ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E

MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO : 2

EXPERIMENT DATE : 09.10.2019

LAB SESSION : WEDNESDAY - 13.30

GROUP NO : G10

GROUP MEMBERS:

150170062 : Mehmet Fatih YILDIRIM

150180704 : Cihat AKKİRAZ

150180705 : Batuhan Faik DERİNBAY

150180707 : Fatih ALTINPINAR

FALL 2019-2020

Contents

1 INTRODUCTION

Microcontrollers are connected to other electronical devices such as sensors, diodes, displays and System-On-Chip modules for many purposes. One of the interface for communicating with other things is GPIO. GPIO stands for General Purpose Input Output and refers to the fact that the pins can support both output and input functionalities.

2 MATERIALS AND METHODS

This experiment is conducted via using MSP430G2553 microprocessor. This microprocessor is programmed using Code Composer Studio according to desired tasks on the experiment handout. During coding below sources are used:

- MSP430 Education Board Manual [1]
- MSP430 Architecture Chapter 4 [2]
- MSP430 Instruction Set [3]
- Supplementary Chapter 6 General Purpose IO [4]

2.1 Part 1

In the first part of the experiment, a new CCS Project is created as choosing "MSP430G2553" as the target and "Empty Assembly Project" as the project template. Then, the codes are examined given on the experiment booklet [5].
With the following two instructions(see Figure 2.1) P1.2 is set as a switch and will branch according to status of the button.

```
1 ;read the switch at P1.2 and set flags
2 bit.b   #00000100b, &P1IN
3 jnz     ON
4
```

Figure 1: Assembly Code of Part 1.1

Then, with the instructions on the Figure 2.1, LED 5 is cleared because we random value isn't wanted on it and then it was set.

```
;read the switch at P1.2 and set flags
bic.b   #00010000b,&P1OUT; clear P1.4
bis.b   #00010000b,&P1OUT; set P1.4

```

Figure 2: Assembly Code of Part 1.2

2.2 Part 2

In the second part of the experiment, the microprocessor was programmed to flash one of the third and fourth LEDs on the second column and change which is lit by the push of a button, i.e. P1.5.

In order to implement the given task, team created following piece of code given in Figure 2.2

```
SetupLED        mov.b   #000Ch,         &P2DIR
                bic.b   #0020h,         &P1DIR
                mov.b   #00000100b,     &P2OUT
ButtonLoop      bit.b   #00100000b,     &P1IN
                jz      ButtonLoop
                xor.b   #001100b,       &P2OUT
PressLoop       bit.b   #00100000b,     &P1IN
                jnz     PressLoop
Wait            mov.w   #250000,        R15
L1              dec.w   R15
                jnz     L1
                jmp     ButtonLoop


```

Figure 3: Assembly Code of Part 2

For better understanding, further examination of the code, line by line if necessary, is required:

- Line 1: The second column of LEDs are set up. Notice that only third and fourth LEDs in the second column will be needed. Therefore $(0x0C)$ is moved to the absolute address $P2DIR$ in order to activate third and fourth LEDs in the second column.

- Line 2: P1.5 which will be used as the button is cleared (set to 0). It is done by clearing the bits (in this case, there is one bit) in $P1DIR$ corresponding to the ones in the binary number 00100000_2 ($0x20$).

- Line 3: Initially, either P2.2 or P2.3 should be lit because that should be the case in any given instant as mentioned in the experiment paper. Therefore 00000100_2 is moved to the absolute address $P2OUT$ to light P2.2.

- Line 4-5: These are the lines that program constantly iterates through until P1.5 is pressed. In line 4, bit test instruction does a bitwise AND operation between 00100000_2 and the value in $P1IN$ which indicates which buttons in the first column are pressed down at that moment. In order for this operation to yield non-zero, the sixth least significant bit of the value in $P1IN$ must be 1 which is when P1.5 is pressed down. Therefore, unless P1.5 is pressed down, zero flag will be raised and in line 5, program will jump to line 4 again. If P1.5 is pressed down, thus, zero flag is not raised, program will continue from line 6.

- Line 6: Now that P1.5 is pressed, P2.2 and P2.3 should be changed. In line 6, an XOR operation is applied to revert the state of third and fourth LEDs. Note that an XOR operation with 1 results in the reverse of the other operand, namely, $1 \oplus 0 = 1$ and $1 \oplus 1 = 0$.

- Line 7-8: The very same way with lines 4 and 5, this time the program will iterate through these 2 lines till P1.5 is released because this time if P1.5 is at down position, namely, zero flag is raised, it jumps to line 7.

- Line 9-11: This is the Wait function that will be executed in order for the button to work properly and as expected. In line 9, the value 250000_{10} is moved to register 15 and in line 10, it is decreased. In line 11, until the zero flag is raised the program jumps back to 10^th line to decrease the value in R15. When the value in R15 hits 0_{10}, the zero flag is raised and the program skips to the next line.

- Line 12: After Wait function, it jumps back to ButtonLoop to wait for the next press of P1.5.

2.3 Part 3

In the part 3 of the second experiment an Assembly program that counts how many times the push button P2.1 is pressed is written.

In order to achieve the given task the team decided on the following piece of code given in Figure 4.

```
1  SetupLED        mov.b   #00FFh,         &P1DIR
2                  bic.b   #0002h,         &P2DIR
3                  mov.b   #00000000b,     &P1OUT
4  ButtonLoop      bit.b   #00000010b,     &P2IN
5                  jz      ButtonLoop
6                  inc.b   Counter
7                  mov.b   Counter,        &P1OUT
8  PressLoop       bit.b   #00000010b,     &P2IN
9                  jnz     PressLoop
10 Wait            mov.w   #250000,        R15
11 L1              dec.w   R15
12                 jnz     L1
13                 jmp     ButtonLoop
14
15                 .data
16 Counter         .word   0
```

Figure 4: Assembly Code of Part 3

For better understanding, further examination the code, line by line if necessary, is required:

- Line 1: The first column of LEDs are set up. Notice that all LEDs in the first column will be needed in order to show as great number as possible because that number will represent how many times P2.1 is pressed. $(0xFF)$ is moved to the absolute address $P1DIR$ in order to activate all LEDs in the first column.
- Line 2: P2.1 which will be used as the button is cleared (set to 0). It is done by clearing the bits (in this case, there is one bit) in $P2DIR$ corresponding to the ones in the binary number 00000010_2 $(0x02)$.
- Line 3: $(0x00)$ is moved to the absolute address $P1OUT$ in order to reset all LEDs in first column to unlit state.

- Line 4-5: These are the lines that program constantly iterates through until P2.1 is pressed. In line 4, bit test instruction does a bitwise AND operation between 00000010_2 and the value in $P2IN$ which indicates which buttons in the second column are pressed down at that moment. In order for this operation to yield non-zero, the second least significant bit of the value in $P2IN$ must be 1 which is when P2.1 is pressed down. Therefore, unless P2.1 is pressed down, zero flag will be raised and in line 5, program will jump to line 4 again. If P2.1 is pressed down, thus, zero flag is not raised, program will continue from line 6.

- Line 6-7: Now that P2.1 is pressed, counter should be incremented. In line 6, the variable Counter which is defined in line 16 is incremented and in line 7, this new value is moved to $P1OUT$ so that it will show up in the first column of LEDs.

- Line 8-9: The very same way with lines 4 and 5, this time the program will iterate through these 2 lines till P2.1 is released because this time if P2.1 is at down position, namely, zero flag is raised, it jumps to line 8.

- Line 10-12: This is the Wait function that will be executed in order for the button to work properly and as expected. In line 10, the value 250000_{10} is moved to register 15 and in line 11, it is decreased. In line 12, until the zero flag is raised the program jumps back to 11^th line to decrease the value in R15. When the value in R15 hits 0_{10}, the zero flag is raised and the program skips to the next line.

- Line 13: After Wait function, it jumps back to ButtonLoop to wait for the next press of P2.1.

- Line 15: This line indicates that the data segment has started.

- Line 16: This line defines a variable that has the name Counter and the size of a word.

2.4 Part 4

In the final part of the experiment, one's complement of the counter value had to be taken by pushing a button. With the extra time, a bonus functionality, resetting the counter has been added with the same technique used in Part 3 of the experiment. The whole assembly code can be seen in Figure 5.

To provide these functionalities several lines required to be altered and a few more had to be added to the code given in Figure 4. For better understanding; modified and new lines in Figure 5 can be explained such:

- Line 2: In order to provide 2: 3^{rd} and 4^{th} buttons are enabled.
- Line 6-7: Checking if 3^{rd} button is pressed or not by using the same method described in Part 3. If pressed jump to *Complement* label.
- Line 8-9: Checking if 4^{th} button is pressed or not. If pressed jump to *Reset* label.
- Line 11-13: Setting all bits to 0 in both $P1OUT$ and $Counter$ which completes the Reset functionality.
- Line 14-16: In these lines complement functionality of the program is implemented. An XOR operation is applied to the $Counter$ by the value $0xFF$ which inverts all the bits. This accomplishes One's complement operation. New value of $Counter$ is moved to $P1OUT$ in or to be observed from the lights in $P1$ column.
- Line 21-24: Same technique is used to prevent repeated calls of *Complement* and *Reset*.

```
1  SetupLED         mov.b   #00FFh,          &P1DIR
2                   bic.b   #00001110b,      &P2DIR
3                   mov.b   #00000000b,      &P1OUT
4  ButtonLoop       bit.b   #00000010b,      &P2IN
5                   jnz     Increment
6                   bit.b   #00000100b,      &P2IN
7                   jnz     Complement
8                   bit.b   #00001000b,      &P2IN
9                   jnz     Reset
10                  jz      ButtonLoop
11 Reset            mov.b   #0000h,          &P1OUT
12                  mov.b   #0000h,          Counter
13                  jmp     PressLoop
14 Complement       xor.b   #0FFh,           Counter
15                  mov.b   Counter,         &P1OUT
16                  jmp     PressLoop
17 Increment        inc.b   Counter
18                  mov.b   Counter,         &P1OUT
19 PressLoop        bit.b   #00000100b,      &P2IN
20                  jnz     PressLoop
21                  bit.b   #00000010b,      &P2IN
22                  jnz     PressLoop
23                  bit.b   #00001000b,      &P2IN
24                  jnz     PressLoop
25 Wait             mov.w   #250000,         R15
26 L1               dec.w   R15
27                  jnz     L1
28                  jmp     ButtonLoop
29
30                          .data
31 Counter          .word   0
32
```

Figure 5: Assembly Code of Part 4

3 RESULTS

In the second part of the experiment, when the button that is connected to P1.5 pushed down, only one of the the third and fourth LEDS on the second column was ON and other one was OFF and pushing to the button results a change in the state of the LEDs.

In the third part of the experiment, when the button that is connected to P2.1 is pushed, counter that is declared on the memory is increased by 1. The LEDs on the first column is lit according to the counter's binary value. Each push of the button changes status of LEDs on the first columns.

In the fourth part of the experiment, microcontroller is programmed in a way where a new button that will do 1'st complement operation in addition to incrementing a value on the memory with the push button. Our team has easily finished desired tasks and added extra functionality to the microcontroller. With the extra functionality added, LEDs can be reset when the extra button is pushed.

4 DISCUSSION

Our team coded all the tasks using interrupts before coming to the lab session. In the lab session, we tried to make the code run. But when the team received help from the instructer, we have realized that the desired way of doing the task is much simpler. There actually is no need to use interrupts. The code is already doing it on the background. The code on the booklet were reexamined by the team and we finished the experiment quickly afterwards with a little brainstorming.

5 CONCLUSION

Our team learned that it is necessary to think in a more simplistic manner while performing the desired tasks. We also saw the importance of having a good understanding of the task we are required to do and how to put it into code. We learned how to make a simple gpio application using Msp430 microcontroller and how to use this variable in order to create a variable in memory.

REFERENCES

[1] Texas Instruments. Msp430 education board document. 2009.

[2] Texas Instruments. Msp430 architecture. 2009.

[3] Texas Instruments. Msp430 architecture. December 2004 - Revised July 2013.

[4] Texas Instruments. Supplementary chapter 6 general purpose io. December 2004 - Revised July 2013.

[5] Istanbul Technical University Department of Computer Engineering. Blg 351e wednesday experiment 1: Basic assembly coding, Spring 2019.

[6] Overleaf documentation https://tr.overleaf.com/learn.

[7] Detailed info on writing reports https://projects.ncsu.edu/labwrite/res/res-studntintro-labparts.html.