12. STM32的看门狗机制（WTD）

北京科技大学
计算机与通信工程学院

目录

CONTENTS.

STM32-12

看门狗的作用

- 看门狗的作用

看门狗是在程序跑飞的情况下，将CPU自恢复的一种方式，当软件在选定的时间间隔内不能置位看门狗定时器，看门狗就复位系统。

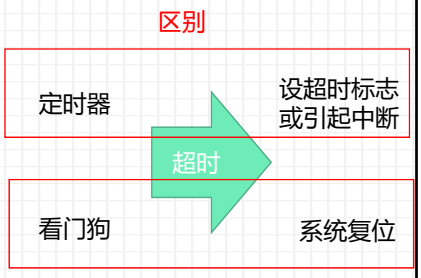
- 使用场景

1. 看门狗可用于电噪声、电源故障或静电放电等恶劣工作环境或高可靠性要求的环境。
2. 嵌入式软件在产品不成熟的部署期间为了防止出错死机，也应该使用看门狗避免高昂的人工维护。

看门狗的工作原理

看门狗(Watch Dog Timer， WDT)是一种专门用于监测单片机程序运行状态的芯片组件。

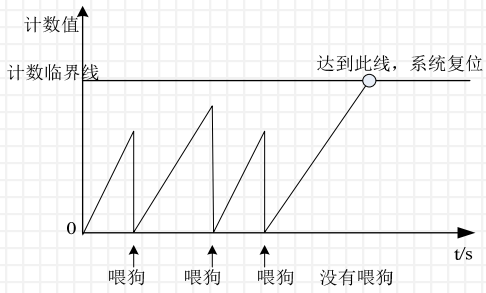
看门狗实质是一个计数器，一般给看门狗一个大数，程序开始运行后，看门狗开始倒计数。如果程序运行正常，过一段时间CPU应发出指令让看门狗复位，重新开始倒计数。如果看门狗减到0，就认为程序没有正常工作，将强制整个系统复位。



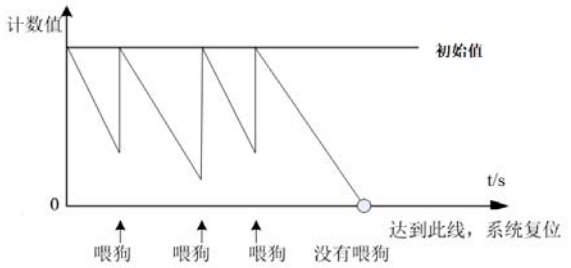
看门狗工作过程：及时喂狗，否则会重启

当启动看门狗定时器后，它就会从初始值开始计数，若程序在规定的时间间隔内没有及时对其重置为初始值（喂狗），看门狗定时器就会复位系统（相当于重启），如图所示。

递增计数的看门狗

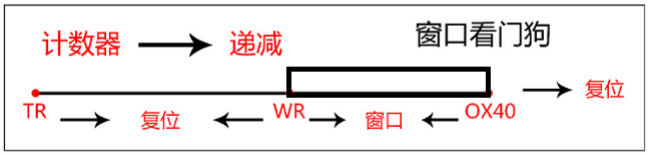


递减计数的看门狗



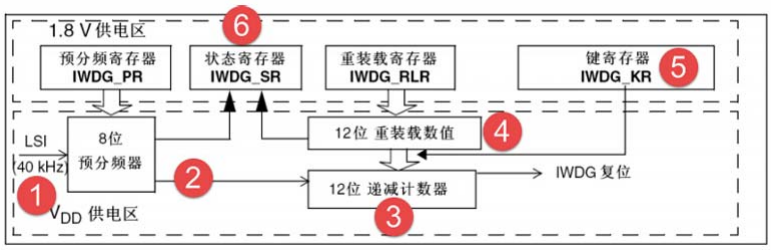
STM32的独立看门狗和窗口看门狗

- 独立看门狗就是常规的看门狗，工作原理就是一个递减计数器不断的往下递减计数，当减到 0之前如果没有喂狗的话， 产生复位。宠物狗。
- 窗口看门狗跟独立看门狗一样，也是一个递减计数器不断的往下递减计数，当减到一个固定值0x40时还不喂狗的话，产生复位，这个值叫**窗口的下限**，是固定的值，不能改变。这个是跟独立看门狗类似的地方，不同的地方是窗口看门狗的计数器的值在减到某一个数之前喂狗的话也会产生复位，这个值叫**窗口的上限**，上限值由用户独立设置。**窗口看门狗计数器的值必须在上窗口和下窗口之间才可以喂狗**，这就是窗口看门狗中窗口两个字的含义 。警犬。



IWDG 与 WWDG 区别

IWDG功能框图剖析

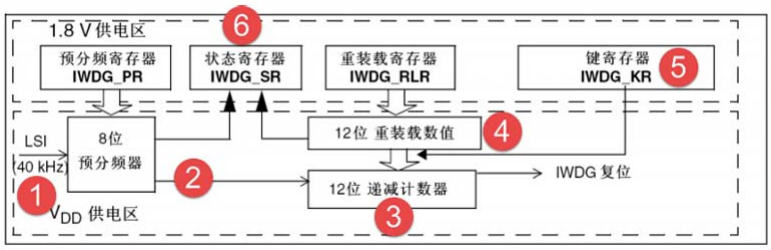


①独立看门狗时钟

- 独立看门狗的时钟由独立的**RC振荡器LSI**提供，即使主时钟发生故障它仍然有效，非常**独立。**
- LSI的频率一般在30~60KHZ之间，根据温度和工作场合会有一定的漂移，**一般取40KHZ**，所以独立看门狗的定时时间并一定非常精确，只适用于对时间精度要求比较低的场合。

7

IWDG功能框图剖析

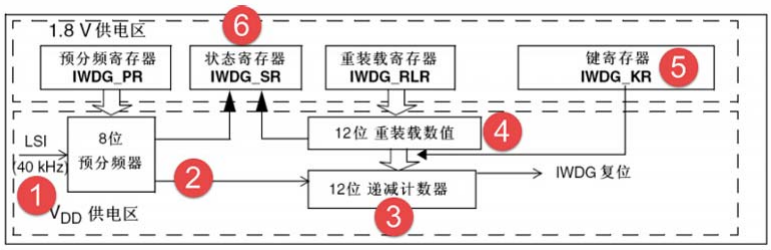


②计数器时钟

- 递减计数器的时钟由 LSI经过一个8位的预分频器得到
- 预分频器寄存器 IWDG_PR用来设置分频因子，可以是：[4,8,16,32,64,128,256,256]，
- 计数器时钟 CK_CNT= 40/ 4*2^PRV
- 一个计数器时钟计数器就减一

8

IWDG功能框图剖析

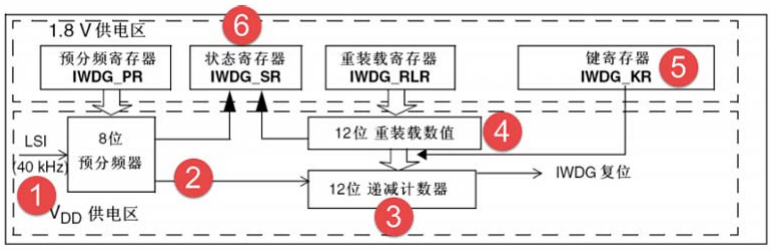


③计数器

- 独立看门狗的计数器是一个**12位**的递减计数器，最大值为0xFFF，
- 当计数器减到0时， 会产生一个**复位信号**:IWDG_RESET，让程序重新启动运行
- 如果在计数器减到0之前刷新了计数器的值的话，就不会产生复位信号
- 重新刷新计数器值的这个动作俗称为**喂狗**

9

IWDG功能框图剖析

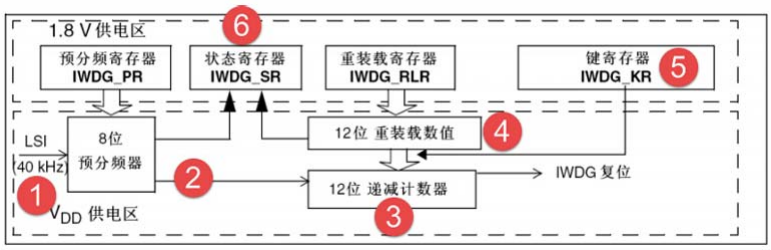


④重装载寄存器

- 重装载寄存器是一个12位的寄存器，里面装着要刷新到计数器的值，
  这个值的大小决定着独立看门狗的溢出时间
- 超时时间Tout = (4*2^prv) / 40 * rlv (s) ，prv是预分频器寄存器
  的值，rlv是重装载寄存器的值

10

IWDG功能框图剖析



⑤键寄存器

- 键寄存器 IWDG_KR可以说是独立看门狗的一个控制寄存器，主要有三种控制方式，往该寄存器写入下面三个不同的键值有不同的效果（键值作用）

  0xAAAA ---- 把RLR的值重装载到CNT

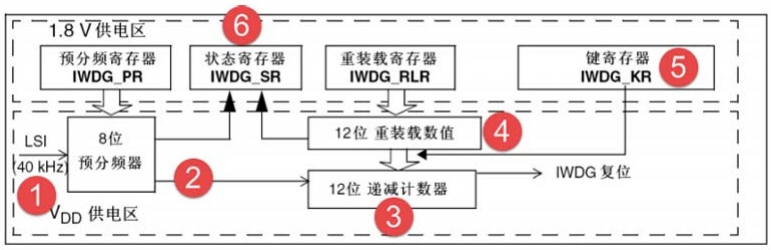
  0x5555 ----- PR和RLR 这两个寄存器可写

  0xCCCC ----- 启动 IWDG

- 通过写往键寄存器写0XCCC 来启动看门狗是属于软件启动的方式，一旦独立看门狗启动，它就关不掉，只有复位才能关掉。

11

IWDG功能框图剖析



- **⑥状态寄存器**
- SR只有位0：PVU 和位1：RVU有效，只能由硬件操作，软件操作不了。
- RVU：看门狗计数器重装载值更新，硬件置1表示重装载值的更新正在进行中，更新完毕之后由硬件清0。
- PVU： 看门狗预分频值更新，硬件置1指示预分频值的更新正在进行中，当更新完成后，由硬件清0。
- 所以只有当RVU/PVU 等于 0 的时候才可以更新重装载寄存器/预分频寄存器。

12

目录

CONTENTS.

STM32-12

13

IWDG库函数

void IWDG_WriteAccessCmd(uint16_t IWDG_WriteAccess);
功能：使能/禁止 预分频寄存器PR和重装载寄存器RLR可写 // IWDG->KR = IWDG_WriteAccess;
参数取值：IWDG_WriteAccess_Enable（0x5555） 或 IWDG_WriteAccess_Disable （0x0000）

void IWDG_SetPrescaler(uint8_t IWDG_Prescaler);
功能： // 设置预分频器值
参数取值： IWDG_Prescaler_4；IWDG_Prescaler_8;…; IWDG_Prescaler_256

void IWDG_SetReload(uint16_t Reload);
// 设置重装载寄存器值
//参数取值：between 0 and 0x0FFF.

void IWDG_ReloadCounter(void);
// 把重装载寄存器的值放到计数器中，等同于： IWDG->KR = (uint16_t)0xAAAA;

void IWDG_Enable(void);
// 使能 IWDG，等同于： IWDG->KR = ((uint16_t)0xCCCC);

14

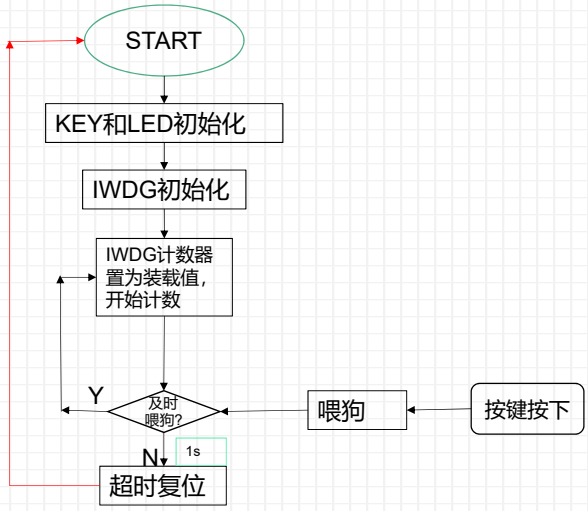
目录

CONTENTS.

STM32-12

15

IWDG例程流程



16

Main.c

```
int main(void)
{
        // 配置LED GPIO，点亮蓝色LED
        LED_GPIO_Config();
        LED_BLUE;

        // 配置按键GPIO
        Key_GPIO_Config();
        // IWDG 1s 超时溢出
        IWDG_Config(IWDG_Prescaler_64 ,625);

        while(1) {
                // 这里添加需要被监控的代码，如果有就去掉按键模拟喂狗，把按键扫描程序去掉
                if( Key_Scan(KEY1_GPIO_PORT,KEY1_GPIO_PIN) == KEY_ON  ) {
                        // 喂狗，如果不喂狗，系统则会复位，LED1则会灭一次，如果在1s
                        // 时间内准时喂狗的话，则绿会常亮
                        IWDG_Feed();
                        //喂狗后亮绿灯
                        LED_GREEN;
                }
        }
```

while部分是我们在项目中具体需要写的代码，这部分的程序可以用独立看门狗来监控。
如果我们知道这部分代码的执行时间，比如是500ms，那么我们可以设置独立看门狗的溢出时间是600ms，比500ms多一点，如果要被监控的程序没有跑飞正常执行的话，那么执行完毕之后就会执行喂狗的程序，如果程序跑飞了那程序就会超时，到达不了喂狗的程序,此时就会产生系统复位。

17

KEY设置

```
#include "stm32f10x.h"
//  引脚定义
#define    KEY1_GPIO_CLK      RCC_APB2Periph_GPIOA
#define    KEY1_GPIO_PORT     GPIOA
#define    KEY1_GPIO_PIN      GPIO_Pin_0
#define KEY_ON         1
#define KEY_OFF        0

void Key_GPIO_Config(void) {
        GPIO_InitTypeDef GPIO_InitStructure;

        /*开启按键端口的时钟*/
        RCC_APB2PeriphClockCmd(KEY1_GPIO_CLK,ENABLE);
        //选择按键的引脚
        GPIO_InitStructure.GPIO_Pin = KEY1_GPIO_PIN;
        // 设置按键的引脚为浮空输入
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        //使用结构体初始化按键
        GPIO_Init(KEY1_GPIO_PORT, &GPIO_InitStructure);
}

uint8_t Key_Scan(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin) {
        if(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON ) {/*检测是否有按键按下 */
                while(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON); /*等待按键释放 */
                return        KEY_ON;
        }
        else        return KEY_OFF;
}
```

18

IWDG初始化

```
void IWDG_Config(uint8_t prv ,uint16_t rlv){
        // 使能 预分频寄存器PR和重装载寄存器RLR可写
        IWDG_WriteAccessCmd( IWDG_WriteAccess_Enable );

        // 设置预分频器值
        IWDG_SetPrescaler( prv );

        // 设置重装载寄存器值
        IWDG_SetReload( rlv );

        // 把重装载寄存器的值放到计数器中
        IWDG_ReloadCounter();

        // 使能 IWDG
        IWDG_Enable();
}
```

```
// IWDG 1s 超时溢出
IWDG_Config(IWDG_Prescaler_64 ,625);
```

Why?

19

喂狗代码

```
// 喂狗
void IWDG_Feed(void)
{
        // 把重装载寄存器的值放到计数器中，喂狗，防止IWDG复位
        // 当计数器的值减到0的时候会产生系统复位
        IWDG_ReloadCounter();
}
```

20

Main.c改进

```
int main(void)
{
        // 配置按键GPIO
        Key_GPIO_Config();

        // 配置LED GPIO，点亮蓝色LED
        LED_GPIO_Config();
        LED_BLUE;

                // IWDG 1s 超时溢出
        IWDG_Config(IWDG_Prescaler_64 ,625);

        while(1) {
                // 这里添加需要被监控的代码，如果有就去掉按键模拟喂狗，把按键扫描程序去掉
                if( Key_Scan(KEY1_GPIO_PORT,KEY1_GPIO_PIN) == KEY_ON  ) {
                        // 喂狗，如果不喂狗，系统则会复位，LED1则会灭一次，如果在1s
                        // 时间内准时喂狗的话，则绿会常亮
                        IWDG_Feed();
                        //喂狗后亮绿灯
                        LED_GREEN;
                }
        }
```

替换

```
Delay(0X8FFFFF);
 /* 检查是否为独立看门狗复位 */
  if (RCC_GetFlagStatus(RCC_FLAG_IWDGRST) != RESET)
  {
    /* 独立看门狗复位 */
    /*  亮红灯 */
    LED_RED;

    /* 清除标志 */
    RCC_ClearFlag();

    /*如果一直不喂狗，会一直复位，加上前面的延时，会看到红灯闪烁
     在1s 时间内喂狗的话，则会持续亮绿灯*/
  }
  else
  {
    /*不是独立看门狗复位(可能为上电复位或者手动按键复位之类的) */
    /* 亮蓝灯 */
    LED_BLUE;
  }
```


Reference：RCC functions

```
1311    *   For @b other_STM32_devices, this parameter can be one of the following values:
1312    *     @arg RCC_FLAG_HSIRDY: HSI oscillator clock ready
1313    *     @arg RCC_FLAG_HSERDY: HSE oscillator clock ready
1314    *     @arg RCC_FLAG_PLLRDY: PLL clock ready
1315    *     @arg RCC_FLAG_LSERDY: LSE oscillator clock ready
1316    *     @arg RCC_FLAG_LSIRDY: LSI oscillator clock ready
1317    *     @arg RCC_FLAG_PINRST: Pin reset
1318    *     @arg RCC_FLAG_PORRST: POR/PDR reset
1319    *     @arg RCC_FLAG_SFTRST: Software reset
1320    *     @arg RCC_FLAG_IWDGRST: Independent Watchdog reset
1321    *     @arg RCC_FLAG_WWDGRST: Window Watchdog reset
1322    *     @arg RCC_FLAG_LPWRRST: Low Power reset
1323    *
1324    * @retval The new state of RCC_FLAG (SET or RESET).
1325    */
1326  FlagStatus RCC_GetFlagStatus(uint8_t RCC_FLAG)
1327  {
1328    uint32_t tmp = 0;
1329    uint32_t statusreg = 0;
1330    FlagStatus bitstatus = RESET;
1331    /* Check the parameters */
1332    assert_param(IS_RCC_FLAG(RCC_FLAG));
```

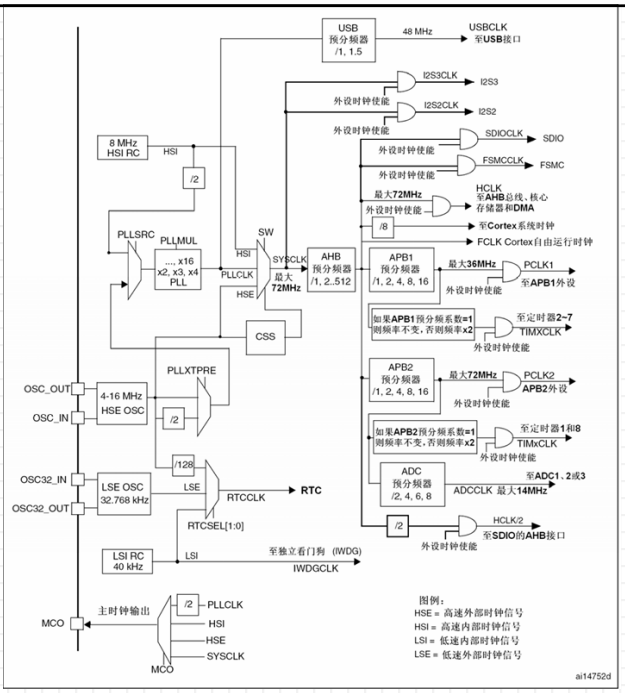
```
/**
  * @brief  Clears the RCC reset flags.
  * @note   The reset flags are: RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST,
  *   RCC_FLAG_IWDGRST, RCC_FLAG_WWDGRST, RCC_FLAG_LPWRRST
  * @param  None
  * @retval None
  */
void RCC_ClearFlag(void)
{
  /* Set RMVF bit to clear the reset flags */
  RCC->CSR |= CSR_RMVF_Set;
}
```

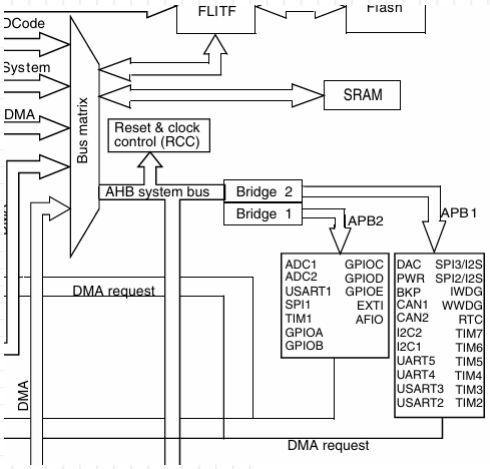
WDG有时钟使能吗？重温时钟树

独立看门狗(IWDG)由专用的低速时钟(LSI)驱动，即使主时钟发生故障它也仍然有效。

窗口看门狗由从APB1时钟分频后得到的时钟驱动，通过可配置的时间窗口来检测应用程序非正常的过迟 或过早的操作。

IWDG最适合应用于那些需要看门狗作为一个在主程序之外，能够完全独立工作，并且对时间精 度要求较低的场合。WWDG最适合那些要求看门狗在精确计时窗口起作用的应用程序。





WWDG有时钟使能

```
/* WWDG 配置函数
 * tr : 递减计时器的值， 取值范围为：0x7f~0x40
 * wr : 窗口值，取值范围为：0x7f~0x40
 * prv: 预分频器值，取值可以是
 *      @arg WWDG_Prescaler_1: WWDG counter clock = (PCLK1/4096)/1
 *      @arg WWDG_Prescaler_2: WWDG counter clock = (PCLK1/4096)/2
 *      @arg WWDG_Prescaler_4: WWDG counter clock = (PCLK1/4096)/4
 *      @arg WWDG_Prescaler_8: WWDG counter clock = (PCLK1/4096)/8
 */
void WWDG_Config(uint8_t tr, uint8_t wr, uint32_t prv)
{
    // 开启 WWDG 时钟
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_WWDG, ENABLE);

```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留		DACEN	PWR EN	BKP EN	保留	CAN EN	保留	USB EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	保留
		rw	rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	保留		WWDG EN	保留					TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw						rw	rw	rw	rw	rw	rw