



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Embedded System and Microcomputer Principle

LAB9 Watchdog

2024 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

- A watchdog timer, usually simplified as watchdog, is an electronic timer that used to detect the device fault and reset the device to recover it when there is a fault.
- It is widely used in embedded systems.
- There is a down counter in a watchdog timer. The system will restart when the counter counts to a reload value.
- In normal case, a system should refresh the counter of watchdog periodically to maintain the system not to be restart.
- If the system has fault and cannot refresh the counter periodically, it will count to reload value and restart the system.



1. IWDG Description

-- Watchdog of STM32F103

- The STM32F10xxx have two embedded watchdogs peripherals which offer a combination of high safety level, timing accuracy and flexibility of use.
- Both watchdog peripherals (Independent and Window) serve to detect and resolve malfunctions due to software failure, and to trigger system reset or an interrupt (window watchdog only) when the counter reaches a given timeout value.



1. IWDG Description

-- Watchdog of STM32F103(continued)

- The independent watchdog (**IWDG**) is clocked by its own dedicated low-speed clock (**LSI**) and thus stays active even if the main clock fails.
- The IWDG is best suited to applications which require the watchdog to run as a totally independent process outside the main application, but have lower timing accuracy constraints.
- The window watchdog (**WWDG**) clock is prescaled from the **APB1** clock and has a configurable time-window that can be programmed to detect abnormally late or early application behavior.
- The WWDG is best suited to applications which require the watchdog to react within an accurate timing window.



1. IWDG Description

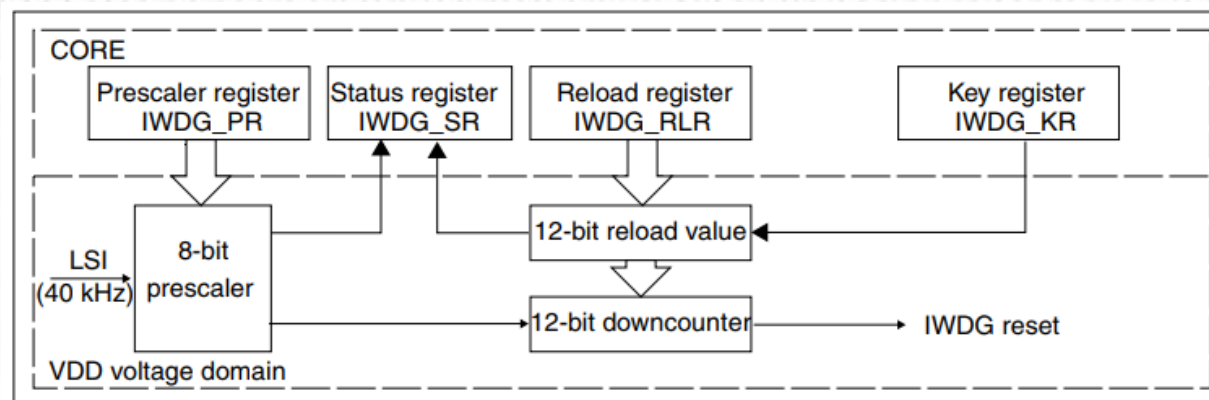
-- IWDG main features

- Independent watchdog (IWDG)
- Free-running downcounter
- clocked from an independent RC oscillator (can operate in Standby and Stop modes)
- Reset (if watchdog activated) when the downcounter value of 0x000 is reached

1. IWDG Description

-- IWDG functional description

- When the IWDG is started by writing the value 0xCCCC in the Key register(IWDG_KR), the counter starts counting down from the reset value of 0xFFF. When it reaches the end of count value (0x000) a reset signal is generated (IWDG reset).
- Whenever the key value 0xAAAA is written in the IWDG_KR register, the IWDG_RLR value is reloaded in the counter and the watchdog reset is prevented.



Independent watchdog block diagram

1. IWDG Description

-- IWDG timeout calculation

- $T_{out} = \text{presaler} * \text{reload} / f_{LSI}$
- $f_{LSI} = 40\text{kHz}$, $\text{presaler} = 4 * 2^{PR[2:0]}$, $\text{reload} = \text{RLR}$
- Minimum timeout: one watchdog clock cycle
- Maximum timeout: (maximum value of IWDG_RLR register) * watchdog clock cycle

Prescaler divider	PR[2:0] bits	Min timeout (ms) RL[11:0]= 0x000	Max timeout (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 (or 7)	6.4	26214.4

Min/max IWDG timeout period at 40kHz (LSI)



1. IWDG Description

-- IWDG_KR

- Key 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
Reserved																KEY[15:0]															
																W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **KEY[15:0]**: Key value (write only, read 0000h)

These bits must be written by software at regular intervals with the key value AAAAh, otherwise the watchdog generates a reset when the counter reaches 0.

Writing the key value 5555h to enable access to the IWDG_PR and IWDG_RLR registers (see [Section 19.3.2](#))

Writing the key value CCCCh starts the watchdog (except if the hardware watchdog option is selected)



1. IWDG Description

-- IWDG_PR

- Prescaler 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
Reserved																													PR[2:0]		
																													r/w	r/w	r/w

Bits 31:3 Reserved, must be kept at reset value.

Bits 2:0 **PR[2:0]**: Prescaler divider

These bits are write access protected see [Section 19.3.2](#). They are written by software to select the prescaler divider feeding the counter clock. PVU bit of IWDG_SR must be reset in order to be able to change the prescaler divider.

000: divider /4

001: divider /8

010: divider /16

011: divider /32

100: divider /64

101: divider /128

110: divider /256

111: divider /256



1. IWDG Description

-- IWDG_RLR

- Reload 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
Reserved																				RL[11:0]											
																				rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **RL[11:0]**: Watchdog counter reload value

These bits are write access protected see [Section 19.3.2](#). They are written by software to define the value to be loaded in the watchdog counter each time the value AAAAh is written in the IWDG_KR register. The watchdog counter counts down from this value. The timeout period is a function of this value and the clock prescaler. Refer to [Table 96](#).

The RVU bit in the IWDG_SR register must be reset in order to be able to change the reload value.

Note: Reading this register returns the reload value from the VDD voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing on this register. For this reason the value read from this register is valid only when the RVU bit in the IWDG_SR register is reset.



1. IWDG Description

-- IWDG_SR

- Status 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
Reserved																												RVU	PVU		
																												r	r		

Bits 31:2 Reserved, must be kept at reset value.

Bit 1 **RVU**: Watchdog counter reload value update

This bit is set by hardware to indicate that an update of the reload value is ongoing. It is reset by hardware when the reload value update operation is completed in the V_{DD} voltage domain (takes up to 5 RC 40 kHz cycles).

Reload value can be updated only when RVU bit is reset.

Bit 0 **PVU**: Watchdog prescaler value update

This bit is set by hardware to indicate that an update of the prescaler value is ongoing. It is reset by hardware when the prescaler update operation is completed in the V_{DD} voltage domain (takes up to 5 RC 40 kHz cycles).

Prescaler value can be updated only when PVU bit is reset.



1. IWDG Description

-- IWDG configuration steps

- Cancel register write protection
- Set the prescaled coefficient of the independent watchdog to determine the clock
- Set the reload value to determine the overflow time
- Enable watchdog
- The application keeps refreshing (feeding the dog)



02

WWDG Description



2. WWDG Description

-- Window Watchdog of STM32F103

- The STM32F10xxx have two embedded watchdogs peripherals which offer a combination of high safety level, timing accuracy and flexibility of use.
- The window watchdog (**WWDG**) clock is prescaled from the **APB1** clock and has a configurable time-window that can be programmed to detect abnormally late or early application behavior.
- The WWDG is best suited to applications which require the watchdog to react within an accurate timing window.



2. WWDG Description

-- WWDG introduction

- The window watchdog is used to detect the occurrence of a software fault, usually generated by external interference or by unforeseen logical conditions, which causes the application program to abandon its normal sequence.
- The watchdog circuit generates an MCU reset on expiry of a programmed time period, unless the program refreshes the contents of the downcounter before the T6 bit becomes cleared.
- An MCU reset is also generated if the 7-bit downcounter value (in the control register) is refreshed before the downcounter has reached the window register value. This implies that the counter must be refreshed in a limited window.



2. WWDG Description

-- WWDG main features

- Window watchdog (WWDG)
- Programmable free-running downcounter
- Conditional reset
 - Reset (if watchdog activated) when the downcounter value becomes less than 0x40
 - Reset (if watchdog activated) if the downcounter is reloaded outside the window
- Early wakeup interrupt (EWI): triggered (if enabled and the watchdog activated) when the downcounter is equal to 0x40.



2. WWDG Description

-- WWDG functional description

- If the watchdog is activated (the WDGA bit is set in WWDG_CR register) and when the 7-bit downcounter (T[6:0] bits) rolls over from 0x40 to 0x3F (T6 becomes cleared), it initiates a reset.
- If the software reloads the counter while the counter is greater than the value stored in window register, then a reset is generated.
- The application program must write in the WWDG_CR register at regular intervals during normal operation to prevent an MCU reset. This operation must occur only when the counter value is lower than the window register value. The value to be stored in the WWDG_CR register must be between 0xFF and 0xC0:
 - Enabling the watch
 - Controlling the downcounter



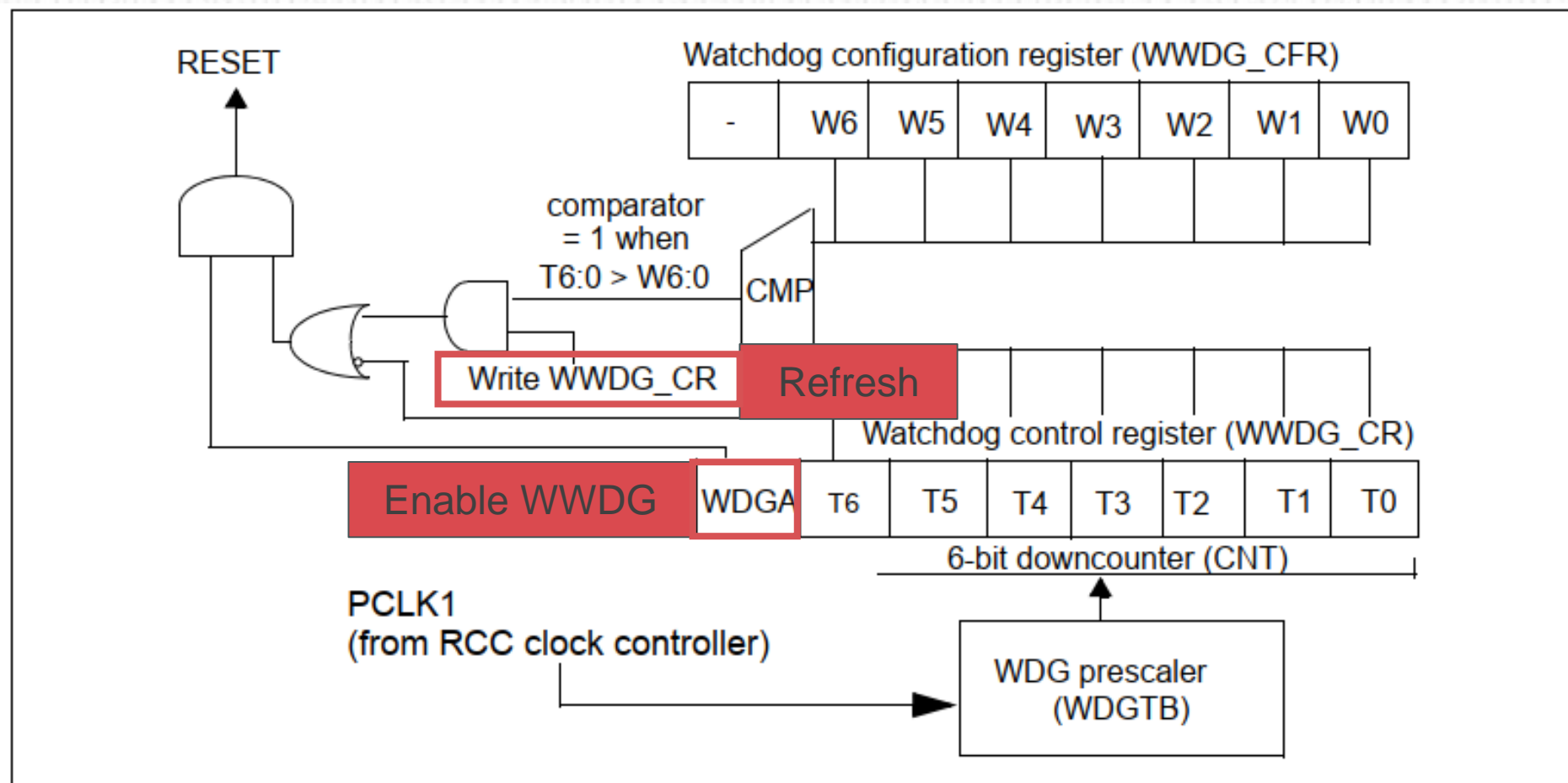
2. WWDG Description

-- WWDG functional description(continued)

- Controlling the downcounter
 - This downcounter is free-running: It counts down even if the watchdog is disabled.
 - When the watchdog is enabled, the T6 bit must be set to prevent generating an immediate reset.
 - The T[5:0] bits contain the number of increments which represents the time delay before the watchdog produces a reset. The timing varies between a minimum and a maximum value due to the unknown status of the prescaler when writing to the WWDG_CR register.
 - The Configuration register (WWDG_CFR) contains the high limit of the window: To prevent a reset, the downcounter must be reloaded when its value is lower than the window register value and greater than 0x3F.
 - Note: The T6 bit can be used to generate a software reset (the WDGA bit is set and the T6 bit is cleared).

2. WWDG Description

-- WWDG functional description(continued)

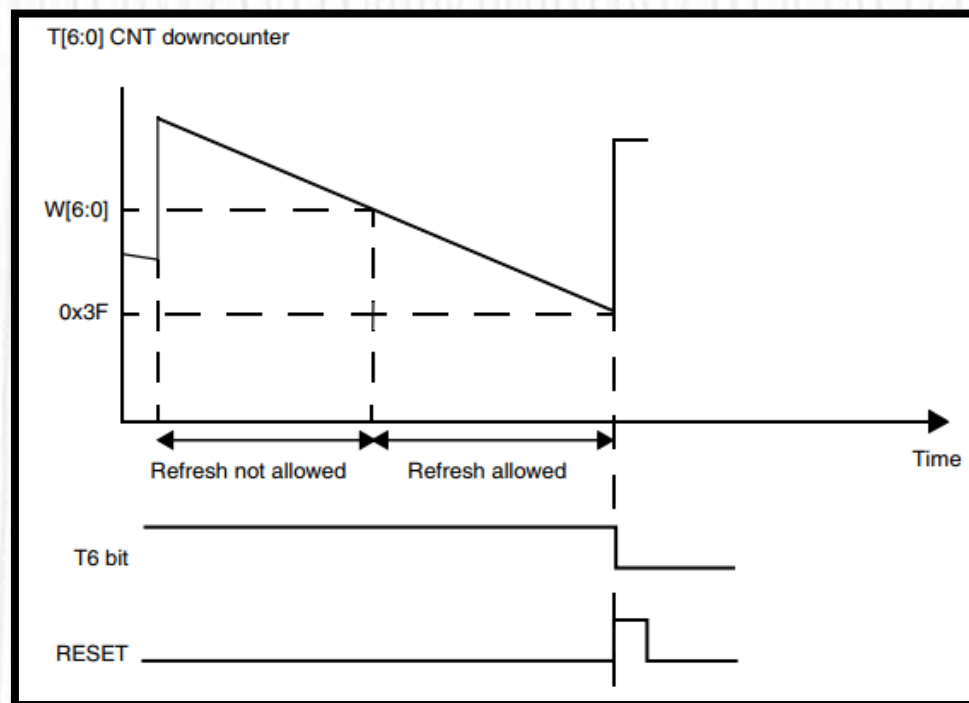


WWDG block diagram

2. WWDG Description

-- WWDG timeout calculation

- $T_{\text{WWDG}} = T_{\text{pclk1}} * 4096 * 2^{\text{WDGTB}} * (T[5:0] + 1)$
 - T_{pclk1} : APB1 clock period



Window watchdog timing diagram

Min-max timeout value @36 MHz

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								WDGA	T[6:0]						
								rs	rw						

Bits 31:8 Reserved, must be kept at reset value.

Bit 7 **WDGA**: Activation bit

This bit is set by software and only cleared by hardware after a reset. When WDGA = 1, the watchdog can generate a reset.

0: Watchdog disabled

1: Watchdog enabled

Bits 6:0 **T[6:0]**: 7-bit counter (MSB to LSB)

These bits contain the value of the watchdog counter. It is decremented every $(4096 \times 2^{\text{WDGTB}})$ PCLK1 cycles. A reset is produced when it rolls over from 0x40 to 0x3F (T6 becomes cleared).



2. WWDG Description

-- WWDG_CFR

- Configuration register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						EWI	WDGTB[1:0]		W[6:0]						
						rs	rw		rw						

Bit 31:10 **Reserved**, must be kept at reset value.

Bit 9 **EWI**: Early wakeup interrupt

When set, an interrupt occurs whenever the counter reaches the value 0x40. This interrupt is only cleared by hardware after a reset.

Bits 8:7 **WDGTB[1:0]**: Timer base

The time base of the prescaler can be modified as follows:

00: CK Counter Clock (PCLK1 div 4096) div 1

01: CK Counter Clock (PCLK1 div 4096) div 2

10: CK Counter Clock (PCLK1 div 4096) div 4

11: CK Counter Clock (PCLK1 div 4096) div 8

Bits 6:0 **W[6:0]**: 7-bit window value

These bits contain the window value to be compared to the downcounter.



2. WWDG Description

-- WWDG_SR

- Status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														EWIF	
														rc_w0	

Bits 31:1 Reserved, must be kept at reset value.

Bit 0 **EWIF**: Early wakeup interrupt flag

This bit is set by hardware when the counter has reached the value 0x40. It must be cleared by software by writing '0. A write of '1 has no effect. This bit is also set if the interrupt is not enabled.



2. WWDG Description

-- WWDG configuration steps

- Enable watchdog clock
- Set frequency division coefficient
- Set upper window value
- Enable early wakeup interrupt (EWI) and group (optional)
- Enable watchdog
- Refresh the value of watchdog t [6:0] (feed dog)
- Write interrupt service function



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

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 GPIO pins. The pin PA15 is highlighted, and its configuration is shown in the bottom right panel. The configuration for PA15 is as follows:

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	✓

The bottom right panel shows the configuration for the selected pin (PA15):

- 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

The screenshot displays the STM32CubeMX configuration tool. On the left, a list of peripherals includes SPI1, SPI2, SPI3, SYS (with a warning icon), TIM1 through TIM8, UART4, UART5, USART1 (highlighted with a green checkmark), USART2, USART3, and USB. The right panel shows 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 shows 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 -- IWDG

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

Configuration

✓ NVIC ✓ Code generation

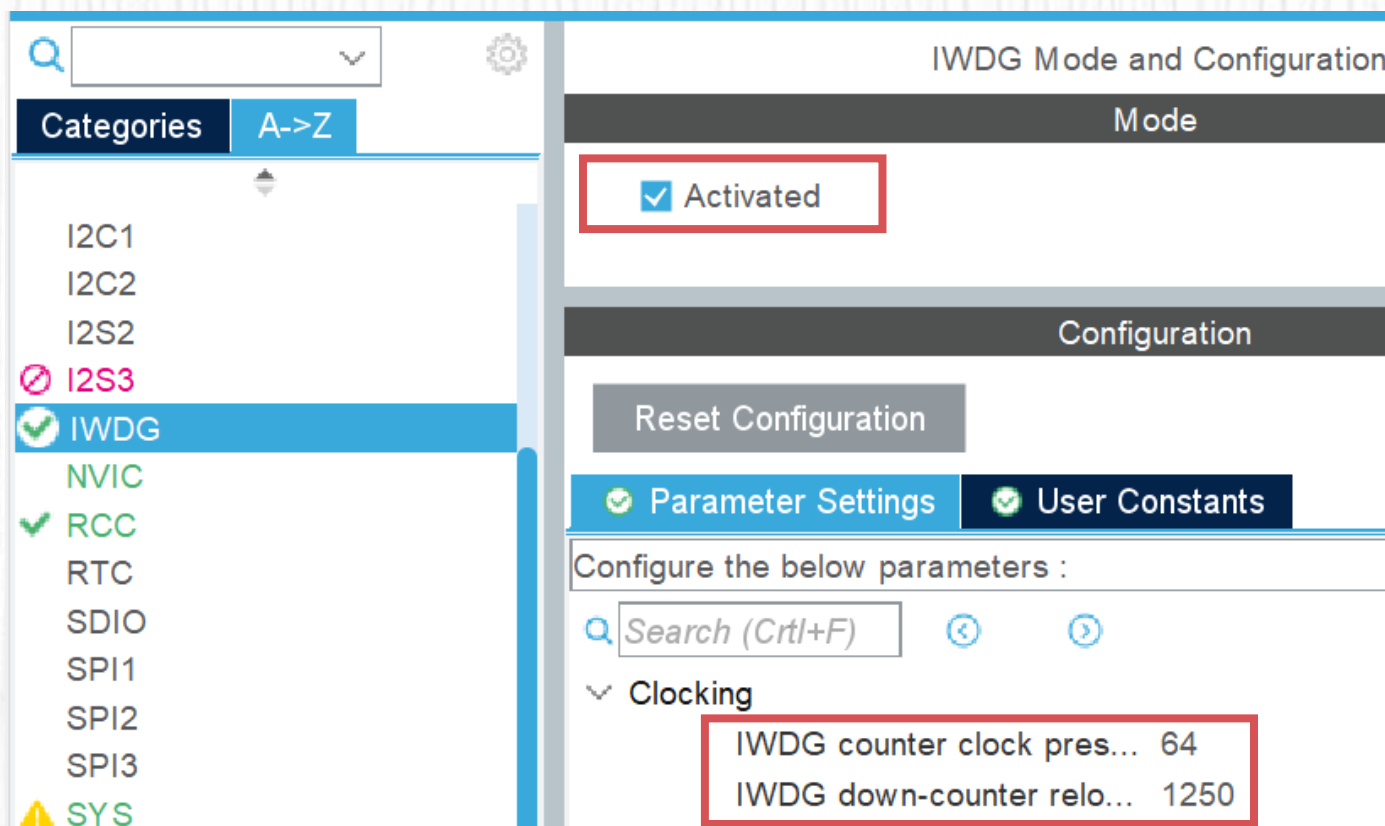
Priority Group 2 bits for pre-emption priority... ☐ Sort by Preemption Priority and Sub Priority ☐ Sort by interrupts name

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 **stm32f1xx_it.c**

```
extern IWDG_HandleTypeDef hiwdg;

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

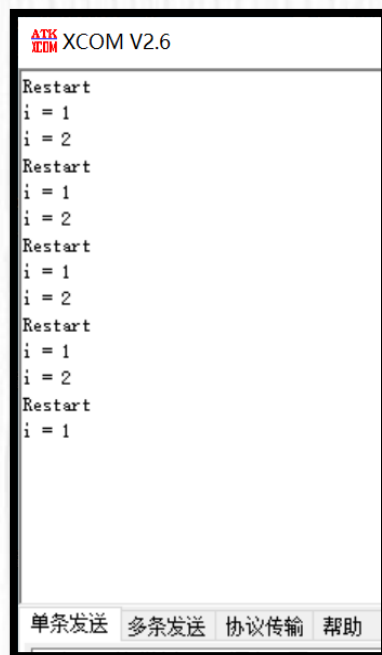
- Set a variable in **main.c** to show if the program is reset
- If the program is reset, the variable i will reset to 0.

```
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 */
```

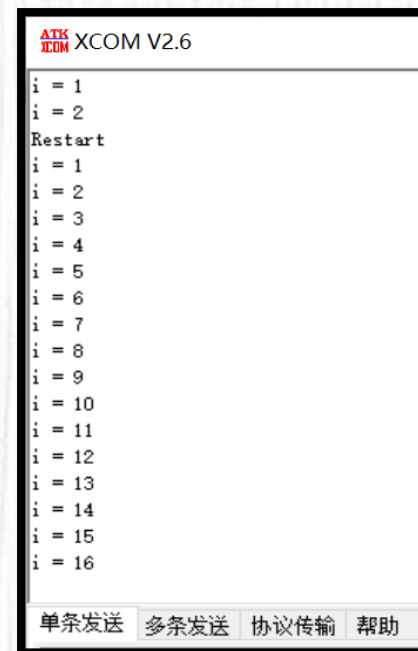

3. How to Program -- IWDG

- 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

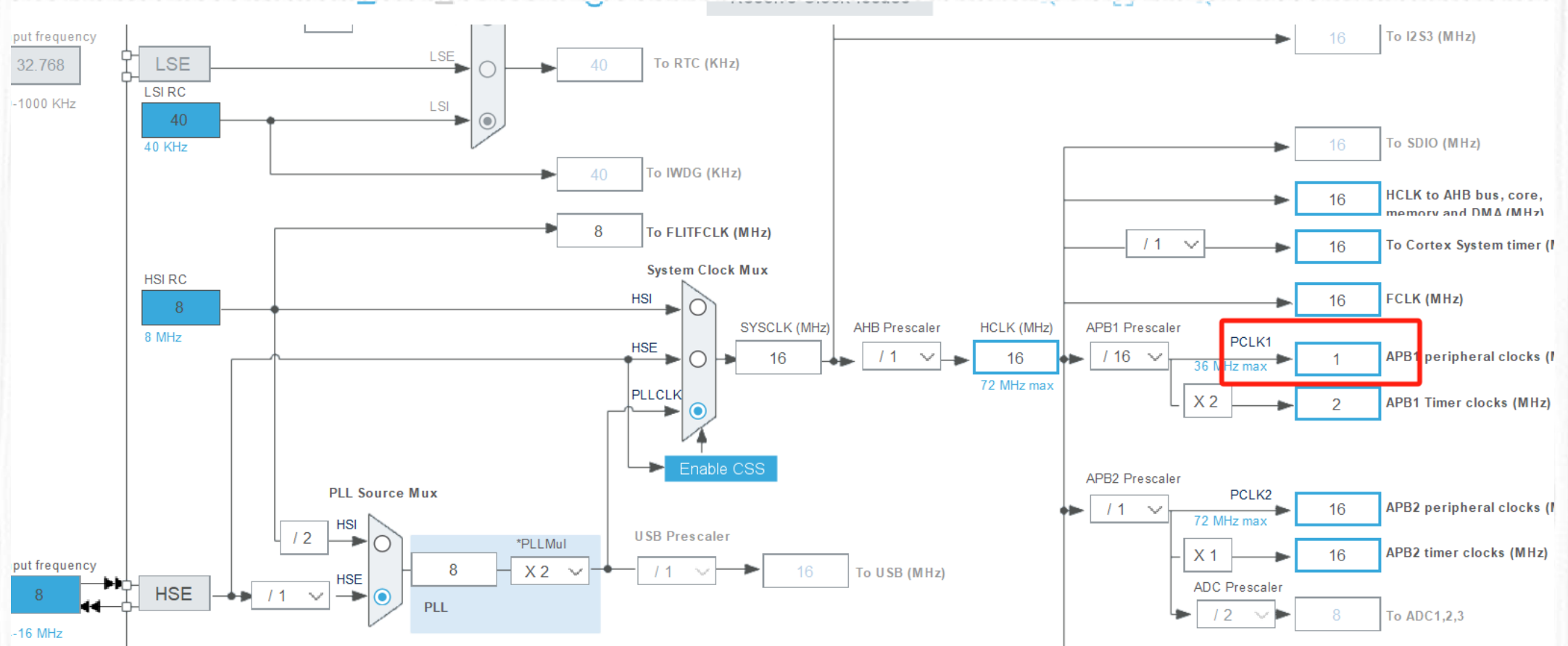


3. How to Program -- WWDG

- Our Goal
 - Use EWI to make LED1 light on.
 - 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 period.

3. How to Program -- WWDG

- Configure RCC





3. How to Program -- WWDG

- Configure GPIO
 - Set PA15(KEY1) as external interrupt source
 - Set PD2(LED1) as output push-pull

The screenshot displays the STM32CubeMX configuration interface. On the left, a sidebar lists various system components: GPIO (selected), IWDG, NVIC, RCC, SYS, and WWDG. Below this, other system modules like Analog, Timers, Connectivity, Multimedia, and Computing are listed with expandable arrows.

The main area is divided into two panels. The left panel shows the 'Search Signals' table, which lists pins and their configurations. The right panel shows the 'PD2 Configuration' settings.

Search Signals Table:

Pin ...	Signal ...	GPIO ...	GPIO ...	GPIO ...	Maxim...	User L...	Modified
PA15	n/a	n/a	Extern...	Pull-up	n/a	KEY1	✓
PD2	n/a	High	Output...	No pull...	Low	LED1	✓

PA15 Configuration:

- GPIO mode: External Interrupt Mode with Falling edge trigger
- GPIO Pull-up/Pull-down: Pull-up

PD2 Configuration:

- GPIO output level: High
- GPIO mode: Output Push Pull
- GPIO Pull-up/Pull-down: No pull-up and no pull-down



3. How to Program -- WWDG

- Configure USART1
 - Set the USART1 as asynchronous mode

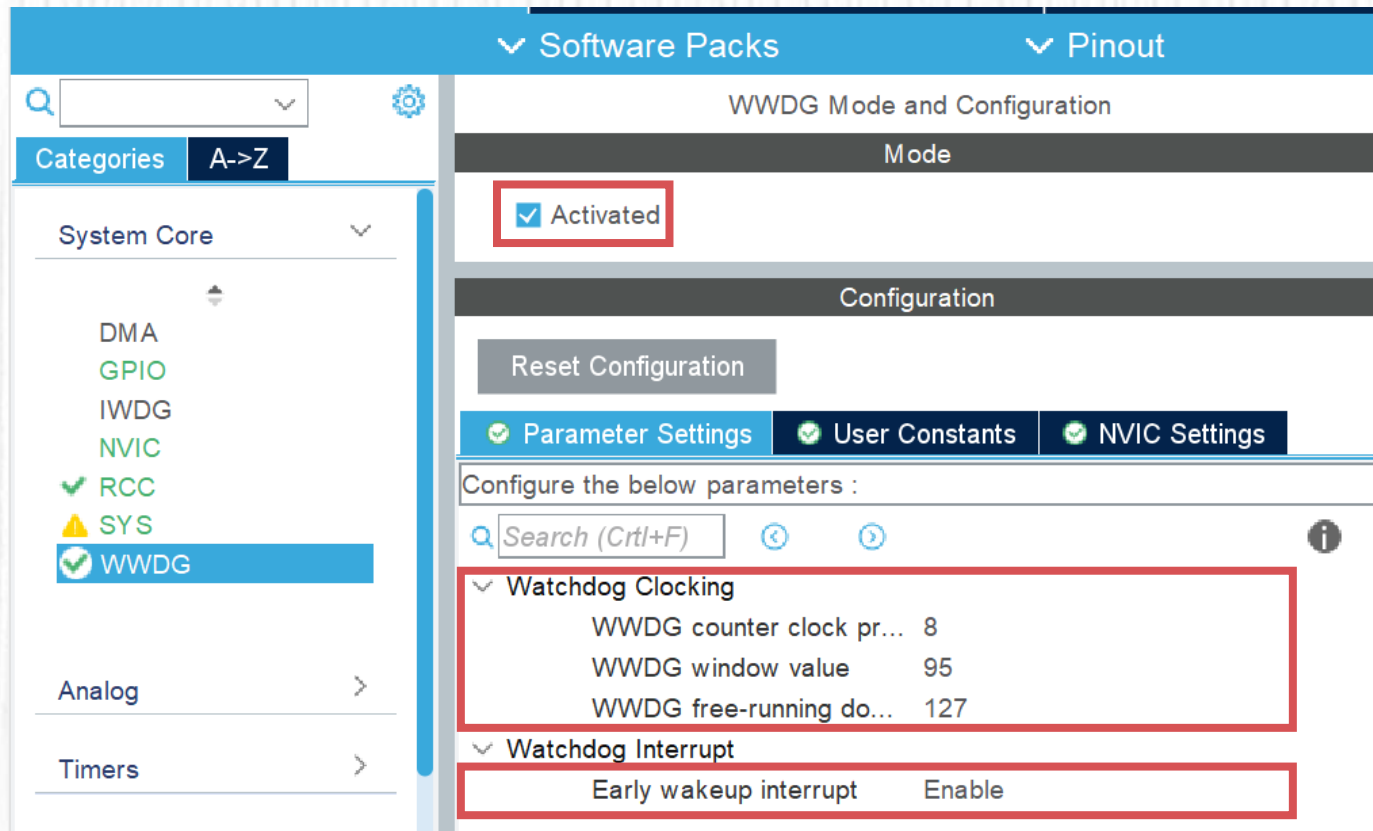
The screenshot shows the STM32CubeMX configuration interface. On the left, a list of peripherals includes SPI1, SPI2, SPI3, SYS (with a warning icon), TIM1, TIM2, TIM3, TIM4, TIM5, TIM6, TIM7, TIM8, UART4, UART5, USART1 (selected with a green checkmark), USART2, USART3, and USB. The right panel shows the configuration for USART1. The 'Mode' is set to 'Asynchronous' (highlighted with a red box). 'Hardware Flow Control (RS232)' is set to 'Disable'. Below this is a 'Configuration' section with a 'Reset Configuration' button. A row of tabs includes 'NVIC Settings', 'DMA Settings', 'GPIO Settings', 'Parameter Settings' (selected), and 'User Constraints'. The 'Parameter Settings' tab displays the following parameters:

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 **`stm32f1xx_it.c`**
- In this demo, we light up the Green LED(PD2) in EWI callback function, to show critical information: when EWI is triggered, the system has software fault already.
- Let the interrupt finish in 1 cycle, because when the counter is 0x3F, MCU will be reset.

```
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

```
PIO_WritePin(GPIOD, LED1_Pin, SET);
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

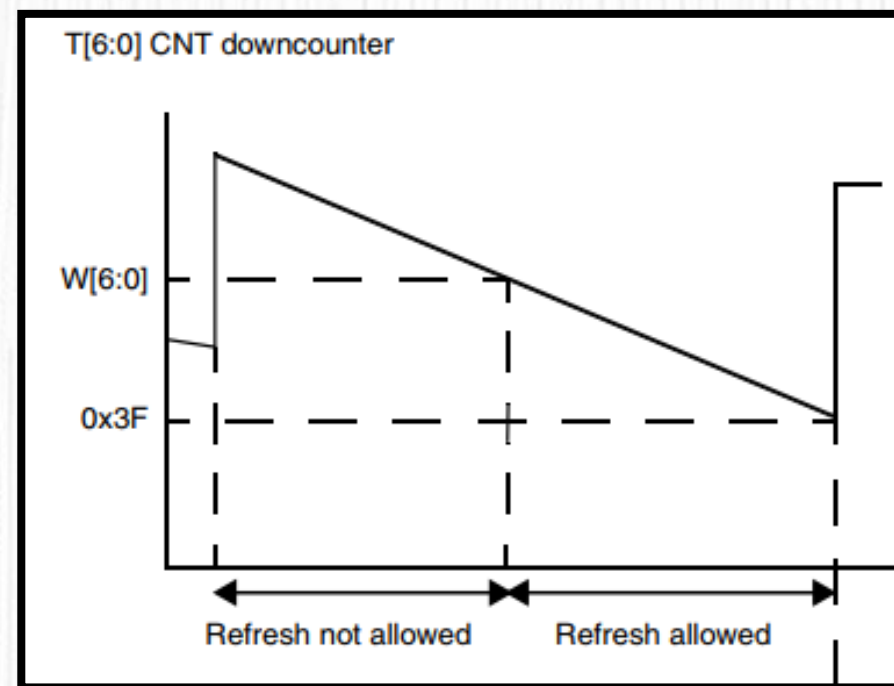
- The program is periodically reset, the value of variable *i* comes back to 1 (shown as Fig.3).The Green LED blinks.
- While pushing button KEY1, the program still can not run continuously(shown as Fig.4).

```
ATK XCOM 正点原子串口调试助手
Restart
i = 1
i = 2
i = 3
Restart
i = 1
i = 2
i = 3
Restart
i = 1
i = 2
i = 3
```

Fig.3 By default: The program resets periodically, Green LED blinks

```
ATK XCOM 正点原子串口调试
i = 4
Restart
i = 1
Restart
i = 1
Restart
i = 1
i = 2
i = 3
```

Fig.4 Pushing button can not correctly feed dog





04

Practice



4.1 Practice(optional)

- Run the IWDG demo on MiniSTM32 board.
- Tell what the timeout of IWDG is.
- Use the timer interrupt to refresh the IWDG in a suitable time interval instead of external interrupt of KEY1, and tell the parameters of IWDG and TIMER of your design.

4.2 Practice



- Run the WWDG demo on MiniSTM32 board.
- Explain the reason why pushing button KEY1, the program still can not run continuously.
- Tell the time range in the demo to refresh WWDG without resetting it.
- Disable the EWI and EXTI, use a timer interrupt to refresh WWDG in a suitable time to avoid resetting CMU, and tell the parameters of WWDG and TIMER of your design.

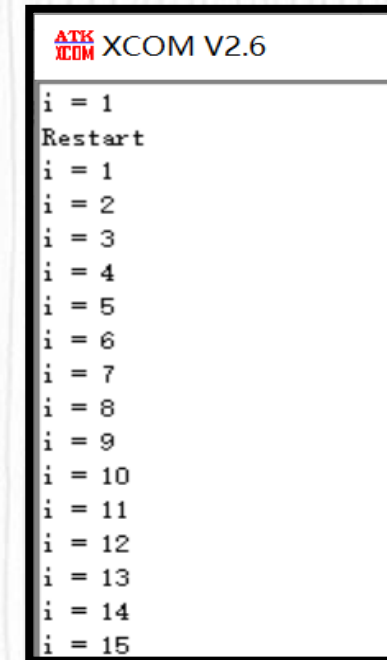


Fig.5 The program runs continuously