait(1);
43.         break;
44.     case 3:
45.         bit_set(P2OUT, BIT0);
46.         bit_set(P2OUT, BIT1);
47.         bit_clear(P2OUT, BIT2);
48.         DECODER_ENABLE;
49.         wait(1);
50.         break;
51.     case 4:
52.         bit_clear(P2OUT, BIT0);
```

```
53.             bit_clear(P2OUT, BIT1);
54.             bit_set(P2OUT, BIT2);
55.             DECODER_ENABLE;
56.             wait(1);
57.             break;
58.         case 5:
59.             bit_set(P2OUT, BIT0);
60.             bit_clear(P2OUT, BIT1);
61.             bit_set(P2OUT, BIT2);
62.             DECODER_ENABLE;
63.             wait(1);
64.             break;
65.         case 6:
66.             bit_clear(P2OUT, BIT0);
67.             bit_set(P2OUT, BIT1);
68.             bit_set(P2OUT, BIT2);
69.             DECODER_ENABLE;
70.             wait(1);
71.             break;
72.         case 7:
73.             bit_set(P2OUT, BIT0);
74.             bit_set(P2OUT, BIT1);
75.             bit_set(P2OUT, BIT2);
76.             DECODER_ENABLE;
77.             wait(1);
78.             break;
79.         }
80.     }
81.
82.     /* Set an LMP91000 to 3-lead amperometric cell mode */
83.     void LMP91000_three_lead_amperometric_mode(unsigned char device_number)
84.     {
85.         // Set MENB pin low to enable I2C communication
86.         enable_LMP91000_i2c(device_number);
87.
88.         // Initialize I2C communication with LMP91000
89.         i2c_initialize(LMP_I2C_ADDR, WRITE);
90.
91.         // Transmit Mode control register address
92.         i2c_transmitByte(MODECN);
93.
94.         // Set device to 3-lead amperometric cell mode
95.         i2c_transmitByte(Three_Lead_Amperometric);
96.
97.         // Generate stop condition
98.         i2c_stop();
99.
100.        // Set MENB pin high to disable I2C communication
101.        DISABLE_LMP91000_I2C;
102.    }
103.
104.
105.    /* Set an LMP91000 to temperature measurement mode (TIA on) */
106.    void LMP91000_temperature_mode(unsigned char device_number)
107.    {
108.        // Set MENB pin low to enable I2C communication
109.        enable_LMP91000_i2c(device_number);
110.
```

```
111.     // Initialize I2C communication with LMP91000
112.     i2c_initialize(LMP_I2C_ADDR, WRITE);
113.
114.     // Transmit Mode control register address
115.     i2c_transmitByte(MODECN);
116.
117.     // Set device to temperature measurement (TIA on)
118.     i2c_transmitByte(Temperature_meas_TIA_On);
119.
120.     // Generate stop condition
121.     i2c_stop();
122.
123.     // Set MENB pin high to disable I2C communication
124.     DISABLE_LMP91000_I2C;
125. }
126.
127.
128. /* Set an LMP91000 to deep sleep mode */
129. void LMP91000_deep_sleep_mode(unsigned char device_number)
130. {
131.     // Set MENB pin low to enable I2C communication
132.     enable_LMP91000_i2c(device_number);
133.
134.     // Initialize I2C communication with LMP91000
135.     i2c_initialize(LMP_I2C_ADDR, WRITE);
136.
137.     // Transmit Mode control register address
138.     i2c_transmitByte(MODECN);
139.
140.     // Set device to deep sleep mode
141.     i2c_transmitByte(Deep_sleep);
142.
143.     // Generate stop condition
144.     i2c_stop();
145.
146.     // Set MENB pin high to disable I2C communication
147.     DISABLE_LMP91000_I2C;
148. }
149.
150.
151. /* Puts all LMP91000 devices in deep sleep mode */
152. void LMP_all_sleep()
153. {
154.     unsigned char device;
155.
156.     for (device = 0; device < 8; device++)
157.     {
158.         LMP91000_deep_sleep_mode(device);
159.     }
160. }
161.
162.
163. /*
164.  * Initialize all LMP91000 devices.
165.  * Based on Pot_Init from sensors.c
166.  * Line 205-241 are the same, but with some new comments.
167.  * The gain in line 235 is chosen from function parameter
168.  * instead of fixed external resistance as in ez Sense.
```

```
 */
void initialize_LMP91000(unsigned char UART_buffer)
{
    unsigned char device_number;
    unsigned char gain;

    // Choose TIA gain
    switch (UART_buffer)
    {
    case '0': { gain = Gain_2p75KOhm; break; }
    case '1': { gain = Gain_3p5KOhm; break; }
    case '2': { gain = Gain_7KOhm; break; }
    case '3': { gain = Gain_14KOhm; break; }
    case '4': { gain = Gain_35KOhm; break; }
    case '5': { gain = Gain_120KOhm; break; }
    case '6': { gain = Gain_350KOhm; break; }
    case '7': { gain = Gain_External_Resistance; break; }
    }


    // Initialize all eight LMP91000 devices
    for (device_number = 0; device_number < 8; device_number++)
    {
        // Set MENB pin low
        enable_LMP91000_i2c(device_number);

        // Initialize I2C communication with LMP91000
        i2c_initialize(LMP_I2C_ADDR, WRITE);

        // Transmit Mode control register address
        i2c_transmitByte(MODECN);

        // Set device to 3-lead amperometric cell mode
        i2c_transmitByte(Three_Lead_Amperometric);

        // Generate stop condition
        i2c_stop();

        // Protection register: TIACN and REFCN write mode
        i2c_initialize(LMP_I2C_ADDR, WRITE);
        i2c_transmitByte(LOCKCN);
        i2c_transmitByte(Write_TIACN_REFCN);
        i2c_stop();

        /* Set the Reference Control Register to select the following parameters:
        * External reference voltage     | Internal zero selection = VREF*0.2 |
        * Bias polarity = positive    | BIAS selection = VREF*0.24        */
        i2c_initialize(LMP_I2C_ADDR, WRITE);
        i2c_transmitByte(REFCN);
        i2c_transmitByte(External_Vref | Internal_Zero_20
            | Bias_Positive | Bias_24_Percent);
        i2c_stop();

        // Set the TIA Control Register to select gain and load resistance values
        i2c_initialize(LMP_I2C_ADDR, WRITE);
        i2c_transmitByte(TIACN);
        i2c_transmitByte(gain | Rload_100_Ohm);
        i2c_stop();
```

```
227.
228.         // Protection register: TIACN and REFCN read only mode
229.         i2c_initialize(LMP_I2C_ADDR, WRITE);
230.         i2c_transmitByte(LOCKCN);
231.         i2c_transmitByte(Read_TIACN_REFCN);
232.         i2c_stop();
233.
234.         // Set MENB pin high to disable I2C communication
235.         DISABLE_LMP91000_I2C;
236.     }
237. }
238.
239.
240. /* ------------------------ Functions for troubleshooting ----------------------------*/
241.
242. /*
243. * Check for LMP91000 response on I2C bus.
244. * Returns 1 if the device is responding, otherwise 0
245. */
246. int check_LMP_status(unsigned char device_number)
247. {
248.     enable_LMP91000_i2c(device_number);
249.     i2c_initialize(LMP_I2C_ADDR, WRITE);
250.
251.     /* Wait 1 ms before checking if the USCI_B0 status register
252.     * not-acknowledge interrupt flag has been set */
253.     wait(1);
254.
255.     if ((UCB0STAT & UCNACKIFG) == UCNACKIFG)
256.     {
257.         // set MENB pin high and clear the flag and return 0 if the flag is set
258.         DISABLE_LMP91000_I2C;
259.         UCB0STAT &= ~UCB0STAT;
260.         return 0;
261.     }
262.     else
263.     {
264.         // dummy byte needed for i2c stop to work
265.         i2c_transmitByte(0x00);
266.         i2c_stop();
267.         DISABLE_LMP91000_I2C;
268.         return 1;
269.     }
270. }
271.
272.
273. /* Turn on LED when device is responding to I2C communication */
274. void test_LMP_status(unsigned char device_number)
275. {
276.     int response = check_LMP_status(device_number);
277.     if (response)
278.         // Turn on led if device responds
279.         LED_ON;
280.     else
281.         LED_OFF;
282. }
283.
284.
```

```
285. /*
286. * Give values to the array containing LMP91000 status
287. * Array is used as parameter in alternate version of initialize_LMP91000()
288. */
289. void set_LMP_status_array(unsigned char status_arr[])
290. {
291.     int device_number;
292. 
293.     for (device_number = 0; device_number < 8; device_number++)
294.     {
295.         // 1 if device responds, 0 if not
296.         status_arr[device_number] = check_LMP_status(device_number);
297.     }
298. }
299. 
300. 
301. /* Alternate version of the initiation function. In main, fill an 8-bit array with 1's
302.  * or 0's using set_LMP_status_array(). The array is passed as a parameter to this
303.  * function so that non-responsive LMP's are skipped in the initialization process.
304.  */
305. /*void initialize_LMP91000(unsigned char LMP_status[], unsigned char gainSetting)
306. {
307.     unsigned char device_number;
308.     unsigned char gain;
309. 
310.     // Choose TIA gain
311.     switch (UART_buffer)
312.     {
313.         case '0': { gain = Gain_2p75KOhm; break; }
314.         case '1': { gain = Gain_3p5KOhm; break; }
315.         case '2': { gain = Gain_7KOhm; break; }
316.         case '3': { gain = Gain_14KOhm; break; }
317.         case '4': { gain = Gain_35KOhm; break; }
318.         case '5': { gain = Gain_120KOhm; break; }
319.         case '6': { gain = Gain_350KOhm;break; }
320.         case '7': { gain = Gain_External_Resistance; break; }
321.     }
322. 
323.     for (device_number = 0; device_number < 8; device_number++)
324.     {
325.         // Check for I2C response before initializing
326.         if (LMP_status[device_number] != 0)
327.         {
328.             // Set MENB pin low
329.             enable_LMP91000_i2c(device_number);
330. 
331.             // Initialize I2C communication with LMP91000
332.             i2c_initialize(LMP_I2C_ADDR,WRITE);
333. 
334.             // Transmit Mode control register address
335.             i2c_transmitByte(MODECN);
336. 
337.             // Set device to 3-lead amperometric cell mode
338.             i2c_transmitByte(Three_Lead_Amperometric);
339. 
340.             // Generate stop condition
341.             i2c_stop();
342. 
```

```
343.             // Protection register: TIACN and REFCN write mode
344.             i2c_initialize(LMP_I2C_ADDR,WRITE);
345.             i2c_transmitByte(LOCKCN);
346.             i2c_transmitByte(Write_TIACN_REFCN);
347.             i2c_stop();
348.
349.             /* Set the Reference Control Register to select the following parameters:
350.             * External reference voltage     | Internal zero selection = VREF*0.2 |
351.             * Bias polarity = positive    | BIAS selection = VREF*0.24        */ /*
352.             i2c_initialize(LMP_I2C_ADDR,WRITE);
353.             i2c_transmitByte(REFCN);
354.             i2c_transmitByte(External_Vref | Internal_Zero_20
355.             | Bias_Positive | Bias_24_Percent);
356.             i2c_stop();
357.
358.             // Set the TIA Control Register to select gain and load resistance values
359.             i2c_initialize(LMP_I2C_ADDR,WRITE);
360.             i2c_transmitByte(TIACN);
361.             i2c_transmitByte(gain | Rload_100_Ohm);
362.             i2c_stop();
363.
364.             // Protection register: TIACN and REFCN read only mode
365.             i2c_initialize(LMP_I2C_ADDR,WRITE);
366.             i2c_transmitByte(LOCKCN);
367.             i2c_transmitByte(Read_TIACN_REFCN);
368.             i2c_stop();
369.
370.             // Set MENB pin high to disable I2C communication
371.             DISABLE_LMP91000_I2C;
372.         }
373.     }
374. }*/
```

1.2 ADS1115.c

```
1.  ////////////////////////////////////////////////////////////////////////////
2.  //
3.  //  Module Name               : ADS1115 A/D converter
4.  //  File Name                 : ADS1115.c
5.  //  Purpose                   : Functions to control the ADS1115
6.  //  Author                    : Torgrim Rudland Næss
7.  //
8.  ////////////////////////////////////////////////////////////////////////////
9.
10. #include <msp430.h>
11. #include "timer.h"
12. #include "ADS1115.h"
13. #include "i2c.h"
14. #include "stdio.h"
15. #include "LMP91000.h"
16. #include "serial.h"
17.
18. /*
19. * Start a conversion on a single channel, decided by channel_number.
20. * Channel 1 to 4 use the ADS1115 with i2c address 1001 001 (AD_I2C_ADDR0)
21. * and channel 5-8 use the address 1001 011 (AD_I2C_ADDR1).
22. * Based on of ADS_DoConversion() in sensors.c.
23. * Code line 29-42 is the same, but the data transmitted is changed.
24. * The switch statement for channel selection is new.
25. */
26. void ADS_DoConversion(unsigned char channel_number)
27. {
28.     switch (channel_number)
29.     {
30.     case 0:
31.         // Initialize I2C communication with ADS1115
32.         i2c_initialize(AD_I2C_ADDR0, WRITE);
33.
34.         // Transmit config register address
35.         i2c_transmitByte(AD_CONF_REG);
36.
37.         // Single conversion on AIN0. FS = +/- 2.048V. Power-down single-shot mode
38.         i2c_transmitByte(ADS1115_MUX_AIN0);
39.
40.         // Data rate = 860 samples per second. Disable comparator
41.         i2c_transmitByte(ADS1115_CFG_RATE_860 | ADS1115_DISABLE_COMP);
42.
43.         // generate stop condition
44.         i2c_stop();
45.         break;
46.     case 1:
47.         i2c_initialize(AD_I2C_ADDR0, WRITE);
48.         i2c_transmitByte(AD_CONF_REG);
49.         i2c_transmitByte(ADS1115_MUX_AIN1);
50.         i2c_transmitByte(ADS1115_CFG_RATE_860 | ADS1115_DISABLE_COMP);
51.         i2c_stop();
52.         break;
53.     case 2:
54.         i2c_initialize(AD_I2C_ADDR0, WRITE);
55.         i2c_transmitByte(AD_CONF_REG);
56.         i2c_transmitByte(ADS1115_MUX_AIN2);
```

```
57.           i2c_transmitByte(ADS1115_CFG_RATE_860 | ADS1115_DISABLE_COMP);
58.           i2c_stop();
59.           break;
60.       case 3:
61.           i2c_initialize(AD_I2C_ADDR0, WRITE);
62.           i2c_transmitByte(AD_CONF_REG);
63.           i2c_transmitByte(ADS1115_MUX_AIN3);
64.           i2c_transmitByte(ADS1115_CFG_RATE_860 | ADS1115_DISABLE_COMP);
65.           i2c_stop();
66.           break;
67.       case 4:
68.           i2c_initialize(AD_I2C_ADDR1, WRITE);
69.           i2c_transmitByte(AD_CONF_REG);
70.           i2c_transmitByte(ADS1115_MUX_AIN0);
71.           i2c_transmitByte(ADS1115_CFG_RATE_860 | ADS1115_DISABLE_COMP);
72.           i2c_stop();
73.           break;
74.       case 5:
75.           i2c_initialize(AD_I2C_ADDR1, WRITE);
76.           i2c_transmitByte(AD_CONF_REG);
77.           i2c_transmitByte(ADS1115_MUX_AIN1);
78.           i2c_transmitByte(ADS1115_CFG_RATE_860 | ADS1115_DISABLE_COMP);
79.           i2c_stop();
80.           break;
81.       case 6:
82.           i2c_initialize(AD_I2C_ADDR1, WRITE);
83.           i2c_transmitByte(AD_CONF_REG);
84.           i2c_transmitByte(ADS1115_MUX_AIN2);
85.           i2c_transmitByte(ADS1115_CFG_RATE_860 | ADS1115_DISABLE_COMP);
86.           i2c_stop();
87.           break;
88.       case 7:
89.           i2c_initialize(AD_I2C_ADDR1, WRITE);
90.           i2c_transmitByte(AD_CONF_REG);
91.           i2c_transmitByte(ADS1115_MUX_AIN3);
92.           i2c_transmitByte(ADS1115_CFG_RATE_860 | ADS1115_DISABLE_COMP);
93.           i2c_stop();
94.           break;
95.       }
96.   }
97.
98.
99.   /*
100.  * Return conversion data from ADS1115
101.  * Based on ADS_Read() from sensors.c.
102.  * Changes:
103.  * - i2c address is no longer static, but taken as an function parameter.
104.  * - Added comments
105.  */
106.  unsigned int ADS_Read(unsigned char i2c_address)
107.  {
108.      unsigned int ADC_val;
109.
110.      i2c_initialize(i2c_address, WRITE);
111.      i2c_transmitByte(AD_CONV_REG);
112.      i2c_stop();
113.
114.      i2c_initialize(i2c_address, READ);
```

```
115.
116.     /* Read MSB from ADS1115 conversion register, then rotate places
117.     * and OR with the LSB to create a 16-bit sequence */
118.     ADC_val = i2c_readByte();
119.     ADC_val = ADC_val << 8 | i2c_readByte();
120.     i2c_stop();
121.
122.     return ADC_val;
123. }
124.
125.
126. /* Measure sensor data and temperature values on selected channels and transfer
127. * results to the GUI with uart_transfer().
128. * Instead of performing conversion on channel 1 through 8 in ascending order,
129. * one conversion is initiated on each ADS1115 before values are read:
130. *  1. Channel 1 - begin conversion
131. *  2. Channel 5 - begin conversion
132. *  3. Channel 1 - read conversion data
133. *  4. Channel 5 - read conversion data
134. *  5. Channel 2 - begin conversion
135. *  6. Channel 6 - begin conversion
136. *  7. Channel 2 - read conversion data
137. *  8. Channel 6 - read conversion data
138. *  9. Channel 3 - begin conversion
139. * 10. Channel 7 - begin conversion
140. * 11. Channel 3 - read conversion data
141. * 12. Channel 7 - read conversion data
142. * 13. Channel 4 - begin conversion
143. * 14. Channel 8 - begin conversion
144. * 15. Channel 4 - read conversion data
145. * 16. Channel 8 - read conversion data
146. */
147. void take_measurements(char* UART_buffer)
148. {
149.     unsigned int conversion_data[16];
150.     unsigned char ch_count0;
151.     unsigned char ch_count1;
152.     unsigned char device_numb;
153.
154.     // Set LMP91000 on all channels to 3-lead amperometric mode
155.     for (device_numb = 0; device_numb < 8; device_numb++)
156.     {
157.         LMP91000_three_lead_amperometric_mode(device_numb);
158.     }
159.
160.     // Wait for LMP91000 to finish the transition between operation modes
161.     wait(5);
162.
163.     // Measure glucose on all channels
164.     for (ch_count0 = 0, ch_count1 = 4; ch_count0 < 4 && ch_count1 < 8;
165.         ch_count0++, ch_count1++)
166.     {
167.         // See if the channel has been selected for measurements
168.         if (UART_buffer[ch_count0 + 1] == '1')
169.         {
170.             // Begin a conversion if selected
171.             ADS_DoConversion(ch_count0);
172.         }
```

```
173.
174.         if (UART_buffer[ch_count1 + 1] == '1')
175.         {
176.             ADS_DoConversion(ch_count1);
177.         }
178.
179.         wait(2); // Conversion with data rate 860sps takes about 1.2 ms
180.
181.         if (UART_buffer[ch_count0 + 1] == '1')
182.         {
183.             // Copy config register value if channel has been selected
184.             //conversion_data[ch_count0] = ADS_Read(AD_I2C_ADDR0);
185.             //conversion_data[ch_count0] = 0x1F40;
186.             conversion_data[ch_count0] = ADS_Read(AD_I2C_ADDR0);
187.             //            wait(5);
188.         }
189.         else
190.         {
191.             // 0 if channel has not been selected
192.             conversion_data[ch_count0] = 0;
193.             //        wait(5);
194.         }
195.
196.         if (UART_buffer[ch_count1 + 1] == '1')
197.         {
198.             conversion_data[ch_count1] = ADS_Read(AD_I2C_ADDR1);
199.             //conversion_data[ch_count1] = 0x1F41;
200.             //        wait(5);
201.         }
202.         else
203.         {
204.             conversion_data[ch_count1] = 0;
205.         }
206.     }
207.
208.
209.     // Set all channels to temperature sensor mode
210.     for (device_numb = 0; device_numb < 8; device_numb++)
211.     {
212.         LMP91000_temperature_mode(device_numb);
213.     }
214.
215.     // Wait for LMP91000 to finish the transition between operation modes
216.     wait(5);
217.
218.     // Measure temperature on all channels
219.     for (ch_count0 = 0, ch_count1 = 4; ch_count0 < 4 && ch_count1 < 8;
220.         ch_count0++, ch_count1++)
221.     {
222.         if (UART_buffer[ch_count0 + 1] == '1')
223.         {
224.             ADS_DoConversion(ch_count0);
225.         }
226.
227.         if (UART_buffer[ch_count1 + 1] == '1')
228.         {
229.             ADS_DoConversion(ch_count1);
230.         }
```

```
231.
232.         wait(2);
233.
234.         if (UART_buffer[ch_count0 + 1] == '1')
235.         {
236.             conversion_data[ch_count0 + 8] = ADS_Read(AD_I2C_ADDR0);
237.             //conversion_data[ch_count0 + 8] = 0x55F0;
238.         }
239.         else
240.         {
241.             conversion_data[ch_count0 + 8] = 0;
242.         }
243.
244.         if (UART_buffer[ch_count1 + 1] == '1')
245.         {
246.             conversion_data[ch_count1 + 8] = ADS_Read(AD_I2C_ADDR1);
247.         }
248.         else
249.         {
250.             conversion_data[ch_count1 + 8] = 0;
251.         }
252.     }
253.
254.     uart_transmit((char*)conversion_data, 32);
255.
256.     // Blink LED to indicate data has been sent
257.     LED_ON;
258.     wait(50);
259.     LED_OFF;
260. }
261.
262.
263. /* Check for ADS1115 response on I2C bus. Returns 1 if the device responds, otherwise 0 */
264. int check_ADS_status(unsigned char i2c_address)
265. {
266.     i2c_initialize(i2c_address, WRITE);
267.
268.     /* Wait 1 ms before checking if the USCI_B0 status register
269.     * not-acknowledge interrupt flag has been set */
270.     wait(1);
271.
272.     if ((UCB0STAT & UCNACKIFG) == UCNACKIFG) {
273.         // Clear the flag and return 0 if the flag is set
274.         UCB0STAT &= ~UCB0STAT;
275.         return 0;
276.     }
277.     else
278.     {
279.         // dummy byte needed for i2c stop to work
280.         i2c_transmitByte(0x00);
281.         i2c_stop();
282.         return 1;
283.     }
284. }
```

1.3 main.c.

```
1.  /*
2.  * ZP2015 - main.c
3.  * Author: Torgrim Rudland Næss
4.  * Based on main.c from eZ Sense
5.  * Changes :
6.  * - Unused pins are set as outputs
7.  * - Removed LCD display code
8.  * - Removed LMP91000 initialization on startup
9.  * - New case structure
10. * - Added comments
11. */
12.
13. #include <msp430.h>
14. #include "macros.h"
15. #include "timer.h"
16. #include "watchdog.h"
17. #include "ADS1115.h"
18. #include "serial.h"
19. #include "LMP91000.h"
20. #include "i2c.h"
21.
22. char UART_buffer[16];
23. unsigned char* pNoChar;
24.
25. int main(void)
26. {
27.     // Configure ports and registers
28.
29.     // pin 1, 2 and 3 are inputs for CBUS
30.     P1DIR = 0xF1;
31.     P1REN = 0x0E;
32.     P1OUT = 0x80;
33.
34.     // pin 0-3 set as outputs to control the demux
35.     P2DIR = 0xFF;
36.     P2REN = 0x00;
37.     P2OUT = 0x00;
38.
39.     // Enable peripheral module on bit 1, 2, 4 and 5 (I2C & UART)
40.     P3SEL = 0x36;
41.     P3DIR = 0xC9;
42.     P3REN = 0x00;
43.     P3OUT = 0x00;
44.
45.     // Port 4 not used
46.     P4DIR = 0xFF;
47.     P4REN = 0x00;
48.     P4OUT = 0x00;
49.
50.     /* set DCO frequency to 12 MHz, USCI clock to 93.5 kHz, UART 9600 baud
51.     * TIMERA_ONE_MS must be set in timer.h according to selected clock speed */
52.     DCOCTL = CALDCO_12MHZ;
53.     BCSCTL1 = CALBC1_12MHZ;
54.     BCSCTL2 = 0x00;
55.
56.     // I2C bitrate setting
```

```
57.         UCB0BR0 = 0x80;
58.
59.         // prescaler 128
60.         UCB0BR1 = 0x00;
61.
62.         // uart baud settings 9600 at 12MHz
63.         UCA0BR0 = 0xE2;
64.         UCA0BR1 = 0x04;
65.
66.         timerInitA();
67.         SEI;                // enable interrupts
68.
69.         watchdogDisable();  // Stop watchdog timer to prevent timeout reset
70.
71.         // Infinite loop checks for instructions from the GUI
72.         while (1)
73.         {
74.             // pointer to the number of characters received
75.             pNoChar = uart_Enable_Receiver(&UART_buffer[0]);
76.
77.             /* Hold until the UART line has been idle for 10 bits, indicating
78.             * that all characters have been received. */
79.             while ((UCA0STAT & UCIDLE) != UCIDLE);
80.
81.             // If pointer is not empty - characters have been received
82.             if (*pNoChar != 0)
83.             {
84.                 // Disable UART RX interrupt
85.                 uart_Disable_Receiver();
86.
87.                 switch (UART_buffer[0])
88.                 {
89.                     case '1':
90.                     {
91.                         // Initialize LMP91000 devices
92.                         initialize_LMP91000(UART_buffer[1]);
93.                         break;
94.                     }
95.                     case '2':
96.                     {
97.                         // Perform a A/D conversion
98.                         take_measurements(UART_buffer);
99.                         break;
100.                    }
101.                    case '3':
102.                        // Deep sleep mode for all LMP91000 devices
103.                        LMP_all_sleep();
104.                        break;
105.                    default: break;
106.                }
107.            }
108.        }
109.    }
```

1.4 I2C.c

```
1.  ////////////////////////////////////////////////////////////////////////////
2.  //
3.  //  Module Name              : i2c
4.  //  File Name                : i2c.c
5.  //  Purpose                  : Low-level i2c commands
6.  //  Author                   : Sindre Søpstad
7.  //  Modififed                : Torgrim Rudland Næss
8.  //
9.  //    Changes:
10. //    - Added enable NACK interrupt in i2c_initialize
11. //    - Moved I2C start condition from read in i2c_initialize to i2c_readByte
12. ////////////////////////////////////////////////////////////////////////////
13.
14. #include <msp430.h>
15. #include "i2c.h"
16.
17. void i2c_initialize(unsigned char SLV_addr, unsigned char r_w)
18. {
19.     IE2 &= ~UCA0TXIE;
20.     IE2 &= ~UCA0RXIE;
21.     UCB0CTL1 |= UCSWRST;
22.     UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC;
23.
24.     // Sub-module clock, software reset enable
25.     UCB0CTL1 = UCSSEL_2 + UCSWRST;
26.     UCB0I2CSA = SLV_addr;
27.
28.     // Enable UCSI module B0
29.     UCB0CTL1 &= ~UCSWRST;
30.     IFG2 &= ~(UCB0TXIFG | UCB0RXIFG);
31.
32.     // Enable NACK interrupt
33.     UCB0I2CIE |= UCNACKIE;
34.
35.     if (r_w == 1) // Read
36.     {
37.         // Receiver mode
38.         UCB0CTL1 &= ~UCTR;
39.     }
40.     else // Write
41.     {
42.         UCB0CTL1 |= UCTR;
43.         IFG2 &= ~(UCB0TXIFG | UCB0RXIFG);
44.
45.         // I2C TX, start condition
46.         UCB0CTL1 |= UCTXSTT;
47.
48.         //IntFlagRegister is set when TxBuffer is empty
49.         while ((IFG2 & UCB0TXIFG) == 0);
50.     }
51. }
52.
53. void i2c_stop()
54. {
55.     // I2C stop condition
56.     UCB0CTL1 |= UCTXSTP;
```

```
57.
58.     // Ensure stop condition got sent
59.     while (UCB0CTL1 & UCTXSTP);
60.
61.     // Empty receive buffer
62.     UCB0RXBUF = 0;
63.     IE2 |= UCA0TXIE;
64.     IE2 |= UCA0RXIE;
65. }
66.
67.
68. void i2c_transmitByte(unsigned char data)
69. {
70.     //Slave Write
71.     UCB0TXBUF = data;
72.     while ((IFG2 & UCB0TXIFG) == 0);
73. }
74.
75. unsigned char i2c_readByte()
76. {
77.     unsigned char read_byte;
78.
79.     // I2C TX, start condition
80.     UCB0CTL1 |= UCTXSTT;
81.     while ((IFG2 & UCB0RXIFG) == 0);
82.     read_byte = UCB0RXBUF;
83.     return read_byte;
84. }
```

1.5 serial.c

```
1.  /*****************************************************************
2.  Functions to control the USCI modules
3.  Written 07.05.13 by Tore Syversen, SINTEF ICT
4.  Sindre Søpstad, ZimmerTech
5.  Modified by Torgrim Rudland Næss
6.
7.  Changes:
8.  - removed excess functions
9.  - Added more comments
10. - Changed __interrupt void USCI_RX_ISR (void) to send repeated
11. start condition on I2C NACK
12.
13. *****************************************************************/
14.
15.
16. #include <msp430.h>
17. #include "serial.h"
18. #include "macros.h"
19. #include "timer.h"
20. #include "intrinsics.h"
21. #include "stdio.h"
22. #include "string.h"
23.
24. volatile unsigned char usci_TXint, usci_RXint, uart_TXint, uart_RXint, I2C_NACK,
    I2C_RXint;
25. char* pUART_RX_buffer = NULL;
26. unsigned char NoRXchar;
27.
28.
29. void uart_init()
30. {
31.     CLI;
32.
33.     // USCI clock is sub-master clock, reset mode
34.     UCA0CTL1 = UCSSEL_2 | UCSWRST;
35.     UCA0CTL0 = 0x02;
36.
37.     // SMCLK is USCI clock
38.     UCA0CTL1 = 0x80;
39.
40.     // Enable USCI
41.     UCA0CTL1 &= ~UCSWRST;
42.     IE2 = UCA0TXIE;
43.     SEI;
44. }
45.
46.
47. /* Write one byte at a time into UCA0TXBUF when the transmit interrupt flag is high.
48. * The data is transferred into the transmit shift register and transmitted as soon
49. * as the shift register is ready to accept new data */
50. void uart_transmit(char* pBuffer, unsigned char NoBytes)
51. {
52.     unsigned char i;
53.     uart_init();
54.
55.     for (i = NoBytes; i > 0; i--)
```

```
56.       {
57.           uart_TXint = 0;
58.
59.           // Place byte in the UART output buffer register
60.           UCA0TXBUF = *(pBuffer++);
61.
62.           // Wait for transmit interrupt flag to be set high
63.           while (uart_TXint == 0);
64.       }
65.       wait(5);
66.   }
67.
68.   unsigned char* uart_Enable_Receiver(char* pBuffer)
69.   {
70.       buffer_clear(pBuffer);
71.
72.       // clear number of char received
73.       NoRXchar = 0;
74.
75.       // set RX buffer to correct address
76.       pUART_RX_buffer = pBuffer;
77.       uart_init();
78.
79.       // enable interrupt
80.       IE2 |= UCA0RXIE;
81.
82.       // return address to number of char received. It is incremented when interrupts happen
83.       return (&NoRXchar);
84.   }
85.
86.   void uart_Disable_Receiver()
87.   {
88.       // Disable UART RX interrupt
89.       IE2 &= ~UCA0RXIE;
90.   }
91.
92.   void buffer_clear(char* pBuffer)
93.   {
94.       unsigned char i;
95.       for (i = strlen(pBuffer); i>0; i--)
96.       {
97.           *(pBuffer++) = 0;
98.       }
99.   }
100.
101.
102. #pragma vector = USCIAB0TX_VECTOR
103. __interrupt void USCI_TX_ISR(void)
104. {
105.     if ((IFG2 & UCB0TXIFG) == UCB0TXIFG)
106.     {
107.         usci_TXint = 1;
108.     }
109.
110.     // For I2C, the RX interrupt comes at the TX interrupt vector.
111.     if ((IFG2 & UCB0RXIFG) == UCB0RXIFG)
112.     {
113.         I2C_RXint = 1;
```

```
114.     }
115.
116.     if ((IFG2 & UCA0TXIFG) == UCA0TXIFG)
117.     {
118.         uart_TXint = 1;
119.     }
120.
121.     // reset flag
122.     IFG2 = 0;
123. }
124.
125. #pragma vector = USCIAB0RX_VECTOR
126. __interrupt void USCI_RX_ISR(void)
127. {
128.     if ((UCB0STAT & UCNACKIFG) == UCNACKIFG)
129.     {
130.         // I2C TX, repeated start condition
131.         UCB0CTL1 |= UCTXSTT;
132.     }
133.
134.     if ((IFG2 & UCA0RXIFG) == UCA0RXIFG)
135.     {
136.         *(pUART_RX_buffer++) = UCA0RXBUF;
137.         NoRXchar++;
138.     }
139.
140.     if ((IFG2 & UCB0RXIFG) == UCB0RXIFG)
141.     {
142.         usci_RXint = 1;
143.     }
144.
145.     IFG2 = 0;
146. }
```

1.6 timer.c

```
1.  /*
2.  * ZP2015 - timer.c
3.  * Author: Tore Syversen
4.  * Modified by: Torgrim Rudland Næss
5.  *
6.  * Changes:
7.  * - Removed excess functions
8.  * - Removed some commented-out code
9.  * - Cleared up some comments
10. */
11.
12. #include <msp430.h>
13. #include "macros.h"
14. #include "timer.h"
15. #include "intrinsics.h"
16. #include "watchdog.h"
17.
18. // locally defined variables
19. volatile unsigned int timerWait = 0;
20.
21.
22. /*****************************************************************************
23. Function    :    void wait(DWORD)
24. Description :     wait the specified number of milliseconds
25. Parameters  :    DWORD millisecond
26. Returns     :
27. *****************************************************************************/
28. void wait(unsigned int millisec)
29. {
30.     timerWait = 0;
31.     TIMERA_START;
32.     while (timerWait < millisec)
33.     {
34.         watchdogKick();
35.     }
36.     TIMERA_STOP;
37.     return;
38. }
39.
40.
41. /*****************************************************************************
42. Function    :    void timerInitA(void)
43. Description :     Setup timer A to generate interrupt every ms
44. Bason clock speed 8 MHz
45. Timer A is used for SW timers
46. Parameters  :
47. Returns     :
48. *****************************************************************************/
49. void timerInitA()
50. {
51.     // stop timer
52.     TACTL = 0x0000;
53.
54.     // Set TACCIE0 (enable interrupt on compare)
55.     TACCTL0 = 0x0010;
56.
```

```
57.       // The counter value for selected clock speed, one millisec between interrupts
58.       TACCR0 = TIMERA_ONE_MS;
59.
60.       // Clear timer counter
61.       TAR = 0x0000;
62.
63.       // Setup TimerA control register:
64.       // SMCLK clock source, no input divider, up counting, clear timer,
65.       //    interrupt disabled, cler interrupt flag
66.       TACTL = 0x0214;   // TASSEL.1, TAMC.0,1, TAIE
67.   }
68.
69.
70.   /*****************************************************************************
71.   Function    :    void timerAInterrupt(void)
72.   Description :     Timer A interrupt service routine
73.   Called each ms, updating SW timers
74.   Parameters  :
75.   Returns     :
76.   *****************************************************************************/
77.   #pragma vector= TIMERA0_VECTOR
78.   __interrupt void timerAInterrupt(void)
79.   {
80.       timerWait++;
81.   }
82.
83.   #pragma vector=TIMERA1_VECTOR
84.   __interrupt void taivInterrupt(void)
85.   {
86.       // defined, but not used
87.       __no_operation();
88.   }
```

1.7 watchdog.c

```
1.  ////////////////////////////////////////////////////////////////////////////
2.  //
3.  //  Module Name               : Watchdog
4.  //  File Name                 : watchdog.c
5.  //  Software Package          : Telespor -  Eartag package
6.  //  Purpose of the file       : Define Watchdog functions
7.  //  Author                    : Tore Syversen
8.  //  Modified by               : Torgrim Rudland Næss
9.  //
10. //    Changes:
11. //     - Cleared up some comments
12. //
13. ////////////////////////////////////////////////////////////////////////////
14.
15. #include <msp430x23x0.h>
16. #include "watchdog.h"
17. #include "macros.h"
18.
19.
20. /*****************************************************************************
21. Function    :    void watchdogEnable(void)
22. Description :     Enable the watchdog, timeout after xx ms
23. Parameters  :
24. Returns     :
25. *****************************************************************************/
26. void watchdogEnable()
27. {
28.     // WTDHOLD = 0 (Watchdog timer is not stopped),
29.     // WDTCNTCL = 1 (Clears count value to 0000h)
30.     WDTCTL = WDTPW | WDTCNTCL | WDTSSEL;
31.     return;
32. }
33.
34.
35. /*****************************************************************************
36. Function    :    void watchdogDisable(void)
37. Description :     Disable the watchdog
38. Parameters  :
39. Returns     :
40. *****************************************************************************/
41. void watchdogDisable()
42. {
43.     // WTDHOLD = 1 (Watchdog timer is stopped)
44.     WDTCTL = WDTPW | WDTHOLD | WDTCNTCL | WDTSSEL;
45.     return;
46. }
47.
48.
49. /*****************************************************************************
50. Function    :    void watchdogKick(void)
51. Description :     Reset the watchdog to avoid timeout
52. Parameters  :
53. Returns     :
54. *****************************************************************************/
55. void watchdogKick()
56. {
```

```
57.      // Kicking the watchdog if it is stopped, will start it!
58.      if (!(WDTCTL & WDTHOLD))
59.      {
60.          // Periodically clear an active watchdog
61.          // Password + Count Control + Source Select
62.          WDTCTL = WDTPW | WDTCNTCL | WDTSSEL;
63.      }
64.      return;
65.  }
66.
67.
68.  /*************************************************************************
69.  Function    :    bool watchdogEnabled(void)
70.  Description :     Test if the watchdog is enabled
71.  Parameters  :
72.  Returns     :    True if enabled, false if disabled
73.  *************************************************************************/
74.  unsigned char watchdogEnabled()
75.  {
76.      if (WDTCTL & WDTHOLD)
77.          return  0; //false
78.
79.      return  1; //true
80.  }
```

2 ZP2015 Header Files

2.1 LMP91000.h

```
1.  /*
2.  * ZP2015 - LMP91000.h
3.  * Author: Torgrim Rudland Næss
4.  * Based on LMP91000.h from eZ Sense
5.  * Changes:
6.  * - removed unused definitions
7.  * - Created a mroe structural setup
8.  * - Added definitions for enabling and disabling I2C communication
9.  * - Removed old function prototypes and added new ones
10. */
11.
12. #ifndef _LMP91000_H
13. #define _LMP91000_H
14.
15. #include "macros.h"
16.
17. /* ----------------- LMP91000 I2C address  -------------------- */
18. #define LMP_I2C_ADDR                    0x48
19.
20. /* -- Enable/disable decoder to activate I2C communication ---- */
21. #define DECODER_ENABLE                  bit_set(P2OUT, BIT3)
22. #define DISABLE_LMP91000_I2C            bit_clear(P2OUT, BIT3)
23.
24. /* ------------------- Register addresses --------------------- */
25.
26. // Status register
27. #define STATUS                          0x00
28.
29. // Protection register
30. #define LOCKCN                          0x01
31.
32. // TIA control register
33. #define TIACN                           0x10
34.
35. // Reference control register
36. #define REFCN                           0x11
37.
38. // Mode control register
39. #define MODECN                          0x12
40.
41. /* -- Enable/disable writing to TIACN and REFCN registers ---- */
42. #define Write_TIACN_REFCN               0x00
43. #define Read_TIACN_REFCN                0x01
44.
45. /* ------------------- Operation modes ----------------------- */
46. #define Deep_sleep                      0x00
47. #define Two_lead_gnd                    0x01
48. #define Standby                         0x02
49. #define Three_Lead_Amperometric         0x03
50. #define Temperature_meas_TIA_Off        0x06
51. #define Temperature_meas_TIA_On         0x07
52.
53. /* ---------------- Gain and load resistors ----------------- */
```

```
54.  #define Gain_External_Resistance      0x00
55.  #define Gain_2p75KOhm                0x04
56.  #define Gain_3p5KOhm                 0x08
57.  #define Gain_7KOhm                   0x0C
58.  #define Gain_14KOhm                  0x10
59.  #define Gain_35KOhm                  0x14
60.  #define Gain_120KOhm                 0x18
61.  #define Gain_350KOhm                 0x1C
62.
63.  #define Rload_10_Ohm                 0x00
64.  #define Rload_33_Ohm                 0x01
65.  #define Rload_50_Ohm                 0x02
66.  #define Rload_100_Ohm                0x03
67.
68.  /* ------------------- Reference voltage -------------------- */
69.  #define Internal_Vref                0x00
70.  #define External_Vref                0x80
71.  #define Internal_Zero_20             0x00
72.  #define Internal_Zero_50             0x20
73.  #define Internal_Zero_67             0x40
74.  #define Internal_Zero_Bypassed       0x60
75.
76.  /* --------------------- Bias settings ---------------------- */
77.  #define Bias_Negative                0x00
78.  #define Bias_Positive                0x10
79.  #define Bias_0_Percent               0x00
80.  #define Bias_1_Percent               0x01
81.  #define Bias_2_Percent               0x02
82.  #define Bias_4_Percent               0x03
83.  #define Bias_6_Percent               0x04
84.  #define Bias_8_Percent               0x05
85.  #define Bias_10_Percent              0x06
86.  #define Bias_12_Percent              0x07
87.  #define Bias_14_Percent              0x08
88.  #define Bias_16_Percent              0x09
89.  #define Bias_18_Percent              0x0A
90.  #define Bias_20_Percent              0x0B
91.  #define Bias_22_Percent              0x0C
92.  #define Bias_24_Percent              0x0D
93.
94.  /* -------------------- Function prototypes ----------------- */
95.  void Pot_init(void);
96.  void enable_LMP91000_i2c(unsigned char device_numb);
97.  void initialize_LMP91000(unsigned char UART_buffer);
98.  void LMP91000_temperature_mode(unsigned char device_numb);
99.  void LMP91000_three_lead_amperometric_mode(unsigned char device_numb);
100. void LMP91000_deep_sleep_mode(unsigned char device_numb);
101. int check_LMP_status(unsigned char device_numb);
102. void LMP_all_sleep();
103. void set_LMP_status_array(unsigned char status_arr[]);
104.
105. #endif
```

2.2 ADS1115.h

```
1.  /*
2.  * ZP2015 - ADS1115.h
3.  * Author: Torgrim Rudland Næss
4.  * Based on ADS1113.h from eZ Sense
5.  *
6.  * Changes:
7.  * - Created a more structured setup
8.  * - Added definitions for single-shot power-down mode and input mux settings
9.  * - Removed old function prototypes and added new ones
10. */
11.
12.
13. #ifndef _ADS1115_H
14. #define _ADS1115_H
15.
16. /* ------------------------------ I2C address ------------------------------ */
17. #define AD_I2C_ADDR0                0x49    // ADDR pin to Vcc
18. #define AD_I2C_ADDR1                0x4B    // ADDR pin to SCL
19.
20. /* -------------------------- Register address ------------------------------ */
21. #define AD_CONV_REG                 0x00
22. #define AD_CONF_REG                 0x01
23. #define AD_LO_REG                   0x02
24. #define AD_HI_REG                   0x03
25.
26. /* ----------------------- Config register MSB ------------------------------ */
27.
28. // Single conversion on AIN0-AIN3. FS = +/- 2.048V. Power-down single-shot mode
29. #define ADS1115_MUX_AIN0            0xC5
30. #define ADS1115_MUX_AIN1            0xD5
31. #define ADS1115_MUX_AIN2            0xE5
32. #define ADS1115_MUX_AIN3            0xF5
33.
34. /* ----------------------- Config register LSB ------------------------------ */
35.
36. // Data rate (Samples Per Second)
37. #define ADS1115_CFG_RATE_8          0x00
38. #define ADS1115_CFG_RATE_16         0x20
39. #define ADS1115_CFG_RATE_32         0x40
40. #define ADS1115_CFG_RATE_64         0x60
41. #define ADS1115_CFG_RATE_128        0x80
42. #define ADS1115_CFG_RATE_250        0xA0
43. #define ADS1115_CFG_RATE_475        0xC0
44. #define ADS1115_CFG_RATE_860        0xE0
45. #define ADS1115_DISABLE_COMP        0x03    // Disable comparator
46.
47. /* ----------------------- Function prototypes ------------------------------ */
48. unsigned int ADS_Read(unsigned char i2c_address);
49. void ADS_DoConversion(unsigned char channel_number);
50. int check_ADS_status(unsigned char i2c_address);
51. void take_measurements(char* UART_buffer);
52.
53. #endif
```

2.3 I2C.h

```
1.  ////////////////////////////////////////////////////////////////////////////
2.  //
3.  //  Module Name               : i2c
4.  //  File Name                 : i2c.h
5.  //  Purpose                   : Header file fort i2c bus
6.  //  Author                    : Sindre Søpstad
7.  ////////////////////////////////////////////////////////////////////////////
8.
9.  #ifndef I2C_H_
10. #define I2C_H_
11.
12. #define READ     1
13. #define WRITE    0
14.
15. /* -------- Function prototypes---------*/
16. void i2c_initialize(unsigned char SLV_addr, unsigned char r_w);
17. void i2c_stop();
18. void i2c_transmitByte(unsigned char data);
19. unsigned char i2c_readByte();
20.
21. #endif /*I2C_H_*/
```

2.4 serial.h

```
1.  /*
2.  * ZP2015 - serial.h
3.  * Author: Torgrim Rudland Næss
4.  */
5.  #ifndef SERIAL_H
6.  #define SERIAL_H
7.
8.  #include "macros.h"
9.
10. /* --------------- Function prototypes---------------*/
11. void uart_init();
12. void uart_transmit(char* pBuffer, unsigned char NoBytes);
13. unsigned char* uart_Enable_Receiver(char* pBuffer);
14. void uart_Disable_Receiver();
15. void buffer_clear(char* pBuffer);
16.
17. #endif
```

2.5 timer.h

```
1.  ////////////////////////////////////////////////////////////////////////////
2.  //
3.  //  Module Name               : Timer
4.  //  File Name                 : Timer.h
5.  //  Software Package          : Telespor -  Eartag package
6.  //  Purpose                   : Header file, Providing standard timers
7.  //  Author                    : Tore Syversen
8.  //  Modified by               : Torgrim Rudland Næss
9.  //
10. //  Changes:
11. //  - Removed excess function prototype
12. ////////////////////////////////////////////////////////////////////////////
13.
14. #ifndef _TIMER_H
15. #define _TIMER_H
16.
17. // 12 MHz clock
18. #define TIMERA_ONE_MS       0x2EE0
19. // 8 MHz clock
20. //#define TIMERA_ONE_MS     0x1F40
21.
22. // 1.0 MHz clock
23. //#define TIMERA_ONE_MS     0x0411
24.
25. // 32.768 kHz clock (32.7)
26. //#define TIMERA_ONE_MS     0x21
27.
28. // 100 kHz clock (100)
29. //#define TIMERA_ONE_MS     0x0075
30.
31. // 32768*5/8 ; interrupt every 5 second with 8 divider
32. //#define TIMERA_ONE_SEC    0x5000
33.
34. // 10800*1/8 ; interrupt every second with 8 divider, VLO
35. #define TIMERA_ONE_SEC      0x0546
36.
37. // Function prototypes
38. void wait(unsigned int millisec);
39. void timerInitA(void);
40.
41. #endif
```

2.6 watchdog.h

```
1.  ////////////////////////////////////////////////////////////////////////////
2.  //
3.  //  Module Name               : Watchdog
4.  //  File Name                 : watchdog.h
5.  //  Software Package          : Telespor -  Eartag package
6.  //  Purpose of the file       : Header file for watchdog functions
7.  //  Author                    : Tore Syversen
8.  //  Modified                  : Torgrim Rudland Næss
9.  //
10. //    Changes:
11. //    - Removed one excess code line
12. ////////////////////////////////////////////////////////////////////////////
13.
14. // Function prototypes
15. void watchdogKick(void);
16. void watchdogDisable(void);
17. void watchdogEnable(void);
```

2.7 macros.h

```
1.  /*
2.  * ZP2015 - macros.h
3.  * Author: Torgrim Rudland Næss
4.  * Based on macros.h from eZ Sense
5.  *
6.  * Changes:
7.  * - Removed definitions for LCD display
8.  * - Added comments
9.  */
10.
11. #ifndef _MACROS_H
12. #define _MACROS_H
13.
14. #include <msp430.h>
15.
16. #define bit_set(A,B)          A |= B        // Set bit
17. #define bit_clear(A,B)        A &= ~B       // Clear bit
18.
19. // Set general interrupt enable bit
20. #define SEI                   __bis_SR_register (GIE)
21.
22. // Clear general interrupt enable bit
23. #define CLI                   __bic_SR_register (GIE)
24.
25. #define TIMERA_START          TACTL = 0x214;
26. #define TIMERA_STOP           TACTL = 0x0000;
27.
28. #define LED_ON                bit_clear(P1OUT, BIT7)
29. #define LED_OFF               bit_set(P1OUT, BIT7)
30.
31. #endif
```

3 ZP GUI Source Code

3.1 Form1.cs

```
1.  /*
2.  * ZP GUI Main Form
3.  * Author: Torgrim Rudland Næss
4.  * Based on ZT GUI written by Sindre Søpstad
5.  */
6.
7.  using System;
8.  using System.IO.Ports;
9.  using System.Collections.Generic;
10. using System.ComponentModel;
11. using System.Data;
12. using System.Drawing;
13. using System.Threading;
14. using System.Linq;
15. using System.Text;
16. using System.Threading.Tasks;
17. using System.Windows.Forms;
18. using System.IO;
19.
20.
21. namespace ZP_GUI
22. {
23.     public partial class mainWindow : Form
24.     {
25.
26. ////////////////////// Class variables and Main Form Window Design ////////////////////////
27.
28.         private string logFileName;             // Log file name
29.         private UInt16 windowWidthCollapsed;    // Window width when panel 2 is collapsed
30.         private UInt16 windowWidthExpanded;     // Window width when panel 2 is expanded
31.         private UInt16 windowHeight;            // Window height
32.         private Int16 NoChannels;               // Number of channels
33.         private Int16 nAvg;                     // Number of values for averaging
34.         private Label[] runningAverageLabels;   // label array for running average values
35.         private Label[] unitLabels;             // label array for units
36.         private CheckBox[] channelCheckBoxes;   // Checkbox array for channel selection
37.         public Int16 T1;                        // Minimum temperature
38.         public Int16 T2;                        // Maximum temperature
39.         private bool allChecked;                // true if selectAllBtn is pressed
40.         private bool sensing;                   // True when sensing
41.         public static bool readOK;              // Indicator of successful transmission
42.
43.         // LMP91000 transimpedance feedback resistor value
44.         private int R_TIA;
45.
46.         // Number of bytes to be received from the microcontroller
47.         private static int NoBytes;
48.
49.         // string with gain settings for the LMP91000
50.         private string gainSettings;
51.
52.         // LMP91000 internal zero reference (percentage of source reference)
53.         private double internal_zero;
```

```
54.
55.         // Calibration constants for adjusting glucose consentration offset
56.         public double[] c_val;
57.
58.         // Constants to provide the glucose value proportional to sensor current
59.         public double[] k_val;
60.
61.         // LMP91000 temperature sensor minimum voltage
62.         private Int16 V1;
63.
64.         // LMP91000 temperature sensor maximum voltage
65.         private Int16 V2;
66.
67.         // Array to hold arrays of the last nAvg samples for each channel
68.         private double[][] voltsForAvg;
69.
70.         // Indicates if sensing process has been paused
71.         private bool pause;
72.
73.         // Number of active channels (used for temmperature averaging)
74.         private int activeChannels;
75.
76.
77.         public mainWindow()
78.         {
79.             // Window size
80.             windowWidthCollapsed = 475;
81.             windowWidthExpanded = 1435;
82.             windowHeight = 675;
83.
84.             NoChannels = 8;
85.             NoBytes = 32;
86.             internal_zero = 0.5;
87.             T1 = 15;
88.             T2 = 25;
89.             allChecked = false;
90.             pause = false;
91.             voltsForAvg = new double[NoChannels][];
92.             sensing = false;
93.             c_val = new double[8];
94.             k_val = new double[8];
95.
96.             // Set default k- and c values
97.             for (int i = 0; i < NoChannels; i++)
98.             {
99.                 k_val[i] = 1;
100.                c_val[i] = 0;
101.            }
102.
103.            InitializeComponent();
104.        }
105.
106.
107.        private void mainWindow_Load(object sender, EventArgs e)
108.        {
109.            this.Width = windowWidthCollapsed;
110.            this.Height = windowHeight;
111.            splitContainer.Panel2Collapsed = true;
```

```
112.             buttonExpand.Text = ">>";
113.             mmolRadBtn.Checked = true;
114.             externalGainTxtBox.ReadOnly = true;
115.             yLabel.Visible = false;
116.             logCheckBox.Checked = true;
117.             pauseBtn.Enabled = false;
118.             initGrid();
119.
120.             // Default TIA gain of 120 kOhm (12.9 ÂµA current range)
121.             gainsComboBox.SelectedIndex = 1;
122.
123.             runningAverageLabels = new Label[8];
124.             unitLabels = new Label[8];
125.             channelCheckBoxes = new CheckBox[8];
126.
127.             // Assign labels and checkboxes to arrays
128.             runningAverageLabels[0] = runningAverage_Disp1;
129.             runningAverageLabels[1] = runningAverage_Disp2;
130.             runningAverageLabels[2] = runningAverage_Disp3;
131.             runningAverageLabels[3] = runningAverage_Disp4;
132.             runningAverageLabels[4] = runningAverage_Disp5;
133.             runningAverageLabels[5] = runningAverage_Disp6;
134.             runningAverageLabels[6] = runningAverage_Disp7;
135.             runningAverageLabels[7] = runningAverage_Disp8;
136.             unitLabels[0] = label_unitDisp1;
137.             unitLabels[1] = label_unitDisp2;
138.             unitLabels[2] = label_unitDisp3;
139.             unitLabels[3] = label_unitDisp4;
140.             unitLabels[4] = label_unitDisp5;
141.             unitLabels[5] = label_unitDisp6;
142.             unitLabels[6] = label_unitDisp7;
143.             unitLabels[7] = label_unitDisp8;
144.             channelCheckBoxes[0] = chBox_channel1;
145.             channelCheckBoxes[1] = chBox_channel2;
146.             channelCheckBoxes[2] = chBox_channel3;
147.             channelCheckBoxes[3] = chBox_channel4;
148.             channelCheckBoxes[4] = chBox_channel5;
149.             channelCheckBoxes[5] = chBox_channel6;
150.             channelCheckBoxes[6] = chBox_channel7;
151.             channelCheckBoxes[7] = chBox_channel8;
152.         }
153.
154. ////////////////////////////////////////////////////////////////////////////////////
155.
156.
157. //////////////////////////      Controls and User Interacion    //////////////////////////
158.
159.         /* Expand and collapse panel 2.
160.          * This method is copied from ZT GUI */
161.         private void buttonExpand_Click(object sender, EventArgs e)
162.         {
163.             if (splitContainer.Panel2Collapsed == true)
164.             {
165.                 splitContainer.Panel2Collapsed = false;
166.                 buttonExpand.Text = "<<";
167.                 this.Width = windowWidthExpanded;
168.             }
169.             else
```

```
170.             {
171.                 splitContainer.Panel2Collapsed = true;
172.                 buttonExpand.Text = ">>";
173.                 this.Width = windowWidthCollapsed;
174.             }
175.         }
176.
177.
178.         /* Select/unselect all checkbuttons (channel selection) */
179.         private void selectAllBtn_Click(object sender, EventArgs e)
180.         {
181.             if (allChecked == false)
182.             {
183.                 foreach (var checkbox in channelCheckBoxes)
184.                 {
185.                     // Check all check boxes
186.                     checkbox.Checked = true;
187.                 }
188.
189.                 // allChecked indicates that all channels have been selected
190.                 allChecked = true;
191.
192.                 // Change button text
193.                 selectAllBtn.Text = "Remove all";
194.             }
195.             else
196.             {
197.                 foreach (var checkbox in channelCheckBoxes)
198.                 {
199.                     // Uncheck all check boxes
200.                     checkbox.Checked = false;
201.                 }
202.
203.                 allChecked = false;
204.                 selectAllBtn.Text = "Select all";
205.             }
206.         }
207.
208.
209.         /* Add list of COM ports to the combo box
210.          * Copied from comboBox1_MouseClick1 in ZT GUI */
211.         private void comboBoxChoosePort_MouseClick(object sender, MouseEventArgs e)
212.         {
213.             string[] sPorts = SerialPort.GetPortNames();
214.             comboBoxChoosePort.Items.Clear();
215.             comboBoxChoosePort.Items.AddRange(sPorts);
216.         }
217.
218.
219.         /* Pause the sensing process */
220.         private void pauseBtn_Click(object sender, EventArgs e)
221.         {
222.             // If not paused
223.             if (pause == false)
224.             {
225.                 stop_sense(true);
226.                 pause = true;
227.                 pauseBtn.Text = "Resume";
```

```
228.                 statusTextBox.Text = "Paused";
229.                 pauseBtn.Enabled = true;
230.             }
231.             else
232.             {
233.                 start_sense();
234.                 pause = false;
235.                 pauseBtn.Text = "Pause";
236.             }
237.         }
238.
239.
240.         /* Show form window with calibration- and offset settings */
241.         private void calBtn_Click(object sender, EventArgs e)
242.         {
243.             Calibration calWindow = new Calibration(this);
244.             calWindow.Show();
245.         }
246.
247.
248.         /* Enable or disable user interaction with controls
249.          * Same procedure as in UInteract_Allowed in ZT GUI,
250.          * but with new controls */
251.         private void InteractAllowed(bool interact)
252.         {
253.             intervalControl.Enabled = interact;
254.             timeOutControl.Enabled = interact;
255.             meanElementsControl.Enabled = interact;
256.             gainsComboBox.Enabled = interact;
257.             externalGainTxtBox.Enabled = interact;
258.             advBtn.Enabled = interact;
259.             chartMinValueControl.Enabled = interact;
260.             chartMaxValueControl.Enabled = interact;
261.             axisIntervalControl.Enabled = interact;
262.             comboBoxChoosePort.Enabled = interact;
263.             logCheckBox.Enabled = interact;
264.             selectAllBtn.Enabled = interact;
265.             selectAllBtn.Enabled = interact;
266.             mmolRadBtn.Enabled = interact;
267.             microAmpRadBtn.Enabled = interact;
268.
269.             foreach (var checkbox in channelCheckBoxes)
270.             {
271.                 checkbox.Enabled = interact;
272.             }
273.         }
274.
275.
276.         /* When the X (closing) button is pressed */
277.         protected override void OnFormClosing(FormClosingEventArgs e)
278.         {
279.             base.OnFormClosing(e);
280.
281.             if (e.CloseReason == CloseReason.WindowsShutDown) return;
282.
283.             // Show warning dialog box if sensing is in progress
284.             if (sensing)
285.             {
```

```
286.                 switch (MessageBox.Show(this, "Sensing in progress. "
287.                     + "Are you sure you want to quit?",
288.                       "Closing", MessageBoxButtons.YesNo))
289.                 {
290.                     case DialogResult.No:
291.                         e.Cancel = true;
292.                         break;
293.                     default:
294.                         break;
295.                 }
296.             }
297.         }
298.
299.
300.         /* Change serial port name to one of the items in the COM port combo box
301.          * Copied from comboBox_ChosePort_SelectedIndexChanged in ZT GUI */
302.         private void comboBox_ChosePort_SelectedIndexChanged(object sender, EventArgs e)
303.         {
304.             try
305.             {
306.                 serialPort.PortName = comboBoxChoosePort.Text;
307.             }
308.             catch
309.             {
310.                 statusTextBox.Text = "Error: Could not open serial port";
311.                 serialPort.Close();
312.             }
313.         }
314.
315.
316.         /* Create the string with TIA gain settings for the LMP91000 */
317.         private void setLMPgain(object sender, EventArgs e)
318.         {
319.             switch (gainsComboBox.SelectedIndex)
320.             {
321.                 case 0:
322.                     // Current range: 4.4 ÂµA
323.                     R_TIA = 350000;
324.                     gainSettings = "16";
325.                     externalGainTxtBox.Clear();
326.                     externalGainTxtBox.ReadOnly = true;
327.                     break;
328.                 case 1:
329.                     // Current range: 12.9 ÂµA
330.                     R_TIA = 120000;
331.                     gainSettings = "15";
332.                     externalGainTxtBox.Clear();
333.                     externalGainTxtBox.ReadOnly = true;
334.                     break;
335.                 case 2:
336.                     // Current range: 44 ÂµA
337.                     R_TIA = 35000;
338.                     gainSettings = "14";
339.                     externalGainTxtBox.Clear();
340.                     externalGainTxtBox.ReadOnly = true;
341.                     break;
342.                 case 3:
343.                     // Current range: 110 ÂµA
```

```
344.                     R_TIA = 14000;
345.                     gainSettings = "13";
346.                     externalGainTxtBox.Clear();
347.                     externalGainTxtBox.ReadOnly = true;
348.                     break;
349.                 case 4:
350.                     // Current range: 221 ÂµA
351.                     R_TIA = 7000;
352.                     gainSettings = "12";
353.                     externalGainTxtBox.Clear();
354.                     externalGainTxtBox.ReadOnly = true;
355.                     break;
356.                 case 5:
357.                     // Current range: 442 ÂµA
358.                     R_TIA = 3500;
359.                     gainSettings = "11";
360.                     externalGainTxtBox.Clear();
361.                     externalGainTxtBox.ReadOnly = true;
362.                     break;
363.                 case 6:
364.                     // Current range: 562 ÂµA
365.                     R_TIA = 2750;
366.                     gainSettings = "10";
367.                     externalGainTxtBox.Clear();
368.                     externalGainTxtBox.ReadOnly = true;
369.                     break;
370.                 case 7:
371.                     // External gain
372.                     externalGainTxtBox.ReadOnly = false;
373.                     gainSettings = "17";
374.                     break;
375.             }
376.         }
377.
378. ////////////////////////////////////////////////////////////////////////////////////////
379.
380.
381. ////////////////////////////////////    Start     ////////////////////////////////////
382.
383.         /* Start sensing when start button is clicked */
384.         private void startBtn_Click(object sender, EventArgs e)
385.         {
386.             string warningText = "This will clear all previous measurement data."
387.                 + "\nDo you want to continue?";
388.
389.             // A valid external resistor value must be entered in before starting
390.             if (gainsComboBox.Text == "external")
391.             {
392.                 try
393.                 {
394.                     R_TIA = Convert.ToInt32(externalGainTxtBox.Text);
395.                 }
396.                 catch
397.                 {
398.                     MessageBox.Show("Please enter a valid gain setting");
399.                     return;
400.                 }
401.             }
```

```
402.
403.            // Get number of average samples
404.            nAvg = Convert.ToInt16(meanElementsControl.Text);
405.
406.            // Get lowest and highest temperature in terms of mV
407.            V1 = CelsiusToMilliVolts(T1);
408.            V2 = CelsiusToMilliVolts(T2);
409.
410.            // Arrays for the nAvg last voltage samples of each channel
411.            for (int i = 0; i < NoChannels; i++)
412.            {
413.                voltsForAvg[i] = new double[nAvg];
414.            }
415.
416.            // If not already sensing
417.            if (sensing == false)
418.            {
419.                if (comboBoxChoosePort.Text == "Choose port")
420.                {
421.                    // If no port has been chosen by the user
422.                    MessageBox.Show("Please choose a COM port");
423.                }
424.                else
425.                {
426.                    // Initialize log file if log file check box is checked
427.                    if (logCheckBox.Checked)
428.                    {
429.                        LogInit();
430.                    }
431.
432.                    // Prevent user for interacting with controls while sensing
433.                    InteractAllowed(false);
434.                    sensing = true;
435.                    startBtn.Text = "Stop sensing";
436.
437.                    if (AD_grid.Rows.Count != 1)
438.                    {
439.                        // yes/no dialog box if grid contains previous data values
440.                        DialogResult confirm =
441.                            MessageBox.Show(warningText,
442.                            "WARNING", MessageBoxButtons.YesNo);
443.
444.                        // Clear all data and start sensing if 'yes'
445.                        if (confirm == DialogResult.Yes)
446.                        {
447.                            // Clear grid and chart
448.                            AD_grid.Rows.Clear();
449.                            tempADCGrid.Rows.Clear();
450.                            currentGrid.Rows.Clear();
451.                            glucoseGrid.Rows.Clear();
452.                            temperatureGrid.Rows.Clear();
453.
454.                            foreach (var series in chart1.Series)
455.                            {
456.                                series.Points.Clear();
457.                            }
458.
459.                            // Start sensing
```

```
460.                             start_sense();
461.                         }
462.                         else if (confirm == DialogResult.No)
463.                         {
464.                             sensing = false;
465.                             startBtn.Text = "Start sensing";
466.                             stop_sense(true);
467.                             InteractAllowed(true);
468.                         }
469.                     }
470.                     else
471.                     {
472.                         // grid has no values. Sensing can start without loss of old data
473.                         start_sense();
474.                     }
475.                 }
476.             }
477.             else
478.             {   // Stop sensing if sensing variable is true
479.                 sensing = false;
480.                 startBtn.Text = "Start sensing";
481.                 stop_sense(true);
482.                 InteractAllowed(true);
483.
484.                 if (pause == true)
485.                 {
486.                     pause = false;
487.                     pauseBtn.Text = "Pause";
488.                 }
489.
490.                 // Update status text box with location of the saved log file
491.                 if (logCheckBox.Checked)
492.                 {
493.                     statusTextBox.Text = "Log file saved to " + logFileName;
494.                 }
495.             }
496.         }
497.
498.
499.         /* Start sensing
500.          * The method is based on start_sense from ZT GUI.
501.          * The following procedures are the same:
502.          * - opening of serial ports
503.          * - Using timers
504.          * - Set timer intervals */
505.         private void start_sense()
506.         {
507.             InitDisplay();  // Set up displays for running average
508.             InitChart();    // Set up the line chart
509.             pauseBtn.Enabled = true; // Enable pause button
510.             statusTextBox.Text = "Sensing...";
511.
512.             // Find the number of active channels to calculate correct average temperature
513.             activeChannels = 0;
514.
515.             foreach (var checkbox in channelCheckBoxes)
516.             {
517.                 if (checkbox.Checked)
```

```
518.                 {
519.                     activeChannels += 1;
520.                 }
521.             }
522.
523.             // Reset pause button
524.             if (pause == true)
525.             {
526.                 pause = false;
527.                 pauseBtn.Text = "Pause";
528.             }
529.
530.             try
531.             {
532.                 // Open serial port if not already open.
533.                 if (!serialPort.IsOpen)
534.                     serialPort.Open();
535.
536.                 // Send TIA gain settings to the microcontroller
537.                 serialPort.Write(gainSettings);
538.
539.                 // Get sampling interval from user input
540.                 timerData.Interval = Convert.ToInt16(intervalControl.Text) * 1000;
541.                 timerTimeout.Interval = Convert.ToInt16(timeOutControl.Text) * 1000;
542.
543.                 // Enable timers
544.                 timerData.Start();
545.                 timerTimeout.Start();
546.             }
547.             catch
548.             {
549.                 statusTextBox.Text = "Error: Could not open serial port";
550.             }
551.         }
552.
553. ////////////////////////////////////////////////////////////////////////////////////
554.
555.
556. ////////////////////////////    Stop and Timeouts     ////////////////////////////////
557.
558.
559.         /* Stop and reset when there is no response from the microcontroller */
560.         private void timeout(object sender, EventArgs e)
561.         {
562.             stop_sense(true);
563.             statusTextBox.Text = "Timed out...";
564.             sensing = false;
565.             startBtn.Text = "Start sensing";
566.         }
567.
568.
569.         /* Stop sensing
570.          * Based on stop_sense in ZT GUI.
571.          * Timer control and user interaction control is the same
572.          */
573.         private void stop_sense(bool WRITELOG_FLAG)
574.         {
575.             if (serialPort.IsOpen)
```

```
576.             {
577.                 serialPort.Write("3");
578.                 serialPort.Close();
579.             }
580.
581.             // Stop timers
582.             timerData.Stop();
583.             timerTimeout.Stop();
584.
585.             statusTextBox.Text = "Stopped";
586.
587.             // Allow user interaction
588.             InteractAllowed(true);
589.
590.             // Disable pause button
591.             pauseBtn.Enabled = false;
592.         }
593.
594. ////////////////////////////////////////////////////////////////////////////////
595.
596.
597.
598. ////////////////////////////    Send and Receive Data       ////////////////////////////
599.
600.         /* Send a data request to the microcontroller
601.          * Copied from ZT GUI.
602.          * Changes:
603.          * - Only data request is sent; not read instruction
604.          * - removed code related to the listbox
605.          * - Changed error message
606.          */
607.         private void ReadADC(object sender, EventArgs e)
608.         {
609.             // A 2 indicates request for data
610.             string serialCommand = "2";
611.
612.             foreach (var checkbox in channelCheckBoxes)
613.             {
614.                 if (checkbox.Checked)
615.                 {
616.                     // Request data for this channel
617.                     serialCommand += "1";
618.                 }
619.                 else
620.                 {
621.                     // Microcontroller will return 0 for this channel
622.                     serialCommand += "0";
623.                 }
624.             }
625.
626.             try
627.             {
628.                 // Send data request to microcontroller
629.                 serialPort.Write(serialCommand);
630.             }
631.             catch
632.             {
633.                 statusTextBox.Text = "Error: Could not write to serial port.";
```

```
634.                 timerTimeout.Stop();
635.             }
636.         }
637.
638.
639.         /* Runs when 32 bytes has been received on the serial port
640.          * Copied from ZT GUI */
641.         private void serialPort_DataReceived(object sender,
642.             System.IO.Ports.SerialDataReceivedEventArgs e)
643.         {
644.             // BeginInvoke instead of Invoke to allow port closing
645.             this.BeginInvoke(new EventHandler(RS232DataRecieved));
646.             Thread.Sleep(800); // Begininvoke needs this
647.         }
648.
649.
650.         /* Fetch data from the serial port's input buffer and return a boolean
651.          * value to indicate if the number of bytes read was correct */
652.         bool ReadBuffer(byte[] ADCbuffer, Int16[] AD_values)
653.         {
654.             int bytesRead = 0;
655.
656.             try
657.             {
658.                 if (serialPort.BytesToRead <= NoBytes)
659.                 {
660.                     // Fill ADCbuffer with data from the serial port
661.                     bytesRead = serialPort.Read(ADCbuffer, 0, NoBytes);
662.                     if (bytesRead != 32)
663.                     {
664.                         // Bytes missing
665.                         readOK = false;
666.                     }
667.                     else
668.                     {
669.                         // No bytes missing
670.                         readOK = true;
671.                     }
672.                 }
673.             }
674.             catch (TimeoutException) { }
675.
676.             // Clear buffer
677.             serialPort.DiscardInBuffer();
678.
679.             for (int i = 0; i < 16; i++)
680.             {
681.                 // Ch.1 gets values from ADCbuffer[0-1], Ch.2 from ADCbuffer[2-3], etc.
682.                 AD_values[i] = BitConverter.ToInt16(ADCbuffer, 2 * i);
683.             }
684.
685.             return readOK;
686.         }
687.
688. ////////////////////////////////////////////////////////////////////////////////
689.
690.
691. ////////////////////////////     Signal Processing       ////////////////////////////////
```

```
692.
693.        /* Read data from serial port and perform calculations
694.         * Based on RS232DataRecieved in ZT GUI.
695.         */
696.        public void RS232DataRecieved(object s, EventArgs e)
697.        {
698.            Int16[] AD_values = new Int16[16];              // AD values
699.            double[] volts = new double[8];                 // Voltage values
700.            Int16[] temperatureADC = new Int16[8];          // Temperature ADC
701.            double[] avgVolts = new double[NoChannels];     // average voltage values
702.            string[] AD_Rows = new string[9];               // Rows for AD value grid
703.            double[] microCurrents = new double[8];         // Current values
704.            double[] avgCurrent = new double[NoChannels];   // Average current values
705.            double[] temperature = new double[8];           // Temperature values
706.            double avgTemperature;                          // Average temperature
707.            string[] temperatureRows = new string[9];       // Rows for temperatures
708.            string temperatureString;                       // Temperature string
709.            double[] milliGlucose = new double[8];          // Glucose values
710.            string[] glucoseRows = new string[9];           // Rows for glucose value grid
711.
712.            // Rows for temperature ADC grid
713.            string[] temperatureADCRows = new string[9];
714.
715.            // Rows for microcurrent value grid
716.            string[] microCurrentRows = new string[9];
717.
718.            // Sum for average temperature
719.            double temperatureSum = 0;
720.
721.            // Average mmol/L values
722.            double[] avgMilliGLucose = new double[NoChannels];
723.
724.            // string of data for the log file
725.            string logData;
726.
727.            // Get current time
728.            string time = DateTime.Now.ToString("HH:mm:ss");
729.
730.            // Buffer array for ADC values
731.            byte[] ADCbuffer = new byte[NoBytes];
732.
733.            // Stop timeout timer
734.            timerTimeout.Stop();
735.
736.            // Stop timeout timer
737.            timerTimeout.Start();
738.
739.            serialPort.ReadTimeout = 100;
740.
741.            // Read bytes from input buffer into AD_values
742.            readOK = ReadBuffer(ADCbuffer, AD_values);
743.
744.            // Add an X to the timestamp if less than 32 bytes were received
745.            if (readOK == false)
746.            {
747.                time = time + " X";
748.            }
749.
```

```
750.            /* --- Calculation of temperature, voltages, currents and glucose values--- */
751.
752.            for (int i = 0; i < 8; i++)
753.            {
754.                // Calculate voltages, currents, glucose and temperatures
755.                volts[i] = (2.048 * AD_values[i] / 32767) - internal_zero;
756.                //milliVolts[i] = volts[i] * 1e3;
757.                microCurrents[i] = (volts[i] / R_TIA) * 1e6;
758.                milliGlucose[i] = k_val[i] * microCurrents[i] + c_val[i];
759.                temperatureADC[i] = AD_values[i + 8];
760.
761.                // Calculate temperature on all selected channels
762.                if (channelCheckBoxes[i].Checked)
763.                {
764.                    temperature[i] = CalculateTemperature(temperatureADC[i]);
765.                }
766.                else
767.                {
768.                    temperature[i] = 0;
769.                }
770.
771.                // Create sum of temperatures
772.                temperatureSum += temperature[i];
773.            }
774.
775.            // Display average temperature if input buffer was read successfully
776.            if(readOK)
777.            {
778.                // Calculate the average temperature of all active channels
779.                avgTemperature = temperatureSum / activeChannels;
780.
781.                // Rounded up to closest integer value
782.                temperatureString = Convert.ToString(Math.Round(avgTemperature, 2));
783.
784.                // Display temperature in text box
785.                temperatureBox.Text = " Temperature: " + temperatureString + " \u00b0C";
786.            }
787.
788.
789.            /* -------------- Add values to grid, chart and log file ------------------ */
790.
791.            // Add timestamp to log file and grid headers
792.            logData = time + ";";
793.            AD_Rows[0] = time;
794.            temperatureADCRows[0] = time;
795.            microCurrentRows[0] = time;
796.            glucoseRows[0] = time;
797.            temperatureRows[0] = time;
798.
799.            for (int i = 0; i < 8; i++)
800.            {
801.                if (channelCheckBoxes[i].Checked)
802.                {
803.                    // Create string arrays for the grid view
804.                    AD_Rows[i + 1] = Convert.ToString(AD_values[i]);
805.                    temperatureADCRows[i + 1] = Convert.ToString(AD_values[i + 8]);
806.                    microCurrentRows[i + 1] =
807.                        string.Format("{0:0.000}", Math.Round(microCurrents[i], 3));
```

```
808.                     glucoseRows[i + 1] =
809.                         string.Format("{0:0.000}", Math.Round(milliGlucose[i], 3));
810.                     temperatureRows[i + 1] =
811.                         string.Format("{0:0.000}", Math.Round(temperature[i], 3));
812.
813.                     // Add glucose values to chart if 32 bytes were received
814.                     if (readOK)
815.                     {
816.                         chart1.Series[i].Points.AddXY(time, milliGlucose[i]);
817.                     }
818.
819.                     // Add data values to the log file string
820.                     logData += AD_Rows[i + 1].ToString() + ";";
821.                     logData += temperatureADCRows[i + 1].ToString() + ";";
822.                     logData += microCurrentRows[i + 1].ToString() + ";";
823.                     logData += glucoseRows[i + 1].ToString() + ";";
824.                     logData += temperatureRows[i + 1].ToString() + ";";
825.                 }
826.                 else
827.                 {
828.                     // Channels not selected. Add not available instead
829.                     AD_Rows[i + 1] = "N/A";
830.                     microCurrentRows[i + 1] = "N/A";
831.                     glucoseRows[i + 1] = "N/A";
832.                     temperatureRows[i + 1] = "N/A";
833.                     temperatureADCRows[i + 1] = "N/A";
834.                 }
835.             }
836.
837.             // Add string arrays to the grid views
838.             AD_grid.Rows.Add(AD_Rows);
839.             tempADCGrid.Rows.Add(temperatureADCRows);
840.             currentGrid.Rows.Add(microCurrentRows);
841.             glucoseGrid.Rows.Add(glucoseRows);
842.             temperatureGrid.Rows.Add(temperatureRows);
843.
844.             // Scroll down the grid views if necessary
845.             AutoscrollGrids();
846.
847.             // add newline to end the log data string
848.             logData += Environment.NewLine;
849.
850.             // Add string to the log file if the log file check box is selected
851.             if (logCheckBox.Checked)
852.             {
853.                 File.AppendAllText(logFileName, logData);
854.             }
855.
856.
857.             /* ---------------------------- Averaging ---------------------------- */
858.
859.             // Calculate running average values if input buffer was read successfully
860.             if(readOK)
861.             {
862.                 for (int i = 0; i < NoChannels; i++)
863.                 {
864.                     // Average volts
865.                     avgVolts[i] = FindAverage(voltsForAvg[i], volts[i]);
```

```
866.
867.                     // Average microcurrent
868.                     avgCurrent[i] = (FindAverage(voltsForAvg[i], volts[i]) / R_TIA) * 1e6;
869.
870.                     // Average glucose
871.                     avgMilliGLucose[i] = k_val[i] * (FindAverage(voltsForAvg[i],
872.                         volts[i]) / R_TIA) * 1e6 + c_val[i];
873.                 }
874.
875.                 // Display running average values
876.                 if (mmolRadBtn.Checked)
877.                 {
878.                     DisplayRunningAverage(milliGlucose, avgMilliGLucose);
879.                 }
880.                 else
881.                 {
882.                     DisplayRunningAverage(microCurrents, avgCurrent);
883.                 } // End if
884.             }
885.         }
886.
887.
888.         /* Return average values
889.          * Based on the averaging process in RS232DataRecieved in ZT GUI */
890.         private double FindAverage(double[] averageValues, double newValue)
891.         {
892.             double average = 0;
893.
894.             // Shift Register
895.             for (int i = 0; i < nAvg - 1; i++)
896.             {
897.                 averageValues[i] = averageValues[i + 1];
898.             }
899.
900.             // Add new value
901.             averageValues[nAvg - 1] = newValue;
902.
903.             // Return the average value of the last nAvg samples
904.             for (int i = 0; i < nAvg; i++)
905.             {
906.                 average = (average + averageValues[i]);
907.             }
908.             return average / nAvg;
909.         }
910.
911.
912.         /* Return a temperature value in terms of milliVolts
913.          * (From table in LMP91000 datasheet, p. 13-14) */
914.         private Int16 CelsiusToMilliVolts(Int16 temperature)
915.         {
916.             switch (temperature)
917.             {
918.                 case 0: return 1560; break;
919.                 case 1: return 1552; break;
920.                 case 2: return 1544; break;
921.                 case 3: return 1536; break;
922.                 case 4: return 1528; break;
923.                 case 5: return 1520; break;
```

```
924.                 case 6: return 1512; break;
925.                 case 7: return 1504; break;
926.                 case 8: return 1496; break;
927.                 case 9: return 1488; break;
928.                 case 10: return 1480; break;
929.                 case 11: return 1472; break;
930.                 case 12: return 1464; break;
931.                 case 13: return 1456; break;
932.                 case 14: return 1448; break;
933.                 case 15: return 1440; break;
934.                 case 16: return 1432; break;
935.                 case 17: return 1424; break;
936.                 case 18: return 1415; break;
937.                 case 19: return 1407; break;
938.                 case 20: return 1399; break;
939.                 case 21: return 1391; break;
940.                 case 22: return 1383; break;
941.                 case 23: return 1375; break;
942.                 case 24: return 1367; break;
943.                 case 25: return 1359; break;
944.                 case 26: return 1351; break;
945.                 case 27: return 1342; break;
946.                 case 28: return 1334; break;
947.                 case 29: return 1326; break;
948.                 case 30: return 1318; break;
949.                 case 31: return 1310; break;
950.                 case 32: return 1302; break;
951.                 case 33: return 1293; break;
952.                 case 34: return 1285; break;
953.                 case 35: return 1277; break;
954.                 case 36: return 1269; break;
955.                 case 37: return 1261; break;
956.                 case 38: return 1253; break;
957.                 case 39: return 1244; break;
958.                 case 40: return 1236; break;
959.                 case 41: return 1228; break;
960.                 case 42: return 1220; break;
961.                 case 43: return 1212; break;
962.                 case 44: return 1203; break;
963.                 case 45: return 1195; break;
964.                 case 46: return 1187; break;
965.                 case 47: return 1179; break;
966.                 case 48: return 1170; break;
967.                 case 49: return 1162; break;
968.                 case 50: return 1154; break;
969.                 default: return 1;
970.             }
971.         }
972.
973.
974.         /* Calculate and return temperatures (Eq. from the LMP91000 datasheet, p. 14) */
975.         private double CalculateTemperature(Int16 ADC)
976.         {
977.             double V = (2.048 * ADC / 32767) * 1e3;             // milliVolts
978.             double T = (V - V1) * (T2 - T1) / (V2 - V1) + T1;   // Temperature
979.             return T;
980.         }
981.
```

```
982. ////////////////////////////////////////////////////////////////////////////////////
983.
984.
985. ////////////////////////////     Data Presentation       ////////////////////////////
986.
987.         /* Initialize running average displays */
988.         private void InitDisplay()
989.         {
990.             foreach (var label in runningAverageLabels)
991.             {
992.                 label.Text = "00.000";
993.             }
994.
995.             if (mmolRadBtn.Checked)
996.             {
997.                 foreach (var label in unitLabels)
998.                 {
999.                     label.Text = "mM";
1000.                }
1001.            }
1002.            else
1003.            {
1004.                foreach (var label in unitLabels)
1005.                {
1006.                    label.Text = "ÂµA";
1007.                }
1008.            }
1009.        }
1010.
1011.
1012.        /* Initialize the gridview */
1013.        private void initGrid()
1014.        {
1015.            // Add nine columns to grids
1016.            AD_grid.ColumnCount = 9;
1017.            currentGrid.ColumnCount = 9;
1018.            glucoseGrid.ColumnCount = 9;
1019.            temperatureGrid.ColumnCount = 9;
1020.            tempADCGrid.ColumnCount = 9;
1021.
1022.            // Leftmost columns gets "Time" header text
1023.            AD_grid.Columns[0].HeaderText = "Time";
1024.            currentGrid.Columns[0].HeaderText = "Time";
1025.            glucoseGrid.Columns[0].HeaderText = "Time";
1026.            temperatureGrid.Columns[0].HeaderText = "Time";
1027.            tempADCGrid.Columns[0].HeaderText = "Time";
1028.
1029.            // Add Channel number and data type to remaining column headers
1030.            for (int i = 1; i <= 8; i++)
1031.            {
1032.                string chaNumb = Convert.ToString(i);
1033.                AD_grid.Columns[i].HeaderText = "Channel " + chaNumb + "\nsensor ADC";
1034.                tempADCGrid.Columns[i].HeaderText = "Channel " + chaNumb + "\ntemp. ADC";
1035.                currentGrid.Columns[i].HeaderText = "Channel " + chaNumb + "\nÂµA";
1036.                glucoseGrid.Columns[i].HeaderText = "Channel " + chaNumb + "\nmmol/L";
1037.                temperatureGrid.Columns[i].HeaderText = "Channel "
1038.                    + chaNumb + "\n\u00B0C";
1039.            }
```

```
1040.         }
1041.
1042.
1043.         /* Initialize the chart */
1044.         private void InitChart()
1045.         {
1046.             // Choose line colors for the chart
1047.             chart1.Series[0].Color = Color.Brown;
1048.             chart1.Series[1].Color = Color.Magenta;
1049.             chart1.Series[2].Color = Color.Green;
1050.             chart1.Series[3].Color = Color.Orange;
1051.             chart1.Series[4].Color = Color.Red;
1052.             chart1.Series[5].Color = Color.Purple;
1053.             chart1.Series[6].Color = Color.Blue;
1054.             chart1.Series[7].Color = Color.Black;
1055.
1056.             // Set Y-axis minimum value
1057.             chart1.ChartAreas[0].AxisY.Minimum =
1058.                 Convert.ToDouble(chartMinValueControl.Text);
1059.
1060.             // Set Y-axis maximum value
1061.             chart1.ChartAreas[0].AxisY.Maximum =
1062.                 Convert.ToDouble(chartMaxValueControl.Text);
1063.
1064.             // Set Y-axis intervals
1065.             chart1.ChartAreas[0].AxisY.Interval =
1066.                 Convert.ToDouble(axisIntervalControl.Text);
1067.
1068.             chart1.ChartAreas[0].AxisX.IsMarginVisible = false;
1069.
1070.             yLabel.Visible = true; // Show Y-axis label
1071.
1072.             // Add grid lines
1073.             chart1.ChartAreas[0].AxisY.MajorGrid.Enabled = true;
1074.             chart1.ChartAreas[0].AxisY.MajorGrid.LineColor = Color.DarkGray;
1075.             chart1.ChartAreas[0].AxisY.MajorGrid.LineDashStyle =
1076.                 System.Windows.Forms.DataVisualization.Charting.ChartDashStyle.Dot;
1077.         }
1078.
1079.
1080.         /* Initialize log file */
1081.         private void LogInit()
1082.         {
1083.             string path;
1084.             string startTime;
1085.             string logFileHeader;
1086.
1087.             // Set log file path to the folder ZP GUI in My Documents
1088.             path = System.Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
1089.                 + "\\ZP GUI\\logs\\";
1090.
1091.             // Create log file folder if it does not already exist
1092.             if (!Directory.Exists(path))
1093.             {
1094.                 Directory.CreateDirectory(path);
1095.             }
1096.
1097.             // Get start time for log file name
```

```
1098.             startTime = DateTime.Now.ToString("yyyy.MM.dd_HH.mm.ss");
1099.
1100.             // Can change file type here: .csv or .txt
1101.             logFileName = @path + "Log_" + startTime + ".csv";
1102.             logFileHeader = "Time;";
1103.
1104.             // Add channel numbers and data types to column headers for selected channels
1105.             for (int i = 0; i < 8; i++)
1106.             {
1107.                 if (channelCheckBoxes[i].Checked)
1108.                 {
1109.                     logFileHeader += "Ch." + Convert.ToString(i + 1)
1110.                         + " (Sensor ADC);";
1111.
1112.                     logFileHeader += "Ch." + Convert.ToString(i + 1)
1113.                         + " (Temperature ADC);";
1114.
1115.                     logFileHeader += "Ch." + Convert.ToString(i + 1)
1116.                         + " (microAmps);";
1117.
1118.                     logFileHeader += "Ch." + Convert.ToString(i + 1)
1119.                         + " (mmol/L);";
1120.
1121.                     logFileHeader += "Ch." + Convert.ToString(i + 1)
1122.                         + " (Celsius);";
1123.                 }
1124.             }
1125.
1126.             // Add columns headers to log file
1127.             logFileHeader += Environment.NewLine;
1128.             File.WriteAllText(logFileName, logFileHeader);
1129.         }
1130.
1131.
1132.         /* Change the unit labels */
1133.         private void ChangeUnitLabels(object sender, EventArgs e)
1134.         {
1135.             if (mmolRadBtn.Checked)
1136.             {
1137.                 foreach (var label in unitLabels)
1138.                 {
1139.                     label.Text = "mM";
1140.                 }
1141.             }
1142.             else
1143.             {
1144.                 foreach (var label in unitLabels)
1145.                 {
1146.                     label.Text = "ÂµA";
1147.                 }
1148.             }
1149.         }
1150.
1151.
1152.         /* Scroll down the grid */
1153.         private void AutoscrollGrids()
1154.         {
1155.             try
```

```
1156.             {
1157.                 AD_grid.FirstDisplayedScrollingRowIndex =
1158.                     AD_grid.RowCount - 1;
1159.                 tempADCGrid.FirstDisplayedScrollingRowIndex =
1160.                     tempADCGrid.RowCount - 1;
1161.                 currentGrid.FirstDisplayedScrollingRowIndex =
1162.                     currentGrid.RowCount - 1;
1163.                 temperatureGrid.FirstDisplayedScrollingRowIndex =
1164.                     temperatureGrid.RowCount - 1;
1165.                 glucoseGrid.FirstDisplayedScrollingRowIndex =
1166.                     temperatureGrid.RowCount - 1;
1167.             }
1168.             catch { }
1169.         }
1170.
1171.
1172.         /* Show running average values on the displays
1173.          * Based on the process in RS232DataRecieved in ZT GUI */
1174.         private void DisplayRunningAverage(double[] value, double[] averageValue)
1175.         {
1176.             for (int i = 0; i < NoChannels; i++)
1177.             {
1178.                 // If channel is selected
1179.                 if (channelCheckBoxes[i].Checked)
1180.                 {
1181.                     if (currentGrid.RowCount - 1 < nAvg)
1182.                     {
1183.                         /* Add last sample to the display if number of samples is
1184.                          * less than the number of samples to be used for averaging */
1185.                         runningAverageLabels[i].Text =
1186.                             string.Format("{0:0.000}", Math.Round(value[i], 3));
1187.                     }
1188.                     else
1189.                     {
1190.                         // Add average value if number of samples > nAvg
1191.                         runningAverageLabels[i].Text =
1192.                             string.Format("{0:0.000}", Math.Round(averageValue[i], 3));
1193.                     }
1194.                 }
1195.                 else
1196.                 {
1197.                     // If channel is not selected
1198.                     runningAverageLabels[i].Text = "N/A";
1199.                     unitLabels[i].Text = "";
1200.                 }
1201.             }
1202.         }
1203.     }
1204. }
```

3.2 Calibration.cs

```
1.  /*
2.  * Form to set Calibration and offset values for each channel
3.  * Author: Torgrim Rudland Næss */
4.
5.  using System;
6.  using System.Collections.Generic;
7.  using System.ComponentModel;
8.  using System.Data;
9.  using System.Drawing;
10. using System.Linq;
11. using System.Text;
12. using System.Threading.Tasks;
13. using System.Windows.Forms;
14.
15. namespace ZP_GUI
16. {
17.     public partial class Calibration : Form
18.     {
19.         private mainWindow mainForm = null;
20.
21.
22.         bool save;
23.
24.         public Calibration()
25.         {
26.             InitializeComponent();
27.         }
28.
29.
30.         public Calibration(Form callingForm)
31.         {
32.             mainForm = callingForm as mainWindow;
33.             InitializeComponent();
34.             setBoxValues();
35.         }
36.
37.
38.         /* Update k_val, c_val and temperature range values in main form */
39.         private void closeBtn_Click(object sender, EventArgs e)
40.         {
41.             this.mainForm.k_val[0] = Convert.ToDouble(kValueControl1.Text);
42.             this.mainForm.k_val[1] = Convert.ToDouble(kValueControl2.Text);
43.             this.mainForm.k_val[2] = Convert.ToDouble(kValueControl3.Text);
44.             this.mainForm.k_val[3] = Convert.ToDouble(kValueControl4.Text);
45.             this.mainForm.k_val[4] = Convert.ToDouble(kValueControl5.Text);
46.             this.mainForm.k_val[5] = Convert.ToDouble(kValueControl6.Text);
47.             this.mainForm.k_val[6] = Convert.ToDouble(kValueControl7.Text);
48.             this.mainForm.k_val[7] = Convert.ToDouble(kValueControl8.Text);
49.             this.mainForm.c_val[0] = Convert.ToDouble(cValueControl1.Text);
50.             this.mainForm.c_val[1] = Convert.ToDouble(cValueControl2.Text);
51.             this.mainForm.c_val[2] = Convert.ToDouble(cValueControl3.Text);
52.             this.mainForm.c_val[3] = Convert.ToDouble(cValueControl4.Text);
53.             this.mainForm.c_val[4] = Convert.ToDouble(cValueControl5.Text);
54.             this.mainForm.c_val[5] = Convert.ToDouble(cValueControl6.Text);
55.             this.mainForm.c_val[6] = Convert.ToDouble(cValueControl7.Text);
56.             this.mainForm.c_val[7] = Convert.ToDouble(cValueControl8.Text);
```

```
57.                 this.mainForm.T1 = Convert.ToInt16(minTempBox.Text);
58.                 this.mainForm.T2 = Convert.ToInt16(maxTempBox.Text);
59.
60.                 save = true;
61.                 this.Close();
62.             }
63.
64.             /* Fetch k_val, c_val and temperature range values from main form */
65.             private void setBoxValues()
66.             {
67.                 kValueControl1.Text = Convert.ToString(this.mainForm.k_val[0]);
68.                 kValueControl2.Text = Convert.ToString(this.mainForm.k_val[1]);
69.                 kValueControl3.Text = Convert.ToString(this.mainForm.k_val[2]);
70.                 kValueControl4.Text = Convert.ToString(this.mainForm.k_val[3]);
71.                 kValueControl5.Text = Convert.ToString(this.mainForm.k_val[4]);
72.                 kValueControl6.Text = Convert.ToString(this.mainForm.k_val[5]);
73.                 kValueControl7.Text = Convert.ToString(this.mainForm.k_val[6]);
74.                 kValueControl8.Text = Convert.ToString(this.mainForm.k_val[7]);
75.
76.                 cValueControl1.Text = Convert.ToString(this.mainForm.c_val[0]);
77.                 cValueControl2.Text = Convert.ToString(this.mainForm.c_val[1]);
78.                 cValueControl3.Text = Convert.ToString(this.mainForm.c_val[2]);
79.                 cValueControl4.Text = Convert.ToString(this.mainForm.c_val[3]);
80.                 cValueControl5.Text = Convert.ToString(this.mainForm.c_val[4]);
81.                 cValueControl6.Text = Convert.ToString(this.mainForm.c_val[5]);
82.                 cValueControl7.Text = Convert.ToString(this.mainForm.c_val[6]);
83.                 cValueControl8.Text = Convert.ToString(this.mainForm.c_val[7]);
84.
85.                 minTempBox.Text = Convert.ToString(this.mainForm.T1);
86.                 maxTempBox.Text = Convert.ToString(this.mainForm.T2);
87.             }
88.
89.
90.             /* When the X (closing) button is pressed */
91.             protected override void OnFormClosing(FormClosingEventArgs e)
92.             {
93.                 base.OnFormClosing(e);
94.
95.                 if (e.CloseReason == CloseReason.WindowsShutDown) return;
96.
97.                 // Save is false when trying to close with the X button
98.                 if (save == false)
99.                 {
100.                    switch (MessageBox.Show(this,
101.                        "Are you sure you want to close the settings window? "
102.                        + "This will discard any changes made.",
103.                        "Closing", MessageBoxButtons.YesNo))
104.                    {
105.                        case DialogResult.No:
106.                            e.Cancel = true;
107.                            break;
108.                        default:
109.                            break;
110.                    }
111.                }
112.            }
113.        }
114.    }
```