ECE 375 Lab 5

External Interrupts

Lab Time: Friday 2 PM - 3:50 PM

Student 1: Winnie Woo
Student 2: Joseph Borisch

TA Signature

1 Introduction

The purpose of this lab was to understand the difference in communication protocols, specifically polling and external interrupts. The BumpBot program from the first lab implemented polling but this lab is understanding how to use interrupts to improve the previous program. Additionally, the LCD display is used in this lab in order to display a counter for the BumpBot routine.

2 Program Overview

This program provides the basic behavior to allow the Tekbot to react to whisker input, with the exception of using interrupts to communicate. The Tekbot moves forward by default until either one of the left or right whiskers are triggered. The LCD display will display how count how many times each whisker is triggered.
Besides the standard INIT and MAIN routines, there are three additional routines. The HitRight and HitLeft handles whether the right or left whisker was triggered. There is also a HandleClear routine to clear the LCD display.

3 Initialization Routine

The INIT routine initializes the stack pointer, PORT B and PORT D for output. We are also initializing the LCD display and the interrupt sense control (EICRA) trigger state to falling edge and configuring the external interrupt mask (EIMSK) to enable INT0, INT1 and INT3 and lastly turn on the global interrupt.

4 Main Routine

The Main routine executes an infinite move forward command for the Tekbot.

5 HitRight

The HitRight routine when triggered moves the Tekbot backwards for 1 second and turns left before resuming forward motion, with the addition of clearing the queued interrupts.

6 HitLeft

The HitLeft routine is similar to the HitRight routine with the exception of the Tekbot turning right if triggered.

7 Wait

The Wait routine requires a single argument provided in the waitcnt register. A triple-nested loop will provide busy cycles as such that cycles will be executed, or roughly. 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.

8 HandleClear

This routine clears the counters for the left and right whisker hits.

9 CountHitRight

This function controls the counter for how many times the HitRight function has been executed. Each time the right whisker is triggered, this function is called from within HitRight. This function also handles the LCD display for line 1, which is the counter for the right whisker.

10 CountHitLeft

This function controls the counter for how many times the HitLeft function has been executed. Each time the left whisker is triggered, this function is called from within HitLeft. This function also handles the LCD display for line 2, which is the counter for the left whisker.

11 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.

   To begin, there are several advantages and disadvantages between polling based operation and interrupt based operation. Both have their own unique characteristics that set them apart and help them apply to different scenarios. For instance, polling is good for simple program's that do not need to carry out more than one action. Interrupts, on the other hand, should be used when a program requires paying attention to a few tasks at a time. Polling is also an inefficient use of a resources if the an event isn't expected to occur frequently. Looking at programming in C or assembly, there are several things to keep in mind. To start, writing in a higher assembly language, such as C, allows for more complex instructions to be carried out in fewer lines of code. This can speed up programming times by making programs physically shorter, and is usually simpler to understand. However, these higher languages can also make assumptions about hardware that cause frustrating compile errors that could be avoided had the program been written in assembly. Assembly may not be as intuitive, but it

allows for a more intuitive understanding of what is happening on the hardware and how data is physically being processed. Finally, another thing to keep in mind is the size of actual programs. Typically, higher level program files are much larger than their assembly equivalents. This is due to the fact that higher languages often require additional libraries to be compiled, even if not in use.

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.

   Replacing the wait function within the BumpBot routine with a timer/counter interrupt should give the same results either way. This is because the timer/-counter interrupt is mapped to it's own set of control registers that are separate from the external interrupts. External interrupts are prioritized higher in the program memory, but this should not matter due to the fact that a HitRight or HitLeft routine should never need to be called when the wait counter needs to be called. The wait counter is only called from within these functions, so the priority should not have an effect on the functionality.

12 Difficulties

When initially uploading the code, we forgot to implement the BIN2ASCII function to convert binary to ASCII values so the counter was not displaying on the LCD properly. In addition, there were several compile errors encountered when incorporating the LCD display that were unclear and difficult to fix.

13 Conclusion

In conclusion, implementing the counter for the HitRight and HitLeft took up the majority of the time but the overall lab was a good way of learning how interrupts work.

14 Source Code

```
;***********************************************************
;* This is the skeleton file for Lab 5 of ECE 375
;*
;*  Author: Winnie Woo and Joseph Borisch
;*  Date: 11/7/2022
;*
;***********************************************************
.include "m32U4def.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  rightcnt = r23     ; HitRight counter
.def  leftcnt = r24      ; HitLeft counter
.equ  WskrR = 0          ; Right Whisker Input Bit
.equ  WskrL = 1          ; Left Whisker Input Bit
.equ  WTime = 100        ; Time to wait in wait loop
.equ  EngEnR = 5         ; Right Engine Enable Bit
.equ  EngEnL = 6         ; Left Engine Enable Bit
.equ  EngDirR = 4        ; Right Engine Direction Bit
.equ  EngDirL = 7        ; 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
.org  $0002              ; INT0 => pin0, PORTD
      rcall HitRight     ; Call HitRight subroutine
      reti               ; Return from interrupt
.org  $0004              ; INT1 => pin1, PORTD
      rcall HitLeft       ; Call HitLeft subroutine
      reti               ; Return from interrupt
.org  $0008              ; INT3 => pin3, PORTD
      rcall ClearButton   ; Call ClearButton subroutine
      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, $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 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 Register
      out     PORTD, mpr    ; so all Port D inputs are Tri-State
      ; Initialize LCD Display
      rcall LCDInit      ; Initialize LCD Display
      rcall lcdbacklighton ; backlight on
      clr     rightcnt   ; clear the counter for the right whisker
      call  CountHitRight ; call function to display right whisker counter to LCD
          (initially 0)
      clr     leftcnt       ; clear the counter for the left whisker
      call  CountHitLeft  ; call function to display left whisker counter to LCD
          (initially 0)
      ; Initialize external interrupts
      ; Set the Interrupt Sense Control to falling edge
      ldi     mpr, 0b10001010;
          (1<<ISC01)|(0<<ISC00)|(1<<ISC11)|(0<<ISC10)|(1<<ISC31)|(0<<ISC30) => sets
          trigger state to falling
      sts     EICRA, mpr      ; use sts, EICRA is in extended I\O space
      ; Configure the External Interrupt Mask
      ldi     mpr, 0b00001011; (1<<INT0)|(1<<INT1)|(1<<INT3) => enables INT0, INT1,
          INT3
      out     EIMSK, mpr
      ; Turn on interrupts
      sei                ; NOTE: This must be the last thing to do in the INIT function
;***********************************************************
;* Main Program
;***********************************************************
MAIN:                 ; The Main program
      ldi     mpr, $90   ; Load Move Forward Command (actual command wouldn't compile,
          had to use bianry value instead)
      out     PORTB, mpr    ; Send command to motors
      rjmp  MAIN       ; Create an infinite while loop to signify the
                       ; end of the program.
;***********************************************************
;* Functions and Subroutines
;***********************************************************
;-----------------------------------------------------------
; Func: HitRight
; Desc: If hit, moves backward, turns left, continues forward
;-----------------------------------------------------------
HitRight:
      ; Save variable by pushing them to the stack
      push  mpr          ; Save mpr
      push  waitcnt      ; Save wait register
```

```
	in	mpr, SREG	; Save program state
	push	mpr
	inc	rightcnt	; Increment HitRight counter
	rcall	CountHitRight ; Write updated coutner value to LCD display
	; Move Backwards for a second
	ldi	mpr, MovBck	; Load Move Backward 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, $90	; Load Move Forward command
	out	PORTB, mpr	; Send command to port
	; Clear queued interrupts
	ldi	mpr, $0F	; Cleared by writing a 1 to it
	out	EIFR, mpr
	; Restore variable by popping them from the stack in reverse order
	pop	mpr		; Restore program state
	out	SREG, mpr
	pop	waitcnt		; Restore wait register
	pop	mpr		; Restore mpr
	ret			; End a function with RET
;-----------------------------------------------------------
; Func: HitLeft
; Desc: If hit, moves backward, turns right
;-----------------------------------------------------------
HitLeft:
	; Save variable by pushing them to the stack
	push	mpr		; Save mpr register
	push	waitcnt		; Save wait register
	in	mpr, SREG	; Save program state
	push	mpr
	inc	leftcnt		; Increment HitLeft counter
	rcall	CountHitLeft ; Write updated coutner value to LCD display
	; Move Backwards for a second
	ldi	mpr, MovBck	; Load Move Backward 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, $90   ; Load Move Forward command
     out     PORTB, mpr    ; Send command to port
     ; Clear queued interrupts
     ldi     mpr, $0F   ; Cleared by writing a 1 to it
     out     EIFR, mpr
     ; Restore variable by popping them from the stack in reverse order
     pop     mpr           ; Restore program state
     out     SREG, mpr
     pop     waitcnt       ; Restore wait register
     pop     mpr           ; Restore mpr
     ret                ; Return from subroutine
     ret                ; End a function with RET
;-----------------------------------------------------------
; Func: HandleClear
; Desc: Clears LCD display
;-----------------------------------------------------------
ClearButton:
     ; Save variables by pushing them to the stack
     push mpr
     push waitcnt
     in   mpr, SREG     ;save program state
     push mpr
     ; Execute the function here
     clr     rightcnt
     rcall CountHitRight
     clr     leftcnt
     rcall CountHitLeft
     ldi     mpr, $0F   ; clear queued interrupts
     out     EIFR, mpr
     ; Restore variables by popping them from the stack,
     ; in reverse order
     pop     mpr
     pop     waitcnt
     out     EIFR, mpr
     pop     mpr
     ret        ; End a function with RET
;-----------------------------------------------------------
; Func: CountHitRight
; Desc: Handles functionality of the LCD display when HitRight function is called
   (called within HitRight)
;-----------------------------------------------------------
CountHitRight:
     ; Save variables by pushing them to the stack
     push mpr
     push rightcnt
     push waitcnt       ; Save wait register
     in   mpr, SREG     ; Save program state
     push mpr
     ; Execute the function here
```

```
     rcall LCDClrln1
     mov     mpr, rightcnt ; load to-be converted value into mpr
     ldi     XL, low($0100) ; ; load X with beginning address to be stored (i.e.
        store directly to where LCD will be looking)
     ldi     XH, high($0100) ;
     rcall Bin2ASCII    ; convert value in ASCII
     rcall LCDWrLn1  ; write ASCII value to LCD line 1
     ; Restore variables by popping them from the stack, in reverse order
     pop     mpr
     out     SREG, mpr
     pop     waitcnt
     pop     rightcnt
     pop     mpr
     ret                ; End a function with RET
;-----------------------------------------------------------
; Func: CountHitLeft
; Desc: Handles functionality of the LCD display when HitLeft function is called
   (called within HitLeft)
;-----------------------------------------------------------
CountHitLeft:
     ; Save variables by pushing them to the stack
     push  mpr
     push  leftcnt
     push  waitcnt       ; Save wait register
     in    mpr, SREG     ; Save program state
     push  mpr
     ; Execute the function here
     rcall LCDClrln2    ; clear line 2 of the LCD
     mov     mpr, leftcnt  ; load to-be converted value into mpr
     ldi     XL, low($0110) ; load X with beginning address to be stored (i.e. store
        directly to where LCD will be looking)
     ldi     XH, high($0110) ;
     rcall Bin2ASCII    ; convert value in ASCII
     rcall LCDWrLn2  ; write ASCII value to LCD line 2
     ; Restore variables by popping them from the stack, in reverse order
     pop     mpr
     out     SREG, mpr
     pop     waitcnt
     pop     rightcnt
     pop     mpr
     ret                ; End a function with RET
;-----------------------------------------------------------------
; Func: 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)-1+4)*olcnt)-1+4)*waitcnt)-1+16
;-----------------------------------------------------------------
```

```
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
;***********************************************************

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