

ECE 375 Lab 6

Timer/Counters

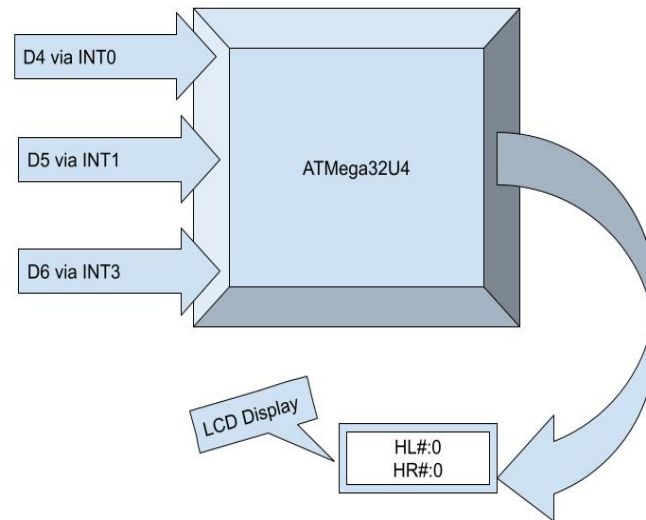
Lab session: 015
Time: 12:00-13:50

Author: Astrid Delestine
Programming partner: Lucas Plastid

TA Signature

1 Introduction

2 Design



3 Assembly Overview

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

3.1 Internal Register Definitions and Constants

3.2 Interrupt Vectors

3.3 Initialization Routine

3.4 Main Routine

3.5 Subroutines

3.6 INCSPD

3.7 DECSPD

3.7.1 MAXSPD

3.7.2 WRITESPD

3.7.3 Wait

The standard wait function.

4 Testing

Tested Each button press and compared to external calculations.

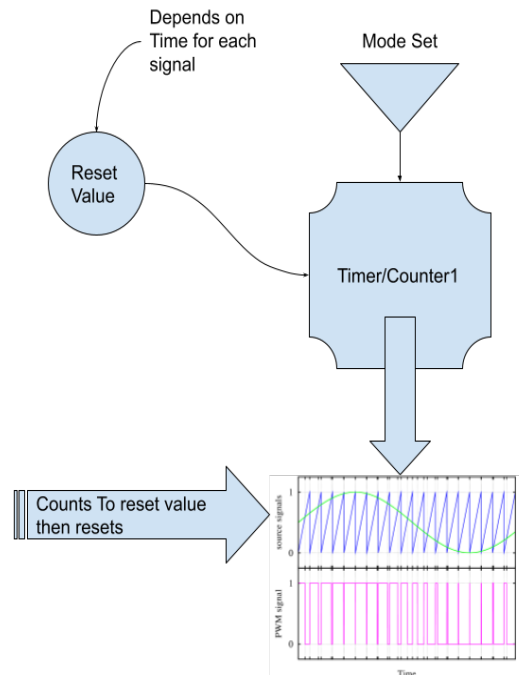
Case	Expected	Actual meet expected
d4	an increment on the LCD, and the bump bot right hit function to be called	✓
d5	an increment on the LCD, and the bump bot left hit function to be called	✓
d6	The two numbers listed on the screen reset	✓
d7	nothing	✓

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.
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 drawing of how this might work can be seen below



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.
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.
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

6 Difficulties

This Lab challenged the thinking power of implementation of ideas we have learned in lecture. It was not too difficult however did require referencing both the AVR manual and the atmega32u4 datasheet.

7 Conclusion

In conclusion, this lab introduced and allowed the student to understand many more aspects of how to program a function using interrupts and modify an existing program to work with an LCD in conjunction with those interrupts. This lab proves that the student is learning how to modify certain code structures and is becoming more fluent in the AVR programming scheme

8 Source Code

Listing 1: Assembly Script

```
1  ;*****
2  ;*
3  ;*   This is the skeleton file for Lab 6 of ECE 375
4  ;*
5  ;*   Author: Astrid Delestine & Lucas Plaisted
6  ;*   Date: 3/1/23
7  ;*
8  ;*****
9
10 .include "m32U4def.inc"           ; Include definition file
11
12 ;*****
13 ;*   Internal Register Definitions and Constants
14 ;*****
15 .def      mpr = r16                ; Multipurpose register
16 .def      waitcnt = r17            ; Wait Loop Counter,
17                                           ; waitcnt*10ms for delay
18 .def      ilcnt = r18              ; Inner Loop Counter
19 .def      olcnt = r19              ; Outer Loop Counter
20 .def      speeded = r20            ; Speed register, max of 15
21
22 ;*****
23 ;*   Start of Code Segment
24 ;*****
25 .cseg                             ; beginning of code segment
26
27 ;*****
28 ;*   Interrupt Vectors
29 ;*****
30 .org      $0000
31          rjmp      INIT            ; reset interrupt
32
33          ; place instructions in interrupt vectors here, if needed
34
35 .org      $0056                    ; end of interrupt vectors
36
37 ;*****
38 ;*   Program Initialization
39 ;*****
40 INIT:
41          ; Initialize the Stack Pointer
42          ldi        mpr, low(RAMEND)
43          out        SPL, mpr        ; Load SPL with low byte of RAMEND
```

```

44      ldi      mpr, high(RAMEND)
45      out      SPH, mpr          ; Load SPH with high byte of RAMEND
46
47      ; Configure I/O ports
48      ; Initialize Port B for output
49      ldi      mpr, $FF          ; Set Port B Data Direction Register
50      out      DDRB, mpr        ; for output
51      ldi      mpr, $00          ; Initialize Port B Data Register
52      out      PORTB, mpr       ; so all Port B outputs are low
53      ; Initialize Port D for input
54      ldi      mpr, $00          ; Set Port D Data Direction Register
55      out      DDRD, mpr        ; for input
56      ldi      mpr, $FF          ; Initialize Port D Data Register
57      out      PORTD, mpr       ; so all Port D inputs are Tri-State
58
59      ; Configure 16-bit Timer/Counter 1A and 1B
60      ; TCCR1A Bits:
61          ; 7:6 - Timer/CounterA compare mode, 10 = non-inverting mode
62          ; On compare match clears port B pin 5
63          ; 5:4 - Timer/CounterB compare mode, 10 = non-inverting mode
64          ; On compare match clears port B pin 6
65          ; 3:2 - Timer/CounterC compare mode, 00 = disabled
66          ; 1:0 - Wave gen mode low half, 01 for 8-bit fast pwm
67      ldi mpr, 0b10_10_00_01
68      sts TCCR1A, mpr
69      ; TCCR1B Bits:
70          ; 7:5 - not relevant, 0's
71          ; 4:3 - Wave gen mode high half, 01 for 8 bit fast pwm
72          ; 2:0 - Clock selection, 001 = no prescale
73      ldi mpr, 0b000_01_001
74      sts TCCR1B, mpr
75      ; Fast PWM, 8-bit mode, no prescaling
76          ; In inverting Compare Output mode output is cleared on compare
77
78      ; Set initial speed, display on Port B pins 3:0
79      ldi speed, $0F
80      rcall WRITESPD
81
82      ldi waitcnt, 5 ; Set wait timer to be 100ms
83
84      ; *****
85      ; * Main Program
86      ; *****
87 MAIN:
88      ; poll Port D pushbuttons (if needed)
89      in mpr, PIND

```

```

90      sbrs mpr, 7 ; Run next command if button 7 presed (active low)
91      rcall MAXSPD
92      sbrs mpr, 5
93      rcall DECSPD
94      sbrs mpr, 4
95      rcall INCSPD
96
97      ldi mpr, $00
98
99      rjmp  MAIN ; return to top of MAIN
100
101 ; *****
102 ; * Functions and Subroutines
103 ; *****
104
105 ;-----
106 ; Func: INCSPD
107 ; Desc: Increases the "speed" of the motor by increasing
108 ;       the width of the pulse. Has built in debouncing.
109 ;       Prevents going over the max speed.
110 ;-----
111 INCSPD:
112     ; Push to stack
113     push mpr
114     cpi speed, 15 ; check if we are at max speed
115     breq INCSKIP ; Don't increment
116     inc speed ; increase the speed
117     rcall WRITESPD
118 INCHOLD: ; Don't leave until we let go of the button
119     rcall Wait ; Wait 50ms, debouncing
120     in mpr, PIND ; Grab current button value
121     sbrs mpr, 4 ; Check if button is still held
122     rjmp INCHOLD ; Stay in loop if held
123 INCSKIP:
124     pop mpr
125     ret ; End a function with RET
126
127 ;-----
128 ; Func: DECSPD
129 ; Desc: Decreases the "speed" of the motor by decreasing
130 ;       the width of the pulse. Has built in debouncing
131 ;-----
132 DECSPD:
133     ; Push to stack
134     push mpr
135     cpi speed, 0 ; If speed is 0

```



```

136      breq DECSKIP      ; Don't decrement
137      dec speed
138      rcall WRITESPD
139      ; Pop from stack
140 DECHOLD:                ; Don't leave until we let go of the button
141      rcall Wait        ; Wait 50ms, debouncing
142      in mpr, PIND       ; Grab current button value
143      sbrs mpr, 5        ; Check if button is still held
144      rjmp DECHOLD      ; Stay in loop if held
145 DECSKIP:
146      pop mpr
147      ret                ; End a function with RET
148
149 ;-----
150 ; Func: MAXSPD
151 ; Desc: Increases the "speed" to max (15)
152 ;-----
153 MAXSPD:
154      push mpr
155
156      ldi speed, 15
157      rcall WRITESPD
158 MAXHOLD:                ; Don't leave until we let go of the button
159      rcall Wait        ; Wait 100ms, debouncing
160      in mpr, PIND       ; Grab current button value
161      sbrs mpr, 7        ; Check if button is still held
162      rjmp MAXHOLD      ; Stay in loop if held
163
164      pop mpr
165      ret                ; End a function with RET
166
167 ;-----
168 ; Func: WRITESPD
169 ; Desc: Sets the timer compares for the current speed as
170 ;       well as setting the lower nibble of
171 ;-----
172 WRITESPD:
173      push mpr
174      push R0
175      push R1
176
177      ldi mpr, 17        ; 255/15 = 17
178      mul speed, mpr    ; speed*17 = pulse width, result in R0
179      clr mpr           ; set mpr to 0
180      sts OCR1AH, mpr   ; write to high byte of compare A
181      mov mpr, R0       ; place output into mpr. Max 255 = 1 reg

```

```

182     sts OCR1AL, mpr ; write to low byte of compare B
183     clr mpr
184     sts OCR1BH, mpr ; clear high of compare B
185     mov mpr, R0      ; copy output to mpr again
186     sts OCR1BL, mpr ; write to low byte of compare B
187     ldi mpr, 0b10010000
188     add mpr, speed
189     out PORTB, mpr
190
191     pop R1
192     pop R0
193     pop mpr
194     ret
195
196     ;-----
197 ; Sub:  Wait
198 ; Desc: A wait loop that is 16 + 159975*waitcnt cycles or roughly
199 ;       waitcnt*10ms. Just initialize wait for the specific amount
200 ;       of time in 10ms intervals. Here is the general equation
201 ;       for the number of clock cycles in the wait loop:
202 ;       (((((3*ilcnt)-1+4)*olcnt)-1+4)*waitcnt)-1+16
203 ;-----
204 Wait:
205     push    waitcnt      ; Save wait register
206     push    ilcnt        ; Save ilcnt register
207     push    olcnt        ; Save olcnt register
208
209 Loop:   ldi      olcnt, 224      ; load olcnt register
210 OLoop:  ldi      ilcnt, 237      ; load ilcnt register
211 ILoop:  dec      ilcnt          ; decrement ilcnt
212         brne     ILoop          ; Continue Inner Loop
213         dec      olcnt          ; decrement olcnt
214         brne     OLoop          ; Continue Outer Loop
215         dec      waitcnt        ; Decrement wait
216         brne     Loop           ; Continue Wait loop
217
218     pop      olcnt            ; Restore olcnt register
219     pop      ilcnt            ; Restore ilcnt register
220     pop      waitcnt          ; Restore wait register
221     ret                    ; Return from subroutine
222
223 ; *****
224 ; *   Stored Program Data
225 ; *****
226 ; Enter any stored data you might need here
227

```

```
228 ;*****
229 ;*  Additional Program Includes
230 ;*****
231      ; There are no additional file includes for this program
```