Line Tracer 08

- Interrupt -

1. Interrupt

How to Handle Hardware Events

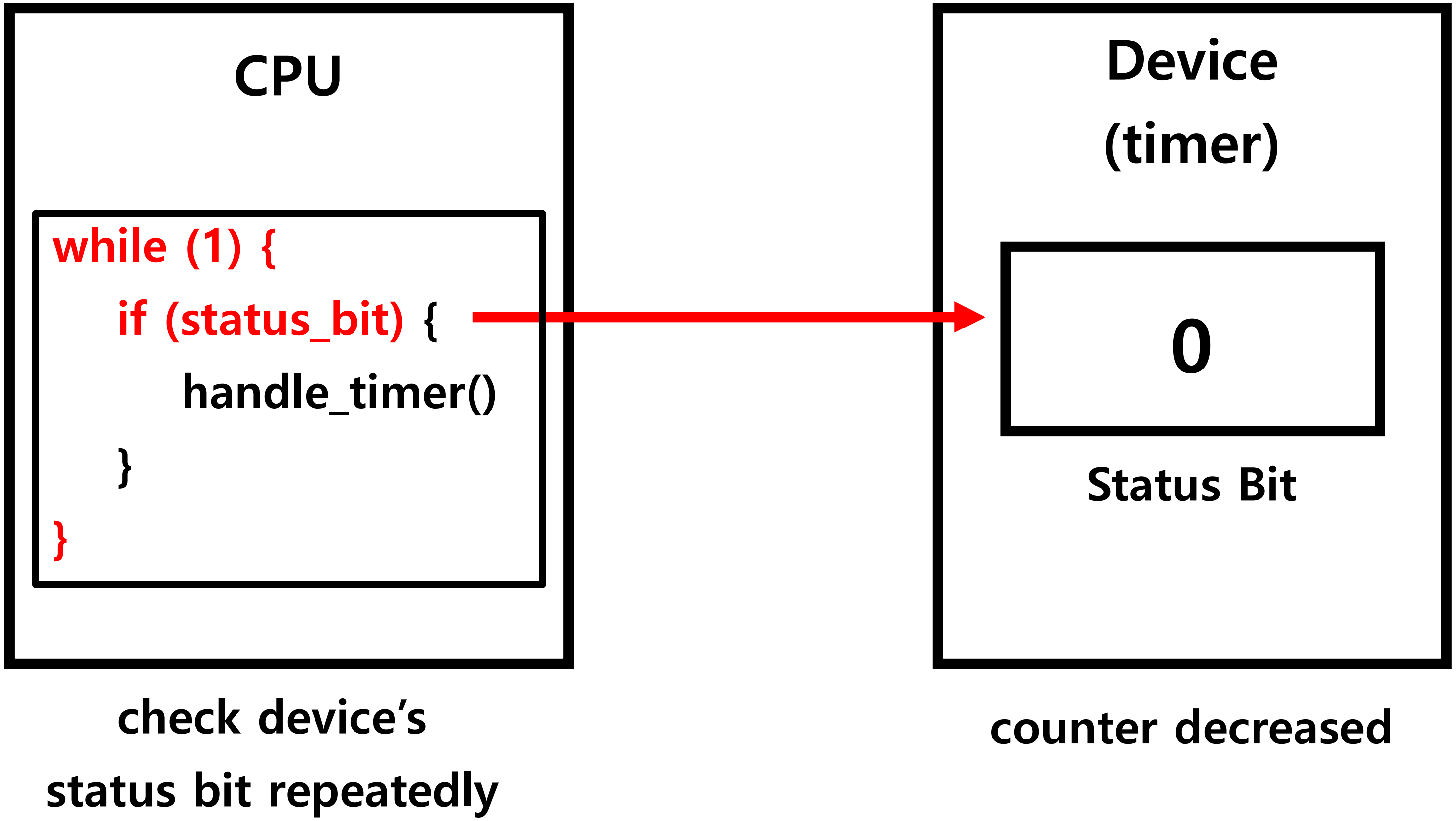
Polling

- CPU (Program) steadily checks whether event occurs
- Not hardware mechanism

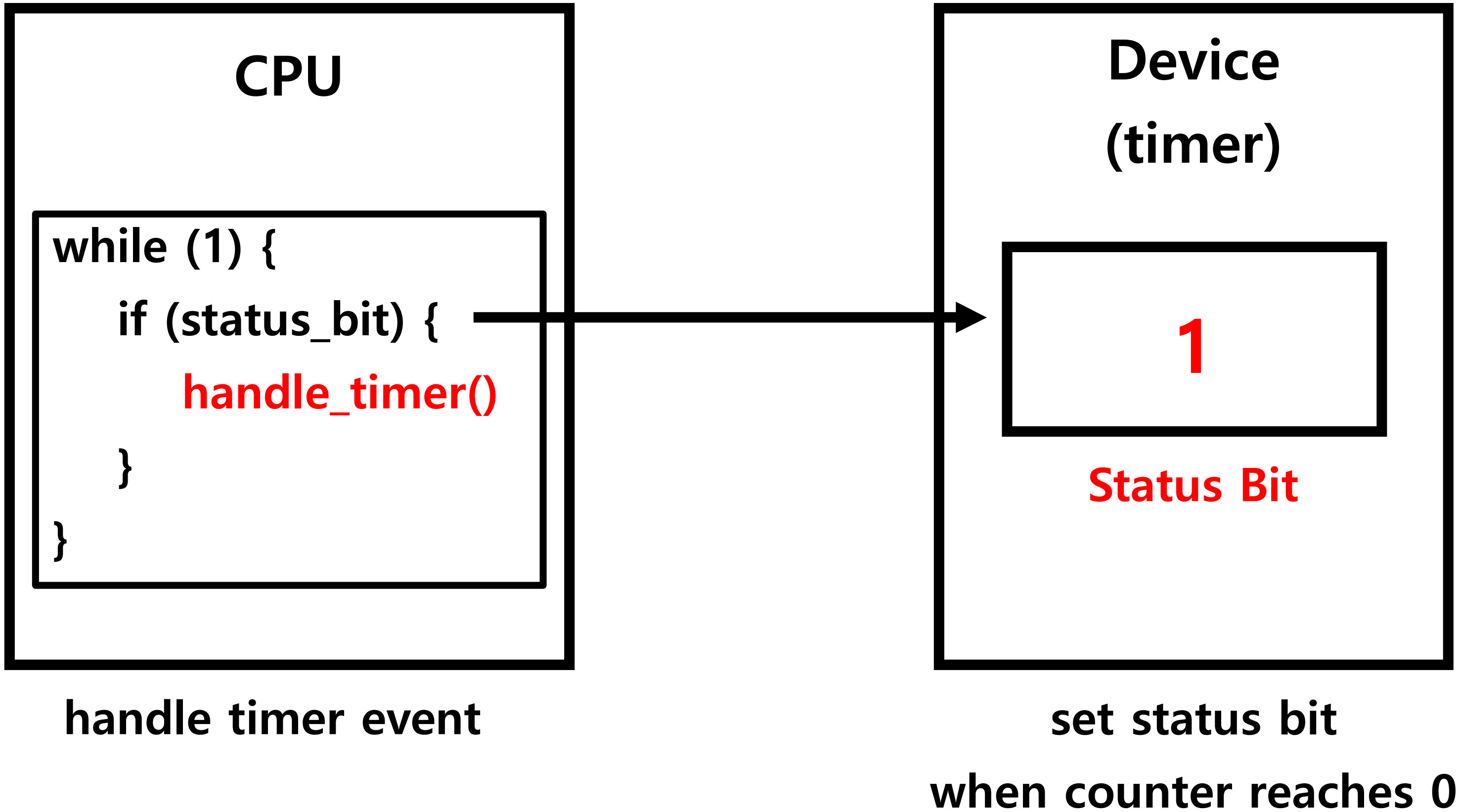
Interrupt

- Devices notice the CPU when event occurs
- Hardware mechanism

Polling


CPU
while (1) {
if (status_bit) {
handle_timer()
}
}
Device
(timer)
0
Status Bit
check device's
status bit repeatedly
counter decreased

Polling

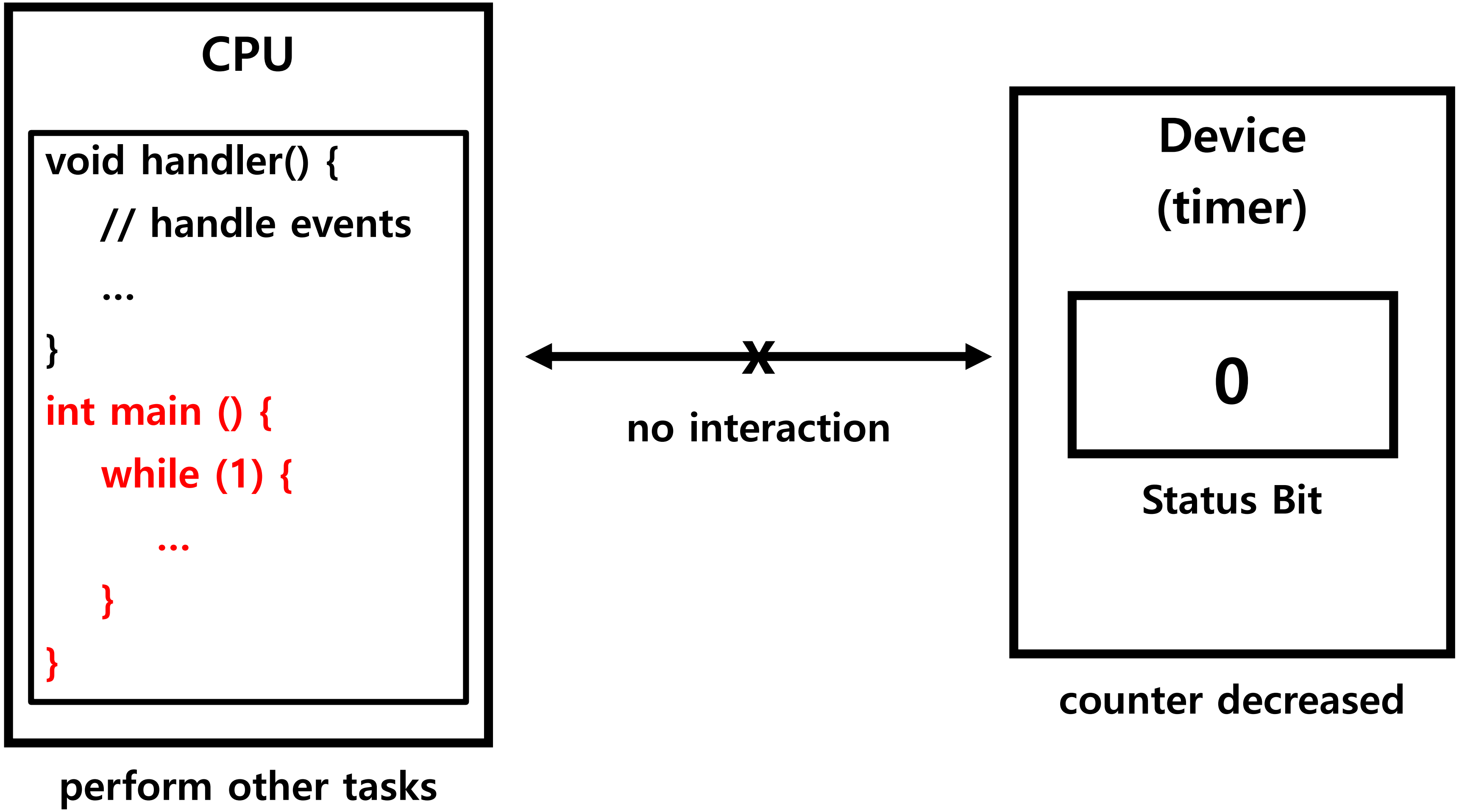

CPU
while (1) {
if (status_bit) {
handle_timer()
}
}
handle timer event
Device
(timer)
1
Status Bit
set status bit
when counter reaches 0

Polling Example

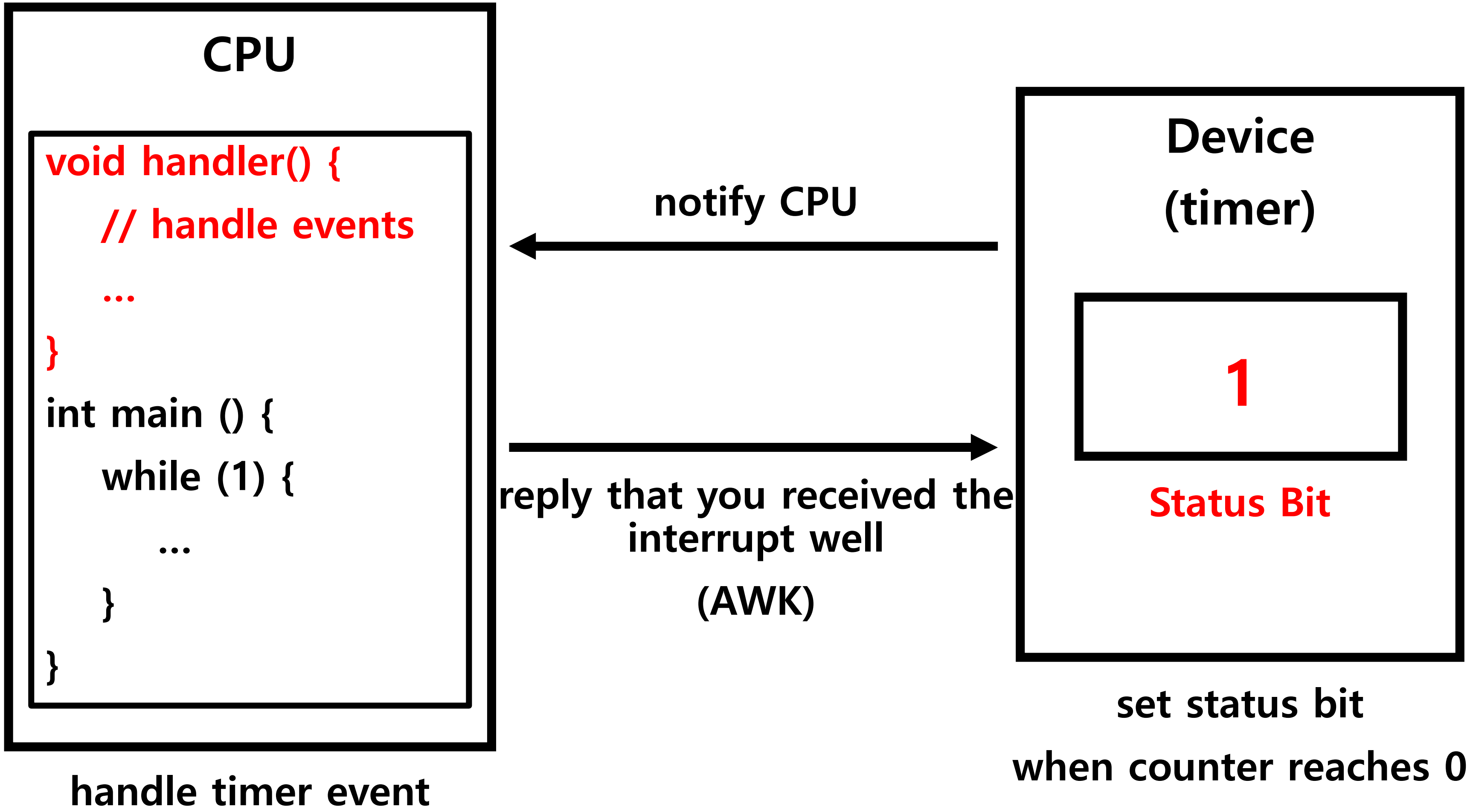
```c
void main(void)
{
    int sw1;

    // Initialization
    Clock_Init48MHz();
    led_init();
    switch_init();

    while (1) {
        sw1 = P1->IN & 0x02;
        if (!sw1) {
            printf("Pressed!\n");
        }
        Clock_Delay1ms(100);
    }
}
```

CPU checks the status of the switch periodically

Interrupt


CPU
void handler() {
// handle events
...
}
int main () {
while (1) {
...
}
}
perform other tasks
no interaction
Device
(timer)
0
Status Bit
counter decreased

Interrupt


CPU
void handler() {
// handle events
...
}
int main () {
while (1) {
...
}
}
handle timer event
notify CPU
reply that you received the interrupt well
(AWK)
Device
(timer)
1
Status Bit
set status bit
when counter reaches 0

Interrupt Handling

```
int main () {
    int a, b = 0
    while (1) {
        a = 1
        b += 1
    }
}
```

main routine

```
void handler() {
    // handle events
    ...
}
```

interrupt handling routine

when interrupt occurs while executing a= 1,

save main routine’s state and jump to the handler

Interrupt Handling

```
int main () {
    int a, b = 0
    while (1) {
        a = 1
        b += 1
    }
}
```

main routine

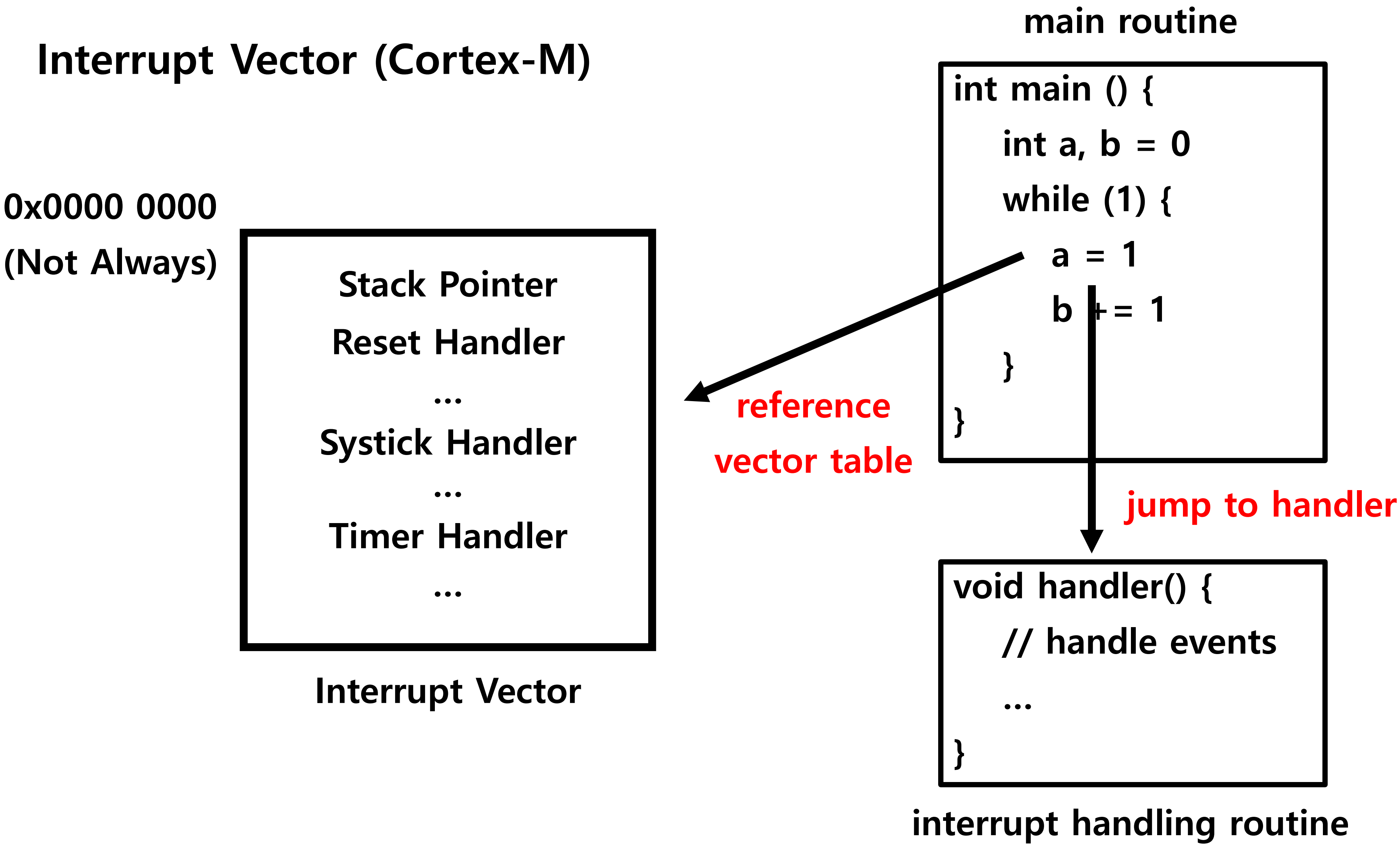
```
void handler() {
    // handle events
    ...
}
```

interrupt handling routine

after executing handler,

go back to the location previously executed

Interrupt Vector (Cortex-M)



Interrupt Vector (Cortex-M)

```c
void (* const interruptVectors[])(void) =
{
    (void (*)(void))((uint32_t)&__STACK_END),
                                            /* The initial stack pointer */
    Reset_Handler,                          /* The reset handler         */
    NMI_Handler,                            /* The NMI handler           */
    HardFault_Handler,                      /* The hard fault handler    */
    MemManage_Handler,                      /* The MPU fault handler     */
    BusFault_Handler,                       /* The bus fault handler     */
    UsageFault_Handler,                     /* The usage fault handler   */
    0,                                      /* Reserved                  */
    0,                                      /* Reserved                  */
    0,                                      /* Reserved                  */
    0,                                      /* Reserved                  */
    SVC_Handler,                            /* SVCall handler            */
    DebugMon_Handler,                       /* Debug monitor handler     */
    0,                                      /* Reserved                  */
    PendSV_Handler,                         /* The PendSV handler        */
    SysTick_Handler,                        /* The SysTick handler       */
    PSS_IRQHandler,                         /* PSS Interrupt             */
    CS_IRQHandler,                          /* CS Interrupt              */
```

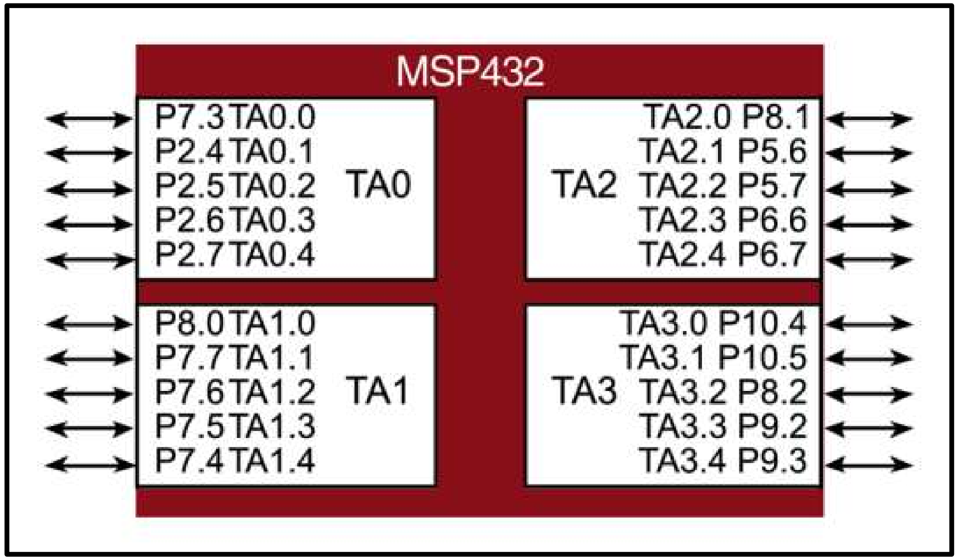
startup_msp432p401r_ccs.c, line 112

2. Timer Interrupt

MSP432 Timer A

MSP432 Timer A consists of

- **4 Timers TA0, TA1, TA2, TA3**
- **Each timer has 7 submodules**



Timer A Registers

	15-10	9-8	7-6	5-4	3	2	1	0	Name
0.0000		TASSEL	ID	MC		TACLR	TAIE	TAIFG	TA0CTL

	15-14	13-12	11	10	9	8	7-5	4	3	2	1	0	
0.0002	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL0
0.0004	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL1
0.0006	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL2
0.0008	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL3
0.000A	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL4
0.000C	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL5
0.000E	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL6

	15–0	
0.0010	16-bit counter	TA0R
0.0012	16-bit Capture/Compare 0 Register	TA0CCR0
0.0014	16-bit Capture/Compare 1 Register	TA0CCR1
0.0016	16-bit Capture/Compare 2 Register	TA0CCR2
0.0018	16-bit Capture/Compare 3 Register	TA0CCR3
0.001A	16-bit Capture/Compare 4 Register	TA0CCR4
0.001C	16-bit Capture/Compare 5 Register	TA0CCR5
0.001E	16-bit Capture/Compare 6 Register	TA0CCR6

	15-3	2-0	
0.0020		TAIDEX	TA0EX0

	15-0	
0.002E	TAIV	TA0IV

Timer A Register – CTL

Table 17-4. TAxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit resets TAxR, the timer clock divider logic, and the count direction. The TACLR bit is automatically reset and is always read as zero.
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

Timer A Register – CCTL

Bit	Field	Type	Reset	Description
15-14	CM	RW	0h	Capture mode 00b = No capture 01b = Capture on rising edge 10b = Capture on falling edge 11b = Capture on both rising and falling edges
13-12	CCIS	RW	0h	Capture/compare input select. These bits select the TAxCCR0 input signal. See the device-specific data sheet for specific signal connections. 00b = CCIxA 01b = CCIxB 10b = GND 11b = VCC
11	SCS	RW	0h	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0b = Asynchronous capture 1b = Synchronous capture
10	SCCI	RW	0h	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit.
9	Reserved	R	0h	Reserved. Reads as 0.
8	CAP	RW	0h	Capture mode 0b = Compare mode 1b = Capture mode
7-5	OUTMOD	RW	0h	Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCR0 because EQUx = EQU0. 000b = OUT bit value 001b = Set 010b = Toggle/reset 011b = Set/reset 100b = Toggle 101b = Reset 110b = Toggle/set 111b = Reset/set
4	CCIE	RW	0h	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0b = Interrupt disabled 1b = Interrupt enabled
3	CCI	R	0h	Capture/compare input. The selected input signal can be read by this bit.
2	OUT	RW	0h	Output. For output mode 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high

Bit	Field	Type	Reset	Description
1	COV	RW	0h	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0b = No capture overflow occurred 1b = Capture overflow occurred
0	CCIFG	RW	0h	Capture/compare interrupt flag 0b = No interrupt pending 1b = Interrupt pending

Timer A Register – TAxEX0

Table 17-9. TAxEX0 Register Description

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved. Reads as 0.
2-0	TAIDEX	RW	0h	Input divider expansion. These bits along with the ID bits select the divider for the input clock. 000b = Divide by 1 001b = Divide by 2 010b = Divide by 3 011b = Divide by 4 100b = Divide by 5 101b = Divide by 6 110b = Divide by 7 111b = Divide by 8

Timer Interrupt Initialization

```
void (*TimerA2Task)(void);
void TimerA2_Init(void(*task)(void), uint16_t period) {
    TimerA2Task = task;
    TIMER_A2->CTL = 0x0280;
    TIMER_A2->CCTL[0] = 0x0010;
    TIMER_A2->CCR[0] = (period - 1);
    TIMER_A2->EX0 = 0x0005;
    NVIC->IP[3] = (NVIC->IP[3]&0xFFFFFF00)|0x00000040;
    NVIC->ISER[0] = 0x00001000;
    TIMER_A2->CTL |= 0x0014;
}
```

TIMER_A2->CTL = 0x0280

-> TIMER_A2->CTL = 0b 0000 0010 1000 0000

7~6 bit : 1/4 input divider

9~8 bit : timer clock source, SMCLK = 12MHz

Timer Interrupt Initialization

```c
void (*TimerA2Task)(void);
void TimerA2_Init(void(*task)(void), uint16_t period) {
    TimerA2Task = task;
    TIMER_A2->CTL = 0x0280;
    TIMER_A2->CCTL[0] = 0x0010;
    TIMER_A2->CCR[0] = (period - 1);
    TIMER_A2->EX0 = 0x0005;
    NVIC->IP[3] = (NVIC->IP[3]&0xFFFFFF00)|0x00000040;
    NVIC->ISER[0] = 0x00001000;
    TIMER_A2->CTL |= 0x0014;
}
```

TIMER_A2->CCTL = 0x0010

-> TIMER_A2->CCTL = 0b 0000 0000 0001 0000

8 bit : compare mode

4 bit : enable compare interrupt

Timer Interrupt Initialization

```
void (*TimerA2Task)(void);
void TimerA2_Init(void(*task)(void), uint16_t period) {
    TimerA2Task = task;
    TIMER_A2->CTL = 0x0280;
    TIMER_A2->CCTL[0] = 0x0010;
    TIMER_A2->CCR[0] = (period - 1);
    TIMER_A2->EX0 = 0x0005;
    NVIC->IP[3] = (NVIC->IP[3]&0xFFFFFF00)|0x00000040;
    NVIC->ISER[0] = 0x00001000;
    TIMER_A2->CTL |= 0x0014;
}
```

TIMER_A2->CCR[0] : compare match value

when the counter meets CCR[0], interrupt occurs

TIMER_A2->EX0 : input divider 2

Timer Interrupt Initialization

```
void (*TimerA2Task)(void);
void TimerA2_Init(void(*task)(void), uint16_t period) {
    TimerA2Task = task;
    TIMER_A2->CTL = 0x0280;
    TIMER_A2->CCTL[0] = 0x0010;
    TIMER_A2->CCR[0] = (period - 1);
    TIMER_A2->EX0 = 0x0005;
    NVIC->IP[3] = (NVIC->IP[3]&0xFFFFFF00)|0x00000040;
    NVIC->ISER[0] = 0x00001000;
    TIMER_A2->CTL |= 0x0014;
}
```

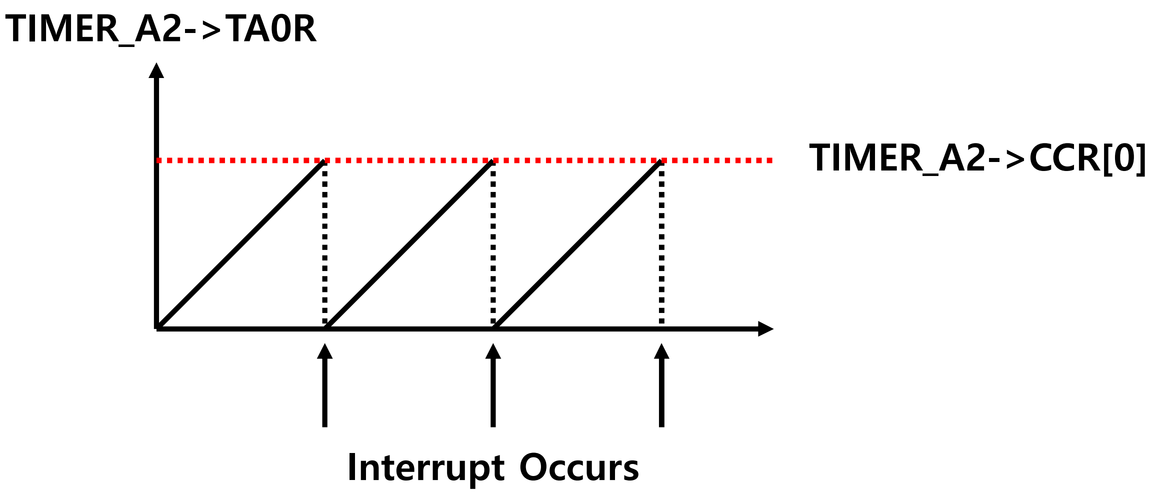
TIMER_A2->CTL |= 0x0014

-> TIMER_A2->CTL |= 0b 0000 0000 0001 0100

5~4 bit : up mode, increment counter

2 bit : clear TA0R register

Timer Interrupt Initialization



period = 50,000

-> 0.1s

$$\text{Resolution} = \frac{1}{\text{Clock Source}} * (\text{input divider}) * (\text{EX0} + 1)$$

$$= \frac{1}{12\text{MHz}} * (4) * (5 + 1) = 2\text{us}$$

Timer Interrupt Handler

```
void TA2_0_IRQHandler(void){
  TIMER_A2->CCTL[0] &= ~0x0001;
  (*TimerA2Task)();
}
```

TIMER_A2->CCTL[0] &= ~0x0001;

-> send ack to the timer

-> If the timer does not receive an ack,
the timer continuously send an interrupt

```
void task() {
    printf("interrupt occurs!\n");
}

int main(void) {
    Clock_Init48MHz();
    TimerA2_Init(&task, 50000);

    while(1) {};
}
```

- Tachometer -

1. Tachomter

What is Tachometer?

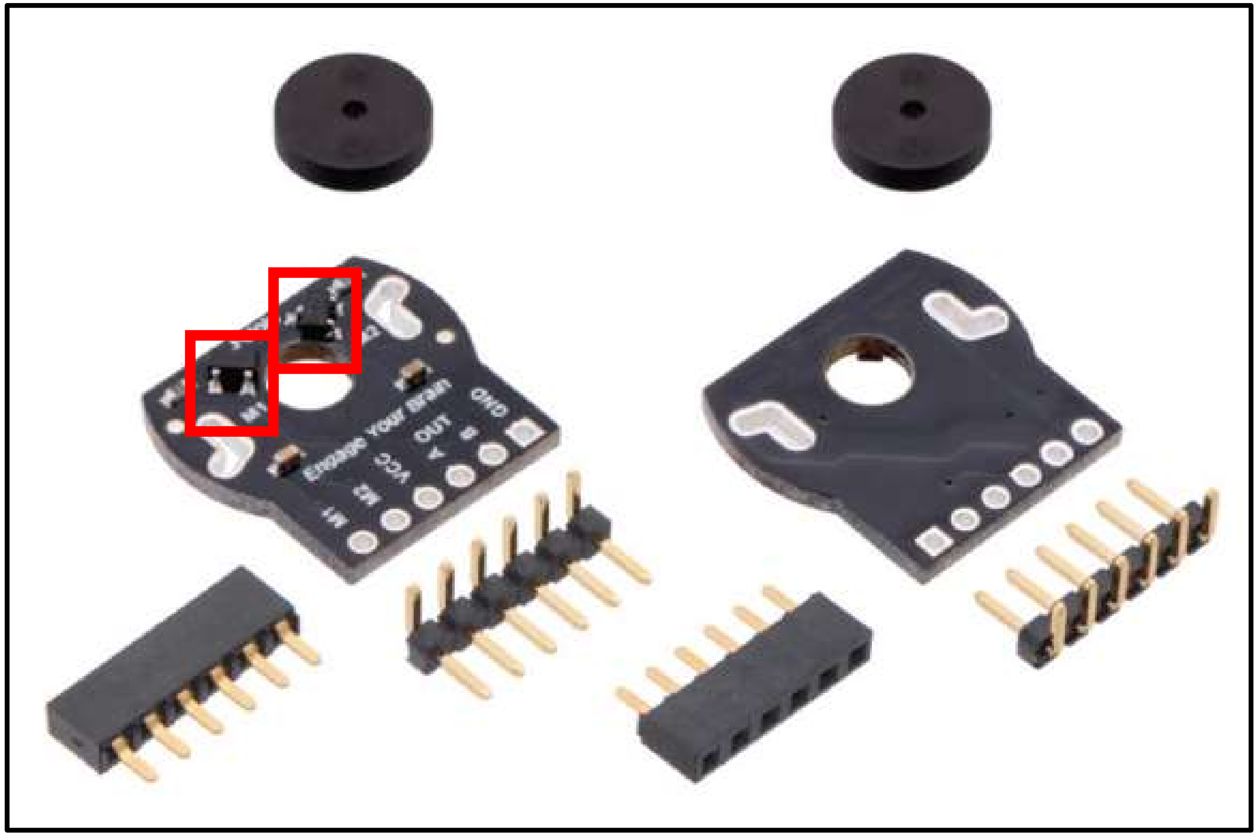
Definition

" A tachometer is an instrument measuring the rotation of a shaft or disk"

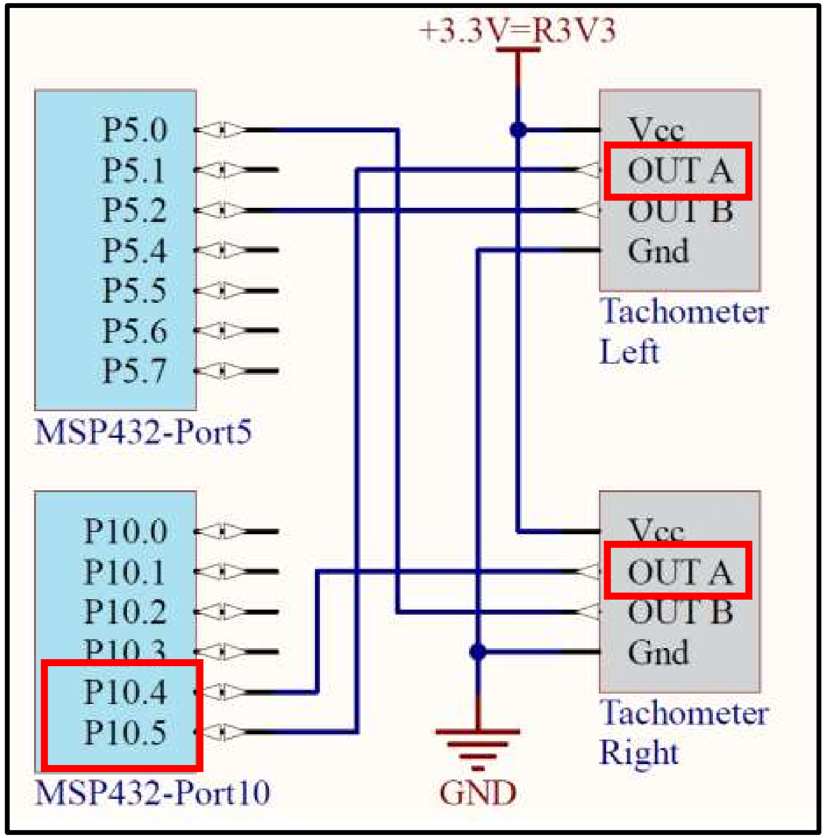
What can we do with a tachometer?

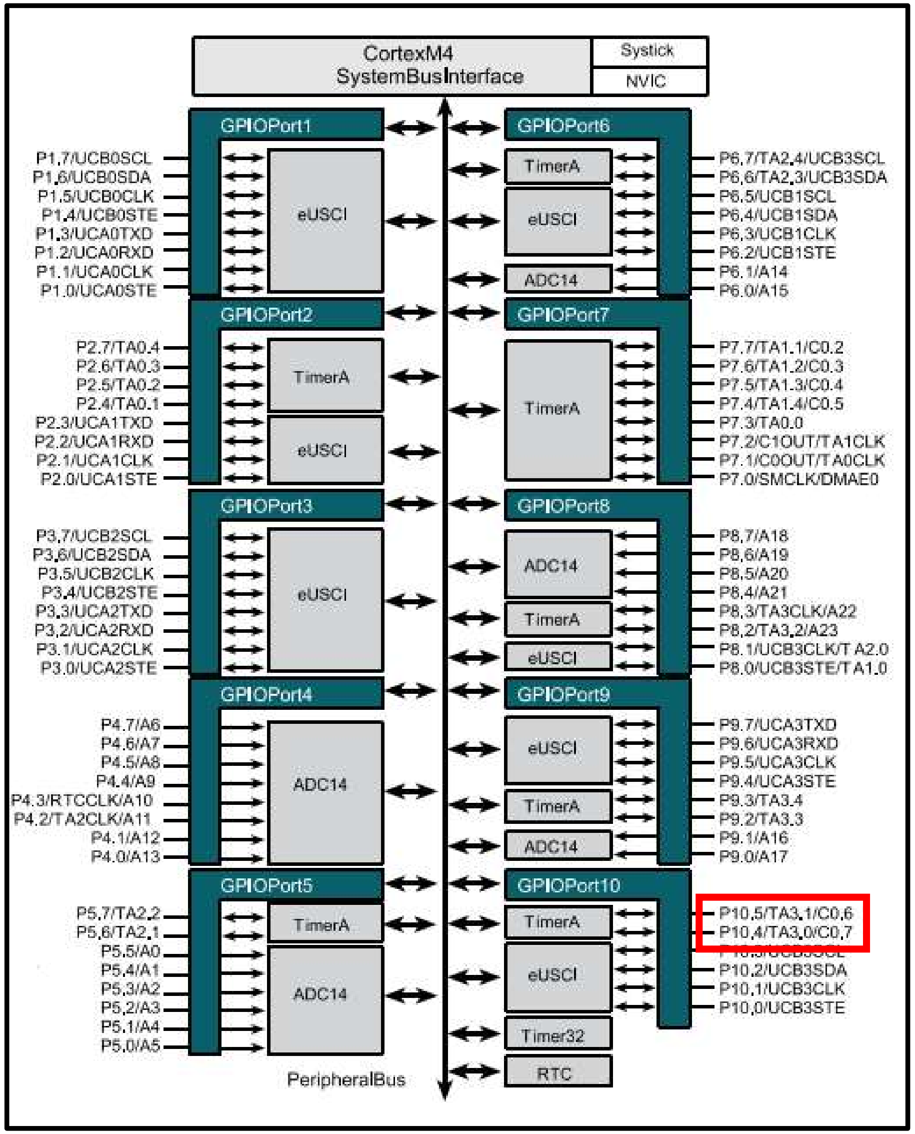
- We can measure RPM (rotations per minute) of the robot

TI-RSLK Tachometer

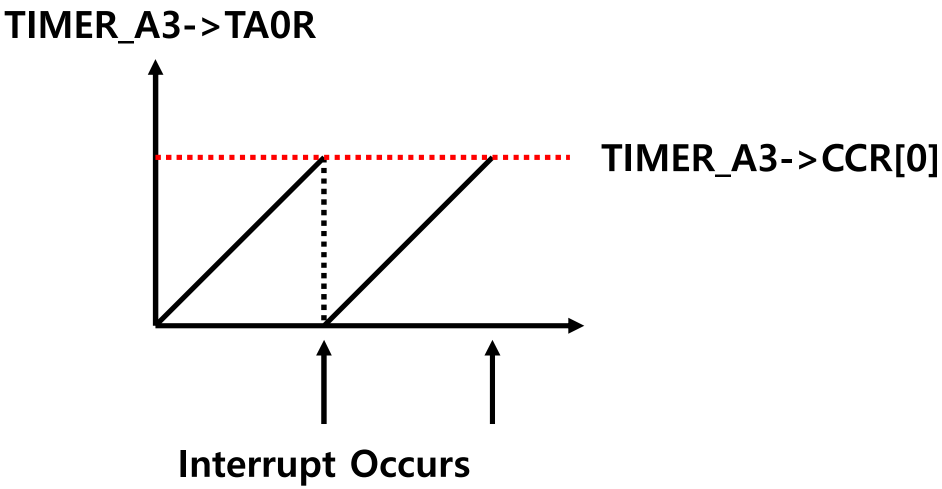


TI-RSLK Tachometer

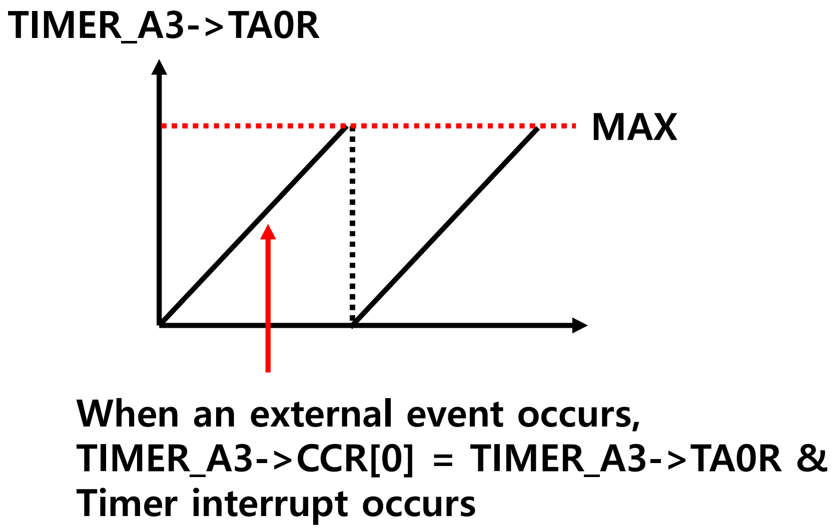




Timer A Input Capture



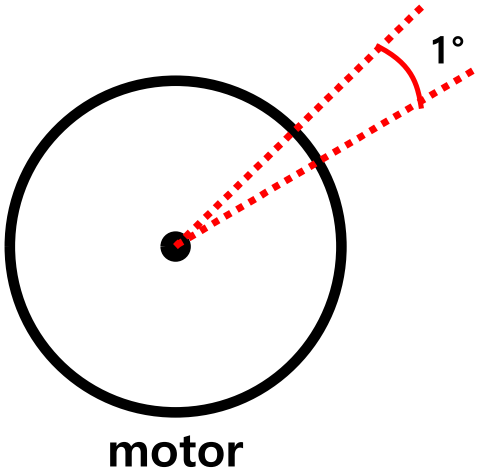
Output Compare



Input Capture

When an event occurs, the counter value is stored in the CCR!

Tachometer & Timer's Input Capture



Timer Interrupt Occurs

→

CCR = TA0R

Period
= Previous CCR – Current CCR

Timer Interrupt Handler

2. Implementation

Initialization

```c
void timer_A3_capture_init() {
    P10->SEL0 |= 0x30;
    P10->SEL1 &= ~0x30;
    P10->DIR &= ~0x30;

    TIMER_A3->CTL &= ~0x0030;
    TIMER_A3->CTL = 0x0200;

    TIMER_A3->CCTL[0] = 0x4910;
    TIMER_A3->CCTL[1] = 0x4910;
    TIMER_A3->EX0 &= ~0x0007;

    NVIC->IP[3] = (NVIC->IP[3]&0x0000FFFF) | 0x40400000;
    NVIC->ISER[0] = 0x0000C000;
    TIMER_A3->CTL |= 0x0024;
}
```

alternative mode

setup timer &
input capture mode

configure interrupts

Interrupt Handler

```
uint16_t first_left;
uint16_t first_right;

uint16_t period_left;
uint16_t period_right;

void TA3_0_IRQHandler(void) {
    TIMER_A3->CCTL[0] &= ~0x0001;
    period_right = TIMER_A3->CCR[0] - first_right;
    first_right = TIMER_A3->CCR[0];
}

void TA3_N_IRQHandler(void) {
    TIMER_A3->CCTL[1] &= ~0x0001;
    period_left = TIMER_A3->CCR[1] - first_left;
    first_left = TIMER_A3->CCR[1];
}
```

Measure RPM

```
uint32_t get_left_rpm() {
    return 2000000 / period_left;
}
```

$$RPM = \frac{1°}{360°} * \frac{1}{period} * \frac{60s}{1m} * \frac{1}{timer\ cycle} * \frac{1{,}000{,}000{,}000ns}{s}$$

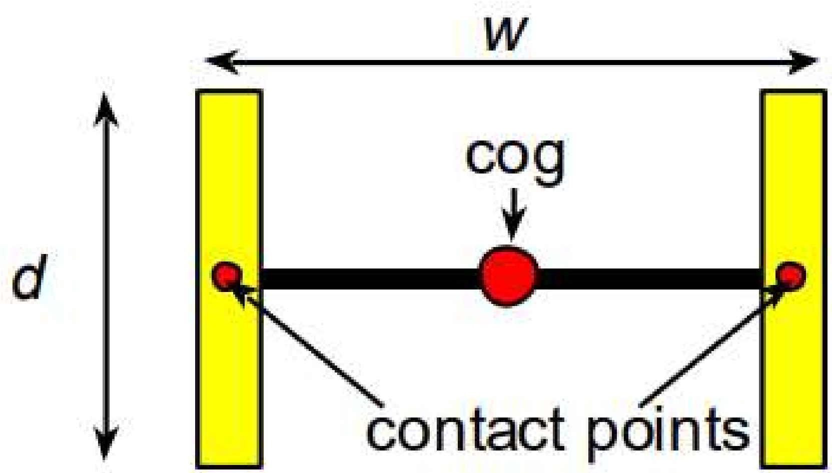
180° Wheel Rotation

```
uint32_t left_count;
void TA3_N_IRQHandler(void) {
    TIMER_A3->CCTL[1] &= ~0x0001;
    left_count++;
}
```

```
while (1) {
    if (left_count > 180) {
        move(0, 0);
    }
};
```

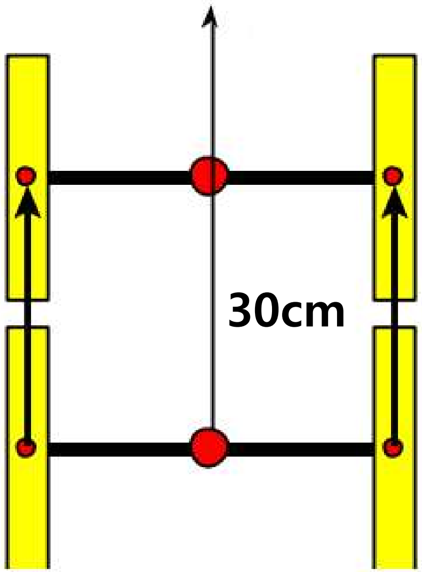
3. Activity
Please Upload LMS

Robot Spec



d = 7cm
w = 14cm

1. Go Straight 30cm



2. Rotate 30°(Assignments)