10.05.2021

16:00 – 20:00

Urszula Chmielewska

300167

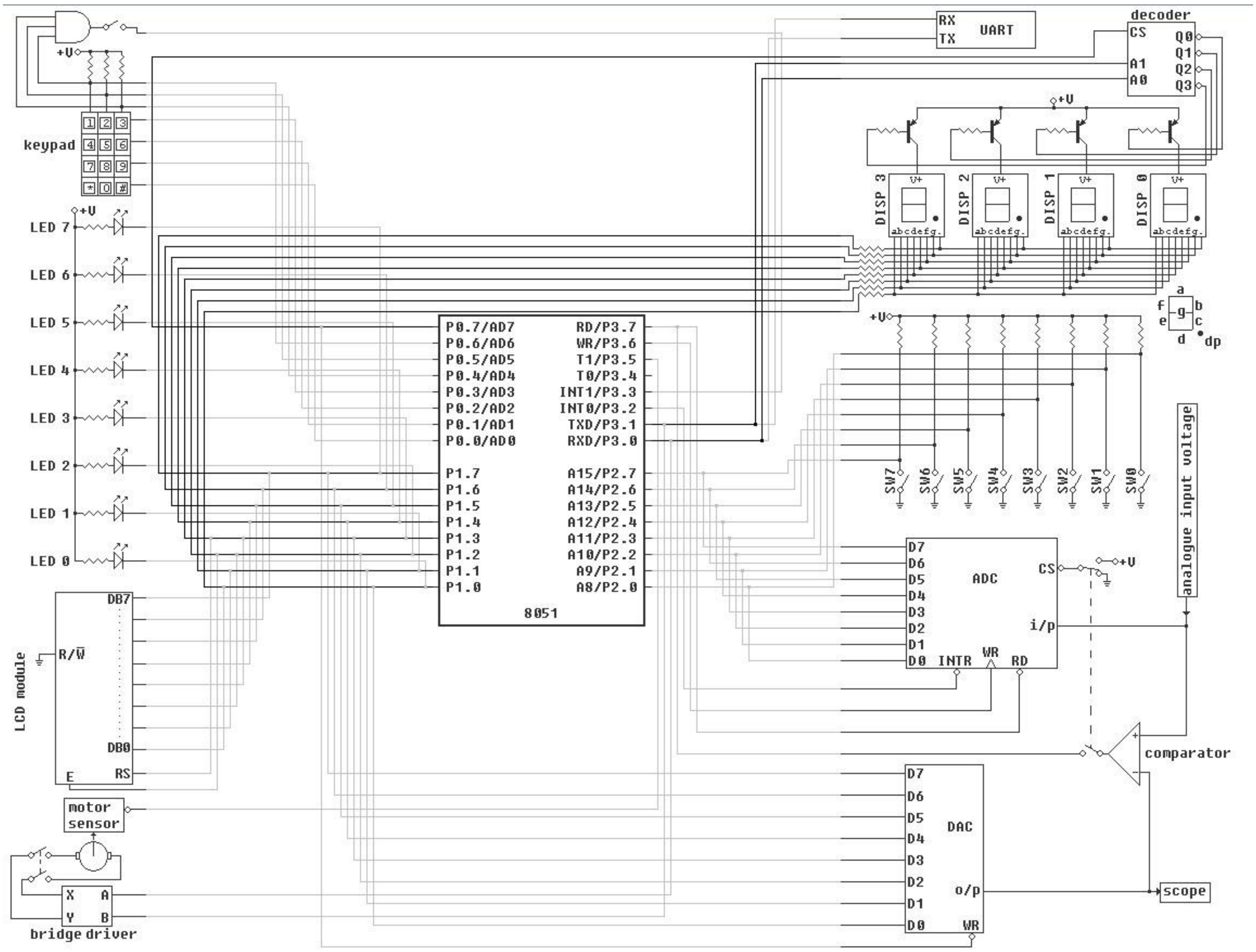
EMISY

Lab 1

INTRODUCTION

The goal of this laboratory was to familiarize with handling seven segment LED displays in multiplexing mode. In the first part of the task it was necessary to configure 8051 Timer0 peripheral in correct mode and settings and use it to generate time intervals. In the second part of the task it was necessary to use the Timer0 interrupt in order to control LED displays.

SCHEMATIC



Hardware – general approach

The schematic presented above is the same for all of the tasks. Chip Select pin responsible for enabling the decoder is connected to the pin P0.7. Address pins A0 and A1 are connected to pins P3.0 and P3.1 respectively. Such connection of address pins will be useful in tasks solutions what will be explained later. Finally, LED display's cathodes are connected to P1 GPIO port. To enable the decoder, Chip Select pin has to be set to "1". In order to turn on a single LED display segment it is necessary to set corresponding pin to "0".

TASK 1

Assembly code

```
1  DEC_CS     EQU    P0.7          ;pin for Display-select Decoder CS
2  A0A1_BUS   EQU    P3            ;pin for Display-select Input 0 and Input 1
3  SEG_BUS    EQU    P1            ;segment bus
4
5  clr     DEC_CS                              ;clear Chip Select
6  mov     SEG_BUS, #11111111b              ;set to 1 all segment pins (then nothing is on the display)
7
8  mov     R0, #3                              ;display digit on the first place from the left
9  mov     R1, #10100100b                   ;load binary number which will display number "2" on the display
10
11 lcall   display
12
13 jmp     $
14
15         display:
16                 mov     A0A1_BUS, R0             ;set A0 and A1 to 1 (it will display on the first from the left position)
17
18                 mov     SEG_BUS, R1               ;set segment busses in a way to display "2"
19
20                 setb    DEC_CS                    ;set Chip Select to 1 to enable decoder
```

Code description

At the very beginning I have assigned names to proper GPIO pins for an easier data handling. First of all I clear Chip Select pin, to make sure that the decoder is not enabled yet. Then I set to P1 GPIO port value 11111111b. After that all segment pins are set to “1” so they are disabled (there is nothing on the display when all segments are set to “1”). In R0 register I store value “3” which indicates the display, first from the left. Due to this fact the number will be displayed on this particular display. In R1 register I store combination of 1’s and 0’s which represent number to be displayed. In my case it will be number “2” visible on the display. Then I call subroutine “display” which particularly loads proper values, previously loaded to registers, to address bus and segment bus. Due to the fact that address pins are connected to the least significant pins, after writing “3” to the bus, both A0 and A1 are set to 1. Finally I set bit to Chip Select what results with showing a digit on the display.

It is easy to modify the position of the digit and the displayed value. It is enough to modify values which are loader to R0 and R1 in lines 8 and 9.

TASK 2

Assembly code

```
1   DEC_CS      EQU     P0.7                ;pin for Display-select Decoder CS
2   A0A1_BUS    EQU     P3                  ;pin for Display-select Input 0 and Input 1
3   SEG_BUS     EQU     P1                  ;segment bus
4
5
6           jmp     skip_handler                    ;skip interrupt handler
7
8           ORG     0Bh                             ;timer overflow interrupt handler
9
10          mov     A0A1_BUS, #3                    ;display digit on the first place from the left
11
12          mov     A, SEG_BUS                      ;load register A with segment bits
13          xrl     A, #4h                          ;xor operation with segment bits vs #00000100
14          mov     SEG_BUS, A                      ;only one segment bit will be changed (to 1 and 0 alternatively)
15
16          setb    DEC_CS                          ;enable decoder
17
18          mov     TH0, #0B1h                      ;reset timer - upper half
19          mov     TL0, #0DFh                      ;reset timer - lower half
20          reti                                    ;return
21
22
23          skip_handler:
24                  setb    TR0                     ;turn T0 by setting it to 1
25                  mov     TMOD, #01h              ;set Timer0 to mode 1
26                  setb    ET0                     ;ET0 set to 1 enables Timer0 Interrupt Bit
27                  setb    EA                      ;turn on global interrupt
28
29                  mov     TH0, #0B1h              ;setting timer interval to 20ms - upper half
30                  mov     TL0, #0DFh              ;setting timer interval to 20ms - lower half
31
32                  jmp     $
```

Code description

At the beginning code jumps to the subroutine which initializes the timer. I set TR0 to "1" to turn on the T0 timer. Then I load '1h' to TMOD register, which sets Timer 0 to work in mode 1. Then I set to "1" ET0 to enable overflow interrupt and EA to enable global interrupt.

Secondly, it was necessary to implement time interval equal to 20ms. It was done by loading register TH0 (upper half) with 'B1h' and register TL0 (lower half) with 'DFh'. Such values were chosen from the calculation FFFFh – 20000d what resulted with value 'B1DFh'. Timer works in a way that it increments its value in each machine cycle and sends interrupt when overflow happens. Due to this fact, in such solution, time intervals are equal to 20ms.

Finally, the program reaches the state where timer is working and program executes in a loop until interrupt happens. After that, we jump to the line number 8 where program jumps to 'Bh' in memory to execute handler's code. Then I coded display section which is similar to the one in the task 1. However this time, due to the fact that display segment should be blinking, I used logical xor function. It works in a way that only one bit will be changing in each loop. All other bits are set to "1" for the whole time, so they are disabled. After all, program restores timer and runs again in the infinite loop.

TASK 3

Assembly code

```
DEC_CS      EQU     P0.7                ;pin for Display-select Decoder CS
A0A1_BUS    EQU     P3                  ;pin for Display-select Input 0 and Input 1
SEG_BUS     EQU     P1                  ;segment bus

;4 last digits in my index numebr: 0167


        mov     R0, #000000001b         ;set first from the left display at the beginning
        setb    DEC_CS                          ;enable decoder

        jmp     skip_handler                    ;skip interrupt handler


        ORG     0Bh                             ;timer overflow interrupt handler

        mov     ACC, R0                 ;load current R0 value to accumulator
        jb      ACC.0, display0         ;check 0'th bit - first from the left display
        jb      ACC.1, display1         ;check 1'st bit - second from the left display
        jb      ACC.2, display2         ;check 2'nd bit - third from the left display
        jb      ACC.3, display3         ;check 3'rd bit - fourth from the left display


        skip_handler:
                setb    TR0                     ;turn T0 by setting it to 1
                mov     TMOD, #01h              ;set Timer0 to mode 1
                setb    ET0                     ;ET0 set to 1 enables Timer0 Interrupt Bit
                setb    EA                      ;turn on global interrupt

                mov     TH0, #0B1h              ;setting timer interval to 20ms - upper half
                mov     TL0, #0DFh              ;setting timer interval to 20ms - lower half
                jmp     $



        display0:
                mov     R0, #00000010b                  ;load value which denoted display1

                mov     A0A1_BUS, #3                    ;first from the left display
                mov     SEG_BUS, #11000000b             ;display "0"

                mov     TH0, #0B1h                      ;reset timer - upper half
                mov     TL0, #0DFh                      ;reset timer - lower half

                reti

        display1:
```

part 2 below

```
46          display1:
47                  mov     R0, #00000100b              ;load value which denoted display2
48
49                  mov     A0A1_BUS, #2                ;second from the left display
50                  mov     SEG_BUS, #11111001b         ;display "1"
51
52                  mov     TH0, #0B1h                  ;reset timer - upper half
53                  mov     TL0, #0DFh                  ;reset timer - lower half
54
55                  reti
56
57          display2:
58                  mov     R0, #00001000b              ;load value which denoted display3
59
60                  mov     A0A1_BUS, #1                ;third from the left display
61                  mov     SEG_BUS, #10000010b         ;display "6"
62
63                  mov     TH0, #0B1h                  ;reset timer - upper half
64                  mov     TL0, #0DFh                  ;reset timer - lower half
65
66                  reti
67
68          display3:
69                  mov     R0, #000000001b         ;load value which denoted display0
70
71                  mov     A0A1_BUS, #0                ;fourth from the left display
72                  mov     SEG_BUS, #111111000b        ;display "7"
73
74                  mov     TH0, #0B1h                  ;reset timer - upper half
75                  mov     TL0, #0DFh                  ;reset timer - lower half
76
77                  reti
```

Code description

This task was an extended version of tasks 1 and 2. First of all I set the display on which I want to have first digit displayed. It will be the first display from the left. Then I enable the decoder and jump to the infinite 'skip_handler' subroutine which is exactly the same as in the task 2. When the timer reaches overflow, program moves to the line 14 and then loads to the accumulator value currently stored in register R0. This register is responsible for displaying digits on proper displays. Then I check bits 0, 1, 2 and 3. Program jumps to the subroutines in order: display0, display1, display2 display3 and displays digits 0 1 6 7.

FINAL QUESTIONS

1. **Describe how is a display selected in EDSIM simulator?**

Decoder chip is responsible for selecting display. Each of 4 low outputs enables the current to the one of the displays.

2. **What is the difference between common anode and common cathode LED displays in terms of interfacing them with MCU?**

Displays with common anodes require clearing bits corresponding to the segments cathodes to turn them on, while LED displays with common cathode require voltage to be provided to the corresponding segment pin.

3. **How can be a timer peripheral used to generate precise time intervals (precise delays)?**

In case when 12MHz clock is used it is enough to load to the timer’s registers value FFFFh – <delay in milliseconds>. Timer increments value stored in this registers in each machine cycle.

4. **How does multiplexing driving mode of LED displays work? Compare it with static driving mode of LED displays in terms of required GPIO pins and current consumption. When does it make sense to use multiplexing and when does it not?**

Multiplexing driving mode of LED displays works in a way that LED displays are switching between each other as fast as possible that it is almost impossible to be noticed by human eye. Static driving mode od LED displays requires more pins and more current since it has a higher number of enabled LEDs. Due to the fact that breaks between LED displays switching are visible on cameras, it is useful to use static mode of LED displays while such display will be observed through a camera.

I declare that this piece of work which is the basis for recognition of achieving learning outcomes in the EMISY course was completed on my own.

Urszula Chmielewska

300167

10.05.2021