ECE 375 LAB 7

Timer/Counter

Lab Time: Friday 16:00 ~ 17:50

Hyunjae Kim

INTRODUCTION

In this lab, timer/counter interruption will be used for controlling action of TekBot. Especially, Fast PWM mode with inverting mode will be used for adjusting speed of TekBot. TekBot will change its speed depending on the external interrupts.

PROGRAM OVERVIEW

The program keeps moving forward, and change its speed depending on the subroutine. To be specific, there are different 15 speed levels. The first interruption will increase the speed level by one, and the second interruption will decrease the speed level by one. The third interruption is reaching to the maximum speed and maintain that speed until there is another interruption. The last interruption is reaching to the minimum speed and maintain that speed until there is another interruption.

INITIALIZATION ROUTINE

MAIN ROUTINE

The main routine keeps looping and does not do specific functions.

SUBROUTINES

1. PLV1 Routine

   When there's an external interrupt from Port D(INT3), the program increments one level, and adds 17 to OC0 and OC2 in order to increase the speed.

2. MLV1 Routine

   When there's an external interrupt from Port D(INT2), the program decrements one level, and subtracts 17 to OC0 and OC2 in order to decrease the speed.

3. G2Max Routine

   When there's an external interrupt from Port D(INT 1), the program goes to max level(LV.15) and maximum speed(255).

4. G2Min Routine

   When there's an external interrupt from Port D(INT 0), the program goes to min level(LV.0) and minimum speed (0).

STUDY QUESTIONS

1. In this lab, you used the Fast PWM mode of both 8-bit Timer/Counters, which is only one of many possible ways to implement variable speed on a TekBot. Suppose instead that you used just one of the 8-bit Timer/Counters in Normal mode, and had it generate an interrupt for every overflow. In the overflow ISR, you manually toggled both Motor Enable pins of the TekBot, and wrote a new value into the Timer/Counter's register. (If you used the correct sequence of values, you would be manually performing PWM.) Give a detailed assessment (in 1-2 paragraphs) of the advantages and disadvantages of this new approach, in comparison to the PWM approach used in this lab.

The advantage of using Normal mode instead of Fast PWM mode is that the initialization of the program is much simpler than using Fast PWM mode. Furthermore, through using Normal mode, the programmer is only controlling one register, OC, to deal with pulse.

However, the disadvantage of using Normal mode instead of Fast PWM mode is that there are some waste of clock cycle and registers. To make a pulse with Normal mode, the program has to toggle every time to make a pulse, and needs to use more instruction whenever loading value to TCNT register.

2. The previous question outlined a way of using a single 8-bit Timer/Counter in Normal mode to implement variable speed. How would you accomplish the same task (variable TekBot speed) using one or both of the 8- bit Timer/Counters in CTC mode? Provide a rough-draft sketch of the Timer/Counter-related parts of your design, using either a flow chart or some pseudocode (but not actual assembly code).

By using CTC mode, the same task can be implemented with keep toggling to OC0 register and setting value of CTC mode.

<Pseudo Code>

```
.ORG

RCALL INCREASE

RETI

.ORG (ADDRESS OF INTERRUPT VECOTR)

RCALL      INTERRUPTREUSE

.ORG $0046

INITIALIZE:

        ;Initialize stack

        ;Initialize TCNT0

        SBI        DDRB, PB4

        LDI        A, $0F    ; CTC mode with pre-scale 1024
```

```
        OUT     TCCR0, A
        LDI     A, 156
        OUT     OCR0, A
        SEI
MAIN:
        RCALL TOGGLEANDRELOAD
        RJMP MAIN
INCREASE:
        IN      A, PORTB
        INC     A
        OUT     PORTB, A
        RET
TOGGLEANDRELOAD:
        IN      A, PORTB
        LDI     B,(1 <<PB4)
        EOR     A, B
        OUT     PORTB, A
        RET
INTERRUPTREUSE:
        LDI     A, $02
        OUT     TIFR, A
RETI
```

DIFFICULTIES

The counter did not count the level one by one. To solve this challenge, I delayed the printing the level on LED for a short period. By delaying the debouncing of the button, the challenge is resolved.

CONCLUSION

Through using Fast PWM mode, it is able to control the rpm of the motor, and other I/O devices. By using and understanding the concept of timer/counter feature of AVR board, it is able to adapt and control other I/O devices for future projects.

SOURCE CODE

```
;***********************************************************
;*
;*       This is the skeleton file for Lab 7 of ECE 375
;*
;*        Author: Hyunjae Kim
;*          Date: 02/25/2022
;*
;***********************************************************

.include "m128def.inc"                  ; Include definition file

;***********************************************************
;*       Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                               ; Multipurpose register
.def    LVcnt = r17
.def    mmpr    = r18

;Delay Register
.def    waitcnt = r20                           ; Wait Loop Counter
.def    ilcnt = r21                             ; Inner Loop Counter
.def    olcnt = r22                             ; Outer Loop Counter

.equ    WTime = 1

.equ    EngEnR = 4                              ; right Engine Enable Bit
.equ    EngEnL = 7                              ; left Engine Enable Bit
.equ    EngDirR = 5                             ; right Engine Direction Bit
.equ    EngDirL = 6                             ; left Engine Direction Bit

;***********************************************************
;*       Start of Code Segment
;***********************************************************
.cseg                                                   ; beginning of code segment

;***********************************************************
;*       Interrupt Vectors
;***********************************************************
.org    $0000
                rjmp    INIT                    ; reset interrupt
; The S1 switch on the board does not work properly, so I reversed the order of the interrupts.
.org    $0008
                rcall   PLV1
                reti
.org    $0006
                rcall   MLV1
                reti
.org    $0004
                rcall   G2Max
                reti
.org    $0002
                rcall   G2Min
                reti

                ; place instructions in interrupt vectors here, if needed

.org    $0046                                   ; end of interrupt vectors

;***********************************************************
;*       Program Initialization
;***********************************************************
INIT:
                ; Initialize the Stack Pointer
                ldi             mpr, LOW(RAMEND)
                out             SPL, mpr
                ldi             mpr, HIGH(RAMEND)
                out             SPH, mpr
                ; Configure I/O ports
```

```
                ldi             mpr, $ff
                out             DDRB, mpr
                ldi             mpr, $00
                out             PORTB, mpr

                ldi             mpr, $00
                out             DDRD, mpr
                ldi             mpr, $0f
                out             PORTD, mpr
                ; Configure External Interrupts, if needed
                ldi             mpr, $AA
                sts             EICRA, mpr
                ldi             mpr, $0f
                out             EIMSK, mpr
                ;out            EIFR, mpr

                ; Configure 8-bit Timer/Counters
                ldi             mpr, $79
                out             TCCR0, mpr
                out             TCCR2, mpr
                ldi             mpr, $00
                out             OCR0, mpr
                out             OCR2, mpr


                                                                ; no prescaling

                ; Set TekBot to Move Forward (1<<EngDirR|1<<EngDirL)
                ldi             mpr, (1 << 5 | 1 << 6)

                ; Set initial speed, display on Port B pins 3:0
                out             PORTB, mpr

                ldi             LVcnt, 0


                clr             mmpr
                ; Enable global interrupts (if any are used)
                sei
;***********************************************************
;*      Main Program
;***********************************************************
MAIN:

                rjmp    MAIN                    ; return to top of MAIN

;***********************************************************
;*      Functions and Subroutines
;***********************************************************

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
PLV1:
        ;Prevents 15 -> 0
        cpi             LVcnt, 15
        breq    Nothing

        ;Push to stack in order to return to the previous state before interruption
        push    mpr
        in              mpr, SREG
        push    mpr


        ;Adds 17 to OC0 & OC2 to increase the pulse
        ldi             mmpr, 17
        in              mpr, OCR0
        add             mmpr, mpr
        out             OCR0, mmpr
```

```
        out             OCR2, mmpr

        rcall   Pend ; Delay for increasing only one level
        rcall   Pend

        ;Print out Level to PORTB
        inc             LVcnt
        in              mpr, PORTB
        andi    mpr, $f0
        add             mpr, LVcnt
        out             PORTB, mpr


        ; Clear the interruption queue
        ldi             mpr, $0f
        out             EIFR, mpr

        pop             mpr
        out             SREG, mpr
        pop             mpr

        ret
Nothing:
        jmp             Exit
MLV1:
        ;Prevent 0 -> 15
        cpi             LVcnt, 0
        breq    Nothing

        ;Push to stack in order to return to the previous state before interruption
        push    mpr
        in              mpr, SREG
        push    mpr



        ;Subtract        17 from OC0 & OC2 to decrease the pulse
        ldi             mmpr, 17
        in              mpr, OCR0
        sub             mpr, mmpr
        out             OCR0, mpr
        out             OCR2, mpr

        rcall   Pend ; Delay for decreasing only one level
        rcall   Pend
        ;Print Level on PORTB
        dec             LVcnt
        in              mpr, PORTB
        andi    mpr, $f0
        add             mpr, LVcnt
        out             PORTB, mpr


        ;Clear the interruption queue
        ldi             mpr, $0f
        out             EIFR, mpr

        pop             mpr
        out             SREG, mpr
        pop             mpr

        ret

;Goes to Lv.15
G2Max:
        ldi             mpr, 255
        out             OCR0, mpr
        out             OCR2, mpr
        ldi             LVcnt, 15
        in              mpr, PORTB
        andi    mpr, $f0
```

```
        or              mpr, LVcnt
        out             PORTB, mpr
        jmp             Exit

;Goes to Level0
G2Min:
        ldi             mpr, 0
        out             OCR0, mpr
        out             OCR2, mpr
        clr             LVcnt
        in              mpr, PORTB
        andi    mpr, $f0
        or              mpr, LVcnt
        out             PORTB, mpr
        jmp             Exit

;To make a delay of adequately short time
;If it is extremely short, then there are multiple level jumps in the result.
Pend:
                push    waitcnt                 ; Save wait register
                push    ilcnt                   ; Save ilcnt register
                push    olcnt                   ; Save olcnt register

Loop:   ldi             olcnt, 150              ; load olcnt register
OLoop:  ldi             ilcnt, 60               ; load ilcnt register
ILoop:  dec             ilcnt                   ; decrement ilcnt
                brne    ILoop                   ; Continue Inner Loop
                dec             olcnt           ; decrement olcnt
                brne    OLoop                   ; Continue Outer Loop
                dec             waitcnt         ; Decrement wait
                brne    Loop                    ; Continue Wait loop

                pop             olcnt           ; Restore olcnt register
                pop             ilcnt           ; Restore ilcnt register
                pop             waitcnt         ; Restore wait register
                ret                             ; Return from subroutine



Exit:
        ldi             mpr, $0f
        out             EIFR, mpr
        ret
;***********************************************************
;*      Stored Program Data
;***********************************************************
                ; Enter any stored data you might need here

;***********************************************************
;*      Additional Program Includes
;***********************************************************
                ; There are no additional file includes for this program
```