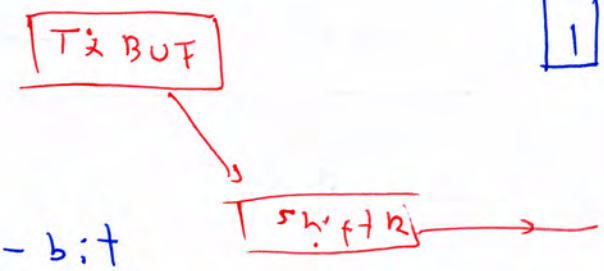
SPI Registers



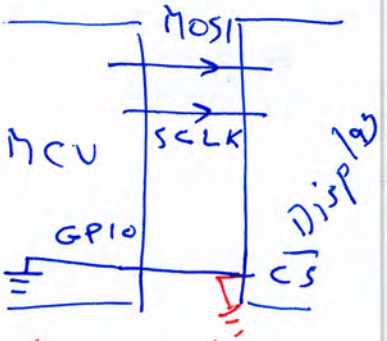
8-bit

TX BUFFER
RX BUFFER

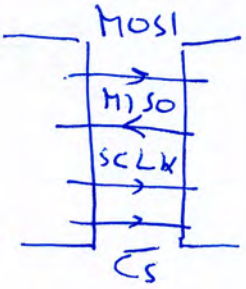
1-bit

BUSY → in CTL

```
void spi_write_byte(
    unsigned char data) {
  // wait for BUSY bit to clear
  while ((CTL & BUSY) != 0) { }
  TXBUFFER = data; // Transmission begins!
  return;
}
```



⊛ Full-duplex exchange




```
void spi_exchange(unsigned char *
    data_rx, unsigned char data_tx) {
  while ((CTL & BUSY) != 0) { }
  TXBUFFER = data_tx;
  // wait for our communication to finish
  while ( (CTL & BUSY) != 0) { }
  *data_rx = RXBUFFER;
  return;
}
```

UART vs. I2C vs. SPI

- **Data Rate:** in eUSCI

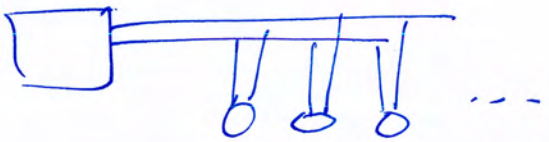
UART	100 kHz
I2C	400 kHz (Fast Mode)
SPI	16 MHz

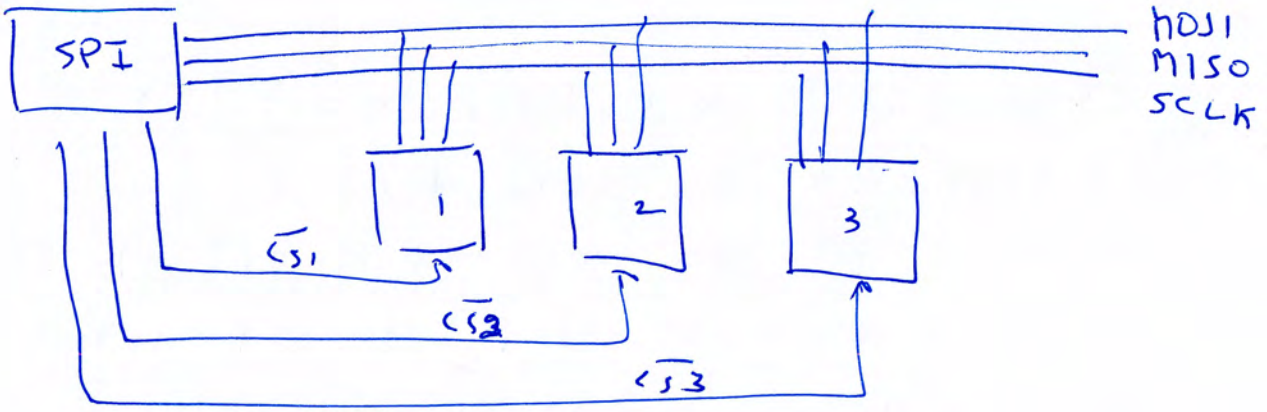
 # pins

UART	1, 2
I2C	2
SPI	4, 3, 2, ≥4

Extensibility

I2C





- Simplicity : SPI
- Support by devices : I2C, SPI
- Full-duplex capability: UART, SPI.

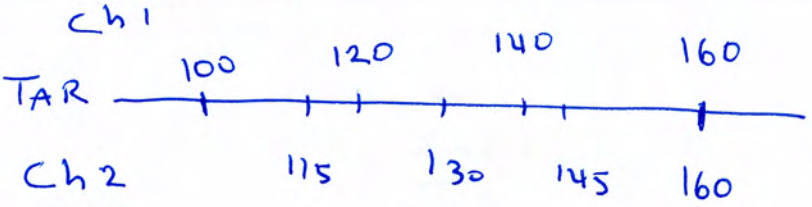
. Quad SPI (4 data wires)
  ↳ SD cards

. CANBUS

. I2S (Audio)
  ↳ Microphones, Speakers

Using multiple channels

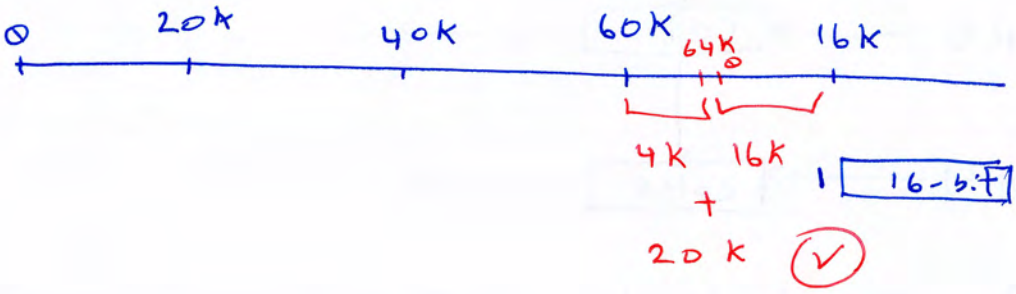
20x



15x

Interrupt every 20K cycles w/ Ch2

└ 1024

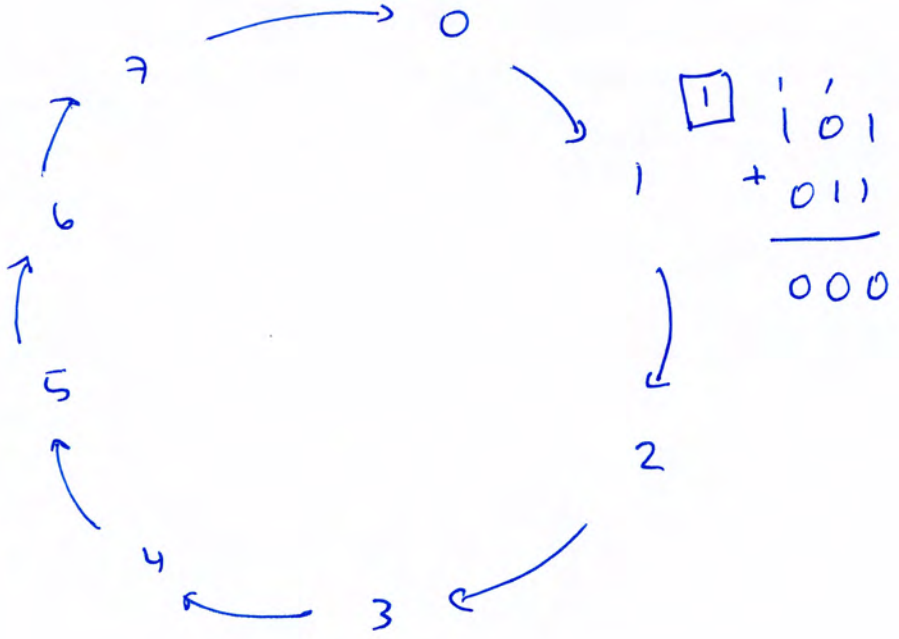


16-bit

(TACCR2) (on 16-bit)

60K
+20K
80K
-64K
16K

E.g. TAR is 3-bit



1

101
+011
000

Interrupt every 3x

2

2 + 3 = 5

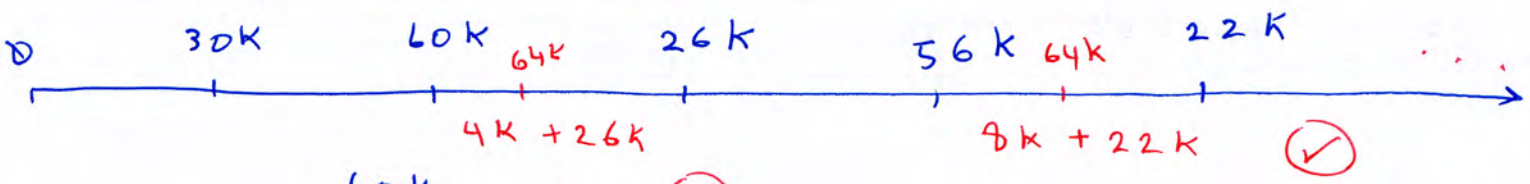
5 + 3 = 8 - 8 = 0

0 + 3 = 3

3 + 3 = 6

6 + 3 = 9 - 8 = 1

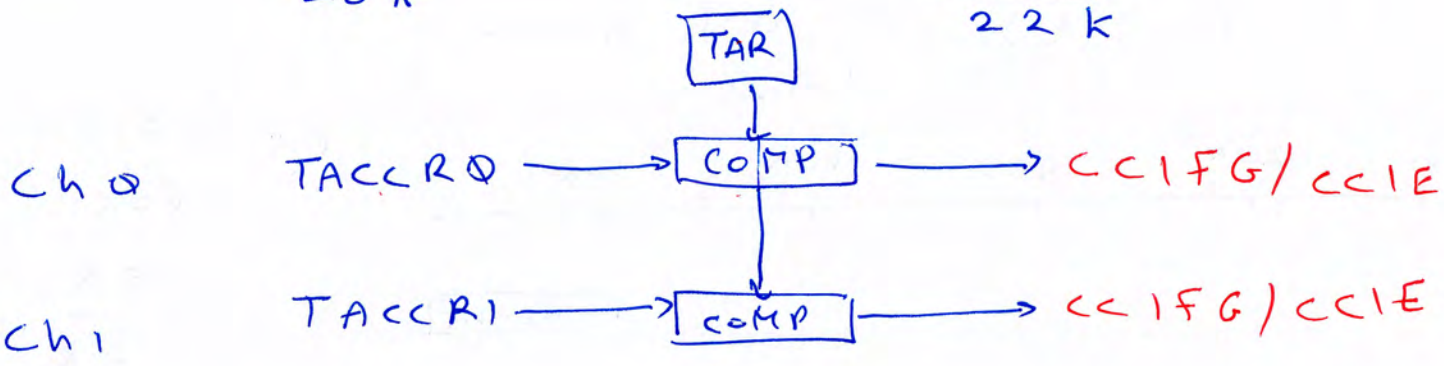
30k periodic interrupts



60k
+ 30k
90k
- 64k
26k

30k

56k
+ 30k
86k
- 64k
22k



① Use Timer_A, Cont. Mode, ACLK = VLO = 10 kHz (10,000)

Use Ch0 to flash the Red LED every 0.75 s

— — Ch1 - - - - - Green — — · — - 0.3 s

3,000 7,500

Ch0: CCIE / CCIFG → TACCTL0 [A0]

Ch1: CCIE / CCIFG → TACCTL1 [A1]

```
#define  Red   BIT0  // P1.0
#define  Green BIT6  // P1.6
;
```

Main

```
// Code: ACLK ← VLO
P1DIR |= Red | Green;   // Output
// Config Ch0 interrupt
TACCR0 = (7500 - 1);
TACCTL0 &= ~CCIFG;   // clear the flag
TACCTL0 |= CCIE;
// Config. Ch1 interrupt
TACCR1 = (3000 - 1);
TACCTL1 &= ~CCIFG;
TACCTL1 |= CCIE;
```

Cont.

```
TACTL = TASSEL_1 | ID_0 | MC_2 | TACLR;
__low_power_mode_3();  // Sets GIE
while(1) { }
```

Main ends

```
#pragma vector = TIMER0_A0_VECTOR
__interrupt void TA0_A0_ISR() {
    P1OUT ^= Red;
    TACCR0 += 7500;
    ↳ // Schedule the next interrupt
    // H.W clears the flag
    return;
}
```

TACCR0
CCIFG
CCIE

vs.

```
TACCR0 = TAR + 7500;
```


```
= TIMER_A1_VECTOR
void TA0_A1_ISR() {
    // Detect Ch1 flag
    if ((TACCTL1 & CCIFG) != 0) {
        P1OUT ^= Green;
        TACCR1 += 3000;
        ↳ // Next interrupt
        TACCTL1 &= ~CCIFG;
    }
    return;
}
```

Can be removed

Since only Ch1 interrupt was enabled.

Rollback	Ch1	Ch2
X	✓	X

Ch 0 flashes the Red LED every .75 s (Green OFF)

Ch 1 flashes the Green LED every 0.3 s (Red OFF)

Repeat

status = 0 — 4 sec — Ch 2

status = 1 — 4 sec

Disable ch 0 interrupt (CC1E ← 0)

Reenable Ch 1 interrupt

Disable Ch 1 int.

Reenable Ch 0 int.

Ch 0 : 7,500 — A0

Ch 1 : 3,000 — A1

Ch 2 : 40,000 — A1

ACLK = 10 kHz

10,000

Main

```
P1DIR |= Red | Green ;

// config Ch0
TACCR0 = (7500 -1);
TACCTL0 &= ~CCIFG;
TACCTL0 |= CCIE;

// Config Ch2
TACCR2 = (40000 -1);
TACCTL2 &= ~CCIFG;
TACCTL2 |= CCIE;

TACTL = TASSEL_1 | ID_0 | MC_2 | TACLR;
__low_power_mode_3();
while (1) {}
```

Cont.

Main ends.

AO

```
void TA0_A0_ISR() {
    P1OUT ^= Red;
    TACCR0 += 7500;
    // H.W. clears the flag
}
```

TACCR0
CCIFG
CCIE

AI

```
void TA0_A1_ISR() {
    static int status = 0;
    // Detect Ch1 flag
    if ((TACCTL1 & CCIFG) != 0) {
        P1OUT ^= Green;
        TACCR1 += 3000;
        TACCTL1 &= ~CCIFG;
    }
```

```
// Detect Ch2 flag
if( (TACCTL2 & CCIFG) != 0 ) {
    if( status == 0 ) {
        P1OUT &= ~Red; // OFF
        // Disable Ch0 interrupt
        TACCTL0 &= ~CCIE;
        // Re-enable Ch1
        TACCR1 = TAR + 3000; // Relative to TAR
        TACCTL1 &= ~CCIFG;
        TACCTL1 |= CCIE;
        P1OUT ^= Green; // Optional
        status = 1;
    }
    else if( status == 1 ) {
        P1OUT &= ~Green;
        // Disable Ch1 interrupt
        TACCTL1 &= ~CCIE;
        // Re enable Ch0
        TACCR0 = TAR + 7500;
        TACCTL0 &= ~CCIFG;
        TACCTL0 |= CCIE;
        P1OUT ^= Red; // optional
        status = 0;
    }
    // Renew Ch2
    TACCR2 += 40000;
    TACCTL2 &= ~CCIFG; } return; }
```

TACCR0
CCIFG
CCIE

Example of last lecture

ch0, ch1, Ch2 interrupts

ch0 interrupts flashing the Red LED

ch1 interrupts flashing the Green LED

Ch2 — 4 s

Ch2 — 4 s

Ch0 & Ch1 interrupts are enabled / disabled ...

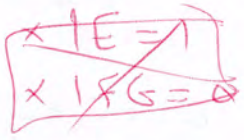
Ch2 interrupt is ON all the time.

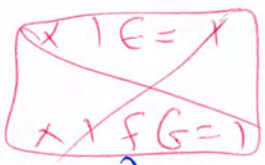
```
A0_ISR () {
    ;
}

A1_ISR () {
    // ch1
    if ( ch1 flag) {        AND ch1 is enabled
        ;
    }
    // ch2
    if ( ch2 flag ) {
        ;
    }
}
```

x1E=0, x1FG=1

x1E=1, x1FG=1

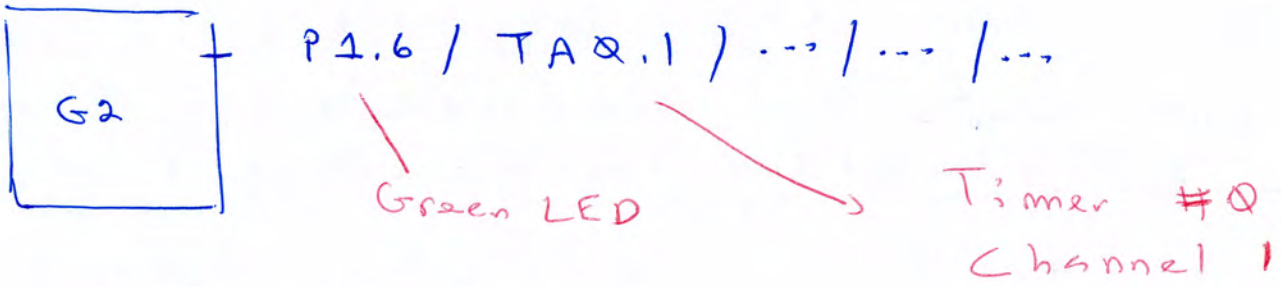




- Interrupt is cycled ON/OFF (enabled) in various stages of the code
  if(x1E=1 AND x1FG=1)
- Interrupt is ON (enabled) all the time
  if(x1FG=1)

Timer - Based Output

- Supported at pins described as TAx.y



Data Sheet

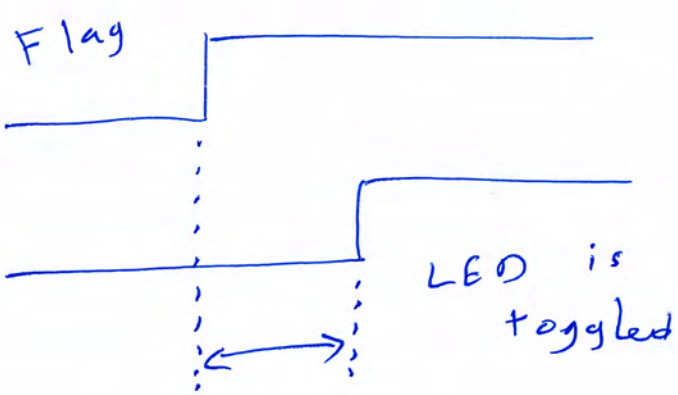
```
P1DIR = 1
P1SEL = 1
P1SEL2 = 0
```

```
// Config pin to TA0.1
P1DIR |= BIT6;
P1SEL |= BIT6;
P1SEL2 &= ~BIT6;
```

Latency

Timer - Based Output

Interrupt - Based



Timer Output Patterns

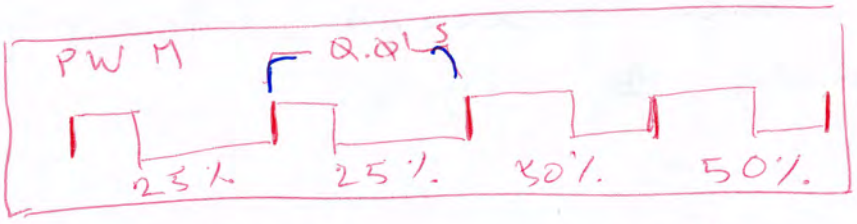
0 : None
1 : Set
2 : Toggle / Reset
3 : Set / Reset
4 : Toggle
5 : Reset
6 : Toggle / Set
7 : Reset / Set

(*) E.g. Set / Reset w/ Channel 1

Ch 1 Compare event (TAR = TACCR1)

Cont. Mode
Rollback event

Up Mode
Ch 0 compare event (TAR = TACCR0)



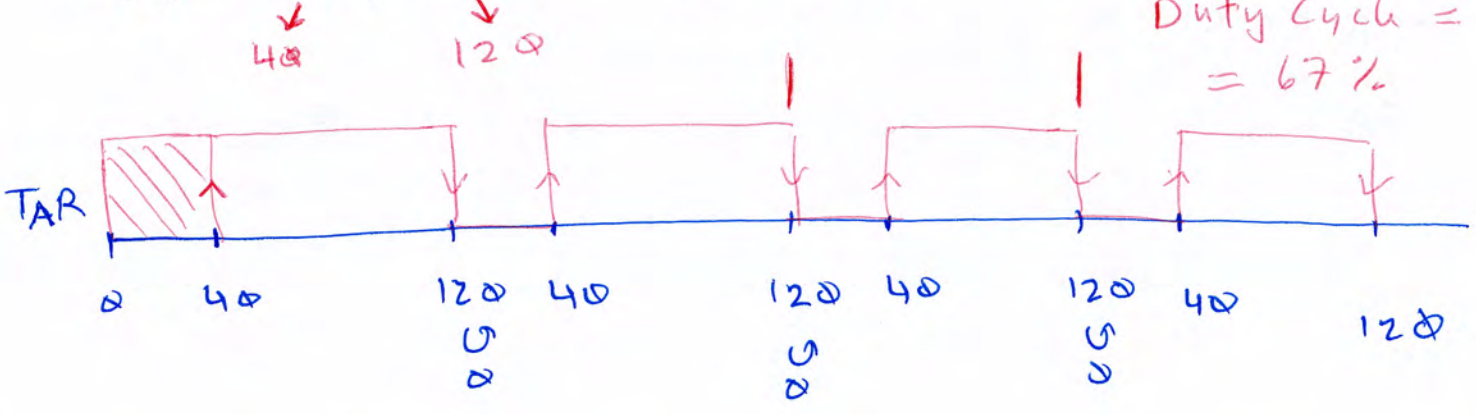
PWM Frequency = 100 Hz
↳ PWM Period = 1/100 s

E.g. PWM Frequency = 100 Hz; ACLK = VLO = 12 KHz (12,000). Up Mode.

Up Mode Cycles = $\frac{1}{100}$ s * 12,000 Hz = 120

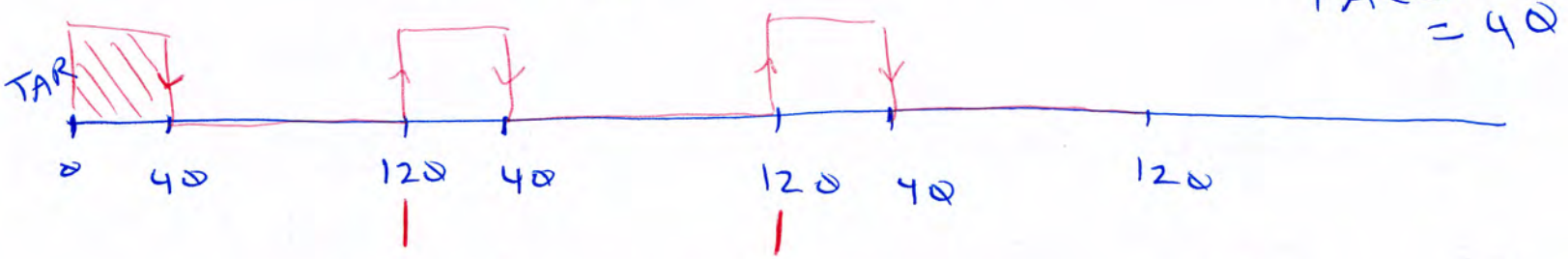
Use Set / Reset w/ Ch 1 TACCR1 = 40

Duty Cycle = $\frac{80}{120}$ = 67%



Redo w/ Reset / Set

Ch1 @ 40 → Ch0 @ 120



Duty cycle = $\frac{40}{120}$ = 33%

[20, 40, 60, 80] / 120

(*) Implement the PWM above.

Try these duty cycles (0, 20, 40, 60, 80) out of 120.

Use Timer #1 to raise 1-second interrupts & change the duty cycle.

Main

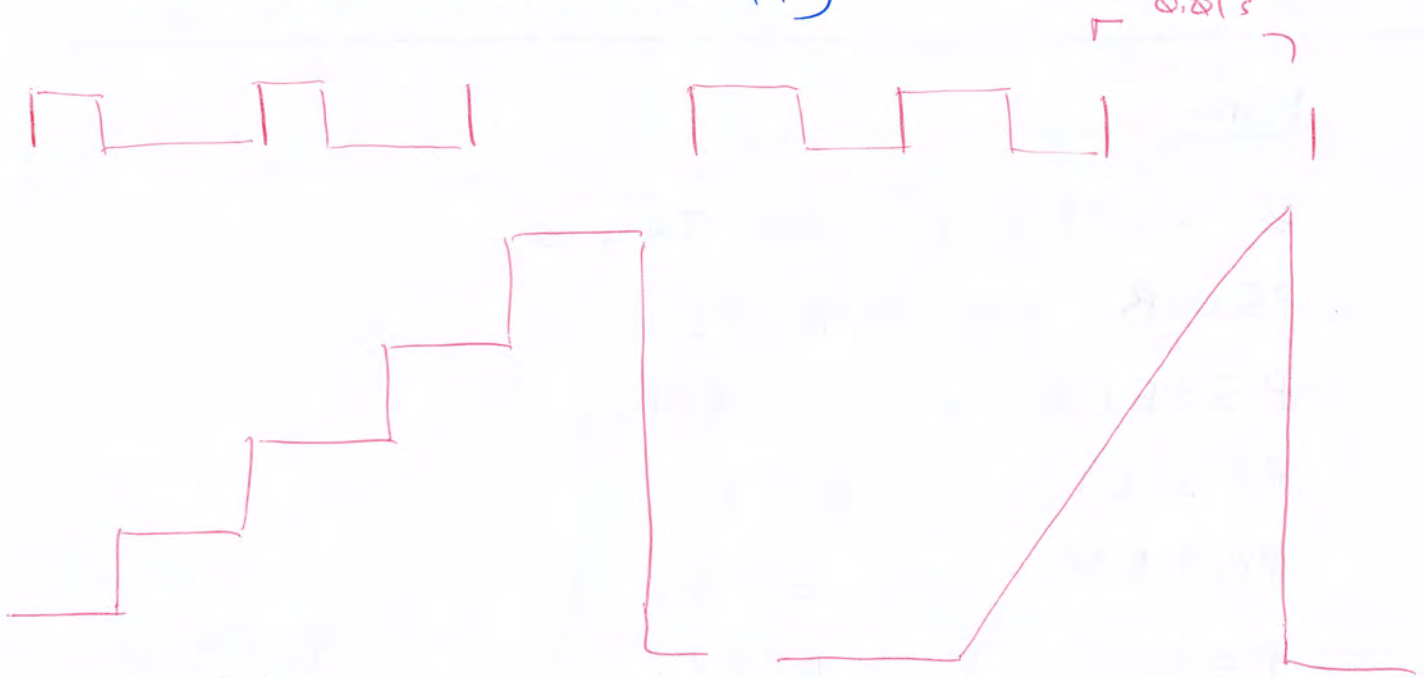
```
// config pin to TA0.1
(P1DIR ...   P1SEL ...   P1SEL2 ...)
// Timer #0 (PWM)
TA0CCR0 = 120;
TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
// Channel 1 : Reset / Set
TA0CCR1 = 20;
TA0CCTL1 |= OUTMOD_7; // Reset/Set
```

```
// Timer #1 , Up , 1-sec interrupt
TA1CCR0 = (12000 -1); // @ 12kHz
TA1CCTL0 &= ~CCIFG;
TA1CCTL0 |= CCIE;
TA1CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
- low-power, mode - 3 ( );
```

Main ends

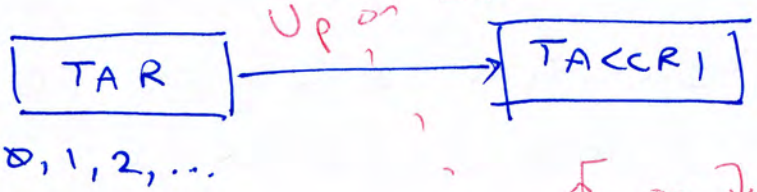
```
--- vector = TIMER1 - A0 - VECTOR
--- void TA1 - A0_ ISR () {
    if ( TA0CCR1 >= 80)
        TA0 CCR1 = 0;
    else TA0 CCR1 += 20;
    // H.W. clears the flag
}
```

1000 Hz
↳ 0.001 s



Timer Input Capture (Time stamp)

w/ Channel 1



or both on a pin.

Available at pins described as

TA x.y.

E.g. FR5994

Button

P5.6 / TA4.0 / ... / ...

↳ Timer #4 Channel 0

Data Sheet

P5.6 → TA4.CCI0A {→ A, → B}

P5DIR = 0 P5SEL0 = 0 P5SEL1 = 1

Main

= Config pin as TA4.0

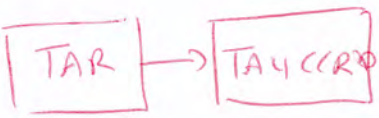
```
P5DIR  &= ~BIT6;
P5SEL0 &= ~BIT6;
P5SEL1 |= BIT6;
P5REN  |= BIT6;
P5OUT  |= BIT6;
```

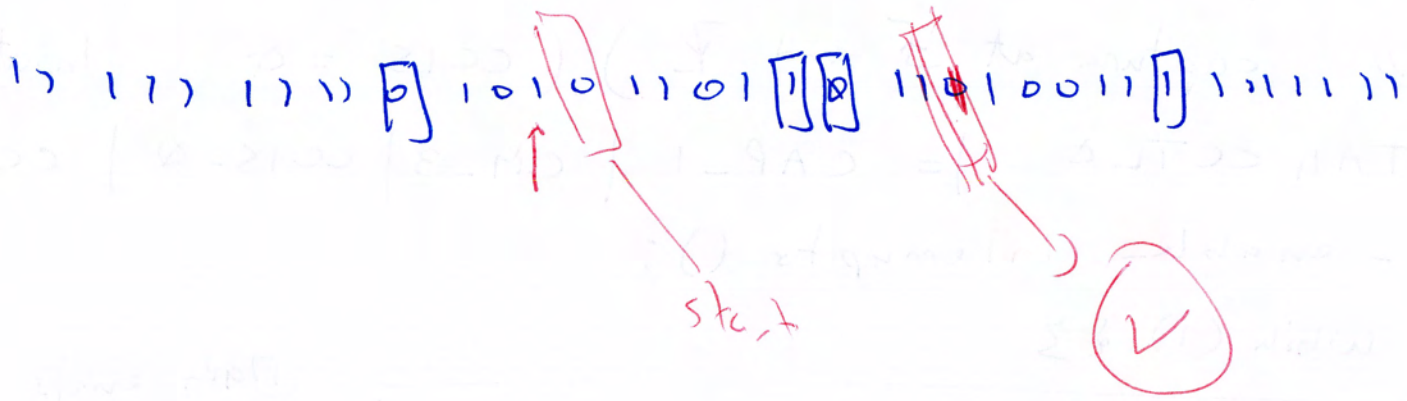
} Pull-up resistor

```
// Timer #4 , Cont. ,  ACLK = 32 kHz
TA4CTL = TASSEL_1 | ID_0 | MC_2 | TACLR;
// Channel 0  Capture Mode
// ( CAP=1  Capture Mode) ( CM = 3
// capture at ↓ and ↑ ) ( CCIS = 0  Input A)
TA4CCTL0 |= CAP_1 | CM_3 | CCIS_0 | CCIE;
_enable_interrupts ();
while (1) { }
```

Main ends

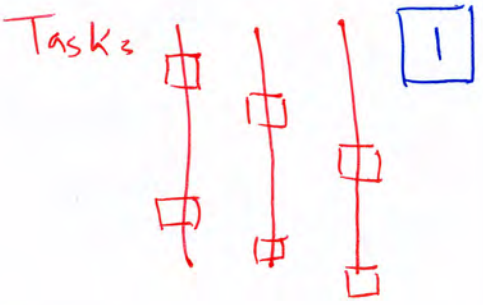
```
.... vector = TIMER4_A0_VECTOR
.... void TA4_A0_ISR () {
      unsigned int temp;
      temp = TA4CCR0;
      uart_write_uint16 ( temp);
      __delay_cycles (1500); // @ 1MHz → 15 ms
      // HW clears the flag at start of ISR
      // Clear it again
      TA4CCTL0 &= ~CCIFG;
      return;
}
```




state
V

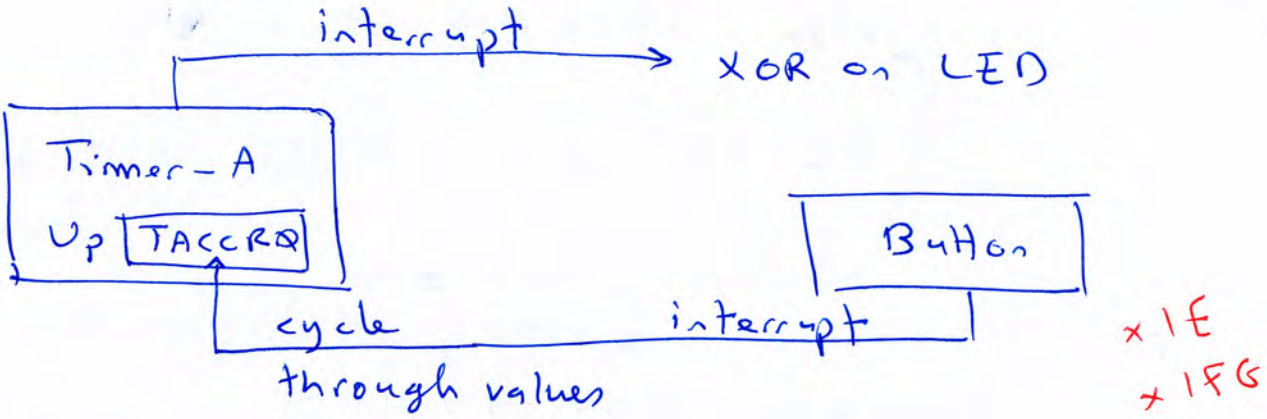
EEL 4742C - 04/03/2024

Concurrent Events Handling



* Timer, 32 kHz, Up Mode,
  Interrupts → Toggle the LED.

Push button, interrupts, change the timer's period. (4000 → 5000 → 6000 → ... → 11000)



Main → initial state

```
P1DIR |= LED;  // Output
// Config. button interrupts (6x)
P1DIR ← 0; P1REN ← 1; P1OUT ← 1;
P1IES ← 1; P1IFG ← 0; P1IE ← 1;

// Config Ch0 interrupt
TACCR0 = 4000;
TACCTL0 &= ~CCIFG;
TACCTL0 |= CCIE;

TACTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
__low_power_mode_3(); // Sets GIE
```

main ends

```
void TA0_A0_ISR () {
    P1OUT ^= LED;
    // HW clears the flag
}

void Port1_ISR () {
    if ((P1IFG & BUT) != 0) {
        if ( TACCR0 >= 11000)
            TACCR0 = 4000;
        else TACCR0 += 1000;
        -delay_cycles (15000); // 15 ms
        P1IFG &= ~BUT; // clear button's flag
    }
}
```

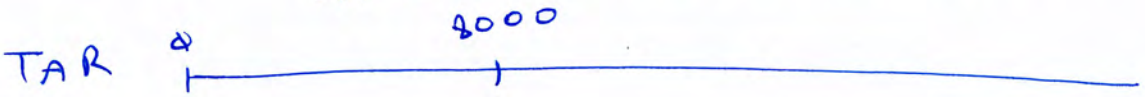
TACCR0
CCIE
CCIFG

@ 1 MHz

```
{ TAR = 0;
  TACTL |= TACLR;
```

TACCR0 = 4000 → 5006

TACCR0 = 11000 → 4000

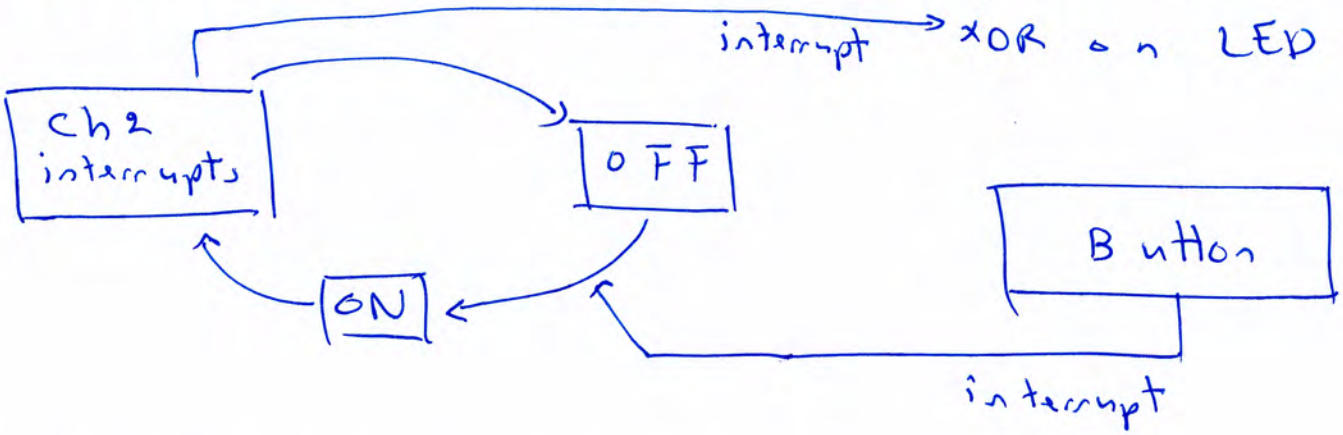


⊛ Flash light functionality.
Button interrupt cycles the LED through:

STATUS

→ OFF ⟶ ON ⟶ Flashing (1/4 sec)
0 1 2

→ keep the timer running all the time in Cont. Mode.
→ Use Channel 2 to time the 1/4 sec interval.



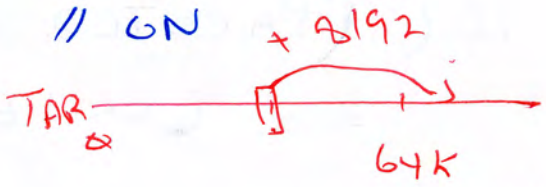

```
int status = 0 ;   // Global
```

Main

```
P1DIR |= LED;
P1OUT &= ~LED; // OFF (status = 0)
// Config button interrupts
...
TACTL = TASSEL-1 | ID_0 | MC-2 | TACLR;
-low-power-mode-3();
```

Main ends

```
void Port1_ISR() {
    if (status == 0) {
        P1OUT |= LED;  // ON
        status = 1;
    }
    else if (status == 1) {
        // Schedule Ch2 interrupt in 1/4s (8192 cycles)
        TACCR2 = TAR + 8192;
        TACCTL2 &= ~CCIFG;
        TACCTL2 |= CCIE;
        P1OUT ^= LED;  // Optional
        status = 2;
    }
    else {
        P1OUT &= ~LED;
        // Disable Ch2 interrupt
        TACCTL2 &= ~CCIE;
        status = 0;
    }
    __delay_cycles(15000);  ---> Rebounce
    P1IFG &= ~BUT;
}
```



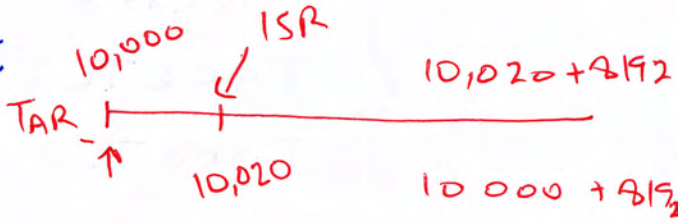
32 kHz

A1 → A2 → Rollback

```
void TA0_A1_ISR () {
  // Detect channel 2 interrupt
  if ( (TACCTL2 & (CCIE | CCIFG))
       == (CCIE | CCIFG)) {
    P1OUT ^= LED;
    TACCR2 += 8192;
    TACCR2 = TAR + 8192;
    TACCTL2 &= ~CCIFG;
  }
}
```

CCIE = 0
CCIFG = 1

TACCR2
CCIE
CCIFG

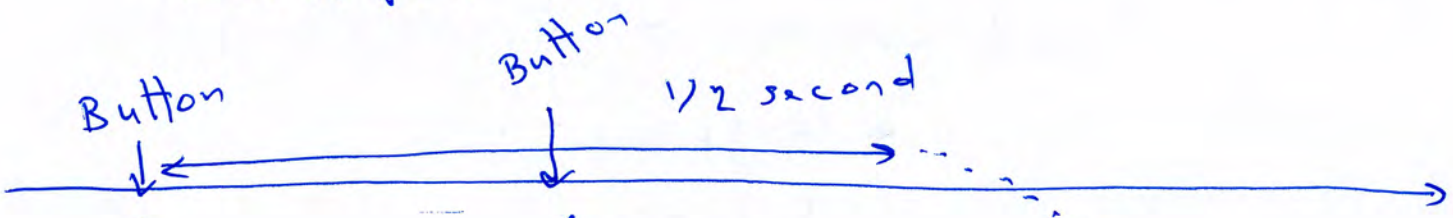


⊛ Double-click on button toggle the LED.

↳ Two clicks within 1/2 s (32kHz) [16384 cycles]

→ Keep the timer running in Cont. Mode

→ Use Channel 1 to time 1/2 second interval w/ interrupt.



if status = 0

- Time 1/2 interrupt w/ Ch1
- status = 1

if status = 1

- Double click! Do action.
- Cancel Ch1 interrupt
- status = 0

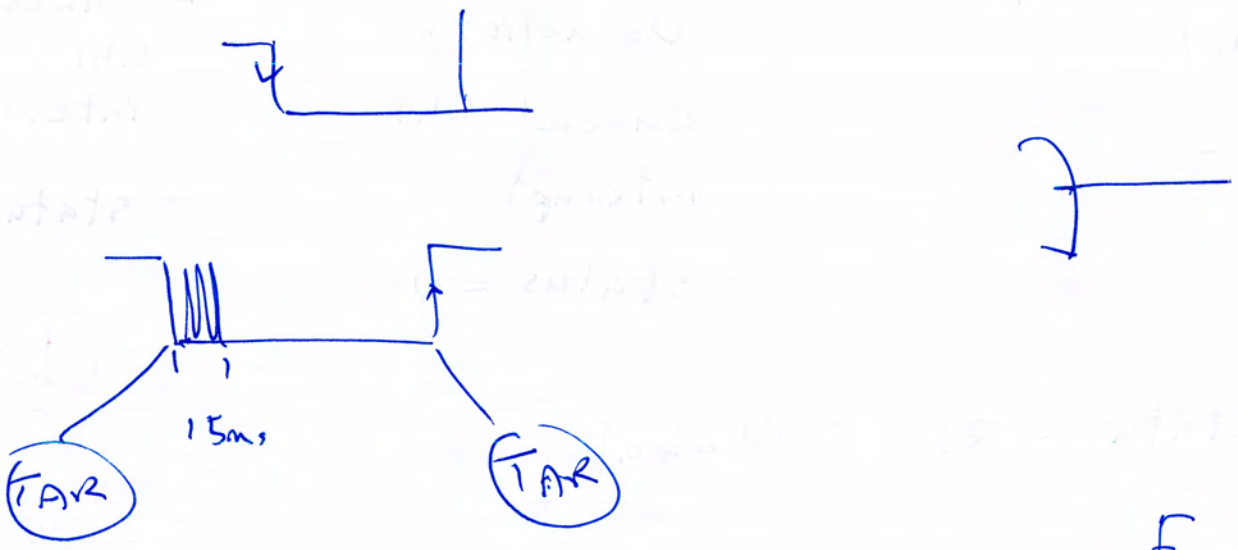
- Cancel Ch1 interrupt
- status = 0

```
int status = 0;  // Global

Main
P1DIR |= LED;  // Output
// Config button interrupt
....
TACTL = TASSEL-1 | ID-0 | MC-2 | TACLR;
- low-power-mode-3();
```

Main ends

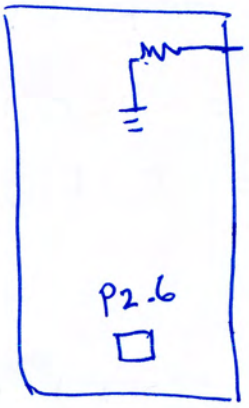
```
longpress = 0;
counter = 0;
while ( Button is down) {
    counter ++;
    if ( counter > 40000) {
        longpress = 1;
        break;
    }
}
```



```
void Port1_ISR() {
    if( status == 0) {
        TACCR1 = TAR + 16384; // 1/2 s
        TACCTL1 &= ~CCIFG;
        TACCTL1 |= CCIE;
        status = 1;
    }
    else if ( status == 1) {
        // Double click!
        P1OUT ^= LED;
        // cancel ch1 interrupt
        TACCTL1 &= ~CCIE;
        status = 0;
    }
    - delay - cycles (15,000);
    P1IFG &= ~BUT;
}
void TA0_A1_ISR() {
    if( (TACCTL1 & (CCIFG | CCIE))
        == (CCIE | CCIFG)) {
        TACCTL1 &= ~CCIE;
        status = 0;
    }
}
```

Software interrupt

Software Interrupt



Port 2

7 0

```
P2IFG &= ~BIT5;
P2IE  |= BIT5;
P2REN |= BIT5;
P2OUT &= ~BIT5
```

To trigger a software interrupt...

```
P2IFG |= BIT5;
```