Department of Computer Engineering

BLG 351E
Microcomputer Laboratory
Experiment Report

Experiment No : 7

Experiment Date : 27.11.2017

Group Number : Monday - 8

Group Members :

ID	Name	Surname
150150114	EGE	APAK
150140066	ECEM	GÜLDÖŞÜREN
150150701	YUNUS	GÜNGÖR

Laboratory Assistant : Ahmet Arış

1 INTRODUCTION

In this experiment, we expanded our knowledge about MSP430 Education Board and 16x2 dot matrix LCD design. We used the LCD, which is placed on the board, in order to display the predefined char array. We printed a text on two lines of the LED. Since LCD's default mode is 8-bit mode, we configured our LCD display such that we can communicate in 4-bit mode. Then we divided our 8-bit ACII characters into two nibbles, which contain 4 bits, and send these nibbles to LCD. We followed the initialization and configuration steps from the flowchart, which was provided alongside with experiment datasheet.

2 EXPERIMENT

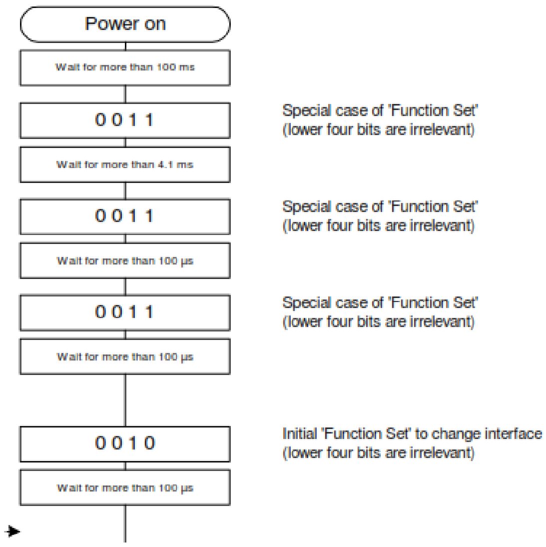
2.1 PART 1

In this experiment, we are obliged to implement a print function. This function uses a char array as input and display it on the LCD screen. One of the main points was that, we need to keep track of the character "\n" and "\0" in order to understand when to break a line and continue from a line below and when our input array has ended, respectively.

At the beginning of our code, we dealed with P1 and P2 ports of the board. The most significant 4 bits of P1 port are directly connected with LCD and the other bits of the P1 is grounded. Since we want to send data to LCD, we need to use these 4 most significants bits as outputs. So we arranged the directions of P1 port as output. Then we arranged the most significant 2 bits of Port 2 as outputs too. Because P2.6 enables the LCD and P2.7 refers to R/S bit. When R/S is equal to 0; LCD reads instruction, otherwise it reads data. And we also adjusted the modes of P1 and P2 ports since we need to use these ports in I/O mode.

After the arrangements about the ports, our code follows predefined initialization and configuration steps in the subroutine "initLCD". When we first started using LCD works in 8-bit mode by default. And we want to convert LCD such that it works in 4-bit mode. Inside this subroutine we used a "delay" subroutine in order to obtain the delays as showed as flowchart. Delay subroutine makes our program to wait for 20 mikro seconds, so before we called this subroutine we calculated how many times we need to call this routine in order to obtain desired delay time. We stored this value in R14 register and then called delay subroutine. We also used "sendCMD" subroutine which will be explained later.

Beginning from line 58, LCD start to work in 4-bit mode. So, lines 39-57 of the initLCD subroutine refers the part of the flowchart which is inserted below.



Following the flow chart again, we set N and F bits as N=1 and F=0. Since we arranged N=1 we make ourselves able to use the 2 lines of the LCD and by arranging F=0, we make sure that we are going to use 5X7 font.

In the following instruction, we firstly close display and call "clear " with the aim of cleaning the screen. After calling display OFF instruction, we call display on instruction. After this call, our LCD is actually ready for display since we took care of initialization part.

After finishing the initialization part, our code goes into the subroutine "print". The first thing that we did in this subroutine was returning the cursor to the home. In R12 we store the number of the element that we are going to print on the screen, and in R11 we stored the address value in which we will print the character. Because we start to print the array in the next line when we see the new line character. During the print subroutine we always check the character that is going to be written on the LCD and do comparison in order to check if we accessed at the end of the char array. If so, we return from print function. Otherwise, we check if the character is equal to #0Dh or not in order to break a line when it is necessary. If the character is not equal to an indicator of new line or end of the array, we store the value of R11 in R13. We use R13 and R14 as parameters of sendDATA subroutine. R13 holds, which address of the DDRAM should the data be written and R14 holds what to write in that address. At the end of the print routine, we increment R12 and R11 values in order to write the right character to the right address.

The working principles of sendDATA and sendCMD is actually pretty similar. So we created a new subroutine sendLCD and grouped these two subroutines under this subroutine. As it has already been explained, R14 holds what to write. sendLCD firstly moves the value of R14 into the most 4 significant bits of Port 1. And then calls triggerEN subroutine. If LCD is in 8-bit mode it returns directly. Otherwise, it shifts the content of the R14 in order to obtain the 4 LSB bits as MSB 4 bits. After the shifting, it sends the R14 value into port 1 and pulses again.

The difference between the subroutines sendCMD and sendDATA is the value of R/S bit. In sendCMD the value R/S bit(P2.7) is equal to 0, since we want to send command. To do so, we clear the 7^{th} bit. And then call sendLCD subroutine.

In sendDATA subroutine, we set the most significant bit of R13 as 1. Since it is required according to datasheet. We set the R/S bit as 1 in order to write to DDRAM, and call sendCMD.

We use triggerEN subroutine in order to change the value of EN to high then change it back to 0.

```
                                                    ; references to current section.

        .data
string  .byte "ITU - Comp. Eng.", 0Dh, "MC Lab. 2017", 00h
;-------------------------------------------------------------------------------
RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer


;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
            mov.b #0FFh, P1DIR
            mov.b #0C0h, P2DIR
            mov.b #000h, P1OUT
            mov.b #000h, P2OUT
            mov.b #000h, &P2SEL
            mov.b #000h, &P2SEL2

            call #initLCD
loop        call #print
            jmp loop

initLCD     mov.w #02AF8h, R14
            call #delay

            mov.b #03h, R13
initSpec    mov.b #00110000b, R14
            mov.b #01h, R5
            call #sendCMD
            mov.w #005F4h, R14
            call #delay
            dec.b R13
            jnz initSpec

            mov.b #00100000b, R14
            mov.b #01h, R5
            call #sendCMD
            mov #00Bh, R14
            call #delay
            mov.b #00h, R5

            mov.b #00101000b, R14 ; NF
            call #sendCMD
            mov #006h, R14
            call #delay


            mov.b #00001000b, R14 ; Display OFF
            call #sendCMD
            mov #06h, R14
            call #delay


            mov.b #00000001b, R14 ; Clear
            call #sendCMD
            mov #00C1Ch, R14
```

```
                mov.b #00000001b, R14 ; Clear
                call #sendCMD
                mov #00C1Ch, R14
                call #delay


                mov.b #00001100b, R14 ; Display ON
                call #sendCMD
                mov #006h, R14
                call #delay
                ret

print           mov.b #00000010b, R14 ; Return Home
                call #sendCMD
                mov #00C1Ch, R14
                call #delay

                mov.b #00h, R12
                mov.b #00h, R11
print$send      mov.b string(R12), R14
                cmp #000h, R14
                jeq print$end
                cmp #00Dh, R14
                jnz print$cnt
                mov.b #040h, R11
                inc R12
                jmp print$send
print$cnt       mov.b R11, R13
                call #sendDATA
                inc R11
                inc R12
                jmp print$send
print$end       ret



sendCMD         bic   #080h, P2OUT ; Bit clear
                call #sendLCD
                ret

sendLCD         mov.b R14, P1OUT
                call #triggerEn   ; MSB 4bit
                cmp #01h, R5
                jeq sendLCD$e
                rla R14
                rla R14
                rla R14
                rla R14
                mov.b R14, P1OUT
                call #triggerEn   ; LSB 4 bit
sendLCD$e       ret

sendDATA        bis #080h, R13
                push R14
                mov.b R13, R14
                call #sendCMD
                pop R14
```

```
            call #sendCMD
            pop R14
            bis   #080h, P2OUT ; Bit set
            call #sendLCD
            ret


triggerEn   bis #040h, P2OUT
            bic #040h, P2OUT
            ret

delay       mov.w #06h, R15
delay$1     dec.w R15
            jnz delay$1
            dec.w R14
            jnz delay
            ret

;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET

```

3 CONCLUSION

In this experiment, we learned how to work with LCD on the board.