ECE 375 LAB 6

External Interrupts

Lab Time: Wednesday 5-7pm

Rhea Mae Edwards

Jackson Neff

INTRODUCTION

The purpose of this first lab is to provide an introduction on how to use AVRStudio4 software for this course along with connecting the AVR board to the TekBot base. A simple pre-made "BumpBot" program was provided to practice creating a project in AVRStudio4, building the project, and then using the Universal Programmer to download the program onto the AVR board.

PROGRAM OVERVIEW

The BumpBot program provides the basic behavior that allows the TekBot to react to whisker input. The TekBot has two forward facing buttons, or whiskers, a left and a right whisker. By default the TekBot will be moving forward until one of the whiskers are triggered. If the left whisker is hit, then the TekBot will backup and then turn right for a bit, while a right whisker hit will backup and turn left. After the either whisker routine completes, the TekBot resumes its forward motion. These functions are called LeftWhiskerHit and RightWhiskerHit, and are triggered by interrupts, that are thrown when the pins are bumped. As specified in the procedure, the interrupts do not queue, and are not enabled again until the bot has resumed its forward motion.

Interrupt Vectors

There is, of course, the reset interrupt at $0000 that calls init, but there are also handleleftbump and handlerightbump interrupts at $0002 and $0004 respectively. These interrupts simply call their corresponding functions when their pin is triggered.

INITIALIZATION ROUTINE

The initialization routine initializes the stack pointer, port D, port B, and the interrupt control and masking registers to the correct values. SP is initialized to point to RAMEND, as usual. DDRD is written with $00 for input, and PortD is written with $FF for tri-state inputs. DDRB is written with $FF for output, and $00 for low values in all the pins. EICRA is given the value 00001010 so that falling edges trigger interrupts on pins 0 and 1. EIMSK is written with 00000011 to enable interrupts on pins 0 and 1.

MAIN ROUTINE

Main is just assigning 11110000 to portB so that the bot goes forward indefinitely. An interrupts being triggered will cause a branch of program flow away from main, but it will always return.

LeftWhiskerHit

LeftWhiskerHit triggers when pin 1 is hit. It stores SREG, mpr, and waitcount on the stack, and then reverses the robot, turns left briefly, calls the wait() subroutine, and then continues forward.

RightWhiskerHit

RightWhiskerHit is a mirror of LeftWhiskerHit. It triggers when pin 0 is hit. It stores SREG, mpr, and waitcount on the stack, and then reverses the robot, turns right briefly, calls the wait() subroutine, and then continues forward.

Wait

Wait is the function that clears all the interrupt flags by writing $FF to eifr. It also waits for a few thousand clock cycles. It is called by LeftWhiskerHit and RightWhiskerHit and is what avoids the queueing of interrupts.

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 and AVR code are different levels of language. Obviously C is higher level than AVR, and this distinction means several things. It means C is easier to understand at a glance and often easier and faster to write, because it is similar to the written English language in structure and is more intuitive than AVR. However, with this increased usability comes a cost of speed. AVR can do things with bare-bores instructions that get to the heart of the problem directly, while C often ends up making function calls to its own library, which just take more time than AVR code and use up more memory. I personally like AVR better, because it is faster, and while it is hard to understand at first, it provides a deep and comprehensive understanding if you spend the time to examine it.

Between polling and interrupts, polling works best for synchronous, often-executed interrupts. This is because polling checks a certain value at a specific interval to see if a program flow branch should execute. So, if something wants to execute a lot, the program will poll, do a branch, return, and have some time to execute other code before the polling catches the next interval. Interrupts work better for asynchronous interruptions that don't occur too often. Instead of waiting for a specific time, an interrupt can execute at once, so they are better when a fast response time is needed. However, they can queue up (not in our program!) or trigger so rapidly that they stop other processes because all resources are devoted to handling them.

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.

Maybe it can, but just from thinking about the problem, I don't think it's a good idea. If we were using polling still, this would be a viable option because we would know when the branching could potentially occur and time the wait interrupt to coincide with that. However, since we are using external interrupts already, we don't know when they could be triggered. Because of this, we can't match anything up with them based on a timer or a counter. And interrupt priority did not have an effect on my answer, I'm not very familiar with it.

CONCLUSION

So all in all, interrupts seem to be a powerful tool. Our robot now is capable of executing an interrupt routine when either of the whiskers are pressed, but doesn't receive any more interrupts until the current handler has run its course. Personally, I like this lab a lot better than the last one. It required less calculating and more AVR conceptual learning. I found myself loving the Atmel Datasheet for the first time ever, so that's a plus!

SOURCE CODE

```
;***********************************************************
;*
;*      main.asm
;*
;*      Moves a TekBot forward, but when a right or left whisker
;*      is hit, it will cause it to back up for 1 second, turn
;*      away for 1 second, and then move forward again by using
;*      external interrupts to detect a falling edge on either
;*      of the whisker inputs.
;*
;***********************************************************
;*
;*       Author: Rhea Mae Edwards, Jackson Neff
;*         Date: February 21st, 2017
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************

.def    mpr = r16       ; Multipurpose register
.def    waitcnt = r17

.equ    WskrR = 0       ; Right Whisker Input Bit
.equ    WskrL = 1       ; Left Whisker Input Bit
.equ    wTime = 100

.def    ilcnt = r18     ; Inner Loop Counter
.def    olcnt = r19     ; Outer Loop Counter

;***********************************************************
;*      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

                ; This is just an example:

.org    $0002

                rcall   LeftWhiskerHit ; Call function to handle interrupt
                reti    ; Return from interrupt

.org    $0004

                rcall   RightWhiskerHit         ; Call function to handle interrupt
                reti    ; Return from interrupt

.org    $0046   ; End of Interrupt Vectors

;***********************************************************
;*      Program Initialization
;***********************************************************

INIT:   ; The initialization routine
```

```
        ldi     mpr, low(RAMEND)
        out     SPL, mpr
        ldi     mpr, high(RAMEND)
        out     SPH, mpr

        ; Initialize Stack Pointer

        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 B for output

        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 Port D for input

        ; Initialize external interrupts
                ; Set the Interrupt Sense Control to falling edge

        ldi mpr, 0b00001010
        sts EICRA, mpr
        ldi mpr, 0b00000011
        out EIMSK, mpr

        ; Configure the External Interrupt Mask

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

        sei

;***********************************************************
;*      Main Program
;***********************************************************

MAIN:   ; The Main program

        ldi mpr, 0b11110000
        out PORTB, mpr
        rjmp    MAIN

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

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

;-----------------------------------------------------------
; Func: LeftWiskerHit
; Desc: When the left whisker of the TekBot is hit, it will
;               react by backing up for 1 second, turning it away
;               for another second, and continues it by moving
;               forward again by using external interrupts.
;-----------------------------------------------------------

LeftWhiskerHit:

        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, 0b10010000        ; 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, 0b11010000        ; 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, 0b11110000 ; Load Move Forward 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

;-----------------------------------------------------------
; Func: RightWhiskerHit
; Desc: When the right whisker of the TekBot is hit, it will
;               react by backing up for 1 second, turning it away
;               for another second, and continues it by moving
;               forward again by using external interrupts.
;-----------------------------------------------------------

RightWhiskerHit:

        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, 0b10010000        ; 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, 0b10110000        ; 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, 0b11110000 ; Load Move Forward 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

;-----------------------------------------------------------
; Func: Wait
; Desc: It has the TekBot waits, and then clears all queued
;               interrupts.
;-----------------------------------------------------------

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
        ldi mpr, (0xff)
        out eifr, mpr  ; Clears all interrupts, resetting all interrupts flags
        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
```