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

i. 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

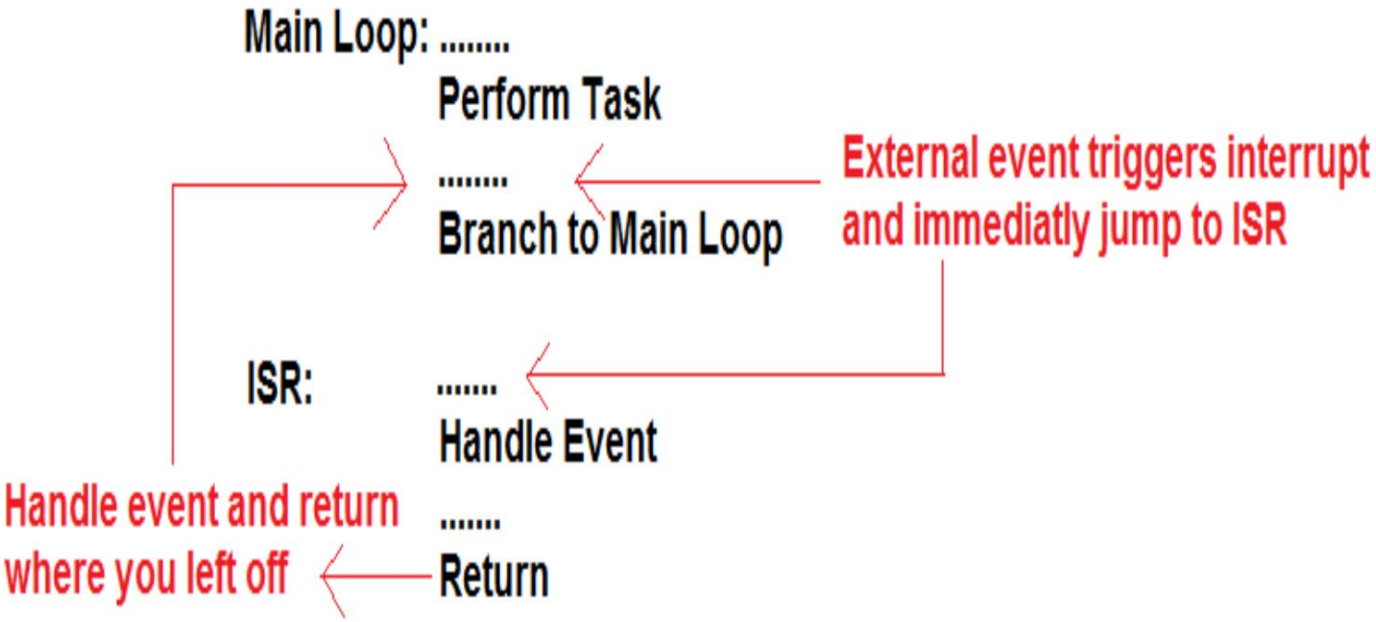

Figure 1: Proper program flow for an interrupt-driven program

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

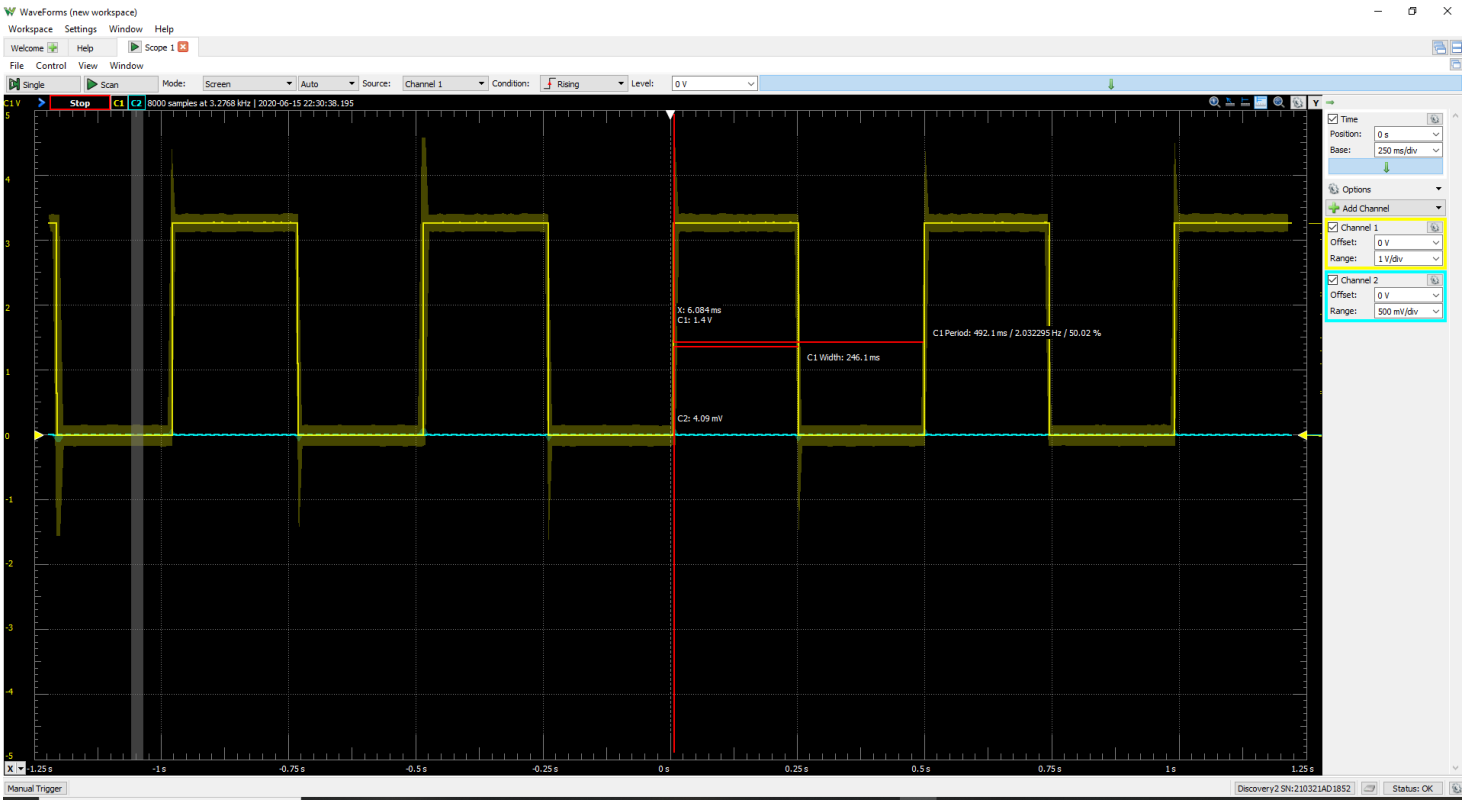

Figure 2: Output toggling every 250ms from interrupt

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:
```