Lab 5 Report EEL4742C - 00446

Yousef Awad

September 2025

Contents

Introduction	2
5.1 Printing on the LCD Display	2
5.2 Implementing a Counter	4
5.3 Utility Chronometer	7
Student Q&A	13
1	13
2	13
3	13

Introduction

5.1 Printing on the LCD Display

```
#include <msp430fr6989.h>
2 #include <stdint.h>
4 #define redLED BITO // Red at P1.0
5 #define greenLED BIT7 // Green at P9.7
_{7} // The array has the shapes of the digits (0 to 9)
8 const unsigned char LCD_Shapes[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67
       , 0xB7, 0xBF, 0xE0, 0xFF, 0xE7 };
11 // Initializes the LCD_C module
void Initialize_LCD()
13 {
    PJSELO = BIT4 | BIT5; // For LFXT
14
    LCDCPCTLO = 0xFFD0;
15
    LCDCPCTL1 = 0xF83F;
    LCDCPCTL2 = 0x00F8;
17
    // Configure LFXT 32kHz crystal
    CSCTLO_H = CSKEY >> 8; // Unlock CS registers
19
    CSCTL4 &= ~LFXTOFF; // Enable LFXT
20
21
    do
22
      CSCTL5 &= ~LFXTOFFG; // Clear LFXT fault flag
23
      SFRIFG1 &= ~OFIFG;
24
25
    while (SFRIFG1 & OFIFG); // Test oscillator fault flag
26
    CSCTLO_H = 0; // Lock CS registers
27
    // Initialize LCD_C
    // ACLK, Divider = 1, Pre-divider = 16; 4-pin MUX LCDCCTLO = LCDDIV_1 | LCDPRE_16 | LCD4MUX | LCDLP;
29
    // VLCD generated internally,
31
    // V2-V4 generated internally, v5 to ground
32
    // Set VLCD voltage to 2.60v
33
    \ensuremath{//} Enable charge pump and select internal reference for it
34
    LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;
    LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled
36
    LCDCMEMCTL = LCDCLRM; // Clear LCD memory
37
    //Turn LCD on (do this at the end!)
38
    LCDCCTLO |= LCDON;
39
40
    return;
41 }
43 void lcd_write_uint16 (uint16_t number)
44 {
    uint8_t digits[5]; // we have a max of 5 possible digits
45
    // Addresses for each possible digit.
46
    volatile unsigned char *display[5] = { &LCDM6, &LCDM4, &LCDM19, &
      LCDM15, &LCDM8 };
49
    uint8_t i = 0;
for (i = 0; i < 5; i++)
```

```
51
52
       // Finding all digits in the number inputted
       digits[4 - i] = number % 10;
53
54
       number /= 10;
55
56
     int leading = 0;
for (i = 0; i < 5; i++)</pre>
57
58
59
       // Incrementing through all 5 digits
60
       if (digits[i])
61
62
         // A digit exists at this display section
63
64
         leading = 1;
65
       if ((i == 4) && (leading == 0))
66
67
         // We don't have any value at this display section
68
69
         *display[i] = LCD_Shapes[0];
70
71
       else
72
73
         // The digit we are on was not the 5th one OR it was the 5th
74
         // one and was not a leading digit
75
         if (leading)
76
         {
           // We have a leading digit
77
            *display[i] = LCD_Shapes[digits[i]];
78
79
         else
80
81
           // We do not have a leading digit
82
            *display[i] = 0;
83
84
       }
85
     }
86
87
     // Exiting function
     return;
89
90 }
91
92 int main(void)
93 {
     volatile unsigned int n;
94
     WDTCTL = WDTPW | WDTHOLD; // Stop WDT
95
     PM5CTLO &= ~LOCKLPM5; // Enable GPIO pins
96
     P1DIR |= redLED; // Pins as output
97
     P9DIR |= greenLED;
98
     P10UT |= redLED; // Red on
P90UT &= ~greenLED; // Green off
99
100
     // Initializes the LCD_C module
     Initialize_LCD();
102
103
     uint16_t number = 51487;
104
     lcd_write_uint16(number);
105
106
// Flash the red LED
```

```
for(;;)
{
    for(n=0; n<=60000; n++)
    {
        // Delay loop
    }
    P1OUT ^= redLED;
}
</pre>
```

5.2 Implementing a Counter

When comparing the time on the counter for 5.2 that I've made it almost perfectly matches it, with the error most likely being due to human error and my own response time.

```
#include <msp430fr6989.h>
#include <stdint.h>
#define redLED BITO // Red at P1.0
5 #define greenLED BIT7 // Green at P9.7
7 #define but1 BIT1 // Port 1.1
8 #define but2 BIT2 // Port 1.2
_{10} // The array has the shapes of the digits (0 to 9)
11 const unsigned char LCD_Shapes[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67
      , 0xB7, 0xBF, 0xE0, 0xFF, 0xE7 };
uint16_t counter = 0;
             *****************
15 // Initializes the LCD_C module
void Initialize_LCD()
17 {
    PJSELO = BIT4 | BIT5; // For LFXT
18
19
    LCDCPCTLO = 0xFFD0;
    LCDCPCTL1 = 0xF83F;
LCDCPCTL2 = 0x00F8;
20
    // Configure LFXT 32kHz crystal
22
    CSCTLO_H = CSKEY >> 8; // Unlock CS registers
23
    CSCTL4 &= ~LFXTOFF; // Enable LFXT
24
25
26
      CSCTL5 &= "LFXTOFFG; // Clear LFXT fault flag
27
      SFRIFG1 &= ~OFIFG;
28
29
    while (SFRIFG1 & OFIFG); // Test oscillator fault flag
30
    CSCTLO_H = 0; // Lock CS registers
31
    // Initialize LCD_C
32
    // ACLK, Divider = 1, Pre-divider = 16; 4-pin MUX
33
34
    LCDCCTL0 = LCDDIV__1 | LCDPRE__16 | LCD4MUX | LCDLP;
    // VLCD generated internally,
35
36
    // V2-V4 generated internally, v5 to ground
    // Set VLCD voltage to 2.60v
37
    // Enable charge pump and select internal reference for it
    LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;
LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled
39
```

```
LCDCMEMCTL = LCDCLRM; // Clear LCD memory
41
42
    //Turn LCD on (do this at the end!)
    LCDCCTLO |= LCDON;
43
44
45 }
46
47 void lcd_write_uint16 (uint16_t number)
48 {
49
    uint8_t digits[5]; // we have a max of 5 possible digits
     // Addresses for each possible digit.
50
     volatile unsigned char *display[5] = { &LCDM6, &LCDM4, &LCDM19, &
51
      LCDM15, &LCDM8 };
52
    uint8_t i = 0;
53
    for (i = 0; i < 5; i++)</pre>
54
55
       // Finding all digits in the number inputted
56
      digits[4 - i] = number % 10;
57
      number /= 10;
58
59
60
    int leading = 0;
61
     for (i = 0; i < 5; i++)
62
63
      // Incrementing through all 5 digits
64
65
       if (digits[i])
66
         // A digit exists at this display section
67
        leading = 1;
68
69
      if ((i == 4) && (leading == 0))
70
71
         // We don't have any value at this display section
72
        *display[i] = LCD_Shapes[0];
73
74
75
       else
76
77
        // The digit we are on was not the 5th one OR it was the 5th
        // one and was not a leading digit
78
79
        if (leading)
80
81
          // We have a leading digit
           *display[i] = LCD_Shapes[digits[i]];
82
83
         else
84
85
           // We do not have a leading digit
86
           *display[i] = 0;
87
88
89
      }
90
91
92
    // Exiting function
93
    return;
94 }
96 // Configures ACLK to 32 KHz crystal
```

```
97 void config_ACLK_to_32KHz_crystal()
98 {
      // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
99
100
     // Reroute pins to LFXIN/LFXOUT functionality
PJSEL1 &= ~BIT4;
102
     PJSELO |= BIT4;
103
104
105
     // Wait until the oscillator fault flags remain cleared
     CSCTLO = CSKEY; // Unlock CS registers
106
107
108
        CSCTL5 &= ~LFXTOFFG; // Local fault flag
109
        SFRIFG1 &= "OFIFG; // Global fault flag
110
111
     while((CSCTL5 & LFXTOFFG) != 0);
112
113
      CSCTLO_H = 0; // Lock CS registers
114
115
      return;
116 }
117
118 int main(void)
119 {
120
     volatile unsigned int n;
     WDTCTL = WDTPW | WDTHOLD; // Stop WDT
PM5CTLO &= ~LOCKLPM5; // Enable GPIO pins
121
122
     P1DIR |= redLED; // Pins as output
123
     P9DIR |= greenLED;
124
     P10UT |= redLED; // Red on
125
     P90UT &= ~greenLED; // Green off
126
127
     P1DIR &= ~(but1 | but2);
128
     P1REN |= (but1 | but2);
129
     P10UT |= (but1 | but2);
130
     P1IES |= (but1 | but2);
131
132
     P1IFG &= (but1 | but2);
     P1IE |= (but1 | but2);
133
134
     P1IFG = 0; // resetting flag for interrupt
135
136
      // Initializes the LCD_C module
     Initialize_LCD();
137
138
      config_ACLK_to_32KHz_crystal();
139
140
      // Starting timer
141
     TAOCCRO = 32767;
142
     TAOCCTLO = CCIE;
143
     TAOCCTLO &= ~CCIFG;
144
     TAOCTL = TASSEL_1 | ID_0 | MC_1 | TAIE;
145
     TAOCTL &= ~TAIFG;
146
147
      _low_power_mode_3();
148
149
     return 0;
150
151 }
#pragma vector = TIMERO_AO_VECTOR
```

```
__interrupt void TAO_ISR()
155 {
     lcd_write_uint16(counter);
156
     counter += 1;
     TAOCTL &= ~TAIFG;
158
159 }
160
#pragma vector = PORT1_VECTOR
   __interrupt void PORT1_ISR()
163 €
     if ((P1IFG & but1))
164
165
       counter = 0;
166
       P1IFG &= "but1; // clear flag
167
168
169
170
     if ((P1IFG & but2))
171
172
       counter += 1000;
       P1IFG &= "but2; // clear flag
173
174
175 }
```

5.3 Utility Chronometer

Just to get this out of the way, the maximum design of my chronometer is that of 12 hours like an AM/PM clock. This is due to the fact that I put the max value that the counter, in seconds, will go to, to be 43199. Now, onto the design. I basically redid the entirety of the 5.2 and 5.1, mainly due to the fact that I started with 5.3 and went backward in writing the code. I specifically chose to make the buttons work via polling once they are already activated, mainly due to the fact that it's just a lot easier for me, and energy consumption is not of a concern. Alongside this, I used a couple of while loops so that the speed-up function of the buttons can be easily looped indefinetely until they are released. That's really it, that's different about the versions from 5.2 and 5.1, when compared to 5.3.

```
uint16_t holding = 1;
19
20 const unsigned char LCD_Shapes[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67
     , 0xB7, 0xBF, 0xE0, 0xFF, 0xE7 };
uint16_t digits[6] = {0};
int checkingBut2And1 = 0;
25 // Configures ACLK to 32 KHz crystal
void config_ACLK_to_32KHz_crystal()
27 {
    // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
28
    // Reroute pins to LFXIN/LFXOUT functionality
PJSEL1 &= ~BIT4;
30
31
    PJSELO |= BIT4;
32
33
    // Wait until the oscillator fault flags remain cleared
34
    CSCTLO = CSKEY; // Unlock CS registers
35
36
    do
37
      CSCTL5 &= ~LFXTOFFG; // Local fault flag
SFRIFG1 &= ~OFIFG; // Global fault flag
38
39
40
    while((CSCTL5 & LFXTOFFG) != 0);
41
42
    CSCTLO_H = 0; // Lock CS registers
43
44
    return;
45 }
47 void toggle_colon()
49 // All this function does is toggle the colon
    LCDM7 ^= BIT2;
50
51 }
52
void toggle_exclamation()
54 {
55
    LCDM3 |= BITO;
   LCDM3 &= ~BIT3;
56
57 }
59 void toggle_stopwatch()
60 {
61 LCDM3 |= BIT3;
   LCDM3 &= ~BITO;
62
63 }
64
65 // Initializes the LCD_C module
66 void Initialize_LCD()
67 {
PJSELO = BIT4 | BIT5; // For LFXT
69 LCDCPCTLO = 0xFFDO;
    LCDCPCTL1 = 0xF83F;
LCDCPCTL2 = 0 \times 00F8;
// Configure LFXT 32kHz crystal
```

```
CSCTLO_H = CSKEY >> 8;
73
74
     CSCTL4 &= ~LFXTOFF;
75
     do
76
       // Unlock CS registers
77
        // Enable LFXT
78
        CSCTL5 &= ~LFXTOFFG;
79
       SFRIFG1 &= ~OFIFG;
80
81
82
     while (SFRIFG1 & OFIFG);
     CSCTLO_H = 0; // Clear LFXT fault flag
83
     // Test oscillator fault flag
84
     // Lock CS registers
85
     // Initialize LCD_C
     // ACLK, Divider = 1, Pre-divider = 16; 4-pin MUX LCDCCTLO = LCDDIV_1 | LCDPRE_16 | LCD4MUX | LCDLP;
87
88
     // VLCD generated internally,
89
     // V2-V4 generated internally, v5 to ground
90
91
     // Set VLCD voltage to 2.60v
     \ensuremath{//} Enable charge pump and select internal reference for it
92
     LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;
93
     LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled
94
     LCDCMEMCTL = LCDCLRM; // Turn LCD on (do this at the end!)
95
96
     LCDCCTLO |= LCDON;
     // Clear LCD memory
97
98
     return;
99 }
100
_{101} // This function does not initialize the timer, but instead is used
         for starting and stopping it.
_{
m 102} // This function returns the time that was inputted if successfull,
       and returns -1 if unsuccessful
int startTime(uint16_t time_given)
104 {
105
     if (time_given < 0)</pre>
106
       // Returning false.
107
108
       return -1;
109
110
     // time_given is a valid value
111
     TAOCCRO = time_given;
112
113
     TAOCCTLO = CCIE;
     TAOCCTLO &= ~CCIFG;
114
     TAOCTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
115
     TAOCTL &= ~TAIFG; // Clearing interrupt flag (if any)
116
117
118
     // Returning success
     return time_given;
119
120 }
121
122 // This function initializes the timer to work.
123 void initializeTime()
124 {
     TAOCTL = TASSEL_1 | ID_0 | MC_2 | TACLR | TAIE;
125
     TAOCTL &= "TAIFG; // Clearing any previous flags (if any)
126
```

```
128
129 void stopTime()
130 {
     // Clearing the timer.
131
     TAOCTL = MC_O | TACLR;
132
133 }
134
135 void LCDframe()
136 {
     // Enable the decimal point
137
     LCDM20 = BIT0;
138
139
     if (state)
140
141
       // we are playing time
142
143
       toggle_stopwatch();
       TAOCTL |= MC_1;
144
        if (totalSec > 43199)
145
146
       {
         totalSec = 0;
147
148
     }
149
     else
150
151
       toggle_exclamation();
152
       TAOCTL &= ~MC_3;
153
154
     toggle_colon();
155
156
     seconds = totalSec % 60;
157
     minutes = (totalSec % 3600) / 60;
158
     hours = totalSec / 3600;
159
160
     uint8_t i = 0; // temporary counter
161
162
     uint8_t *display[6] = {&LCDM10, &LCDM6, &LCDM4, &LCDM19, &LCDM15,
163
        &LCDM8};
     uint16_t temp;
165
166
167
     temp = seconds;
     // Writing seconds
168
     for (i = 0; i < 2; i++)</pre>
169
170
171
       digits[i] = temp % 10;
       temp /= 10; // go down
172
        *display[5 - i] = LCD_Shapes[digits[i]];
173
174
175
     // Writing minutes
176
     temp = minutes;
177
     for (i = 0; i < 2; i++)
178
179
       digits[i] = temp % 10;
180
181
       temp /= 10;
       *display[3 - i] = LCD_Shapes[digits[i]];
182
183
```

```
184
185
      // Writing hours
      temp = hours;
186
      for (i = 0; i < 2; i++)</pre>
187
188
        digits[i] = temp % 10;
189
190
        temp /= 10;
        *display[1 - i] = LCD_Shapes[digits[i]];
191
192
193 }
194
195 int main(void)
196 {
      WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
197
     PM5CTLO &= ~LOCKLPM5;
                                  // Enable the GPIO pins
198
199
200
     P1DIR |= red; // Set output for red led
      P10UT &= "red; // Turn off red LED
201
202
     P9DIR |= green; // Set output for green led P9OUT &= ~green; // Turn off green LED
203
204
205
      // Using default initialize function
206
207
      Initialize_LCD();
     LCDCMEMCTL = LCDCLRM; // Clear LCD
208
209
      // Configure buttons
210
     P1DIR &= ~(but1 | but2); // input
211
     P1REN |= (but1 | but2); // enable resistors
P1OUT |= (but1 | but2); // pull-up
212
213
214
      P1IES |= (but1 | but2); // interrupt on falling edge
215
      P1IE |= (but1 | but2);
                                  // enable interrupts
216
     P1IFG = 0;
                                   // set interrupt flag off
217
218
219
      config_ACLK_to_32KHz_crystal();
220
221
      // initializeTime(); // To set the interrupt flag forever.
      startTime(32767); // Clock max is 1 second per.
222
223
      _low_power_mode_0();
224
225
226
      return 0;
227 }
228
229 #pragma vector = PORT1_VECTOR
   __interrupt void PORT1_ISR()
230
231 {
     if ((P1IFG & but1))
232
233
        holding = 1;
234
        // S1 was pressed
235
236
        while (!(P1IN & but1))
237
238
          // S1 was held.
239
240
          holding += 1;
```

```
_delay_cycles(10000);
241
242
          if (holding > 110)
          {
243
244
            break;
         }
245
246
247
        if (holding > 100)
248
249
         // We held it long enough.
250
          state = false;
251
          totalSec = 0;
252
         // Stop the state.
253
254
       else
255
256
       {
257
         state = !state;
258
259
       P1IFG &= ~but1; // clear flag
260
261
        LCDframe();
          __delay_cycles(50000); // 50k cycles for debouncing
262
263
264
     if ((P1IFG & but2))
265
266
        checkingBut2And1 = 1;
267
       while (!(P1IN & but2))
268
269
270
          LCDframe();
271
          // You are pushing down only button2
         if (!(P1IN & but1))
272
273
           totalSec--;
274
         }
275
276
          else
277
278
            totalSec++;
279
280
          __delay_cycles(100); // debounce
281
282
       P1IFG &= ~but2;
     }
283
284 }
285
286 #pragma vector = TIMERO_AO_VECTOR
__interrupt void TOAO_ISR()
     \ensuremath{//} generate a new frame every time the timer overflows.
289
290
     totalSec += 1;
     LCDframe();
291
292 }
293
294 #pragma vector = TIMERO_A1_VECTOR
295 __interrupt void TOA1_ISR()
296 {
297 // Do nothing
```

Student Q&A

1

Given: Explain whether this statement is true or false. If false, explain the correct operation. "An LCD segment works just like a colored LED. It's turned on/off by writing either digital high/low to it, respectively".

The statement is true. It works the exact same way as an LED, its just that you have more than one LED to turn on and off at a given frequency (as it's not always on, and instead flickers extremely fast so as to conserve energy).

2

Given: What is the name of the LCD controller that interfaces the LCD display of our board? Is the LCD controller located on the display module or in the microcontroller?

The LCD controller is specifically located inside the MCU of the MSP430, specifically it is called 'LCD_C'.

3

Given: In what multiplexing configuration is the LCD module wired (2-way, 4-way, etc)? What does this mean regarding the number of pins used at the microcontroller?

The LCD is wired in 4-way multiplexing so as to conserve the amount of pins that the microcontroller needs by a factor of 4. Since we have a 108 segment display, this therefore means we only need 27 pins to write the high/low values to the entirety of said display.