Department of Computer Engineering

BLG 351E
Microcomputer Laboratory
Experiment Report

Experiment No : 8

Experiment Date : 04.12.2017

Group Number : Monday - 8

Group Members :

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

Laboratory Assistant : Ahmet ARIŞ

1. INTRODUCTION

In this experiment, we have improved ourselves about the various peripherals present on the MSP430 Education Board and especially with the DS18B20 1-Wire Bus Thermometer. In the experiment we have set up Port#2.4 since it was connected to the thermometer via the 1-Wire bus. Before doing anything else, we have initialized the thermometer using the necessary sequences noted in the datasheet of the DS18B20. After that, we have sent our commands to convert the temperature with 12-bit resolution and tried to read the data before displaying the result on the LCD display, with the help of the code we have written in the previous experiment.

2. EXPERIMENT

2.1. PART 1

Before doing anything else we are doing some basic setup: We set the ports directions, reset the output values and set selection of Port#2 to GPIO.

For sending and receiving data from the 1-Wire bus, we had to implement two different subroutines: WriteCMD & ReadDATA.

In the WriteCMD subroutine (starting at line 185), we first pull the wire low since both writing 1 and 0 begins with pulling the wire low. The only difference between them is how early the wire should be released. In order to write 1, we are releasing the wire within 15µs and for writing 0 we are pulling the wire low for at least 60 µs and waiting around 70µs in total in both write operations. After writing the LSB, we are shifting the byte we are writing in order to write the next bit and keep doing this until all the bits in the data we want to send is processed and sent.

In the ReadDATA subroutine (starting at line 204), we first pull the wire low as usual and wait a little more than a couple of µs and start sampling within 15µs of pulling the wire low. Since we are reading the data from LSB to MSB, we are either setting or clearing the 16-bit MSB of the register we want to save the data and right-shift the contents to open space for the next value and keep doing this until full 16-bits are read from the 1-Wire bus.

In order to use the DS18B20 thermometer and to transfer any data either to or from it, we had to run an initialization sequence, in which we (as the master of the 1-Wire bus) are pulling the wire low for at least 480µs and reading the presence pulse of the DS18B20.

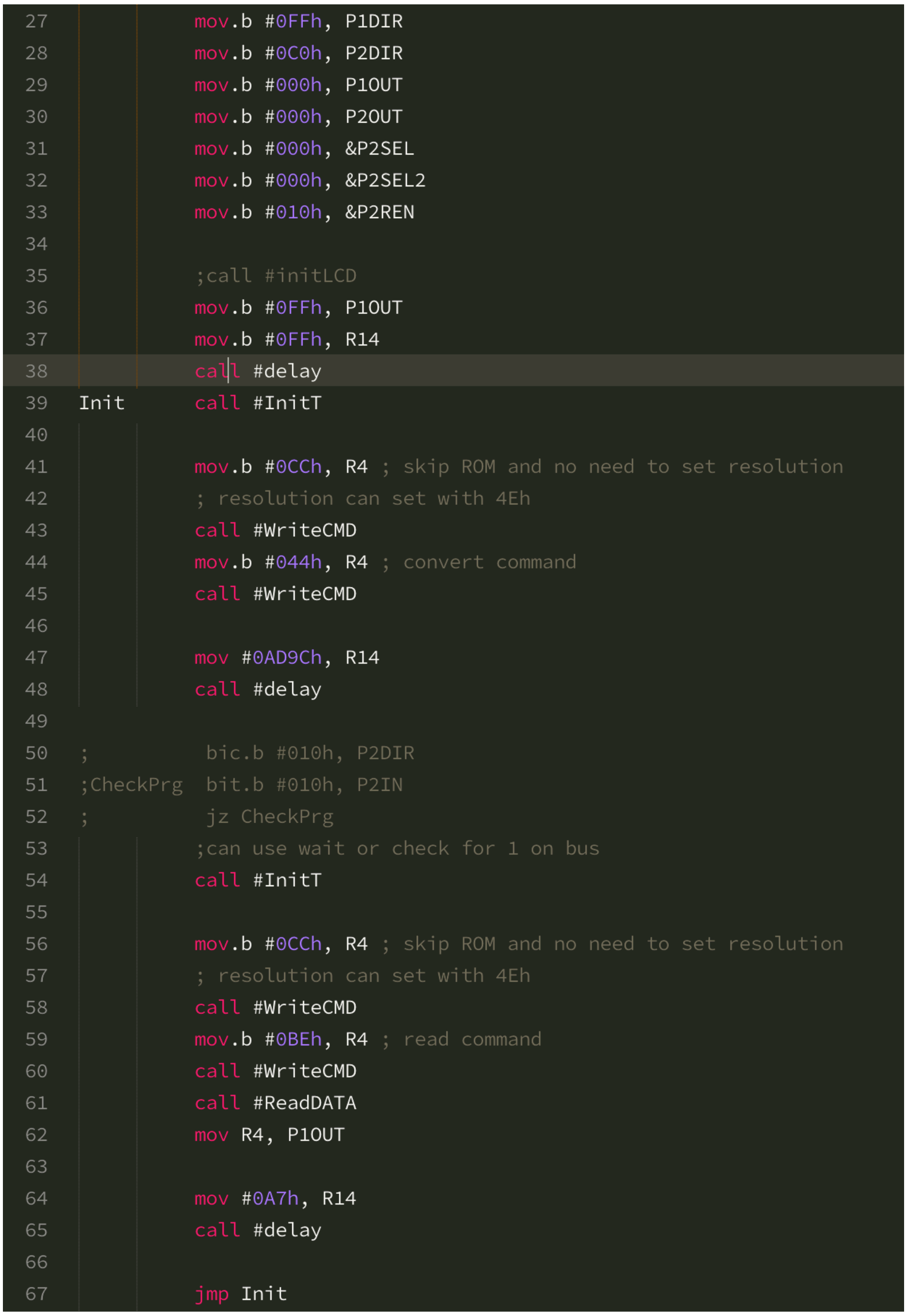
For this sequence (subroutine starting at line 168), we first set the Port#2.4 direction as OUT and set it to 0. After this operation, we are waiting for ~480µs (a little bit more than that) and set the direction of Port#2.4 to IN, just before reading the data in 1-Wire bus, in order to read the presence pulse of the DS18B20. After checking the presence pulse, we make sure that the DS18B20 is responding and return from the subroutine.

After the initialization, we send the Skip ROM command via the WriteCMD subroutine to communicate with the DS18B20 directly and send the ConvertT command right after that. Since the mov.b instruction takes more than 1µs of time, we don’t have to wait between subsequent commands since that time has already passed when the next command is being sent.

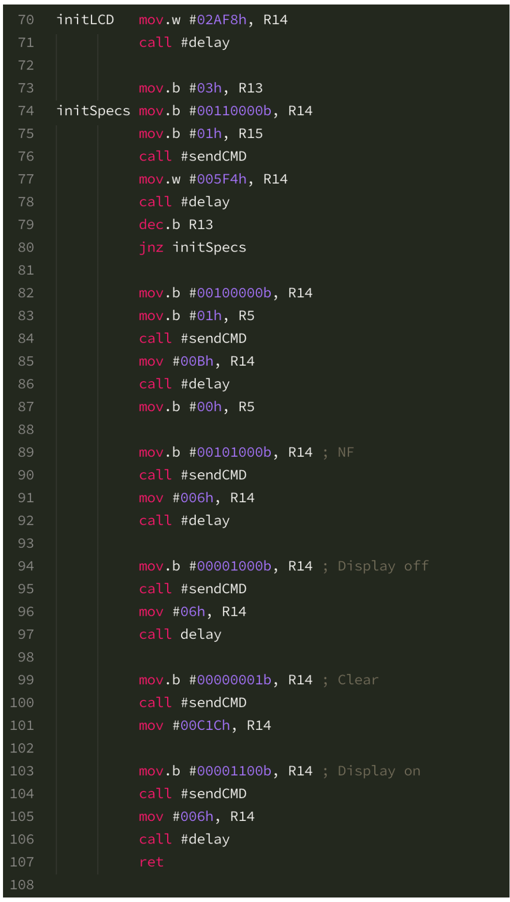
After waiting more than enough time, after sending the ConvertT command, we call the initialization sequence once more, since it is required before sending any commands to the DS18B20 and after sending Skip ROM command, we send the Read command and start reading data with the ReadDATA subroutine and read the data DS18B20 has sent.

Initialization, setup and writing subroutines of the LCD display is exactly same as in the previous experiment. The only difference is how we convert and send data to the LCD display.

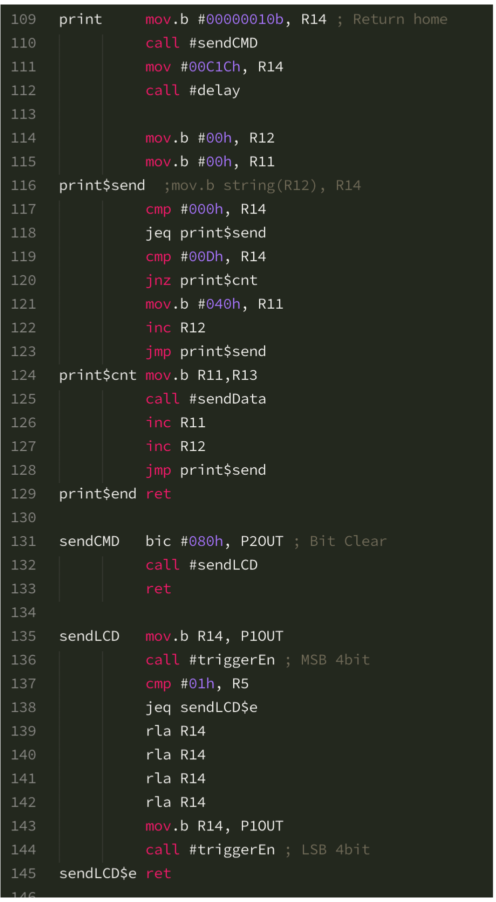
For the testing purposes we have sent the received data directly to Port#1 to observe the value and deferred the writing to the LCD part to the end, we were out of time and had to finish the experiment session.



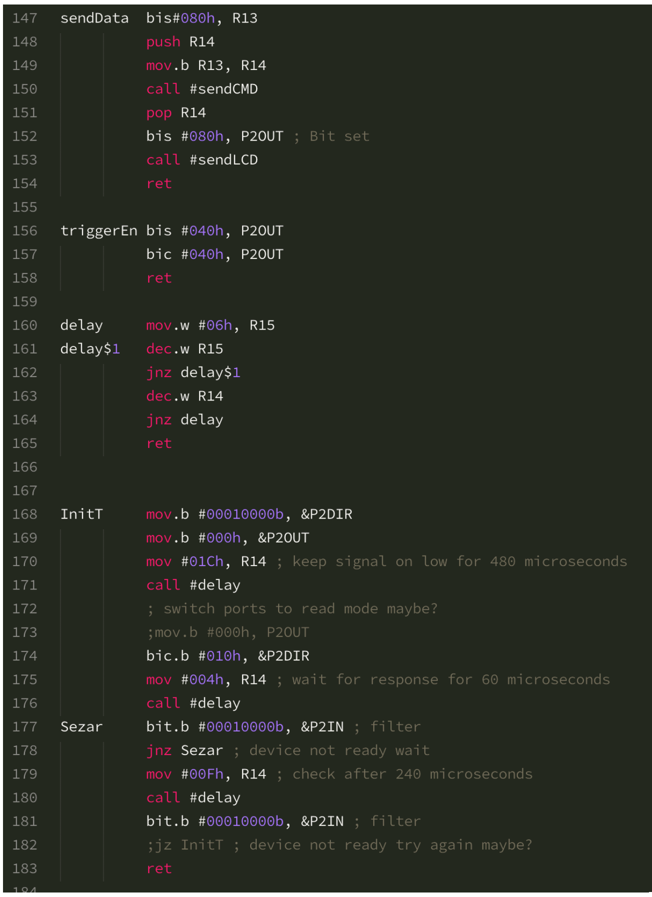
```
          mov.b #0FFh, P1DIR
          mov.b #0C0h, P2DIR
          mov.b #000h, P1OUT
          mov.b #000h, P2OUT
          mov.b #000h, &P2SEL
          mov.b #000h, &P2SEL2
          mov.b #010h, &P2REN

          ;call #initLCD
          mov.b #0FFh, P1OUT
          mov.b #0FFh, R14
          call #delay
Init      call #InitT

          mov.b #0CCh, R4 ; skip ROM and no need to set resolution
          ; resolution can set with 4Eh
          call #WriteCMD
          mov.b #044h, R4 ; convert command
          call #WriteCMD

          mov #0AD9Ch, R14
          call #delay

;          bic.b #010h, P2DIR
;CheckPrg  bit.b #010h, P2IN
;          jz CheckPrg
          ;can use wait or check for 1 on bus
          call #InitT

          mov.b #0CCh, R4 ; skip ROM and no need to set resolution
          ; resolution can set with 4Eh
          call #WriteCMD
          mov.b #0BEh, R4 ; read command
          call #WriteCMD
          call #ReadDATA
          mov R4, P1OUT

          mov #0A7h, R14
          call #delay

          jmp Init
```



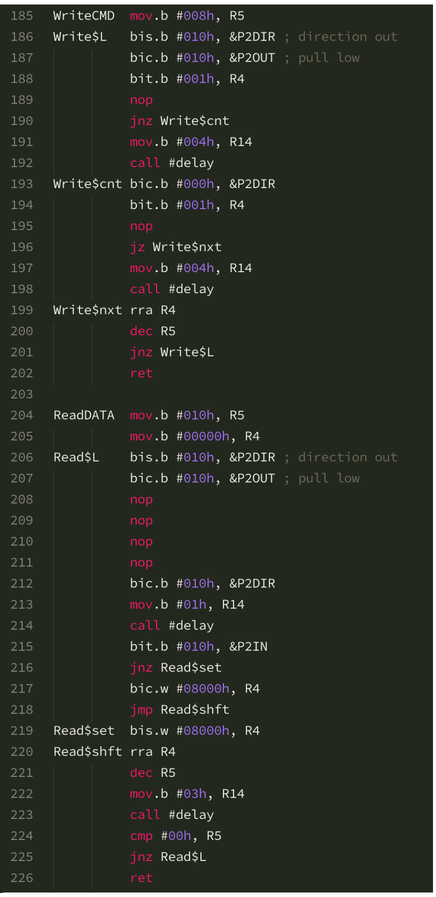
```
initLCD   mov.w #02AF8h, R14
          call #delay

          mov.b #03h, R13
initSpecs mov.b #00110000b, R14
          mov.b #01h, R15
          call #sendCMD
          mov.w #005F4h, R14
          call #delay
          dec.b R13
          jnz initSpecs

          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 #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$send  ;mov.b string(R12), R14
           cmp #000h, R14
           jeq print$send
           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$e
           rla R14
           rla R14
           rla R14
           rla R14
           mov.b R14, P1OUT
           call #triggerEn ; LSB 4bit
sendLCD$e  ret
```



```
sendData  bis#080h, R13
          push R14
          mov.b R13, 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


InitT     mov.b #00010000b, &P2DIR
          mov.b #000h, &P2OUT
          mov #01Ch, R14 ; keep signal on low for 480 microseconds
          call #delay
          ; switch ports to read mode maybe?
          ;mov.b #000h, P2OUT
          bic.b #010h, &P2DIR
          mov #004h, R14 ; wait for response for 60 microseconds
          call #delay
Sezar     bit.b #00010000b, &P2IN ; filter
          jnz Sezar ; device not ready wait
          mov #00Fh, R14 ; check after 240 microseconds
          call #delay
          bit.b #00010000b, &P2IN ; filter
          ;jz InitT ; device not ready try again maybe?
          ret
```



```
WriteCMD  mov.b #008h, R5
Write$L   bis.b #010h, &P2DIR ; direction out
          bic.b #010h, &P2OUT ; pull low
          bit.b #001h, R4
          nop
          jnz Write$cnt
          mov.b #004h, R14
          call #delay
Write$cnt bic.b #000h, &P2DIR
          bit.b #001h, R4
          nop
          jz Write$nxt
          mov.b #004h, R14
          call #delay
Write$nxt rra R4
          dec R5
          jnz Write$L
          ret

ReadDATA  mov.b #010h, R5
          mov.b #00000h, R4
Read$L    bis.b #010h, &P2DIR ; direction out
          bic.b #010h, &P2OUT ; pull low
          nop
          nop
          nop
          nop
          bic.b #010h, &P2DIR
          mov.b #01h, R14
          call #delay
          bit.b #010h, &P2IN
          jnz Read$set
          bic.w #08000h, R4
          jmp Read$shft
Read$set  bis.w #08000h, R4
Read$shft rra R4
          dec R5
          mov.b #03h, R14
          call #delay
          cmp #00h, R5
          jnz Read$L
          ret
```

3. CONCLUSION

In this experiment, the biggest obstacle we have faced was the 1-Wire Bus and the timings. Since we had zero experience with 1-Wire protocol and the timings of the instructions had to be precise, we paid utmost attention to the code we have written. However, even though we have followed the instructions in the data sheet completely, there seemed to be a problem with the communication between the DS18B20 and MSP430 since we couldn’t get any data back from the DS18B20. Even the presence pulse wasn’t present when we pulled the wire low and we have checked that wire is pulled low successfully and couldn’t find any problems. We have even double checked with other assistants but even though we have sent data to DS18B20 from MSP430, we couldn’t get any data back. But from reading the data sheet and experimenting, we have learned about 1-Wire protocol, understood the significance of timings and the reason of this significance through the thorough explanation in the data sheet.