# 看门狗定时器
# (Watchdog Timer)

**程晨闻**

**东南大学电气工程学院**

# 问题

- GPIOA对应的异常编号为16，那么系统重启后，16号异常向量存储在什么位置（具体地址）？存储这个中断向量占用多少存储空间?
- 调用GPIOIntRegister(GPIO_PORTA__BASE,IntGPIOa); 注册中断服务函数后，16号异常向量存储在什么位置（具体地址）？

**Figure 2-6. Vector Table**

| Exception number | IRQ number | Offset | Vector |
|---|---|---|---|
| (N+16) | (N) | 0x040 + 0x(N*4) | IRQ N |
| . . . | . . . | . . . 0x004C | . . . |
| 18 | 2 | 0x0048 | IRQ2 |
| 17 | 1 | 0x0044 | IRQ1 |
| 16 | 0 | 0x0040 | IRQ0 |
| 15 | -1 | 0x003C | Systick |
| 14 | -2 | 0x0038 | PendSV |
| 13 | | | Reserved |
| 12 | | | Reserved for Debug |
| 11 | -5 | 0x002C | SVCall |
| 10 9 8 7 | | | Reserved |
| 6 | -10 | 0x0018 | Usage fault |
| 5 | -11 | 0x0014 | Bus fault |
| 4 | -12 | 0x0010 | Memory management fault |
| 3 | -13 | 0x000C | Hard fault |
| 2 | -14 | 0x0008 | NMI |
| 1 | | 0x0004 | Reset |
| 0 | | 0x0000 | Initial SP value |

Vector Table Offset (VTABLE)

Base 0xE000.E000
Offset 0xD08
Type RW, reset 0x0000.0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET | | | | | | | | | | | | | | | |
| Type RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFFSET | | | | | | reserved | | | | | | | | | |
| Type RW | RW | RW | RW | RW | RW | 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 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | OFFSET | RW | 0x000.00 | Vector Table Offset<br>When configuring the OFFSET field, the offset must be aligned to the number of exception entries in the vector table. Because there are 112 interrupts, the offset must be aligned on a 1024-byte boundary. |
| 9:0 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

- 原本中断向量表位于0x0000 0000（FLASH）
- VTABLE为0x2000 0000,则向量表就位于从0x2000 0000起始的空间中（RAM）

## 系统节拍定时器原理

## 系统节拍定时器的操作方法

- 节拍控制和状态寄存器STCTRL：配置时钟、使能、使能中断、查看状态
- 节拍计数初值寄存器STRELOAD: 计数器从这个值开始计数，向下计数，计到零后，又重新从这个值开始计数
- 节拍当前值寄存器STCURRENT：计数器的当前值
- SysTickPeriodSet()
- SysTickEnable()， SysTickDisable ()
- SysTickIntEnable()， SysTickIntDisable()
- SysTickIntRegister()
- SysTickValueGet()

- 看门狗计数器的工作原理
- 看门狗计数器的使用

## ➢ 作用

– 用于在系统产生错误或者程序出现故障、失去响应时，使系统复位

## ➢ 工作原理：“定时喂狗，狗饿复位”

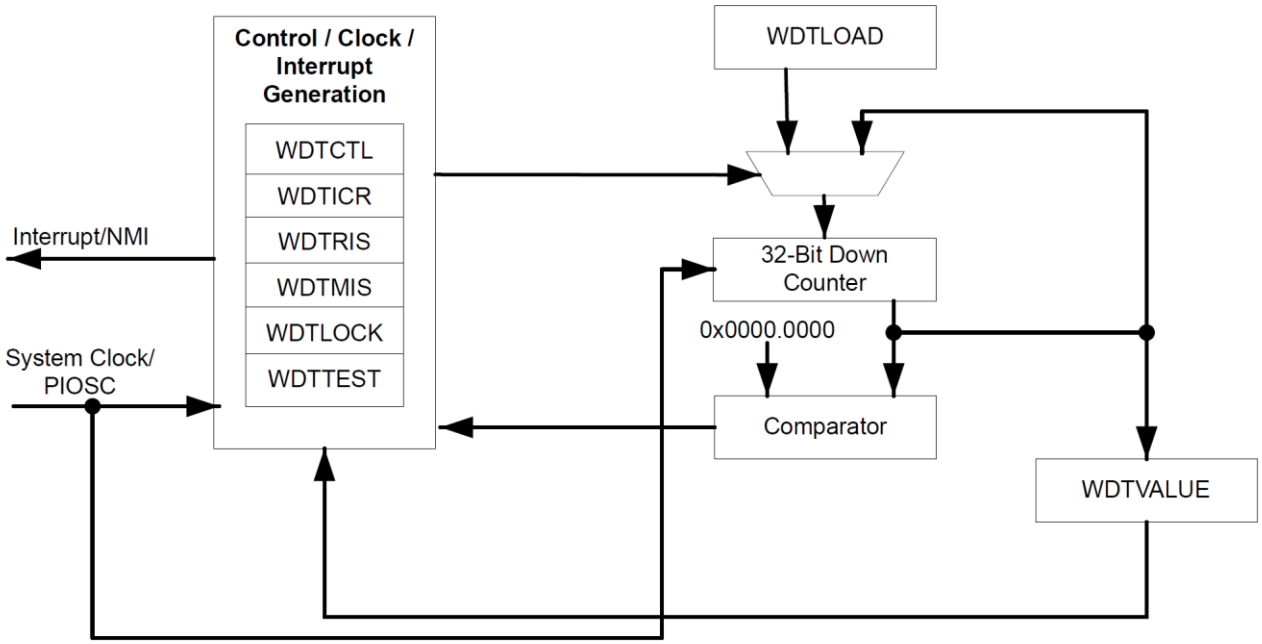
– 看门狗计数器设置一定的计时时间

– 看门狗使能后，计数器开始向下计数

– 当计时时间到后，则触发系统复位

– 如果在定时时间到达之前，进行喂狗（计数器重装）动作，就不会引起系统复位

# ➢ 工作原理:“定时喂狗，狗饿复位”




- 使能后，计数器从WDTLOAD开始向下计数。
- 当看门狗定时器第一次递减到0时，产生一个超时信号，触发中断。
- 如果第一次超时的中断状态没有被清除，计数器第二次递减到零后，系统复位。

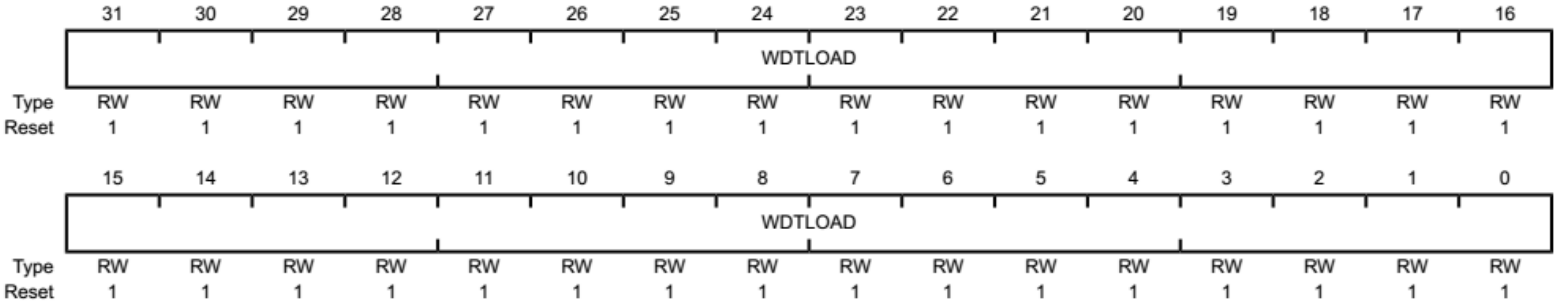
## ➢ TM4C控制器看门狗模块的特点

- 32位递减计数器，计数周期可编程
- 可设置不同的时钟源，带使能控制（两个WDT）
- 可编程中断
- 内部寄存器保护
- 可使能/禁止产生复位信号
- 调试时可以暂停

# ➢ 计数初值寄存器WDTLOAD

Watchdog Load (WDTLOAD)

WDT0 base: 0x4000.0000
WDT1 base: 0x4000.1000
Offset 0x000
Type RW, reset 0xFFFF.FFFF



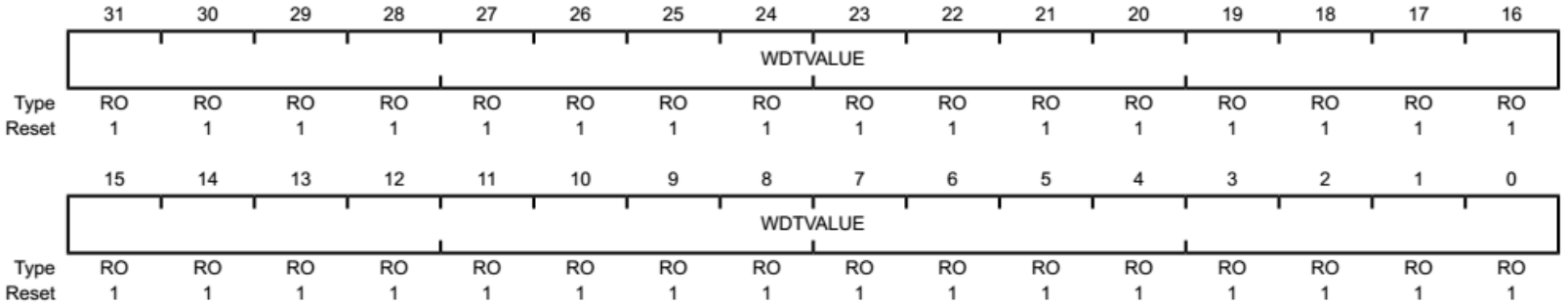

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTLOAD | RW | 0xFFFF.FFFF | Watchdog Load Value |

32位计数初值。

# ➢ 当前计数值寄存器 WDTVALUE

Watchdog Value (WDTVALUE)

WDT0 base: 0x4000.0000
WDT1 base: 0x4000.1000
Offset 0x004
Type RO, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTVALUE | RO | 0xFFFF.FFFF | Watchdog Value<br>Current value of the 32-bit down counter. |

32位当前计数值

# ➢ 看门狗控制寄存器 WDTCTL

Watchdog Control (WDTCTL)

WDT0 base: 0x4000.0000
WDT1 base: 0x4000.1000
Offset 0x008
Type RW, reset 0x0000.0000 (WDT0) and 0x8000.0000 (WDT1)

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | WRC | reserved | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 1 | 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 | | | | | | | | | | | | | INTTYPE | RESEN | INTEN |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- INTTYPE：看门狗产生的中断类型
  - 0：普通可屏蔽中断；1：非可屏蔽中断NMI
- RESEN：重置使能
  - 0：看门狗不能重启系统；1：看门狗可以重启系统。写1后，开门狗定时器启动。
- INTEN：中断使能
  - 0：关闭中断；1：启动中断，一旦写1，除非重启系统，该位的值都不会再改变

# ➢ 中断清除寄存器 WDTICR

Watchdog Interrupt Clear (WDTICR)

WDT0 base: 0x4000.0000
WDT1 base: 0x4000.1000
Offset 0x00C
Type WO, reset -

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | WDTINTCLR | | | | | | | | | | | | | | | |
| Type | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO |
| Reset | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | WDTINTCLR | | | | | | | | | | | | | | | |
| Type | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO |
| Reset | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

– 向该寄存器写任意数值，都可清除看门狗中断，并且重置看门狗计数值为WDTLOAD

# ➢ 看门狗锁定寄存器 WDTLOCK

Watchdog Lock (WDTLOCK)

WDT0 base: 0x4000.0000
WDT1 base: 0x4000.1000
Offset 0xC00
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | WDTLOCK | | | | | | | | | | | | | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | WDTLOCK | | | | | | | | | | | | | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

– 向该寄存器中写入0x1ACC E551，可以解锁看门狗寄存器组，使其可编辑
– 写入其他数值，锁定看门狗寄存器组，是其不可编辑
– 如果看门狗寄存器组已经锁定，读取该寄存器返回1。如果未锁定，返回0

## ➢ 看门狗定时器的使用

- 1. 在系统控制模块中开启看门狗模块的时钟
- 2. 解锁看门狗寄存器组
- 3. 设置看门狗计数器的计数初值
- 4. 注册中断服务函数
- 5. 使能看门狗模块的中断和重启功能
- 6. 锁定看门狗寄存器组
- 7. 在无限循环中喂狗，确保程序没有死机
- 8. 编写中断服务函数，在程序没有及时喂狗的情况下，做特殊处理

# ➢ 看门狗定时器的使用

## – 1. 在系统控制模块中开启看门狗模块的时钟

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_WDOG0);
```

其中#define SYSCTL_PERIPH_WDOG0  0xf0000000  // Watchdog 0

SysCtlPeripheralEnable函数定义如下

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

```
#define SYSCTL_RCGCBASE          0x400fe600
```

Watchdog Timer Run Mode Clock Gating Control (RCGCWD)

Base 0x400F.E000
Offset 0x600
Type RW, reset 0x0000.0000

| 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 | | | | | | | | | | | | | | R1 | R0 |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# ➢ 看门狗定时器的使用

## – 2. 解锁看门狗寄存器组

```
if(WatchdogLockState(WATCHDOG0_BASE) == true){
    WatchdogUnlock(WATCHDOG0_BASE);
}
```

```
bool
WatchdogLockState(uint32_t ui32Base)
{
    …
    // Get the lock state.
return((HWREG(ui32Base + WDT_O_LOCK) == WDT_LOCK_LOCKED) ? true : false);
}
```

其中：#define WDT_O_LOCK 0x00000C00 // Watchdog Lock
#define WDT_LOCK_UNLOCKED 0x00000000 // Unlocked
#define WDT_LOCK_LOCKED 0x00000001 // Locked

```
void
WatchdogUnlock(uint32_t ui32Base)
{
    …
    // Unlock watchdog register writes.
    HWREG(ui32Base + WDT_O_LOCK) = WDT_LOCK_UNLOCK;
}
```

其中 #define WDT_LOCK_UNLOCK 0x1ACCE551 // Unlocks the watchdog timer

# ➢ 看门狗锁定寄存器 WDTLOCK

Watchdog Lock (WDTLOCK)

WDT0 base: 0x4000.0000
WDT1 base: 0x4000.1000
Offset 0xC00
Type RW, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | WDTLOCK | | | | | | | | | | | | | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | WDTLOCK | | | | | | | | | | | | | | | |
| Type | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

– 向该寄存器中写入0x1ACC E551，可以解锁看门狗寄存器组，使其可编辑。
– 写入其他数值，锁定看门狗寄存器，是其不可编辑。
– 如果看门狗寄存器组已经锁定，读取该寄存器返回1。如果未锁定，返回0。

# ➢ 看门狗定时器的使用

## – 3. 设置看门狗计数器的计数初值

```
WatchdogReloadSet(WATCHDOG0_BASE, ui32SysClock*5);
```

```
void
WatchdogReloadSet(uint32_t ui32Base, uint32_t ui32LoadVal)
{
    //
    // Check the arguments.
    //
    ASSERT((ui32Base == WATCHDOG0_BASE) || (ui32Base == WATCHDOG1_BASE));

    //
    // Set the load register.
    //
    HWREG(ui32Base + WDT_O_LOAD) = ui32LoadVal;
}
```

WatchdogReloadSet函数通过写WDTLOAD寄存器，设置初始值。

以上代码中，看门狗的超时时间是多长?

# ➢ 看门狗定时器的使用

## – 3. 读取看门狗计数器的计数初值

```
WatchdogReloadGet(WATCHDOG0_BASE);
```

```
uint32_t
WatchdogReloadGet(uint32_t ui32Base)
{
    //
    // Check the arguments.
    //
    ASSERT((ui32Base == WATCHDOG0_BASE) || (ui32Base == WATCHDOG1_BASE));

    //
    // Get the load register.
    //
    return(HWREG(ui32Base + WDT_O_LOAD));
}
```

# ➢ 看门狗定时器的使用

## – 4. 注册中断服务函数

```
WatchdogIntRegister(WATCHDOG0_BASE,WatchdogIntHandler);
```

```
void
WatchdogIntRegister(uint32_t ui32Base, void (*pfnHandler)(void))
{
    //
    // Check the arguments.
    //
    ASSERT((ui32Base == WATCHDOG0_BASE) || (ui32Base == WATCHDOG1_BASE));

    //
    // Register the interrupt handler.
    //
    IntRegister(INT_WATCHDOG_TM4C123, pfnHandler);

    //
    // Enable the watchdog timer interrupt.
    //
    IntEnable(INT_WATCHDOG_TM4C123);
}
```

```
#define INT_WATCHDOG_TM4C123    34          // Watchdog Timers 0 and 1
```

# ➢ 看门狗定时器的使用

## – 5. 使能看门狗模块的中断和重启功能

```
WatchdogIntEnable(WATCHDOG0_BASE);
WatchdogResetEnable(WATCHDOG0_BASE);
```

```
void
WatchdogIntEnable(uint32_t ui32Base)
{
    ...
    // Enable the watchdog interrupt.
    HWREG(ui32Base + WDT_O_CTL) |= WDT_CTL_INTEN;
}
```

```
void
WatchdogResetEnable(uint32_t ui32Base)
{
    …
    // Enable the watchdog reset.
    HWREG(ui32Base + WDT_O_CTL) |= WDT_CTL_RESEN;
}
```

- 这两个函数操作WDCTRL寄存器的INTEN和RESEN两个位
- 这两个位中的任意一个被写1后，看门狗都会自动开始运行

```
HWREG(WATCHDOG0_BASE + WDT_O_CTL) |= (WDT_CTL_INTEN | WDT_CTL_RESEN);
```

或如果直接使用寄存器操作，可以用一句代码完成上述两个功能，请写出代码

```
HWREG(WATCHDOG0_BASE + WDT_O_CTL) |= 0x03;
```

# ➢ 看门狗控制寄存器 WDTCTL

Watchdog Control (WDTCTL)

WDT0 base: 0x4000.0000
WDT1 base: 0x4000.1000
Offset 0x008
Type RW, reset 0x0000.0000 (WDT0) and 0x8000.0000 (WDT1)

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | WRC | reserved | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 1 | 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 | | | | | | | | | | | | | INTTYPE | RESEN | INTEN |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RW | RW | RW |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- INTTYPE：看门狗产生的中断类型。
  - 0：普通可屏蔽中断。1：非可屏蔽中断NMI
- RESEN：重置使能。
  - 0：看门狗不能重启系统。1：看门口可以重启系统。写1后，开门狗定时器启动。
- INTEN：中断使能。
  - 0：关闭中断。1：启动中断，一旦写1，除非重启系统，该位的值都不会再改变

## ➢ WDTTEST

– 调试器暂停CPU的时候，看门狗还会继续工作，这会导致系统意外重启。

– WDTTEST寄存器可以在这种情况下，暂停看门狗的计数。

This register provides user-enabled stalling when the microcontroller asserts the CPU halt flag during debug.

Watchdog Test (WDTTEST)

WDT0 base: 0x4000.0000
WDT1 base: 0x4000.1000
Offset 0x418
Type RW, reset 0x0000.0000

| 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 | | | | | | | STALL | reserved | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RW | 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 |

```
WatchdogStallEnable(WATCHDOG0_BASE);
```

```
void
WatchdogStallEnable(uint32_t ui32Base)
{
    …
    // Enable timer stalling.
    HWREG(ui32Base + WDT_O_TEST) |= WDT_TEST_STALL;
}
```

# ➢ 看门狗定时器的使用

## – 6. 锁定看门狗寄存器组

- WatchdogLock将看门狗寄存器锁定，不能随意更改

```
WatchdogLock(WATCHDOG0_BASE);
```

```
void
WatchdogLock(uint32_t ui32Base)
{
    …
    // Lock out watchdog register writes.  Writing anything to the WDT_O_LOCK
    // register causes the lock to go into effect.
    HWREG(ui32Base + WDT_O_LOCK) = WDT_LOCK_LOCKED;
}
```

```
#define WDT_LOCK_LOCKED         0x00000001  // Locked
```

# ➢ 看门狗定时器的使用

## – 7. 在无限循环中喂狗，确保程序没有死机

```
while(1)
    {
    WatchdogIntClear(WATCHDOG0_BASE);

    //do something
    ……
    }
```

```
void
WatchdogIntClear(uint32_t ui32Base)
{
    …
    // Clear the interrupt source.
    HWREG(ui32Base + WDT_O_ICR) = WDT_RIS_WDTRIS;
}
```

# ➢ 中断清除寄存器 WDTICR

Watchdog Interrupt Clear (WDTICR)

WDT0 base: 0x4000.0000
WDT1 base: 0x4000.1000
Offset 0x00C
Type WO, reset -

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | WDTINTCLR | | | | | | | | | | | | | | | |
| Type | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO |
| Reset | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | WDTINTCLR | | | | | | | | | | | | | | | |
| Type | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO | WO |
| Reset | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

– 向该寄存器写任意数值，都可清除看门狗中断，并且重置看门狗计数值为WDTLOAD

# 看门狗定时器的使用

– 8. 编写中断服务函数，在程序没有及时喂狗的情况下，做特殊处理

```
void WatchdogIntHandler(void)
{
    WatchdogIntClear(WATCHDOG0_BASE);

}
```

- #include "driverlib/watchdog.h"
- ….
- SysCtlPeripheralEnable(SYSCTL_PERIPH_WDOG0);
- if(WatchdogLockState(WATCHDOG0_BASE) == true){
- WatchdogUnlock(WATCHDOG0_BASE);}
- WatchdogReloadSet(WATCHDOG0_BASE, ui32SysClock*10);
- WatchdogResetEnable(WATCHDOG0_BASE);
- WatchdogIntRegister(WATCHDOG0_BASE, WatchdogIntHandler);
- WatchdogStallEnable(WATCHDOG0_BASE);
- WatchdogEnable(WATCHDOG0_BASE);
- WatchdogLock(WATCHDOG0_BASE);
- ……

Void WatchdogIntHandler(void)

{WatchdogIntClear(WATCHDOG0_BASE); }

## ➢ 练习

- 利用看门狗计数器，计算主循环函数的运行时间，以us为单位，填入float类型变量 TimePassed 中
- 假设主循环中只有一个简单的延时函数 SysCtlDelay(ui32SysClock/3);

## 答案

定义全局变量

```
uint32_t WDTCount;
float TimePassed;
```

初始化：

```
uint32_t ui32SysClock;

 ui32SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
        SYSCTL_OSC_MAIN |
        SYSCTL_USE_PLL |
        SYSCTL_CFG_VCO_480), 120000000);


SysCtlPeripheralEnable(SYSCTL_PERIPH_WDOG0);
if(WatchdogLockState(WATCHDOG0_BASE) == true){
    WatchdogUnlock(WATCHDOG0_BASE);
}
WatchdogReloadSet(WATCHDOG0_BASE, ui32SysClock*3);
WatchdogIntRegister(WATCHDOG0_BASE,WatchdogIntHandler);
WatchdogIntEnable(WATCHDOG0_BASE);
WatchdogResetEnable(WATCHDOG0_BASE);
WatchdogStallEnable(WATCHDOG0_BASE);
WatchdogEnable(WATCHDOG0_BASE);

WatchdogLock(WATCHDOG0_BASE);
```

## 答案

主循环：

```
while(1)
 {
    WatchdogIntClear(WATCHDOG0_BASE);

    SysCtlDelay(ui32SysClock/3);

    WDTCount=WatchdogReloadGet(WATCHDOG0_BASE)-WatchdogValueGet(WATCHDOG0_BASE);
    TimePassed=((float)WDTCount)/120.0;

}
```

## 小节

- 系统节拍定时器原理
- 系统节拍定时器的操作
- 看门狗计数器的工作原理
- 看门狗计数器的使用

## 实验任务

– 用PN0和PN1驱动开发板上的两个LED灯，PN0控制D2，PN1控制D1。PJ0和PJ1接两个简单按键，USR_SW1和USR_SW2。

– 基本要求：

• 每按一次USR_SW1，D2快闪三次（间隔0.33s）。每按一次USR_SW2，D1慢闪三次（间隔2s）。（60分）

# 谢谢！