ECE 375 LAB 6

Introduction to AVR Development Tools

Lab Time: Wednesday 5-7

Abhishek Raol

Zhenggan Zheng

INTRODUCTION

In this lab we will be performing the same functions as in Lab 1 and Lab 2 however this time we will be using external interrupts of our ATmega128 microcontroller. When the left or right whisker is hit, our tekbot should react by backing up, turning away then continuing forward. The purpose of this lab is to familiarize ourselves with interrupts.

PROGRAM OVERVIEW

This program most notable functions are the interrupt vector setup, main function, call left and right, hit left and right and flag function. The interrupt vector function assigns interrupt vectors to memory. The main begins by setting the whisker flag to 0 so that the tekbot moves forward. When a whisker is hit, the main function will call a flag function which sets a flag to a value corresponding to which bumper is hit which will call the call right or left functions which simply calls the hit left and right functions which are taken directly from earlier labs.

INTERRUPT VECTORS

The interrupt vectors are mapped to addresses $0002 and $0004 and control the external interrupt requests with values 0 and 1. $0002 corresponds to the HitRight routines while $0004 corresponds to the HitLeft routine. Address $0000 corresponds to the rjmp INIT routine which will reset the interrupt.

MAIN ROUTINE

The main rountines begins by enabling both motors forward. It then checks for flags (which we set), that determine if a whisker was hit. If the flag is set to value 1 or 2 then we break to run the CALLRIGHT or CALLLEFT routine depending on which whisker was hit. The CALLLEFT and CALLRIGHT call HItRight and HitLeft and then jump back to main where the flag will be reset to 0 and the tekbot can continue forward. Main also calls FLAG routines to set the flag when the whiskers are hit.

HITRIGHT ROUTINE

(The Hit Right routine is the same as what was used in lab1)

The Hit Right routine first moves the TekBot backwards for roughly 1 second by first sending the Move Backwards command to PORTB followed by a call to the Wait routine. Upon returning from the Wait routine, the Turn Left command is sent to PORTB to get the TekBot to turn left and then another call to the Wait routine to have the TekBot turn left for roughly another second. Finally, the HitRight Routine sends a Move Forward command to PORTB to get the TekBot moving forward and then returns from the routine.

HITLEFT ROUTINE

(The Hit Left routine is the same as what was used in lab1)

The HitLeft routine is identical to the HitRight routine, except that a Turn Right command is sent to PORTB instead. This then fills the requirement for the basic BumpBot behavior.

WAIT ROUTINE

(Same as lab1)

The Wait routine requires a single argument provided in the *waitcnt* register. A triple-nested loop will provide busy cycles as such that $16 + 159975 \cdot waitcnt$ cycles will be executed, or roughly $waitcnt \cdot 10\text{ms}$. In order to use this routine, first the *waitcnt* register must be loaded with the number of 10ms intervals, i.e. for one second, the *waitcnt* must contain a value of 100. Then a call to the routine will perform the precision wait cycle.

FLAG ROUTINES

These routines are called by main and set the flag to 1 if the right whisker is hit and set the flag to 2 if the left whisker is hit. The flag routine resets to 0 in main.

CALL(RIGHT/LEFT) ROUTINE

Simply calls HitLeft or HitRight depending on flag value and once the hit routines are finished, jumps back to main.

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.**

   The Polling method was least efficient since it checks for external input by continuously checking in a loop and will continue to loop even if no inputs will be registered for a long time. In lab1, using C was the simplest way to accomplish the desired behavior because of C's brevity and familiarity compared to assembly however we have slightly less control with timing and controlling I/O ports when we want to. Using interrupts as we did in this lab is the most efficient since the infinite loops is only responsible for moving the tekbot forward and waits for a change in the value of the flag to change behavior rather than continuously checking itself.

2. **Imagine that you were required to implement a wait loop inside the external interrupt ISRs, instead of outside the ISRs like you had to do for this lab. Putting aside the fact that this would not be a good design, would it be possible to use a timer/counter interrupt to perform this wait loop? Give a reasonable argument either way, and be sure to mention if interrupt priority had any effect on your answer.**

   Using a timer/counter interrupt would not be the best solution because we would have to measure the duration of a low or high level pulse and call an interrupt and a certain level and this would inherently give this method lower priority than having external interrupt requests.

DIFFICULTIES

We were stuck in a loop where the CALLRIGHT would be run continuously. This is because we forgot to reset our flag to a value of 0 since the cpi function doesn't change the value of the flag. Since the flag was stuck with a value of 1, it would keep running the CALLLRIGHT function and never move forward again. We fixed this by setting flag to 0 at the beginning of main.

CONCLUSION

The conclusion should sum up the report along with maybe a personal though on the lab. For example, in this lab, we were simply required to set up an AVRStudio4 project with an example program, compile this project and then download it onto our TekBot bases. The result of this program allowed the TekBot to behave in a BumpBot fashion. The lab was great and allowed us the time to build the TekBot with the AVR board and learn the software for this lab.

SOURCE CODE

```
;***********************************************************
;*
;*      AssemblerApplication6.asm
;*
;*      Interrupt implementation of BumpBot
;*
;*      This is the skeleton file for Lab 6 of ECE 375
;*
;***********************************************************
;*
;*       Author: Zhenggan Zheng and Abhishek Raol
;*         Date: 2/16/2016
;*
;***********************************************************

.include "m128def.inc"              ; Include definition file

;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                           ; Multipurpose register
.def    waitcnt = r17                ; Wait loop counter
.def    ilcnt = r18                         ; Inner loop counter
.def    olcnt = r19                         ; Outer loop counter
.def    flag = r24                          ; Flag for calling hit functions
; Constants for interactions such as
.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
.equ    MovFwd = (1<<EngDirR|1<<EngDirL)    ; Move forward command
.equ    MovBck = $00                                    ; Move back command
.equ    TurnR = (1<<EngDirL)                            ; Turn right command
.equ    TurnL = (1<<EngDirR)                            ; Turn left command
.equ    Halt = (1<<EngEnR|1<<EngEnL)        ; Stop 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                  ; Call HitRight
               reti                                    ; Return from interrupt
.org $0004                                              ; INT1 => pin1, PORTD
               rcall HitLeft                   ; Call HitLeft
               reti                                    ; Return from interrupt
.org    $0046                                   ; End of Interrupt Vectors

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

               ; Initialize Port B for output
               ldi             mpr, (1<<EngEnL)|(1<<EngEnR)|(1<<EngDirR)|(1<<EngDirL)
               out             DDRB, mpr
               ; Initialize Port D for input
               ldi             mpr, (0<<WskrL)|(0<<WskrR)
               out             DDRD, mpr
               ldi             mpr, (1<<WskrL)|(1<<WskrR)
               out             PORTD, mpr
               ; Initialize external interrupts to trigger on falling edge
               ldi mpr, (1<<ISC01)|(0<<ISC00)|(1<<ISC11)|(0<<ISC10)
               sts EICRA, mpr
               ; Set external interrupt mask
               ldi mpr, (1<<INT0)|(1<<INT1)
               out EIMSK, mpr

               sei


               ; Initialize external interrupts
               ; Set the Interrupt Sense Control to falling edge
               ; NOTE: To be safe, initialize both EICRA and EICRB

               ; Configure the External Interrupt Mask

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

;***********************************************************
;*      Main Program
;***********************************************************
MAIN:   ; The Main program
               ldi flag, $00  ; Load 0 into flag
               cpi     flag, $01       ; See if flag has 1
               breq CALLRIGHT ; If it does, go to CALLRIGHT
               cpi flag, $02  ; See if flag has 2
               breq CALLLEFT  ; If it does, go to CALLLEFT
               ldi mpr, MovFwd         ; Load MovFwd command
               out PORTB, mpr ; Ouput MovFwd to PORTB
               in mpr, PIND   ; Takes input from PIND
               com mpr                 ; Complements it since TekBot is active low
               andi mpr, (1<<WskrL)|(1<<WskrR)       ;Mask out other bits
               cpi mpr, (1<<WskrR)     ; See if right whisker is hit
               breq FLAG1      ; If it is, go to FLAG1
               cpi mpr, (1<<WskrL)     ; See if left whisker is hit
               breq FLAG2      ; If it is go to FLAG2
               rjmp MAIN       ; Infinite loop
FLAG1:
               ldi flag, $01 ; Set flag=1
               rjmp MAIN ; Goes back to MAIN
FLAG2:
               ldi flag, $02; Set flag=2
               rjmp MAIN; Goes back to MAIN
CALLRIGHT:
```

```
                rcall HitRight; Calls HitRight
                rjmp MAIN; Goes back to MAIN
CALLLEFT:
                rcall HitLeft; Calls HitLeft
                rjmp MAIN; Goes back to MAIN
                ; TODO: ???

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

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

;-----------------------------------------------------------
;       You will probably want several functions, one to handle the
;       left whisker interrupt, one to handle the right whisker
;       interrupt, and maybe a wait function
;-----------------------------------------------------------

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
HitRight:       ; Begin a function with a label

                push mpr        ; Save mpr register
                push waitcnt; Save wait register
                in mpr, SREG; save program state
                push mpr

                ldi mpr, MovBck; Load MovBck command
                out PORTB, mpr ; Output MovBck command to PORTB
                ldi waitcnt, WTime; Load wait for 1 second
                rcall Wait              ; Call Wait function

                ldi mpr, TurnL ; Load TurnL command
                out PORTB, mpr ; Output TurnL command to PORTB
                ldi waitcnt, WTime; Load wait for 1 second
                rcall Wait                      ; Call Wait function

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

HitLeft:

                push mpr        ; Save mpr register
                push waitcnt; Save wait register
                in mpr, SREG; Save program state
                push mpr

                ldi mpr, MovBck; Load MovBck command
                out PORTB, mpr; Output MovBck command to PORTB
                ldi waitcnt, WTime; Load wait for 1 second
                rcall Wait              ; Call Wait function

                ldi mpr, TurnR ; Load TurnR command
                out PORTB, mpr ; Output TurnR command to PORTB
                ldi waitcnt, WTime; Load Wait for 1 second
                rcall Wait              ; Call Wait function

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

```
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

;***********************************************************
;*      Stored Program Data
;***********************************************************

; Enter any stored data you might need here

;***********************************************************
;*      Additional Program Includes
;***********************************************************
; There are no additional file includes for this program
```