# ECE 375 Lab 7

## External Interrupts

Lab Time: Wednesday 10a-12n

Robert Detjens

David Headrick

---

TA Signature

# Introduction

In this lab, we learned how to use the Timer/Counters on the ATMega128 to perform PWM management on output pins. In addition, we also learned how to write to different portions of an I/O port at different times. The timers on the board can be used to generate square waves at a user-configured duty cycle, allowing for easy PWM output on some of the output pins.

# Program Overview

As this lab does not require the full BumpBot functionality, this program is relatively simple in structure. This program uses interrupts to react to button inputs and adjusts the necessary outputs in the interrupt handlers. The main loop of this program does nothing, as all of the need functionality of this program is contained within those handlers.

This program keeps track of the current speed level in one of the chip registers. This level ranges from 0 to 15, the same as what is shown in the lower half of the LEDs. This makes updating the LED I/O port easy, as this is a direct store without any conversion necessary (aside from loading the current state of the other half of the pins). To convert this internal speed value into the correct value to store into the Timers, it is multiplied by 17 to convert to a range of 0 to 255. This multiplication is done manually via shifting (for $\times 16$) and an subsequent add to get the final $\times 17$ value. This is then stored into both of the timer count registers to set the new duty cycle.

As both the LED update and the timer update are the same across all of the interrupt handlers, these were broken out into subprocedures and called from each handler in order to de-duplicate code.

# 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.

   One advantage to doing this approach would be that you have an extra 8-bit Timer/Counter at your disposal for any other use case. Another advantage is that you'd have more control over the semantics of an overflow in relation to the motors.

   Advantages to the PWM method used in this lab would be that you don't have to use an interrupt to control PWM for the motor pins, they just get controlled in the background after everything is set up. A drawback of this approach is that it occupies both 8-bit counters to essentially do the same operation. If you need more counters, and interrupts don't bother you, the previously mentioned method would be optimal.

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).

   One 8-bit timer in CTC mode and it's corresponding interrupt could be used to perform the same action. The interrupt would be manually changing OCR0 every time it overflows to manipulate the duty cycle. The OCR pin would be set to toggle every overflow.

```
INIT:
 set timer/counter 0 to CTC mode
 set OCR0 to 0x08
 set OCR pin to toggle every overflow

ISR for Timer/Counter0:
 XOR OCR0 and 0x88 to switch between duty cycle counts
```

# Difficulties

We spent a large amount of time troubleshooting why our timer was not being updated by our button interrupts. We were using the `mul` instruction to convert our stored speed level (0 to 15) into the appropriate timer count level (0 to 255). This was done by multiplying the speed level by 15, and storing that result into the timer. However, we did not fully understand how the `mul` instruction worked and did not realize it stored the result in a different register pair instead of updating the first operand with the result. We eventually double checked how it worked and realized this, and decided instead to switch to a more efficient manual shift-and-add instead of using `mul` since 17 is very close to a power of two and easily done via shifting.

Aside from this one problem, no other components of this lab gave us much trouble.

# Conclusion

Barring the one issue with using `mul` incorrectly, this lab went smoothly. We feel confident now in our ability to configure and use the 8-bit Tiemr/Counters functionality for both timing and PWM. There are several roughly-equivalent ways of performing PWM (as discussed in the previous Questions section), and in researching those for said questions, we are confident in our ability to use those approaches as well.

## Source Code

```
;***********************************************************
;*
;*  Robert Detjens & David Headrick Lab 7 Source Code
;*
;***********************************************************
;*
;*   Author: Robert Detjens
;*           David Headrick
;*     Date: 11/16/21
;*
;***********************************************************

.include "m128def.inc"    ; Include definition file

;***********************************************************
;*  Internal Register Definitions and Constants
;***********************************************************
.def  mpr        = r16  ; Multipurpose register
.def  curr_speed = r17  ; current speed level (0-15) * 17

.def  waitcnt = r20   ; WaitFunc Loop Counter
.def  ilcnt = r21     ; Inner Loop Counter
.def  olcnt = r22     ; Outer Loop Counter

.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

; Timer config bits
.equ  FOCx   = 1 << 7
.equ  WGMx0  = 1 << 6
.equ  COMx1  = 1 << 5
.equ  COMx0  = 1 << 4
.equ  WGMx1  = 1 << 3
.equ  CSx2   = 1 << 2
.equ  CSx1   = 1 << 1
.equ  CSx0   = 1 << 0

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

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

.org $0002
  rcall SpeedMax    ; IRQ0 Handler - set maximum speed
  reti
.org $0004
  rcall SpeedInc    ; IRQ1 Handler - increment speed
  reti
.org $0006
  rcall SpeedDec    ; IRQ2 Handler - decrement speed
  reti
.org $0008
  rcall SpeedMin    ; IRQ3 Handler - set minimum speed
  reti

.org  $0046      ; 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 Directional Register
  out DDRB, mpr ; for output
  ldi mpr, $00 ; Initialize Port B for outputs
  out PORTB, mpr ; Port B outputs low

  ; Initialize Port D for input
  ldi mpr, $00 ; Set Port D Directional Register
  out DDRD, mpr ; for inputs
  ldi mpr, $FF ; Initialize Port D for inputs
  out PORTD, mpr ; with pull-up
```

```
  ; Configure External Interrupts, if needed

  ; Set the Interrupt Sense Control to falling edge
  ; Set INT0:3 to be on falling edge
  ldi mpr, 0b10101010
  sts EICRA, mpr

  ; Configure the External Interrupt Mask
  ldi mpr, 0b00001111
  out EIMSK, mpr

  ; Configure 8-bit Timer/Counters

  ; Enable PWM with no prescaling, set on OCR clear on overflow.
  ldi mpr,    WGMx0 | WGMx1 | COMx1 | CSx0
  out TCCR0,  mpr   ; T/C 0
  out TCCR2,  mpr   ; T/C 2

  ; Set TekBot to Move Forward (1<<EngDirR|1<<EngDirL)
  ldi mpr, (1<<EngDirR|1<<EngDirL)
  out PORTB, mpr    ; Send command to motors

  ; Set initial speed, display on Port B pins 3:0
  ldi   curr_speed,  0  ; start at full speed
  rcall UpdateTimers

  ; Enable global interrupts (if any are used)
  sei

;************************************************************
;*  Main Program
;************************************************************
MAIN:
  ; everything is handled in interrput handlers
  rjmp  MAIN    ; return to top of MAIN

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

;-----------------------------------------------------------
; Func:   SpeedMax
; Desc:   GO TO PLAID
;-----------------------------------------------------------
```

```
SpeedMax:

  ldi   curr_speed,   15
  rcall UpdateTimers

  ; delay a bit for debounce
  ldi   mpr,   1
  rcall WaitFunc

  ; clear interrupt
  ldi   mpr,    0b00001111
  out   EIFR,   mpr

  ret

;------------------------------------------------------------
; Func:   SpeedInc
; Desc:   Increment speed level by one
;------------------------------------------------------------
SpeedInc:

  cpi   curr_speed, 15
  brge  Inc_noop
    ; only increment if below max
    inc   curr_speed
  Inc_noop:
  rcall UpdateTimers

  ; delay a bit for debounce
  ldi   mpr,   1
  rcall WaitFunc

  ; clear interrupt
  ldi   mpr,    0b00001111
  out   EIFR,   mpr

  ret

;------------------------------------------------------------
; Func:   SpeedDec
; Desc:   Decrement speed level by one
;------------------------------------------------------------
SpeedDec:

  cpi   curr_speed,   0
```

```
  breq  Dec_noop
    ; only decrement if above min
    dec   curr_speed
  Dec_noop:
  rcall UpdateTimers

  ; delay a bit for debounce
  ldi   mpr,   1
  rcall WaitFunc

  ; clear interrupt
  ldi   mpr,    0b00001111
  out   EIFR,   mpr

  ret

;------------------------------------------------------------
; Func:   SpeedMin
; Desc:   Set minimum speed
;------------------------------------------------------------
SpeedMin:

  ldi   curr_speed,   0
  rcall UpdateTimers

  ; delay a bit for debounce
  ldi   mpr,   1
  rcall WaitFunc

  ; clear interrupt
  ldi   mpr,    0b00001111
  out   EIFR,   mpr

  ret

;------------------------------------------------------------
; Func:   UpdateTimers
; Desc:   Update timers and display with new speed value
;------------------------------------------------------------
UpdateTimers:
  push  mpr

  ; update leds with current speed
  ldi mpr,    (1<<EngDirR|1<<EngDirL)
  or    mpr,    curr_speed
```

```
  out   PORTB,  mpr

  ; convert speed level to timer match value
  ; mpr * 17 == mpr * 16 + mpr == mpr << 4 + mpr
  mov   mpr,    curr_speed
  lsl   mpr
  lsl   mpr
  lsl   mpr
  lsl   mpr
  add   mpr,    curr_speed

  out   OCR0,   mpr
  out   OCR2,   mpr

  pop   mpr
  ret

;-----------------------------------------------------------------
; Sub:  WaitFunc
; 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 + 3) * olcnt + 3) * waitcnt + 13 + call
;-----------------------------------------------------------------
WaitFunc:
  push        waitcnt         ; Save waitregister
  push        ilcnt           ; Save ilcntregister
  push        olcnt           ; Save olcntregister

  Loop:   ldi   olcnt, 224        ; load olcnt register
    OLoop:  ldi   ilcnt, 50        ; load ilcnt register
      ILoop:  dec   ilcnt            ; decrement ilcnt
        brne        ILoop            ; Continue InnerLoop
      dec         olcnt            ; decrementolcnt
      brne        OLoop            ; Continue OuterLoop
    dec         waitcnt          ; Decrementwait
    brne        Loop             ; Continue Funcloop

  pop         olcnt           ; Restore olcntregister
  pop         ilcnt           ; Restore ilcntregister
  pop         waitcnt         ; Restore waitregister
  ret                         ; Return fromsubroutine

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

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

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