ECE 375 LAB 7

Timer/Counters

Lab Time: Tuesday 4-6

Alexander Uong

INTRODUCTION

The purpose of this lab is to understand how implement Timers/Counters into an assembly level program. In this lab, we use two 8-bit Timer/Counters, Timer/Counter0 and Timer/Counter2. Using input through the PORTD pins and output via the PORTB LED's, we use the Timer/Counters provided on the AVR board in Fast PWM mode to modify the speed of the Tekbot. We will be using PORTD pins 0-3 to modify the speed.

PROGRAM OVERVIEW

In this program, we are asked to use two of the AVR board's 8-bit Timer/Counters, Timer/Counter0 and Timer/Counter2, to modify the speed of the Tekbot using Fast PWM mode. Using PORTD pins 0 through 3, we are asked to implement four different functionalities: increase the speed, decrease the speed, set at max speed, and set at minimum speed. There are 15 different speed levels, with the Tekbot speed differing by roughly 17 each level. The value 17 comes from taking the maximum values of OCR0 and OCR2 (output compare registers), 255, and dividing the value by 15.

Pressing pin0 decrease the speed by one level each time it's pressed. However, once the speed of the Tekbot is at minimum speed, pressing pin0 should not do anything. Pressing pin1 increases the speed by one level each time it's pressed. However, like before, if the speed of the Tekbot is at maximum speed, pressing pin1 should do nothing. Pressing Pin3 should set the speed of the Tekbot at the minimum speed, or the lowest level. Pressing Pin4 should set the speed of the Tekbot at the maximum speed, or the highest level.

Besides the INIT and MAIN routine, several subroutines are used, including decreaseSpeed, increaseSpeed, lowestSpeed, and maximumSpeed. decreaseSpeed lowers the speed of the Tekbot by one level when triggered. increaseSpeed increases the speed of the Tekbot by one level when triggered. lowestSpeed sets the Tekbot at the lowest speed when triggered. highestSpeed sets the Tekbot at the highest speed when triggered.

INITIALIZATION ROUTINE

Within the initialization routine, the stack pointer is initialized, PORT B is initialized for output, PORT D is initialized for input, external interrupts are initialized through writing values into EICRA and EIMSK registers, the two 8-bit Timer/Counters are configured, the output compare registers are set, and the initial LED layout is configured.

MAIN ROUTINE

The main routine isn't used in this case.

SUBROUTINES

1. decreaseSpeed Routine

   Checks if the duty cycle is already at 100%; if it is, exit routine, as you can't decrease speed anymore. If it's not, add 17 to the speedRegister and store into output compare registers OCR0 and OCR2. This will modify the brightness of pins 4 and 7. To modify the LED's 0 through 4, load PORTB into mpr and use

instructions logical and immediate and logical or to modify the LED's each time the routine is triggered. Also, clear the flag register to prevent queued interrupts.

2. increaseSpeed Routine

   Checks if the duty cycle is already at 0%; if it is, exit routine, as you can't increase speed anymore. If it's not, subtract 17 to the speedRegister and store into output compare registers OCR0 and OCR2. This will modify the brightness of pins 4 and 7. To modify the LED's 0 through 4, load PORTB into mpr and use instructions logical and immediate and logical or to modify the LED's each time the routine is triggered. Also, clear the flag register to prevent queued interrupts.

3. lowestSpeed Routine

   Sets the speedRegister to the maximum value of 255, which is 100% duty cycle. Store into output compare registers OCR0 and OCR2. To modify the LED's 0 through 4, load PORTB into mpr and use instructions logical and immediate and logical or to modify the LED's each time the routine is triggered. Also, clear the flag register to prevent queued interrupts.

4. maximumSpeed Routine

   Sets the speedRegister to the minimum value of 0, which is 0% duty cycle. Store into output compare registers OCR0 and OCR2. To modify the LED's 0 through 4, load PORTB into mpr and use instructions logical and immediate and logical or to modify the LED's each time the routine is triggered. Also, clear the flag register to prevent queued interrupts.

ADDITIONAL 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 main advantage to only using one Timer/Counter is you have freed the other only 8-bit Timer/Counters on the AVR board. Using both of the 8-bit Timer/Counters only left the 16-bit Timer/Counters for use. This 8-bit Timer/Counter can now be used for other functionalities and purposes. A disadvantage is this method wouldn't be as efficient as one Timer/Counter is only be used to modify and toggle both of the motor enable pins of the Tekbot.

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).

CONCLUSION

Overall, this lab taught me a lot regarding how to implement Timer/Counters. I felt the concept of Timers/Counters was difficult compared to the other material we have learned, but implementing this lab has allowed me to become much more comfortable with the topic. This lab also allowed me to gain more experience with modifying input and outputs on the AVR board.

SOURCE CODE

```
;
; Alexander_Uong_Lab7_sourcecode.asm
;
; Created: 3/1/2021 4:58:37 PM
; Author : Alex Uong
;

;***********************************************************
;*
;*      Alexander_Uong_Lab7_sourcecode.asm
;*
;*      Utilizes Timers/Counters configure and use the 8-bit Timer/Counters on the ATmega128
;* to generate pulse-width modulation (PWM) signals.
;*
;*      This is the skeleton file for Lab 7 of ECE 375
;*
;***********************************************************
;*
;*       Author: Alexander Uong
;*         Date: 3/1/2021
;*
;***********************************************************

.include "m128def.inc"              ; Include definition file

;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                           ; Multipurpose register
.def    speedLevels = r17             ; register keeping track of speedLevel
.def    flagRegister = r18            ;register to clear interrupt flags
.def    speedRegister = r19           ;register to handle speed functions


.equ    minimum = 0                         ;defined as 0 for 0% duty cycle
.equ    maximum = 255                 ;defined as 255 for 0% duty cycle

.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

.equ    initialOutput = 0b01101111    ;initial configuration of LED's
```

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

;***********************************************************
;*      Interrupt Vectors
;***********************************************************
.org    $0000
               rjmp    INIT                    ; reset interrupt

.org    $0002
               rcall decreaseSpeed             //calls decreaseSpeed subroutine if pin0 is pressed
               reti

.org    $0004
               rcall increaseSpeed             //calls increaseSpeed subroutine if pin1 is pressed
               reti

.org    $0006
               rcall lowestSpeed               //calls lowestSpeed subroutine if pin2 is pressed
               reti

.org    $0008
               rcall maximumSpeed              //calls maximumSpeed subroutine if pin3 is pressed
               reti

.org    $0046                                   ; end of interrupt vectors

;***********************************************************
;*      Program Initialization
;***********************************************************
INIT:


               ; Initialize the Stack Pointer
               LDI mpr, LOW(RAMEND)   ; Low Byte of End SRAM Address
               OUT SPL, mpr                    ; Write Byte to SPL
               LDI mpr, HIGH(RAMEND)  ; High Byte of End SRAM Address
               OUT SPH, mpr                    ; Write Byte to SPH


               ; Configure I/O ports

               ; Initialize Port B for output
               ldi mpr, $FF   ;Set port b data direction register
               out DDRB, mpr  ;set for output
               ldi mpr, $00   ;initialize port b data register
               out PORTB, mpr ;all port b outputs are now low

               ; Initialize Port D for input
               ldi mpr, $00   ;set port d data direction register
               out DDRD, mpr  ;set for input
               ldi mpr, $FF   ;initialize port d data register
               out PORTD, mpr ;all port d inputs are tri-state

               ; Configure External Interrupts, if needed

               ; Set the Interrupt Sense Control to falling edge
               ldi mpr,0b10101010
               sts EICRA, mpr

               ; Configure the External Interrupt Mask
               ldi mpr, 0b00001111             //masks pins 0 through 3
               out EIMSK, mpr

               ; Configure 8-bit Timer/Counters
               ldi mpr, 0b01101001             //fast pwm mode, non inverting mode, no prescaling
               out TCCR0, mpr
               out TCCR2, mpr
```

```
                ;initially set OCR0 AND OCR2 equal to zero, or 0% duty cycle (this means full
speed)
                clr mpr
                out OCR0, mpr
                out OCR2, mpr

                ; initial LED output (pins 0-3, 5-6 are illuminated)
                ldi mpr, initialOutput
                out PORTB, mpr


                ; Enable global interrupts (if any are used)
                sei

;***********************************************************
;*      Main Program
;***********************************************************
MAIN:
                ; poll Port D pushbuttons (if needed)

                                                        ; if pressed, adjust speed
                                                        ; also, adjust speed indication

                rjmp    MAIN                    ; return to top of MAIN

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

;-----------------------------------------------------------
; Func: decreaseSpeed
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------

decreaseSpeed:

        cpi speedRegister, maximum    //compare speedRegister with value 255
        breq minSpeed                         //if speedRegister is equal to 255, that means its
already at max speed or 100% duty cycle. so branch to maxSpeed if equal
        dec speedLevels                               //decrement speedLevel

        ldi mpr, 17                                   //load 17 into mpr. Each level differs by 17
        add speedRegister, mpr        //add 17 to speedRegister

        out OCR0, speedRegister               //store register to OCR0 and OCR2 to modify pin 4
and pin 7 brightness
        out OCR2, speedRegister

        in mpr, PORTB                         //load PORTB into mpr
        andi mpr, 0b11110000          //logical and with immediate to mask out pins of PORTB
        or mpr, speedLevels                   //logical or between mpr and speedLevels
        out PORTB, mpr                        //store register to output PORTB

        ;clear flag register
        ldi flagRegister, 0b00001111          //Setting the respective bit places to 1 resets the
flag register contents
        out EIFR, flagRegister                        //this is to avoid queued interrupts


minSpeed:

        clr mpr
        ret


;-----------------------------------------------------------
; Func: increaseSpeed
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
```

```
;-----------------------------------------------------------
increaseSpeed:

        cpi speedRegister, minimum    //compare speedRegister with value 0
        breq maxSpeed                         //if speedRegister is equal to 0, that means its
already at max speed or 0% duty cycle. so branch to minSpeed if equal
        inc speedLevels                               //increment speedLevels otherwise

        ldi mpr, 17                                   //load 17 into mpr. Each level differs by 17
        sub speedRegister, mpr        //subtract 17 from speedRegister. This lowers brightness

        out OCR0, speedRegister               //store register to OCR0 and OCR2 to modify pin 4
and pin 7 brightness
        out OCR2, speedRegister

        in mpr, PORTB                         //load PORTB into mpr
        andi mpr, 0b11110000          //logical and with immediate to mask out pins of PORTB
        or mpr, speedLevels                   //logical or between mpr and speedLevels
        out PORTB, mpr                        //store register to output PORTB


        ;clear flag register
        ldi flagRegister, 0b00001111          //Setting the respective bit places to 1 resets the
flag register contents
        out EIFR, flagRegister                        //this is to avoid queued interrupts

maxSpeed:

        clr mpr
        ret



;-----------------------------------------------------------
; Func: lowestSpeed
; Desc: Cut and paste this and fill in the info at the
;             beginning of your functions
;-----------------------------------------------------------
lowestSpeed:

        ldi speedRegister, maximum            //100% duty cycle->halt. load into speedRegister
        out OCR0, speedRegister                       //store into OCR0 and OCR2
        out OCR2, speedRegister

        in mpr, PORTB                 //load PORTB into mpr
        andi mpr, 0b11110000   //logical and with immediate to mask out pins of PORTB
        ori mpr, 0                            //logical or between mpr and 0
        out PORTB, mpr                //store register to output PORTB


        ;clear flag register
        ldi flagRegister, 0b00001111          //Setting the respective bit places to 1 resets the
flag register contents
        out EIFR, flagRegister                        //this is to avoid queued interrupts


        ret



;-----------------------------------------------------------
; Func: maximumSpeed
; Desc: Cut and paste this and fill in the info at the
;             beginning of your functions
;-----------------------------------------------------------
maximumSpeed:
        ldi speedRegister, minimum            //0% duty cycle->full speed. load into speedRegister
        out OCR0, speedRegister                       //store into OCR0 and OCR2
        out OCR2, speedRegister
```

```
        in mpr, PINB                            //load PORTB into mpr
        andi mpr, 0b11110000                //AND to mask out first 4 bits of PORTB
        ori mpr, 15                                     //logical or between mpr and 15
        out PORTB, mpr                          //store register to output PORTB


        ;clear flag register
        ldi flagRegister, 0b00001111        ;Setting the respective bit places to 1 resets the
flag register contents
        out EIFR, flagRegister                  ;this is to avoid queued interrupts


        ret



;***********************************************************
;*      Stored Program Data
;***********************************************************
                ; Enter any stored data you might need here

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