REQUIREMENTS NOT MET

N/A

PROBLEMS ENCOUNTERED

I didn't really come across any problems. I just took some time to fully understand some concepts and actually code them well. Also, I feel like I took around 50 screenshots because my monitor is huge, so the screenshots look really small on the document. So, in order to get them to match the rule/policies I had to keep retaking them.

FUTURE WORK/APPLICATIONS

Being able to work with I/O ports is super helpful for anything. This gives your application a simple interface that is easy to debug and visually see what is happening with data. Also, working with timer counters was interesting and I can easily see how we will use them throughout the rest of this class. Being able to manipulate and use our clock in different ways just makes our programs quicker, easier, and more understandable.

PRE-LAB EXERCISES

i. Which configuration register allows the utilization of an I/O port pin configured as an input? Which configuration registers allow the utilization of an I/O port pin configured as an output?

IN for input OUT for output

ii. What is the purpose of the SET/CLR/TGL variants of the DIR and OUT registers?

SET/CLR/TGL modify individual pins/bits corresponding to the DIR register.

iii. Which I/O ports are utilized for the DIP switches and LEDs on the OOTB Switch & LED Backpack?

PortA, PortC, and PortF

iv. Are the LEDs on the OOTB Switch & LED Backpack active-high, or active-low? Draw a schematic diagram for a single LED circuit with the same activation level used on the backpack, as well as one with the opposite activation level. Also, draw a schematic diagram for a single-pole, single-throw (SPST) switch circuit, using the same pull-up or pull-down resistor condition utilized on the backpack, as well as another switch circuit using the opposite configuration.

The LEDs are active-low.
(put drawings here)

v. Would it be possible to interface the OOTB µPAD with an external input device consisting of 24 inputs? If so, describe how many I/O ports would be necessary. If not, explain why.

Yes, you could have 24 inputs. You would need 3 I/O ports.

vi. Assuming a system clock frequency of 2 MHz, a prescaler value of 256, and a desired period of 50 ms, calculate a theoretically-corresponding timer/counter period value two separate times: once using a form of dimensional analysis, providing explanation(s) when appropriate, and another time using the general formula provided within The Most Common Use Case for Timer/Counters.

Dimensional analysis: $\frac{2*10^6\ SCtick}{1\ sec} * \frac{1\ TCtick}{256\ SCtick} * \frac{0.5\ sec}{50\ ms} = \frac{3906.25\ TCtick}{50\ ms}$

General formula: $PRE = 0.5\left(\frac{2*10^6}{256}\right) = 3906.25$

vii. Assuming a system clock frequency of 2 MHz, is a period of two seconds achievable when using a 16-bit timer/counter prescaler value of one? If not, determine if there exists any prescaler value that allows for this period under the assumed circumstances, and if there does, list such a value.

No, you cannot achieve this with a prescaler value of one.

$$2^{16} - 1 = 65{,}535$$

$$2\ sec * \frac{2 * 10^6\ SCtick}{1\ sec} * \frac{1\ TCtick}{1\ SCtick} = 4{,}000{,}000$$

$$4{,}000{,}000 > 65{,}535$$

We can only count to 65,546 with a prescaler value of one, but we would need to count to 4,000,000. If we divide, we can find the least prescaler value that would work for us.

$$\frac{4{,}000{,}000}{65{,}535} = 61.036$$

We need to round up to the nearest whole number, so we round the prescalar up to 62.

$$2\ sec * \frac{2 * 10^6\ SCtick}{1\ sec} * \frac{1\ TCtick}{62\ SCtick} = 64516.12903$$

$$64{,}516.12903 < 65{,}535$$

So, you need at least a prescalar value of 62 to reach a period of 2 seconds. The closest prescalar clock we could use is 64.

viii. What is the maximum time value (to the nearest millisecond) representable by a timer/counter, if the relevant system clock frequency is 2 MHz? What about for a system clock frequency of 32.768 kHz?

$$x\ sec = 65{,}536 * \frac{1}{2 * 10^6}$$

$$x\ sec = 0.032768\ sec$$

$$x\ ms = 3ms$$

$$x\ sec = 65{,}536 * \frac{1}{32.768 * 10^3}$$

$$x\ sec = 2\ sec$$

$$x\ ms = 2000ms$$

ix. 3.2 with a prescalar of 2

```
/*
 * lab2_3.asm
 *
 *  Created: 6/7/2020 4:34:29 PM
 *   Author: Koby Miller
 */


.include "ATxmega128A1Udef.inc"

.equ F_CPU = 2000000
.equ CLK_PRE_TWO = 2
.equ FRAME_PER_A = 1/20 ;50ms
.equ FRAME_PER_RECIP_A = 20

.ORG 0x0000
	rjmp MAIN

.ORG 0x0100

MAIN:

; initialize stack, start at 0x3FFF
ldi r16, 0xFF
sts CPU_SPL, r16
ldi r16, 0x3F
sts CPU_SPH, r16


; initialize ports for switches/LEDs
ldi r16, 0xFF
sts PORTC_OUT, r16		; this sets all LEDs
sts PORTC_DIR, r16		; sets direction as outputs

ldi r16, 0x00
sts PORTA_DIR, r16		; sets direction of switches as inputs

rcall TCC0_INIT

LOOP:

lds r16, TCC0_INTFLAGS
sbrs r16, 0			; if 0, skip the next instruction
rjmp LOOP
ldi r16, 0xFF			; this is used to toggle every light
sts PORTC_OUTTGL, r16

; clear OVFIF
ldi r16, TC0_OVFIF_bm
sts TCC0_INTFLAGS, r16

rjmp LOOP



DONE:
	rjmp DONE


TCC0_INIT:
```

```
push r16

clr r16
sts TCC0_CNT, r16
sts(TCC0_CNT+1), r16

;set TCC0 period register
;TCC0_PER = (fclk/prescalar) * (duration in seconds)
;                     2MH/2              0.05
;when you use the reciprocal, you divide by the duration
;assembler can't do decimals

ldi r16, low((F_CPU/CLK_PRE_TWO)/FRAME_PER_RECIP_A)
sts TCC0_PER, r16                 ; 2,000,000/2 /20      = 1,000,000/20 = 50000

ldi r16, high((F_CPU/CLK_PRE_TWO)/FRAME_PER_RECIP_A)
sts (TCC0_PER + 1), r16

;starts timer counter with prescaler of 2
ldi r16, TC_CLKSEL_DIV2_gc
sts TCC0_CTRLA, r16

pop r16

ret
```

x. Assembly program to keep track of elapsing minutes. Minutes saved in r18

```
.include "ATxmega128A1Udef.inc"

.equ F_CPU = 2000000
.equ CLK_PRE = 64
.equ FRAME_PER_A = 1/1
.equ FRAME_PER_RECIP_A = 1

.ORG 0x0000
	rjmp MAIN

.ORG 0x0100

MAIN:

; initialize stack, start at 0x3FFF
ldi r16, 0xFF
sts CPU_SPL, r16
ldi r16, 0x3F
sts CPU_SPH, r16


; initialize ports for switches/LEDs
ldi r16, 0xFF
sts PORTC_OUT, r16		; this sets all LEDs
sts PORTC_DIR, r16		; sets direction as outputs

ldi r16, 0x00
sts PORTA_DIR, r16		; sets direction of switches as inputs

rcall TCC0_INIT

ldi r17, 60			; We will count the seconds here (count down)
ldi r18, 0			; We will count the minutes here

LOOP:

lds r16, TCC0_INTFLAGS
sbrs r16, 0			; if 0, skip the next instruction
rjmp LOOP
ldi r16, 0xFF			; this is used to toggle every light
sts PORTC_OUTTGL, r16

; clear OVFIF
ldi r16, TC0_OVFIF_bm
sts TCC0_INTFLAGS, r16

dec r17
sbrs r17, 0			; if 0, skip the next instruction
rjmp LOOP
ldi r17, 60			; if here, reset second count and increase minute count
inc r18

rjmp LOOP


DONE:
	rjmp DONE
```

```
TCC0_INIT:
	push r16

	clr r16
	sts TCC0_CNT, r16
	sts(TCC0_CNT+1), r16

	;set TCC0 period register
	;TCC0_PER = (fclk/prescalar) * (duration in seconds)
	;                     2MH/64              1
	;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                       ; 2,000,000/64  /1  = 31250

	ldi r16, high((F_CPU/CLK_PRE)/FRAME_PER_RECIP_A)
	sts (TCC0_PER + 1), r16

	;starts timer counter with prescaler of 64
	ldi r16, TC_CLKSEL_DIV64_gc
	sts TCC0_CTRLA, r16

	pop r16

	ret
```

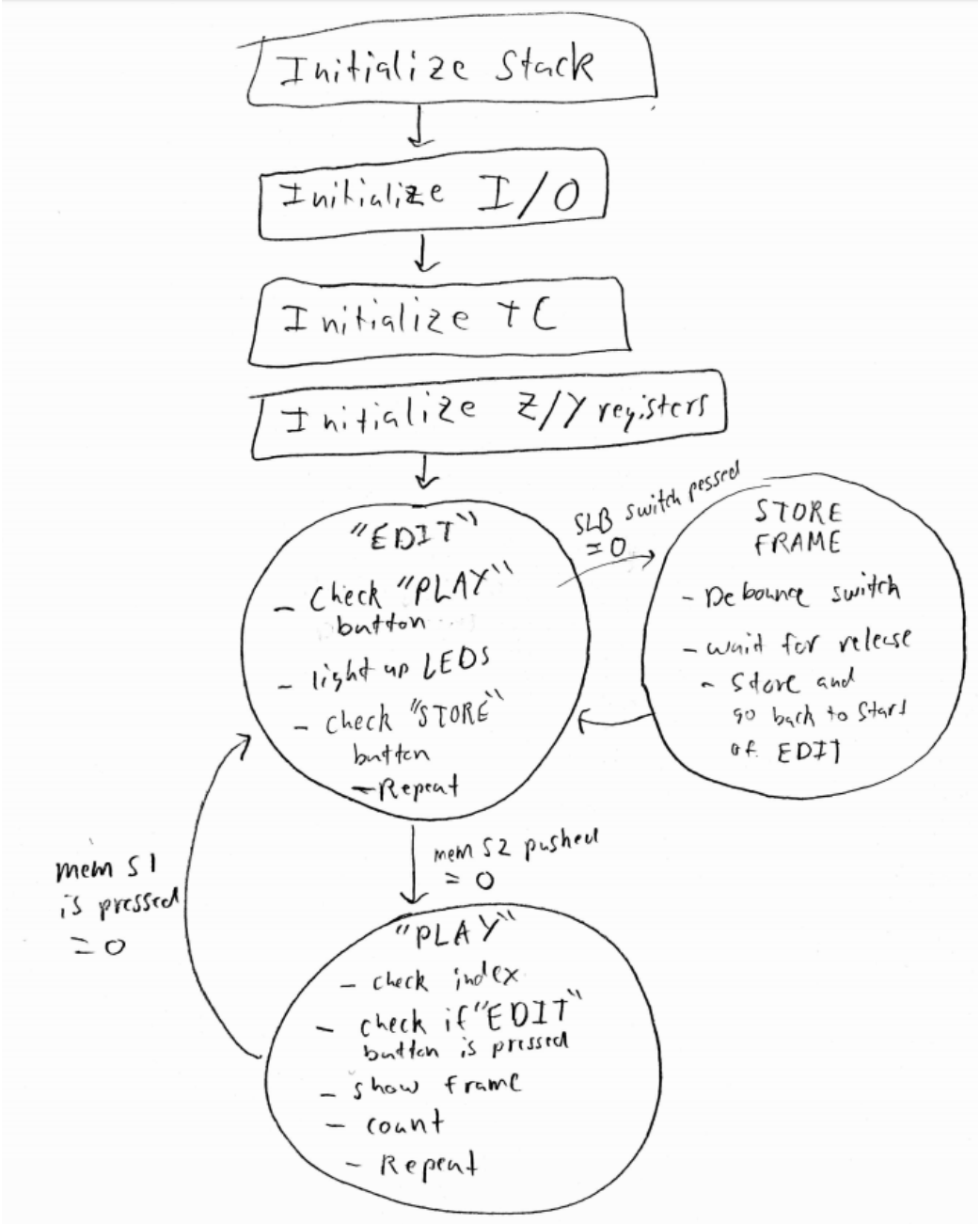
xi. It isn't necessary to debounce those two switches because after the switch is initially pressed, there is no other action that can be done because of that switch. For example, when you press the switch to go into "PLAY" mode, it doesn't matter how long or how many times you press the "PLAY" button. You need to push a different switch in order to change modes. So, the bouncing of the switch would not alter the logic after the first press. Unlike the logic to save a frame. After a frame is saved, the logic can stay in the same "area" and end up saving extra frames.

xii. A scenario in which lab2_4.asm could experience unintended behavior due to tactile switch bouncing would be if the program used the "PLAY" button to also pause the LEDs. So say you start the animation with the "PLAY" button, instead of only being able to go back to "EDIT" mode, you could program it to "PAUSE" if you hit the same switch as the "PLAY" button. This would require you to debounce that switch.
If we are not altering the program to find unintended behavior, than the only thing I can think of is if you tried to go back and forth between "PLAY" and "EDIT" really fast, but nothing would really happen.

PSEUDOCODE/FLOWCHARTS

Part 1 – 3 mainly had the same sort of simple flow:
Initialize stack -> initialize I/O ports -> (Initialize counter for part 3) -> toggle/turn on LED -> count for a certain amount of time -> repeat.

Part 4 Flowchart:



PROGRAM CODE

Starting from MAIN

```
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 the X and Y indices to point to the beginning of the
; animation table. (Although one pointer could be used to both
; store frames and playback the current animation, it is simpler
; to utilize a separate index for each of these operations.)
; Note: recognize that the animation table is in DATA memory

;Y init. Using to store
ldi YL, BYTE3(ANIMATION)
out CPU_RAMPY, YL
ldi YH, BYTE2(ANIMATION)
ldi YL, BYTE1(ANIMATION)

;Z init. Using to playback
ldi ZL, BYTE3(ANIMATION)
out CPU_RAMPZ, ZL
ldi ZH, BYTE2(ANIMATION)
ldi ZL, BYTE1(ANIMATION)



; begin main program loop

; "EDIT" mode
EDIT:
; Check if it is intended that "PLAY" mode be started, i.e.,
; determine if the relevant switch has been pressed.

;For some reason, bit 1 is for switch S2 even though skematic has it as bit 0
lds r16, PORTE_IN    ;read input of memory base tactical switch S2
sbrs r16, 1                ;skip next instruction if bit = 1 meaning not pressed (button not pressed =
pulled high)
rjmp PLAY
; If it is determined that relevant switch was pressed,
; go to "PLAY" mode.


; Otherwise, if the "PLAY" mode switch was not pressed,
; update display LEDs with the voltage values from relevant DIP switches
; and check if it is intended that a frame be stored in the animation
; (determine if this relevant switch has been pressed).

lds r16, PORTA_IN
sts PORTC_OUT, r16
```

```
; If the "STORE_FRAME" switch was not pressed,
; branch back to "EDIT".

lds r16, PORTF_IN
sbrc r16, 2                ;skip next instruction if bit = 0 meaning it is pressed
rjmp EDIT


; Otherwise, if it was determined that relevant switch was pressed,
; perform debouncing process, e.g., start relevant timer/counter
; and wait for it to overflow. (Write to CTRLA and loop until
; the OVFIF flag within INTFLAGS is set.)

;starts timer counter with prescaler of 256
ldi r16, TC_CLKSEL_DIV256_gc
sts TCC0_CTRLA, r16
;I measured around a 10ms debounce rate for the switch. To be safe and to reuse code
;I can use the same TC that counts the frames and just count 50ms

LOOP:
lds r16, TCC0_INTFLAGS
sbrs r16, 0                        ; if 1, skip the next instruction
rjmp LOOP


; After relevant timer/counter has overflowed (i.e., after
; the relevant debounce period), disable this timer/counter,
; clear the relevant timer/counter OVFIF flag,
; and then read switch value again to verify that it was
; actually pressed. If so, perform intended functionality, and
; otherwise, do not; however, in both cases, wait for switch to
; be released before jumping back to "EDIT".

ldi r16, TC_CLKSEL_OFF_gc          ;Turn it off
sts TCC0_CTRLA, r16

ldi r16, TC0_OVFIF_bm              ;clear OVFIF
sts TCC0_INTFLAGS, r16

lds r16, PORTF_IN                  ;read press again
sbrc r16, 2                        ;skip next instruction if bit = 0 meaning pressed
rjmp EDIT


; Wait for the "STORE FRAME" switch to be released
; before jumping to "EDIT".
STORE_FRAME_SWITCH_RELEASE_WAIT_LOOP:
lds r16, PORTF_IN                  ;read until not pressed
sbrs r16, 2                        ;skip next instruction if bit = 1, not pressed
rjmp STORE_FRAME_SWITCH_RELEASE_WAIT_LOOP

;If here, now store
lds r16, PORTA_IN
st Y+, r16

rjmp EDIT
```

```
; "PLAY" mode
PLAY:

; Reload the relevant index to the first memory location
; within the animation table to play animation from first frame.

ldi ZL, BYTE3(ANIMATION)
out CPU_RAMPZ, ZL
ldi ZH, BYTE2(ANIMATION)
ldi ZL, BYTE1(ANIMATION)

PLAY_LOOP:

; Check if it is intended that "EDIT" mode be started
; i.e., check if the relevant switch has been pressed.

;For some reason, bit 0 is for switch S2 even though skematic has it as bit 1
lds r16, PORTE_IN    ;read input of memory base tactical switch S1
sbrs r16, 0          ;skip next instruction if bit = 1 meaning not pressed
rjmp EDIT
; If it is determined that relevant switch was pressed,
; go to "EDIT" mode.


; Otherwise, if the "EDIT" mode switch was not pressed,
; determine if index used to load frames has the same
; address as the index used to store frames, i.e., if the end
; of the animation has been reached during playback.
; (Placing this check here will allow animations of all sizes,
; including zero, to playback properly.)
; To efficiently determine if these index values are equal,
; a combination of the "CP" and "CPC" instructions is recommended.

cp ZL,YL
brne DIFFERENT_ADDRESS
cp ZH,YH
brne DIFFERENT_ADDRESS
;cp CPU_RAMPZ, CPU_RAMPY
;brne DIFFERENT_ADDRESS

;If here, they are at the same address
rjmp PLAY
; If index values are equal, branch back to "PLAY" to
; restart the animation.


DIFFERENT_ADDRESS:
; Otherwise, load animation frame from table,
; display this "frame" on the relevant LEDs,
; start relevant timer/counter,
; wait until this timer/counter overflows (to more or less
; achieve the "frame rate"), and then after the overflow,
; stop the timer/counter,
; clear the relevant OVFIF flag,
; and then jump back to "PLAY_LOOP".

ld r16, Z+
sts PORTC_OUT, r16

;starts timer counter with prescaler of 256
ldi r16, TC_CLKSEL_DIV256_gc
```

```
sts TCC0_CTRLA, r16

FRAME_LOOP:
lds r16, TCC0_INTFLAGS
sbrs r16, 0                ; if 0, skip the next instruction
rjmp FRAME_LOOP

; clear OVFIF
ldi r16, TC0_OVFIF_bm
sts TCC0_INTFLAGS, r16

rjmp PLAY_LOOP


; end of program (never reached)
DONE:
	rjmp DONE
;*****************************END OF MAIN PROGRAM ****************************

;*******************************SUBROUTINES**********************************

;*****************************************************************************
; Name: IO_INIT
; Purpose: To initialize the relevant input/output modules, as pertains to the
;               application.
; Input(s): OOTB SLB DIP switches/tactical switch 1
; Output: LEDs
;*****************************************************************************
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

	; DIP switches on SLB
	ldi r16, 0x00
	sts PORTA_DIR, r16          ; sets switch direction as inputs

	; switch on OOTB SLB
	ldi r16, 0b00000100         ; tactical switch 1
	sts PORTF_DIRCLR, r16       ; used DIRCLR for practice, really it sets every bit that = 1 in r16
	                            ; to 0. That sets them to inputs

	;switches on OOTB EBIBB
	ldi r16, 0b00000011
	sts PORTE_DIRCLR, r16

	; recover relevant registers
	pop r16
	; return from subroutine
	ret

;*****************************************************************************
; Name: TC_INIT
; Purpose: To initialize the relevant timer/counter modules, as pertains to
;               application.
; Input(s): N/A
```

```
; Output: N/A
;***************************************************************************
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                    0.05
	;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   /20  = 7812/20 = 3906
	sts (TCC0_PER + 1), r16

	; recover relevant registers
		pop r16

	; return from subroutine
	ret

;*****************************END OF SUBROUTINES*******************************

;*****************************END OF "lab2_4.asm"******************************
```

APPENDIX

Supporting code above MAIN:

```
;***********************************************************************
;  File name: lab2_4.asm
;  Author: Christopher Crary
;  Last Modified By: Koby Miller
;  Last Modified On: 8 June 2020
;  Purpose: To allow LED animations to be created with the OOTB ?AD,
;                   OOTB SLB, and OOTB MB (or EBIBB, if a previous version of the kit
;                   is used).
;
;                   NOTE: The use of this file is NOT required! This file is just given
;                   as an example for how to potentially write code more effectively.
;***********************************************************************

;*********************************INCLUDES************************************

; The inclusion of the following file is REQUIRED for our course, since
; it is intended that you understand concepts regarding how to specify an
; "include file" to an assembler.
.include "ATxmega128a1udef.inc"
;*****************************END OF INCLUDES*********************************

;*****************************DEFINED SYMBOLS*********************************
.equ ANIMATION_START_ADDR   =      0x2000 ;useful, but not required
.equ ANIMATION_SIZE                =      (0x3FFF - 0x2000);useful, but not required
.equ F_CPU = 2000000
.equ CLK_PRE = 256
.equ FRAME_PER_A = 1/20 ;50ms
.equ FRAME_PER_RECIP_A = 20
;*************************END OF DEFINED SYMBOLS******************************

;*****************************MEMORY CONSTANTS********************************
; data memory allocation
.dseg

.org ANIMATION_START_ADDR
ANIMATION:
.byte ANIMATION_SIZE
;*************************END OF MEMORY CONSTANTS*****************************

;*********************************MAIN PROGRAM********************************
.cseg
; upon system reset, jump to main program (instead of executing
; instructions meant for interrupt vectors)
.org 0x0000
       rjmp MAIN

; place the main program somewhere after interrupt vectors (ignore for now)
.org 0x0100          ; >= 0xFD
MAIN:
```

Note: C1 Width = time before toggle.

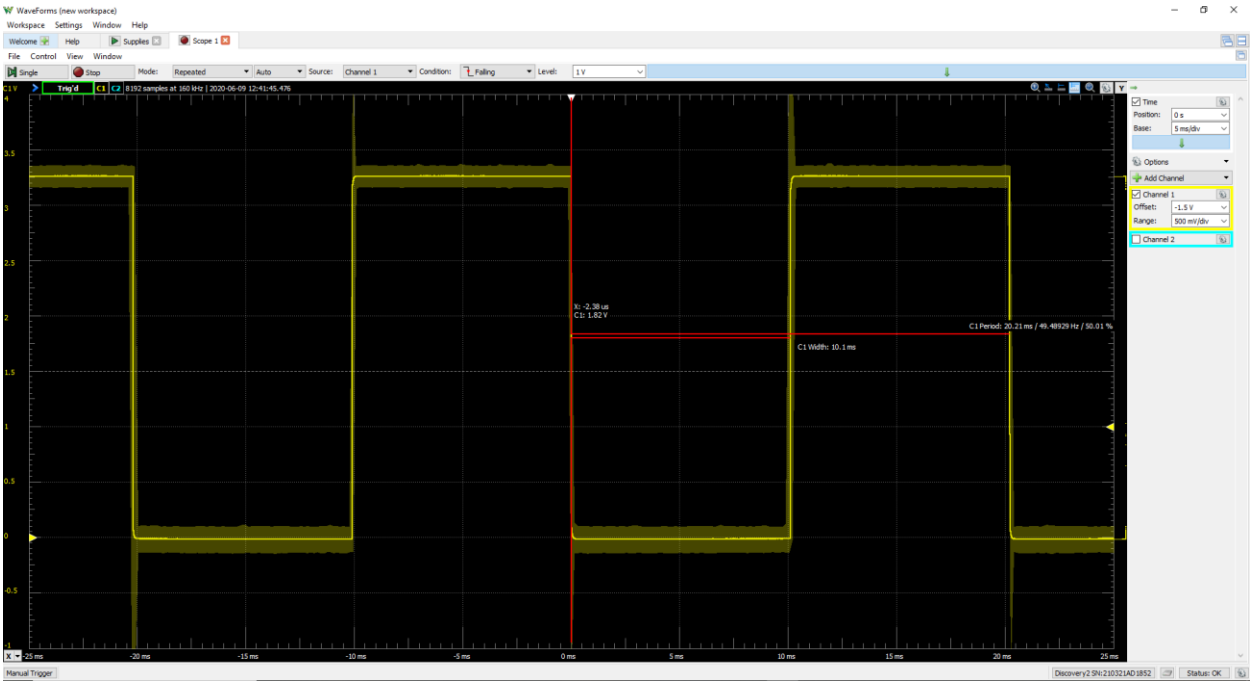


Figure 1: Section 2 DELAY_10MS
C1 Width = 10.1ms
Frequency = 49.48Hz

Note: In my code I reset my count 34 times. Mathematically resetting 33 times is closer to the amount of cycles I am supposed to count, however it was further from toggling every 10ms. When resetting 33 times I had a delay of 9.8ms, but when resetting 34 times I had a delay of 10.1ms

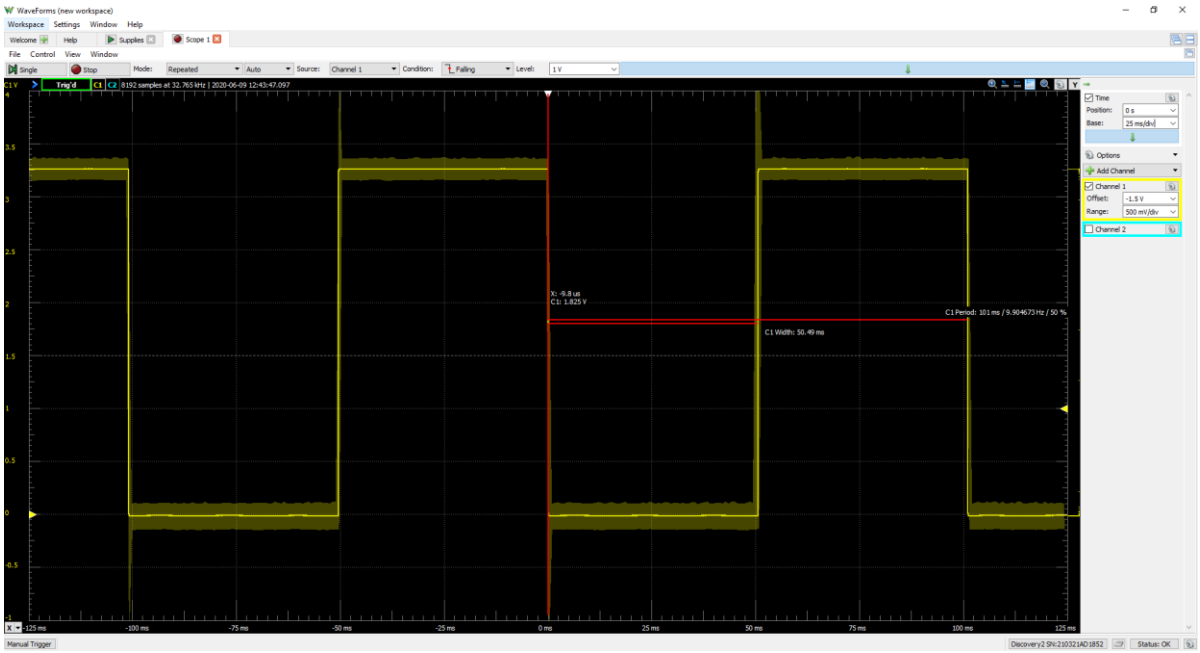


Figure 2: Section 2 DELAY_X_10MS where X = 5
C1 Width = 50.49ms
Frequency = 9.9Hz

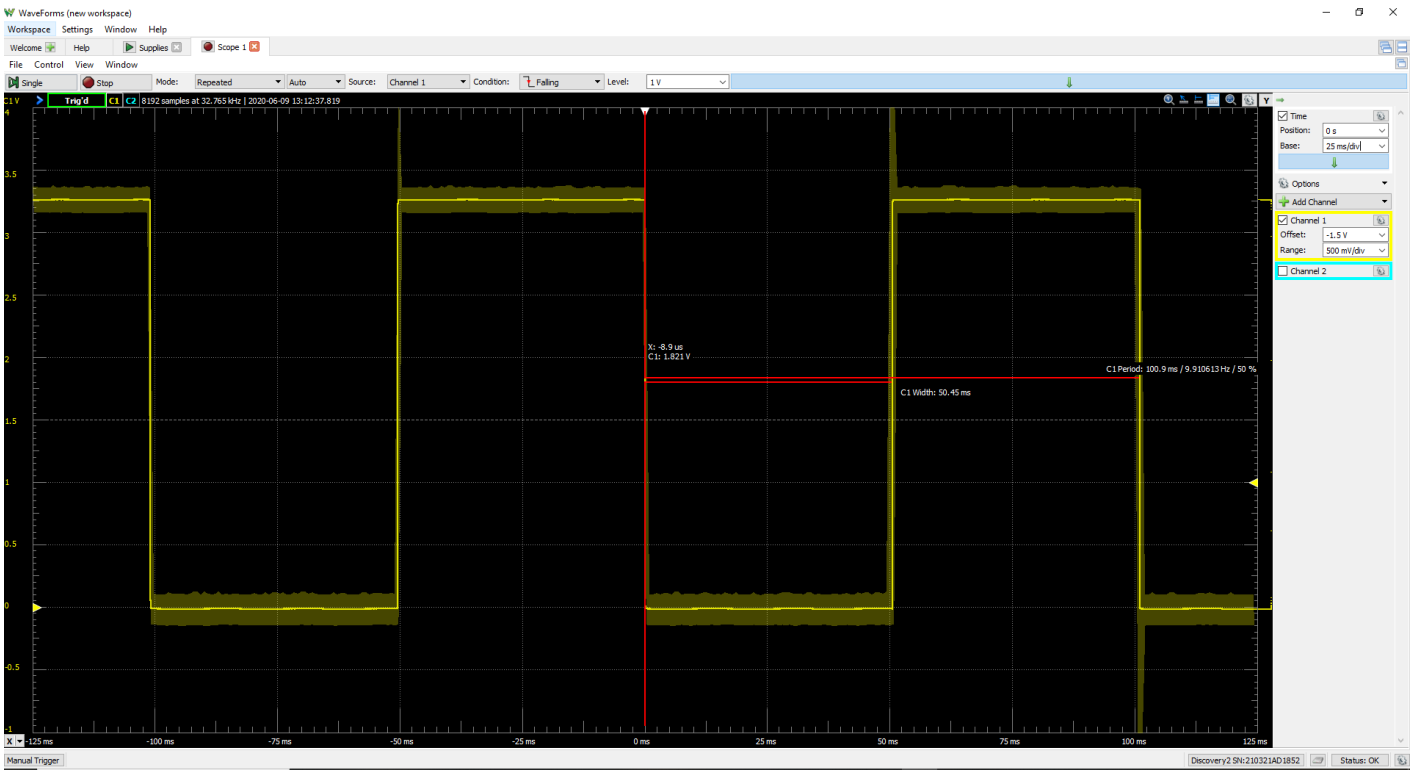


Figure 3: Section 2 Aiming for Frequency of 10Hz
C1 Width = 50.45ms
Frequency = 9.91Hz

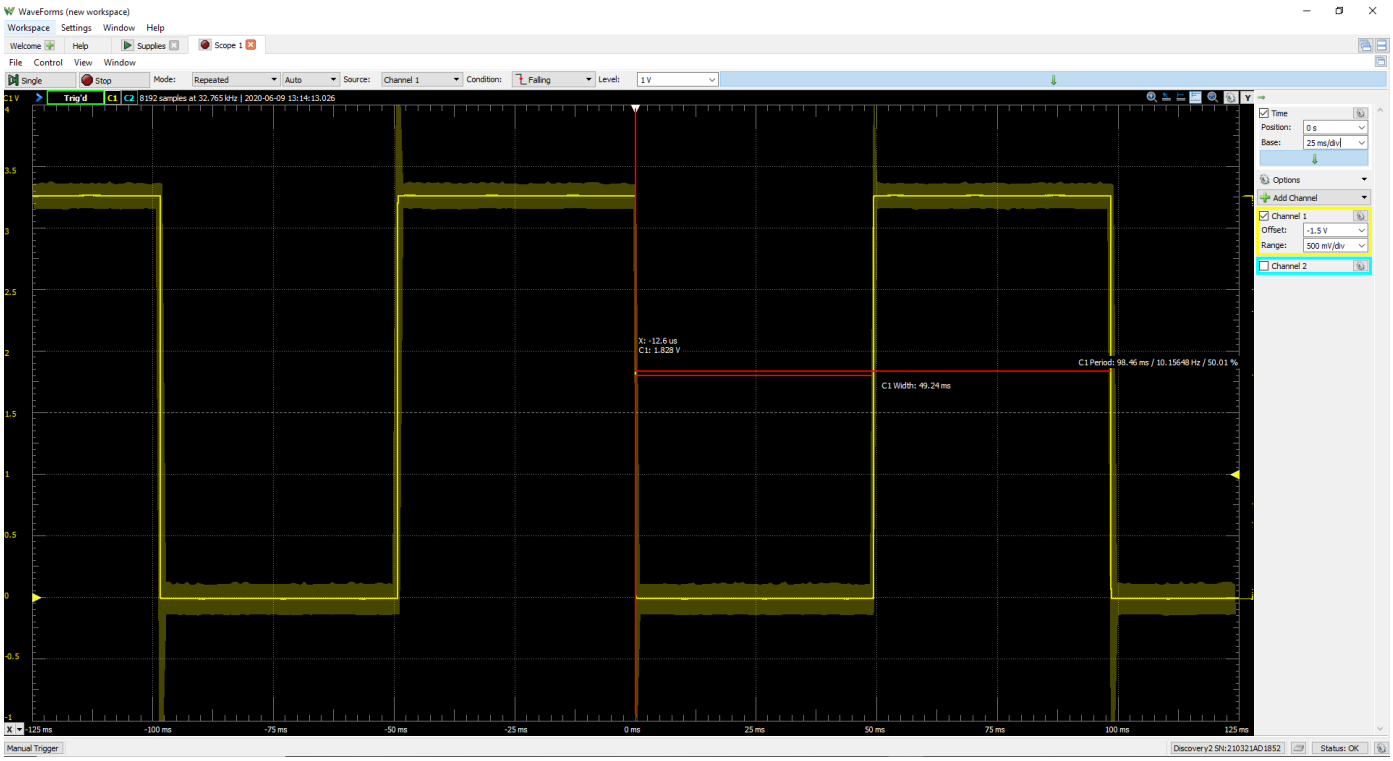


Figure 4: Section 3 Aiming for 50ms delay
Prescalar = 256
Period = 3906
C1 Width = 49.24ms
Frequency = 10.15Hz

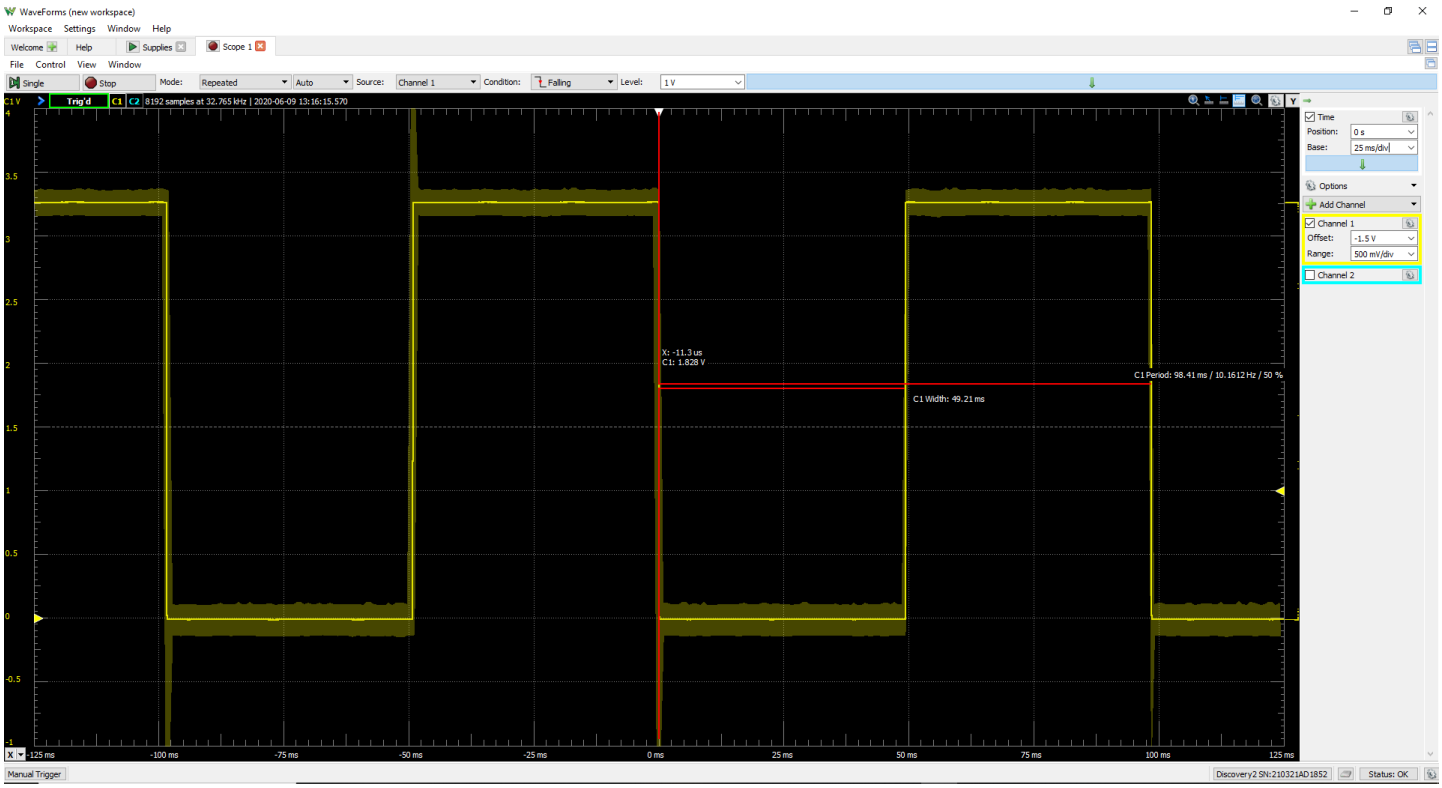


Figure 5: Section 3 Aiming for 50ms delay
Prescalar = 2
Period = 50,000
C1 Width = 49.21ms
Frequency = 10.16Hz

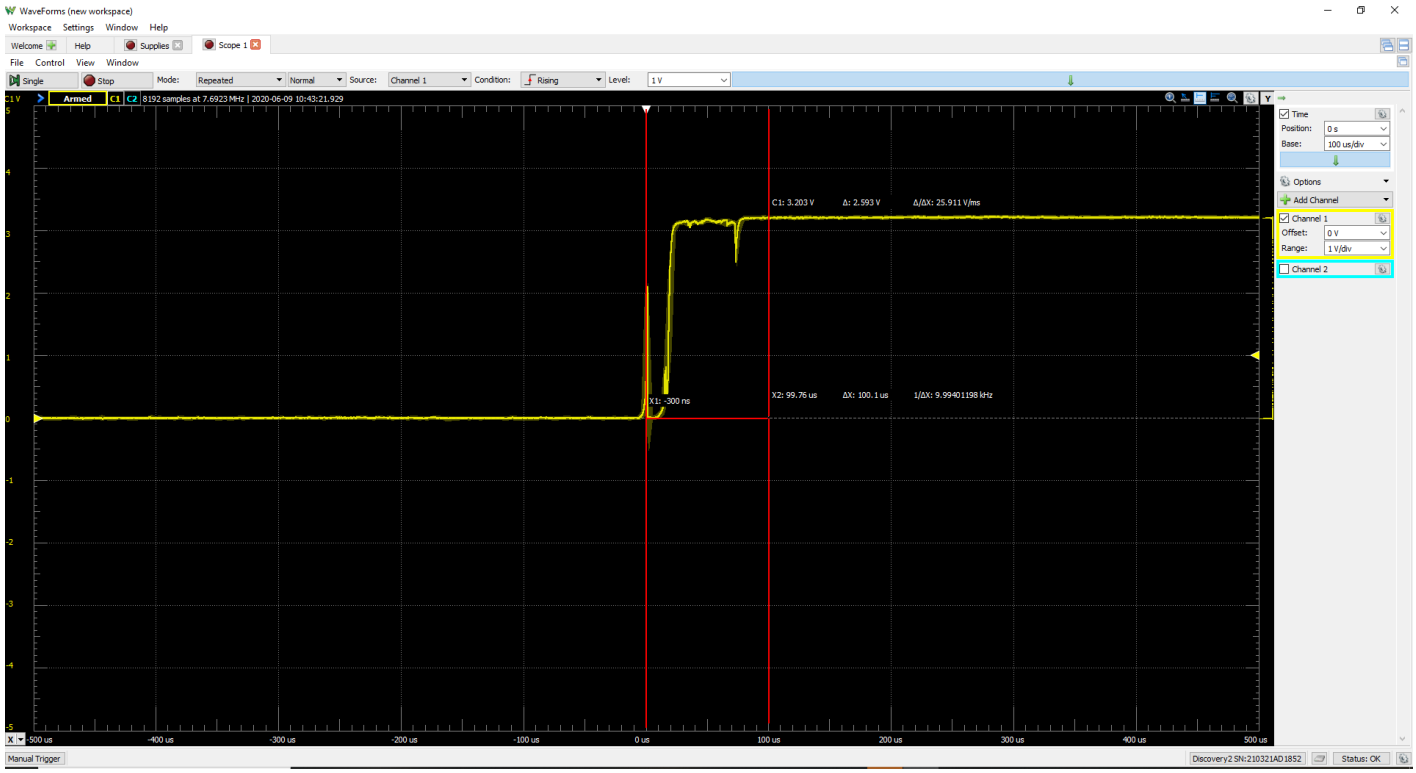


Figure 6: Section 4 Release Debouncing around 0.1ms

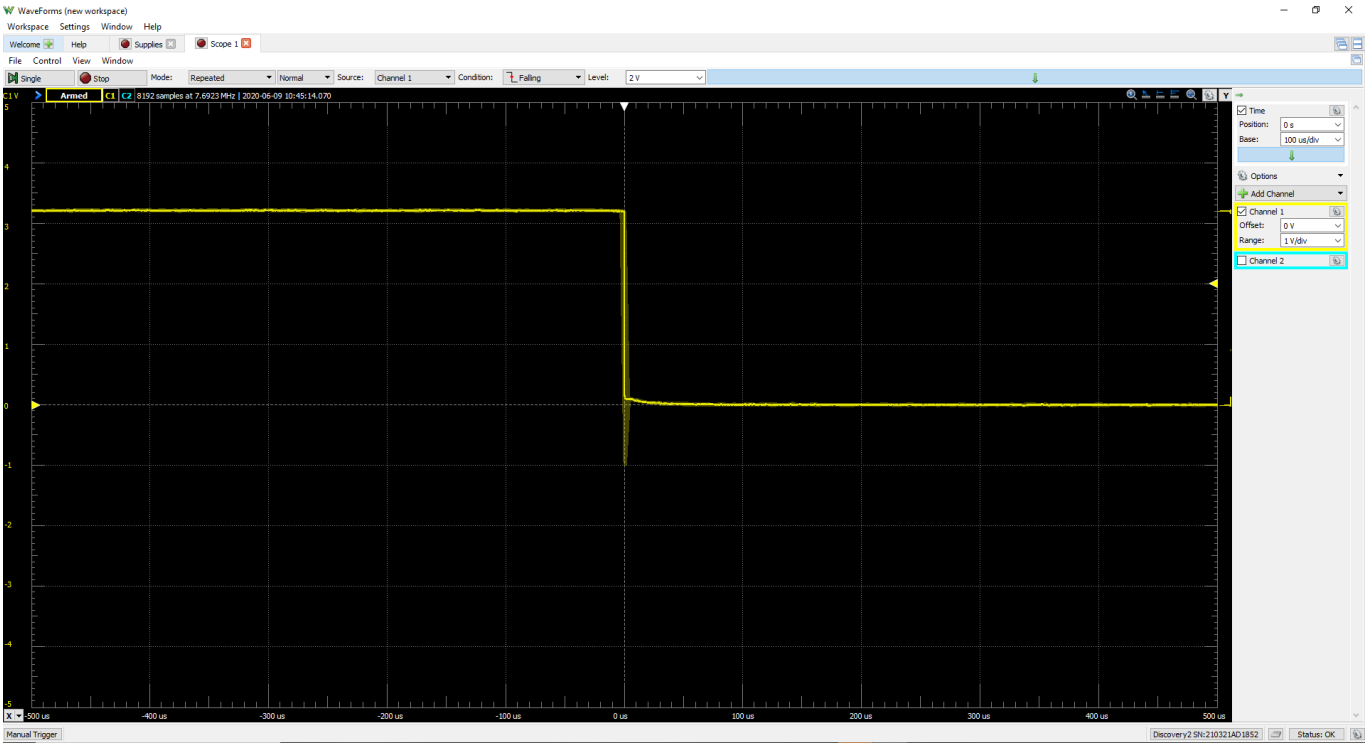


Figure 7: Section 4 Press Debouncing
I could not really get a debounce while pressing the switch but here is a screenshot