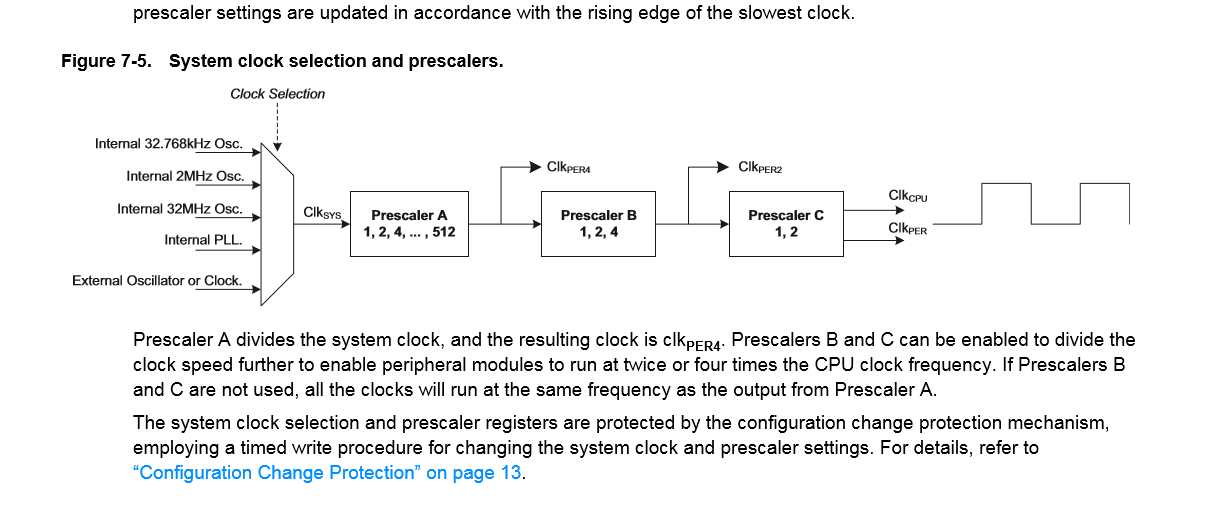
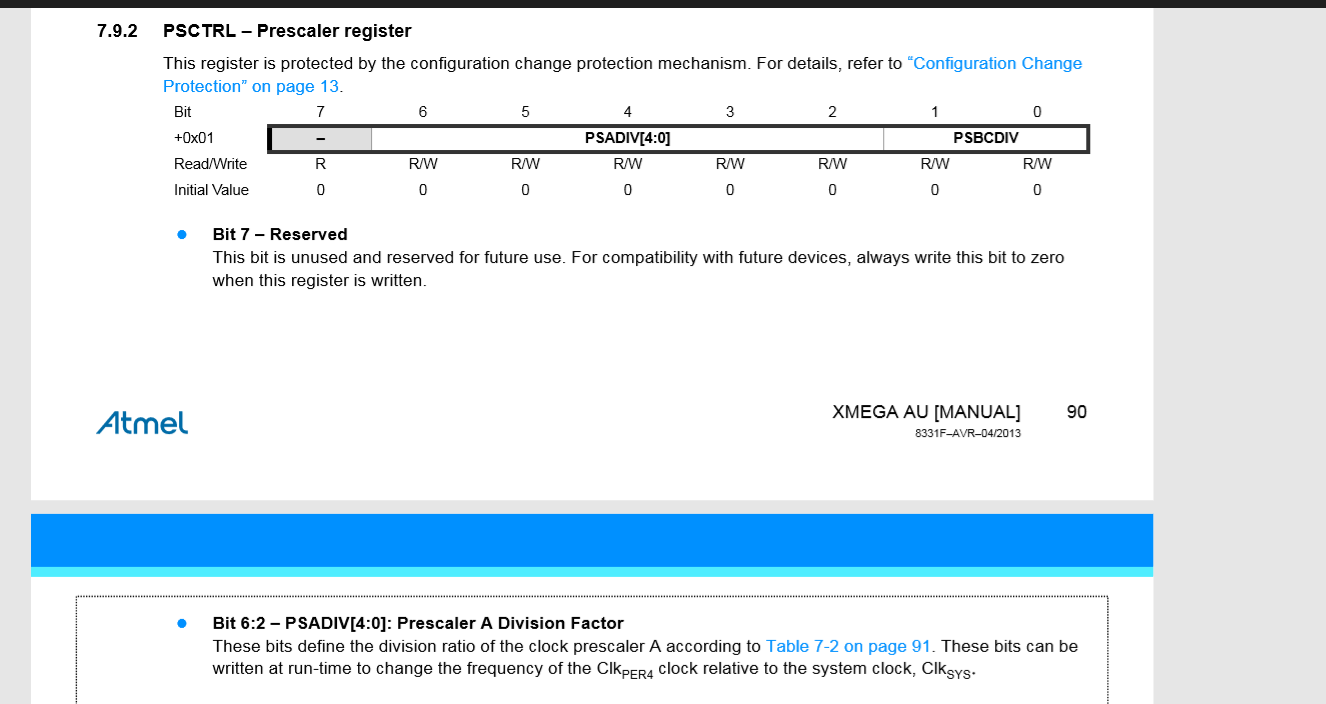
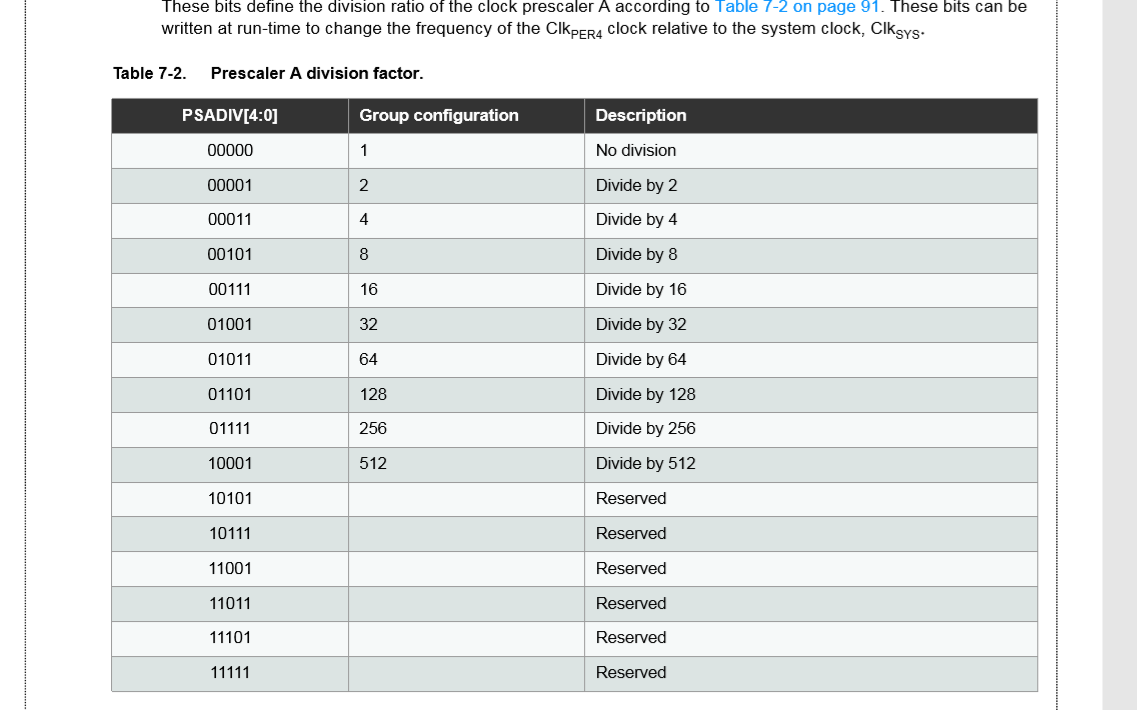
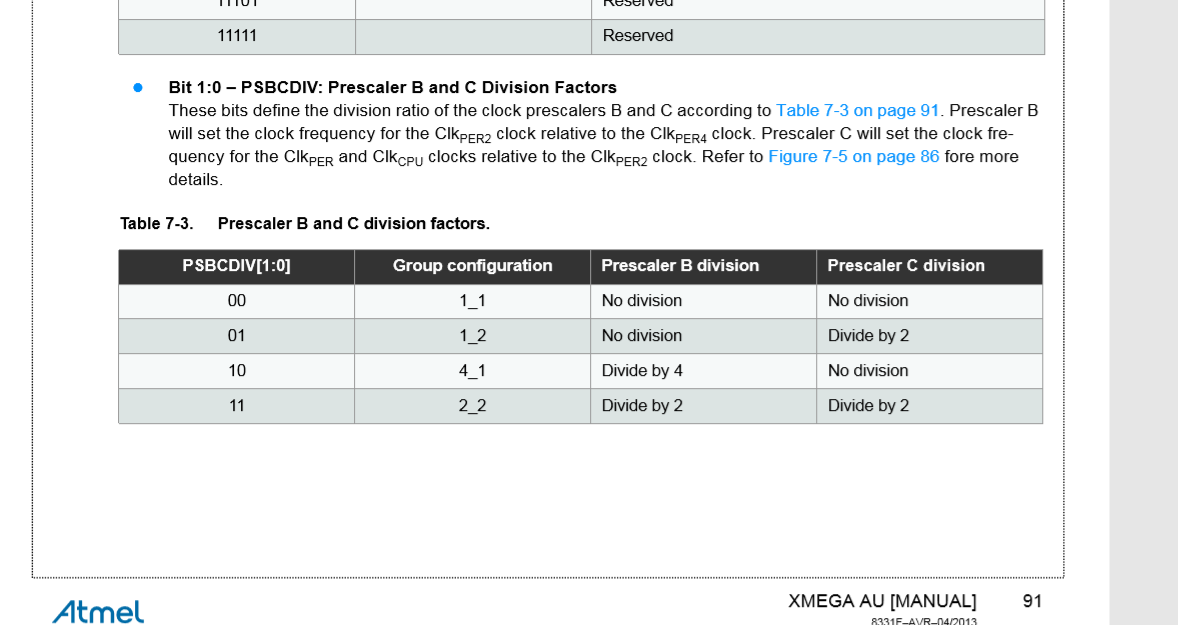
Lab 3 Notes



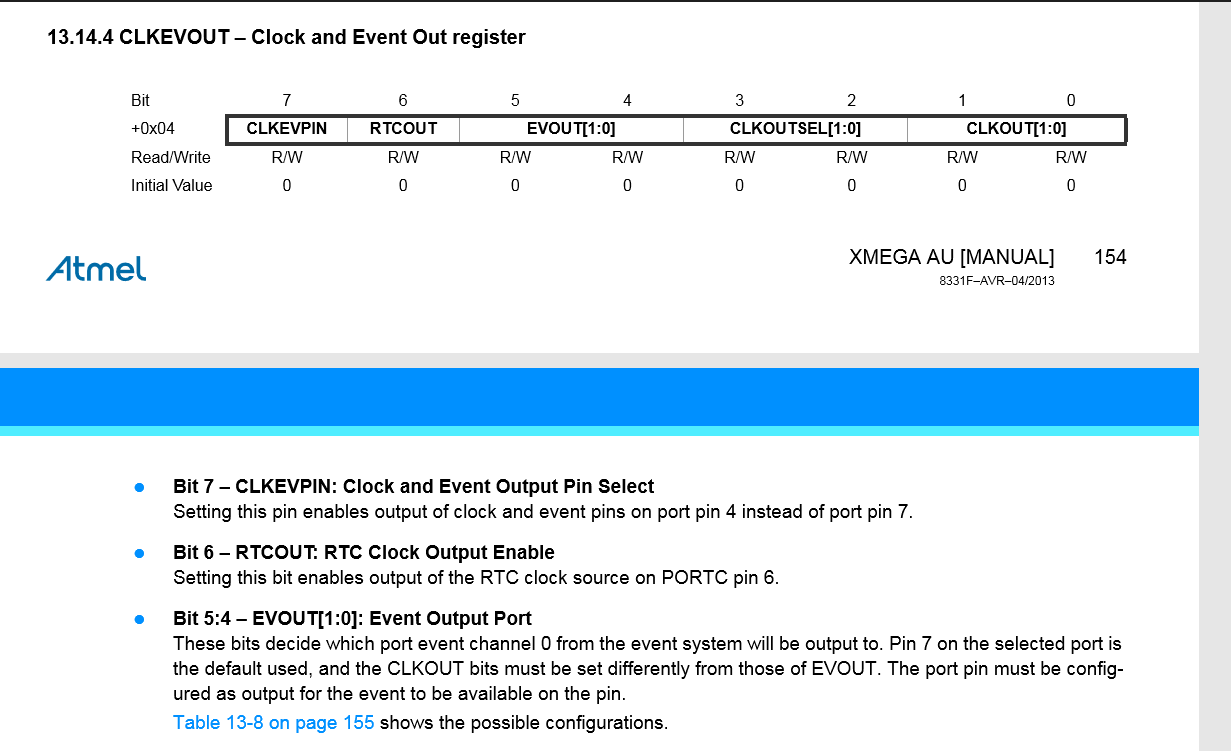
Prescale

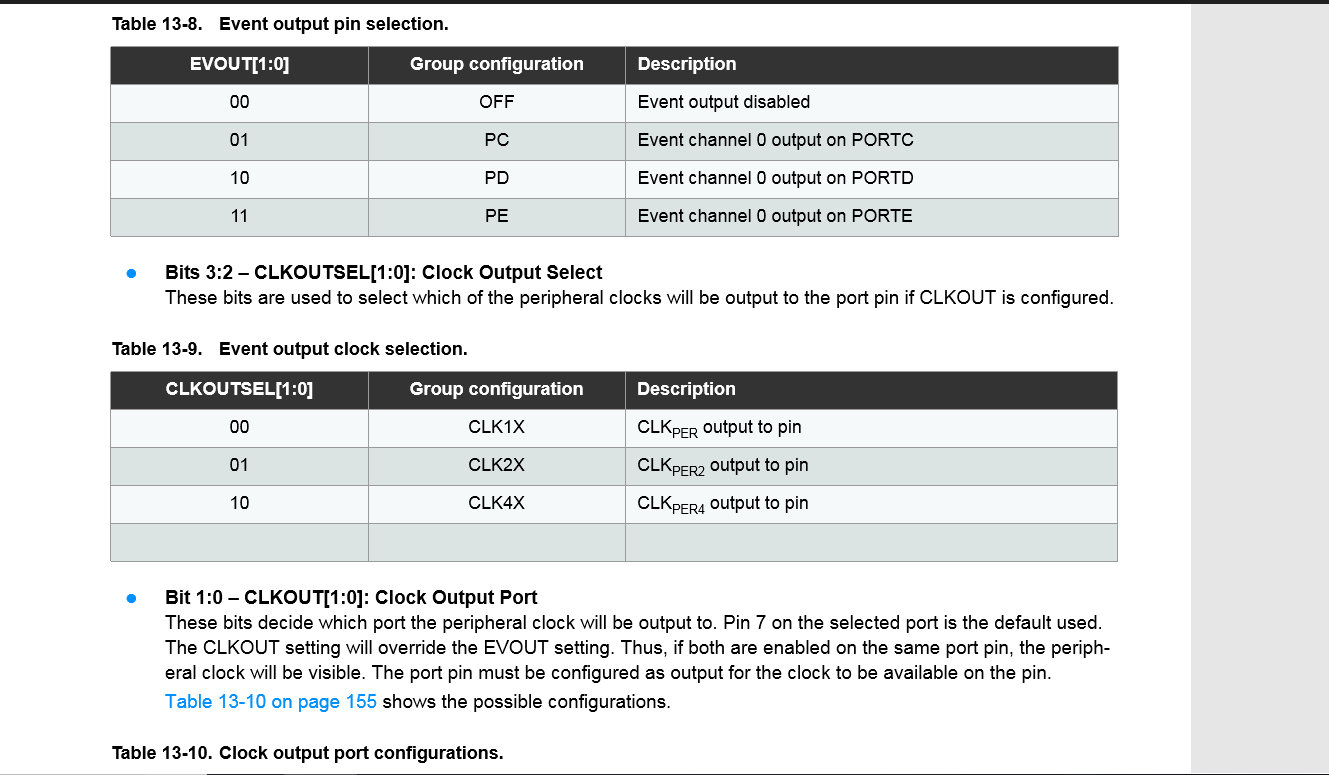


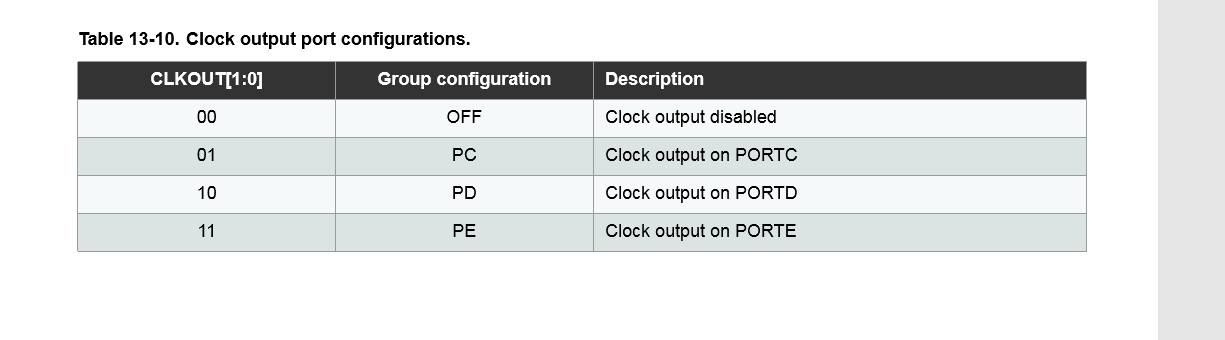




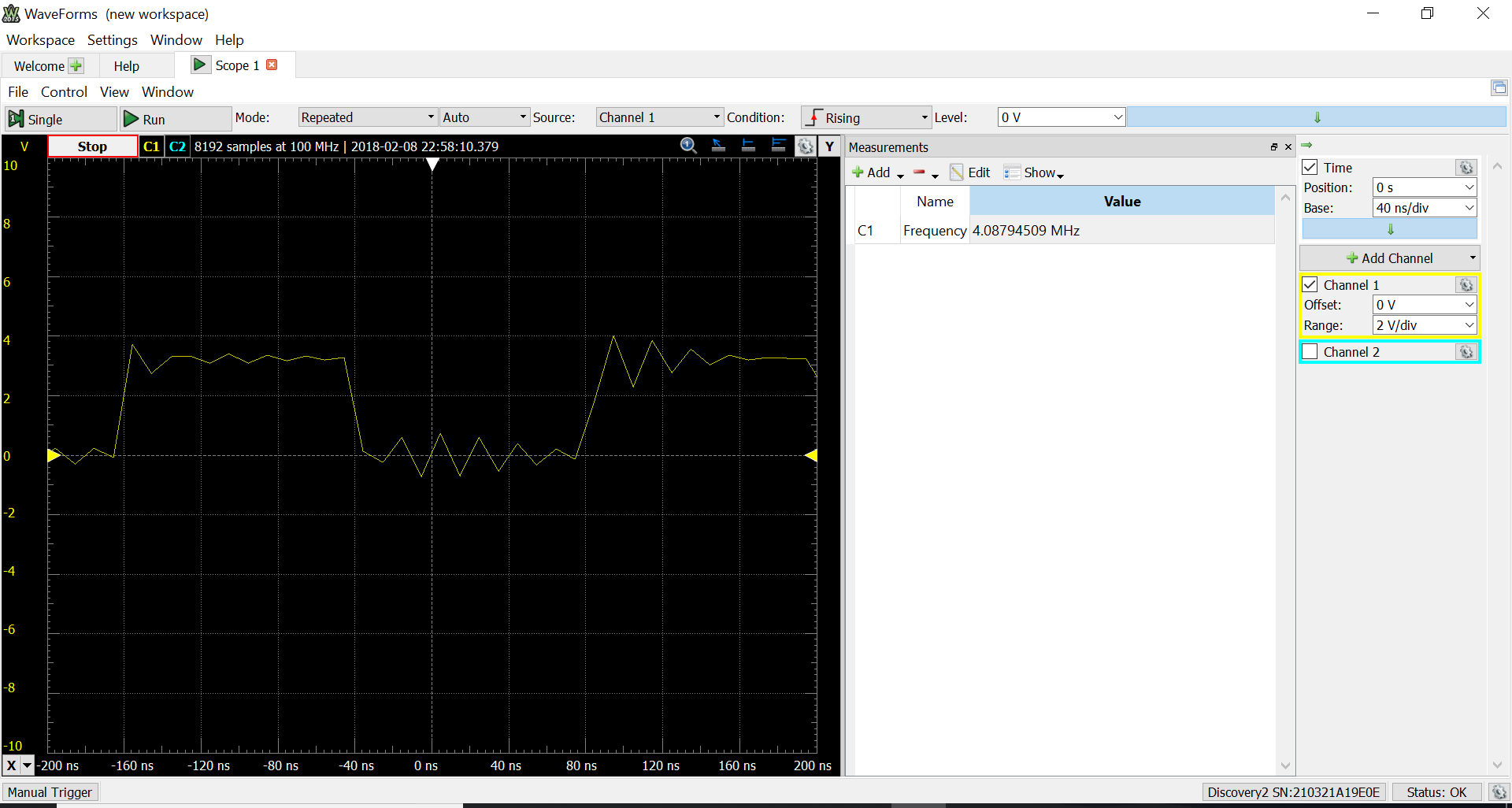
CLKEVOUT Notes



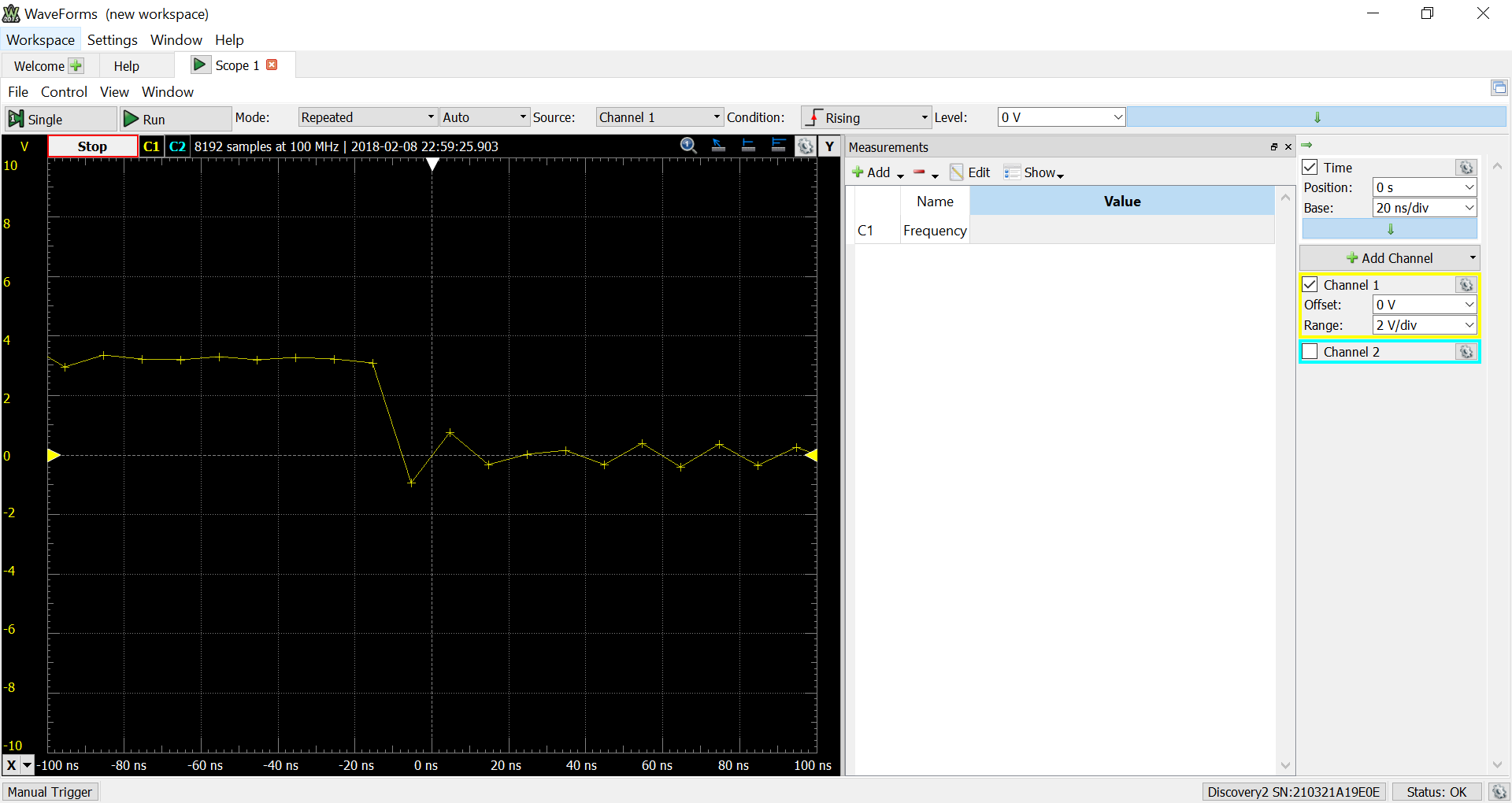




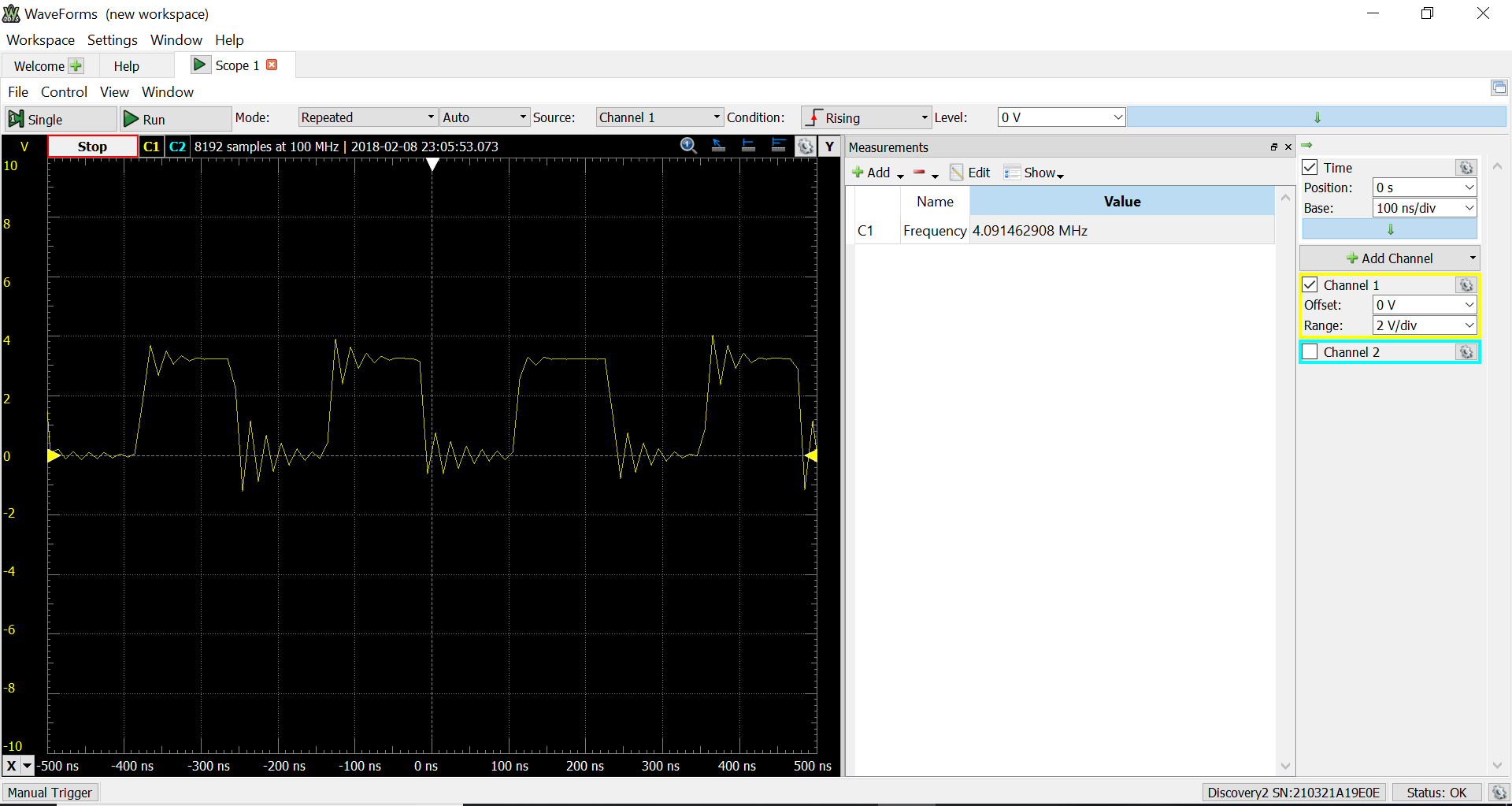
Part A

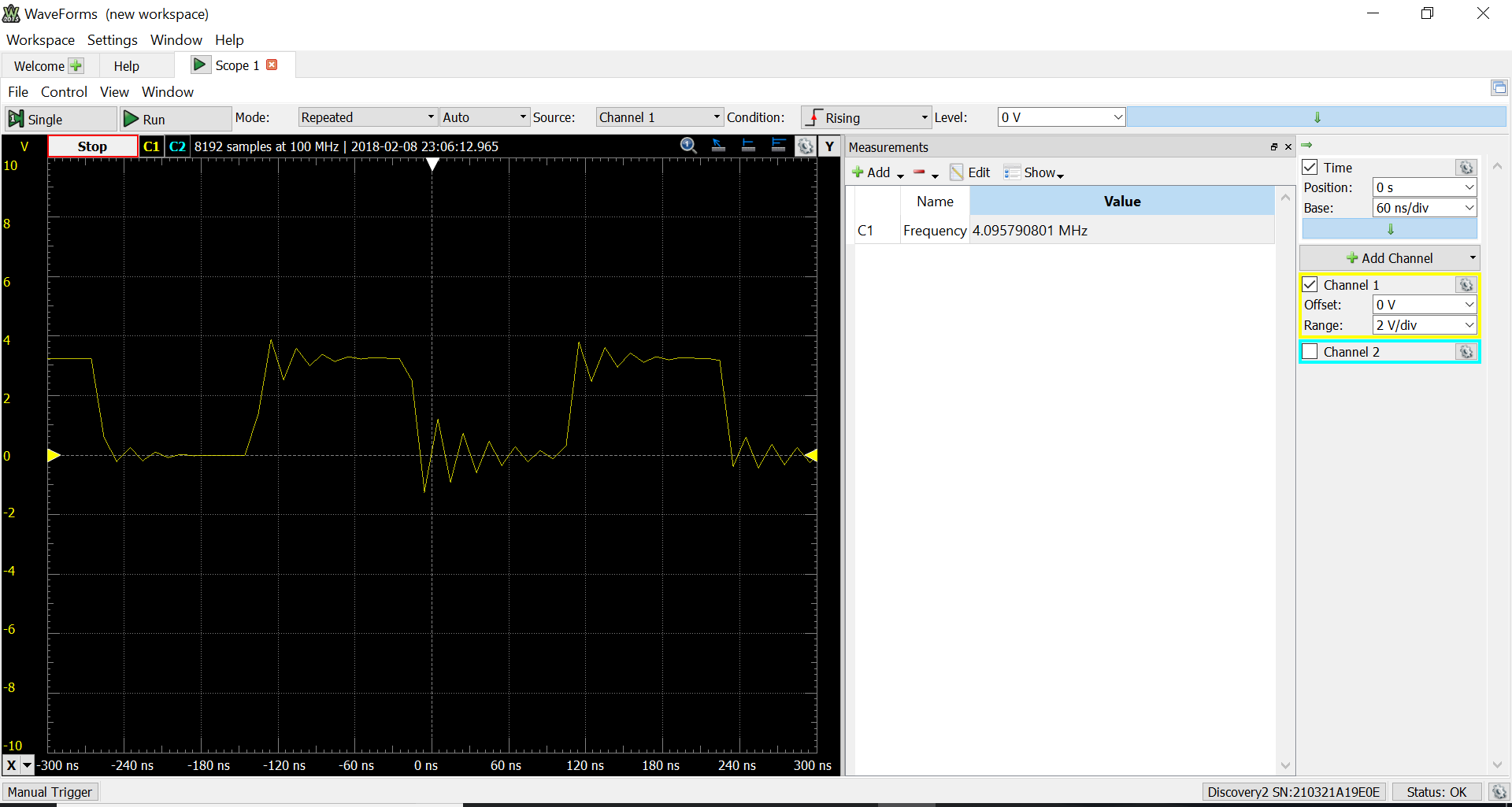


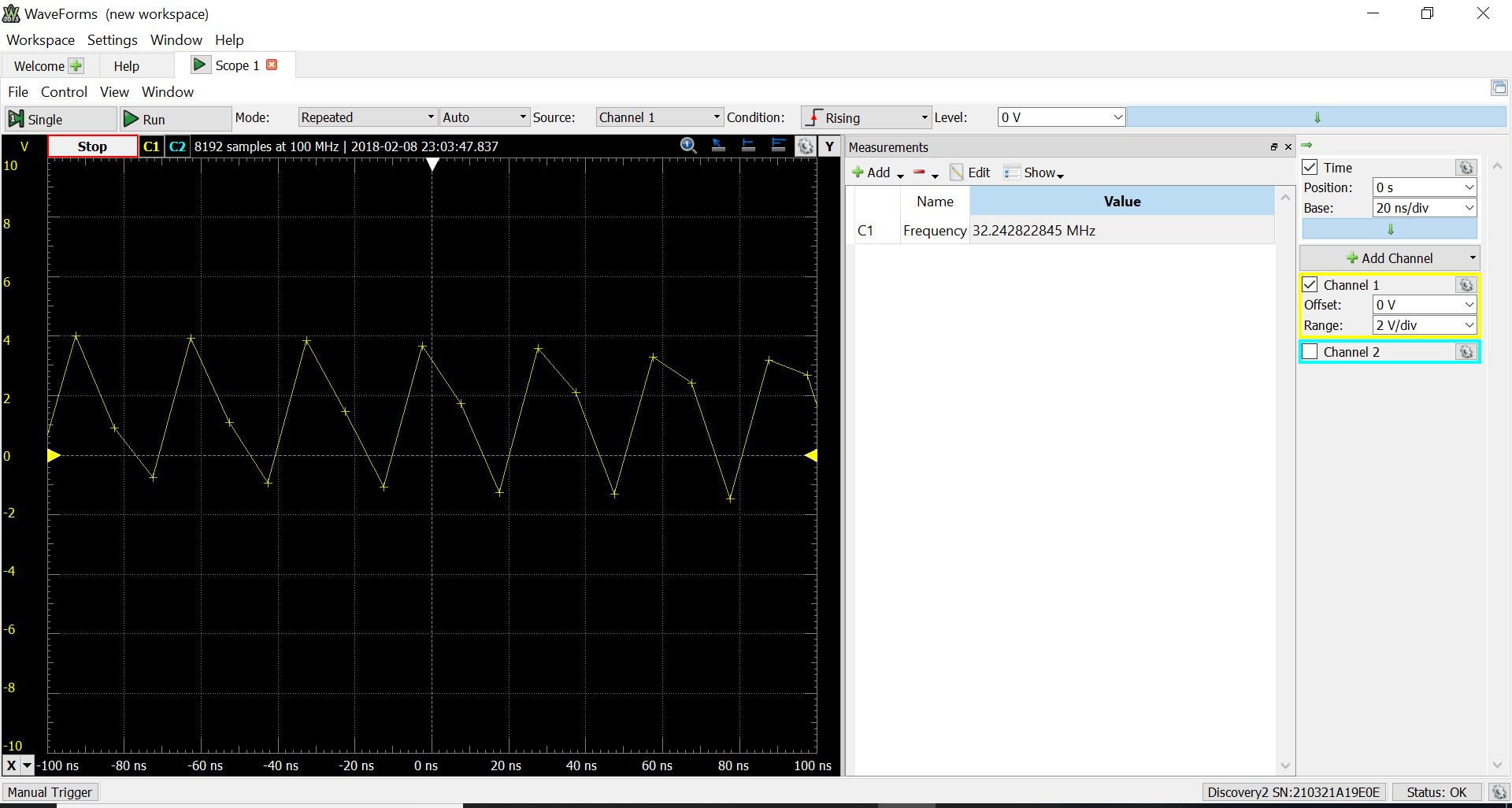
4 MHZ (40 ns/div)



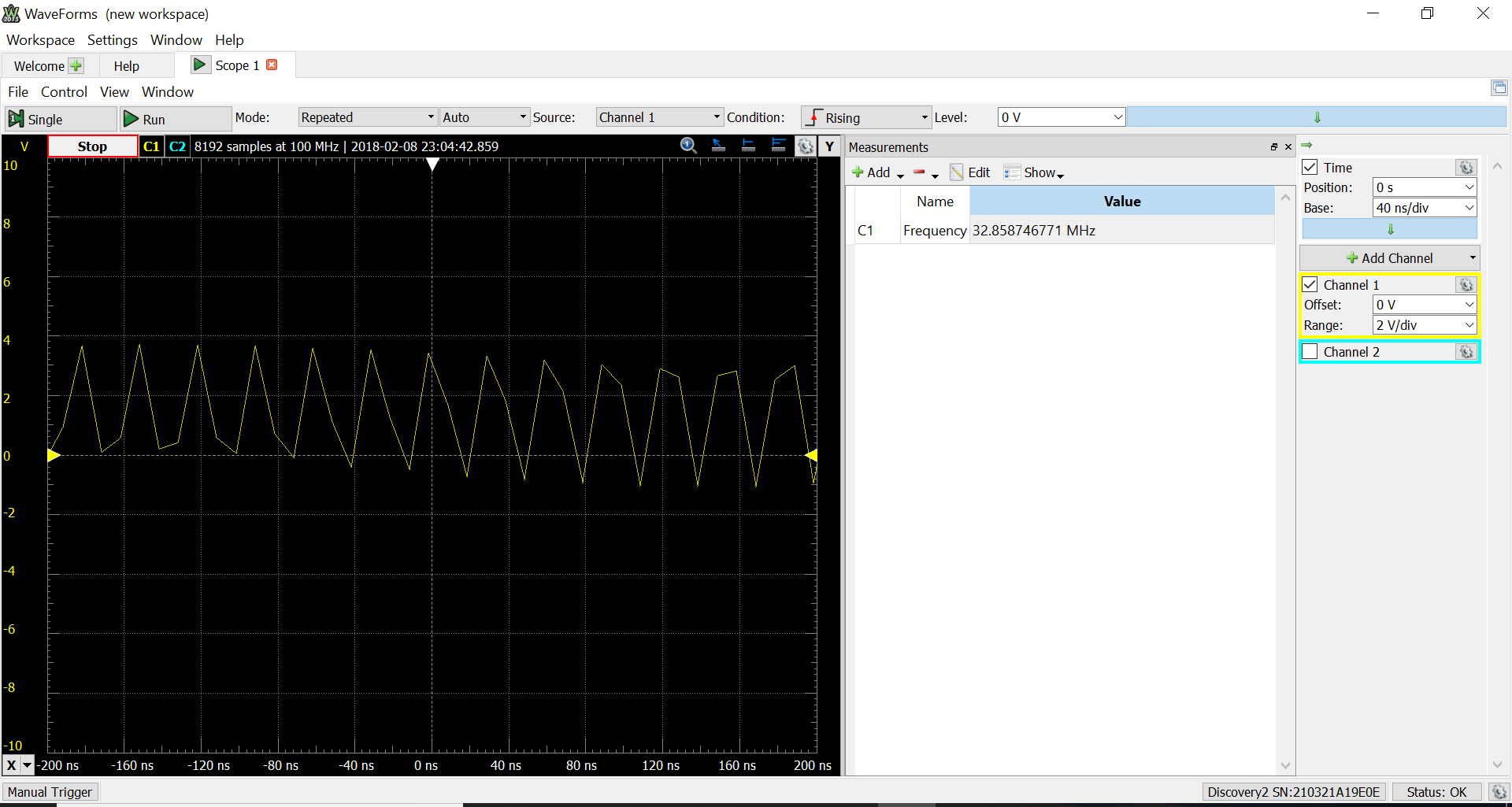
4 MHZ (20 ns/div)







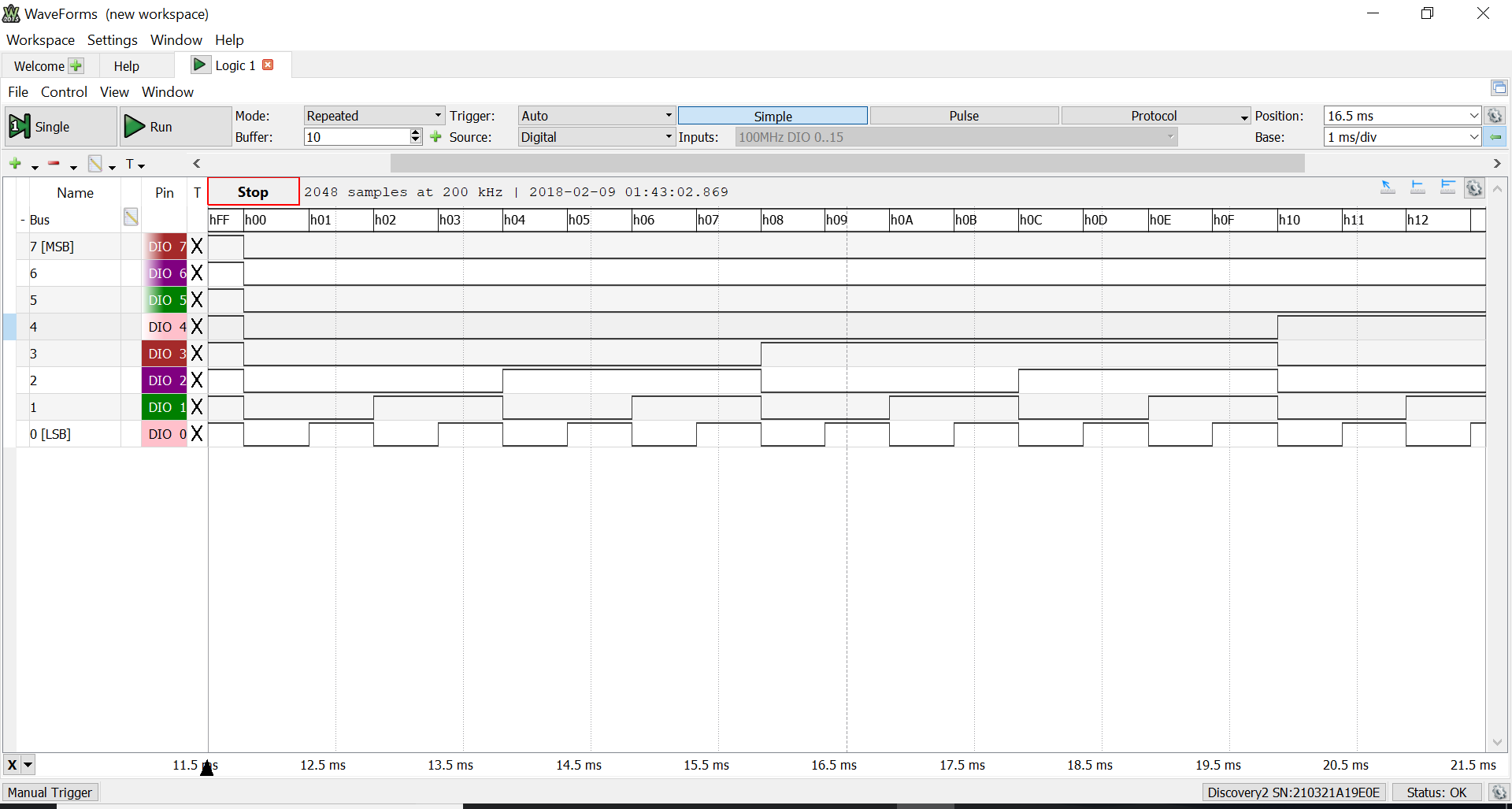
32 MHZ



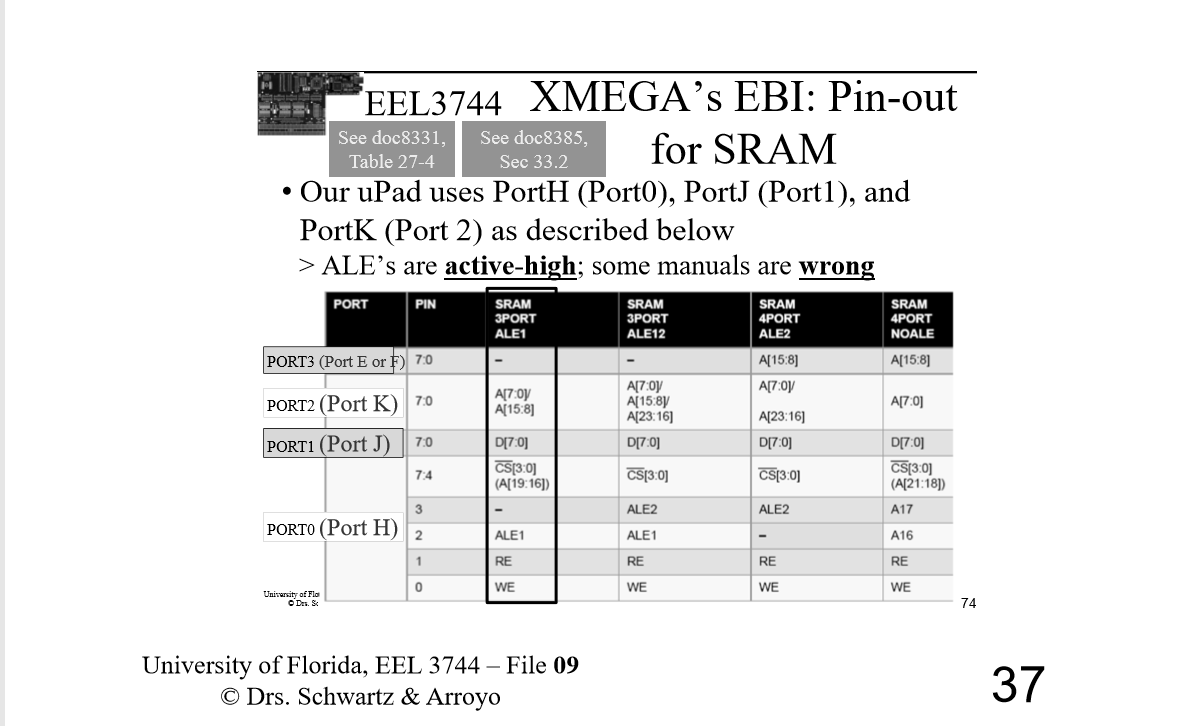
32 MHZ

PART B

CTRLA controls how fast CNT is counted up. Then CNT is compared to PER



**EBI**



Part A Code

.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

.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

sts PORTC\_OUT, r17

rjmp REPEAT

Part C

.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:

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, 0b00001000 ;bit 3= ALE1

sts PORTH\_OUTCLR, r16 ;set false value to ALE

ldi r16, 0xFF

sts PORTJ\_DIRSET, r16

ldi r16, 0xFF

sts PORTK\_DIRSET, r16

ldi ZL, low(EBI\_CS0\_BASEADDR) ;Y pointer point to base address

ldi ZH, high(EBI\_CS0\_BASEADDR)

;ldi r16, byte3(EBI\_CS0\_BASEADDR)

;sts CPU\_RAMPY, r16

ldi r16, byte2(IN\_PORT) ;transfer middle byte of base address to lower byte of base address

st Z+, r16

ldi r16, byte3(IN\_PORT) ;transfer high byte of base address to upper byte of base address

st Z, r16

ldi ZL, byte3(In\_PORT)

out CPU\_RAMPZ, ZL

ldi ZL, low(IN\_PORT)

ldi ZH, byte2(IN\_PORT)

REPEAT:

ldi r16, 0xA5

st Z, r16

ld r17, Z

rjmp REPEAT

Part C pins

0-2 = A2:0

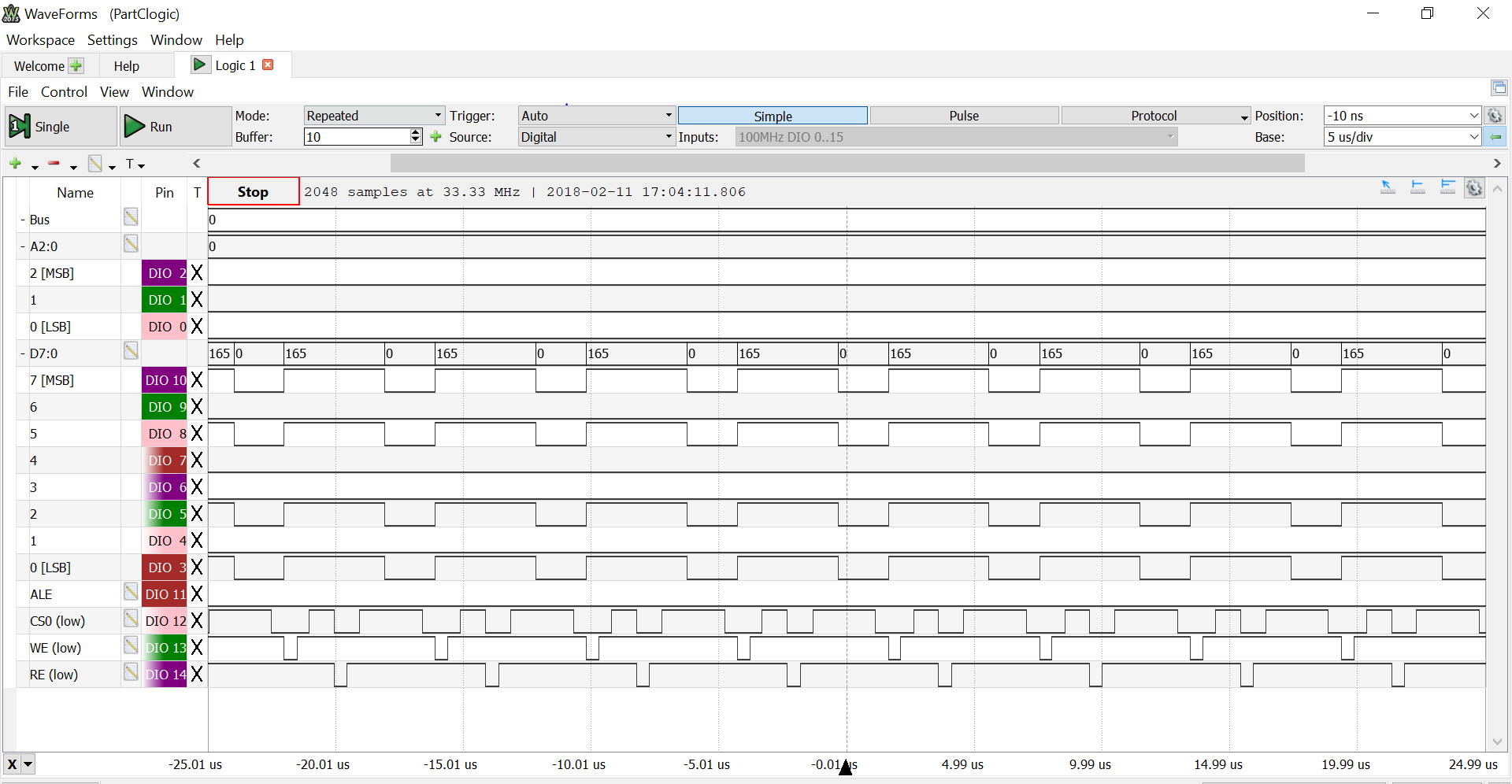
3-10 =D7:0

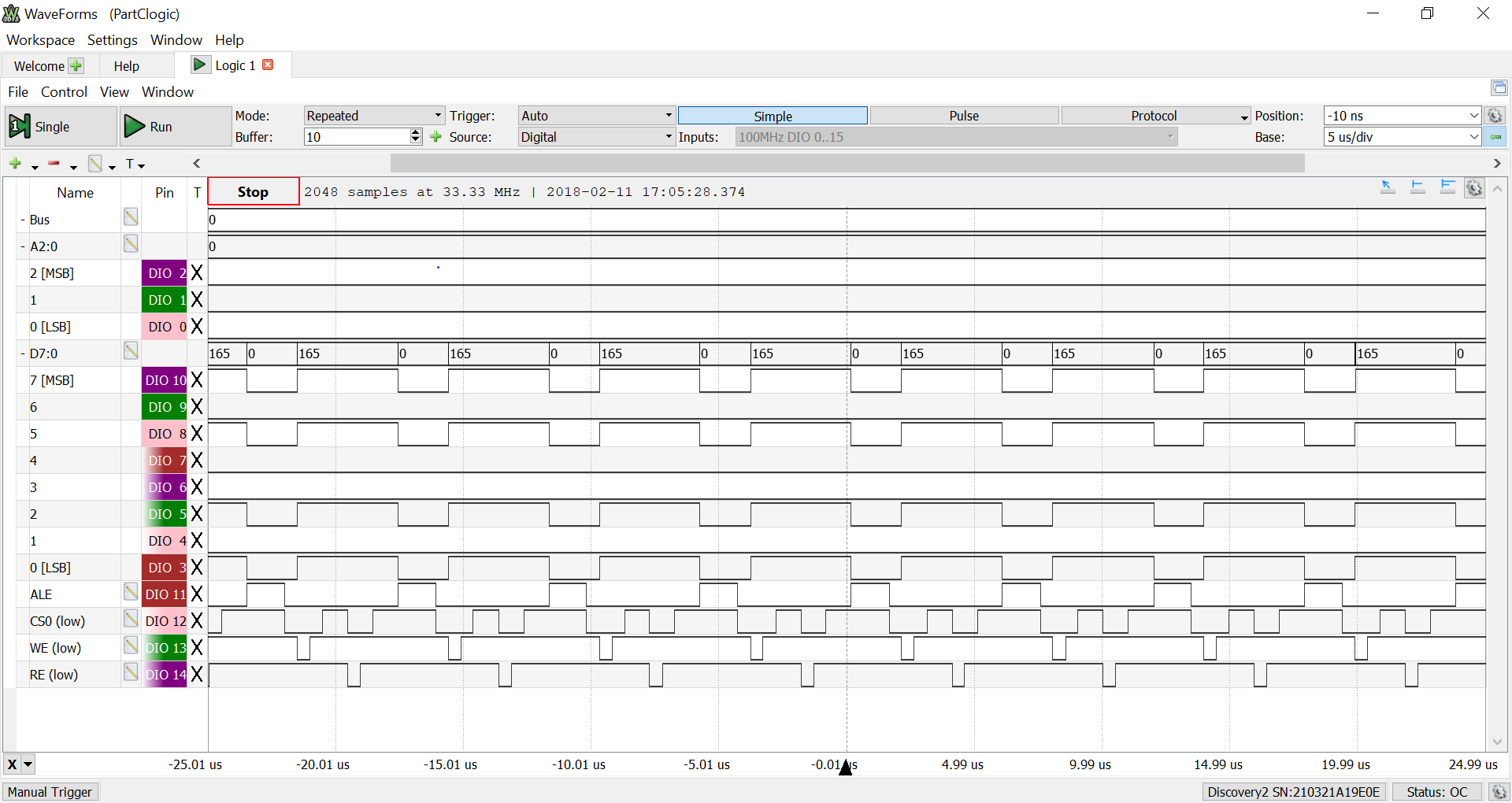
11= ALE

12= CS0

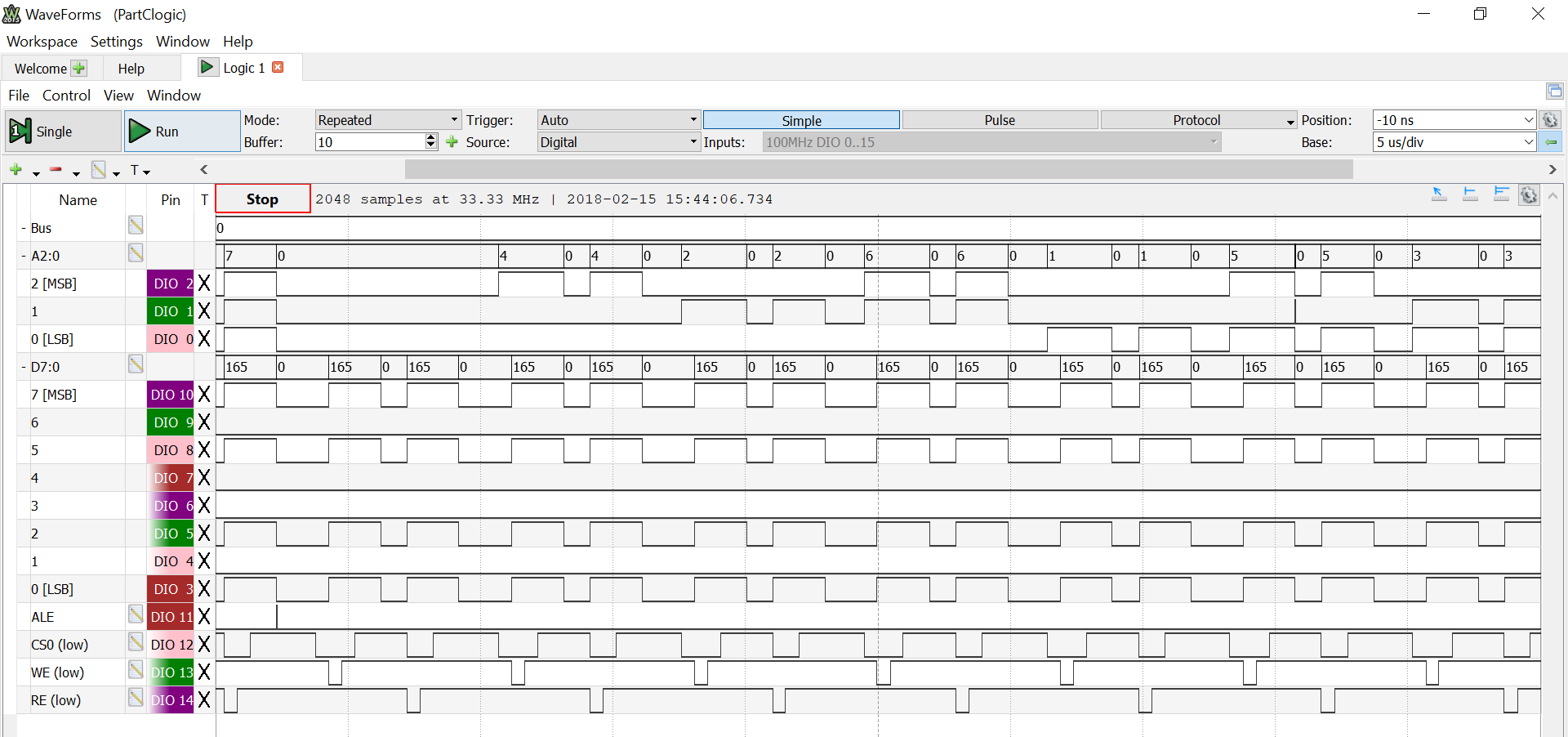
13= WE

14=RE



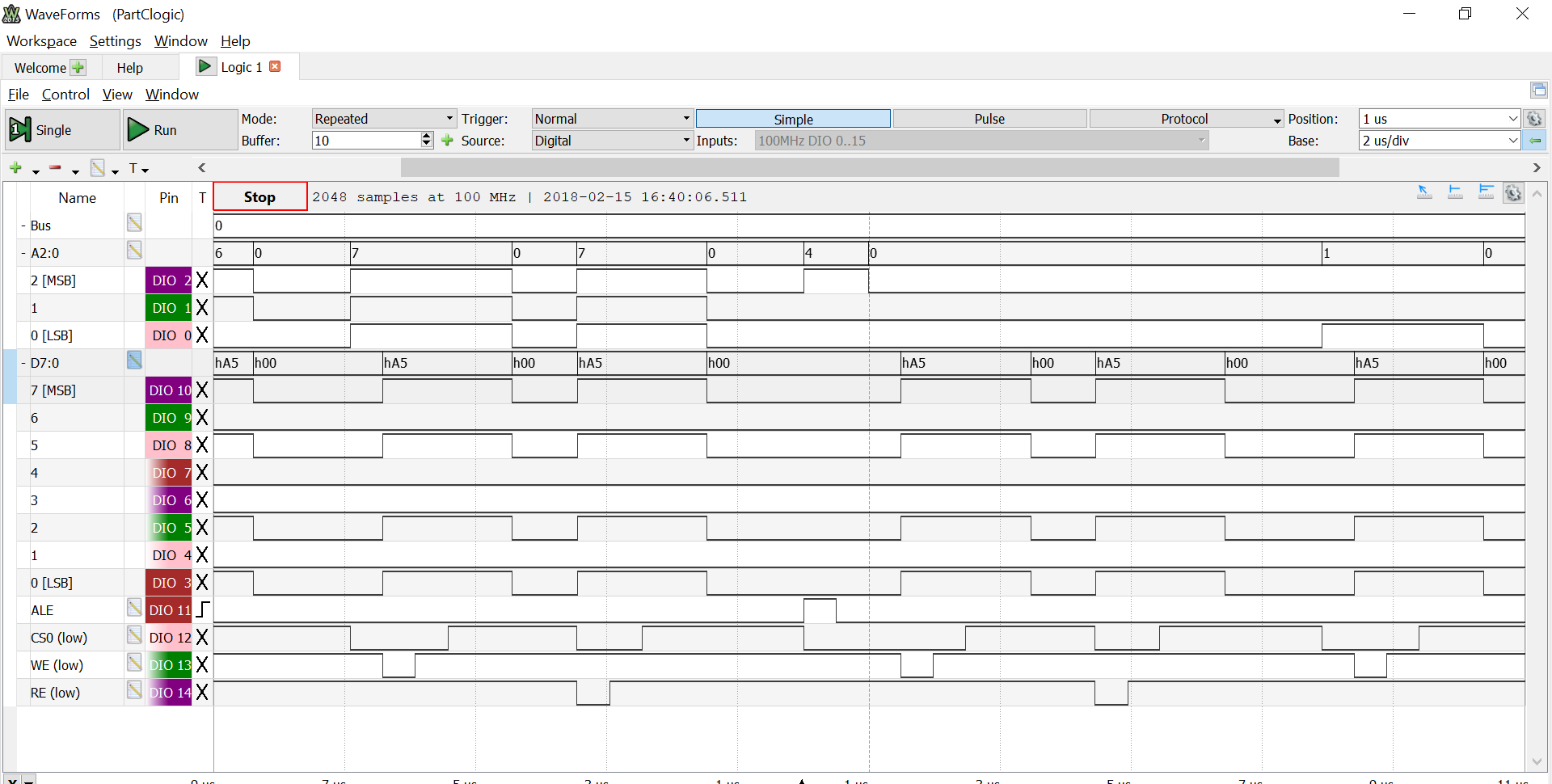


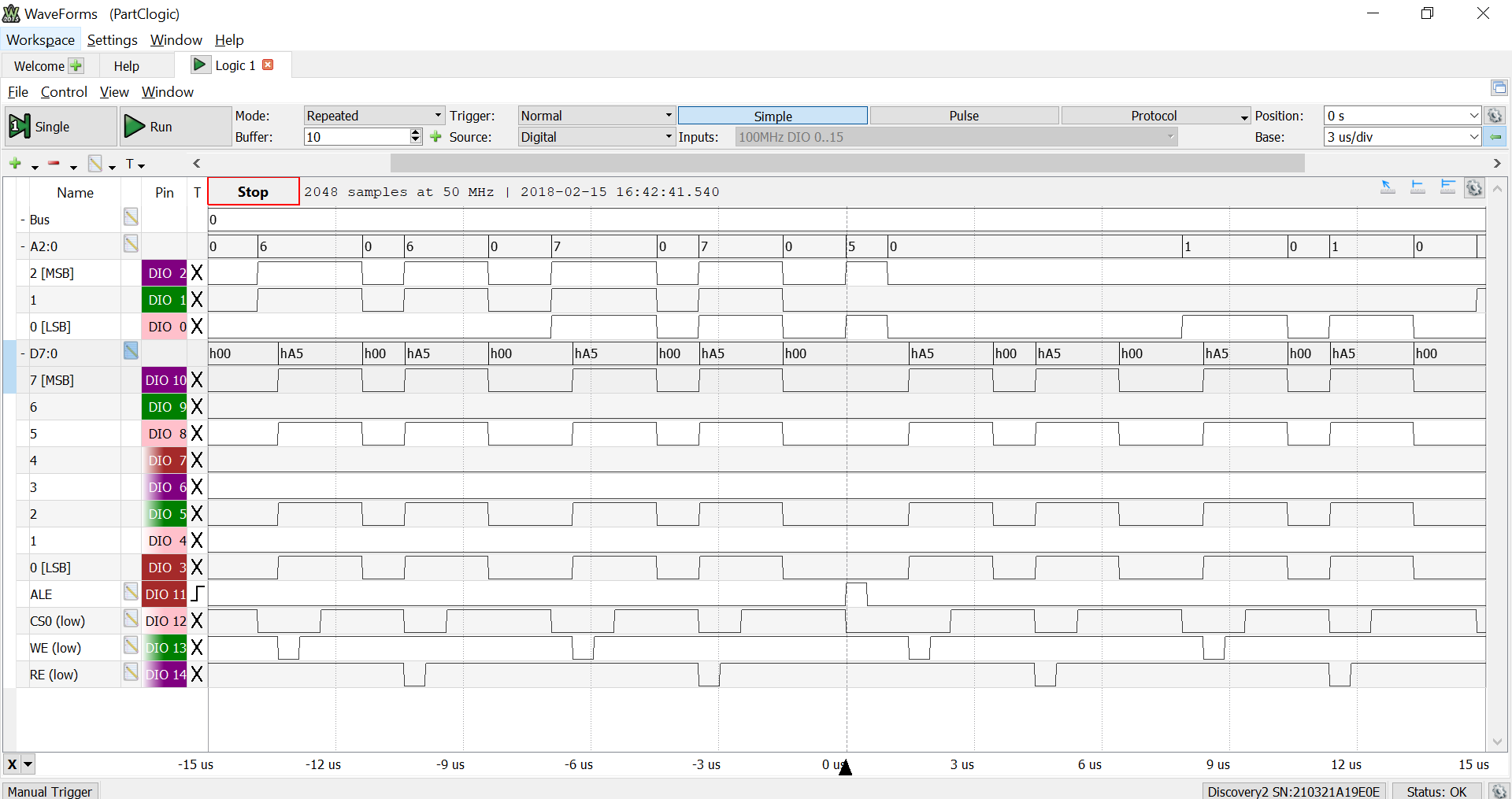
What is this?

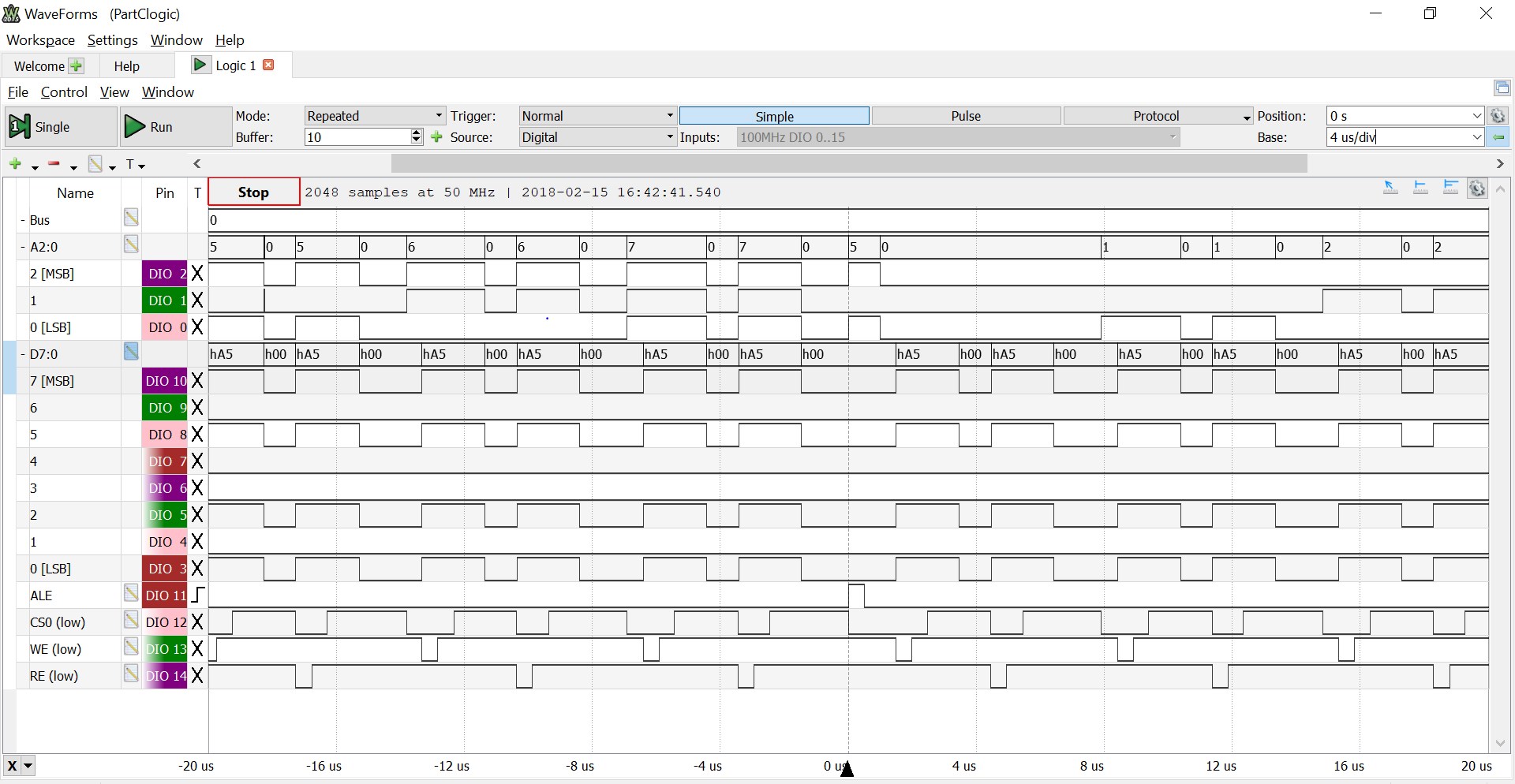


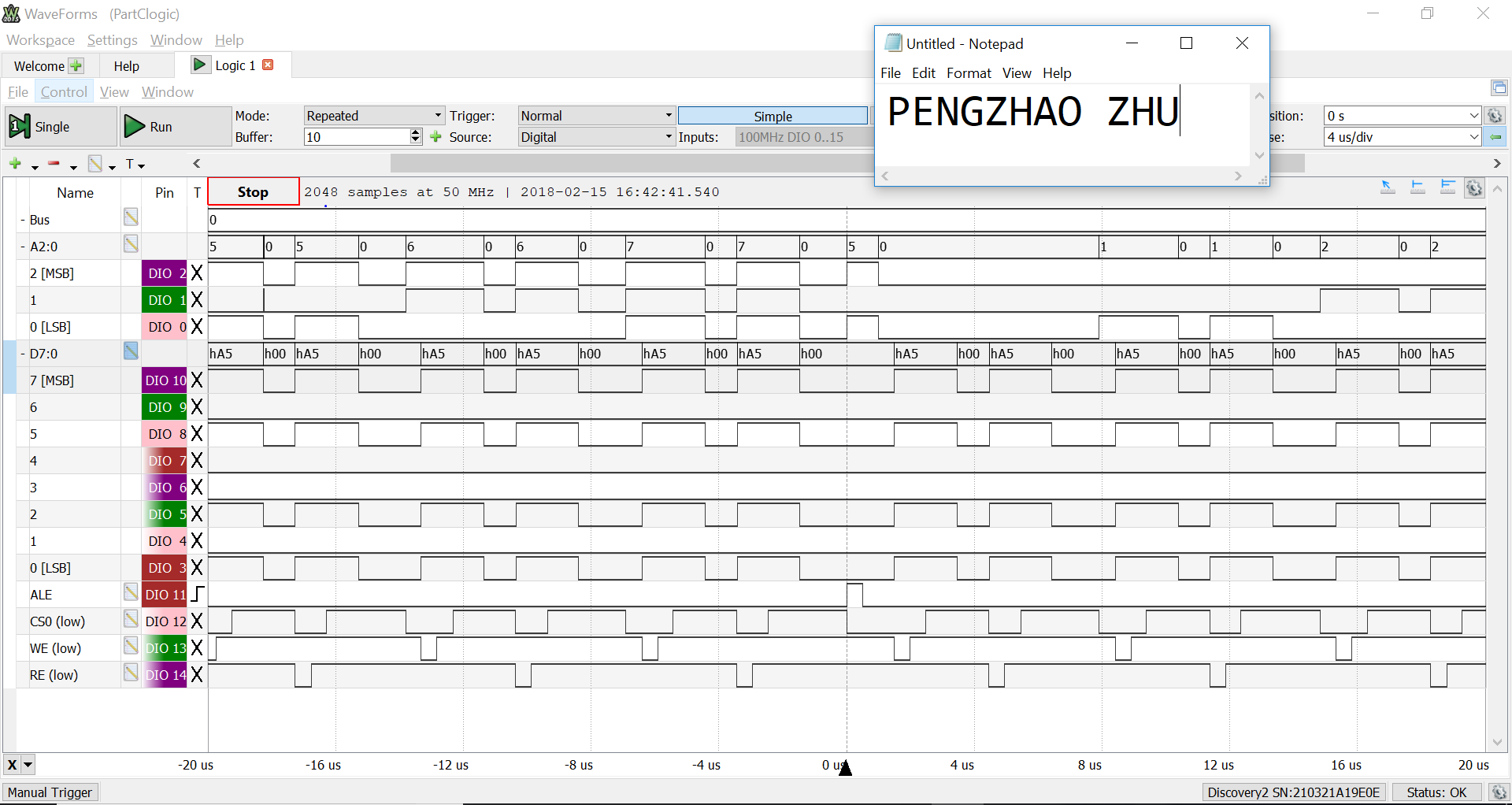
Part C LSA screenshot

Screenshots are all basically the same thing









Change per, calculate time it takes to do one second

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

/\* 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

/\* 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

/\* 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