ECE 375 LAB 7

Timers / Counters

Lab Time: Thursday 1000-1200

Eric Prather

PRELAB

1. List the correct sequence of AVR assembly instructions needed to store the contents of registers R25:R24 into Timer/Counter1's 16-bit register, TCNT1. (You may assume that registers R25:R24 have already been initialized to contain some 16-bit value.)

```
STS LOW(TCNT1), r24 ; NOT "OUT" ;
STS HIGH(TCNT1), r25
```

2. List the correct sequence of AVR assembly instructions needed to load the contents of Timer/Counter1's 16-bit register, TCNT1, into registers R25:R24.

```
LDS r24, LOW(TCNT1) ; NOT "IN"
LDS r25, HIGH(TCNT1)
```

3. Suppose Timer/Counter0 (an 8-bit timer) has been configured to operate in Normal mode, and with no prescaling (i.e., clkT0 = clkI/O = 16 MHz). The decimal value "128" has just been written into Timer/Counter0's 8-bit register, TCNT0. How long will it take for the TOV0 flag to become set? Give your answer as an amount of time, not as a number of cycles.

Important: Assuming the clock source is the system clock and the counter increments once per cycle, as default.

(255-128) * (1/(16*10^6)) = 8 * 10^-6 seconds = 4 microseconds.

INTRODUCTION

This lab introduces timers/counters in a use case of moderate complexity- to rapidly toggle some output pins so that they appear to be analog output. At a broader level, these output pins would correspond to motor power, so causing this behavior would result in a gradual increase or decrease in the speed of the TekBot. However, in our case, we are just sending to LEDs, so we are really just simulating dim-ness or bright-ness.

This is not the only skill tested by this lab; students are additionally required to synthesize their own I/O graphical user interface as well. This means using the push-button inputs to send commands to the ATMega128 and displaying the current speed of the output oscillation as a binary number. Notably, the following commands must be supported:

- Increase speed
- Decrease speed
- Max speed
- Min speed

Some additional qualifying requirements are added to this as well, such as overflow avoidance and anti-bounce.

PROGRAM OVERVIEW

Initialization Routine

This program's initialization routine prepares the following components. Some of these were discussed in previous labs, some were introduced in this lab, but overall they are all just a couple of lines per item. There is no considerable logic happening during this part of the program's execution, they are just simple hardcoded statements. It would be an exaggeration to say any decision making happened here, either; most functionality was simply pulled from the data sheet.

- Stack pointer
- Port D Input (Tri-state buffered input)
- Port B Output (0xZ11Z1111)
- Clock PWM (TCCR0 & TCCR2)
    - Pre-scaling
    - Interrupts
- PIND Interrupt
    - EICRA; Falling edge for all 4
    - EIMSK; 00001111

Interrupt Vectors

Branching conditional is implemented as follows:

PIND == 0bXXXXXXX1:
    Increase speed
PIND == 0bXXXXXX1X:
    Decrease speed
PIND == 0bXXXXX1XX:
    Max speed
PIND == 0bXXXX1XXX:
    Min speed
These commands issue on the falling edge of the interrupt.

The clock interrupts toggle the value being output to PORTB[0bX__X_____]. This produces the intended behavior because of the pulse-width modulation.

ResetOCR

Calculates a number [0..255] from the speed [0-15] to put in timer/counter 0's output compare register, then puts it there. Must be called every time the speed is changed.

ClockCompareInterrupt

Turns off motor power and calculates next compare interrupt based on speed.

ClockOverflowInterrupt

Turns on motor power

Main Routine

Loops forever.

IncreaseSpeed

Increments a value in data memory, not past 15, then writes it to:

a) The clock speed controller

b) The lower nibble of PortB

DecreaseSpeed

Decrements a value in data memory, not past 15, then writes it to:

a) The clock speed controller

b) The lower nibble of PortB

MaxSpeed

Sets a value in data memory to 15, then writes it to:

a) The clock speed controller

b) The lower nibble of PortB

MinSpeed

Sets a value in data memory to 15, then writes it to:

a) The clock speed controller

b) The lower nibble of PortB

DoSpeed

Writes a value in data memory to:

a) The clock speed controller

b) The lower nibble of PortB

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 very detailed set of instructions provided in this question introduce an effective "manual" pulse width modulation. This would take far more clock cycles, code, and room for error than using the built in PWM system. On the AT128Mega board at least, there is no good way to do concurrent processing. The processor itself is very simple. So any logic which has to be executed via machine instructions will inherently take longer than something built into the board itself using digital building blocks. The development cost of replicating PWM is also non-negligible. Lastly, the arithmetic and memory storage required to implement PWM manually would be time and memory inefficient, although the impact in the broader scheme of things is debatable depending on the pre-scaling of the clock (running once very 255 cycles is much less efficient than once every several thousand cycles).

However, it may be desirable in some circumstances to manually replicate PWM despite the above limitations. Firstly, if the assembly code needs to be portable between devices which are not known to support implicit PWM, it may be mandatory. Furthermore, it is possible that the clocks on the board are overloaded with too many responsibilities. There are only four timer/counters on the AT128Mega, for example, so this is a plausible concern. In this case, it may be advantageous to "double up" the responsibility of some timer/counters so that when their interrupt fires, in addition to the regular behavior, the PWM is emulated in separate data space.

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 8bit 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).

On the ATMega 128 board, CTC mode is "Clear Timer on Compare Match" mode. It works by using a dedicated timer comparison register to decide when a reset/interrupt happens instead of defaulting to the clock cycle that increments the counter from 255 to 0. Implementing PWM through CTC would be much simpler than through normal mode. All that needs to happen is to change the comparison register every time an interrupt occurs so that interrupts occur concurrently with a predefined waveform. This will require a conditional and toggle. See the following pseudocode for a specific implementation.

```
PWM Duty cycle: 40% (103 cycles on, 153 cycles off) implemented via CTC

DATA MEMORY:
        Active = false

CTC Interrupt ROUTINE:
         if(Active)
                Compare register = 152
                LED => OFF
         else:
                Compare register = 103
                LED => ON
        Active = !Active ; toggle active
```

DIFFICULTIES

At first, I tried to do the **alternative** method for modulation as described in more detail in the study questions. I only realized later that the intended method was to use two separate timers.

My output compare register didn't work in the debugger. I talked to some folks about it and they said just don't use the simulator.

Even since last lab, I was unable to figure out how to remove button bounce.

For some reason, Button S4 did not always exhibit the intended behavior if it was pressed repeatedly; sometimes it triggered the interrupt bound to button S3 instead.

I had some very bizarre edge cases at 255 and 0 for my duty cycle. I ended up writing a hard-coded solution instead of using multiplication to write the correct numbers to the OCR0 register.

I did not understand what the lab prompt meant by using two clocks. I was able to accomplish this lab using just one clock. I hope this is not a problem.

CONCLUSION

This lab demonstrated in depth the usage of one of the non-basic forms of the clock in the ATMega128 board. Working closely with FastPWM mode and no prescaling required a very precise management of the system resources, especially clock cycles. Completing this lab helped to reinforce good assembly practice and helped make the process of using interrupt vectors much easier through practice. Conceptualizing the clock signal as a waveform was also good preparatory material for Lab 8, when a detailed dissection of exact waveforms will be mandatory for inter-board communication.

SOURCE CODE

```
;***********************************************************
;*
;*      Prather_Eric_Lab7_sourcecode
;*
;*  Drives the tekbot forward, but allows the speed to be
;*  configured via interrupts from PORTD. Enabling
;*  technology is PWM mode of TCNT0.
;*
;***********************************************************
;*
;*       Author: Eric Prather (prathere@oregonstate.edu)
;*         Date: Feb 26, 2020
;*
;***********************************************************

.include "m128def.inc"               ; Include definition file

;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                           ; Multipurpose register
.def    mpr2 = r17

.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

; Macros for tekbot movement (From lab 1)
.equ    MovFwd = (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 INT0addr
               rjmp IncreaseSpeed
               ;reti
.org INT1addr
               rjmp DecreaseSpeed
               ;reti
.org INT2addr
               rjmp MaxSpeed
               ;reti
.org INT3addr
               rjmp MinSpeed
               ;reti
.org    $001E ; Timer 0 Compare Match (FALLING EDGE)
               rjmp ClockCompareInterrupt
               ;reti
.org    $0020 ; Timer 0 Overflow (RISING EDGE)
               rjmp ClockOverflowInterrupt
               ;reti


.org    $0046                                   ; end of interrupt vectors

;***********************************************************
;*      Program Initialization
;***********************************************************
INIT:
               ;;; BEGIN LAB 1 CODE SEGMENT ;;;
               ; Initialize the Stack Pointer (VERY IMPORTANT!!!!)
               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

           ; 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

               ; Initialize TekBot Forward Movement
               ldi             mpr, MovFwd             ; Load Move Forward Command
               out             PORTB, mpr              ; Send command to motors
               ;;; END LAB 1 CODE SEGMENT ;;;


               ; Configure External Interrupts, if needed
               ; See lab 6 for more detail
               ldi mpr, 0b10101010 ; Falling edge for all 4 buttons
                                                       ; See datasheet page 89-90
               sts EICRA, mpr ; Can't understand why we use sts?
               ; Fortunately, we do not have to use EICRB
               ldi mpr, 0b00001111 ; Initalize interrupts to clear flags?
                                                       ;  I  don't  know  if  this  is  actually
necessary,
                                                       ; but whatever, I'll bite.
               out EIFR, mpr
               out EIMSK, mpr      ; We want to enable the same interrupts as those
```

```
                                                ; whose flags we just cleared.

                ; Configure 8-bit Timer/Counters
                ; See page 104 of datasheet for register descriptions
                ; See pages 98-99 of datasheet for FastPWM description
                ; Set TCCR0 to No prescaling, non-inverting fast PWM
                ldi mpr, 0b01001011 ; Prescaler bits (CS00,CS01,CS02 => clk_T0S:001), fast PWM
                                    ; (WGM01:0=3) must be set. COM01:0 => Non inverting.
Clear FOC.
                out TCCR0, mpr      ; Write to control register


                ; Output compare register should be based on duty cycle.
                ldi mpr, 0x7F ; About speed 7
                out OCR0, mpr
                ; Enable timer interrupts on falling and rising edge of PWM mode
                ldi mpr, 0b00000011 ; (1<<OCIE0)| (1<<OCIE1)  ; Mask
                out TIMSK, mpr
                ldi mpr, 0b00000011 ;(1<<TOV0) | (1<<OFC0) ; Flags
                out TIFR, mpr


                ; Set TekBot to Move Forward (1<<EngDirR|1<<EngDirL)
                ; Set initial speed, display on Port B pins 3:0
                ldi ZL, low(SPEED)
                ldi ZH, high(SPEED)
                ldi  mpr,  0b00000111  |  MovFwd  ;  Initial  speed  =  7,  for  kicks,  in  the  forward
direction
                st Z, mpr
                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: ResetOCR
; Desc: Calculates a number [0..255] from the speed [0-15]
;               to put in timer/counter 0's output compare register,
;           then puts it there. Must be called every time the
;               speed is changed.
;-----------------------------------------------------------
ResetOCR:
        push mpr2
        push ZL
        push ZH
        push mpr
        in mpr, sreg
        push mpr
        ; Logical body
        ld mpr, Z ; mpr = SPEED
        andi mpr, 0b00001111 ; 0-15 speed number
        ldi mpr2, 0b00010001 ; 0d16

        ; CONDITION 1: MAX SPEED
        cpi mpr, 0b00001111
        brne CON_2
```

```
        ldi mpr, 0b00000000 ; This is the only way to circumvent any compare=
        rjmp OCR_CALCULATED
CON_2:
        ; CONDItiON 2: MIN SPEED
        cpi mpr, 0b00000000
        brne CON_3
        ldi mpr, 0b00000011 ; Very close to, but not quite, 0- see above
        rjmp OCR_CALCULATED
CON_3:
        mul mpr, mpr2 ; Moves to R1:R0. We only care about least significant byte, since max =
255
        mov mpr, r0   ; Now contains # of active high duty cycles
OCR_CALCULATED:
        out OCR0, mpr ; Set next time to do the falling edge based on speed.

        ; End Logical body; Cleanup
        pop mpr
        out sreg, mpr
        pop  mpr
        pop ZH
        pop ZL
        pop mpr2
        ret

;-----------------------------------------------------------
; Func: ClockCompareInterrupt
; Desc: Turns off the motor
;-----------------------------------------------------------
ClockCompareInterrupt: ; FALLING EDGE
        push mpr2
        push ZL
        push ZH
        push mpr
        in mpr, sreg
        push mpr
        ; Logical body
        ldi ZL, low(SPEED)
        ldi ZH, high(SPEED)

        ; RESET OCR
        rcall ResetOCR

        ; DISPLAY
        ld mpr, Z ; mpr = SPEED
        ori mpr, 0b10010000 ; Turn on motors
        st Z, mpr

        st Z, mpr
        rcall DoSpeed
        ; Pre-Cleanup (Flush flags)
        ldi mpr, 0b00000011 ; (1<<OFC0) | (1<<TOV0) ; Flags
        out TIFR, mpr
        ; Cleanup
        pop mpr
        out sreg, mpr
        pop  mpr
        pop ZH
        pop ZL
        pop mpr2
        reti

;-----------------------------------------------------------
; Func: ClockOverflowInterrupt
; Desc: Turns the motor back on
;-----------------------------------------------------------
ClockOverflowInterrupt: ; RISING EDGE
        push mpr2
        push ZL
        push ZH
        push mpr
        in mpr, sreg
```

```
        push mpr
        ; Logical body
        ldi ZL, low(SPEED)
        ldi ZH, high(SPEED)

        ; DISPLAY
        ld mpr, Z ; mpr = SPEED
        andi mpr, 0b01101111 ; Turn off motors
        st Z, mpr

        st Z, mpr
        rcall DoSpeed
        ; Pre-Cleanup (Flush flags)
        ldi mpr, 0b00000011 ;(1<<TOV0) | (1<<OFC0); Flags
        out TIFR, mpr
        ; Cleanup
        pop mpr
        out sreg, mpr
        pop  mpr
        pop ZH
        pop ZL
        pop mpr2
        reti

;-----------------------------------------------------------
; Func: IncreaseSpeed
; Desc: Increments the lower nibble of SPEED by 1, but not
;              past 15
;-----------------------------------------------------------
IncreaseSpeed:
        push mpr2
        push ZL
        push ZH
        push mpr
        in mpr, sreg
        push mpr
        ; Logical body
        ldi ZL, low(SPEED)
        ldi ZH, high(SPEED)
        ld mpr, Z
        mov mpr2, mpr ; Split tekbot nibble and number nibble
        andi mpr2, 0b11110000 ; Mask to just tekbot output
        andi mpr, 0b00001111 ; Mask to just a 4 bit number
        cpi mpr, 0b00001111
        brge AT_MAX_SPEED ; Don't go out of bounds
        inc mpr
        or mpr, mpr2 ; Merge nibbles
        st Z, mpr
AT_MAX_SPEED:
        rcall ResetOCR
        ; Pre-cleanup
        ldi mpr, 0b00001111
        out EIFR, mpr
        ; Cleanup
        pop mpr
        out sreg, mpr
        pop  mpr
        pop ZH
        pop ZL
        pop mpr2
        reti

;-----------------------------------------------------------
; Func: DecreaseSpeed
; Desc: Decrements the lower nibble of SPEED by 1, but not
;              below 0.
;-----------------------------------------------------------
DecreaseSpeed:
        push mpr2
        push ZL
        push ZH
```

```
        push mpr
        in mpr, sreg
        push mpr
        ; Logical body
        ldi ZL, low(SPEED)
        ldi ZH, high(SPEED)
        ld mpr, Z
        mov mpr2, mpr ; Split tekbot nibble and number nibble
        andi mpr2, 0b11110000 ; Mask to just tekbot output
        andi mpr, 0b00001111 ; Mask to just a 4 bit number
        cpi mpr, 0b00000001
        brlt AT_MIN_SPEED ; Don't go out of bounds
        dec mpr
        or mpr, mpr2 ; Merge nibbles
        st Z, mpr
AT_MIN_SPEED:
        rcall ResetOCR
        ; Pre-cleanup
        ldi mpr, 0b00001111
        out EIFR, mpr
        ; Cleanup
        pop mpr
        out sreg, mpr
        pop  mpr
        pop ZH
        pop ZL
        pop mpr2
        reti

;-----------------------------------------------------------
; Func: MaxSpeed
; Desc: Sets the lower nibble of SPEED to 0b1111
;-----------------------------------------------------------
MaxSpeed:
        push ZL
        push ZH
        push mpr
        in mpr, sreg
        push mpr
        ; Logical body
        ldi ZL, low(SPEED)
        ldi ZH, high(SPEED)
        ld mpr, Z
        ori mpr, 0b00001111
        st Z, mpr

        rcall ResetOCR
        ; Pre-cleanup
        ldi mpr, 0b00001111
        out EIFR, mpr
        ; Cleanup
        pop mpr
        out sreg, mpr
        pop  mpr
        pop ZH
        pop ZL
        reti

;-----------------------------------------------------------
; Func: MinSpeed
; Desc: Sets the lower nibble of SPEED to 0b0000
;-----------------------------------------------------------
MinSpeed:
        push ZL
        push ZH
        push mpr
        in mpr, sreg
        push mpr
        ; Logical body
        ldi ZL, low(SPEED)
        ldi ZH, high(SPEED)
```

```
        ld mpr, Z
        andi mpr, 0b11110000
        st Z, mpr

        rcall ResetOCR
        ; Pre-cleanup
        ldi mpr, 0b00001111
        out EIFR, mpr
        ; Cleanup
        pop mpr
        out sreg, mpr
        pop  mpr
        pop ZH
        pop ZL
        reti

;-----------------------------------------------------------
; Func: DoSpeed
; Desc: Displays SPEED to PORTB.
;-----------------------------------------------------------
DoSpeed:
        push ZL
        push ZH
        push mpr
        in mpr, sreg
        push mpr
        ; Logical body
        ldi ZL, low(SPEED)
        ldi ZH, high(SPEED)
        ld mpr, Z
        out PORTB, mpr
        ; Cleanup
        pop mpr
        out sreg, mpr
        pop  mpr
        pop ZH
        pop ZL
        ret

;***********************************************************
;*      Stored Program Data
;***********************************************************
               ; Enter any stored data you might need here
.dseg
.org $0100
SPEED: ; The speed (0 - 15)
        .byte 1

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