ECE 375 LAB 7

Remotely connected rock paper scissors

Lab Time: Friday 4-6

Arthur Kichatov

Liuqiao Song

Introduction

The objective of this lab is to show understanding of how to configure and utilize sixteen bit timer and counters as well as utilizing USART1 on the ATmega32U4 microcontroller board. Another predominate goal of conducting this lab is to be able to grasp the timing and patterns to be able to recreate the rock paper scissors game using two Atmega32U4 controllers. The intention of this lab is to make an application of a sixteen bit timer/counter and utilizing USART1 communication so that the two boards can communicate with each other.

Program Overview

The takeaway of this program revolves around the code being written in assembly language and permitting the Bots to communicate with each other. To manage this, USART1 needs to be initialized correctly so that when a button is pressed on one controller, the code will not proceed unless a button is pressed on the second controller. The user will then have a time limit to input a response, in which then the program will compare the two responses and give a verdict on the winner. If no buttons are pressed in the time span, the program will reset and both players will have to start over. The game should function exactly like a rock, paper, scissors game except the response will depend on button inputs from both controllers, aka, players.

Initialization Routine

In this routine, the stack pointer is initialized so that the pointer can be set to the low and high bits of memory. The outputs for port B and input for port D will also be initialized. Furthermore, initialization of the LCD screen is necessary so that the screen can display both players responses. The external interrupts would be in this function in order to execute this lab. These interrupts will react to the falling edge. The sixteen bit timer/counter will need to be configured accordingly for this lab to be done correctly. Normal mode is the mode of operation. In terms of counting, it increments to the maximum value where rock beats scissors, scissors beat paper, and paper beats rock. The Leds will be illuminated and disabled depending on the time that is left before a reset of the game.

Main Routine

The main routine is carried out when the user decides to press PD4. After selecting a choice, the LEDS will begin flashing down to represent a timer. Additionally, pushing PD7 will also begin the process as it will act like a reset button for the game. And end the game after a few rounds

PD7 Routine

This routine is responsible for Starting the game. Before the game can begin, both players will have to press the button. This function will be utilizing external interrupts and Usart since both controllers need to respond before the game can begin.

PD4 Routine

This routine, users can change gesture in the order of rock, paper, scissors, rock, paper, ect. This function will wrap once the options were exhausted. Once both players have chosen, the game will then decide a winner based on a comparison of both responses.

C_timer Routine

This routine is responsible for the timer that starts once an action by the user is set. This routine utilizes four LEDs from PB7:4 and will individually turn off at 1.5 second delay. Once the timer reaches zero, the game will automatically start over and display the message, “game over”. In order to start a new game, both users will need to trigger the PD7 routine once more.

RECEIVE ROUTINE

This routine is to check the buffer of another board if there is already data in it. If the buffer is zero which means we could send data. If it is not zero, we need to wait and check again. And compare the data in the receiver have to see who will be the winner and give the result.

Difficulties

This lab felt particularly difficult this time. The lab almost covers all the knowledge we learned in the previous lab and needs another board to send data. The first difficult part we meet is to choose between using interrupt or the loop. We choose to use the interrupt. And another difficult part is how we check the buffer in another board so that we can send data. But we are not doing well in that part. What we realize is only to show some instructions on the LCD screen. There is another reason for this was because we could not effectively test our code during the long weekend. This is because this lab requires two Atmega32u4 boards as well as 3 more wires. This was something we did not have access to during the long weekend. Meaning this project realistically was only one weeklong. In the future, we will try to think more about designing the code and making it more efficient.

CONCLUSION

The main intention was to grasp how the configuration of the sixteen-bit timers and counters as well as the use of USART1 so that both boards can communicate with each other. This lab required comprehension in being able to implement PD4 and PD7 routines. The goal that is worth mentioning is being able to understand how USART1 works in combination with another board. This helps enhance previous knowledge by showing the concepts and objections of this lab. It also showcased understanding and continuing familiarity with microchip studio software.

SOURCE CODE

```
;***********************************************************

;*

;*          Author: Arthur Kichatov, Liu Song

;*           Date: 12/2/2022

;*

;***********************************************************


.include "m32U4def.inc"       ; Include definition file
```

```
;***********************************************************
;*  Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16             ; Multi-Purpose Register
.def        OuterLoop = r17
.def        InnerLoop = r18
.def        Change = r19
.def        Input = r23
.def        Counter = r24
; Use this signal code between two boards for their game ready
.equu   SendReady = 0b11111111


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


;***********************************************************
;*  Interrupt Vectors
;***********************************************************
.org    $0000
   rjmp    INIT    ;initlization


.org        $0002
rcall       P_D4 ;change current gesture
reti
```

```
.org	$0004
rcall	P_D7 ;start/ready
reti

.org	$0028
rcall	C_timer ;count down indicator
reti

.org	$00032
rcall	Compare  ;function that decides winner
reti

.org   $0056		; End of Interrupt Vectors

;***********************************************************
;*  Program Initialization
;***********************************************************
INIT:
;Stack Pointer initliazation
ldi		mpr, low(RAMEND) ; load spl with low byte
out		SPL, mpr ;put mpr into spl
ldi		mpr, high(RAMEND) ; load spH with high byte
out		SPH, mpr ; put mpr into sph

rcall	LCDInit
rcall	LCDBacklightOn
```

```
; Initialize Port B for output
ldi     mpr, $FF        ; Set Port B Data Direction
out     DDRB, mpr       ; for output
ldi     mpr, $00        ; Initialize Port B Data Register
out     PORTB, mpr      ; so all Port B outputs are low

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


;Set baudrate at 2400bps
ldi     mpr, 0xCF       ;from the lab ppt
sts     UBRR1L, mpr     ; store from mpr
ldi     mpr, 0x00       ; load 0x00 to mpr
sts     UBRR1H, mpr     ; store from mpr

ldi     mpr, (1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1) ;values from data sheet
sts     UCSR1B, mpr     ;store from mpr

ldi     mpr, (1<<USBS1)|(3<<UCSZ10)     ;values from avr data sheet
sts     UCSR1C, mpr     ;store from mpr
```

```
; initalize timer/counter1
ldi         mpr, 0b00000000       ; Normal mode
sts         TCCR1A, mpr           ; Time/Counter 1 Control Register A

ldi         mpr, 0b00000100       ;falling edge
sts         TCCR1B, mpr ; Time/Counter 1 Control Register B

; initalize External interrupts
ldi         mpr,   (1<<ISC01)|(0<<ISC00)|(1<<ISC11)|(0<<ISC10) ;from data sheet
sts         EICRA, mpr

; Initalize LCD and program to data memory transfer
ldi         ZL, LOW(2*STRING_START) ;initalizing Z low to the stack
ldi         ZH, HIGH(2*STRING_END) ;initalizing Z high to the stack

ldi         XL, LOW(Welcome)      ; Having X Low loaded by having the Welcome string placed in x
ldi         XH, HIGH(Welcome)     ; Having X High loaded by having the Welcome string placed in x

rcall   Data_Mem      ;call function
rcall   LCDClr    ;call function from LCDDRIVER

SEI
```

```
;***********************************************************
;*  Main Program
;***********************************************************
MAIN:


ldi         XL, LOW(Welcome)      ; Having X Low loaded by having the Welcome string placed in x
ldi         XH, HIGH(Welcome)     ; Having X High loaded by having the Welcome string placed in x

ldi         YL, $00   ; Initializing the Y Low addresses to the stack
ldi         YH, $01   ; Initializing the Y High addresses to the stack

rcall   Transfer ;call function

rcall   LCDWrite        ;call function from LCDDRIVER

ldi         Change, 0b00000000      ;load to change

ldi         mpr, 0b11111111         ;load to mpr
out         EIFR, mpr         ;out interrupt

ldi         mpr, (1<<INT1)
out         EIMSK, mpr        ;out interrupt

rcall Start_Loop   ;call function

rcall Receiver_Full        ;call function
```

```
rcall Counter_done          ;call function

rjmp    MAIN

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

Data_Mem:

push    mpr     ;push to stack

ldi             OuterLoop, 16   ;load 16 to outerloop

Loop16:
lpm             mpr, Z+  ;load post increment Z to mpr
st              X+, mpr ;store post increment X to mpr

dec             OuterLoop
brne    Loop16

pop             mpr             ;pop to stack

ret

Transfer:
```

```
push	mpr	;push to stack

push	OuterLoop		;push to stack


ldi			OuterLoop, 16	;load 16 to outerloop


Loop16_2:

ld			mpr, X+	;load post increment X to mpr

st			Y+, mpr	;store post increment Y to mpr


dec			OuterLoop		;decrement outerloop

brne	Loop16_2		;jump back to loop


pop			OuterLoop		;pop to stack

pop			mpr	;pop to stack


ret


Start_Loop:

sbrs	Change, 0		;skip if bit is set to 0

rjmp	Start_Loop		;return to start


Receiver_Full:

sbrs	Input, 7 ;skip if bit is set to 7

rjmp	Receiver_Full	;return to receiver


push	mpr
```

```
ldi         mpr, $48          ;load to mpr
sts         TCNT1H, mpr
ldi         mpr, $1B          ;load to mpr
sts         TCNT1L, mpr

ldi         mpr, 0b11110000
out         PORTB, mpr        ;out to portb

ldi         counter, $0F      ;load to counter

ldi         mpr, $01          ;load to mpr
out         TIFR1, mpr

ldi         mpr, $01          ;load to mpr
sts         TIMSK1, mpr       ;store timer

pop         mpr     ;pop to stack

ret

Counter_done:
sbrc    Counter, 0 ;skip if bit is set to 0
rjmp    Counter_done     ;return to counter
ldi         mpr, $00          ;load to mpr
sts         TIMSK1, mpr       ;stor timer
```

```
P_D7:
push	mpr	;push to stack

ldi		mpr, (0<<INT1)
out		EIMSK, mpr		;store mpr to interrupt

ldi		mpr, 0b11111111		;load to mpr
sts		UDR1, mpr	;store mpr to UDR1

ldi		Change, 0x01	;load 0x01 to change


pop		mpr	;pop mpr

ret



Compare:
lds		Input, UDR1	;load UDR1 from data sheet
ret




C_timer:
push	mpr	;push mpr to stack

ldi		mpr, $48	;load vlaue to mpr
```

```
sts         TCNT1H, mpr
ldi         mpr, $1B        ;load vlaue to mpr
sts         TCNT1L, mpr

lsr         counter

ldi         mpr, $10        ;load vlaue to mpr
mul         Counter, mpr
out         PORTB, r0       ;r0 to portb

pop         mpr     ; pop mpr to stack

ret

P_D4:
rcall   LCDClr  ;LCDDriver
ret

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

;-----------------------------------------------------------
; An example of storing a string. Note the labels before and
; after the .DB directive; these can help to access the data
;-----------------------------------------------------------
```

```
STRING_START:
   .DB          "Welcome!     "        ;declearing data in program mem
STRING_END:

STRING_ROCK:
.DB             "ROCK      "           ;declearing data in program mem
STRING_ROCK_END:

STRING_PAPER:
.DB             "PAPER     "           ;declearing data in program mem
STRING_PAPER_END:

STRING_SCISSOR:
.DB             "SCISSOR   "           ;declearing data in program mem
STRING_SCISSOR_END:


.DSEG
.org    $0120
Welcome:
.BYTE   16
Welcome_end:


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

```
;**********************************************************
.include "LCDDriver.asm"		; Include the LCD Driver
```