目 录

第1章 看门狗(WatchDog)

表 1.1 WatchDog 库函数索引

1.1 什么是看门狗

在实际的 MCU 应用系统中，由于常常会受到来自外界的某些干扰，有可能造成程序跑飞而进入死循环，从而导致整个系统的陷入停滞状态并且不会自动恢复到可控的工作状态。所以出于对 MCU 运行的安全考虑，便引入了一种专门的复位监控电路 WatchDog，俗称看门狗。看门狗电路所起的作用是一旦 MCU 运行出现故障，就强制对 MCU 进行硬件复位，使整个系统重新处于可控状态(要想精确恢复到故障之前的运行状态从技术上讲难度大成本高，而复位是最简单且可靠的处理手段)。

SP706 是 Exar（原 Sipex）公司推出的低功耗、高可靠、低价格的 MCU 复位监控芯片。以下是其关键特性：

- 分为 4 个子型号：SP706P、SP706R、SP706S、SP706T
- 复位输出：P 为高电平有效，R/S/T 为低电平有效
- 精密的低电压监控：P/R 为 2.63V、S 为 2.93V、T 为 3.08V
- 复位脉冲宽度：200ms（额定值）
- 独立的看门狗定时器：1.6 秒超时（额定值）
- 去抖 TTL/CMOS 手动复位输入（/MR 管脚）

图 1.1 给出了 SP706 的一个典型的应用电路：U1 是复位监控芯片 SP706S，其中电源失效检测功能未被用到，因此 PFI 管脚直接连到 GND（接 VCC 也可）；U2 是被监控的 MCU，这是简化的模型。

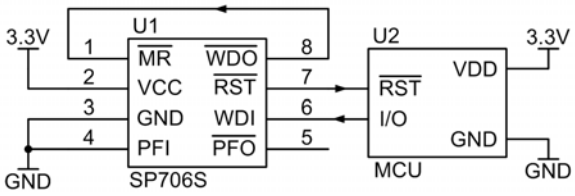

图 1.1 看门狗芯片 SP706 应用电路图

表 1.2 给出了 SP706R/S/T 的管脚功能描述（P 型的复位输出为高电平有效）。

表 1.2 SP706R/S/T 管脚功能

管脚编号	管脚名称	功能描述
1	/MR	手工复位，输入低电平时会导致/RST 管脚输出复位信号
2	VCC	电源，1.0～5.5V
3	GND	地
4	PFI	电源失效输入：接内部比较器的同相端，比较器反相端接内部 1.25V 参考源
5	/PFO	电源失效输出：来自内部比较器的输出端
6	WDI	看门狗输入：浮空时禁止看门狗功能；固定接 HIGH 或 LOW 电平 1.6s 后看门狗定时器溢出导致/WDO 管脚输出低电平；反转输入状态会清除看门狗定时器
7	/RST	复位信号输出，低电平有效
8	/WDO	看门狗输出，内部看门狗定时器溢出时输出低电平

系统上电时，SP706S 自动产生 200ms 低电平复位信号，使 MCU 正常复位。MCU 配置一个 I/O 管脚为输出，并接到 WDI。如果 I/O 固定为 HIGH 或 LOW 电平不变，则 1.6s 后，SP706S 内部的看门狗定时器就会溢出并使/WDO 输出低电平，而/WDO 已连接到手动复位/MR，因此会导致/RST 管脚输出低电平复位信号使 MCU 重新复位。MCU 在正常工作情况下当然是不允许这样反复复位的，因此必须在程序里及时反转 I/O 的状态，该操作被形象地称为“喂狗”。每次反转 WDI 输入状态都能够清除 SP706S 内部的看门狗定时器，从而确保/WDO 不会输出低电平（为保证可靠，喂狗间隔应当小于 1s）。

在程序里如何进行喂狗操作呢？一般的做法是，先编写一个能够使 WDI 状态反转的喂狗函数，然后把函数调用插入到每一个可能导致长时间执行的程序段里，最常见的情况是 while(1)、for(;;)之类的无条件循环语句。

一旦程序因为意外情况跑飞，很可能会陷入一个不含喂狗操作的死循环里，超过 1.6s 后就会自动复位重来，而不会永远停留在故障状态。

1.2 WatchDog 功能概述

在 Stellaris 系列 ARM 里集成有硬件的看门狗定时器模块。看门狗定时器在到达超时值时会产生不可屏蔽的中断或复位。当系统由于软件错误而无法响应或外部器件不能以期望的方式响应时，使用看门狗定时器可重新获得控制。Stellaris 系列的看门狗定时器模块有以下特性：

- 带可编程装载寄存器的 32 位倒计数器
- 带使能控制的独立看门狗时钟
- 带中断屏蔽的可编程中断产生逻辑
- 软件跑飞时由锁定寄存器提供保护

- 带使能/禁止控制的复位产生逻辑
- 在调试过程中用户可控制看门狗暂停

看门狗定时器模块包括 32 位倒计数器（以 6MHz 系统时钟为例，最长定时接近 12 分钟）、可编程的装载寄存器、中断产生逻辑、锁定寄存器以及用户使能的暂停控制。

看门狗定时器具有“二次超时”特性。当 32 位计数器在使能后倒计数到 0 状态时，看门狗定时器模块产生第一个超时信号，并产生中断触发信号。在发生了第一个超时事件后，32 位计数器自动重装并重新递减计数。如果没有清除第一个超时中断状态，则当计数器再次递减到 0 时，且复位功能已使能，则看门狗定时器会向处理器发出复位信号。如果中断状态在 32 位计数器到达其第二次超时之前被清除（即喂狗操作），则自动重装 32 位计数器，并重新开始计数，从而可以避免处理器被复位。

为了防止在程序跑飞时意外修改看门狗模块的配置，特意引入了一个锁定寄存器。在配置看门狗定时器之后，只要写入锁定寄存器一个不是 0x1ACCE551 的任何数值，看门狗模块的所有配置都会被锁定，拒绝软件修改。因此以后要修改看门狗模块的配置，包括清除中断状态（即喂狗操作），都必须要首先解锁。解锁方法是向锁定寄存器写入数值 0x1ACCE551。这是个很特别的数字，程序跑飞本身已是罕见的事件，而在一旦发生此罕见事件的情况下又恰好会把这个特别的数字写入锁定寄存器更是不可能。读锁定寄存器将得到看门狗模块是否被锁定的状态，而非写入的数值。

为了防止在调试软件时看门狗产生复位，看门狗模块还提供了允许其暂停计数的功能。

1.3 如何正确使用看门狗

看门狗真正的用法应当是：在不用看门狗的情况下，硬件和软件经过反复测试已经通过，而在考虑到在实际应用环境中出现的强烈干扰可能造成程序跑飞的意外情况时，再加入看门狗功能以进一步提高整个系统的工作可靠性。可见，看门狗只不过是万不得已的最后手段而已。

但是，有相当多的工程师，尤其是经验不多者，在调试自己的系统时一出现程序跑飞，就马上引入看门狗来解决，而没有真正去思考程序为什么会跑飞。实际上，程序跑飞的大部分原因是程序本身存在 bug，或者已经暗示硬件电路可能存在故障，而并非是受到了外部的干扰。如果试图用看门狗功能来“掩饰”此类潜在的问题，则是相当不明智的，也是危险的，潜在的系统设计缺陷可能一直伴随着您的产品最终到用户手中。

综上，我们建议：在调试自己的系统时，先不要使用看门狗，待完全调通已经稳定工作了，最后再补上看门狗功能。

1.4 WatchDog 库函数

1． 运行控制

函数 WatchdogEnable( )的作用是使能看门狗。该函数实际执行的操作是使能看门狗中断功能，即等同于函数 WatchdogIntEnable( )。中断功能一旦被使能，则只有通过复位才能被清除。因此库函数里不会有对应的 WatchdogDisable( )函数。参见表 1.3 的描述。

函数 WatchdogRunning( )可以探测看门狗是否已被使能。参见表 1.4 的描述。

表 1.3 函数 WatchdogEnable( )

功能	使能看门狗定时器

原型	void WatchdogEnable(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	无

表 1.4 函数 WatchdogRunning( )

功能	确定看门狗定时器是否已经被使能
原型	tBoolean WatchdogRunning(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	如果看门狗定时器已被使能则返回 true，否则返回 false

函数 WatchdogResetEnable( )使能看门狗定时器的复位功能，一旦看门狗定时器产生了二次超时事件，将引起处理器复位。函数 WatchdogResetDisable( )禁止看门狗定时器的复位功能，此时可以把看门狗作为一个普通定时器来使用。参见表 1.5 和表 1.6 的描述。

表 1.5 函数 WatchdogResetEnable( )

功能	使能看门狗定时器的复位功能
原型	void WatchdogResetEnable(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	无

表 1.6 函数 WatchdogResetDisable( )

功能	禁止看门狗定时器的复位功能
原型	void WatchdogResetDisable(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	无

在进行单步调试时，看门狗定时器仍然会独立地运行，这将很快导致处理器复位，从而破坏调试过程。函数 WatchdogStallEnable( )允许看门狗定时器暂停计数，可防止在调试时引起不期望的处理器复位。函数 WatchdogStallDisable( )将禁止看门狗定时器暂停。参见表 1.7 和表 1.8 的描述。

表 1.7 函数 WatchdogStallEnable( )

功能	允许在调试过程中暂停看门狗定时器
原型	void WatchdogStallEnable(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	无

表 1.8 函数 WatchdogStallDisable( )

功能	禁止在调试过程中暂停看门狗定时器
原型	void WatchdogStallDisable(unsigned long ulBase)

参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	无

2． 装载与锁定

函数 WatchdogReloadSet( )设置看门狗定时器的装载值，WatchdogReloadGet( )获取装载值。参见表 1.9 和表 1.10 的描述。

函数 WatchdogValueGet( )能够获取看门狗定时器当前的计数值。参见表 1.11 的描述。

表 1.9 函数 WatchdogReloadSet( )

功能	设置看门狗定时器的重装值
原型	void WatchdogReloadSet(unsigned long ulBase, unsigned long ulLoadVal)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE ulLoadVal：32 位装载值
返回	无

表 1.10 函数 WatchdogReloadGet( )

功能	获取看门狗定时器的重装值
原型	unsigned long WatchdogReloadGet(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	已设置的 32 位装载值

表 1.11 函数 WatchdogValueGet( )

功能	获取看门狗定时器的计数值
原型	unsigned long WatchdogValueGet(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	当前的 32 位计数值

函数 WatchdogLock( )用来锁定看门狗定时器的配置，一旦锁定，则拒绝软件对配置的修改操作。函数 WatchdogUnlock( )用来解除锁定。参见表 1.12 和表 1.13 的描述。

函数 WatchdogLockState( )用来探测看门狗定时器的锁定状态。参见表 1.14 的描述。

表 1.12 函数 WatchdogLock( )

功能	使能看门狗定时器的锁定机制
原型	void WatchdogLock(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	无

表 1.13 函数 WatchdogUnlock( )

功能	解除看门狗定时器的锁定机制

原型	void WatchdogUnlock(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	无

表 1.14 函数 WatchdogLockState( )

功能	获取看门狗定时器的锁定状态
原型	tBoolean WatchdogLockState(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	已锁定返回 true，未锁定返回 false

3． 中断控制

函数 WatchdogIntEnable( )用来使能看门狗定时器中断。中断功能一旦被使能，则只有通过复位才能被清除。因此库函数里不会有对应的 WatchdogIntDisable( )函数。参见表 1.15 的描述。

函数 WatchdogIntStatus( )可获取看门狗定时器的中断状态，函数 WatchdogIntClear( )用来清除中断状态。参见表 1.16 和表 1.17 的描述。

函数 WatchdogIntRegister( )用来注册一个看门狗定时器的中断服务函数，而函数 WatchdogIntUnregister( )用来注销。参见表 1.18 和表 1.19 的描述。

表 1.15 函数 WatchdogIntEnable( )

功能	使能看门狗定时器中断
原型	void WatchdogIntEnable(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	无

表 1.16 函数 WatchdogIntStatus( )

功能	获取看门狗定时器的中断状态
原型	unsigned long WatchdogIntStatus(unsigned long ulBase, tBoolean bMasked)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE bMasked：如果需要原始的中断状态则取值 false，如果需要获取屏蔽的中断状态则取值 true
返回	原始的或屏蔽的中断状态

表 1.17 函数 WatchdogIntClear( )

功能	清除看门狗定时器的中断状态
原型	void WatchdogIntClear(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	无

表 1.18　函数 WatchdogIntRegister( )

功能	注册一个看门狗定时器的中断服务函数
原型	void WatchdogIntRegister(unsigned long ulBase, void(*pfnHandler)(void))
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE pfnHandler：函数指针，指向要注册的中断服务函数
返回	无

表 1.19　函数 WatchdogIntUnregister( )

功能	注销看门狗定时器的中断服务函数
原型	void WatchdogIntUnregister(unsigned long ulBase)
参数	ulBase：看门狗定时器模块的基址，取值 WATCHDOG_BASE
返回	无

1.5　WatchDog 例程

1．　看门狗复位

程序清单 1.1 演示了看门狗定时器监控处理器的用法。函数 wdogInit( )初始化看门狗模块，已知系统时钟为 6MHz，设置的定时时间为 350ms，使能复位功能，配置后锁定。函数 wdogFeed( )是喂狗操作，解锁→喂狗→锁定，并且使 LED 闪亮。程序一开始便点亮 LED，延时，再熄灭，以表示已复位。然后在主循环里每隔 500ms 喂狗一次，由于看门狗具有二次超时特性，因此不会产生复位，除非喂狗间隔超过了 2×350ms。

程序清单 1.1　WatchDog 例程：看门狗复位

文件：main.c

```
#include    "systemInit.h"
#include    "watchdog.h"

//  定义 LED
#define   LED_PERIPH              SYSCTL_PERIPH_GPIOG
#define   LED_PORT                GPIO_PORTG_BASE
#define   LED_PIN                 GPIO_PIN_2

//  LED 初始化
void ledInit(void)
{
      SysCtlPeriEnable(LED_PERIPH);                    //  使能 LED 所在的 GPIO 端口
      GPIOPinTypeOut(LED_PORT, LED_PIN);               //  设置 LED 所在管脚为输出
      GPIOPinWrite(LED_PORT, LED_PIN, 0xFF);           //  熄灭 LED
}

//  看门狗初始化
void wdogInit(void)
```

```
{
    unsigned long ulValue = 350 * (TheSysClock / 1000);          //  准备定时 350ms

    SysCtlPeriEnable(SYSCTL_PERIPH_WDOG);                        //  使能看门狗模块
    WatchdogResetEnable(WATCHDOG_BASE);                          //  使能看门狗复位功能
    WatchdogStallEnable(WATCHDOG_BASE);                          //  使能调试器暂停看门狗计数
    WatchdogReloadSet(WATCHDOG_BASE, ulValue);                   //  设置看门狗装载值
    WatchdogEnable(WATCHDOG_BASE);                               //  使能看门狗
    WatchdogLock(WATCHDOG_BASE);                                 //  锁定看门狗
}

//  喂狗操作
void wdogFeed(void)
{
    WatchdogUnlock(WATCHDOG_BASE);                               //  解除锁定
    WatchdogIntClear(WATCHDOG_BASE);                             //  清除中断状态，即喂狗操作
    WatchdogLock(WATCHDOG_BASE);                                 //  重新锁定

    GPIOPinWrite(LED_PORT, LED_PIN, 0x00);                       //  点亮 LED
    SysCtlDelay(2 * (TheSysClock / 3000));                       //  短暂延时
    GPIOPinWrite(LED_PORT, LED_PIN, 0xFF);                       //  熄灭 LED
}

//  主函数（程序入口）
int main(void)
{
    jtagWait( );                                                 //  防止 JTAG 失效，重要！
    clockInit( );                                                //  时钟初始化：晶振，6MHz
    ledInit( );                                                  //  LED 初始化

    GPIOPinWrite(LED_PORT, LED_PIN, 0x00);                       //  点亮 LED，表明已复位
    SysCtlDelay(1500 * (TheSysClock / 3000));
    GPIOPinWrite(LED_PORT, LED_PIN, 0xFF);                       //  熄灭 LED
    SysCtlDelay(1500 * (TheSysClock / 3000));

    wdogInit( );                                                 //  看门狗初始化

    for (;;)
    {
        wdogFeed( );                                             //  喂狗，每喂一次 LED 闪一下
        SysCtlDelay(500 * (TheSysClock / 3000));                 //  延时超过 2×350ms 才会复位
    }
}
```

2．　作为普通定时器

程序清单 1.2 演示了看门狗作为普通定时器的用法，关键是用 WatchdogResetDisable( ) 函数禁止其复位处理器的功能，也不需要锁定和解锁操作。程序的功能是：配置看门狗定时器，并使能其中断功能，当计数器归 0 时产生超时中断，在中断服务函数里反转 LED，使之不断闪烁发光。

程序清单 1.2　WatchDog 例程：作为普通定时器

文件：main.c

```
#include    "systemInit.h"
#include    "watchdog.h"

//  定义 LED
#define   LED_PERIPH              SYSCTL_PERIPH_GPIOG
#define   LED_PORT                GPIO_PORTG_BASE
#define   LED_PIN                 GPIO_PIN_2

//  LED 初始化
void ledInit(void)
{
      SysCtlPeriEnable(LED_PERIPH);                          //  使能 LED 所在的 GPIO 端口
      GPIOPinTypeOut(LED_PORT, LED_PIN);                     //  设置 LED 所在管脚为输出
      GPIOPinWrite(LED_PORT, LED_PIN, 0xFF);                 //  熄灭 LED
}

//  看门狗初始化
void wdogInit(void)
{
      unsigned long ulValue = 350 * (TheSysClock / 1000);    //  准备定时 350ms

      SysCtlPeriEnable(SYSCTL_PERIPH_WDOG);                  //  使能看门狗模块
      WatchdogResetDisable(WATCHDOG_BASE);                   //  禁止看门狗复位功能
      WatchdogStallEnable(WATCHDOG_BASE);                    //  使能调试器暂停看门狗计数
      WatchdogReloadSet(WATCHDOG_BASE, ulValue);             //  设置看门狗装载值
      WatchdogIntEnable(WATCHDOG_BASE);                      //  使能看门狗中断
      IntEnable(INT_WATCHDOG);                               //  使能看门狗模块中断
      IntMasterEnable( );                                    //  使能处理器中断
      WatchdogEnable(WATCHDOG_BASE);                         //  使能看门狗
}

//  主函数（程序入口）
int main(void)
{
      jtagWait( );                                           //  防止 JTAG 失效，重要！
```

```
    clockInit( );                                   //  时钟初始化：晶振，6MHz
    ledInit( );                                     //  LED 初始化
    wdogInit( );                                    //  看门狗初始化

    for (;;)
    {
    }
}

//  看门狗中断服务函数
void Watchdog_Timer_ISR(void)
{
    unsigned char ucValue;
    unsigned long ulStatus;

    ulStatus = WatchdogIntStatus(WATCHDOG_BASE, true);      //  获取看门狗中断状态
    WatchdogIntClear(WATCHDOG_BASE);                        //  清除中断状态，重要！

    if (ulStatus != 0)
    {
        ucValue = GPIOPinRead(LED_PORT, LED_PIN);           //  反转 LED
        GPIOPinWrite(LED_PORT, LED_PIN, ~ucValue);
    }
}
```