ECE 375 Lab 7

Timer/Counters

Lab Time: Thursday 10:00 - 11:50

Jeremy Fischer

TA Signature

1 Introduction

The purpose of this lab was to give students more practice with interrupts, as well as teach them how to use timers/counters. Specifically, how to use Fast Pulse Width Modulation to control the intensity of a connected device.

2 Internal Register Definitions and Constants

This program utilized Interrupts INT0, INT1, INT2, and INT3. All three were initialized to trigger on the falling edge of the clock. INT0 was responsible for handling a button press that requests an increase in speed. INT1 was responsible for handling a button press that requests a decrease in speed. INT2 was responsible for handling a button press that requests a jump tp max speed. INT3 was responsible for handling a button press that requests the engines to halt.

Below are the labels I gave definitions in this program. They are mainly definitions which represent bit locations for enable engine, engine direction, and speed changes within Port D.

```
.def mpr = R16                                          ;Multipurpose register
.def speedLevel = R17                                   ;register that holds the current speed
.def changePerLevel = R18                               ;holds the speed change per level
.def newSpeed = R0                                      ;holds the newSpeed to put into OCRx
.def waitcnt = R19                                      ;wait loop counter
.equ WTime = 10                                         ;(wait time in MS) / 10ms. So 1s
.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 MovFwd = (1<< EngDirR)|(1 << EngDirL)              ;move forward
.equ ddrdLayout = 0b11110000                            ;upper nibble = output, lower = input
.equ IncSpeed = 0                                       ;increment speed input bit
.equ DecSpeed = 1                                       ;decrement speed input bit
.equ FullSpeed = 2                                      ;full speed input bit
.equ SlowSpeed = 3                                      ;slowest speed input bit
```

3 Init()

Init is the first function that is ran and it is responsible for initializing the stack pointer, initializing each bit of Port B for output, initializing the lower nibble of Port D for input, and setting Port D to function as a pull-up resistor since the buttons are passive devices. Init also sets bit in EICRA to have INT0, INT1, INT2, and INT3 to trigger on the falling edge, and sets bits 0, 1, 2, and 3 to a one in EIMSK to allow INT0, INT1, INT2, and INT3 to be detected.

TCCR0 and TCCR2 are then set up to work in non-inverting Fast PWM mode with a prescale of 1024.

Lastly, the initial speed-level is zero and the move forward command is loaded into Port B to turn on the two engine LEDs.

4 Main()

The main program is in an infinite loop which OR's together the move forward command and the current speed level. The result of that OR is sent to Port B so the corresponding LEDs are lit up. Specifially, Bits 6 and 5 are always on because the robot moves forward indefinitely, and the lower four LEDs are lit to match the current speed level represented in binary, with the highest bit being LED 4.

5 INCRSPEED()

When INCRSPEED() is first entered it waits for 100ms. Then, the current speedlevel is checked. If the speedLevel is 15 (the fastest speedlevel), then nothing happens. If the speedLevel is less than 15, then the speedLevel is increased by one and the the multiplication of the new speedLevel and the intensity per level is output to both OCR0 and OCR2, which will make the engine LEDs brighter. On exiting the function, EIFR is cleared of any staged interrupts.

6 DECRSPEED()

When DECRSPEED() is first entered it waits for 100ms. Then, the current speedlevel is checked. If the speedLevel is 0 (the slowest speedlevel), then nothing happens. If the speedLevel is greater than 0, then the speedLevel is decreased by one and the the multiplication of the new speedLevel and the intensity per level is output to both OCR0 and OCR2, which will make the engine LEDs dimmer. On exiting the function, EIFR is cleared of any staged interrupts.

7 JMPFASTEST()

When JMPFASTEST() is first entered it waits for 100ms. The speedLevel is changed to 15 (the fastest speedLevel) and the the multiplication of the new speedLevel and the intensity per level is output to both OCR0 and OCR2, which will make the engine LEDs the brightest possible. On exiting the function, EIFR is cleared of any staged interrupts.

8 JMPSLOWEST()

When JMPSLOWEST() is first entered it waits for 100ms. The speedLevel is changed to 0 (the slowest speedLevel) and the the multiplication of the new speedLevel and the intensity per level is output to both OCR0 and OCR2, which will turn off the engine LEDs, because the robot is stopped. On exiting the function, EIFR is cleared of any staged interrupts.

9 WAIT()

WAIT exploits the fact that each instruction$_i$ takes X_i processing time. WAIT ends up working out as 16 + 159975*waitcnt cycles or roughly waitcnt*10ms. So, for one second I set waitcnt to 100 since 100*10ms = 1000ms = 1 second.

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/Counters 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.*

   Advantages:

   - The main advantage to manually performing PWM is that it allows the developer to use only one timer/counter. This allows them to leverage the other timer for another purpose.

   Disadvantages:

   - The developer would have to write more code
   - The developer having to write more code means there is more room for possible (logic) errors.
   - The interrupt will be fired more often, causing the CPU to work harder due to it repetitively saving and restoring the state.
   - Due to the interrupt being fired more frequently, it may block out other interrupts.

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).*

   The same approach wold be taken as the above question in normal mode, just instead of calculating a new delayValue to place in the Timer/Counter0 register, the developer would either increment or decrement the OCR0 by some changePerLevel number, so the interrupt is interrupted more or less frequently. This would result in toggling the motor enable pins faster or slower.

```
Set up stack
Set up TCCR0 to be CTC mode and prescale
Set up TIMSK to call ISR on match
Set OCR0 to be highest value (toggled less frequently)

INCSPEED (so ISR is toggled more frequently):
Toggle motor enable pins
Set OCR0 to current value minus some changePerLevel variable
reset timer
```

```
DECSPEED (so ISR is toggled less frequently):
Toggle motor enable pins
Set OCR0 to current value plus some changePerLevel variable
reset timer
```

11 Difficulties

An obstacle I ran into in this lab was the *increase speed* button would sometimes jump two speed levels with one press, and the *jump to halt* button wouldn't turn off the lower four LEDs. After some searching on the internet I found that I should wait for roughly 100ms before running the ISR code.

12 Conclusion

This lab was fun. It ws interesting to see how a microcontroller can control engine speeds, and how to automate that process via interrupts.

13 Source Code

Jeremy_Fischer_Lab7_Sourcecode.asm

```
;***********************************************************
;*
;* Enter Name of file here
;*
;* Enter the description of the program here
;*
;* This is the skeleton file for Lab 7 of ECE 375
;*
;***********************************************************
;*
;*  Author: Jeremy Fischer
;*    Date: Feb. 21 2018
;*
;***********************************************************

.include "m128def.inc" ;Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = R16 ;Multipurpose register
.def speedLevel = R17 ;register that holds the current speed
```

```
.def changePerLevel = R18 ;holds the speed change per level
.def newSpeed = R0 ;holds the newSpeed to put into OCRx

.def   waitcnt = R19 ;wait loop counter
.equ   WTime = 10 ;(wait time in MS) / 10ms.

.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   MovFwd = (1<<EngDirR)|(1<<EngDirL)  ;move forward
.equ    ddrdLayout = 0b11110000 ;upper nibble = output, lower = input
.equ   IncSpeed = 0 ;increment speed input bit
.equ   DecSpeed = 1 ;decrement speed input bit
.equ   FullSpeed = 2 ;full speed input bit
.equ   SlowSpeed = 3 ;slowest speed input bit
;***********************************************************
;* Start of Code Segment
;***********************************************************
.cseg ;beginning of code segment

;***********************************************************
;* Interrupt Vectors
;***********************************************************
.org $0000 ;Beginning of IVs
rjmp  INIT ;Reset interrupt
.org $0002 ;INT0 => pin0, PORTD
rcall INCRSPEED ;run INCSPEED ISR
reti ;return from interrupt
.org $0004 ;INT1 => pin1, PORTD
rcall DECRSPEED ;run DECSPEED ISR
reti ;return from interrupt
.org $0006 ;INT2 => pin2, PORTD
rcall JMPFASTEST ;run JMPFASTEST ISR
reti ;return from interrupt
.org $0008 ;INT3 => pin3, PORTD
rcall JMPSLOWEST ;run JMPSLOWEST ISR
reti ;return from interrupt
.org $0046 ;End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT:
;Initialize Stack Pointer
ldi R16, low(RAMEND)  ;Low Byte of End SRAM Address
```

```
out SPL, R16  ;Write byte to SPL
ldi R16, high(RAMEND)  ;High Byte of End SRAM Address
out SPH, R16  ;Write byte to SPH

;Port Set-up
;Initialize Port B for output
ldi   mpr, $FF ;set all bits, cuz all bits will be used
out   DDRB, mpr          ;Set PortB to be output

;Initialize Port D for input
ldi   mpr, ddrdLayout
out   DDRD, mpr          ;Set PortD to be input
ldi   mpr, (1<<IncSpeed)|(1<<DecSpeed)|(1<<FullSpeed)|(1<<SlowSpeed)
out   PORTD, mpr         ;Set PortD to be pull-up

;Interrupt Masking
;Initialize external interrupts (to trigger on falling edge)
ldi   mpr, (1<<ISC31)|(0<<ISC30)|(1<<ISC21)|(0<<ISC20)|(1<<ISC11)|(0<<ISC10)|(1<<ISC01)|(0<
sts   EICRA, mpr         ;Use sts, EICRA is in extended I/O space

;Set the External Interrupt Mask
ldi   mpr, (1<<INT3)|(1<<INT2)|(1<<INT1)|(1<<INT0) ;setting EIMSKx to 1 enables it to be de
out   EIMSK, mpr

;Counters/Timers
;Configure 8-bit Timer/Counter0
ldi   mpr, 0b01101111 ;Activate Fast PWM mode with toggle. Done by bit 3 and 6 being 1
out   TCCR0, mpr  ;(non-inverting), and set prescaler to 1024. Normal Mode
;Configure 8-bit Timer/Counter2
ldi   mpr, 0b01101101 ;Activate Fast PWM mode with toggle. Done by bit 3 and 6 being 1
out   TCCR2, mpr  ;(non-inverting), and set prescaler to 1024


;Init before Main
;Set tekbot to move forward at slowest speed to start
ldi speedLevel, 0
mov mpr, speedLevel
ori   mpr, MovFwd ;load speed and direction of motor to portB
out   PORTB, mpr ;tell TekBot change speed

;Turn on interrupts
sei



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

```
;* Main Program
;***********************************************************
MAIN:
;Move Robot Forward
mov mpr, speedLevel
ori   mpr, MovFwd ;load speed and direction of motor to portB
out   PORTB, mpr ;tell TekBot change speed

rjmp  MAIN ;infinite loop. End of program.

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


;-----------------------------------------------------------
; 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 equation
;  for the number of clock cycles in the wait loop:
; ((3 * ilcnt + 3) * olcnt + 3) * waitcnt + 13 + call
;-----------------------------------------------------------
WAIT: ;Begin a function with a label
push R18
push R19

OLoop:
ldi R19, 224 ;Load middle-loop count
MLoop:
ldi R18, 237   ;Load inner-loop count
ILoop:
dec R18   ;Decrement inner-loop count
brne Iloop  ;Continue inner-loop
dec R19    ;Decrement middle-loop
brne Mloop  ;Continue middle-loop
dec waitcnt    ;Decrement outer-loop count
brne OLoop  ;Continue outer-loop

pop R19
pop R18
ret  ;Return from subroutine

;-----------------------------------------------------------
; INCSPEED:
; Desc: Increments the speedlevel if it is below 15
```

```
;-----------------------------------------------------------
INCRSPEED:
;Save used variables
push mpr

ldi   waitcnt, WTime
rcall Wait

;Perform Function
cpi speedLevel, 15 ;check if the speedlevel is already at its highest
breq EXITINCR ;if it is then disregard
;I can go up a speed level
inc speedLevel ;increment speedLevel
mul speedLevel, changePerLevel ;get new intensity by (seedLevel * changePerLevel)
out OCR0, newSpeed ;set new speed level (newSpeed = R0 = MUL result)
out OCR2, newSpeed

EXITINCR:
clr mpr
sbr mpr, (1<<IncSpeed)|(1<<DecSpeed)|(1<<FullSpeed)|(1<<SlowSpeed) ;clear out any staged II
out EIFR, mpr ;cancel out any staged interrupts

;Restore used variables
pop mpr
ret ;End a function with RET

;-----------------------------------------------------------
; DECSPEED:
; Desc: Decrements the speedlevel if it is above 0
;-----------------------------------------------------------
DECRSPEED:
;Save used variables
push mpr

ldi   waitcnt, WTime
rcall Wait

;Perform Function
cpi speedLevel, 0 ;check if the speedlevel is already at its lowest
breq EXITDECR ;if it is then disregard
;I can go down a speed level
dec speedLevel ;decrement speedLevel
mul speedLevel, changePerLevel ;get new intensity by (seedLevel * changePerLevel)
out OCR0, newSpeed ;set new speed level (newSpeed = R0 = MUL result)
out OCR2, newSpeed
```

```
EXITDECR:
clr mpr
sbr mpr, (1<<IncSpeed)|(1<<DecSpeed)|(1<<FullSpeed)|(1<<SlowSpeed) ;clear out any staged I
out EIFR, mpr ;cancel out any staged interrupts

;Restore used variables
pop mpr
ret ;End a function with RET
;-----------------------------------------------------------
; JMPFASTEST:
; Desc: Jumps the the highest speed level, 15
;-----------------------------------------------------------
JMPFASTEST:
;Save used variables
push mpr

ldi   waitcnt, WTime
rcall Wait

;Perform Function
ldi speedLevel, 15 ;load fastest speed to speedlevel
mul speedLevel, changePerLevel ;get new intensity by (seedLevel * changePerLevel)
out OCR0, newSpeed ;set new speed level (newSpeed = R0 = MUL result)
out OCR2, newSpeed

clr mpr
sbr mpr, (1<<IncSpeed)|(1<<DecSpeed)|(1<<FullSpeed)|(1<<SlowSpeed) ;clear out any staged I
out EIFR, mpr ;cancel out any staged interrupts

;Restore used variables
pop mpr
ret ;End a function with RET

;-----------------------------------------------------------
; JMPSLOWEST:
; Desc: Jumps the the slowest speed level, 0
;-----------------------------------------------------------
JMPSLOWEST:
;Save used variables
push mpr

ldi   waitcnt, WTime
rcall Wait

;Perform Function
ldi speedLevel, 0 ;load slowest speed to speedlevel
```

```
mul speedLevel, changePerLevel ;get new intensity by (seedLevel * changePerLevel)
out OCR0, newSpeed ;set new speed level (newSpeed = R0 = MUL result)
out OCR2, newSpeed

clr mpr
sbr mpr, (1<<IncSpeed)|(1<<DecSpeed)|(1<<FullSpeed)|(1<<SlowSpeed) ;clear out any staged I
out EIFR, mpr ;cancel out any staged interrupts

;Restore used variables
pop mpr
ret ;End a function with RET
```