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

i. The sampling rate of a UART receiver is usually faster than the baud rate of the overall system. Why is this so?

ii. 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.

$$f_{BAUD} = \frac{f_{PER}}{2^{BSCALE} * 16(BSEL + 1)}$$

f_{PER} = 2,000,000

BSCALE = -7

BSEL = 0

f_{BAUD} =16,000,000 Hz

sts USARTD0_BAUDCTRLA, (low(BSEL))

sts USARTD0_BAUDCTRLB, ((BSCALE<<4) | high(BSEL))

iii. 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.

iv. 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

$$\frac{10bits}{1frame} * \frac{1sec}{150{,}000Hz} * 67frames = 4.46667 * 10^{-3}sec$$

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
```