

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

```
1 #include <msp430fr6989.h>
2 #include <stdint.h>
3
4 #define redLED BIT0 // Red at P1.0
5 #define greenLED BIT7 // Green at P9.7
6
7 // The array has the shapes of the digits (0 to 9)
8 const unsigned char LCD_Shapes[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67
9     , 0xB7, 0xBF, 0xE0, 0xFF, 0xE7 };
10
11 //*****
12 // Initializes the LCD_C module
13 void Initialize_LCD()
14 {
15     PJSEL0 = BIT4 | BIT5; // For LFXT
16     LCDCPCTL0 = 0xFFD0;
17     LCDCPCTL1 = 0xF83F;
18     LCDCPCTL2 = 0x00F8;
19     // Configure LFXT 32kHz crystal
20     CSCTL0_H = CSKEY >> 8; // Unlock CS registers
21     CSCTL4 &= ~LFXTOFF; // Enable LFXT
22     do
23     {
24         CSCTL5 &= ~LFXTOFFG; // Clear LFXT fault flag
25         SFRIFG1 &= ~OFIFG;
26     }
27     while (SFRIFG1 & OFIFG); // Test oscillator fault flag
28     CSCTL0_H = 0; // Lock CS registers
29     // Initialize LCD_C
30     // ACLK, Divider = 1, Pre-divider = 16; 4-pin MUX
31     LCDCCCTL0 = LCDDIV__1 | LCDPRE__16 | LCD4MUX | LCDLP;
32     // VLCD generated internally,
33     // V2-V4 generated internally, v5 to ground
34     // Set VLCD voltage to 2.60v
35     // Enable charge pump and select internal reference for it
36     LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;
37     LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled
38     LCDCMEMCTL = LCDCLRM; // Clear LCD memory
39     //Turn LCD on (do this at the end!)
40     LCDCCCTL0 |= LCDON;
41     return;
42 }
43
44 void lcd_write_uint16 (uint16_t number)
45 {
46     uint8_t digits[5]; // we have a max of 5 possible digits
47     // Addresses for each possible digit.
48     volatile unsigned char *display[5] = { &LCDM6, &LCDM4, &LCDM19, &
49         LCDM15, &LCDM8 };
50
51     uint8_t i = 0;
52     for (i = 0; i < 5; i++)
```

```

51 {
52     // Finding all digits in the number inputted
53     digits[4 - i] = number % 10;
54     number /= 10;
55 }
56
57 int leading = 0;
58 for (i = 0; i < 5; i++)
59 {
60     // Incrementing through all 5 digits
61     if (digits[i])
62     {
63         // A digit exists at this display section
64         leading = 1;
65     }
66     if ((i == 4) && (leading == 0))
67     {
68         // We don't have any value at this display section
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         {
77             // We have a leading digit
78             *display[i] = LCD_Shapes[digits[i]];
79         }
80         else
81         {
82             // We do not have a leading digit
83             *display[i] = 0;
84         }
85     }
86 }
87
88 // Exiting function
89 return;
90 }
91
92 int main(void)
93 {
94     volatile unsigned int n;
95     WDTCTL = WDTPW | WDTHOLD; // Stop WDT
96     PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
97     P1DIR |= redLED; // Pins as output
98     P9DIR |= greenLED;
99     P1OUT |= redLED; // Red on
100    P9OUT &= ~greenLED; // Green off
101    // Initializes the LCD_C module
102    Initialize_LCD();
103
104    uint16_t number = 51487;
105    lcd_write_uint16(number);
106
107    // Flash the red LED

```

```

108     for(;;)
109     {
110         for(n=0; n<=60000; n++)
111         {
112             // Delay loop
113         }
114         P10UT ^= redLED;
115     }
116 }

```

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.

```

1  #include <msp430fr6989.h>
2  #include <stdint.h>
3
4  #define redLED BIT0 // Red at P1.0
5  #define greenLED BIT7 // Green at P9.7
6
7  #define but1 BIT1 // Port 1.1
8  #define but2 BIT2 // Port 1.2
9
10 // The array has the shapes of the digits (0 to 9)
11 const unsigned char LCD_Shapes[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67
12     , 0xB7, 0xBF, 0xE0, 0xFF, 0xE7 };
13 uint16_t counter = 0;
14 //*****
15 // Initializes the LCD_C module
16 void Initialize_LCD()
17 {
18     PJSEL0 = BIT4 | BIT5; // For LFXT
19     LCDCPCTL0 = 0xFFD0;
20     LCDCPCTL1 = 0xF83F;
21     LCDCPCTL2 = 0x00F8;
22     // Configure LFXT 32kHz crystal
23     CSCTL0_H = CSKEY >> 8; // Unlock CS registers
24     CSCTL4 &= ~LFXTOFF; // Enable LFXT
25     do
26     {
27         CSCTL5 &= ~LFXTOFFG; // Clear LFXT fault flag
28         SFRIFG1 &= ~OFIFG;
29     }
30     while (SFRIFG1 & OFIFG); // Test oscillator fault flag
31     CSCTL0_H = 0; // Lock CS registers
32     // Initialize LCD_C
33     // ACLK, Divider = 1, Pre-divider = 16; 4-pin MUX
34     LCDCCTL0 = LCDDIV__1 | LCDPRE__16 | LCD4MUX | LCDLP;
35     // VLCD generated internally,
36     // V2-V4 generated internally, v5 to ground
37     // Set VLCD voltage to 2.60v
38     // Enable charge pump and select internal reference for it
39     LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;
40     LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled

```

```

41 LCDCMEMCTL = LCDCLRM; // Clear LCD memory
42 //Turn LCD on (do this at the end!)
43 LCDCCTL0 |= LCDON;
44 return;
45 }
46
47 void lcd_write_uint16 (uint16_t number)
48 {
49     uint8_t digits[5]; // we have a max of 5 possible digits
50     // Addresses for each possible digit.
51     volatile unsigned char *display[5] = { &LCDM6, &LCDM4, &LCDM19, &
        LCDM15, &LCDM8 };
52
53     uint8_t i = 0;
54     for (i = 0; i < 5; i++)
55     {
56         // Finding all digits in the number inputted
57         digits[4 - i] = number % 10;
58         number /= 10;
59     }
60
61     int leading = 0;
62     for (i = 0; i < 5; i++)
63     {
64         // Incrementing through all 5 digits
65         if (digits[i])
66         {
67             // A digit exists at this display section
68             leading = 1;
69         }
70         if ((i == 4) && (leading == 0))
71         {
72             // We don't have any value at this display section
73             *display[i] = LCD_Shapes[0];
74         }
75         else
76         {
77             // The digit we are on was not the 5th one OR it was the 5th
78             // one and was not a leading digit
79             if (leading)
80             {
81                 // We have a leading digit
82                 *display[i] = LCD_Shapes[digits[i]];
83             }
84             else
85             {
86                 // We do not have a leading digit
87                 *display[i] = 0;
88             }
89         }
90     }
91
92     // Exiting function
93     return;
94 }
95
96 // Configures ACLK to 32 KHz crystal

```

```

97 void config_ACLK_to_32KHz_crystal()
98 {
99     // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
100
101     // Reroute pins to LFXIN/LFXOUT functionality
102     PJSEL1 &= ~BIT4;
103     PJSEL0 |= BIT4;
104
105     // Wait until the oscillator fault flags remain cleared
106     CSCTL0 = CSKEY; // Unlock CS registers
107     do
108     {
109         CSCTL5 &= ~LFXTOFFG; // Local fault flag
110         SFRIFG1 &= ~OFIFG; // Global fault flag
111     }
112     while((CSCTL5 & LFXTOFFG) != 0);
113
114     CSCTL0_H = 0; // Lock CS registers
115     return;
116 }
117
118 int main(void)
119 {
120     volatile unsigned int n;
121     WDCTL = WDTWPW | WDTOLD; // Stop WDT
122     PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
123     P1DIR |= redLED; // Pins as output
124     P9DIR |= greenLED;
125     P1OUT |= redLED; // Red on
126     P9OUT &= ~greenLED; // Green off
127
128     P1DIR &= ~(but1 | but2);
129     P1REN |= (but1 | but2);
130     P1OUT |= (but1 | but2);
131     P1IES |= (but1 | but2);
132     P1IFG &= (but1 | but2);
133     P1IE |= (but1 | but2);
134     P1IFG = 0; // resetting flag for interrupt
135
136     // Initializes the LCD_C module
137     Initialize_LCD();
138
139     config_ACLK_to_32KHz_crystal();
140
141     // Starting timer
142     TAOCCRO = 32767;
143     TAOCTL0 = CCIE;
144     TAOCTL0 &= ~CCIFG;
145     TAOCTL = TASSEL_1 | ID_0 | MC_1 | TAIE;
146     TAOCTL &= ~TAIFG;
147
148     _low_power_mode_3();
149
150     return 0;
151 }
152
153 #pragma vector = TIMER0_A0_VECTOR

```

```

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

```

1 #include <msp430fr6989.h>
2 #include <stdbool.h>
3 #include <stdint.h>
4
5 #define red BIT0 // Port 1.0
6 #define green BIT7 // Port 9.7
7
8 #define but1 BIT1 // Port 1.1
9 #define but2 BIT2 // Port 1.2
10
11 bool state = true; // true is unpaused
12 volatile uint16_t totalSec = 0;
13 volatile uint16_t seconds = 0;
14 volatile uint16_t minutes = 0;
15 volatile uint16_t hours = 0;
16 volatile uint16_t counter = 0;

```

```

17 uint16_t holding = 1;
18
19
20 const unsigned char LCD_Shapes[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67
    , 0xB7, 0xBF, 0xE0, 0xFF, 0xE7 };
21 uint16_t digits[6] = {0};
22
23 int checkingBut2And1 = 0;
24
25 // Configures ACLK to 32 KHz crystal
26 void config_ACLK_to_32KHz_crystal()
27 {
28     // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
29
30     // Reroute pins to LFXIN/LFXOUT functionality
31     PJSEL1 &= ~BIT4;
32     PJSEL0 |= BIT4;
33
34     // Wait until the oscillator fault flags remain cleared
35     CSCTL0 = CSKEY; // Unlock CS registers
36     do
37     {
38         CSCTL5 &= ~LFXTOFFG; // Local fault flag
39         SFRIFG1 &= ~OFIFG; // Global fault flag
40     }
41     while((CSCTL5 & LFXTOFFG) != 0);
42
43     CSCTL0_H = 0; // Lock CS registers
44     return;
45 }
46
47 void toggle_colon()
48 {
49     // All this function does is toggle the colon
50     LCDM7 ^= BIT2;
51 }
52
53 void toggle_exclamation()
54 {
55     LCDM3 |= BIT0;
56     LCDM3 &= ~BIT3;
57 }
58
59 void toggle_stopwatch()
60 {
61     LCDM3 |= BIT3;
62     LCDM3 &= ~BIT0;
63 }
64
65 // Initializes the LCD_C module
66 void Initialize_LCD()
67 {
68     PJSEL0 = BIT4 | BIT5; // For LFXT
69     LCDCPCTL0 = 0xFFD0;
70     LCDCPCTL1 = 0xF83F;
71     LCDCPCTL2 = 0x00F8;
72     // Configure LFXT 32kHz crystal

```



```

73 CSCTL0_H = CSKEY >> 8;
74 CSCTL4 &= ~LFXTOFF;
75 do
76 {
77     // Unlock CS registers
78     // Enable LFXT
79     CSCTL5 &= ~LFXTOFFG;
80     SFRIFG1 &= ~OFIFG;
81 }
82 while (SFRIFG1 & OFIFG);
83 CSCTL0_H = 0; // Clear LFXT fault flag
84 // Test oscillator fault flag
85 // Lock CS registers
86 // Initialize LCD_C
87 // ACLK, Divider = 1, Pre-divider = 16; 4-pin MUX
88 LCDCTL0 = LCDDIV_1 | LCDPRE__16 | LCD4MUX | LCDLP;
89 // VLCD generated internally,
90 // V2-V4 generated internally, v5 to ground
91 // Set VLCD voltage to 2.60v
92 // Enable charge pump and select internal reference for it
93 LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;
94 LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled
95 LCDMEMCTL = LCDCLRM; // Turn LCD on (do this at the end!)
96 LCDCTL0 |= LCDON;
97 // Clear LCD memory
98 return;
99 }
100
101 // This function does not initialize the timer, but instead is used
102 // for starting and stopping it.
103 // This function returns the time that was inputted if succesfull,
104 // and returns -1 if unsuccessful
105 int startTime(uint16_t time_given)
106 {
107     if (time_given < 0)
108     {
109         // Returning false.
110         return -1;
111     }
112
113     // time_given is a valid value
114     TAOCCRO = time_given;
115     TAOCTL0 = CCIE;
116     TAOCTL0 &= ~CCIFG;
117     TAOCTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
118     TAOCTL &= ~TAIFG; // Clearing interrupt flag (if any)
119
120     // Returning success
121     return time_given;
122 }
123
124 // This function initializes the timer to work.
125 void initializeTime()
126 {
127     TAOCTL = TASSEL_1 | ID_0 | MC_2 | TACLR | TAIE;
128     TAOCTL &= ~TAIFG; // Clearing any previous flags (if any)
129 }

```

```

128
129 void stopTime()
130 {
131     // Clearing the timer.
132     TAOCTL = MC_0 | TACLR;
133 }
134
135 void LCDframe()
136 {
137     // Enable the decimal point
138     LCDM20 = BIT0;
139
140     if (state)
141     {
142         // we are playing time
143         toggle_stopwatch();
144         TAOCTL |= MC_1;
145         if (totalSec > 43199)
146         {
147             totalSec = 0;
148         }
149     }
150     else
151     {
152         toggle_exclamation();
153         TAOCTL &= ~MC_3;
154     }
155     toggle_colon();
156
157     seconds = totalSec % 60;
158     minutes = (totalSec % 3600) / 60;
159     hours = totalSec / 3600;
160
161     uint8_t i = 0; // temporary counter
162
163     uint8_t *display[6] = {&LCDM10, &LCDM6, &LCDM4, &LCDM19, &LCDM15,
164                             &LCDM8};
165
166     uint16_t temp;
167
168     temp = seconds;
169     // Writing seconds
170     for (i = 0; i < 2; i++)
171     {
172         digits[i] = temp % 10;
173         temp /= 10; // go down
174         *display[5 - i] = LCD_Shapes[digits[i]];
175     }
176
177     // Writing minutes
178     temp = minutes;
179     for (i = 0; i < 2; i++)
180     {
181         digits[i] = temp % 10;
182         temp /= 10;
183         *display[3 - i] = LCD_Shapes[digits[i]];
184     }

```

```

184
185 // Writing hours
186 temp = hours;
187 for (i = 0; i < 2; i++)
188 {
189     digits[i] = temp % 10;
190     temp /= 10;
191     *display[1 - i] = LCD_Shapes[digits[i]];
192 }
193 }
194
195 int main(void)
196 {
197     WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
198     PM5CTL0 &= ~LOCKLPM5;     // Enable the GPIO pins
199
200     P1DIR |= red; // Set output for red led
201     P1OUT &= ~red; // Turn off red LED
202
203     P9DIR |= green; // Set output for green led
204     P9OUT &= ~green; // Turn off green LED
205
206     // Using default initialize function
207     Initialize_LCD();
208     LCDMEMCTL = LCDCLRM; // Clear LCD
209
210     // Configure buttons
211     P1DIR &= ~(but1 | but2); // input
212     P1REN |= (but1 | but2); // enable resistors
213     P1OUT |= (but1 | but2); // pull-up
214
215     P1IES |= (but1 | but2); // interrupt on falling edge
216     P1IE |= (but1 | but2); // enable interrupts
217     P1IFG = 0; // set interrupt flag off
218
219     config_ACLK_to_32KHz_crystal();
220
221     // initializeTime(); // To set the interrupt flag forever.
222     startTime(32767); // Clock max is 1 second per.
223
224     _low_power_mode_0();
225
226     return 0;
227 }
228
229 #pragma vector = PORT1_VECTOR
230 __interrupt void PORT1_ISR()
231 {
232     if ((P1IFG & but1))
233     {
234         holding = 1;
235         // S1 was pressed
236
237         while (!(P1IN & but1))
238         {
239             // S1 was held.
240             holding += 1;

```

```

241     _delay_cycles(10000);
242     if (holding > 110)
243     {
244         break;
245     }
246 }
247
248 if (holding > 100)
249 {
250     // We held it long enough.
251     state = false;
252     totalSec = 0;
253     // Stop the state.
254 }
255 else
256 {
257     state = !state;
258 }
259
260 P1IFG &= ~but1; // clear flag
261 LCDframe();
262     __delay_cycles(50000); // 50k cycles for debouncing
263 }
264
265 if ((P1IFG & but2))
266 {
267     checkingBut2And1 = 1;
268     while (!(P1IN & but2))
269     {
270         LCDframe();
271         // You are pushing down only button2
272         if (!(P1IN & but1))
273         {
274             totalSec--;
275         }
276         else
277         {
278             totalSec++;
279         }
280         __delay_cycles(100); // debounce
281     }
282     P1IFG &= ~but2;
283 }
284 }
285
286 #pragma vector = TIMERO_A0_VECTOR
287 __interrupt void TOA0_ISR()
288 {
289     // generate a new frame every time the timer overflows.
290     totalSec += 1;
291     LCDframe();
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.