**REQUIREMENTS NOT MET**

N/A

**PROBLEMS ENCOUNTERED**

N/A

**FUTURE WORK/APPLICATIONS**

This can/will be used in many future applications. The ability to run multiple pieces of code asynchronously is incredible! This allows for more complex and faster programs. It can also simplify how the code is read/written as seen in the third help session. In stead of trying to configure some complicated loops in order to get everything to fit together right, we can just use interrupts to run certain code when requirements are met.

**PRE-LAB EXERCISES**

1. Assuming that no interrupt has been previously configured, devise and describe a generalized series of steps for configuring any interrupt within the ATxmega128A1U, i.e., not just an interrupt within the TC system.

Initialize your interrupt:

configure the interrupt source

set the level of the interrupt

turn on that level of interrupt

Create your ISR  
 preserve status register

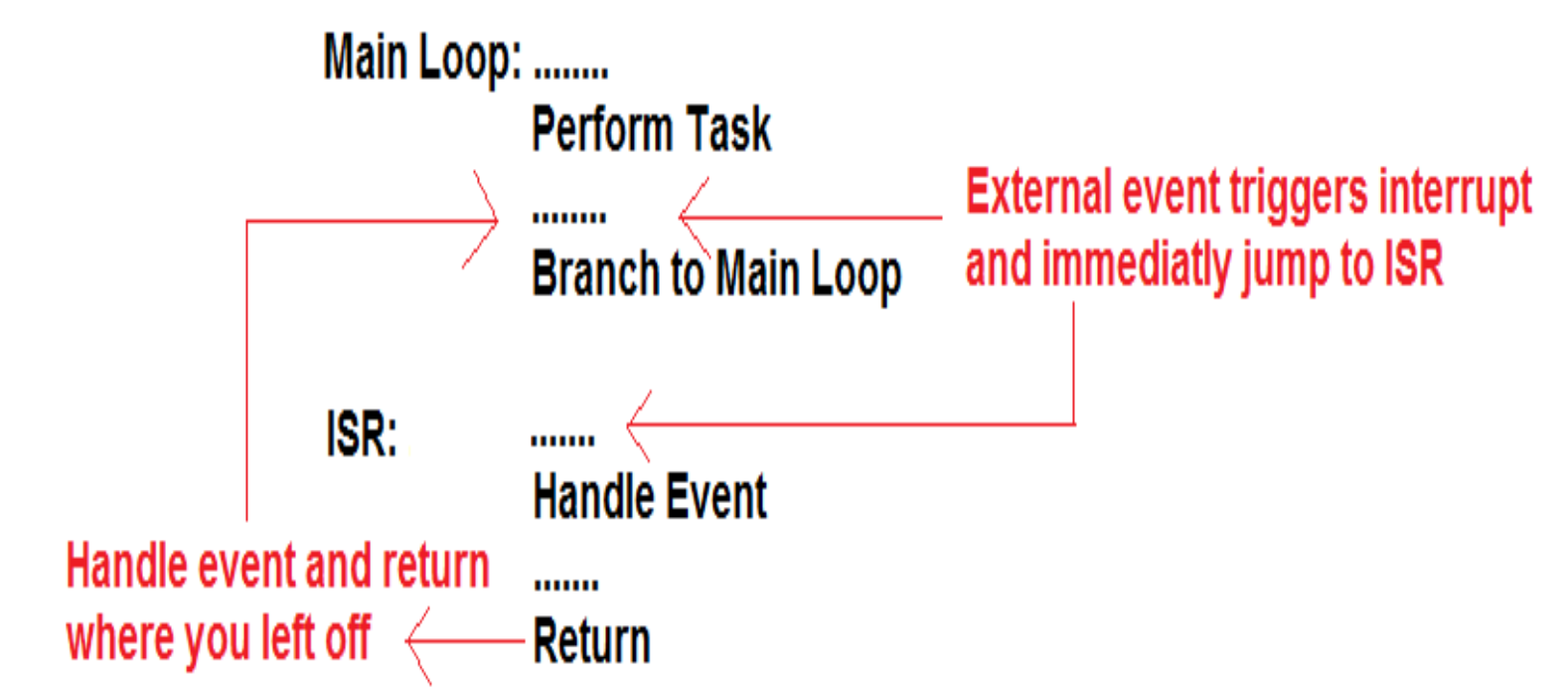
Do whatever logic you need

Recover status register

Put your ISR in memory after the vector that your interrupt uses

**PSEUDOCODE/FLOWCHARTS**

**SECTION 1**



**Figure 1: Proper program flow for an interrupt-driven program**

**SECTION 2**

Thought process for interrupt and debouncing in second half of section 2.

Loop:  
 toggle blue light  
 just jump back to Loop

Interrupt when button is pressed:  
 Start a timer, that’s it

Interrupt when timer ends:  
 check if button is still pressed  
 based on that ^, either increment, or do nothing

**PROGRAM CODE**

**SECTION 1**

MAIN:

; initialize the stack pointer

ldi r16, 0xFF

sts CPU\_SPL, r16

ldi r16, 0x3F

sts CPU\_SPH, r16

; initialize relevant I/O modules (switches and LEDs)

rcall IO\_INIT

; initialize (but do not start) the relevant timer/counter module(s)

rcall TC\_INIT

; initialize Interrupts

rcall INTR\_INIT

TOGGLE\_LOOP:

;Nothing goes here?

;Just keep looping and the timer will count

rjmp TOGGLE\_LOOP

DONE:

rjmp DONE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; I/O Initializations

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IO\_INIT:

; protect relevant registers

push r16

; initialize the relevant I/O

; LEDs on SLB

ldi r16, 0xFF

sts PORTC\_OUT, r16 ; this sets all LEDs

sts PORTC\_DIR, r16 ; sets direction as outputs

; recover relevant registers

pop r16

; return from subroutine

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Timer Counter initializations

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TC\_INIT:

; protect relevant registers

push r16

; initialize the relevant TC modules

clr r16

sts TCC0\_CNT, r16

sts(TCC0\_CNT+1), r16

;set TCC0 period register

;TCC0\_PER = (fclk/prescalar) \* (duration in seconds)

; 2MH/256 0.25

;when you use the reciprocal, you divide by the duration

;assembler can't do decimals

ldi r16, low((F\_CPU/CLK\_PRE)/FRAME\_PER\_RECIP\_A)

sts TCC0\_PER, r16

ldi r16, high((F\_CPU/CLK\_PRE)/FRAME\_PER\_RECIP\_A) ; 2,000,000/256 /4 = 19536

sts (TCC0\_PER + 1), r16

ldi r16, TC\_CLKSEL\_DIV256\_gc ;start the timer

sts TCC0\_CTRLA, r16

; recover relevant registers

pop r16

; return from subroutine

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Interrupt initializations

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INTR\_INIT:

;protect registers

push r16

ldi r16, TC\_OVFINTLVL\_LO\_gc ; low level. System level

sts TCC0\_INTCTRLA, r16

;Turn on low level interrupts

ldi r16, PMIC\_LOLVLEN\_bm

sts PMIC\_CTRL, r16

;enable global interrupt bit

sei

;recover registers

pop r16

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Interrupts

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TOGGLE\_ISR:

; first, always preserve the status register

push r16

lds r16, CPU\_SREG

push r16

; this is used to toggle every light

ldi r16, 0xFF

sts PORTC\_OUTTGL, r16

; clear OVFIF

ldi r16, TC0\_OVFIF\_bm

sts TCC0\_INTFLAGS, r16

; recover the status register

pop r16

sts CPU\_SREG, r16

pop r16

reti

**SECTION 2a**

MAIN:

; initialize the stack pointer

ldi r16, 0xFF

sts CPU\_SPL, r16

ldi r16, 0x3F

sts CPU\_SPH, r16

; initialize relevant I/O modules (switches and LEDs)

rcall IO\_INIT

; initialize relevant interrupts

rcall INTR\_INIT

;default r20 to have all the lights off. Use this to store our count

ldi r20, 0xFF

OVERALL\_LOOP:

;toggle blue here

ldi r16, 0b01000000

sts PORTD\_OUTTGL, r16

rjmp OVERALL\_LOOP

DONE:

rjmp DONE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; I/O Initializations

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IO\_INIT:

; protect relevant registers

push r16

; initialize the relevant I/O

; LEDs on SLB

ldi r16, 0xFF

sts PORTC\_OUT, r16 ; this sets all LEDs

sts PORTC\_DIR, r16 ; sets direction as outputs

; BLUE\_PMW

ldi r16, 0b01000000

sts PORTD\_OUTSET, r16 ; set led to off

sts PORTD\_DIRSET, r16 ;make it an output

; switch on OOTB SLB

ldi r16, 0b00000100 ; tactical switch 1

sts PORTF\_DIRCLR, r16

; recover relevant registers

pop r16

; return from subroutine

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Interrupt initializations

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INTR\_INIT:

;protect registers

push r16

;Select pin2 as the interrupt source

ldi r16, 0b00000100

sts PORTF\_INT0MASK, r16

;Set as low level interrupt

ldi r16, 1

sts PORTF\_INTCTRL, r16

;Only call the interrupt on a falling edge. When the button is pressed

ldi r16, 0b00000010

sts PORTF\_PIN2CTRL, r16

;Turn on low level interrupts

ldi r16, PMIC\_LOLVLEN\_bm

sts PMIC\_CTRL, r16

;enable global interrupt bit

sei

;recover registers

pop r16

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Interrupts

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

COUNT\_ISR:

; first, always preserve the status register

push r16

lds r16, CPU\_SREG

push r16

dec r20 ;LEDs are active low, so this is really like adding 1

sts PORTC\_OUT, r20

; recover the status register

pop r16

sts CPU\_SREG, r16

pop r16

; return from interrupt

reti ;not 'ret'!

**SECTION 2b**

MAIN:

; initialize the stack pointer

ldi r16, 0xFF

sts CPU\_SPL, r16

ldi r16, 0x3F

sts CPU\_SPH, r16

; initialize relevant I/O modules (switches and LEDs)

rcall IO\_INIT

; initialize (but do not start) the relevant timer/counter module(s)

rcall TC\_INIT

; initialize relevant interrupts

rcall INTR\_INIT

;default r20 to have all the lights off. Use this to store our count

ldi r20, 0xFF

OVERALL\_LOOP:

;Just toggle blue light here

ldi r16, 0b01000000

sts PORTD\_OUTTGL, r16

rjmp OVERALL\_LOOP

DONE:

rjmp DONE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; I/O Initializations

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IO\_INIT:

; protect relevant registers

push r16

; initialize the relevant I/O

; LEDs on SLB

ldi r16, 0xFF

sts PORTC\_OUT, r16 ; this sets all LEDs

sts PORTC\_DIR, r16 ; sets direction as outputs

; BLUE\_PMW

ldi r16, 0b01000000

sts PORTD\_OUTSET, r16 ; set led to off

sts PORTD\_DIRSET, r16 ;make it an output

; switch on OOTB SLB

ldi r16, 0b00000100 ; tactical switch 1

sts PORTF\_DIRCLR, r16

; recover relevant registers

pop r16

; return from subroutine

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Timer Counter initializations

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TC\_INIT:

; protect relevant registers

push r16

; initialize the relevant TC modules

clr r16

sts TCC0\_CNT, r16

sts(TCC0\_CNT+1), r16

;set TCC0 period register

;TCC0\_PER = (fclk/prescalar) \* (duration in seconds)

; 2MH/1 0.01

;when you use the reciprocal, you divide by the duration

;assembler can't do decimals

ldi r16, low((F\_CPU/CLK\_PRE)/FRAME\_PER\_RECIP\_A)

sts TCC0\_PER, r16

ldi r16, high((F\_CPU/CLK\_PRE)/FRAME\_PER\_RECIP\_A) ; 2,000,000/1 /100 = 20000

sts (TCC0\_PER + 1), r16

; recover relevant registers

pop r16

; return from subroutine

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Interrupt initializations

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INTR\_INIT:

;protect registers

push r16

;Select pin2 as the interrupt source

ldi r16, 0b00000100

sts PORTF\_INT0MASK, r16

;Set as low level interrupt

ldi r16, 1

sts PORTF\_INTCTRL, r16

;Only call the interrupt on a falling edge. When the button is pressed

ldi r16, 0b00000010

sts PORTF\_PIN2CTRL, r16

;TC interrupt. Low level

ldi r16, TC\_OVFINTLVL\_LO\_gc

sts TCC0\_INTCTRLA, r16

;Turn on low level interrupts

ldi r16, PMIC\_LOLVLEN\_bm

sts PMIC\_CTRL, r16

;enable global interrupt bit

sei

;recover registers

pop r16

ret

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Interrupts

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BUTTON\_ISR:

;button has been pressed

; first, always preserve the status register

push r16

lds r16, CPU\_SREG

push r16

;Debounce switch here. Just start TC

;do the logic of adding and showing LEDs in the TC interrupt

ldi r16, TC\_CLKSEL\_DIV1\_gc

sts TCC0\_CTRLA, r16

; recover the status register

pop r16

sts CPU\_SREG, r16

pop r16

; return from interrupt

reti ;not 'ret'!

TC\_ISR:

;if here, timer has overflowed

; first, always preserve the status register

push r16

lds r16, CPU\_SREG

push r16

lds r16, PORTF\_IN

sbrs r16, 2 ;skip next instruction if bit = 1 meaning depressed

;if it isn't pressed, debouncing isn't done, skip the decriment

dec r20 ;LEDs are active low, so this is really like adding 1

sts PORTC\_OUT, r20

; clear OVFIF

ldi r16, TC0\_OVFIF\_bm

sts TCC0\_INTFLAGS, r16

;Turn off TC

ldi r16, TC\_CLKSEL\_OFF\_gc ;Turn it off

sts TCC0\_CTRLA, r16

; recover the status register

pop r16

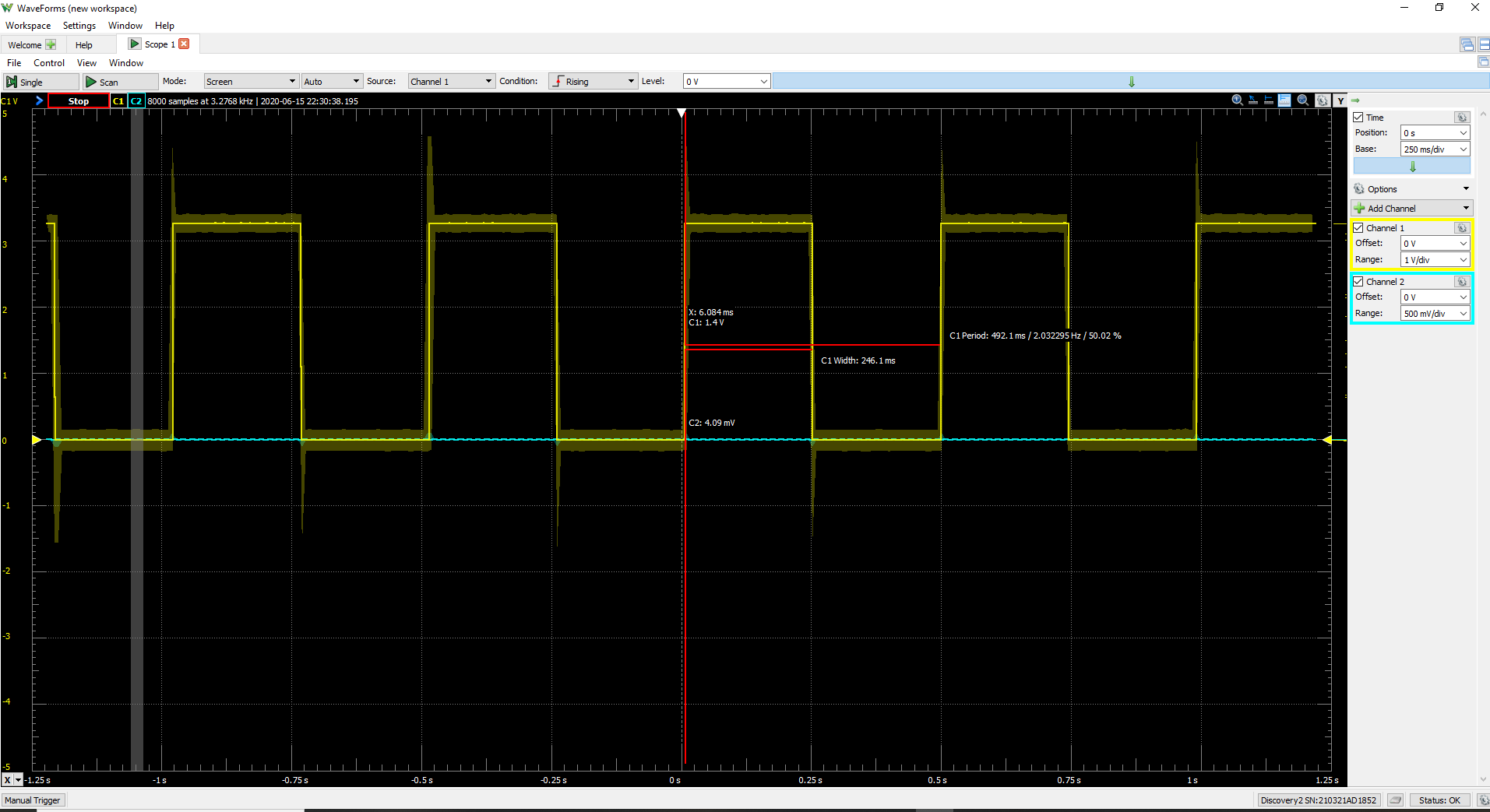
sts CPU\_SREG, r16

pop r16

reti

**APPENDIX**

**SECTION 1**

****

**Figure 2: Output toggling every 250ms from interrupt**

**Code above MAIN**

.include "ATxmega128A1Udef.inc"

.equ F\_CPU = 2000000

.equ CLK\_PRE = 256

.equ FRAME\_PER\_A = 1/4 ; 250ms

.equ FRAME\_PER\_RECIP\_A = 4

.ORG 0x0000

rjmp MAIN

.ORG TCC0\_OVF\_vect

rjmp TOGGLE\_ISR

.ORG 0x0100

MAIN:

**SECTION 2a**

**Code above MAIN**

.include "ATxmega128A1Udef.inc"

.ORG 0x0000

rjmp MAIN

.ORG PORTF\_INT0\_vect

rjmp COUNT\_ISR

.ORG 0x0100

MAIN:

**SECTION 2b**

**Code above MAIN**

.include "ATxmega128A1Udef.inc"

.equ F\_CPU = 2000000

.equ CLK\_PRE = 1

.equ FRAME\_PER\_A = 1/100 ; 10ms

.equ FRAME\_PER\_RECIP\_A = 100

.ORG 0x0000

rjmp MAIN

.ORG PORTF\_INT0\_vect

rjmp BUTTON\_ISR

.ORG TCC0\_OVF\_vect

rjmp TC\_ISR

.ORG 0x0100

MAIN: