17.05.2021 16:30

EMISY LAB 2

Krzysztof Piotrowski

Index. No.: 300175

Task 1.

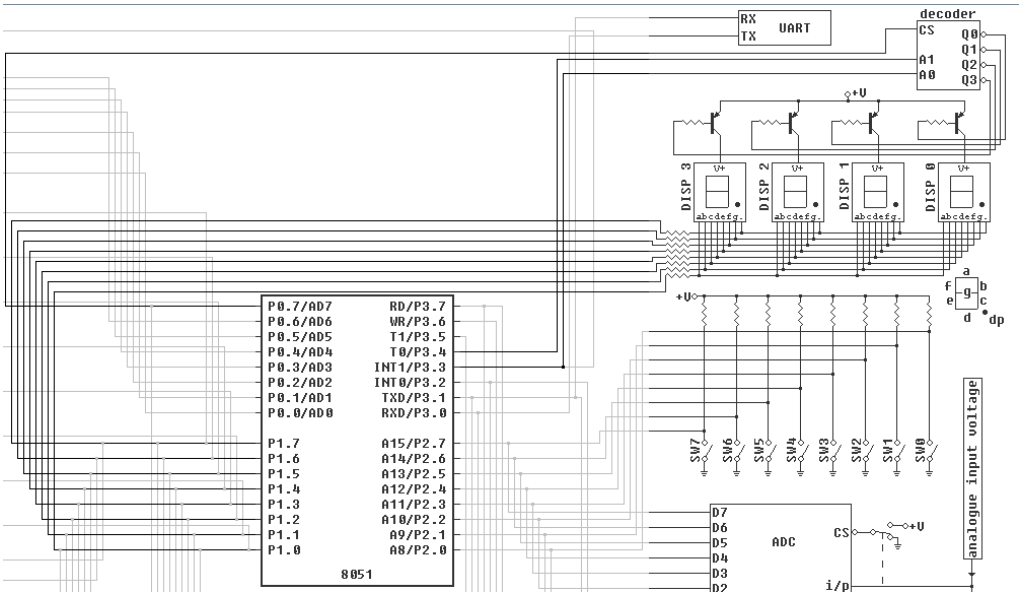
- **Code for Task 1**

```
DISPLAY_BUS equ P1   ; Label for all 7 segments of display
DECODE_CS equ P0.7  ; Label for Chip Select pin to enable decoding
DECODE_ADR equ P3  ; Specifically Pins: 3.3 and 3.4 for choosing which display will display
main:
        setb DISPLAY_BUS        ; clear display
        mov R0, #00001000B      ; fill segments in display #1
        mov R1, #10010000B      ; write '9' to the display
        call display            ; execute
        jmp $                   ; infinite loop
display:
        clr DECODE_CS           ; turn off decoder to choose display
        mov DECODE_ADR, R0  ; user choice display
        mov DISPLAY_BUS, R1   ; user choice pattern
        setb DECODE_CS          ; turn on decoder
        ret                     ; return to main
```

- **Code Description**

  At the beginning I assigned labels for each used pins in this task for good readability of the code. In main part, first I clear display by setting each pin logical value to 1. (As we have common anode configuration). Next, in register R0, the value of 3rd and 4th bits are adjusted, as P3.3 and P3.4 are used in decorder, in order to choose which display (from 0 to 3) we want to use in given moment (I have chosen display number 1). In R1, according to which display segments we want to turn on, bits are set to the value 0. I have chosen number 9 so only 2 segments are not turned on: ‘e’ and ‘dp’. During display call process decoder is turned off to apply changes and logical values of proper pins are set with use of labels. After that we turn on decoder and go to infinite loop to keep digits on the display.

- **Schematic of display connection**



Task 2.

- **Code for Task 2**

```
DISPLAY_BUS equ P1  ; Label for all 7 segments of display
DECODE_CS equ P0.7 ; Label for Chip Select pin to enable decoding
DECODE_ADR equ P3 ; Specifically Pins: 3.3 and 3.4 for choosing which display will display
main:
        setb DISPLAY_BUS ; clear display

        jmp initialize                  ; go to initialization process of timer
        org 00BH                        ; interrupt address
        clr TF0                         ; clear overflow flag

        clr DECODE_CS                   ; turn off decoder to choose display
        mov DECODE_ADR, #00011000B  ; use 3rd display
        xrl DISPLAY_BUS, #10000000B     ; xor display bus to make decimal point blinking
        setb DECODE_CS                  ; turn on decoder

        mov TH0, #11011000B             ;restore timer
        mov TL0, #11101111B
        reti

initialize:
        setb TR0                ; turn on T0
        mov TMOD, #00000001B ; set time to work in mode 1, gate = 0
        setb ET0                ; set overflow interrupt of timer T0
        setb EA                 ; enable global interrupt

        mov TH0, #11011000B     ; load timer with value 65535 - 10000
        mov TL0, #11101111B     ; splitted into to registers TH0 and TL0
        jmp $                   ; infinite loop till interrupt
```

- **Code Description**

  As in previous task, labels were again assigned for better readability of code and display was cleared. Next, we jump to the initialization process. Here we set up counter to work in first mode: as 16 bit counter. Only lower nibble of TMOD register is responsible for Timer 0. Zero and first bit are responsible for mode (that is why lowest bit is set to 1) and third bit is responsible for gate (It is zero because we need to set gate to zero). Lastly we set up EA and ET0 which are responsible respectively for enabling global interrupt and enabling overflow interrupt of used timer. Also, we need to set TH0 and TL0 to value equal 65535 – 10000 in order to get desired delay as counter counts to maximum value of 16 bit number which is equal 65535. Counter is incremented by 1 in one machine cycle which last 1 μs so in total we will wait about 10 ms. While counter is increasing its value we are stuck in infinite loop until interrupt flag will be set by reaching maximum value of the counter. It moves to the place of interrupt address where the code is defined for blinking decimal point on 3rd display. To make decimal point blink we use xor to change value of MSB after each execution of code. At the end timer is restored and with use of reti we come back to infinite loop. Whole process repeats.

Task 3.

- **Code for Task 3**

```
DISPLAY_BUS equ P1 ; Label for all 7 segments of display
DECODE_CS equ P0.7 ; Label for Chip Select pin to enable decoding
DECODE_ADR equ P3 ; Specifically Pins: 3.3 and 3.4 for choosing which display will display

main:
        setb DISPLAY_BUS   ; clear display
        mov R2, #00000001B ; set value of R2 for Display 0
        jmp initialize ; go to initialize process of timer

        org 0BH  ; interrupt address
        clr TF0  ; clear overflow flag

        mov ACC, R2 ; load R2 to ACC in order to know on which display we operate
        jb ACC.0, DISPLAY0 ; go to display 0
        jb ACC.1, DISPLAY1 ; go to display 0
        jb ACC.2, DISPLAY2 ; go to display 0
        jb ACC.3, DISPLAY3 ; go to display 0
        reti

display0:
        mov R0, #00000000B ; set to fill segments in display  0
        mov R1, #10010010B ; turn on segments that are responsible for '5'
        mov R2, #00000010B ; set value of R2 to move to Display 1 label
        call display

        mov TH0, #11011000B ; load timer with value 65535 - 10000
        mov TL0, #11101111B ; splitted into to registers TH0 and TL0
        reti
```

```
display1:
        mov R0, #00001000B ; set to fill segments in display  1
        mov R1, #11111000B ; turn on segments that are responsible for '7'
        mov R2, #00000100B ; set value of R2 to move to Display 2 label
        call display

        mov TH0, #11011000B ; load timer with value 65535 - 10000
        mov TL0, #11101111B ; splitted into to registers TH0 and TL0
        reti

display2:
        mov R0, #00010000B ; set to fill segments in display  2
        mov R1, #11111001B ; turn on segments that are responsible for '1'
        mov R2, #00001000B ; set value of R2 to move to Display 3 label
        call display

        mov TH0, #11011000B ; load timer with value 65535 - 10000
        mov TL0, #11101111B ; splitted into to registers TH0 and TL0
        reti


display3:
        mov R0, #00011000B ; set to fill segments in display  3
        mov R1, #11000000B ; turn on segments that are responsible for '0'
        mov R2, #00000001B ; set value of R2 to move to Display 0 label
        call display

        mov TH0, #11011000B ; load timer with value 65535 - 10000
        mov TL0, #11101111B ; splitted into to registers TH0 and TL0
        reti

initialize:
        setb TR0 ; turn on T0
        mov TMOD, #00000001B ; set time to work in mode 1, gate = 0
        setb ET0 ; set overflow interrupt
        setb EA ; enable global interrupt

        mov TH0, #11011000B ; load timer with value 65535 - 10000
        mov TL0, #11101111B ; splitted into to registers TH0 and TL0

        jmp $ ; infinite loop

display:
        clr DECODE_CS      ; turn off decoder to choose display
        mov DECODE_ADR, R0  ; user choice display
        mov DISPLAY_BUS, R1 ; user choice pattern
        setb DECODE_CS     ; turn on decoder

        ret     ; return to main
```

- **Code Description**

  In task 3 there is basically connection of task 1 and task 2. Displays have to show 4 last digits of my index number (0175) with multiplexing animation. Program include initialization part of timer to wait 10 ms between each display, display routine for setting proper display by decoder and pass information which segments should be turned on and lastly displayX routine for creating the order in which each display works. The order is set with help of R2 and ACC registers and jb condition. We start from display 0 and end with display 3.

Final Questions

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

   With the help of decoder and its two pins. Depending on their state, one of the output Q0 - Q3 is set to 0 and rest is set to 1 (in common anode config.). With their use decoder tied to logic low level one of the transistors which is connected to certain display. The one is tied, the one is working at that time.

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

   In common anode to turn on display segments we have to set logical value ‘0’ on GPIO pins. In common cathode we need to provide voltage to GPIO pins to turn on displays.

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

   Timer registers can load up to maximal value of 16 bit number (65535). Counter increment by one in each machine cycle and after reaching maximum value the flag is set. By substracting desired delay in microseconds from maximal value and loading it to registers it will stop counting exactly after time we substracted from 65535.

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?**

   In multiplexing driving mode microcontroller turns on display one after another with very small delay which makes for human eye effect of static image. In Static driving mode we would have to connect every display to separate pins and provide them constant power which is not good from economic and ergonomic point of view. The only drawback of multiplexing system is that it will not work for devices that have high refresh rate as they will clearly see changing displays.

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.

Krzysztof Piotrowski

300175

18.05.2021