ECE 375 LAB 6

Introduction to AVR Development Tools

Lab Time: Friday 2-4

Hao Truong

INTRODUCTION

The purpose of this lab is to understand the fundamental of external interrupts such as when interrupts are used and how they are used. We will also learn about some interrupt facilities that are available on the ATmega128 microcontroller and how to configure and enable specific interrupts on the mega128 microcontroller board. In previous labs, polling was used to detect whisker inputs, but this time we will use external interrupts 0, 1, 2, and 3 to detect whisker inputs on a falling edge of the clock.

PROGRAM OVERVIEW

The External Interrupts program provides the basic behavior that allows the TekBot to react to external interrupts. The TekBot will be moving forward until an interrupt is detected by default. If an interrupt is detected, the program will pause whatever it is doing to perform the interrupt service, and then resume the program (moving forward) when the service is done. The program also counts the numbers of left and whisker hits and displays them on the LCD.

Besides the standard INIT and MAIN routines within the program, five additional routines were created and used. The HitRight and HitLeft routines provide the basic functionality for handling either a Right or Left whisker hit and display the number of times the Right or Left whisker is hit respectively. The ClearLeftCounter and ClearRightCounter clear the number of times the Right or Left whisker was hit and display zero on the LCD. The Wait1 routine provide an extremely accurate busy wait, allowing time for the TekBot backup and turn.

INITIALIZATION ROUTINE

The Initialization routine provides. A one-time initialization of key registers that allow the program to execute correctly. The SP is initialized in order to use function and subroutine calls properly. Port D and Port B were initialized as input and output. The LDC Display was initialized for the use of functions inside LCDDriver.asm file. Left whisker counter and right whisker counter were initialized to 0 and displayed on LCD. Finally, the external interrupts were initialized turned on.

MAIN ROUTINE

The Main routine executes a simple loop that makes the TekBot moves forward constantly.

SUBROUTINES

1. HitRight Routine

When interrupt 0 (INT0) is detected, the TekBot moves backwards for a second, then turns away (left) from the object for a second, and finally resume to its initial motion. This routine also displays the number of times the INT0 is triggered on the LCD.

2. HitLeft Routine

When interrupt 1 (INT1) is detected, the TekBot moves backwards for a second, then turns away (right) from the object for a second, and finally resume to its initial motion. This routine also displays the number of times the INT1 is triggered on the LCD.

3. ClearRightCounter

This routine clears the number of times the INT0 was triggered and display 0 on line 1 of the LCD

4. ClearLeftCounter

This routine clears the number of times the INT1 was triggered and display 0 on line 1 of the LCD

5. Wait1

This routine performs a delay of 1 second when it is called.

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.

With assembly, we have complete control of which register is going to be used to store content providing sufficient speed for coders. On the other hand, C takes care of a lot of details behind the scene for us, therefore it is more abstractive when it comes to C. Interrupts can be very beneficial because they allow the program to continue to be executed until something happens. When an interrupt is triggered, the process pauses the program and saves its current state, services the interrupt, and then resume to its state when done.

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.

It is not possible to use a time/counter interrupt to perform the one-second delays that are a part of the BumpBot behavior while still using external interrupts for the bumpers because only one interrupt can be handle at a time. When an interrupt is triggered, let’s say HitRight, this routine will be serviced first while other interrupts will be put in queue. Therefore, the timer/counter interrupt will not be executed until the HitRight routine is done.

CONCLUSION

In this lab, we learned when to use interrupts and how they are used. We also learned how to configure and enable interrupts on the Atmega128 microcontroller board. The interrupts were set to triggered on a falling edge of clock meaning that every time a button was pressed down, an interrupt was triggered. Also, some of the routines were reused and improved from the BasicBumpBot.asm file.

SOURCE CODE

```
;***********************************************************
;*
;*      Hao_Truong_Lab6_sourcecode.asm
;*
;*      Enternal Interrupt
;*
;*      This is the skeleton file for Lab 6 of ECE 375
;*
;***********************************************************
;*
;*       Author: Hao Truong
;*         Date: 11/12/2021
;*
;***********************************************************

.include "m128def.inc"               ; Include definition file

;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                            ; Multipurpose register
.def    waitcnt = r23                ; wait loop counter
.def    ilcnt = r24                          ; Inner Loop Counter
.def    olcnt = r25                          ; Outer Loop Counter

.def    rightcnt = r3
.def    leftcnt = r4


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


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

;***********************************************************
;*      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
; enternal interrupt request 0 (INT0) is located at Program address $0002
.org    $0002
               rcall   HitRight
               reti

; enternal interrupt request 1 (INT1) is located at Program address $0004
.org    $0004
               rcall   HitLeft
               reti

; enternal interrupt request 2 (INT2) is located at Program address $0006
.org    $0006
               rcall ClearRightCounter
               reti

; enternal interrupt request 3 (INT3) is located at Program address $0008
.org    $0008
               rcall ClearLeftCounter
               reti

               ; This is just an example:
;.org   $002E                                ; Analog Comparator IV
;              rcall   HandleAC              ; Call function to handle interrupt
;              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 RAM
               ldi                           mpr, high(RAMEND)
               out                           SPH, mpr                      ; load SPH with high
byte of RAM

               ; Initialize Port B for output
               ldi                           mpr, $FF                      ; set Port B Data
Direction Register
               out                           DDRB, mpr                     ; for output
               ldi                           mpr, $00                      ; initialize Port B
Data Register
               out                           PORTB, mpr                    ; so all port B data
outputs are low

               ; 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
Direction Register
               out                   PORTD, mpr                    ; so all Port D inputs are
Tri-State

               ; Initialize LCD Display
               RCALL          LCDInit



               ; Display left counter and right counter on LCD (both counters are initially 0)
               ; initialize right counter
               ldi            mpr, 0                ; load 0 to rightcnt
               ldi            XL, $00                      ; Y <- data memory address of line 1
               ldi            XH, $01
;              st             X, mpr
               ; call Bin2ASCII function
               rcall   Bin2ASCII       ; convert value in ASCII

               ; initialize left counter
;              ldi            mpr, 0b00000000                       ; load 0 to leftcnt
               ldi            XL, $10                      ; Y <- data memory address of line 2
               ldi            XH, $01
;              st             X, mpr
               ; call Bin2ASCII function
               rcall   Bin2ASCII       ; convert value in ASCII
               rcall  LCDWrite

               ; initialize left and right counters
               clr            leftcnt
               clr            rightcnt




               ; Initialize external interrupts
               ; Set the Interrupt Sense Control to falling edge
               ldi                   mpr, 0b10101010               ; ISCn1 is set to 1, ISCn0 is
cleared -> falling edge
               sts                   EICRA, mpr

               ; Configure the External Interrupt Mask
               ldi                   mpr, 0b00001111
               out                  EIMSK, mpr                    ; enable interrupts 0, 1, 2,
and 3

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

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

               ; TODO: ???



               ; The BumpBot initially moves forwards
               ldi            mpr, 0b01100000
               out            PORTB, mpr

               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
;------------------------------------------------------------

;----------------------------------------------------------------
; Sub: HitRight
; Desc: Handles functionality of the TekBot when the right whisker
;            is triggered and displays number of right whikser hit on LCD.
;----------------------------------------------------------------
HitRight:
               push    mpr                                ; Save mpr register
               push    waitcnt                    ; Save wait register
               in             mpr, SREG                   ; Save program state
               push    mpr                                ;
               push    XL                                 ; save YL
               push    XH                                 ; save YH
               push    ZL
               push    ZH



               ldi            mpr, 0
               inc            mpr
               add            rightcnt, mpr
               mov            mpr, rightcnt

               ldi            XL, $00                     ; X <- data memory address of line 1
               ldi            XH, $01

               ; call Bin2ASCII function
               rcall    Bin2ASCII       ; convert value in ASCII

               rcall   LCDWrLn1              ; write both lines onto LCD




               ; Move Backwards for a second
               ldi            mpr, 0b00000000             ; Load Move Backward command
               out            PORTB, mpr                  ; Send command to port
               ldi            waitcnt, WTime        ; Wait for 1 second
               rcall   Wait1                      ; Call wait function
               ; Turn left for a second
               ldi            mpr, 0b00100000               ; Load Turn Left Command
               out            PORTB, mpr                    ; Send command to port
               ldi            waitcnt, WTime        ; Wait for 1 second
               rcall  Wait1                         ; Call wait function

               ; Move Forward again
               ldi            mpr, 0b01100000               ; Load Move Forward command
               out            PORTB, mpr                    ; Send command to port

               ; Avoid queued interrupts by writing 1 to EIFR
               ldi            mpr, 0b00001111
               out            EIFR, mpr


               pop            ZH
               pop            ZL
               pop            XH
               pop            XL
               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 and displays number of left whisker hit on LCD.
;----------------------------------------------------------------
HitLeft:
               push   mpr                                   ; Save mpr register
               push   waitcnt                       ; Save wait register
               in             mpr, SREG                     ; Save program state
               push   mpr                                   ;
               push   XL                                    ; save YL
               push   XH                                    ; save YH


               ldi            mpr, 0
               inc            mpr
               add            leftcnt, mpr
               mov            mpr, leftcnt

\

               ldi            XL, $10                       ; X <- data memory address of line 2
               ldi            XH, $01

               ; call Bin2ASCII function
               rcall   Bin2ASCII      ; convert value in ASCII

               rcall  LCDWrLn2              ; write both lines onto LCD
               mov            leftcnt, mpr

               ; Move Backwards for a second
               ldi            mpr, 0b00000000               ; Load Move Backward command
               out            PORTB, mpr                    ; Send command to port
               ldi            waitcnt, WTime        ; Wait for 1 second
               rcall  Wait1                         ; Call wait function

               ; Turn right for a second
               ldi            mpr, 0b01000000               ; Load Turn Left Command
               out            PORTB, mpr                    ; Send command to port
               ldi            waitcnt, WTime        ; Wait for 1 second
               rcall  Wait1                         ; Call wait function

               ; Move Forward again
               ldi            mpr, 0b00000000               ; Load Move Forward command
               out            PORTB, mpr                    ; Send command to port
               ; Avoid queued interrupts by writing 1 to EIFR
               ldi            mpr, 0b00001111
               out            EIFR, mpr


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


;--------------------------------------------------------------
; Sub: ClearRightCounter
; Desc: Clear right whisker counter and display 0 to line 1 of LCD
;--------------------------------------------------------------
ClearRightCounter:
               push   mpr
               push   XL
               push   XH



               clr            rightcnt
               mov            mpr, rightcnt

               ldi            XL, $00                      ; X <- data memory address of line 1
               ldi            xH, $01

               ; call Bin2ASCII function
               rcall   Bin2ASCII      ; convert value in ASCII

               rcall  LCDWrLn1              ; write both lines onto LCD

               ; Avoid queued interrupts by writing 1 to EIFR
               ldi            mpr, 0b00001111
               out            EIFR, mpr


               pop            XH
               pop            XL
               pop            mpr


;--------------------------------------------------------------
; Sub: ClearRightCounter
; Desc: Clear left whisker counter and display 0 to line 2 of LCD
;--------------------------------------------------------------
ClearLeftCounter:
               push   mpr
               push   XL
               push   XH


               clr            leftcnt               ; clear rightcnt
               mov            mpr, leftcnt


               ldi            XL, $10                      ; X <- data memory address of line 1
               ldi            XH, $01

               ; call Bin2ASCII function
               rcall   Bin2ASCII      ; convert value in ASCII

               rcall  LCDWrLn2              ; write both lines onto LCD

               ; Avoid queued interrupts by writing 1 to EIFR
               ldi            mpr, 0b00001111
               out            EIFR, mpr


               pop            XH
               pop            XL
               pop            mpr

;----------------------------------------------------------------
; 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
;----------------------------------------------------------------
Wait1:
               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

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

               ; Save variable by pushing them to the stack

               ; Execute the function here

               ; Restore variable by popping them from the stack in reverse order

               ret                                          ; End a function with RET

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

; Enter any stored data you might need here

;***********************************************************
;*     Data memory allocation
;***********************************************************


;***********************************************************
;*     Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"             ; Include the LCD Driver
```