**REQUIREMENTS NOT MET**

Section 3

**PROBLEMS ENCOUNTERED**

I was a little confused with how exactly to do section 3. I tried to skip it and come back to it, but I was using too much time on it that I needed to use to focus on other classes and things going on in life.

**FUTURE WORK/APPLICATIONS**

This gives us experience passing information in between multiple sources which is extremely helpful. Just by creating simple programs to interact with the ATxmega128A1U through our computers we can see how impactful serial communication can be. If we were not able to pass data the way we do, we would not be able to have the complex systems that we all use everyday.

**PRE-LAB EXERCISES**

1. The sampling rate of a UART receiver is usually faster than the baud rate of the overall system. Why is this so?
2. What is the maximum possible baud rate for asynchronous communication within the USART system of the ATxmega128A1U, assuming that the microcontroller has a system clock frequency of 2 MHz and that the USART “double-speed mode” is disabled (i.e., the relevant bit CLK2X is set to 0)? In addition to the maximum rate, provide the values of the relevant registers used to configure that rate. Whenever appropriate, support your answer with calculations.

fPER = 2,000,000

BSCALE = -7

BSEL = 0

16,000,000 Hz

sts USARTD0\_BAUDCTRLA, (low(BSEL))

sts USARTD0\_BAUDCTRLB, ( (BSCALE<<4) | high(BSEL) )

1. In the context of the USART system within the ATxmega128A1U, how many buffers (i.e., memory locations that store temporary data) are used by a transmitter? How many are used by a receiver? Additionally, for both transmitters and receivers, explain how the use of buffers provides greater flexibility to an application involving these components.
2. If an asynchronous serial communication protocol of 8 data bits, one start bit, one stop bit, no parity, and baud rate of 150 kHz was chosen, calculate how many seconds it would take to transmit the ASCII character string “Dr. Schwartz saw seven slick slimy snakes slowly sliding southward.” (This string has 67 characters.) Show all work.

8 + 1 + 1 = 10 total bits

**PSEUDOCODE/FLOWCHARTS**

**N/A**

**PROGRAM CODE**

**SECTION 2 (consisted of making USART\_INIT, and OUT\_CHAR)**

**MAIN code:**

MAIN:

; initialize the stack pointer

ldi r16, 0xFF

sts CPU\_SPL, r16

ldi r16, 0x3F

sts CPU\_SPH, r16

; initialize relevant I/O modules (switches and LEDs)

rcall IO\_INIT

; initialize USART

rcall USART\_INIT

; We are only outputting 'U' so store in r17

ldi r17, 'U'

LOOP:

rcall OUT\_CHAR

rjmp LOOP

**USART\_INIT code:**

/\*Initialize USARTD0 to utilize an async communication protocol with the following characteristics:

Baud rate: 115,200 bps

Parity: odd

No. data bits: 8

No. stop bits: 1

\*/

USART\_INIT:

; Baud rate symbols

.equ BSEL = 1 ; 12-bit value

.equ BSCALE = -4; 4-bit 2's complement value

push r16

; Configure the UART frame.

ldi r16, ( USART\_CMODE\_ASYNCHRONOUS\_GC | USART\_PMODE\_ODD\_gc | USART\_CHSIZE\_8BIT\_gc )

sts USARTD0\_CTRLC, r16

; Initialize the baud rate.

ldi r16, low(BSEL)

sts USARTD0\_BAUDCTRLA, r16

ldi r16, ( (BSCALE<<4) | high(BSEL) )

sts USARTD0\_BAUDCTRLB, r16

; Enable the transmitter

ldi r16, USART\_TXEN\_bm

sts USARTD0\_CTRLB, r16

pop r16

ret

**OUT\_CHAR code:**

;output a single character to the transmit pin of a chosen USART module

OUT\_CHAR: ; passing in r17

push r16

TX\_POLL:

;Wait until the data register is empty.

lds r16, USARTD0\_STATUS

sbrs r16, USART\_DREIF\_bp

rjmp TX\_POLL

; Transmit the character that was passed in via r17

sts USARTD0\_DATA, r17

pop r16

ret

**SECTION 3 (consisted of connecting Tx signal from the USART to an I/O pin)**

**SECTION 4 (consisted of making OUT\_STRING)**

**MAIN code:**

MAIN:

; initialize the stack pointer

ldi r16, 0xFF

sts CPU\_SPL, r16

ldi r16, 0x3F

sts CPU\_SPH, r16

; initialize relevant I/O modules (switches and LEDs)

rcall IO\_INIT

; initialize USART

rcall USART\_INIT

;\*\*\*NOTE: I basically just took this from my LAB 1

; I have a lot of other school work to do so I am trying to just re-use code to save time because I know it works

;point appropriate indices to input/output tables

ldi ZL, BYTE1(IN\_TABLE << 1) ;load the first value in the table into the Z register

ldi ZH, BYTE2(IN\_TABLE << 1)

;To read program memory we must multiply the table address by 2

;0xABCD \* 2 = 0x1579A

;ZL = 9A

;ZH = 57

;we still need the most significant 1

ldi r20, BYTE3(IN\_TABLE << 1) ;so we load it into the RAMPZ register

out CPU\_RAMPZ, r20 ;we will need to extend load to use the RAMPZ register

LOOP:

rcall OUT\_STRING

DONE:

rjmp DONE

**OUT\_STRING code:**

;output a character string stored in program memory

OUT\_STRING:

ldi r16, NULL

NEXT\_LETTER:

elpm r17, Z+ ;Load next value from table

rcall OUT\_CHAR

cpse r16, r17

rjmp NEXT\_LETTER

ret

**SECTION 5 (consisted of editing USART\_INIT, and making IN\_CHAR)**

**MAIN code:**

MAIN:

; initialize the stack pointer

ldi r16, 0xFF

sts CPU\_SPL, r16

ldi r16, 0x3F

sts CPU\_SPH, r16

; initialize relevant I/O modules (switches and LEDs)

rcall IO\_INIT

; initialize USART

rcall USART\_INIT

LOOP:

rcall IN\_CHAR

cpi r17, NULL

breq LOOP

rcall OUT\_CHAR

;To make it look cleaner, after each character, start new line

ldi r17, '\r'

rcall OUT\_CHAR

ldi r17, '\n'

rcall OUT\_CHAR

rjmp LOOP

DONE:

rjmp DONE

**USART\_INIT code:**

/\*Initialize USARTD0 to utilize an async communication protocol with the following characteristics:

Baud rate: 115,200 bps

Parity: odd

No. data bits: 8

No. stop bits: 1

\*/

USART\_INIT:

; Baud rate symbols

.equ BSEL = 1 ; 12-bit value

.equ BSCALE = -4; 4-bit 2's complement value

push r16

; Configure the UART frame.

ldi r16, ( USART\_CMODE\_ASYNCHRONOUS\_GC | USART\_PMODE\_ODD\_gc | USART\_CHSIZE\_8BIT\_gc )

sts USARTD0\_CTRLC, r16

; Initialize the baud rate.

ldi r16, low(BSEL)

sts USARTD0\_BAUDCTRLA, r16

ldi r16, ( (BSCALE<<4) | high(BSEL) )

sts USARTD0\_BAUDCTRLB, r16

; Enable the transmitter and receiver

ldi r16, ( USART\_TXEN\_bm | USART\_RXEN\_bm )

sts USARTD0\_CTRLB, r16

pop r16

ret

**IN\_CHAR code:**

; receive a single character and return the received character to the calling procedure

IN\_CHAR:

RX\_POLL:

; Wait until a character is received.

lds r17, USARTD0\_STATUS

sbrs r17, USART\_RXCIF\_bp ;skip instruction if bit is set

rjmp RX\_POLL

;Read the received character and save it in r17 for the calling program

lds r17, USARTD0\_DATA

;r17 contains what data has been recieved

ret

**SECTION 6 (consisted of making IN\_STRING. Also made OUT\_STRING\_TWO to test)**

**MAIN code:**

MAIN:

; initialize the stack pointer

ldi r16, 0xFF

sts CPU\_SPL, r16

ldi r16, 0x3F

sts CPU\_SPH, r16

; initialize relevant I/O modules (switches and LEDs)

rcall IO\_INIT

; initialize USART

rcall USART\_INIT

;initialize Y

ldi YL, low(0x3700)

ldi YH, high(0x3700)

LOOP:

ldi YL, low(0x3700)

ldi YH, high(0x3700)

rcall IN\_STRING

rcall OUT\_STRING\_TWO

;To make it look cleaner, after each character, start new line

ldi r17, '\r'

rcall OUT\_CHAR

ldi r17, '\n'

rcall OUT\_CHAR

rjmp LOOP

DONE:

rjmp DONE

**IN\_STRING code:**

IN\_STRING:

READ\_NEXT:

rcall IN\_CHAR

;document says 0x0A, but that didn't work for me

;I don't know how to type a line feed, so I used the enter key

;in ascii that is 0x0D

cpi r17, 0x0D ;carriage

breq CARRIAGE

cpi r17, 0x08 ;backspace

breq BACKSPACE

cpi r17, 0x7F ;Delete character

breq BACKSPACE

;if here, not any of the above, store

st Y+, r17

rjmp READ\_NEXT

CARRIAGE:

ldi r17, NULL

st Y+, r17

ret

BACKSPACE:

ldi r17, NULL

st -Y, r17

rjmp READ\_NEXT

;shouldn't reach this, but in case

ret

**OUT\_STRING\_TWO code:**

OUT\_STRING\_TWO:

;reset Y

ldi YL, low(0x3700)

ldi YH, high(0x3700)

ldi r16, NULL

NEXT\_LETTER\_TWO:

ld r17, Y+

rcall OUT\_CHAR

cpse r16, r17

rjmp NEXT\_LETTER\_TWO

ret

**SECTION 7 (consisted of making RECEIVE\_COMPLETE\_ISR)**

**MAIN code:**

MAIN\_LOOP:

;toggle green here

ldi r16, 0b00100000

sts PORTD\_OUTTGL, r16

rjmp MAIN\_LOOP

DONE:

rjmp DONE

**RECEIVE\_COMPLETE\_ISR code:**

RECEIVE\_COMPLETE\_ISR:

; first, always preserve the status register

push r16

lds r16, CPU\_SREG

push r16

lds r17, USARTD0\_DATA

EMPTY\_REG:

;Wait until the data register is empty.

lds r16, USARTD0\_STATUS

sbrs r16, USART\_DREIF\_bp

rjmp EMPTY\_REG

sts USARTD0\_DATA, r17

cpi r17, 0x0D ;carriage

brne SKIP

ldi r17, '\n'

sts USARTD0\_DATA, r17 ; just to make it look nicer

SKIP:

; recover the status register

pop r16

sts CPU\_SREG, r16

pop r16

reti

**APPENDIX**

Below is my full lab5\_7.asm which has every single subroutine made throughout this lab in it. By showing you this, it gives all my code not shown previously such as my I/O initialization that is very similar in all of my assembly files.

lab5\_7.asm:

/\*

\* lab5\_7.asm

\*

\* Author: Koby

\* Description: Uses USART and interrupts to echo to computer characters received

\*/

.include "ATxmega128A1Udef.inc"

//.equ here

.equ NULL = 0

; USART0

.equ USART0\_RX\_bp = (2)

.equ USART0\_RX\_bm = (1<<USART0\_RX\_bp)

.equ USART0\_TX\_bp = (3)

.equ USART0\_TX\_bm = (1<<USART0\_TX\_bp)

.equ TABLE\_SIZE = 100

.ORG 0x0000

rjmp MAIN

//Other interupt vectors here

.ORG USARTD0\_RXC\_vect

rjmp RECEIVE\_COMPLETE\_ISR

.ORG 0x0100

MAIN:

; initialize the stack pointer

ldi r16, 0xFF

sts CPU\_SPL, r16

ldi r16, 0x3F

sts CPU\_SPH, r16

; initialize relevant I/O modules (switches and LEDs)

rcall IO\_INIT

; initialize USART

rcall USART\_INIT

;initialize interrupts

rcall INTR\_INIT

MAIN\_LOOP:

;toggle green here

ldi r16, 0b00100000

sts PORTD\_OUTTGL, r16

rjmp MAIN\_LOOP

DONE:

rjmp DONE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; I/O Initializations

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IO\_INIT:

; protect relevant registers

push r16

; GREEN\_PMW

ldi r16, 0b00100000

sts PORTD\_OUTSET, r16 ; set led to off

sts PORTD\_DIRSET, r16 ;make it an output

; Initialize transmit pin as a high voltage output

ldi r16, USART0\_TX\_bm

sts PORTD\_OUTSET, r16

sts PORTD\_DIRSET, r16

pop r16

; return from subroutine

ret

/\*Initialize USARTD0 to utilize an async communication protocol with the following characteristics:

Baud rate: 115,200 bps

Parity: odd

No. data bits: 8

No. stop bits: 1

\*/

USART\_INIT:

; Baud rate symbols

.equ BSEL = 1 ; 12-bit value

.equ BSCALE = -4; 4-bit 2's complement value

push r16

; Configure the UART frame.

ldi r16, ( USART\_CMODE\_ASYNCHRONOUS\_GC | USART\_PMODE\_ODD\_gc | USART\_CHSIZE\_8BIT\_gc )

sts USARTD0\_CTRLC, r16

; Initialize the baud rate.

ldi r16, low(BSEL)

sts USARTD0\_BAUDCTRLA, r16

ldi r16, ( (BSCALE<<4) | high(BSEL) )

sts USARTD0\_BAUDCTRLB, r16

; Enable the transmitter and receiver

ldi r16, ( USART\_TXEN\_bm | USART\_RXEN\_bm )

sts USARTD0\_CTRLB, r16

pop r16

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Interrupt initializations

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INTR\_INIT:

;protect registers

push r16

;RXC interrupt. Low level

ldi r16, 0b010000

sts USARTD0\_CTRLA, r16

;Turn on low level interrupts

ldi r16, PMIC\_LOLVLEN\_bm

sts PMIC\_CTRL, r16

;enable global interrupt bit

sei

;recover registers

pop r16

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; IN/OUT SUBROUTINES

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;output a single character to the transmit pin of a chosen USART module

OUT\_CHAR: ; passing in r17

push r16

TX\_POLL:

;Wait until the data register is empty.

lds r16, USARTD0\_STATUS

sbrs r16, USART\_DREIF\_bp

rjmp TX\_POLL

; Transmit the character that was passed in via r17

sts USARTD0\_DATA, r17

pop r16

ret

;output a character string stored in program memory

OUT\_STRING:

ldi r16, NULL

NEXT\_LETTER:

elpm r17, Z+ ;Load next value from table

rcall OUT\_CHAR

cpse r16, r17

rjmp NEXT\_LETTER

ret

OUT\_STRING\_TWO:

;reset Y

ldi YL, low(0x3700)

ldi YH, high(0x3700)

ldi r16, NULL

NEXT\_LETTER\_TWO:

ld r17, Y+

rcall OUT\_CHAR

cpse r16, r17

rjmp NEXT\_LETTER\_TWO

ret

; receive a single character and return the received character to the calling procedure

IN\_CHAR:

RX\_POLL:

; Wait until a character is received.

lds r17, USARTD0\_STATUS

sbrs r17, USART\_RXCIF\_bp ;skip instruction if bit is set

rjmp RX\_POLL

;Read the received character and save it in r17 for the calling program

lds r17, USARTD0\_DATA

;r17 contains what data has been recieved

ret

IN\_STRING:

READ\_NEXT:

rcall IN\_CHAR

;document says 0x0A, but that didn't work for me

;I don't know how to type a line feed, so I used the enter key

;in ascii that is 0x0D

cpi r17, 0x0D ;carriage

breq CARRIAGE

cpi r17, 0x08 ;backspace

breq BACKSPACE

cpi r17, 0x7F ;Delete character

breq BACKSPACE

;if here, not any of the above, store

st Y+, r17

rjmp READ\_NEXT

CARRIAGE:

ldi r17, NULL

st Y+, r17

ret

BACKSPACE:

ldi r17, NULL

st -Y, r17

rjmp READ\_NEXT

;shouldn't reach this, but in case

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Interrupt SERVICE ROUTINES

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RECEIVE\_COMPLETE\_ISR:

; first, always preserve the status register

push r16

lds r16, CPU\_SREG

push r16

lds r17, USARTD0\_DATA

EMPTY\_REG:

;Wait until the data register is empty.

lds r16, USARTD0\_STATUS

sbrs r16, USART\_DREIF\_bp

rjmp EMPTY\_REG

sts USARTD0\_DATA, r17

cpi r17, 0x0D ;carriage

brne SKIP

ldi r17, '\n'

sts USARTD0\_DATA, r17 ; just to make it look nicer

SKIP:

; recover the status register

pop r16

sts CPU\_SREG, r16

pop r16

reti