ECE 375 Lab 6

Introduction to AVR Development Tools

**Lab Time: Tuesday 8-10**

Alexander Wood

Zack DeVita

# Introduction

This labs goal was to duplicate the default tekbot behavior implemented in Lab 1, which used polling t determine if a one of the tekbot bumpers was activated. The change in this lab was to use interrupts to implement this behavior. The goal of the lab is to illustrate how interrupts can be used to implement behavior that is an occasional operation. When poling you dedicate CPU time to checking if there is an input at a port, then implementing the desired behavior. This is a waste of CPU processing time, so it is not ideal for this application. Utilizing interrupts allows the CPU to not waste cycles on checking a particular port for inputs. With the use of interrupts, the processor will get a signal to halt operations and to handle the interrupt.

# Program Overview

The lab 6 program is set up to use interrupts to initiate a change in state of the tekbots motors. First the initialize routine is called to set up the stack pointer, which is required for the use of subroutines. Next the interrupt vectors are setup using the .org directive. The hit right vector holds the label of the subroutine to call when the interrupt is triggered. Next the values are set up to configure PORTB (I/O Port) and PORTD for their various functions. This includes configuring the EICRA and EIMSK. Following the initialization routine, the main goes into the standard endless loop, which causes the tekbot to continuously stay in the move forward state. In the main loop, the system also keeps track of how many times the bumpers have been hit to deal with the potential of the robot getting stuck in a loop. During the interrupts, no other interrupts will be recognized or queued for execution until the current interrupt is executed.

## Initialization Routine

The initialization routine starts off by setting up the programs stack pointer. The Value of RAMEND is stored to accomplish this task. The next step is the configuration of the I/O port registers. In this step the data direction register for port B (DDRB) is configured to output data on pins 7 through 4. The next step is to set the value for the port output register. The values are all set to 0. Next port D is configured so that all pins are outputs, and pins 0 and 1 (used for the interrupt signal from the whiskers). The EICRA is set to 00001111. This means that an interrupt detected on interrupt 0 or 1, will be latched on the rising edge. The EIMSK is set to 00000011. This sets the unwanted pins to off, and will only allow an external interrupt from interrupt pin 0 or 1. The values used to track the number of times an interrupt is triggered in a specific time period are finally loaded with their default values. Finally, the SREG flag for allowing interrupts is enabled using the sei AVR instruction.

## Main Routine

The main routine serves the function of returning the tekbot to forward motion. Once the tekbot has commenced forward motion, the main loop then waits to see if an interrupt is initiated. If not interrupt is sent in this time period, the interrupt will be executed. However, if no interrupt is received the program will “clear” its memory of number of times in a row a single bumper is hit, or its memory that detects if it is potentially stuck in a corner. This functionality will be discussed in the Individual hit routines.

## HitRight Routine

The HitRight routine has several features built into it to account for the standard behavior of the tekbot when the right bumper is hit, it also deals with the challenge problem of getting caught in a corner and the other challenge problem which is if the tekbot hits the same whisker twice. Unlike typical sub routine calls, interrupt sub-routine calls for us to temporarily disable interrupts while the routine is running. This is done by writing all 0’s to the EICRA register. Next the values for equal and counter (equal is the counter that indicates two bumper hits on the same side and cntr is used to count the number of times the whisker is bumped) are incremented. Now comes the standard implementation of a subroutine, where the values of the processor state are store in order to execute the interrupt, then return the processor to its original state prior to the interrupt call (including the SREG, as that can be changed in the subroutine). The subroutine is now ready to execute its primary function, which is to send a signal to the tekbot motors to go into reverse. Then the value for a wait time of 1 second is loaded into the loop counter for the wait function. Then the wait function is called to initiate a 1 second delay during which the tekbot will reverse. Now the function hits its conditional branches to determine if there has been alternating whisker hits 5 times in a row, if there has been a right whisker hit twice in a row, or if this is a normal whisker hit event.

First the subroutine will compare the counter value to 5. If the counter equals 5, the program will think it has gotten stuck in a corner. The tekbot will then backup for another three seconds before turning left by 180˚. After the wait cycles it will jump into the final subroutine which send the forward signal to the tekbot.

If the first compare is not true, the subroutine SKIPR will be entered. At this point the equal value will be checked to see if the same whisker has been activated 2 times in a row. If it has not been activated twice in a row the program will jump to the subroutine called JUMPR. The subroutine JUMPR is the standard tekbot behavior of turning left for one second.

Finally, if the comparison in SKIPR sees that the right whisker has been hit twice in a row, the rest of the SKIPR routine will be carried out. This functionality is designed to detect if the tekbot is potentially running into the same object repeatedly. The SKIPR routine will make the tekbot reverse for 2 seconds, then turn left in hopes of avoiding the object.

## HitLeft Routine

The HitLeft routine is identical to the HitRight routine, except that when entering the hit left routine the equal register is decremented. This is how the tekbot know whether or not the whiskers are being hit repeatedly or are being hit in an alternating fashion. This change is also reflected in the SKIPL routine, at which point it is compared to the value 1. This allows the hit left function to use the same register as hit right and still be able to use that value to figure out if the left bumper has been hit twice in a row. The final change in the hit left routine from the hit right, is that when turning the tekbot turns right instead of let.

## Wait Routine

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.

This pre-built wait routine was augmented by creating two extra wait time variables. This is to accomdate the added wait time for the challenge problems.

# Additional Questions

1. In lab one and two the Tekbot’s bump bot behavior was implemented through the process of polling. In lab 6 the goal was to implement the same behavior using interrupts instead of polling. For this particular behavior one could argue either method is an acceptable approach. Polling is an easier method when it comes writing assembly code. The program took lees logic to implement. Being as both behaviors were executed using the same hardware it doesn’t appear that cost would be a factor. The main downside to using polling in a situation like this is that you are using the CPU clock cycles to check if an input which may or may not be there, is there. This is more of an efficiency problem, which could cause complications if the tekbot were to engage in several other tasks, other than just driving around and running into walls. This overhead on the CPU’s available time could slow down other more important functionality of the tekbot.

Using interrupts is a good fit for the behavior of the tekbot. This is because using interrupts allows the processor to do whatever it is programmed to do until a particular event trigger the interrupt. At that point the program would store the processors state and execute the command defined by the interrupt. Once this is complete the tekbots CPU could continue doing whatever tasks it was doing prior to the interrupt. This method is well suited for events that are infrequent or very short in execution time. The use of interrupts does require more coding than the previous method, and the code is a little more complex.

Finally, the use of a high level language like C, over using assembly seems to be much easier. The code is easier for me to follow, as it is more familiar to me. It is also more concise, and takes much less time to code. The downfall of using a high level language is you are totally reliant on a compiler to translate and compile your code into instructions that can be understood by the hardware. This means a programmer has no granular control of the hardware, and has a much harder time determining how long operations will take or the actual hardware usage required to tune the program. This could lead to microcontrollers with more or less power than required for the job, which will result in higher hardware costs then necessary, or nonfunctioning devices.

1. Yes, this behavior would be possible using the internal timers. This implementation would take some work. You would need to make sure that when your whisker interrupts come in, that the SREG interrupt flag gets enabled while you are in the Hit right or Left subroutines. If set up correctly this method shoud work. The reason interrupt priority isn’t an issue, is that the priority is based on several interrupt requests at one time. For example, in this lab the HitRight routine was mapped to IO interrupt 0, and HitLeft to interupt1. If both bumpers are hit at the same time, hit right will take precedence because it has a higher priority. This would not be an issue for the bump bot behavior.

# Difficulties

This lab presented a few challenges, but overall the tasks were pretty straight forward once the interrupt process was understood. The challenge took some time to figure out. Finding an efficient way to determine which behavior was desired took some time.

# Conclusion

In Lab 6 we were to duplicate the simple BumpBot behavior from lab 1, but instead of using polling to detect a whisker getting hit, Lab 6 is to use interrupts. Through the use of the interrupts the program would call one of two subroutines. These subroutines direct the tekbot to reverse and turn away from the object that triggered the interrupt. These subroutines also included the ability to solve the challenge problem of giving the tekbot “memory”. This memory would allow the tekbot to determine if the same bumper was hit twice in a row, which would trigger a longer turn event away from the object. This also allowed the tekbot to determine if it got stuck in a corner, and how to back up and turn 180̐˚. This lab clearly illustrated the how interrupts could be used in the place of polling for bump bot behavior on the tekbot platform.

# Source Code

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* main.asm

;\*

;\* Lab 6 - External Interrupts

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\*  Author: Zachary DeVita

;\*    Date: November 3, 2016

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.include "m128def.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 equal = r20 ; whisker equilibrium

.def cntr = r21 ; Counts right whisker bumps

.def one = r22

;.def two = r23

.def five = r24

.equ WTime = 100 ; Time to wait in wait loop

.equ WTime2 = 200

.equ WTime3 = 150

.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

; Set up interrupt vectors for any interrupts being used

.org $0002 ; {IRQ0 => pin0, PORTD}

rcall HitRight ; Call function to handle interrupt

reti ; Return from interrupt

.org $0004 ; {IRQ1 => pin1, PORTD}

rcall HitLeft ; Call function to handle interrupt

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, $F0 ; 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 external interrupts

; Set the Interrupt Sense Control to falling edge

ldi mpr, $0A ; We don't want the higher interrupts

sts EICRA, mpr

;ldi mpr, $0A

;out EICRB, mpr

; Configure the External Interrupt Mask

ldi mpr, $03

out EIMSK, mpr

ldi one, 1

;ldi two, 2

ldi five, 5

; Turn on interrupts

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

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Main Program

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MAIN: ; The Main program

; Start TekBot moving forward

ldi mpr, MovFwd ; Load Move Foward Command

out PORTB, mpr ; Send command to motors

; If no bumper has been hit for one second after interrupt

; then reset the interrupt count and equalibrium

ldi waitcnt, WTime2 ; Wait for 2 second

rcall Wait ; Call wait function

ldi equal, 3

clr cntr

rjmp MAIN ; Create an infinite while loop to signify the

; end of the program.

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Functions and Subroutines

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

; Sub: HitRight

; Desc: Handles functionality of the TekBot when the right whisker

; is triggered.

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

HitRight:

ldi mpr, $00 ; Temporarily Disable interrupts

sts EICRA, mpr

inc equal

inc cntr

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, 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

cp cntr, five

brne SKIPR

ldi waitcnt, WTime3 ; Wait for 3 seconds

ldi mpr, TurnR ; Load Turn Left Command

out PORTB, mpr ; Send command to port

rcall Wait

rcall Wait

rcall LEAPR

SKIPR:

cp equal, five

brne JUMPR

ldi waitcnt, WTime2 ; Wait for 2 seconds

ldi mpr, TurnR ; Load Turn Left Command

out PORTB, mpr ; Send command to port

rcall Wait

rcall LEAPR

JUMPR:

ldi waitcnt, WTime ; Wait for 1 second

ldi mpr, TurnR ; Load Turn Left Command

out PORTB, mpr ; Send command to port

rcall Wait ; Call wait function

LEAPR:

; Move Forward again

ldi mpr, MovFwd ; 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

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

; Sub: HitLeft

; Desc: Handles functionality of the TekBot when the left whisker

; is triggered.

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

HitLeft:

ldi mpr, $00 ; Temporarily Disable interrupts

sts EICRA, mpr

dec equal

inc cntr

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, 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

cp cntr, five

brne SKIPL

ldi waitcnt, WTime3 ; Wait for 3 seconds

ldi mpr, TurnL ; Load Turn Left Command

out PORTB, mpr ; Send command to port

rcall Wait

rcall Wait

rcall LEAPL

SKIPL:

cp equal, one

brne JUMPL

ldi waitcnt, WTime2 ; Wait for 2 seconds

ldi mpr, TurnL ; Load Turn Left Command

out PORTB, mpr ; Send command to port

rcall Wait

rcall LEAPL

JUMPL:

ldi waitcnt, WTime ; Wait for 1 second

ldi mpr, TurnL ; Load Turn Left Command

out PORTB, mpr ; Send command to port

rcall Wait ; Call wait function

LEAPL:

; Move Forward again

ldi mpr, MovFwd ; 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

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

; 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 + 3) \* olcnt + 3) \* waitcnt + 13 + call

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

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