一起学习 CC3200 系列教程之 WatchDog

阿汤哥

序：

能力有限，难免有错，有问题请联系我，

QQ1519256298 hytga@163.com

Pdf 下载 http://pan.baidu.com/s/1hqiWB56

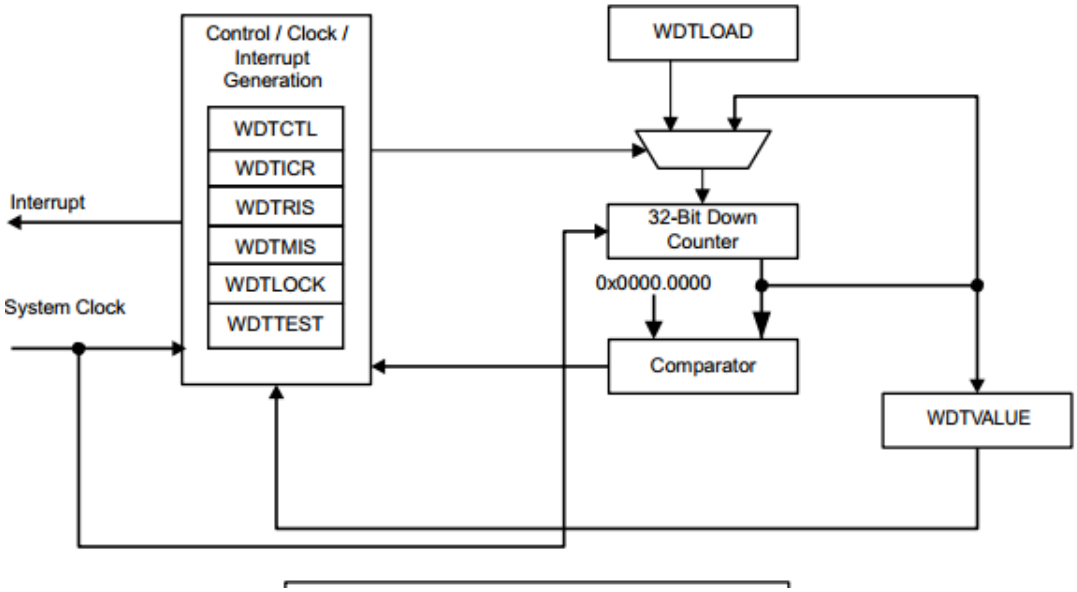
看门狗：

特征：位宽：32bit 向下计数器

时钟：80Mhz（RUN 模式）

锁保护：手册是写保护看门狗寄存器（避免被意外更改），清除中断状态寄存器并不在保护范围内，

系统框图



我把看门狗分成两个过程，第一过程是中断产生过程，第二过程是复位过程，

第一过程：

1、计数器=重装值 2、计数器递减，3、当计数器==0，触发了中断，并且计数器=重装值

第二过程：

4、计数器递减，5、当计数器==0，触发了系统复位。

注意点：

1、 中断函数是可以没有的，那么就不会导致了中断的触发

2、 清除中断状态标志都会导致计数器=重装值，并且可以从中断函数跳出来，不然一旦开启了中断，却不清除中断标志程序会死在中断函数中。

3、 在程序的前面可以用**PRCMSysResetCauseGet()**;这个函数来判断是上电开机，还是看门狗复位开机。

4、 附上 datasheet 一张图，

10.4.1 System WatchDog

Behavior:

- The system WDOG timer expiry forces the MCU and network processor through a reset cycle, but the WLAN domain (MAC and Baseband) is not reset.
- Subsequent recovery takes MCU and NWP out of reset at the same time.

Issue:

- The above behavior does not allow a WLAN domain clean reset.
- The recovery flow order is not congruent to the software flow (NWP start-up is always initiated by the MCU once the MCU boot loading is completed).

Resolution: The MCU application must detect a recovery from the WDOG trigger and force the device into complete hibernation with a wake-up associated with an internal RTC timer. This ensures a complete system cleanup.

寄存器介绍下，可以直接跳过，，，

锁保护寄存器：为了避免看门狗的其他寄存器被意外地更爱了。

在配置看门狗其他寄存器的时候需要进行解锁操作，

解锁：写 0x1acce551，上锁：其他任意值

读锁寄存器：返回 0x00000001，锁住了， 返回 0x00000000 解锁了

10.3.1.1.7 WDTLOCK Register (offset = C00h) [reset = 0h]

WDTLOCK is shown in Figure 10-8 and described in Table 10-9.

Writing 0x1ACC.E551 to the WDTLOCK register enables write access to all other registers. Writing any other value to the WDTLOCK register re-enables the locked state for register writes to all the other registers, except for the Watchdog Test (WDTTEST) register. The locked state will be enabled after 2 clock cycles. Reading the WDTLOCK register returns the lock status rather than the 32-bit value written. Therefore, when write accesses are disabled, reading the WDTLOCK register returns 0x0000.0001 (when locked; otherwise, the returned value is 0x0000.0000 (unlocked)).

Figure 10-8. WDTLOCK Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDTLOCK																															
R/W-0h																															

LEGEND: R/W = Read/Write; R = Read only; W1toCl = Write 1 to clear bit; -*n* = value after reset

Table 10-9. WDTLOCK Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	WDTLOCK	R/W	0h	Watchdog Lock A write of the value 0x1ACC.E551 unlocks the watchdog registers for write access. A write of any other value reapplies the lock, preventing any register updates, except for the WDTTEST register. Avoid writes to the WDTTEST register when the watchdog registers are locked. A read of this register returns the following values: 0h = Unlocked 1h = Locked

重装载寄存器

重装载寄存器决定了看门狗的溢出时间

10.3.1.1.1 WDTLOAD Register (offset = 0h) [reset = FFFFFFFFh]

WDTLOAD is shown in Figure 10-2 and described in Table 10-3.

This register is the 32-bit interval value used by the 32-bit counter. When this register is written, the value is immediately loaded and the counter restarts counting down from the new value. If the WDTLOAD register is loaded with 0x0000.0000, an interrupt is immediately generated.

Figure 10-2. WDTLOAD Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDTLOAD																															
R/W-FFFFFFFFh																															

LEGEND: R/W = Read/Write; R = Read only; W1toCl = Write 1 to clear bit; *-n* = value after reset

Table 10-3. WDTLOAD Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	WDTLOAD	R/W	FFFFFFFFh	Watchdog Load Value

控制寄存器

貌似只有 INTEN 有效，使能看门狗

10.3.1.1.3 WDTCTL Register (offset = 8h) [reset = 80000000h]

WDTCTL is shown in Figure 10-4 and described in Table 10-5.

This register is the watchdog control register. The watchdog timer can be used to generate a reset signal (on second time-out) or an interrupt on time-out.

Figure 10-4. WDTCTL Register

31	30	29	28	27	26	25	24
WRC	RESERVED						
R-1h	R-0h						
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED					INTTYPE	RESERVED	INTEN
R-0h					R/W-0h	R/W-0h	R/W-0h

LEGEND: R/W = Read/Write; R = Read only; W1toCl = Write 1 to clear bit; *-n* = value after reset

Table 10-5. WDTCTL Register Field Descriptions

Bit	Field	Type	Reset	Description
31	WRC	R	1h	Write Complete The WRC values are defined as follows: Note: This bit is reserved for WDT0 and has a reset value of 0. 0h = A write access to one of the WDT1 registers is in progress. 1h = A write access is not in progress, and WDT1 registers can be read or written.
30-3	RESERVED	R	0h	
2	INTTYPE	R/W	0h	Watchdog Interrupt Type The INTTYPE values are defined as follows: 0h = Watchdog interrupt is a standard interrupt. 1h = Not Valid Value
1	RESERVED	R/W	0h	
0	INTEN 使能看门狗	R/W	0h	Watchdog Interrupt Enable The INTEN values are defined as follows: 0h = Interrupt event disabled (once this bit is set, it can only be cleared by a hardware reset). 1h = Interrupt event enabled. Once enabled, all writes are ignored. Setting this bit enables the Watchdog Timer.

代码

Main 函数，我是直接在 SDK 上改的，用于测试

```
void main(void)
{
    tBoolean bRetcode;
    //
    // Initialize the board
    //
    BoardInit();
    //
    // Pinmuxing for LEDs
    //
    PinMuxConfig();
    //
    // configure RED LED
    //
    uart0_Init();
    GPIO_IF_LedConfigure(LED1);

    GPIO_IF_LedOff(MCU_RED_LED_GPIO);


#if 0
    //
    // 设置成没有中断函数的看门狗.主函数不做任何操作，
    // 通过输出看门狗的值，我们发现：
    //值变化 ：重新载入值 -》 0 -》重新载入值-》复位
    //第一个参数是中断函数，第二个是重装值
    WDT_IF_Init(NULL, MILLISECONDS_TO_TICKS(WD_PERIOD_MS));
    //使能看门狗
    bRetcode = MAP_WatchdogRunning(WDT_BASE);
    if(!bRetcode)
    {
       WDT_IF_DeInit();
    }
    my_printf("start_____________________________________________\r\n");
    while(1) {
    delay(20000);
    //获取计数器的值
    my_printf("%d\r\n",MAP_WatchdogValueGet(WDT_BASE));
    }
#endif
```

```
#if 0
    //
    // 设置成没有中断函数的看门狗.主函数一直清除中断状态
    // 通过输出看门狗的值，我们发现：
    //值变化 ： 重新载入值 -》某一个值-》清除中断状态导致了重新载入值-。。。。。
    //
    WDT_IF_Init(NULL, MILLISECONDS_TO_TICKS(WD_PERIOD_MS));

    bRetcode = MAP_WatchdogRunning(WDT_BASE);
    if(!bRetcode)
    {
       WDT_IF_DeInit();
    }
    my_printf("start_____________________________________________\r\n");
    while(1) {
    delay(20000);
    //清除中断状态
    MAP_WatchdogIntClear(WDT_BASE);
    //获取计数器的值
    my_printf("%d\r\n",MAP_WatchdogValueGet(WDT_BASE));
    }
#endif

#if 1
    //
    // 设置成中断函数的看门狗.
    // 中断函数没有任何作用，仅仅打印调试信息
    // 通过输出看门狗的值，我们发现：
    //值变化 ： 重新载入值 -》 0 -》中断函数触发（一直触发中断，实际上是有重新载入
值，并递减） -》触发复位
    //
    WDT_IF_Init(WatchdogIntHandler_NULL,
MILLISECONDS_TO_TICKS(WD_PERIOD_MS));

    bRetcode = MAP_WatchdogRunning(WDT_BASE);
    if(!bRetcode)
    {
       WDT_IF_DeInit();
    }
    my_printf("start_____________________________________________\r\n");
    while(1) {
    delay(20000);
```

```
    //获取计数器的值
    my_printf("%d\r\n",MAP_WatchdogValueGet(WDT_BASE));
    }
#endif

#if 0
    //
    // 设置成中断函数的看门狗.
    // 中断函数清楚了中断触发状态，
    // 通过输出看门狗的值，我们发现：
    //值变化 ： 重新载入值 -》 0 -》中断函数触发（清楚中断状态）-》重新载入值-》
0-》中断触发（清除中断状态）。。。。。。。。
    //
    WDT_IF_Init(WatchdogIntHandler, MILLISECONDS_TO_TICKS(WD_PERIOD_MS));

    bRetcode = MAP_WatchdogRunning(WDT_BASE);
    if(!bRetcode)
    {
       WDT_IF_DeInit();
    }
    my_printf("start_____________________________________________\r\n");
    while(1) {
    delay(20000);
    //获取计数器的值
    my_printf("%d\r\n",MAP_WatchdogValueGet(WDT_BASE));
    }
#endif


#if 0
    //
    // 设置成中断函数的看门狗.主任务清除中断状态
    // 中断函数没有任何作用，仅仅打印调试信息
    // 通过输出看门狗的值，我们发现：
    //值变化 ： 重新载入值 -》某一个值-》清除中断状态导致了重新载入值-》某一个值
-。。。。。
    //
    WDT_IF_Init(WatchdogIntHandler_NULL,
MILLISECONDS_TO_TICKS(WD_PERIOD_MS));

    bRetcode = MAP_WatchdogRunning(WDT_BASE);
    if(!bRetcode)
    {
       WDT_IF_DeInit();
```

```
    }
    my_printf("start_____________________________________________\r\n");
    while(1) {
    delay(20000);
    //清除中断状态
    MAP_WatchdogIntClear(WDT_BASE);
    //获取计数器的值
    my_printf("%d\r\n",MAP_WatchdogValueGet(WDT_BASE));
    }
#endif

}
```

中断回调函数

```
void WatchdogIntHandler(void)
{
    my_printf("interrupt  clear
state_____________________________________\r\n");
    MAP_WatchdogIntClear(WDT_BASE);
}
void WatchdogIntHandler_NULL(void)
{
    my_printf("interrupt no clear
state_____________________________________\r\n");
    my_printf("%d\r\n",MAP_WatchdogValueGet(WDT_BASE));

}
```