ECE 375 Lab 5

Simple Interrupts

Lab Time: Wednesday 5-7

Sean Rettig

TA Signature

1 Introduction

The purpose of this fifth lab was to familiarize ourselves with the concept of interrupts and implement the basic ”BumpBot” program that we used earlier using interrupts instead of polling.

2 Program Overview

This is a very simple program that implements basic ”BumpBot” functionality by moving forward until one of its two front whiskers is hit, at which point it will reverse for a short time, turn away from the object it hit (left if the right whisker was hit, or right if the left or both whiskers were hit), and resume moving forward indefinitely.

2.1 Interrupt Vector Initialization

First, the INT0 and INT1 interrupts are set to activate the HitRight and HitLeft routines, respectively.

2.2 Initialization Routine

All the initialization routine does is initialize the stack pointer, set up the ports to be used for input (PORTD) and output (PORTB) (allowing the control of the motors and the reading of whisker data, respectively), and initialize the external interrupts INT0 and INT1 (in addition to enabling global interrupts in general).

2.3 Main Routine

The main routine just makes the Tekbot move forward and keeps looping over itself.

2.4 HitRight Routine

The HitRight Routine simply moves backward for 1 second, turns left for 1 second, and starts moving forward again. It accomplishes the timed events by setting movement signals to PORTB and then calling the Wait routine to wait a short amount of time before setting a new movement signal.

2.5 HitLeft Routine

The HitLeft Routine simply moves backward for 1 second, turns right for 1 second, and starts moving forward again. It accomplishes the timed events by setting movement signals to PORTB and then calling the Wait routine to wait a short amount of time before setting a new movement signal.

2.6 Wait Routine

The Wait routine simply runs a long loop during which it does nothing in order to ”wait” a certain amount of time.

3 Conclusion

In this lab, I learned about how to initialize and use basic external interrupts. Additionally, since this lab involved the reading and writing of a lot of individual specific bits, I started learning the value of using equate definitions and bit shifting for addressing particular byte constants (such as for setting I/O port configurations or the interrupt control registers).

4 Source Code

```
;***********************************************************
;*
;* Lab 5
;*
;* Enter the description of the program here
;*
;* This is the skeleton file Lab 5 of ECE 375
;*
;***********************************************************
;*
;*  Author: Sean Rettig
;*    Date: 2015-02-04
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def waitcnt = r17 ; Wait Loop Counter
.def ilcnt = r18 ; Inner Loop Counter
.def olcnt = r19 ; Outer Loop Counter

.equ WTime = 100 ; Time to wait in wait loop

.equ WskrR = 0 ; Right Whisker Input Bit
.equ WskrL = 1 ; Left Whisker Input Bit
.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

; Using the constants from above, create the movement
; commands, Forwards, Backwards, Stop, Turn Left, and Turn Right
```

```
.equ MovFwd = (1<<EngDirR|1<<EngDirL) ; Move Forwards Command
.equ MovBck = $00 ; Move Backwards 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 the interrupt vectors for the interrupts, .i.e
;.org $002E ; Analog Comparator IV
; rcall HandleAC ; Function to handle Interupt request
; reti ; Return from interrupt
.org $0002
rcall HitRight
reti
.org $0004
rcall HitLeft
reti

.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 Stack Pointer
ldi mpr, high(RAMEND)
out SPH, mpr
ldi mpr, low(RAMEND)
out SPL, mpr
; Initialize Port B for output
```

```
ldi mpr, (1<<EngEnL)|(1<<EngEnR)|(1<<EngDirR)|(1<<EngDirL)
out DDRB, mpr ; Set the DDR register for Port B
ldi mpr, $00
out PORTB, mpr ; Set the default output for Port B
; Initialize Port D for input
ldi mpr, (0<<WskrL)|(0<<WskrR)
out DDRD, mpr ; Set the DDR register for Port D
ldi mpr, (1<<WskrL)|(1<<WskrR)
out PORTD, mpr ; Set the Port D to Input with Hi-Z
; Initialize external interrupts
; Set the Interrupt Sense Control to level low
ldi mpr, (0<<ISC01)|(0<<ISC00)|(0<<ISC11)|(0<<ISC10)
sts EICRA, mpr ; Use sts, EICRA in extended I/O space
; Set the External Interrupt Mask
ldi mpr, (1<<INT0)|(1<<INT1)
out EIMSK, mpr
; Turn on interrupts
sei

;------------------------------------------------------------
; Main Program
;------------------------------------------------------------
MAIN: ; The Main program

; Initialize TekBot Foward Movement
ldi mpr, MovFwd ; Load Move Forward Command
out PORTB, mpr ; Send command to motors

rjmp MAIN ; Create an infinite while loop to signify the
; end of the program.

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

;-----------------------------------------------------------------
; Sub: HitRight
; Desc: Handles functionality of the TekBot when the right whisker
; is triggered.
;-----------------------------------------------------------------
HitRight:
push mpr ; Save mpr register
push waitcnt ; Save wait register
in mpr, SREG ; Save program state
push mpr ;
```

```
; Move Backwards for a second
ldi mpr, MovBck ; Load Move Backwards command
out PORTB, mpr ; Send command to port
ldi waitcnt, WTime ; Wait for 1 second
rcall Wait ; Call wait function

; Turn left for a second
ldi mpr, TurnL ; Load Turn Left Command
out PORTB, mpr ; Send command to port
ldi waitcnt, WTime ; Wait for 1 second
rcall Wait ; Call wait function

; Move Forward again
ldi mpr, MovFwd ; Load Move Forwards command
out PORTB, mpr ; Send command to port

pop mpr ; Restore program state
out SREG, mpr ;
pop waitcnt ; Restore wait register
pop mpr ; Restore mpr
ret ; Return from subroutine

;----------------------------------------------------------------
; Sub: HitLeft
; Desc: Handles functionality of the TekBot when the left whisker
; is triggered.
;----------------------------------------------------------------
HitLeft:
push mpr ; Save mpr register
push waitcnt ; Save wait register
in mpr, SREG ; Save program state
push mpr ;

; Move Backwards for a second
ldi mpr, MovBck ; Load Move Backwards command
out PORTB, mpr ; Send command to port
ldi waitcnt, WTime ; Wait for 1 second
rcall Wait ; Call wait function

; Turn right for a second
ldi mpr, TurnR ; Load Turn Left Command
out PORTB, mpr ; Send command to port
ldi waitcnt, WTime ; Wait for 1 second
rcall Wait ; Call wait function

; Move Forward again
```

```
ldi mpr, MovFwd ; Load Move Forwards command
out PORTB, mpr ; Send command to port

pop mpr ; Restore program state
out SREG, mpr ;
pop waitcnt ; Restore wait register
pop mpr ; Restore mpr
ret ; Return from subroutine

;----------------------------------------------------------------
; 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
;----------------------------------------------------------------
Wait:
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, 237 ; 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
```

5 Challenge Source Code

```
;***********************************************************
;*
;* Lab 5
;*
;* Enter the description of the program here
;*
;* This is the skeleton file Lab 5 of ECE 375
```

```
;*
;***********************************************************
;*
;*  Author: Sean Rettig
;*    Date: 2015-02-04
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def waitcnt = r17 ; Wait Loop Counter
.def ilcnt = r18 ; Inner Loop Counter
.def olcnt = r19 ; Outer Loop Counter

; if hitlast = 1, right was hit last
; if hitlast = 2, left was hit last
.def    hitlast = r20           ; var for loop detection
; once hitcount = 5, turn more
.def    hitcount = r21          ; var for loop detection
.def turndouble = r22

.equ WTime = 100 ; Time to wait in wait loop

.equ WskrR = 0 ; Right Whisker Input Bit
.equ WskrL = 1 ; Left Whisker Input Bit
.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

; Using the constants from above, create the movement
; commands, Forwards, Backwards, Stop, Turn Left, and Turn Right

.equ MovFwd = (1<<EngDirR|1<<EngDirL) ; Move Forwards Command
.equ MovBck = $00 ; Move Backwards 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 the interrupt vectors for the interrupts, .i.e
;.org $002E ; Analog Comparator IV
; rcall HandleAC ; Function to handle Interupt request
; reti ; Return from interrupt
.org $0002
rcall HitRight
reti
.org $0004
rcall HitLeft
reti

.org $0046 ; End of Interrupt Vectors

;-------------------------------------------------------------
; Program Initialization
;-------------------------------------------------------------
INIT: ; The initialization routine

; Initialize Stack Pointer
ldi mpr, high(RAMEND)
out SPH, mpr
ldi mpr, low(RAMEND)
out SPL, mpr

        ; set variables
        ldi hitlast, 0
        ldi hitcount, 0
ldi turndouble, 0

; Initialize Port B for output
ldi mpr, (1<<EngEnL)|(1<<EngEnR)|(1<<EngDirR)|(1<<EngDirL)
out DDRB, mpr ; Set the DDR register for Port B
ldi mpr, $00
out PORTB, mpr ; Set the default output for Port B
; Initialize Port D for input
ldi mpr, (0<<WskrL)|(0<<WskrR)
out DDRD, mpr ; Set the DDR register for Port D
ldi mpr, (1<<WskrL)|(1<<WskrR)
```

```
out PORTD, mpr ; Set the Port D to Input with Hi-Z
; Initialize external interrupts
; Set the Interrupt Sense Control to level low
ldi mpr, (0<<ISC01)|(0<<ISC00)|(0<<ISC11)|(0<<ISC10)
sts EICRA, mpr ; Use sts, EICRA in extended I/O space
; Set the External Interrupt Mask
ldi mpr, (1<<INT0)|(1<<INT1)
out EIMSK, mpr
; Turn on interrupts
sei

;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN: ; The Main program

; Initialize TekBot Foward Movement
ldi mpr, MovFwd ; Load Move Forward Command
out PORTB, mpr ; Send command to motors

rjmp MAIN ; Create an infinite while loop to signify the
; end of the program.

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

;----------------------------------------------------------------
; Sub: HitRight
; Desc: Handles functionality of the TekBot when the right whisker
; is triggered.
;----------------------------------------------------------------
HitRight:
push mpr ; Save mpr register
push waitcnt ; Save wait register
in mpr, SREG ; Save program state
push mpr ;

        cpi     hitlast, 1
        brne    SkipR1      ; If we didn't last hit on the right, skip
        ldi     hitcount, 1 ; If we did, reset the hit count
ldi turndouble, 1

SkipR1:

        cpi     hitlast, 2
        brne    SkipR2      ; If we didn’t last hit on the left, skip
        inc     hitcount    ; If we did, increment the hit count

SkipR2:

        ldi     hitlast, 1

; Move Backwards for a second
ldi mpr, MovBck ; Load Move Backwards command
out PORTB, mpr ; Send command to port
ldi waitcnt, WTime ; Wait for 1 second
rcall Wait ; Call wait function

; Turn left for a second
ldi mpr, TurnL ; Load Turn Left Command
out PORTB, mpr ; Send command to port
ldi waitcnt, WTime ; Wait for 1 second
rcall Wait ; Call wait function
        cpi     hitcount, 5
        brne    SkipR3
rcall Wait ; wait again to turn 180 degrees

SkipR3:

cpi     turndouble, 1
        brne    SkipR4
rcall Wait ; wait again to turn 180 degrees
        ldi     hitcount, 0

SkipR4:

; Move Forward again
ldi mpr, MovFwd ; Load Move Forwards command
out PORTB, mpr ; Send command to port

ldi turndouble, 0

pop mpr ; Restore program state
out SREG, mpr ;
pop waitcnt ; Restore wait register
pop mpr ; Restore mpr
ret ; Return from subroutine

;-----------------------------------------------------------------
; Sub: HitLeft
; Desc: Handles functionality of the TekBot when the left whisker
; is triggered.
;------------------------------------------------------------------
HitLeft:
push mpr ; Save mpr register
push waitcnt ; Save wait register
in mpr, SREG ; Save program state
push mpr ;

        cpi     hitlast, 1
        brne    SkipL1      ; If we didn't last hit on the right, skip
        inc     hitcount    ; If we did, increment the hit count

SkipL1:

        cpi     hitlast, 2
        brne    SkipL2      ; If we didn't last hit on the left, skip
        ldi     hitcount, 1 ; If we did, reset the hit count
ldi turndouble, 1

SkipL2:

        ldi     hitlast, 2

; Move Backwards for a second
ldi mpr, MovBck ; Load Move Backwards command
out PORTB, mpr ; Send command to port
ldi waitcnt, WTime ; Wait for 1 second
rcall Wait ; Call wait function

; Turn right for a second
ldi mpr, TurnR ; Load Turn Left Command
out PORTB, mpr ; Send command to port
ldi waitcnt, WTime ; Wait for 1 second
rcall Wait ; Call wait function
        cpi     hitcount, 5
        brne    SkipL3
cpi     turndouble, 1
        brne    SkipL3
rcall Wait ; wait again to turn 180 degrees

SkipL3:

cpi     turndouble, 1
        brne    SkipL4
rcall Wait ; wait again to turn 180 degrees
        ldi     hitcount, 0
```

```
SkipL4:

; Move Forward again
ldi mpr, MovFwd ; Load Move Forwards command
out PORTB, mpr ; Send command to port

ldi turndouble, 0

pop mpr ; Restore program state
out SREG, mpr ;
pop waitcnt ; Restore wait register
pop mpr ; Restore mpr
ret ; Return from subroutine

;----------------------------------------------------------------
; 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
;----------------------------------------------------------------
Wait:
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, 237 ; 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
```