Middle East Technical University

Department of Computer Engineering

CENG 336

Introduction to Embedded Systems Development

Spring 2021

THE3

Due: 07 June 2022, 23:55

Submission: via **ODTUClass**

Note: *This PDF uses animated images and is best viewed on PDF viewers that support javascript. On linux you can use Okular and on Windows you can use Adobe Acrobat. If your PDF viewer doesn't support javascript, animated images are also provided as .gif files.*

1 Objectives

This assigment aims to familiarize you with the usage of analog-to-digital converter (ADC), LCD, timers, and LEDs. Your task is to create a custom text entry program for the LCD. You will use the C programming language to create your program for the PIC microcontroller. This is a group assignment. Any clarifications or updates regarding this assignment will be posted on **ODTUClass**. There is a discussion forum dedicated for your questions about this assignment on **ODTUClass**.

2 Scenario

Using the custom text entry program, the user will be able to enter any character he wants at any point in the first line of the LCD. By using the potentiometer, the user will be able to move the cursor to the cell he wants to insert a character. By using the buttons on the board, the user can select a character from the predefined set of characters (`[a-z0-9]`) or he can insert a custom character he defines using the LEDs. When the user is done with the text entry, the text will scroll on the LCD infinitely.

When the user wants to define a custom character, he will go into the custom character definition (CCD) mode. In this mode, the user will turn on and off LEDs in a 4x8 LED grid on the board. To move around this grid the user will use an invisible cursor which will be controlled by the buttons on the board. Since the cursor is invisible, the coordinates of the cursor will be displayed in the rightmost two digits of the seven-segment display.

Due to constraints of the LCD, user can define at most 8 custom characters. Number of characters defined by the user will be shown in the leftmost digit of the seven-segment display.

3 Specifications

3.1 Initial Configuration

The program will use the LCD (PORTB & PORTD), seven-segment display (PORTH & PORTJ), buttons (PORTE & PORTF), and LEDs (PORT[A-D]). Initially the LCD will be empty, except the cursor. Cursor's initial position will depend on the state of the potentiometer when the program starts. Additionally, the seven-segment display will show 0 for number of custom characters defined and "0 0" for the position of the LED cursor. At this stage the status of LEDs is not important.

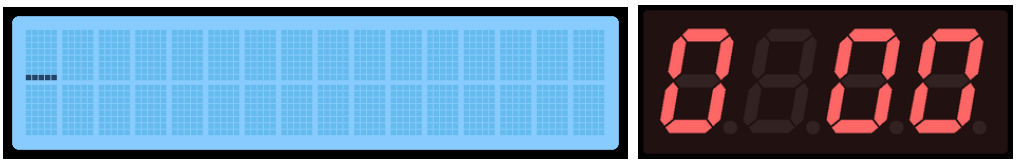


Figure 1: Initial status of the LCD and the seven-segment display. Assuming that the potentiometer is turned counter-clockwise all the way.

3.2 Text Entry Mode

When the program starts it will go directly into the *Text Entry Mode* (TEM). In this state, the user is able to move the cursor around the first line of the LCD using the potentiometer. Which of the 16 cells in the first line of the LCD the cursor points to will depend on the value of the ADC. The 10-bit ADC gives us values between 0 and 1023. This range will be divided into 16 equal parts, each part corresponding to a cell in the LCD. The cursor will move to the cell which the value of ADC corresponds.

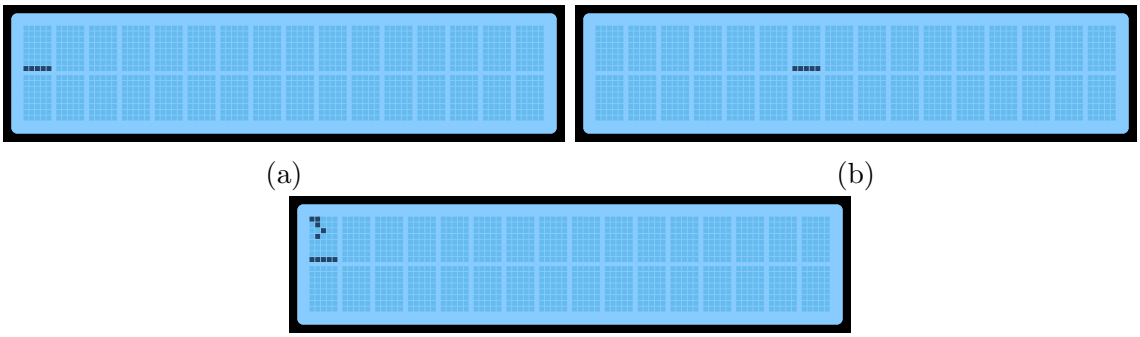
To change the text at the cursor's position, the user will use the `RF1` and `RE1` buttons. A single press of the `RF1` button will cycle the displayed character through the set of predefined characters in the forward direction. And pressing the `RE1` button will cycle the displayed character in the backward direction. While the last character of the set is being displayed pressing the `RF1` button will make the displayed character the first one in the predefined character in the set. Similarly, pressing the `RE1` button while the first character in the set is being displayed will change the character to the last one in the set. The predefined character set will be defined as:

```
PREDEFINED = " abcdefghijklmnopqrstuvwxyz0123456789"
```

Note that the predefined characters start with a space character. Throughout this assignment you can assume that and empty cell is equal to a space character.

In addition to the ability to insert predefined characters, the user can also insert the custom characters he/she already defined to the cursor's position. Pressing the `RE2` and `RE0` buttons will cycle through the defined custom characters. However, this time the cycle length will be determined by the number of defined custom characters. For example, if the user has defined 3 custom characters (out of 8 available) after the 3rd custom character the

characters you display should wrap around. That is, if the current character is space (␣) and user has defined 3 custom characters, continuously pressing `RE0` will cycle like this: ␣, 1, 2, 3, ␣, 1, 2 . . . (1, 2, 3 denote the index of custom characters and not the actual characters defined. See Figure 2).



(a) (b)

(c)

Figure 2: a) Moving the cursor with the potentiometer (cursor_move.gif). b) Cycling through predefined characters (char_cycle.gif). c) Cycling through 3 custom defined characters (custom_cycle.gif).

Finally while in the *Text Entry Mode*, pressing the `RF0` button will put the program into the *Character Define Mode*, and pressing the `RF2` button will put the program into the *Text Scroll Mode*.

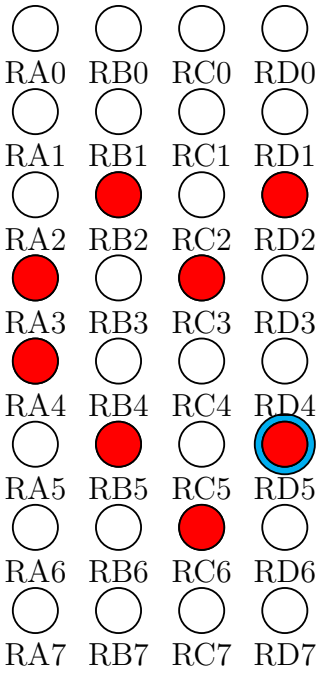
3.3 Character Define Mode

In *Character Define Mode* (CDM), the user will use the R[ABCD][0-7] LEDs to enter a custom character. Each cell in the LCD is a 5x8 dot matrix and each LED corresponds to a single dot in the LCD cell. However, since only 4 ports for LEDs are used, the program will assume that the rightmost column of defined character is always empty. The user will use the `RF2` button to turn on/off the LEDs. If a LED is turned on, corresponding dot in the dot matrix will be filled and vice versa.

The user will use the `RF1`, `RE1`, `RE2`, and `RE0` buttons to move an invisible cursor up, down, left, and right over the LEDs, respectively. With the press of the `RF2` button, the LED under the cursor will toggle its state. Meanwhile the cursor's coordinates will be displayed in the seven-segment display. RA7 LED's coordinates are `(0, 0)` and the RD0 LED's coordinates are `(3, 7)`. The cursor should not go over the edges of the LED grid or wrap around. For example, pressing `RE0` when the cursor is at RD5 `(3, 2)` will have no effect and the cursor will stay at `(3, 2)`.

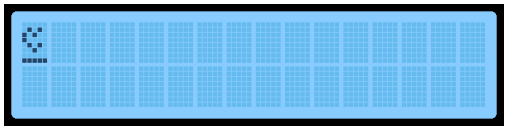
Note: *Due to connections of the board, RA0 and RA1 LEDs will always be turned off and their state cannot be changed. Therefore, all of the custom characters will have their 2 dots on the bottom left empty.*

After entering a custom character by turning LEDs on and off, the user will press the `RF0` button to go back into the TEM. At the same time the character defined by the LEDs will be shown on the LCD. Also the custom character counter at the seven-segment display will increase by 1 and the coordinates of the invisible cursor will be reset to (0, 0).



(a) LEDs and invisible cursor position (blue circle) in CDM.

(b) seven-segment display during (a)



(c) LCD status after pressing the `RF0` button. Character defined by the LEDs is transferred to LCD.

(d) seven-segment display status after pressing the `RF0` button. Number of defined custom characters increased by 1 and invisible cursor position is reset.

4 Text Scroll Mode

In *Text Scroll Mode* (TSM), the text entered by the user will start to scroll on the second line of the LCD. Meanwhile there will be a static "`finished`" text on the first line (Figure 4). The cursor on the LCD will also disappear. At this mode no inputs will be accepted, and the program will stay in this state until the board has been reset.

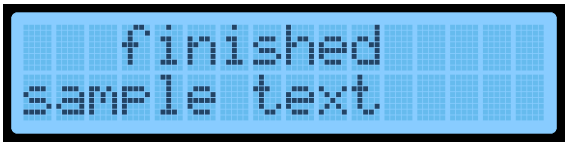


Figure 4: State of the LCD in TSM (text_scroll.gif).

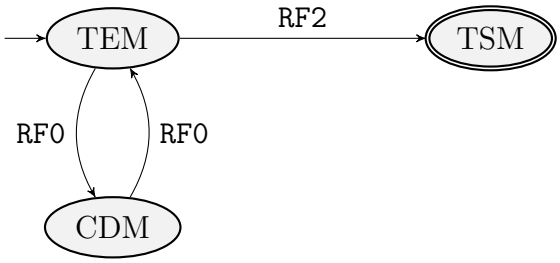


Figure 5: State diagram of the whole program

5 ADC Usage

ADC module allows conversion of an analog input signal to a corresponding 10-bit digital number. ADC module is controlled and configured by five registers:

1. A/D Result High Register (`ADRESH`)
2. A/D Result Low Register (`ADRESL`)
3. A/D Control Register 0 (`ADCON0`)
4. A/D Control Register 1 (`ADCON1`)
5. A/D Control Register 2 (`ADCON2`)

`ADCON0` controls the operation of the module, `ADCON1` configures the pin functions, `ADCON2` configures the clock source, acquisition time and justification. `ADRESH` and `ADRESL` registers contain the result of the conversion operation.

When the `GO/`$\overline{\texttt{DONE}}$ bit is set the ADC module starts the conversion. When the conversion is complete the result is loaded into the `ADRESH:ADRESL` registers and the `GO/`$\overline{\texttt{DONE}}$ bit is cleared and the `ADIF` interrupt bit is set. Which 10-bits of the `ADRESH:ADRESL` is used is defined by the `ADFM` bit of the `ADCON2` register.

As described in the datasheet, the following steps should be followed to perform an A/D conversion:

1. Configure the A/D module:
   - Configure analog pins, mark corresponding `TRIS` bits as input. Configure voltage reference and digital I/O (`ADCON1`)
   - Select the input channel for the conversion (`ADCON0`)
   - Select acquisition time (`ADCON2`)
   - Select conversion clock (`ADCON2`)
   - Turn on the module (`ADCON0`)
2. Configure the A/D interrupt (optional):
   - Clear `ADIF` bit
   - Set `ADIE` bit
   - Set `GIE` bit
3. Wait for the required acquisition time (if required)
4. Start conversion by setting `GO/`$\overline{\texttt{DONE}}$ bit (`ADCON0`)
5. Wait for conversion to finish by either polling the `GO/`$\overline{\texttt{DONE}}$ bit to be cleared or by waiting for the A/D interrupt.
6. Read `ADRESH:ADRESL`, clear `ADIF` bit (if required)
7. Go to Step 1 or Step 2 for the next conversion.

See the datasheet for acquisition time and clock calculations.

6 LCD Usage

Note:*Some functions related to the LCD will be provided for you (e.g., LCD Initialization sequence). To give you some ideas, some definitions and function declarations will be provided as comments. You* ***don't*** *need to implement these functions, you can just ignore them and write your own functions.*

Our development board has the HD44780 (16×2) LCD connected to it. It has 16 pins that we can use to communicate with it. 5 of these 16 pins are used for ground(`VSS`), power(`VDD`), contrast of display(`V0`) and backlight(`LED+/LED-`) and aren't directly related to the operation of the LCD. These are already connected properly and we don't need to look into these 5 pins. Moreover, the `R/W` pin on the board is hardwired as 0 (connected to ground). When `R/W` pin is set `HIGH`, it is used to read values from the LCD's memory but we do not need such functionality in our program. Therefore, we do not need to consider `R/W` pin as well.

So the pins we need to learn how to operate are Register Select(`RS`), Enable(`E`) and data bus(`D0:D7`).

LCD has two operating modes: 8-bit and 4-bit. 8-bit and 4-bit describes how many pins of the data bus we will use to operate the LCD. Since on our board, only 4 bits of the data bus is connected to the microcontroller we must use the 4 bit mode. LCD uses 8-bits for all of its instructions. In 8-bit mode the complete instruction can be put on `D0:D7` and LCD can read the whole instruction in one go. In 4-bit mode however, we can only send 4 bits of an instruction at a time via `D4:D7`. Therefore, LCD needs to read 2 4-bit words to complete the instruction.

We can send either *commands* or *data* to the LCD using the data pins. We use the `RS` pin to specify how the LCD should interpret the values of `D4:D7`. When `RS` is set as `LOW`, LCD interprets the values on the bus as a *command*, and whent `RS` is `HIGH` it interprets the values on the bus as *data*. `E` pin is used to notify the LCD that the contents of the bus and `RS` are ready to be read. When we pulse (i.e., setting `HIGH` and then `LOW` again) the `E` pin, LCD reads the information and performs the operation we specified. Since we are operating in the 4-bit mode, to send a single instruction we need to set the values on the bus and pulse the `E` pin two times.

With commands, we can perform actions such as clearing the display, setting cursor position, turning cursor display on/off, shifting the display etc. And by sending data we can write characters to the display. The next section goes over the usage of the LCD via an example.

→ Datasheet for the LCD can be found here. Also, Ben Eater has a great video on operating the HD44780.

Hint: *Take a look at the Instructions table on page 24 (Table 6) of the LCD's datasheet.*

Example Operation

In this example, we will clear the display, put the cursor on the beginning of the second line, and then write the letter 'A' to the screen. We assume that the LCD has been properly initialized beforehand.

I. **Clear Display**: We need to send the `0x01` command to LCD.

1. Set `RS = LOW`: we are sending a command.
2. Set `D4:D7 = 0b0000`: 4-bit mode, send high 4 bits first.
3. Pulse `E`: set `E HIGH` wait the required amount and set `LOW` again.

- *LCD reads the high 4-bits of the command.*

4. Set `D4:D7 = 0b0001`: 4-bit mode, send low 4 bits next.
5. Pulse `E`: set `E HIGH` wait the required amount and set `LOW` again.

- *LCD now has the whole command and clears the display.*

II. **Move Cursor**: We need to send the *DDRAM Address* command with the address corresponding to the first cell of the second line (`0xC0`).

1. Set `RS = LOW`: we are sending a command.
2. Set `D4:D7 = 0b1100`: 4-bit mode, send high 4 bits first.
3. Pulse `E`

- *LCD reads the high 4-bits of the command.*

4. Set `D4:D7 = 0b0000`: 4-bit mode, send low 4 bits next.
5. Pulse `E`

- *Cursor is now at the beginning of the second line.*

III. **Write** `A`: We need to send the data 'A' (`0x41`) to be written at the cursor.

1. Set `RS = HIGH`: we are sending data.
2. Set `D4:D7 = 0b0100`: 4-bit mode, send high 4 bits first.
3. Pulse `E`

- *LCD reads the high 4-bits of the data.*

4. Set `D4:D7 = 0b0001`: 4-bit mode, send low 4 bits next.
5. Pulse `E`

- *A is printed at the beginning of the second line.*

7 Hints

1. Here are the ports and pins the components use:
   - **LCD:** `PORTD[4:7]` for data bus, `PORTB[5]` for enable, `PORTB[2]` for register select.
   - **7-Segment Display:** `PORTH[0:3]`, `PORTJ`.
   - **ADC:** `PORTH[4](AN12)` is used for the potentiometer. Don't forget to set it as input.
   - **Buttons:** `RE0`, `RE1`, `RE2`, `RF0`, `RF1` , `RF2`
   - **LEDs:** `RA[0-7]`,`RB[0-7]`,`RC[0-7]`,`RD[0-7]`,
2. You might have to store and restore port values (just like callee-saved registers from Computer Organization course) to have a stable program depending on your design.
3. You are free to use as many timer modules as you want. However, you need to clearly state which timer(s) are used for what purpose in your comments. You should also state the modes (8/16-bit) and prescalers if any.
4. It is beneficial to avoid function calls in ISRs. You may use some counters or flags and do the function calls or calculations outside the ISR.

8 Regulations

1. You will code your program using PIC C language. You should use XC8 C compiler for MPLAB X.
2. Your program should be written for `PIC18F8722` working at 40MHz.
3. **Your code MUST be properly commented and you must provide a complete description of the flow of your program just before the `main` function.** 5% of your grade will be given by the quality of your comments.
4. All the timing related tasks should be performed by timers. You cannot use busy waits. However, you can use `__delay_us()` for LCD related tasks only.
5. You must get the ADC result by using its interrupt, polling is not allowed.
6. There is already a discussion forum for THE3, please ask your questions there. Unless you have *really specific* question about **your code** use the forum! If you think that your question is too specific to ask on the forum or private you can ask your questions via email to Hakan`hbostan@ceng.metu.edu.tr`.
7. RTFM! Use your lecture notes and datasheets extensively. Consult these resources first before asking any questions to the assistants or on the forums.
8. You will submit your codes as a `.tar.gz` archive through ODTUClass. Please tarball the entire MPLAB project folder and not just your source code.