ECE 375 Lab 6

Timer/Counters

Lab session: 015
Time: 12:00-13:50

Author: Astrid Delestine
Programming partner: Lucas Plastid

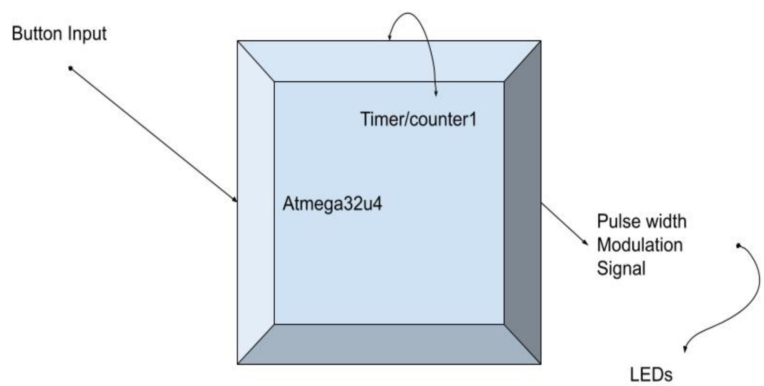
TA Signature

1 Introduction

In this lab the student is expected to learn how to configure the 16 bit Timer/Counter on the ATMEGA32U4 to generate pulse width modulation signals. This is then applied to the LED'S on the bump bot script, to show how speed could be modified on the motors.

2 Design

The design for this Lab was created to fulfill the requirements of this lab. External button inputs will be used to increment and decrement the speed at which the PWM signal operates. This is done by offsetting the Timer/Counter 1.



3 Assembly Overview

As for the Assembly program an overview can be seen below.

3.1 Internal Register Definitions and Constants

The standard mpr and waitcnt registers are assigned to registers 16 and 17 respectively, as are ilcnt and olcnt to 18 and 19. Additionally a register named speeed is set to r20. This register will control the speed at which out PWM signal operates.

3.2 Interrupt Vectors

no interrupt vectors were included in this assignment.

3.3 Initialization Routine

The init routine first setup the stack pointer, then the IO ports B and D are configured fro output and input respectively. Next the 16 bit timer/counter 1 is initialized, this is done through the modification of TCCR1A and TCCR1B. The initial speed is then set by calling the writespd command. Finally the wait counter register is initialized to 100 ms, done by setting it to 5.

3.4 Main Routine

The main subroutine just polls the buttons for input, testing to see if any of them are pressed, and if they are, triggering the expected input. This main routine then loops.

3.5 Subroutines

3.6 INCSPD

This subroutine first checks to see if the max speed has not already been reached, then if not it increments the speed counter. After incrementing the speeed register the WRITESPD subroutine is called. This function will then continue checking to see if the button is being held, and will increment if the button is held, this will handle any bouncing of the button input.

3.7 DECSPD

This subroutine works the same way that the INCSPD subroutine does, except for the fact that it decrements the speeed register instead of incrementing it.

3.7.1 MAXSPD

This subroutine sets the max speed for the PWM. To do this it sets the speeed register to 15, the max value. The hold part of the function is then operated, this part fixes any denouncing.

3.7.2 WRITESPD

The writespd subroutine sets the high byte and the low byte of the speeed register multiplied by 17, this will result in a max value of 255, as the r0 register is moved to mpr. Mpr is then copied to the high and low sides of OCR1A and OCR1B.

3.7.3 Wait

The standard wait function.

4 Testing

Tested Each button press and compared to external calculations.

Case	Expected	Actual meet expected
d4	speed increments	✓
d5	speed decrements	✓
d6	nothing	✓
d7	the max speed for the PWM is set	✓

Table 1: Assembly Testing Cases

5 Study 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 Tek-Bot. 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.

   Some of the advantages of this approach would be having a more granular understanding and control of the signal, the signal would have to be manually interpreted and whenever it overflows, that overflow would need to be handled. This could be considered either a good thing or a bad thing, however with regards to how this timer was setup in this lab, having more understanding might have been easier to program. While it may be easier to program a more direct approach, using the method taken in this lab results in a clearer end product, in the form of code. In the end however they both function in an externally similar way.

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)

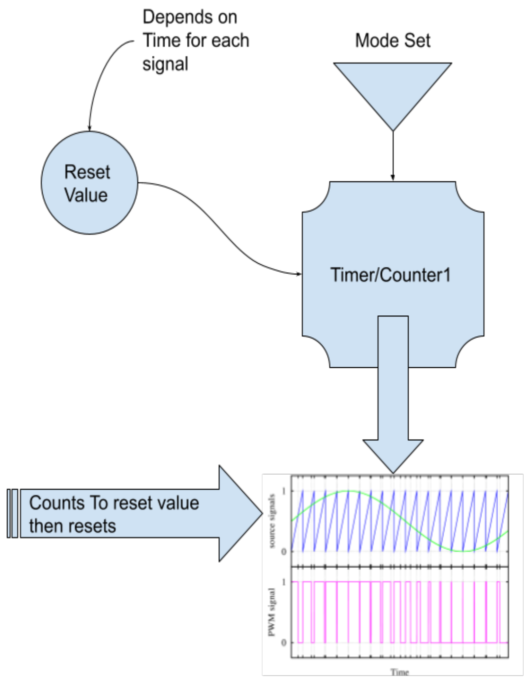
3. In the next lab, you will be utilizing Timer/Counter1, which can make use of several 16 bit timer registers. The datasheet describes a particular manner in which these registers must be manipulated. To illustrate the process, write a snippet of assembly code that configures OCR1A with a value of 0x1234. For the sake of simplicity, you may assume that no interrupts are triggered during your code's operation.

   ldi mpr, High(0x1234) ; You have to set the high first because if you set the low first, you will not actually have set the lower bits correctly. sts OCR1AH, mpr ldi mpr Low(0x1234) sts OCR1AL, mpr

4. Each ATmega32U4 USART module has two flags used to indicate its current transmitter state: the Data Register Empty (UDRE) flag and Transmit Complete (TXC) flag. What is the difference between these two flags, and which one always gets set first as the transmitter runs? You will probably need to read about the Data Transmission process in the datasheet (including looking at any relevant USART diagrams) to answer this question.

   The UDRE flag determines if the USART bus is ready to reccive data. It is set to a 1 when the register of the USART is empty. On the other hand the TXC flag is set when the entire

A drawing of how this might work can be seen below



message has been sent. Thus we know that the UDRE flag is set first as the transmitter runs, waiting for the transmit buffer to be empty before loading it with new data.

5. Each ATmega32U4 USART module has one flag used to indicate its current receiver state (not including the error flags). For USART1 specifically, what is the name of this flag, and what is the interrupt vector address for the interrupt associated with this flag? This time, you will probably need to read about Data Reception in the datasheet to answer this question

   The name of the flag for USART1 is UDRE1, and its interrupt vector location is $0034

6 Difficulties

This Lab was only diffucult due to the fact that there was a lot of reading and direct understanding of the AVR manual. Besides that the lab was quite simple to understand.

7 Conclusion

In conclusion, this lab allowed the student to experiment with using the 16 bit timer/counter and gain a better understanding for how dimming LEDs work and how motor speed control can work. Additionally the student learned how to find specific item in the AVR manual in a faster and clearer way. While this may not have been an intended goal of the lab, it was a useful outcome.

8 Source Code

Listing 1: Assembely Script

```
;***********************************************************
;*
;*   This is the skeleton file for Lab 6 of ECE 375
;*
;*    Author: Astrid Delestine & Lucas Plaisted
;*       Date: 3/1/23
;*
;***********************************************************

.include "m32U4def.inc"          ; Include definition file

;***********************************************************
;*   Internal Register Definitions and Constants
;***********************************************************
.def     mpr = r16               ; Multipurpose register
.def     waitcnt = r17           ; Wait Loop Counter,
                                 ; waitcnt*10ms for delay
.def     ilcnt = r18             ; Inner Loop Counter
.def     olcnt = r19             ; Outer Loop Counter
.def     speeed = r20                ; Speed register, max of 15

;***********************************************************
;*   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     $0056                   ; end of interrupt vectors

;***********************************************************
;*  Program Initialization
;***********************************************************
INIT:
         ; Initialize the Stack Pointer
         ldi     mpr, low(RAMEND)
         out     SPL, mpr        ; Load SPL with low byte of RAMEND
```

```
        ldi     mpr, high(RAMEND)
        out     SPH, mpr            ; Load SPH with high byte of RAMEND

        ; Configure I/O ports
            ; Initialize Port B for output
            ldi     mpr, $FF        ; Set Port B Data Direction Register
            out     DDRB, mpr       ; for output
            ldi     mpr, $00        ; Initialize Port B Data Register
            out     PORTB, mpr      ; so all Port B outputs are low
            ; Initialize Port D for input
            ldi     mpr, $00        ; Set Port D Data Direction Register
            out     DDRD, mpr       ; for input
            ldi     mpr, $FF        ; Initialize Port D Data Register
            out     PORTD, mpr      ; so all Port D inputs are Tri-State

        ; Configure 16-bit Timer/Counter 1A and 1B
            ; TCCRIA Bits:
                ; 7:6 - Timer/CounterA compare mode, 10 = non-inverting mode
                    ; On compare match clears port B pin 5
                ; 5:4 - Timer/CounterB compare mode, 10 = non-inverting mode
                    ; On compare match clears port B pin 6
                ; 3:2 - Timer/CounterC compare mode, 00 = disabled
                ; 1:0 - Wave gen mode low half, 01 for 8-bit fast pwm
            ldi mpr, 0b10_10_00_01
            sts TCCR1A, mpr
            ; TCCRIB Bits:
                ; 7:5 - not relevant, 0's
                ; 4:3 - Wave gen mode high half, 01 for 8 bit fast pwm
                ; 2:0 - Clock selection, 001 = no prescale
            ldi mpr, 0b000_01_001
            sts TCCR1B, mpr
            ; Fast PWM, 8-bit mode, no prescaling
                ; In inverting Compare Output mode output is cleared on compare

        ; Set initial speeed, display on Port B pins 3:0
            ldi speeed, $0F
            rcall WRITESPD

        ldi waitcnt, 5   ; Set wait timer to be 100ms

;***********************************************************
;*  Main Program
;***********************************************************
MAIN:
        ; poll Port D pushbuttons (if needed)
        in mpr, PIND
```

```
        sbrs mpr, 7 ; Run next command if button 7 presed (active low)
        rcall MAXSPD
        sbrs mpr, 5
        rcall DECSPD
        sbrs mpr, 4
        rcall INCSPD

        ldi mpr, $00

        rjmp    MAIN            ; return to top of MAIN

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

;-----------------------------------------------------------
; Func: INCSPD
; Desc: Increases the "speeed" of the motor by increasing
;       the width of the pulse. Has built in debouncing.
;       Prevents going over the max speeed.
;-----------------------------------------------------------
INCSPD:
        ; Push to stack
        push mpr
        cpi speeed, 15  ; check if we are at max speed
        breq INCSKIP    ; Don't incremend
        inc speeed      ; increase the speeed
        rcall WRITESPD
INCHOLD:                ; Don't leave until we let go of the button
        rcall Wait      ; Wait 50ms, debouncing
        in mpr, PIND    ; Grab current button value
        sbrs mpr, 4     ; Check if button is still held
        rjmp INCHOLD    ; Stay in loop if held
INCSKIP:
        pop mpr
        ret                     ; End a function with RET

;-----------------------------------------------------------
; Func: DECSPD
; Desc: Decreases the "speeed" of the motor by decreasing
;       the width of the pulse. Has built in debouncing
;-----------------------------------------------------------
DECSPD:
        ; Push to stack
        push mpr
        cpi speeed, 0   ; If speeed is 0
```

```
        breq DECSKIP        ; Don't decrement
        dec speeed
        rcall WRITESPD
        ; Pop from stack
DECHOLD:                    ; Don't leave until we let go of the button
        rcall Wait          ; Wait 50ms, debouncing
        in mpr, PIND        ; Grab current button value
        sbrs mpr, 5         ; Check if button is still held
        rjmp DECHOLD        ; Stay in loop if held
DECSKIP:
        pop mpr
        ret                         ; End a function with RET

;-----------------------------------------------------------
; Func: MAXSPD
; Desc: Increases the "speeed" to max (15)
;-----------------------------------------------------------
MAXSPD:
        push mpr

        ldi speeed, 15
        rcall WRITESPD
MAXHOLD:                    ; Don't leave until we let go of the button
        rcall Wait          ; Wait 100ms, debouncing
        in mpr, PIND        ; Grab current button value
        sbrs mpr, 7         ; Check if button is still held
        rjmp MAXHOLD        ; Stay in loop if held

        pop mpr
        ret                         ; End a function with RET

;-----------------------------------------------------------
; Func: WRITESPD
; Desc: Sets the timer compares for the current speeed as
;       well as setting the lower nibble of
;-----------------------------------------------------------
WRITESPD:
        push mpr
        push R0
        push R1

        ldi mpr, 17         ; 255/15 = 17
        mul speeed, mpr     ; speeed*17 = pulse width, result in R0
        clr mpr             ; set mpr to 0
        sts OCR1AH, mpr     ; write to high byte of compare A
        mov mpr, R0         ; place output into mpr. Max 255 = 1 reg
```

```
        sts OCR1AL, mpr ; write to low byte of compare B
        clr mpr
        sts OCR1BH, mpr ; clear high of compare B
        mov mpr, R0     ; copy output to mpr again
        sts OCR1BL, mpr ; write to low byte of compare B
        ldi mpr, 0b10010000
        add mpr, speeed
        out PORTB, mpr

        pop R1
        pop R0
        pop mpr
        ret

        ;-----------------------------------------------------------
; Sub:  Wait
; Desc: A wait loop that is 16 + 159975*waitcnt cycles or roughly
;       waitcnt*10ms.  Just initialize wait for the specific amount
;       of time in 10ms intervals. Here is the general eqaution
;       for the number of clock cycles in the wait loop:
;           (((((3*ilcnt)-1+4)*olcnt)-1+4)*waitcnt)-1+16
;-----------------------------------------------------------
Wait:
        push    waitcnt         ; Save wait register
        push    ilcnt           ; Save ilcnt register
        push    olcnt           ; Save olcnt register

Loop:   ldi     olcnt, 224      ; load olcnt register
OLoop:  ldi     ilcnt, 237      ; 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

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

```

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