通用定时器

程晨闻
东南大学电气工程学院

➢ 作业

– 如果12位逐次逼近ADC的参考电压为3.3V，转换的结果为0x101，那么输入电压是多少伏？

$$U_{in} = 3.3V \cdot (\frac{1}{2^4} + \frac{1}{2^{12}}) \approx 0.2071\text{V} \qquad U_{in\_max} = 3.3V \cdot (\frac{1}{2^4} + \frac{1}{2^{11}}) \approx 0.2079\text{V}$$

DAC为4位，参考电压V_{REF}= 8V，输入电压U_i = 5.52V

顺 序	d_3 d_2 d_1 d_0	U_A(V)	比 较 判 断	“1”留否
1	1 0 0 0	4V	$U_A < U_I$	留
2	1 1 0 0	6V	$U_A > U_I$	去
3	1 0 1 0	5V	$U_A < U_I$	留
4	1 0 1 1	5. 5V	$U_A \approx U_I$	留

➢ 内容概要

- 通用定时器的用途
- TM4C1294的通用定时器模块
- 通用定时器的功能和使用方法：32位单次/周期计数功能
- 通用定时器的功能和使用方法：16位边沿计时功能

➢ 内容概要

– 通用定时器的用途

– TM4C1294的通用定时器模块

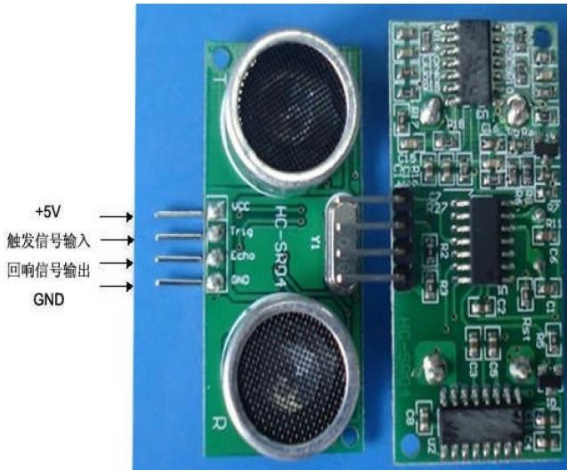
– 通用定时器的功能和使用方法：32位单次/周期计数功能

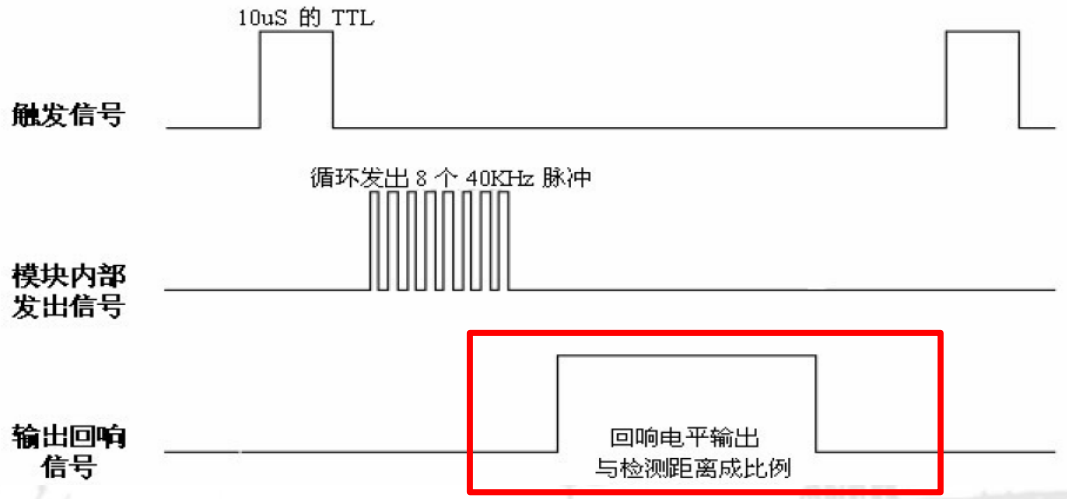
– 通用定时器的功能和使用方法：16位边沿计时功能

电网频率测量

先把正弦信号转换为方波信号
再测量两次上升沿的间隔时间

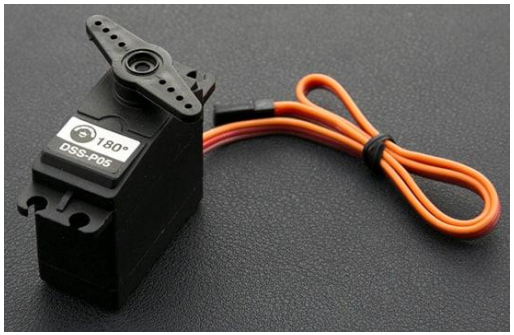
➢ 超声测距波传感器





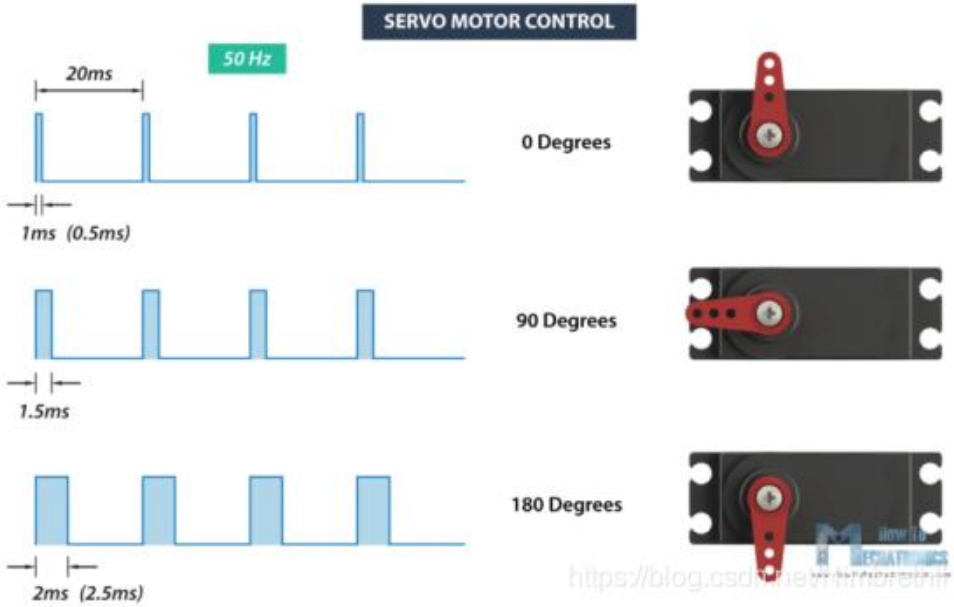
测量回响信号的高电平时间，可以得知声波传输的距离。

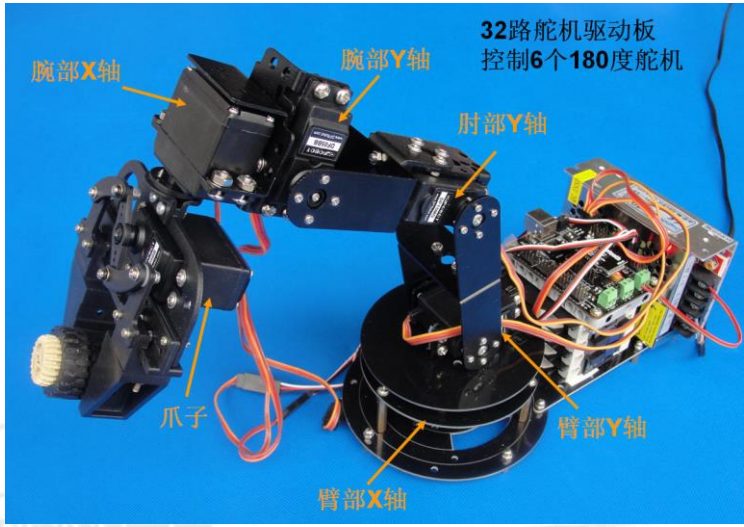
舵机控制



- 具备角度转动的一种执行部件
- 是一种位置（角度）伺服的驱动器，适用于那些需要角度不断变化并可以保持的闭环控制执行模块

伺服系统：是使物体的位置、方位、状态等输出，能够跟随输入量（或给定值）的任意变化而变化的自动控制系统。



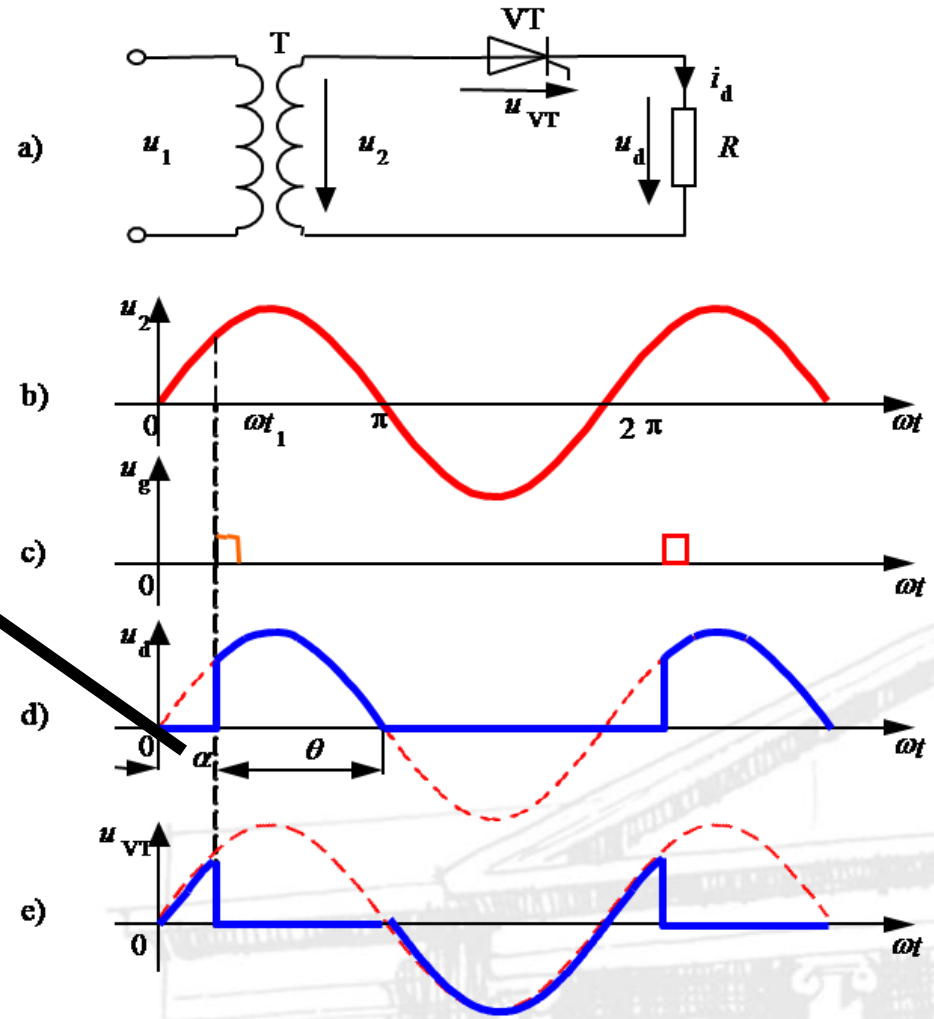


控制控制线上高电平脉冲的宽度，来控制旋转的角度

➢ 晶闸管整流电路的控制

精确的控制导通角α

延迟t=α/ω的时间后，
产生触发电压u_g



- **频率测量、超声波测距**
  - 精确测量两个电平边沿之间的时间
- **舵机控制、晶闸管触发**
  - 精确控制信号跳变的时间

内容概要

- 通用定时器的用途
- **TM4C1294的通用定时器模块**
- 通用定时器的功能和使用方法：32位单次/周期计数功能
- 通用定时器的功能和使用方法：16位边沿计时功能

TM4C1294定时器模块

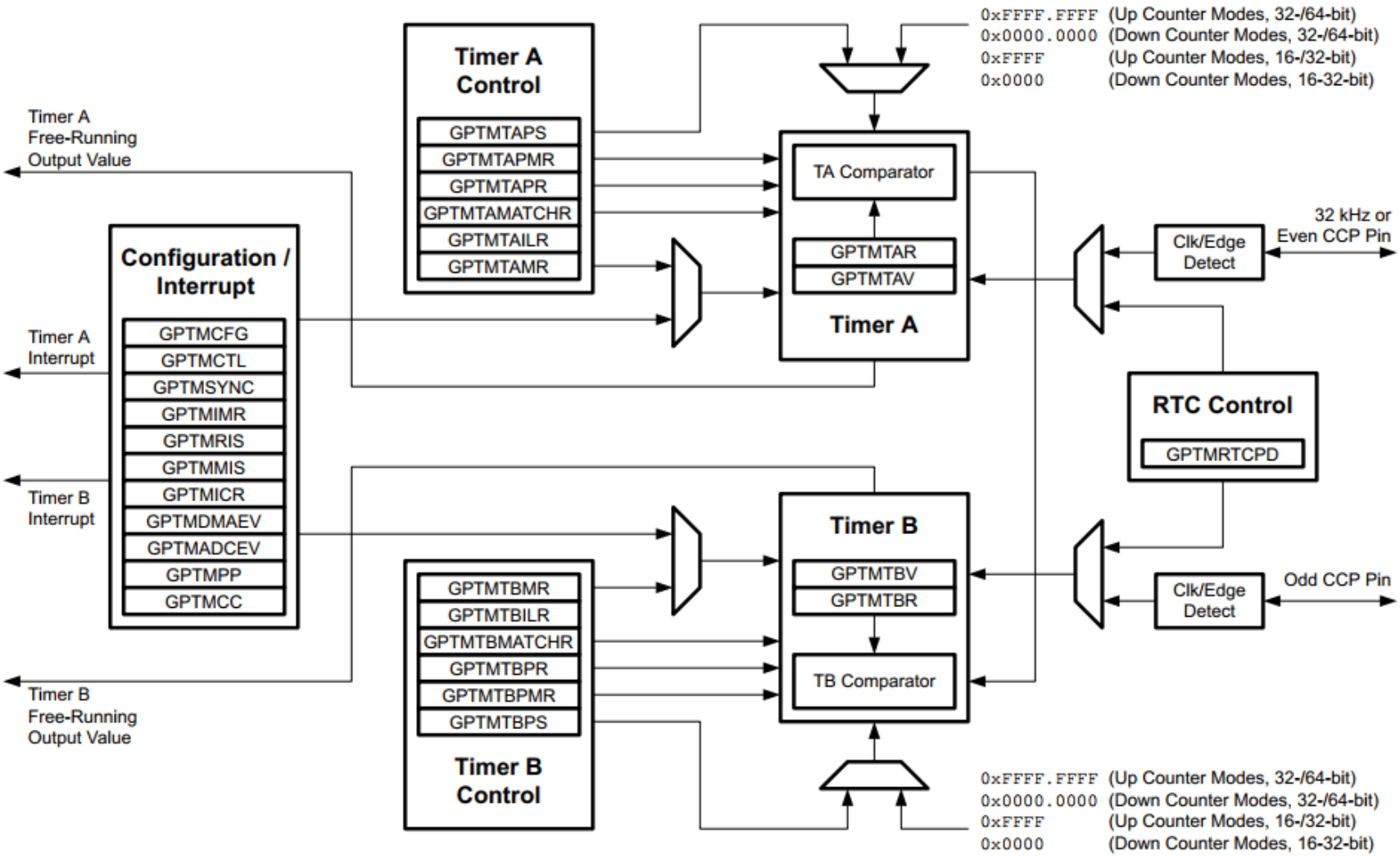
- 8个定时器模块
- 每个模块有两个16位定时计数器，可以单独运行，也可以组成一个32位定时器运行
- 计数的时钟来源可设置
- 计数方向可设置
- 有比较器功能
- 可以感知引脚的边沿变化
- 可控制引脚输出

定时器模块的功能

- **单次计数**
- **周期计数**
- **边沿计时**
- 边沿计数
- PWM模式
- 实时时钟

➢ TM4C1294定时器模块的结构

Figure 13-1. GPTM Module Block Diagram

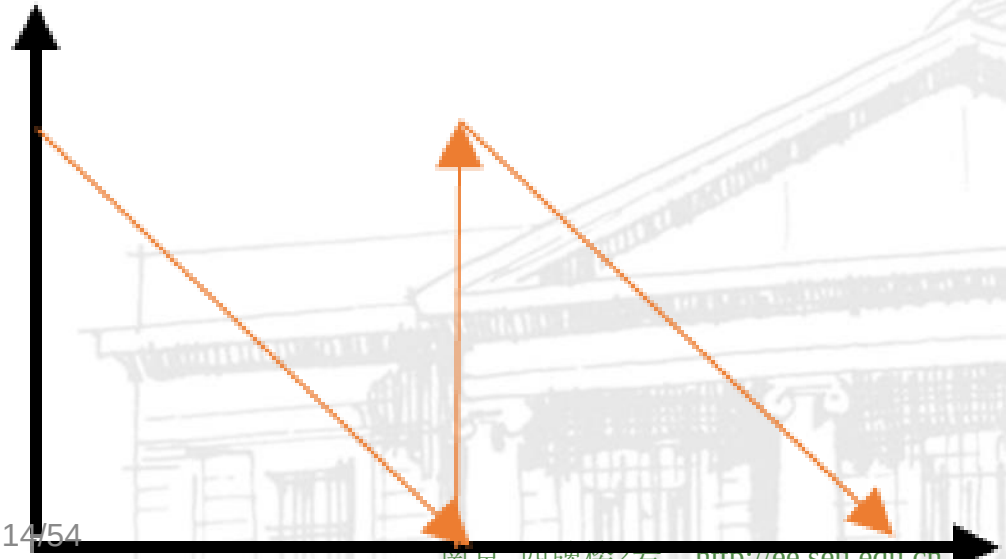


内容概要

- 通用定时器的用途
- TM4C1294的通用定时器模块
- **通用定时器的功能和使用方法：32位单次/周期计数功能**
- 通用定时器的功能和使用方法：16位边沿计时功能

32位单次/周期计数

- 单次或者周期性的重复计数，TimerA和TimerB联合使用组成32位计数器
- 默认为向下计数，也可以设置为向上计数
- 计数周期/时长由GPTM Timer A Interval Load (GPTM**TAILR**)寄存器决定
- 计数值存放在GPTM Timer A (GPTM**TAR**)和GPTM Timer A Value (GPTM**TAV**)寄存器中。其中低16位是TimerA的计数值，高16位是TimerB的计数值，与GPTM Timer B (GPTM**TBR**)和GPTM Timer B (GPTM**TBV**)寄存器的值一致。
- 计数超时后，会触发中断



➢ 32位单次/周期计数

- 1.使能定时器模块的时钟
- 2.配置定时器的功能
- 3.设置定时器的计数时长
- 4.打开并设置中断
- 5.开始计数
- 6.编写中断服务函数，清除中断，完成功能

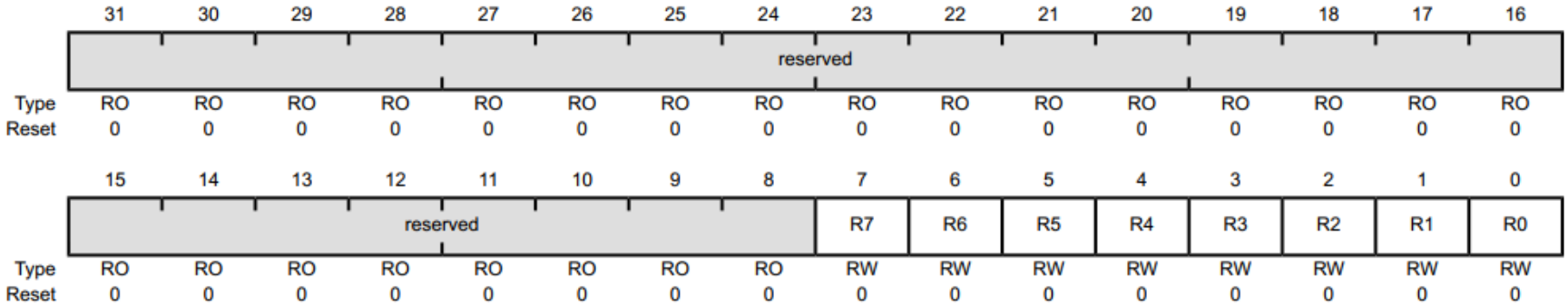
➢ 32位单次/周期计数

– **1.使能定时器模块的时钟**

SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

16/32-Bit General-Purpose Timer Run Mode Clock Gating Control (RCGCTIMER)

Base 0x400F.E000
Offset 0x604
Type RW, reset 0x0000.0000




```
#define SYSCTL_RCGCBASE         0x400fe600
#define SYSCTL_PERIPH_TIMER0    0xf0000400  // Timer 0
```

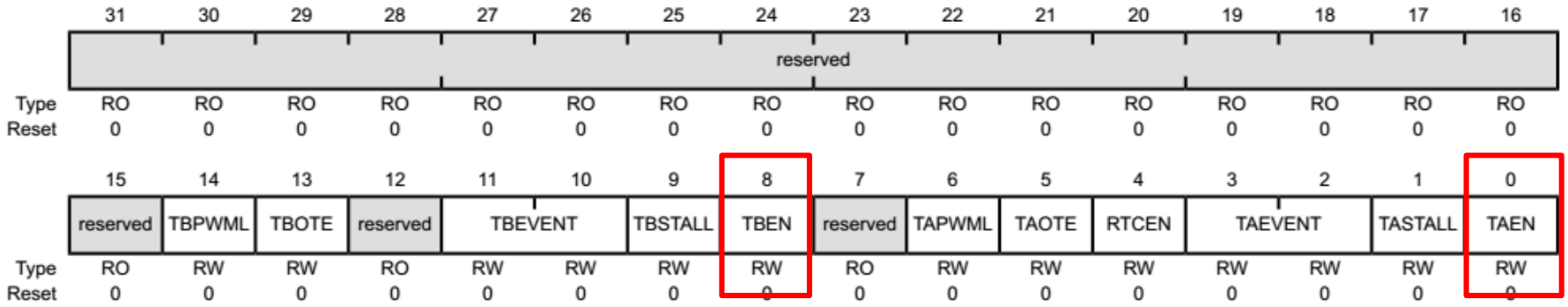
```
void
SysCtlPeripheralEnable(uint32_t ui32Peripheral)
{
    // Check the arguments.
    ASSERT(_SysCtlPeripheralValid(ui32Peripheral));
    // Enable this peripheral.
    HWREGBITW(SYSCTL_RCGCBASE + ((ui32Peripheral & 0xff00) >> 8),
              ui32Peripheral & 0xff) = 1;
}
```

➢ 32位单次/周期计数

– **2.配置定时器的功能**

2.1 配置之前，先关闭A、B两个计数器

GPTM Control (GPTM**CTL**)寄存器：



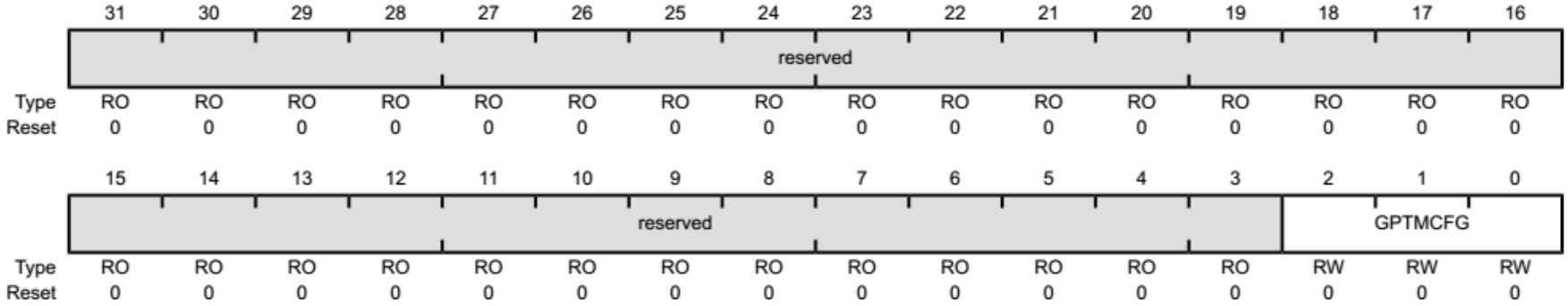
TAEN：计数器A使能控制，写0关闭，写1开启
TBEN：计数器B使能控制，写0关闭，写1开启

➢ 32位单次/周期计数

– **2.配置定时器的功能**

2.2 设置本通用定时器模块的计数模式

GPTM Configuration (**GPTMCFG**)寄存器



GPTMCFG域:

0：32位计数，使用TimerA作为32位计数器，对TimerB的配置无效

1：实时时钟模式

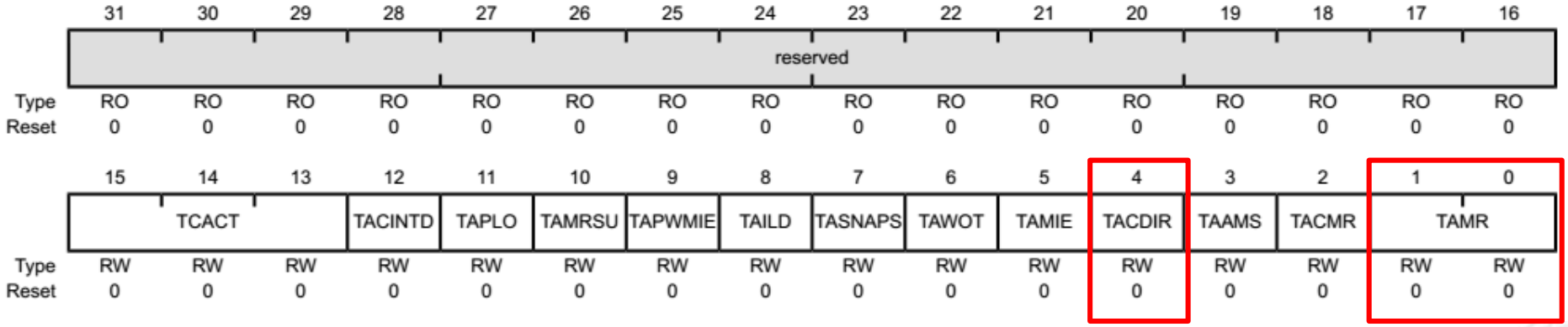
4：16位计数，即TimerA和TimerB分开计数

➢ 32位单次/周期计数

– 2.配置定时器的功能

2.3 设置TimerA的计数模式

GPTM Timer A Mode (**GPTMTAMR**)寄存器



TAMR域：

1：单次计数

2：周期计数

3：捕获模式

TACDIR域：

0：向下计数

1：从0开始向上计数

➢ 32位单次/周期计数

– **2.配置定时器的功能**

2.4 使用TivaWare提供的函数完成以上设置

```
void
TimerConfigure(uint32_t ui32Base, uint32_t ui32Config)
{
    // ...
    // Disable the timers.
    HWREG(ui32Base + TIMER_O_CTL) &= ~(TIMER_CTL_TAEN | TIMER_CTL_TBEN);
    // Set the global timer configuration.
    HWREG(ui32Base + TIMER_O_CFG) = ui32Config >> 24;
    // Set the configuration of the A and B timers and set the TxPWMIE bit.
    // Note that the B timer configuration is ignored by the hardware in 32-bit modes.
    if(NEW_TIMER_CONFIGURATION)
    {
        HWREG(ui32Base + TIMER_O_TAMR) = (((ui32Config & 0x000f0000) >> 4) |
                                          (ui32Config & 0xff) |
                                          TIMER_TAMR_TAPWMIE);
        HWREG(ui32Base + TIMER_O_TBMR) = (((ui32Config & 0x00f00000) >> 8) |
                                          ((ui32Config >> 8) & 0xff) |
                                          TIMER_TBMR_TBPWMIE);
    }
    else
    {
        HWREG(ui32Base + TIMER_O_TAMR) = ((ui32Config & 0xff) |
                                          TIMER_TAMR_TAPWMIE);
        HWREG(ui32Base + TIMER_O_TBMR) = (((ui32Config >> 8) & 0xff) |
                                          TIMER_TBMR_TBPWMIE);
    }
}
```

➢ 32位单次/周期计数

– **2.配置定时器的功能**

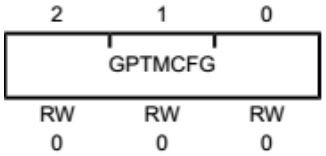
2.4 使用TivaWare提供的函数完成以上设置

TimerConfigure函数根据参数ui32Config的编码，设置GPTM**CFG**、GPTM**TAMR**和GPTM**TBMR**寄存器

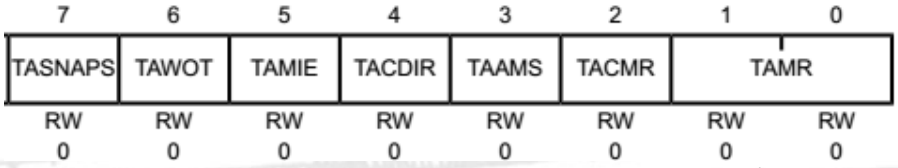
对于单次、周期计数，可选的参数有：

```
#define TIMER_CFG_ONE_SHOT          0x00000021
#define TIMER_CFG_ONE_SHOT_UP       0x00000031
#define TIMER_CFG_PERIODIC          0x00000022
#define TIMER_CFG_PERIODIC_UP       0x00000032
```

TIMER_O_CFG



GPTM**TAMR**



0：32位计数，使用TimerA作为32位计数器，对TimerB的配置无效。

1：实时时钟模式

4：16位计数，即TimerA和TimerB分开计数

TACDIR域：

0：向下计数

1：从0开始向上计数

TAMR域：

1：单次计数

2：周期计数

3：捕获模式

➢ 32位单次/周期计数

– **2.配置定时器的功能**

2.4 使用TivaWare提供的函数完成以上设置

TimerConfigure函数根据参数ui32Config的编码，设置GPTMCFG、GPTMTAMR和GPTMTBMR寄存器。

对于单次、周期计数，可选的参数有：

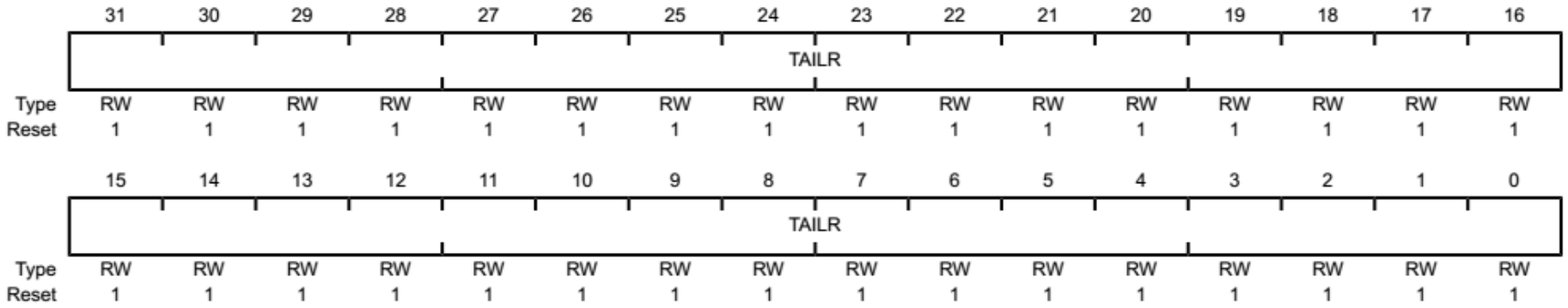
```
#define TIMER_CFG_ONE_SHOT          0x00000021
#define TIMER_CFG_ONE_SHOT_UP       0x00000031
#define TIMER_CFG_PERIODIC          0x00000022
#define TIMER_CFG_PERIODIC_UP       0x00000032
```

TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);

➢ 32位单次/周期计数

– **3. 设置定时器的计数时长**

GPTM Timer A Interval Load (**GPTMTAILR**)寄存器



如果是向下计数，计数开始后，计数器从**GPTMTAILR**开始计数，直到零为止。

如果是向上计数，计数开始后，计数器从零开始计数，直到**GPTMTAILR**为止。

➢ 32位单次/周期计数

– **3. 设置定时器的计数时长**

TivaWare提供了TimerLoadSet函数设置计数时长，其定义为：

```
void
TimerLoadSet(uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value)
{
    // Check the arguments.
    ASSERT(_TimerBaseValid(ui32Base));
    ASSERT((ui32Timer == TIMER_A) || (ui32Timer == TIMER_B) ||
           (ui32Timer == TIMER_BOTH));
    //
    // Set the timer A load value if requested.
    //
    if(ui32Timer & TIMER_A)
    {
        HWREG(ui32Base + TIMER_O_TAILR) = ui32Value;
    }

    //
    // Set the timer B load value if requested.
    //
    if(ui32Timer & TIMER_B)
    {
        HWREG(ui32Base + TIMER_O_TBILR) = ui32Value;
    }
}
```

➢ 32位单次/周期计数

– 3. 设置定时器的计数时长

TivaWare提供了TimerLoadSet函数设置计数时长，其用法为：

```
TimerLoadSet(TIMER0_BASE, TIMER_A, 1200000);
```

32位单次/周期计数

- **4. 打开并设置中断**
  - 一个定时器模块里的两个计数器，都可以向中断管理器发出中断请求
  - 32位计数，只用到了TimerA定时器，因此打开TimerA中断：

```
IntMasterEnable();
IntEnable(INT_TIMER0A);
```

➢ 32位单次/周期计数

– **4. 打开并设置中断**

GPTM Interrupt Mask (**GPTMIMR**)寄存器

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved		DMABIM	reserved	TBMIM	CBEIM	CBMIM	TBTOIM	reserved		DMAAIM	TAMIM	RTCIM	CAEIM	CAMIM	TATOIM
Type	RO	RO	RW	RO	RW	RW	RW	RW	RO	RO	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

计数器向下计数到零后或向上计数到GPTMTAILR，会触发**超时**中断

TATOIM：TimerA的超时中断请求使能，写1使能

```
void
TimerIntEnable(uint32_t ui32Base, uint32_t ui32IntFlags)
{
    ASSERT(_TimerBaseValid(ui32Base));
    HWREG(ui32Base + TIMER_O_IMR) |= ui32IntFlags;
}
```

```
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
```

➢ 32位单次/周期计数

– 4. 打开并设置中断

TivaWare提供了TimerIntRegister函数，用于注册中断服务函数

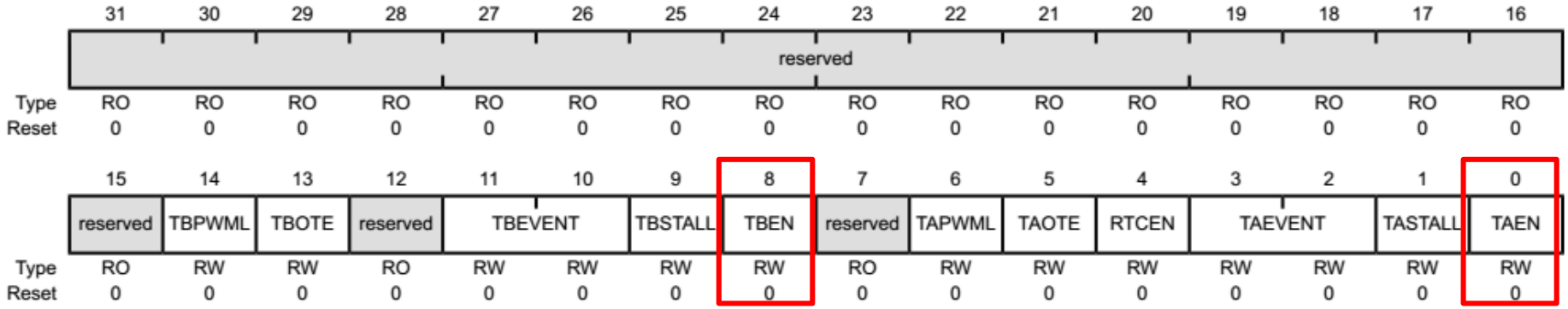
- 一个定时器模块里的两个计数器，都可以向中断管理器发出中断请求
- 32位计数，只用到了TimerA定时器，因此注册TimerA中断服务函数：

```
TimerIntRegister(TIMER0_BASE,TIMER_A,Timer0AIntHandler);
```

➢ 32位单次/周期计数

– **5.** 开始计数

GPTM Control (GPTMCTL)寄存器:



TAEN：计数器A使能控制，写0关闭，写1开启

TBEN：计数器B使能控制，写0关闭，写1开启

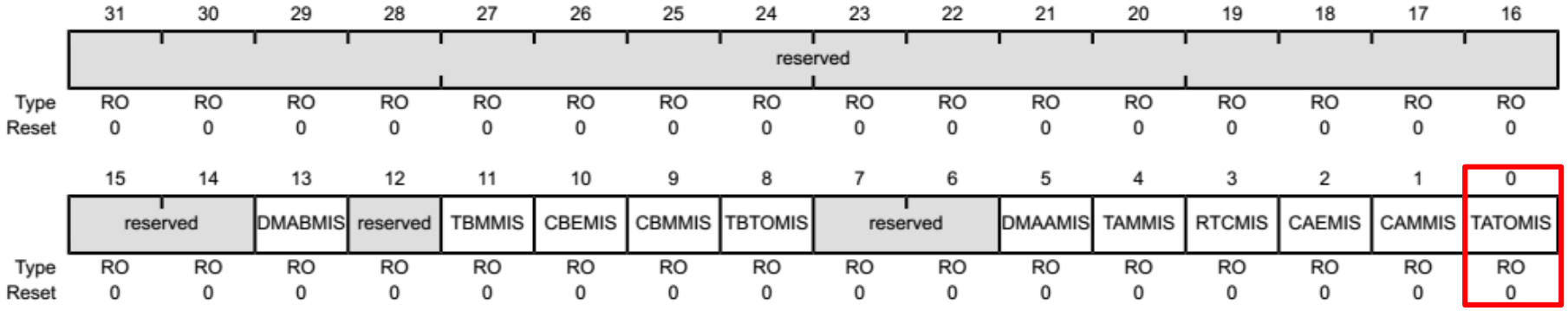
```
void
TimerEnable(uint32_t ui32Base, uint32_t ui32Timer)
{
    // …
    // Enable the timer(s) module.
    HWREG(ui32Base + TIMER_O_CTL) |= ui32Timer & (TIMER_CTL_TAEN |
                                                  TIMER_CTL_TBEN);
}
```

```
TimerEnable(TIMER0_BASE, TIMER_A);
#define TIMER_A     0x000000ff  // Timer A
#define TIMER_B     0x0000ff00  // Timer B
```

➢ 32位单次/周期计数

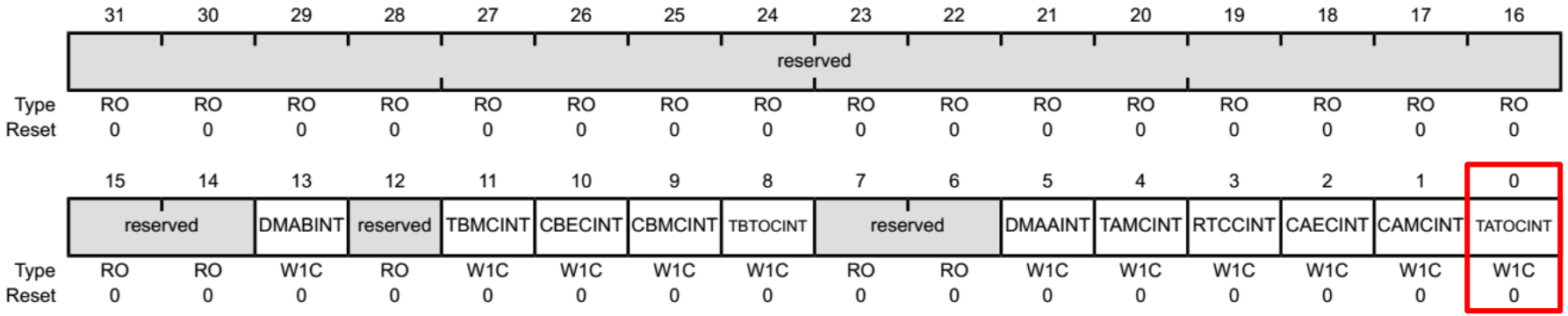
– **6. 编写中断服务函数，清除中断，完成功能**

GPTM Masked Interrupt Status (**GPTMMIS**)寄存器



TATOMIS：TimerA的超时中断的状态

GPTM Interrupt Clear (**GPTMICR**)寄存器



向**TATOCINT**位写1，清除TimerA的超时中断

➢ 32位单次/周期计数

– **6. 编写中断服务函数，清除中断，完成功能**

TivaWare提供了**TimerIntClear**，操作**GPTMICR**寄存器，清除中断：

```
void
TimerIntClear(uint32_t ui32Base, uint32_t ui32IntFlags)
{
    //
    // Check the arguments.
    ASSERT(_TimerBaseValid(ui32Base));
    // Clear the requested interrupt sources.
    //
    HWREG(ui32Base + TIMER_O_ICR) = ui32IntFlags;
}
```

```
TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
```

其中：

```
#define TIMER_TIMA_TIMEOUT      0x00000001
```

32位单次/周期计数

– 6. 编写中断服务函数，清除中断，完成功能

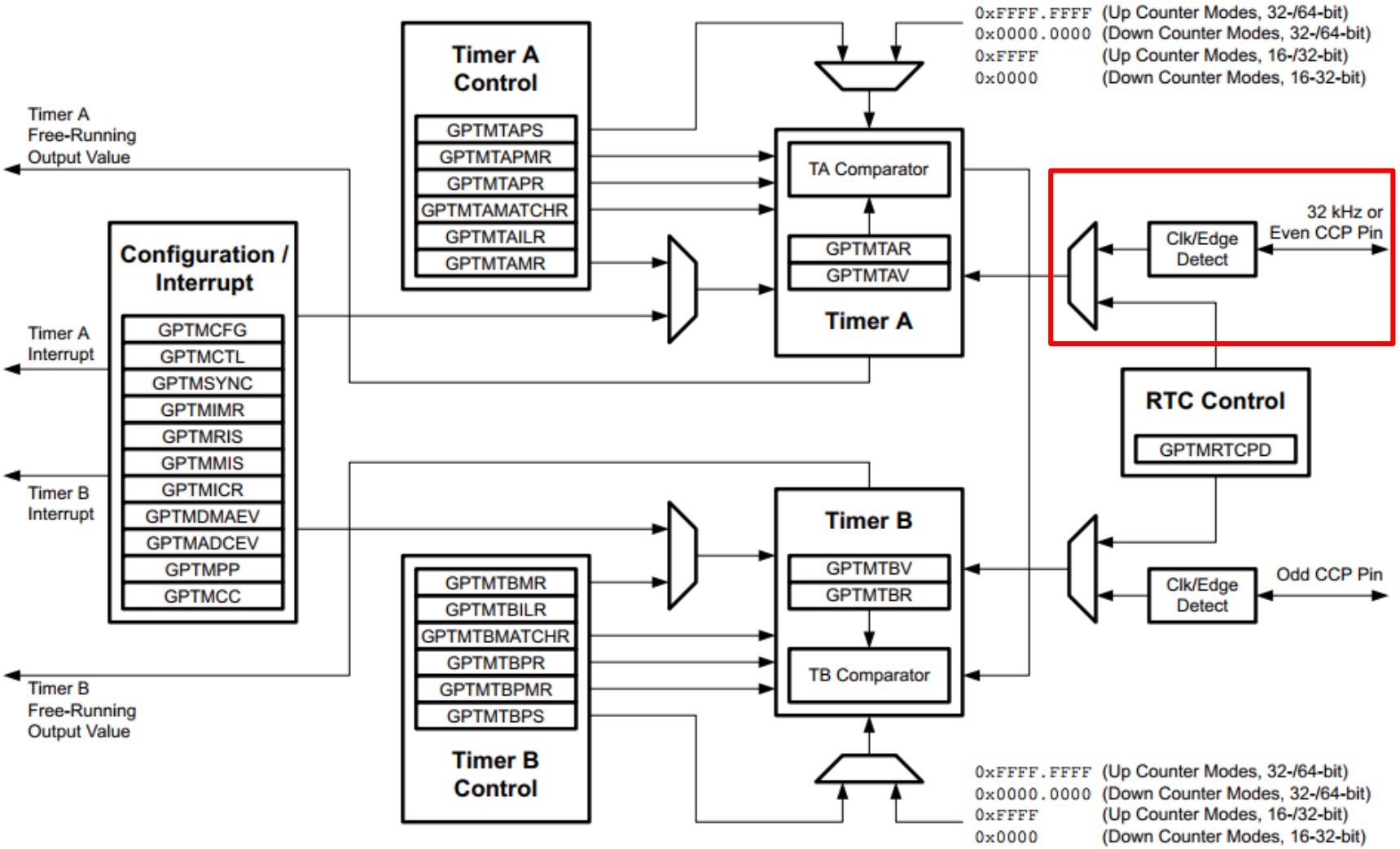
```
void
Timer0AIntHandler(void)
{
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    // …
}
```

➢ 内容概要

- 通用定时器的用途
- TM4C1294的通用定时器模块
- 通用定时器的功能和使用方法：32位单次/周期计数功能
- **通用定时器的功能和使用方法：16位边沿计时功能**

➢ 16位边沿计时

Figure 13-1. GPTM Module Block Diagram



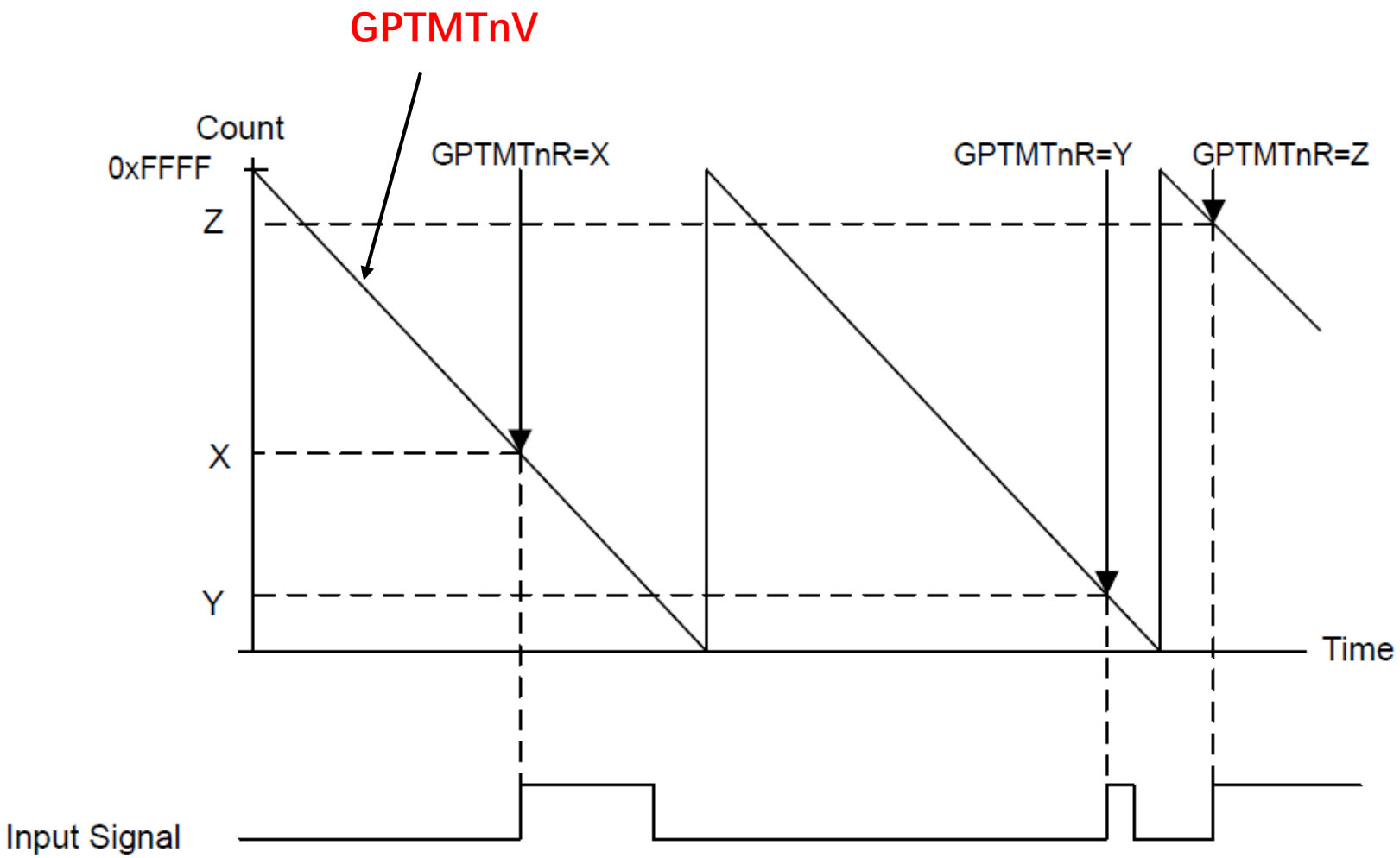
16位边沿计时

- 计数器A或B按16位周期计数模式运行，计数值存储在GPTMTnV寄存器里
- 如果检测到CCP引脚上的边沿（上升沿或下降沿）后，边沿到来时刻的计数值，存储在GPTMTnR寄存器中

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type	Description
T0CCP0	1 33 85	PD0 (3) PA0 (3) PL4 (3)	I/O	TTL	16/32-Bit Timer 0 Capture/Compare/PWM 0.
T0CCP1	2 34 86	PD1 (3) PA1 (3) PL5 (3)	I/O	TTL	16/32-Bit Timer 0 Capture/Compare/PWM 1.
T1CCP0	3 35 94	PD2 (3) PA2 (3) PL6 (3)	I/O	TTL	16/32-Bit Timer 1 Capture/Compare/PWM 0.
T1CCP1	4 36 93	PD3 (3) PA3 (3) PL7 (3)	I/O	TTL	16/32-Bit Timer 1 Capture/Compare/PWM 1.

- 之后GPTMTnR寄存器的值保持不变，GTTMTnV寄存器继续计数
- 边沿到来会触发中断，通知CPU读取GPTMTnR寄存器的值并保存
- 通过计算前后两次GPTMTnR寄存器的值，就可以计算两次边沿之间时间

➢ 16位边沿计时


GPTMTnV
Count
0xFFFF
Z
X
Y
GPTMTnR=X
GPTMTnR=Y
GPTMTnR=Z
Time
Input Signal

➢ 16位边沿计时

- 1. 使能定时器模块的时钟
- 2. 使能输入引脚所在的GPIO模块的时钟
- 3. 打开GPIO引脚的复用功能，并把引脚与定时器模块相连
- 4. 配置定时器模块的运行模式和功能
- 5. 配置定时器模块的计数周期
- 6. 配置定时器模块要感知的引脚上的电平的边沿（上升沿/下降沿）
- 7. 打开并设置中断
- 8. 开始计数
- 9. 编写中断服务函数，清除中断

➢ 16位边沿计时

– 1. 使能定时器模块的时钟

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
```

– 2. 使能输入引脚所在的GPIO模块的时钟

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
```

– 3. 打开GPIO引脚的复用功能，并把引脚与定时器模块相连

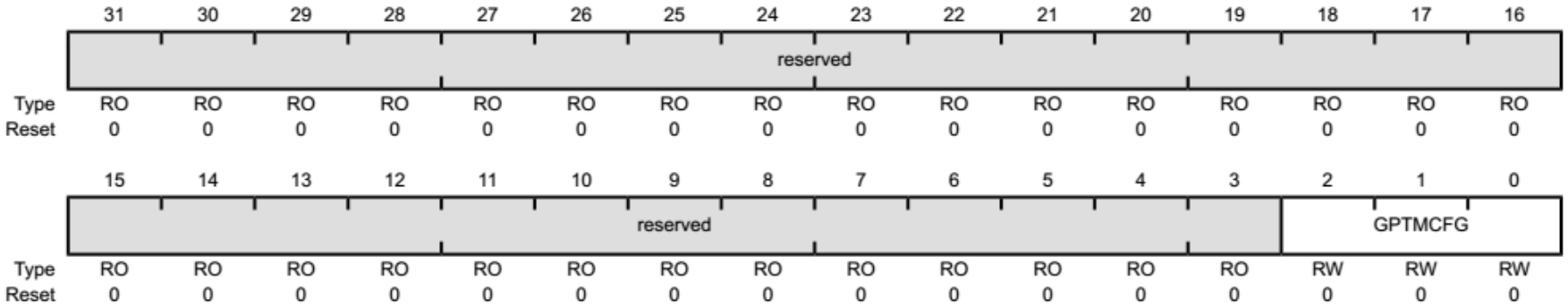
```
GPIOPinTypeTimer(GPIO_PORTD_BASE, GPIO_PIN_2);
GPIOPinConfigure(GPIO_PD2_T1CCP0);
```

```
void
GPIOPinTypeTimer(uint32_t ui32Port, uint8_t ui8Pins)
{
    // …
    // Make the pin(s) be peripheral controlled.
    GPIODirModeSet(ui32Port, ui8Pins, GPIO_DIR_MODE_HW);
    // Set the pad(s) for standard push-pull operation.
    GPIOPadConfigSet(ui32Port, ui8Pins, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD);
}
```

➢ 16位边沿计时

– **4. 配置定时器模块的运行模式和功能**

GPTM Configuration (GPTM**CFG**)寄存器



GPTMCFG域：

0：32位计数，使用TimerA作为32位计数器，对TimerB的配置无效。

1：实时时钟模式

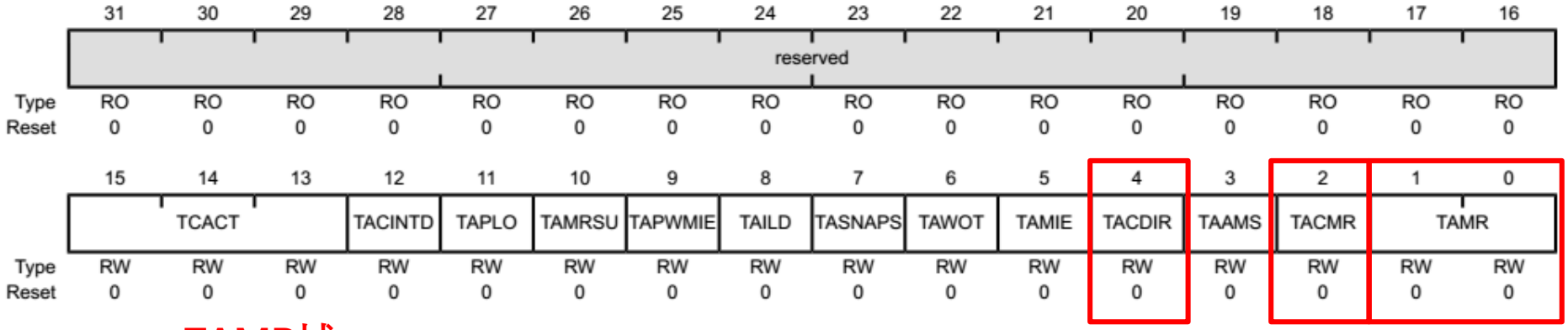
4：16位计数，即TimerA和TimerB分开计数

边沿计时模式**只支持**单独使用TimerA或TimerB，因此GPTMCFG要设置位**0x04**。

➢ 16位边沿计时

– **4. 配置定时器模块的运行模式和功能**

GPTM Timer A Mode (GPTM**TAMR**)寄存器



TAMR域：

1：单次计数

2：周期计数

3：捕获模式

TACDIR域：

0：向下计数

1：从0开始向上计数

TACMR域：

0：边沿计数模式

1：边沿计时模式

使用**边沿计时**模式：

- GPTMTAMR 寄存器的**TAMR**域设置位捕获模式（**0x03**）
- GPTMTAMR 寄存器的**TACMR**域设置位边沿计时模式（**0x01**）

➢ 16位边沿计时

– **4. 配置定时器模块的运行模式和功能**

```
TimerConfigure(TIMER1_BASE,
TIMER_CFG_SPLIT_PAIR|TIMER_CFG_A_CAP_TIME);
```

```
#define TIMER_CFG_SPLIT_PAIR      0x04000000
```

GPTM**CFG**

4：16位计数，即TimerA和TimerB分开计数

```
#define TIMER_CFG_A_CAP_TIME      0x00000007
#define TIMER_CFG_A_CAP_TIME_UP   0x00000017
```

TACDIR域：1为向上计数

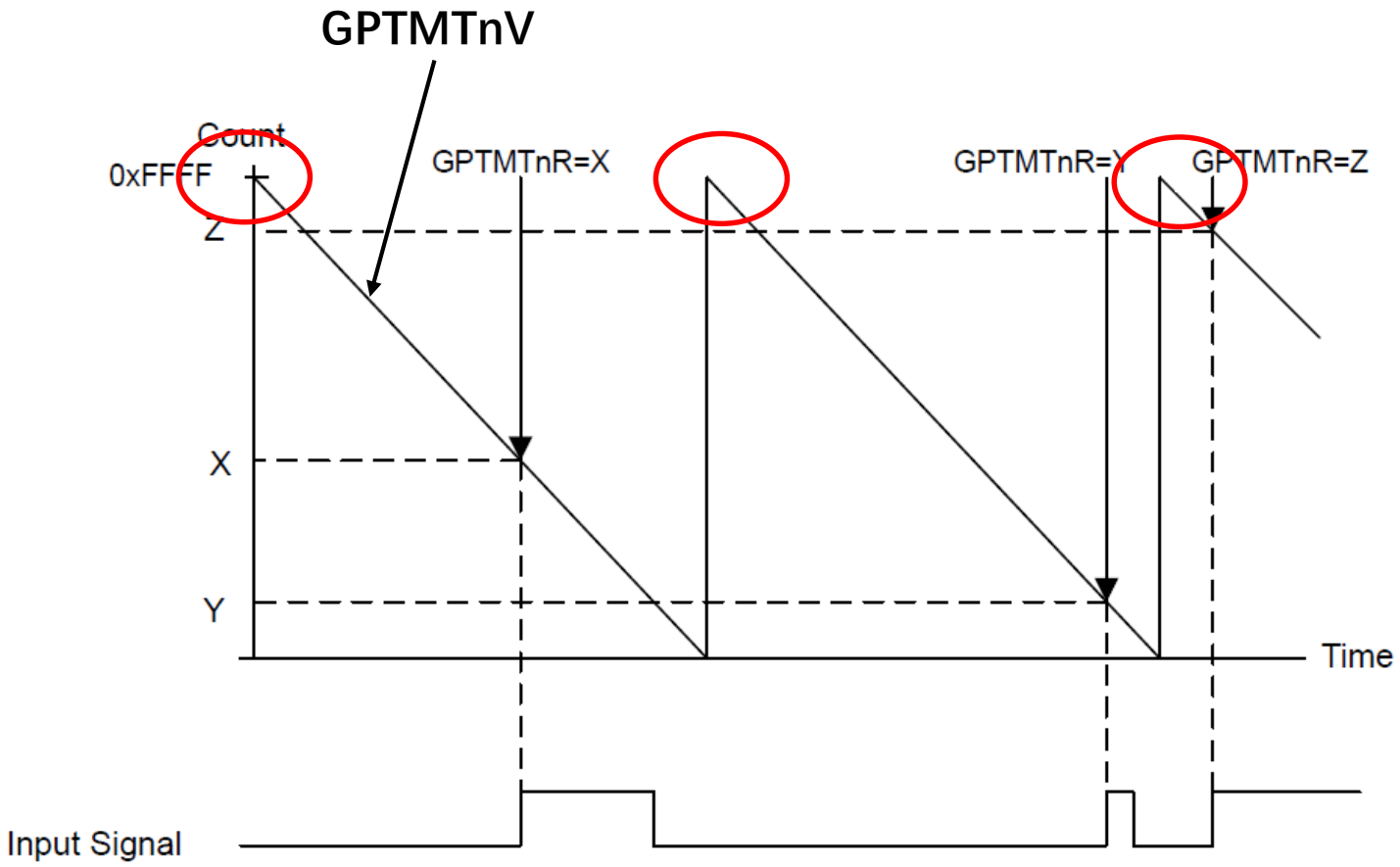
- TAMR域: **0x03**
- TACMR域: **0x01**

```
void
TimerConfigure(uint32_t ui32Base, uint32_t ui32Config)
{
    // ...
    // Disable the timers.
    HWREG(ui32Base + TIMER_O_CTL) &= ~(TIMER_CTL_TAEN | TIMER_CTL_TBEN);
    // Set the global timer configuration.
    HWREG(ui32Base + TIMER_O_CFG) = ui32Config >> 24;
    // Set the configuration of the A and B timers and set the TxPWMIE bit.
    // Note that the B timer configuration is ignored by the hardware in 32-bit modes.
    if(NEW_TIMER_CONFIGURATION)
    {
        HWREG(ui32Base + TIMER_O_TAMR) = (((ui32Config & 0x000f0000) >> 4) |
                                          (ui32Config & 0xff) |
                                          TIMER_TAMR_TAPWMIE);
        HWREG(ui32Base + TIMER_O_TBMR) = (((ui32Config & 0x00f00000) >> 8) |
                                          ((ui32Config >> 8) & 0xff) |
                                          TIMER_TBMR_TBPWMIE);
    }
    else
    {
        HWREG(ui32Base + TIMER_O_TAMR) = ((ui32Config & 0xff) |
                                          TIMER_TAMR_TAPWMIE);
        HWREG(ui32Base + TIMER_O_TBMR) = (((ui32Config >> 8) & 0xff) |
                                          TIMER_TBMR_TBPWMIE);
    }
}
```

➢ 16位边沿计时

– **5. 配置定时器模块的计数周期**

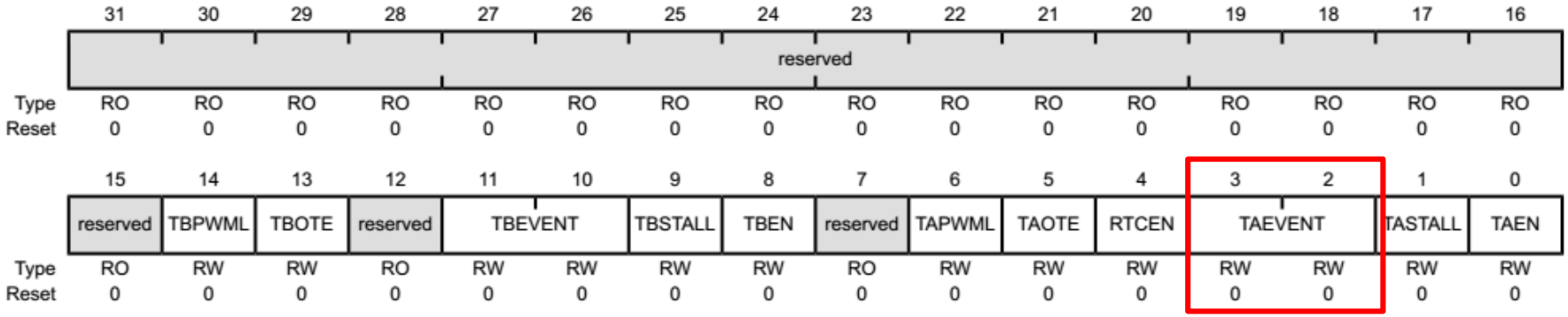
```
TimerLoadSet(TIMER1_BASE, TIMER_A, 0xffff);
```



➢ 16位边沿计时

– **6. 配置定时器模块要感知的引脚上的电平的边沿（上升沿/下降沿）**

GPTM Control (**GPTMCTL**)寄存器:



TAEN：计数器A使能控制

TBEN：计数器B使能控制

TAEVENT：边沿计数或者计时要感知的边沿

0：上升沿，1：下降沿，3：上升沿和下降沿

➢ 16位边沿计时

– **6. 配置定时器模块要感知的引脚上的电平的边沿（上升沿/下降沿）**

TivaWare提供**TimerControlEvent**函数。设置要感知的边沿，其定义为：

```
void
TimerControlEvent(uint32_t ui32Base, uint32_t ui32Timer,
                  uint32_t ui32Event)
{
    // …
    // Set the event type.
    ui32Timer &= TIMER_CTL_TAEVENT_M | TIMER_CTL_TBEVENT_M;
    HWREG(ui32Base + TIMER_O_CTL) = ((HWREG(ui32Base + TIMER_O_CTL) &
                                      ~ui32Timer) | (ui32Event &
ui32Timer));
}
```

```
#define TIMER_CTL_TAEVENT_M     0x0000000C  // GPTM Timer A Event Mode
#define TIMER_CTL_TBEVENT_M     0x00000C00  // GPTM Timer B Event Mode
```

```
#define TIMER_EVENT_POS_EDGE    0x00000000  // Count positive edges
#define TIMER_EVENT_NEG_EDGE    0x00000404  // Count negative edges
#define TIMER_EVENT_BOTH_EDGES  0x00000C0C  // Count both edges
```

例：

```
TimerControlEvent(TIMER1_BASE, TIMER_A, TIMER_EVENT_BOTH_EDGES);
```

➢ 16位边沿计时

– 7. 打开并设置中断

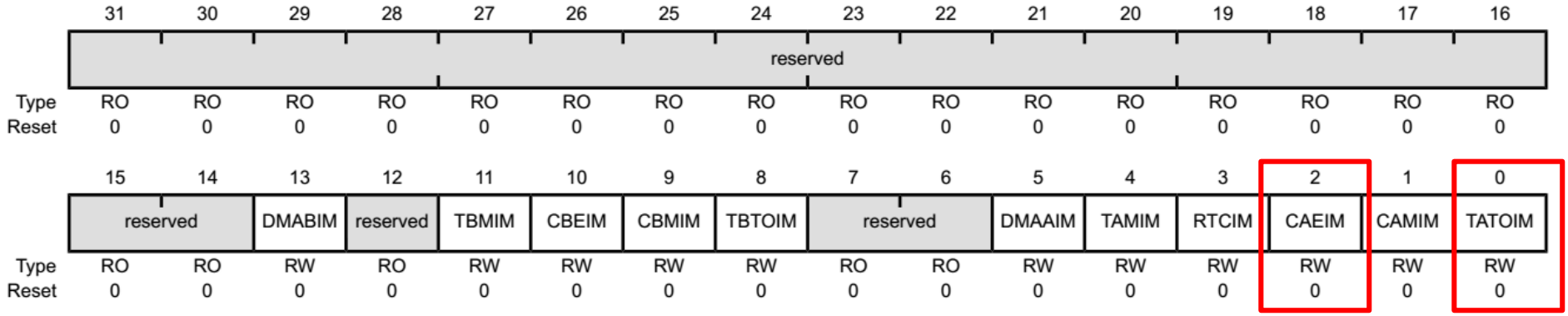
```
IntMasterEnable();

IntEnable(INT_TIMER1A);
```

➢ 16位边沿计时

– **7. 打开并设置中断**

GPTM Interrupt Mask (GPTM**IMR**)寄存器



TATOIM：TimerA的超时中断请求使能，写1使能

CAEIM：捕获事件中断使能，写1使能

使用TimerIntEnable函数使能超时中断和捕获中断：

```
TimerIntEnable(TIMER1_BASE, TIMER_CAPA_EVENT|TIMER_TIMA_TIMEOUT);

#define TIMER_CAPA_EVENT          0x00000004  // CaptureA event interrupt
#define TIMER_TIMA_TIMEOUT        0x00000001  // TimerA time out interrupt
```

16位边沿计时

- **7. 打开并设置中断**
- 注册中断服务函数

```
TimerIntRegister(TIMER1_BASE,TIMER_A,Timer1IntHandler);
```

➢ 16位边沿计时

– **8.** 开始计数

```
TimerEnable(TIMER1_BASE, TIMER_A);
```

```
void
TimerEnable(uint32_t ui32Base, uint32_t ui32Timer)
{
    // …
    // Enable the timer(s) module.
    HWREG(ui32Base + TIMER_O_CTL) |= ui32Timer & (TIMER_CTL_TAEN |
                                                  TIMER_CTL_TBEN);
}
```

```
#define TIMER_A                 0x000000ff  // Timer A
#define TIMER_B                 0x0000ff00  // Timer B
```

➢ 16位边沿计时

– **9. 编写中断服务函数，清除中断，完成功能**

同时使能了超时中断和捕获中断，可以通过TimerIntStatus查看中断的来源，该函数的定义为：

```
uint32_t
TimerIntStatus(uint32_t ui32Base, bool bMasked)
{
    //
    // Check the arguments.
    //
    ASSERT(_TimerBaseValid(ui32Base));

    //
    // Return either the interrupt status or the raw interrupt status as
    // requested.
    //
    return(bMasked ? HWREG(ui32Base + TIMER_O_MIS) :
           HWREG(ui32Base + TIMER_O_RIS));
}
```

➢ 16位边沿计时

– **9. 编写中断服务函数，清除中断，完成功能**

同时使能了超时中断和捕获中断，可以通过TimerIntStatus查看中断的来源

```
TimerIntStatus(TIMER1_BASE,true)
```

边沿到来时刻的计数值存放在GPTM**TAR**寄存器中，使用TimerValueGet函数获取，其定义为：

```
uint32_t
TimerValueGet(uint32_t ui32Base, uint32_t ui32Timer)
{
    //
    // Check the arguments.
    ASSERT(_TimerBaseValid(ui32Base));
    ASSERT((ui32Timer == TIMER_A) || (ui32Timer == TIMER_B));
    //
    // Return the appropriate timer value.
    return((ui32Timer == TIMER_A) ? HWREG(ui32Base + TIMER_O_TAR) :
           HWREG(ui32Base + TIMER_O_TBR));
}
```

```
C0=TimerValueGet(TIMER1_BASE,TIMER_A);
```

➢ 16位边沿计时

– **9. 编写中断服务函数，清除中断，完成功能**

同时使能了超时中断和捕获中断，可以通过TimerIntStatus查看中断的来源

边沿到来时刻的计数值存放在GPTM**TAR**寄存器中，使用TimerValueGet函数获取

```
void
Timer1IntHandler(void)
{
        if(TimerIntStatus(TIMER1_BASE,true)&TIMER_TIMA_TIMEOUT){
                TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
        // …
        }
        if(TimerIntStatus(TIMER1_BASE,true)&TIMER_CAPA_EVENT){

                TimerIntClear(TIMER1_BASE, TIMER_CAPA_EVENT);
               C0=C1;
               C1=TimerValueGet(TIMER1_BASE,TIMER_A);
      // …
        }
}
```

➢ 思考

– 如果两次边沿之间有多个计数周期，如何处理？

➢ 对照数据手册自学

– 利用预分频寄存器扩展为24位边沿计时

– 比较器功能

– 边沿计数

– 实时时钟

– PWM功能

谢谢！