Ch.8增强型捕获（eCAP）模块

8.1 eCAP的捕获模式

8.2 eCAP重要寄存器

8.3 eCAP的应用举例

eCap的学习思路

思路：

1. 什么是捕获？捕获与外部中断的差别？

2.捕获单元的构成和使用要点？

3.捕获有哪几种方式？各有何应用背景？

4. 捕获单元的中断使用方法？

5. 重要的寄存器？

Cap的基本概念

捕获（CAP）模块：

对外部事件进行捕捉，并记录此事件发生的时刻。

相比于外部中断：都是对外部事件处理

1）CAP记录功能更强，32位计时器
2）更加灵活，
更多样的事件的定义
多级缓冲
丰富的中断设置

增强型捕获（eCAP）模块应用在外部事件精确定时等比较重要的系统中。

主要应用场合：

1. 旋转机械（电机）的速度检测；
2. 电机位置传感器之间的时间检测；
3. 脉冲序列信号的周期和占空比检测；
4. 根据电压/电流传感器编码的占空比周期计算电压/电流实际幅值大小。（F/V）

eCap的特性

- 4个事件的时间戳寄存器（每个32位）；
- 多达4个顺序时间戳捕获事件的边沿极性选择；
- 任一事件（共4个事件）均可产生中断；
- 多达4个事件时间戳的单触发捕获；
- 时间戳的连续模式捕获，循环缓冲区的深度为4；
- 绝对时间戳捕获；
- 差分（Delta）模式时间戳捕获；
- 所有上述资源特定用于一个输入管脚；
- ~~当不用于捕获模式时，ECAP模块可配置为单通道PWM输出。~~

一个eCAP通道具有下列独立的重要资源

1. 专用的捕捉输入引脚；
2. 32位时基（计数器）
3. 4个32位时间戳捕捉寄存器（CAP1~CAP4），用于存储捕捉的计数值
4. mod4序列发生器（mod4位计数器），能够根据ECAP引脚的上升或者下降沿实现与外部事件的同步；
5. 可以为4个事件独立选择边沿极性（上升/下降沿）；

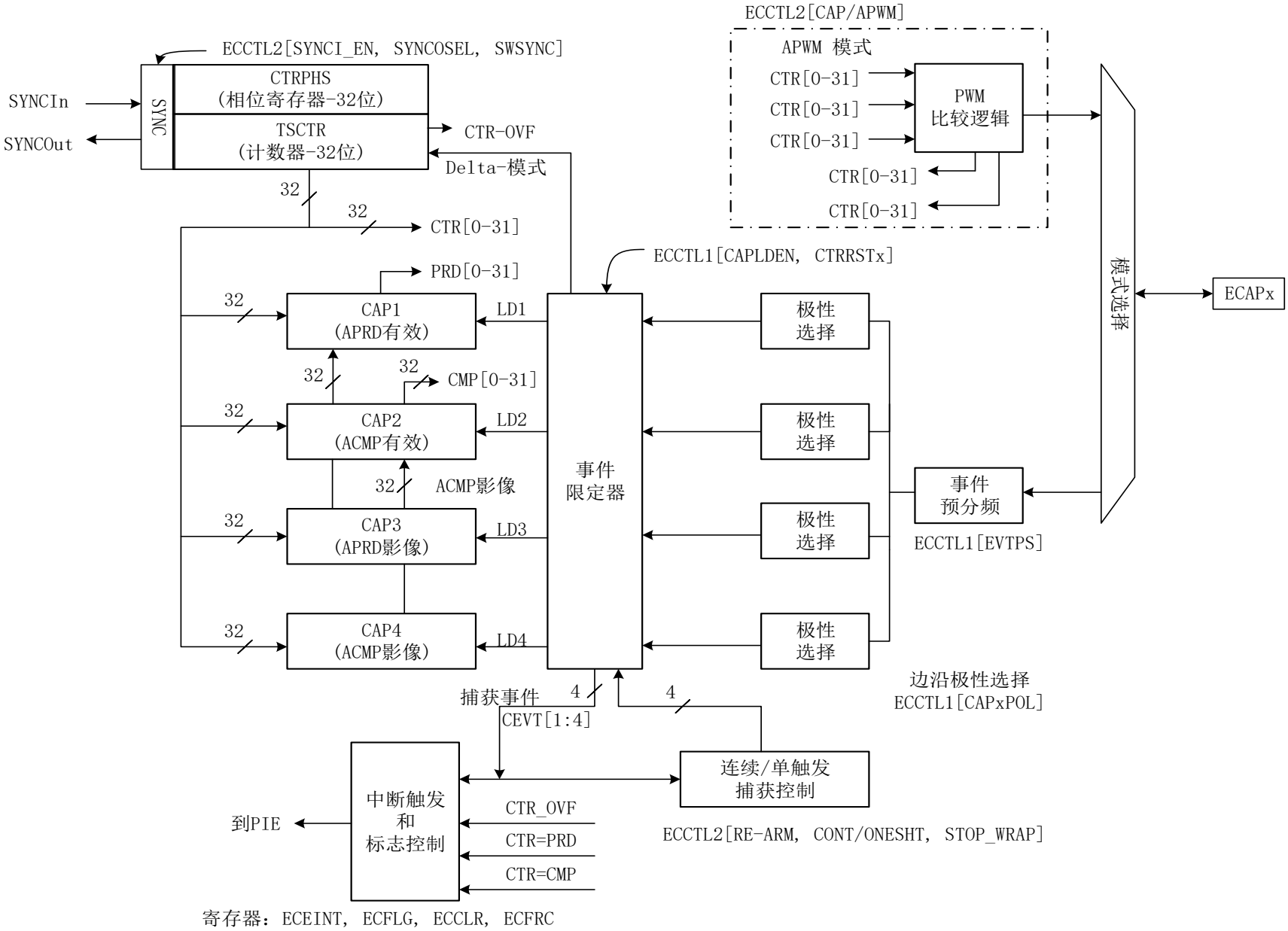
6. 支持输入捕捉信号预分频（2-62）

7. 单触发比较寄存器（2位），用于在1~4个时间标签事件发生后停止捕捉；

8. 采用一个4级深度的循环缓冲器（ CAP1~CAP4 ）控制连续时间标签事件的捕捉操作；

9. 4个捕捉事件每个都可以产生中断。

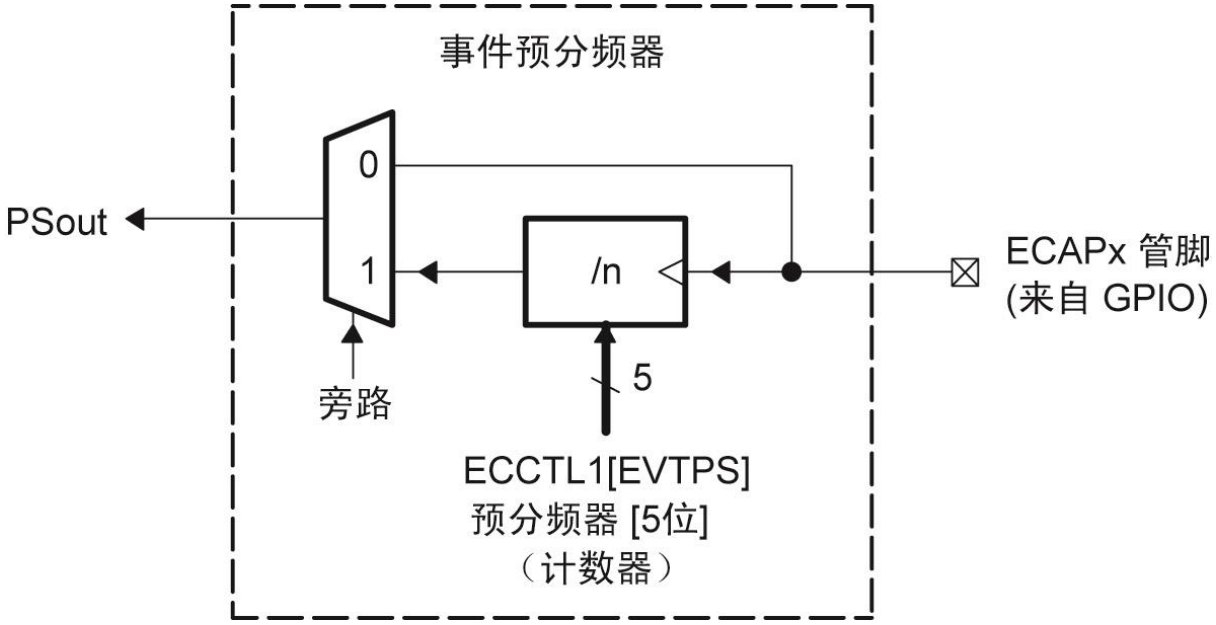
•8.1 eCAP的捕获模式



8.1.1 事件捕获

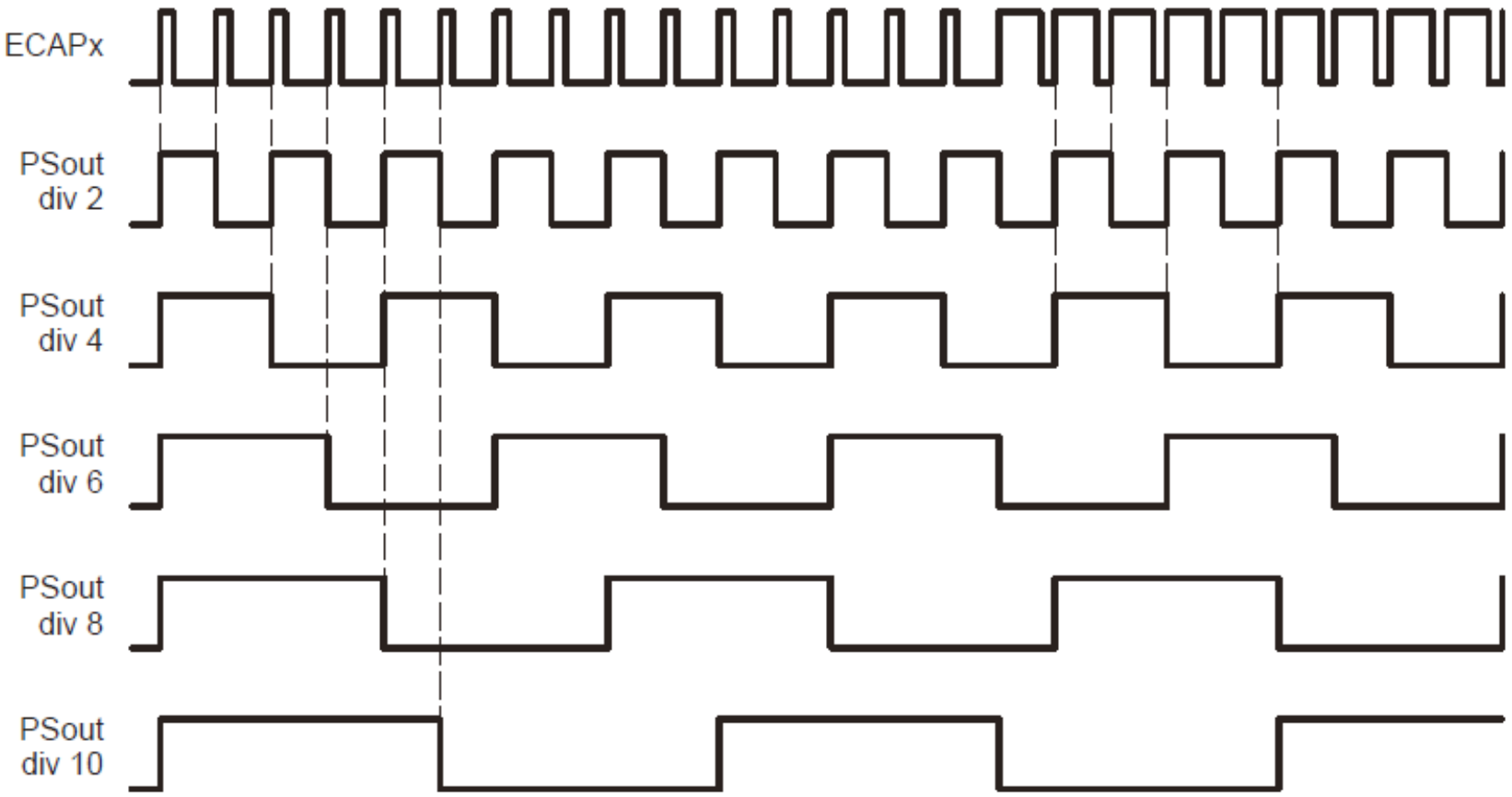
预分频器（Event-Prescaler）

输入捕获信号（脉冲群）可通过N=2-62（2的倍数）预分频或可旁路预分频器。当高频信号用作输入时该操作是十分有用的。



A 当选择预分频值为1（即，ECCTL1[13:9]=0,0,0,0,0）时，输入捕获信号完全旁路预分频逻辑。

预分频器（Event-Prescaler）



边缘极性选择与限定器

1. 事件捕捉引脚捕捉的脉冲信号经过预分频以后，采用4个独立的边缘极性（上升/下降沿）选择MUX，每个对应一个捕捉事件；

2. 采用mod4序列发生器对每个边沿（最多4个）进行量化；

3. 通过mod4计数器将边沿事件锁存到相应的CAPx寄存器中，CAPx寄存器在下降沿被装载

eCap连续/单触发控制

1. 根据边沿量化事件（CEVT1~CEVT4），mod4计数器（2位）递增计数；

2. 计数器mod4连续循环计数（0,1,2,3,0）和循环，直至设置计数停止；

3. 用一个2位停止寄存器和mod4计数器比较输出，当停止寄存器的值等于mod4计数器的值时停止mod4计数器计数，不再装载CAP1~CAP4寄存器，该操作通过**单触发模式（one—shot）**实现

单触发：

连续/单触发模块可通过单触发类型的操作来控制Mod4计数器的开始/停止和复位（零）功能；

该单触发类型操作可通过停止值比较器触发以及通过软件控制重新产生（re-armed）。

一旦产生，eCAP模块等待1-4个（由停止值定义）捕获事件，然后中止（freezing）Mod4计数器和CAP1-4寄存器的内容（即，时间戳）。

重新产生（re-arming）是为另一个捕获序列提供eCAP模块。同时重新产生将清除Mod4计数器并允许再次装载CAP1-4寄存器，假设CAPLDEN位置位。

连续模式：

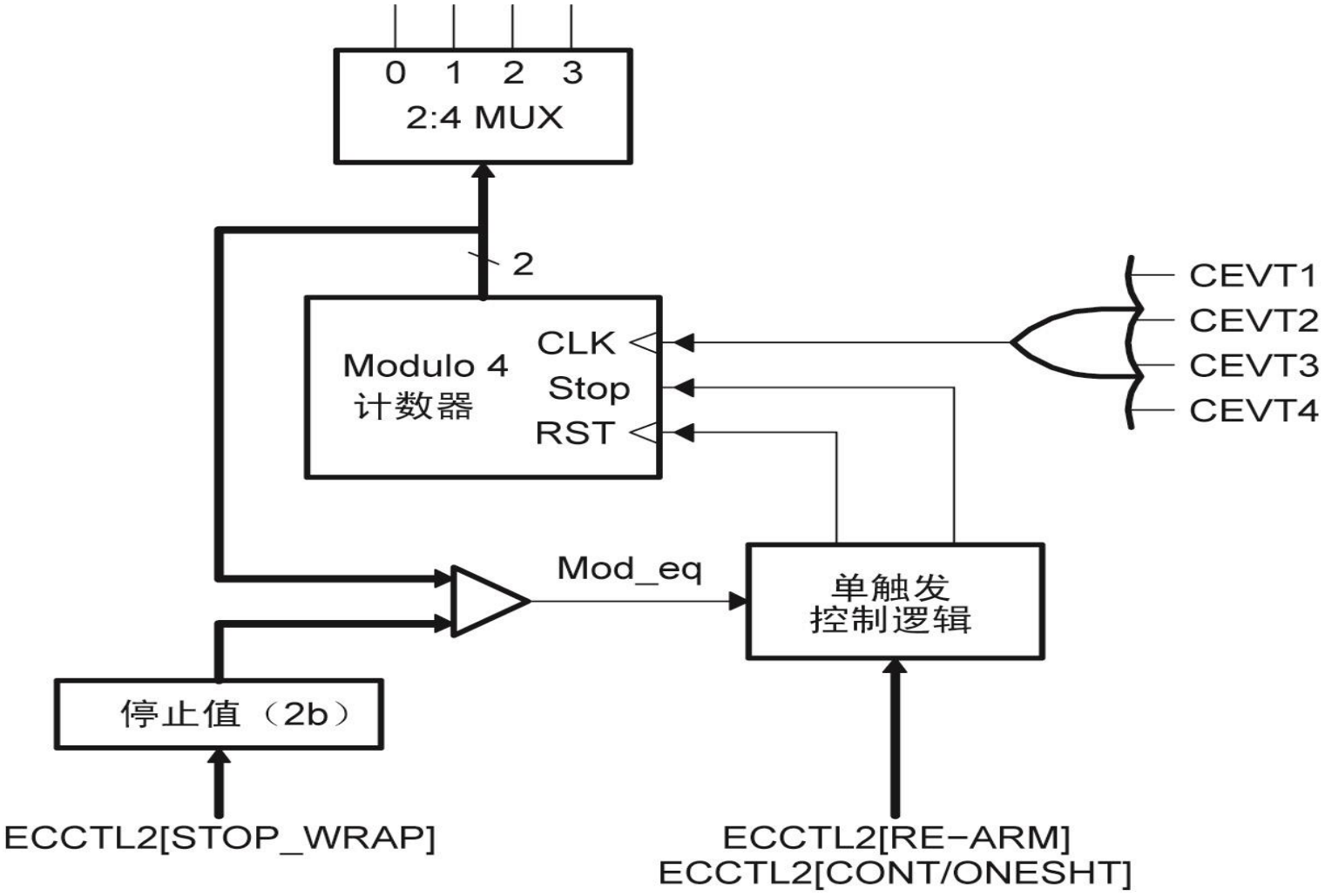
在连续模式中，

Mod4计数器连续运行（**0->1->2->3->0**），

单触发操作被忽略，

捕获值在循环缓冲区序列中继续被写入**CAP1-4**。

eCap 连续/单触发模块结构图



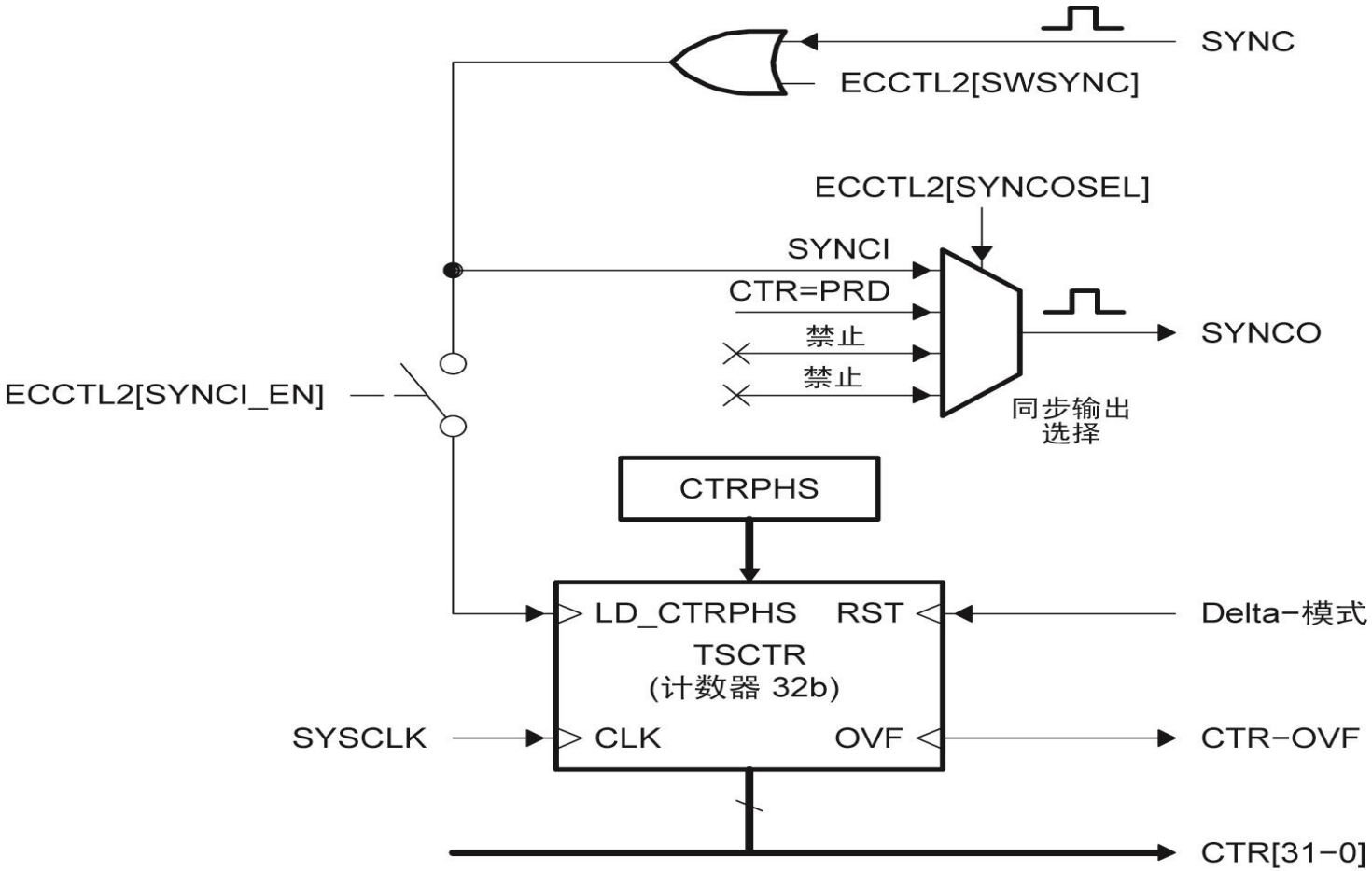
32位计数器和相位控制

- **32位计数器（TSCTR）通过系统时钟为事件捕捉提供事件基准。也就是：该计数器提供了事件捕获的时基，由系统时钟来驱动。**

- 相位寄存器（CTRPHS）通过软件和硬件实现计数器之间的同步。~~Apwm模块下，当模块之间需要相位偏差时，该同步操作十分有效。~~

- **4个事件中的任何一个都可以复位32位计数器，这对于偏差的捕捉非常有用。先捕捉32位计数器的值，之后任何一个LD1~LD4信号都可以将其复位为0。**

eCap计数器和同步模块框图



CAP1-CAP4寄存器

- 这些32位寄存器都连接在32位计数器定时器总线CTR[0-31]上，当选择了相应的装载指令（LD）时进行装载。

- 可以通过使能位CAPLDEN控制寄存器的装载。在单次控制模式中，当mod4计数器满足停止条件时，改位自动清零（禁止装载），即，StopValue=Mod4

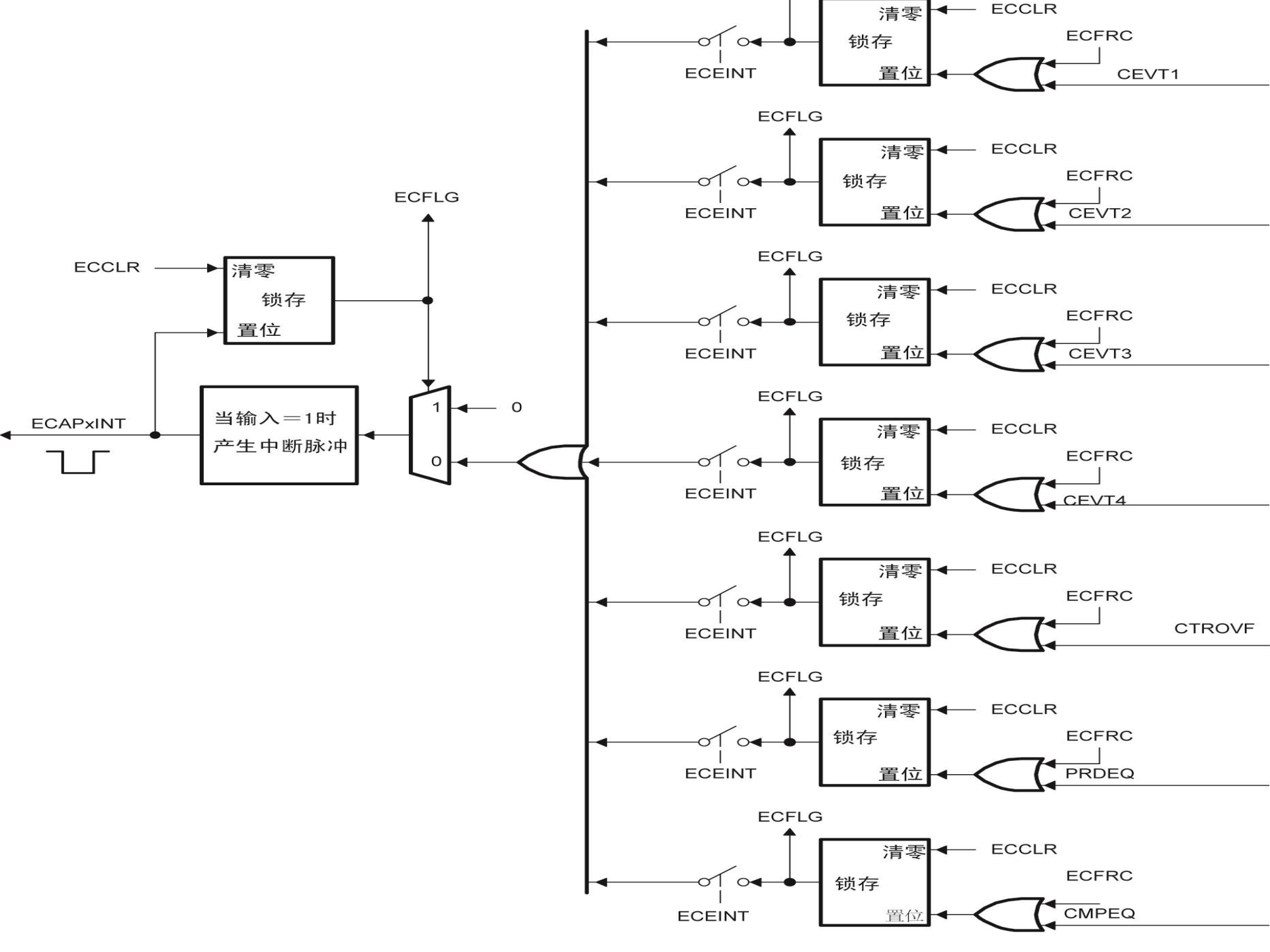
8.1.2 中断控制

1. 在捕捉事件（CEVT1-CEVT4,CTROVF）~~或APWM事件（CTR=PRD，CTR=CMP）~~中，都可以产生中断。

2. eCap一共可以产生以上7种中断。

3. 中断使能寄存器（ECEINT）可以使能/禁止单独的中断事件。如果任何一个中断事件被锁住，中断标志寄存器（ECEFLG）将进行标记，这其中包括全局中断标志位INT。

4. 在任何其他中断脉冲产生之前，中断服务程序必须通过中断清除寄存器（ECCCLR）清除全局中断标志位。通过中断强制寄存器（ECFRC）可以强制产生中断，这对于系统的调试是非常有用的


ECCLR
清零
锁存
置位
ECFLG
ECAPxINT
当输入＝1时
产生中断脉冲
1
0
0
ECEINT
ECFRC
CEVT1
CEVT2
CEVT3
CEVT4
CTROVF
PRDEQ
CMPEQ

8.2 eCap重要寄存器

表8.1　eCAP相关寄存器

名称	偏移量	大小（×16）	描述
TSCTR	0x0000	2	时间戳计数器：用作捕获时基
CTRPHS	0x0002	2	计数器相位偏移值寄存器：用于SYNCI事件或通过控制位进行软件强制同步
CAP1	0x0004	2	捕获1寄存器
CAP2	0x0006	2	捕获2寄存器
CAP3	0x0008	2	捕获3寄存器
CAP4	0x000A	2	捕获4寄存器
ECCTL1	0x0014	1	捕获控制寄存器1
ECCTL2	0x0015	1	捕获控制寄存器2
ECEINT	0x0016	1	捕获中断使能寄存器
ECFLG	0x0017	1	捕获中断标志寄存器
ECCLR	0x0018	1	捕获中断清除寄存器
ECFRC	0x0019	1	捕获中断强制寄存器

1）时间戳计数器寄存器（TSCTR）

TSCTR 为32位计数器寄存器，用作捕获时基。该寄存器可读可修改，复位时取值为0。

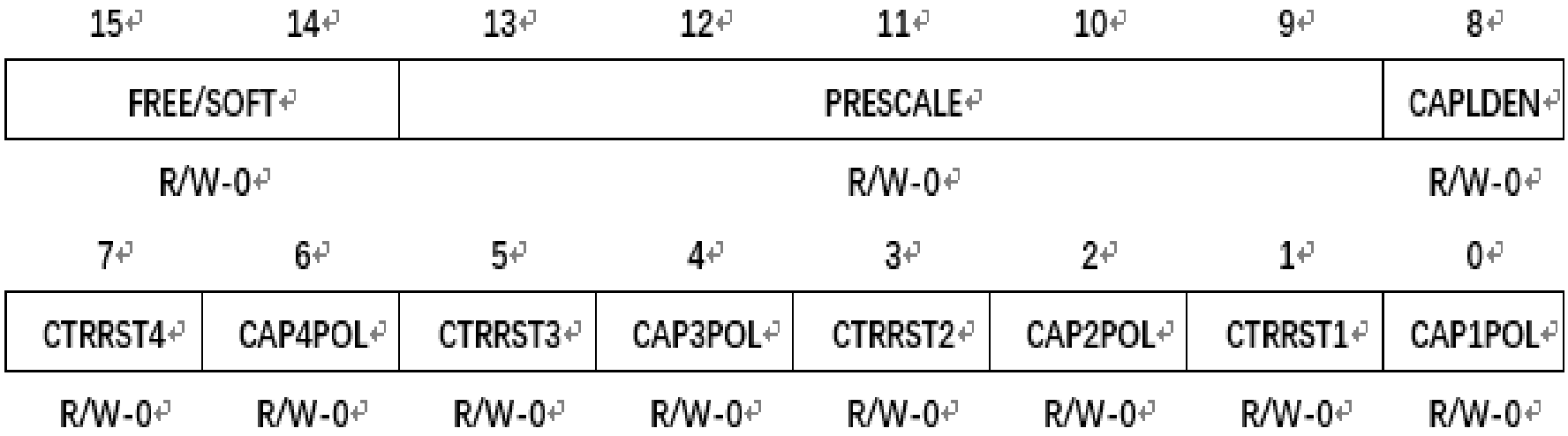
2）计数器相位控制寄存器（CTRPHS）

CTRPHS为可编程计数器相位值寄存器，用于相位滞后/超前。该寄存器在发生SYNCI事件或通过控制位进行软件S/W强制时映像TSCTR并装载到TSCTR，用于实现与其它eCAP和ePWM时基相关的相位控制同步。该寄存器可读可修改，复位时取值为0。

3）捕获寄存器（CAP1~CAP4）

CAP1~CAP4为捕获事件过程的时间戳（即计数器值TSCTR）；可软件写入，以用作测试和初始化。该寄存器可读可修改，复位时取值为0。

4) eCAP控制寄存器1（ECCTL1）字段描述



15	14	13	12	11	10	9	8
FREE/SOFT		PRESCALE					CAPLDEN
R/W-0		R/W-0					R/W-0

7	6	5	4	3	2	1	0
CTRRST4	CAP4POL	CTRRST3	CAP3POL	CTRRST2	CAP2POL	CTRRST1	CAP1POL
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

ECAP控制寄存器1（ECCTL1）字段描述

位	字段	值	描述
15:14	FREE/SOFT		模拟控制
		00	在模拟中止时TSCTR计数器立即停止
		01	TSCTR计数器运行直至＝0
		1x	TSCTR计数器不受模拟中止的影响（自由运行）
13:9	PRESCALE		事件过滤器预分频选择
		00000	通过1分频（即，无预分频、旁路预分频器）
		00001	通过2分频
		00010	通过4分频
		00011	通过6分频
		00100	通过8分频
		00101	通过10分频
		…	
		11110	通过60分频
		11111	通过62分频
8	CAPLDEN		在出现捕获事件时使能装载CAP1-4寄存器
		0	在捕获事件时禁止装载CAP1-4寄存器
		1	在捕获事件时使能装载CAP1-4寄存器

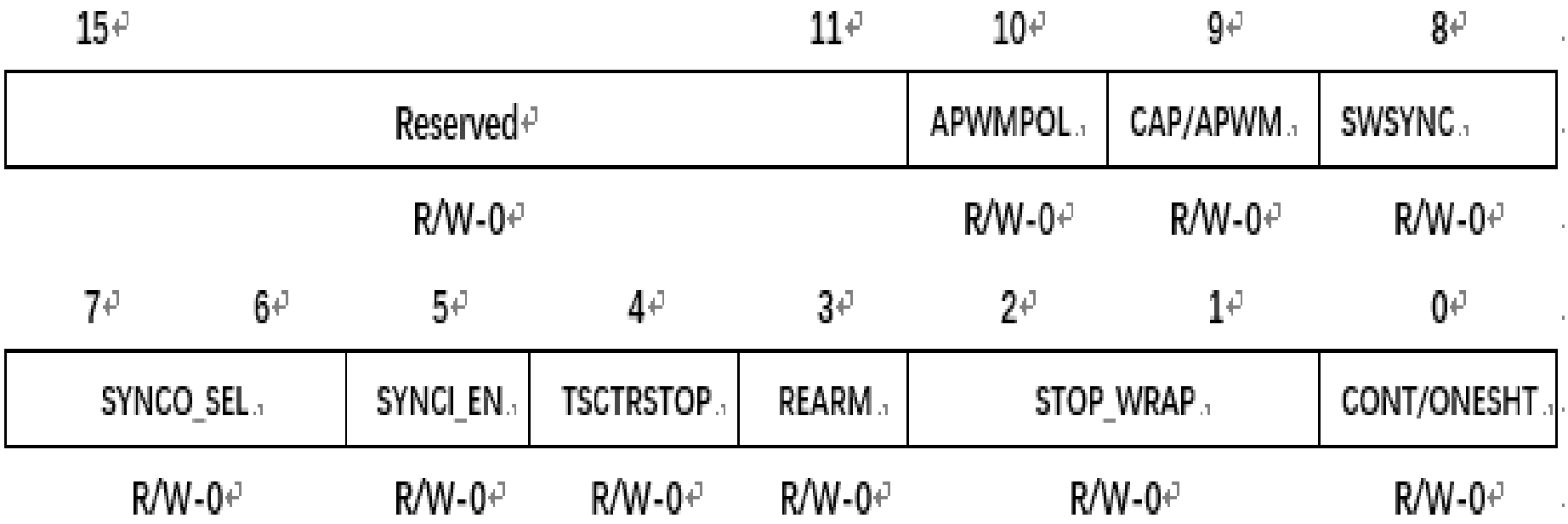
ECAP控制寄存器1（ECCTL1）字段描述

位	字段	值	描述
7	CTRRST4	0 1	在出现捕获事件4时计数器复位 在捕获事件4出现时不复位计数器（绝对时间戳操作） 在捕获事件4时间戳已被捕获后复位计数器（用于差分模式操作）
6	CAP4POL	0 1	捕获事件4极性选择 在上升沿（RE）上触发捕获事件4 在下降沿（FE）上触发捕获事件4
5	CTRRST3	0 1	在出现捕获事件3时计数器复位 在捕获事件3出现时不复位计数器（绝对时间戳操作） 在事件3时间戳已被捕获后复位计数器（用于差分模式操作）
4	CAP3POL	0 1	捕获事件3极性选择 在上升沿（RE）上触发捕获事件3 在下降沿（FE）上触发捕获事件3

ECAP控制寄存器1（ECCTL1）字段描述

位	字段	值	描述
3	CTRRST2	0 1	在出现捕获事件2时计数器复位 在捕获事件2出现时不复位计数器（绝对时间戳操作） 在事件2时间戳已被捕获后复位计数器（用于差分模式操作）
2	**CAP2POL**	0 1	捕获事件2极性选择 在上升沿（RE）上触发捕获事件2 在下降沿（FE）上触发捕获事件2
1	CTRRST1	0 1	在出现捕获事件1时计数器复位 在捕获事件1出现时不复位计数器（绝对时间戳操作） 在事件1时间戳已被捕获后复位计数器（用于差分模式操作）
0	CAP1POL	0 1	捕获事件1极性选择 在上升沿（RE）上触发捕获事件1 在下降沿（FE）上触发捕获事件1

5) ECAP控制寄存器2（ECCTL2）



位	字段		描述
15:11	Reserved		保留
10	APWMPOL		APWM输出极性选择。这仅可用于APWM操作模式
		0	输出高电平有效（即，比较值定义了高电平时间）
		1	输出低电平有效（即，比较值定义了低电平时间）
9	CAP/APWM		CAP/APWM操作模式选择
		0	ECAP模块在捕获模式下操作。该模式强制下面的配置： 禁止TSCTR通过CTR=PRD事件复位； 禁止在CAP1和CAP2寄存器上装载影像； 允许用户使能CAP1-4寄存器装载； CAPx/APWMx管脚用作捕获输入
		1	ECAP模块在APWM模式下操作。该模式强制下面的配置： 在出现CTR=PRD事件时复位TSCTR（周期边界）； 允许在CAP1和CAP2寄存器上实现影像装载； 禁止将时间戳装载到CAP1-4寄存器中； CAPx/APWMx管脚用作APWM输出

位	字段		描述
8	SWSYNC	0 1	软件强制的计数器（TSCTR）同步。这提供了一种便利的软件方法来同步某些或所有ECAP时基。在APWM模式中，同步也可通过CTR=PRD事件来实现 写0没有影响。读总是返回0 假设SYNCO_SEL位为0,0，则写1强制当前ECAP模块和任何ECAP模块下行（down-stream）的TSCTR影像装载。在写1后，该位返回0 注：选择CTR=PRD仅在APWM模式中有意义；但是，如果你认为该操作有用，则可在CAP模式中对其进行选择
7:6	SYNCO_SEL	00 01 10 11	同步输出选择 把同步输入事件选为同步输出信号（直接通过） 把CTR=PRD事件选为同步输出信号 禁止同步输出信号 禁止同步输出信号
5	SYNCI_EN	0 1	计数器（TSCTR）同步输入选择模式 禁止同步输入选项 在出现SYNCI信号或S/W强制事件时使能计数器（TSCTR）从CTRPHS寄存器中装载

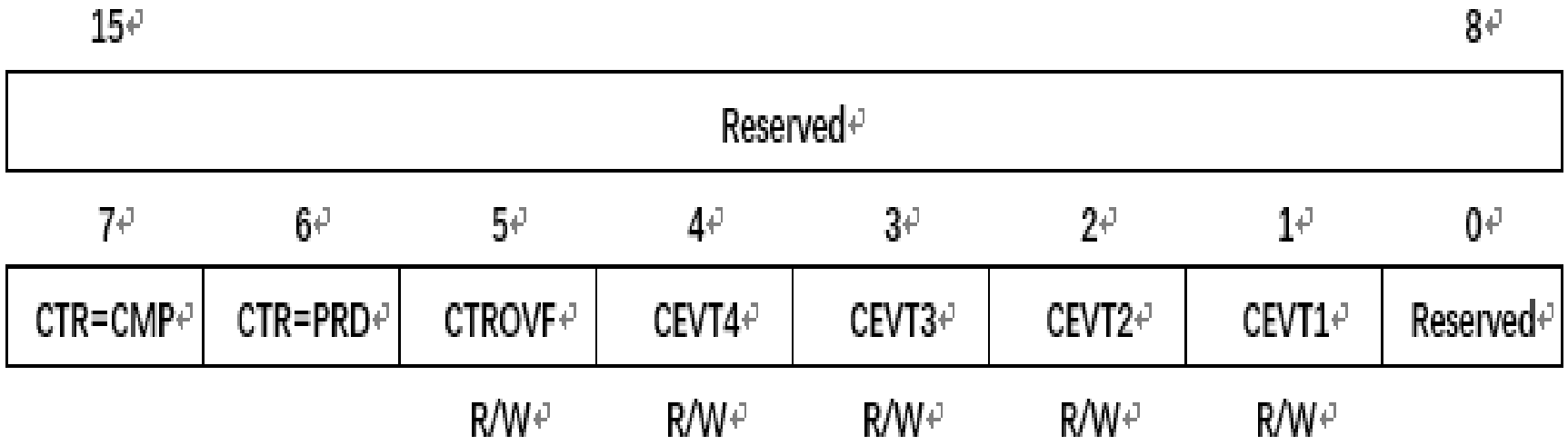
4	TSCTRSTOP	0 1	时间戳（TSCTR）计数器停止（冻结）控制 TSCTR停止 TSCTR自由运行
3	RE-ARM	0 1	单触发重新产生（re-arming）控制，也就是等待停止触发。注：重新产生功能在单触发模式或连续模式中均有效 没有影响（读总是返回0） 产生单触发的顺序如下： 1）将Mod4计数器复位为0 2）恢复Mod4计数器 3）使能捕获寄存器装载

2:1	STOP_WRAP		单触发模式的停止值。这是在CAP（1-4）寄存器冻结（即捕获序列停止）之前允许出现的捕获数（1-4） 连续模式的循环值。这是在循环缓冲区中不断循环的捕获寄存器的数（在1至4间）
		00	在单触发模式中捕获事件1后停止 在连续模式中捕获事件1后循环
		01	在单触发模式中捕获事件2后停止 在连续模式中捕获事件2后循环
		10	在单触发模式中捕获事件3后停止 在连续模式中捕获事件3后循环
		11	在单触发模式中捕获事件4后停止 在连续模式中捕获事件4后循环 注：STOP_WRAP与Mod4计数器相比较，当相等时，出现2种操作： Mod4计数器停止（冻结）； 禁止捕获寄存器装载 在单触发模式中，阻止更多的中断事件直至重新产生中断事件
0	CONT/ONESHT		连续或单触发模式控制（仅在捕获模式中可以应用）
		0	在连续模式中操作
		1	在单触发模式中操作

6）中断相关寄存器

包括中断使能寄存器（ECEINT）、
中断标志寄存器（ECFLAG）、
中断清零寄存器（ECCLR）
和中断强制寄存器（ECFRC）。

（1）ECAP 中断使能寄存器（ECEINT）



位	字段	值	描述
15:8	Reserved		
7	CTR=CMP		计数器相等比较中断使能
		0	禁止比较相等事件作为中断源
		1	使能比较相等事件作为中断源
6	CTR=PRD		计数器相等周期中断使能
		0	禁止周期相等事件作为中断源
		1	使能周期相等事件作为中断源
5	CTROVF		计数器溢出中断使能
		0	禁止计数器溢出事件作为中断源
		1	使能计数器溢出事件作为中断源

4	CEVT4	0 1	捕获事件4中断使能 禁止捕获事件4作为中断源 捕获事件4中断使能
3	CEVT3	0 1	捕获事件3中断使能 禁止捕获事件3作为中断源 使能捕获事件3作为中断源
2	CEVT2	0 1	捕获事件2中断使能 禁止捕获事件2作为中断源 使能捕获事件2作为中断源
1	CEVT1	0 1	捕获事件1中断使能 禁止捕获事件1作为中断源 使能捕获事件1作为中断源
0	Reserved		

（2）ECAP 中断标志寄存器（ECFLAG）

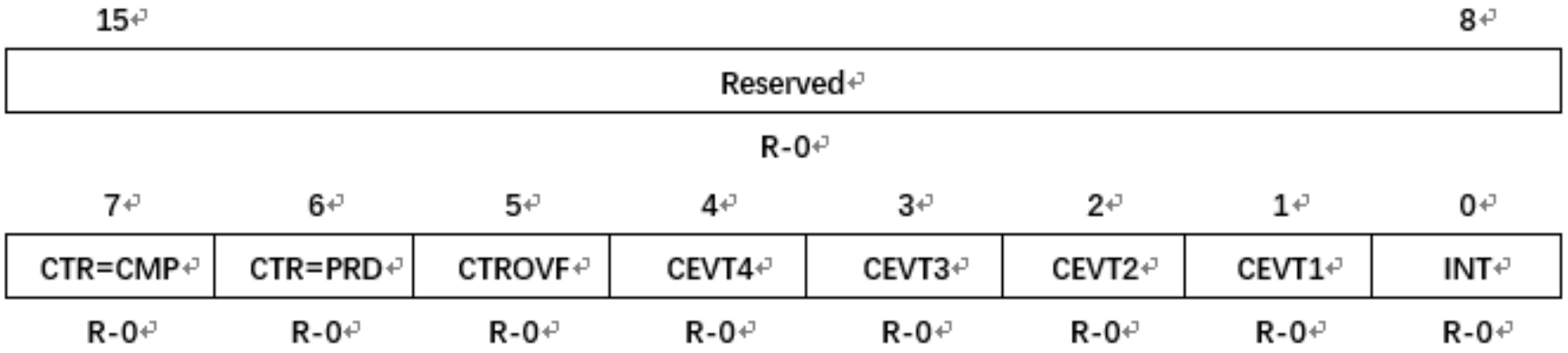


图 8.7　ECAP 中断标志寄存器（ECFLAG）

位	字段	值	描述
15:8	Reserved		
7	CTR=CMP	0 1	计数器相等比较状态标志。该标志仅在APWM模式中有效 表示没有出现事件 表示计数器（TSCTR）到达比较寄存器值（ACMP）
6	CTR=PRD	0 1	计数器相等周期状态标志。该标志仅在APWM模式中有效 表示没有出现事件 表示计数器（TSCTR）到达周期寄存器值（APRD）并复位
5	CTROVF	0 1	计数器溢出状态标志。该标志在CAP和APWM模式中有效 表示没有出现事件 表示计数器（TSCTR）从FFFFFFFF跳变为00000000

4	CEVT4	0 1	捕获事件4状态标志。该标志仅在CAP模式中有效 表示没有出现事件 表示第4个事件在ECAPx管脚出现
3	CEVT3	0 1	捕获事件3状态标志。该标志仅在CAP模式中有效 表示没有出现事件 表示第3个事件在ECAPx管脚出现
2	CEVT2	0 1	捕获事件2状态标志。该标志仅在CAP模式中有效 表示没有出现事件 表示第2个事件在ECAPx管脚出现
1	CEVT1	0 1	捕获事件1状态标志。该标志仅在CAP模式中有效 表示没有出现事件 表示第1个事件在ECAPx管脚出现
0	INT	0 1	全局中断状态标志 表示没有产生中断 表示已产生一个中断

（3） ECAP 中断清零寄存器（ECCLR）

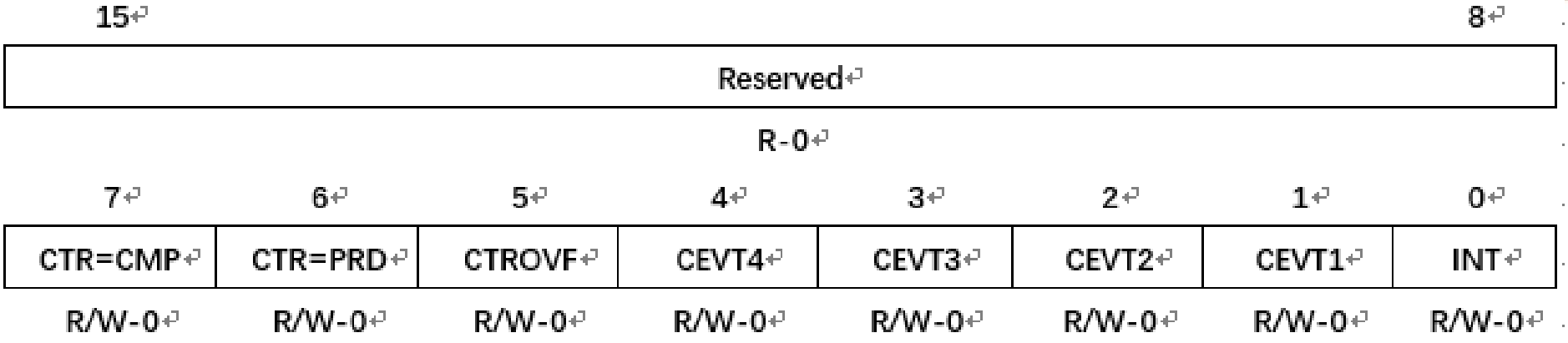


图 8.8 ECAP 中断清零寄存器（ECCLR）

位	字段	值	描述
15:8	Reserved		
7：0	XXXX	0 1	写0没有影响，总是读回0 写1清除相应的标志条件

（4）ECAP 中断强制寄存器（ECFRC）

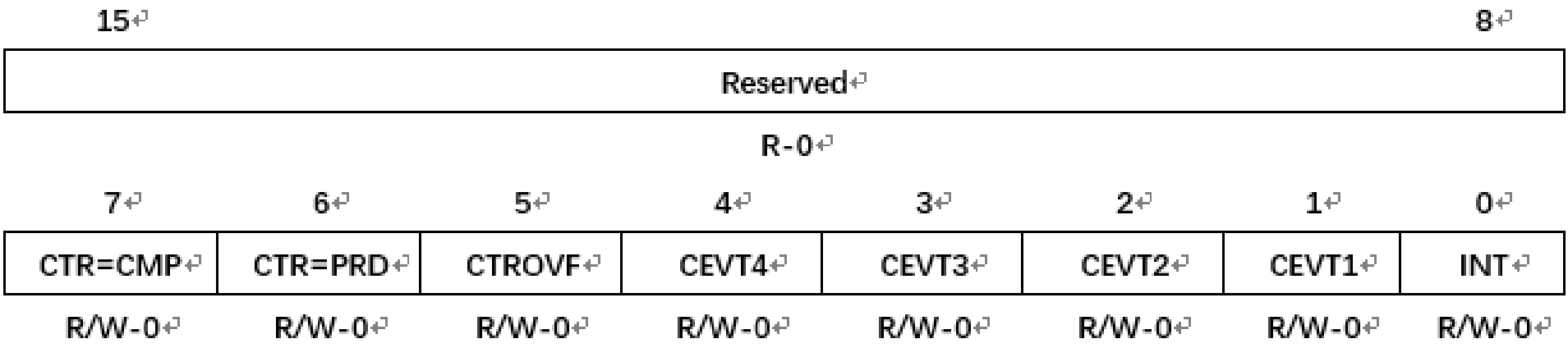


图 8.9　ECAP 中断强制寄存器（ECFRC）

位	字段	值	描述
15:8	Reserved		
7：0	XXXX	0 1	强制 XXXX 中断 写0没有影响，总是读回0 写1，置位XXXX 标志位

7）eCAP头文件中寄存器结构体说明

头文件中包括一个寄存器结构体ECAP_REGS寄存器结构体中的位定义与前面介绍的寄存器位定义相同。

```
struct ECAP_REGS {
   Uint32     TSCTR;                    // 时间戳计数器寄存器
   Uint32     CTRPHS;                   // 计数器相位控制寄存器
   Uint32     CAP1;             // 捕获-1寄存器
   Uint32     CAP2;             // 捕获-2寄存器
   Uint32     CAP3;             // 捕获-3寄存器
   Uint32     CAP4;             // 捕获-4寄存器
   Uint16      rsvd1[8];                // 保留
   union   ECCTL1_REG  ECCTL1;     // ECAP控制寄存器1
   union   ECCTL2_REG  ECCTL2;     // ECAP控制寄存器2
   union   ECEINT_REG  ECEINT;     // ECAP中断使能寄存器
   union   ECFLG_REG   ECFLG;                  // ECAP中断标志寄存器
   union   ECFLG_REG   ECCLR;      // ECAP中断清零寄存器
   union   ECEINT_REG  ECFRC;      // ECAP中断强制寄存器
   Uint16     rsvd2[6];                // 保留
};
volatile struct ECAP_REGS ECap1Regs;
```

8.3 eCap模块应用举例

•8.3.1 eCAP的使用

eCAP的应用程序一般包含两部分：
第一部分为初始化部分，对eCAP进行合理配置

第二部分为中断服务程序部分，读取CAP1~CAP4寄存器，并进行处理。

1）初始化程序

初始化程序包括配置外设模式和中断初始化

初始化正确步骤如下：

配置外设寄存器，选择eCAP模式；

禁止全局中断；

禁止eCAP中断；

填写中断向量；

使能eCAP中断；

使能全局中断。

2）中断服务程序

中断服务程序完成测量功能，主要包括：

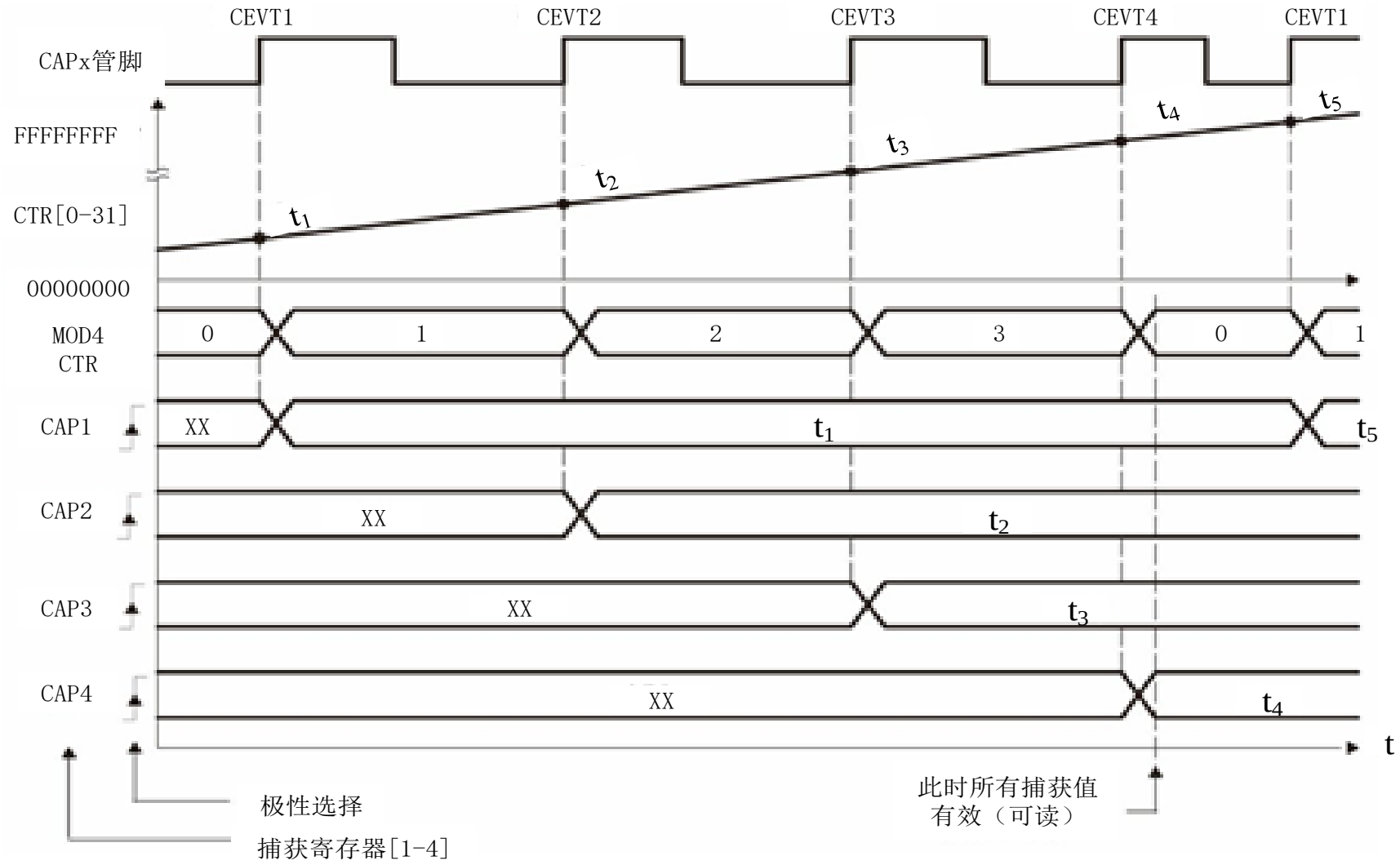
相关CAP的读取或溢出次数的累加；

所读取CAP的处理应用；

中断现场恢复，外设中断标志处理、PIEACK的处理。

8.3.2 eCAP的应用举例

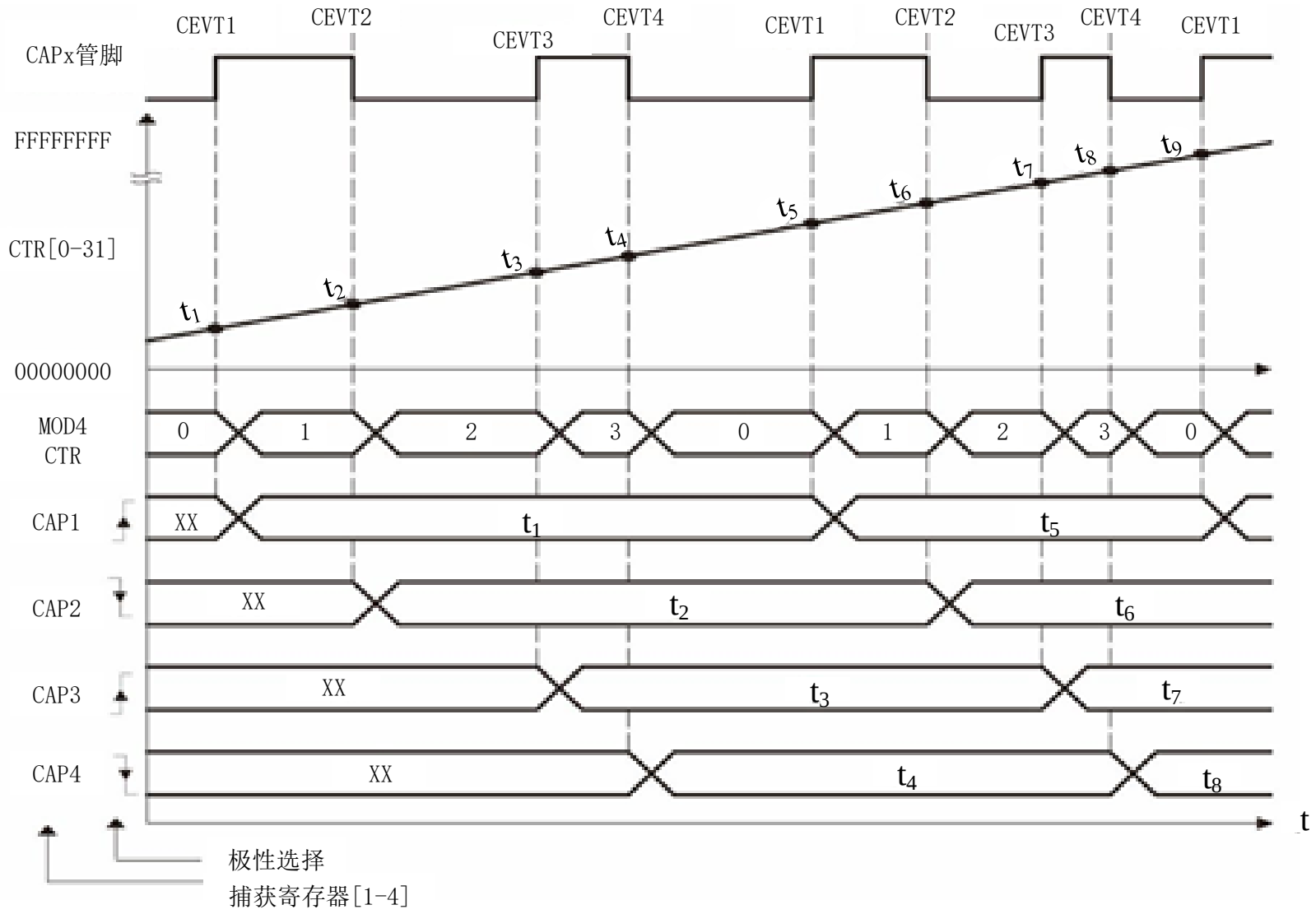
示例1—绝对时间戳操作，上升沿触发



```
ECap1Regs.ECCTL1.bit.CAP1POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP2POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP3POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP4POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CTRRST1 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CTRRST2 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CTRRST3 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CTRRST4 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CAPLDEN = EC_ENABLE;
ECap1Regs.ECCTL1.bit.PRESCALE = EC_DIV1;
ECap1Regs.ECCTL2.bit.CAP_APWM = EC_CAP_MODE;
ECap1Regs.ECCTL2.bit.CONT_ONESHT = EC_CONTINUOUS;
ECap1Regs.ECCTL2.bit.SYNCO_SEL = EC_SYNCO_DIS;
```

```
ECap1Regs.ECCTL2.bit.SYNCI_EN = EC_DISABLE;
ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN; // 允许TSCTR运行
ECap1Regs.ECEINT.bit.CEVT4 = 1; // CEVT4
// 运行时间（例如，CEVT4触发的ISR调用）
//===================================
TSt1 = ECap1Regs.CAP1; // 在t1捕获预取指时间戳
TSt2 = ECap1Regs.CAP2; //在t2捕获预取指时间戳
TSt3 = ECap1Regs.CAP3; //在t3捕获预取指时间戳
TSt4 = ECap1Regs.CAP4; //在t4捕获预取指时间戳
Period1 = TSt2-TSt1; // 计算第1个周期
Period2 = TSt3-TSt2; //计算第2个周期
Period3 = TSt4-TSt3; //计算第3个周期
ECap1Regs.ECCLR.bit.CEVT4 = 1;
ECap1Regs.ECCLR.bit.INT = 1;
PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;
```

示例2—绝对时间戳操作，上升沿和下降沿触发



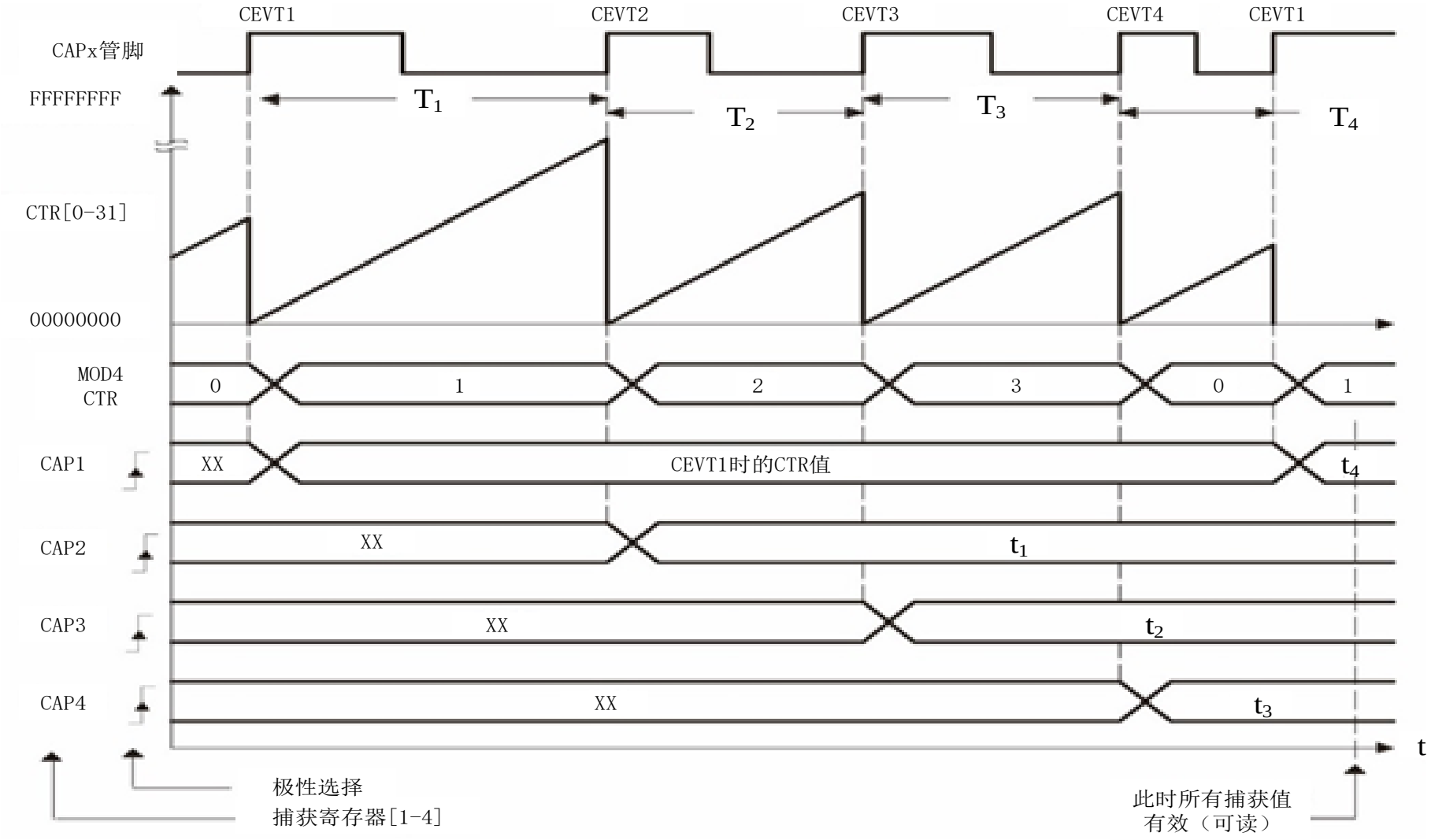
eCap

```
ECap1Regs.ECCTL1.bit.CAP1POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP2POL = EC_FALLING;
ECap1Regs.ECCTL1.bit.CAP3POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP4POL = EC_FALLING;
ECap1Regs.ECCTL1.bit.CTRRST1 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CTRRST2 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CTRRST3 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CTRRST4 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CAPLDEN = EC_ENABLE;
ECap1Regs.ECCTL1.bit.PRESCALE = EC_DIV1;
ECap1Regs.ECCTL2.bit.CAP_APWM = EC_CAP_MODE;
ECap1Regs.ECCTL2.bit.CONT_ONESHT = EC_CONTINUOUS;
ECap1Regs.ECCTL2.bit.SYNCO_SEL = EC_SYNCO_DIS;
```

eCap

```
ECap1Regs.ECCTL2.bit.SYNCI_EN = EC_DISABLE;
ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN; // 允许TSCTR运行
ECap1Regs.ECEINT.bit.CEVT4 = 1; // CEVT4
// 运行时间（例如，CEVT4触发ISR调用）
//===================================
TSt1 = ECap1Regs.CAP1; //在t1捕获预取指时间戳
TSt2 = ECap1Regs.CAP2; //在t2捕获预取指时间戳
TSt3 = ECap1Regs.CAP3; //在t3捕获预取指时间戳
TSt4 = ECap1Regs.CAP4; //在t4捕获预取指时间戳
Period1 = TSt3-TSt1; // 计算第1个周期
DutyOnTime1 = TSt2-TSt1; // 计算导通时间
DutyOffTime1 = TSt3-TSt2; // 计算断开时间
ECap1Regs.ECCLR.bit.CEVT4 = 1;
ECap1Regs.ECCLR.bit.INT = 1;
PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;
```

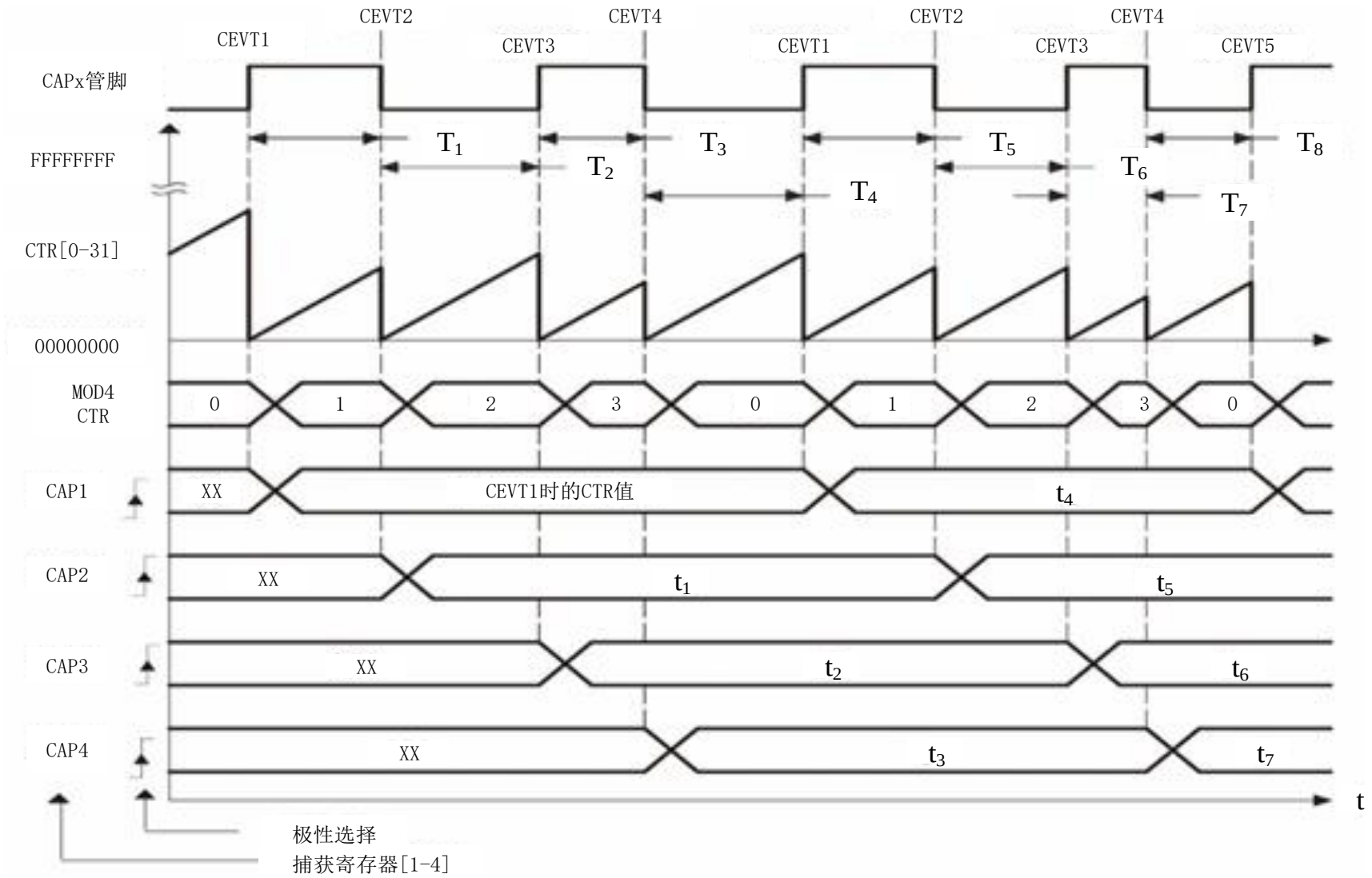
示例3—时间差（Delta）操作，上升沿触发



```
ECap1Regs.ECCTL1.bit.CAP1POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP2POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP3POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP4POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CTRRST1 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CTRRST2 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CTRRST3 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CTRRST4 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CAPLDEN = EC_ENABLE;
ECap1Regs.ECCTL1.bit.PRESCALE = EC_DIV1;
ECap1Regs.ECCTL2.bit.CAP_APWM = EC_CAP_MODE;
ECap1Regs.ECCTL2.bit.CONT_ONESHT = EC_CONTINUOUS;
ECap1Regs.ECCTL2.bit.SYNCO_SEL = EC_SYNCO_DIS;
```

```
ECap1Regs.ECCTL2.bit.SYNCI_EN = EC_DISABLE;
ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN; // 允许TSCTR运行
ECap1Regs.ECEINT.bit.CEVT1 = 1; // CEVT1
// 运行时间（例如，CEVT1触发ISR调用）
//=========================================
// 注：这里的时间戳直接反映周期值。
Period4 = ECap1Regs.CAP1; //在T1捕获预取指时间戳
Period1 = ECap1Regs.CAP2; //在T2捕获预取指时间戳
Period2 = ECap1Regs.CAP3; //在T3捕获预取指时间戳
Period3 = ECap1Regs.CAP4; //在T4捕获预取指时间戳
ECap1Regs.ECCLR.bit.CEVT1 = 1;
ECap1Regs.ECCLR.bit.INT = 1;
PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;
```

示例4—时间差（Delta）操作，上升沿和下降沿触发



```
ECap1Regs.ECCTL1.bit.CAP1POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP2POL = EC_FALLING;
ECap1Regs.ECCTL1.bit.CAP3POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP4POL = EC_FALLING;
ECap1Regs.ECCTL1.bit.CTRRST1 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CTRRST2 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CTRRST3 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CTRRST4 = EC_DELTA_MODE;
ECap1Regs.ECCTL1.bit.CAPLDEN = EC_ENABLE;
ECap1Regs.ECCTL1.bit.PRESCALE = EC_DIV1;
ECap1Regs.ECCTL2.bit.CAP_APWM = EC_CAP_MODE;
ECap1Regs.ECCTL2.bit.CONT_ONESHT = EC_CONTINUOUS;
ECap1Regs.ECCTL2.bit.SYNCO_SEL = EC_SYNCO_DIS;
```

```
ECap1Regs.ECCTL2.bit.SYNCI_EN = EC_DISABLE;
ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN; // 允许TSCTR运行
// 运行时间（例如，CEVT1触发的ISR调用）
//===========================================
//注：这里的时间戳直接反映周期值。
DutyOnTime1 = ECap1Regs.CAP2; //在T2捕获预取指时间戳
DutyOffTime1 = ECap1Regs.CAP3; //在T3捕获预取指时间戳
DutyOnTime2 = ECap1Regs.CAP4; //在T4捕获预取指时间戳
DutyOffTime2 = ECap1Regs.CAP1; //在T1捕获预取指时间戳
Period1 = DutyOnTime1 + DutyOffTime1;
Period2 = DutyOnTime2 + DutyOffTime2;
ECap1Regs.ECCLR.bit.CEVT1 = 1;
ECap1Regs.ECCLR.bit.INT = 1;
PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;
```

需要强调的是：

当eCAP用作低频脉冲信号测量时，往往需要允许CTROVF（计数器溢出）中断，周期、占空比的计算需要考虑计数器溢出的次数；

当eCAP用作高频脉冲信号测量时，为了提高测量的精度可以应用事件预分频的功能。

•**实验三 PWM和eCAP实验**

•**1） 实验要求**

•输出带死区的互补PWM波形；

•通过eCAP单元捕获其中一路PWM波形并获取其周期和占空比。

•**2） 实验目的**

•掌握PWM的周期、死区电平和死区时间的设定方法；

•掌握PWM计数周期中断和比较值匹配中断的配置方法。

•掌握eCAP单元的操作方法及其中断的配置；

•掌握eCAP单元时间测量和占空比测量的方法。

•**3） 实验说明**

•将EPWM1A/EPWM1B、EPWM4A/EPWM4B配置成输出带死区的互补PWM波形。

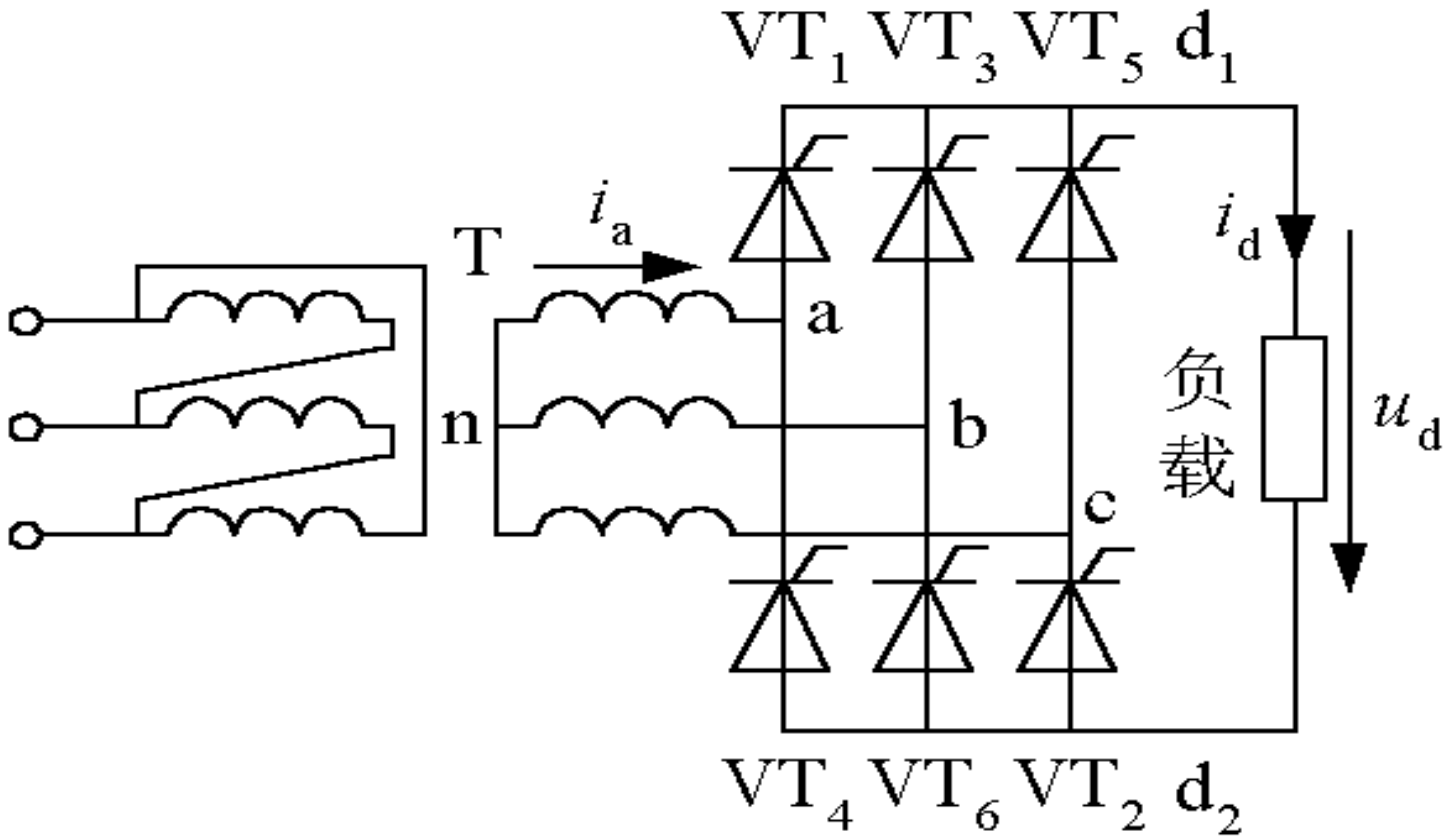
•在输出PWM的基础上，将其中一路PWM输出引脚接到eCAP1捕获引脚上，通过捕获单元获取PWM的周期和占空比信息。

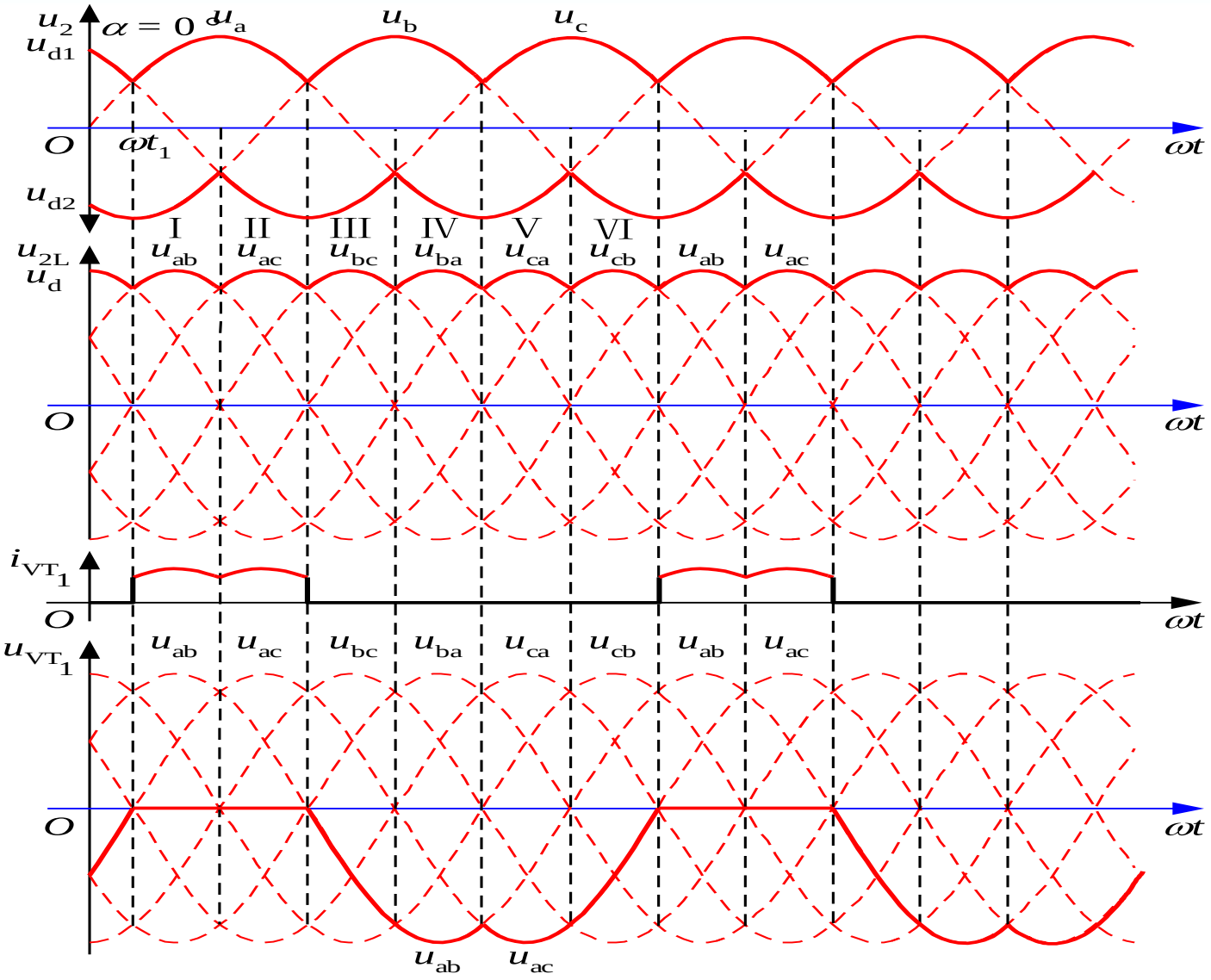
```
void InitCAP()
{
  ECap1Regs.ECEINT.all = 0x0000;           // Disable all capture interrupts
  ECap1Regs.ECCLR.all = 0xFFFF;            // Clear all CAP interrupt flags
  ECap1Regs.ECCTL1.bit.CAP1POL = 0;
  ECap1Regs.ECCTL1.bit.CAP2POL = 1;
  ECap1Regs.ECCTL1.bit.CAP3POL = 0;
  ECap1Regs.ECCTL1.bit.CAP4POL = 1;
  ECap1Regs.ECCTL1.bit.CTRRST1 = 0;
  ECap1Regs.ECCTL1.bit.CTRRST2 = 0;
  ECap1Regs.ECCTL1.bit.CTRRST3 = 0;
  ECap1Regs.ECCTL1.bit.CTRRST4 = 0;
  ECap1Regs.ECCTL1.bit.CAPLDEN = 1;
  ECap1Regs.ECCTL1.bit.PRESCALE = 0;
  ECap1Regs.ECCTL2.bit.CAP_APWM = 0;
  ECap1Regs.ECCTL2.bit.CONT_ONESHT = 0;
  ECap1Regs.ECCTL2.bit.SYNCO_SEL = 2;
  ECap1Regs.ECCTL2.bit.SYNCI_EN = 0;
  ECap1Regs.ECCTL2.bit.TSCTRSTOP = 1; // 允许TSCTR
  ECap1Regs.ECEINT.bit.CEVT4 = 1; // CEVT4
}
```

```
void InitCAPGpio()
{
    EALLOW;
    GpioCtrlRegs.GPAPUD.bit.GPIO5 = 0;    // Enable pull-up on GPIO5 (ECAP1)
    GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 3;   // Configure GPIO5 as ECAP1
     EDIS;
}

interrupt void Ecap1Int_isr(void)
{
  li1=ECap1Regs.CAP1;
  li2=ECap1Regs.CAP2;
  li3=ECap1Regs.CAP3;
  li4=ECap1Regs.CAP4;
  PWM_HI=((li2-li1)+(li4-li3)) >> 1;
  PWM_LO=li3-li2;
  PWM_PRD=((li3-li1)+(li4-li2)) >> 1;
  ECap1Regs.ECCLR.bit.CEVT4 = 1;
  ECap1Regs.ECCLR.bit.INT = 1;
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;
}
```

应用举例：三相全控整流滤波电路的数字触发控制




u_2
$\alpha = 0$
u_a
u_b
u_c
u_{d1}
O
ωt_1
ωt
u_{d2}
I
II
III
IV
V
VI
u_{ab}
u_{ac}
u_{bc}
u_{ba}
u_{ca}
u_{cb}
u_{2L}
u_d
i_{VT_1}
u_{VT_1}

解题思路：

测出Ua的过零点和周期，根据这个过零点和周期，可以得到下一个控制周期a的时刻，进而可以得到晶闸管的门极触发脉冲时刻。

1：捕获

初始化：

引脚配置，→ eCAP1

事件类型，→ 双沿捕获

用到的变量初试化，→捕获时刻备份值，周期值，驱动序列值

中断配置

1：捕获

中断服务程序：

现场保护

读取读取最近捕获的时刻

读取eCAP1的引脚状态，判断Ua的相位，

计算Ua周期

根据触发角a,确定下次CpuTimer0中断的时刻值和触发序列值

恢复现场

返回

2：定时器CpuTimer0

初始化：

CpuTimer0的周期值，

CpuTimer0中断初试化

中断服务程序：

现场保护

根据触发序列号，触发相应的晶闸管

重置CpuTimer0周期值

恢复现场

返回

小结：

1. 捕获：**能够记录事件发生（类型和时间）的外设**
   捕获与外部中断的差别：**事件更丰富、时间戳更强大、中断更灵活**

2. 捕获有哪几种方式：**绝对模式、△模式，+极性**
   各有何应用背景：

3. 捕获单元的构成和使用要点：**预分频、极性选择、时间戳、触发器、中断模块**

4. 捕获单元的中断使用方法：**ECEINT**

5. 重要的寄存器：**ECCTL1，ECCTL2，CAP1~4**