Experiment 5: 7-Segment Display and Interrupt Subroutine

(09-13).12.2024
Res. Asst. Fırat Öncel
oncelf@itu.edu.tr

1 Introduction

This experiments aims to enhance the practical experience about driving 7-segment displays and initializing interrupts. Students are recommended both to read the supplementary material MSP430 User Guide - Chapter 8 on Ninova.

A 7-segment display contains 7+1 (including the dot at the lower corner) distinct LEDs which are controlled by the distinct inputs. In this experiment, you are requested to use these displays to show decimal integers. Thus, the first objective you need to do is to decide which inputs have to be set high or low in order to represent the integers between 0-9. For example, in order to show 0 on the 7-segment display, LEDs A, B, C, D, E and F have to be lit. This means that the related bits of the GPIO ports have to be set high. Before, the experiment, please fill the table 1 for the other decimal integers.

GPIO Port 1 and Port 2 are physically connected to the 4-digit 7-segment display as in Figure 2.

Integer	**H**	**G**	**F**	**E**	**D**	**C**	**B**	**A**
0	0	0	1	1	1	1	1	1
1	0							
2	0							
3	0							
4	0							
5	0							
6	0							
7	0							
8	0							
9	0							

Table 1: Some inputs for 7-segment display

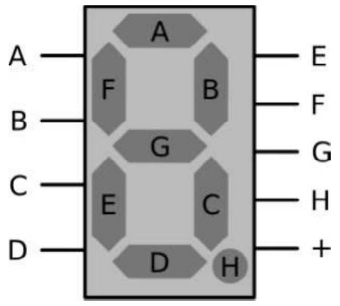

Figure 1: 7-segment display

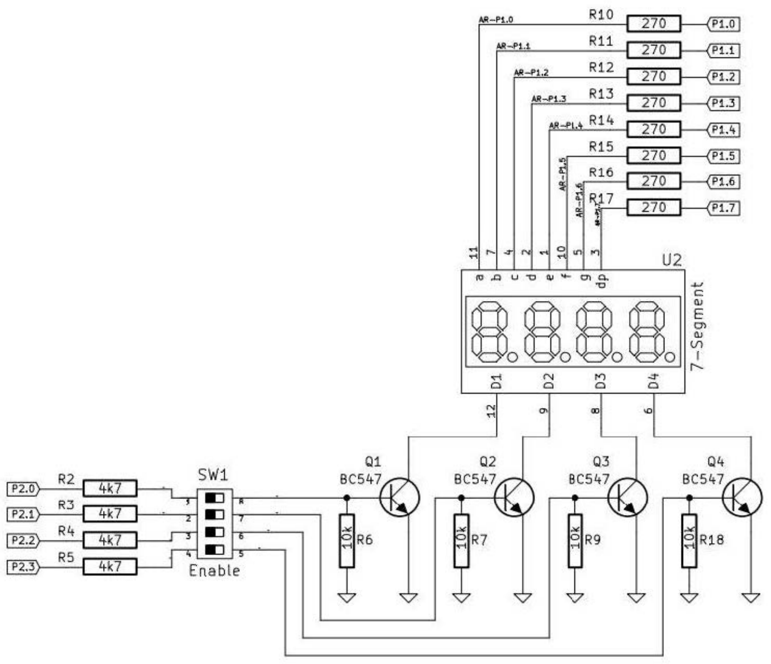

Figure 2: Port connections of 7-segment displays

2 Part 1

In the first part of the experiment, you are requested to write a main program which counts from 0 to 9 in ten seconds and shows the value of the counter at one of the digits of the 7-segment display. Details of this program are explained below. Firstly, you should to initialize GPIO Port 1 and Port 2 to activate and use 7-segment display. Then, you should write a loop which increments a counter you defined and waits one-second at each iteration. Notice that, the counter has to be reset when it reaches the decimal number 10 (0Ah). You could use the following Delay function to implement one-second delay.

```
Delay   mov.w #0Ah,R14          ; Delay to R14
L2      mov.w #07A00h, R15
L1      dec.w R15               ; Decrement R15
        jnz L1
        dec.w R14
        jnz L2
        ret
```

Moreover, you could call this function in your code as follows

```
    ...
    call #Delay
    ...
```

Lastly, at each iteration, you should convert the value of the counter (integer) to values that you determine in the Table 1 in order to show meaningful integers on 7-segment display. One of the simple ways of this conversion is defining an array to hold the values of integers for 7-segment display. So, the number you would like to convert could be used as the index of the array. In CCS editor, an array could be defined as follows.

```
;Integer to 7-segment array
array .byte 00111111b, ... , 00111111b    ; contains 10 values
lastElement
```

3 Part 2

In this part, you are requested to write an interrupt subroutine and enhance your main program so that your program is able to count even numbers (e.g., 0, 2, ... , 8) or odd numbers (e.g., 1, 3, ... , 9) by an external interrupt. You can define a Boolean variable in your program which represents the mode of the counting. So, in your main loop, you can check the value of this variable and decide to count in the even or odd mode. And, you can simply toggle the the mode of the counting using the interrupt subroutine. After, changing your main program, you should implement necessary part to utilize interrupts. In the following steps, use of interrupts is explained in detail. First, you should activate a mask-able interrupt in one of the GPIO Port Pins. The following code is given as an example to initialize the interrupt at the 6th bit of Port 2.

```
init_INT  bis.b #040h, &P2IE          ; enable interrupt at P2.6
          and.b #0BFh, &P2SEL         ; set 0 P2SEL.6
          and.b #0BFh, &P2SEL2        ; set 0 P2SEL2.6

          bis.b #040h, &P2IES         ; high-to-low interrupt mode
          clr   &P2IFG                ; clear the flag
          eint                        ; enable interrupts
```

Then, you need to store the address of your interrupt subroutine to the interrupt vector of the GPIO Port 2. So that, whenever an interrupt comes from the Port 2, your interrupt subroutine begins running. To do this, you should implement the following code at the end of your main program.

```
;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------

            .sect ".reset"          ; MSP430 RESET Vector
            .short RESET
            .sect ".int03"          ; Port Interrupt Vector
            .short ISR
```

Lastly, you should write your interrupt subroutine which is pointed in the interrupt vector of the GPIO Port 2. The following code part is given as a template for your interrupt subroutine. You need the fill the body of the subroutine in order to accomplish the requested behavior.

```
ISR         dint            ; disable interrupts

            ; YOUR CODE goes here

            eint            ; enable interrupts
            reti            ; return from ISR
```