	\sim $-$	~ :	7 -		\circ
\mathbf{F}	(⊢	~	/5	I AR	8
			, ,	-	

Remotely Operated Vehicle (USART)

Lab Time: Thursday 12-2

Mason Obery Benjamin Leonard

INTRODUCTION

The purpose of this lab is to learn the basics of USART transmission by using the IR light/sensor. The lab uses two TekBot boards, one which transmits a signal and another which receives the signal. When a button is pressed, the transmitter board selects an opcode to send to the board, and transmits the code along with the receiving bots individual address. The receiver functions as a normal BumpBot, but will set its behavior to that of the most recently issued command from the transmitter. This lab will demonstrate and understanding of how to efficiently make use of all the I/O capabilities of the board, including buttons, interrupts, IR USART, timer/counters, and LEDs.

PROGRAM OVERVIEW

There are two programs that were created for this lab: the Receiver code (often abbreviated as RX) and the Transmitter code (often referred to as TX). The transmitter determines if one of the buttons is pressed, and if they have been pressed, to send the bot ID via USART. The program waits until the full signal has been transmitted, and then loads the opcode for the movement indicated. The receiver waits until an receive interrupt has been triggered, which then sets the output LEDs to the correct settings based on the message received. If the robot receives the Send Freeze signal, the TekBot switches the settings of the USART port to Transmit mode, and sends the signal 01010101. If the 01010101 signal is received, the TekBot stops and does not respond to external commands for 5 seconds. The TekBot can survive 3 Freezes until it ceases to function entirely.

For the TX remote, there are no interrupts. The body of the program consists of an initialization routine, followed by a main program which polls PORTD to check if any buttons have been pressed. If buttons have been pressed, it jumps to a function that transmits the data, and uses polling to determine when the transmitter has sent the first bit and is prepared to send another message.

For the RX remote, interrupts are used extensively. The program retains the BumpBot functionality from Lab 2, which means that when External Interrupt1 and External Interrupt0 are triggered, the robot will back up and turn away from the object it bumped into. When the robot receives a signal from the transmitter with the corresponding Bot ID, it will begin executing the command and await another instruction.

TX Initialization Routine

First, the stack pointer is set up so that function calls can be made. PORTD is configured to input on all pins except PIND4, and the pull-up resistor is enabled for all input bits. PORTB is set up to allow for LED output. These LEDs are not part of the assignment, but help to display the opcode that was most recently transmitted to the receiver. For setting up the USART communication, the value 832 is set into the UBBR1 Registers, the double data rate is enabled, the transmitter is enabled, asynchronous mode is enabled, parity is disabled, n-bit data transfer is set to n=8, and 2 stop bits is selected.

RX INITIALIZATION ROUTINE

For the initialization of the RX TekBot, first the stack pointer is set up. PIND3 is set as an output, and the pullup resistor is enabled. PORTB (LEDs) is set up as an output, and the initial output of the PORTB is set to \$00. For setting up the USART communication, the value 832 is set into the UBBR1 Registers, the double data rate is enabled, the receiver mode is enabled, asynchronous mode is enabled, parity is disabled, n-bit data transfer is set to n=8, and 2 stop bits is selected. The number of health points (HP) is initialized to 3. The external interrupts are set to falling edge, the robot is set to MoveForward, and the interrupt is enabled.

TX MAIN ROUTINE

The TX remote program is relatively straightforward. The program polls for button inputs every 100ms. If any of the buttons have been pressed, their respective buttons presses are converted into the opcode pattern. The USART_Transmit function is called twice, first to transmit the specified BotID, then to transmit the opcode. The USART_Transmit function is described in detail below. The program then jumps back to the top of the main function.

TX SUBROUTINES

1. USART Transmit Routine

This function loops until the USART data register is empty, and then loads the next piece of data into the transmit data buffer. This is a polling implementation of USART communication. While an interrupt-based implementation would be more resource-efficient, the polling system is sufficient for the demonstration of concept.

2. Wait Routine

This subroutine is directly taken from the BumpBot code. This function cycles repeatedly to wait roughly waitcnt * 10ms. This is used to prevent button debouncing, which could result in multiple freeze signals sent simultaneously.

RX MAIN ROUTINE

The TX remote program is relatively straightforward. The program polls for button inputs every 100ms. If any of the buttons have been pressed, their respective buttons presses are converted into the opcode pattern. The USART_Transmit function is called twice, first to transmit the specified BotID, then to transmit the opcode. The USART_Transmit function is described in detail below. Additionally, external interrupts 1 and 0 are enabled so that the TekBot will turn if it bumps into an object.

RX SUBROUTINES

1. rCommand Routine

This subroutine is triggered when a message is received by the USART module. When the USART module receives data, it stores the data and clears the buffer. The program checks if the data was equal to the freeze command. If the TekBot has been frozen, it jumps to a portion of the routine that will decrement the total number of health points, wait for 5 seconds, and stops the robot permanently if the robot is out of health points.

If the robot did not receive a freeze command, it checks if it received the BotID. If is equal to the bot ID, it sets register ereg(r17) to \$FF and returns to main. If not, it checks first if ereg=\$FF, then checks if a valid command was transmitted. For all the standard functions (Forward, Left, Right, Stop, Back), the LEDs are updated to the correct configuration. If the 'Issue Freeze' opcode was received, the Tekbot changes the configuration of the USART module to a transmitter, identical to that of the TX program. The freeze command uses the USART_Transmit1 function (polling) to set the freeze command '01010101.' After the transmission has been sent, the USART module is restored to its initial configuration.

2. HitLeft Routine

The function is taken directly from the standard BumpBot code. If the External interrupt 1 is triggered (S2), the robot backs up for 1 second, drives forward for one second with the left motor enabled, and then continues forward.

3. HitRightRoutine

The function is taken directly from the standard BumpBot code. If the External interrupt 0 is triggered (S1), the robot backs up for 1 second, drives forward for one second with the left motor enabled, and then continues forward.

3. Wait Routine

This subroutine is directly taken from the BumpBot code. This function cycles repeatedly to wait roughly waitcnt * 10ms. This is used to prevent button debouncing, which could result in multiple freeze signals sent simultaneously.

4. USART Transmit1 Routine

This function loops until the USART data register Is empty (UDRE1), loads the next piece of data into the transmit data buffer, and then loops until the data buffer is empty again. This is a polling implementation of USART communication. While an interrupt-based implementation would be more resource-efficient, the polling system is sufficient.

DIFFICULTIES

One of the main difficulties we had was correctly implementing the freeze and frozen command because we did not have possession of a third TekBot to test on. Coordination of schedules with three or four people to test the bots is very challenging, especially if not everyone is on the same page. Correctly configuring the timer/counters to work consistently for the challenge code was also rather difficult and took considerably more time than implementing the PWM mode. Most of the trouble encountered in the challenge section was due to unfamiliar hardware and software, as there is a steep learning curve when experimenting with new AVR material. Our understanding of timers/counters and their implementation on the ATMEGA128 has greatly improved after hours of poring over the ATMEGA128 Datasheet in an attempt to correctly configure the control registers.

CONCLUSION

In this lab, we were asked to utilize USART communication to a create a 'remote control' for a TekBot, giving it a unique BotID to ensure communication came from the associated remote, as well as the TekBot itself. We implemented forward, backward, turn left, turn right, halt, speed control, and freeze commands. The lab was an opportunity to learn how to use the USART module and how to utilize the datasheet to correctly configure I/O ports on the microcontroller. Over the course of the multi-week lab, we gained a more in depth understanding of the material covered in class, and this lab helped to demonstrate the importance of good programming techniques and etiquette, especially for larger, collaborative programming projects.

TX Source/Challenge Code

```
; *
           This is the TRANSMIT skeleton file for Lab 8 of ECE 375
     ; *
           Author: Mason Obery and Benjamin Leonard
     ; *
            Date: 2/24/2022 12:23:27 PM
     .include "m128def.inc"
                                ; Include definition file
     Internal Register Definitions and Constants
     .def mpr = r16
                                      ; Multi-Purpose Register
     .def ereg = r17
                                      ;extra register'
     .def
          waitcnt = r23
                                      ; Wait Loop Counter
     .def ilcnt = r24
                                      ; Inner Loop Counter
     .def olcnt = r25
                                ; Outer Loop Counter
     .equ EngEnR = 4
                                      ; Right Engine Enable Bit
     .equ EngEnL = 7
                                      ; Left Engine Enable Bit
     .equ EngDirR = 5
                                      ; Right Engine Direction Bit
                                      ; Left Engine Direction Bit
     .equ
          EngDirL = 6
     ; Use these action codes between the remote and robot
     ; MSB = 1 thus:
     ; control signals are shifted right by one and ORed with 0b100000000 = $80
     .equ MovFwd = ($80|1<<(EngDirR-1)|1<<(EngDirL-1)) ;0b10110000 Move Forward
Action Code
         MovBck = ($80|$00)
     .equ
     ;0b10000000 Move Backward Action Code
                                                            ;0b10100000
     .equ TurnR = (\$80|1 << (EngDirL-1))
Turn Right Action Code
     .equ TurnL = ($80|1 << (EngDirR-1))
                                                            :0b10010000
Turn Left Action Code
     .equ Halt = (\$80|1 << (EngEnR-1)|1 << (EngEnL-1))
                                                ;0b11001000 Halt Action Code
         BotAddress = 0x2B; (Enter your robot's address here (8 bits))
     ;* Start of Code Segment
                                           ; Beginning of code segment
     ; ***********************************
     ; * Interrupt Vectors
     .org $0000
rjmp INIT
                         ; Beginning of IVs
                          ; Reset interrupt
     ;.org UDRE1addr
     ; rcall finishsend
                ret
     .org $0046
                                      ; End of Interrupt Vectors
     ; *
         Program Initialization
     INIT:
     ;Stack Pointer (VERY IMPORTANT!!!!)
     ldi mpr, low(RAMEND)
     out SPL, mpr
     ldi mpr, high (RAMEND)
     out SPH, mpr
     ;I/O Ports
     ldi mpr, 0b0000 1000
     out DDRD, mpr
     ldi mpr, 0b11110111
     out PORTD, mpr
```

```
ldi
           mpr, $FF
out
            DDRB, mpr
            PORTB, mpr
out
ldi mpr, (1<<U2X1)
sts UCSR1A, mpr
ldi mpr, high (832)
sts UBRR1H, mpr
ldi mpr, low(832)
sts UBRR1L, mpr
ldi mpr, ((1<<TXEN1))</pre>
sts UCSR1B, mpr
ldi mpr, (1<<UCSZ11 | 1<<UCSZ10 | 1<<USBS1 | 0<<UMSEL1)
sts UCSR1C, mpr
; Set baudrate at 2400bps
;SEI
rjmp MAIN
;Other
Main Program
; TODO: ???
rcall transmit
rjmp MAIN
;* Functions and Subroutines
transmit:
in mpr, \mbox{PIND}
andi mpr, 0b1111_0111
cpi mpr, 0b1111 0110
breq forward
cpi mpr, 0b1111 0101
breq backward
cpi mpr, 0b1110 0111
breq right
cpi mpr, 0b1101_0111
breq left
cpi mpr, 0b1011 0111
breq stop
cpi mpr, 0b0111_0111
breq freeze
forward:
ldi mpr, 0b10110000
rjmp check
backward:
ldi mpr, 0b1000_0000
rjmp check
left:
ldi mpr, 0b1010 0000
rjmp check
right:
ldi mpr, 0b1001 0000
rjmp check
stop:
ldi mpr, 0b1100 1000
```

```
rjmp check
freeze:
ldi mpr, Ob1111 1000 ;set to freeze
;ldi mpr, 0b0101 0101
rjmp check
check:
push mpr
ldi mpr, BotAddress; transmit bot ID
RCALL USART Transmit
pop mpr
out PORTB, mpr
RCALL USART Transmit
ret
USART Transmit:
lds ereg, UCSR1A
sbrs ereg, UDRE1 ; Loop until UDR0 is empty
rjmp USART Transmit
sts UDR1, mpr; Move data to transmit data buffer
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 + 3) * olcnt + 3) * waitcnt + 13 + call
Wait:
push waitcnt
push ilcnt
                            ; Save wait register
; Save ilcnt register
push olcnt
                            ; Save olcnt register
Loop: ldi
OLoop: ldi
                     olcnt, 224
                                          ; load olcnt register
                                          ; load ilcnt register
                    ilcnt, 237
                                          ; decrement ilcnt
ILoop: dec
                    ilcnt
                      ; Continue Inner Loop ; decrement olcnt
brne ILoop
dec
              olcnt
brne
       OLoop
                           ; Continue Outer Loop
dec
              waitcnt
                           ; Decrement wait
brne Loop
                            ; Continue Wait loop
pop
              olcnt
                           ; Restore olcnt register
                            ; Restore ilcnt register
              ilcnt
qoq
                            ; Restore wait register
рор
              waitcnt
                            ; Return from subroutine
ret
;* Stored Program Data
.org $0200
.dseg
data: .byte 1
```

RX Source Code

```
; *
    This is the RECEIVE file for Lab 8 of ECE 375
; *
; *
     Author: Mason Obery and Benjamin Leonard
; *
      Date: 2/24/2022
.include "m128def.inc"
                      ; Include definition file
```

```
;* Internal Register Definitions and Constants
    ***********
.def mpr = r16
                              ; Multi-Purpose Register
.def
   ereg = r17
.def
    waitcnt = r23
                              ; Wait Loop Counter
   ilcnt = r18
.def
                              ; Inner Loop Counter
.def olcnt = r19
                              ; Outer Loop Counter
   WTime = 100
                              ; Time to wait in wait loop
.equ WskrR = 0
                              ; Right Whisker Input Bit
    WskrL = 1
                              ; Left Whisker Input Bit
.eau
    EngEnR = 4
                              ; Right Engine Enable Bit
.equ
    EngEnL = 7
                              ; Left Engine Enable Bit
.equ
    EngDirR = 5
.equ
                              ; Right Engine Direction Bit
    EngDirL = 6
                              ; Left Engine Direction Bit
.equ
.equ
    BotAddress = 0x2B; (Enter your robot's address here (8 bits))
; These macros are the values to make the TekBot Move.
;0b00000000 Move Backward Action Code
.equ TurnR = (1<<EngDirL)</pre>
                                   ;0b01000000 Turn Right Action Code
.equ TurnL = (1<<EngDirR)
.equ Halt = (1<<EngEnR|1<<EngEnL)</pre>
                                   ;0b00100000 Turn Left Action Code
                                   ;0b10010000 Halt Action Code
Start of Code Segment
.cseg
                                   ; Beginning of code segment
Interrupt Vectors
.org $0000
                             ; Beginning of IVs
    INIT
rjmp
                   ; Reset interrupt
.org INTOaddr
rcall HitRight
ret.i
    INT1addr
.org
rcall HitLeft
.org $003C
rcall rCommand
reti
; Should have Interrupt vectors for:
:- Left whisker
; - Right whisker
; - USART receive
.org $0046
                              ; End of Interrupt Vectors
;* Program Initialization
TNTT:
ldi ereg, $00
; Stack Pointer (VERY IMPORTANT!!!!)
ldi
     mpr, low(RAMEND)
out
          SPL, mpr
ldi
         mpr, high(RAMEND)
out
         SPH, mpr
;I/O Ports
         mpr, 0b0000 1000 ; 1 is for IR output for Freeze
1 di
```

```
out
           DDRD, mpr
ldi
            mpr, $FF
out
             PORTD, mpr
; Initialize Port B for output
                                ; Set Port B Data Direction Register
ldi
            mpr, $FF
                                ; for output
out
             DDRB, mpr
             mpr, $00
                                ; Initialize Port B Data Register
ldi
             PORTB, mpr
                                ; so all Port B outputs are low
out
;USARTO to recieve
; Set baudrate at 2400bps
ldi mpr, high (832)
sts UBRR1H, mpr
ldi mpr, low(832)
sts UBRR1L, mpr
; Enable receiver and enable receive interrupts
;Set frame format: 8 data bits, 2 stop bits
ldi mpr, (1<<U2X1)
sts UCSR1A, mpr
ldi mpr, ((1<<RXCIE1) | (1<<RXEN1));</pre>
sts UCSR1B, mpr
ldi mpr, ((1<<UCSZ11)|(1<<UCSZ10)); modified the paratheses
sts UCSR1C, mpr
; USART1 (to send freexe)
ldi
             ZL, low(HP)
            ZH, high (HP)
ldi
ldi
            mpr, 3
st
             Z, mpr
; External Interrupts
; Set the External Interrupt Mask
; Set the Interrupt Sense Control to falling edge detection
ldi
            mpr, 0b00001010; falling edge
sts
            EICRA, mpr
ldi ZL, low(preFreeze)
ldi ZH, high(preFreeze)
in mpr, PORTB
st \mathbf{Z}_{\bullet} mpr
             mpr, 0b00000011; int0-3 activated
ldi
            EIMSK, mpr
out.
;Other
; Initialize TekBot Forward Movement
            mpr, MovFwd ; Load Move Forward Command PORTB, mpr ; Send command to motors
ldi
out
SET
rjmp Main
; * Main Program
      ***********
MAIN:
; TODO: ???
rjmp MAIN
; * Subroutines and Functions
; ;-----
  HitRight
; Desc: Handles functionality of the TekBot when the right whisker
   is triggered.
HitRight:
                         ; Save mpr register
push mpr
push
                         ; Save wait register
          mpr, SREG
in
                         ; Save program state
push mpr
```

```
in
            mpr, PORTB
push mpr
; Move Backwards for a second
             ldi
out
ldi
             waitcnt, WTime ; Wait for 1 second
rcall Wait
                          ; Call wait function
; Turn left for a second
                          ; Load Turn Left Command
ldi
             mpr, TurnL
                         ; Send command to port
out
             PORTB, mpr
ldi
             waitcnt, WTime; Wait for 1 second
rcall Wait
                           ; Call wait function
; Move Forward again
ldi
             mpr, MovFwd
                          ; Load Move Forward command
             PORTB, mpr
                          ; Send command to port
out.
ldi mpr, $0F
out EIFR, mpr
pop
      mpr
out
             PORTB, mpr
                          ; Restore program state
рор
             SREG, mpr
out.
рор
             waitcnt
                          ; Restore wait register
             mpr
                           ; Restore mpr
pop
                           ; Return from subroutine
ret
; Sub: HitLeft
; Desc: Handles functionality of the TekBot when the left whisker
          is triggered.
HitLeft:
push mpr
push wai
                          ; Save mpr register
      waitcnt
                          ; Save wait register
in
           mpr, SREG
                          ; Save program state
push
      mpr
in
            mpr, PORTB
push
     mpr
; Move Backwards for a second
     out.
ldi
             waitcnt, WTime ; Wait for 1 second
rcall Wait
                          ; Call wait function
; Turn right for a second
             mpr, TurnR
                          ; Load Turn Left Command
ldi
out
             PORTB, mpr
                          ; Send command to port
             waitcnt, WTime; Wait for 1 second
ldi
rcall Wait
                          ; Call wait function
; Move Forward again
ldi
             mpr, MovFwd ; Load Move Forward command
out
             PORTB, mpr
                          ; Send command to port
ldi mpr, $0F
out EIFR, mpr
     mpr
pop
out
             PORTB, mpr
                           ; Restore program state
pop
             mpr
             SREG, mpr
out
pop
             waitcnt
                          ; Restore wait register
рор
             mpr
                           ; Restore mpr
                           ; Return from subroutine
ret
```

```
lds mpr, UDR1 ; reads transmitted info
push mpr
ldi mpr, $00
sts UCSR1B, mpr
ldi mpr, ((1<<RXCIE1) | (1<<RXEN1))</pre>
sts UCSR1B, mpr
pop mpr
cpi mpr, 0b01010101; check if freeze command recieved
breq frozen
cpi mpr, BotAddress
                    ;botID recieved
brne notEqual
ldi ereg, $FF
ret
NotEqual:
cpi ereg, $FF ; check that the botID preceded it
breq ValidCommand
ret
ValidCommand: ; check for valid opcodes
cpi mpr, 0b1011 0000
breq forward
cpi mpr, 0b1000_0000
breq backward
cpi mpr, 0b1010 0000
breq right
cpi mpr, 0b1001_0000
breq left
cpi mpr, 0b1100 1000
breq stop
cpi mpr, 0b1111_1000
breq freeze
rjmp toEnd
forward:
ldi mpr, MovFwd
out PORTB, mpr
rjmp toEnd
backward:
ldi mpr, MovBck
out PORTB, mpr
rjmp toEnd
right:
ldi mpr, TurnR
out PORTB, mpr
rjmp toEnd
left:
ldi mpr, TurnL
out PORTB, mpr
rjmp toEnd
stop:
ldi mpr, Halt
out PORTB, mpr
rjmp toEnd
freeze:; set up transmitter, disable recever
ldi mpr, ((1<<TXEN1))
sts UCSR1B, mpr
ldi mpr, 0b01010101
rcall USART Transmit1
ldi mpr, ((\overline{1} < RXCIE1) | (1 < RXEN1)); reenable reciever to work properly
sts UCSR1B, mpr
rjmp toEnd
```

```
toEnd:
ldi ereq, $00
ret
USART Transmit1:
lds ereg, UCSR1A
sbrs ereg, UDRE1 ; Loop until UDR0 is empty
rjmp USART Transmit1
sts UDR1, mpr; Move data to transmit data buffer
USART Transmit2:
lds ereg, UCSR1A
sbrs ereg, UDRE1; Loop until UDR0 is empty
rjmp USART Transmit2
ret
frozen:
ldi ZL, low(preFreeze)
ldi ZH, high(preFreeze)
in mpr, PORTB
st Z, mpr
ldi
      mpr, halt
out PORTB, mpr
ldi ZL, low(HP)
ldi ZH, high(HP)
ld mpr, Z
dec mpr
st Z, mpr
ld mpr, Z
cpi mpr, 0
breq killed
ldi waitcnt, 250
rcall Wait
ldi waitcnt, 240
rcall Wait
ldi ZL, low(preFreeze)
ldi ZH, high(preFreeze)
ld
     mpr, Z
      PORTB, mpr
out.
rjmp toEnd
killed:
ldi mpr, 0
out EIMSK, mpr
sts UCSR1B, mpr
rjmp toEnd
; 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
                              ; Save wait register
       waitcnt
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
                              ; Continue Inner Loop
brne ILoop
             olcnt
                              ; decrement olcnt
dec
```

```
OLoop ; Continue Outer Loop waitcnt ; Decrement wait Loop ; Continue Wait loop
brne OLoop
dec
    Loop
brne
                      ; Restore olcnt register
pop
          olcnt
                      ; Restore ilcnt register ; Restore wait register
           ilcnt
pop
          waitcnt
pop
                       ; Return from subroutine
ret
; * Stored Program Data
.dseq
.org $0200
HP: .byte 1
PreFreeze: .byte 1
```

RX CHALLENGE CODE

```
; *
; *
     This is the RECEIVE skeleton file for
                                                Lab 8 of ECE 375
; *
     Author: Mason
                             Obery and
; *
                                                 Benjamin
                                                                 Leonard
; *
      Date:
                                                                2/24/2022
.include "m128def.inc"
                                                  definition
                                                                   file
                                    Include
;* Internal Register Definitions and
                                                               Constants
.def mpr = r16
                                        Multi-Purpose
                                                                Register
                                               Loop
.def
                                                                 r17
   ereq
.def waitcnt = r23
.def ilcnt = r18
.def olcnt = r19
                                         Wait
                                                                 Counter
                                ;
                                                    Loop
Loop
                                ;
                                          Inner
                                                                 Counter
                                         Outer
                                                                 Counter
                                ;
                                ; Time to wait in wait loop
.equ WTime = 100
                                     Right Whisker Input Bit
Left Whisker Input Bit
Right Engine Enable Bit
Left Engine Enable Bit
Right Engine Direction Bit
Left Engine Direction Bit
.equ WskrR = 0
                                ;
.equ WskrL = 1
                                ;
     EngEnR = 4
.equ
                                ;
.equ
     EngEnL = 7
.equ EngDirR = 5
                                :
.equ EngDirL = 6
.equ BotAddress = 0x2B; (Enter your robot's address here (8 bits))
;These macros are the values to make the TekBot Move.
.equ MovFwd = (1<<EngDirR|1<<EngDirL) ;0b01100000 Move Forward Action
                                           ;0b0000000 Move Backward Action Code
.equ MovBck = $00
.equ TurnR = (1<<EngDirL)
.equ TurnL = (1<<EngDirR)</pre>
                                     ;0b01000000 Turn Right Action Code
;0b00100000 Turn Left Action Code
.equ Halt = (1<<EngEnR|1<<EngEnL)
                                     ;0b10010000
                                                 Halt
                                                          Action
;* Start
              of
                                             Code
                                                                 Seament
                                      ; Beginning of code segment
.csea
```

; *	Interr	upt	* * * * * * * * * * * * * * * * * * * *						Vectors
.org	\$0000				;	Beginnin	~	of	IVs
.019	70000	rjmp	INIT		;	-	set	OI	interrupt
.org	INT0ad	dr							
		rcall							HitRight
		reti							
.org	INT1ad								11: LT - EL
		rcall reti							HitLeft
		Tett							
ora	\$0034								
.org	Ş 00 34	rcall							nowgoleft
		reti							nowgoicic
.org	\$0036								
_		rcall						r	nowgoright
		reti							
.org	\$003A								
		rcall							finish
		reti							
	¢0020								
.org	\$003C	rcall							rCommand
		reti							1 COMMATIC
;Shoul	.d	1001	have		Interrupt		vectors		for:
; -					Left				whisker
; -					Right				whisker
; –					USART				receive
.org	\$0046				; En	nd of	Inte	rrupt	Vectors
; ***** ; *	Progra		* * * * * * * * * * * * *	****	****	****		Tniti	Lalization
			* * * * * * * * * * * * *	*****	*****	****		111101	Lalizacion
INIT:									
	; new								plan
	/*ldi				mpr,				\$00
	sts				TCCR3A,				mpr*/
	ldi				ereg,				\$00
	Q1 1			.		(I.T.D.)		T1/D0	
	;Stack ldi			Pointer	mnr	(VERY			RTANT!!!!) ow(RAMEND)
	out				mpr, SPL,			Τ.	mpr
	ldi				mpr,			hio	gh (RAMEND)
	out				SPH,			-	mpr
	; I/O								Ports
		mpr,	0b0000_1000	;button		1 is fo	r IR ou	tput for	
	out				DDRD,				mpr
	ldi out				mpr, PORTD,				\$FF mnr
	;		Initialize		Port	В	fo	r	mpr output
	ldi		mpr, \$FF			Port B			Register
	out		DDRB, mpr		;		for		output
	ldi		mpr, \$00				ort B	Data	Register
	out		PORTB, mpr		; so all Po	ort B output	s are low		
	2 21		4			_	~~	_	
	ldi out	mpı	r, \$FF	;	output DDRE,	for	COM	of	timer3 mpr

```
recieve
      ;USARTO
            ;Set
                                                         at
                                                                              2400bps
                                baudrate
            ldi
                                                                            high (832)
            sts
                                            UBRR1H,
                                                                              mpr
            ldi
                                                                             low (832)
                                            mpr,
                                                                              mpr
            sts
                                             UBRR1L,
            ; Enable receiver and enable receive interrupts
            ;Set frame format: 8 data bits, 2 stop bits
                                                                            (1<<U2X1)
                                            mpr,
                                             UCSR1A,
            sts
                                                                                mpr
            ldi
                          mpr,
                                         ((1<<RXCIE1)
                                                             (1<<RXEN1));
                                            UCSR1B,
            sts
                                                                           mpr
            ldi mpr, ((1<<UCSZ11)|(1<<UCSZ10)|(1<<USBS1)) ;modified the
                                                                           paratheses
                                             UCSR1C,
      ; USART1
                                (to
                                                      send
                                                                              freexe)
      ldi
                                          ZL,
                                                                              low(HP)
      ldi
                                          ZH,
                                                                             high(HP)
      ldi
                                          mpr,
      st
                                           Z,
                                                                                 mpr
                                          8-bit mpr,
                        Configure
                                                                       Timer/Counters
      ;
            ldi
                                                                        0b01 11 1 001
                                            TCCR2,
            out
                                                                                 mpr
                                          mpr,
TCCR0,
                                                                         0b01_11_1_001
            out
      ldi mpr, 0b0000 0000; outputcompare interrupt match enable for timer0 and timer2. and com1
          com2 and ovf for timer1
and
            out
                                             TIMSK,
                                                                                mpr
      ldi
                                                                           low (speed)
                                         ZL,
                                                                           high (speed)
            ldi
                                           ZH,
            ldi
                                              mpr,
                                                                                 $0F
            st
                                               z,
                                                                                 mpr
      ;External
                                                      Interrupt
to falling edge
            ;Set
                          the
                                         External
                                                                           Mask
                                    External
Sense Control to
0b00001010 ;
EICRA,
                       Interrupt
                                                           falling edge
falling
             ;Set
                                                                            detection
            ldi
                                                                           edge
                        mpr,
                                            EICRA,
            sts
                                                                                 mpr
                                                             int0-3
                                    0b00000011 ;
            ldi
                       mpr,
                                                                            activated
            out
                                             EIMSK,
      ;Other
                                        TekBot
; Load
; Send
                                                          Forward
Move Forward
                   Initialize
                                                                            Movement
                                                           rorward Command command to motore
            ldi
                    mpr, MovFwd
                         PORTB, mpr
            out
                   ldi
                                                                           HIGH (49911)
                                             mpr,
            sts
                                             OCR3AH,
                                                                              mpr
                                           mpr,
                                                                           LOW (49911)
            ldi
             sts
                                           OCR3AL,
                                                                                mpr
                   ;right
            ldi
                                                                           HIGH (49911)
                                           mpr,
                                            OCR3BH,
            sts
                                                                             mpr
            ldi
                                                                           LOW (49911)
                                           mpr,
                                             OCR3BL,
            sts
                                                                                mpr
                   ldi
                                                                        0b1111 0000*/
                                             mpr,
                                                                          0000 0000d0
                   ldi
                                              mpr,
```

sts ldi sts		TCCR3A, mpr, TCCR3B,		ı	mpr 0b0000_0101 mpr
; * Main	* * * * * * * * * * * * * * * * * * *				Main Program
MAIN: ;TODO: nop nop					???
nop busywait: rjmp					busywait
rjmp MAIN	*****	****			
•	*****************************			:	Subroutines
;* Stored	* * * * * * * * * * * * * * * * * * * *	Program			Data
;* ;**************** ;	**************************************	**************************************	and	than	Functions
INCspeed: PUSH PUSH PUSH PUSH					ZL ZH mpr low(speed)
ldi ld		ZL, ZH, mpr,		1	nigh (speed)
CPI BREQ exit inc mpr st	;check if speed ;if it's	mpr, level is not Z,	at 15,	exit if speed	it is value mpr
rjmp					exit
; Func: DECspeed ; Desc: Decrements ;	speed	of	less	than	0
DECspeed: PUSH PUSH					ZL ZH

```
PUSH
                                                                          mpr
     ldi
                                     ZL,
                                                                     low(speed)
     ldi
                                     ZH,
                                                                    high (speed)
                                                                           Z
0
                                      mpr,
      1 d
                          if
      CPI mpr, 0 ; check
                                               level is at
                                     speed
     BREQ
                                                                          exit
                ;if
                                      decrement speed
                                                                 bу
      dec mpr
                          not,
                                                                           1
      st
                                       Z,
                                                                          mpr
exit:
          ; all functions share the same exit sequence, so to save space it is abbrieviated to
      ldi
                                      waitcnt,
      rcall
                                                                          Wait
      rcall
                                                                        Left0n
     rcall
                                                                       RightOn
                                 ;clear external interrupt register
      ldi
         mpr, $0F
                                       EIFR,
     out.
                                                                          mpr
     POP
                                                                          MPR
     POP
                                                                           ZH
                                                                           ZL
     POP
      ret
; Func: LeftOn
; Desc: Sets
                            PWM
                                                   signal
                                                                           on
     push
                                                                           mpr
     push
                                                                           ZL
     push
                                                                           ZH
                                                                        waitcnt
     push
     ldi
                                     ZL,
                                                                     low(speed)
     ldi
                                     ZH,
                                                                    high (speed)
      1d
                                       mpr,
                                      waitcnt,
     mul mpr, waitcnt ; convert 4-bit nibble to 8-bit even
                                                                    distirbution
     mov
                                       mpr,
                                                                           r0
                                       OCR2,
     out
                                                                          mpr
      /*ldi
                                                                          ŚFF
                                       mpr,
     out TIFR, mpr ;clear
                                 timer
                                                  interrupt
                                                                     register*/
                                                                        waitcnt
                                                                          ZH
     pop
     рор
                                                                           ZL
                                                                          mpr
     pop
      ret
;-----
; Func: RightOn
                PWM
; Desc: Sets
                                                     to
                                                                    approriate
     push
                                                                           mpr
     push
                                                                           ZL
```

push							ZH
push							waitcnt
ldi ldi ld		ZI ZH	•		low(speed) high(speed) Z		
ldi			waitcnt,				17
mul mpr, waitent mov out	; convert	4-bit	nibble mpr, OCRO,	to	8-bit	even	distirbution r0 mpr
/*ldi			mpr,				\$FF
out TIFR, mpr ;clear		timer		in	terrupt		register*/
pop pop pop pop ret							waitent ZH ZL mpr

```
;------
; Sub: HitRight
; Desc: Handles
              functionality of the TekBot when the right whisker
                                                                      triggered.
; is
HitRight:
                                                          mpr
wait
                                               Save
                                                                       register
            push
                 mpr
                  waitcnt
                                             Save
Save
            push
                                                                       register
                 mpr, SREG
                                                                        state
            in
                                                          program
            push mpr
           ,
ldi
out
/*
                       Move Backwards for a mpr, MovBck ; Load Move Backward PORTB, mpr ; Send command to
                    Move
                                                                         second
                      mpr, MovBck
                                                                        command
                       PORTB, mpr ; Send command to waitcnt, WTime; Wait for 1 ; Call wait RWait ; right side
           out
                                                                        port
second
            ldi
            rcall Wait;
                                                                        function
            rcall
                                          Load Turn Left
Send command to
Wait for 1
Call
                                    left; Load;
                       Turn
                                                                         second
                       mpr, TurnL
            ldi
                                                                        Command
                                                                        port
            out
                       PORTB, mpr
                       waitcnt, WTime;
            ldi
                                                                         second
            ;rcall Wait
                                                                       function
            rcall
                                                                       NewWait*/
                        mpr, MovBck ;
                                         Load
                                                    Move Backward
                                                                        command
            out
                        PORTB, mpr ;
                                          Send
                                                   command to
                                                                         port
            rjmp
                                                                 rdisableandenable
                                         ; Restore
/*hitrmid:
                  pop
                             mpr
                                                           program
                                                                        state
                                             Restore wait register
Restore mpr
Return from subroutine*/
                        SREG, mpr
            out
                                         Restore
            pop
                        waitcnt
            рор
                        mpr
                                            Return
            ret
```

- Page 18 -

```
; Sub: HitLeft
                                             TekBot when the left whisker
; Desc: Handles
                functionality of the
; is
                                                                         triggered.
HitLeft:
             pusn waitcnt ; in mpr, SREG ; push mpr
                                                   Save mpr
Save wait
Save program
                                                                                register
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 ; reall Wait ; Call wait function reall RWait ; right side wait
/*
             ; 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 ; reall Wait ; Call wait function NewWait*/
                                                                                NewWait*/
     ldi
                  mpr, MovBck ; Load Move Backward command PORTB, mpr ; Send command to port
             out
                                                                       ldisableandenable;
             rjmp
                          mpr ; Restore program state SREG, mpr ; waitcnt ; Restore wait register
hitlmid:
             pop
             out
                                        ; Restore wait register; Restore mpr; Return from subroutine*/
             pop
                          mpr
             pop
             ret
rdisableandenable:
      int0-3 de
                                                                               activated
                  mpr, MovBck ; Load Move Backward PORTB, mpr ; Send command to
                                                                                 command
      out
                                                                                    port
      ldi
                                             mpr,
                                                                                     $00
                                            UCSR1B,
                                                                                      mpr
      sts
                                     different output compares and the overflow
      ; use timer 1 with two
                                                                               0b0011 0000
      ldi
                                          mpr,
      sts
                                           TCCR3A,
                                                                                 mpr*/
/*
      ldi
                                          mpr,
                                                                                0b0000 0101
                                                                                mpr*/
                                          TCCR3B,
      sts
                                        used to force compare
      ; nothing for TCCR1C,
                                                                                   match
                           new
                                           load
      ; now
                                                       attempt
                                                                                  values
                                            timer HIGH (34286)
                                                                                   HERE:
      ;starting
                                             HIGH (34286)
TCNT3H,
                                                                                 ;34286
mpr
           ldi
                            mpr,
             sts
             ldi
                                             mpr,
                                                                                LOW (34286)
                                               TCNT3L,
             sts
                                                                                    mpr
```

	;confi;ldi mpr,	gure 0b0001_1100 bo	th, bit 4 is mpr,	A, 3 is	B, and	TIMSK 2 is OVF 0b0001_0100
	sts		ETIMSK,			mpr
	ldi out	mpr, MovBck ; PORTB, mpr ;		Move command	Backward to	command port
	ldi sts ETIFR, mp	or ;clear qued tim	mpr, mer compare matches	s that set too e	early because	0b0001_1100 e they queued
	pop out pop pop		SRE	eg,		mpr mpr waitcnt mpr
	ret					
ldisab	leandenable: ldi m out	pr, 0b0000	_00000 ; EIMSK,	int0-3	de	activated mpr
	ldi out	mpr, MovBck ; PORTB, mpr ;		Move command	Backward to	command port
	ldi sts		mpr, UCSR1B,			\$00 mpr
/* /*	;use timer ldi sts ldi sts	1 with tw	o different ou mpr, TCCR3A, mpr, TCCR3B,	utput compares	s and th	ne overflow 0b1100_0000 mpr*/ 0b0000_0101 mpr*/
	;nothing	for TCCR10	·	to force	compar	_
	;now ;starting ;right	new	load timer	at	tempt	values HERE: side
	ldi sts ldi sts		mpr, TCNT mpr, TCNT	·		HIGH (34286) mpr LOW (34286) mpr
	;confi;ldi mpr, ldi sts		th, bit 4 is mpr, ETIMSK,	A, 3 is	B, and	TIMSK 2 is OVF 0b0000_1100 mpr

	ldi out			mpr, Mo		;		oad Send	гоМ	ve command	Backward to	command port
	ldi sts E	TIFR,	mpr	;clear	qued	timer	mpr compare		that	set too	early because	0b0001_1100 they queued
	pop	out pop pop		SREG,								
;;;;;;	ret	;;;;;	;;;;	,,,,,,,	;;;;;	;;;;;	,,,,,,,,	;;;;;				
rComma						did						
	;in lds		mp	r,	U	DR1	;		reads	not s	transmitted	work info
	cpi brne ldi ret						mp	r, ereg,				BotAddress notEqual \$FF
NotEqua	al: cpi breq ret							ereg,				\$FF ValidCommand
ValidCo	ommand: cpi breq rjmp						mpr mpr mpr					0b1011_0000 forward 0b1000_0000 backward 0b1010_0000 right 0b1001_0000 left 0b1100_1000 speedUp 0b1111_1000 speedDown toEnd
forward	d:											
	ldi out rjmp							mpr, PORTB,				MovFwd mpr toEnd
backwa:	rd: ldi out rjmp							mpr, PORTB,				MovBck mpr toEnd
right:	ldi out rjmp							mpr, PORTB,				TurnR mpr toEnd
left:	ldi out rjmp							mpr, PORTB,				TurnL mpr toEnd
stop:	ldi out rjmp							mpr, PORTB,				Halt mpr toEnd

```
speedUp:
  rcall
                                                                         INCSPEED
     rjmp
                                                                           toEnd
speedDown:
     rcall
                                                                         DECSPEED
     rjmp
                                                                           toEnd
toEnd:
     ldi
                                                                            $00
                                        ereq,
     ret
                       com
                                                         turn
                                                                          signal
     ;stop
nowgoleft:
                                                                       0b0000 0000
     /*ldi
                                       mpr,
                                       TCCR3A,
     sts
                                                                          mpr*/
                                                 Turn Left to Move Backward
     ldi
                  mpr, TurnL
                                       Load
                                                                          Command
                  PORTB, mpr ;
                                    Send
Load
; Send
                                                                          port
                  mpr, MovBck
                             ;
     ldi
                                                                         command
           out
                  PORTB, mpr
                                                 command to
                                                                         port*/
     ret
nowgoright:
                                                                       0b0000 0000
     ldi
                                      mpr,
                                      TCCR3A,
                                                                           mpr*/
                  mpr, TurnR ;
PORTB, mpr ;
mpr, MovBck ;
PORTB, mpr ;
                                    Load
Send
                                                            Left
to
                                                  Turn
     ldi
                                                                          Command
                                                                          port
     out
                                                  command
                                     Load
Send
      ldi
                                                 Move
                                                            Backward
                                                                          command
                                                            to
                                                  command
                                                                          port*/
     out.
     ret.
finish:
                                                 Move Forward
      ldi
                                   ; Load
                  mpr, MovFwd
                                                                        Command
                 PORTB, mpr
                                          Send
      out
                                                    command to
                                                                          motors
      ldi
                           mpr,
                                                  0b0000 0000
                                        ETIMSK,
      sts
                                                                             mpr
      ldi
                                                                             $FF
                                         mpr,
                                        EIFR,
      out
                                                                             mpr
               mpr, 0b00000011 ; int0-1 activated mpr
      ldi
      out
           ldi
                                      ((1<<RXCIE1)
                                                          (1<<RXEN1));
                       mpr,
            sts
                                          UCSR1B,
     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 + 3) * olcnt + 3) * waitcnt + 13 + call
; Wait:
```

		push	waitcn	t	;	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		;	decr	rement	ilcnt
		brne	ILoop		;	Continue	Inner	Loop
		dec	_	olcnt	;	decr	rement	olcnt
		brne	OLoop		;	Continue	Outer	Loop
		dec	-	waitcnt	;	De c	rement	wait
		brne	Loop		; C	ontinue Wait loop		
		pop		olcnt	;	Restore	olcnt	register
		pop		ilcnt	;	Restore	ilcnt	register
		pop		waitcnt	;	Restore	wait	register
		ret			;	Return	from	subroutine
; ****	*****	*****	*****	* * * * * * * * * * * * * * * * * * * *	* * * * * *	** *** ** **		
<i>;</i> *	Additi					Program		Includes
; ****	*****	*****	*****	* * * * * * * * * * * * * * * *	* * * * * * *	****		
.dseg								
								20000
.org								\$0200
	HP:					.byte		1
	speed:	.byte 1	L					