

## Lab 5

# LCD Display

---

In this lab, we will learn how to use the segmented LCD display.

### 5.1 Printing on the LCD Display

Our LaunchPad board is equipped with a segmented LCD display, which is the ADKOM model FH-1138P that has 108 segments and its layout is shown in Figure 5.1. Unlike simple colored LEDs, LCD segments can't be controlled by simply writing low/high to turn them off/on, respectively. Liquid crystal burns out if a continuous voltage is applied to it for a long time. Therefore, the LCD display is interfaced via a specialized controller that continually oscillates the signals that are applied to the segments. The controller can be either part of the display module or part of the MCU. In the case of our board, the LCD controller is a module on the MSP430 called the LCD\_C module. It is described in the FR6xx Family User's Guide ([slau367o](#)) in Chapter 36.

When the LCD controller is part of the MCU, this introduces the problem of having too many wires between the LCD display and the microcontroller. Our display has 108 segments. Using a basic configuration called static drive, we would need 108 pins at the MCU to control the segments, which is an excessive number of pins. Therefore, the idea of multiplexing has been devised in which one pin of the

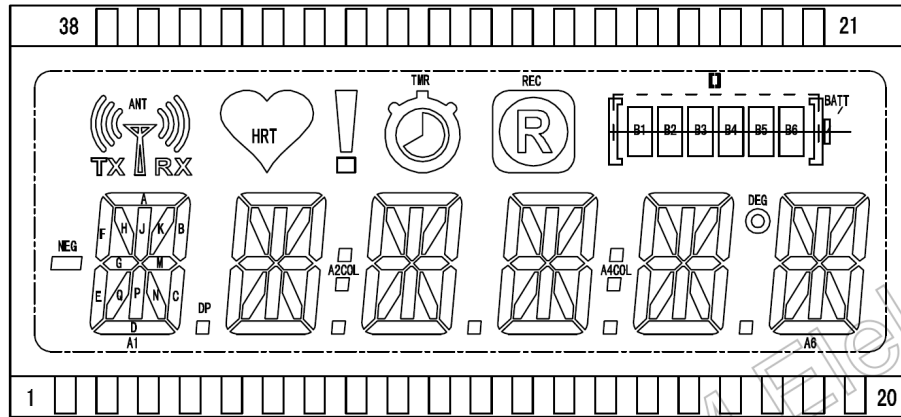


Figure 5.1: ADKOM FH-1138P LCD Monitor

microcontroller controls multiple segments on the display by switching fast between them. With two-way multiplexing, one pin on the MCU controls two segments on the display and, in addition, there are two lines called common backplanes which act as the ground lines. The display on our board is interfaced based on four-way multiplexing. Therefore, the number of pins needed on the MCU is  $(108/4)$  27 and there are four common backplane lines. The LCD controller uses a clock signal in order to generate the waveforms that control the segments. Our configuration will use ACLK based on the 32 KHz crystal.

### Segment Memory-Mapping

Our display has six digits and each one is 14-segment, as shown in Figure 5.1. Such 14-segment digits are alphanumeric and can show the whole alphabet. An example font is shown in Figure 5.2.

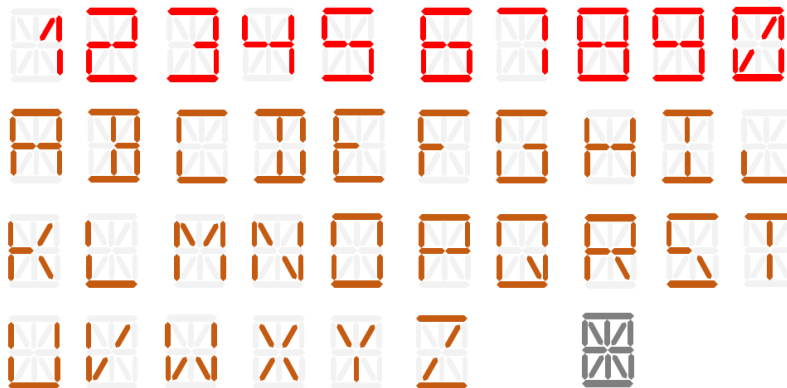


Figure 5.2: Alphanumeric font for a 14-segment display.

The leftmost digit in Figure 5.1 shows how the segments are labeled (A, B, C...). The outer ring and the two middle horizontal bars constitute an 8-segment display that resembles the classic 7-segment display (G and M would be one segment in the 7-segment display). Therefore, if we only want to display numbers, we can use these eight segments and keep the other segments turned off. The leftmost digit on

LCDMEM	Port Pin	FR6989 Pin	LCD Pin	COM3	COM2	COM1	COM0	Port Pin	FR6989 Pin	LCD Pin	COM3	COM2	COM1	COM0
LCDM22	P2.4	S43						P2.5	S42					
LCDM21	P2.6	S41						P2.7	S40					
LCDM20	P10.2	S39	16	A4H	A4J	A4K	A4P	P5.0	S38	15	A4Q	A4COL	A4N	A4DP
LCDM19	P5.1	S37	14	A4A	A4B	A4C	A4D	P5.2	S36	13	A4R	A4F	A4G	A4M
LCDM18	P5.3	S35	34	B5	B3	B1	␣	P3.0	S34					
LCDM17	P3.1	S33						P3.2	S32					
LCDM16	P6.7	S31	20	A5H	A5J	A5K	A5P	P7.5	S30	19	A5Q	DEG	A5N	A5DP
LCDM15	P7.6	S29	18	A5A	A5B	A5C	A5D	P10.1	S28	17	A5E	A5F	A5G	A5M
LCDM14	P7.7	S27	33	B6	B4	B2	BATT	P3.3	S26					
LCDM13	P3.4	S25						P3.5	S24					
LCDM12	P3.6	S23						P3.7	S22					
LCDM11	P8.0	S21	4	A1H	A1J	A1K	A1P	P8.1	S20	3	A1Q	NEG	A1N	A1DP
LCDM10	P8.2	S19	2	A1A	A1B	A1C	A1D	P8.3	S18	1	A1E	A1F	A1G	A1M
LCDM9	P7.0	S17	38	A6H	A6J	A6K	A6P	P7.1	S16	37	A6Q	TX	A6N	RX
LCDM8	P7.2	S15	36	A6A	A6B	A6C	A6D	P7.3	S14	35	A6E	A6F	A6G	A6M
LCDM7	P7.4	S13	8	A2H	A2J	A2K	A2P	P5.4	S12	7	A2Q	A2COL	A2N	A2DP
LCDM6	P5.5	S11	6	A2A	A2B	A2C	A2D	P5.6	S10	5	A2E	A2F	A2G	A2M
LCDM5	P5.7	S9	12	A3H	A3J	A3K	A3P	P4.4	S8	11	A3Q	ANT	A3N	A3DP
LCDM4	P4.5	S7	10	A3A	A3B	A3C	A3D	P4.6	S6	9	A3R	A3F	A3G	A3M
LCDM3	P4.7	S5						P10.0	S4	32	TMR	HRT	REC	!
LCDM2	P4.0	S3						P4.1	S2					
LCDM1	P1.4	S1						P1.5	S0					

Figure 5.3: Segment mapping (LaunchPad User's Guide (slau627a) p. 13)

the display is labeled A1 and the rightmost is labeled A6, as shown in Figure 5.1.

In order to program the display, the segments are associated with memory-mapped variables called LCDMx (e.g. LCDM1, LCDM2...) that are shown in Figure 5.3. This figure was obtained from the LaunchPad User's Guide (slau627a) as it depends on the wiring between the display and the MCU. Each LCDMx variable is 8-bit and, therefore, maps eight segments on the display. Let's find the mapping of the rightmost digit on the display, A6. We can see that the variable LCDM8 maps the following segments of A6: A, B, C, D, E, F, G and M, in this order, starting from the most significant bit. Therefore, this variable acts like a 7-segment display (technically 8-segment) on the rightmost digit. We can also see that the variable LCDM9 maps the other segments of A6 which are: H, J, K, P, Q and N. It also maps the TX and RX symbols on the display. Accordingly, the table in Figure 5.3 enables us to find the variables LCDMx that map all the segments on the display.

The LCDMx variables are active high. Writing 1 to the bit position turns the corresponding segment on. For example, to display 8 on digit A6, we would need to turn on all the segments in LCDM8. This can be done in the software with: `LCDM8 = 0xFF`; although it's not a good practice to use hex numbers in the code because they are not easily readable. Instead, we'll store the shapes of the digits in an array.

To facilitate displaying digits (0 to 9) on the display, let's declare an array that stores the shapes of all the digits. The array is shown below:

```
unsigned char LCD_Shapes[10] = {0xFC, 0x60, 0xDB, ...};
```

In this array, we store the shape of 0 at index 0, the shape of 1 at index 1, etc. This makes it easy to retrieve the shapes and display them. Let's find the shape of 0. By looking at the layout of LCDM8 in Figure 5.3, we can see that segments A, B, C, D, E and F should be turned on while segments G and M should be turned off. Based on the format of LCDM8, this corresponds to the binary value 1111 1100 or 0xFC. Therefore, we store 0xFC (shape of 0) at index 0 in the array above. Similarly, we found the shapes of 1 and 2 to be 0x60 and 0xDB. Based on the array, we can show the digit 8 on A6 with the following line of code: LCDM8 = LCD\_Shapes[8]; This is much more readable than using hex values.

The code below prints the number 430 on the right side of the display (on digits A4, A5, A6). The code contains an LCD initialization function that maps ACLK to the 32 KHz crystal so that it's used by the LCD controller and configures the latter based on 4-way multiplexing. For this code to work, complete the array that stores the shapes of the digits.

```
// Sample code that prints 430 on the LCD monitor
#include <msp430fr6989.h>
#define redLED BIT0           // Red at P1.0
#define greenLED BIT7         // Green at P9.7
void Initialize_LCD();

// The array has the shapes of the digits (0 to 9)
// Complete this array...
const unsigned char LCD_Shapes[10] = {0xFC, 0x60, 0xDB, ...};

int main(void) {
    volatile unsigned int n;
    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    PM5CTL0 &= ~LOCKLPM5;     // Enable GPIO pins

    P1DIR |= redLED;           // Pins as output
    P9DIR |= greenLED;
    P1OUT |= redLED;           // Red on
    P9OUT &= ~greenLED;        // Green off

    // Initializes the LCD_C module
    Initialize_LCD();

    // The line below can be used to clear all the segments
    //LCDCMEMCTL = LCDCLRM;     // Clears all the segments
```

```

// Display 430 on the rightmost three digits
LCDM19 = LCD_Shapes[4];
LCDM15 = LCD_Shapes[3];
LCDM8  = LCD_Shapes[0];

// Flash the red LED
for(;;) {
    for(n=0; n<=60000; n++) {} // Delay loop
    P1OUT ^= redLED;
}

return 0;
}

//*****
// Initializes the LCD_C module
// *** Source: Function obtained from MSP430FR6989's Sample Code ***
void Initialize_LCD() {
    PJSEL0 = BIT4 | BIT5;          // For LFXT

    // Initialize LCD segments 0 - 21; 26 - 43
    LCDCPCTL0 = 0xFFFF;
    LCDCPCTL1 = 0xFC3F;
    LCDCPCTL2 = 0x0FFF;

    // Configure LFXT 32kHz crystal
    CSCTL0_H = CSKEY >> 8;        // Unlock CS registers
    CSCTL4 &= ~LFXTOFF;           // Enable LFXT
    do {
        CSCTL5 &= ~LFXTOFFG;      // Clear LFXT fault flag
        SFRIFG1 &= ~OFIFG;
    }while (SFRIFG1 & OFIFG);      // Test oscillator fault flag
    CSCTL0_H = 0;                  // Lock CS registers

    // Initialize LCD_C
    // ACLK, Divider = 1, Pre-divider = 16; 4-pin MUX
    LCDCCCTL0 = LCDDIV__1 | LCDPRE__16 | LCD4MUX | LCDLP;

    // VLCD generated internally,
    // V2-V4 generated internally, v5 to ground
    // Set VLCD voltage to 2.60v
    // Enable charge pump and select internal reference for it
    LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;

```

```

    LCDCCPCTL = LCDCPCLKSYNC;    // Clock synchronization enabled

    LCDCMEMCTL = LCDCLRM;        // Clear LCD memory

    //Turn LCD on
    LCDCTL0 |= LCDON;

    return;
}

```

### Displaying a 16-bit Unsigned Number

Modify the code by writing a function that prints a 16-bit unsigned integer to the display. The function's header is shown below. The parameter is of type 'unsigned int' since this type is 16-bit in the MSP430 compiler. An unsigned 16-bit value goes up to 65,535, therefore at most five digits are needed and can be accommodated on our display.

```
void lcd_write_uint16 (unsigned int n);
```

Perform the following and answer the questions in your report:

- Complete the code and demo it to the TA.
- Test the function with small values and with large values; also test transitions from large values to small values to ensure unused digits are being cleared.
- Submit the code in your report.

## 5.2 Implementing a Counter

Write a code that shows an incrementing unsigned 16-bit counter on the display (0, 1, 2...). The counter goes up by one every second. Use the timer with interrupts based on the 32 KHz crystal. When button S1 is pushed, the counter goes to zero and continues counting. When button S2 is pushed, the counter jumps up by 1000 and continues counting. Interface the buttons with interrupts.

Perform the following and answer the questions in your report:

- Complete the code and demo it to the TA.
- Compare the timing to your phone's stopwatch for at least 20 seconds and make sure the timing matches closely (the crystal is accurate).
- Submit the code in your report.

### 5.3 Application: Utility Chronometer

You are working as an embedded engineer at a sports equipment company and you are asked to design software that implements a utility chronometer that can be used as either a chronometer or a clock. The chronometer counts seconds, minutes and hours. The application has the following requirements. Pushing S1 starts/stops the timer. Long pressing S1 resets the time to zero and stops the counting. The chronometer can be used as a clock. Pushing S2 fast forwards the time value in order to set the current time. Holding S2 then pushing S1 rewinds the clock backwards (in case we went past the time that we're trying to set). Finally, when the timer is counting, the colon sign between hours and minutes should blink and the chronometer logo should be on. When the timer is stopped, the exclamation point sign on the display should be on. Use the timer with interrupts in order to advance the time. Interface the push buttons with interrupts. For some tasks that happen intermittently (e.g. long press, fast forward and fast rewind), it's acceptable to poll the buttons.

A demo of the utility chronometer can be found at the link below:

<https://youtu.be/WflUzULpUeQ>

Perform the following:

- Complete the code and demo it to the TA.
- Explain your design.
- What is the maximum duration your chronometer application supports? Explain.
- Submit the code in your report.

### Student Q & A

1. 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".
2. 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?
3. 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?