ECE 375 Lab 7

Timers

Lab Time: Tuesday 10-12

Miles Van de Wetering
Mathew Popowski

TA Signature

1 Introduction

In this lab, we practiced working with interrupts and timers in order to mimic speed by creating a PWM signal to control an LED's brightness. In this case, very bright meant 'stopped' and very dim meant 'fast'.

2 Internal Register Definitions and Constants

EngEnR, EngEnL, EngDirR, and EngDirL were constants used to address specific bits in order to more easily control the motors. mpr was a general use variable. r16 was defined as mpr, r17 defined as 'speed', and r18 was essentially a constant in register form called 'change'. We set it to 17 and left it.

3 Program Initialization

To initialize our program we directed the stack pointer to the end of the program memory section, allowing us to push and pop from the stack as we change scopes (we did very little of this). We then set up Port B for output and D to listen for input, and started the tekbot moving forward. Lastly, we set up interrupts for the falling edges to turn left and right, set the interrupt mask to enable those interrupts, and turned on the 8 bit timer using PWM. We started the duty cycle at 50

4 Main Program

There was effectively no main (one line infinite loop).

5 SpeedUp

This function incremented speed if it wasn't already at the max, and set the timer to use a new duty cycle calculated from the new speed.

6 SpeedDown

This function decremented speed if it wasn't already at the min, and set the timer to use a new duty cycle calculated from the new speed.

7 SpeedMax

This function set the speed to zero, and the duty cycle to 100

8 SpeedMax

This function set the speed to max, and the duty cycle to 0

9 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 of the new approach include the fact that we are only using one timer, which reduces overhead significantly and frees up hardware for other important tasks. On the other hand, the code would be more complicated and likely use more of the CPU because it's not making use of built in functionality. It would also give you more control over the actual values being written (e.g. you could create an exponential speed curve or something instead of the basic linear mapping). It would also take up more memory because it would just be more 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).

   For this we would need to put in a comparison value (the value at which the timer generates an interrupt and starts over) which would be the appropriate duty cycle just like before. In this case, we would modify OCR0 to be the desired speed (designated as a portion of the clock's max value). Lastly, we set OC0 to toggle each time a match is generated.

10 Difficulties

We had minor difficulties getting things initialized correctly, which were fixed when we decided to use binary constants rather than converting them to hex. Made bits much clearer.

11 Conclusion

This was an interesting lab, working with duty cycles was very cool and a useful way to perform regular timing operations (operations on regular intervals).

12 Source Code

Base Code:

```
;
; Lab7.asm
;
```

```
; Created: 11/8/2016 10:12:25 AM
; Author : Mathew
;***********************************************************
;*
;*
;* This is the skeleton file for Lab 7 of ECE 375
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multipurpose register
.def speed = r17
.def change = r18

.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 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

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

;***********************************************************
;* Interrupt Vectors
;***********************************************************
.org $0000 ; Beginning of IVs
rjmp  INIT ; Reset interrupt

; Set up interrupt vectors for any interrupts being used
.org $0002
rcall SpeedUp
reti
```

```
.org $0004
rcall SpeedDown
reti

.org $0006
rcall SpeedMax
reti

.org $0008
rcall SpeedMin
reti

.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT: ; The initialization routine
; 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, 0b01101000 ; Initialize Port B Data Register
out PORTB, mpr ; so move forward is active

; 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

;init constant-ish thingys
ldi change, 17

; Set the Interrupt Sense Control to falling edge
ldi mpr, $AA
sts EICRA, mpr
; Configure the External Interrupt Mask
ldi mpr, $0f
out EIMSK, mpr
```

```
; Turn on interrupts
; NOTE: This must be the last thing to do in the INIT function

;Initialize TCCR0
ldi mpr, 0b01111001
out TCCR0, mpr
clr mpr
out TCNT0, mpr
ldi mpr, $77
out OCR0, mpr

;Initialize TCCR1
ldi mpr, 0b01111001
out TCCR2, mpr
clr mpr
out TCNT2, mpr
ldi mpr, $77
out OCR2, mpr

ldi speed, 7
sei



;***********************************************************
;* Main Program
;***********************************************************
MAIN: ; The Main program
rjmp MAIN

;***********************************************************
;* Functions and Subroutines
;***********************************************************
SpeedUp:
IN mpr, OCR0
CPI mpr, $00
BREQ doneup
SUB mpr, change
OUT OCR0, mpr
OUT OCR2, mpr
DEC speed
ldi mpr, 0b01100000
add mpr, speed
OUT PORTB, mpr
doneup:
ret
```

```
SpeedDown:
IN mpr, OCR0
CPI mpr, $FF
BREQ donedown
ADD mpr, change
OUT OCR0, mpr
OUT OCR2, mpr
INC speed
ldi mpr, 0b01100000
add mpr, speed
OUT PORTB, mpr
donedown:
ret

SpeedMax:
ldi mpr, $ff
out OCR2, mpr
out OCR0, mpr
ldi speed, 15
ldi mpr, 0b01100000
add mpr, speed
OUT PORTB, mpr
ret

SpeedMin:
ldi mpr, $00
out OCR2, mpr
out OCR0, mpr
ldi speed, 0
ldi mpr, 0b01100000
OUT PORTB, mpr
ret
```