ECE 375 Lab 6

External Interrupts

Lab Time: Thursday 10:00 - 11:50

Jeremy Fischer

TA Signature

1 Introduction

The purpose of this lab was to teach the students how to set up and use interrupts. Specifically, how to use interrupts instead of busy waiting (polling).

2 Internal Register Definitions and Constants

Interrupt vector for INT0 is setup to call the HITRIGHT subroutine and interrupt vector for INT1 is setup to call the HITLEFT subroutine.

Below are the labels I gave definitions in this program. They are mainly definitions which represent bit locations to turn left, turn right, enable left engine, enable right engine, etc.

```
.def mpr = R16
.def waitcnt = R17                                  ;wait loop counter
.def ilcnt = R18                                    ;inner loop counter
.def olcnt = R19                                    ;outer loop counter
.equ WTime = 100                                    ;(wait time in MS) / 10ms. So 1s
.equ WskrR = 0                                      ;right whisker input bit
.equ WskrL = 1                                      ;left whisker input bit
.equ EngEnR = 4                                     ;right engine direction 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 MovBck = $00                                   ;move backward
.equ TurnR = (1<< EngDirL)                          ;turn right
.equ TurnL = (1<< EngDirR)                          ;turn left
.equ Halt = (1<< EngEnR|1 << EngEnL)                ;halt command
```

3 Init()

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

4 Main()

The main program is in an infinite loop which commands the tekbot to move forward. It does so by setting the MovFwd command in PORTB.

5 HITRIGHT()

HITRIGHT first pushes the registers, including SREG, to the stack to ensure the state is restored when the ISR is finished. HITRIGHT then backs up for a second by setting the MovBck command

in PORTB and then calling the Wait subroutine. Then, HITRIGHT instructs the tekbot to turn left for a second by setting the TurnL command in PORTB and then calling the Wait subroutine. Finally, HITRIGHT pops the values off the stack back into their respective registers.

6 HITLEFT()

HITLEFT first pushes the registers, including SREG, to the stack to ensure the state is restored when the ISR is finished. HITLEFT then backs up for a second by setting the MovBck command in PORTB and then calling the Wait subroutine. Then, HITLEFT instructs the tekbot to turn right for a second by setting the TurnR command in PORTB and then calling the Wait subroutine. Finally, HITLEFT pops the values off the stack back into their respective registers.

7 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.

8 Additional Questions

1. *As this lab, Lab 1, and Lab 2 have demonstrated, there are always multiple ways to accomplish the same task when programming (this is especially true for assembly programming). As an engineer, you will need to be able to justify your design choices. You have now seen the BumpBot behavior implemented using two different programming languages (AVR assembly and C), and also using two different methods of receiving external input (polling and interrupts).*

   Explain the benefits and costs of each of these approaches. Some important areas of interest include, but are not limited to: efficiency, speed, cost of context switching, programming time, understandability, etc.

   C: The pros of writing in C is that it is much faster to program and much easier to understand. The downside is that compared to the exact program written in assembly, the compiled binary is roughly twice as big and runs slower.

   Assembly: The pros of writing in assembly is that the code executes much faster than a semantically identical program written in C. Since Assembly is the lowest language to the hardware, the user has more control of timing. Therefore time sensitive programs are often written in Assembly. The cons, obviously, is that Assembly programs are much more complicated to write and harder to understand.

   Polling: Polling is the right choice when the device that you are polling is triggered very often. The downside is that if the device that you are polling needs serviced (nearly) immediately than polling could be a problem, because the program isn't aware of the trigger until it executes the check in the loop again.

   Interrupts: Interrupts are very handy because they don't hold up the processor, and they are serviced immediately when the interrupt is signaled. The downside is that the cost

of context switching is high, due to having to save the state of the program. This means interrupts probably aren't a good choice if the interrupt is expected to be triggered often.

2. *Instead of using the **Wait** function that was provided in **BasicBumpBot.asm**, is it possible to use a timer/counter interrupt to perform the one-second delays that are a part of the BumpBot behavior, while still using external interrupts for the bumpers? Give a reasonable argument either way, and be sure to mention if interrupt priority had any effect on your answer.*

   Yes, it is possible to use a timer/counter interrupt to perform the one-second delays. Essentially what would happen is the developer would move Xms into TCNT0 where X is the number of milliseconds to wait, and then a loop is ran to back up/turn left/turn right until the timer is done. The TOV0 flag is repeatedly tested until it is set indicating Xms has elapsed, at which point the loop finishes.

9 Difficulties

The only difficulty I had was understanding why clearing bits in EIFR didn't work when trying to discard queued interrupts. The book says that if $\text{EIFR}_n = 1$ then interrupt n has been triggered. So, I thought that clearing it would discard the interrupt's signal. It took me awhile to realize that since Port-B is pull-up I had to instead set bits in EIFR not clear them.

10 Conclusion

This lab was straightforward due to us going over it in class, and then it also appearing int the textbook. It was interesting to see the interrupt alternative to this lab compared to the busy waiting option. It's interesting that the interrupt is hardware driven not software driven like most of us CS students are use to.

11 Source Code

Jeremy_Fischer_Lab5_Sourcecode.asm

```
;***********************************************************
;*
;* Jeremy_Fischer_Lab6_sourcecode.adm
;*
;***********************************************************
;*
;*  Author: Jeremy Fischer
;*    Date: 02/12/2018
;*
;***********************************************************

.include "m128def.inc" ;Include definition file
```

```
;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def  mpr = R16
.def  waitcnt = R17 ;wait loop counter
.def  ilcnt = R18 ;inner loop counter
.def  olcnt = R19 ;outer loop counter
.equ  WTime = 100 ;(wait time in MS) / 10ms. So 1s
.equ  WskrR = 0 ;right whisker input bit
.equ  WskrL = 1 ;left whisker input bit
.equ  EngEnR = 4 ;right engine direction 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  MovBck = $00 ;move backward
.equ  TurnR = (1<<EngDirL) ;turn right
.equ  TurnL = (1<<EngDirR) ;turn left
.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
.org $0002 ;INT0 => pin0, PORTD
rcall HITRIGHT ;run hitright ISR
reti ;return from interrupt
.org $0004 ;INT1 => pin1, PORTD
rcall HITLEFT ;run hitleft ISR
reti ;return from interrupt
.org $0046 ;End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT: ;The initialization routine
;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
```

```
;Initialize Port B for output
ldi   mpr, (1<<EngEnL)|(1<<EngEnR)|(1<<EngDirR)|(1<<EngDirL)
out   DDRB, mpr          ;Set PortB to be output

; Initialize Port D for input
ldi   mpr, (0<<WskrL)|(0<<WskrR)
out   DDRD, mpr          ;Set PortD to be input
ldi   mpr, (1<<WskrL)|(1<<WskrR)
out   PORTD, mpr         ;Set PortD to be pull-up

;Initialize external interrupts (to trigger on falling edge)
ldi   mpr, (1<<ISC01)|(0<<ISC00)|(1<<ISC11)|(0<<ISC10) ;'10' is falling edge
sts   EICRA, mpr         ;Use sts, EICRA is in extended I/O space

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

;Turn on interrupts
sei



;***********************************************************
;* Main Program
;***********************************************************
MAIN: ;The Main program
; Move Robot Forward
ldi   mpr, MovFwd ;load move forward bit sequence
out   PORTB, mpr ;tell TekBot to move forward
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
```

```
OLoop:
ldi olcnt, 224 ;Load middle-loop count
MLoop:
ldi ilcnt, 237   ;Load inner-loop count
ILoop:
dec ilcnt   ;Decrement inner-loop count
brne Iloop  ;Continue inner-loop
dec olcnt    ;Decrement middle-loop
brne Mloop  ;Continue middle-loop
dec waitcnt    ;Decrement outer-loop count
brne OLoop  ;Continue outer-loop
ret  ;Return from subroutine

;-----------------------------------------------------------
; HITLEFT:
; Desc: This function is an ISR that handles the tekbot
; hitting its left trigger. The tekbot backs up
; for one second, then turns right for one second
;-----------------------------------------------------------
HITLEFT: ; Begin a function with a label
;Save variable by pushing them to the stack
push  mpr
push  waitcnt
in    mpr, SREG
push  mpr

; Move Backwards for a second
ldi   mpr, MovBck
out   PORTB, mpr
ldi   waitcnt, WTime
rcall Wait

; Turn left for a second
ldi   mpr, TurnR
out   PORTB, mpr
ldi   waitcnt, WTime
rcall Wait

;clear out any staged INT0 or INT1
sbr mpr, (1<<WskrR)|(1<<WskrL) ;setting bits in flag register to 1 since pull up resistor
out EIFR, mpr ;cancel out any staged interrupts

;Restore variable by popping them from the stack in reverse order
pop   mpr
out   SREG, mpr
pop   waitcnt
```

```
pop   mpr
ret ;return to caller
;-----------------------------------------------------
; HITRIGHT:
; Desc: This function is an ISR that handles the tekbot
; hitting its right trigger. The tekbot backs up
; for one second, then turns left for one second
;-----------------------------------------------------------
HITRIGHT: ; Begin a function with a label

;Save variable by pushing them to the stack
push  mpr
push  waitcnt
in    mpr, SREG
push  mpr

; Move Backwards for a second
ldi   mpr, MovBck
out   PORTB, mpr
ldi   waitcnt, WTime
rcall Wait

; Turn left for a second
ldi   mpr, TurnL
out   PORTB, mpr
ldi   waitcnt, WTime
rcall Wait

;clear out any staged INT0 or INT1
sbr mpr, (1<<WskrR)|(1<<WskrL) ;setting bits in flag register to 1 since pull up resistor
out EIFR, mpr ;cancel out any staged interrupts

;Restore variable by popping them from the stack in reverse order
pop   mpr
out   SREG, mpr
pop   waitcnt
pop   mpr
ret ;return to caller
```