ECE 375 Lab 6

External Interrupts

Lab Time: Wednesday 10a-12n

Robert Detjens

David Headrick

TA Signature

Introduction

In this lab, we re-implemented the Basic Bump Bot program from previous labs using interrupts instead of polling to trigger the `HitRight` and `HitLeft` subroutines from before. In addition, we added LCD functionality to the program. The LCD now displays the number of times the `HitLeft` and `HitRight` functions have been called since the last time the `ClearLeft` and `ClearRight` subroutines were triggered by interrupts. The main routine of this program does nothing, since all interactions are triggered by interrupts and run on-demand.

Program Overview

We setup interrupt vectors for INT0:3. INT0 and INT1 correspond to `HitRight` and `HitLeft` respectively. We also setup INT2 and INT3 to correspond to `ClearRight` and `ClearLeft`. Within the `HitRight/Left` routines, the original routines from previous labs were ran and were adapted to increment a general register corresponding to the subroutine hit count. At the beginning of each routine, the general register was incremented, and the LCD was updated with the new count.

The `ClearLeft/Right` routines simply cleared the general registers corresponding to the respective `HitLeft/Right` counter. Then, the LCD was updated.

Additional Questions

- 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.

  One con to polling is that it takes up processor time while the processor could be doing something else. This slows the speed of the processor. An advantage of polling is that the processor knows the context and doesn't have to be interrupted while doing something else. No context switching needed, thus it doesn't have to be accounted for while programming, and makes the flow of the program easier to understand.

  One con to interrupts is that it halts what the processor was doing to service it's subroutine. An advantage to interrupts is that the processor doesn't have to waste time polling for the state, thus increasing speed. Reading the program from an engineer's view would be harder, because the flow of the program would be less obvious.

  An advantage of C is that it's much more easy to read and can be written faster. C can be compiled to many different architectures, and is widely known. A con of C is that your final generated machine code is up to the mercy of the compiler. The compiler

does the translation of your C code into machine code, and can make good, or bad decisions. In the case that the compiler is generating non-optimal, or incorrect machine code, the engineer would still have to peak into the assembly to fix the issue.

An advantage of AVR assembly, and assembly in general, is that the engineer has complete control over the machine instructions being produced. This comes at the cost of hard to read, complex code. When writing in just assembly, the engineer also doesn't have the assistance of the compiler to them out.

- 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

  We could use the timer, however we would have to re-enable interrupts in our interrupt handlers in order to receive the timer interrupt. The timer interrupt also has a lower priority than the external interrupts do, so we would be at risk of stacking interrupts, which is not desirable for this lab.

Difficulties

Our "whisker" interrupts were implemented and handled correctly, however one of our clear handlers was falling through to the next one, causing both counters to be cleared when only one should have been cleared. This was fixed after we changed the interrupt vector from `jmp ClearLeft` to `rcall ClearLeft; reti`. We aren't sure why this fixed the issue. The inspiration to change this came from the lecture slides.

Conclusion

In conclusion, we have learned the basics behind using interrupts, instead of polling, to interact and respond to outside input. `EIMSK` and `EICRA` were used to initialize INT0:3 with the settings required by the lab.

Source Code

```
;***********************************************************
;*
;* Robert Detjens & David Headrick Lab 6 Source Code
;*
;* Basic Bump Bot, but now with interrupts!
;*
;***********************************************************
;*
;*  Author: Robert Detjens
;*          David Headrick
;*    Date: 11/9/21
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Variable and Constant Declarations
;***********************************************************
.def    mpr = r16                       ; Multi-Purpose Register
.def    waitcnt = r17                   ; WaitFunc Loop Counter
.def    ilcnt = r18                     ; Inner Loop Counter
.def    olcnt = r19                     ; Outer Loop Counter

.def    LW_count = r23
.def    RW_count = r24

.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

;/////////////////////////////////////////////////////////////
;These macros are the values to make the TekBot Move.
;/////////////////////////////////////////////////////////////

.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
  rcall HitRight    ; IRQ0 Handler - right whisker input
  reti
.org $0004
  rcall HitLeft     ; IRQ1 Handler - left whisker input
  reti
.org $0006
  rcall ClearRight  ; IRQ2 Handler - right whisker count clear
  reti
.org $0008
  rcall ClearLeft   ; IRQ3 Handler - left whisker count clear
  reti

.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, $00 ; Initialize Port B for outputs
  out PORTB, mpr ; Port B outputs low
  ldi mpr, $FF ; Set Port B Directional Register
  out DDRB, mpr ; for output

  ; Initialize Port D for input
  ldi mpr, $FF ; Initialize Port D for inputs
  out PORTD, mpr ; with Tri-State
```

```
  ldi mpr, $00 ; Set Port D Directional Register
  out DDRD, mpr ; for inputs

  ; Initialize TekBot Foward Movement
  ldi mpr, MovFwd ; Load Move Foward Command
  out PORTB, mpr ; Send command to motors

  ; Clear registers
  clr   LW_count
  clr   RW_count

  ; Clear LCD memory
  ldi   olcnt,    $20
  ldi   XL,       low(LCD_Line1)
  ldi   XH,       high(LCD_Line1)
  clr   mpr
  Mem_init:
    st      X+,   mpr
    dec     olcnt
    brne    Mem_init

  ; init LCD
  call      LCDInit
  rcall     UpdateLCD

  ; Initialize external interrupts
  ; Set the Interrupt Sense Control to falling edge
  ; Set INT0:3 to be on falling edge
  ldi mpr, 0b10101010
  sts EICRA, mpr

  ; Configure the External Interrupt Mask
  ldi mpr, 0b00001111
  out EIMSK, mpr

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


;***********************************************************
;* Main Program
;***********************************************************
MAIN: ; The Main program
  ; do nothing
```

```
  rjmp      MAIN


;-----------------------------------------------------------
; Func: UpdateLCD
; Desc: Clear the hit count register for left whisker
;-----------------------------------------------------------
UpdateLCD:
  ; convert left count to string in LCD mem
  mov       mpr,  LW_count
  ldi       XL,   LOW(LCD_Line1)
  ldi       XH,   HIGH(LCD_Line1)
  call      Bin2ASCII

  ; convert right count to string in LCD mem
  mov       mpr,  RW_count
  ldi       XL,   LOW(LCD_Line2)
  ldi       XH,   HIGH(LCD_Line2)
  call      Bin2ASCII

  call      LCDWrite

  ret


;-----------------------------------------------------------
; Func: ClearLeft
; Desc: Clear the hit count register for left whisker
;-----------------------------------------------------------
ClearLeft:
  clr   LW_count  ; clear counter register

  rcall   UpdateLCD

  ; clear interrupt
  ldi   mpr,    0b00001111
  out   EIFR,   mpr

  ret

;-----------------------------------------------------------
; Func: ClearRight
; Desc: Clear the hit count register for right whisker
;-----------------------------------------------------------
```

```
ClearRight:
  clr   RW_count  ; clear counter register

  rcall   UpdateLCD

  ; clear interrupt
  ldi   mpr,    0b00001111
  out   EIFR,   mpr

  ret

;-----------------------------------------------------------------
; Sub:  HitRight
; Desc: Handles functionality of the TekBot when the right whisker
;       is triggered.
;-----------------------------------------------------------------
HitRight:
  push        mpr               ; Save mprregister
  push        waitcnt           ; Save waitregister
  in          mpr, SREG         ; Save programstate
  push        mpr               ;

  inc         RW_count          ; increment right whisker hit count
  rcall       UpdateLCD

  ; Move Backwards for a second
  ldi         mpr, MovBck       ; Load Move Backwardcommand
  out         PORTB, mpr        ; Send command toport
  ldi         waitcnt, 100      ; WaitFunc for 1 second
  rcall       WaitFunc             ; Call waitfunction

  ; Turn left for a second
  ldi         mpr, TurnL        ; Load Turn LeftCommand
  out         PORTB, mpr        ; Send command toport
  ldi         waitcnt, 100      ; WaitFunc for 1second
  rcall       WaitFunc             ; Call waitfunction

  ; Move Forward again
  ldi         mpr, MovFwd       ; Load Move Forwardcommand
  out         PORTB, mpr        ; Send command to port
  ldi         waitcnt, 50       ; move forward for 0.5s
  rcall       WaitFunc


  pop         mpr               ; Restore programstate
```

```
  out         SREG, mpr         ;
  pop         waitcnt           ; Restore waitregister
  pop         mpr               ; Restorempr

  ; clear interrupt
  ldi   mpr,    0b00001111
  out   EIFR,   mpr

  ret                           ; Return from interrupt

;-----------------------------------------------------------------
; Sub:  HitLeft
; Desc: Handles functionality of the TekBot when the left whisker
;       is triggered.
;-----------------------------------------------------------------
HitLeft:
  push        mpr               ; Save mprregister
  push        waitcnt           ; Save waitregister
  in          mpr, SREG         ; Save programstate
  push        mpr               ;

  inc         LW_count          ; increment left whisker hit count
  rcall       UpdateLCD

  ; Move Backwards for a second
  ldi         mpr, MovBck       ; Load Move Backward command
  out         PORTB, mpr        ; Send command to port
  ldi         waitcnt, 100      ; WaitFunc for 1 second
  rcall       WaitFunc             ; Call wait function

  ; Turn right for a second
  ldi         mpr, TurnR        ; Load Turn Left Command
  out         PORTB, mpr        ; Send command toport
  ldi         waitcnt, 100      ; WaitFunc for 1second
  rcall       WaitFunc             ; Call waitfunction

  ; Move Forward again
  ldi         mpr, MovFwd       ; Load Move Forward command
  out         PORTB, mpr        ; Send command to port
  ldi         waitcnt, 50       ; move forward for 0.5s
  rcall       WaitFunc

  pop         mpr               ; Restore program state
  out         SREG, mpr         ;
  pop         waitcnt           ; Restore wait register
```

```
  pop         mpr               ; Restorempr

  ; clear interrupt
  ldi   mpr,    0b00001111
  out   EIFR,   mpr

  ret                           ; Return from interrupt

;-----------------------------------------------------------------
; Sub:  WaitFunc
; 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
;-----------------------------------------------------------------
WaitFunc:
  push         waitcnt          ; Save waitregister
  push         ilcnt            ; Save ilcntregister
  push         olcnt            ; Save olcntregister

  Loop:   ldi   olcnt, 224        ; load olcnt register
    OLoop:  ldi   ilcnt, 237        ; load ilcnt register
      ILoop:  dec   ilcnt             ; decrement ilcnt
        brne        ILoop             ; Continue InnerLoop
      dec         olcnt             ; decrementolcnt
    brne        OLoop             ; Continue OuterLoop
    dec         waitcnt           ; Decrementwait
  brne        Loop              ; Continue Funcloop

  pop         olcnt             ; Restore olcntregister
  pop         ilcnt             ; Restore ilcntregister
  pop         waitcnt           ; Restore waitregister
  ret                           ; Return fromsubroutine

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

; Enter any stored data you might need here

;***********************************************************
;* Additional Program Includes
;***********************************************************
```

```
.include "LCDDriver.asm"

;***********************************************************
;* Data Memory Allocation
;***********************************************************

.dseg
.org $0100
LCD_Line1: .byte $10
.org $0110
LCD_Line2: .byte $10
```