Lab 4
Pengzhao Zhu
Section: 112D

B) Prelab Questions

*Will it be necessary to remap the specified port?

*Yes, PD6 (BLUE_LED) doesn't have its own channel. So it will be necessary to remap

*the port for PART A of this lab (to output to the BLUE_LED).

1. What would happen if the RGB period was $FFFF instead of $FF?
   If the RGB period was $FFFF, we would have more options for different resolution and different light intensity. If the RGB was $FFFF instead of $FF but with the same $0E for compare (as in Part A), the BLUE_LED would be a lot less brighter.

2. How many TC0 channels are necessary to control all three LEDs in the µPAD's RGB LED?
   Three TC0 channels are necessary to control all three LEDs in the uPad's RGB LED

C) Problems Encountered

The first problem I encountered in this lab was the initialization of PWM and remap of the capture pin. The section on Timer/Compare/PWM was very confusing in the manual and I didn't know which waveform generator mode to use. It wasn't until I went to office hour that I figured out I can use single slope PWM mode for the lab (to output to LED).

The second problem I encountered in this lab was setting up the two interrupts used to debounce the switch. I couldn't get my code to debounce properly, so the count value would increment more than once per switch press. To solve the problem, I set the timer delay to 5ms for the interrupt to debounce properly.

D) Future Application

By completing this lab, I gained more knowledge of the timer system and the use of interrupts in microprocessors. The use of the timer capture and waveform generation functions will allow me to write more efficient codes for microprocessors. A precise count of real time using the timer system will be necessary for complex time dependent systems.

The second major skill I learned in this lab was the use of interrupts. Interrupts allow me to trigger a section of code that I want to run without polling the condition. It saves time, memory, and processing speed and will be necessary in any embedded systems that I will work on in the future.

E) Schematics

N/A

F) Pseudocode/Flowcharts

Part A Pseudocode:

```
Set up r23 to be 0x00 for the 32Mhz subroutine
rcall CLK  (to set up 32Mhz clock)

Initialize Stack Pointer to 0x3FFF

Set Blue LED to be output

Remap PORTD       ; move location of 0C02 from Px2 to Px6

Set the PER register of PORTD to 0xFF     ;0xFF for 0-255 of RGB LED

Set Timer clock on Port D to be CLK/1024

Use CTRLB to enable Compare/Capture C and enable single-slope PWM

Set TCD0_CCC to 0x0E

Invert the signal outputted to PORTD pin 6 using PORTD_PIN6CTRL (because LED is low-true)

DONE:
        rjmp DONE


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 r23 initialized outside the subroutine to set it up so it remains 32Mhz
```

```
pop r16

ret
```

Part C Pseudocode:

```
.org 0x0000
        rjmp MAIN

set up ISR name to jump to when encountered PORTF_INTO_vect

.org 0x100

MAIN:

Set up r17 to be a counter register for how many time the ISR has been executed

Set up r23 to be 0x00 for the 32Mhz subroutine

rcall CLK

Initialize Stack Pointer to 0x3FFF

Set 8 LED on PORTC to be output

Turn the 8 active low LED off

Enable low level interrupt for INT0

Set PF2 (Tactile switch S1) as the source for INT0

Set PF2 (Tactile switch S1) as input

Configure Pin2 of PortF to be falling edge trigger using PORTF_PIN2CTRL

Enable low level interrupt in the PMIC

sei

ldi r16, 0x70    ; to toggle the LED in an infinite loop

LOOP:

        sts PORTD_OUTTGL, r16

        rjmp LOOP


ISR:        ;ISR to output count to 8 LED

push r19
push r18
put CPU_SREG into r18
push r18
increment r17
Takes one’s complement of r17 because LED are active low
sts PORTC_OUT, r17
Takes one’s complement of r17 to ensure correct count

Clear interrupt flag using PORTF_INTFLAGS. Write a one to it
pop r18
Put value of r18 back to CPU_SREG
Pop r18
Pop r19
reti

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 r23 initialized outside the subroutine to set it up so it remains 32Mhz
```

```
pop r16

ret
```

Part D Pseudocode:

```
.org 0x0000
        rjmp MAIN

set up ISR name to jump to when encountered PORTF_INTO_vect

set up TIMER_ISR name to jump to when encountered TCD0_OVF_vect

.equ debounce_timer= (32000000*.005)/1024

.org 0x100

Set up r23 to use in the 32Mhz subroutine

Set r17 to be a register for how many times the ISR has been executed

rcall CLK

Initialize Stack Pointer to 0x3FFF

Set 8 LED on PORTC to be output

Turn the 8 active low LED off

rcall DEBOUNCE (to set up interrupt)

ldi r16, 0x70   ; to toggle the LED in an infinite loop

LOOP:

        sts PORTD_OUTTGL, r16

        rjmp LOOP


DEBOUCNE:

Push r16

Enable low level interrupt using PORTF_INTCTRL

Set PF2 (Tactile switch S1) as the source for INT0

Set PF2 (Tactile switch S1) as input

Configure the source to be falling edge trigger using PORTF_PIN2CTRL
```

```
Enable low level interrupt in the PMIC
Sei
Pop r16
Ret

ISR:
Push r18
Put CPU_SREG into r18
Push r18
Push r16
Setting the CNT back to zero
Set the PER for 5ms (math before the main code)
Configure timer clock for CLK/1024
Enable low level timer interrupt
Disable PORTF external interrupt
Pop r16
Pop r18
Value in r18 back to CPU_SREG
Pop r18
Reti

Timer_ISR:
Push r18
Value in CPU_SREG t or18
Push r18
Push r16
Disable Timer
Disable Timer Interrupt
Check if switch is set
```

```
brts NOT_ACTIVE
increment r17
Take one’s complement of r17
Value in r17 to PORTC_OUT
Take one’s complement of r17

NOT_ACTIVE:
Enable PORTF external interrupt
Clear the Interrupt Flag
Pop r16
Pop r18
Value in r18 to CPU_SREG
Pop r18
reti

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 r23 initialized outside the subroutine to set it up so it remains 32Mhz
```

```
pop r16

ret
```

Part E Pseudocode:

```
.org 0x0000
        rjmp MAIN

set up EDGE_ISR name to jump to when encountered PORTF_INTO_vect

set up DEBOUNCE_TIMER_ISR name to jump to when encountered TCC0_OVF_vect

set up BLINK_ISR name to jump to when encountered TCE0_OVF_vect

.equ debounce_timer= (32000000*.005)/1024

.equ blink_timer =(32000000*.1)/1024

Put values of RGB into a Table in Program Memory (use 4 tables)

.org 0x100

Set up r23 to use in the 32Mhz subroutine

Set r17 to be a register for how many times the ISR has been executed

Set r20 as the count register for the 8 LED

Set r22 to be 0

rcall CLK

Initialize Stack Pointer to 0x3FFF

Set 8 LED on PORTC to be output

Turn the 8 active low LED off

rcall DEBOUNCE

rcall PWM

rcall BLINK


DONE:
        rjmp DONE


DEBOUNCE:
```

```
Initialize the EDGE_ISR system
Ret


PWM:
Initialize the PWM system
Ret


BLINK::
Initialize the .1 second blink ISR
Ret


BLINK_ISR:
If r17 = 0  {       ;there is be a single bit in a register that will be inverted so it will alternate
                     ; the blinking of LED
Load first table to output nothing
} else if r17 =1 {
Load second table to blink between 2 colors of INCREDIBLE HULK
} else if r17 =2 {
Load third table to blink between 2 colors of HOLIDAY
} else if r17 =3 {
Load third table to blink between 2 colors of HOLIDAY


If r17 = 4 or greater than 4 {
Set r17=4
}
Setting the CNT back to zero
Clear the interrupt flag
Reti
```

```
EDGE_ISR:    ; to trigger when S1 is pressed
Initialize debounce timer for .5 ms
Enable Timer interrupt
Disable Port Interrupt
Reti

DEBOUNCE_TIMER_ISR:
Disable Timer
Disable Timer Interrupt
Enable Port Interrupt
Reti

BLINK_ISR:
Disable Timer
Disable Timer interrupt
Move to the next value in table because .1 second has been pressed already
Reti

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 r23 initialized outside the subroutine to set it up so it remains 32Mhz

pop r16

ret

G) Program Code

Part A Code

```
/* Lab 4 Part A
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program configures the PWM system and output a blue hue of $0E to
the blue LED
*/

.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

.org 0x100               ;start at 0x100

MAIN:
ldi r17, 0x00   ;setting for 32MHZ subroutine

rcall CLK                  ;call subroutine to set up 32MHZ clock

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 r16, 0b01000000         ;load r16
sts PORTD_DIRSET, r16    ;set blue LED to be output

ldi r16, 0b00000100         ;load r16
sts PORTD_REMAP, r16 ; to move location of OC02 from Px2 and Px6

ldi r16, 0xFF           ;load r16
sts TCD0_PER, r16           ;load lower byte of PER
ldi r16, 0x00               ;load r16
sts TCD0_PER+1, r16         ;load high byte of PER

ldi r16, 0b00000111         ;load r16
sts TCD0_CTRLA, r16         ;Timer clock for clk/1024

ldi r16, 0b01000011         ;load r16
sts TCD0_CTRLB, r16         ; bit 6 is Compare C enable. bit 2-0 is single-slope PWM

ldi r16, 0x0E               ;load r16
sts TCD0_CCC, r16           ;load CCC
ldi r16, 0x00               ;load r16
sts TCD0_CCC+1, r16      ;setting the compare value to 0x000E

ldi r16, 0b01000000                ;load r16
```

```
sts PORTD_PIN6CTRL, r16            ;invert the signal because the LED are low true

DONE:
       rjmp DONE                          ;infinite loop


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
```

Part C Code

```
/* Lab 4 Part C
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program configures the Xmega port interrupt to trigger when S1 is
pressed.
                            It will then output how many time S1 has been pressed to the 8
LED.
                            No debounce performed
*/



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

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

```
.org PORTF_INT0_vect                  ;tell the ISR where to jump to
       rjmp ISR

.equ stack_init=0x3FFF   ;initialize stack pointer

.org 0x100

MAIN:
ldi r23, 0x00   ;setting for 32MHZ subroutine.

ldi r17, 0x00   ;counter register for how many time the ISR has been executed

rcall CLK

ldi YL, low(stack_init)    ;Load 0xFF to YL
sts CPU_SPL, YL                      ;transfer to CPU_SPL
ldi YL, high(stack_init)   ;Load 0x3F to YH
sts CPU_SPH, YL                      ;transfer to CPU_SPH

ldi r16, 0xFF
sts PORTC_DIRSET, r16    ;set the 8 LED to be output
sts PORTC_OUT, r16       ;turn the 8 active low LED off

ldi r16, 0x40
sts PORTD_DIRSET, r16    ;set the BLUE LED to be output
sts PORTD_OUT, r16       ;set the BLUE LED to be output


ldi r16, 0x01         ;enable low level interrupt for INT0
sts PORTF_INTCTRL, r16

ldi r16, 0x04
sts PORTF_INT0MASK, r16    ;set PF2 (tactile switch S1) as the source for INT0

sts PORTF_DIRCLR, r16    ;set PF2 (tactile switch S1) as input

ldi r16, 0b00000010     ; the last 3 bits "010" corresponds to falling edge sense trigger
sts PORTF_PIN2CTRL, r16

ldi r16, 0x01
sts PMIC_CTRL, r16      ;enable low level interrupt in the PMIC

sei    ;setting the I bit

ldi r16, 0x70                                         ;0x70 to toggle the pin. Used in
the infinite loop

LOOP:
       sts PORTD_OUTTGL, r16              ;keep toggling the LED
       rjmp LOOP                                        ;infinite loop in the main
code


ISR:
       push r19                                       ;push the necessary registers and
also the status register
       push r18
       lds r18, CPU_SREG
       push r18
       inc r17             ;to increase r17 by 1 everytime the ISR is executed
       com r17             ;takes one's complement of r17 because LED are active low
       sts PORTC_OUT, r17
       com r17             ;takes one's complement again so r17 is correct the next time 
we execuate the ISR

       ldi r19, 0x01                               ;load r19
       sts PORTF_INTFLAGS, r19                     ;clear the interrupt flag

       pop r18                                            ;return the registers to 
their original value. including the status register
       sts CPU_SREG, r18
       pop r18
       pop r19
       reti


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, r23  ;r17 will be initialized outside the subroutine for prescale. 
32/8=4MHZ

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

Part D Code

```
/* Lab 4 Part D
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program configures the Xmega port interrupt to trigger when S1 is 
pressed.
                            It will then output how many time S1 has been pressed to the 8 
LED.
                            Debounced performed with a second timer interrupt. Added to 
the first interrupt
*/


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

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

.org PORTF_INT0_vect                     ;tell the uP where to jump to when edge 
interrupt is triggered
       rjmp ISR

.org TCD0_OVF_vect                              ;tell the uP where to jump to when timre 
interrupt is triggered
       rjmp TIMER_ISR

.equ stack_init=0x3FFF   ;initialize stack pointer
.equ debounce_timer= (32000000*.005)/1024              ;.equ for the PER value of the 
timer interrupt

.org 0x100

MAIN:
ldi r23, 0x00   ;setting for 32MHZ subroutine.

ldi r17, 0x00   ;counter register for how many time the ISR has been executed

rcall CLK                   ;call CLK to configure 32MHZ clock

ldi YL, low(stack_init)    ;Load 0xFF to YL
sts CPU_SPL, YL                       ;transfer to CPU_SPL
ldi YL, high(stack_init)   ;Load 0x3F to YH
sts CPU_SPH, YL                       ;transfer to CPU_SPH

ldi r16, 0xFF
sts PORTC_DIRSET, r16    ;set the 8 LED to be output
sts PORTC_OUT, r16       ;turn the 8 active low LED off

ldi r16, 0x40
sts PORTD_DIRSET, r16    ;set the RGB LED to be output
sts PORTD_OUT, r16       ;turn the RGB LED off

rcall DEBOUNCE


ldi r16, 0x70

LOOP:
sts PORTD_OUTTGL, r16                 ;keep toggling the BLUE LED
rjmp LOOP
```

```
DEBOUNCE:
push r16
ldi r16, 0x01         ;enable low level interrupt for INT0
sts PORTF_INTCTRL, r16

ldi r16, 0x04
sts PORTF_INT0MASK, r16    ;set PF2 (tactile switch S1) as the source for INT0

sts PORTF_DIRCLR, r16    ;set PF2 (tactile switch S1) as input

ldi r16, 0b00000010     ; the last 3 bits "010" corresponds to falling edge sense trigger
sts PORTF_PIN2CTRL, r16

ldi r16, 0x01
sts PMIC_CTRL, r16      ;enable low level interrupt in the PMIC

sei    ;setting the I bit

pop r16
ret

ISR:
push r18                                                                    ;push the
necessary registers. including the status register
lds r18, CPU_SREG
push r18
push r16

ldi r16, 0x00                                  ;setting the CNT back to zero
sts TCD0_CNT, r16

ldi r16, low(debounce_timer)                   ;set the timer for 5ms to debounce
sts TCD0_PER, r16                                                    ;need to load low
and high byte of PER
ldi r16, high(debounce_timer)
sts TCD0_PER+1, r16

ldi r16, 0b00000111                           ;Timer clock for clk/1024
sts TCD0_CTRLA, r16

ldi r16, 0x01                                    ;enable timer interrupt
sts TCD0_INTCTRLA, r16

ldi r16, 0x00                                     ;disable PortF external interrupt
sts PORTF_INTCTRL, r16

pop r16                                                                             ;pop
the necessary registers. including the status register
pop r18
sts CPU_SREG, r18
pop r18
reti

TIMER_ISR:
push r18                                                      ;push the necessary
registers. including the status register
lds r18, CPU_SREG
```

```
push r18
push r16

ldi r16, 0x00                           ;disable timer
sts TCD0_CTRLA, r16

ldi r16, 0x00
sts TCD0_INTCTRLA, r16                                 ;disable timer interrupt

lds r16, PORTF_IN       ;to check if switch is still active
bst r16, 2
brts NOT_ACTIVE

inc r17             ;to increase r17 by 1 everytime the ISR is executed
com r17             ;takes one's complement of r17 because LED are active low

sts PORTC_OUT, r17
com r17             ;takes one's complement again so r17 is correct the next time we
execuate the ISR

NOT_ACTIVE:
ldi r16, 0x01                                    ;enable PortF external interrupt
sts PORTF_INTCTRL, r16

ldi r16, 0x01                                    ;clear the interrupt flag
sts PORTF_INTFLAGS, r16

pop r16                                          ;prepare to return from interrupt
pop r18
sts CPU_SREG, r18                                                    ;pop the necessary
registers. including the status register
pop r18
reti



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, r23  ;r17 will be initialized outside the subroutine for prescale.
32/8=4MHZ
```

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

Part E Code

```
/* Lab 4 Part E
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This Program blinks the RGB between two color combination with a .1 blink 
alternation period
                            Four settings in total (4 sets of combination for RGB blink )
                            It will also output the number of times S1 is pressed to the 8 
LEDs
*/


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

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

.org PORTF_INT0_vect       ;for the edge trigger debounce interrupt
       rjmp EDGE_ISR

.org TCC0_OVF_vect       ;for the 5 ms timer debounce interrupt
       rjmp DEBOUNCE_TIMER_ISR

.org TCE0_OVF_vect       ;for the .1 second blink timer interrupt
       rjmp BLINK_ISR

;Have to use TCC0 and TCE0 because PER OF TCD0 is used for PWM

.equ stack_init=0x3FFF   ;initialize stack pointer
.equ debounce_timer= (32000000*.005)/1024      ;.equ for PER of debounce timer ISR
.equ blink_timer =(32000000*.1)/1024                   ;.equ for PER of blink timer ISR

.org 0x100
Table :.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00        ;table for no LED light up
Table1:.db 0x9A, 0x2C, 0x8A, 0x07, 0xFF, 0x49           ;table for INCREDIBLE HULK
Table2:.db 0x1F, 0x1F, 0xC2, 0x0D, 0x8D, 0x3C           ;table for HOLIDAY
Table3:.db 0x16, 0x46, 0xFA, 0xA5, 0x21, 0x00           ;table for UF

.org 0x200

MAIN:
ldi r23, 0x00   ;setting for 32MHZ subroutine.
ldi r17, 0x00   ;counter register for how many time the ISR has been executed
ldi r22, 0x00   ;bit0, capture D(BLUE). BIT1, capture C(Green). Bit 2, capture (RED)    . 
0 means forward, 1 means back
                            ;register so I know which data to take
```

```
ldi r20, 0x00   ;counter register for the 8 LED
rcall CLK

ldi YL, low(stack_init)    ;Load 0xFF to YL
sts CPU_SPL, YL                       ;transfer to CPU_SPL
ldi YL, high(stack_init)   ;Load 0x3F to YH
sts CPU_SPH, YL                       ;transfer to CPU_SPH

ldi r16, 0xFF              ;just want to see the outputted
sts PORTC_DIRSET, r16    ;set the 8 LED to be output
sts PORTC_OUT, r16       ;turn the 8 active low LED off

rcall DEBOUNCE   ;call subroutine to initialze port edge interrupt
rcall PWM    ;call subroutine to initialize PWM
rcall BLINK  ;call subroutine to set up .1 second switch interrupt

DONE:
       rjmp DONE

PWM:
push r16
ldi r16, 0x70
sts PORTD_DIRSET, r16    ;set the RGB LED to be output
sts PORTD_OUT, r16       ;turn the RGB LED off

ldi r16, 0b00000111   ; remap the RGB LED
sts PORTD_REMAP, r16

ldi r16, 0xFF           ;load r16
sts TCD0_PER, r16       ;load low byte of PER
ldi r16, 0x00           ;load r16
sts TCD0_PER+1, r16         ;load high byte of PER

ldi r16, 0b00000111   ;Timer clock for clk/1024
sts TCD0_CTRLA, r16

ldi r16, 0b01110011    ; enablc Compare A,B,C. bit 2-0 is single-slope PWM
sts TCD0_CTRLB, r16

ldi r16, 0x00
sts TCD0_CCC, r16                 ;CCC is for blue LED
sts TCD0_CCC+1, r16                      ;need to load low and high byte

sts TCD0_CCB, r16                  ;CCB is for Green LED
sts TCD0_CCB+1, r16                ;need to load low and high byte

sts TCD0_CCA, r16                  ;CCA is for Red LED
sts TCD0_CCA+1, r16                ;need to load low and high byte

ldi r16, 0b01000000                     ;invert the signal because they are low-true LED
sts PORTD_PIN6CTRL, r16                                ;invert 3 pins between there are
Red, Green, and Blue LED. 3 of them
sts PORTD_PIN5CTRL, r16
sts PORTD_PIN4CTRL, r16

pop r16
ret                                                                  ;return
```

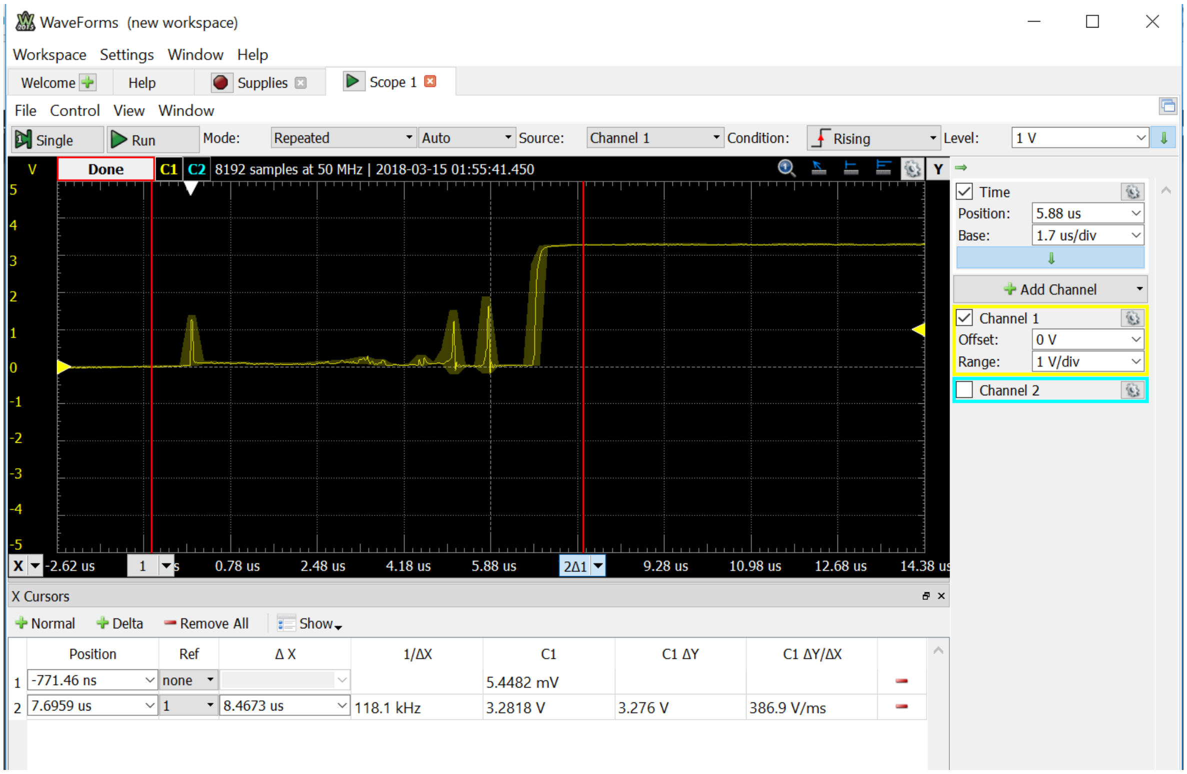
```
DEBOUNCE:
push r16
ldi r16, 0x02         ;enable medium level interrupt for INT0
sts PORTF_INTCTRL, r16

ldi r16, 0x04
sts PORTF_INT0MASK, r16    ;set PF2 (tactile switch S1) as the source for INT0

sts PORTF_DIRCLR, r16    ;set PF2 (tactile switch S1) as input

ldi r16, 0b00000010     ; the last 3 bits "010" corresponds to falling edge sense trigger
sts PORTF_PIN2CTRL, r16

ldi r16, 0x03
sts PMIC_CTRL, r16      ;enable low level and medium level interrupt in the PMIC

sei    ;setting the I bit

pop r16
ret

BLINK:                                       ;initialize the timer/timer interrupt for .1
second blink
push r16
ldi r16, low(blink_timer)                    ;set the timer for .1 between blink
sts TCE0_PER, r16                                              ;need to load low and high
byte
ldi r16, high(blink_timer)
sts TCE0_PER+1, r16

ldi r16, 0b00000111                           ;Timer clock for clk/1024
sts TCE0_CTRLA, r16

ldi r16, 0x01                                    ;enable low level timer interrrupt
sts TCE0_INTCTRLA, r16


pop r16
ret
       ;return from subroutine

EDGE_ISR:                                        ;initialization for edge interrupt
push r18                                                                    ;push the
necessary registers and status register
lds r18, CPU_SREG
push r18
push r16

ldi r16, 0x00                                  ;setting the CNT back to zero
sts TCC0_CNT, r16

ldi r16, low(debounce_timer)                    ;set the timer for 5ms to debounce
sts TCC0_PER, r16                                                   ;need to load low
and high byte of PER
ldi r16, high(debounce_timer)
```

```
sts TCC0_PER+1, r16

ldi r16, 0b00000111                           ;Timer clock for clk/1024
sts TCC0_CTRLA, r16

ldi r16, 0x02                                    ;enable medium timer interrupt
sts TCC0_INTCTRLA, r16

ldi r16, 0x00                                     ;disable PortF external interrupt
sts PORTF_INTCTRL, r16

pop r16                                                                     ;pop the
necessary registers. including the status register
pop r18
sts CPU_SREG, r18
pop r18
reti                                                                 ;return from
interrupt

DEBOUNCE_TIMER_ISR:                  ;initialization for the 5ms timer used for debounce
push r18
lds r18, CPU_SREG                               ;push the necessary registers and status
register
push r18
push r16

ldi r16, 0x00                           ;disable timer
sts TCC0_CTRLA, r16

ldi r16, 0x00
sts TCC0_INTCTRLA, r16                                 ;disable timer interrupt

lds r16, PORTF_IN       ;to check if switch is still active
bst r16, 2
brts NOT_ACTIVE             ;if set, it means it is not active

inc r17                     ;if active, increment r17. r17 is a counter
                                                ;the rest of separate. the next 4 line of
code are for 8 LED count
inc r20             ;to increase r20 by 1 everytime the ISR is executed
com r20             ;takes one's complement of r20 because LED are active low

sts PORTC_OUT, r20                ;output the value
com r20             ;takes one's complement again so r20 is correct the next time we
execuate the ISR

NOT_ACTIVE:

ldi r16, 0x02                                    ;enable PortF external medium level
interrupt
sts PORTF_INTCTRL, r16

ldi r16, 0x01                                    ;clear the interrupt flag
sts PORTF_INTFLAGS, r16


pop r16                                          ;prepare to return from interrupt
pop r18                                                                              ;pop
the necessary registers. including the status register
sts CPU_SREG, r18
pop r18
reti


BLINK_ISR:
push ZL                               ;pushes the necessary registers. including the
status register
push ZH
push r18
lds r18, CPU_SREG
push r18
push r16



cpi r17, 0                          ;check the counter for how many times tactile switch
S1 is pressed
breq STORE                                                ;Jump to different part of the
code depending on r17 value
cpi r17, 1
breq STORE1
cpi r17, 2                                                ;should be self explanatory
breq STORE2
cpi r17, 3
breq STORE3

STORE:                                           ;load corresponding tables
ldi ZL, low(Table<<1)                                              ;r17=0, no output
ldi ZH, high(Table<<1)
rjmp CHECK

STORE1:
ldi ZL, low(Table1<<1)                                    ;load corresponding tables
ldi ZH, high(Table1<<1)                                   ;r17=1, INCREDIBLE HULK color
rjmp CHECK

STORE2:
ldi ZL, low(Table2<<1)                                    ;load corresponding tables
ldi ZH, high(Table2<<1)                                   ;r17=2, HOLIDAY color
rjmp CHECK

STORE3:
ldi ZL, low(Table3<<1)                                    ;load corresponding tables
ldi ZH, high(Table3<<1)                                   ;r17=3, UF Color

CHECK:
bst r22, 0                                                ;r22 is just a register so I how
which half of the table to load
brtc LOAD                            ;will be inverted everytime

adiw ZH:ZL, 3                        ;add 3 to counter for blinking

LOAD:
ldi r16, 0x08                        ;force restart of the TC system
sts TCD0_CTRLFSET, r16
```

```
lpm r16, Z+                         ;load corresponding table values for each capture
sts TCD0_CCC, r16                               ;CCC is for blue LED
ldi r16, 0x00                                   ;need to load low and high byte
sts TCD0_CCC+1, r16

lpm r16, Z+                                            ;load corresponding table values
for each capture
sts TCD0_CCB, r16                               ;CCB is for green LED
ldi r16, 0x00                                   ;need to load low and high byte
sts TCD0_CCB+1, r16

lpm r16, Z+                                            ;load corresponding table values
for each capture
sts TCD0_CCA, r16                               ;CCA is for red LED
ldi r16, 0x00                                   ;need to load low and high byte
sts TCD0_CCA+1, r16

com r22  ;take one's complement of r22. so it will load different half of the table set
next time

cpi r17, 4                                             ;compare r17 with 4
brsh RESET                                             ;branch if greater or equal to 4
brne SKIP                                              ;otherwise, branch to SKIP
RESET:
ldi r17, 0                           ;when counter half reach 4. start over. load r17=0

SKIP:

ldi r16, 0x00                                  ;setting the CNT back to zero
sts TCE0_CNT, r16

ldi r16, 0x01
sts TCE0_INTFLAGS, r16                         ;clear the interrupt flag

pop r16
pop r18                                                                     ;pop the
necessary register and prepare to return from interrupt
sts CPU_SREG, r18
pop r18
pop ZH
pop ZL
reti




CLK:   ;take in a r23 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, r23  ;r17 will be initialized outside the subroutine for prescale.
32/8=4MHZ

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

H) Appendix



Mode: Repeated · Auto · Source: Channel 1 · Condition: Rising · Level: 1 V

Done · C1 · C2 · 8192 samples at 50 MHz | 2018-03-15 01:55:41.450

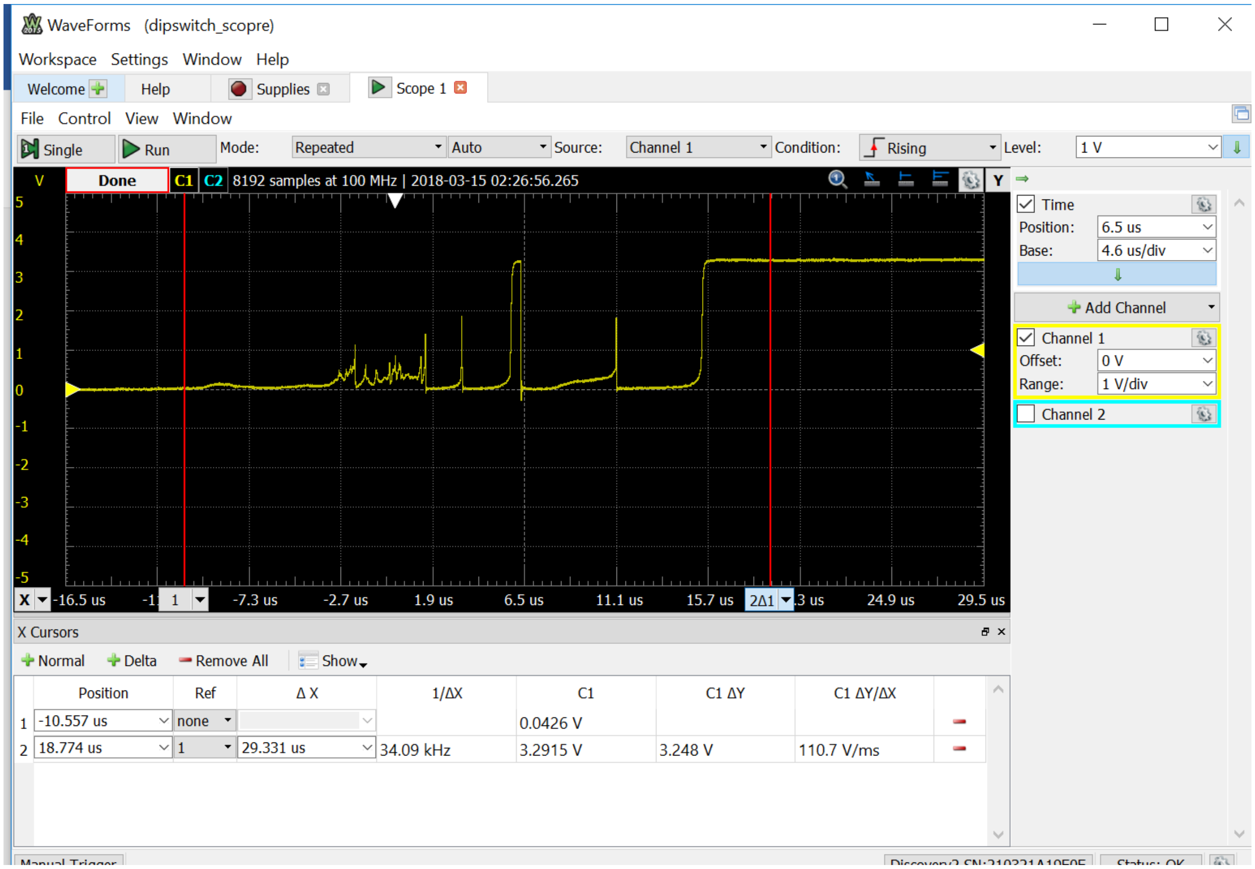
☑ Time
Position: 5.88 us
Base: 1.7 us/div

☑ Channel 1
Offset: 0 V
Range: 1 V/div

□ Channel 2

	Position	Ref	Δ X	1/ΔX	C1	C1 ΔY	C1 ΔY/ΔX
1	-771.46 ns	none			5.4482 mV		
2	7.6959 us	1	8.4673 us	118.1 kHz	3.2818 V	3.276 V	386.9 V/ms

Figure 1: Rising Edge Trigger of DIP Switch Screenshot 1 (On to Off position)



Mode: Repeated
Auto
Source: Channel 1
Condition: Rising
Level: 1 V

Done | 8192 samples at 100 MHz | 2018-03-15 02:26:56.265

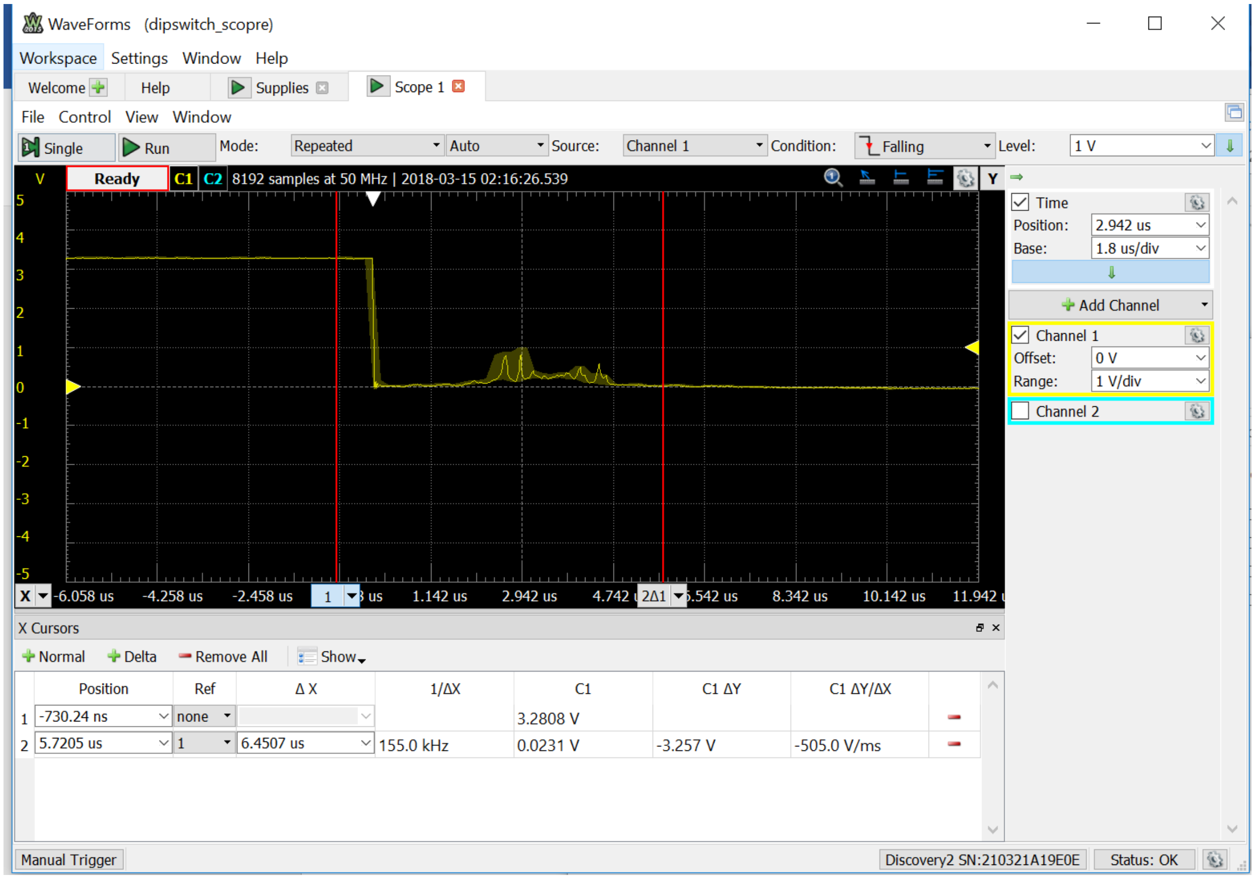
☑ Time
Position: 6.5 us
Base: 4.6 us/div

☑ Channel 1
Offset: 0 V
Range: 1 V/div

□ Channel 2

	Position	Ref	Δ X	1/ΔX	C1	C1 ΔY	C1 ΔY/ΔX
1	-10.557 us	none			0.0426 V		
2	18.774 us	1	29.331 us	34.09 kHz	3.2915 V	3.248 V	110.7 V/ms

Figure 2: Rising Edge Trigger of DIP Switch Screenshot 2 (On to Off position)



Mode: Repeated

Auto

Source: Channel 1

Condition: Falling

Level: 1 V

8192 samples at 50 MHz | 2018-03-15 02:16:26.539

☑ Time

Position: 2.942 us

Base: 1.8 us/div

☑ Channel 1

Offset: 0 V

Range: 1 V/div

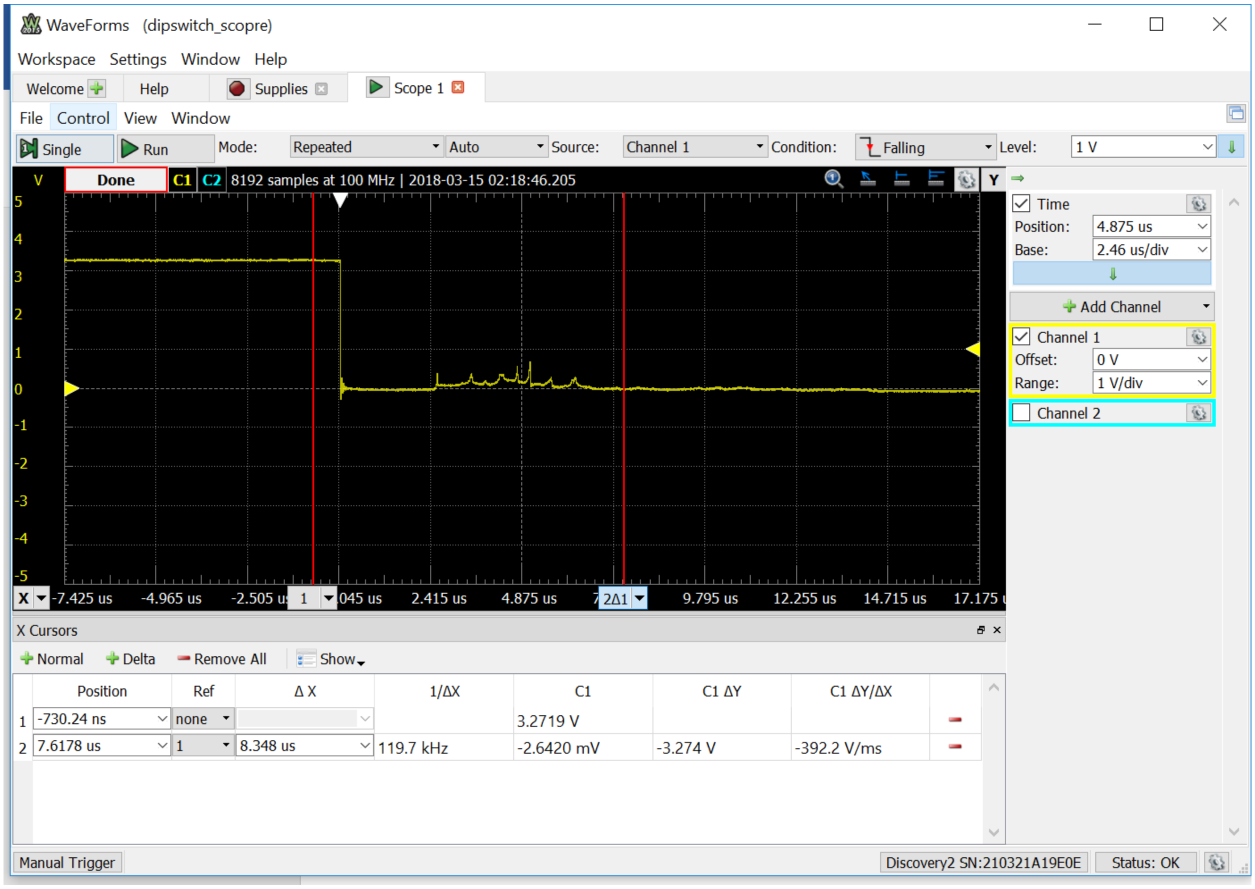
☐ Channel 2

	Position	Ref	Δ X	1/ΔX	C1	C1 ΔY	C1 ΔY/ΔX
1	-730.24 ns	none			3.2808 V		
2	5.7205 us	1	6.4507 us	155.0 kHz	0.0231 V	-3.257 V	-505.0 V/ms

Discovery2 SN:210321A19E0E

Status: OK

Figure 3: Falling Edge Trigger of DIP Switch Screenshot 1 (Off to On position)



Mode: Repeated | Auto | Source: Channel 1 | Condition: Falling | Level: 1 V

Done | 8192 samples at 100 MHz | 2018-03-15 02:18:46.205

☑ Time
Position: 4.875 us
Base: 2.46 us/div

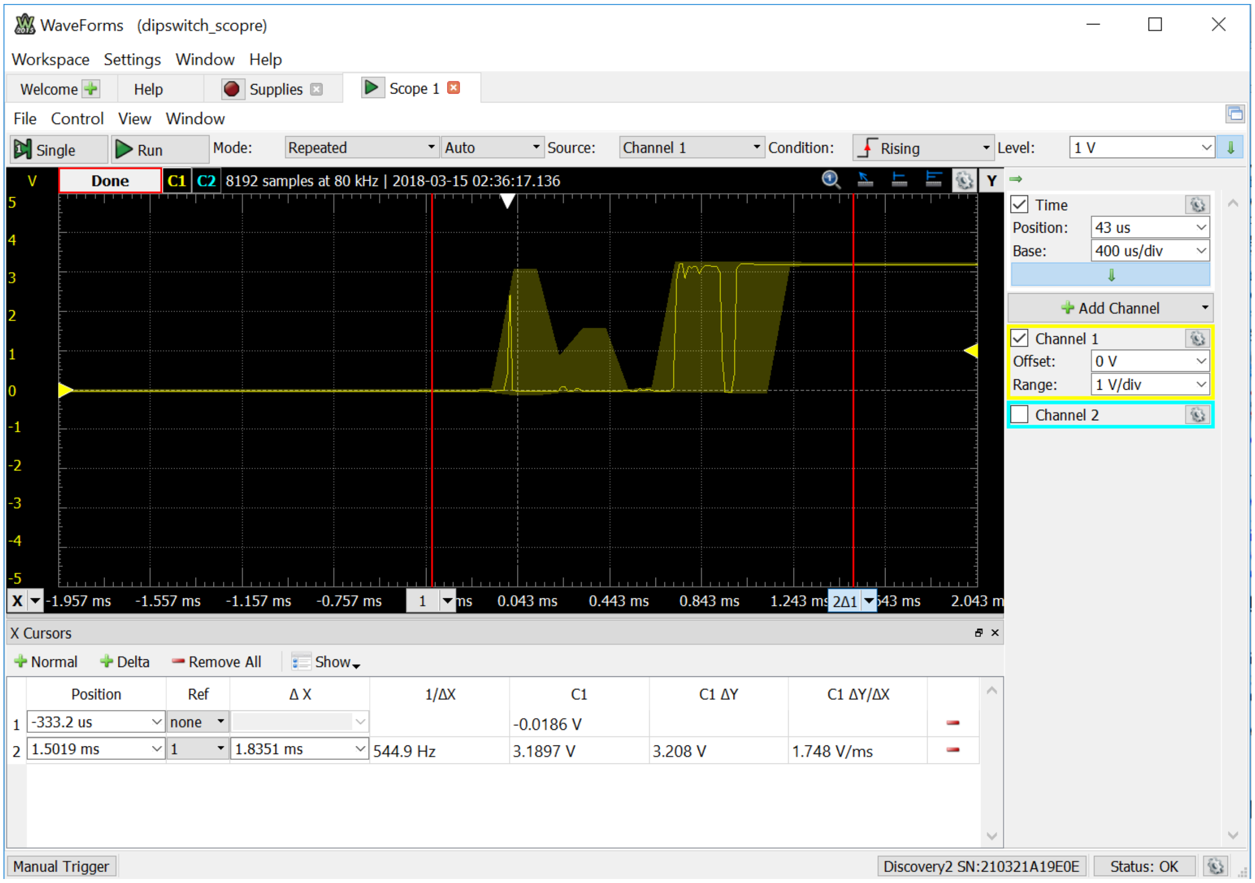
☑ Channel 1
Offset: 0 V
Range: 1 V/div

□ Channel 2

	Position	Ref	Δ X	1/ΔX	C1	C1 ΔY	C1 ΔY/ΔX
1	-730.24 ns	none			3.2719 V		
2	7.6178 us	1	8.348 us	119.7 kHz	-2.6420 mV	-3.274 V	-392.2 V/ms

Manual Trigger | Discovery2 SN:210321A19E0E | Status: OK

Figure 4: Falling Edge Trigger of DIP Switch Screenshot 2 (Off to On position)



Mode: Repeated | Auto | Source: Channel 1 | Condition: Rising | Level: 1 V

Done | 8192 samples at 80 kHz | 2018-03-15 02:36:17.136

☑ Time
Position: 43 us
Base: 400 us/div

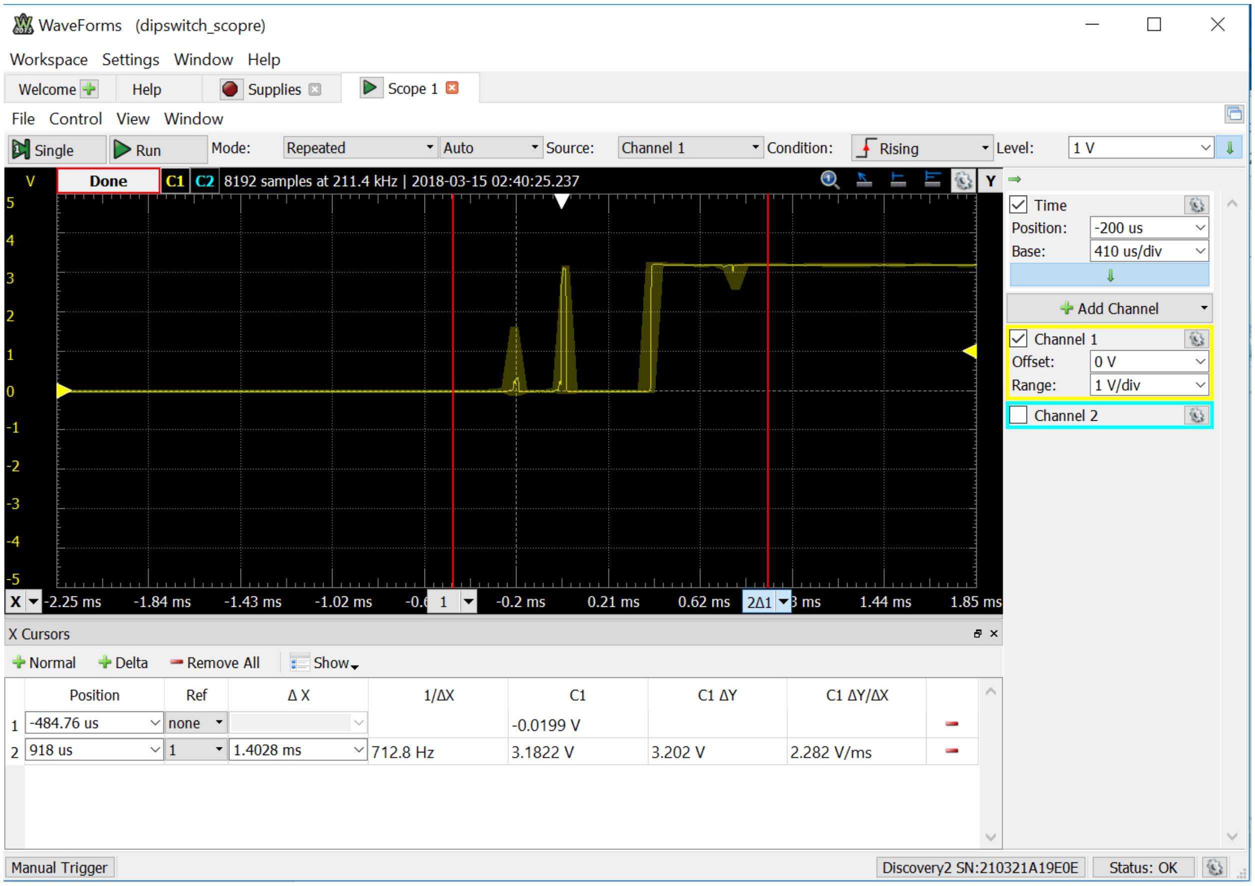
☑ Channel 1
Offset: 0 V
Range: 1 V/div

☐ Channel 2

	Position	Ref	Δ X	1/ΔX	C1	C1 ΔY	C1 ΔY/ΔX
1	-333.2 us	none			-0.0186 V		
2	1.5019 ms	1	1.8351 ms	544.9 Hz	3.1897 V	3.208 V	1.748 V/ms

Discovery2 SN:210321A19E0E | Status: OK

Figure 5: Rising Edge Trigger of Tactile Switch Screenshot 1 (Switch Release)



Mode: Repeated
Auto
Source: Channel 1
Condition: Rising
Level: 1 V

8192 samples at 211.4 kHz | 2018-03-15 02:40:25.237

☑ Time
Position: -200 us
Base: 410 us/div

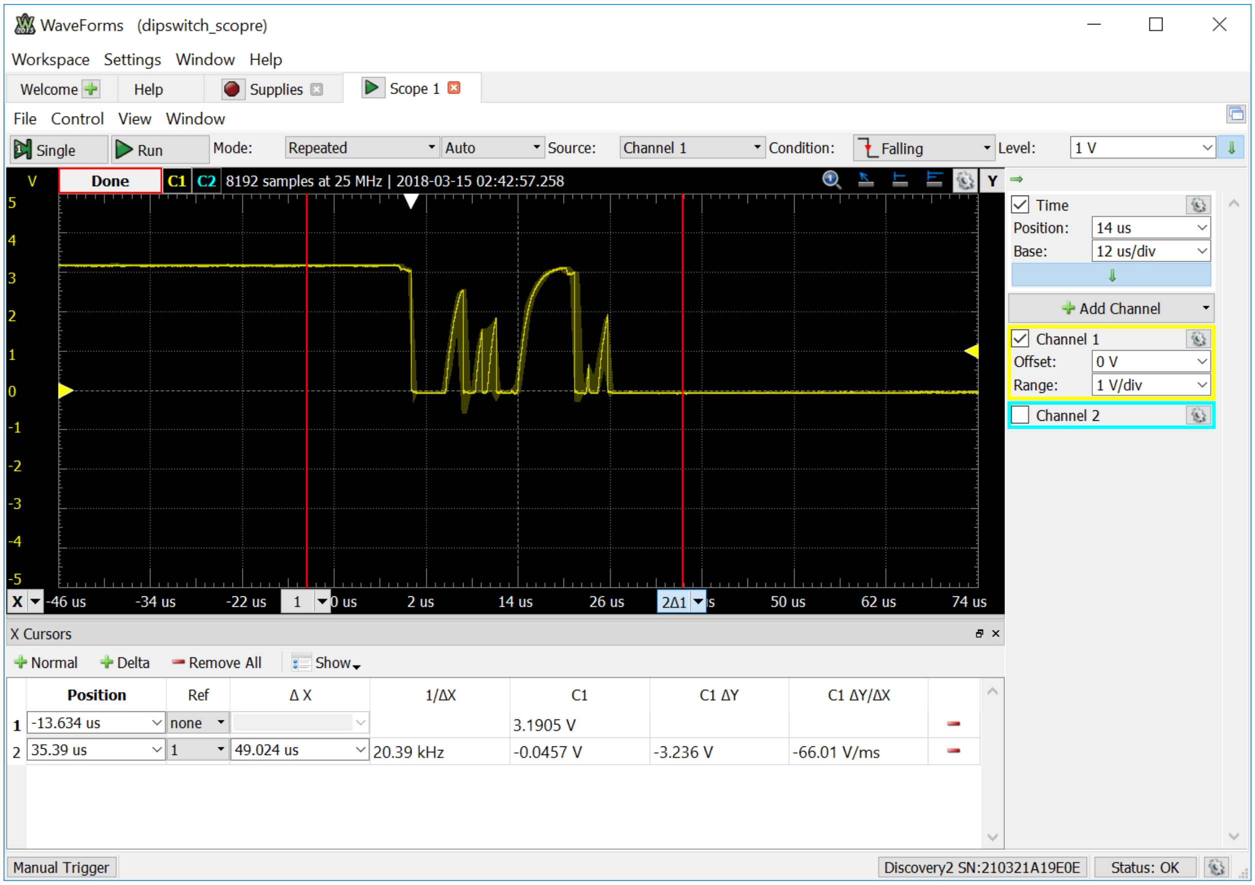
☑ Channel 1
Offset: 0 V
Range: 1 V/div

□ Channel 2

	Position	Ref	Δ X	1/ΔX	C1	C1 ΔY	C1 ΔY/ΔX
1	-484.76 us	none			-0.0199 V		
2	918 us	1	1.4028 ms	712.8 Hz	3.1822 V	3.202 V	2.282 V/ms

Discovery2 SN:210321A19E0E
Status: OK

Figure 6: Rising Edge Trigger of Tactile Switch Screenshot 2 (Switch Release)



Mode: Repeated Auto Source: Channel 1 Condition: Falling Level: 1 V

8192 samples at 25 MHz | 2018-03-15 02:42:57.258

☑ Time
Position: 14 us
Base: 12 us/div

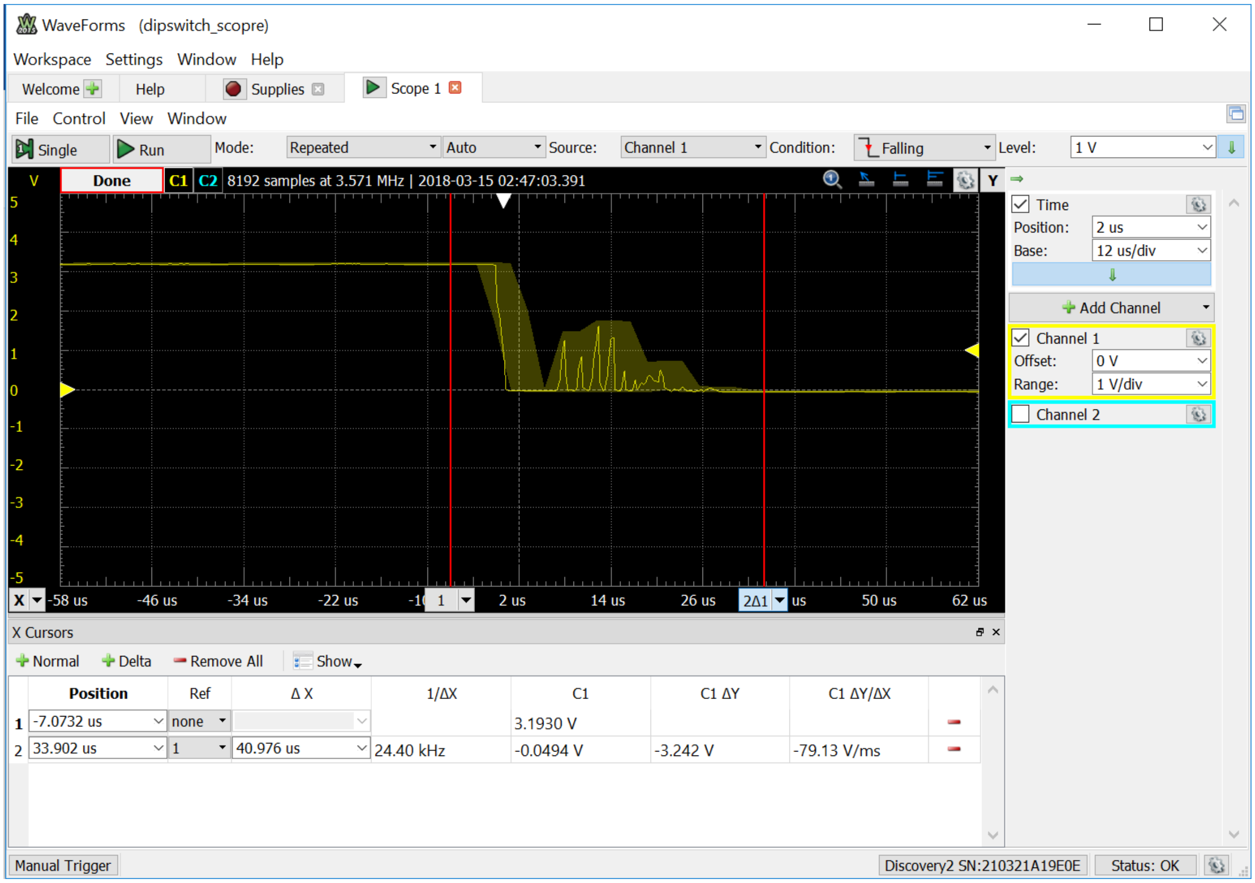
☑ Channel 1
Offset: 0 V
Range: 1 V/div

☐ Channel 2

	Position	Ref	Δ X	1/ΔX	C1	C1 ΔY	C1 ΔY/ΔX
1	-13.634 us	none			3.1905 V		
2	35.39 us	1	49.024 us	20.39 kHz	-0.0457 V	-3.236 V	-66.01 V/ms

Discovery2 SN:210321A19E0E Status: OK

Figure 7: Falling Edge Trigger of Tactile Switch Screenshot 1 (Switch Press)



Mode: Repeated | Auto | Source: Channel 1 | Condition: Falling | Level: 1 V

8192 samples at 3.571 MHz | 2018-03-15 02:47:03.391

☑ Time
Position: 2 us
Base: 12 us/div

☑ Channel 1
Offset: 0 V
Range: 1 V/div

☐ Channel 2

	Position	Ref	Δ X	1/ΔX	C1	C1 ΔY	C1 ΔY/ΔX
1	-7.0732 us	none			3.1930 V		
2	33.902 us	1	40.976 us	24.40 kHz	-0.0494 V	-3.242 V	-79.13 V/ms

Discovery2 SN:210321A19E0E
Status: OK

Figure 8: Falling Edge Trigger of Tactile Switch Screenshot 2 (Switch Press)