第7章　定时器

第1节　概述

一、定时器的作用

二、MSP430内部的定时器

第2节　看门狗定时器(Watch Dog Timer)

一、看门狗定时器作用

二、看门狗定时器特点

第3节　MSP430F1xx的定时器A(Timer_A3)

一、TA的特点和结构图

二、TA的计数定时器单元

三、TA的捕捉/比较器单元

第1节 概述

一、定时器的作用

二、MSP430 内部的定时器

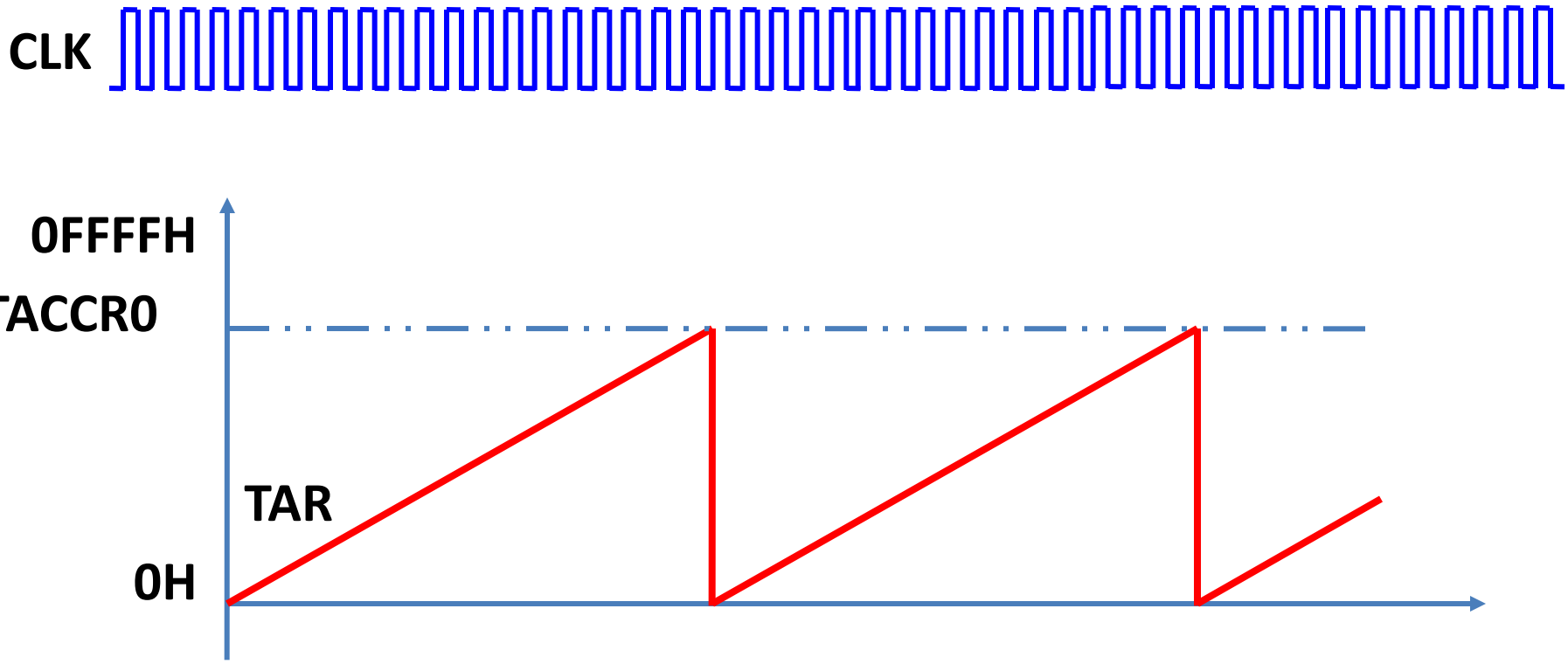
一、定时器的作用

时钟信号CLK是一个周期固定的方波信号(脉冲信号)

利用时钟信号可以做什么?

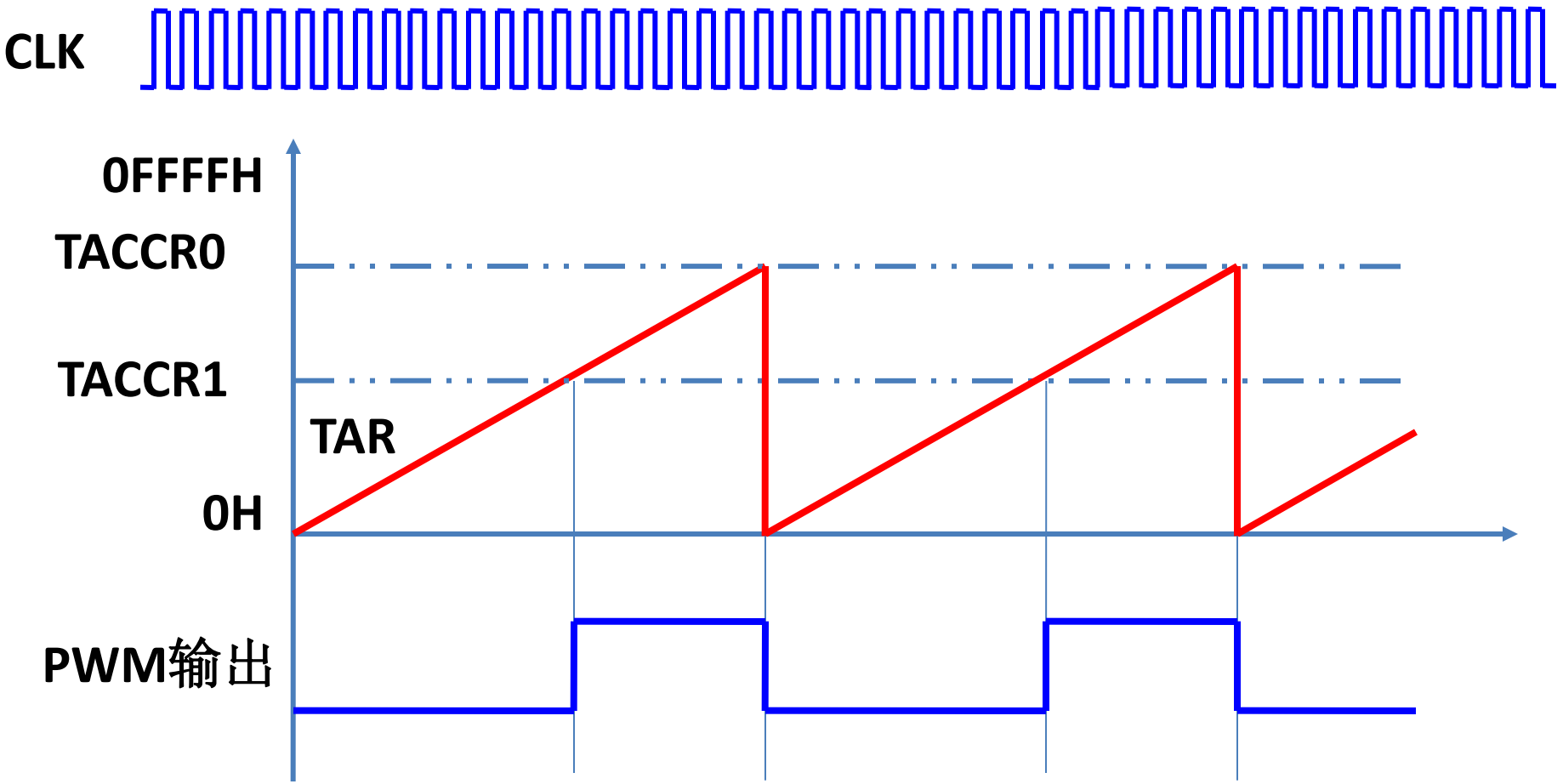
1. 定时
2. PWM脉宽调制(输出比较)
3. 事件发生时刻的捕捉(输入捕捉)
4. 对外部事件计数(输入捕捉)
5. 测量脉宽(输入捕捉)
6. 速度测量、周期/频率测量(输入捕捉)

●定时



- TAR计数脉冲个数，
  到设定的TACCR0值，发中断申请
  定时t= TACCR0*T，　T为一个脉冲的周期
  每来一次中断申请，意味着定时到了
  调整TACCR0可调整定时时间长短

Pulse Width Modulation脉宽调制输出(输出比较)



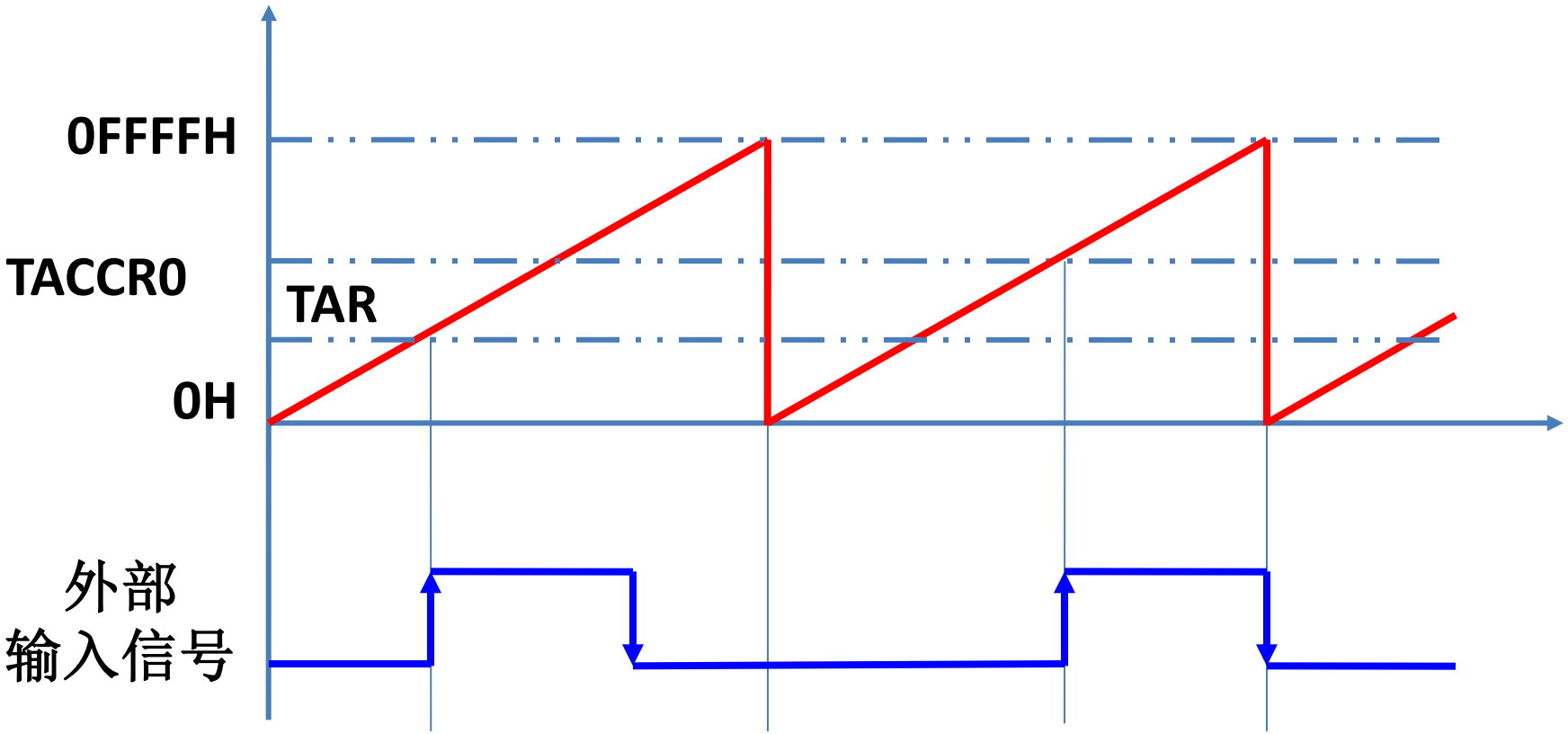
- TAR=TACCR1，输出1

  TAR=TACCR0，输出0，并置TAR=0

  改变TACCR0，改变脉冲周期

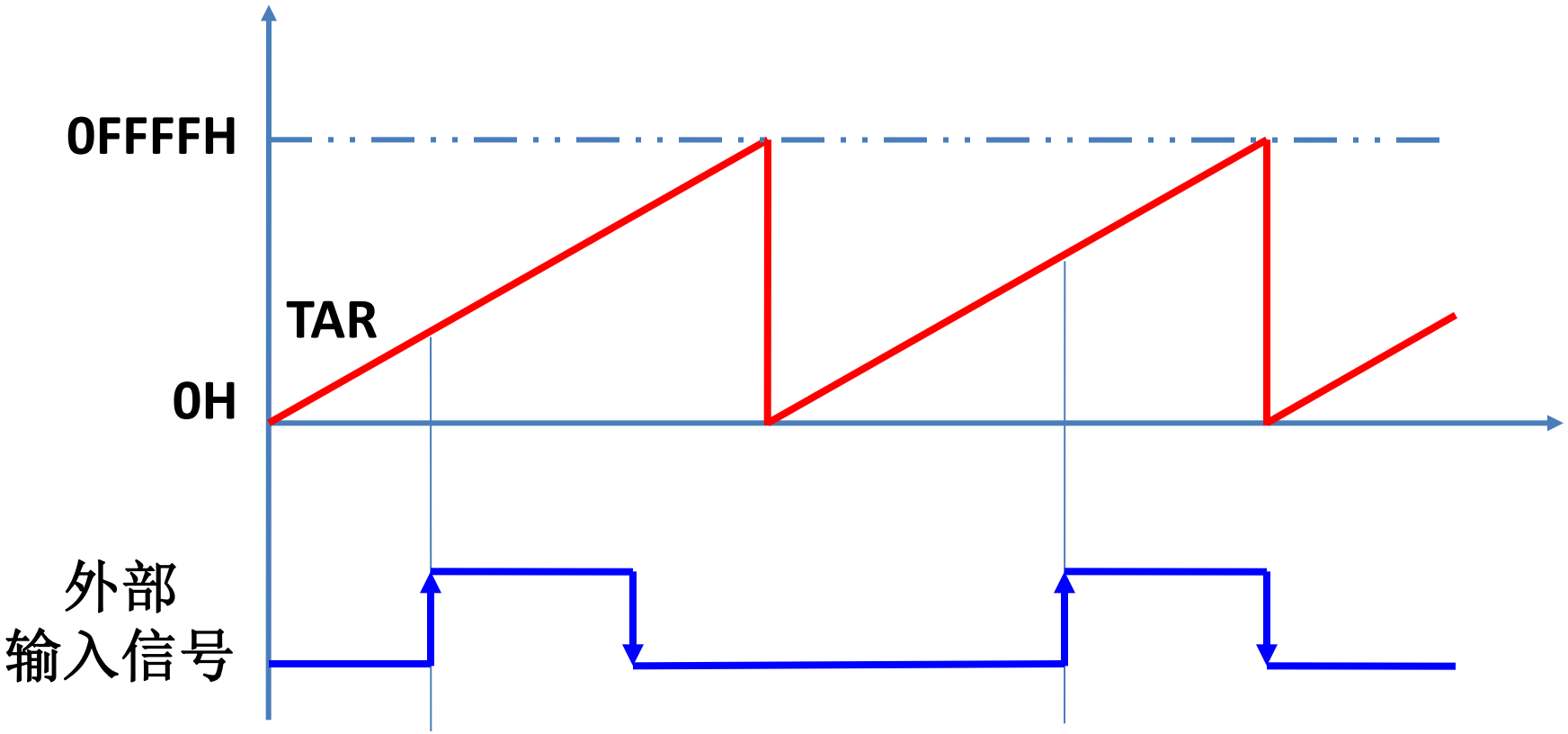
  改变TACCR1，改变占空比

●事件发生时刻的捕捉（输入捕捉)



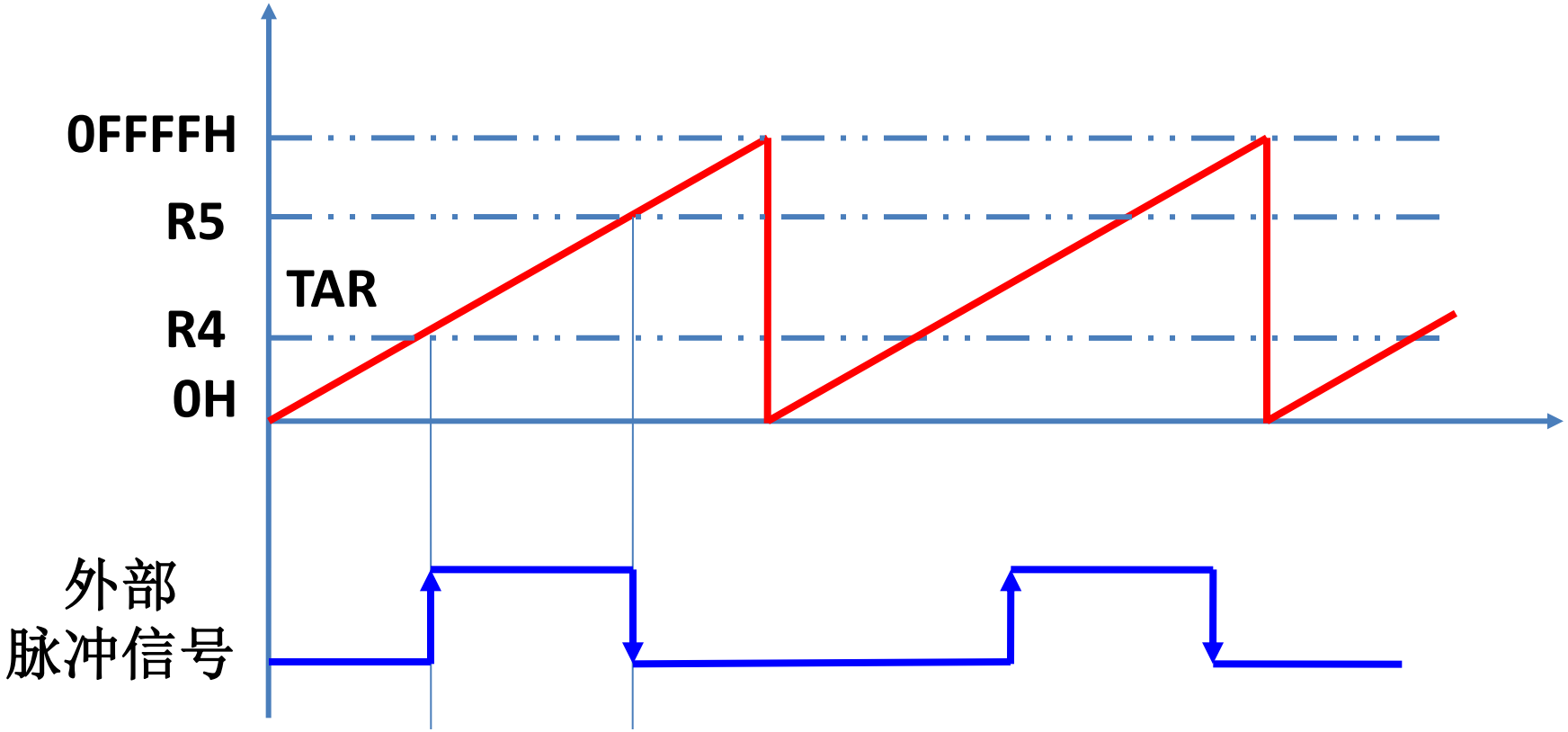
●事件(如上升沿)产生时，
将事件产生时刻的TAR计数值写入TACCR0

●对外部事件计数（输入捕捉)



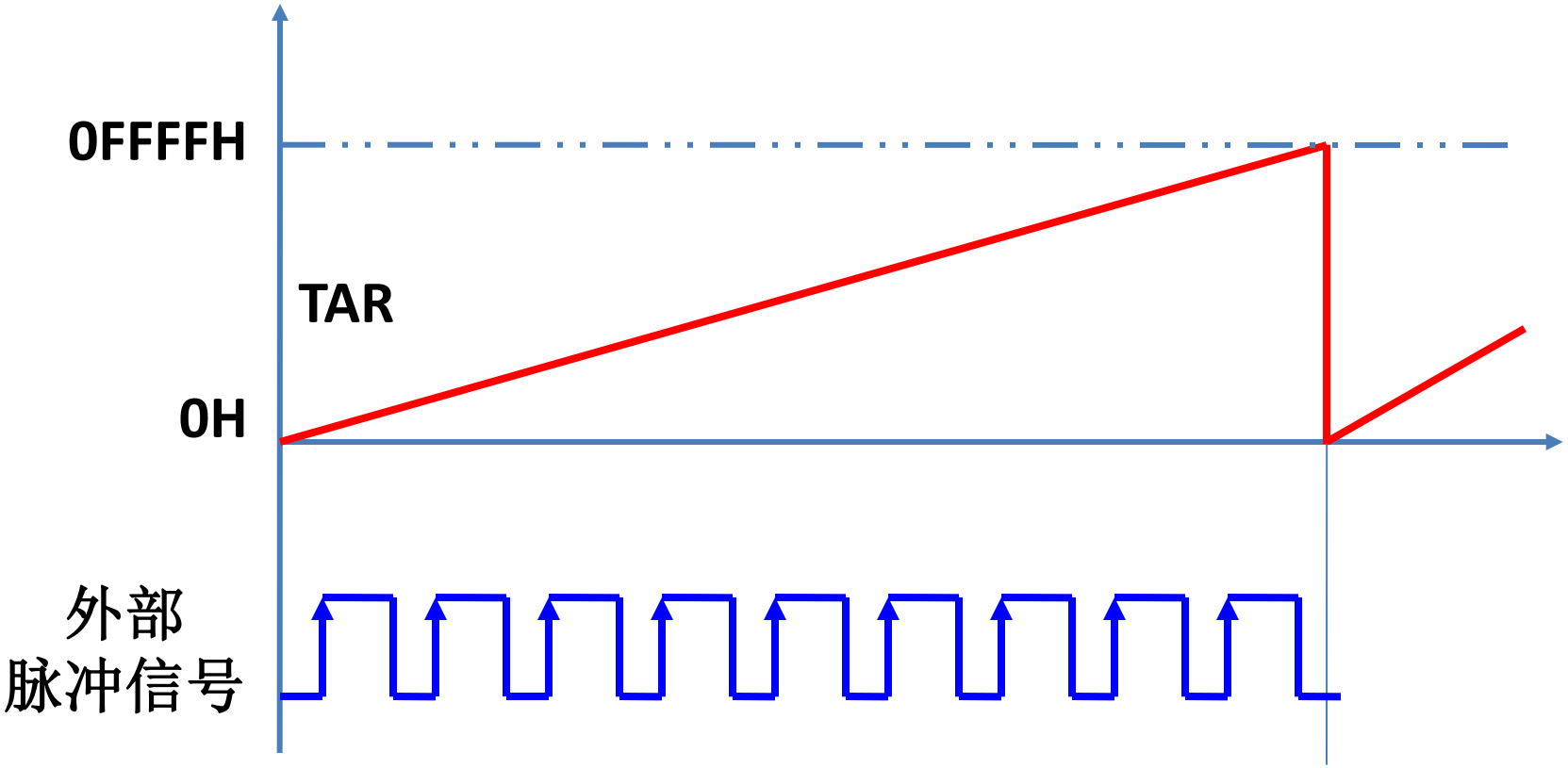
●事件(如上升沿)产生时，发出中断申请，
通过计数中断次数，可对事件计数。

●测量脉宽（输入捕捉）



- 事件产生时（如上升沿，下降沿），
  将上升沿产生时刻的TAR计数值记录下来，如R4
  再将下降沿产生时刻的TAR计数值记录下来，如R5
  两者相减，即可得到脉冲宽度

●利用定时和输入捕捉(或外部中断）结合，
可以进行频率/周期、速度测量



计数一定时间内(如10s)，外部事件发生的次数,即可获取频率

二、MSP430内部的定时器

MSP430单片机含丰富的定时器，各自的特点：

看门狗定时器：定时，可产生复位MCU的中断

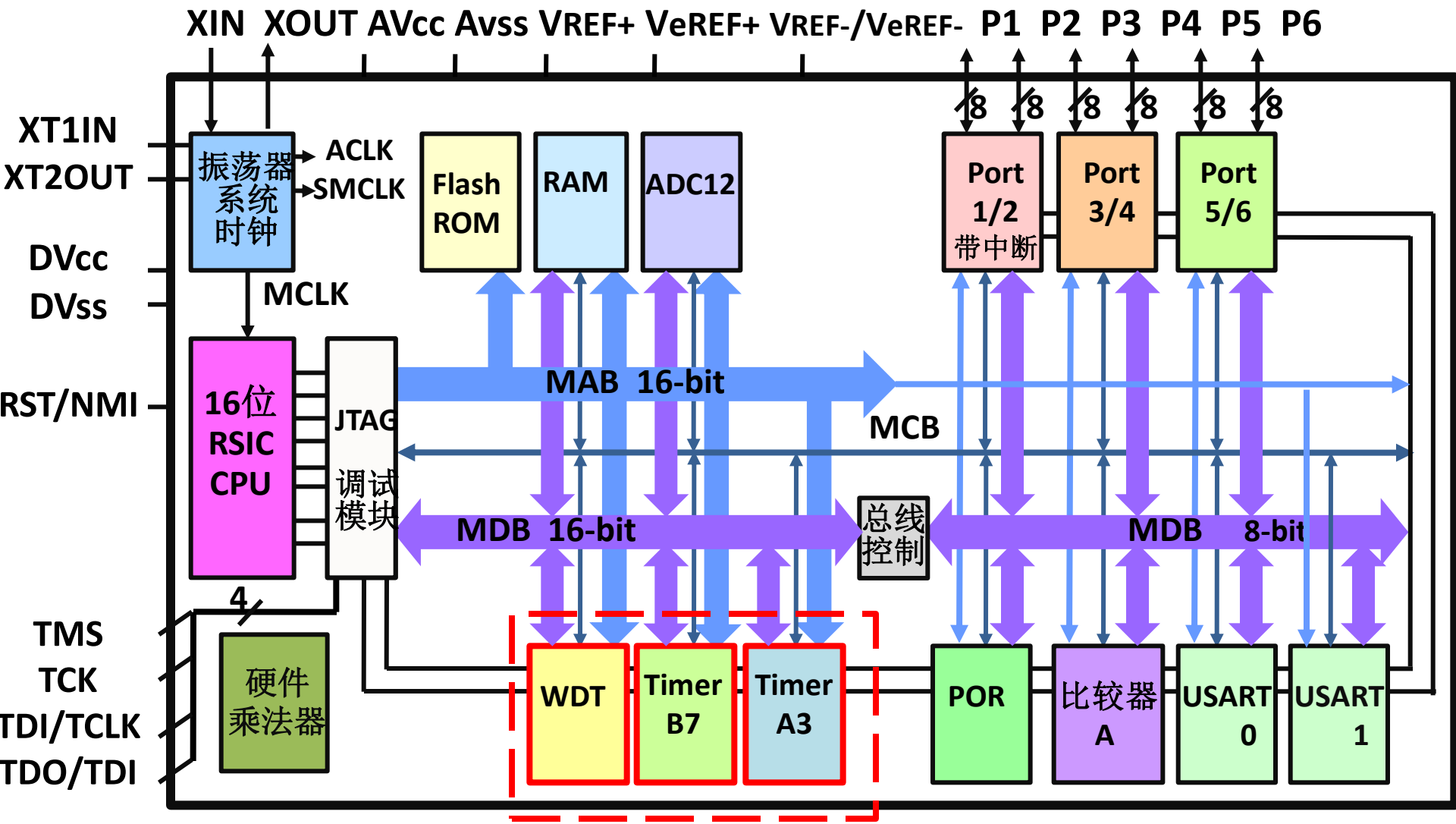
定时器A：　　定时，

具有多个捕捉/比较器、可输出PWW波形

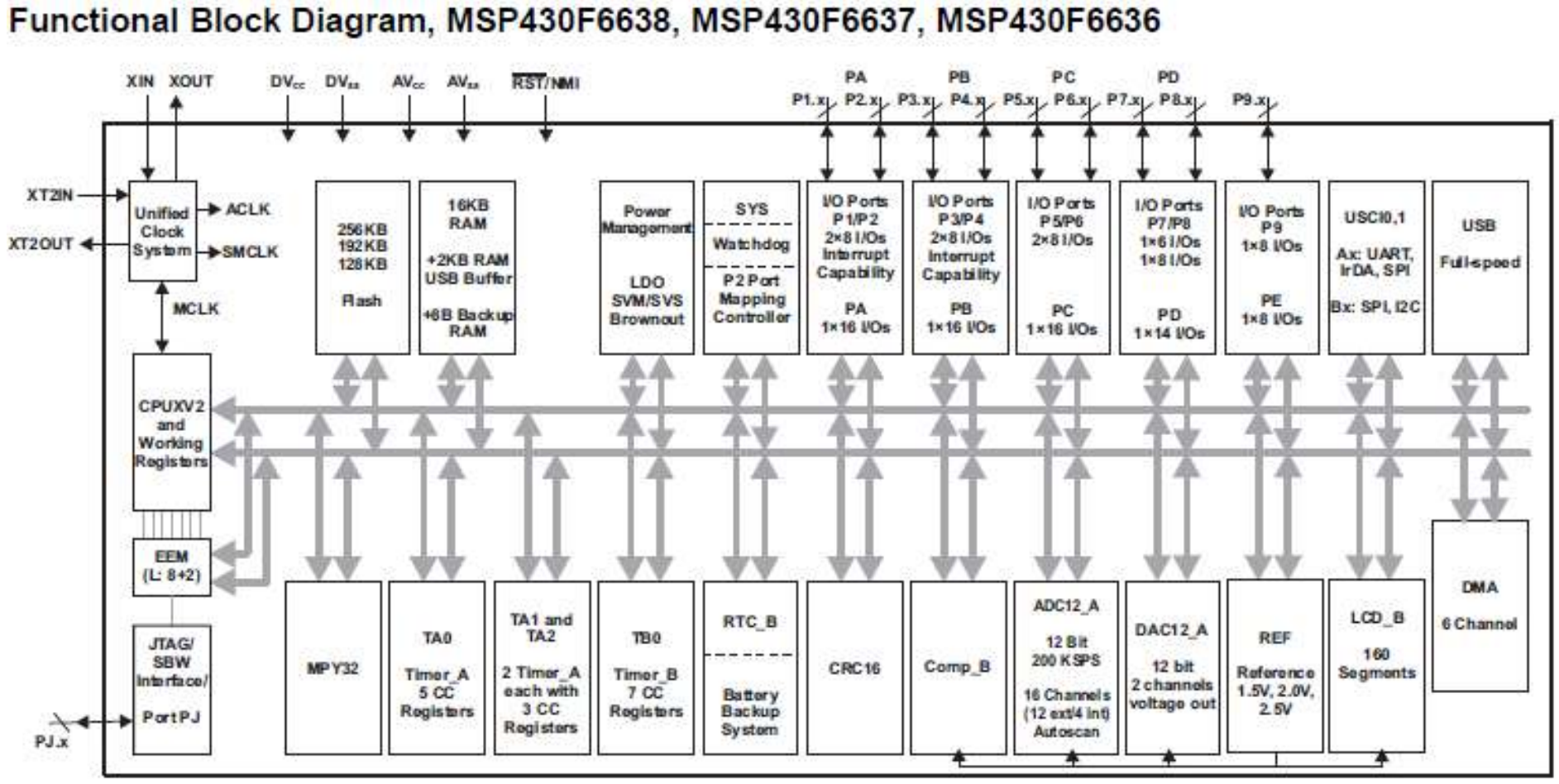
定时器B:　　与定时器A相似，但更灵活，功能更强大

定时器D：　　与定时器B相似，但更灵活，功能更强大

msp430F1xx内含定时器A、定时器B、看门狗定时器



msp430F663x内含定时器A(TA0/TA1/TA2)、定时器B(TB0)、看门狗定时器(WDT)



以下几个方面来掌握定时器：

1. 时钟信号的选择(内部时钟、外部时钟)
2. 计数寄存器开始计数、停止计数的控制
3. 计数溢出的产生、标志置位、分中断允许的设置
4. 比较匹配、输入捕捉相关寄存器的设置
5. 比较匹配、输入捕捉的产生、标志置位、分中断允许的设置
6. 比较匹配对输出波形的控制(PWM)

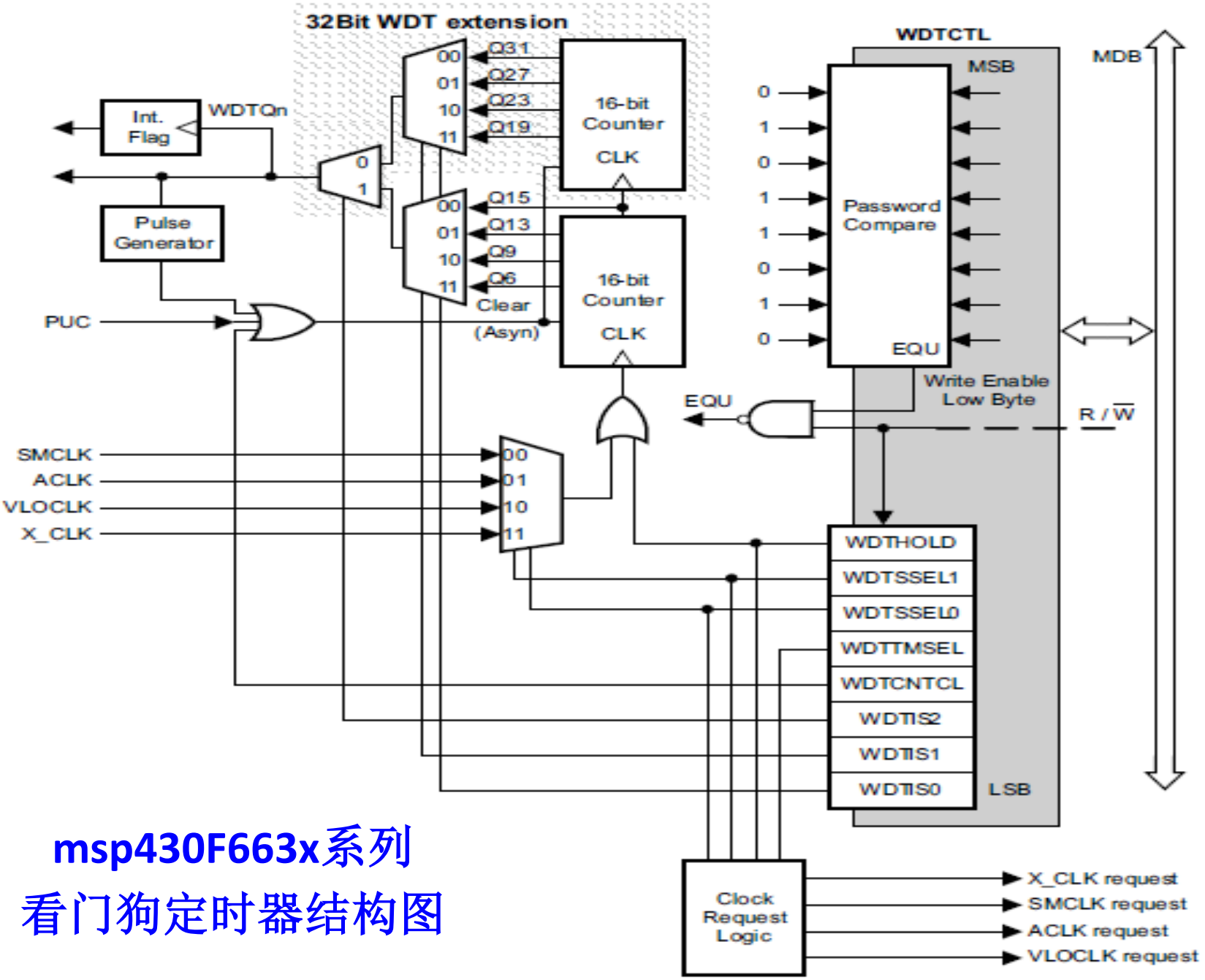
第2节　看门狗定时器(Watch Dog Timer)

一、看门狗定时器WDT作用

二、看门狗定时器特点

一、看门狗定时器的作用

- 由于干扰噪声的出现，造成程序运行“跑飞”，即CPU 的取指执行不在用户程序控制范围内，使程序未能对WDT的计数器清零，产生WDT的计数溢出，可发出中断，复位系统，使CPU重新运行用户程序，系统回到正常工作状态。
- 当不使用WDT的复位功能时，WDT可作为一个定时器使用。



msp430F663x系列

看门狗定时器结构图

Watchdog Timer Control Register (WDTCTL)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Read as 069h WDTPW, Must be written as 05Ah							

7	6	5	4	3	2	1	0
WDTHOLD	**WDTSSEL**		**WDTTMSEL**	**WDTCNTCL**	**WDTIS**		
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-1	rw-0	rw-0

WDTPW Bits 15-8 Watchdog timer password. Always read as 069h. Must be written as 05Ah, or a PUC is generated.

WDTHOLD Bit 7 Watchdog timer hold. This bit stops the watchdog timer. Setting WDTHOLD = 1 when the WDT is not in use conserves power.

- 0 Watchdog timer is not stopped.
- 1 Watchdog timer is stopped.

WDTSSEL Bits 6-5 Watchdog timer clock source select

- 00 SMCLK
- 01 ACLK
- 10 VLOCLK
- 11 X_CLK; VLOCLK in devices that do not support X_CLK

WDTTMSEL Bit 4 Watchdog timer mode select

- 0 Watchdog mode
- 1 Interval timer mode

WDTCNTCL Bit 3 Watchdog timer counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset.

- 0 No action
- 1 WDTCNT = 0000h

WDTIS Bits 2-0 Watchdog timer interval select. These bits select the watchdog timer interval to set the WDTIFG flag and/or generate a PUC.

- 000 Watchdog clock source /2G (18:12:16 at 32 kHz)
- 001 Watchdog clock source /128M (01:08:16 at 32 kHz
- 010 Watchdog clock source /8192k (00:04:16 at 32 kHz)
- 011 Watchdog clock source /512k (00:00:16 at 32 kHz)
- 100 Watchdog clock source /32k (1 s at 32 kHz)
- 101 Watchdog clock source /8192 (250 ms at 32 kHz)
- 110 Watchdog clock source /512 (15,6 ms at 32 kHz)
- 111 Watchdog clock source /64 (1.95 ms at 32 kHz)

二、MSP430 看门狗模块特性

- 16位定时器/计数器WDTCNT
- 8 种可编程选择的定时时间
- 看门狗/定时器两种工作模式
- 控制寄存器带密码保护
- 计数时钟源可编程选择
- 可编程关闭WDT, 降低功耗
- 具备时钟失效保护

看门狗工作模式

- ✓ 该模式下，当计数器**WDTCNT**超过了**WDTIS**设定的定时时间，将产生系统复位中断(类型号63)，使系统复位。

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
System Reset Power-Up, External Reset Watchdog Timeout, Key Violation Flash Memory Key Violation	WDTIFG, KEYV (SYSRSTIV)(1) (2)	Reset	0FFFEh	63, highest
System NMI PMM Vacant Memory Access JTAG Mailbox	SVMLIFG, SVMHIFG, DLYLIFG, DLYHIFG, VLRLIFG, VLRHIFG, VMAIFG, JMBNIFG, JMBOUTIFG (SYSSNIV)(1)	(Non)maskable	0FFFCh	62
User NMI NMI Oscillator Fault Flash Memory Access Violation	NMIIFG, OFIFG, ACCVIFG, BUSIFG (SYSUNIV)(1) (2)	(Non)maskable	0FFFAh	61
Comp_B	Comparator B interrupt flags (CBIV)(1) (3)	Maskable	0FFF8h	60
Timer TB0	TB0CCR0 CCIFG0 (3)	Maskable	0FFF6h	59
Timer TB0	TB0CCR1 CCIFG1 to TB0CCR6 CCIFG6, TB0IFG (TBIV)(1) (3)	Maskable	0FFF4h	58
Watchdog Interval Timer Mode	WDTIFG	Maskable	0FFF2h	57

看门狗工作模式(续）

- ✓ 在看门狗模式下，用户需通过编程完成下面操作：
  1) 初始化 WDTCTL中WDTIS设定的定时时间
  2) 周期性对WDTCNT清零，防止WDT溢出

注意： 复位后，WDT默认设置是看门狗模式，
如果未编程处理WDTCNT计数器，不用看门狗功能，
应在程序开始处禁止看门狗，否则MCU会反复重启，
导致用户程序不能正常运行。

```
int main( void )
{       // Stop watchdog timer to prevent time out reset
        WDTCTL = WDTPW + WDTHOLD;     //关闭看门狗
        ……
}
```

第3节　MSP430F66x的定时器A1和A2(Timer_A3)

一、TA的特点和结构图

二、TA的计数定时器单元

三、TA的捕捉/比较器单元

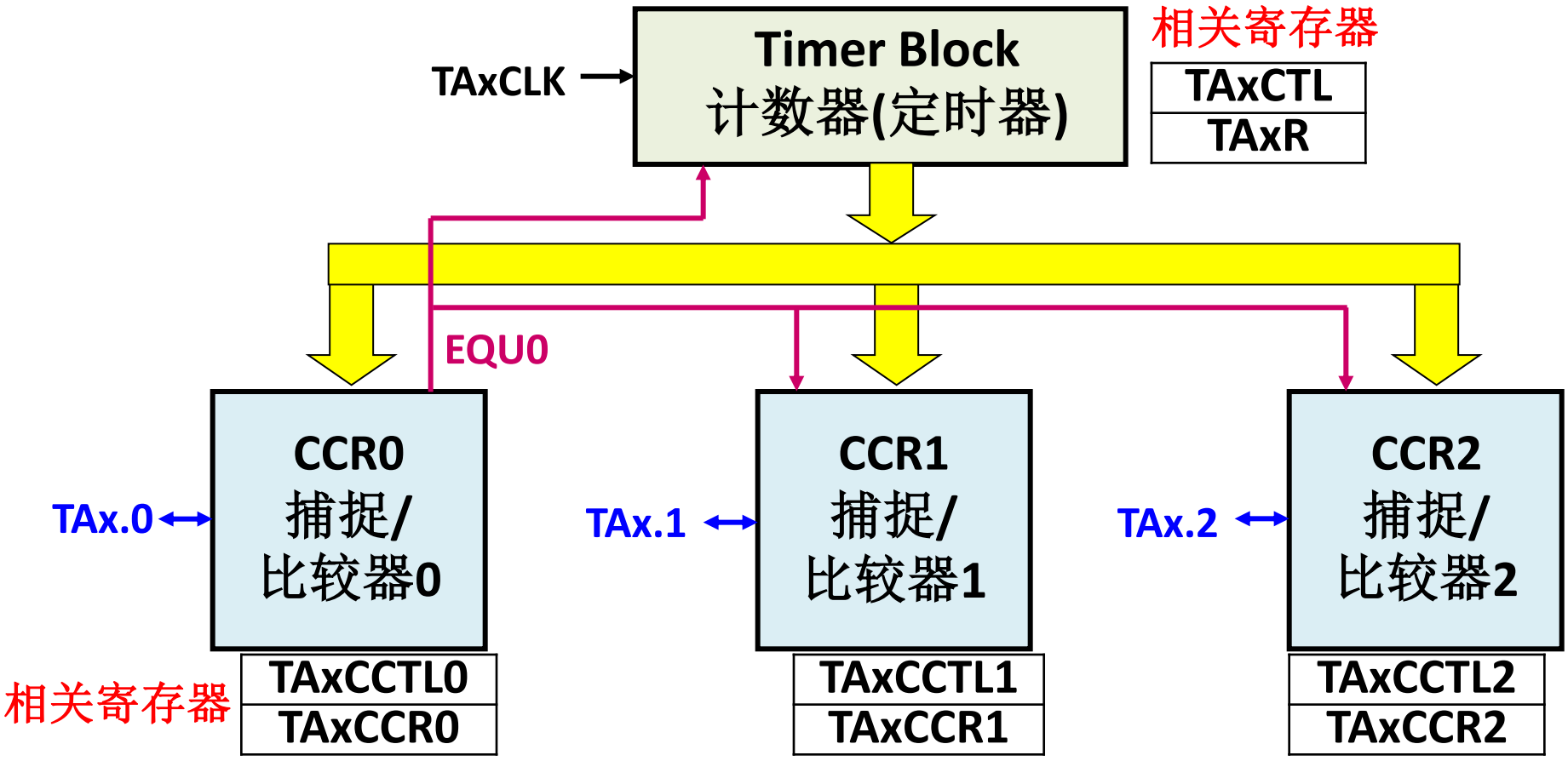
一、TA的特点和结构图

- 16位定时器/计数器，4种工作模式
- 可编程选择的时钟源
- 3个可编程的捕捉/比较寄存器，8 种比较输出方式
- 多种中断功能（计数器溢出、捕捉/比较中断）

中断源	中断标志	向量地址	优先级,类型号
…	…	…	…
定时器A1	TA1CCR0 CCIFG	0FFE2h	49
定时器A1	TA1CCR1 CCIFG TA1CCR2 CCIFG TA1IFG	0FFE0h	48
…	…	…	…
定时器A2	TA2CCR0 CCIFG	0FFD0h	40
定时器A2	TA2CCR1 CCIFG TA2CCR2 CCIFG TA2IFG	0FFCEh	39
…	…	…	…

Timer_A3的结构图

由1个计数定时器单元和3个捕捉/比较器单元构成



↔ 捕捉输入（CCIxA）/比较输出引脚

定时器A相关寄存器

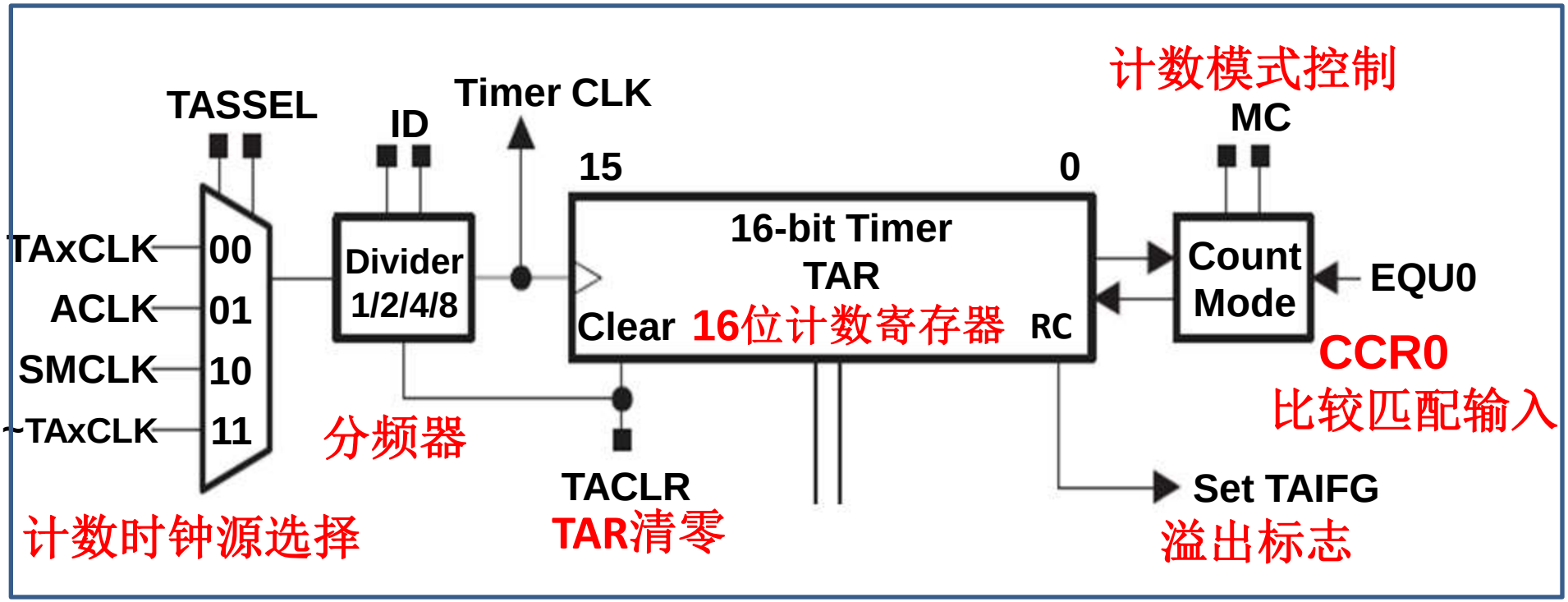
寄 存 器	缩写	读写类型
TAx控制寄存器	TAxCTL	读/写
TAx计数寄存器	TAxR	读/写
TAx捕捉/比较控制寄存器0	TAxCCTL0	读/写
TAx捕捉/比较寄存器0	TAxCCR0	读/写
TAx捕捉/比较控制寄存器1	TAxCCTL1	读/写
TAx捕捉/比较寄存器1	TAxCCR1	读/写
TAx捕捉/比较控制寄存器2	TAxCCTL2	读/写
TAx捕捉/比较寄存器2	TAxCCR2	读/写
TAx中断向量寄存器	TAxIV	只读
Tax扩展寄存器	TAxER0	

CC：Capture/Compare

二、TA的计数器单元(Timer Block)

1. TA计数器单元结构图
2. TA的4种计数模式
3. TA的计数溢出中断TAIFG

1. TA 计数定时器部分结构图



相关寄存器：

TAxCTL：TA控制寄存器

TAxR：TA计数寄存器

TA计数寄存器 TAR

- TAR是1个16位计数寄存器，其内容可读可写;
- Timer CLK时钟 的上升沿触发1次计数器TAR计数，
- 由计数方式决定TAR自动随时钟个数加 1 或减 1、
  到何值时设置溢出中断标志，
- TAR 可由程序设置初值，可由控制寄存器的TACLR 位清零

15	14	13	12	11	10	9	8
TAR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

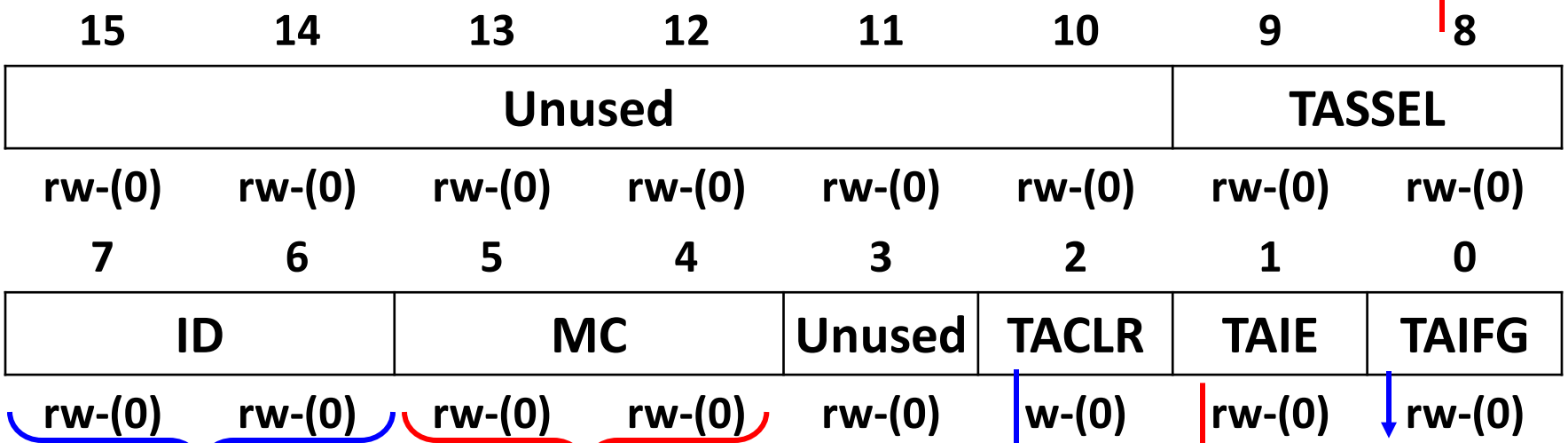
7	6	5	4	3	2	1	0
TAR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

TA控制寄存器 TACTL

Timer_A control Register

控制计数定时器的操作

Source Select
时钟源选择
00: TACLK
01: ACLK
10: SMCLK
11: ~TACLK



分频控制
00: /1
01: /2
10: /4
11: /8
Input Divider

计数方式
00: 停止计数
01: 增计数模式
10: 连续计数模式
11: 增/减计数模式
Mode Control

计数溢出标志
1: 有溢出产生
0: 无溢出产生

计数溢出中断允许
1：允许
0：禁止

1:复位TAR 、分频值和计数方向
该位置1后由硬件自动复位，且读出总为0

msp430F6638.h

用位和值的方式对寄存器的各位、多位的值做了定义，

以便用与、或、以及 +等方式进行对寄存器的编程。

```
/* TAxCTL Control Bits */
#define TASSEL1          (0x0200)      /* Timer A clock source select 1 */
#define TASSEL0          (0x0100)      /* Timer A clock source select 0 */
#define ID1              (0x0080)      /* Timer A clock input divider 1 */
#define ID0              (0x0040)      /* Timer A clock input divider 0 */
#define MC1              (0x0020)      /* Timer A mode control 1 */
#define MC0              (0x0010)      /* Timer A mode control 0 */
#define TACLR            (0x0004)      /* Timer A counter clear */
#define TAIE             (0x0002)      /* Timer A counter interrupt enable */
#define TAIFG            (0x0001)      /* Timer A counter interrupt flag */
……
#define MC__STOP         (0*0x10u)     /* Timer A mode control: 0 - Stop */
#define MC__UP           (1*0x10u)     /* Timer A mode control: 1 - Up to CCR0 */
#define MC__CONTINUOUS   (2*0x10u)     /* Timer A mode control: 2 - Continuous up */
#define MC__CONTINOUS    (2*0x10u)     /* Legacy define */
#define MC__UPDOWN       (3*0x10u)     /* Timer A mode control: 3 - Up/Down */
……
#define TASSEL__TACLK    (0*0x100u)    /* Timer A clock source select: 0 - TACLK */
#define TASSEL__ACLK     (1*0x100u)    /* Timer A clock source select: 1 - ACLK  */
#define TASSEL__SMCLK    (2*0x100u)    /* Timer A clock source select: 2 - SMCLK */
#define TASSEL__INCLK    (3*0x100u)    /* Timer A clock source select: 3 - INCLK */
```

2. TA 的4种计数方式 Count Mode

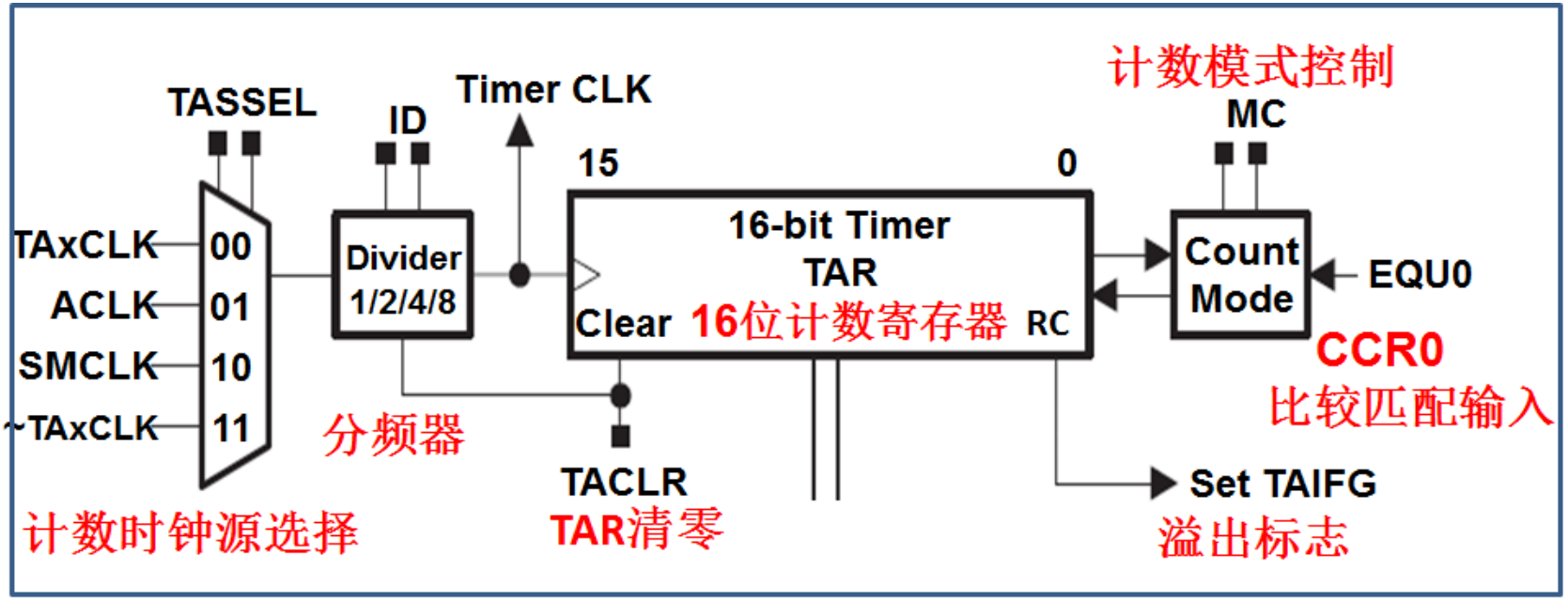
MC

0 0: 停止计数 stop mode

0 1: 增计数模式 up mode

1 0: 连续计数模式 continuous mode

1 1: 增/减计数模式 up/down mode



1) 停止方式　MCx= 00

Stop mode : the timer is halted

- 定时器暂停计数，不复位TA，

  所有寄存器的内容在停止方式结束后都可用。

- 当不使用Timer时，

  将Timer配置为Stop mode ， 可降低芯片的功耗

- MCU复位后定时器 A 的计数方式为 stop mode

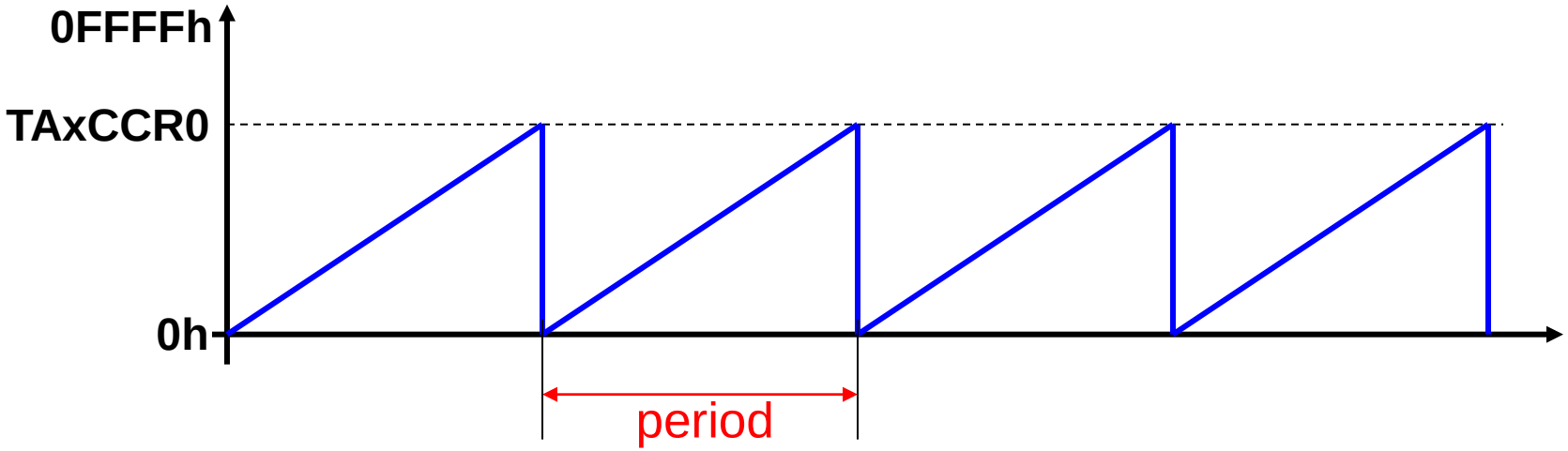
例　编程设置Timer A1为停止方式

C语言　　TA1CTL = MC__STOP;

2) 增计数方式(锯齿波方式)　MC= 01

Up mode　必需要**CCR0(比较方式)**协助

Timer CLK的上升沿触发 TAxR加1，

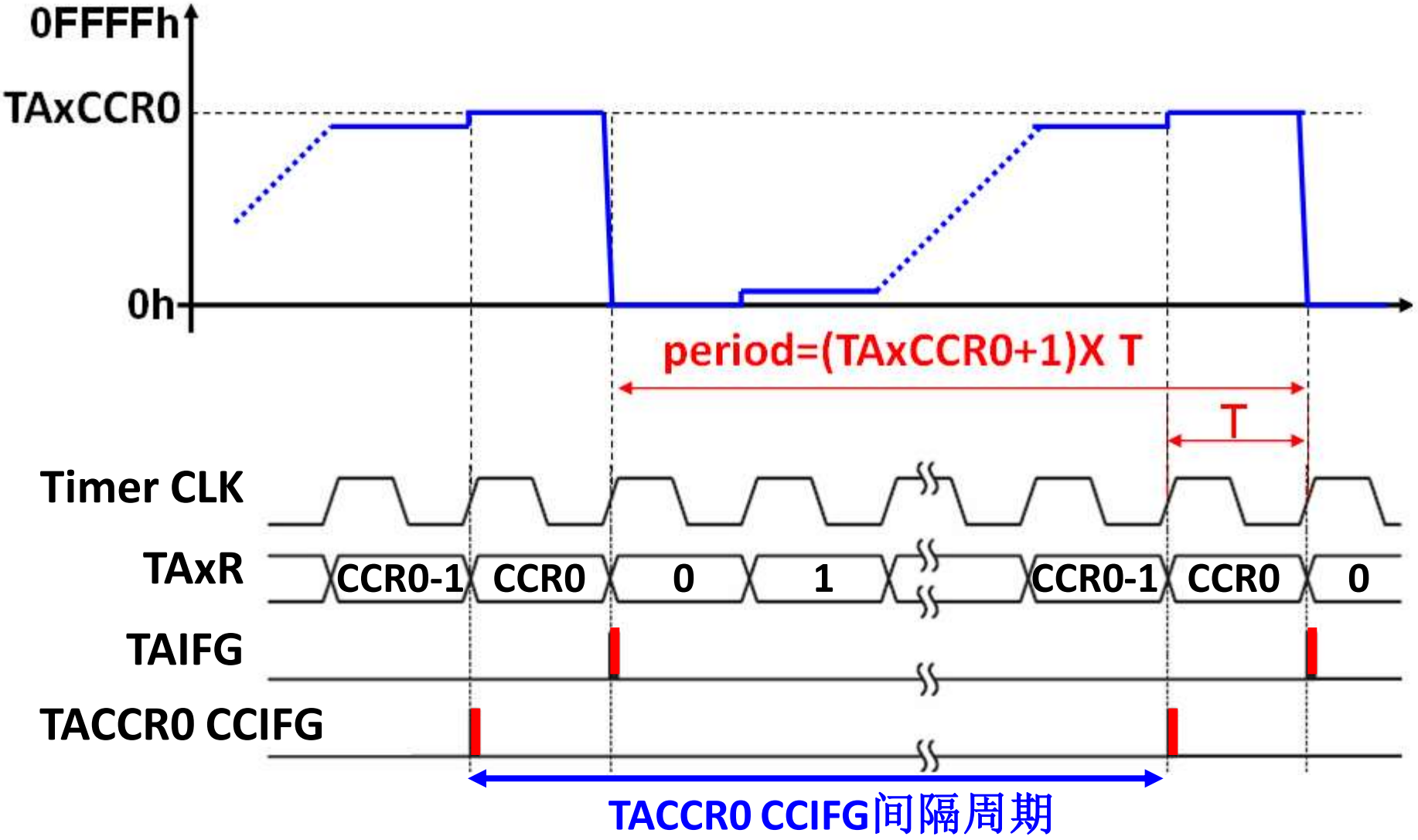


period = (TAxCCR0 +1)×T

T = Timer Clock 的周期

TA1CTL = MC__UP ;

增计数方式下标志位的设置



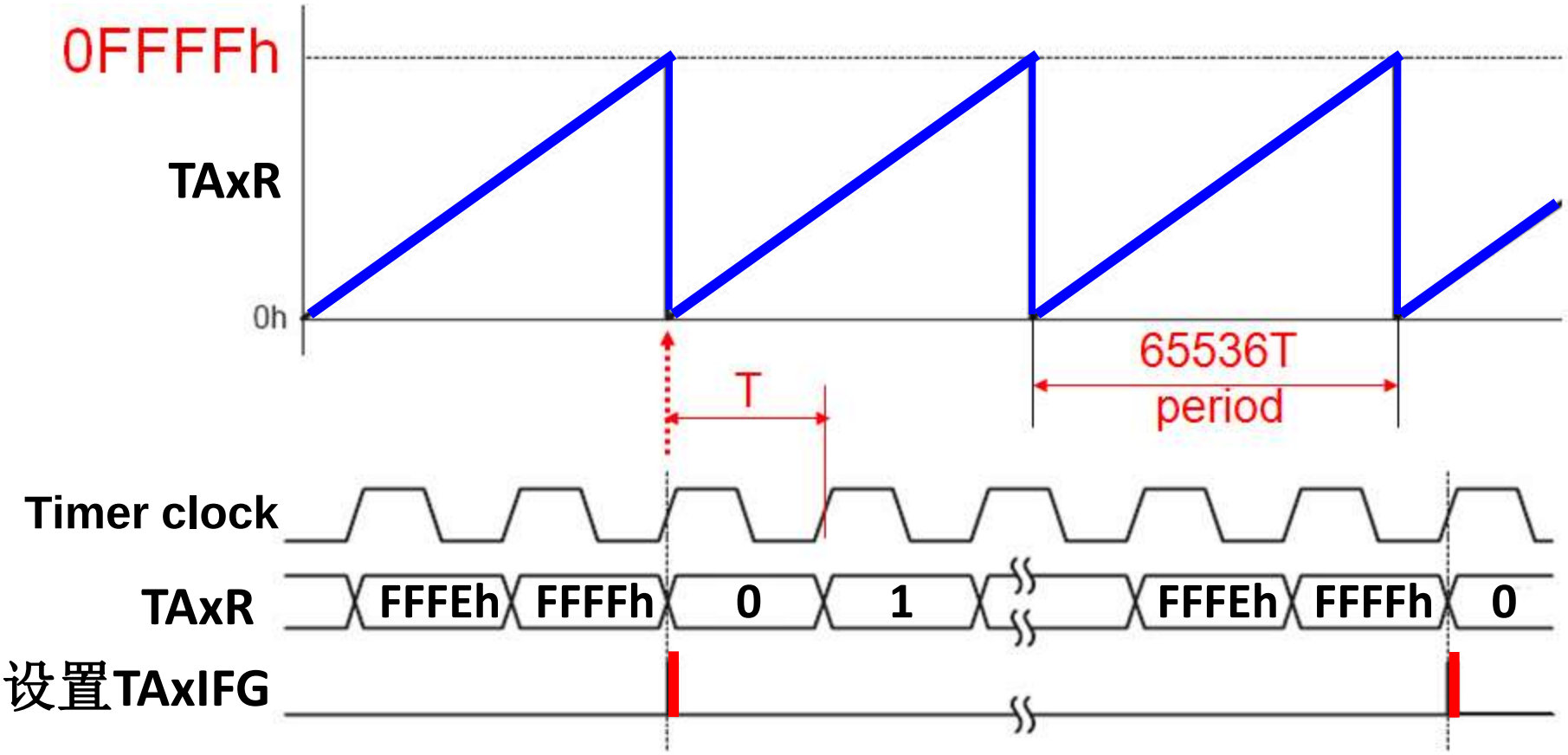
当TAxR计数到 TACCR0时，置TACCR0 CCIFG =1

当TAxR计数从值TACCR0变为0时， 置 TAIFG =1

3) 连续计数方式(最大锯齿波方式)　MC= 10

Continuous mode

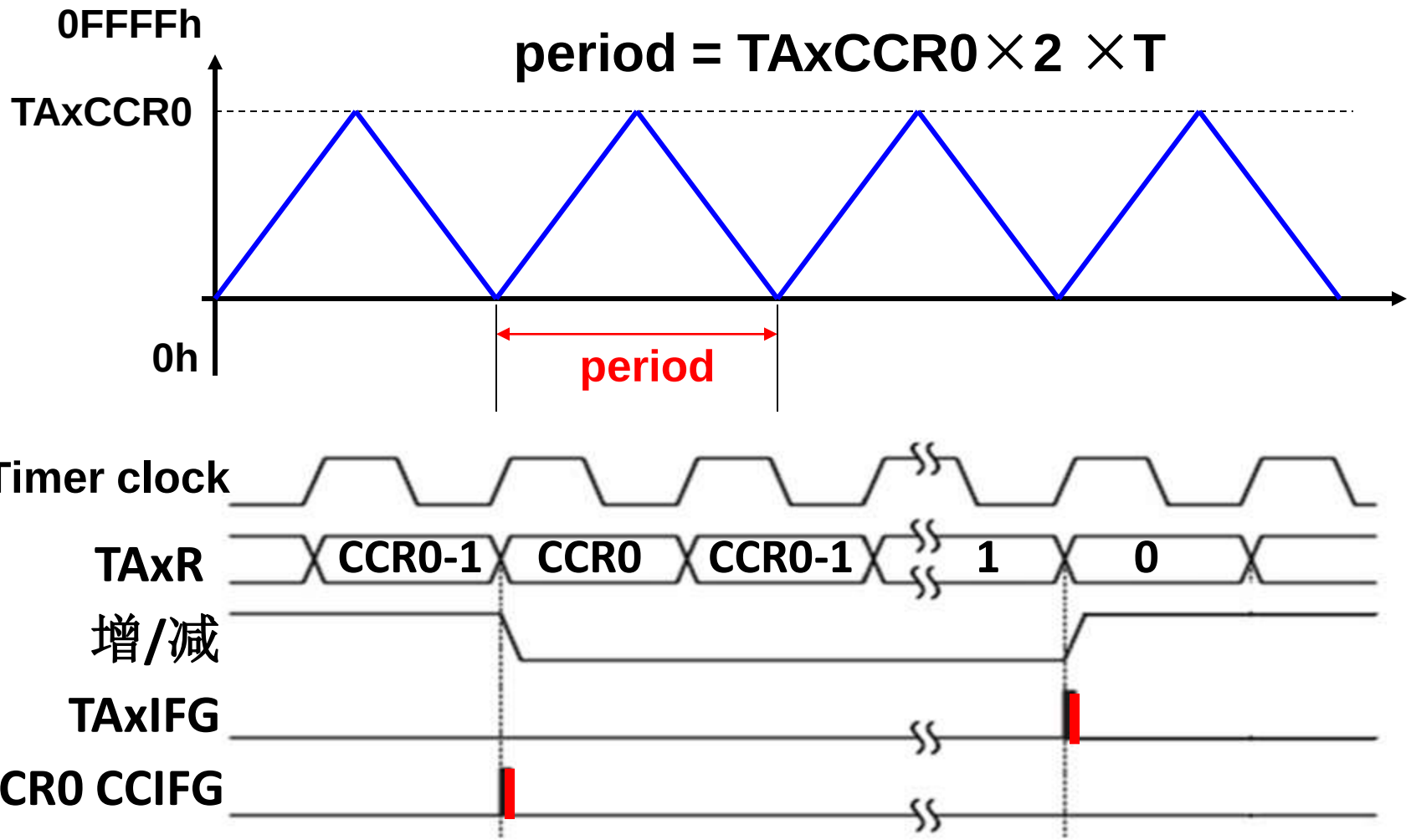
本方式不需要 **CCR0** 协助，使用方便，
但难以得到所希望的 **period**



当TAxR计数从值0FFFFh变为0时，置 TAxIFG =1

4) 增/减计数方式(三角波方式) MC= 11

Up/Down mode 需要 CCR0 协助



当TAR增计数到 TACCR0时，置TAxCCR0 CCIFG =1, 并开始减计数,
当TAR减计数从值1变为0时，置 TAxIFG =1，并开始增计数

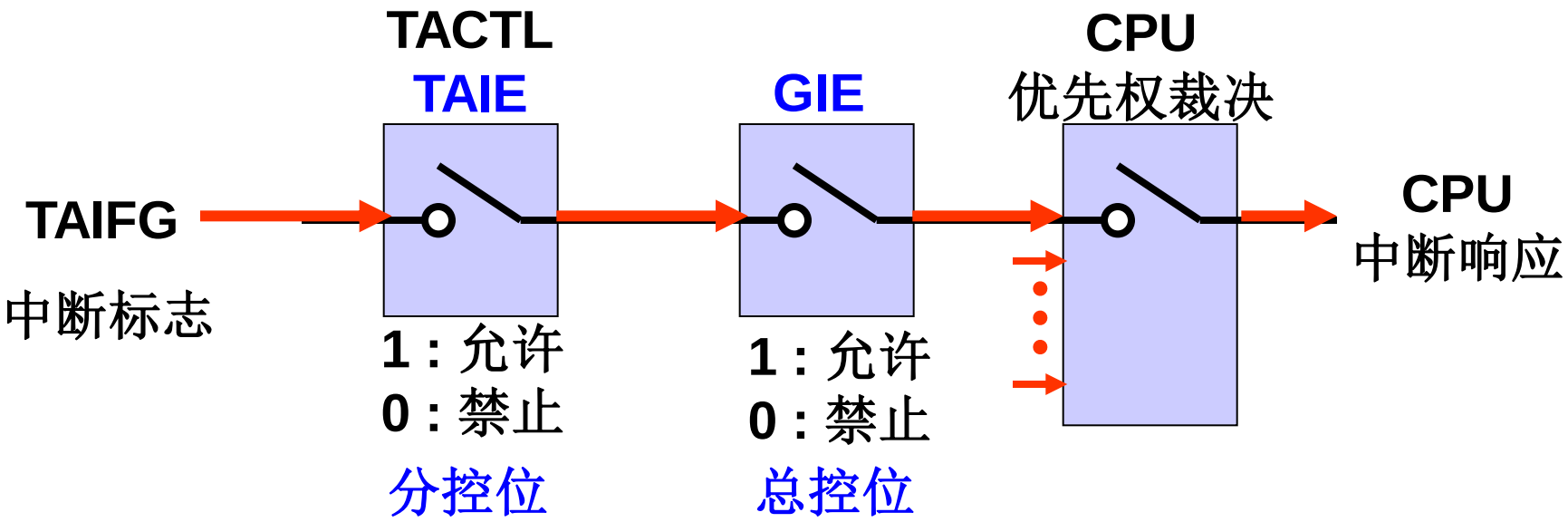
3. TA的计数溢出中断

- 如果溢出中断标志TAIFG被置位，

  若中断允许位TAIE=1，TA向CPU发出中断申请，

  如果CPU的 GIE=1，则CPU将响应该中断，

  转去执行相应类型的中断程序



三个条件：TAIFG=1， TAIE=1， GIE=1

- TAxIGF 产生的溢出中断与其他2个中断共享的一个中断类型

中断源	中断标志	向量地址	优先级,类型号
…	…	…	…
定时器A1	TA1CCR0 CCIFG	0FFE2h	49
定时器A1	TA1CCR1 CCIFG TA1CCR2 CCIFG TA1IFG	0FFE0h	48
…	…	…	…
定时器A2	TA2CCR0 CCIFG	0FFD0h	40
定时器A2	TA2CCR1 CCIFG TA2CCR2 CCIFG TA2IFG	0FFCEh	39

需利用中断向量寄存器TAxIV确定具体的中断源

三、TA的捕捉/比较器单元

Capture / Compare Blocks

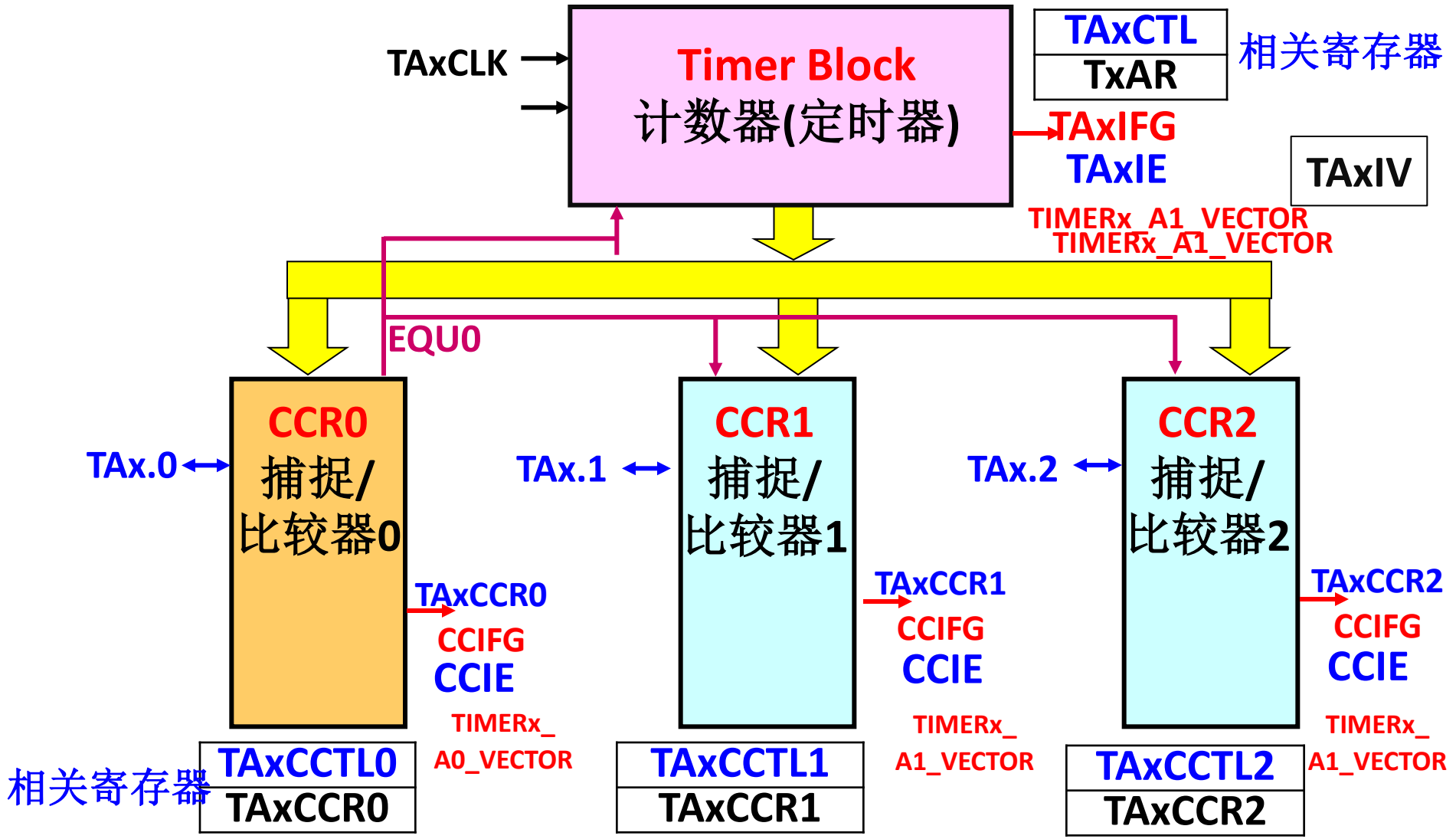
1. TA的捕捉/比较器单元作用和结构图
2. TA的比较匹配部分及其应用
3. TA的比较输出部分及其应用
4. TA的输入捕捉部分及其应用(不介绍不要求)

1. TA的捕捉/比较器单元作用和结构图

- **TA捕捉/比较器单元作用**

  - 产生时间间隔(比较匹配)
  - 输出PWM波形(比较输出)
  - 用于捕捉事件的发生和时刻(输入捕捉)

● TA的捕捉/比较器单元结构图



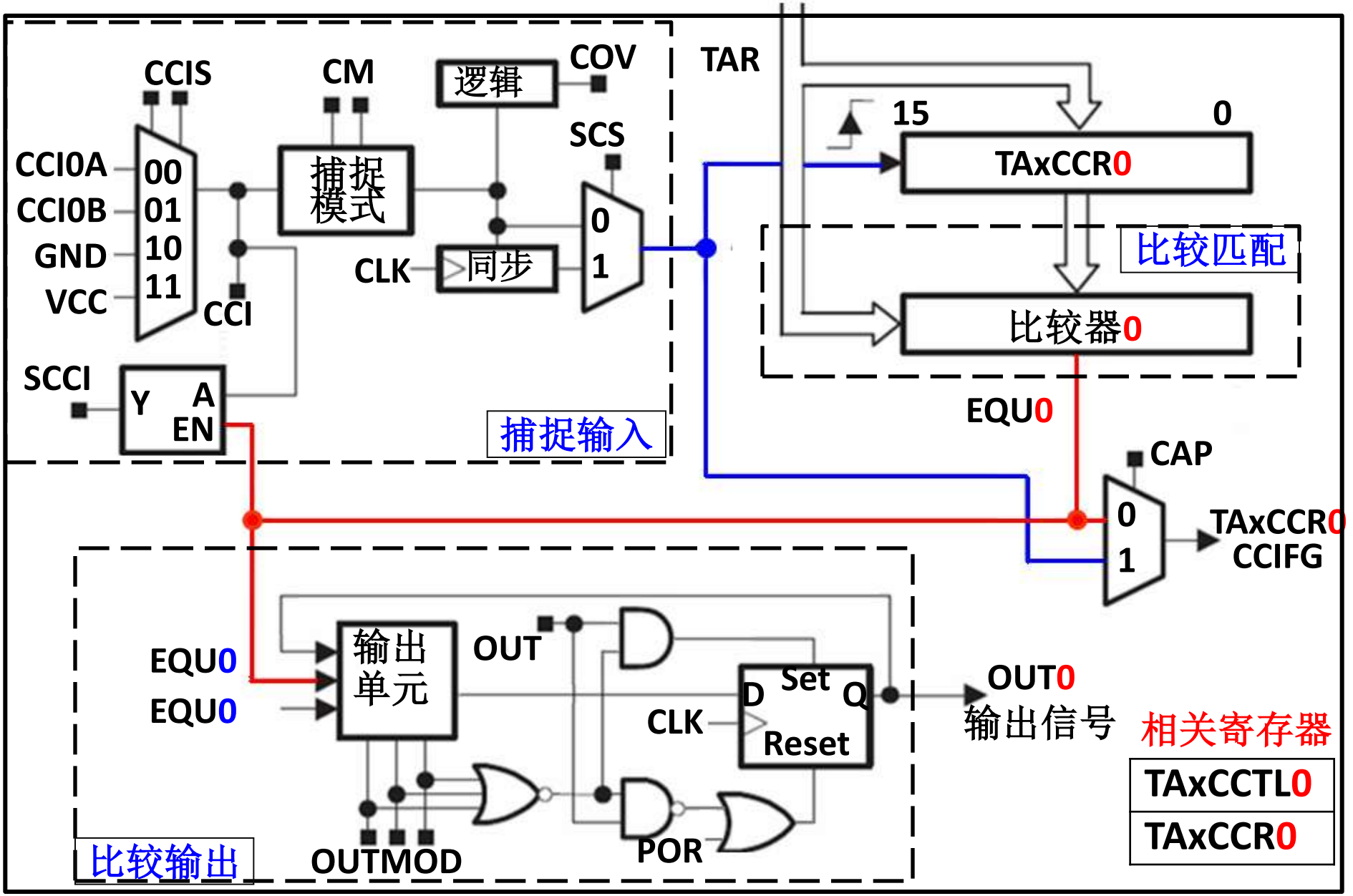
捕捉/比较寄存器TAxCCRy

在比较器方式中用于存放与TAxR进行比较的设定值，

当TAxR的值等于TAxCCRy时，产生比较匹配标志TACCRy CCIFG

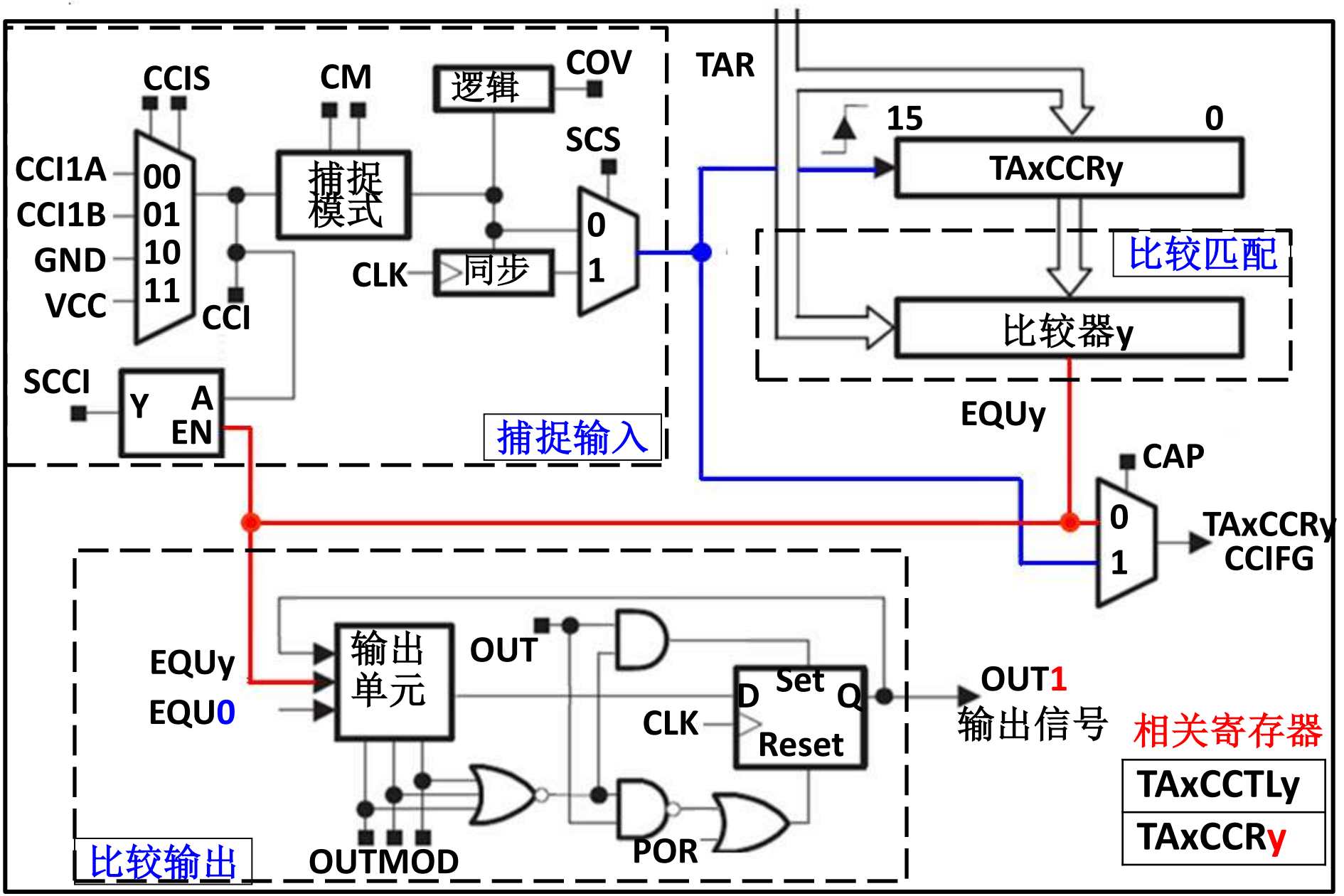
TAx捕获/比较器0结构图(CCR0)

TAx Capture/Compare 0 Block



TAx捕获/比较器y结构图(CCRy)

TAx Capture/Compare y Block



TAxCCTLy捕捉/比较器控制寄存器 (x=0,1,2， y=0,1,2)

capture/compare control register

决定CCRx的工作方式
0: 比较方式
1: 捕捉方式

与比较有关的设置

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)

7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

000~111
比较输出方式
8种

比较匹配中断允许
1: 允许
0: 禁止

当**OUTMODx=0**时
决定比较输出引脚上输出的电平值

比较匹配中断标志
1: 有中断
0: 无中断

msp430F6638.h中TAxCCTLx的各位及位值的符号定义

```
/* TAxCCTLx Control Bits */
#define CM1             (0x8000)     /* Capture mode 1 */
#define CM0             (0x4000)     /* Capture mode 0 */
#define CCIS1           (0x2000)     /* Capture input select 1 */
#define CCIS0           (0x1000)     /* Capture input select 0 */
#define SCS             (0x0800)     /* Capture sychronize */
#define SCCI            (0x0400)     /* Latched capture signal (read) */
#define CAP             (0x0100)     /* Capture mode: 1 /Compare mode : 0 */
#define OUTMOD2           (0x0080)     /* Output mode 2 */
#define OUTMOD1           (0x0040)     /* Output mode 1 */
#define OUTMOD0           (0x0020)     /* Output mode 0 */
#define CCIE            (0x0010)     /* Capture/compare interrupt enable */
#define CCI             (0x0008)     /* Capture input signal (read) */
#define OUT             (0x0004)     /* PWM Output signal if output mode 0 */
#define COV             (0x0002)     /* Capture/compare overflow flag */
#define CCIFG           (0x0001)     /* Capture/compare interrupt flag */

#define OUTMOD_0          (0*0x20u)    /* PWM output mode: 0 - output only */
#define OUTMOD_1          (1*0x20u)    /* PWM output mode: 1 - set */
#define OUTMOD_2          (2*0x20u)    /* PWM output mode: 2 - PWM toggle/reset */
#define OUTMOD_3          (3*0x20u)    /* PWM output mode: 3 - PWM set/reset */
#define OUTMOD_4          (4*0x20u)    /* PWM output mode: 4 - toggle */
#define OUTMOD_5          (5*0x20u)    /* PWM output mode: 5 - Reset */
#define OUTMOD_6          (6*0x20u)    /* PWM output mode: 6 - PWM toggle/set */
#define OUTMOD_7          (7*0x20u)    /* PWM output mode: 7 - PWM reset/set */
```

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Unused	CAP

7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG

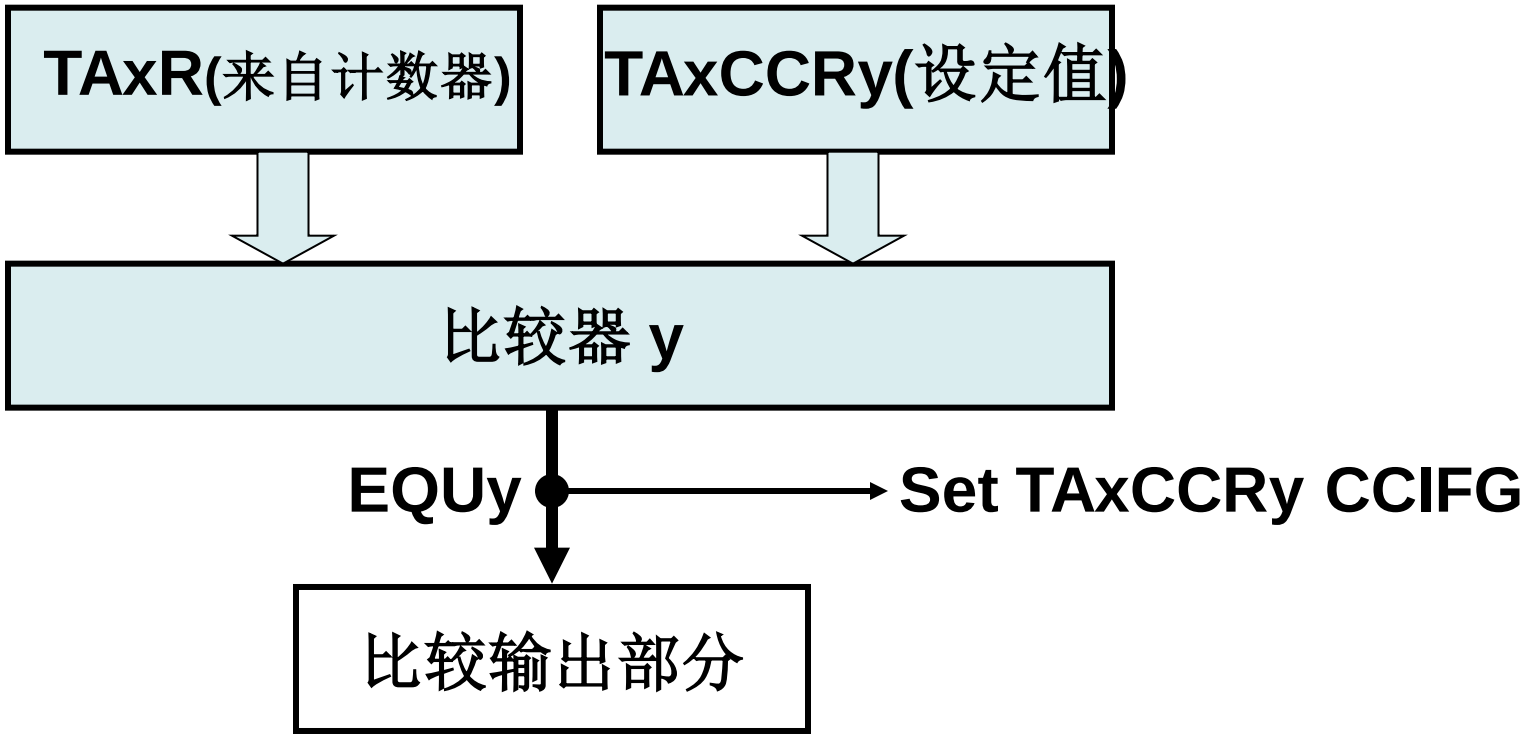
2. 比较匹配部分及其应用
(Compare Mode , CAP=0)

1) 比较匹配部分结构

2) 比较匹配中断

3) 比较匹配应用举例

1) TA 比较匹配电路 (CAP=0)

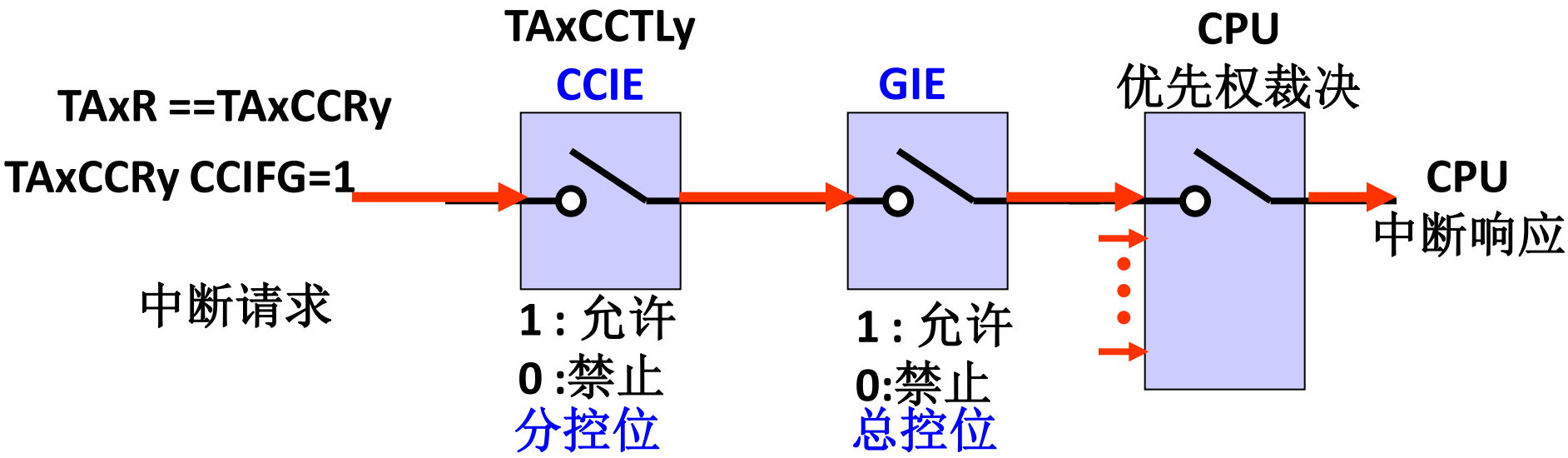


当 TAxR 计数到 TAxCCRy 设定的值时：

- 置内部信号 EQUy=1;
- 置比较匹配中断标志 TAxCCRy CCIFG=1

2) TA的比较匹配中断

- 当 TAxR 计数到 TAxCCRy 设定的值时，
  置比较匹配中断标志TAxCCRy CCIFG=1，
  如果分中断允许位TAxCCTLy CCIE =1，TAx向CPU发出中断申请，
  如果CPU的 GIE=1，则CPU将响应该中断，
  转去执行相应类型的中断程序



- TAxCCR0 CCIFG对应的中断类型为6，向量地址独占
- TAxCCR1 CCIFG、 TAxCCR2 CCIFG、TAxIFG三个标志共享一个类型中断向量，需通过TAxIV确定具体中断源

中断源	中断标志	向量地址	优先级,类型号
…	…	…	…
定时器A1	TA1CCR0 CCIFG	0FFE2h	49
定时器A1	TA1CCR1 CCIFG TA1CCR2 CCIFG TA1IFG	0FFE0h	48
…	…	…	…
定时器A2	TA2CCR0 CCIFG	0FFD0h	40
定时器A2	TA2CCR1 CCIFG TA2CCR2 CCIFG TA2IFG	0FFCEh	39

● TAxIV 中断向量寄存器 (Timer A Interrupt Vector Register)

用于确定TAx共享中断向量的中断源产生中断的具体原因;
如当TAxIV=000Eh时，表示是定时器溢出TAxIFG产生的中断

TAxIV内容	中断源	中断标志	优先级
0000h	无中断	-	
0002h	捕捉/比较1	TACCR1 CCIFG	最高
0004h	捕捉/比较2	TACCR2 CCIFG	
0006h	保留	-	
0008h	保留	-	
000Ah	保留	-	
000Ch	保留	-	
000Eh	定时器溢出	TAIFG	最低

共享中断源的中断标志的清除

3个中断标志(TAxCCR1 CCIFG和TAxCCR2 CCIFG 、TAxIFG)，

在读中断向量寄存器TAxIV后，相应中断标志将自动复位；

如果未读TAxIV寄存器，标志则不能自动清除，需用户软件清除。

msp430F6638.h 中断向量相关的符号定义

```
#define PORT4_VECTOR        (37 * 1u)          /* 0xFFCA Port 4 */
#define PORT3_VECTOR        (38 * 1u)          /* 0xFFCC Port 3 */
#define TIMER2_A1_VECTOR (39 * 1u)         /* 0xFFCE Timer0_A5 CC1-4, TA */
#define TIMER2_A0_VECTOR   (40 * 1u)              /* 0xFFD0 Timer0_A5 CC0 */
……
#define PORT2_VECTOR        (44 * 1u)              /* 0xFFD8 Port 2 */
……
#define PORT1_VECTOR        (47 * 1u)              /* 0xFFDE Port 1 */
#define TIMER1_A1_VECTOR (48 * 1u)              /* 0xFFE0 Timer1_A3 CC1-2, TA1 */
#define TIMER1_A0_VECTOR (49 * 1u)              /* 0xFFE2 Timer1_A3 CC0 */
……
```

TACCR0 中断子程结构

```
……
int main( void )
{     WDTCTL = WDTPW + WDTHOLD;
      ……
      TAxCCTL0|=CCIE;          //置TACCR0 CCIFG中断允许
      _EINT();
       ……
      while(1){};
}

#pragma  vector=TIMER1_A0_VECTOR        //TimerA1 CCR0中断
__interrupt void  Timer1_CCR0_isr()
{
      …..;
}
```

●TAIV 中断子程结构框架

```
……
int main( void )
{     WDTCTL = WDTPW + WDTHOLD;
      ……
//    TA1CCTL|=TAIE;              //置TA1  TAIFG中断允许
      TA1CCTL1|=CCIE;             //置TA1CCR1 CCIFG中断允许
//    TA1CCTL2|=CCIE;             //置TA1CCR2 CCIFG中断允许
      _EINT();
      ……
      while(1){  };
}

#pragma  vector=TIMER1_A1_VECTOR
__interrupt void  Timer1_A1_isr( )
{   unsigned char Taiv_temp;
      Taiv_temp=TAIV;                          //取TAIV值
      if ( Taiv_temp==2) then
      { …… }                                   //TAxCCR1  CCIFG=1的处理
      else if( Taiv_temp==4) then
      { …… }                                   //TAxCCR2  CCIFG=1的处理
      else if( Taiv_temp==0xE) then
      { …… };                                  //TAx  TAxIFG=1处理
}
```

3) 比较匹配应用举例

利用比较匹配中断，设计间隔的时间触发系统

例1 要求每隔 1s 进行一次系统自检，

假设自检功能由self_check子程完成

为便于实验观察，系统自检功能用“对P3.0求反一次”代表，

P3.0上可以连接一个发光二极管，或连到一个蜂鸣器上。

如果选用ACLK=32.768KHz作为计数时钟，采用增计数方式，利用TA1CCR0 CCIFG控制时间间隔，问:

1) TA1CCR0应设置为多少?
2) TA1CTL应如何设置？
3) TA1CCTL0应如何设置？
4) 给出程序流程和程序清单？

增计数方式下：TAxCCR0 CCIFG中断周期=(TAxCCR0+1) * T

TAxCCR0的最大值=65535，即0FFFFh

TAxCCR0 CCIFG中断产生的最大时间间隔=

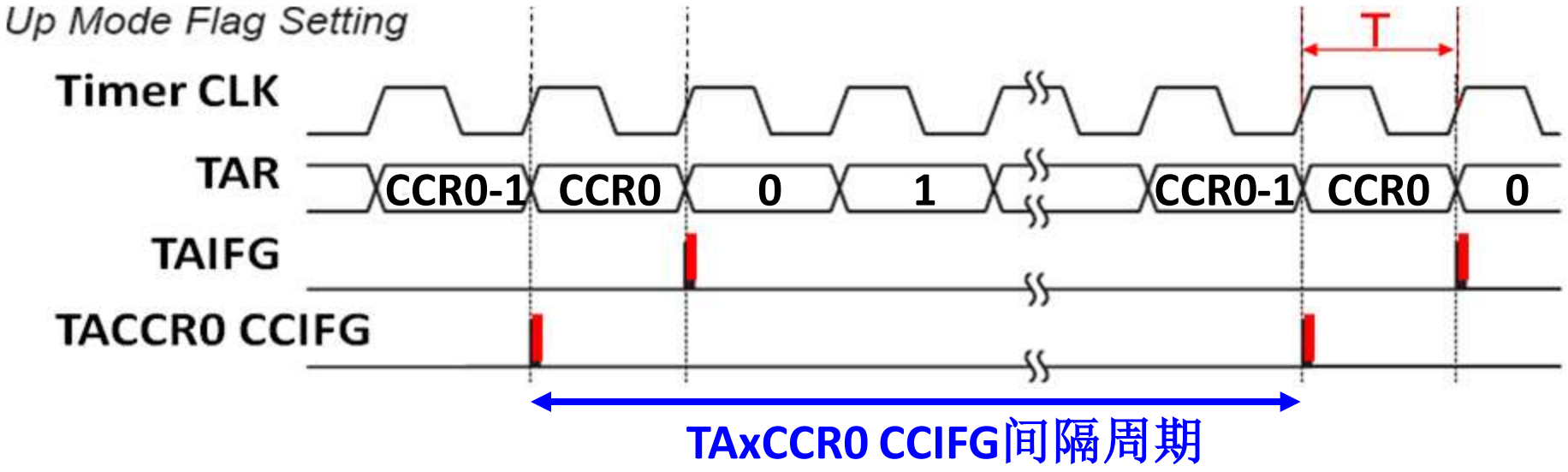
(65535+1)*(计数时钟周期)=65536*(1/32768) = 2S

1) 计算TA1CCR0的设定值

事件的时间间隔周期为1S < CCIFG中断产生的最大时间间隔

需要的时钟个数=事件周期/时钟周期

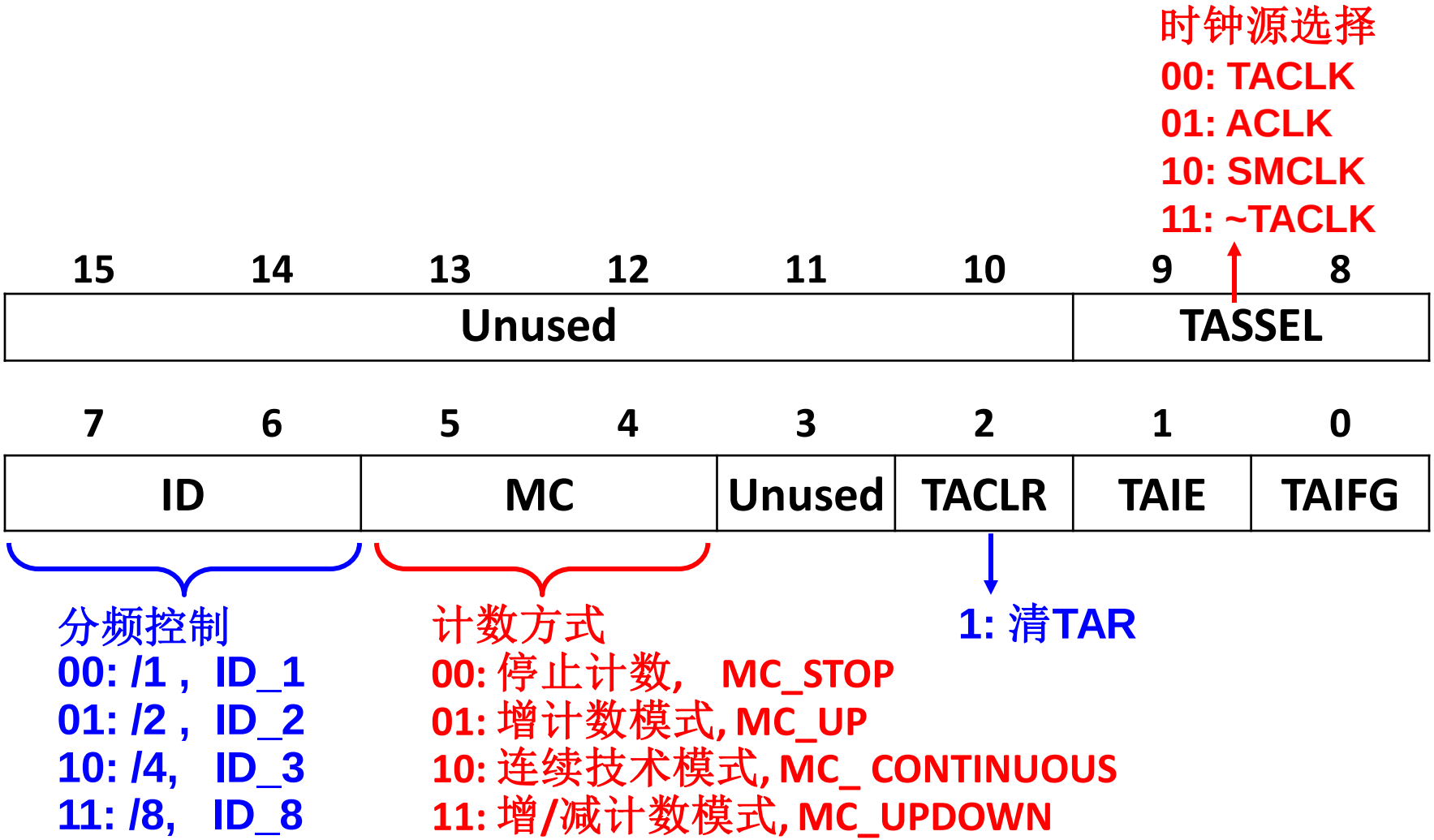
所以：TA1CCR0=需要的时钟个数 – 1 = 1s/(1/32768) – 1 = 32767



2) 设置定时器 A 控制寄存器 TAxCTL

//清TA1R=0，选ACLK计数，不分频，增计数方式，禁止溢出中断

TA1CTL = TACLR+TASSEL__ACLK + MC__UP + ID__1;



3)设置捕捉/比较器控制寄存器 0(TA1CCTL0)

TACCTL0的复位值为： CAP=0, 比较方式；
所以，只用打开CCIE中断允许即可

```
TA1CCTL0|= CCIE ;
```

设置与比较匹配有关的位

决定CCRx的工作方式
0: 比较方式
1: 捕捉方式

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)
7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

比较输出方式
8种

比较匹配
中断允许
1: 允许
0: 禁止

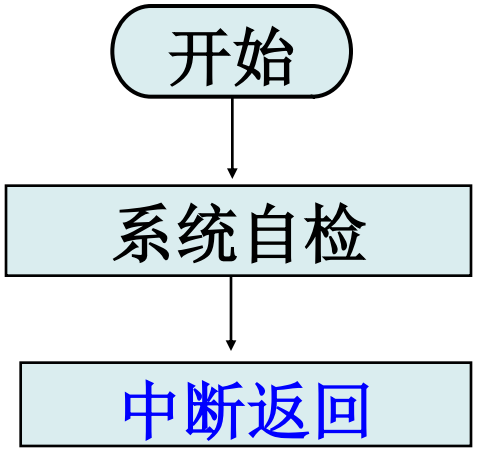
当OUTMOD=0时
决定比较输出引脚
上输出的电平值

比较匹配
中断标志
1: 有中断
0: 无中断

4) 程序流程

1. 主程序

2. 中断子程



3. 设置中断向量

将中断子程设置在向量地址处

chap7_TACCR0_1s.c程序清单

```
#include “msp430F6638.h"
void self_check( );

int    main ( void )
{    WDTCTL = WDTPW + WDTHOLD;      //关闭看门狗

     P3SEL &= ~BIT0;                //置P3.0基本I/O功能
     P3OUT &= ~BIT0;                //置输出的初值
     P3DIR |= BIT0;                 //置P3.0 输出方向

     //清TAR=0, 选ACLK计数, 不分频, 增计数方式, 禁止溢出中断
     TA1CTL = TACLR+TASSEL__ACLK + MC__UP + ID__1;//
     TA1CCR0=32767;                 //置TACCR0值
     TA1CCTL0|= CCIE ;              //允许CCR0中断

     _EINT();                       //开CPU总中断允许

     while(1){  };
}
```

chap7_TACCR0_1s.c程序清单(续)

```
#pragma  vector=TIMER1_A0_VECTOR
__interrupt  void  timer1_A0_ISR( )
{
      self_check();
}

void self_check( )
{
      P3OUT ^= BIT0;                    //P3.0求反
 }
```

例2. 事件的 时间间隔周期大于最大的CCIFG中断周期

如何编程使系统每3秒自检查一次?

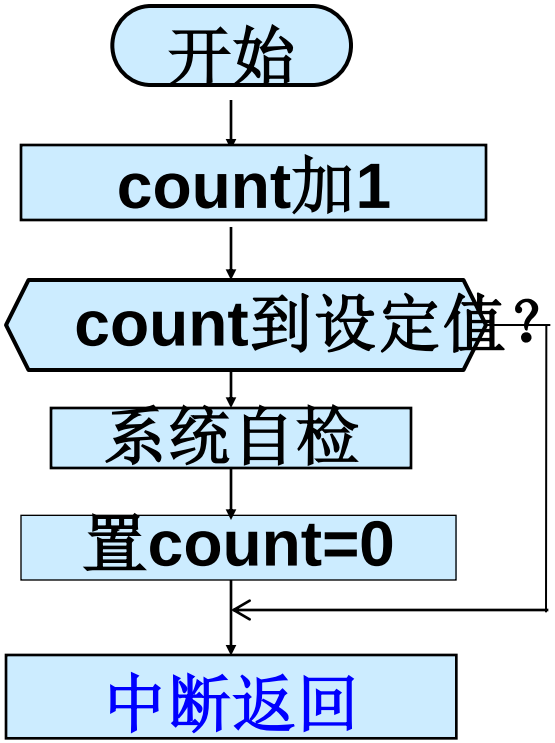
- 要求的事件间隔3秒> CCIFG中断产生的最大时间间隔2秒,
  如何编程?

- 1秒进1次中断，进3次是3秒，
  可以在例1程序基础上，用变量计数CCIFG响应中断的次数，
  从0开始计数，到了3时自检；否则只计数

程序流程

1. 主程序

2. 中断子程



3. 设置中断向量

将中断子程设置在向量地址处

chap7_TACCR0_3s.c程序清单

```
#include “msp430F6638.h"
void self_check( );
unsigned int count;
int    main ( void )
{    WDTCTL = WDTPW + WDTHOLD;      //关闭看门狗

     P3SEL &= ~BIT0;                //置P3.0基本I/O功能
     P3OUT &= ~BIT0;                //置输出的初值
     P3DIR |= BIT0;                 //置P3.0 输出方向

     //清TAR=0, 选ACLK计数, 不分频, 增计数方式,禁止溢出中断
     TA1CTL = TACLR+TASSEL__ACLK + MC__UP + ID__1;//
     TA1CCR0=32767;                 //置TACCR0值
     TA1CCTL0|= CCIE ;              //允许CCR0中断

     _EINT();                       //开CPU总中断允许

     while(1){  };
}
```

chap7_TACCR0_3s.c程序清单(续)

```
#pragma  vector=TIMER1_A0_VECTOR
__interrupt void timer1_A0_ISR( )
{
      count++;
      if (count==3)
        {   self_check( );
              count=0;
        }
}

void self_check( )
{       P3OUT^=BIT0;     //P3.0取反
}
```

思考：

1. 如何将主程序改为 低功耗模式？
2. 可否选择LMP4低功耗模式?
3. 如果计数时钟源选SMCLK=8MHz, 如何编程?
4. 例2的中断程序中，如果没有count=0的语句，结果如何？
5. 如果改成TA1CCR1 CCIFG中断来控制时间间隔,
   在例1 Chap7_TACCR0_1s.c程序基础上如何修改?

例3 如果改成TA1CCR1 CCIFG中断来控制时间间隔，

在例1 Chap7_TACCR0_1s.c程序基础上如何修改?

1) 中断类型号
2) 中断程序的编写，需访问TAxIV
3) 设置TAxCTL1中的CCIE中断允许
4) 增计数模式需要CCR0的协助，故需设置TAxCCR0=周期设定值，否则由于TAxCCR0=0，TAxR将不计数
5) TAxCCR1可以是0~TAxCCR0中的任意数。

在TAxR计数到TAxCCR0的过程中，

当TAxR=TAxCCR1时，EQU1=1, 置TAxCCR1 CCIFG=1，

发出中断申请

chap7_TACCR1_1s.c程序清单

```
#include “msp430F6638.h"
void self_check( );

int    main ( void )
{    WDTCTL = WDTPW + WDTHOLD;      //关闭看门狗

     P3SEL &= ~BIT0;                //置P3.0基本I/O功能
     P3OUT &= ~BIT0;                //置输出的初值
     P3DIR |= BIT0;                 //置P3.0 输出方向

     //清TAR=0, 选ACLK计数, 不分频, 增计数方式, 禁止溢出中断
     TA1CTL = TACLR+TASSEL__ACLK + MC__UP + ID__1;//
     TA1CCR0=32767;                 //置TACCR0值
     TA1CCTL1|= CCIE ;              //允许CCR1中断

     _EINT();                       //开CPU总中断允许

     while(1){  };
}
```

chap7_TACCR1_1s.s43程序清单(续)

```
#pragma  vector=TIMER1_A1_VECTOR          //TimerA CCR1中断
__interrupt  void  Timer1_A1_ISR( )
{   unsigned char Taiv;
          Taiv=TA1IV;                          //读TAIV值
          if ( Taiv==2)
          {                                 //TACCR1 CCIFG处理
          self_check( );
          };
}

void self_check( )
{
          P3OUT ^= BIT0;                       //P3.0求反
}
```

3. TA的比较输出部分及其应用(自学)

参看MSP430x1xx Family User's Guide .pdf 中11.2.5 Output Unit

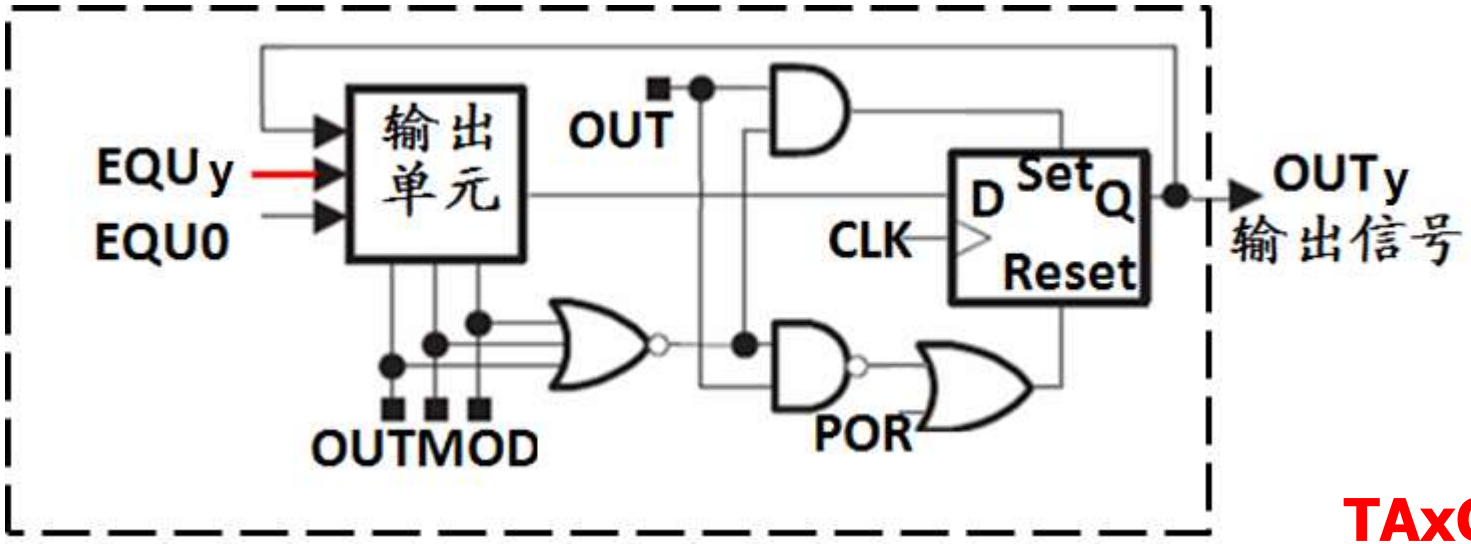
1) TA的比较输出部分电路

2) TA的8种输出方式

3) TA的比较输出应用举例

1) TA的比较输出部分电路

利用比较匹配的内部输出EQUy,
以及TAxCCLy控制寄存器中输出方式的控制，
在相应的MCU引脚上输出波形.



TAxCCTLy寄存器

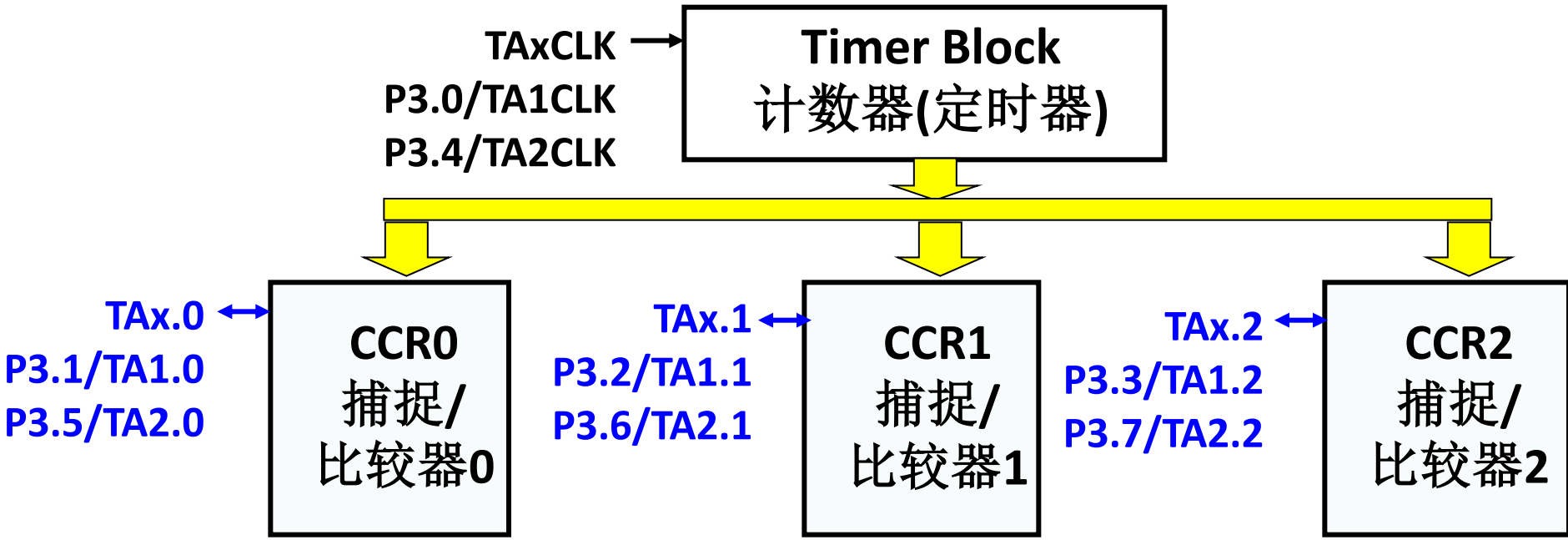
7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

8种比较输出方式

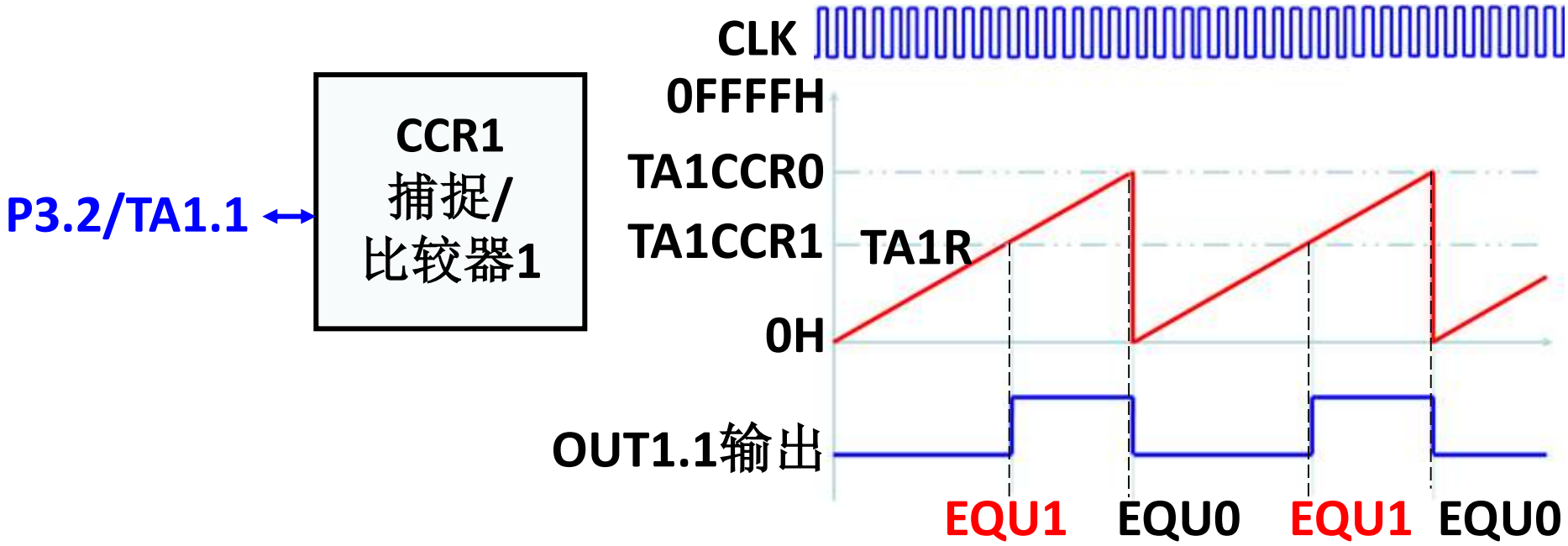
当OUTMOD=0时
决定比较输出引脚上输出的电平值

TA的相关 输出引脚

- 将模块对应的端口引脚功能选择寄存器SEL选择为模块功能，并将方向寄存器DIR相应位设置为输出，引脚即作为TA输出引脚



例　引脚P3.1上可以输出OUT1.1的波形

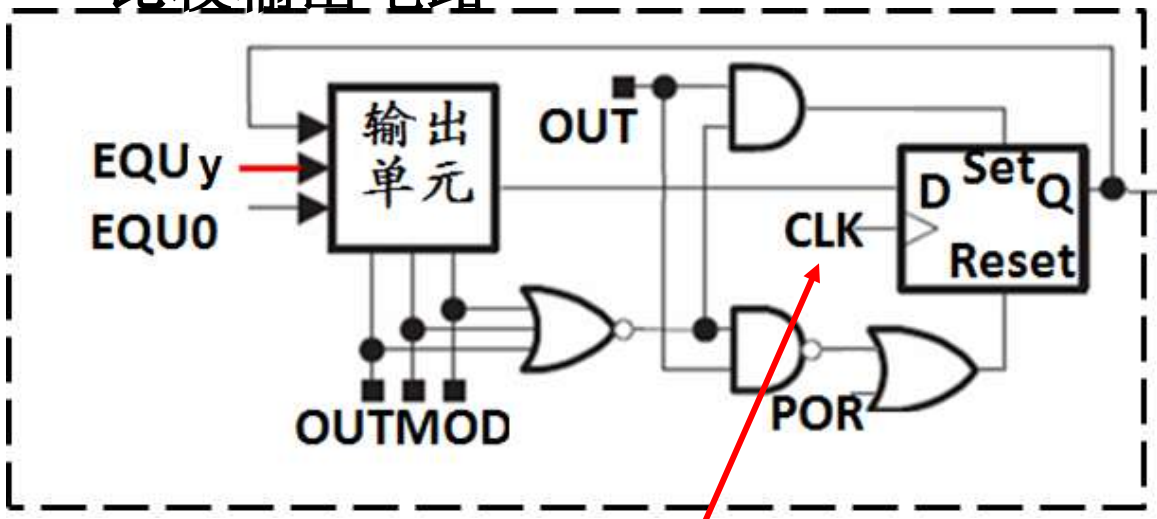


下面设置使P3.2作为TA1的输出引脚，

OUT1.1的波形将通过P3.2输出

```
P3SEL|=BIT2;          //置P3.2模块功能
P3DIR|=BIT2;          //置P3.2输出方向
```

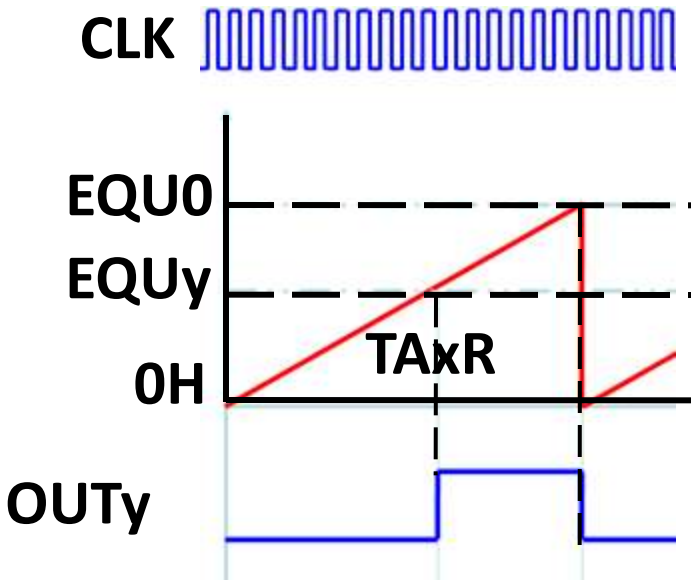
比较输出电路



2) OUT1和OUT2有8种输出方式

EQU1和EQU0控制OUT1
EQU2和EQU0控制OUT2

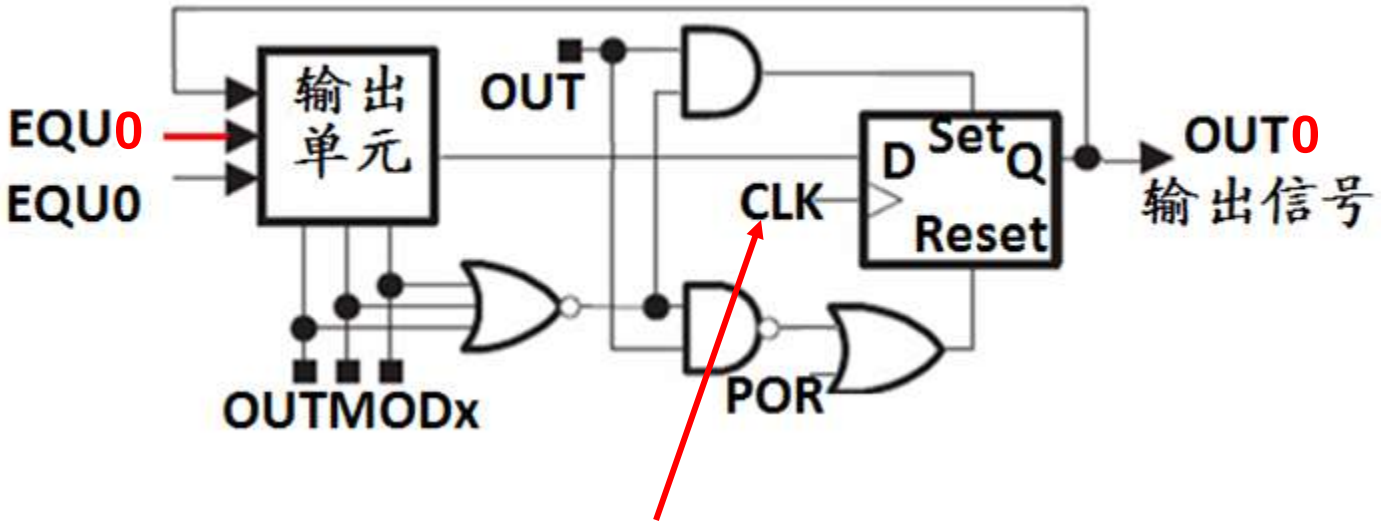
OUTy在Timer CLK的上升沿处改变



OUTMOD	EQUy时输出	EQU0时输出
000	=OUT位	不影响
001	置位	不影响
010	翻转	复位
011	置位	复位
100	翻转	不影响
101	复位	不影响
110	翻转	置位
111	复位	置位

OUT0只有4种输出方式

比较输出电路



OUTx在Timer CLK的上升沿处改变

OUTMOD	EQU0时输出
000	=OUT位
001	置位
100	翻转
101	复位

OUT0，只受EQU0控制

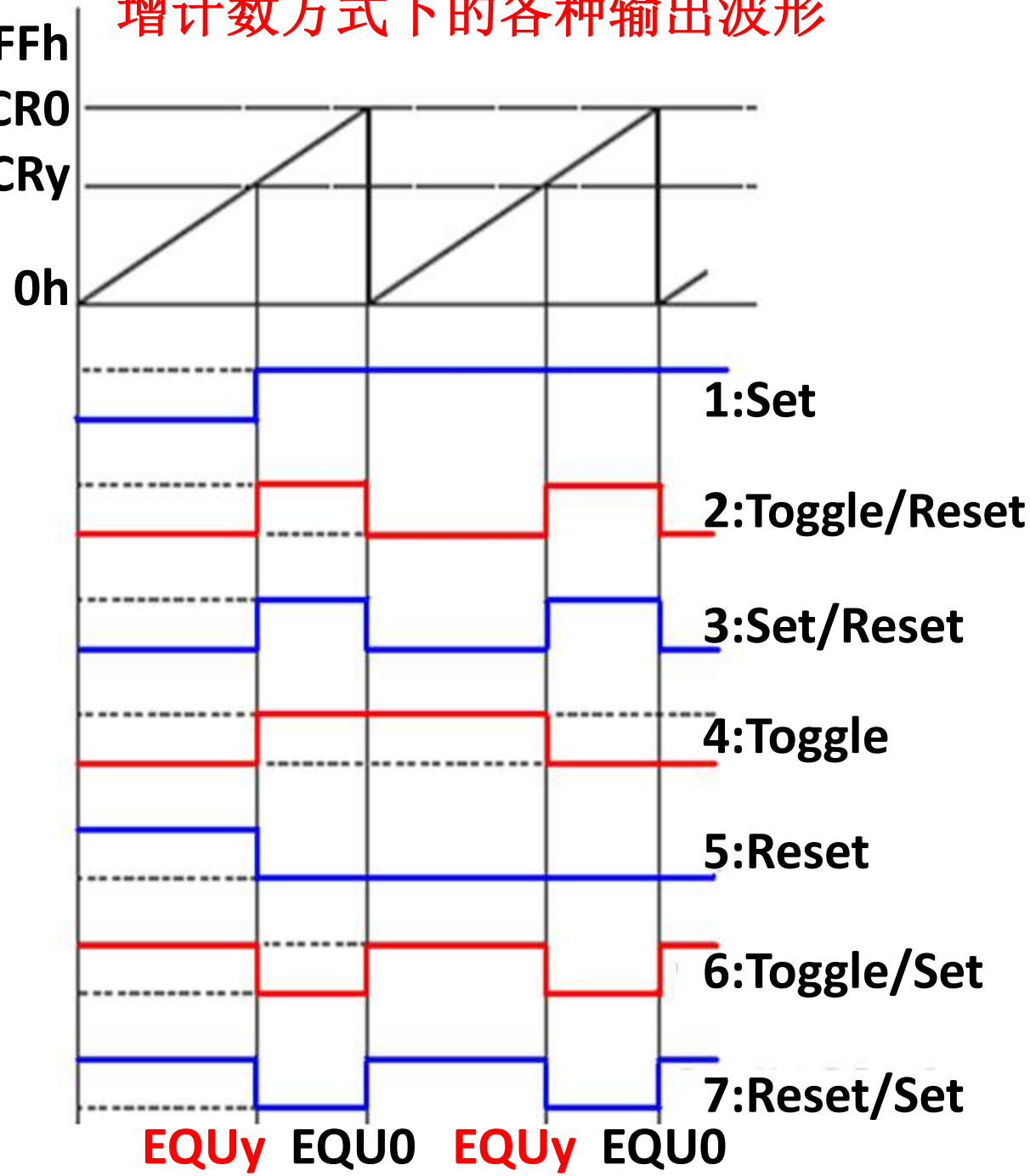
只有0/1/4/5 输出方式

没有2/3/6/7 输出方式

增计数方式下的各种输出波形

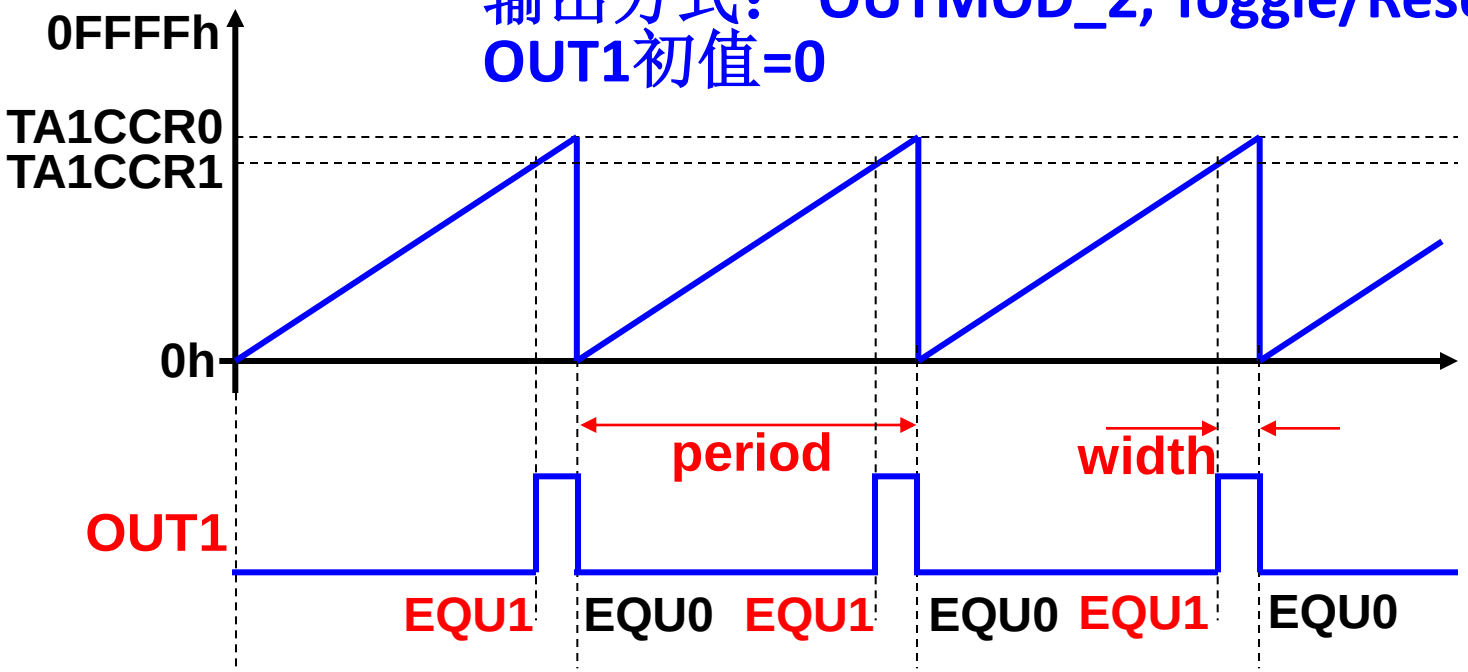
OUT MOD	EQUy时输出	EQU0时输出
000	=OUT位	不影响
001	置位	不影响
010	翻转	复位
011	置位	复位
100	翻转	不影响
101	复位	不影响
110	翻转	置位
111	复位	置位

利用模式0，
可通过OUT位设置
波形的初始值



输出举例

定时器：增计数方式，MC_UP
使用 TA1CCR1和 TA1CCR0
输出方式： OUTMOD_2, Toggle/Reset
OUT1初值=0

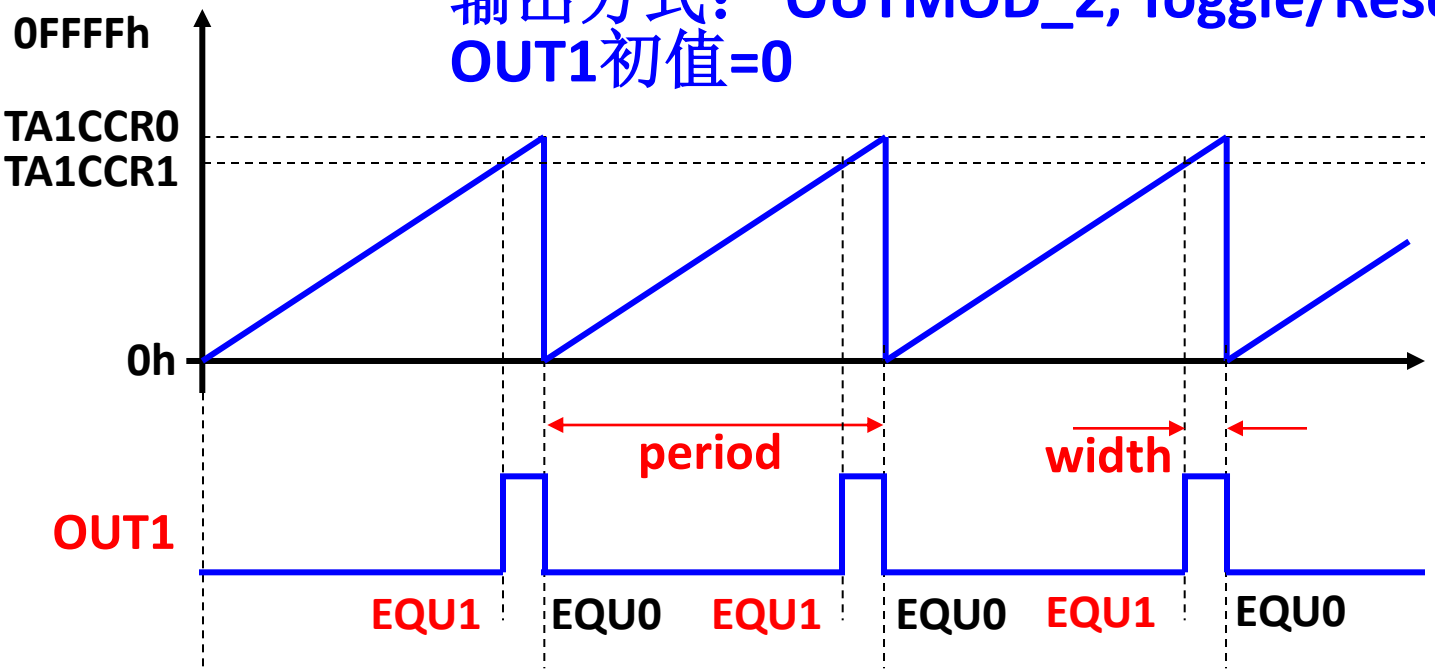


- 改变TA1CCR1和TA1CCR0可改变脉冲的周期和占空比：TA1CCR0决定脉冲周期；(TA1CCR0-TA1CCR1)决定脉冲宽度；实现 PWM 输出

*例如：*使TA1CCR1=TA1CCR0/2，则可得占空比=50%的输出信号

输出举例(续)

定时器：增计数方式， MC_UP
使用 TA1CCR1和 TA1CCR0
输出方式： OUTMOD_2, Toggle/Reset
OUT1初值=0



当 TA1CCR0 和 TA1CCR1 都固定时，可以得到固定周期 period和正脉冲宽度 width的脉冲信号输出；

周期period = (TA1CCR0 + 1)×T

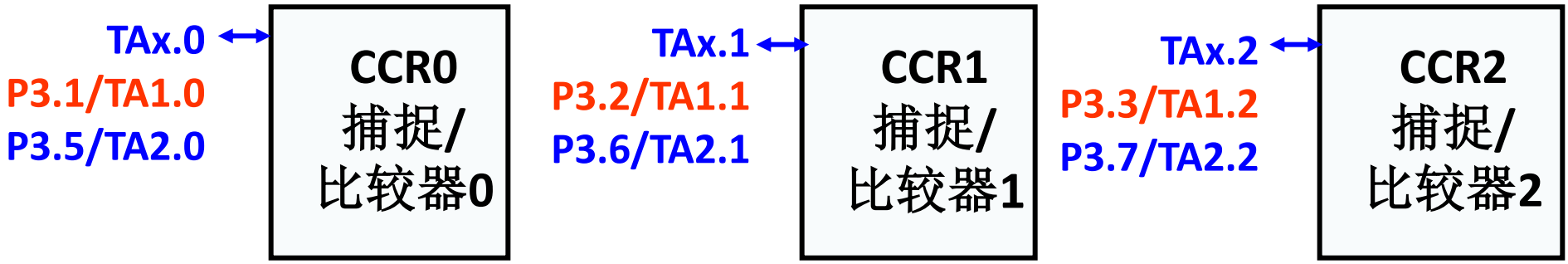
正脉冲宽度width = (TA1CCR0 – TA1CCR1)×T

T = Timer Clock 的周期

3) TA的比较输出应用举例

PWM脉冲波形输出

编程选用ACLK=32.768KHz作为计数时钟，采用增计数方式，利用TA1的输出引脚输出占空比分别为25%和50%的PWM脉冲信号



注意：

PWM输出，与中断无关，

设置后，CPU不介入，由TA模块独立完成

程序流程


开始
相关输出引脚初始化设置
据周期设置TA1CCR0值
据占空比设置TA1CCR1值
和TA1CCR1中的输出模式
据占空比设置TA1CCR2值
和TA1CCR2中的输出模式
设置TA1CTL
清TA1R, 选计数时钟,计数方式
主程其它工作处理
(可用无限循环延时代替,
或进入低功耗模式)

Chap7_PWM.c程序清单

```
#include “msp430F6638.h"
int main ( void )
{   WDTCTL = WDTPW + WDTHOLD;          //关闭看门狗
    P3SEL |=BIT2+BIT3;                 //置P2.3,P2.4模块功能
    P3DIR |=BIT2+BIT3;          //置P2.3,P2.4输出方向
   TA1CCR0=32767;                      //PERIOD= (TA1CCR0+1)*T
   TA1CCR1=24575;                      //25%占空比32768*(1-0.25
   TA1CCTL1=OUTMOD_2;                  //OUT1.1模式2输出 toggle/reset
   TA1CCR2=16383;                      //50%占空比32768*(1-0.5)
   TA1CCTL2=OUTMOD_2;                  //OUT1.2模式2输出 toggle/reset

//清TAR=0，选ACLK计数，不分频，增计数方式，禁止溢出中断
    TA1CTL = TACLR+TASSEL__ACLK + MC__UP + ID__1;//

    LPM0;                              //进入低功耗模式0
    while(1){  };
}
```