

# CS301

## Embedded System and Microcomputer Principle

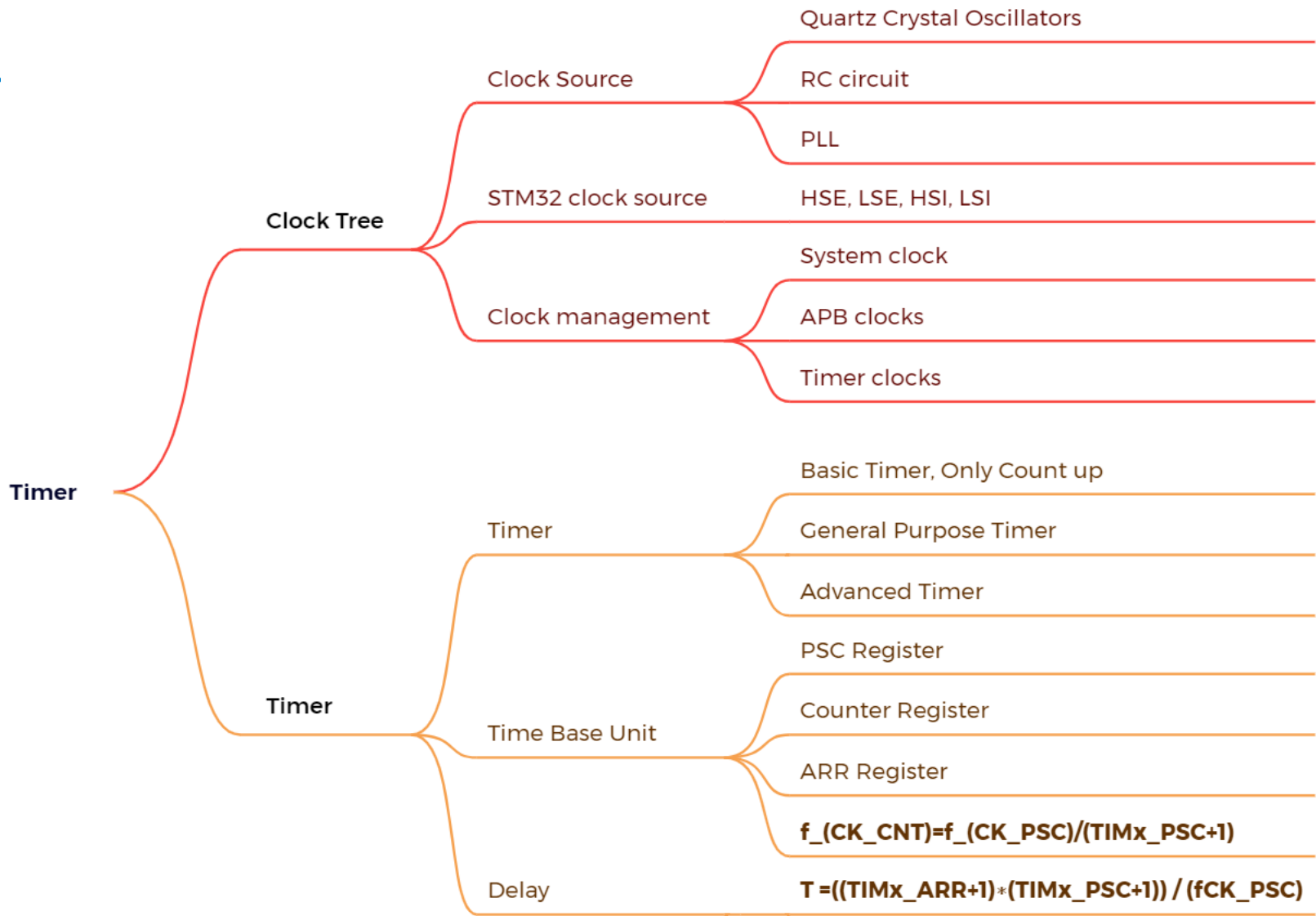
### Lecture 8: Advanced Timer Functionality

2024 Fall

This PowerPoint is for internal use only at Southern University of Science and Technology.  
Please do not repost it on other platforms without permission from the instructor.

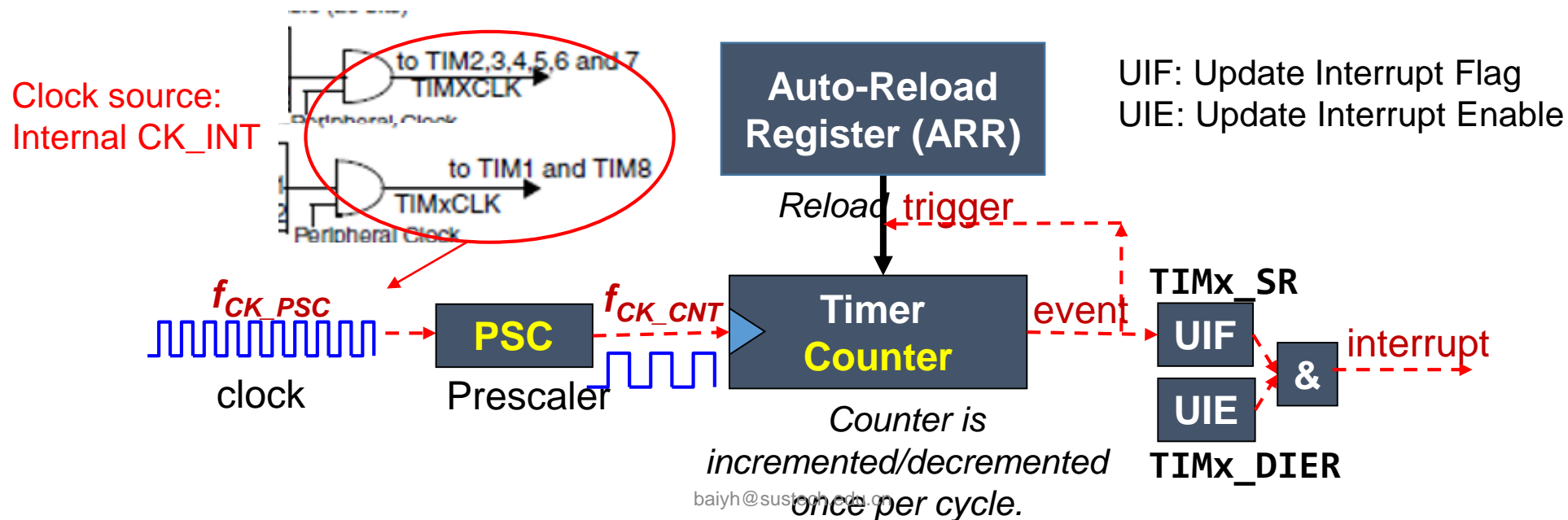


# Recap



# Basic Block Diagram

- Basic timer block diagram
  - Prescaler
    - divide counter clock frequency by any factor between 1~65536
  - Counter
    - counts from 0 to the auto-reload value (contents of the ARR register), then restarts from 0 and generates an overflow event.
  - Auto-Reload Register



# TIMx\_DIER Registers

- TIMx\_DIER (DMA/Interrupt Enable Register)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

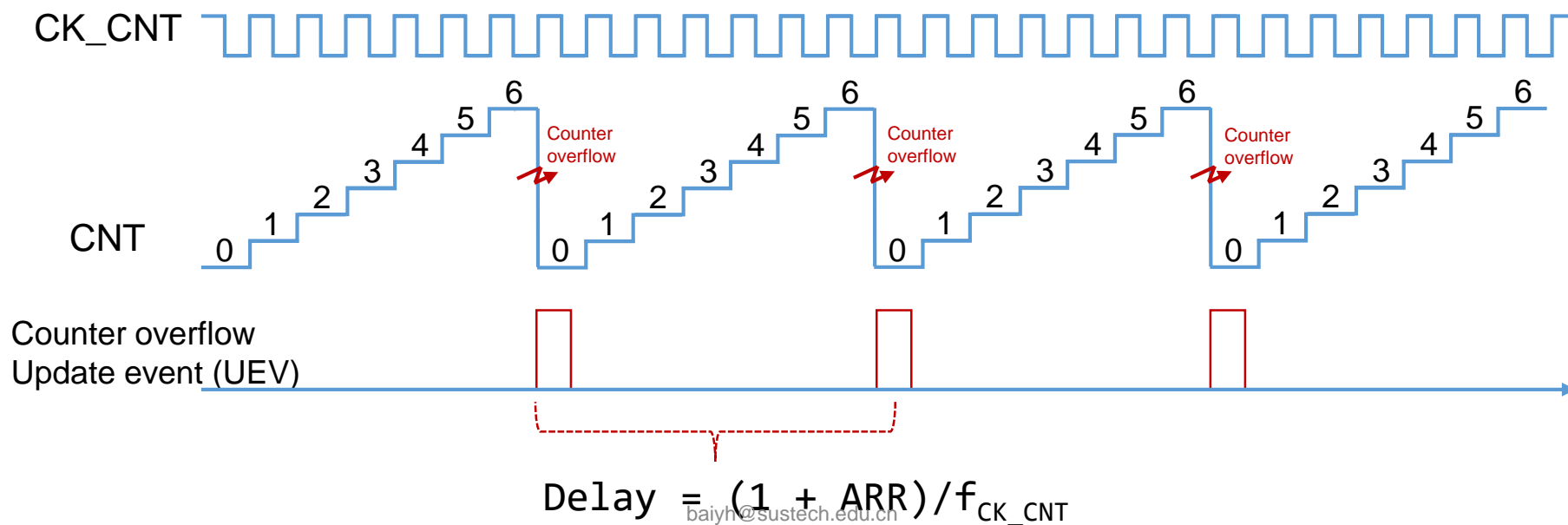
- UIE: Update Interrupt Enable
- CC1IE: Capture/compare Channel 1 Interrupt Enable
- CC2IE: Capture/compare Channel 2 Interrupt Enable
- CC3IE: Capture/compare Channel 3 Interrupt Enable
- CC4IE: Capture/compare Channel 4 Interrupt Enable
- ...

# Delay and Overflow Frequency

- Counting Up Example
- What's the CK\_CNT frequency
- What is the counter overflow frequency?

$$f_{CK\_CNT} = \frac{f_{CK\_PSC}}{PSC + 1}$$

$$f_{CK\_OV} = \frac{1}{Delay} = \frac{f_{CK\_PSC}}{(PSC + 1)(ARR + 1)}$$

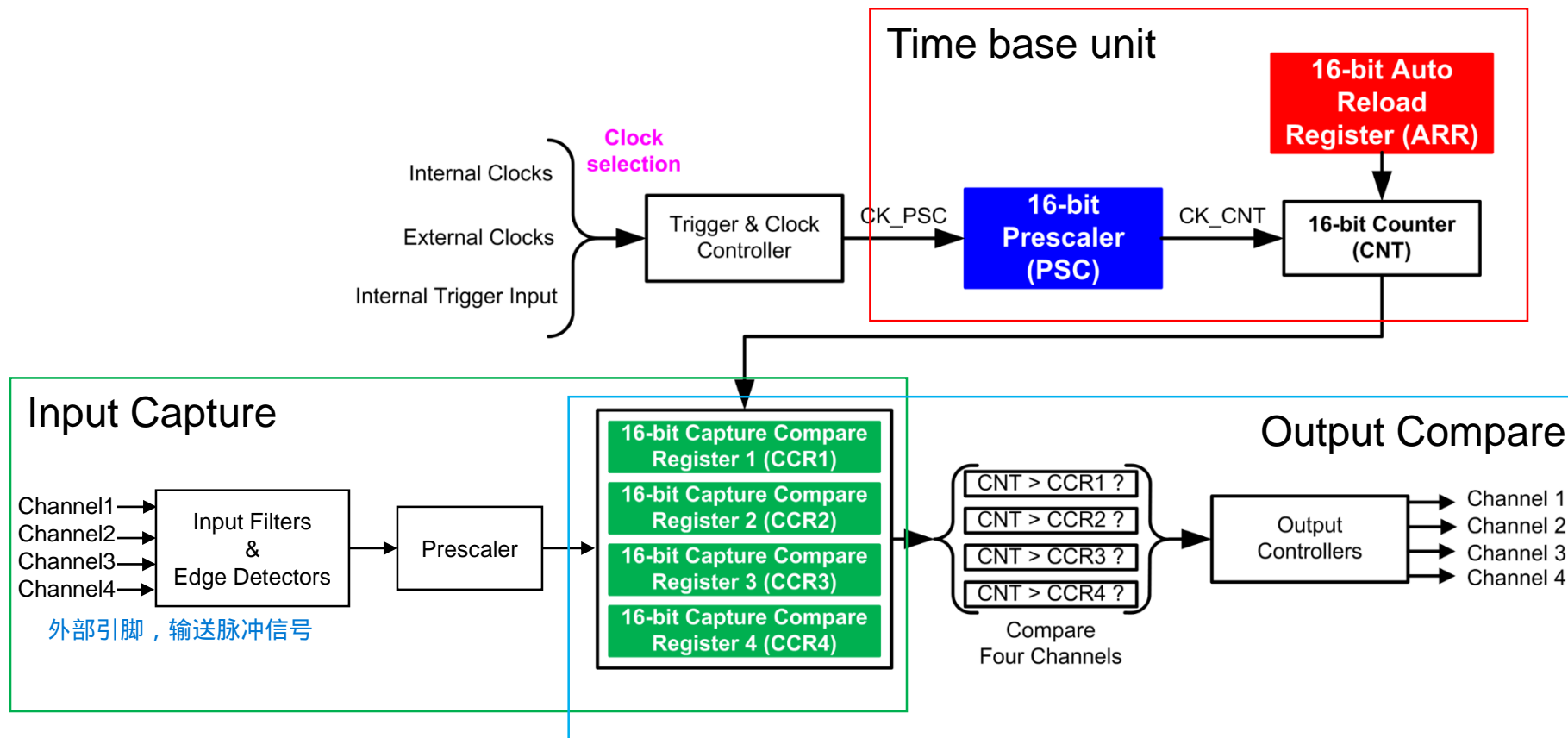


# Outline

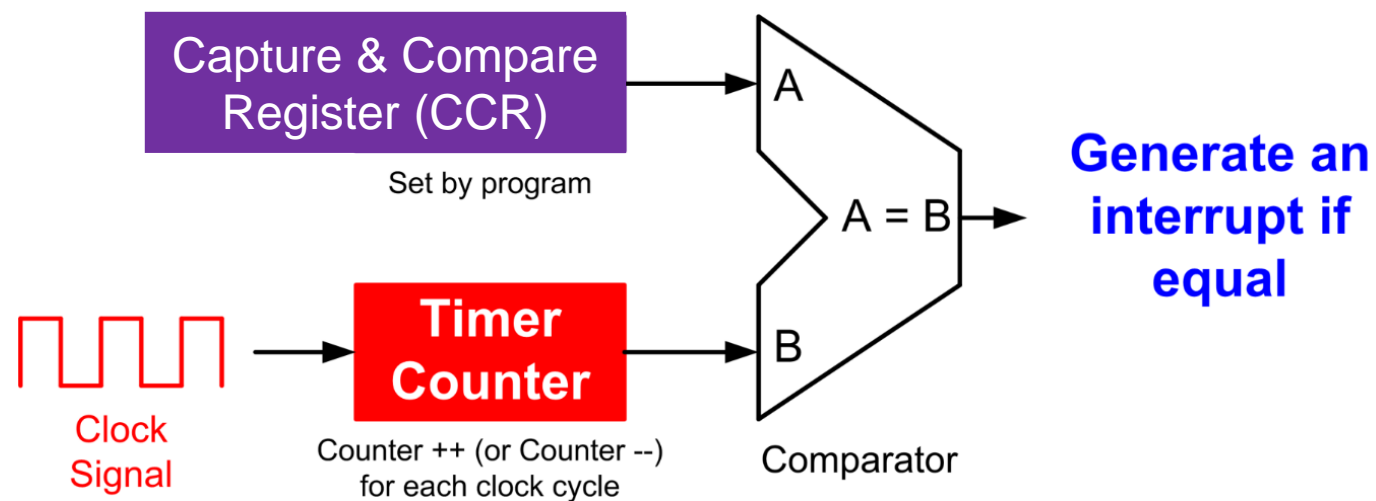
- **Output Compare PWM**
- Input Capture
- SysTick

# Timer

- Time Generation & Input Capture & Output Compare



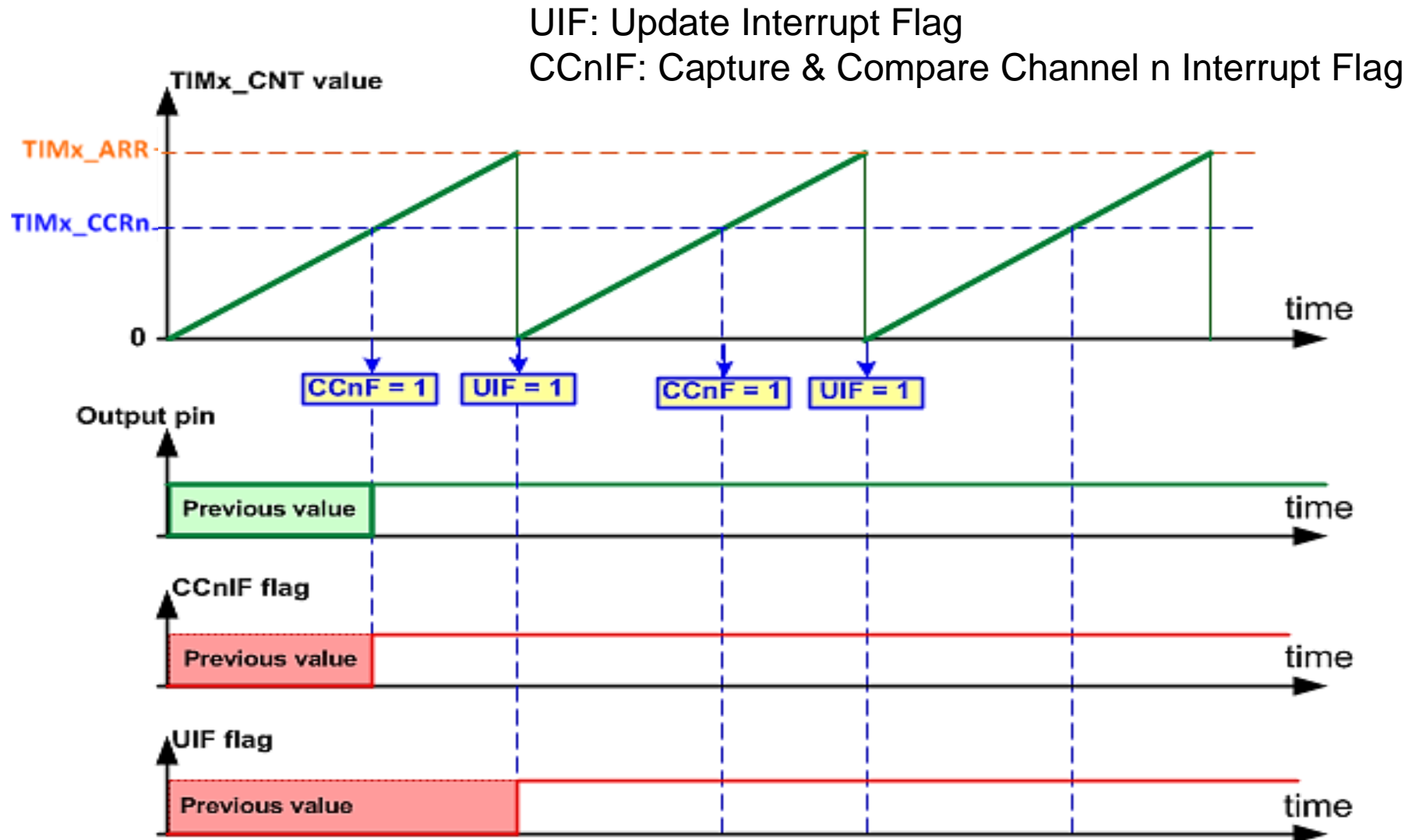
# Output Compare



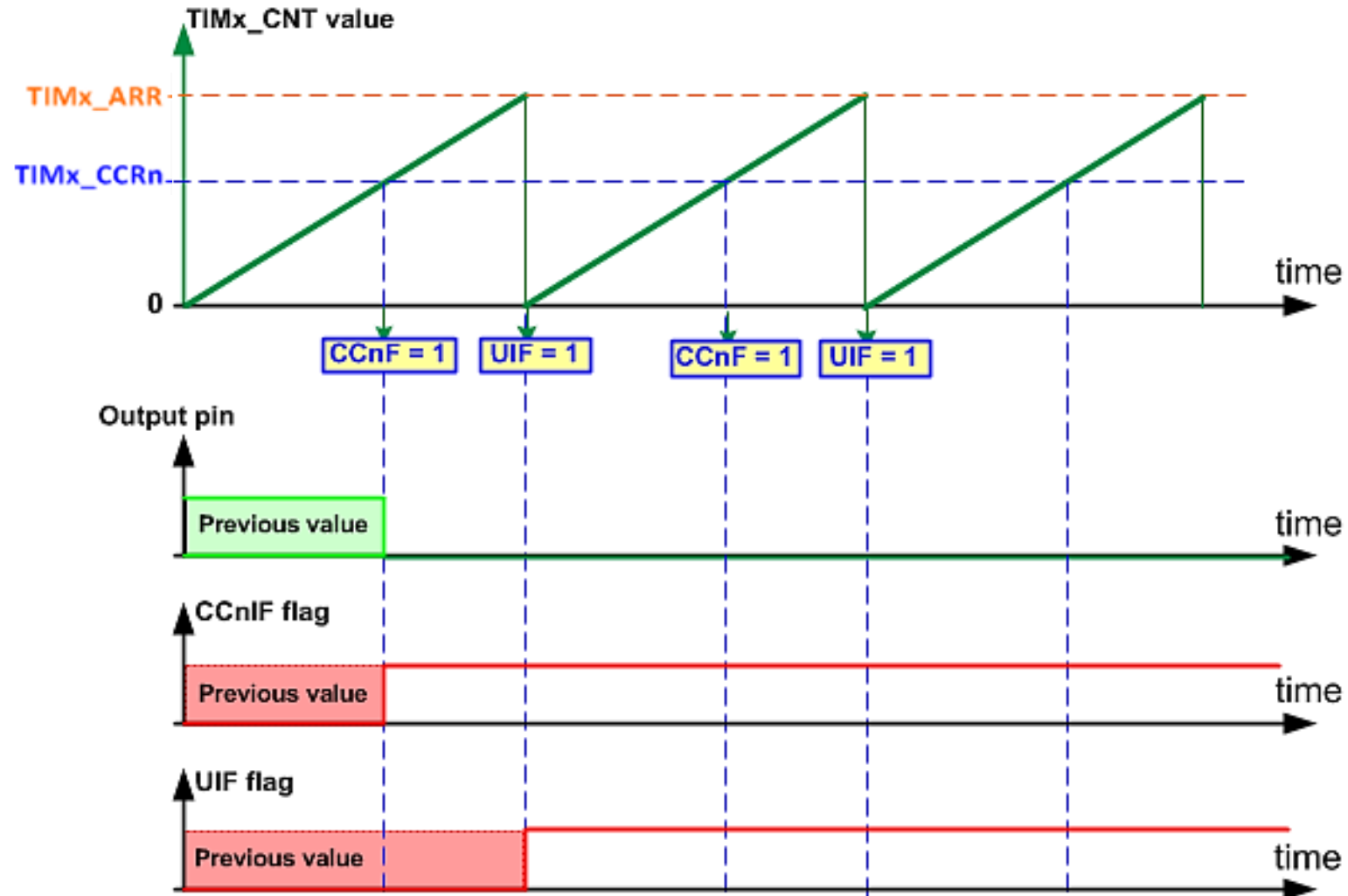
Output Compare Mode (OCnM)	Timer Output (OCREF)
000	Frozen
001	High if CNT == CCR
010	Low if CNT == CCR
011	Toggle if CNT == CCR
100	Forced low (always low)
101	Forced high (always high)
110	PWM Mode 1
111	PWM Mode 2



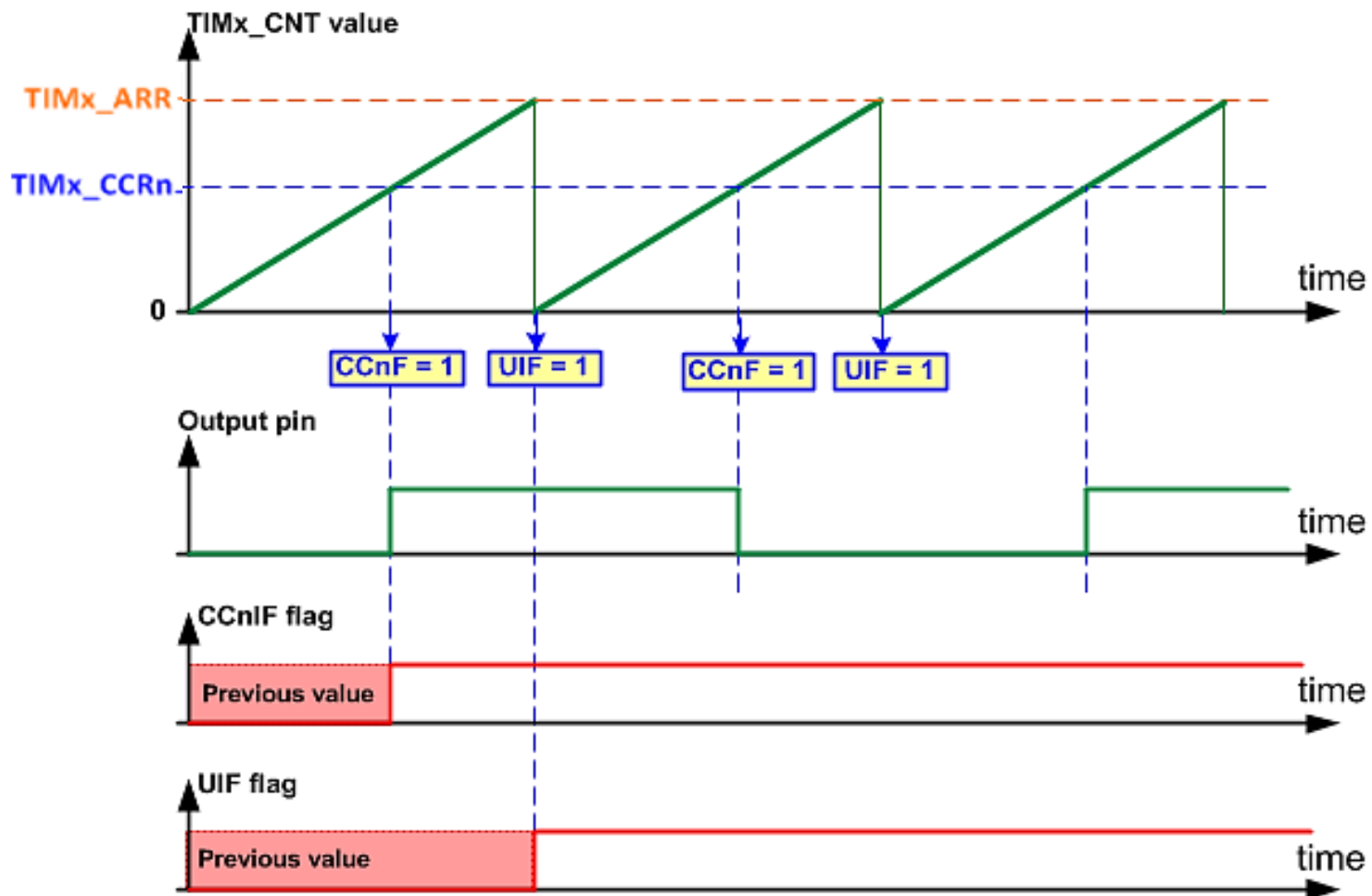
# Set Mode (OCnM = 001)



# Clear Mode (OCnM = 010)

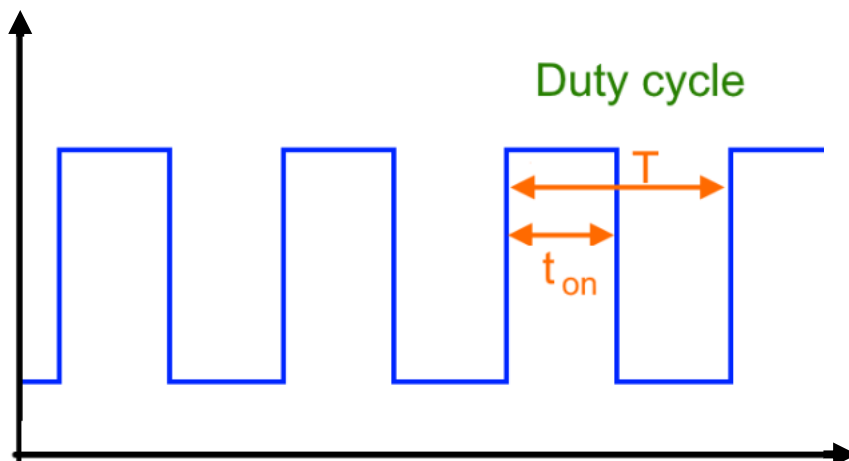


# Toggle mode (OCnM = 011)



# Duty Cycle

- Duty cycle(占空比) of a periodic waveform is the ratio of  $t_{on}$  to  $T$
- usually expressed as a percentage
- measures fraction of period for which signal is HIGH
  - Frequency
  - Pulse-Width
  - Duty cycle



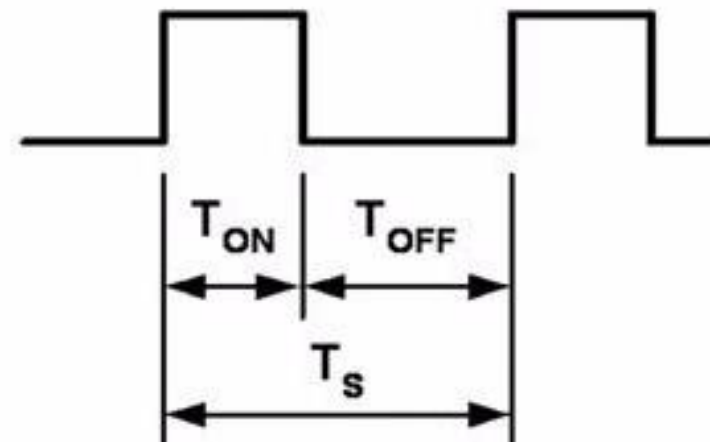
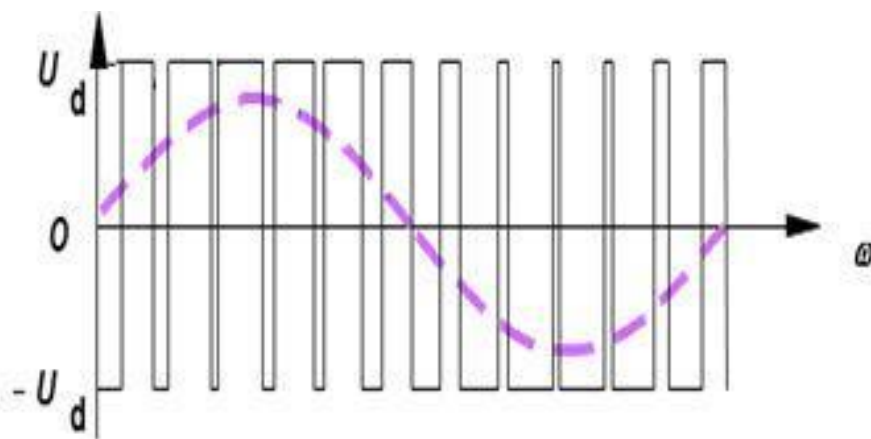
# PWM

- PWM (Pulse Width Modulation) is a technique to effectively obtain desired analog parameters by modulating the width of a series of pulses. It is commonly applied in fields such as motor speed control.
- PWM Parameters:
  - Frequency =  $1 / T_S$
  - Duty Cycle =  $T_{ON} / T_S = T_{ON} / (T_{ON} + T_{OFF})$

占空比：对于一个周期性的信号，高电平的时间占整个信号时间的比例

一般用于惯性系统，例如呼吸灯和电机

通过调整占空比，调整LED的亮灭时间



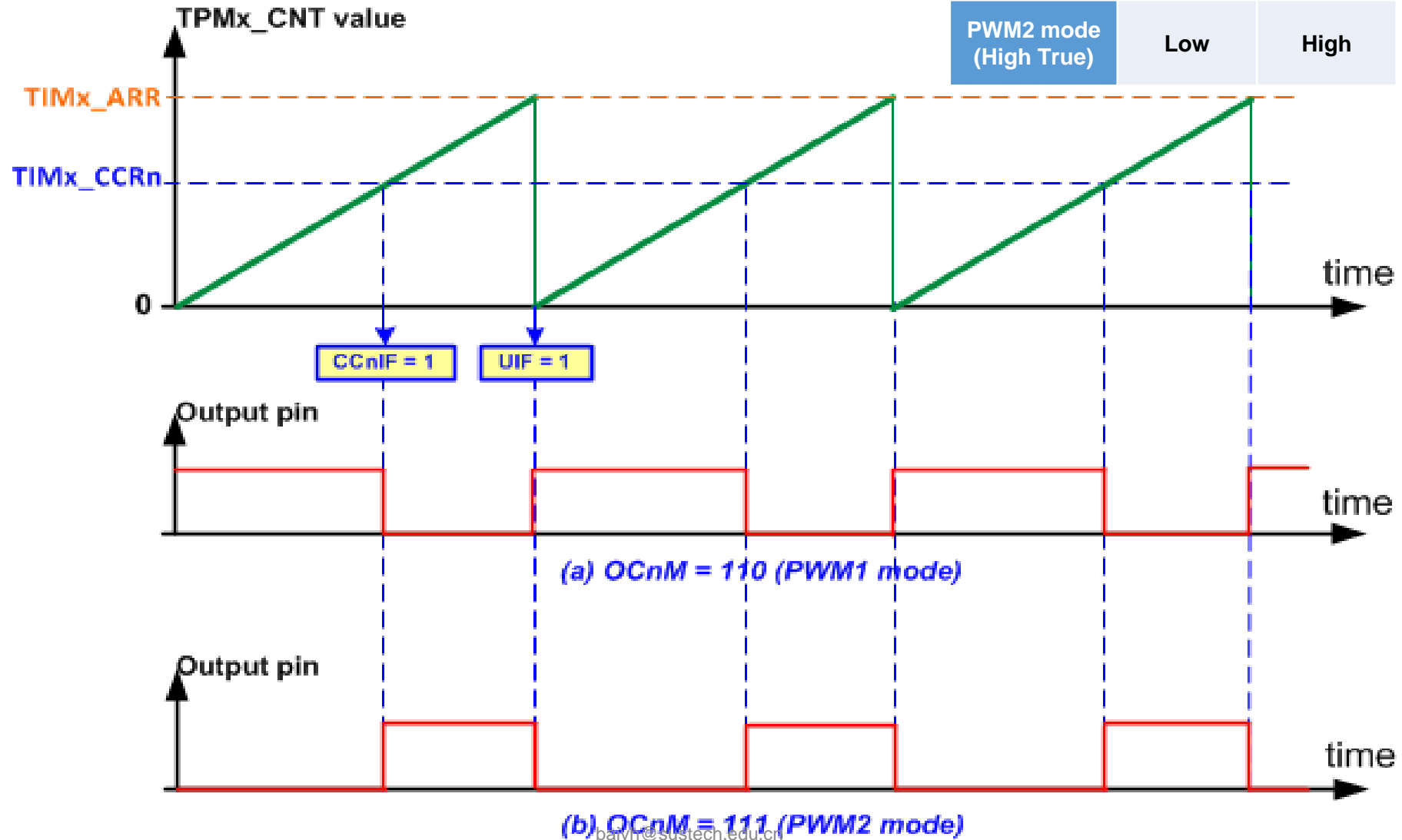
# PWM Mode

- PWM1 mode: The PWM signal is high as long as  $TIMx\_CNT < TIMx\_CCRx$  else it becomes low
- PWM2 mode: reverse of PWM1 mode

Mode	Counter < CCR	Counter $\geq$ CCR
PWM1 mode (Low True)	High	Low
PWM2 mode (High True)	Low	High

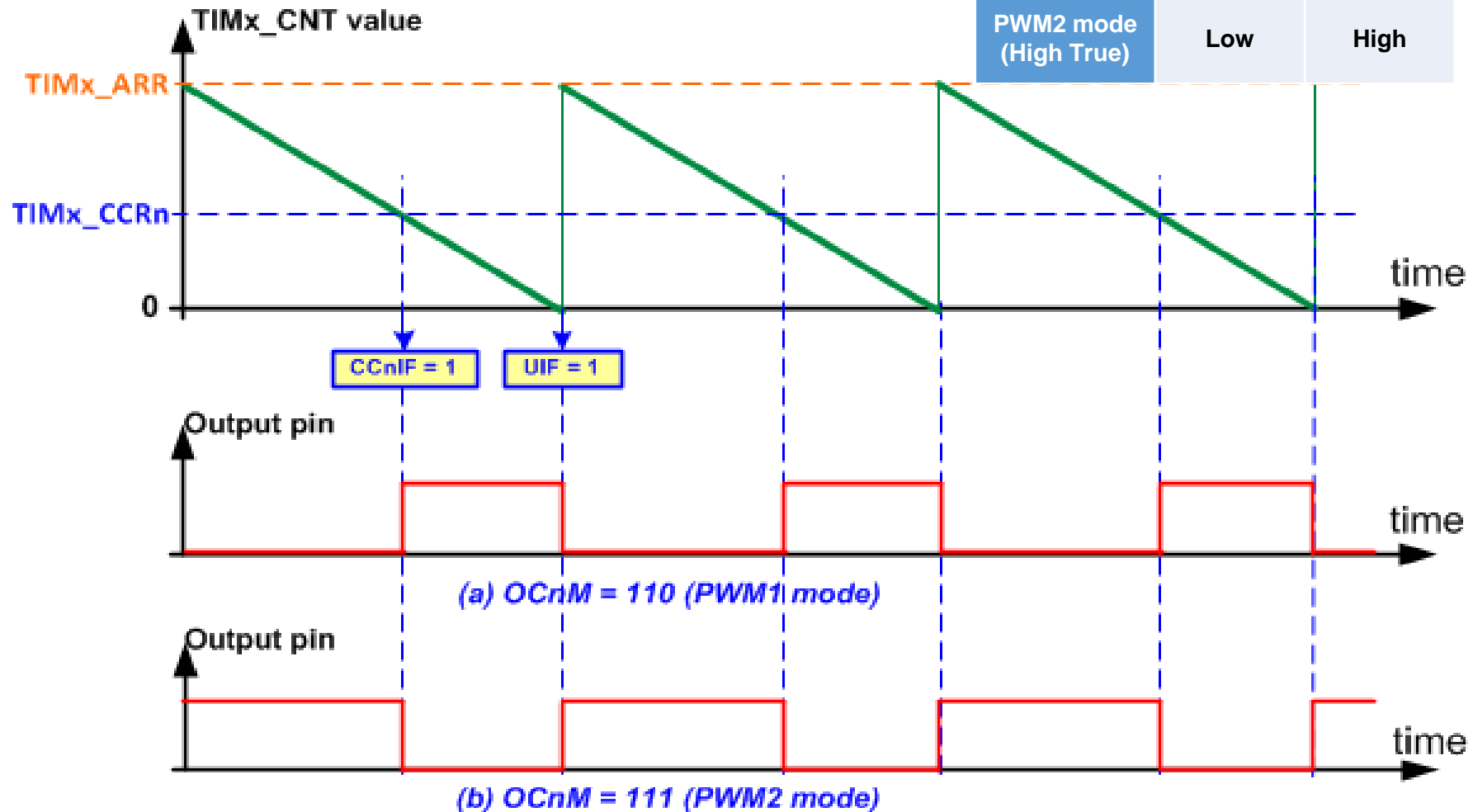
# PWM in Up-Counting Mode

Mode	Counter < CCR	Counter ≥ CCR
PWM1 mode (Low True)	High	Low
PWM2 mode (High True)	Low	High



# Down-Counting Mode

Mode	Counter < CCR	Counter ≥ CCR
PWM1 mode (Low True)	High	Low
PWM2 mode (High True)	Low	High

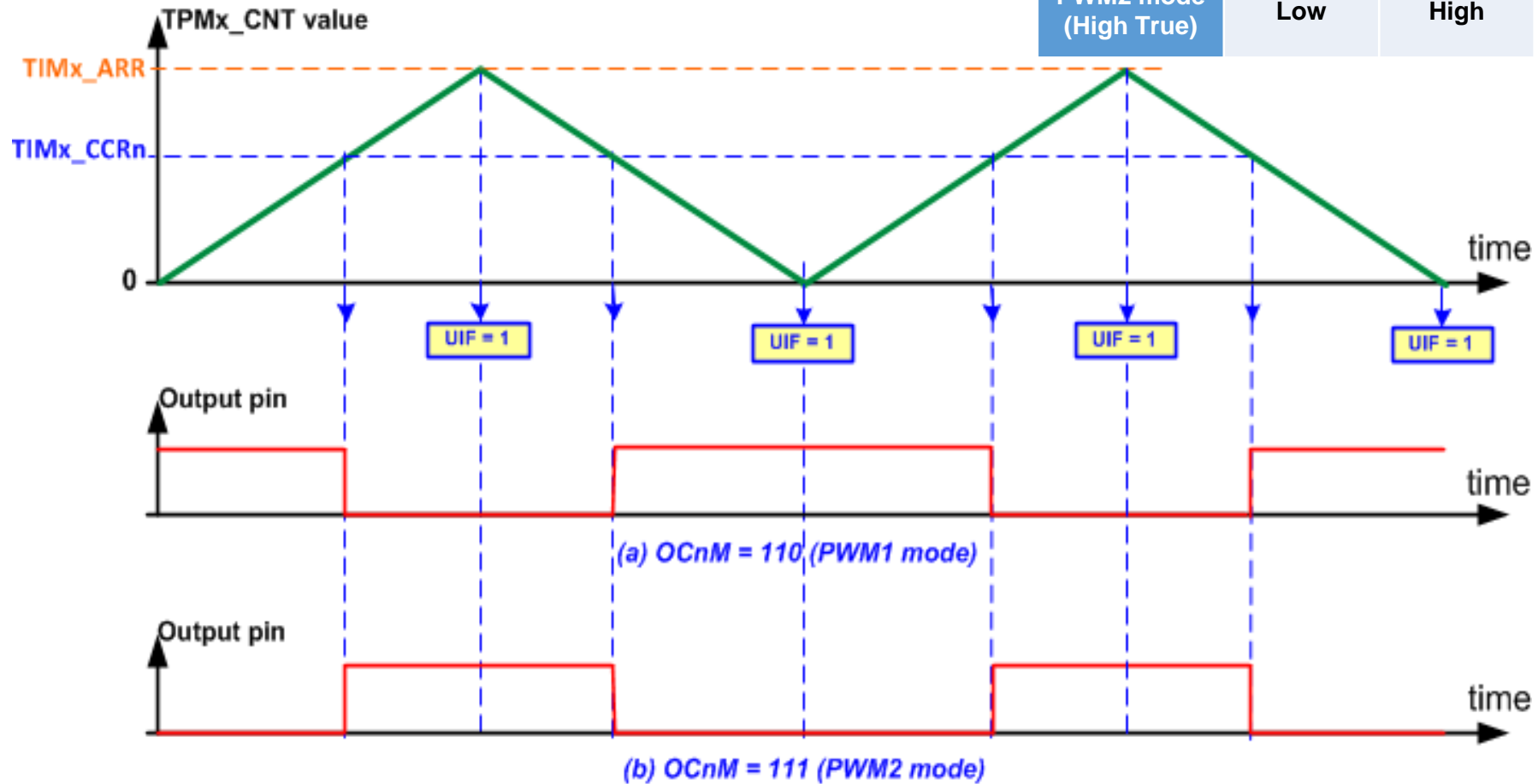




# Up-down Counting (Center-aligned)



Mode	Counter < CCR	Counter ≥ CCR
PWM1 mode (Low True)	High	Low
PWM2 mode (High True)	Low	High

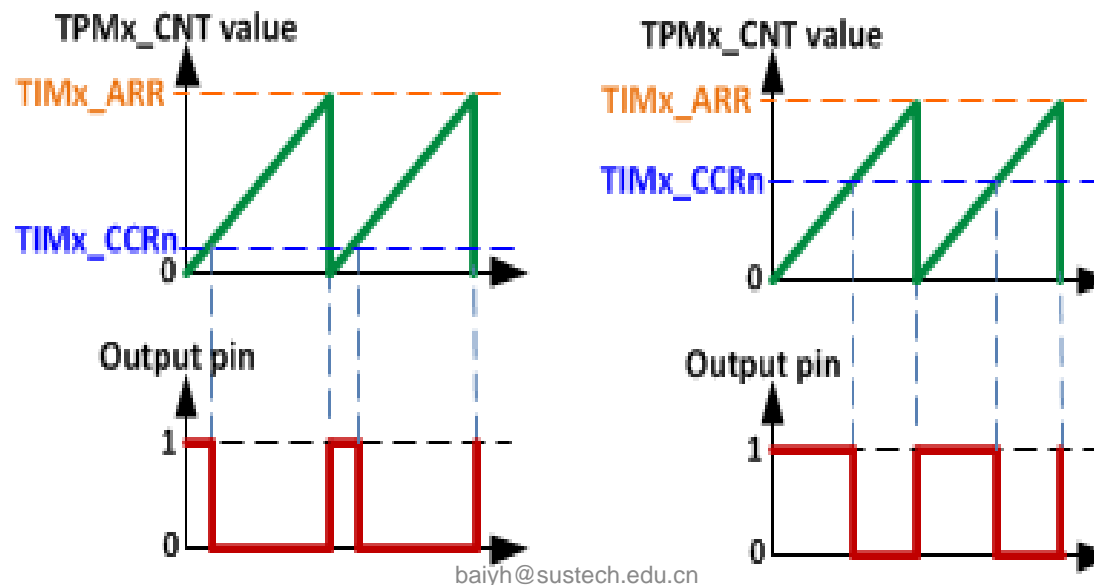


# PWM Parameters

- PWM Frequency is determined by ARR (Auto-reload Register)
- PWM Duty Cycle is determined by CCRx (Capture/Compare Register)

$$\text{PWM Frequency} = \frac{f_{\text{CK\_PSC}}}{(\text{ARR} + 1) \times (\text{PSC} + 1)}$$

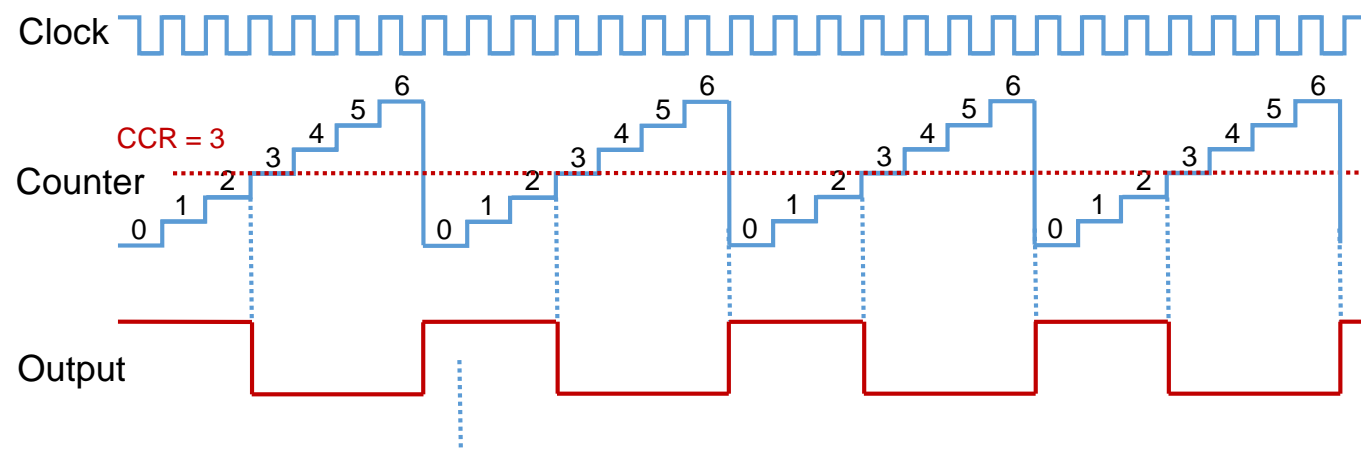
$$\begin{aligned} \text{PWM Duty Cycle} &= \text{CCR} / (\text{ARR} + 1) \times 100\% \\ \text{PWM Duty Cycle} &= 1 - \text{CCR} / (\text{ARR} + 1) \text{ if Mode2} \end{aligned}$$



# Example 1

- PWM Mode 1 (Low True)
- Upcounting mode, ARR = 6, CCR = 3

Mode	Counter < CCR	Counter ≥ CCR
PWM1 mode (Low True)	High	Low
PWM2 mode (High True)	Low	High

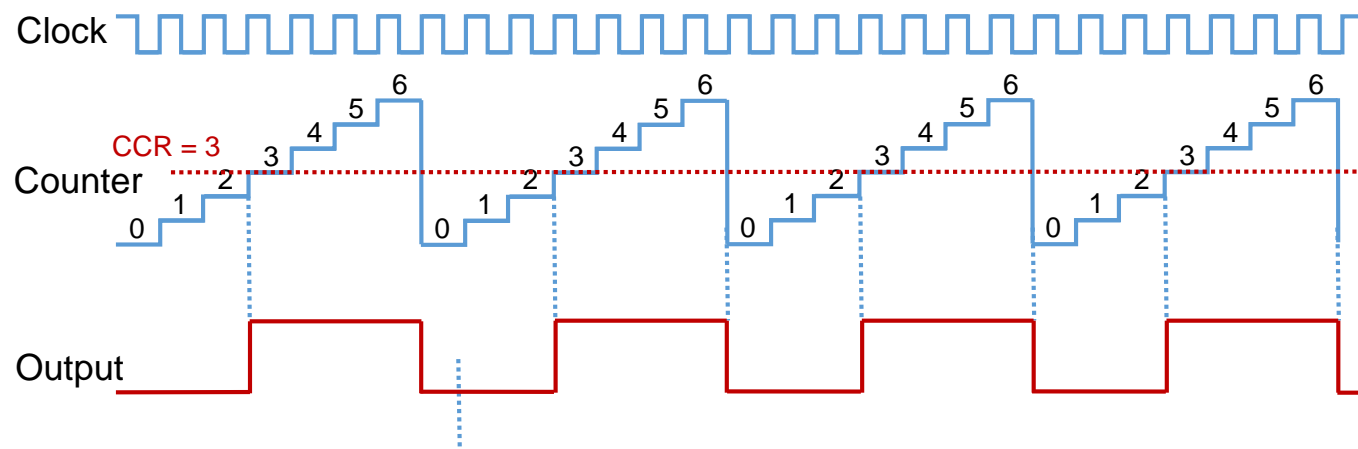


$$\begin{aligned}\text{Duty Cycle} &= \frac{\text{CCR}}{\text{ARR} + 1} \\ &= \frac{3}{7}\end{aligned}$$

$$\begin{aligned}\text{Period} &= (1 + \text{ARR}) * \text{Clock Period} \\ &= 7 * \text{Clock Period}\end{aligned}$$

## Example 2

- PWM Mode 2 (High-True)
- Upcounting mode,  $ARR = 6$ ,  $CCR = 3$

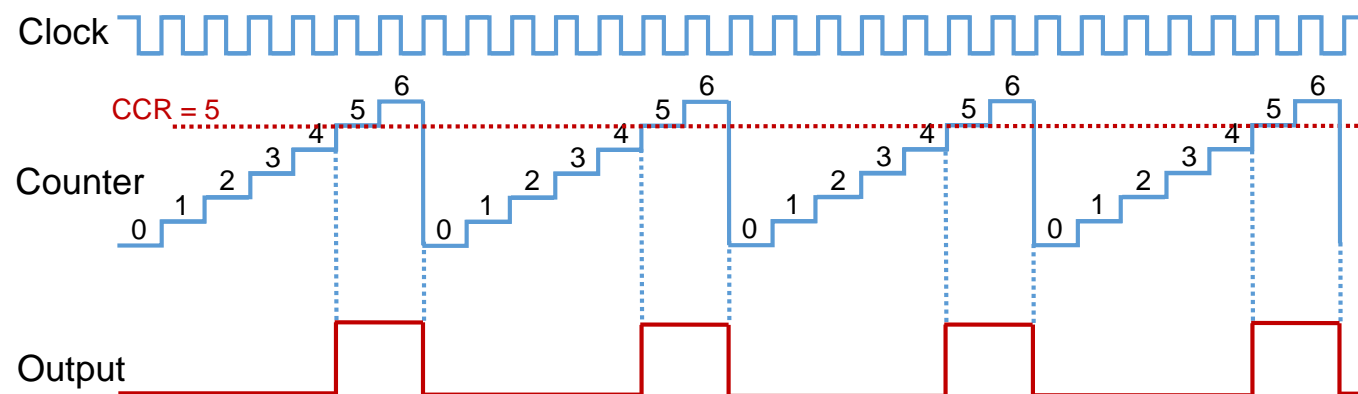


$$\begin{aligned}\text{Duty Cycle} &= 1 - \frac{CCR}{ARR + 1} \\ &= \frac{4}{7}\end{aligned}$$

$$\begin{aligned}\text{Period} &= (1 + ARR) * \text{Clock Period} \\ &= 7 * \text{Clock Period}\end{aligned}$$

# Example 3

- PWM Mode 2 (High-True)
- Upcounting mode, ARR = 6, CCR = 5



$$\begin{aligned}\text{Duty Cycle} &= 1 - \frac{\text{CCR}}{\text{ARR} + 1} \\ &= \frac{2}{7}\end{aligned}$$

$$\begin{aligned}\text{Period} &= (1 + \text{ARR}) * \text{Clock Period} \\ &= 7 * \text{Clock Period}\end{aligned}$$

## Example 4

- Find the frequency (F) and Duty cycle(DC) of a PWM if TIMx\_ARR = 999 and TIMx\_CCRn = 250. Assume OCnM = 110 (PWM1), no prescaler, and TIMx clock frequency of 72MHz.
- Solution:**
- Frequency=72M/(999+1)=72KHz=72000Hz.
- Duty Cycle = TIMx\_CCR/(TIMx\_ARR +1)  $\times$  100% = (250/1000)  $\times$  100% = 25%.

$$\text{PWM Frequency} = \frac{f_{\text{CK\_PSC}}}{(\text{ARR} + 1) \times (\text{PSC} + 1)}$$

$$\text{PWM Duty Cycle} = \text{CCR} / (\text{ARR} + 1) \times 100\%$$

## Example 5

- Assume the TIMx Module clock frequency is 72MHz. Using no prescaler, find the value of the TIMx\_ARR register if we want the PWM output Frequency of (a) 5KHz, (b) 10KHz, and (c) 25KHz.
- Solution
- (a)  $ARR+1=72\text{MHz}/5\text{KHz} = 14400 \rightarrow ARR = 14399$ .
- (b)  $ARR+1=72\text{MHz}/10\text{KHz} = 7200 \rightarrow ARR = 7199$ .
- (c)  $ARR+1=72\text{MHz}/25\text{KHz} = 2880 \rightarrow ARR = 2879$ .

$$\text{PWM Frequency} = \frac{f_{\text{CK\_PSC}}}{(ARR + 1) \times (PSC + 1)}$$

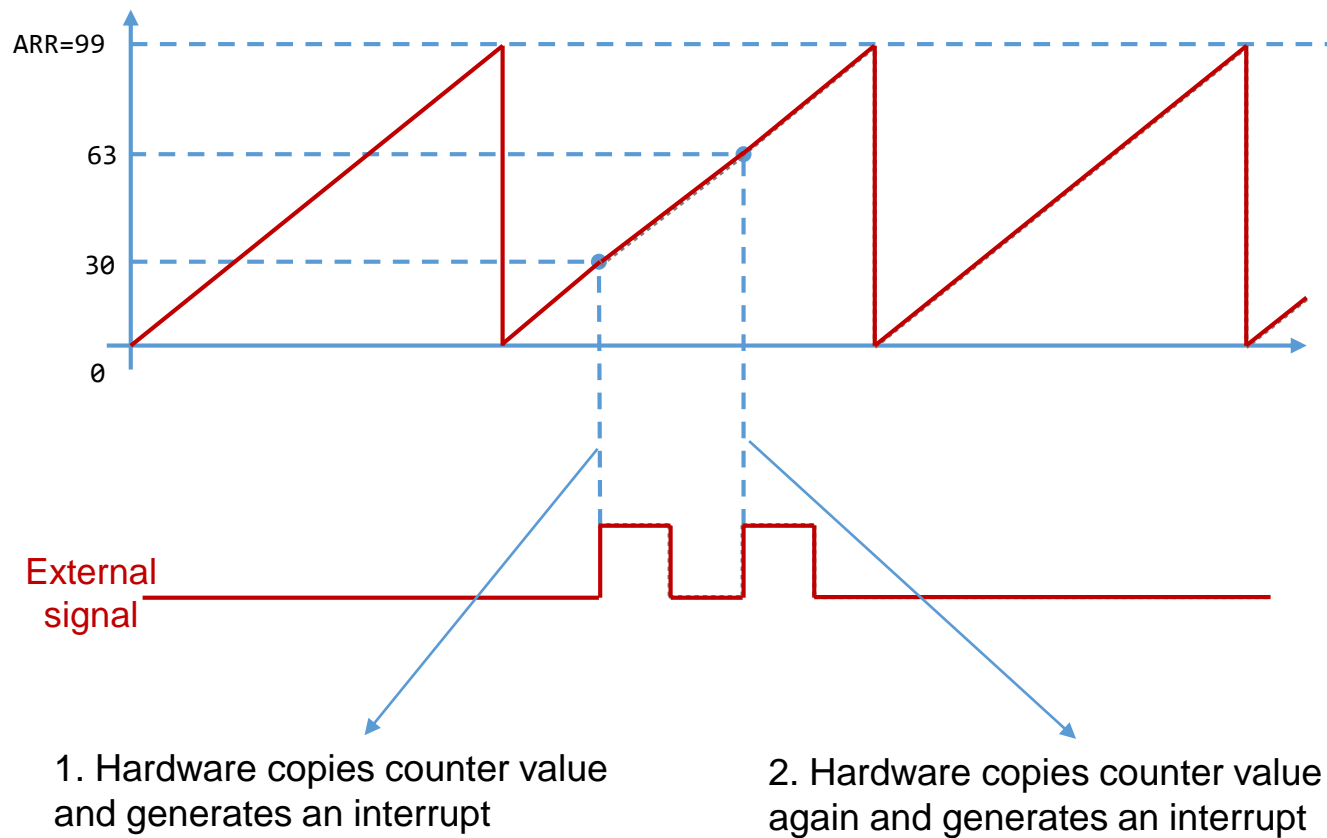
# Outline

- Output Compare PWM
- **Input Capture**
- SysTick



# Measure Time Span

- How to measure time span between two events?

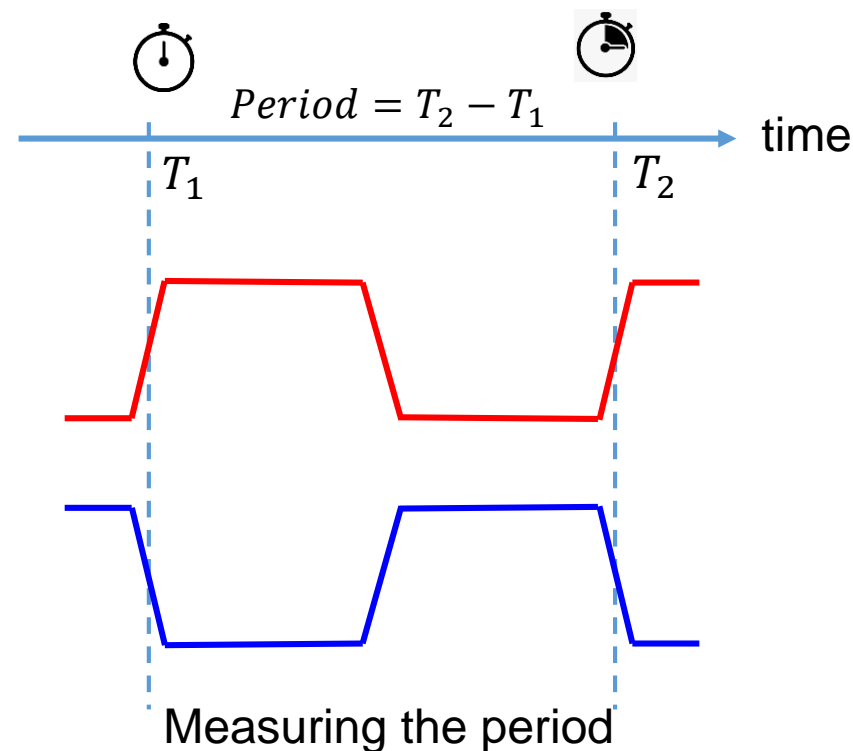
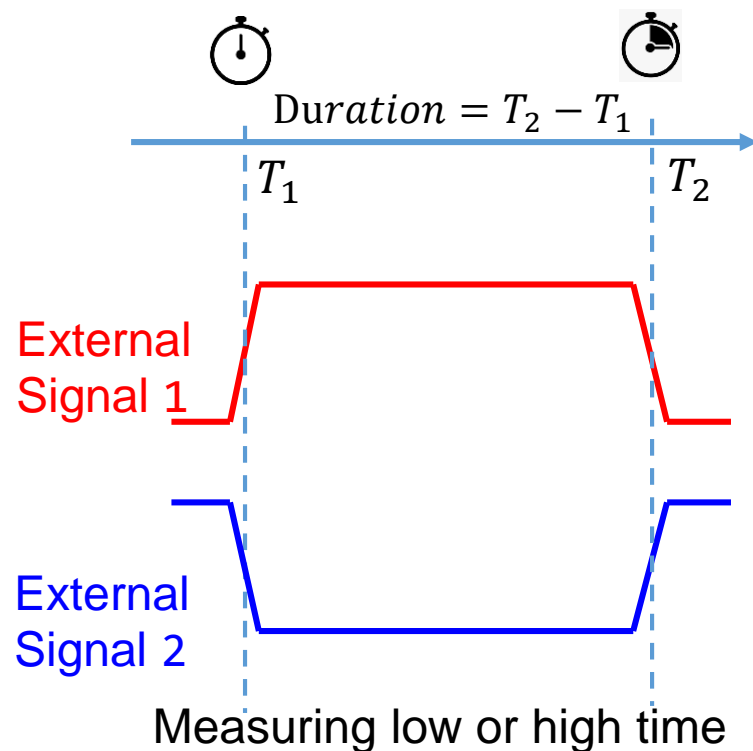


3. Substitute two counter values to get Time span between two events

$$\text{Time span} = (63 - 30) \times \text{Time Unit}$$

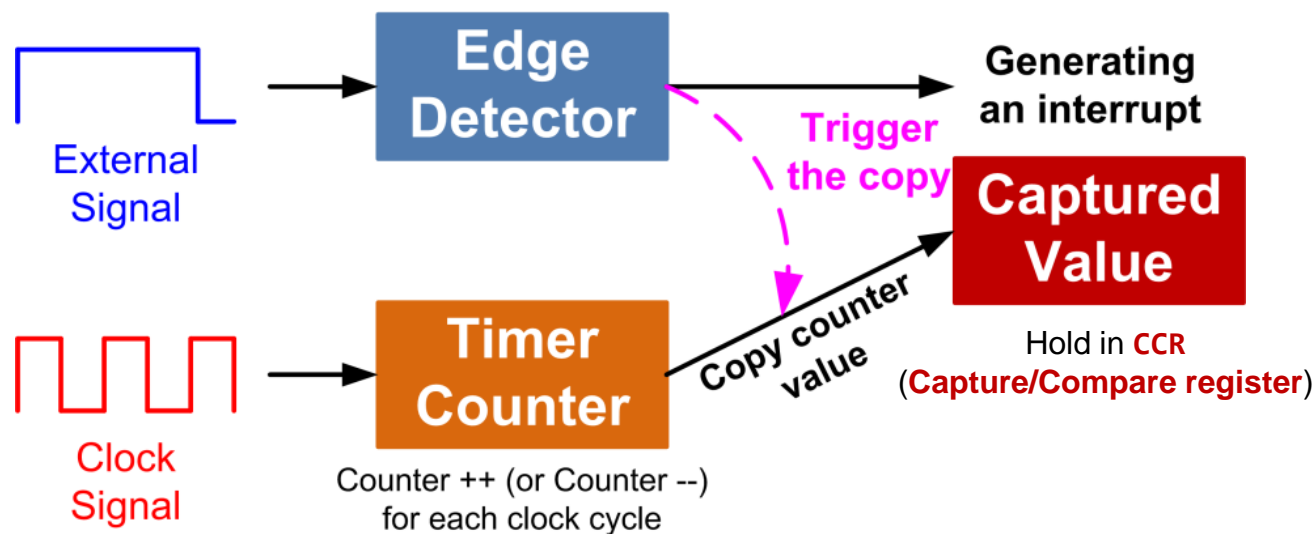
# Input Capture

- A Timer function to record the timestamp of an external event
  - Capture both rising and falling edges
  - Capture only rising edges or only falling edges

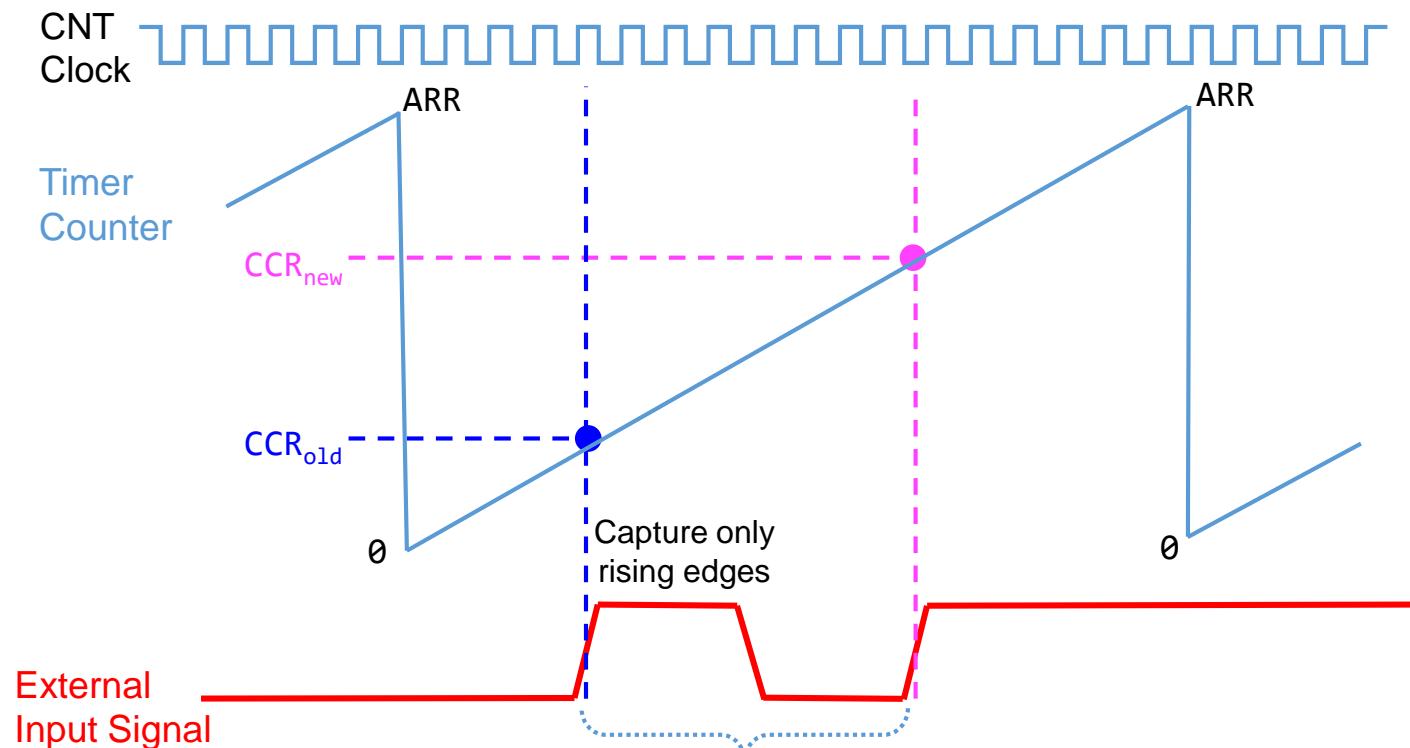


# Input Capture

- Latch the counter value into CCR(Capture & Compare Register) after a transition is detected
- If enabled, generate an interrupt to inform the processor to read CCR



# Measure Time Span



$$Time\ Span = (CCR_{new} - CCR_{old}) \times Clock\ period$$

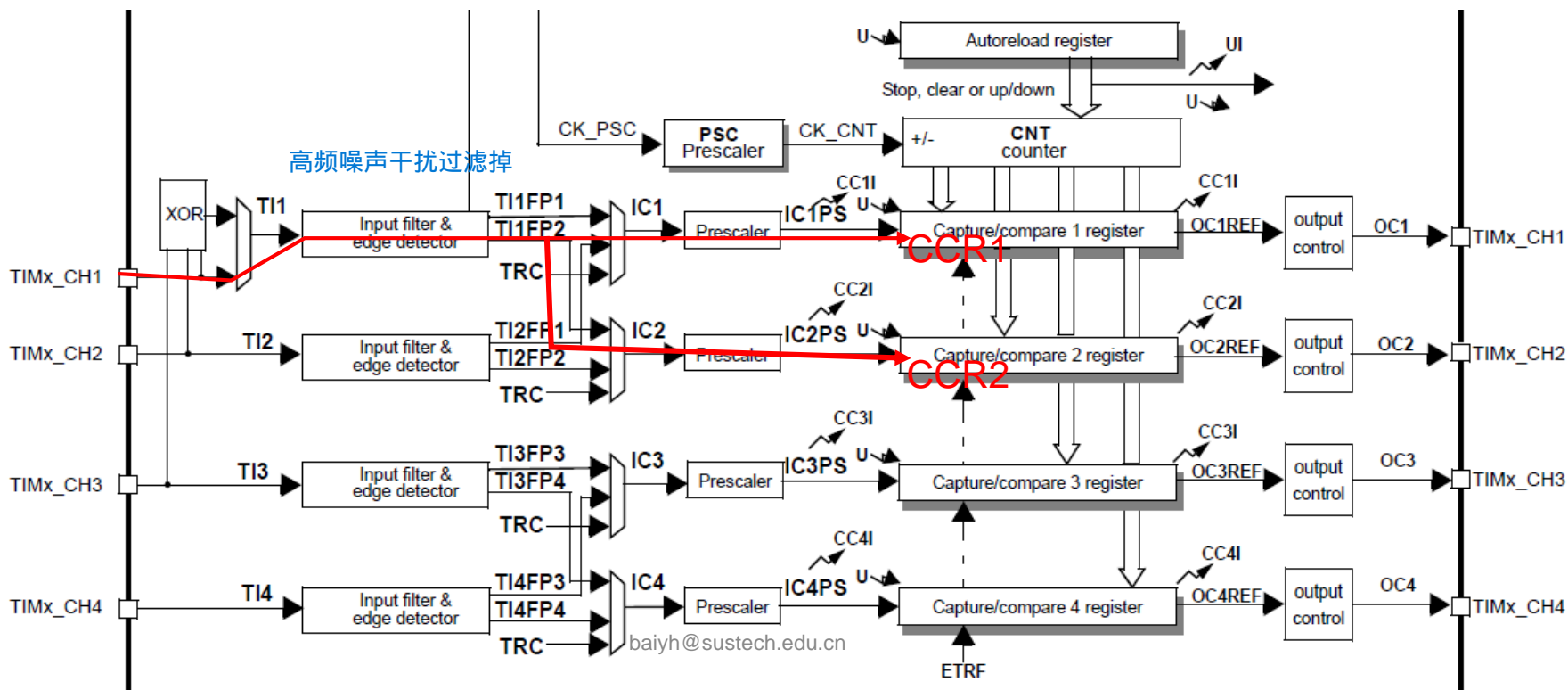
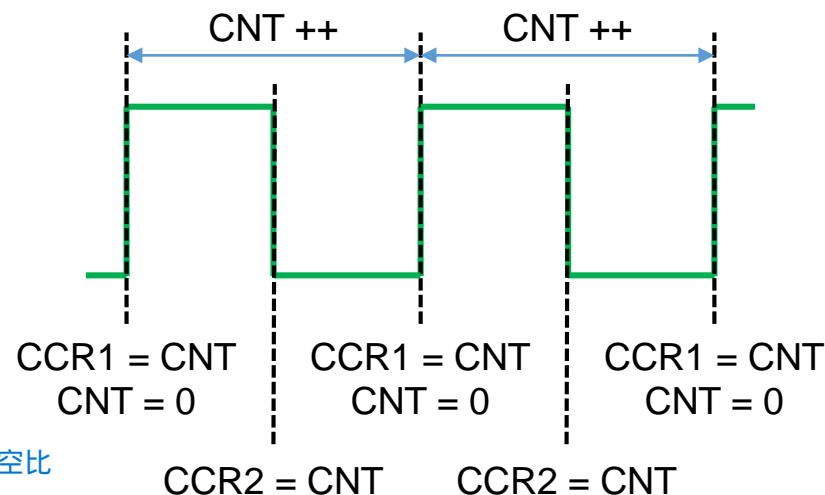
$$Clock\ period = \frac{1}{f_{CK\_CNT}}$$

$$f_{CK\_CNT} = \frac{f_{CK\_PSC}}{PSC + 1}$$

# Input Capture Channels

- Measure frequency and Duty Cycle at the same time

通过两个通道复用，得到两个上升沿和两个下降沿的时间戳，然后新的减去旧的计算占空比



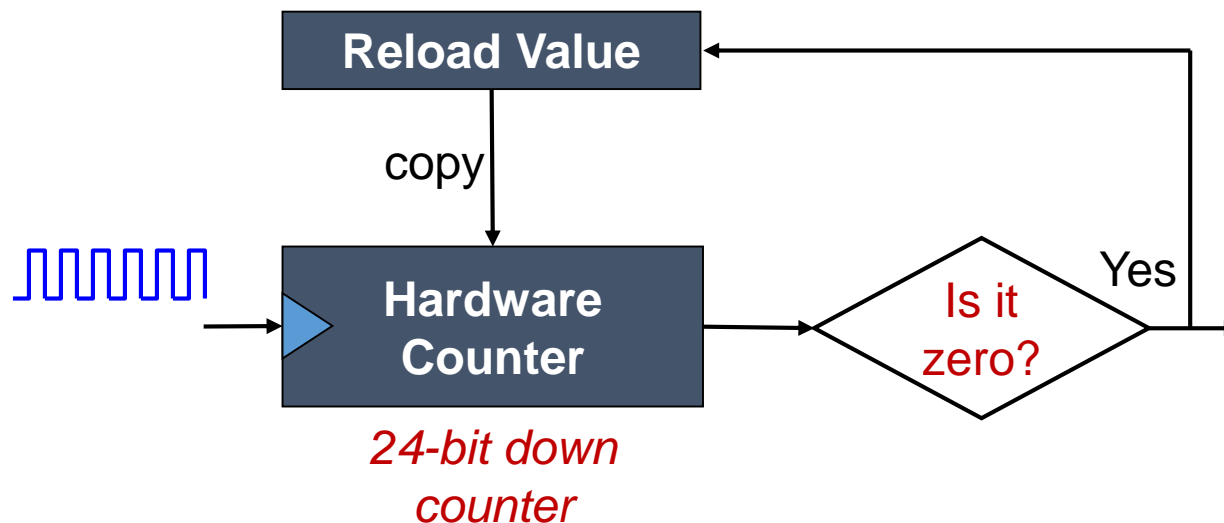
# Outline

- Output Compare PWM
- Input Capture
- **SysTick**

# System Timer (SysTick)

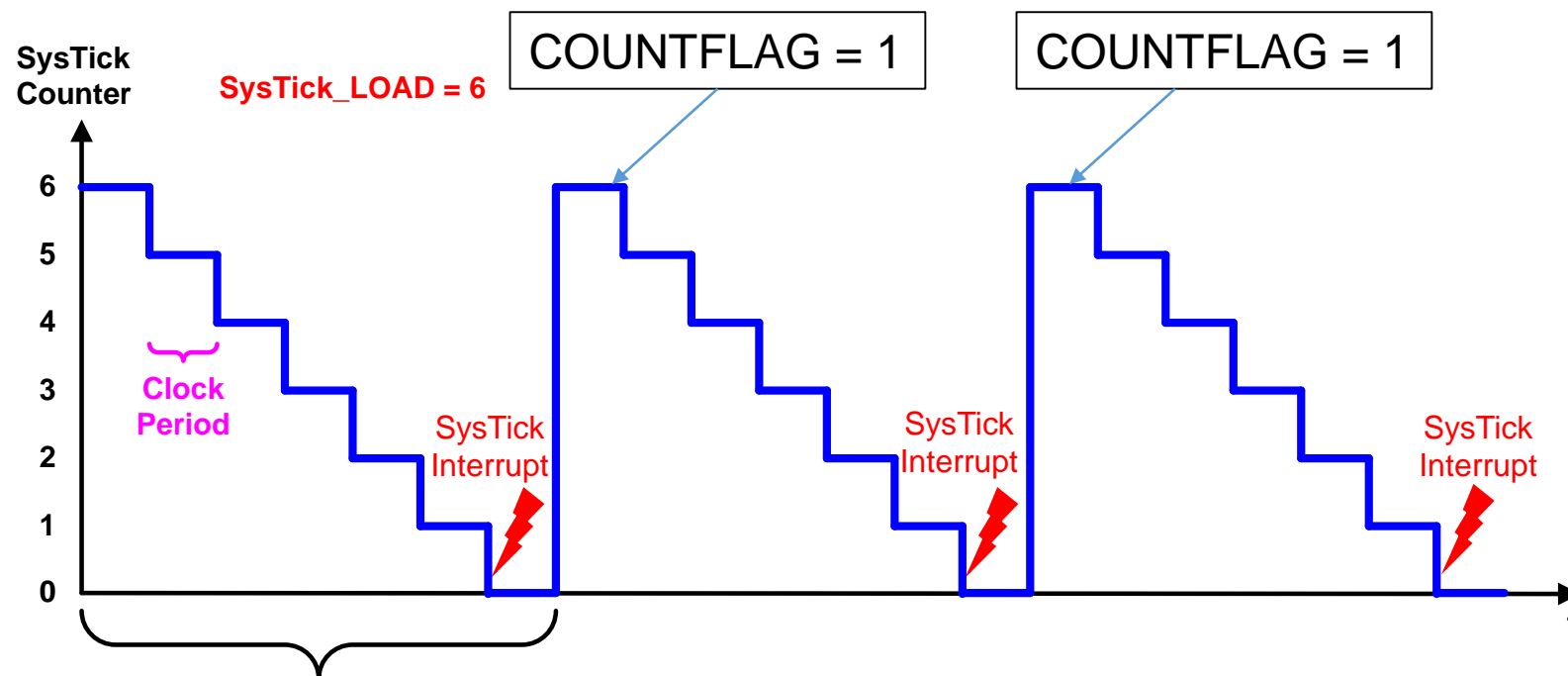
只能作为24 bit的down counter

- Available in all Cortex-M MCUs
- It is a 24-bit down counter. It counts down from an initial value to 0.
- Used to initiate an action on a periodic basis
  - OS ticks



# SysTick Counting

- SysTick only supports downcounting

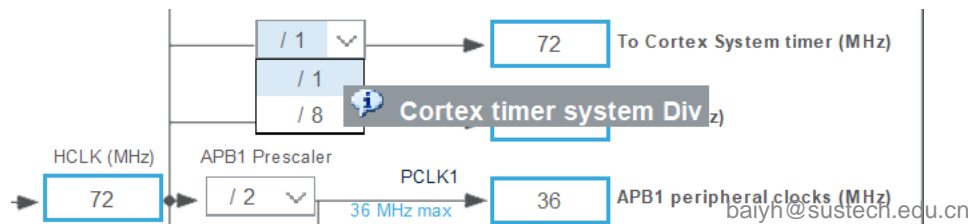
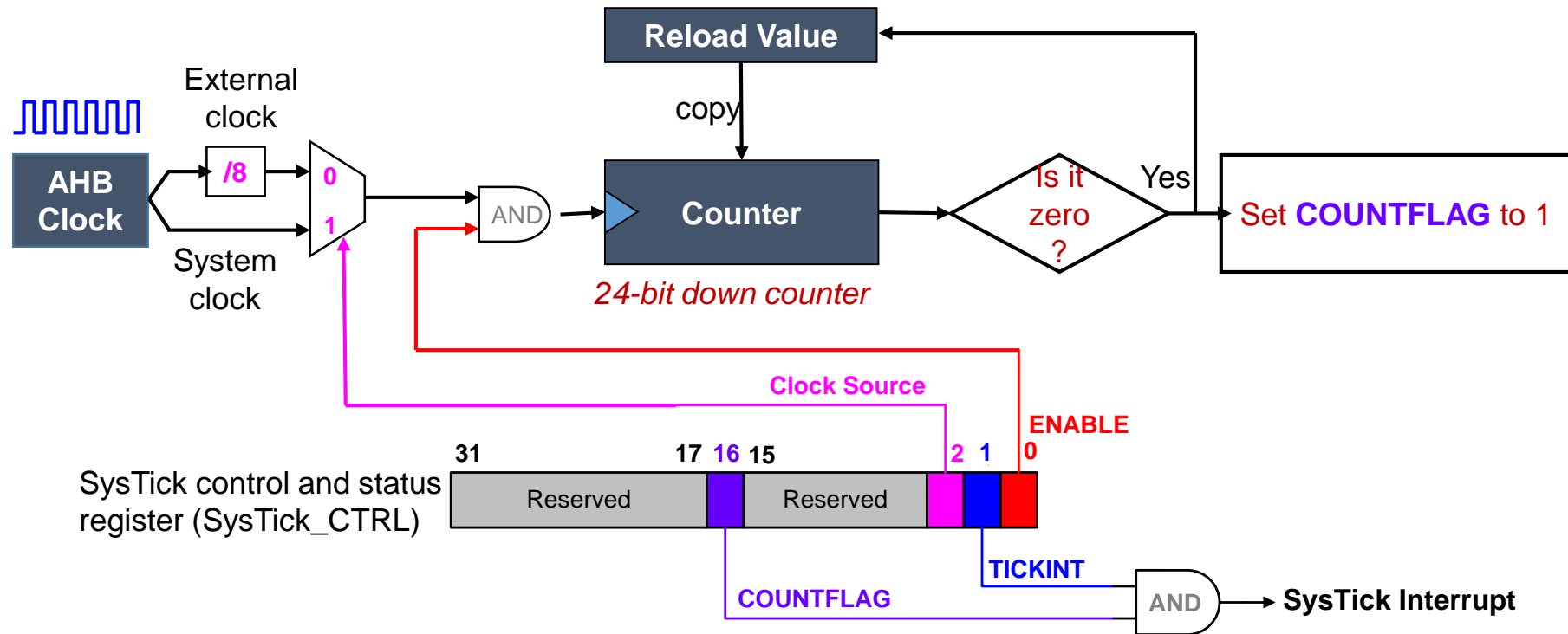


**SysTick Interrupt Time Period = (SysTick\_LOAD + 1) × Clock Period = 7 × Clock Period**



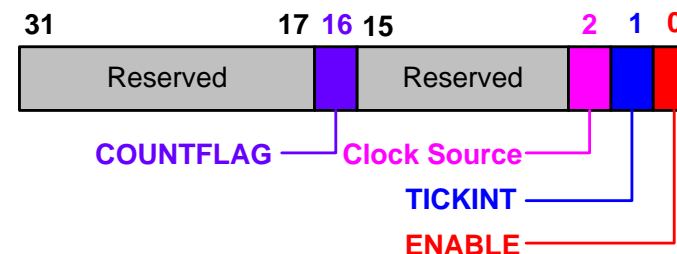
# SysTick Diagram

- Diagram of System Timer



# SysTick Registers

- SysTick\_CTRL
  - SysTick control and status register



Bits	Name	Type	Reset Value	Description
16	COUNTFLAG	R	0	Read as 1 if counter reaches 0 since last time this register is read; clear to 0 automatically when read or when current counter value is cleared
2	CLKSOURCE	R/W	0	0 = External reference clock (STCLK) 1 = Use processor free running clock
1	TICKINT	R/W	0	1 = Enable SYSTICK interrupt generation when SYSTICK Timer reaches 0 0 = Do not generate interrupt
0	ENABLE	R/W	0	SYSTICK Timer enable

# SysTick Registers

- SysTick\_LOAD

- SysTick reload value register



Bits	Name	Type	Reset Value	Description
23:0	RELOAD	R/W	0	Reload value when timer reaches 0

- 24 bits, maximum value 0x00FF.FFFF (16,777,215)
- Counter counts down from RELOAD value to 0.
- Writing RELOAD to 0 disables SysTick
- Time interval between two SysTick interrupts

$$\text{Interval} = (\text{RELOAD} + 1) \times \text{Clock\_Period}$$

- If 100 clock periods between two SysTick interrupts
  - RELOAD = 99

# SysTick Registers

- SysTick\_VAL
  - SysTick current value register



Bits	Name	Type	Reset Value	Description
23:0	CURRENT	R/Wc	0	Read to return current value of the timer. Write to clear counter to 0. Clearing of current value also clears COUNTFLAG in SYSTICK Control and Status register

- Reading it returns the current value of the counter
- When it transits from 1 to 0, it generates an interrupt
- Writing to SysTick\_VAL clears the counter and COUNTFLAG to zero
  - Cause the counter to reload on the next timer clock
  - But, does not trigger an SysTick interrupt
- It has random value on reset.
  - Always clear it before enabling the timer

# Delay Function Example

- Example: Assuming SysTick clock frequency = 8 MHz, calculate the delay which is made by the following function.

```
void delay() {  
    SysTick->LOAD = 9; /*Timer start value*/  
    SysTick->CTRL = 5; /*Enable the timer and choose  
system clock as the clock source */  
    while ((SysTick->CTRL & 0x10000) == 0) /*wait until  
the Count flag is set */  
    {}  
    SysTick->CTRL = 0; /*Stop the timer (Enable = 0) */  
}
```

$$\begin{aligned}\text{Delay} &= (\text{RELOAD} + 1) \times \text{Clock\_Period} \\ &= (9+1) / 8\text{MHz} = 10 \times 0.125\mu\text{s} \\ &= 1.25\mu\text{s} = 1250\text{ns}\end{aligned}$$

# Recall First Sample Code

## • Toggling PA2

```
#include <stm32f10x.h>
```

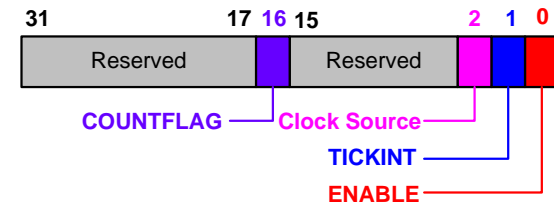
```
void delay_ms(uint16_t t);
```

```
int main()
{
```

```
/* System clock initial */
RCC->APB2ENR |= 0xFC; /* Enable clocks for GPIO ports */
GPIOA->CRL = 0x44444344; /* PA2 as output */
while(1)
{
    GPIOA->ODR ^= (1<<2); /* toggle PA2 */
    delay_ms(1000);
}
}
```

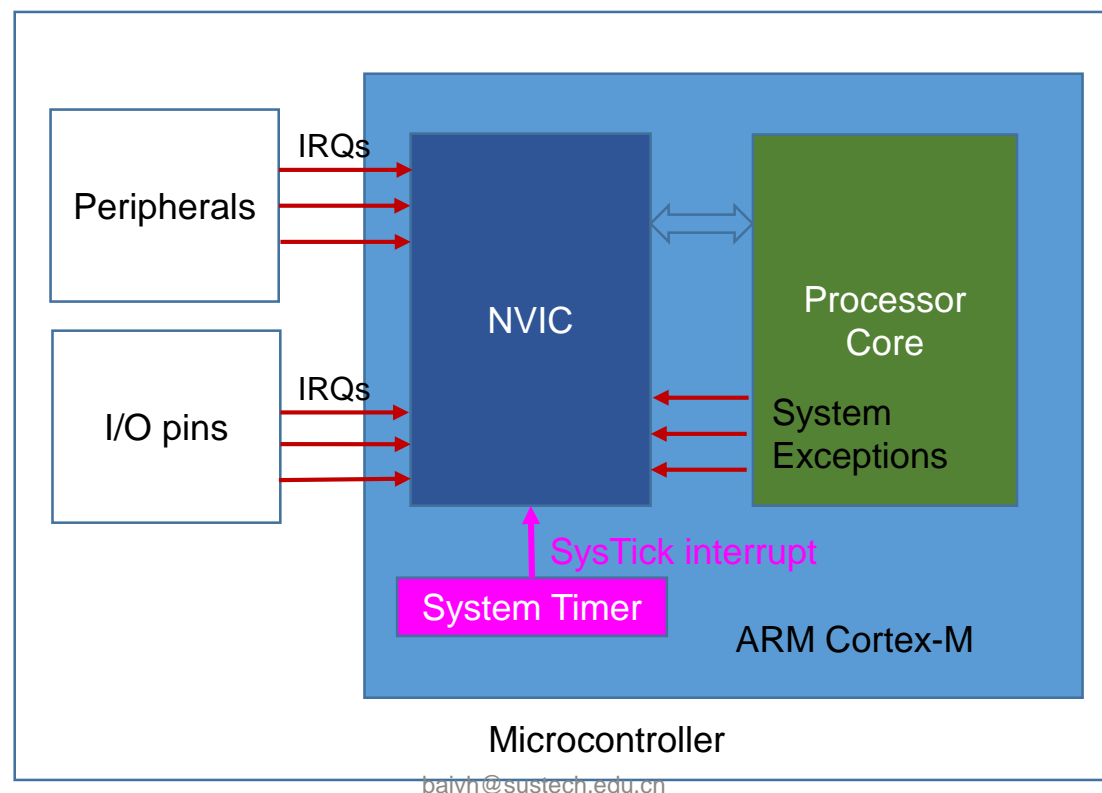
```
201 //延时nms
202 //注意nms的范围
203 //SysTick->LOAD为24位寄存器,所以,最大延时为:
204 //nms<=0xffffffff*8*1000/SYSCLK
205 //SYSCLK单位为Hz,nms单位为ms
206 //对72M条件下,nms<=1864
207 void delay_ms (u16 nms)
208 {
209     u32 temp;
210     SysTick->LOAD= (u32)nms*fac_ms;
211     SysTick->VAL =0x00;
212     SysTick->CTRL=0x01 ;
213     do
214     {
215         temp=SysTick->CTRL;
216     }while ((temp&0x01)&&!(temp&(1<<16))); //等待时间到达
217     SysTick->CTRL=0x00;
218     SysTick->VAL =0x00;
219 }
220 #endif
```

Source code of delay.c



# SysTick Interrupt

- System timer
  - standard hardware component built into ARM Cortex-M.
  - This hardware periodically forces the processor to execute the `SysTick_Handler()` ISR:



# SysTick Interrupt Example

- Toggle PC13 with SysTick interrupt

```
#include <stm32f10x.h>
void SysTick_Handler()
{
    GPIOC->ODR ^= (1 << 13); /* toggle PC13 */
}
int main()
{
    RCC->APB2ENR = 0xFC;          /* enable GPIO clocks */
    GPIOC->CRH = 0x44344444;      /* PC13 as output */
    SysTick->LOAD = 9000000 - 1; /* STRELOAD = 72,000,000/8 -1 */
    SysTick->CTRL = 0x03;          /* Clock = AHB clock/8, TickInt enable,
Enable = 1 */
    while (1)
    {
    }
}
```