ECE 375 LAB 5

External Interrupts

Lab Time: Friday 4-6

Arthur Kichatov

Liuqiao Song

INTRODUCTION

The purpose of this lab is to show understanding of how to configure and enable specific interrupts using the ATmega32U4 microcontroller board. Another goal from conducting this lab is to grasp the concepts of external interrupts and be able to comprehend how to operate them as well.

PROGRAM OVERVIEW

The takeaway of this program revolves around the code being written in assembly language and permitting the Bot to move when either of the whiskers has been hit. The external interrupts will be used to identify if the TekBot has had any of the whiskers pressed. When the interrupt has been activated, the ATmega32U4 microcontroller will come to a halt from the task that it was previously doing and by servicing the interrupt

INITIALIZATION ROUTINE

In this routine, the stack pointer is initialized so that the functions and subroutines can be called. The output for Port B and the input for Port D require initialization as well. Initialization for the external interrupts are also set in this section.

MAIN ROUTINE

The TekBot will move forward, and when either left or right whisker is hit the bot will first move back for a second then turn away for another second and continue moving forward. The bot will keep track of the number of times each whisker has been pressed by counting the number of times it has occurred while being displayed on the LCD display (using two counters for both the left and right whisker). The external interrupts must identify a falling edge on either the left or right whisker.

HITLEFT ROUTINE

The HitLeft routine reacts to the INT1 being pushed which causes the Bot to move backwards by transmitting the command to both of the motors. The command involving the turn right will be delivered to both of the motors which will authorize the TekBot to turn right for one second. After all of this, it will increment the counter by one.

HITRIGHT ROUTINE

The Hitright routine reacts to the INT0 being pushed which causes the Bot to move backwards by transmitting the command to both of the motors. The command involving the turn right will be delivered to both of the motors which will authorize the TekBot to turn left for one second. After all of this, it will increment the counter by one.

Clear Routine

The clear routine interrupt will clear the number counter on the screen of the LCD by having it become a zero. This will correlate to INT3 being pushed.

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.

a. External interrupts were the most efficient method when conducting this lab. The benefit of using assembly languages is that the size of the file is half the size of the C file. In addition, using external interrupts allowed to conduct the tasks by waiting for interrupts rather of detecting all the time of weather or not the whiskers are pressed. The benefit of C languages is that it is higher level of programming which has the benefit of having similarity with other languages. This also has the benefit of using libraries that can be used in other languages as well, which also reduces the length of code.

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.

a. It is not possible to use timers or counter interrupts to perform one-second delays in regards of the BumpBot behavior (while using external interrupts). This is because the Atmega32U4 can't service two interrupts in a clock cycle. The interrupts work by prioritizing interrupts in order from low to high.

Difficulties

This lab seemed tricky, especially the external interrupts since this was the first time ive dealt with implementing them. Later in the lab, it became more trivial as the repetition helped.

Conclusion

The intention of lab 5 was to show how to implement the external interruptions along with lab 1 stuff such as HitLeft, HitRight and Clear routines. Another goal is being able to understand how external interruptions operate in assembly language. After this concept was understood, it enhanced our previous knowledge by magnifying the objectives of this lab. Lastly, this lab helped gain a better understanding and familiarity with Microchip studio.

Source Code

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

;*          This is the skeleton file for Lab 5 of ECE 375

;*

;*           Author: Enter your name

;*             Date: Enter Date

;*

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


.include "m32U4def.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


.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 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

rcall HitRight ;relative call the HitRight Function

reti ;Returns back to main


.org $0004 ;INT1

 rcall HitLeft ;relative call the HitLeft Function

reti ;Returns back to main
```

```
.org $0008 ;INT3

rcall Clearboth ;relative call the ClearLeft Function

reti ;Returns back to main


; This is just an example:

;.org       $002E                                       ; Analog Comparator IV

;                     rcall       HandleAC                  ; Call function to handle interrupt

;                     reti                                            ; Return from interrupt


.org        $0056                                       ; End of Interrupt Vectors


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

;*           Program Initialization

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

INIT:                                                                 ; The initialization routine

; Initialize Stack Pointer

ldi mpr, LOW(RAMEND) ;Loading LOW(RAMEND) and storing it into mpr

out SPL, mpr ;Putting mpr into SPL

ldi mpr, HIGH(RAMEND) ;Loading HIGH(RAMEND) and storing it into mpr

out SPH, mpr ;Putting mpr into SPH


ldi r24, 0 ;Sets r24 to 0

ldi r25, 0 ;Sets r25 to 0


; Initialize Port B for output

ldi mpr, (1<<EngEnL)|(1<<EngEnR)|(1<<EngDirR)|(1<<EngDirL); 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, (0<<WskrL)|(0<<WskrR) ; Set Port D Data Direction Register

out DDRD, mpr ; for input

ldi mpr, (1<<WskrL)|(1<<WskrR) ; Initialize Port D Data Register

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


; Initialize external interrupts

; Set the Interrupt Sense Control to falling edge

ldi mpr, 0b10101010 ; Generating the ISC (Interrupt Sense Control)

sts EICRA, mpr ; Setting INT0 to activate on the raising edge


; Configure the External Interrupt Mask

ldi mpr,(1<<INT0)|(1<<INT1)|(1<<INT2)|(1<<INT3) ; Configurating the external interrupts for pins INT0, INT1, INT2,
and INT3

out EIMSK, mpr ; Putting mpr into EIMSK

; Initialize LCD Display

rcall LCDInit ; relative call the LCDInit


ldi mpr, 0 ; loading mpr to 0

ldi XL, low($0100) ; loading address into Xlow

ldi XH, high($0110) ; loading address into Xhigh

rcall Bin2ASCII ; relative calling function Page 6


ldi mpr, 0 ; loading mpr to 0
ldi XL, low($0010) ; loading address into Xlow

ldi XH, high($0101) ; loading address into Xhigh

rcall Bin2ASCII ; relative calling function

rcall LCDWrite ; Calling LCDWrite

SEI

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


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

;*          Main Program

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

MAIN:                                                        ; The Main program


ldi mpr, MovFwd ; loading MovFwd into mpr

out PORTB, mpr ; outputing mpr into PortB


rjmp       MAIN                          ; Create an infinite while loop to signify the

; end of the program.


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

;*          Functions and Subroutines

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

HitRight:

push mpr ; Save mpr register

push waitcnt ; Save wait register

in mpr, SREG ; Save program state

push mpr ; Save mpr register
```

```
; 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 Waitln ; 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 Waitln ; Call wait function


; Move Forward again

ldi mpr, MovFwd ; Load Move Forward command

out PORTB, mpr ; Send command to port

ldi mpr, 0b00001111 ; Chooses ITNT(3-0)

out EIMSK, mpr ; outputs mpr into EIMSK interrupt


; Having the counter implemented

ldi mpr, 0b00001111 ; Chooses ITNT(3-0)

out EIFR, mpr ; Clearing every interrupt to prevent the interrupts from stacking


; Having the counter implemented

inc r24 ; incrementing r24 register

mov mpr, r24 ; moving r24 register into mpr

ldi XL, low($0100) ; loading address into Xlow

ldi XH, high($0110) ; loading address into Xhigh
```

```
rcall Bin2ASCII ; relative calling function


 rjmp FirstEndLoop ; jumping to the FirstEndLoop function

 pop mpr ; Restore program state

 out SREG, mpr ;

 pop waitcnt ; Restore wait register

 pop mpr ; Restore mpr

 ret ; Return from subroutine

FirstEndLoop:

rcall LCDWrite ; Having everything restored

pop mpr ; Restore program state

out SREG, mpr ; Putting mpr into SREG

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 ; Save mpr register


 ; 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 Waitln ; 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 Waitln ; Call wait function


 ; Move Forward again

ldi mpr, MovFwd ; Load Move Forward command

out PORTB, mpr ; Send command to port

ldi mpr, 0b00001111 ;Chooses ITNT(3-0)

out EIMSK, mpr ;outputs mpr into EIMSK interrupt

ldi mpr, 0b00001111 ;Chooses ITNT(3-0)

out EIFR, mpr ; Clearing every interrupt to prevent the interrupts from stacking


; Having the counter implemented

inc r25 ; incrementing r25 register

mov mpr, r25 ; moving r25 register into mpr

 ldi XL, low($0010) ; loading address into Xlow

ldi XH, high($0101) ; loading address into Xhigh

rcall Bin2ASCII ; relative calling function

rjmp SecondEndLoop ; Then jump to the SecondEndLoop


 pop mpr ; Restore program state

 out SREG, mpr ;

 pop waitcnt ; Restore wait register

 pop mpr ; Restore mpr
 ret ; Return from subroutine

SecondEndLoop:
 rcall LCDWrite ; Having everything restored
 pop mpr ; Restore program state
 out SREG, mpr ; Putting mpr into SREG
 pop waitcnt ; Restore wait register
 pop mpr ; Restore mpr

ret ; Return from subroutine

FirstLine:
 lpm mpr, Z+ ; Putting the program memory into mpr
 st Y+, mpr ; Assigning a value that will be in mpr by putting it in the Y address
cp r26, r30 ; Checking to see if the string matches with beg. If it is a match, it will continue in the loop
brne FirstLine ; If the string doesn't match with string name. Then it will branch off
rcall LCDWrLn1 ; Calling the LCDWrLn1
 ldi mpr, 0b00001111
out EIFR, mpr ; Clearing every interrupt to prevent the interrupts from stacking
 ret ; End a function with RET

Clearboth:
 rcall LCDClrLn2 ; Calling the LCDClrLn1
 ldi r25, 0 ; Loading 0 into r25 register
 mov mpr, r25 ; moving r25 register into mpr
 ldi XL, low($0010) ; loading address into Xlow
 ldi XH, high($0101) ; loading address into Xhigh
 rcall LCDClrLn1 ; Calling the LCDClrLn1

 ldi r24, 0 ; Loading 0 into r24 register

 mov mpr, r24 ; moving r24 register into mpr

 ldi XL, low($0100) ; loading address into Xlow

 ldi XH, high($0110) ; loading address into Xhigh


 rcall Bin2ASCII ; relative calling function


SecondLine:

 lpm mpr, Z+ ; Putting the program memory into mpr

 st Y+, mpr ; Assigning a value that will be in mpr by putting it in the Y address

cp r26, r30 ; Checking to see if the string matches with beg. If it is a match, it will continue in the loop

brne SecondLine ; If the string doesn't match with the string name. Then it will branch off

rcall LCDWrLn2 ; Calling the LCDWrLn2

 ldi mpr, 0b00001111

out EIFR, mpr ; Clearing every interrupt to prevent the interrupts from stacking

 ret ; End a function with RET


Waitln:

 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


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

;***********************************************************
;*      Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"

; There are no additional file includes for this program
```