**Microcontroller Experiment**

Independent Watchdog (IWDG) Experiment

Name: Han Yichen 韩一尘

Student ID: 22722051

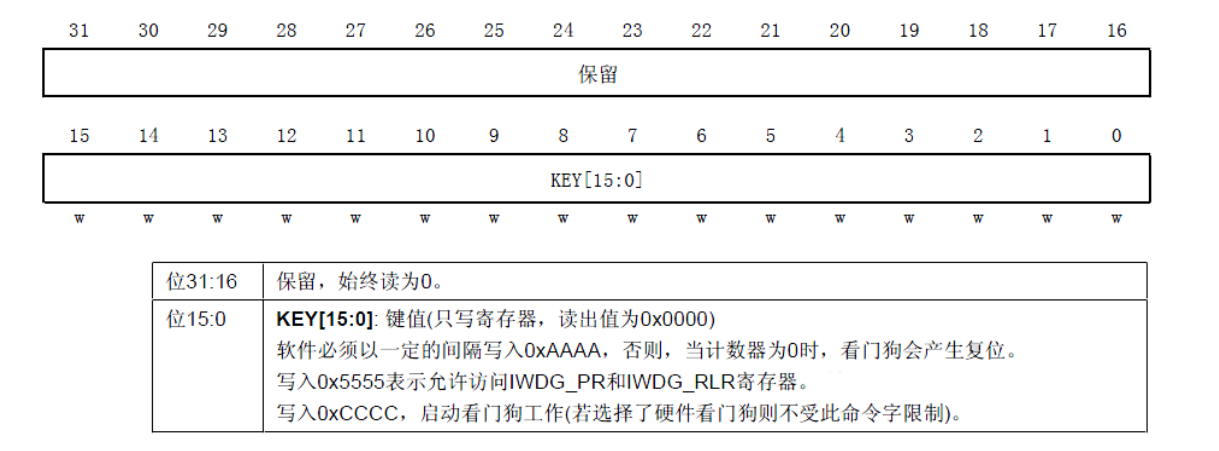
Date: 2023/11/1

1. **Experimental Principles**

The independent watchdog of STM32 is driven by an internal dedicated 40Khz low-speed clock, which remains effective even if the main clock malfunctions. It should be noted that the clock of the independent watchdog is an internal RC clock, so it is not an accurate 40Khz, but a variable clock between 30 and 60Khz. However, when we estimate using a frequency of 40Khz, the watchdog's time requirements are not very precise, so some clock deviations are acceptable. The principle is as follows:

The microcontroller system may experience a dead loop in the program under external interference, and the watchdog circuit is designed to avoid this situation. The function of a watchdog is to automatically reset and restart the processor (sending a reset signal) within a certain period of time (achieved through a timer counter) without receiving a dog feeding signal (indicating that the MCU has died).

After understanding several registers associated with independent watchdog, we will explain how to implement configuration through library functions. Firstly, the key value register IWDG\_ KR, the descriptions of each register are shown in Figure 11.1.1:



Write 0xCCCC in the key value register (IWDG\_KR) to start enabling the independent watchdog; At this point, the counter starts to count down from its reset value 0xFFF. When the counter reaches the end of 0x000, a reset signal (IWDG\_RESET) will be generated. Whenever the key register IWDG\_ WRITED 0xAAAA, IWDG in KR\_ The value in RLR will be reloaded into the counter to avoid a watchdog reset.

IWDG\_ PR and IWDG\_ The RLR register has write protection function. To modify the values of these two registers, you must first report to IWDG\_ Write 0x5555 to the KR register. Writing other values to this register will disrupt the order of operations and the register will be protected again. The reload operation (i.e. writing 0xAAAA) will also activate the write protection function.

There are also two registers, one pre division register (IWDG\_PR), which is used to set the division coefficient of the watchdog clock. Another reload register. This register is used to store the values reloaded into the counter. This register is also a 32-bit register, but only the lower 12 bits are valid.

As long as the above three registers are set accordingly, we can start the independent watchdog of STM32. The startup process can be implemented as follows (the library functions and definitions related to the independent watchdog are distributed in the files stm32f10x\_iwdg. h and stm32f10x\_iwdg. c):

(1) Cancel register write protection (write 0X5555 to IWDG\_KR)

Through this step, we cancel IWDG\_ PR and IWDG\_ RLR write protection allows for later operation of these two registers and setting IWDG\_ PR and IWDG\_ The value of RLR. The implementation function in the library function is:

**IWDG\_ WriteAccessCommand (IWDG\_WriteAccessEnable);**

This function is very simple, as the name suggests, it means to enable/disable write protection, which means to enable/disable write permissions.

(2) The function for setting the pre division coefficient and reload value of an independent watchdog is:

**Void IWDG\_ SetPrescaler (uint8\_t IWDG\_Prescaler)// The function for setting the IWDG pre division value**

and the watchdog's reload value is:

**Void IWDG\_ SetReload (uint16\_t Reload)// Set IWDG reload value**

Setting the frequency division coefficient prer and reload value of the good-looking watchdog can determine the feeding time (i.e. the watchdog overflow time) of the watchdog. The calculation method for this time is:

**Tout=(4 × 2 ^ prer) × Rlr)/40**

Among them, Tout is the watchdog overflow time (in ms); Prer is the pre division value (IWDG\_PR value) of the watchdog clock, with a range of 0 to 7; Rlr is the reload value of the watchdog (the value of IWDG\_RLR); For example, if we set the pre r value to 4 and the rlr value to 625, we can obtain Tout=64 × 625/40=1000ms, so the overflow time of the watchdog is 1s. As long as you write 0XAAAA to IWDG once within one second\_ KR will not cause the watchdog to reset (of course, writing multiple times is also possible). It should be noted that the watchdog clock is not accurate at 40Khz, so it is best not to feed the dog too late, otherwise there may be a watchdog reset.

(3) Heavy load counting value feeding dog (write 0XAAAA to IWDG\_KR)

The function that overloads the count value in the library function is:

**IWDG\_ ReloadCounter()// Reload the IWDG counter according to the value of the IWDG reload register**

By using this sentence, STM32 will be reloaded with IWDG\_ The value of RLR is entered into the watchdog counter. Realize independent gatekeeping

Dog feeding operation.

(4) Start watchdog (write 0XCCCC to IWDG\_KR)

The function to start the independent watchdog in the library function is:

**IWDG\_ Enable ()// Enable IWDG**

Activate the watchdog of STM32 through this sentence. Note that once enabled, IWDG cannot be turned off again! If you want to close it, you can only restart it, and after restarting, you cannot open IWDG. Otherwise, the problem still persists. Therefore, I remind everyone not to open IWDG if it is not used to avoid trouble.

Through the above four steps, we can start the watchdog of STM32. If the watchdog is enabled, it must be fed at certain intervals within the program, otherwise it will cause the program to reset. Taking advantage of this, we will use an LED light in this chapter to refer to

Verify whether the program restarts to verify the independent watchdog of STM32.

After configuring the watchdog, DS0 will remain on, if WK\_ Press the UP button to feed the dog, as long as WK\_ If the UP button is pressed continuously, the watchdog will not reset and DS0 will remain on. Once the watchdog overflow time (Tout) is exceeded and the button is not pressed, it will cause the program to restart, which will cause DS0 to turn off once.

1. **Main program analysis**

The code in wdg. c is as follows:

#include "wdg.h"

//初始化独立看门狗

//prer:分频数:0~7(只有低 3 位有效!)

//分频因子=4\*2^prer.但最大值只能是 256!

//rlr:重装载寄存器值:低 11 位有效.

//时间计算(大概):Tout=((4\*2^prer)\*rlr)/40 (ms).

void IWDG\_Init(u8 prer,u16 rlr) {

IWDG\_WriteAccessCmd(IWDG\_WriteAccess\_Enable); //①使能对寄存器 I 写操作 IWDG\_SetPrescaler(prer); //②设置 IWDG 预分频值:设置 IWDG 预分频值 IWDG\_SetReload(rlr); //②设置 IWDG 重装载值

IWDG\_ReloadCounter(); //③按照 IWDG 重装载寄存器的值重装载 IWDG 计数器 IWDG\_Enable(); //④使能 IWDG

} //喂独立看门狗

void IWDG\_Feed(void) {

IWDG\_ReloadCounter();//reload

}

In the main program, we first initialize the system code, then activate the key input and watchdog. After the watchdog is turned on, Ma Shan lights up LED0 (DS0) and enters a dead cycle waiting for the input of the key. Once WK\_ If there is a button on UP, feed the dog, otherwise wait for the IWDG reset to arrive. This code is easy to understand, as follows:

int main(void) {

delay\_init(); //延时函数初始化

NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_2); //设置 NVIC 中断分组 2 uart\_init(115200); //串口初始化波特率为 115200

LED\_Init(); //初始化与 LED 连接的硬件接口

KEY\_Init(); //按键初始化

delay\_ms(500); //让人看得到灭

IWDG\_Init(4,625); //与分频数为 64,重载值为 625,溢出时间为 1s LED0=0; //点亮 LED0 while(1) {

if(KEY\_Scan(0)==WKUP\_PRESS)

{ IWDG\_Feed(); //如果 WK\_UP 按下,则喂狗

} delay\_ms(10);

};

}

1. **Experimental result**

After downloading the code, you can see DS0 constantly flashing, indicating that the program is constantly resetting, otherwise only DS0 will remain on. At this point, let's try pressing WK continuously\_ By pressing the UP button, you can see that DS0 is constantly on and will not flash again.

**4.Improvement and perfection**

By modifying parameters such as frequency division and overflow time, the feeding and death time of the watchdog can be adjusted, which can make the experiment more suitable for time control in practical applications.