**a) Title.**

**Lab 4**

**Mitchell Irvin**

**Section: 1E83**

**b) Answers to all pre-lab questions.**

**1. What is the difference between synchronous and asynchronous communication?**

Synchronous communication relies on the rising/falling edge of the transmission clock to determine when data is read/transmitted. If data is transmitted on the rising edge then it is received on the falling edge, and vice versa. Asynchronous communication doesn’t rely on the normal clock edge to determine when data is read/transmitted. Asynchronous communication relies on a clock recovery unit to sync the clock with the incoming serial frames.

**2. What is the difference between serial and parallel communication?**

Parallel communication is reading X number of pins at one time to produce some value, while serial communication is reading one pin X (in the case of the Xmega, 10) times, checking the value of the pin at a certain frequency and interpreting each new read as a different bit.

**3. List the XMEGA’s USART registers used in your programs and briefly describe their functions.**

DATA- where a byte will be transferred from or received to

STATUS- holds various flags determining whether data has been received, transmitted, and whether or not the data register is empty. Includes other flags, see manual for full details.

CTRLA- allows for interrupts in serial communication. Allows for specification of interrupt levels for Receive Complete, Transmit Complete, and Data Register Empty.

CTRLB- allows you to enable the receiver/transmitter for this USART port. Also allows enabling of doubling transmission clock speed and multiprocessor communication mode.

CTRLC- allows you to decide which comm mode (sync/async/other), which parity mode, how many stop bits, and the character size of the communications

BAUDCTRLA- lower 8 bits of BSel value determined by baud rate calculator

BAUDCTRLB- upper 4 bits = Bscale3:0, lower 4 bits Bsel11:8. these bits select the baud rate generator scale factor

**4. List the number of bounces from part A of this lab. How long (in ms) is your delay routine for debouncing?**

There were roughly 3-4 bounces per change of switch position. I used a 100ms delay to ensure that bouncing would cause no problems, but judging by the wave form in the appendix I’m sure 30ms would’ve worked well also.

**5. What is the maximum possible baud you can use for asynchronous communication if your board runs at 2 MHz? Support your answer with the values you would place in any special registers that are needed.**

The maximum Baud value is fper/16 for normal clock speed, for double it’s fper/8.

**c) Problems Encountered.**

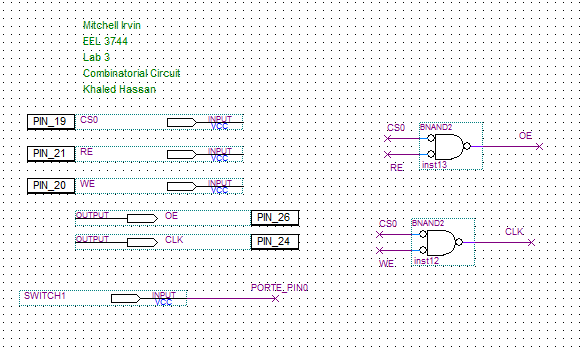
The only big problem encountered was figuring out how to use PORTE for asynchronous communication. At the time of writing this lab document I’m still not sure how to do it properly. I’ve programmed it precisely how I used PORTD but it’s not successfully transmitting anything.

**d) Future Work/Applications.** Given more time (or ambition), how could you adjust your work on this lab for another purpose or with different hardware? Think about how what you did connects with specific applications or a different way of doing the lab. This should be a short description.

This communication could be how multiple more sophisticated devices communicate. This communication could be expanded to be used for any manner of devices and more complex microprocessors.

**e) Schematics.** You must submit a comprehensive and complete schematic every time you add/modify hardware on your board. (You will add to the schematic throughout the semester.)

added switch1 as input to PE0



**f) Decoding Logic.** When we start using the CPLD, you should include screenshots of either your Quartus schematic entry or the text of your VHDL files. You must use Quartus to put your name and lab number at the top of the design file.

N/A for this lab

**g) Pseudocode/Flowcharts.** You must submit either a flowchart or pseudocode for EVERY (even very simple) part of the lab.

**Part A pseudocode:**

do EBI initializations

place jmp to ISR at correct interrupt vector in memory

infinite loop writing counter to LEDs to interrupt

initialize portE’s INTMASK, INTCTRL, and PIN0CTRL registers appropriately

initialize PMIC\_CTRL register

enable global interrupt

ISR:

preserve CPU\_SREG

delay .1s

confirm falling edge

increment counter if true

clear interrupt flags

restore CPU\_SREG

**Part C pseudocode**

set equate statements for Bsel and Bscale

init stack pointer

init PortD and PortQ appropriately w/ subroutine

init USARTD0 registers appropriately w/ subroutine

modify OUT\_CHAR and IN\_CHAR subroutines from example on site to check the correct flags in the status register

create OUT\_STRING subroutine to iterate through characters at a place in memory and use OUT\_CHAR to transmit them

create LOOP to read/transmit characters to confirm functionality

**Part D pseudocode**

Use the code from part C but instead of using PortD’s USART, use PORTE’s USARTE0.

**Part E Pseudocode**

extend part C’s code again.

Place necessary strings in memory for the “favorites” game.

init GPIO (same as part C)

init USART (same as part C)

point Z to favorite’s prompt in memory

OUT\_STRING

LOOP:

read IN\_CHAR

based on IN\_CHAR, call OUT\_STRING after appropriately pointing Z to correct string in memory

if IN\_CHAR was not one of the appropriate values, loop back to rcall IN\_CHAR

**Part F Pseudocode**

at USARTD0\_RXC\_vect in memory jump to ISR

init USART (same as part E)

init GPIO (same as part E)

init EBI (same as part A)

create same OUT\_CHAR, IN\_CHAR, and OUT\_STRING subroutines as part E

create toggleLED subroutine to

output R16 to LED

com R16 (invert R16)

loop

ISR:

preserve current register values

preserve CPU\_SREG

IN\_CHAR

OUT\_CHAR

clear interrupt flag

restore CPU\_SREG

restore register values

reti

**h) Program Code.** Submit your main ASM, C, and/or header file(s) for every part of the lab. Include all supporting asm/libraries in the appendix.

**Lab4\_ext\_intr.asm**

; Lab4\_ext\_intr.asm

; Lab 4 Part A

; Name: Mitchell Irvin

; Section: 1E83

; TA Name: Khaled Hassan

; Description: this program will use an external interrupt and create

; an ISR to display a count of how many times the interrupt has executed

; and output that to the LED array

;Definitions for all the registers in the processor. ALWAYS REQUIRED.

.include "ATxmega128A1Udef.inc"

.set IO\_PORT = 0x4000 ; Set address start to beginning of external memory

.set IO\_PORT\_END = 0x4FFF ; might as well be a comment, going to use 4k CS memory

.org 0x0000

rjmp MAIN

.org PORTE\_INT0\_vect ;using reg def from included file, this is the address of

;our interrupt vector (interrupt0 for PORTE)

jmp EXT\_INT\_ISR ;when our interrupt0 occurs, rjmp to our ISR

.org 0x200

MAIN:

;begin standard 3port/ALE1/SRAM initializations

ldi R16, 0x17 ;Configure the PORTH bits 0,1,2,4 as outputs.

sts PORTH\_DIRSET, R16 ; These are the CS0(L), ALE1(H), WE(L), and RE(L) outputs.

; (CS0 is bit 4; ALE1 is bit 2; RE is bit 1; WE is bit 0)

ldi R16, 0x13 ; Since RE(L), WE(L) and CS0(L) are active low, we must set

sts PORTH\_OUTSET, R16 ; the default output to 1 = H = false.

; -> bits 4,1,0 high, bit 2 low, all others low = 0001 0011

ldi R16, 0xFF ; Set all PORTK pins (A15-A0) to be outputs. As requried

sts PORTK\_DIRSET, R16 ; in the data sheet. See 8331, sec 27.9.

; only 8 bits b/c A7:0 mux'd with A15:8

ldi R16, 0xFF ;Set all PORTJ pins (D7-D0) to be outputs. As requried

sts PORTJ\_DIRSET, R16 ; in the data sheet. See 8331, sec 27.9.

ldi R16, 0x01 ;Store in EBI\_CTRL register to select 3 port EBI(H,J,K)

sts EBI\_CTRL, R16 ; mode and SRAM ALE1 mode. SRAM ALE1 mode is bits 3 and 2 = 00

; 3PORT EBI mode is bits 1 and 0 = 01 therefore lower 4 bits = 0x1

;Reserve a chip-select zone for our input port. The base address register is made up of

; 12 bits for the address (A23:A12). The lower 12 bits of the address (A11-A0) are

; assumed to be zero. This limits our choice of the base addresses.

;Initialize the Z pointer to point to the base address for chip select 0 (CS0) in memory

ldi ZH, high(EBI\_CS0\_BASEADDR)

ldi ZL, low(EBI\_CS0\_BASEADDR)

;Load the middle byte (A15:8) of the three byte address into a register and store it as the

; LOW Byte of the Base Address, BASEADDRL. This will store only bits A15:A12 and ignore

; anything in A11:8 as again, they are assumed to be zero. We increment the Z pointer

; so that we can load the upper byte of the base address register.

ldi R16, byte2(IO\_PORT)

st Z+, R16 ;

;Load the highest byte (A23:16) of the three byte address into a register and store it as the

; HIGH byte of the Base Address, BASEADDRH.

ldi R16, byte3(IO\_PORT)

st Z, R16

ldi R16, 0x11 ; Set to 4K chip select space and turn on SRAM mode, 0x28 8000 - 0x28 9FFF

; in manual under EBI CS ctrl register A: bit 7 = 0, bits 6:2 are ASIZE, 4k = 00100

; bits 1:0 are mode, SRAM = 01 therefore, load R16 with 00010001

sts EBI\_CS0\_CTRLA, R16

rcall INIT\_INTERRUPT

ldi XH, high(IO\_PORT) ; set the middle (XH) and low (XL) bytes of the pointer as usual

ldi XL, low(IO\_PORT)

;init the stack pointer to 0x3FFF (top of internal SRAM)

ldi YL, 0xFF

out CPU\_SPL, YL

ldi YL, 0x3F

out CPU\_SPH, YL

ldi R25, 0x00 ;interrupt counter, starts at 0

LOOP: ;loop forever while the interrupt fires

st X, R25

rjmp LOOP

INIT\_INTERRUPT:

; now begin the interrupt initializations

ldi R16, 0x01

; load R16 with 0b0000 0001 b/c we're enabling interrupt 0 (bits 0 and 1) with a low priority (01)

; via the PORTE\_INTCTRL control register, see doc8331 13.13.10

sts PORTE\_OUT, R16 ;set output to default to 1

sts PORTE\_DIRCLR, R16 ;set pin0 as input

sts PORTE\_INTCTRL, R16 ; enable PORTE interrupt 0

sts PORTE\_INT0MASK, R16

;store 0x01 to interrupt 0 mask register, marking pin0 as interrupt source

ldi R16, 0x02 ;load R16 with 0x02 because pins2:0 are input sense config (010 = falling edge)

;pins5:3 000 for totem pole, pins 7 and 6 0 because INVEN and RREN are set to 0

sts PORTE\_PIN0CTRL, R16 ;set pin0 of portE to sense on falling edge

ldi R16, 0x01 ;load R16 with 0b 00 (RREN and INVEN disabled) 00 0 (reserved)

; 0 (HILVLEN disabled) 0 (MDLVLEN disabled) 1 (LOLVLEN enabled)

sts PMIC\_CTRL, R16

; set programmable multilevel interrupt controller to enable low level interrupts

sei ;enable global interrupt

ret

; interrupt service routine for external interrupt

EXT\_INT\_ISR:

lds R16, CPU\_SREG ;need to push because it may change below

push R16

;shortest delay possible inside the ISR

ldi R21, 0x0A ;set x to be 10, delay 10 \* 10ms

rcall DELAYx10ms ;delay by .1s

;make sure we're on a falling edge

lds R16, PortE\_IN ;read PORTE

andi R16, 0x01 ;AND with 0000 0001 to check the 0th bit

sbrc R16, 0 ;if the 0th bit was high, then we're on a rising edge, don't increment

rjmp NOINC

inc R25 ;otherwise it's a falling edge, increment R25

NOINC:

ldi R17, 0x01

sts PORTE\_INTFLAGS, R17 ; Clear the PORTE\_INTFLAG, not necessary but "can't hurt"

pop R16 ;pop and restore CPU\_SREG to prev value

sts CPU\_SREG, R16

reti ;special return from interrupt directive

;program to delay by x times 10ms, where x is the val in R21

DELAYx10ms:

;load 0x14 into R20, this value will allow for the standard 10ms delay

ldi R20, 0x14

DELAY10ms:

;load 0xFF into R19. when R19 is decremented to 0, reset to 0xFF

;and decremented again over and over (0x14 times) the delay is 10ms

ldi R19, 0xFF

SUBDELAY:

dec R19

cpi R19, 0x00

brne SUBDELAY

dec R20

cpi R20, 0x00

brne DELAY10ms

;do the 10ms delay R21 number of times

dec R21

cpi R21, 0x00

brne DELAYx10ms

ret

**Lab4\_serial.asm**

; Lab4\_serial.asm

; Lab 4 Part C

; Name: Mitchell Irvin

; Section: 1E83

; TA Name: Khaled Hassan

; Description: this program will send and receive data between

; the uPad and the user's machine using USARTD0

;Definitions for all the registers in the processor. ALWAYS REQUIRED.

.include "ATxmega128A1Udef.inc"

.equ BSel = 9 ; these values for 38400Hz yield error = 0.16

.equ BScale = -2 ; 38400 Hz

.equ Prompt = '?'

.equ StringLength = 10

.org 0x0000

rjmp MAIN

.dseg

.org 0x2000 ;where test data will be placed

STRING: .byte StringLength

.cseg

.org 0x0200

MAIN:

ldi YL, 0xFF ;initialize low byte of stack pointer

out CPU\_SPL, YL

ldi YL, 0x3F

out CPU\_SPH, YL

REPEAT:

ldi R17, StringLength

ldi XL, low(STRING)

ldi XH, high(STRING)

; rcall DELAY\_500ms ; This can help with extra characters being displayed

call INIT\_GPIO

call USART\_INIT

ldi R16, Prompt ;load our Prompt character

;OUT\_CHAR expects the arguments on the stack

LOOP:

call OUT\_CHAR ;echo character

call OUT\_CHAR

call OUT\_CHAR

; save value and increment pointer

call IN\_CHAR ;read in character

st X+, R16 ;store the last value

dec R17

brne LOOP

ldi R16, 'W' ;space

rcall OUT\_CHAR ;echo character

ldi R16, 'X'

rcall OUT\_CHAR ;echo character

ldi R16, 'Y' ;space

rcall OUT\_CHAR ;echo character

ldi R16, 'Z'

rcall OUT\_CHAR ;echo character

ldi R16, 0x20 ;space

rcall OUT\_CHAR ;echo character

rjmp REPEAT ;repeat

USART\_INIT:

ldi R16, 0x18

sts USARTD0\_CTRLB, R16 ;turn on TXEN, RXEN lines, bits 4 and 3

ldi R16, 0x03

sts USARTD0\_CTRLC, R16 ;Set Parity to none, 8 bit frame, 1 stop bit

ldi R16, (BSel & 0xFF) ;select only the lower 8 bits of BSel

sts USARTD0\_BAUDCTRLA, R16 ;set baudctrla to lower 8 bites of BSel

ldi R16, ((BScale << 4) & 0xF0) | ((BSel >> 8) & 0x0F)

sts USARTD0\_BAUDCTRLB, R16 ;set baudctrlb to BScale | BSel. Lower

; 4 bits are upper 4 bits of BSel

; and upper 4 bits are the BScale.

ret

INIT\_GPIO:

; portD bits 3:2 are TXD, RXD respectively

; TXD is transmit data buffer, RXD is receive data buffer

ldi R16, 0x04 ; pin 2 is receive = input

sts PortD\_DIRCLR, R16 ; set pin 2 as input

ldi R16, 0x08 ;pin 3 is transmit = output

sts PortD\_DIRSET, R16 ; set pin 3 as output

sts PortD\_OUTSET, R16 ; set activation level as high (pin3)

; PORTQ pins 3 and 1 are USB switch SEL and USB switch ENable respectively

; enable is low true, SEL is high true. we want to set both to 0, to enable

; the output, and to select FTDI\_D- and FTDI\_D+ as the outputs of the USB switch

ldi R16, 0x0A ; pins3 and 1 high

sts PORTQ\_DIRSET, R16 ; Set pins 3 and 1 as output

sts PORTQ\_OUTCLR, R16 ; pins 3 and 1 low

; set PQ3 and PQ1 as low (enable true, sel off (FTDI))

ret

OUT\_CHAR:

push R17

TX\_POLL:

lds R17, USARTD0\_STATUS ;load status register

sbrs R17, 5 ;proceed to writing out the char if

; the DREIF flag is set (bit 5)

rjmp TX\_POLL ;else go back to polling

sts USARTD0\_DATA, R16 ;send the character out over the USART

pop R17

ret

OUT\_STRING:

lpm R16, Z+ ;read character pointed to by Z and inc Z

cpi R16, 0x00 ;check if char is null

breq RETURN ;if char is null return from subroutine

rcall OUT\_CHAR ;char is not null, call OUT\_CHAR

rjmp OUT\_STRING ;repeat for each non-null char

RETURN:

ret

IN\_CHAR:

RX\_POLL:

lds R16, USARTD0\_STATUS ;load the status register

sbrs R16, 7 ;proceed to reading in a char if the receive flag is set

; if bit 7 is 1, there is unread data in the receive buffer

rjmp RX\_POLL ;else continue polling

lds R16, USARTD0\_DATA ;read the character into R16

ret

**Lab4\_serial\_baud\_test.asm**

; Lab4\_serial\_baud\_test.asm

; Lab 4 Part D

; Name: Mitchell Irvin

; Section: 1E83

; TA Name: Khaled Hassan

; Description: this program will send and receive data between

; the uPad and the user's machine using USARTE0, and test to confirm

; that the correct baud rate (38400) is being used

;Definitions for all the registers in the processor. ALWAYS REQUIRED.

.include "ATxmega128A1Udef.inc"

.equ BSel = 9 ; these values for 38400Hz yield error = 0.16

.equ BScale = -2 ; 38400 Hz

.org 0x0000

rjmp MAIN

.cseg

.org 0x0200

MAIN:

ldi YL, 0xFF ;initialize low byte of stack pointer

out CPU\_SPL, YL

ldi YL, 0x3F

out CPU\_SPH, YL

rjmp INIT

INIT:

call INIT\_GPIO

call USART\_INIT

rjmp LOOP

LOOP:

ldi R16, 'U' ;load R16 for OUT\_CHAR b/c it expects arguments on the stack

call OUT\_CHAR

ldi R21, 0x02 ;set delay for 2 \* 500us

call DELAYx500us ;delay 2 \* 500us

rjmp LOOP ;repeat

USART\_INIT:

ldi R16, 0x18

sts USARTE0\_CTRLB, R16 ;turn on TXEN, RXEN lines, bits 4 and 3

ldi R16, 0x03

sts USARTE0\_CTRLC, R16 ;Set Parity to none, 8 bit frame, 1 stop bit

ldi R16, (BSel & 0xFF) ;select only the lower 8 bits of BSel

sts USARTE0\_BAUDCTRLA, R16 ;set baudctrla to lower 8 bites of BSel

ldi R16, ((BScale << 4) & 0xF0) | ((BSel >> 8) & 0x0F)

sts USARTE0\_BAUDCTRLB, R16 ;set baudctrlb to BScale | BSel. Lower

; 4 bits are upper 4 bits of BSel

; and upper 4 bits are the BScale.

ret

INIT\_GPIO:

; portE bits 3:2 are TXD, RXD respectively

; TXD is transmit data buffer, RXD is receive data buffer

ldi R16, 0x04 ; pin 2 is receive = input

sts PORTE\_DIRCLR, R16 ; set pin 2 as input

ldi R16, 0x08 ;pin 3 is transmit = output

sts PORTE\_DIRSET, R16 ; set pin 3 as output

sts PORTE\_OUTSET, R16 ; set activation level as high (pin3)

; PORTQ pins 3 and 1 are USB switch SEL and USB switch ENable respectively

; enable is low true, SEL is high true. we want to set both to 0, to enable

; the output, and to select FTDI\_D- and FTDI\_D+ as the outputs of the USB switch

ldi R16, 0x0A ; pins3 and 1 high

sts PORTQ\_DIRSET, R16 ; Set pins 3 and 1 as output

sts PORTQ\_OUTCLR, R16 ; pins 3 and 1 low

; set PQ3 and PQ1 as low (enable true, sel off (FTDI))

ret

OUT\_CHAR:

push R17

TX\_POLL:

lds R17, USARTE0\_STATUS ;load status register

sbrs R17, 5 ;proceed to writing out the char if

; the DREIF flag is set (bit 5)

rjmp TX\_POLL ;else go back to polling

sts USARTE0\_DATA, R16 ;send the character out over the USART

pop R17

ret

OUT\_STRING:

lpm R16, Z+ ;read character pointed to by Z and inc Z

cpi R16, 0x00 ;check if char is null

breq RETURN ;if char is null return from subroutine

rcall OUT\_CHAR ;char is not null, call OUT\_CHAR

rjmp OUT\_STRING ;repeat for each non-null char

RETURN:

ret

IN\_CHAR:

RX\_POLL:

lds R16, USARTE0\_STATUS ;load the status register

sbrs R16, 7 ;proceed to reading in a char if the receive flag is set

; if bit 7 is 1, there is unread data in the receive buffer

rjmp RX\_POLL ;else continue polling

lds R16, USARTE0\_DATA ;read the character into R16

ret

;subroutine to delay by 500us \* R20

DELAYx500us:

;load 0xFF into R19. when R19 is decremented to 0, reset to 0xFF

;and decremented again over and over (0x14 times) the delay is 10ms

ldi R19, 0xFF

SUBDELAY:

dec R19

cpi R19, 0x00

brne SUBDELAY

dec R20

cpi R20, 0x00

brne DELAYx500us

ret

**Lab4\_serial\_menu.asm**

; Lab4\_serial\_menu.asm

; Lab 4 Part E

; Name: Mitchell Irvin

; Section: 1E83

; TA Name: Khaled Hassan

; Description: this program will display a list of categories to the user,

; with each category numbered. When the user sends one of the corresponding

; numbers, the program will respond with "Mitch's favorite \_\_\_\_ is \_\_\_\_\_".

;Definitions for all the registers in the processor. ALWAYS REQUIRED.

.include "ATxmega128A1Udef.inc"

.equ BSel = 9 ; these values for 38400Hz yield error = 0.16

.equ BScale = -2 ; 38400 Hz

.equ CR = 0x0D

.equ LF = 0x0A

.equ ESC = 0x1B

.org 0x0000

jmp MAIN

; need to use EEPROM memory (1000-17FF) for storing our strings in data memory

; store all the strings we need for our "Favorites" program to work

.org 0x1000

MENU: .db "Mitch's favorite:", CR, LF, \

"1. OS/Computer", CR, LF, \

"2. EE/CE Course", CR, LF, \

"3. Hobby", CR, LF, \

"4. Quote", CR, LF, \

"5. Movie", CR, LF, \

"6. Re-display Menu", CR, LF, \

"ESC: exit", CR, LF, CR, LF, '\0'

.org 0x1070

COMP: .db "Mitch's favorite OS/Computer is Windows/PC", CR, LF, CR, LF, '\0'

.org 0x1100

COURSE: .db "Mitch's favorite EE/CE Course is Digital Logic", CR, LF, CR, LF, '\0'

.org 0x1150

HOBBY: .db "Mitch's favorite Hobby is weightlifting", CR, LF, CR, LF, '\0'

.org 0x1200

QUOTE: .db "Mitch's favorite Quote is: We made Iran great again. -Trump", CR, LF, CR, LF, '\0'

.org 0x1250

MOVIE: .db "Mitch's favorite Movie is Edge of Tomorrow", CR, LF, CR, LF, '\0'

.org 0x1300

TERMINATE: .db "Done!", '\0'

.org 0x0200

MAIN:

ldi YL, 0xFF ;initialize low byte of stack pointer

out CPU\_SPL, YL

ldi YL, 0x3F

out CPU\_SPH, YL

call INIT\_GPIO ;init GPIO ports and USART

call USART\_INIT

LOOP:

ldi ZL, low(MENU << 1) ;point Z at MENU label in memory

ldi ZH, high(MENU << 1)

call OUT\_STRING ;output the string where Z is pointing

INVALIDCHAR:

;wait for character to be input

call IN\_CHAR

cpi R16, '1' ;if user input 1, branch to ONE

breq ONE

cpi R16, '2' ;if user input 2, branch to TWO

breq TWO

cpi R16, '3' ;if user input 2, branch to THREE

breq THREE

cpi R16, '4' ;if user input 2, branch to FOUR

breq FOUR

cpi R16, '5' ;if user input 2, branch to FIVE

breq FIVE

cpi R16, '6' ;if user input 6, LOOP

breq LOOP

cpi R16, ESC ;if user input ESC, quit

breq DONE

rjmp INVALIDCHAR ;user didn't input a valid character, jump to

;INVALID CHAR to receive input again

ONE:

ldi ZL, low(COMP << 1) ;point Z to response string 1

ldi ZH, high(COMP << 1)

call OUT\_STRING ;print string at location Z is pointing to

jmp LOOP ;display menu again and wait for input

TWO:

ldi ZL, low(COURSE << 1) ;point Z to response string 2

ldi ZH, high(COURSE << 1)

call OUT\_STRING ;print string at location Z is pointing to

jmp LOOP ;display menu again and wait for input

THREE:

ldi ZL, low(HOBBY << 1) ;point Z to response string 3

ldi ZH, high(HOBBY << 1)

call OUT\_STRING ;print string at location Z is pointing to

jmp LOOP ;display menu again and wait for input

FOUR:

ldi ZL, low(QUOTE << 1) ;point Z to response string 4

ldi ZH, high(QUOTE << 1)

call OUT\_STRING ;print string at location Z is pointing to

jmp LOOP ;display menu again and wait for input

FIVE:

ldi ZL, low(MOVIE << 1) ;point Z to response string 5

ldi ZH, high(MOVIE << 1)

call OUT\_STRING ;print string at location Z is pointing to

jmp LOOP ;display menu again and wait for input

;output "Done!" and terminate execution

DONE:

ldi ZL, low(TERMINATE << 1) ;point Z to response string 5

ldi ZH, high(TERMINATE << 1)

call OUT\_STRING ;print string at location Z is pointing to

rjmp END

END: ;terminate program w/ infinite loop

rjmp END

USART\_INIT:

ldi R16, 0x18

sts USARTD0\_CTRLB, R16 ;turn on TXEN, RXEN lines, bits 4 and 3

ldi R16, 0x03

sts USARTD0\_CTRLC, R16 ;Set Parity to none, 8 bit frame, 1 stop bit

ldi R16, (BSel & 0xFF) ;select only the lower 8 bits of BSel

sts USARTD0\_BAUDCTRLA, R16 ;set baudctrla to lower 8 bites of BSel

ldi R16, ((BScale << 4) & 0xF0) | ((BSel >> 8) & 0x0F)

sts USARTD0\_BAUDCTRLB, R16 ;set baudctrlb to BScale | BSel. Lower

; 4 bits are upper 4 bits of BSel

; and upper 4 bits are the BScale.

ret

INIT\_GPIO:

; portD bits 3:2 are TXD, RXD respectively

; TXD is transmit data buffer, RXD is receive data buffer

ldi R16, 0x04 ; pin 2 is receive = input

sts PORTD\_DIRCLR, R16 ; set pin 2 as input

ldi R16, 0x08 ;pin 3 is transmit = output

sts PORTD\_DIRSET, R16 ; set pin 3 as output

sts PoRTD\_OUTSET, R16 ; set activation level as high (pin3)

; PORTQ pins 3 and 1 are USB switch SEL and USB switch ENable respectively

; enable is low true, SEL is high true. we want to set both to 0, to enable

; the output, and to select FTDI\_D- and FTDI\_D+ as the outputs of the USB switch

ldi R16, 0x0A ; pins3 and 1 high

sts PORTQ\_DIRSET, R16 ; Set pins 3 and 1 as output

sts PORTQ\_OUTCLR, R16 ; pins 3 and 1 low

; set PQ3 and PQ1 as low (enable true, sel off (FTDI))

ret

OUT\_CHAR:

push R17

TX\_POLL:

lds R17, USARTD0\_STATUS ;load status register

sbrs R17, 5 ;proceed to writing out the char if

; the DREIF flag is set (bit 5)

rjmp TX\_POLL ;else go back to polling

sts USARTD0\_DATA, R16 ;send the character out over the USART

pop R17

ret

OUT\_STRING:

lpm R16, Z+ ;read character pointed to by Z and inc Z

cpi R16, 0x00 ;check if char is null

breq RETURN ;if char is null return from subroutine

rcall OUT\_CHAR ;char is not null, call OUT\_CHAR

rjmp OUT\_STRING ;repeat for each non-null char

RETURN:

ret

IN\_CHAR:

RX\_POLL:

lds R16, USARTD0\_STATUS ;load the status register

sbrs R16, 7 ;proceed to reading in a char if the receive flag is set

; if bit 7 is 1, there is unread data in the receive buffer

rjmp RX\_POLL ;else continue polling

lds R16, USARTD0\_DATA ;read the character into R16

ret

**Lab4\_serial\_int.asm**

; Lab4\_serial\_int.asm

; Lab 4 Part F

; Name: Mitchell Irvin

; Section: 1E83

; TA Name: Khaled Hassan

; Description: this program is an interrupt driven echo system

;Definitions for all the registers in the processor. ALWAYS REQUIRED.

.include "ATxmega128A1Udef.inc"

.equ BSel = 9 ; these values for 38400Hz yield error = 0.16

.equ BScale = -2 ; 38400 Hz

.equ Prompt = '?'

.equ StringLength = 10

.set IO\_PORT = 0x4000 ; Set address start to beginning of external memory

.set IO\_PORT\_END = 0x4FFF ; might as well be a comment, going to use 4k CS memory

.org USARTD0\_RXC\_vect ;using reg def from included file, this is the address of

;our interrupt vector (receive confirmed vector

;from portD's USART functionality)

jmp EXT\_INT\_ISR ;when our interrupt0 occurs, rjmp to our ISR

.org 0x0000

rjmp MAIN

.cseg

.org 0x0200

MAIN:

ldi YL, 0xFF ;initialize low byte of stack pointer

out CPU\_SPL, YL

ldi YL, 0x3F

out CPU\_SPH, YL

ldi R16, 0x01 ;load R16 with 0b 00 (RREN and INVEN disabled) 00 0 (reserved)

; 0 (HILVLEN disabled) 0 (MDLVLEN disabled) 1 (LOLVLEN enabled)

sts PMIC\_CTRL, R16

; set programmable multilevel interrupt controller to enable low level interrupts

sei ;enable global interrupt

REPEAT:

; rcall DELAY\_500ms ; This can help with extra characters being displayed

call INIT\_GPIO

call INIT\_EBI

call USART\_INIT

ldi R16, 0x00

TOGGLELED:

st X, R16 ;light up LED

com R16 ;toggle value

ldi R21, 0x32

call DELAYx10ms ;delay for .5s

rjmp TOGGLELED ;repeat

USART\_INIT:

ldi R16, 0x10

sts USARTD0\_CTRLA, R16 ;enable low level receiver interrupt w/ CTRLA

ldi R16, 0x18

sts USARTD0\_CTRLB, R16 ;turn on TXEN, RXEN lines, bits 4 and 3

ldi R16, 0x03

sts USARTD0\_CTRLC, R16 ;Set Parity to none, 8 bit frame, 1 stop bit

ldi R16, (BSel & 0xFF) ;select only the lower 8 bits of BSel

sts USARTD0\_BAUDCTRLA, R16 ;set baudctrla to lower 8 bites of BSel

ldi R16, ((BScale << 4) & 0xF0) | ((BSel >> 8) & 0x0F)

sts USARTD0\_BAUDCTRLB, R16 ;set baudctrlb to BScale | BSel. Lower

; 4 bits are upper 4 bits of BSel

; and upper 4 bits are the BScale.

ret

INIT\_GPIO:

; portD bits 3:2 are TXD, RXD respectively

; TXD is transmit data buffer, RXD is receive data buffer

ldi R16, 0x04 ; pin 2 is receive = input

sts PortD\_DIRCLR, R16 ; set pin 2 as input

ldi R16, 0x08 ;pin 3 is transmit = output

sts PortD\_DIRSET, R16 ; set pin 3 as output

sts PortD\_OUTSET, R16 ; set activation level as high (pin3)

; PORTQ pins 3 and 1 are USB switch SEL and USB switch ENable respectively

; enable is low true, SEL is high true. we want to set both to 0, to enable

; the output, and to select FTDI\_D- and FTDI\_D+ as the outputs of the USB switch

ldi R16, 0x0A ; pins3 and 1 high

sts PORTQ\_DIRSET, R16 ; Set pins 3 and 1 as output

sts PORTQ\_OUTCLR, R16 ; pins 3 and 1 low

; set PQ3 and PQ1 as low (enable true, sel off (FTDI))

ret

INIT\_EBI:

;begin standard 3port/ALE1/SRAM initializations

ldi R16, 0x17 ;Configure the PORTH bits 0,1,2,4 as outputs.

sts PORTH\_DIRSET, R16 ; These are the CS0(L), ALE1(H), WE(L), and RE(L) outputs.

; (CS0 is bit 4; ALE1 is bit 2; RE is bit 1; WE is bit 0

ldi R16, 0x13 ; Since RE(L), WE(L) and CS0(L) are active low, we must set

sts PORTH\_OUTSET, R16 ; the default output to 1 = H = false.

; -> bits 4,1,0 high, bit 2 low, all others low = 0001 0011

ldi R16, 0xFF ; Set all PORTK pins (A15-A0) to be outputs. As requried

sts PORTK\_DIRSET, R16 ; in the data sheet. See 8331, sec 27.9.

; only 8 bits b/c A7:0 mux'd with A15:8

ldi R16, 0xFF ;Set all PORTJ pins (D7-D0) to be outputs. As requried

sts PORTJ\_DIRSET, R16 ; in the data sheet. See 8331, sec 27.9.

ldi R16, 0x01 ;Store in EBI\_CTRL register to select 3 port EBI(H,J,K)

sts EBI\_CTRL, R16 ; mode and SRAM ALE1 mode. SRAM ALE1 mode is bits 3 and 2 = 00

; 3PORT EBI mode is bits 1 and 0 = 01 therefore lower 4 bits = 0x1

;Reserve a chip-select zone for our input port. The base address register is made up of

; 12 bits for the address (A23:A12). The lower 12 bits of the address (A11-A0) are

; assumed to be zero. This limits our choice of the base addresses.

;Initialize the Z pointer to point to the base address for chip select 0 (CS0) in memory

ldi ZH, high(EBI\_CS0\_BASEADDR)

ldi ZL, low(EBI\_CS0\_BASEADDR)

;Load the middle byte (A15:8) of the three byte address into a register and store it as the

; LOW Byte of the Base Address, BASEADDRL. This will store only bits A15:A12 and ignore

; anything in A11:8 as again, they are assumed to be zero. We increment the Z pointer

; so that we can load the upper byte of the base address register.

ldi R16, byte2(IO\_PORT)

st Z+, R16 ;

;Load the highest byte (A23:16) of the three byte address into a register and store it as the

; HIGH byte of the Base Address, BASEADDRH.

ldi R16, byte3(IO\_PORT)

st Z, R16

ldi R16, 0x11 ; Set to 4K chip select space and turn on SRAM mode, 0x28 8000 - 0x28 9FFF

; in manual under EBI CS ctrl register A: bit 7 = 0, bits 6:2 are ASIZE, 4k = 00100

; bits 1:0 are mode, SRAM = 01 therefore, load R16 with 00010001

sts EBI\_CS0\_CTRLA, R16

ldi XH, high(IO\_PORT) ; set the middle (XH) and low (XL) bytes of the pointer as usual

ldi XL, low(IO\_PORT)

ret

OUT\_CHAR:

push R17

TX\_POLL:

lds R17, USARTD0\_STATUS ;load status register

sbrs R17, 5 ;proceed to writing out the char if

; the DREIF flag is set (bit 5)

rjmp TX\_POLL ;else go back to polling

sts USARTD0\_DATA, R16 ;send the character out over the USART

pop R17

ret

OUT\_STRING:

lpm R16, Z+ ;read character pointed to by Z and inc Z

cpi R16, 0x00 ;check if char is null

breq RETURN ;if char is null return from subroutine

rcall OUT\_CHAR ;char is not null, call OUT\_CHAR

rjmp OUT\_STRING ;repeat for each non-null char

RETURN:

ret

IN\_CHAR:

RX\_POLL:

lds R16, USARTD0\_STATUS ;load the status register

sbrs R16, 7 ;proceed to reading in a char if the receive flag is set

; if bit 7 is 1, there is unread data in the receive buffer

rjmp RX\_POLL ;else continue polling

lds R16, USARTD0\_DATA ;read the character into R16

ret

EXT\_INT\_ISR:

;push any registers that might be holding sensitive values to the stack

;so we don't accidentally change them

push R16

push R17

push R18

push R19

push R20

push R21

lds R16, CPU\_SREG ;need to push because it may change below

push R16

call IN\_CHAR ;read input character

call OUT\_CHAR ;echo character

ldi R17, 0x80 ;pin7 is the NMI flag, must clear before returning

sts USARTD0\_STATUS, R17 ; Clear the USARTD0's non-maskable interrupt flag,

; not necessary but "can't hurt"

pop R16

sts CPU\_SREG, R16 ;restore CPU\_SREG value

;pop reg values previously pushed back into place

pop R21

pop R20

pop R19

pop R18

pop R17

pop R16

reti

;program to delay by x times 10ms, where x is the val in R21

DELAYx10ms:

;load 0x14 into R20, this value will allow for the standard 10ms delay

ldi R20, 0x14

DELAY10ms:

;load 0xFF into R19. when R19 is decremented to 0, reset to 0xFF

;and decremented again over and over (0x14 times) the delay is 10ms

ldi R19, 0xFF

SUBDELAY:

dec R19

cpi R19, 0x00

brne SUBDELAY

dec R20

cpi R20, 0x00

brne DELAY10ms

;do the 10ms delay R21 number of times

dec R21

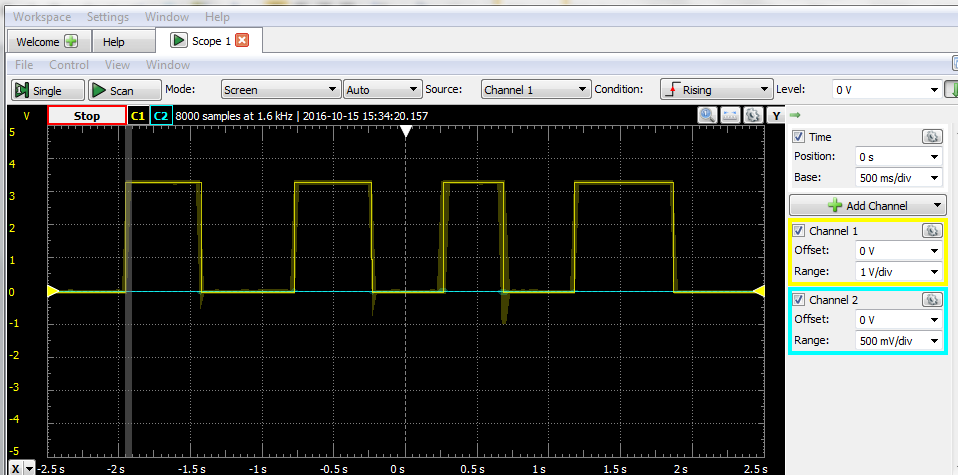
cpi R21, 0x00

brne DELAYx10ms

ret

**i) Appendix.** Include all supporting ASM, C, or header files that you used in your main programs. Also include any other relevant information.

screenshot of waveform for switch input



close up screenshot of waveform for switch input



based on this screenshot I decided to have a delay of .1s before reading the input value in my ISR

**PART D Baud Rate Confirmation**

the width of our bits appears to be 26us on our image below. 1/26us = 38461 bps, which is approximately our correct baud rate.

