ECE 375 Lab 5

External Interrupts

Lab session: 015
Time: 12:00-13:50

Author: Astrid Delestine
Programming partner: Lucas Plastid

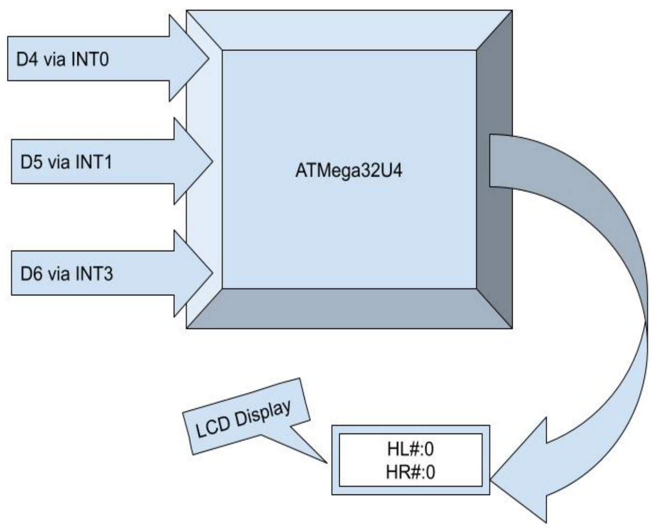
TA Signature

1 Introduction

This is the Fifth lab in the ECE 375 series and it covers using hardware interrupts to preform predescribed "bump bot" operations. Additionally it incorporated use of the LCD Display to show the user how many times the bump bot had been triggered on its left or right side.

2 Design

In this lab Lucas and I setup several different interrupt vectors that were able to trigger certain functions. These functions made the program function similarly to the Lab 1 and 2 bump bot script. Once these interrupts were created and working we moved to creating counters and displays for each of the buttons pressed. In the image seen below, one can see an example of what the LCD display would look like upon boot up.



3 Assembly Overview

As for the Assembly program an overview can be seen below.

3.1 Internal Register Definitions and Constants

Many different constants and registers are assigned in this program, and due to this they will not all be listed. Some more important registers will be highlighted however. The hlcnt and the hrcnt registers were created to count the number of times each button was pressed on either bumper. The strSize is a constant that determines how long the steady state numbers are that need to be patched in every time to the LCD are. All the other values and register assignments are either taken from Lab 1 or Lab 3 and connect to the bump bot script or the LCD scripts.

3.2 Interrupt Vectors

Vectors setup are; hit right on interrupt 0, hit left on interrupt 1, and clear counters on interrupt 3.

3.3 Initialization Routine

Firstly the stack pointer is initialized then ports B and D are initialized for output and input respectively. The LCD is then initialized in its own subroutines as we set it to turn its backlight on and clear any remaining text on the screen. Then we set it such that it displays clear delimiters for each of our button presses. Next we load up the interrupt control for falling edge detection, and configure the interrupt mask for just the 3 interrupts we had setup earlier. Finally we run the sei command to set the interrupt flag in SREG so that the interrupts can work at all.

3.4 Main Routine

The main routine is very simple due to the fact that most operations are handled outside of the main routine by interrupts. All it does is send the move forward command to the LEDs.

3.5 Subroutines

3.6 ClearCounters

This subroutine clears the counters for each button press, then clears the LCD of any overflowing numbers, and resets it back to its initial state. This is done by loading all 16 characters into the data memory that the display looks at for its characters.

3.7 toLCD

The toLCD subroutine is quite simple with regards to what we have already completed. It sets the first four bits of each row to the characters in data memory then uses the built in Bin2ASCII command to take the mpr register and print it to the LCD display. It then enables the LCD to write the characters to the screen.

3.7.1 HitRight

This subroutine is nearly the same as the subroutine built for the original bump bot script. The major changes to it are that it increments a register such that it keeps track of how many times the subroutine is called, and it also calls the toLCD command, allowing the update to be pushed to the LCD. Finally it also has a debounce filter, to disable any interrupts that may have run during the method.

3.7.2 HitLeft

This subroutine is nearly the same as the one built for the original bump bot script. It's major changes are the same as HitRight's, those being the associated counter, the toLCD command and the filter.

3.7.3 Wait

This is the stock wait function. It is unchanged from the original bump bot script.

4 Testing

Tested Each button press and compared to external calculations.

Case	Expected	Actual meet expected
d4	an increment on the LCD, and the bump bot right hit function to be called	✓
d5	an increment on the LCD, and the bump bot left hit function to be called	✓
d6	The two numbers listed on the screen reset	✓
d7	nothing	✓

Table 1: Assembly Testing Cases

5 Study Questions

1. In this lab, you used the Fast PWM mode of 16-bit Timer/Counter, which is only one of many possible ways to implement variable speed on a Tek-Bot. Suppose instead that you used Normal mode, and had it generate an interrupt for every overflow. In the overflow ISR, you manually toggled both Motor Enable pins of the TekBot, and wrote a new value into the Timer/Counter's register. (If you used the correct sequence of values, you would be manually performing PWM.) Give a detailed assessment (in 1-2 paragraphs) of the advantages and disadvantages of this new approach, in comparison to the PWM approach used in this lab.

2. the previous question outlined a way of using a single 16-bit Timer/Counter in Normal mode to implement variable speed. How would you accomplish the same task (variable TekBot speed) using in CTC mode? Provide a rough-draft sketch of the Timer/Counter-related parts of your design, using either a flow chart or some pseudocode (but not actual assembly code)

3. In the next lab, you will be utilizing Timer/Counter1, which can make use of several 16 bit timer registers. The datasheet describes a particular manner in which these registers must be manipulated. To illustrate the process, write a snippet of assembly code that configures OCR1A with a value of 0x1234. For the sake of simplicity, you may assume that no interrupts are triggered during your code's operation.

4. Each ATmega32U4 USART module has two flags used to indicate its current transmitter state: the Data Register Empty (UDRE) flag and Transmit Complete (TXC) flag. What is the difference between these two flags, and which one always gets set first as the transmitter

runs? You will probably need to read about the Data Transmission process in the datasheet (including looking at any relevant USART diagrams) to answer this question.

5. Each ATmega32U4 USART module has one flag used to indicate its current receiver state (not including the error flags). For USART1 specifically, what is the name of this flag, and what is the interrupt vector address for the interrupt associated with this flag? This time, you will probably need to read about Data Reception in the datasheet to answer this question

6 Difficulties

This Lab challenged the thinking power of implementation of ideas we have learned in lecture. It was not too difficult however did require refrenceing both the AVR manual and the atmega32u4 datasheet.

7 Conclusion

In conclusion, this lab introduced and allowed the student to understand many more aspects of how to program a function using interrupts and modify an existing program to work with an LCD in conjunction with those interrupts. This lab proves that the student is learning how to modify certain code structures and is becoming more fluent in the AVR programming scheme

8 Source Code

Listing 1: Assembely Bump Bot Script

```
;***********************************************************
;*
;*   This is the skeleton file for Lab 6 of ECE 375
;*
;*    Author: Astrid Delestine & Lucas Plaisted
;*       Date: 3/1/23
;*
;***********************************************************

.include "m32U4def.inc"          ; Include definition file

;***********************************************************
;*  Internal Register Definitions and Constants
;***********************************************************
.def     mpr = r16               ; Multipurpose register
.def     waitcnt = r17           ; Wait Loop Counter,
                                 ; waitcnt*10ms for delay
.def     ilcnt = r18             ; Inner Loop Counter
.def     olcnt = r19             ; Outer Loop Counter
.def     speeed = r20                ; Speed register, max of 15

;***********************************************************
;*  Start of Code Segment
;***********************************************************
.cseg                            ; beginning of code segment

;***********************************************************
;*  Interrupt Vectors
;***********************************************************
.org     $0000
         rjmp    INIT            ; reset interrupt

         ; place instructions in interrupt vectors here, if needed

.org     $0056                   ; end of interrupt vectors

;***********************************************************
;*  Program Initialization
;***********************************************************
INIT:
         ; Initialize the 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

		; Configure I/O ports
			; 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
		; Configure External Interrupts, if needed
			; Should not need any
		; Configure 16-bit Timer/Counter 1A and 1B
			; TCCRIA Bits:
				; 7:6 - Timer/CounterA compare mode, 10 = non-inverting mode
					; On compare match clears port B pin 5
				; 5:4 - Timer/CounterB compare mode, 10 = non-inverting mode
					; On compare match clears port B pin 6
				; 3:2 - Timer/CounterC compare mode, 00 = disabled
				; 1:0 - Wave gen mode low half, 01 for 8-bit fast pwm
			ldi mpr, 0b10_10_00_01
			sts TCCR1A, mpr
			; TCCRIB Bits:
				; 7:5 - not relevant, 0's
				; 4:3 - Wave gen mode high half, 01 for 8 bit fast pwm
				; 2:0 - Clock selection, 001 = no prescale
			ldi mpr, 0b000_01_001
			sts TCCR1B, mpr
			; Fast PWM, 8-bit mode, no prescaling
				; In inverting Compare Output mode output is cleared on compare
				;

		; Set TekBot to Move Forward (1<<EngDirR|1<<EngDirL) on Port B
			ldi mpr, $F0
			out PINB, mpr
		; Set initial speeed, display on Port B pins 3:0
			ldi speeed, $0F
			rcall WRITESPD
		; Enable global interrupts (if any are used)
			; Not used
		ldi waitcnt, 5	; Set wait timer to be 100ms

```

```
;***********************************************************
;*  Main Program
;***********************************************************
MAIN:
        ; poll Port D pushbuttons (if needed)
        in mpr, PIND
        sbrs mpr, 7 ; Run next command if button 7 presed (active low)
        rcall MAXSPD
        sbrs mpr, 5
        rcall DECSPD
        sbrs mpr, 4
        rcall INCSPD

        rjmp    MAIN            ; return to top of MAIN

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

;-----------------------------------------------------------
; Func: INCSPD
; Desc: Increases the "speeed" of the motor by increasing
;       the width of the pulse. Has built in debouncing.
;       Prevents going over the max speeed.
;-----------------------------------------------------------
INCSPD:
        ; Push to stack
        push mpr

        inc speeed          ; increase the speeed
        sbrc speeed, 5      ; Skip next command if bit 5 is cleared
                            ; If bit 5 is set then we are 16+, 15 is max
        ldi speeed, 15      ; If we are over 15, set speeed to 15
        rcall WRITESPD
INCHOLD:                    ; Don't leave until we let go of the button
        rcall Wait          ; Wait 50ms, debouncing
        in mpr, PIND        ; Grab current button value
        sbrs mpr, 4         ; Check if button is still held
        rjmp INCHOLD        ; Stay in loop if held
        ; Pop from stack
        pop mpr
        ret                         ; End a function with RET

;-----------------------------------------------------------
; Func: DECSPD
; Desc: Decreases the "speeed" of the motor by decreasing
```

```
;         the width of the pulse. Has built in debouncing
;--------------------------------------------------------------
DECSPD:
          ; Push to stack
          push mpr
          cpi speeed, 0    ; If speeed is 0
          breq DECSKIP     ; Don't decrement
          dec speeed
          rcall WRITESPD
          ; Pop from stack
DECHOLD:                   ; Don't leave until we let go of the button
          rcall Wait       ; Wait 50ms, debouncing
          in mpr, PIND     ; Grab current button value
          sbrs mpr, 5      ; Check if button is still held
          rjmp DECHOLD     ; Stay in loop if held
DECSKIP:
          pop mpr
          ret                        ; End a function with RET

;--------------------------------------------------------------
; Func: MAXSPD
; Desc: Increases the "speeed"
;--------------------------------------------------------------
MAXSPD:
          push mpr
          ldi speeed, 15
          rcall WRITESPD
MAXHOLD:                   ; Don't leave until we let go of the button
          rcall Wait       ; Wait 100ms, debouncing
          in mpr, PIND     ; Grab current button value
          sbrs mpr, 7      ; Check if button is still held
          rjmp MAXHOLD     ; Stay in loop if held
          pop mpr
          ret                        ; End a function with RET

;--------------------------------------------------------------
; Func: WRITESPD
; Desc: Sets the timer compares for the current speeed as
;         well as setting the lower nibble of
;--------------------------------------------------------------
WRITESPD:
          push mpr
          ldi mpr, 17      ; 255/15 = 17
          mul speeed, mpr  ; speeed*17 = pulse width
          clr mpr          ; set mpr to 0
          sts OCR1AH, mpr  ; write to high byte of both compares
```

```
        mov mpr, R0       ; place output into mpr. Max 255 = 1 reg
        sts OCR1AL, mpr   ; write to low byte of both compares
        clr mpr
        sts OCR1BH, mpr   ; only done because requried
        mov mpr, R0       ; place output into mpr. Max 255 = 1 reg
        sts OCR1BL, mpr   ; write to low byte of both compares
        ldi mpr, 0b10010000
        add mpr, speeed
        out PINB, mpr
        pop mpr
        ret

        ;-----------------------------------------------------------
; 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)-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
;***********************************************************
        ; Enter any stored data you might need here

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

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