ECE 375 Lab 7

timer/counters

Lab Time: Friday 4-6

Aaron Vaughan

Bradley Heenk

TA Signature

1 Introduction

The purpose of this lab is to familiarize ourselves with the functionality and initialization of the timer/counters on the atmega128. We were tasked with creating a variable speed selection motor driver simulator. We kept track of the current speed with LEDs 3:0 via a binary counter that could not overflow. Having good awareness of how to mask outputs and update the current state based on the previous state was crucial for the implementation of the functionality of our design. The extra credit portion keeps track of the seconds between each speed increase and decrease and displays it as a binary value devoid of all leading zeros.

2 Program Overview

This program provides a simulation of a motor control circuit via the fast pulse wave modulation output from the 8 bit timer/counters. The drive motor direction is constantly in the forward drive state and the enable lines are pulsed at a high frequency to change the speed output. We implemented the challenge code which was to display the binary count in seconds since the last change in speed occurred. This meant that we were limited in the use of multipurpose registers again. To get around this limitation, we used push and pop to reuse multi purpose registers over and over again in the same subroutine. The LCD count used timer/counter3 because that is the only counter that does not have dual functionality on the LED output (PORTB). Also, it is a 16-bit counter, which helps to simplify the one second wait loop that we created. All of this code uses interrupt routines to simplify the code. It simply counts up to max value from 3036 with a 256 prescaler. When the max value is reached the interrupt is set and the LCD display is incremented.

3 Internal Register Definitions and Constants

We use 4 multipurpose registers for the main functionality of our program. One non multi purpose register, R5, was used in the LCD to keep track of the global counter value. We use some common names for readability of the drive state of the TekBot as well, such as MovFwd and MovBck. There is a step value of 17 that is kept as a constant to use as the increment value for each speed output on the board.

4 Interrupt Vectors

The main reset vector is always used. We also used interrupts INT0-INT3 to execute the four main functions of the speed selection, Max speed, Min speed, Inc speed, and Dec speed. The timer/Counter3 interrupt vector is also used to control the LCD timer.

5 Program Initialization

We initialize the stack pointer, PORTD for input to use the buttons on the board, PORTB for output to use the LEDs on the board, configure the INT3:0 interrupts to act on falling edge, mask interrupts INT7:4, configure the two 8-bit counters as Fast PWM mode with no prescale and inverted output, timer/counter3 is set as normal mode with a 256 prescale to elapse 1 second before triggering an interrupt, set the global interrupt, initialize the LCD screen, and set the starting speed and direction of the TekBot.

6 Main Program

The main program is blank. This program just loops and waits for an interrupt command.

7 Subroutines

Function name: Time Elapsed
Description: When this function is executed it resets the LCD driver to start counting in binary from 0 to 11111111

Function Name: LCD Binary
Description: An Extension of the LCDDrive.asm library to take a specific value in mpr and display it on the lcd in binary output with no leading zeroes and update the display.

Function Name : Display Zero
Description: Loads a Zero into the display and post-inc's X

Function Name: Display One
Description: Loads a One into the display and post-inc's X

Function Name: Max Speed
Description: Set the output to show motors in max speed arrangement

Function Name: Min Speed
Description: Set the output to show motors in min speed arrangement

Function Name: Increment Speed
Description: This function should increment the binary output on PORTB(0..3) and increment OCCR0 by "step"

Function Name: Decrement Speed
Description: This function should increment the binary output on PORTB(0..3) and decrement OCCR0 by "step"

Subroutine Name: Wait
Description: 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. This wait cycle is used to debounce the mechanical buttons.

8 Stored Program Data

There is not any stored program data for this lab.

9 Additional Program Includes

The LCD driver file must be included in this lab to utilize the functionality of the LCD screen. "LCDDriver.asm"

10 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 clear advantage would be that we would only be using one timer/counter to execute this. If we were concerned about power consumption of our device this would better utilize the resources of the atmega128. The disadvantage of this aproach would be the added complexity of the code. It may affect the readability of it and complicate the process of adding features or functionality at a later date due to the complexity of code.

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)

```
Enable Interrupts Set up interrupt handler value ← 17
timer/Counter0 ← CTC Mode & enable interrupts, prescale = 1
INCREMENT Function:
OCR0 ← OCR0 + value
Send OC0 to PORTB 7,4
```

```
DECREMENT Function:
OCR0 ← OCR0 - value
Send OC0 to PORTB 7,4
```

11 Difficulties

The debounce sequence for the buttons had me tripped up for an hour or so. I kept getting multiple function calls. The process of waiting until the button signal is stabilized makes sense now. Setting the LCD screen up to ignore the leading zeros took more lines of code than we expected.

12 Conclusion

The main code for this lab was extremely simple. It took a bit of time to search the atmega manual to find the correct initialization values for Fast PWM mode, but the code was extremely easy to write. The utility of having the knowledge needed to implement a motor controller will prove to be invaluable in my career. This lab is by far the most useful so far. The extra credit was a bit tricky but only because I though that we should be using CTC mode for the other counter sequence. The normal mode proved to be an easier option. By the time I realized this, we had already completed the homework which had us set up the timer/counter1 for normal mode and create a 1 second wait sequence. The extra credit followed this same logic, so it was basically cut and paste code and poof it worked. Each time we enter a subroutine that changes the speed of the motor output, the timer is reset. Otherwise, it simply counts up until $FF at which point it overflows and resets to 0.

13 Source Code & Challenge Code

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

```
;* Aaron_Vaughan_and_Brandley_Heenk_Lab7_challengecode.avr
;*
;* This program uses fast PWM 8-bit timers
;*  to simulate a motor controller. The speed
;*  is displayed via a binary counter on LED 3:0.
;*  The challenge code creates a counter on the LCD
;*  that tracks the time in binary until a button
;*  is pressed. To see the program operate in the
;*  extra credit mode, uncomment the LCDinit line
;*
;*
;***********************************************************
;*
;*  Author: Aaron Vaughan and Bradley Heenk
;*    Date: 11/15/2019
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multipurpose register
; Registers R17 to R22 are
; reserved for the LCD driver
.def waitcnt = r23
.def ilcnt = r24 ; Also the speed counter
.def olcnt = r25 ; Also the step register
.def counter = r5

.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

.equ Normal_Mode = 0

.equ MovFwd = (1<<EngDirR|1<<EngDirL) ; Move Forward Command
.equ MovBck = $00         ; Move Backward Command
```

```
.equ TurnR = (1<<EngDirL)      ; Turn Right Command
.equ TurnL = (1<<EngDirR)      ; Turn Left Command
.equ Halt = (1<<EngEnR|1<<EngEnL) ; Halt Command
.equ Step = $11      ; Each Step is decimal 17
.equ One = $31
.equ Zero = $30

;***********************************************************
;* 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
rcall MaxSpd ; call max speed handler
reti
.org $0004
rcall MinSpd ; call min speed handler
reti
.org $0006
rcall IncSpd ; call inc speed handler
reti
.org $0008
rcall DecSpd ; call dec speed handler
reti
.org $003A
rcall TimeElapsed
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 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
mov counter, mpr
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 external interrupts
ldi mpr, $AA ; Set the Interrupt Sense Control to falling edge
sts EICRA, mpr

; Configure the External Interrupt Mask
ldi mpr, $0F ; Set value to what we want to hide
out EIMSK, mpr

; Configure 8-bit Timer/Counters
ldi mpr, $79 ; No prescaling, Fast PWM, Inverted Compare
out TCCR0, mpr
ldi mpr, $79 ; No prescaling, Fast PWM, Inverted Compare
out TCCR2, mpr

; init the counter/timer3 normal, 256, interrupt enable
ldi mpr, high(3036) ; Set the starting value to 3036
sts TCNT3H, mpr
ldi mpr, low(3036)
sts TCNT3L, mpr
```

```
ldi mpr, Normal_Mode ; Set normal mode
sts TCCR3A, mpr
ldi mpr, (1<<CS32) ; 256 prescaler
sts TCCR3B, mpr
ldi mpr, (1<<TOIE3) ; Enable the TOV interrupt
sts ETIMSK, mpr



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

; Start up the LCD screen
rcall LCDInit

; Set initial speed, display on Port B pins 3:0
ldi olcnt, $00 ; Start at min speed
ldi ilcnt, $00 ; Start at min speed
or mpr, ilcnt
out PORTB, mpr



; Enable global interrupts (if any are used)
sei

;***********************************************************
;* Main Program
;***********************************************************
MAIN:
; Wait for an interrupt and just loop around

rjmp MAIN ; return to top of MAIN

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

```
;-----------------------------------------------------------
; Func: Time Elapsed
; Desc: When executed resets the LCD driver to start
; counting in binary from 0 to 11111111 where each
; time it gets incremented it is based on
;-----------------------------------------------------------
TimeElapsed:
; Save the states of the registers and push onto the stack
push waitcnt
push mpr
push ilcnt
in mpr, SREG ; Save the state of SREG
push mpr
push olcnt

mov mpr, counter
rcall LCDBinary

inc mpr
mov counter, mpr

; Restore the old values to our registers and remove off the stack
pop olcnt
pop mpr
out SREG, mpr ; Restore the state of SREG
pop ilcnt
pop mpr
pop waitcnt

ret

;-----------------------------------------------------------
; Func: LCD Binary
; Desc: An Extension of the LCDDrive.asm library to take
; a specific value in mpr and display its on the lcd
; in binary output with no leading zeroes and update
; the display
;-----------------------------------------------------------
LCDBinary:
```

```
ldi XL, low(LCDLn1Addr) ; Load the low byte of LCDLn1Addr into XL
ldi XH, high(LCDLn1Addr) ; Load the high byte of LCDLn1Addr into XH

ldi waitcnt, 0 ; Changes to a 1 when we detect our first 1
rcall LCDClr ; Clear the display

sbrc mpr, 7 ; Check if the 7th bit has leading zero if
; If so then skip the next instruction
ldi waitcnt, 1 ; If executed we hit our first one enable writing
sbrs mpr, 7 ; Check the 7th bit if set skip the next instruction
rcall DisplayZero ; Since we were cleared go ahead and display a one
sbrc    mpr, 7 ; Check the 7th bit if cleared skip the next instruction
rcall DisplayOne ; Since we were set go ahead and display a one

; Same sort of setup for the rest of these...
; Except for the last step which will always display
; The leading zero

sbrc mpr, 6
ldi waitcnt, 1
sbrs mpr, 6
rcall DisplayZero
sbrc    mpr, 6
rcall DisplayOne

sbrc mpr, 5
ldi waitcnt, 1
sbrs mpr, 5
rcall DisplayZero
sbrc    mpr, 5
rcall DisplayOne

sbrc mpr, 4
ldi waitcnt, 1
sbrs mpr, 4
rcall DisplayZero
sbrc    mpr, 4
rcall DisplayOne

sbrc mpr, 3
```

```
ldi waitcnt, 1
sbrs mpr, 3
rcall DisplayZero
sbrc    mpr, 3
rcall DisplayOne

sbrc mpr, 2
ldi waitcnt, 1
sbrs mpr, 2
rcall DisplayZero
sbrc    mpr, 2
rcall DisplayOne

sbrc mpr, 1
ldi waitcnt, 1
sbrs mpr, 1
rcall DisplayZero
sbrc    mpr, 1
rcall DisplayOne

ldi waitcnt, 1
sbrs mpr, 0
rcall DisplayZero
sbrc    mpr, 0
rcall DisplayOne

rcall LCDWrite

ret

;-------------------------------------------------------------
; Func: Display Zero
; Desc: Loads a Zero into the display and post-inc's X
;-------------------------------------------------------------
DisplayZero:

cpi waitcnt,1 ; Check if we can start writing Zeroes
brne DISPZEROEXIT ; If not then exit

ldi ilcnt, Zero ; Load our Zero value into a register ilcnt
```

```
st X+, ilcnt ; Load the value in ilcnt into the display
; and post inc X

DISPZEROEXIT:

ret

;-----------------------------------------------------------
; Func: Display One
; Desc: Loads a One into the display and post-inc's X
;-----------------------------------------------------------
DisplayOne:
ldi ilcnt, One ; Load a one into ilcnt register
st X+, ilcnt ; Load the value in ilcnt into our display
; then post in X so we point to the next spot

DISPONEEXIT:

ret

;-----------------------------------------------------------
; Func: Max Speed
; Desc: Set the output to show motors in max speed arrangement
;-----------------------------------------------------------
MaxSpd: ; Begin a function with a label

; If needed, save variables by pushing to the stack
push mpr
push ilcnt

in mpr, portB
andi mpr, $0F
cpi mpr, $0F
breq START
clr counter

START:
; Execute the function here
ldi mpr, $ff
out OCR0, mpr
out OCR2, mpr
ldi mpr, 0b01101111 ; Force motors to max speed
out PORTB, mpr ; Send command to motors

; Restore any saved variables by popping from stack
pop ilcnt
pop mpr

ret ; End a function with RET

;-----------------------------------------------------------
; Func: Min Speed
; Desc: Set the output to show motors in min speed arrangement
;-----------------------------------------------------------
MinSpd: ; Begin a function with a label

; If needed, save variables by pushing to the stack
push mpr
; Execute the function here

in mpr, portB
andi mpr, $0F
cpi mpr, $00
breq START1
clr counter

START1:
ldi ilcnt, $00
push ilcnt
ldi mpr, $00
out OCR0, mpr
out OCR2, mpr
ldi mpr, 0b11110000 ; Force motors to min speed
out PORTB, mpr ; Send command to motors

; Restore any saved variables by popping from stack
pop ilcnt
pop mpr

ret ; End a function with RET
```

```
;-----------------------------------------------------------
; Func: Increment Speed
; Desc: This funciton should increment the binary output
; on PORTB(0..3) and increment OCCR0 by "step"
;-----------------------------------------------------------
IncSpd: ; Begin a function with a label

; If needed, save variables by pushing to the stack
push mpr
push ilcnt
in mpr, SREG
push mpr
push olcnt

; Execute the function here

in ilcnt, PORTB ; Get the current state
mov mpr, ilcnt ; Get a copy of current state
andi ilcnt, $0F ; Mask the motor state
andi mpr, $F0 ; Mask the counter state
cpi ilcnt, $0F ; Make sure we arent at max speed
breq SKIP_ADD ; Skip the update if we are

; Step the motor speed by 1/16 speed increments
clr counter
push mpr ; Save mpr
ldi mpr, step ; Load in the step value (decimal 17)
in olcnt, OCR0 ; Get the current speed
add olcnt, mpr ; Increment the speed by the step value
out OCR0, olcnt ; Set the new speed
out OCR2, olcnt ; Set the new speed
pop mpr
; Increment the speed display
inc ilcnt ; Increment the Speed counter to update the led count
or ilcnt, mpr
out PORTB, ilcnt ; Update the LEDs

SKIP_ADD:
; This is for Debounce sequence, wait and flag...
```

```
rcall WAITFUNC
ldi mpr, $0F
out EIFR, mpr

; Restore any saved variables by popping from stack
pop olcnt
pop mpr
out SREG, mpr
pop ilcnt
pop mpr

ret ; End a function with RET

;-----------------------------------------------------------
; Func: Decrement Speed
; Desc: Cut and paste this and fill in the info at the
; beginning of your functions
;-----------------------------------------------------------
DecSpd: ; Begin a function with a label

; If needed, save variables by pushing to the stack
push mpr
push ilcnt
in mpr, SREG
push mpr
push olcnt

; Execute the function here

in ilcnt, PORTB ; Get the current Speed count
mov mpr, ilcnt ; Get a copy of the current state of the output
andi ilcnt, $0F ; Mask the motor state
andi mpr, $F0 ; Mask the counter state
cpi ilcnt, $00 ; Make sure we aren't at min speed already
breq SKIP_ADD2 ; Skip the rest if we are

; Step the motor speed by 1/16 speed increments
clr counter
push mpr ; Save mpr
ldi mpr, step ; Load in the step value (decimal 17)
```

```
in olcnt, OCR0 ; Get the current speed
sub olcnt, mpr ; decrement the speed by the step value
out OCR0, olcnt ; Set the new speed
out OCR2, olcnt ; Set the new speed
pop mpr
; Decrement the speed count display
dec ilcnt ; Increment the Speed counter to update the led count
or ilcnt, mpr ; Get the new state of our motors and count
out PORTB, ilcnt ; Update the LEDs to display new state

SKIP_ADD2:
; This is our debounce sequence, wait and flag...
rcall WAITFUNC
ldi mpr, $0F
out EIFR, mpr

; Restore any saved variables by popping from stack
pop olcnt
pop mpr
out SREG, mpr
pop ilcnt
pop mpr

ret ; End a function with 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 + 3) * olcnt + 3) * waitcnt + 13 + call
;-----------------------------------------------------------------
WAITFUNC:
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, 30 ; 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
;***********************************************************
.include "LCDDriver.asm"
```