ECE 375 Lab 6

Freeze Tag

Lab Time: Wednesday 5-7

Jacques Uber
Riley Hickman

TA Signature

1 Introduction

Make two AVR boards play freeze tag.

2 What We did and Why

We used two AVR boards, one slave and one master, to implement freeze tag. We used an additional board to demonstrate freeze tag.

Our master board used busy waiting to listen to input from the buttons. When a certain button was pressed we called a function that would transmit BotID and command to our slave AVR board. We had three commands. Left motor, Right motor, and "Freeze". We sent signals using the USART IR transmitter built into the AVR board.

Our slave board listened for commands from the master board via interrupts. When an interrupt was triggered it would check whether or not the received signal was from the master by comparing the signal to the BotID (both master and slave used the same BotID). If the BotID didn't match, the command was ignored. This meant that other people's bot controllers could not interfere with our slave.

Before we checked for the BotID we checked to see if the signal received was a freeze command. It it was a freeze command the bot would freeze for 3 seconds. After being frozen three times the bot would stay frozen. We also would print the freeze command to PORTB. When our slave froze for the third time we counted in binary and printed the output to PORTB.

There is verbose pseudo code contained in the source code explaining how we handled signals from the master.

3 Difficulties

Debugging was hard. It was hard to identify where bugs were. We used PORTB (the LEDs) to debug some things. In one instance we found that the function "Wait" was not working. We got around this by placing the contents of the Wait function inline.

Keeping track of which registers were being used and for what purpose was critical. We found that bugs would happen if we were using a register for two different things at the same time.

4 Conclusion

I/O is hard.

5 Source Code

MASTER Code

```
;***********************************************************
;*
;* Enter Name of file here
;*
```

```
;* Enter the description of the program here
;*
;* This is the RECEIVE skeleton file for Lab 6 of ECE 375
;*
;***********************************************************
;*
;*  Author: Enter your name
;*    Date: Enter Date
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************

.def    mpr = r16               ; Multi-Purpose Register
.def    waitcnt = r17               ; Wait Loop Counter
.def    ilcnt = r18             ; Inner Loop Counter
.def    olcnt = r19             ; Outer Loop Counter
.def    tmp = r20
.def rec = r22 ; Multi-Purpose Register


.equ    WTime = 50              ; Time to wait in wait loop

.equ B0 = 0b11111110 ; Right Whisker Input Bit
.equ B1 = 0b11111101
.equ B2 = 0b11111011
.equ B3 = 0b11110111
.equ B4 = 0b11101111
.equ B5 = 0b11011111
.equ B6 = 0b10111111
.equ B7 = 0b01111111
.equ FREEZE = 0b11111000

.equ WskrR = 0 ; Right Whisker Input Bit
.equ WskrL = 1 ; Left Whisker Input Bit
.equ EngEnR = 4 ; Right Engine Enable Bit
.equ EngEnL = 7 ; Left Engine Enable Bit
.equ EngDirR = 5 ; Right Engine Direction Bit
.equ EngDirL = 6 ; Left Engine Direction Bit

;.equ BotID = ;(Enter you group ID here (8bits)); Unique XD ID (MSB = 0)
.equ BotID = 0b01111111 ;(Enter you group ID here (8bits))
```

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

.equ MovFwd =  (1<<EngDirR|1<<EngDirL) ;0b01100000 Move Forwards Command
.equ MovBck =  $00 ;0b00000000 Move Backwards Command
.equ TurnR =   (1<<EngDirL) ;0b01000000 Turn Right Command
.equ TurnL =   (1<<EngDirR) ;0b00100000 Turn Left Command
.equ Halt =    (1<<EngEnR|1<<EngEnL) ;0b10010000 Halt Command

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

;-----------------------------------------------------------
; Interrupt Vectors
;-----------------------------------------------------------
.org $0000 ; Beginning of IVs
rjmp  INIT ; Reset interrupt

;Should have Interrupt vectors for:
;.org    $003C
;    rcall USART_Receive
;    reti
;- Left wisker
;- Right wisker
;- USART receive


;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:
        ;Stack Pointer (VERY IMPORTANT!!!!)
        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
        ;I/O Ports
        ; Initialize Port B for output
        ldi     mpr, $00        ; Initialize Port B for outputs
        out     PORTB, mpr      ; Port B outputs low
        ldi     mpr, $ff        ; Set Port B Directional Register
        out     DDRB, mpr       ; for output
```

```
        ; Initialize Port D for inputs
        ldi     mpr, $FF        ; Initialize Port D for inputs
        out     PORTD, mpr      ; with Tri-State
        ldi     mpr, $00        ; Set Port D Directional Register
        out     DDRD, mpr       ; for inputs

        ;USART1
USART_INIT:
        ;Set double data rate
        ldi r16, (1<<U2X1)
        ; UCSR1A control register --
        ;Bit 1  U2Xn: Double the USART Transmission Speed
        sts UCSR1A, r16

        ;Set baudrate at 2400bps
        ; UBRR1H Bod rate control register
        ldi r16, high(832)
        sts UBRR1H, r16
        ldi r16, low(832)
        sts UBRR1L, r16


;Set frame format: 8data bits, 2 stop bit
        ldi r16, (0<<UMSEL1|1<<USBS1|1<<UCSZ11|1<<UCSZ10)
        sts UCSR1C, r16

        ;Enable both receiver and transmitter -- needed for Lab 2
        ; RXEN (Receiver enable) TXEN (Transmit enable)
        ldi r16, (1<<RXCIE1|1<<RXEN1|1<<TXEN1)
        sts UCSR1B, r16

;External Interrupts
        ; Turn on interrupts
        sei ; This may be redundant

;Enable receiver and enable receive interrupts
;Set the External Interrupt Mask

        ; Set the Interrupt Sense Control to falling edge
        ldi mpr, (1 <<ISC41)|(0 <<ISC40)|(1 <<ISC51)|(0 <<ISC50)
        out EICRB, mpr

;Other
```

```
;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
        ldi   tmp, $00
MAIN:
        ; Clear lds
        clr mpr
        out PORTB, mpr

        ldi   mpr, $00

        in    mpr, PIND       ; Get whisker input from Port D
        cpi   mpr, B0
        breq   BUTTON0        ; Left

        in    mpr, PIND       ; Get whisker input from Port D
        cpi   mpr, B1
        breq   BUTTON1        ; Right

        in    mpr, PIND       ; Get whisker input from Port D
        cpi   mpr, B5
        breq   SENDFREEZE  ; Button 5



        ;in    mpr, PIND       ; Get whisker input from Port D
        ;cpi   mpr, B2
        ;breq   BUTTON2        ; Forward

        ;in    mpr, PIND       ; Get whisker input from Port D
        ;cpi   mpr, B3
        ;breq   BUTTON3        ; Backward

        in    mpr, PIND       ; Get whisker input from Port D
        cpi   mpr, B4
        breq   BUTTON4        ; Halt

        in    mpr, PIND       ; Get whisker input from Port D
        cpi   mpr, B6
        breq   BUTTON6        ; Halt

        in    mpr, PIND       ; Get whisker input from Port D
        cpi   mpr, B7
        breq   BUTTON7        ; Halt

        ; Clear lds
```

```
        clr mpr
        out PORTB, mpr

        rjmp MAIN


        ;in    mpr, PIND       ; Get whisker input from Port D
        ;cpi   mpr, B6


BUTTON0: ;Left
        ; Load bot id
        ldi mpr, BotID
        ; Send bot id
        call USART_Transmit

        ldi   mpr, 0b00000001
        out PORTB, mpr
        call USART_Transmit
        jmp MAIN
BUTTON1: ;Right
        ; Load bot id
        ldi mpr, BotID
        ; Send bot id
        call USART_Transmit

        ldi   mpr, 0b00000010
        out PORTB, mpr
        call USART_Transmit
        jmp MAIN
SENDFREEZE: ;Freeze
        ; Load bot id
        ldi mpr, BotID
        ; Send bot id
        call USART_Transmit

        ldi   mpr, FREEZE
        out PORTB, mpr
        call USART_Transmit
        jmp MAIN
BUTTON2:
        ; Load bot id
        ldi mpr, BotID
        ; Send bot id
        call USART_Transmit
```

```
        ldi   mpr, 0b00000101
        out PORTB, mpr
        call USART_Transmit
        jmp MAIN
BUTTON4: ;Forward
        ; Load bot id
        ldi mpr, BotID
        ; Send bot id
        call USART_Transmit

        ldi   mpr, MovFwd
        out PORTB, mpr
        call USART_Transmit
        jmp MAIN

BUTTON6: ;Backward
        ; Load bot id
        ldi mpr, BotID
        ; Send bot id
        call USART_Transmit

        ldi   mpr, MovBck
        out PORTB, mpr
        call USART_Transmit
        jmp MAIN

BUTTON7: ;Halt
        ; Load bot id
        ldi mpr, BotID
        ; Send bot id
        call USART_Transmit

        ldi   mpr, Halt
        out PORTB, mpr
        call USART_Transmit
        jmp MAIN


rjmp MAIN


;***********************************************************
;* Functions and Subroutines
;***********************************************************
; USART Receive
USART_Receive:
```

```
    ; Wait for data to be received
    lds rec, UCSR1A
    sbrs rec, RXC1
    rjmp USART_Receive

    ; Get and return receive data from receive buffer
    lds rec, UDR1
    ret

USART_Transmit:
    lds r23, UCSR1A
    sbrs r23, UDRE1
    ; Load status of USART1
    ; Loop until transmit data buffer is ready
    rjmp USART_Transmit

    ; Send data
    sts UDR1, mpr
    ; Move data to transmit data buffer
    ret




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



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

;-----------------------------------------------------------------
; 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
```

Slave Code

```
;***********************************************************
;*
;* Enter Name of file here
;*
;* Enter the description of the program here
;*
;* This is the RECEIVE skeleton file for Lab 6 of ECE 375
;*
;***********************************************************
;*
;*  Author: Enter your name
;*    Date: Enter Date
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def numFrozen = r17 ; Multi-Purpose Register
.def waitcnt = r21 ; Wait Loop Counter
.def rec = r22 ; What we received Register
.def tmp = r20 ; What we received Register
.def tmp2 = r25 ; What we received Register
.def cmd = r24 ; What we received Register
.def state = r23 ; State register.
.def ilcnt = r18 ; Inner Loop Counter
```

```
.def olcnt = r19 ; Outer Loop Counter

.equ WTime = 100 ; Time to wait in wait loop
.equ    FROZEN = 0b01010101
.equ    FREEZE = 0b11111000


.equ WskrR = 0 ; Right Whisker Input Bit
.equ WskrL = 1 ; Left Whisker Input Bit
.equ EngEnR = 4 ; Right Engine Enable Bit
.equ EngEnL = 7 ; Left Engine Enable Bit
.equ EngDirR = 5 ; Right Engine Direction Bit
.equ EngDirL = 6 ; Left Engine Direction Bit

;.equ BotID = ;(Enter you group ID here (8bits)); Unique XD ID (MSB = 0)
.equ BotID = 0b01111111;(Enter you group ID here (8bits)); Unique XD ID (MSB = 0)

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

.equ MovFwd =  (1<<EngDirR|1<<EngDirL) ;0b01100000 Move Forwards Command
.equ MovBck =  $00 ;0b00000000 Move Backwards Command
.equ TurnR =   (1<<EngDirL) ;0b01000000 Turn Right Command
.equ TurnL =   (1<<EngDirR) ;0b00100000 Turn Left Command
.equ Halt =    (1<<EngEnR|1<<EngEnL) ;0b10010000 Halt Command

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

;-----------------------------------------------------------
; Interrupt Vectors
;-----------------------------------------------------------
.org $0000 ; Beginning of IVs
rjmp  INIT ; Reset interrupt

;- Right wisker
; Reset interrupt
.org $0004
rcall HitRight ; Call hit right function
reti

;- Left wisker (For some reason putting this in $0006 broke stuff)
; Return from interrupt
```

```
.org $0002
rcall HitLeft ; Call hit left function
reti

;Should have Interrupt vectors for:
.org    $003C
rcall USART_Receive
reti
;- USART receive


.org $0046 ; End of Interrupt Vectors
;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:
        ;Stack Pointer (VERY IMPORTANT!!!!)
        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
        ;I/O Ports
        ; Initialize Port B for output
        ldi     mpr, $00        ; Initialize Port B for outputs
        out     PORTB, mpr      ; Port B outputs low
        ldi     mpr, $ff        ; Set Port B Directional Register
        out     DDRB, mpr       ; for output

        ; Initialize Port D for inputs
        ldi     mpr, $FF        ; Initialize Port D for inputs
        out     PORTD, mpr      ; with Tri-State
        ldi     mpr, $00        ; Set Port D Directional Register
        out     DDRD, mpr       ; for inputs

        ;USART1
USART_INIT:
        ;Set double data rate
        ldi r16, (1<<U2X1)
        ; UCSR1A control register -- Bit 1  U2Xn: Double the USART Transmission Speed
        sts UCSR1A, r16

        ;Set baudrate at 2400bps
        ; UBRR1H Bod rate control register
        ldi r16, high(832)
        sts UBRR1H, r16
        ldi r16, low(832)
```

```
        sts UBRR1L, r16


;Set frame format: 8data bits, 2 stop bit
        ldi r16, (0<<UMSEL1|1<<USBS1|1<<UCSZ11|1<<UCSZ10)
        sts UCSR1C, r16

        ;Enable both receiver and transmitter -- needed for Lab 2
        ; RXEN (Receiver enable) TXEN (Transmit enable)
        ldi r16, (1<<RXCIE1|1<<RXEN1|1<<TXEN1)
        sts UCSR1B, r16

;External Interrupts
        ; Turn on interrupts
        sei ; This may be redundant

;Enable receiver and enable receive interrupts
;Set the External Interrupt Mask
        ; Set INT4 & 5 to trigger on falling edge
        ldi mpr, $00
        sts EICRA, mpr

        ; Use sts, EICRA in extended I/O space
        ; Set the External Interrupt Mask
        ldi mpr, (1 <<INT2)|(1 <<INT1)|(1 <<INT0)
        out EIMSK, mpr

        ; Set the Interrupt Sense Control to falling edge
        ldi mpr, (1 <<ISC41)|(0 <<ISC40)|(1 <<ISC51)|(0 <<ISC50)
        out EICRB, mpr

;Other


;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
        ldi     mpr, $01
        ldi     state, $00
        clr     numFrozen
        clr     cmd
MAIN:
        clr cmd
        out PORTB, cmd
        ldi waitcnt, 10 ; Wait for 1 second
        call Wait
```

```
        clr cmd
        out PORTB, cmd

rjmp MAIN



;***********************************************************
;* Functions and Subroutines
;***********************************************************
; USART Receive
; Set state to 0
; Listen
;   if received and state = 0 # Check for botid
;       if received == BotId
;            state = 1
;   if received and state = 1 # Accept commands
;       write received as command # Write to LEDs
;       state = 0
USART_Receive:
    ; Wait for data to be received
    lds rec, UCSR1A
    sbrs rec, RXC1
    rjmp USART_Receive

    ; Get and return receive data from receive buffer
    lds rec, UDR1
    ; Data is now in rec
    ; if rec == FROZEN:
    ;   wait n
    ;   numFrozen++
    ;   if numFrozen == 3:
    ;      STUCK  rjmp STUCK
    ;   ret
    ; if state == 0:
    ;   if rec == BotID:
    ;       state = 1
    ;       ret
    ; if state == 1:
    ;   cmd = rec
    ;   mov cmd, rec // Do the command
    ; ============= The Actual Code =============
    ; if rec == FROZEN:

    cpi   rec, FROZEN
    breq  DO_FROZEN
```

```
    ; if state == 0:
    cpi   state, $00
    breq  GO_STATE0
    ; if state == 1:
    cpi   state, $01
    breq  COMMAND
    ret

DO_FROZEN:
    ;   wait n
    out PORTB, rec
    ldi mpr, FROZEN
    out PORTB, mpr
    ldi waitcnt, 500 ; Wait for 1 second
    call Wait
    clr mpr
    out PORTB, mpr
    inc numFrozen
    cpi    numFrozen, 5
    breq LOOP_FOREVER
    ret
LOOP_FOREVER:
    inc mpr
    out PORTB, mpr
    ldi waitcnt, 20; Wait
    call Wait
    rjmp LOOP_FOREVER

GO_STATE0:
    cpi  rec, BotID
    breq  MY_ID
    ret ; It wasn't our ID. Ignore it.
MY_ID:
    ldi   state, $01
    ret
COMMAND:
    ldi state, $00
    cpi rec, FREEZE
    breq DO_FREEZE
    out PORTB, rec
    ret
DO_FREEZE:
    ldi  mpr, FROZEN
    call USART_Transmit
    ret
```

```
USART_Transmit:
    lds tmp, UCSR1A
    sbrs tmp, UDRE1
    ; Load status of USART1
    ; Loop until transmit data buffer is ready
    rjmp USART_Transmit
    ; Send data
    sts UDR1, mpr
    ; Move data to transmit data buffer
    ;ldi waitcnt, 20 ; Wait for 1 second
    ;call Wait
    ret




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



;***********************************************************
;* Additional Program Includes
;***********************************************************
;-----------------------------------------------------------------
; Sub: HitRight
; Desc: Handles functionality of the TekBot when the right whisker
; is triggered.
;-----------------------------------------------------------------
HitRight:
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 Backwards 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
ldi mpr, TurnL ; Load Turn Left Command
out PORTB, mpr ; Send command to port
```

```
ldi waitcnt, WTime ; Wait for 1 second
rcall Wait ; Call wait function

; Move Forward again
ldi mpr, MovFwd ; Load Move Forwards 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:
        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 Backwards command
        out     PORTB, mpr  ; Send command to port
        ldi     waitcnt, WTime  ; Wait for 1 second
        rcall   Wait            ; Call wait function

        ; Turn right for a second
        ldi     mpr, TurnR  ; Load Turn Left Command
        out     PORTB, mpr  ; Send command to port
        ldi     waitcnt, WTime  ; Wait for 1 second
        rcall   Wait            ; Call wait function

        ; Move Forward again
        ldi     mpr, MovFwd ; Load Move Forwards 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
```