ECE 375 LAB 7

Arcade Basketball

Lab Time: Wednesday 5-7

Anton Bilbaeno

Mushfiqur Sarker

Introduction

The purpose of this lab was to apply what we have learned throughout the term in order to implement a project which may or may not include: interrupts, the LCD display, and IR communication.

We chose to implement an arcade basketball game.

Program Overview

The arcade basketball program requires two microcontrollers. The first microcontroller functions as the hoop or backboard. When a bucket is scored, this microcontroller will communicate with the second indicating as such. The second microcontroller is responsible for keeping time, maintaining the score and writing to the LCD display.

Backboard / Hoop

Main Function

The backboard uses polling to check for button presses (or whisker presses). We continuously check the state of PIND and compare it to specific binary values. When there is a match, we load the HitID (an 8-bit binary value) into the UDR and transmit the value. If and when the second microcontroller receives this specific code, it will increment a register that it uses to keep track of the score.

Initialization

To be able to use the IR transmitter of the ATMega 128, we need to do three extra things in the initialize section. First, we need to define the baud rate. To set the baud rate, we need to write to the UBRR register. For this lab we chose a baud rate of 2400bps. Second, we need to set the 'transmit enable bit' (TXEN) to 1. This bit is located on the UCSR1B register. Third, we need to let the ATMega 128 know what the format will be for transmitting data. For this lab, we define a data size of 8bits and use 2 stop bits.

'ReadyCheck' Function

This function gets called when we want to transmit the HitID. The function checks if a specific bit on one of the transmit/receive status registers is set. If this bit (UDRE) is set, the buffer is empty, and we are allowed to write to it. This function loops to itself if this condition is not met, so it may be thought of as a wait function.

RECEIVER / DISPLAY

MAIN ROUTINE

The main routine is only utilized at the start of the game. If any buttons 5-8 on PIND are pressed, we initialize the LCD display to display “Time:” and “Score:”. At this point the timer will enter into the Game function.

INITIALIZATION

Similarly to the transmit code, we need to set up the USART Control and Status registers. Again, the baud rate is set to 2400bps. Because we set the data transfer size to 8bits for transmit, we need to set it to the same for our receive code. We also need to define 2 stop bits. We also need to set the ‘receive enable bit’ (RXEN) on USART Status Register B. This simply enables the IR receiver. We also need to set the ‘RX complete interrupt enable’ bit (RXCIE). We will only be able to get a USART Receive Complete interrupt if this bit is set. Finally, because our remote TekBot has the ability to receive AND transmit, we need to set the TXEN bit as we did for our transmit initialization routine.

We utilize the Timer/Counter2 to keep track of time. In order to change the speed at which the timer counts, we must set the CSn0 – CSn2 bits in the TCCR2 (the timer/counter control register). We also set the WGMn0 – WGMn1 bits to ‘CTC’ (clear timer on compare match mode). OCIEn needs to be set to 1. This enables the Timer/Counter2 compare match if interrupts are also enabled.

GAME

This function first checks to see if the remaining time is equal to 61. If it is, the game is just beginning and the LCD display should be initialized accordingly.

Next, the UpdScore function is called.

Finally, make sure there is some time left. If there isn’t, end the game by breaking to the GG function.

Otherwise this function will stay in a loop. It will continuously update the score.

GG

This function does one thing: update the first line of the LCD display to alert the user the game is over and to press the 8th button on PORTD if they want to restart.

UPDSCORE

This function takes care of writing the score to the LCD display. The LCDDriver.asm contains a function called “Bin2ASCII”. This takes cares of converting the value of the score contained in our register to text which we can easily write to the LCD display.

RECEIVE

This function is called when we receive an interrupt from IR communication. We check to see if the data we are receiving is our specific HitID. If it is, the score register is incremented. The writing of the score to the display is taken care of in the Game function.

CONCLUSION

This somewhat open ended lab tested our resourcefulness with the AVR microcontroller. Although we were not given any specific instructions about how to implement our program, we had already completed similar tasks in the past. This lab was a great challenge.

```
;***********************************************************
;*
;*    ECE 375: Lab 7 - Arcade Basketball
;*
;*    LCD Display Source Code
;*
;***********************************************************
;*
;*     Authors: Anton Bilbaeno, Mushfiqur Sarker
;*        Date: 3/5/11
;*
;***********************************************************

.include "m128def.inc"              ; Include definition file

;***********************************************************
;*    Internal Register Definitions and Constants
;***********************************************************
.def  mpr = r16                     ; Multipurpose register
.def  zero = r2                     ; Zero flag
.def  counter = r4
.def  TimerCnt = r6

; LCD Driver uses registers 17-22
.def  ReadCnt = r23                 ; Register required for LCD Driver for
writing to LCD Displays
.def  remTime = r24
.def  curScore = r25

.equ  HitID = 0b00110011
.equ  WaitTime = $FF
.equ  writeSpace = $0200

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

;-----------------------------------------------------------
; Interrupt Vectors
```

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

;- USART receive
.org  $0002
            rcall       INC_SCORE
            reti

.org  $0004
            rcall       RESET
            reti

.org  $003C
            rcall       Receive
            reti

.org  $0014
            rcall       UpdTime
            reti

.org  $0046                         ; End of Interrupt Vectors

;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:                                       ; The initialization routine
; Initialize stack pointer
            ldi         mpr, HIGH(RAMEND)
            out         SPH, mpr
            ldi         mpr, LOW(RAMEND)
            out         SPL, mpr

; Set baud rate at 2400bps
            ldi mpr, high(416)              // $01A0 = 416 for U2X = 0
            sts UBRR1H, mpr
            ldi mpr, low(416)
            sts UBRR1L, mpr

;Enable receiver and enable receive interrupts
            ldi mpr, (1<<RXEN1)|(1<<RXCIE1)
            sts UCSR1B, mpr

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

; Initialize Port D for input
            ldi   mpr, 0b11111000   ; pin 1,0 whiskers. pin 2 rx, pin 3 tx
output
            out         DDRD, mpr
            ldi         mpr, 0b11110011
            out         PORTD, mpr
```

```
; Timer/Counter2 initialization
            ldi         mpr, (1<<WGM21|0<<CS22|1<<CS21|1<<CS20)
            out         TCCR2, mpr
            ldi         mpr, $30
            out         OCR0, mpr
            ldi         mpr, (1<<OCIE2)

            out         TIMSK, mpr
            ldi         mpr, 0
            mov         counter, mpr
            clr         TimerCnt

; Initialize external interrupts
; Set the Interrupt Sense Control for whiskers to RISING EDGE
            ldi         mpr, (1<<ISC01)|(1<<ISC00)|(1<<ISC11)|(1<<ISC10)
            sts   EICRA, mpr
            ldi         mpr, $00
            out         EICRB, mpr   ; NOTE: must initialize both EICRA and
EICRB

; Set the External Interrupt Mask
            ldi         mpr, (1<<INT0)|(1<<INT1)
            out         EIMSK, mpr

; Initialize LCD Display
            rcall       LCDInit

RESET:
; Initialize Port B
            ldi   mpr, 0b11111111
            out   DDRB, mpr
            ldi   mpr, 0b00000000
            out         PORTB, mpr

; Initialize pre-defined registers
            ldi   curScore, 0
            ldi         remTime, 31

; Init line 1 variable registers

            ldi         ZL, low(INTRO1<<1)
            ldi         ZH, high(INTRO1<<1)
            ldi         YL, low(LCDLn1Addr)
            ldi         YH, high(LCDLn1Addr)
            ldi         ReadCnt, LCDMaxCnt

INIT_LINE1:

            lpm         mpr, Z+                 ; Read Program memory
            st          Y+, mpr                 ; Store into memory
            dec         ReadCnt                 ; Decrement Read Counter
            brne  INIT_LINE1        ; Continue until all data is read
            rcall       LCDWrLn1          ; WRITE LINE 1 DATA
```

```
; Init line 2 variable registers

            ldi         ZL, low(INTRO2<<1);
            ldi         ZH, high(INTRO2<<1);
            ldi         YL, low(LCDLn2Addr)
            ldi         YH, high(LCDLn2Addr)
            ldi         ReadCnt, LCDMaxCnt

INIT_LINE2:

            lpm         mpr, Z+                 ; Read Program memory
            st          Y+, mpr                 ; Store into memory
            dec         ReadCnt                 ; Decrement Read Counter
            brne  INIT_LINE2        ; Continue until all data is read
            rcall       LCDWrLn2            ; WRITE LINE 2 DATA

;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN:                                       ; The Main program
            cli
            ; Display the strings on the LCD Display
            in                  mpr, PIND
            cpi         mpr, 0b10111111   ;compare to 0's
            breq        Game        ;if what we were looking for then jump

            in                  mpr, PIND
            cpi         mpr, 0b11011111   ;compare to 0's
            breq        Game        ;if what we were looking for then jump

            in                  mpr, PIND
            cpi         mpr, 0b01111111   ;compare to 0's
            breq        Game        ;if what we were looking for then jump

            in                  mpr, PIND
            cpi         mpr, 0b11101111   ;compare to 0's
            breq        Game        ;if what we were looking for then jump

            rjmp        MAIN              ; jump back to main and create an
infinite
                                              ; while loop.  Generally,
every main program is an
                                              ; infinite while loop, never
let the main program
                                              ; just run off

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

;-----------------------------------------------------------
; Func: Start
; Desc: Once button is pressed, this is the main function of
```

```
;           the program.
;-----------------------------------------------------------
Game:
            sei
            cpi         remTime, 31
            breq  InitBoard


            rcall UpdScore


            cpi         remTime, 0
            breq  GG

            rjmp  Game

GG:
            cli
            ldi         ZL, low(GAMEOVERTXT<<1)
            ldi         ZH, high(GAMEOVERTXT<<1)
            ldi         YL, low(LCDLn1Addr)
            ldi         YH, high(LCDLn1Addr)
            ldi         ReadCnt, LCDMaxCnt

GG_Line1:

            lpm         mpr, Z+                 ; Read Program memory
            st          Y+, mpr                 ; Store into memory
            dec         ReadCnt                 ; Decrement Read Counter
            brne  GG_Line1          ; Continue until all data is read
            rcall       LCDWrLn1          ; WRITE LINE 1 DATA

UserReset:
            in          mpr, PIND
            cpi   mpr, 0b01111111   ;compare to 0's
            breq  RESET             ;if what we were looking for then jump

            rjmp  UserReset


;-----------------------------------------------------------
; Func: InitBoard
; Desc: When receive interrupt is triggerd, we go here to
;           update line 2 of the LDC Display
;-----------------------------------------------------------

InitBoard:
            cli
            ldi   curScore, 0
            ldi         ZL, low(TIMETXT<<1)
            ldi         ZH, high(TIMETXT<<1)
            ldi         YL, low(LCDLn1Addr)
            ldi         YH, high(LCDLn1Addr)
            ldi         ReadCnt, LCDMaxCnt
```

```
GAME_LINE1:

            lpm         mpr, Z+                 ; Read Program memory
            st          Y+, mpr                 ; Store into memory
            dec         ReadCnt                 ; Decrement Read Counter
            brne  GAME_LINE1        ; Continue until all data is read
            rcall       LCDWrLn1          ; WRITE LINE 1 DATA


; Init line 2 variable registers

            ldi         ZL, low(SCORETXT<<1);
            ldi         ZH, high(SCORETXT<<1);
            ldi         YL, low(LCDLn2Addr)
            ldi         YH, high(LCDLn2Addr)
            ldi         ReadCnt, LCDMaxCnt

GAME_LINE2:

            lpm         mpr, Z+                 ; Read Program memory
            st          Y+, mpr                 ; Store into memory
            dec         ReadCnt                 ; Decrement Read Counter
            brne  GAME_LINE2        ; Continue until all data is read
            rcall       LCDWrLn2          ; WRITE LINE 2 DATA

            ldi         mpr, 0b11110000
            out         PORTB, mpr
            sei

            rjmp  Game
;-----------------------------------------------------------
; Func: Score
; Desc: When receive interrupt is triggerd, we go here to
;           update line 2 of the LDC Display
;-----------------------------------------------------------

UpdScore:
            cli
            push  ReadCnt
            push  line
            push  count
            push  counter
            push  XH
            push  XL
            push  mpr
            in          mpr, SREG
            push  mpr

            ldi         XL, low(writeSpace)
            ldi         XH, high(writeSpace)

            ldi         count, 3
            ldi         mpr, ' '
```

```
ScoreLoadSpace:
            st          X+, mpr
            dec         count
            brne  ScoreLoadSpace

            mov         mpr, curScore

            ldi         XL, low(writeSpace)
            ldi         XH, high(writeSpace)
            /*
            ldi         YL, low(LCDLn2Addr)           determined by the
            ldi         YH, high(LCDLn2Addr)    ldi         line, 2
      (below)
            */
            rcall Bin2ASCII


            ldi         ReadCnt, 3
            ldi         line, 2


            ldi         count, 13

            /*
            cpi         curScore, 0
            breq  CountSet0

            cpi         curScore, 10
            breq  CountSet1

            cpi         curScore, 100
            breq  CountSet2
            */

ScoreWrite:
            ld          mpr, X+
            rcall       LCDWriteByte
            inc   count
            dec         ReadCnt
            brne  ScoreWrite

            pop         mpr
            out         SREG, mpr
            pop   mpr
            pop         XL
            pop         XH
            pop         counter
            pop         count
            pop         line
            pop         ReadCnt

            sei
            ret
```

```
/*
CountSet0:
            ldi         count, 15
            clr         zero
            rjmp  ScoreWrite

CountSet1:
            ldi         count, 14
            clr         zero
            rjmp  ScoreWrite

CountSet2:
            ldi         count, 13
            clr         zero
            rjmp  ScoreWrite
*/

;-----------------------------------------------------------
; Func: Timer
; Desc:
;-----------------------------------------------------------


UpdTime:
            push  ReadCnt
            push  line
            push  count
            push  counter
            push  XH
            push  XL
            push  mpr
            in          mpr, SREG
            push  mpr

            inc         TimerCnt
            brne  EndTime

            ldi         XL, low(writeSpace)
            ldi         XH, high(writeSpace)
            /*
            ldi         YL, low(LCDLn2Addr)           determined by the
            ldi         YH, high(LCDLn2Addr)    ldi         line, 1
      (below)
            */
            ldi         count, 3                      ; count defined by
LCDDriver.asm
            ldi         mpr, ' '                      ; blank character
TimerLoadSpace:
            st          X+, mpr
            dec         count
            brne  TimerLoadSpace

            ; Convert binary counter to ASCII
```

```
            dec         remTime
            mov         mpr, remTime

            ldi         XL, low(writeSpace)
            ldi         XH, high(writeSpace)
            /*
            ldi         YL, low(LCDLn1Addr)
            ldi         YH, high(LCDLn1Addr)
            */
            rcall Bin2ASCII

            ; Write data to LCD Display
            ldi         ReadCnt, 3
            ldi         line, 1
            ldi         count, 13

TimerWrite:
            ld          mpr, X+
            rcall LCDWriteByte
            inc         count
            dec         ReadCnt
            brne  TimerWrite

EndTime:
            pop         mpr
            out         SREG, mpr
            pop   mpr
            pop         XL
            pop         XH
            pop         counter
            pop         count
            pop         line
            pop         ReadCnt


            ret

Hang:
            push  curScore              ; Save curScore register
            push  remTime               ; Save remTime register

Loop: ldi         remTime, 224          ; load remTime register
OLoop:      ldi         curScore, 237           ; load curScore register
ILoop:      dec         curScore                ; decrement curScore
            brne  ILoop             ; Continue Inner Loop
            dec         remTime                 ; decrement remTime
            brne  OLoop             ; Continue Outer Loop
            dec         ReadCnt                 ; Decrement wait
            brne  Loop              ; Continue Wait loop

            pop         remTime                 ; Restore remTime register
            pop         curScore          ; Restore curScore register
                                                ; Restore wait register
            ret
```

```
;-----------------------------------------------------------
; Desc: Receive functions
;-----------------------------------------------------------

Receive:
            out         PORTB, mpr
            rcall RCV_CONFIRM
            cpi         mpr, HitID
            breq  INC_SCORE

            rjmp        RCV_COMPLETE

RCV_CONFIRM:

            lds         mpr, UCSR1A
            sbrs  mpr, RXC1
            rjmp  RCV_CONFIRM
            lds         mpr, UDR1

            ret

INC_SCORE:
            inc         curScore

RCV_COMPLETE:

            ret

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;           beginning of your functions
;-----------------------------------------------------------



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

;-----------------------------------------------------------
; An example of storing a string, note the preceeding and
; appending labels, these help to access the data
;-----------------------------------------------------------
INTRO1:
.DB         "Press button 4-8"            ; Storing the string in Program
Memory

INTRO2:
.DB   "on PortD 2 start"

TIMETXT:
.DB         "Time left:      "
```

```
SCORETXT:
.DB         "Score:          "

GAMEOVERTXT:
.DB         "GG.Press8toreset"

; for testing
;           "Score:          "

;***********************************************************
;*    Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"            ; Include the LCD Driver









;***********************************************************
;*
;*    ECE 375: Lab 7 - Arcade Basketball
;*
;*    Backboard Source Code
;*
;***********************************************************
;*
;*     Author: Anton Bilbaeno, Mushfiqur Sarker
;*       Date: 3/5/11
;*
;***********************************************************

.include "m128def.inc"              ; Include definition file
```

```
;***********************************************************
;*    Internal Register Definitions and Constants
;***********************************************************
.def  mpr = r16                     ; Multipurpose register
.def  waitcnt = r17
.def  ilcnt = r18
.def  olcnt = r19
.def  zero = r2                     ; Zero register, set to zero in INIT,
useful for calculations
.def  data = r24
.def  correctID = r22
.def  recCheck = r21
.def  sent = r25

; Constants for interactions such as
.equ  WskrR = 0                     ; Right Whisker Input Bit
.equ  WskrL = 1                     ; Left Whisker Input Bit
.equ  WTime = 25                    ; Time to wait in wait loop

.equ  HitID = 0b00110011

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

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

/*
.org  $0002
            rcall Hit0
            reti

.org  $0004
            rcall Hit1
            reti

.org  $000A
            rcall Hit0
            reti

.org  $000C
            rcall Hit1
            reti
*/

.org  $0046                         ; End of Interrupt Vectors

;-----------------------------------------------------------
; Program Initialization
```

```
;-----------------------------------------------------------
INIT:
            ; The initialization routine
            ; Initialize Stack Pointer
            ldi         mpr, HIGH(RAMEND)
            out         SPH, mpr
            ldi         mpr, LOW(RAMEND)
            out         SPL, mpr

            clr         zero

            ; Set baud rate at 2400bps
            ldi mpr, high(416)            // $01A0 = 416 for U2X = 0
            sts UBRR1H, mpr
            ldi mpr, low(416)
            sts UBRR1L, mpr

            ;Enable receiver and enable receive interrupts
            ldi mpr, (1<<TXEN1)
            sts UCSR1B, mpr

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

            ; Initialize Port D for input
            ldi   mpr, 0b11001000                    ; pin 4,5,1,0 whiskers.
pin 2 rx, pin 3 tx output
            out         DDRD, mpr
            ldi         mpr, 0b11110011
            out         PORTD, mpr

            ; Initialize Port B
            ldi   mpr, 0b11111111
            out   DDRB, mpr
            ldi   mpr, 0b00000000
            out         PORTB, mpr


            ; Initialize external interrupts
            ; Set the Interrupt Sense Control for whiskers to RISING EDGE
            ldi         mpr, (1<<ISC01)|(0<<ISC00)|(1<<ISC11)|(0<<ISC10)
            sts   EICRA, mpr
            ldi         mpr, (0<<ISC51)|(1<<ISC50)|(0<<ISC41)|(1<<ISC40)
            out         EICRB, mpr

            ; NOTE: must initialize both EICRA and EICRB

            ; Set the External Interrupt Mask
            ldi         mpr, (1<<INT0)|(1<<INT1)|(1<<INT4)|(1<<INT5)
            out         EIMSK, mpr

            sei
```

```
;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN: ; The Main program


            ; Constantly check if pin hit
            ; PIN 0
            in              mpr, PIND
            cpi         mpr, 0b11111110         ;compare to 0's
            breq        Hit0                        ;if what we were
looking for then jump

            ; PIN 1
            in              mpr, PIND
            cpi         mpr, 0b11111101         ;compare to 0's
            breq        Hit1                    ;if what we were looking for
then jump

            ldi             waitcnt, WTime
            rcall       Wait

            rjmp        MAIN            ; Create an infinite while loop to
signify the
                                                ; end of the program

ReadyCheck: ; infinite loop if UDRE is not set, if UDRE is set, then UDR can
be written to
            lds         mpr, UCSR1A
            sbrs  mpr, UDRE1

            rjmp  ReadyCheck

            ret

Hit0:
            push  mpr

            rcall       ReadyCheck
            ldi         mpr, HitID
            sts         UDR1, mpr


            ldi         mpr, 0b01010101
            out         PORTB, mpr

            pop   mpr


            ret

Hit1:

            push  mpr
```

```
            rcall       ReadyCheck
            ldi         mpr, HitID
            sts         UDR1, mpr


            ldi         mpr, 0b10101010
            out         PORTB, mpr

            pop   mpr


            ret

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
```