ECE 375 Lab Final

Remote and Robot (USART)

Lab Time: Thursday 10am - 12

Josh Matteson

TA Signature

1 Introduction

This lab is dedicated to understanding the USART that comes with the ATmega128 and its importance to embedded programming. We learned how to use the transmitter and receiver on the AVR board, and how it transmits packets and in what order. This along with how Bot ID's work. This involved setting up the USART1 beforehand, which involved setting the baud rate, the packet type, and the transmittion type. We also initialized the external interrupts for Ports B and D, and setting up the interrupt masks. The importance of this lab is found in the everyday use of communication between different embedded systems, which occurs everywhere.

2 Program Overview

The overall point of this program is to get two avr boards to communicate with each other, one acting as a remote, and one acting as a robot. Someone will enter one of 6 commands on the remote, and the robot will respond with the appropriete command. This is represented by the LED's instead of actual wheels. We are also able to play a freeze tag game, where a robot receives a freeze command from another robot and has to freeze. We do this by sending a command on the remote which sends it to the board, which, in turn, sends it to all other boards around it.

3 Robot.asm: Vectors

We used three basic vectors for this program, the basic ones we used for the bump bot (0002 for HitRight, and 0004 for HitLeft), and 003C for our Receive routine (USART1).

4 Robot.asm: Initialize Routine

This initialize routine initializes the stack pointer, Initializes Port B for output, D for input. We then intialized the baud rate for USART1, and then configured the receive and transmit and enabled interrupts for receive. After that, we configured the packet to be 8 long with 2 stop bits. We set the interrupt mask and triggers for the bumpers just like the previous labs. Finally, we initialized some values for memory.

5 Robot.asm: Main Program

The main program is very simple, as it just simply loops through itself waiting to be interrupted.

6 Robot.asm: USART Receive

This function controls how we handle the USART1 interrupts. We first determine if it's an ID or action command but checking if the 7th bit is set, and then branching off into separate routines. We also check if we got frozen in this routine, and jump to FreezeRobot if we did.

7 Robot.asm: CheckID

CheckID compares the value that we received to the Bot ID that we have stored, and if they match, we give a value stored in memory a one, if not give it a zero.

8 Robot.asm: CheckCommand

This function test whether we had a correct Bot ID given to us or not, and then performs the command if we did. If we did not, we jump to the end of the routine and ignore the input.

9 Robot.asm: FreezeRobot

FreezeRobot means that our robot received a freeze command from another robot, and will freeze up and not respond to controls or interrupts for 5 seconds. We also decrement the counter in this function, checking if we have been frozen three times.

10 Robot.asm: FreezeOthers

FreezeOthers is a function that gets called when the command sent to us by the remote is the freeze command. We send out a freeze signal to all other robots. Since we are sending this to everybody, our own robot gets hit, which means that we need to clear the receive buffer in order for our robot to not get frozen.

11 Robot.asm: USART Receiver end

This is part of the USART Receive Function, and gets called when we finish the function. It pops all the variables and register off the stack and returns us to main.

12 Robot.asm: Die

Die gets called when we have been frozen three times, and basically causes the robot to go into an infinite loop without a possibility of being interrupted.

13 Robot.asm: HitRight

Handles the interupt received when hitting the right bumper on the tekbot. Causes the bot to move backwards for a second, turn left for a second, and then returns.

14 Robot.asm: HitLeft

Handles the interupt received when hitting the left bumper on the tekbot. Causes the bot to move backwards for a second, turn right for a second, and then returns.

15 Remote.asm: Vectors

No vectors were used for Remote.

16 Remote.asm: Initialize Routine

The Init routine is almost exactly the same as the robot Init routine, except we don't need interrupts or values set to anything. The Stack, i/o ports, and USART1 are the same.

17 Remote.asm: Main

The main program polls the inputs from PORTD, and goes into a specific subroutine depending on which button was pressed on the remote.

18 Remote.asm: XCommand

All the subroutines that have the Command name in them perform a specific action for a specific button pushed. For example, The ForwardCommand first calls a function that sends the ID of our robot, and then sends the command to the robot. It loops the transmit finished flag from UCSR1A until we finish, and then returns back to main.

19 Remote.asm: TransmitAddress

TransmitAddress gets the address of the Bot, and then sends it to the robot before anything else. Just like the other commands, it polls until we've finished sending before continuing on with the rest of the program.

20 Both: Wait

The short function (used from the previous labs), causes the program to wait for 1 second before continuing.

21 Difficulties

By far the biggest difficulty was finding another AVR board to program with, after that, the rest of the lab was simply combining previous knowledge with some new challenges.

22 Conclusion

Overall, the importance of this lab was clearly seen and understood. Communication between multiple embedded systems is used all the time on lots of machinery. The more I learn and get an understanding of it now, the better I'll be able to understand them in the future and understand how to better program machinery in a futue job.

23 Source Code for Robot.asm

```
;***********************************************************
;*
;* Robot.asm
;*
;* The program receives signals from the remote, and
;*  performs specific actions according to it.
;*
;* This is the RECEIVE skeleton file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*  Author: Josh Matteson
;*    Date: 02/02/20017
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def  mpr2 = r17 ; Second Multi-Puporse Register
.def  waitcnt = r18 ; Wait counter
.def ilcnt = r19 ; Inner Loop Counter
.def olcnt = r20 ; Outer Loop Counter

.equ  WTime = 100 ; 1 second

.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 BotAddress = 13 ;(Enter your robot's address here (8 bits))
.equ  Command = $0100
.equ FreezeCount = $0140

;/////////////////////////////////////////////////////////////
;These macros are the values to make the TekBot Move.
;/////////////////////////////////////////////////////////////
.equ MovFwd =  (1<<EngDirR|1<<EngDirL) ;0b01100000 Move Forward Action Code
.equ MovBck =  $00 ;0b00000000 Move Backward Action Code
.equ TurnR =   (1<<EngDirL) ;0b01000000 Turn Right Action Code
.equ TurnL =   (1<<EngDirR) ;0b00100000 Turn Left Action Code
.equ Halt = ($80|1<<(EngEnR-1)|1<<(EngEnL-1)) ;0b10010000 Halt Action Code
.equ  Freeze =  0b01010101 ;Command to freeze this robot
.equ  SendFreeze = 0b11111000
.equ  SendFreezeR = 0b011110000

;***********************************************************
;* 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:
;- Right whisker
.org  $0002 ;{IRQ0 => pin0, PORTD}
rcall HitRight ; Call hit right function
reti ; Return from interrupt
```

```
;- Left whisker
.org  $0004 ;{IRQ1 => pin1, PORTD}
rcall HitLeft ; Call hit left function
reti ; Return from interrupt
;- USART receive
.org  $003C ; USART1, Rx complete interrupt
rcall USART_Receive ; Call USART_Receive
reti ; Return from interrupt

.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT:
;Stack Pointer (VERY IMPORTANT!!!!)
LDI  mpr, LOW(RAMEND) ; Low Byte of End SRAM Address
OUT  SPL, mpr ; Write byte to SPL
LDI  mpr, HIGH(RAMEND) ; High Byte of End SRAM Address
OUT  SPH, mpr ; Write byte to SPH
; I/O Ports
; Initialize Port B for output
ldi mpr, $00 ; Initialize Port B Data Register
out PORTB, mpr ; so all Port B outputs are low
ldi mpr, $FF ; Set Port B Data Direction Register
out DDRB, mpr ; for output

; Initialize Port D for input
ldi mpr, $FF ; Initialize Port D Data Register
out PORTD, mpr ; so all Port D inputs are Tri-State
ldi mpr, $00 ; Set Port D Data Direction Register
out DDRD, mpr ; for input

;USART1
;Set baudrate at 2400bps
; (16 MHz / (16 * 2400 [desired baud rate])) - 1 = 416
ldi  mpr, low(416) ; Load low byte into UBRR1L
sts  UBRR1L, mpr
ldi  mpr, high(416) ; Load high byte into UBRR1H
sts  UBRR1H, mpr
;Don't know if this is necessary, but set UCSR1A to 0b00000000
;ldi  mpr, 0b00000000
;sts  UCSR1A, mpr
;Enable receiver and enable receive interrupts, and transmit
ldi  mpr, (1<<RXEN1 | 1<<RXCIE1 | 1<<TXEN0) ;
sts UCSR1B, mpr
;Set frame format: 8 data bits, 2 stop bits
ldi mpr, (1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10 | 1<<UCSZ11) ; UCSZn1 and UCSZn0 to
; 1's for 8 data bits and USBSn set to 1 for 2 stopping bits
sts  UCSR1C, mpr
;External Interrupts

;Set the External Interrupt Mask
ldi mpr, 0b00000011 ; Enable INTO and INT1 for interrupts
out EIMSK, mpr
;Set the Interrupt Sense Control to falling edge detection
ldi mpr, 0b00001010 ; Set INT0 and INT1 to trigger on falling edge
sts EICRA, mpr

sei
;Other
ldi  YL, low(Command)
ldi  YH, high(command)
ldi mpr, 0
st Y, mpr

ldi  ZL, low(FreezeCount)
ldi  ZH, high(FreezeCount)
ldi mpr, 0b00000011
st Z, mpr
```

```
;***********************************************************
;* Main Program
;***********************************************************
MAIN:
;TODO: ???
rjmp MAIN

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

USART_Receive:

push mpr2
push  mpr
in mpr, SREG
push waitcnt ; Wait counter
push  mpr
push YL
push YH
push ZL
push ZH

lds mpr, UDR1
cpi mpr, Freeze
breq FreezeRobot
sbrc mpr, 7
rjmp CheckCommand
rjmp CheckID

CheckID:

ldi  YL, low(Command)
ldi  YH, high(command)
ld mpr2, Y

cpi  mpr, BotAddress
breq Correct
ldi mpr2, 0
st Y, mpr2

rjmp  USART_Receiver_end

Correct:

ldi mpr2, 1
st Y, mpr2

rjmp  USART_Receiver_end

CheckCommand:

ldi  YL, low(Command)
ldi  YH, high(command)
ld mpr2, Y
sbrs mpr2, 0
rjmp USART_Receiver_end
rol  mpr
;ldi waitcnt, WTime ; Wait for 1 second
;rcall Wait
cpi mpr, SendFreezeR
breq FreezeOthers
out PORTB, mpr

rjmp USART_Receiver_end

FreezeRobot:

; Turn off interupts for duration of function
```

```
cli

ldi mpr2, Halt
rol mpr2
out PORTB, mpr2
ldi waitcnt, WTime ; Wait for 1 second
rcall Wait

ldi ZL, low(FreezeCount)
ldi ZH, high(FreezeCount)
ld mpr, Z

dec mpr
breq Die
st Z, mpr

; Turn interupts back on
ldi mpr, 0b00000011
out  EIFR, mpr ; Clear External Interupt Flag Register
sei

rjmp  USART_Receiver_end

FreezeOthers:

ldi mpr, 0b01010101
out PORTB, mpr
sts UDR1, mpr
Send_Loop:
lds mpr, UCSR1A
sbrs mpr, TXC1
rjmp Send_Loop
Clear_Input:
lds  mpr, UCSR1A
sbrs  mpr, RXC1
rjmp  Clear_Input
lds mpr, UDR1

rjmp USART_Receiver_end


USART_Receiver_end:

pop ZH
pop  ZL
pop  YH
pop YL
pop  mpr
pop waitcnt
out SREG, mpr
pop  mpr
pop  mpr2
ret

Die:
rjmp  Die


HitRight: ; Begin a function with a label

push mpr ; Save mpr register
push waitcnt ; Save wait register
in mpr, SREG ; Save program state
push mpr ;

; Turn off interupts for duration of function
cli

; 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
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 Forward command
out PORTB, mpr ; Send command to port

; Turn interupts back on
ldi mpr, 0b00000011
out  EIFR, mpr ; Clear External Interupt Flag Register
sei

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

ret ; End a function with RET

HitLeft: ; Begin a function with a label

push mpr ; Save mpr register
push waitcnt ; Save wait register
in mpr, SREG ; Save program state
push mpr ;

; Turn off interupts for duration of function
cli

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

; Move Forward again
ldi mpr, MovFwd ; Load Move Forward command
out PORTB, mpr ; Send command to port

; Turn interupts back on
ldi mpr, 0b00000011 ; Left and Right bumper can interrupt again
out  EIFR, mpr ; Clear External Interupt Flag Register
sei

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

ret ; Return from subroutine

Wait:
push waitcnt ; Save wait register
push ilcnt ; Save ilcnt register
push olcnt ; Save olcnt register
; Wait function from previous labs
Loop:
```

```
ldi olcnt, 224 ; load olcnt register
OLoop:
ldi ilcnt, 237 ; load ilcnt register
ILoop:
dec ilcnt ; decrement ilcnt
brne ILoop ; Continue Inner Loop
dec olcnt ; decrement olcnt
brne OLoop ; Continue Outer Loop
dec waitcnt ; Decrement wait
brne Loop ; Continue Wait loop

pop olcnt ; Restore olcnt register
pop ilcnt ; Restore ilcnt register
pop waitcnt ; Restore wait register
ret ; Return from subroutine

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

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

24 Source Code for Robot.asm

```
;***********************************************************
;*
;* Remote.asm
;*
;* Transmits signals to the robot giving it commands.
;*
;* This is the TRANSMIT skeleton file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*  Author: Josh Matteson
;*    Date: 02/02/20017
;*
;***********************************************************

.include "m128def.inc" ; Include definition file

;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16 ; Multi-Purpose Register
.def  waitcnt = r18 ; Wait counter
.def ilcnt = r19 ; Inner Loop Counter
.def olcnt = r20 ; Outer Loop Counter

.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  WTime = 25 ; 1 second

; Use these action codes between the remote and robot
; MSB = 1 thus:
; control signals are shifted right by one and ORed with 0b10000000 = $80
.equ MovFwd =  ($80|1<<(EngDirR-1)|1<<(EngDirL-1)) ;0b10110000 Move Forward Action Code
.equ MovBck =  ($80|$00) ;0b10000000 Move Backward Action Code
.equ TurnR =   ($80|1<<(EngDirL-1)) ;0b10100000 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
.equ Freeze =  0b11111000
```

```
.equ SendComplete = 0b01100000

.equ BotAddress = 13 ;(Enter your robot's address here (8 bits))
; 0b00001101

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

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

.org $0046 ; End of Interrupt Vectors

;***********************************************************
;* Program Initialization
;***********************************************************
INIT:
;Stack Pointer (VERY IMPORTANT!!!!)
LDI  mpr, LOW(RAMEND) ; Low Byte of End SRAM Address
OUT  SPL, mpr ; Write byte to SPL
LDI  mpr, HIGH(RAMEND) ; High Byte of End SRAM Address
OUT  SPH, mpr ; Write byte to SPH
;I/O Ports
; Initialize Port B for output
ldi mpr, $00 ; Initialize Port B Data Register
out PORTB, mpr ; so all Port B outputs are low
ldi mpr, $FF ; Set Port B Data Direction Register
out DDRB, mpr ; for output

; Initialize Port D for input
ldi mpr, $FF ; Initialize Port D Data Register
out PORTD, mpr ; so all Port D inputs are Tri-State
ldi mpr, $00 ; Set Port D Data Direction Register
out DDRD, mpr ; for input
;USART1
;Set baudrate at 2400bps
; (16 MHz / (16 * 2400 [desired baud rate])) - 1 = 416
ldi  mpr, low(416) ; Load low byte into UBRR1L
sts  UBRR1L, mpr
ldi  mpr, high(416) ; Load high byte into UBRR1H
sts  UBRR1H, mpr

;Enable transmitter
ldi mpr, (1<<TXEN0)
sts UCSR1B, mpr
;Set frame format: 8 data bits, 2 stop bits
ldi mpr, (1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10 | 1<<UPM11) ; UCSZn1 and UCSZn0 to
; 1's for 8 data bits and USBSn set to 1 for 2 stopping bits
sts  UCSR1C, mpr

sei

;Other

;***********************************************************
;* Main Program
;***********************************************************
MAIN:
in mpr, PIND ; Get button input from Port D
cpi mpr, 0b11111110 ; Check for first button input
brne NEXT1 ; Continue with next check
rcall ForwardCommand ; Call the subroutine ForwardCommand
rjmp MAIN ; Loop back to main
NEXT1:
```

```
cpi mpr, 0b11111101 ; Check for 2nd button input
brne NEXT2 ; No button input, continue program
rcall BackwardCommand ; Call subroutine BackwardCommand
rjmp MAIN ; Continue through main
NEXT2:
cpi mpr, 0b11101111 ; Check for 5th button input
brne NEXT3 ; No button input, continue program
rcall TurnRCommand ; Call subroutine TurnRCommand
rjmp MAIN ; Continue through main
NEXT3:
cpi mpr, 0b11011111 ; Check for 6th button input
brne NEXT4 ; No button input, continue program
rcall TurnLCommand ; Call subroutine TurnLCommand
rjmp MAIN ; Continue through main
NEXT4:
cpi mpr, 0b10111111 ; Check for 7th button input
brne NEXT5 ; No button input, continue program
rcall HaltCommand ; Call subroutine HaltCommand
rjmp MAIN ; Continue through main
NEXT5:
cpi mpr, 0b01111111 ; Check for 8th button input
brne MAIN ; No button input, continue program
rcall FreezeCommand ; Call subroutine FreezeCommand
rjmp MAIN ; Continue through main

;***********************************************************
;* Functions and Subroutines
;***********************************************************
ForwardCommand:

rcall TransmitAddress

ldi mpr, MovFwd
out PORTB, mpr
sts UDR1, mpr


Forward_Loop:
lds mpr, UCSR1A
cpi mpr, SendComplete
brne Forward_Loop

ret

BackwardCommand:

rcall TransmitAddress

ldi mpr, MovBck
out PORTB, mpr
sts UDR1, mpr

Backward_Loop:
lds mpr, UCSR1A
cpi mpr, SendComplete
brne Backward_Loop

ret

TurnRCommand:

rcall TransmitAddress

ldi mpr, TurnR
out PORTB, mpr
sts UDR1, mpr

TurnR_Loop:
lds mpr, UCSR1A
cpi mpr, SendComplete
```

```
brne TurnR_Loop

ret

TurnLCommand:

rcall TransmitAddress

ldi mpr, TurnL
out PORTB, mpr
sts UDR1, mpr

TurnL_Loop:
lds mpr, UCSR1A
cpi mpr, SendComplete
brne TurnL_Loop

ret

HaltCommand:

rcall TransmitAddress

ldi mpr, Halt
out PORTB, mpr
sts UDR1, mpr

Halt_Loop:
lds mpr, UCSR1A
cpi mpr, SendComplete
brne Halt_Loop

ret

;FreezeCommand:

;ldi mpr, 0b01010101
;out PORTB, mpr
;sts UDR1, mpr

;Freeze_Loop:
;lds mpr, UCSR1A
;sbrs mpr, TXC1
;rjmp Freeze_Loop

;ldi waitcnt, WTime ; Wait for 1 second
;rcall Wait

;ret

FreezeCommand:

rcall TransmitAddress

ldi mpr, Freeze
out PORTB, mpr
sts UDR1, mpr

Freeze_Loop:
lds mpr, UCSR1A
cpi mpr, SendComplete
brne Freeze_Loop

ldi waitcnt, WTime ; Wait for 1 second
rcall Wait

ret

TransmitAddress:
```

```
ldi mpr, BotAddress
sts UDR1, mpr

Transmit_Loop:
lds mpr, UCSR1A
cpi mpr, SendComplete
brne Transmit_Loop

ret
; Wait function from previous labs
Wait:
push waitcnt ; Save wait register
push ilcnt ; Save ilcnt register
push olcnt ; Save olcnt register

Loop:
ldi olcnt, 224 ; load olcnt register
OLoop:
ldi ilcnt, 237 ; load ilcnt register
ILoop:
dec ilcnt ; decrement ilcnt
brne ILoop ; Continue Inner Loop
dec olcnt ; decrement olcnt
brne OLoop ; Continue Outer Loop
dec waitcnt ; Decrement wait
brne Loop ; Continue Wait loop

pop olcnt ; Restore olcnt register
pop ilcnt ; Restore ilcnt register
pop waitcnt ; Restore wait register
ret ; Return from subroutine

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

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