ECE 375 LAB 6

Timers/Counters

Lab Time: Friday 4-6

Arthur Kichatov

Liuqiao Song

INTRODUCTION

The purpose of this lab is to show understanding of how to configure and utilize eight bit timer and counters using the ATmega32U4 microcontroller board by using pulse width modulation PWM signals. Another goal from conducting this lab is to grasp the concepts of lower/upper half-byte. These are nibbles of the I/O ports for two tasks. The intention of this lab is to make applications of PWM signals comprehensible.

PROGRAM OVERVIEW

The takeaway of this program revolves around the code being written in assembly language and permitting the Bot utilize its speed. To manage the bots speed, the program will use four interrupts. Decreasing the bots speed will use the first interrupt. The second interrupt will increase its speed. The third interrupt will manage that the tekBot will have no speed at all. Lastly, the final interrupt will make sure that the bot will have a maximum speed.

Initialization Routine

In this routine, the stack pointer is initialized so that the pointer can be set to the low and high bits of memory. The outputs for port B and input for port D will also be initialized. Lastly the external interrupts would be in this function in order to execute this lab. These interrupts will react to the falling edge.

Main Routine

This routine is carried out when the bot moves forward and will respond to each of the interrupts depending on which is triggered at a current time.

Speed up Routine

This routine interrupts react to Int 1 being pushed. This will make the bots speed increase. The speed is represented in binary fashion on the Atmega32U4 board.

Speed down Routine

These routine interrupts react to Int 0 being pushed. This will make the bots speed decrease. The speed is represented in binary fashion on the Atmega32U4 board.

Speed minimum Routine

This routine interrupt will make the bot operate at no speed which will be displayed by having and three 0-3 LEDs disabled.

Speed Maximum Routine

This routine interrupt will make the bot operate at no speed which will be displayed by having and three 0-3 LEDs enabled.

Additional Questions

1. . In this lab, you used the Fast PWM mode of 16-bit Timer/Counter, which is only one of many possible ways to implement variable speed on a TekBot. Suppose instead that you used 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

a. External interrupts were the most efficient method when conducting this lab. The benefit of using assembly languages is that the size of the file is half the size of the C file. In addition, using external interrupts allowed to conduct the tasks by waiting for interrupts rather of detecting all the time of weather or not the whiskers are pressed. The benefit of C languages is that it is higher level of programming which has the benefit of having similarity with other languages. This also has the benefit of using libraries that can be used in other languages as well, which also reduces the length of code.

a. A benefit of utilizing the normal mode is that it only needs to utilize one of the clock or timer, which makes it much easier to follow and to see what's going on. A disadvantage of using normal mode is that it can occupy a lot of the CPU's time. Advantages of using the PWM is that it's very efficient and it can also utilize frequencies that are high. Using the PWM permits us the ability to control the speed which will affect how the motors will operate. It will also authorize the output signal to have a larger bandwidth which will produce a higher frequency. A disadvantage is that the power can be inconsistent at times due to the varying nature of the pulse itself. Another disadvantage is that the timer can be overflowed, which will make it go unnoticed.

2. . The previous question outlined a way of using a single 16-bit Timer/Counter in Normal mode to implement variable speed. How would you accomplish the same task (variable TekBot speed) using 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).

a. Initialize TCCR0

Enable the interrupts

Have OCR0 set to a desired number

Then have OCR0 compared with TCNT0

Increment the speed

Followed by decrementing the speed

Difficulties

This lab felt particularly difficult this time. Perhaps this was due to the shorter due date, however, I had a tough time trying to make the LED lights and buttons display correctly. The interrupts however, were very consistent throughout both lab 5 and lab 6.

CONCLUSION

The main intention was to grasps how the configuration of the eight bit timers and counters are meant to be used to create pulse width modulation. This lab required comprehension in being able to implement speed down, up, min, and max routines. The goal that is worth mentioning, is being able to understand how the lower and upper half bits work in assembly language. This helps enhance previous knowledge by showing the concepts and objections of this lab. It also showcased understanding and continuing familiarity with microchip studio software.

SOURCE CODE

```
;***********************************************************
```

```
;*
;*          Author: Aruthr Liuqiao
;*           Date: 11/18/2022
;*
;***********************************************************

.include "m32U4def.inc"                    ; Include definition file

;***********************************************************
;*          Internal Register Definitions and Constants
;***********************************************************
.def       mpr = r16                           ; Multipurpose register
.def    CSPEED=r17
.def    onelevel = r18
.def    waitcount = r23
.def    ilcnt = r24
.def    olcnt = r25

.equ    waitime = 100           ; Time to wait in wait loop
.equ       EngEnR = 5                          ; right Engine Enable Bit
.equ       EngEnL = 6                          ; left Engine Enable Bit
.equ       EngDirR = 4                         ; right Engine Direction Bit
.equ       EngDirL = 7                         ; left Engine Direction Bit

.equ       MoveForward = (1<<EngDirR | 1<<EngDirL)  ; Move Forward Command

;***********************************************************
```

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

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

; place instructions in interrupt vectors here, if needed
.org   $0002		; Representing INT0
	rcall SpeedUp	; Calling the Speed_Down function
reti		; Return from interrupt

.org   $0004		; Representing INT1
	rcall SpeedDown	; Calling the Speed_Up function
reti		; Return from interrupt

.org   $0006		; Representing INT2
	rcall SpeedMax	; Calling the Speed_Min function
  reti		; Return from interrupt

.org   $0008		; Representing INT3
	rcall SpeedMin	; Calling the Speed_Max function
  reti		; Return from interrupt
```

```
.org	$0056					; end of interrupt vectors

;***********************************************************
;*	Program Initialization
;***********************************************************
INIT:
; Initialize the Stack Pointer
ldi mpr, LOW(RAMEND)		; Loading LOW(RAMEND) and storing it into mpr
out SPL, mpr				; Putting mpr into SPL
ldi mpr, HIGH(RAMEND)		; Loading HIGH(RAMEND) and storing it into mpr
out SPH, mpr				; Putting mpr into SPH

; Configure I/O ports



ldi		mpr, 0b11111111	; Setting the data direction register for PortB
out		DDRB, mpr		; Putting mpr into DDRB
ldi		mpr, 0b00001111		; Initialize Port B Data Register
out		DDRD, mpr		; so all Port B outputs are low

;  Initialize Port B for output
ldi		mpr, $FF		; Setting the data direction register for PortD
out		DDRB, mpr					; for input / ; Putting mpr into DDRD
ldi		mpr, $00		; Setting the data direction register for PortD
out		PORTB, mpr			; so all Port D inputs are Tri-State

; Initialize Port D for input
```

```
ldi		mpr, $00		; Setting the data direction register for PortD
out		DDRD, mpr				; for input / ; Putting mpr into DDRD
ldi		mpr, $FF		; Setting the data direction register for PortD
out		PORTD, mpr


; Configure External Interrupts, if needed
ldi	mpr, (1<<ISC01) | (0<<ISC00) | (1<<ISC11) | (0<<ISC10)
sts	EICRA, mpr
ldi		mpr, 0b00001111
out		EIMSK, mpr
ldi		mpr, 0b00001111
out		EIFR, mpr



; Configure 8-bit Timer/Counters
ldi	mpr, 0b01101001   ; Having TCCR0 set to no prescaling and having it non-inverted
out	TCCR0A, mpr  ; Having the right counter non-inverted and set to PWM
out	TCCR0B, mpr  ; Having the left counter non-inverted and set to PWM

;set inital speed
ldi		mpr, $00
out		OCR0A, mpr
out		OCR0B, mpr
```

```
;set inital value for speedcounter
clr             CSPEED
ldi             onelevel, 17

;move forward
ldi     mpr, MoveForward
out     PORTB, mpr

sei  ; Setting the global interrupt flag
;***********************************************************
;*      Main Program
;***********************************************************
MAIN:

ldi             mpr, MoveForward
or              mpr,CSPEED
out             PORTB,mpr

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

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
;-----------------------------------------------------------
FUNC:   ; Begin a function with a label

SpeedMin:

        push    mpr     ; Having mpr pushed on the stack
 in             mpr,SREG
 push mpr

 ldi mpr, $00
 out    OCR0A, mpr
out             OCR0B, mpr

clr             CSPEED
ldi             mpr, 0b0001111
out             EIFR, mpr
pop             mpr     ;restore program state
out             SREG, mpr
pop             mpr     ;restore mpr
```

```
ret




SpeedDown:

	;save variable by pushing to the stack

push mpr

in mpr, SREG

push mpr


in	mpr, OCR0A

cpi	mpr,0

breq	DONE

sub	mpr, onelevel

out	OCR0A, mpr

out	OCR0B, mpr


dec			CSPEED


ldi waitcount, waitime

rcall Wait





DONE:

ldi	mpr, 0b00001111
```

```
out     EIFR, mpr

pop     mpr         ; restore program state
out     SREG,mpr
pop     mpr         ; restore mpr
ret


SpeedUp:
    ; save variable by pushing to the stack
push waitcount
      push mpr
in mpr, SREG
push mpr

in      mpr, OCR0A
cpi     mpr,255
breq    DONE2
add     mpr, onelevel
out     OCR0A, mpr
out     OCR0B, mpr

inc             CSPEED

ldi waitcount, waitime
rcall Wait
```

```
DONE2:

ldi     mpr, 0b00001111

out     EIFR, mpr


pop     mpr     ;restore program state

out     SREG,mpr

pop     mpr     ;restore mpr

pop     waitcount

ret


SpeedMax:


push waitcount

        push    mpr     ; Having mpr pushed on the stack

 in             mpr,SREG

 push mpr


 ldi mpr, $FF

 out            OCR0A, mpr

out             OCR0B, mpr


ldi     CSPEED, 0b00001111


;clr            speed
```

```
ldi		mpr, 0b00001111
out		EIFR, mpr

pop		mpr
out		SREG, mpr
pop		mpr
pop waitcount
ret

Wait:
push waitcount
push	ilcnt
push	olcnt
;Excute the function here

Loop1:
 ldi		olcnt, 221		; Load with a 30us value
Loop2:
ldi		ilcnt, 238
Loop3:
dec ilcnt
brne Loop3
dec olcnt
brne Loop2
dec waitcount
brne Loop3
;restore variable by poping them from stack in reverse order
```

```
pop waitcount
pop ilcnt
pop olcnt

ret                                   ; Return from Wait Function


; If needed, save variables by pushing to the stack

; Execute the function here

; Restore any saved variables by popping from stack

;ret                                  ; End a function with RET

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

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