HW4
Pengzhao Zhu
Section: 112D

B) Prelab Questions

1. Which pins on PORTD are used for USARTD0?
   Pin 2 and Pin 3 on PORTD are used for USARTD0. Pin 2 is for Rx AND pin 3 is for Tx.

2. What is the maximum possible baud you can use for asynchronous communication, if your board runs at 32 MHz? Support your answer.
   The maximum possible baud rate for asynchronous communication (at 32 Mhz) of the XMEGA USART system is 2.0 Mbps for CLK2X=0 and 4.0 Mbps for CLK2X=1. The stats are listed in Table 23-5 of the USART section of the XEMGA manual.

3. What is the main difference between serial and parallel communication?
   Serial is one bit at a time. In parallel communication, bits are sent simultaneously over their own channels.

4. What is the main difference between synchronous and asynchronous communication?
   Synchronous communication shares the same clock and they communicate with each other with the same clock speed. Asynchronous communication does not have the same clock. Instead, asynchronous communication relies on a predetermined baud rate (bits/second) to facilitate communication.

5. List the XMEGA's USART registers used in your programs, and briefly describe their functions.
   USARTC0_DATA= I used this register to transmit data and read data. TXB and RXB shares this line.
   USARTC0_STATUS= I used this register to check for the data register empty flag (before writing to it) and for the receive complete interrupt flag. Some of the other bits in this register include TXCIF, FERR, and BUFOVF.

   USARTC0_CTRLA= I used this register to set the receive complete interrupt level (RXCINTLVL). Other uses for this register include setting transmit complete level interrupt level and data register empty interrupt level.

   USARTD0_CTRLB= I used this register in my program to enable receiver and transmitter. Another bits in the program include to enable double transmission speed, multiprocessor communication mode, and enable transmit bit B

   USARTD0_CTRLC= I used this register to enable asynchronous USART communication, set odd parity, set number of stop bits, and set number of data bits. This register is used to set up the above mentioned features.

   USARTC0_BAUDCTRLA= This register sets the lower 8 bits of the 12-bit BSEL value. I used it to set the BSEL value

USARTC0_BAUDCTRLB= This upper 4 bits of this register contains the baud rate generator scale factor. The lower 4 bits of this register sets the upper 4 bits of the 12-bit BSEL value.

C) Problems Encountered

The first problem I encountered in this lab was that I didn't know which register to check before writing data to USART0_DATA register. I checked the transmit complete flag but the flag is never set. I finally realize I had to check the DREIF flag before I write data to USART0_DATA register.

The second problem I encountered in this lab was that I had trouble figuring out the correct BSEL and BSCALE to used. There are two variables and two unknowns in the equations provided. Finally, I realized that I had to keep using different BSCALE values to get a good BSEL (with low errors) value to use.

D) Future Application

The USART system (in this case asynchronous) is a useful knowledge to know because it enables data communication between two devices. By mastering this homework, I will be able to program microprocessor to communicate with other electronic devices. It will come in handy for senior design or during my future jobs if it relates to embedded systems.

E) Schematics

N/A

F) Pseudocode/Flowcharts

Part A Pseudocode:

Set up r23 to be 0x00 for the 32Mhz subroutine
rcall CLK (to set up 32Mhz clock)

Initialize Stack Pointer to 0x3FFF

ldi r17, 0x55

rcall USART

LOOP:

    Rcall OUT_CHAR

    Rjmp LOOP

DONE:
    rjmp DONE

USART:

Set receiver as output

Set transmitter as input

Enable receiver and transmitter

Set USART asynchronous, 8 data bit, odd parity, 1 stop bit

Load lower 8 bit BSEL value into BAUDCTRLA

Low BSCALE and highest 4 bit of BSEL value into BAUDCTRLB

Set the transmit pin idle

Ret

OUT_CHAR:

Check the DREIF flag of USARTD0_STATIS

If not wait, wait for it to be set.

If set, transmit r17 to USARTD0_DATA

Ret

CLK (32 MHZ subroutine):

push r16

set OSC_CTRL to be the 32 MHZ oscillator

NSTABLE:

Check if 32MHZ oscillator is stable

If stable, go to STABLE

If not stable, go back to NSTABLE

STABLE:

Write IOREG (0xD8) to CPU_CCP to enable change

Select the 32 MHZ oscillator

Write IOREG (0XD8) to CPU_CCP to use prescaler

Use r23 initialized outside the subroutine to set it up so it remains 32Mhz

pop r16

ret

Part C Pseudocode:

Put a table that contains the string of my name

Set up r23 to be 0x00 for the 32Mhz subroutine
rcall CLK (to set up 32Mhz clock)

Initialize Stack Pointer to 0x3FFF

Low Z pointer with the starting address of the table (with my name).

rcall USART

rcall OUT_STRING

DONE:
    rjmp DONE

USART:

Set receiver as output

Set transmitter as input

Enable receiver and transmitter

Set USART asynchronous, 8 data bit, odd parity, 1 stop bit

Load lower 8 bit BSEL value into BAUDCTRLA

Low BSCALE and highest 4 bit of BSEL value into BAUDCTRLB

Set the transmit pin idle

Ret

OUT_CHAR:

Check the DREIF flag of USARTD0_STATIS

If not wait, wait for it to be set.

If set, transmit r17 to USARTD0_DATA

Ret

OUT_STRING:

Push r17

CONTINE:

Load value pointed to by Z to r16. Post increment Z

Check if the value is the null character.

breq RETURN

rcall OUT_CHAR

rjmp CONTINUE

RETURN:

Pop r17

Ret

CLK (32 MHZ subroutine):

push r16

set OSC_CTRL to be the 32 MHZ oscillator

NSTABLE:

Check if 32MHZ oscillator is stable

If stable, go to STABLE

If not stable, go back to NSTABLE

STABLE:

Write IOREG (0xD8) to CPU_CCP to enable change

Select the 32 MHZ oscillator

Write IOREG (0XD8) to CPU_CCP to use prescaler

Use r23 initialized outside the subroutine to set it up so it remains 32Mhz

pop r16

ret

Part D Pseudocode:

Put a table that contains the string of my name

Set up r23 to be 0x00 for the 32Mhz subroutine
rcall CLK (to set up 32Mhz clock)

Initialize Stack Pointer to 0x3FFF

Low Z pointer with the starting address of the table (with my name).

rcall USART

LOOP:

    Rcall IN_CHAR

    Rcall OUT_CHAR

    Rjmp LOOP

USART:

Set receiver as output

Set transmitter as input

Enable receiver and transmitter

Set USART asynchronous, 8 data bit, odd parity, 1 stop bit

Load lower 8 bit BSEL value into BAUDCTRLA

Low BSCALE and highest 4 bit of BSEL value into BAUDCTRLB

Set the transmit pin idle

Ret

OUT_CHAR:

Check the DREIF flag of USARTD0_STATIS

If not wait, wait for it to be set.

If set, transmit r17 to USARTD0_DATA

Ret

OUT_STRING:

Push r17

CONTINE:

Load value pointed to by Z to r16. Post increment Z

Check if the value is the null character.

breq RETURN

rcall OUT_CHAR

rjmp CONTINUE

RETURN:

Pop r17

Ret

IN_CHAR:

Push r16

NOT:

Check the receive complete flag

If not set, go to NOT.

If set, go to RECEIVE.

RECEIVE:

Transfer data in USARTD0_DATA to r17

Pop r16

Ret

CLK (32 MHZ subroutine):

push r16

set OSC_CTRL to be the 32 MHZ oscillator

NSTABLE:

Check if 32MHZ oscillator is stable

If stable, go to STABLE

If not stable, go back to NSTABLE

STABLE:

Write IOREG (0xD8) to CPU_CCP to enable change

Select the 32 MHZ oscillator

Write IOREG (0XD8) to CPU_CCP to use prescaler

Use r23 initialized outside the subroutine to set it up so it remains 32Mhz

pop r16

ret

Part E Pseudocode:

```
.org USARTF0_RXC_VECT
        Rjmp USART_ISR
```

Put a table that contains the string of my name

Set up r23 to be 0x00 for the 32Mhz subroutine
rcall CLK (to set up 32Mhz clock)

Initialize Stack Pointer to 0x3FFF

Low Z pointer with the starting address of the table (with my name).

rcall USART

SET green LED as output

Set the timer for 5ms by setting low and high byte of PER

Set the Timer Clock for CLK/1024

LOOP:

Check the interrupt flag.

If set jump to TOGGLE.

If not, back to LOOP.

TOGGLE:

Toggle the green LED.

Set the CNT value back to zero.

Clear the interrupt flag

Rjmp LOOP

USART:

Set receiver as output

Set transmitter as input

Enable receiver and transmitter

Set USART asynchronous, 8 data bit, odd parity, 1 stop bit

Load lower 8 bit BSEL value into BAUDCTRLA

Low BSCALE and highest 4 bit of BSEL value into BAUDCTRLB

Set the transmit pin idle

Enable low level interrupt for receive complete

Enable low level interrupt in the PMIC

sei

Ret

USART_ISR:

Pushes the necessary registers (including CPU_SREG)

Read the information in the receiver buffer.

WAIT:

Check the DREIF flag.

If set, go to TRANSMIT.

If not set, go to WAIT.

TRANSMIT:

Transmit received character to USARTD0_DATA

Clear the receive complete interrupt flag.

Pop necessary registers. Including CPU_SREG.

reti

OUT_CHAR:

Check the DREIF flag of USARTD0_STATIS

If not wait, wait for it to be set.

If set, transmit r17 to USARTD0_DATA

Ret

OUT_STRING:

Push r17

CONTINE:

Load value pointed to by Z to r16. Post increment Z

Check if the value is the null character.

breq RETURN

rcall OUT_CHAR

rjmp CONTINUE

RETURN:

Pop r17

Ret

IN_CHAR:

Push r16

NOT:

Check the receive complete flag

If not set, go to NOT.

If set, go to RECEIVE.

RECEIVE:

Transfer data in USARTD0_DATA to r17

Pop r16

Ret

CLK (32 MHZ subroutine):

push r16

set OSC_CTRL to be the 32 MHZ oscillator

NSTABLE:

Check if 32MHZ oscillator is stable

If stable, go to STABLE

If not stable, go back to NSTABLE

STABLE:

Write IOREG (0xD8) to CPU_CCP to enable change

Select the 32 MHZ oscillator

Write IOREG (0XD8) to CPU_CCP to use prescaler

Use r23 initialized outside the subroutine to set it up so it remains 32Mhz

pop r16

ret

G) Program Code

Part A

```
/* HW 4 Part A
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program outputs the ASCII chacter "U" to Putty continously

*/


.include "ATxmega128A1Udef.inc"        ;include the file
.list                                  ;list it

.org 0x0000                            ;start our program here
rjmp MAIN                              ;jump to main

.equ stack_init=0x3FFF   ;initialize stack pointer
.equ BSELHIGH=(((8)*((32000000/(16*115200))-1))>>8)
.equ BSEL=((8)*((32000000/(16*115200))-1))

.org 0x100

MAIN:
ldi r23, 0x00   ;setting for 32MHZ subroutine

rcall CLK

ldi YL, low(stack_init)    ;Load 0xFF to YL
out CPU_SPL, YL                        ;transfer to CPU_SPL
ldi YL, high(stack_init)   ;Load 0x3F to YH
out CPU_SPH, YL                        ;transfer to CPU_SPH

ldi r17, 0x55       ;ASCII hex code for “U". Used in the OUT_CHAR subroutine

rcall USART        ;call subroutine to set up USART system

LOOP:
rcall OUT_CHAR      ;call OUT_CHAR subroutine
rjmp LOOP                ;infinite loop to output "U"

USART:
push r16                  ;push r16
push r18
ldi r16, 0x08             ;load r16 with 0x08
sts PORTD_DIRSET, r16     ;set receiver as output
ldi r16, 0x04             ;load r16 with 0x04
sts PORTD_DIRCLR, r16     ;set transmitter as input

ldi r16, 0x18
sts USARTD0_CTRLB, r16          ;enable receiver and transmitter

ldi r16, 0x33      ;USART asynchronous, 8 data bit, odd parity, 1 stop bit
```

```
sts USARTD0_CTRLC, r16

ldi r16, low(BSEL)                    ;8 bit BSEL value
sts USARTD0_BAUDCTRLA, r16

ldi r16, low(BSELHIGH)
ldi r18, 0xD0
or r16, r18                    ;BSCALE of -3, ignoring highest 4 bit

sts USARTD0_BAUDCTRLB, r16     ;load BAUDCTRLB with BSCALE and highest 4 bits of BSEL

ldi r16, 0x04                  ;turn the transmit pin idle by writing a one to it. pin2
is the transmit pin
sts PORTD_OUTSET, r16                    ;turn the transmit pin idle

pop r18
pop r16
ret


OUT_CHAR:               ;OUT_CHAR subroutine
push r16

/*
WAIT:
lds r16, USARTD0_STATUS
bst r16, 6    ;check TXCIF (bit 6, TXCIF:Transmit Complete Interrupt Flag) to see if
there is any ongoing transmission
brts COMPLETE
brtc WAIT
*/

COMPLETE:
lds r16, USARTD0_STATUS
bst r16, 5                        ;check the DREIF (Data register empty flag)
brts LOAD
brtc COMPLETE

LOAD:
sts USARTD0_DATA, r17               ;transit "U" to the data register

pop r16
ret




CLK:   ;take in a r17 value for prescaler. 32MHZ = 0x00 for prescale
push r16              ;push r16
ldi r16, 0b00000010   ;bit 1 is the 32Mhz oscillator
sts OSC_CTRL, r16     ;store r16 into the OSC_CTRL

NSTABLE:
lds r16, OSC_STATUS     ;load oscillator status into r16
bst r16, 1              ;check if 32Mhz oscillator is stable
brts STABLE             ;branch if stable
brtc NSTABLE            ;loop again if non-stable
```

```
STABLE:
ldi r16, 0xD8   ;writing IOREG to r16
sts CPU_CCP, r16 ;write IOREG to CPU_CCP to enable change
ldi r16, 0b00000001  ;write this to r16. corresponds to 32Mhz oscillator
sts CLK_CTRL, r16    ;select the 32Mhz oscillator

ldi r16, 0xD8    ;writing IOREG for prescaler
sts CPU_CCP, r16 ;for prescaler
sts CLK_PSCTRL, r23  ;r23 will be initialized outside the subroutine for prescale.
32/8=4MHZ

pop r16          ;pop r16
ret              ;return to main routine
```

Part C

```
/* HW 4 Part C
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program calls the OUT_CHAR and transmit a string to Putty Terminal.
                            In this case, the string transmitted by this program is my
name.
*/


.include "ATxmega128A1Udef.inc"        ;include the file
.list                                  ;list it

.org 0x0000                            ;start our program here
rjmp MAIN                              ;jump to main

.equ stack_init=0x3FFF   ;initialize stack pointer
.equ BSELHIGH=(((8)*((32000000/(16*115200))-1))>>8)
.equ BSEL=((8)*((32000000/(16*115200))-1))

.org 0x100
Table :.db 'P', 'e', 'n', 'g', 'z', 'h', 'a', 'o', ' ', 'Z', 'h', 'u', 0x00


.org 0x200

MAIN:
ldi r23, 0x00   ;setting for 32MHZ subroutine

rcall CLK

ldi YL, low(stack_init)    ;Load 0xFF to YL
out CPU_SPL, YL                        ;transfer to CPU_SPL
ldi YL, high(stack_init)   ;Load 0x3F to YH
out CPU_SPH, YL                        ;transfer to CPU_SPH

ldi ZL, low(Table << 1)    ;load lower byte of table address
ldi ZH, high(Table << 1)   ;load higher byte of table address
```

```
rcall USART         ;call subroutine to set up USART system

rcall OUT_STRING

DONE:
       rjmp DONE

USART:
push r16                  ;push r16
push r18
ldi r16, 0x08             ;load r16 with 0x08
sts PORTD_DIRSET, r16     ;set receiver as output
ldi r16, 0x04             ;load r16 with 0x04
sts PORTD_DIRCLR, r16     ;set transmitter as input

ldi r16, 0x18
sts USARTD0_CTRLB, r16          ;enable receiver and transmitter

ldi r16, 0x33      ;USART asynchronous, 8 data bit, odd parity, 1 stop bit
sts USARTD0_CTRLC, r16

ldi r16, low(BSEL)                    ;8 bit BSEL value
sts USARTD0_BAUDCTRLA, r16

ldi r16, low(BSELHIGH)
ldi r18, 0xD0
or r16, r18                   ;BSCALE of -3, ignoring highest 4 bit

sts USARTD0_BAUDCTRLB, r16     ;load BAUDCTRLB with BSCALE and highest 4 bits of BSEL

ldi r16, 0x04                  ;turn the transmit pin idle by writing a one to it. pin2
is the transmit pin
sts PORTD_OUTSET, r16                    ;turn the transmit pin idle

pop r18
pop r16
ret



OUT_CHAR:               ;OUT_CHAR subroutine
push r16

/*
WAIT:
lds r16, USARTD0_STATUS
bst r16, 6    ;check TXCIF (bit 6, TXCIF:Transmit Complete Interrupt Flag) to see if
there is any ongoing transmission
brts COMPLETE
brtc WAIT
*/

COMPLETE:
lds r16, USARTD0_STATUS  ;check the DREIF (Data register empty flag)
bst r16, 5
brts LOAD
brtc COMPLETE
```

```
LOAD:
sts USARTD0_DATA, r17               ;transit "U" to the data register

pop r16
ret

OUT_STRING:
push r17

CONTINUE:
elpm r17, Z+                ;load value in Z to r16. Post increment Z
cpi r17,0                   ;check if the value is the null character
breq RETURN                 ;if it is the null character. prepare to return from
subroutine
rcall OUT_CHAR              ;call OUT_CHAR subroutine
rjmp CONTINUE               ;Loop until the null character has been detected.

RETURN:
pop r17
ret



CLK:   ;take in a r17 value for prescaler. 32MHZ = 0x00 for prescale
push r16              ;push r16
ldi r16, 0b00000010   ;bit 1 is the 32Mhz oscillator
sts OSC_CTRL, r16     ;store r16 into the OSC_CTRL

NSTABLE:
lds r16, OSC_STATUS     ;load oscillator status into r16
bst r16, 1              ;check if 32Mhz oscillator is stable
brts STABLE             ;branch if stable
brtc NSTABLE            ;loop again if non-stable

STABLE:
ldi r16, 0xD8   ;writing IOREG to r16
sts CPU_CCP, r16 ;write IOREG to CPU_CCP to enable change
ldi r16, 0b00000001  ;write this to r16. corresponds to 32Mhz oscillator
sts CLK_CTRL, r16    ;select the 32Mhz oscillator

ldi r16, 0xD8    ;writing IOREG for prescaler
sts CPU_CCP, r16 ;for prescaler
sts CLK_PSCTRL, r23  ;r23 will be initialized outside the subroutine for prescale.
32/8=4MHZ

pop r16          ;pop r16
ret              ;return to main routine
```

Part D

```
/* HW 4 Part D
   Name: Pengzhao Zhu
```

```
   Section#: 112D
   TA Name: Chris Crary
   Description: This program turns in the input from the keyboard using the IN_CHAR
subroutine.
                        Then it echoes it back to the Putty terminal using the
OUT_CHAR (forever).
*/

.include "ATxmega128A1Udef.inc"        ;include the file
.list                                  ;list it

.org 0x0000                            ;start our program here
rjmp MAIN                              ;jump to main

.equ stack_init=0x3FFF   ;initialize stack pointer
.equ BSELHIGH=(((8)*((32000000/(16*115200))-1))>>8)
.equ BSEL=((8)*((32000000/(16*115200))-1))

.org 0x100
Table :.db 'P', 'e', 'n', 'g', 'z', 'h', 'a', 'o', ' ', 'Z', 'h', 'u', 0x00


.org 0x200

MAIN:
ldi r23, 0x00   ;setting for 32MHZ subroutine

rcall CLK

ldi YL, low(stack_init)    ;Load 0xFF to YL
out CPU_SPL, YL                        ;transfer to CPU_SPL
ldi YL, high(stack_init)   ;Load 0x3F to YH
out CPU_SPH, YL                        ;transfer to CPU_SPH

ldi ZL, low(Table << 1)    ;load lower byte of table address
ldi ZH, high(Table << 1)   ;load higher byte of table address

rcall USART         ;call subroutine to set up USART system

LOOP:
       rcall IN_CHAR
       rcall OUT_CHAR
       rjmp LOOP

USART:
push r16                  ;push r16
push r18
ldi r16, 0x08             ;load r16 with 0x08
sts PORTD_DIRSET, r16     ;set receiver as output
ldi r16, 0x04             ;load r16 with 0x04
sts PORTD_DIRCLR, r16     ;set transmitter as input

ldi r16, 0x18
sts USARTD0_CTRLB, r16          ;enable receiver and transmitter

ldi r16, 0x33      ;USART asynchronous, 8 data bit, odd parity, 1 stop bit
sts USARTD0_CTRLC, r16
```

```
ldi r16, low(BSEL)                    ;8 bit BSEL value
sts USARTD0_BAUDCTRLA, r16

ldi r16, low(BSELHIGH)
ldi r18, 0xD0
or r16, r18                    ;BSCALE of -3, ignoring highest 4 bit

sts USARTD0_BAUDCTRLB, r16     ;load BAUDCTRLB with BSCALE and highest 4 bits of BSEL

ldi r16, 0x04                  ;turn the transmit pin idle by writing a one to it. pin2
is the transmit pin
sts PORTD_OUTSET, r16                    ;turn the transmit pin idle

pop r18
pop r16
ret


OUT_CHAR:               ;OUT_CHAR subroutine
push r16

/*
WAIT:
lds r16, USARTD0_STATUS
bst r16, 6    ;check TXCIF (bit 6, TXCIF:Transmit Complete Interrupt Flag) to see if
there is any ongoing transmission
brts COMPLETE
brtc WAIT
*/

COMPLETE:
lds r16, USARTD0_STATUS  ;check the DREIF (Data register empty flag)
bst r16, 5
brts LOAD
brtc COMPLETE

LOAD:
sts USARTD0_DATA, r17              ;transmit information typed on keypad
pop r16
ret

OUT_STRING:
push r17

CONTINUE:
elpm r17, Z+                ;load value in Z to r16. Post increment Z
cpi r17,0                   ;check if the value is the null character
breq RETURN                 ;if it is the null character. prepare to return from
subroutine
rcall OUT_CHAR              ;call OUT_CHAR subroutine
rjmp CONTINUE               ;Loop until the null character has been detected.

RETURN:
pop r17
ret

IN_CHAR:
push r16
```

```
NOT:
lds r16, USARTD0_STATUS      ;check the receive complete flag
bst r16, 7
brts RECEIVE
brtc NOT

RECEIVE:
lds r17, USARTD0_DATA             ;if set, put the received data into r17 (to be used
later)
pop r16
ret



CLK:   ;take in a r17 value for prescaler. 32MHZ = 0x00 for prescale
push r16              ;push r16
ldi r16, 0b00000010   ;bit 1 is the 32Mhz oscillator
sts OSC_CTRL, r16     ;store r16 into the OSC_CTRL

NSTABLE:
lds r16, OSC_STATUS     ;load oscillator status into r16
bst r16, 1              ;check if 32Mhz oscillator is stable
brts STABLE             ;branch if stable
brtc NSTABLE            ;loop again if non-stable

STABLE:
ldi r16, 0xD8   ;writing IOREG to r16
sts CPU_CCP, r16 ;write IOREG to CPU_CCP to enable change
ldi r16, 0b00000001  ;write this to r16. corresponds to 32Mhz oscillator
sts CLK_CTRL, r16    ;select the 32Mhz oscillator

ldi r16, 0xD8    ;writing IOREG for prescaler
sts CPU_CCP, r16 ;for prescaler
sts CLK_PSCTRL, r23  ;r23 will be initialized outside the subroutine for prescale.
32/8=4MHZ

pop r16          ;pop r16
ret              ;return to main routine
```

Part E

```
/* HW 4 Part E
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This program continously blink the Green LED with a period of .5 second
(toggle every .25 second).
                          While blinking the LED, it will trigger an interrupt when we
receive an input from the keypad and
                          echo it back to Putty.
*/


.include "ATxmega128A1Udef.inc"        ;include the file
.list                                  ;list it
```

```
.org 0x0000                             ;start our program here
rjmp MAIN                               ;jump to main

.equ stack_init=0x3FFF   ;initialize stack pointer
.equ BSELHIGH=(((8)*((32000000/(16*115200))-1))>>8)
.equ BSEL=((8)*((32000000/(16*115200))-1))
.equ toggle_timer= (32000000*.25)/1024

.org USARTF0_RXC_vect
       rjmp USART_ISR

.org 0x100
Table :.db 'P', 'e', 'n', 'g', 'z', 'h', 'a', 'o', ' ', 'Z', 'h', 'u', 0x00


.org 0x200

MAIN:
ldi r23, 0x00   ;setting for 32MHZ subroutine

rcall CLK

ldi YL, low(stack_init)    ;Load 0xFF to YL
out CPU_SPL, YL                         ;transfer to CPU_SPL
ldi YL, high(stack_init)   ;Load 0x3F to YH
out CPU_SPH, YL                         ;transfer to CPU_SPH

ldi ZL, low(Table << 1)    ;load lower byte of table address
ldi ZH, high(Table << 1)   ;load higher byte of table address

rcall USART         ;call subroutine to set up USART system

ldi r16, 0x20      ;load r16 with 0x20
sts PORTD_DIRSET, r16   ;set GREEN LED as output

ldi r16, low(toggle_timer)                    ;set the timer for 5ms to debounce
sts TCD0_PER, r16                                                   ;need to load low
and high byte of PER
ldi r16, high(toggle_timer)
sts TCD0_PER+1, r16

ldi r16, 0b00000111                           ;Timer clock for clk/1024
sts TCD0_CTRLA, r16

ldi r16, 0x00                                  ;setting the CNT back to zero
sts TCD0_CNT, r16

LOOP:
lds r16, TCD0_INTFLAGS                                              ;check the intflag
sbrs r16, 0
rjmp LOOP
rjmp TOGGLE                                    ;if intflag set, rjmp to toggle. in other
work, if CNT reaches per

TOGGLE:
ldi r16, 0x20                                                       ;use to toggle GREEN
LED
```

```
sts PORTD_OUTTGL, r16

ldi r16, 0x00                                ;setting the CNT back to zero
sts TCD0_CNT, r16                            ;resetting the timer CNT value

ldi r16, 0x01
sts TCD0_INTFLAGS, r16                      ;clears the interrupt flag

rjmp LOOP                                                                ;back to LOOP



USART:
push r16                 ;push r16
push r18
ldi r16, 0x08            ;load r16 with 0x08
sts PORTD_DIRSET, r16    ;set receiver as output
ldi r16, 0x04            ;load r16 with 0x04
sts PORTD_DIRCLR, r16    ;set transmitter as input

ldi r16, 0x18
sts USARTD0_CTRLB, r16          ;enable receiver and transmitter

ldi r16, 0x33      ;USART asynchronous, 8 data bit, odd parity, 1 stop bit
sts USARTD0_CTRLC, r16

ldi r16, low(BSEL)                     ;8 bit BSEL value
sts USARTD0_BAUDCTRLA, r16

ldi r16, low(BSELHIGH)
ldi r18, 0xD0
or r16, r18                    ;BSCALE of -3, ignoring highest 4 bit

sts USARTD0_BAUDCTRLB, r16     ;load BAUDCTRLB with BSCALE and highest 4 bits of BSEL

ldi r16, 0x04                  ;turn the transmit pin idle by writing a one to it. pin2
is the transmit pin
sts PORTD_OUTSET, r16                    ;turn the transmit pin idle

ldi r16, 0x10
sts USARTD0_CTRLA, r16         ;enable low level interrupt for "receive complete"

ldi r16, 0x01
sts PMIC_CTRL, r16      ;enable low level interrupt in the PMIC

sei

pop r18
pop r16
ret

USART_ISR:
push r18
lds r18, CPU_SREG
push r18
push r18

lds r18, USARTD0_DATA     ;read the information in the receive buffer
```

```
WAIT:
lds r16, USARTD0_STATUS  ;check the DREIF (Data register empty flag)
bst r16, 5
brts TRANSMIT
brtc WAIT

TRANSMIT:
sts USARTD0_DATA, r18              ;transmit received character out

ldi r16, 0x80
sts USARTD0_STATUS, r16           ;clear the receive complete interrupt flag


pop r16
pop r18
sts CPU_SREG, r18
pop r18

reti


OUT_CHAR:               ;OUT_CHAR subroutine
push r16

/*
WAIT:
lds r16, USARTD0_STATUS
bst r16, 6    ;check TXCIF (bit 6, TXCIF:Transmit Complete Interrupt Flag) to see if
there is any ongoing transmission
brts COMPLETE
brtc WAIT
*/

KEEPCHECK:
lds r16, USARTD0_STATUS  ;check the DREIF (Data register empty flag)
bst r16, 5
brts LOAD
brtc KEEPCHECK

LOAD:
sts USARTD0_DATA, r17              ;transmit information typed on keypad
pop r16
ret

OUT_STRING:
push r17

CONTINUE:
elpm r17, Z+                ;load value in Z to r16. Post increment Z
cpi r17,0                   ;check if the value is the null character
breq RETURN                 ;if it is the null character. prepare to return from
subroutine
rcall OUT_CHAR              ;call OUT_CHAR subroutine
rjmp CONTINUE               ;Loop until the null character has been detected.

RETURN:
pop r17
```

```
ret

IN_CHAR:
push r16

NOT:
lds r16, USARTD0_STATUS             ;check if receive complete interrupt Flag
bst r16, 7
brts RECEIVE
brtc NOT

RECEIVE:
lds r17, USARTD0_DATA                           ;if set, data to r17
pop r16
ret



CLK:   ;take in a r17 value for prescaler. 32MHZ = 0x00 for prescale
push r16              ;push r16
ldi r16, 0b00000010   ;bit 1 is the 32Mhz oscillator
sts OSC_CTRL, r16     ;store r16 into the OSC_CTRL

NSTABLE:
lds r16, OSC_STATUS     ;load oscillator status into r16
bst r16, 1              ;check if 32Mhz oscillator is stable
brts STABLE             ;branch if stable
brtc NSTABLE            ;loop again if non-stable

STABLE:
ldi r16, 0xD8   ;writing IOREG to r16
sts CPU_CCP, r16 ;write IOREG to CPU_CCP to enable change
ldi r16, 0b00000001  ;write this to r16. corresponds to 32Mhz oscillator
sts CLK_CTRL, r16    ;select the 32Mhz oscillator

ldi r16, 0xD8    ;writing IOREG for prescaler
sts CPU_CCP, r16 ;for prescaler
sts CLK_PSCTRL, r23  ;r23 will be initialized outside the subroutine for prescale.
32/8=4MHZ

pop r16          ;pop r16
ret              ;return to main routine
```

H) Appendix

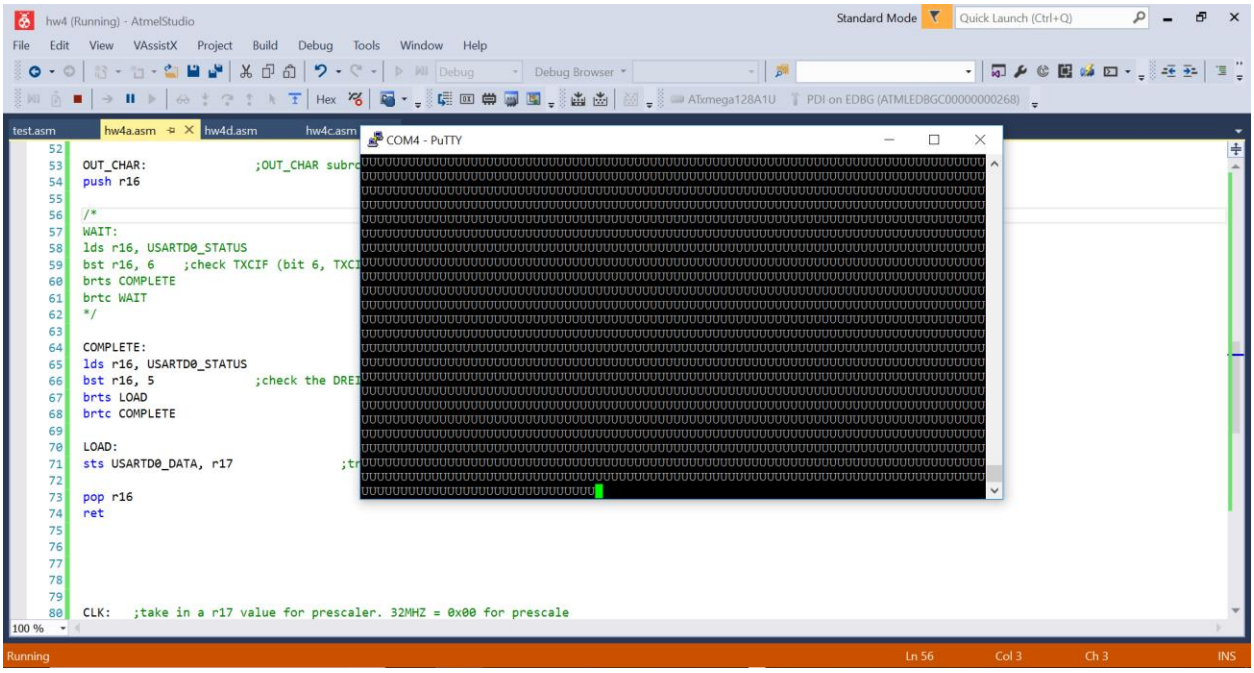

Figure 1: Part A- Transmit “U”

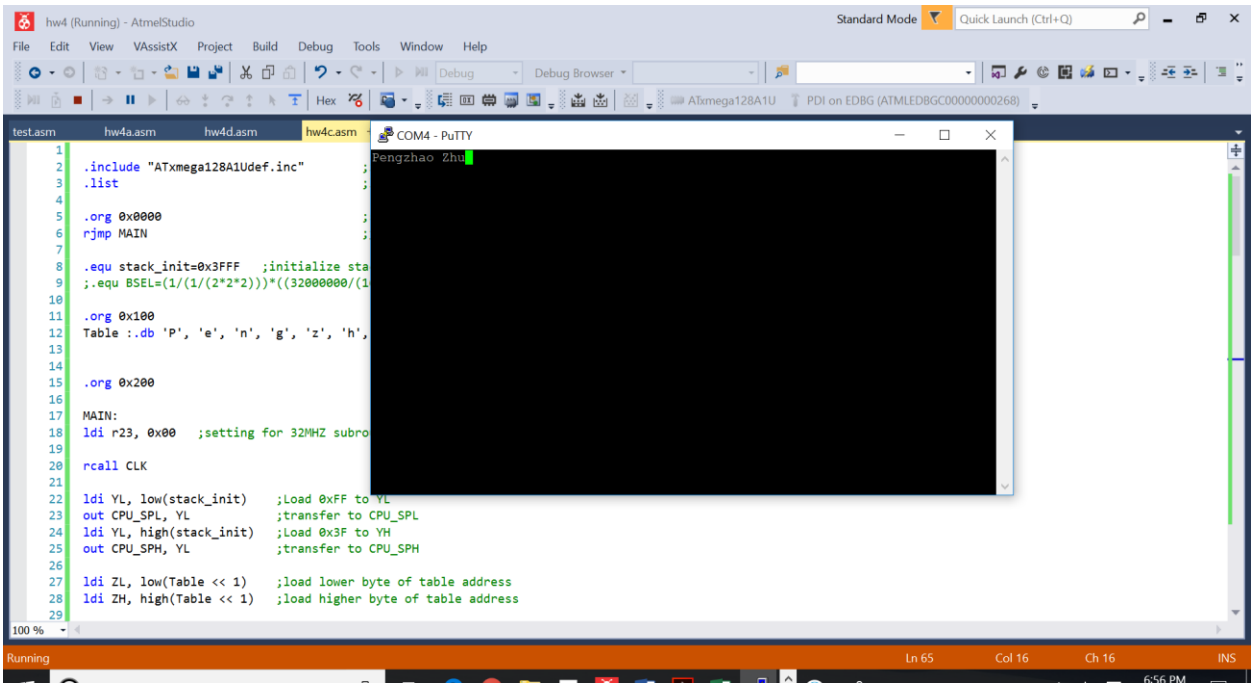

Figure 2: Part C- Transmit String (My Name)

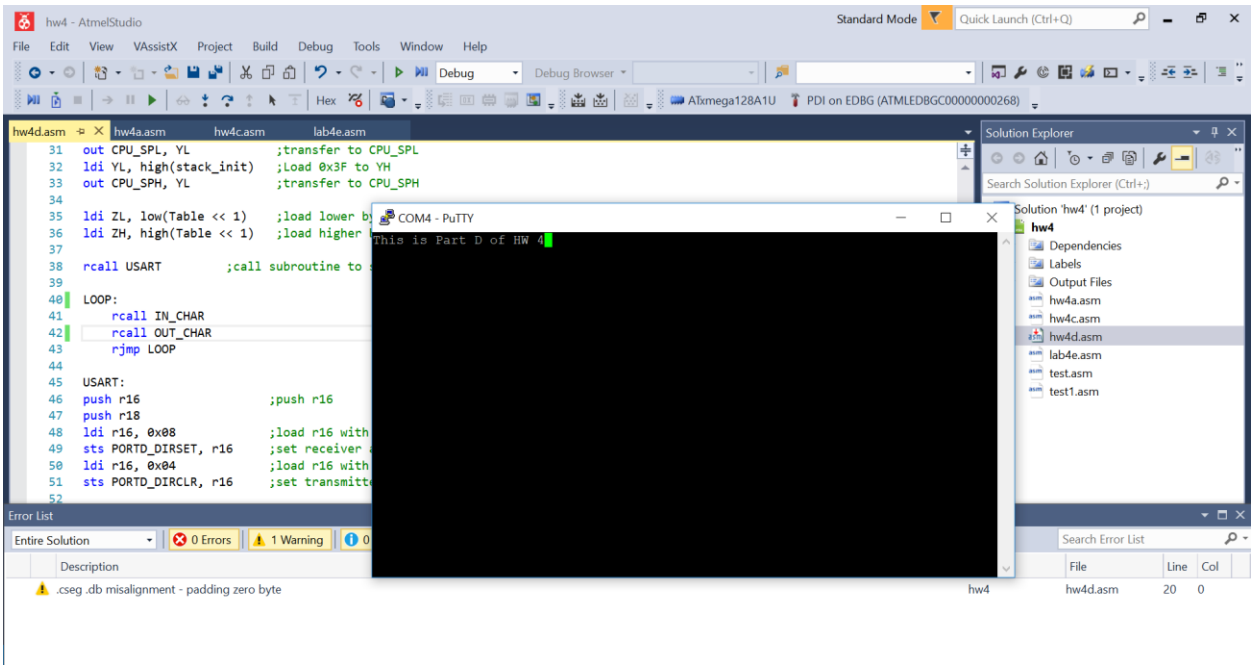

Figure 3: Part D- Keyboard Input/Putty Outside

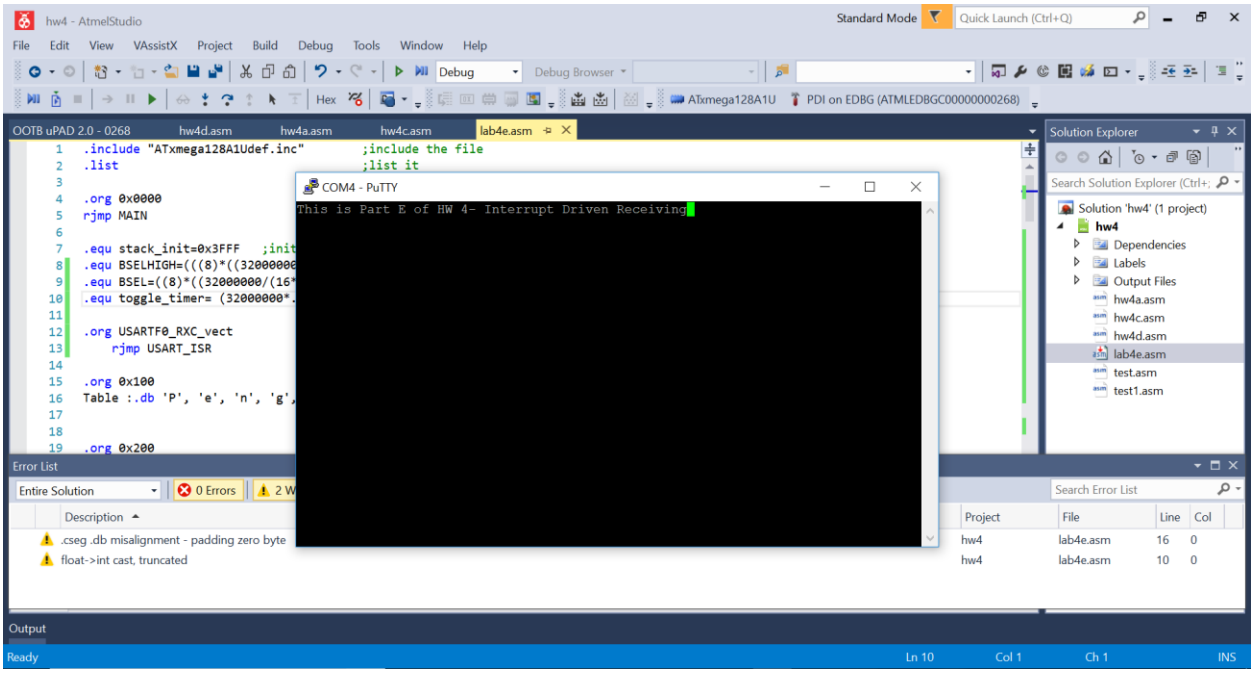

Figure 4: Part E- Interrupt Driven Echo while blinking Green LED