



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Embedded System and Microcomputer Principle

LAB7 Watchdog

2021 Fall
wangq9@mail.sustech.edu.cn



CONTENTS

- 1 IWDG Description
- 2 WWDG Description
- 3 How to Program
- 4 Practice



01

IWDG Description



1. IWDG Description

-- What is watchdog

- 在由单片机构成的微型计算机系统中，由于单片机的工作常常会受到来自外界电磁场的干扰，造成程序的跑飞，而陷入死循环，程序的正常运行被打断，由单片机控制的系统无法继续工作，会造成整个系统的陷入停滞状态，发生不可预料的后果，所以出于对单片机运行状态进行实时监测的考虑，便产生了一种专门用于监测单片机程序运行状态的模块或者芯片，俗称“看门狗” (watchdog)。
- 作用：当系统发生严重错误（如程序进入死循环等）不能恢复的时候， watchdog能够让系统重启。



1. IWDG Description

-- Watchdog of STM32F103

- STM32内置两个看门狗。
- 两个看门狗设备（独立看门狗/窗口看门狗）可以用来检测 and 解决由软件错误引起的故障。
- **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仍有效。IWDG适合应用于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低的场合。
- **窗口看门狗（WWDG）** 由从APB1时钟分频后得到时钟驱动。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操作。WWDG适合那些要求看门狗在精确计时窗口起作用的程序。



1. IWDG Description

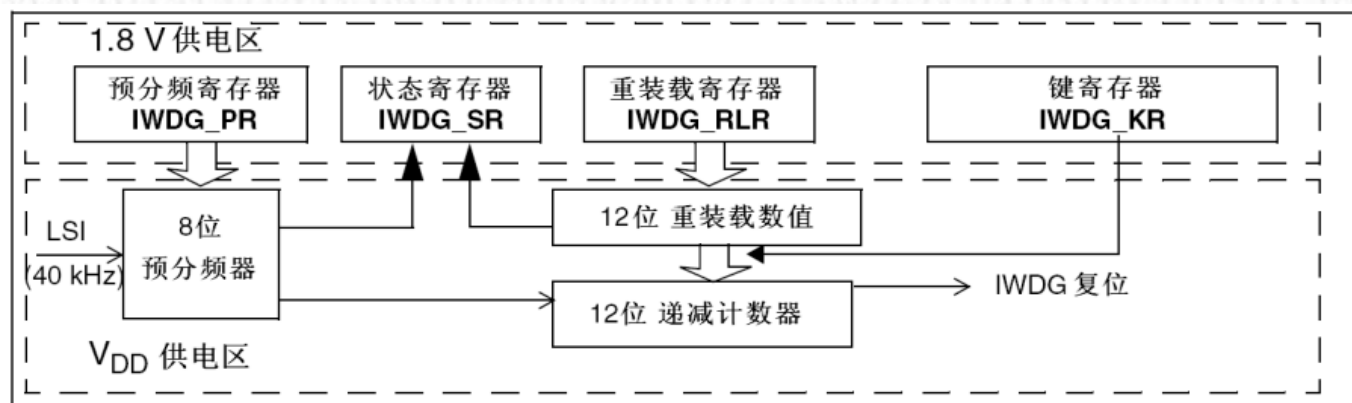
-- IWDG main features

- Independent watchdog (IWDG)
- 自由运行的递减计数器
- 时钟由独立的RC振荡器提供（可在停止和待机模式下工作）
- 看门狗被激活后，则在计数器计数至0x000时产生复位

1. IWDG Description

-- IWDG functional description

- 在键寄存器 (IWDG_KR) 中写入**0xCCCC**，开始启用独立看门狗；此时计数器开始从其复位值**0xFFF**递减计数。当计数器计数到末尾**0x000**时，会产生一个复位信号 (IWDG_RESET)。
- 在IWDG_KR中写入**0xAAAA** (刷新/喂狗)，IWDG_RLR中的值就会被重新加载到计数器，从而避免产生看门狗复位。
- 如果程序异常，就无法正常刷新 (喂狗)，从而系统复位。



看门狗功能处于V_{DD}供电区，即在停机和待机模式时仍能正常工作。

独立看门狗框图



1. IWDG Description

-- IWDG timeout calculation

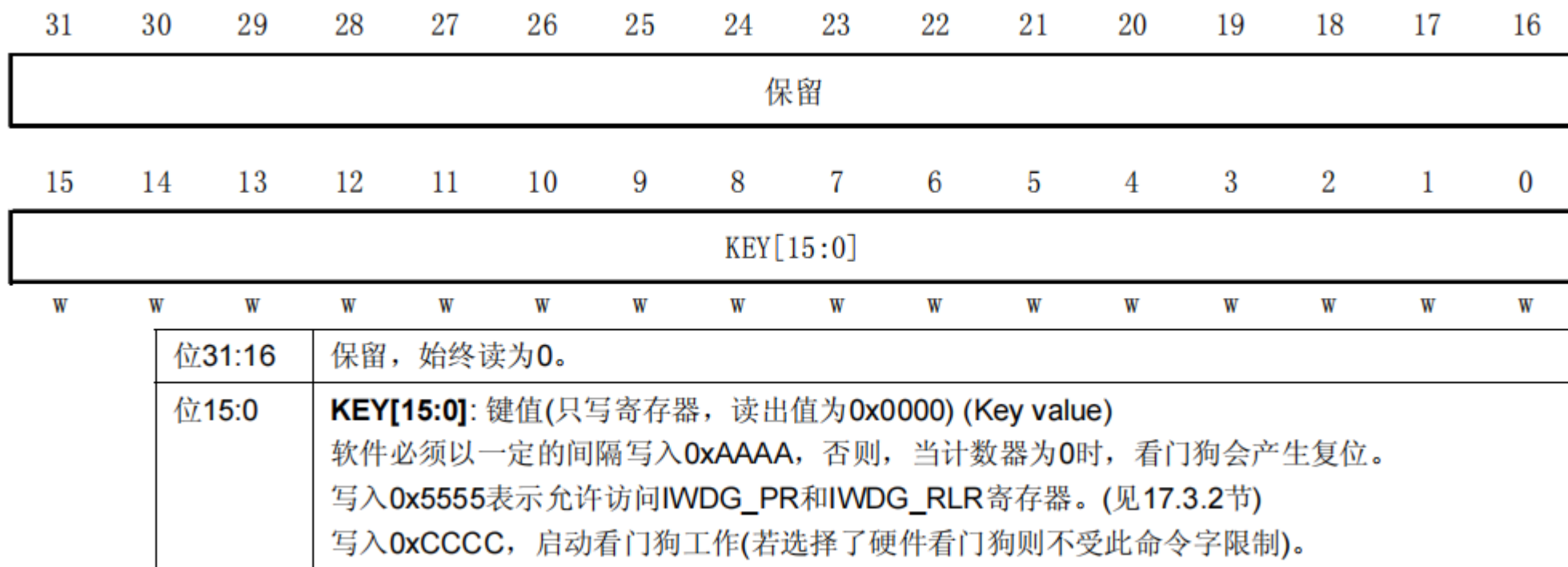
- $T_{out} = presaler * reload / f_{LSI}$
- $f_{LSI} = 40kHz$
- $presaler = 4 * 2^{PR[2:0]}$
- $reload = RLR$
- 最短超时时间：一个看门狗时钟周期
- 最长超时时间：(IWDG_RLR寄存器最大值) * 看门狗时钟周期

预分频系数	PR[2:0]位	最短时间(ms) RL[11:0] = 0x000	最长时间(ms) RL[11:0] = 0xFFFF
/4	0	0.1	409.6
/8	1	0.2	819.2
/16	2	0.4	1638.4
/32	3	0.8	3276.8
/64	4	1.6	6553.6
/128	5	3.2	13107.2
/256	(6或7)	6.4	26214.4

1. IWDG Description

```
-- IWDG_KR
```

- Key register
- 键寄存器

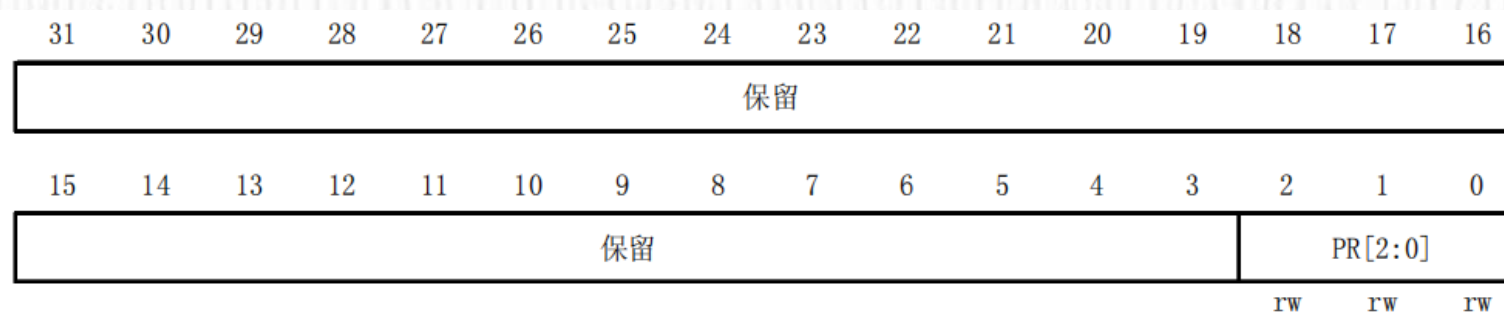




1. IWDG Description

-- IWDG_PR

- Prescaler register
- 预分频寄存器



位31:3	保留，始终读为0。								
位2:0	<p>PR[2:0]: 预分频因子 (Prescaler divider)</p> <p>这些位具有写保护设置，参见17.3.2节。通过设置这些位来选择计数器时钟的预分频因子。要改变预分频因子，IWDG_SR寄存器的PVU位必须为0。</p> <table><tr><td>000: 预分频因子=4</td><td>100: 预分频因子=64</td></tr><tr><td>001: 预分频因子=8</td><td>101: 预分频因子=128</td></tr><tr><td>010: 预分频因子=16</td><td>110: 预分频因子=256</td></tr><tr><td>011: 预分频因子=32</td><td>111: 预分频因子=256</td></tr></table> <p>注意：对此寄存器进行读操作，将从VDD电压域返回预分频值。如果写操作正在进行，则读回的值可能是无效的。因此，只有当IWDG_SR寄存器的PVU位为0时，读出的值才有效。</p>	000: 预分频因子=4	100: 预分频因子=64	001: 预分频因子=8	101: 预分频因子=128	010: 预分频因子=16	110: 预分频因子=256	011: 预分频因子=32	111: 预分频因子=256
000: 预分频因子=4	100: 预分频因子=64								
001: 预分频因子=8	101: 预分频因子=128								
010: 预分频因子=16	110: 预分频因子=256								
011: 预分频因子=32	111: 预分频因子=256								



1. IWDG Description

-- IWDG_RLR

- Reload register
- 重装载寄存器



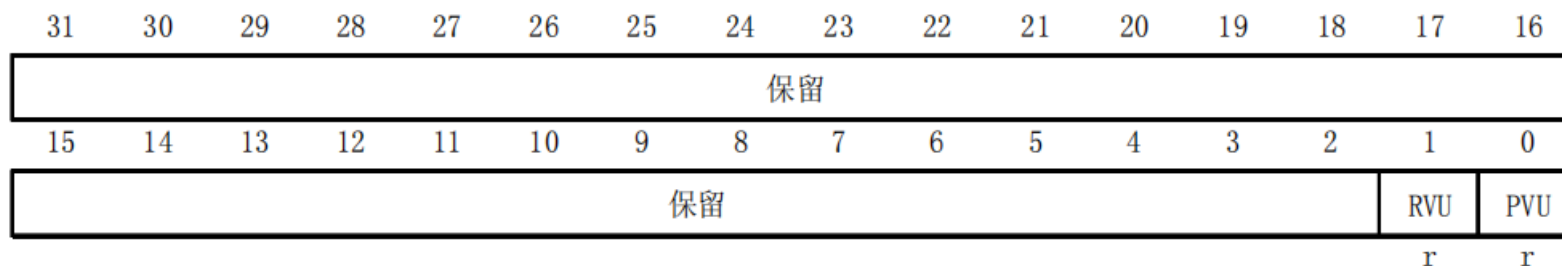
位31:12	保留，始终读为0。
位11:0	RL[11:0]: 看门狗计数器重装载值 (Watchdog counter reload value) 这些位具有写保护功能，参看17.3.2节。用于定义看门狗计数器的重装载值，每当向IWDG_KR寄存器写入0xAAAA时，重装载值会被传送到计数器中。随后计数器从这个值开始递减计数。看门狗超时周期可通过此重装载值和时钟预分频值来计算，参照表83。 只有当IWDG_SR寄存器中的RVU位为0时，才能对此寄存器进行修改。 注：对此寄存器进行读操作，将从VDD电压域返回预分频值。如果写操作正在进行，则读回的值可能是无效的。因此，只有当IWDG_SR寄存器的RVU位为0时，读出的值才有效。



1. IWDG Description

-- IWDG_SR

- Status register
- 状态寄存器



位31:2	保留。
位1	RVU : 看门狗计数器重载值更新 (Watchdog counter reload value update) 此位由硬件置'1'用来指示重载值的更新正在进行中。当在VDD域中的重载更新结束后, 此位由硬件清'0'(最多需5个40kHz的RC周期)。重载值只有在RVU位被清'0'后才可更新。
位0	PVU : 看门狗预分频值更新 (Watchdog prescaler value update) 此位由硬件置'1'用来指示预分频值的更新正在进行中。当在VDD域中的预分频值更新结束后, 此位由硬件清'0'(最多需5个40kHz的RC周期)。预分频值只有在PVU位被清'0'后才可更新。

注: 如果在应用程序中使用了多个重载值或预分频值, 则必须在RVU位被清除后才能重新改变预装载值, 在PVU位被清除后才能重新改变预分频值。然而, 在预分频和/或重载值更新后, 不必等待RVU或PVU复位, 可继续执行下面的代码。(即是在低功耗模式下, 此写操作仍会被继续执行完成。)



1. IWDG Description

-- IWDG configuration steps

- 取消寄存器写保护
- 设置独立看门狗的预分频系数，确定时钟
- 设置看门狗重装载值，确定溢出时间:
- 使能看门狗
- 应用程序不停刷新（喂狗）



02

WWDG Description



2. WWDG Description

-- WWDG introduction

- 窗口看门狗通常被用来监测，由外部干扰或不可预见的逻辑条件造成的应用程序背离正常的运行序列而产生的软件故障。
- 除非递减计数器的值在T6位变成0前被刷新，看门狗电路在达到预置的时间周期时，会产生一个MCU复位。
- 在递减计数器达到窗口寄存器数值之前，如果7位的递减计数器数值(在控制寄存器中)被刷新，那么也将产生一个MCU复位。
- 递减计数器需要在一个有限的时间窗口中被刷新。



2. WWDG Description

-- WWDG main features

- Window watchdog (WWDG)
- 可编程的自由运行递减计数器
- 条件复位
 - 当递减计数器的值小于0x40, (若看门狗被启动) 则产生复位。
 - 当递减计数器在窗口外被重新装载, (若看门狗被启动) 则产生复位。
- 如果启动了看门狗并且允许中断, 当递减计数器等于0x40时产生早期唤醒中断(EWI), 它可以被用于重装载计数器以避免WWDG复位。



2. WWDG Description

-- WWDG functional description

- 看门狗被启动，递减计数器从0x40翻转到0x3F时，产生一个复位。
- 软件在计数器值大于窗口寄存器中的数值时重新装载计数器，将产生一个复位。
- 应用程序在正常运行过程中必须定期地写入WWDG_CR寄存器以防止MCU发生复位。
- 只有当计数器值小于窗口寄存器的值时，才能进行写操作。
- 储存在WWDG_CR寄存器中的数值必须在0xFF和0xC0之间：
 - 启动看门狗：在系统复位后，看门狗总是处于关闭状态，设置WWDG_CR寄存器的WDGA位能够开启看门狗，随后它不能再被关闭，除非发生复位。
 - 控制递减计数器



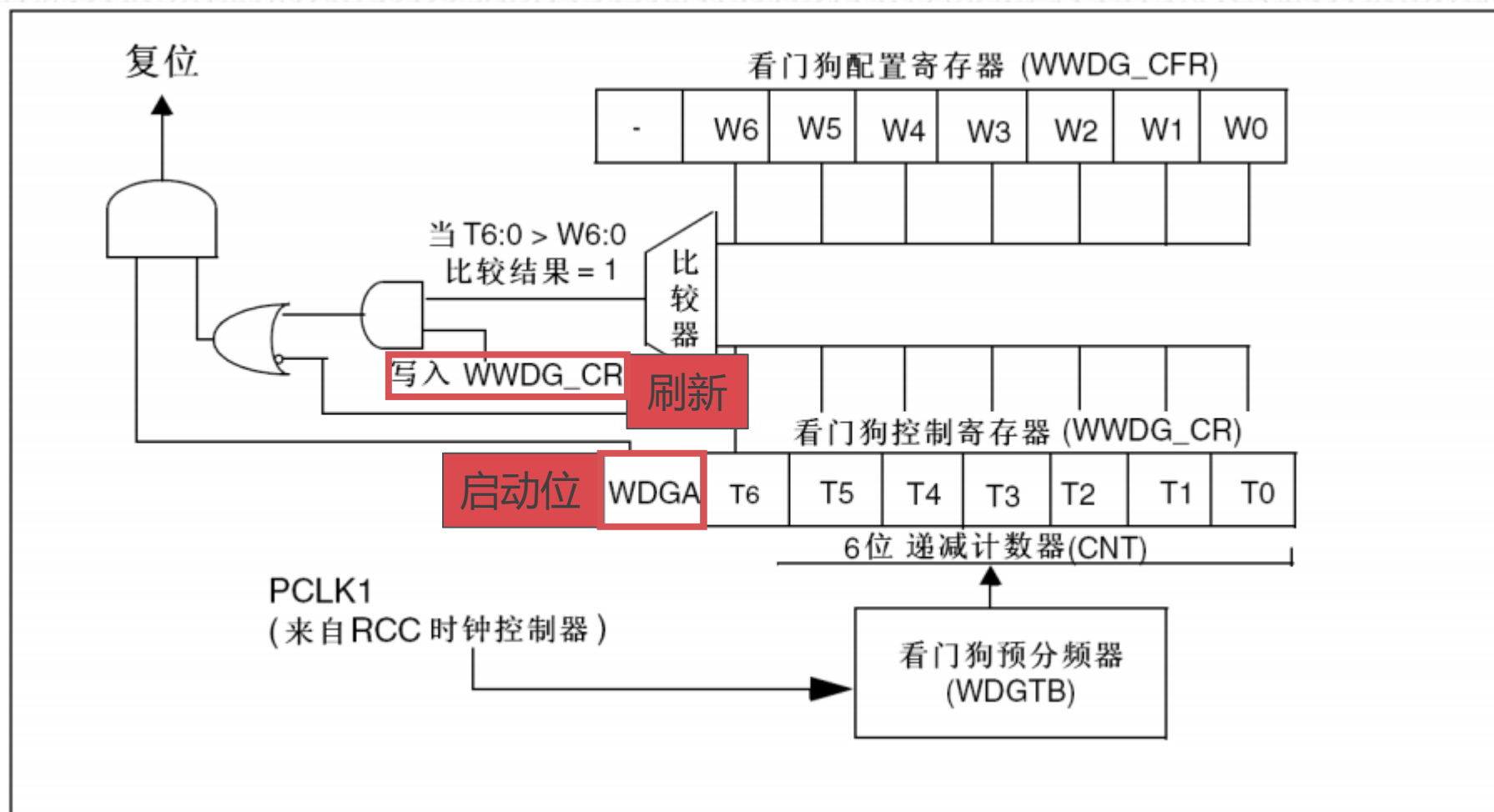
2. WWDG Description

-- WWDG functional description(continued)

- 控制递减计数器
 - 递减计数器处于自由运行状态，即使看门狗被禁止，递减计数器仍继续递减计数。
 - 当看门狗被启用时，T6位必须被设置，以防止立即产生一个复位。
 - T[5:0]位包含了看门狗产生复位之前的计时数目；复位前的延时时间在一个最小值和一个最大值之间变化，这是因为写入WWDG_CR寄存器时，预分频值是未知的。
 - WWDG_CFR中包含窗口的上限值：要避免产生复位，递减计数器必须在其值小于窗口寄存器的数值并且大于0x3F时被重新装载。
 - 另一个重装载计数器的方法是利用早期唤醒中断(EWI)。设置WWDG_CFR寄存器中的EWI位开启该中断。当递减计数器到达0x40时，则产生此中断，相应的中断服务程序(ISR)可以用来加载计数器以防止WWDG复位。在WWDG_SR寄存器中写'0'可以清除该中断。

2. WWDG Description

-- WWDG functional description(continued)



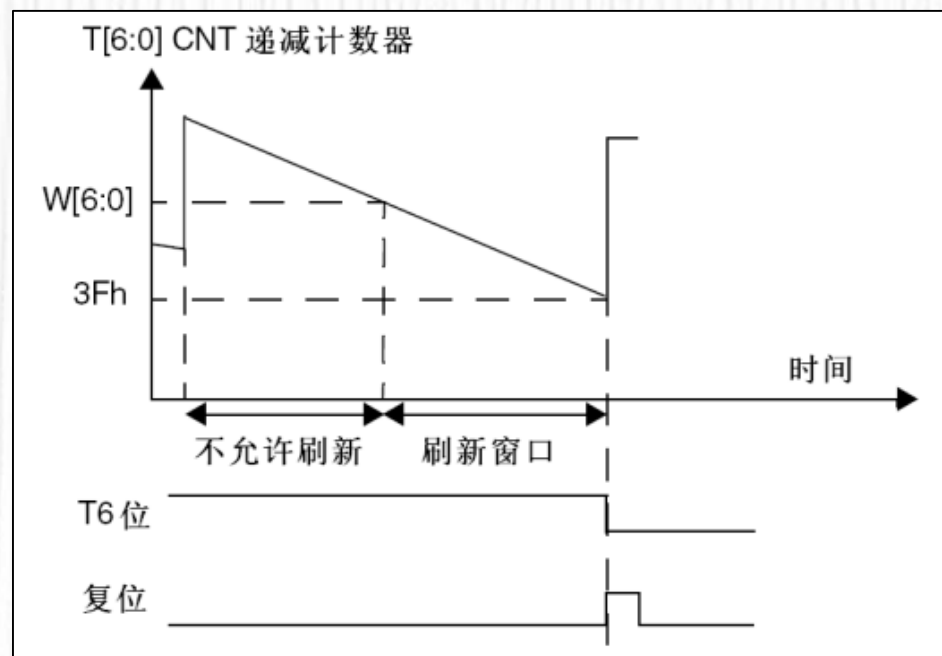
窗口看门狗框图



2. WWDG Description

-- WWDG timeout calculation

- $T_{WWDG} = T_{pclk1} * 4096 * 2^{WDGTB} * (T[5:0] + 1)$
 - T_{pclk1} : APB1时钟周期



窗口看门狗时序图

PCLK1 = 36MHz时的最小-最大超时值

Prescaler	WDGTB	Min timeout value	Max timeout value
1	0	113 μ s	7.28 ms
2	1	227 μ s	14.56 ms
4	2	455 μ s	29.12 ms
8	3	910 μ s	58.25 ms



2. WWDG Description

-- WWDG_CR

- Control register
- 控制寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留								WDGA	T6	T5	T4	T3	T2	T1	T0
								rS	rW	rW	rW	rW	rW	rW	rW

位31:8	保留。
位7	WDGA : 激活位 (Activation bit) 此位由软件置'1', 但仅能由硬件在复位后清'0'。当WDGA=1时, 看门狗可以产生复位。 0 : 禁止看门狗 1 : 启用看门狗
位6:0	T[6:0] : 7位计数器(MSB至LSB) (7-bit counter) 这些位用来存储看门狗的计数器值。每 $(4096 \times 2^{\text{WDGTB}})$ 个PCLK1周期减1。当计数器值从40h变为3Fh时(T6变成0), 产生看门狗复位。

```
-- WWDG_CFR
```

- Configuration register
- 配置寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留						EWI	WDG TB1	WDG TB0	W6	W5	W4	W3	W2	W1	W0
						r/s	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

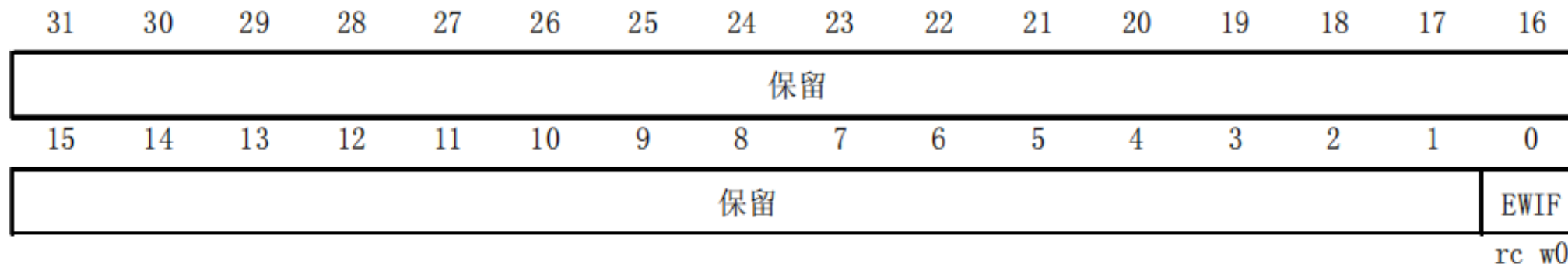
位31:8	保留。
位9	EWI : 提前唤醒中断 (Early wakeup interrupt) 此位若置'1', 则当计数器值达到40h, 即产生中断。 此中断只能由硬件在复位后清除。
位8:7	WDGTB[1:0] : 时基 (Timer base) 预分频器的时基可以设置如下: 00: CK计时器时钟(PCLK1除以4096)除以1 01: CK计时器时钟(PCLK1除以4096)除以2 10: CK计时器时钟(PCLK1除以4096)除以4 11: CK计时器时钟(PCLK1除以4096)除以8
位6:0	W[6:0] : 7位窗口值 (7-bit window value) 这些位包含了用来与递减计数器进行比较用的窗口值。



2. WWDG Description

-- WWDG_SR

- Status register
- 状态寄存器



位31:1	保留。
位0	EWIF: 提前唤醒中断标志 (Early wakeup interrupt flag) 当计数器值达到40h时，此位由硬件置'1'。它必须通过软件写'0'来清除。对此位写'1'无效。若中断未被使能，此位也会被置'1'。



2. WWDG Description

-- WWDG configuration steps

- 使能看门狗时钟
- 设置分频系数
- 设置上窗口值
- 开启提前唤醒中断并分组（可选）
- 使能看门狗
- 刷新看门狗T[6:0]的值（喂狗）
- 编写中断服务函数



03

How to Program



3. How to Program -- IWDG

- Our Goal
 - Use KEY1 interrupt to refresh IWDG
 - If the program is reset, the variable output restarts to count from 0
 - If IWDG is refreshed before the downcounter has reached 0, the variable output counts increasingly by 1 each clock



3. How to Program -- IWDG

- Configure GPIO
 - Set PA15(KEY1) as external interrupt source

Group By Peripherals

GPIO RCC SYS USART NVIC

Search Signals

Search (Ctrl+F) ☐ Show only Modified Pins

Pin Name	Signal on Pin	GPIO outp...	GPIO mode	GPIO P...	Maximum ...	User Label	Modified
PA15	n/a	n/a	External Interrupt ...	Pull-up	n/a	KEY1	<input checked="" type="checkbox"/>

GPIO mode: External Interrupt Mode with Falling edge trigger detection

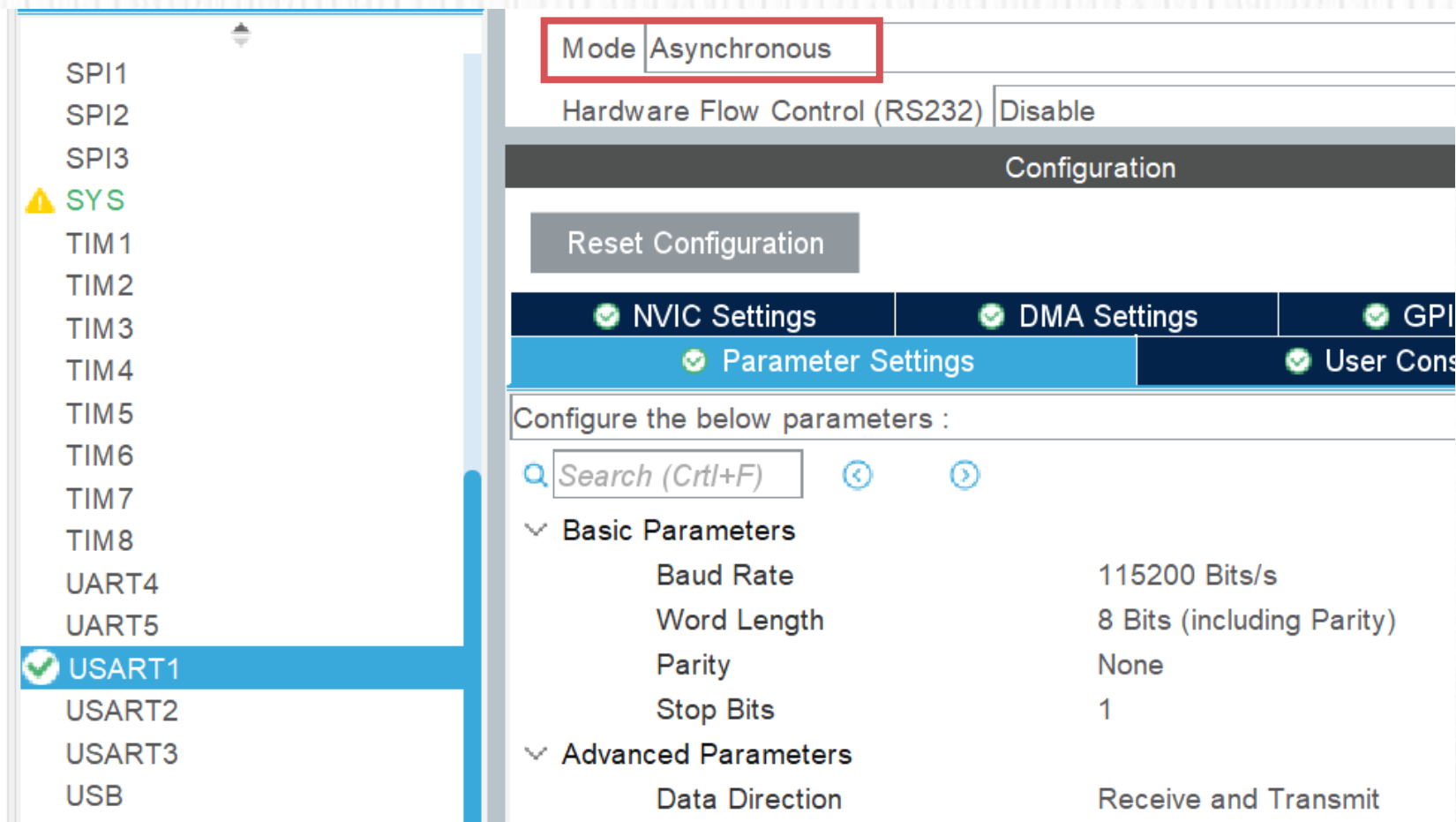
GPIO Pull-up/Pull-down: Pull-up

User Label: KEY1



3. How to Program -- IWDG

- Configure USART1
 - Set the USART1 as asynchronous mode





3. How to Program -- IWDG

- Configure NVIC
 - Enable EXTI line[15:10] interrupt

Configuration

✓ NVIC ✓ Code generation

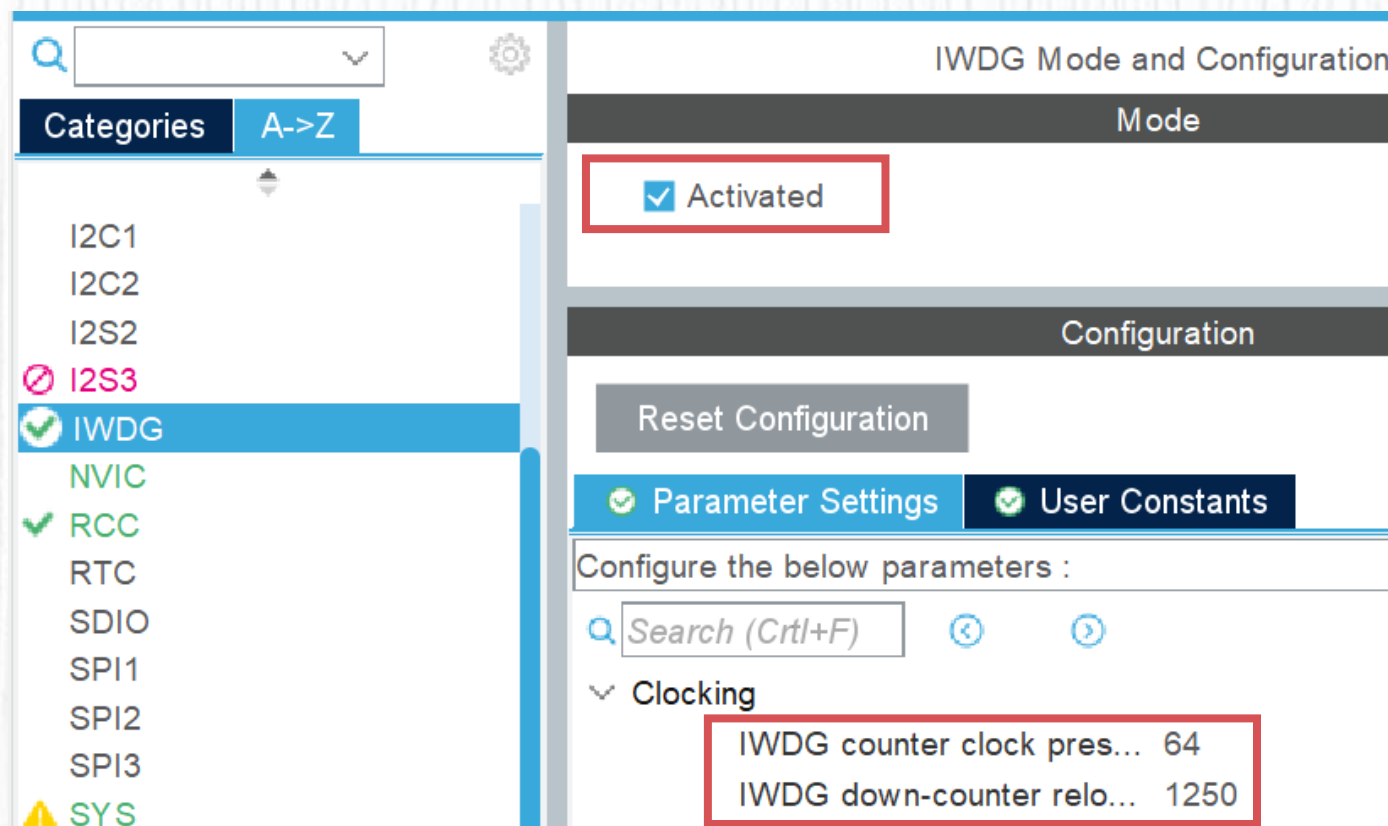
Priority Group 2 bits for pre-emption priority... ☐ Sort by Preemption Priority and Sub Priority ☐ Sort by interrupts name

Search Show available interrupts ☒ Force DMA channel

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0
Memory management fault	<input checked="" type="checkbox"/>	0	0
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0
Debug monitor	<input checked="" type="checkbox"/>	0	0
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0	0
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
USART1 global interrupt	<input type="checkbox"/>	0	0
EXTI line[15:10] interrupts	<input checked="" type="checkbox"/>	1	0

3. How to Program -- IWDG

- Configure IWDG
 - Active the IWDG
 - Set the prescaler and reload value





3. How to Program -- IWDG

- Some functions we used
 - Call HAL_IWDG_Refresh() function to refresh the IWDG (feed dog)

```
HAL_StatusTypeDef HAL_IWDG_Refresh(IWDG_HandleTypeDef *hiwdg)
{
    /* Reload IWDG counter with value defined in the reload register */
    __HAL_IWDG_RELOAD_COUNTER(hiwdg);

    /* Return function status */
    return HAL_OK;
}
```

3. How to Program -- IWDG

- Configure the external interrupt, and refresh the IWDG by pressing the KEY1 in **main.c** or **stm32f1xx_it.c**

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    switch (GPIO_Pin) {
        case KEY1_Pin:
            if (HAL_GPIO_ReadPin(KEY1_GPIO_Port, KEY1_Pin) == GPIO_PIN_RESET){
                HAL_IWDG_Refresh(&hiwdg);
            }
            break;
        default:
            break;
    }
}
```




3. How to Program -- IWDG

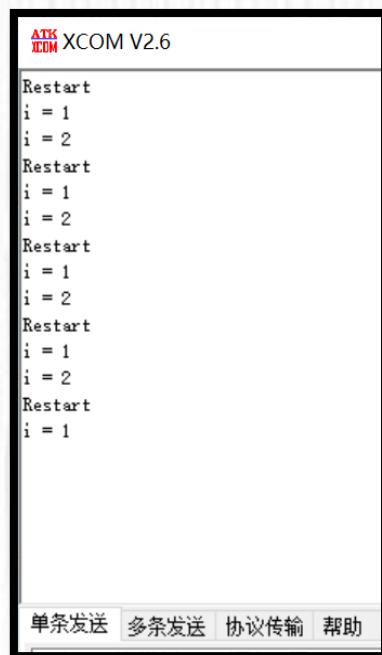
- Set a variable in **main.c** to show if the program is reset
- If the program is reset, the variable i will reset to 0

```
int i = 0;
unsigned char msg[100];
HAL_UART_Transmit(&huart1, "Restart\r\n", 9, HAL_MAX_DELAY);
while (1)
{
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
    i++;
    sprintf(msg, "i = %d\r\n", i);
    HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    HAL_Delay(1000);
}
/* USER CODE END 3 */
```



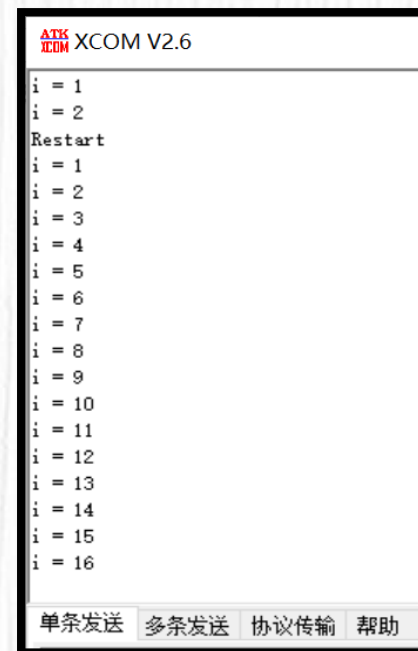
3. How to Program -- IWDG

- When the program is reset, the value of variable *i* comes back to 0, 1, 2 (shown as Fig.1)
- When press the KEY1 continuously, the value of variable *i* will increase (shown as Fig.2)



```
ATX XCOM V2.6
Restart
i = 1
i = 2
Restart
i = 1
i = 2
Restart
i = 1
i = 2
Restart
i = 1
i = 2
Restart
i = 1
```

Fig.1 The program resets periodically



```
ATX XCOM V2.6
i = 1
i = 2
Restart
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
i = 11
i = 12
i = 13
i = 14
i = 15
i = 16
```

Fig.2 The program runs continuously



3. How to Program -- WWDG

- Our Goal
 - Use EWI to refresh WWDG
 - Use KEY1 interrupt to refresh WWDG
 - If the program is reset, the variable output restarts to count from 0
 - If WWDG is refreshed before the downcounter has reached the window register value, the variable output restarts to count from 0
 - If WWDG is refreshed when the downcounter is between the window register value and 0x3F, the variable output counts increasingly by 1 each clock



3. How to Program -- WWDG

- Configure GPIO
 - Set PA15(KEY1) as external interrupt source

The screenshot shows the STM32CubeMX software interface. On the left, a tree view lists various peripherals, with 'GPIO' selected. The main area displays the 'Group By Peripherals' tab, showing a table of pins. The pin PA15 is highlighted, and its configuration is shown in the bottom right. The configuration includes setting the GPIO mode to 'External Interrupt Mode with Falling edge trigger detection', the pull-up/pull-down to 'Pull-up', and the user label to 'KEY1'.

Group By Peripherals

GPIO RCC SYS USART NVIC

Search Signals

Search (Ctrl+F) ☐ Show only Modified Pins

Pin Name	Signal on Pin	GPIO outp...	GPIO mode	GPIO P...	Maximum ...	User Label	Modified
PA15	n/a	n/a	External Interrupt ...	Pull-up	n/a	KEY1	<input checked="" type="checkbox"/>

GPIO mode: External Interrupt Mode with Falling edge trigger detection

GPIO Pull-up/Pull-down: Pull-up

User Label: KEY1



3. How to Program -- WWDG

- Configure USART1
 - Set the USART1 as asynchronous mode

The screenshot shows the STM32CubeMX configuration interface. On the left, a list of peripherals includes SPI1, SPI2, SPI3, SYS (with a warning icon), TIM1, TIM2, TIM3, TIM4, TIM5, TIM6, TIM7, TIM8, UART4, UART5, USART1 (selected with a green checkmark), USART2, USART3, and USB. The right panel displays the configuration for the selected peripheral, USART1. At the top, the 'Mode' is set to 'Asynchronous', which is highlighted with a red box. Below this, 'Hardware Flow Control (RS232)' is set to 'Disable'. A 'Configuration' section contains a 'Reset Configuration' button. Below that are tabs for 'NVIC Settings', 'DMA Settings', 'GPIO Settings', 'Parameter Settings' (which is active), and 'User Constraints'. The 'Parameter Settings' tab shows a search bar and a list of parameters to configure. The 'Basic Parameters' section includes: Baud Rate (115200 Bits/s), Word Length (8 Bits (including Parity)), Parity (None), and Stop Bits (1). The 'Advanced Parameters' section includes: Data Direction (Receive and Transmit).

Mode
Asynchronous

Hardware Flow Control (RS232) | Disable

Configuration

Reset Configuration

✓ NVIC Settings | ✓ DMA Settings | ✓ GPIO Settings | ✓ Parameter Settings | ✓ User Constraints

Configure the below parameters :

Search (Ctrl+F) ⏪ ⏩

▼ Basic Parameters

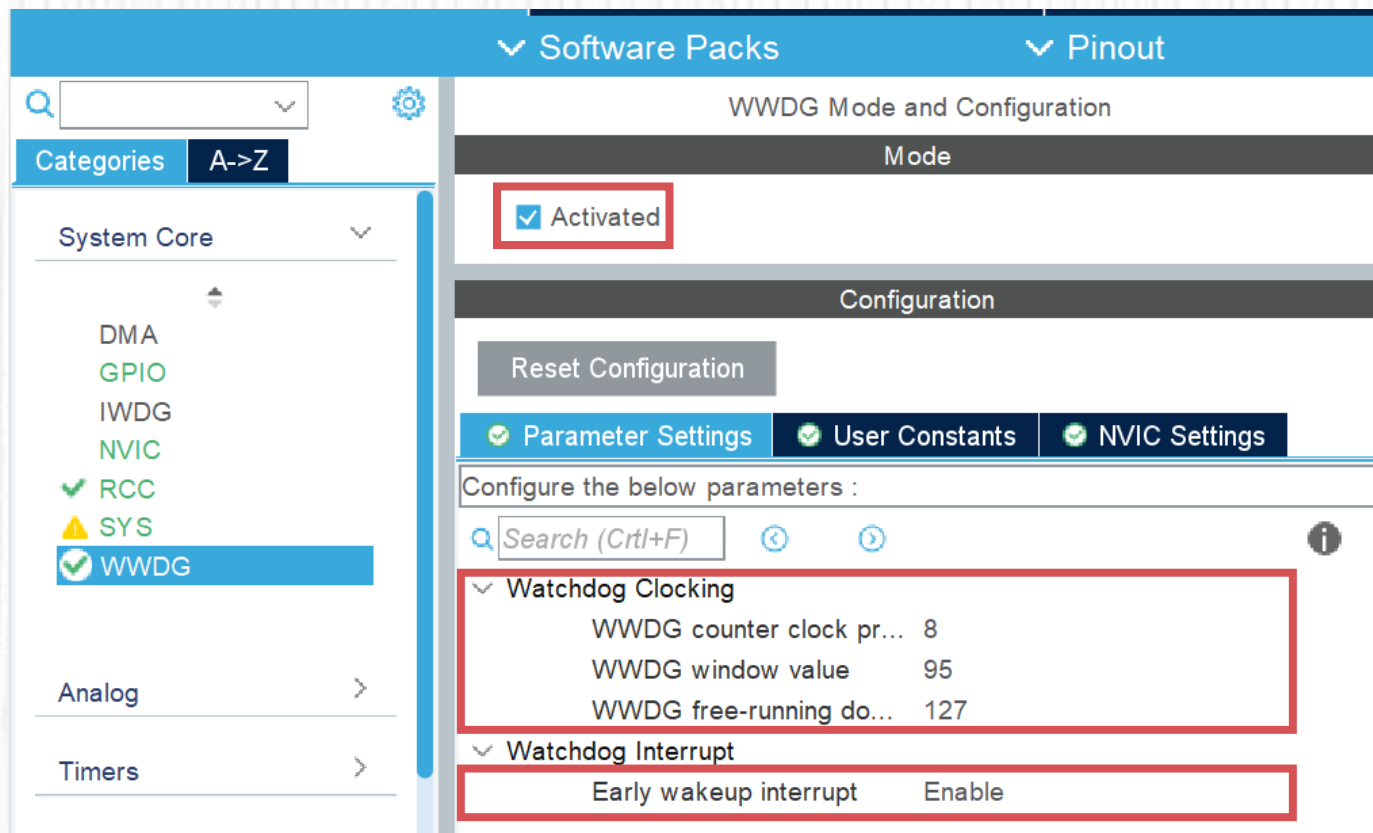
Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

▼ Advanced Parameters

Data Direction	Receive and Transmit
----------------	----------------------

3. How to Program -- WWDG

- Configure WWDG
 - Active the WWDG
 - Set the WWDG parameters, enable EWI





3. How to Program -- WWDG

- Configure NVIC
 - Enable EXTI line[15:10] interrupt and window watchdog interrupt

System Core

DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG

Analog

Timers

Connectivity

Multimedia

Computing

NVIC Code generation

Search Show available interrupts Force DMA channels Interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0
Memory management fault	<input checked="" type="checkbox"/>	0	0
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0
Debug monitor	<input checked="" type="checkbox"/>	0	0
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0	0
Window watchdog interrupt	<input checked="" type="checkbox"/>	1	0
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
USART1 global interrupt	<input type="checkbox"/>	0	0
EXTI line[15:10] interrupts	<input checked="" type="checkbox"/>	1	0



3. How to Program -- WWDG

- Some functions we used
 - Call HAL_WWDG_Refresh() function to refresh the WWDG (feed dog)

```
HAL_StatusTypeDef HAL_WWDG_Refresh(WWDG_HandleTypeDef *hwwdg)
{
    /* Write to WWDG CR the WWDG Counter value to refresh with */
    WRITE_REG(hwwdg->Instance->CR, (hwwdg->Init.Counter));

    /* Return function status */
    return HAL_OK;
}
```



3. How to Program -- WWDG

- When the counter is 0x40, the EWI will be triggered, it will call HAL_WWDG_EarlyWakeupCallback(), which is a weak function, we should reimplement it in **main.c** or **stm32f1xx_it.c**
- In this demo, we refresh the WWDG in EWI callback function, but in real case, we should record some critical informations, for when EWI is triggered, the system has software fault already.
- Let the interrupt finish in 1 cycle, because the interrupt is triggered while the MCU is reseted when the counter is 0x3F.

```
void HAL_WWDG_EarlyWakeupCallback(WWDG_HandleTypeDef* hwwdg)
{
    HAL_WWDG_Refresh(hwwdg);
}
```


3. How to Program -- WWDG

- Configure the external interrupt, and refresh the WWDG by pressing the KEY1 in **main.c** or **stm32f1xx_it.c**

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    switch (GPIO_Pin) {
        case KEY1_Pin:
            if (HAL_GPIO_ReadPin(KEY1_GPIO_Port, KEY1_Pin) == GPIO_PIN_RESET){
                HAL_WWDG_Refresh(&hwwdg);
            }
            break;
        default:
            break;
    }
}
```



3. How to Program -- WWDG

- Set a variable in **main.c** to show if the program is reset
- If the program is reset, the variable i will reset to 0

```
int i = 0;
unsigned char msg[100];
HAL_UART_Transmit(&huart1, "Restart\r\n", 9, HAL_MAX_DELAY);
while (1)
{
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
    i++;
    sprintf(msg, "i = %d\r\n", i);
    HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    HAL_Delay(1000);
}
/* USER CODE END 3 */
```

3. How to Program -- WWDG

- When the program is reset, the value of variable *i* comes back to 0, 1 (shown as Fig.3)
- When the program runs continuously, the value of variable *i* will increase (shown as Fig.4)

```
ATK XCOM V2.6
i = 23
i = 24
Restart
i = 1
Restart
i = 1
Restart
i = 1
Restart
i = 1
Restart
i = 1
i = 2
i = 3
Restart
i = 1
Restart
i = 1
```

Fig.3 The program resets periodically

```
ATK XCOM V2.6
i = 1
Restart
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
i = 11
i = 12
i = 13
i = 14
i = 15
```

Fig.4 The program runs continuously



04

Practice

4. Practice

- Run the IWDG demo on MiniSTM32 board
- Tell what is the timeout of IWDG
- Run the WWDG demo on MiniSTM32 board
- Tell the time range to refresh WWDG without resetting it
- Complete assignment 1 on Sakai site and submit the whole project before DDL (31st, Oct)