Lab 4 Backup Code

Part A

.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

MAIN:

ldi r17, 0x00 ;setting for 32MHZ subroutine

rcall CLK

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

sts PORTD\_DIRSET, r16 ;set blue LED to be output

ldi r16, 0b00000100 ;use TC0C to move location of OC02 from Px2 and Px6

sts PORTD\_REMAP, r16

ldi r16, 0xFF ;RGB color for 0-255

sts TCD0\_PER, r16

ldi r16, 0x00

sts TCD0\_PER+1, r16

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

sts TCD0\_CTRLA, r16

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

sts TCD0\_CTRLB, r16

ldi r16, 0x0E

sts TCD0\_CCC, r16

ldi r16, 0x00

sts TCD0\_CCC+1, r16 ;setting the compare value to 0x000E

ldi r16, 0b01000000

sts PORTD\_PIN6CTRL, r16

DONE:

rjmp DONE

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

.include "ATxmega128A1Udef.inc" ;include the file

.list ;list it

.org 0x0000 ;start our program here

rjmp MAIN ;jump to main

.org PORTF\_INT0\_vect

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, 0xFF

sts PORTD\_DIRSET, r16 ;set the RGB LED to be output

sts PORTD\_OUT, r16 ;turn the RGB LED off

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

LOOP:

sts PORTD\_OUTTGL, r16

rjmp LOOP

ISR:

push r19

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 ;clear the interrupt flag

sts PORTF\_INTFLAGS, r19

pop r18

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

.include "ATxmega128A1Udef.inc" ;include the file

.list ;list it

.org 0x0000 ;start our program here

rjmp MAIN ;jump to main

.org PORTF\_INT0\_vect

rjmp ISR

.org TCD0\_OVF\_vect

rjmp TIMER\_ISR

.equ stack\_init=0x3FFF ;initialize stack pointer

.equ debounce\_timer= (32000000\*.005)/1024

.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, 0xFF

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

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

lds r18, CPU\_SREG

push r18

push r16

\*insert code to SET CNT to zero here

ldi r16, low(debounce\_timer) ;set the timer for 5ms to debounce

sts TCD0\_PER, r16

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 r18

sts CPU\_SREG, r18

pop r18

reti

TIMER\_ISR:

push r18

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 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**

.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 blink\_timer =(32000000\*.1)/1024

.org 0x100

Table :.db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

Table1:.db 0x9A, 0x2C, 0x8A, 0x07, 0xFF, 0x49

Table2:.db 0x1F, 0x1F, 0xC2, 0x0D, 0x8D, 0x3C

Table3:.db 0x16, 0x46, 0xFA, 0xA5, 0x21, 0x00

.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 ;RGB color for 0-255

sts TCD0\_PER, r16

ldi r16, 0x00

sts TCD0\_PER+1, r16

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

sts TCD0\_CCC+1, r16 ;blue

sts TCD0\_CCB, r16

sts TCD0\_CCB+1, r16 ;green

sts TCD0\_CCA, r16

sts TCD0\_CCA+1, r16 ;red

ldi r16, 0b01000000 ;invert the signal because they are low-true LED

sts PORTD\_PIN6CTRL, r16

sts PORTD\_PIN5CTRL, r16

sts PORTD\_PIN4CTRL, r16

pop r16

ret

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

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

EDGE\_ISR: ;initialization for edge interrupt

push r18

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

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 r18

sts CPU\_SREG, r18

pop r18

reti

DEBOUNCE\_TIMER\_ISR: ;initialization for the 5ms timer used for debounce

push r18

lds r18, CPU\_SREG

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

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

sts CPU\_SREG, r18

pop r18

reti

BLINK\_ISR:

push ZL ;pushes

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

cpi r17, 1

breq STORE1

cpi r17, 2

breq STORE2

cpi r17, 3

breq STORE3

STORE: ;load corresponding tables

ldi ZL, low(Table<<1)

ldi ZH, high(Table<<1)

rjmp CHECK

STORE1:

ldi ZL, low(Table1<<1) ;load corresponding tables

ldi ZH, high(Table1<<1)

rjmp CHECK

STORE2:

ldi ZL, low(Table2<<1) ;load corresponding tables

ldi ZH, high(Table2<<1)

rjmp CHECK

STORE3:

ldi ZL, low(Table3<<1) ;load corresponding tables

ldi ZH, high(Table3<<1)

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

ldi r16, 0x00

sts TCD0\_CCC+1, r16 ;blue

lpm r16, Z+ ;load corresponding table values for each capture

sts TCD0\_CCB, r16

ldi r16, 0x00

sts TCD0\_CCB+1, r16 ;green

lpm r16, Z+ ;load corresponding table values for each capture

sts TCD0\_CCA, r16

ldi r16, 0x00

sts TCD0\_CCA+1, r16 ;red

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

cpi r17, 4

brsh RESET

brne SKIP

RESET:

ldi r17, 0 ;when counter half reach 4. start over

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

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