Lab 3
Pengzhao Zhu
Section: 112D

B) Prelab Questions

1. What is the default clock frequency of the XMEGA?
   The default clock frequency of the XMEGA is 2 MHZ.

2. What would you need to do to run the XMEGA at 8 MHz?
   Configure the XEMGA to run at 32MHZ. Then use the Prescalers to divide the 32MHZ by 4 to get 8 Hz. The numbers to load into CLK_PSCTRL is 0b00001100 (use of Prescaler A to divide by 4).

3. The XMEGA is rated to run at frequencies up to 32 MHz, but some peripherals run up to either 2x or 4x of this frequency. How can the XMEGA be configured to run at 32 MHz, the 2x peripheral clock at 64 MHz, and the 4x peripheral clock at 128 MHz without using any external clock sources? (There may be multiple solutions)
   The build-in phase locked loop (PLL) can be used to generate a high-frequency system clock. The PLL has a user-selectable multiplication factor of from 1 to 31. The output frequency, Fout, is given by the input frequency, Fin, multiplied by the multiplication factor, PLL_FAC.

4. Why are timers useful?
   Timers are useful because they are more accurate than software delays that we have used in the last lab. Their capabilities include accurate program execution timing, frequency and waveform generation, and input capture with time and frequency measurement of digital signals.

5. Where does potential external memory space begin and end?
   Potential external memory begin from 0x4000 to 0xFFFFFF

6. Why is our EBI configuration named “3-PORT ALE1 mode”?
   The EBI configuration is names 3-PORT ALE1 mode because it uses 3 ports (H, J K) to connect address lines, data lines, and control signals. It also uses ALE1 to latch address line signal for EBI use.

7. Which ports contain the necessary EBI signals? List each port and which pins are connected to each of the necessary EBI signals.
   Port K contains address lines 7:0 or 15:8. Port J contain data line 7:0. Port H contains ALE1, RE, and WE signals.

8. What is the memory size of the external SRAM? How do you know?
   Memory size of the external SRAM is 32K X 8. I figured this out by looking at the schematics. There are 15 address lines and 8 data lines. 15 address lines corresponds to 32K.

9. Would our external SRAM be aliased (i.e., partially address decoded) if we chose our chip select size to be 32K? How about 64K?
   No, our SRAM would not be aliased if we chose our chip select size to be 32K since it is a 32 K SRAM. It would be aliased (partially address decoded) if we chose 64 K as our chip select since our SRAM is only 32K. You can write to the same hardware address with two different addresses.

10. If you were to remove the trigger on the ALE signal and instead trigger on the falling edge of the CS0 signal, would the ALE signal still be active upon writing to each sequential address? Why or why not?
    No, it will not be active upon writing to each sequential address. ALE would not be active upon writing to each sequential address because it would only go true and latch when A15:8 changes. It would stay false unless A15:8 changes and it needs to latch.

C) Problems Encountered

The first problem I encountered in this lab was that I didn't know how to set the base address of the external SRAM to start at 0x200000 (Part C). To fix the problem, I emailed some of the teaching assistants and realized that I had to load EBI_CS0_BASEADDR with desired values to set the base address. It will then enable a certain amount of addresses according to the chip select.

The second problem I encountered was that I didn't know how to poll the interrupt flag to set the delay to 1 second. I was getting confused with the concepts of polling the interrupt flag and running an actual interrupt. I had to go back and reread the XMEGA manual section regarding interrupt to realize that I need to poll the interrupt timer overflow flag to determine the 1 second delay (without running an interrupt).

D) Future Application

By completing this lab, I gained the ability to add external memories to my microprocessor. This concept will be useful in the future when I need to work with devices that require a large amount of memories or when I need to add extensions to my microprocessor.

This lab also allowed me to learn the use of timer and how to change clock frequencies. Both concepts will be highly useful for later labs and for skills in my future career (for high frequency applications).

E) Schematics

N/A. Confirmed with one of the Teaching Assistants

F) Pseudocode/Flowcharts

Part A Pseudocode:

Initialize stack pointer address at 0x3FFF

Load stack pointer address to CPU_SPL and CPU_SPH

Initialize r17 to be 0x14 to divide by 8 later in the subroutine

Configure pin 7 of PORT C to be output

rcall CLK (subroutine to set the clock frequency to 32 MHZ)

Output to PORTCFG_CLKEVOUT (Pin 7 of Port C)

Infinite loop to end program

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 r17 (initialized outside the subroutine) to divide 32MHZ by 8 to get 4MHZ (or set up no change to remain 32MHZ)

pop r16

ret

Part B Pseudocode

Load low byte of decimal 255 to TCCO_PER

Load high byte of decimal 255 to TCCO_PER+1

Ldi r17, 0b00000111 ;prescaler CLK/1024

Transfer value in r17 to TCC0_CTRLA

Use PORTC_DIRSET to set PORTC as output

REPEAT:

Ldi r17, TCC0_CNT

sts PORTC_OUT, r17

rjmp REPEAT ;infinite loop to output count value to PORTC

Part C Pseudocode

rcall EBI subroutine

STARTOVER:

load Z pointer (using RAMPZ too) with address 0x200000

REPEAT:

ldi r16, 0xA5

st Z, r16

Check if address is at 0x207FFF ; to see if it has reached the 32 K limit

If yes, jump to STARTOVER to load Z pointer at 0x200000 again

If not, write to r17. Post-increment Z pointer

rjmp REPEAT

EBI Subroutine:

push r16, ZL, ZH

load EBI_CTRL with 0x01 to set up 3-port mode

Enable CS0, RE, WE, ALE, on Port H. Set them as output

Set CS0, RE, WE as high because they are active low

Set ALE as low because it is active low

Set Port J and Port K as output

Set pointer to point to EBI_CS0_BASEADDR

Load EBI_CS0_BASEADDR with the desired base address

Pop ZH, ZL, R16 (in order)

Part D Pseudocode

Initialize stack pointer address at 0x3FFF

Load stack pointer address to CPU_SPL and CPU_SPH

Set up r17 to stay at 32 MHZ (used in the subroutine)

rcall CLK ;change CLK frequency to 32MHZ

rcall EBI subroutine

Set Port A to be input

Set Port C to be output

Turn off Port C LED for now

STARTOVER:

load Z pointer (using RAMPZ too) with address 0x300000

REPEAT:

Take in value from input switches

ldi r16, 0xA5

st Z, r16

Check if address is at 0x207FFF ; to see if it has reached the 32 K limit

If yes, jump to STARTOVER to load Z pointer at 0x200000 again

If not, write to r18. Post-increment Z pointer

rcall TIMER (for 1 second delay)

Output to LED

rjmp REPEAT

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 r17 (initialized outside the subroutine) to set up CLK to stay 32 MHZ

pop r16

ret

EBI Subroutine:

push r16, ZL, ZH

load EBI_CTRL with 0x01 to set up 3-port mode

Enable CS0, RE, WE, ALE, on Port H. Set them as output

Set CS0, RE, WE as high because they are active low

Set ALE as low because it is active low

Set Port J and Port K as output

Set pointer to point to EBI_CS0_BASEADDR

Load EBI_CS0_BASEADDR with the desired base address

Pop ZH, ZL, R16 (in order)

TIMER (subroutine):

push r17

Load 0x12 to TCCO_PER

Load 0x7A to TCCO_PER+1

Ldi r17, 0b00000111 ;prescaler CLK/1024

Transfer value in r17 to TCC0_CTRLA

NOTSET:

nop

Check if timer overflow interrupt flag is set

If set, branch to RETURN

If not, branch to NOTSET

RETURN:

ldi r17, 0x01 ;to clear the flag

sts TCC0_INTFLAGS, R17 ;clears the flag

pop r17

ret

G) Program Code

Part A Code

```
/* Lab 3 Part A
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program configures the XMEGA clock to run at 4MHZ or 32MHZ
(depending on situation)
*/


.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

MAIN:
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, 0b00010100  ;divide by 8 to change from 32 MHZ to 4 MHZ in the subroutine. 0x14

ldi r16, 0b10000000   ;load value into r16. configure pin 7 as output
sts PORTC_DIRSET, r16 ;configure pin 7 as output
rcall CLK
ldi r16, 0b00001001   ;output CLKPER 4 on PORT C pin 7. pin 7 is the default. 0000=not
used, 10=output CLKPER4, 01=PORTC
sts PORTCFG_CLKEVOUT, r16    ;output to PORTCFG_CLKEVOUT

DONE:
       rjmp DONE         ;infinite loop to end the program

CLK:
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, r17  ;r17 will be initialized outside the subroutine for prescale.
32/8=4MHZ

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

Part B Code

```
/* Lab 3 Part B
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program ulitizes the Timer system to allow the CNT register to count
to 255. The CNT will increment one time
               every 1024 clock cycles
*/

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

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

MAIN:

ldi r17, 0xFF               ;load low byte
sts TCC0_PER, r17           ;0X826. should
ldi r17, 0x00              ;load high byte
sts TCC0_PER+1, r17         ;0x827
ldi r17, 0b00000111       ;prescaler CLK/1024
sts TCC0_CTRLA, r17       ;CTRLA controls the count (CNT)

ldi r17, 0xFF             ;set as output
sts PORTC_DIRSET, r17     ;set as output

REPEAT:
lds r17, TCC0_CNT         ;load value from TCC0_CNT to r17
sts PORTC_OUT, r17        ;output value at r17 to PORTC
rjmp REPEAT               ;REPEAT
```

Part C Code

```
/* Lab 3 Part C
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program configures the 3-Port EBI system on the XMEGA. It will then
write 0xA5 to all memory addresses
               of the external 32K SRAM starting at address 0x200000.
*/

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

```
.list                                   ;list it

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

.set IN_PORT = 0x200000         ;beginning at end of the 32K SRAM. base address
;.set IPORT_END = 0x207FFF

MAIN:
rcall EBI                   ;call EBI subroutine

STARTOVER:

ldi ZL, byte3(In_PORT)        ;load 0x20 into ZL
out CPU_RAMPZ, ZL             ;ZL value into RAMPZ
ldi ZL, low(IN_PORT)          ;load 0x00 into ZL
ldi ZH, byte2(IN_PORT)        ;load 0x00 into ZH


REPEAT:

ldi r16, 0xA5                 ;load 0xA5 into r16
st Z, r16                     ;r16 value into value pointed by Z pointer
cpi ZH, 0x7F                  ;first check for if 32K limit is reached
breq CHECK                    ;if equal, branch to do second check

LOAD:
ld r17, Z+                    ;load value to Z pointer address to r17. post increment
rjmp REPEAT                   ;jump to REPEAT

CHECK:
cpi ZL, 0xFF                   ;second check for if 32K limit is reached
breq STARTOVER                 ; if limit is reached. start over from 0x200000
brne LOAD                      ;if not, branch to LOAD


EBI:   ;takes in IN_PORT, IN_PORT = 24 bit address for 32 K SRAM
push r16                        ;push r16
push ZL                         ;push ZL
push ZH                         ;push ZH
ldi r16, 0x01   ; 4 bit data bus, data multiplexed with address byte 0 and 1, 3 port
sts EBI_CTRL, r16   ;configure mode to be 3 port

ldi r16, 0b00011101   ;set for 32K SRAM chip select
sts EBI_CS0_CTRLA, r16   ;set for 32K SRAM

ldi r16, 0b00010111   ;hex 0x17, enable CS0, RE, WE, ALE1
sts PORTH_DIRSET, r16 ; set CS0, RE, WE, ALE1 as output
ldi r16, 0b00010011     ;bit 0=we, bit 1=re, bit 4= CS0
sts PORTH_OUTSET, r16 ;set false value to CS0, WE, RE
ldi r16, 0b00000100   ;bit 2= ALE1
sts PORTH_OUTCLR, r16 ;set false value to ALE

ldi r16, 0xFF                    ;0xFF to r16
sts PORTJ_DIRSET, r16            ;set PORTJ to be output
ldi r16, 0xFF                    ;0xFF to r16. unneccesary, but I am still including it
sts PORTK_DIRSET, r16            ;set PORTK to be output
```

```
ldi ZL, low(EBI_CS0_BASEADDR)        ;ZL pointer point to low byte of EBI_CS0_BASEADDR
ldi ZH, high(EBI_CS0_BASEADDR)       ;ZH pointer point to high byte of EBI_CS1_BASEADDR
ldi r16, byte2(IN_PORT)              ;transfer middle byte of base address to lower byte
of base address
st Z+, r16                           ;r16 value to address pointed to Z pointer. post
increment Y pointer
ldi r16, byte3(IN_PORT)              ;transfer high byte of base address to upper byte of
base address
st Z, r16                            ;r16 value to address pointed to Z pointer
pop ZH                               ;pop ZH
pop ZL                               ;pop ZL
pop r16                              ;pop r16
ret                                  ;return from subroutine
```

Part D Code

```
/* Lab 3 Part D
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program stores data read from the DIP switches to sequential memory
locations every one second.
                          It will then read it back, and store the value to the LED
bank.
*/


.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 IN_PORT=0x300000

MAIN:
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, 0x00  ;set up 32MHZ clock in
rcall CLK      ;subroutine to set up 32Mhz clock
rcall EBI      ;subroutine to set up EBI of 32K and base address of 0x300000

ldi r16, 0xFF
sts PORTA_DIRCLR, r16    ;set Port A to be input
sts PORTC_DIRSET, r16    ;set Port C to be output
sts PORTC_OUTSET, r16    ;turn off the LED for now

STARTOVER:

ldi ZL, byte3(In_PORT)    ;load highest byte of 0x300000
out CPU_RAMPZ, ZL         ;RAMPZ point to 30
```

```
ldi ZL, byte1(IN_PORT)    ;ZL point to 00
ldi ZH, byte2(IN_PORT)    ;ZH point to 00

REPEAT:
lds r16, PORTA_IN         ;take in value from input switches
cpi ZH, 0x7F              ;check if middle byte is 0x7F to see if we have reach the limit
of the external SRAM
breq CHECK                ;if equal, branch to check again

LOAD:
st Z, r16          ; write to external memory
ld r18, Z+         ;read it back from external memory
rcall TIMER        ;call timer
sts PORTC_OUT, r18 ;output to LED
rjmp REPEAT        ;repeat

CHECK:
cpi ZL, 0xFF       ;check if the lower byte is 0xFF to see if we have reach the limit of
the 32K external SRAM
breq STARTOVER     ;if we have. start from 0x300000 again
brne LOAD          ; otherwise, branch to load


; the rest are just subroutines


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, r17  ;r17 will be initialized outside the subroutine for prescale.
32/8=4MHZ

pop r16          ;pop r16
ret              ;return to main routine


EBI:   ;takes in IN_PORT, IN_PORT = 24 bit address for 32 K SRAM
push r16
push ZL
push ZH
ldi r16, 0x01   ; 4 bit data bus, data multiplexed with address byte 0 and 1, 3 port
sts EBI_CTRL, r16   ;configure mode to be 3 port
```

```
ldi r16, 0b00011101   ;set for 32K SRAM chip select
sts EBI_CS0_CTRLA, r16   ;set for 32K SRAM

ldi r16, 0b00010111   ;hex 0x17, enable CS0, RE, WE, ALE1
sts PORTH_DIRSET, r16 ; set CS0, RE, WE, ALE1 as output
ldi r16, 0b00010011     ;bit 0=we, bit 1=re, bit 4= CS0
sts PORTH_OUTSET, r16 ;set false value to CS0, WE, RE
ldi r16, 0b00000100   ;bit 2= ALE1
sts PORTH_OUTCLR, r16 ;set false value to ALE

ldi r16, 0xFF
sts PORTJ_DIRSET, r16            ;set port J to be output
sts PORTK_DIRSET, r16            ;set port K to be output

ldi ZL, low(EBI_CS0_BASEADDR)        ;ZL pointer point to low byte of EBI_CS0_BASEADDR
ldi ZH, high(EBI_CS0_BASEADDR)       ;ZH pointer point to high byte of EBI_CS1_BASEADDR
ldi r16, byte2(IN_PORT)              ;transfer middle byte of base address to lower byte
of base address
st Z+, r16                           ;r16 value to Z pointer address. post increment Z
pointer
ldi r16, byte3(IN_PORT)              ;transfer high byte of base address to upper byte of
base address
st Z, r16                            ;r16 value to Z pointer address.
pop ZH                               ;pop ZH
pop ZL                               ;pop ZL
pop r16                              ;pop r16
ret                                  ;return from subroutine

TIMER:   ;delay for 1 second
push r17                              ;push r17
ldi r17, 0x12               ;load low byte of 0x7A12
sts TCC0_PER, r17           ;transfer to TCC0_PER
ldi r17, 0x7A              ;load high byte of 0x7A12
sts TCC0_PER+1, r17         ;transfer to TCC0_PER+1
ldi r17, 0b00000111       ;prescaler CLK/1024
sts TCC0_CTRLA, r17       ;CTRLA controls the count (CNT)

NOTSET:
nop                        ;delay
lds r17, TCC0_INTFLAGS     ;check if the flag is set
bst r17, 0                 ;check the zero bite
brts RETURN                ;if set, branch to return
brtc NOTSET                ;if not, branch to NOTSET and continue

RETURN:
ldi r17, 0x01            ;to clear the flag
sts TCC0_INTFLAGS, r17   ;clears the flag
pop r17                  ;pop r17
ret                      ;return from subroutine
```

H) Appendix

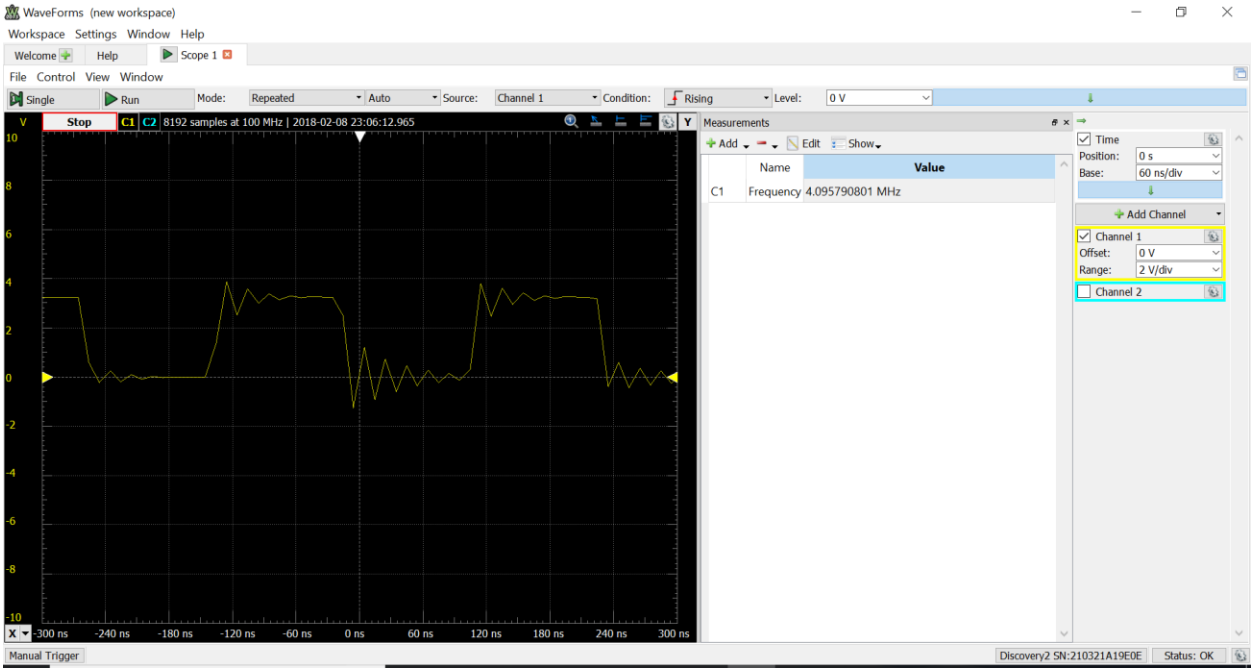

Figure 1: XMEGA 4 MHZ Clock Configuration (PART A)

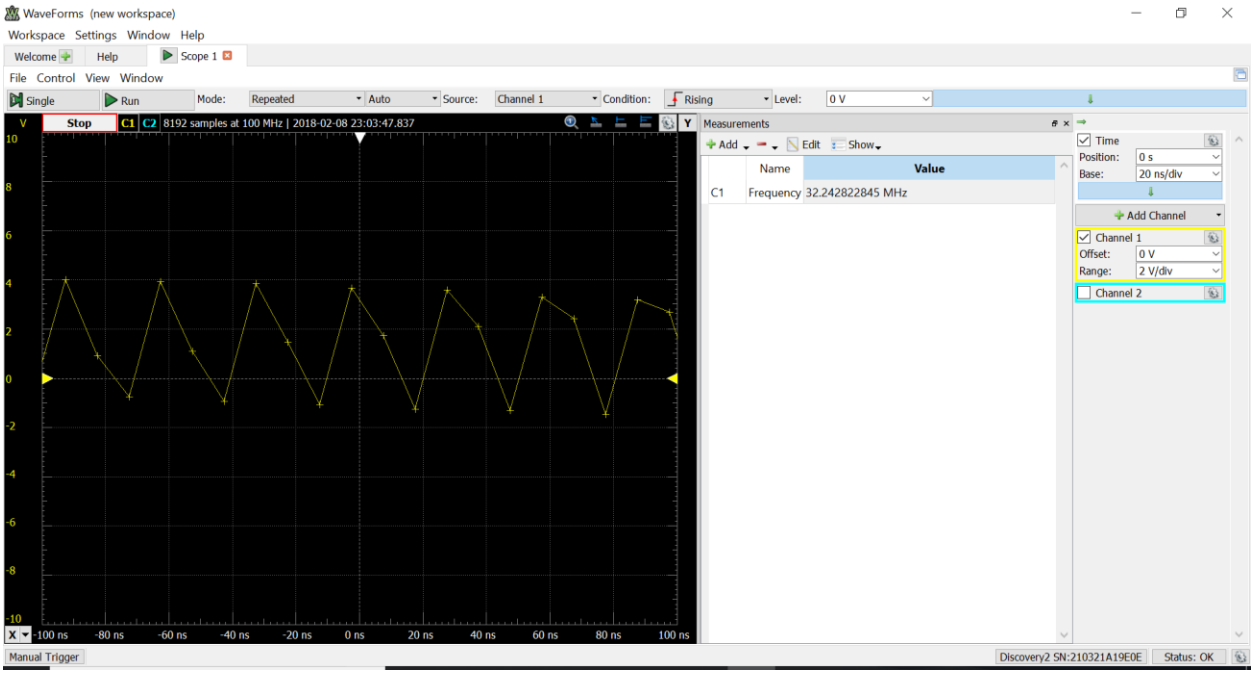

Figure 2: XMEGA 32 MHZ Clock Configuration (PART A)

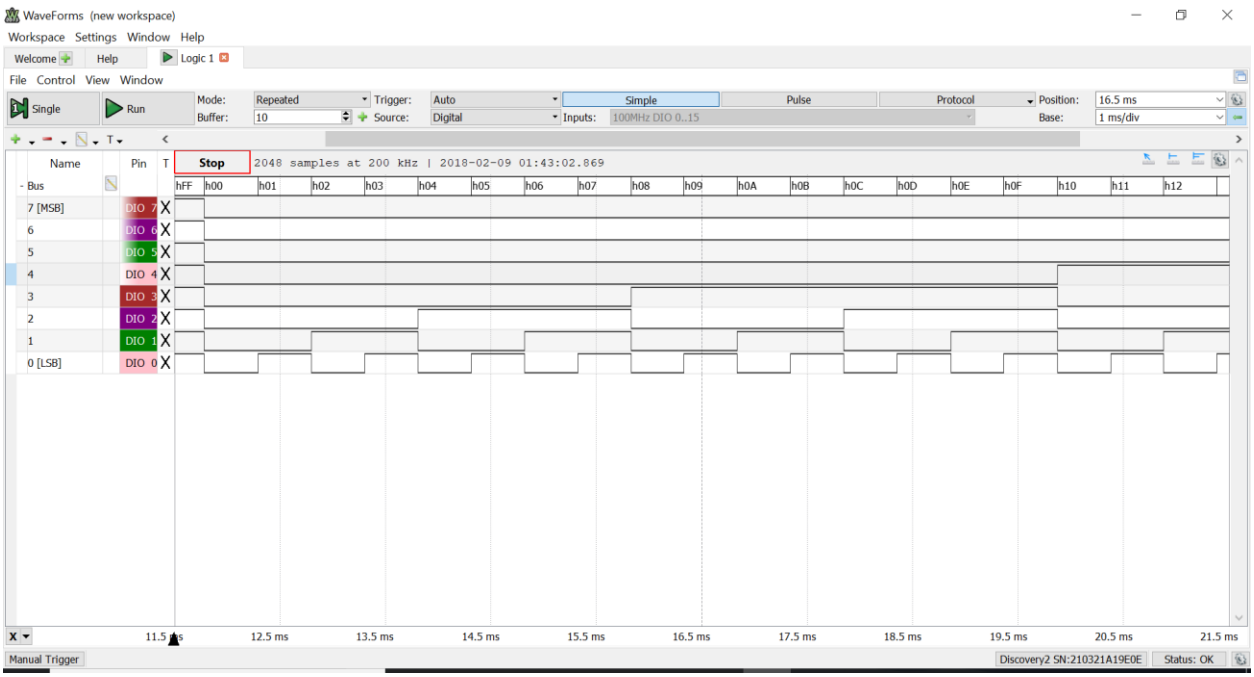

Figure 3: Incrementing CNT Screenshot (PART B)

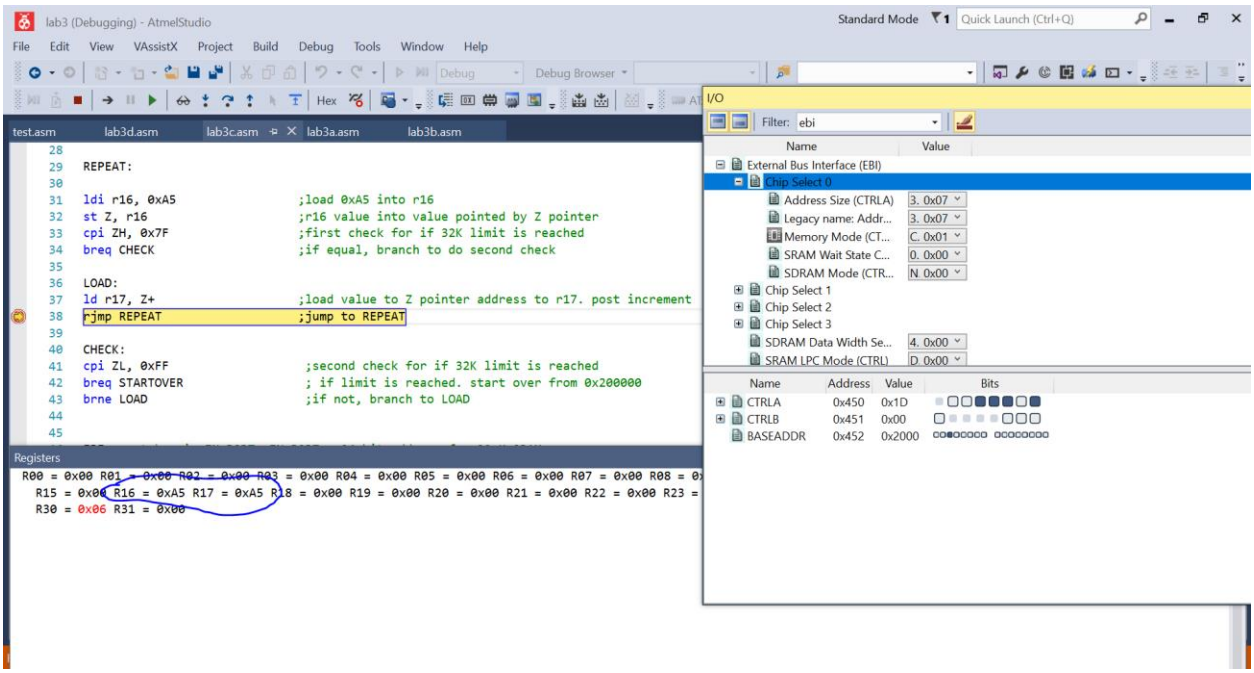

Figure 4: Read/Write to External SRAM confirmation (Part C)

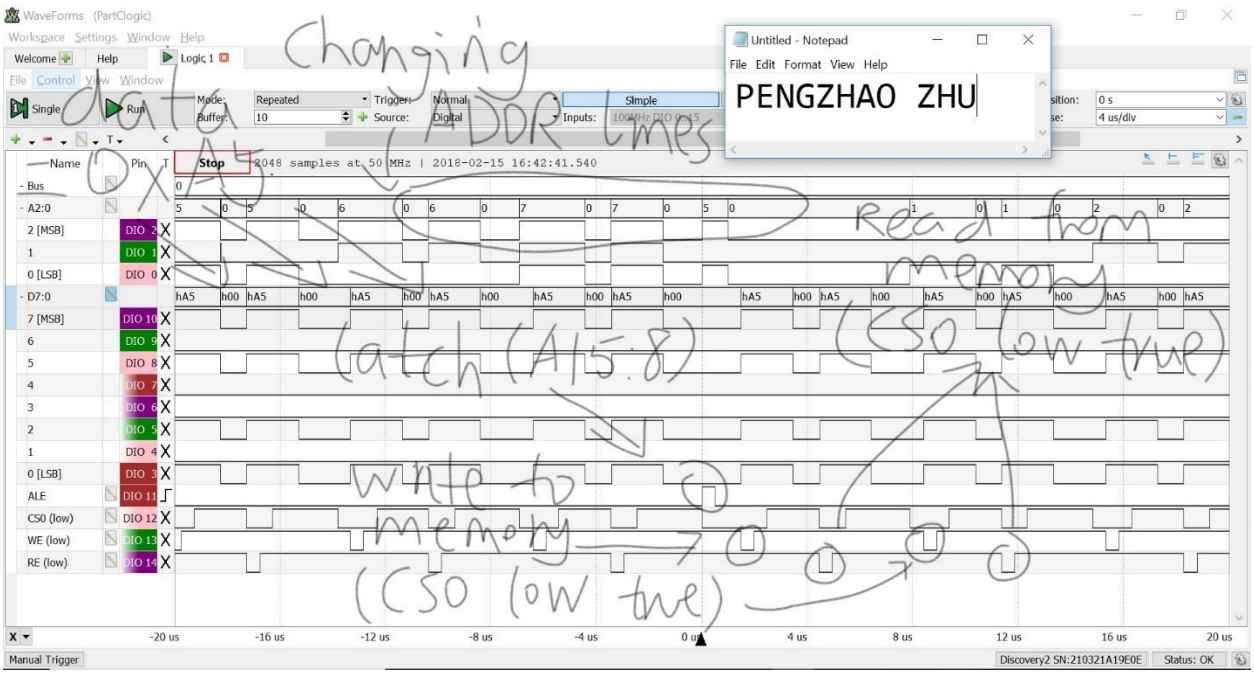

Figure 5: Annotated EBI Configuration Signals (PART C)

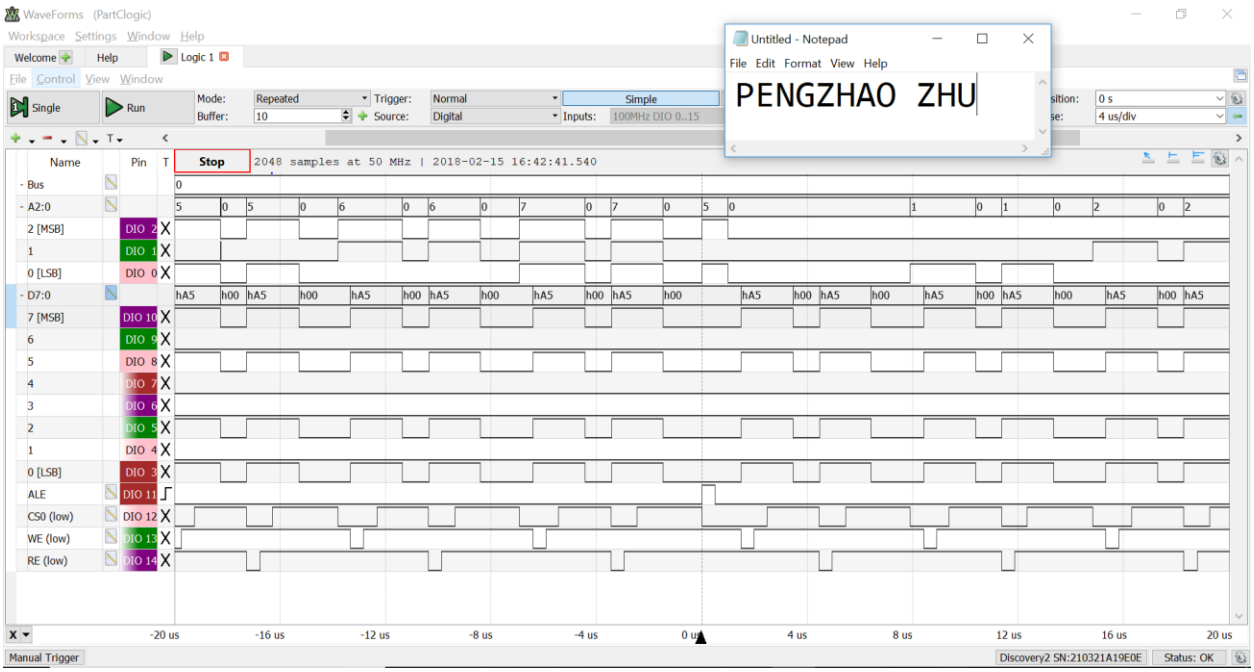

Figure 6: EBI Configuration Signal without Annotation (Part C, same as Figure 4)

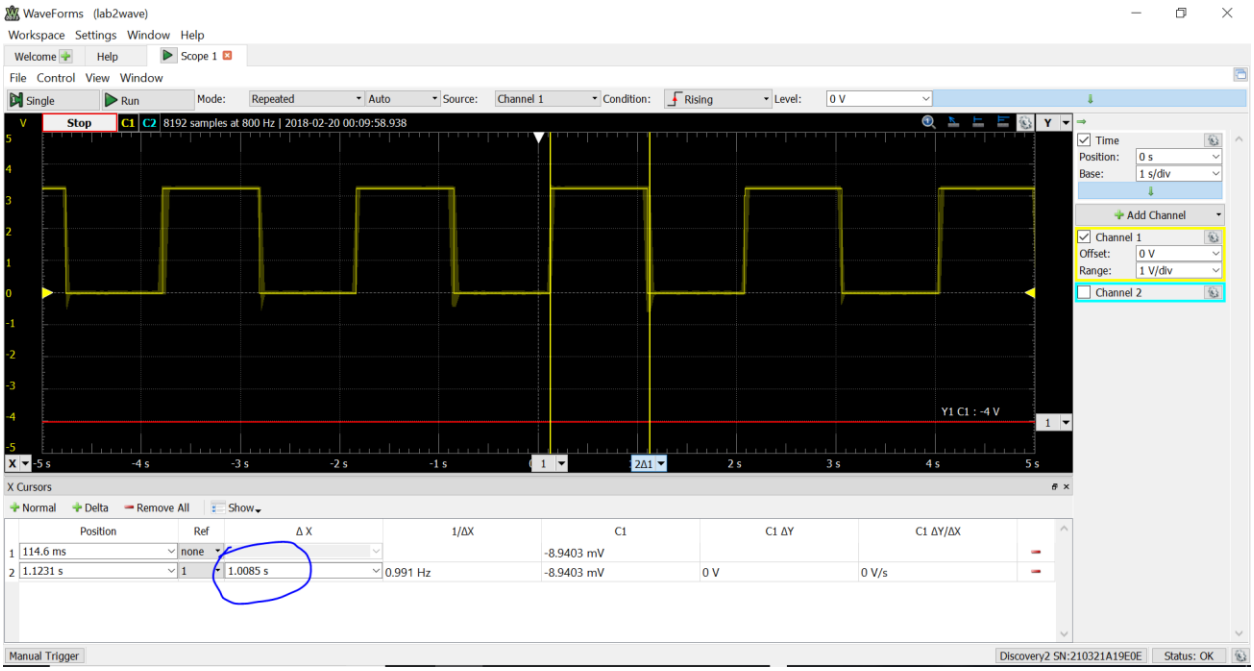

Figure 7: Timer 1 second interval confirmation with test program (Part D)