AN0045
Application Note

AT32 WDT and WWDT User Guide

Introduction

This application note introduces how to use watchdog timer (WDT) and window watchdog timer (WWDT) of AT32 MCUs.

Note: The corresponding code in this application note is developed on the basis of BSP_V2.x.x provided by Artery. For other versions of BSP, please pay attention to the differences in usage.

Applicable products:

Part number	All AT32 series

Contents

List of tables

List of figures

1 Overview

A watchdog timer is mainly used to improve system stability and recover from malfunctions by resetting the MCU when program crash occurs or if it does not receive a signal from the MCU within the set interval due to runtime logic error. It is recommended to use a watchdog timer to guarantee system stability.

AT32 MCUs have two watchdog timers:

- Watchdog timer (WDT): It has a 12-bit downcounter. When the counter counts down to 0, a system reset is generated; if the counter is refreshed before it counts down to 0, a system reset does not generate.
- Window watchdog timer (WWDT): It has a 7-bit downcounter. When the counter counts down to 0x3F, a system reset is generated; if the counter is refreshed within the set time (window time), a system reset does not generate.

1.1 Differences

WWDTs of each model are the same, compatible with programs.

WDTs of each model are basically the same, except that the advanced time window function or optional function of stop counting in low-power mode may not be available for some models (other functions are the same and compatible with programs).

Table 1. Differences between WDTs of each model

Model	Time window	Stop counting in DEEPSLEEP and STANDY mode (optional)
AT32F403xx	×	×
AT32F403Axx	×	×
AT32F407xx	×	×
AT32F413xx	×	×
AT32F415xx	×	×
AT32F421xx	×	×
AT32F425xx	√	√
AT32F435xx	√	√
AT32F437xx	√	√
AT32L021xx	√	√

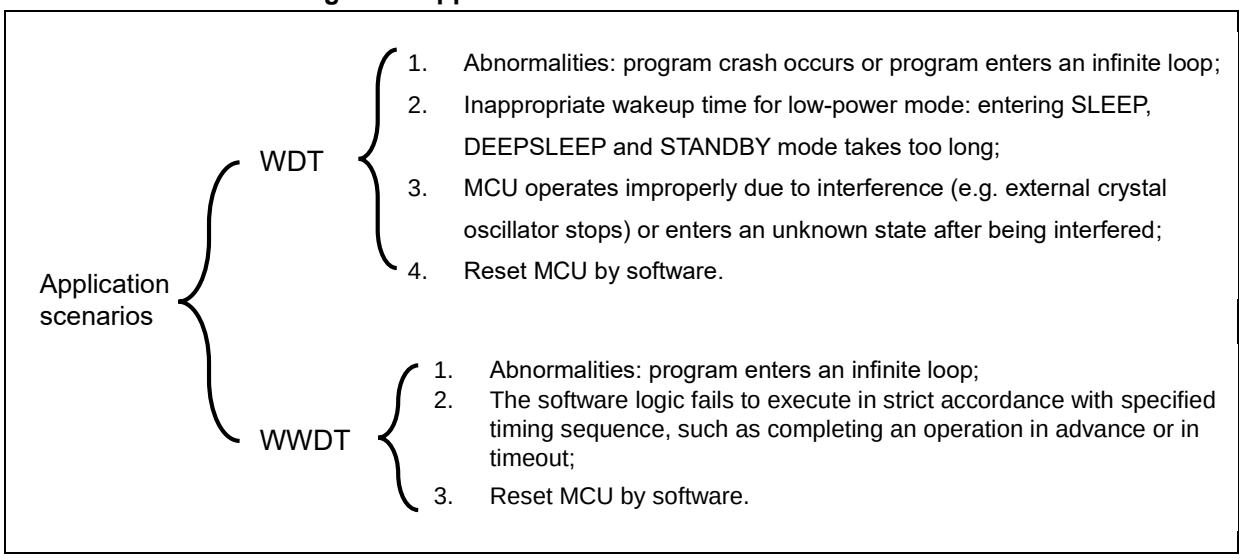
√: This function is available and is the same for different models.

×: This function is not supported.

1.2 Application scenarios

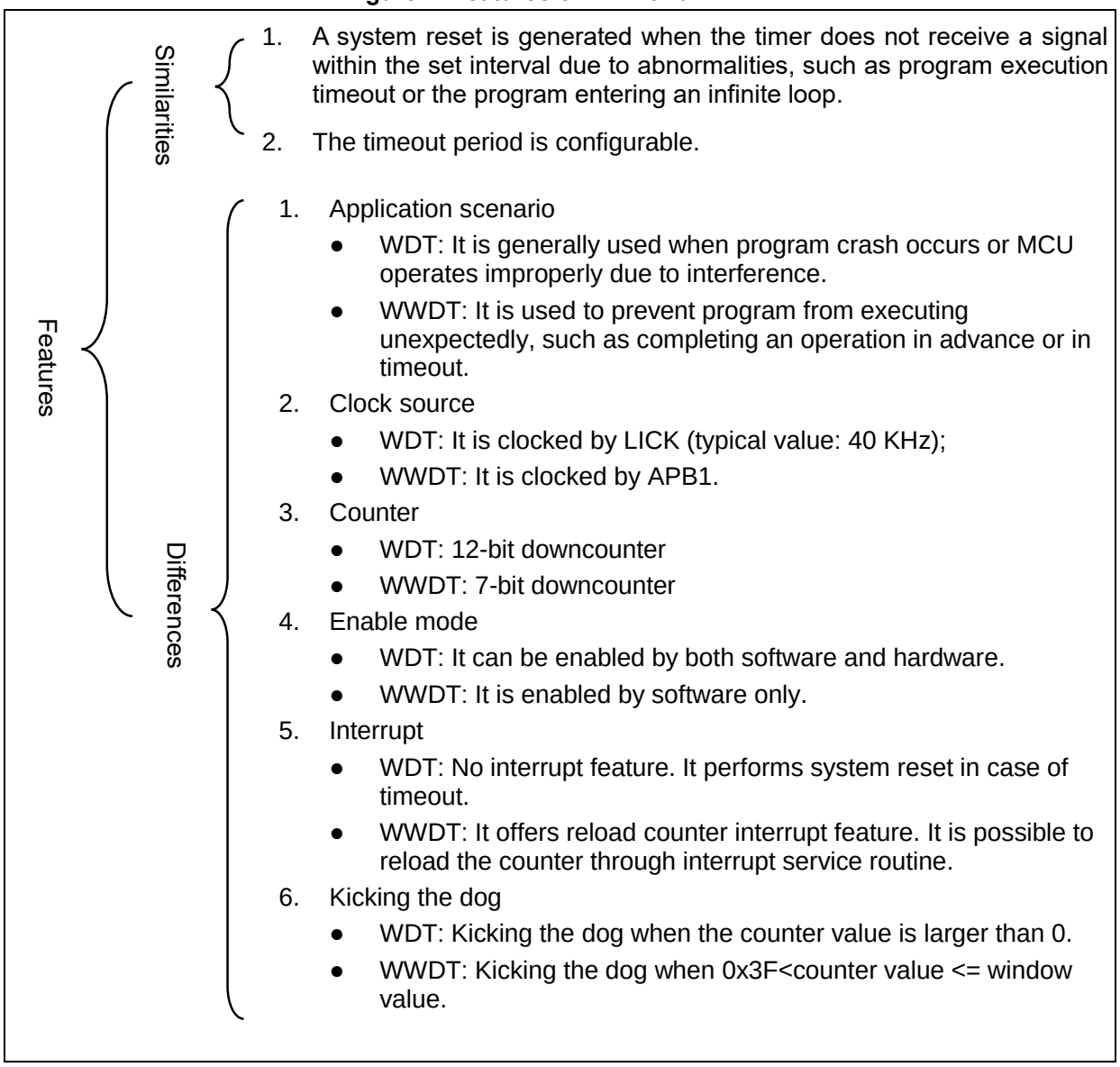
Application scenarios of WDT and WWDT are different, as shown in Figure 1.

Figure 1. Application scenarios of WDT and WWDT



1.3 Features

Figure 2. Features of WDT and WWDT



2 Watchdog timer (WDT)

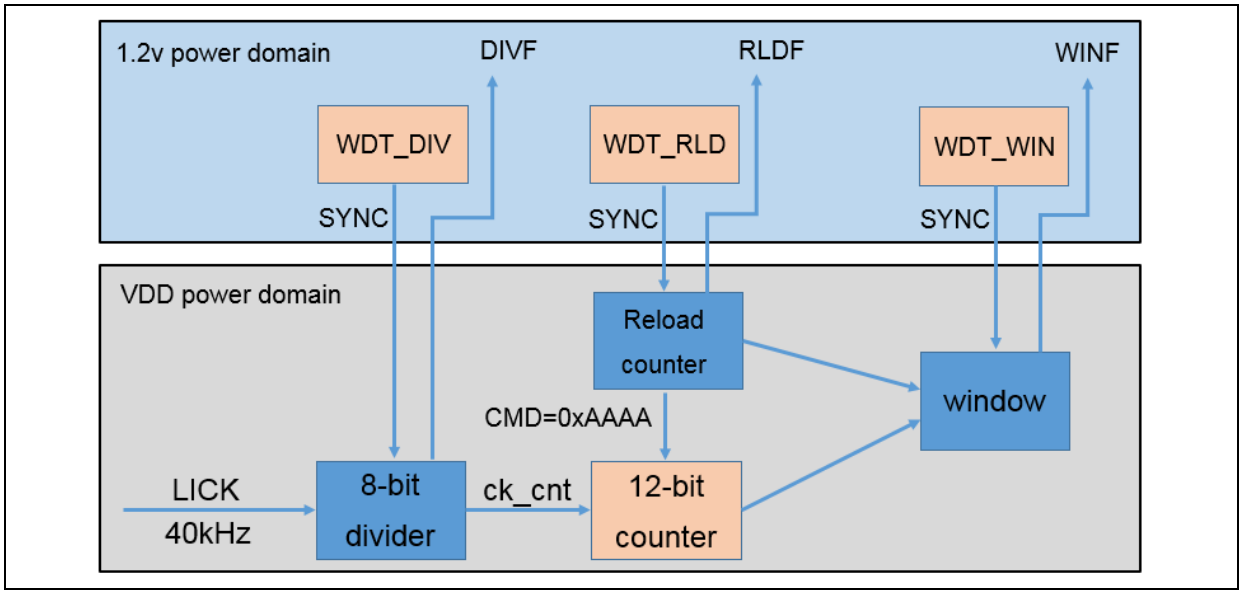
2.1 Access to registers

Status register

WDT registers are in the 1.2 V power domain, and the counter module is in VDD power domain. WDT can work in SLEEP, DEEPSLEEP and STANDBY modes.

The write operation to WDT registers is in the 1.2 V power domain, and values of these registers need to be synchronized to VDD power domain. Each register has a flag indicating whether the synchronization operation is complete. A maximum of four LICK clock cycles (about 125 us) are required for each synchronization. When the register is written, the corresponding synchronization flag is automatically set to 1 and cleared after the synchronization is completed. It is not allowed to write this register before the synchronization flag is cleared.

Figure 3. WDT block diagram



RLDF: When this bit =1, it indicates that the reload value is being synchronized; when this bit=0, it indicates synchronization is complete.

DIVF: When this bit =1, it indicates that the divider value is being synchronized; when this bit=0, it indicates synchronization is complete.

WINF: When this bit =1, it indicates the window value is being synchronized; when this bit=0, it indicates synchronization is complete.

Flag fetch function:

```
flag_status wdt_flag_get(uint16_t wdt_flag);
```

Register write protection

WDT registers are write-protected, which should be unlocked by writing CMD = 0x5555 in the command register before write operation. The read protection is enabled when another value is written. Table 2 lists the read-protected registers:

Table 2. WDT registers

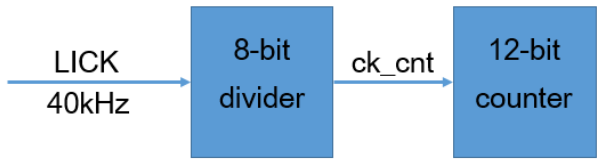
Register	Abbr.	Synchronization complete flag	Write protection	Unlock
Command register	WDT_CMD	-	No	-
Divider register	WDT_DIV	DIVF	Yes	Write CMD=0x5555
Reload register	WDT_RLD	RLDF	Yes	Write CMD=0x5555
Status register	WDT_STS	-	No	-
Window register	WDT_WIN	WINF	Yes	Write CMD=0x5555

Register write enable function:

```
void wdt_register_write_enable( confirm_state new_state);
```

2.2 Clock structure

Figure 4. WDT clock



The WDT downcounter is clocked by LICK (divided by an 8-bit divider). The LICK is an internal RC clock with a typical value of 40 kHz, with its range falling between 30 kHz and 60 kHz (refer to the datasheet of each series for details). The timeout period is also within a certain range, so a margin should be taken into account when configuring timeout period. The LICK can be measured and calibrated to obtain the WDT timeout with a relatively accuracy.

Configure the prescaler divider (4, 8, 16, 32, 64, 128 or 256) by setting the DIV[2:0] bit.

$$f_{ck\_cnt} = \frac{f_{LICK}}{\text{Prescaler divider}}$$

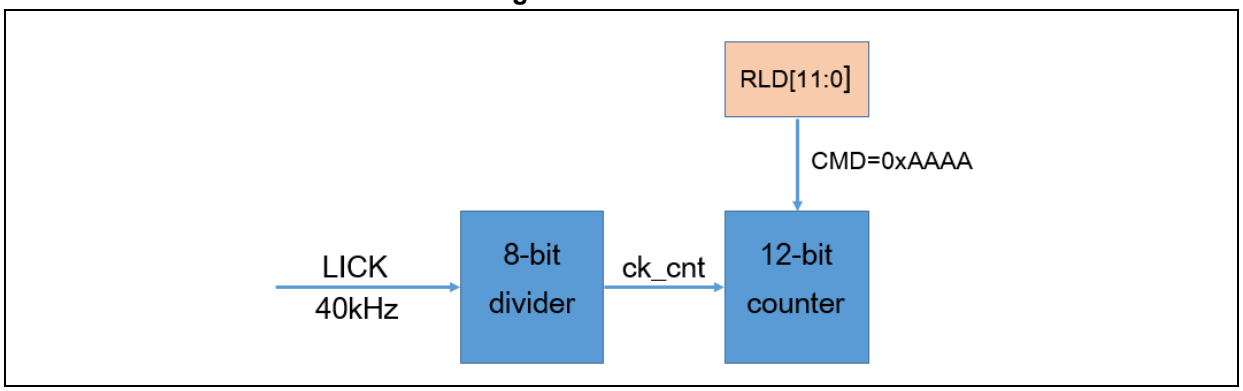
Prescaler divider set function:

```
void wdt_divider_set(wdt_division_type division);
```

2.3 Counter

The WDT is featured with a 12-bit downcounter (maximum value: 0xFFF). Once the WDT is enabled, it starts counting down from the set value, and a system reset is generated when the counter reaches 0.

Figure 5. WDT reload



The counter value is set through the RLD bit in the reload register. When the prescaler divider is set, the RLD bit value determines WDT reset time. Whenever 0xAAAA is written to the WDT_CMD register, the value of this register is updated to the downcounter. This operation is commonly referred to as kicking the dog, which should be performed before the downcounter reaches 0; otherwise, a reset is generated.

The WDT reset time is calculated as below:

$$\text{Timeout period} = (\text{RLD} + 1) \times \frac{\text{Prescaler divider}}{f_{LICK}}$$

Table 3. WDT timeout period (LICK = 40 kHz)

Prescaler divider	Minimum timeout (ms) RLD[11:0] = 0x000	Maximum timeout (ms) RLD[11:0] = 0xFFF
4	0.1	409.6
8	0.2	819.2
16	0.4	1638.4
32	0.8	3276.8
64	1.6	6553.6
128	3.2	13107.2
256	6.4	26214.4

Reload value set function:

```
void wdt_reload_value_set(uint16_t reload_value);
```

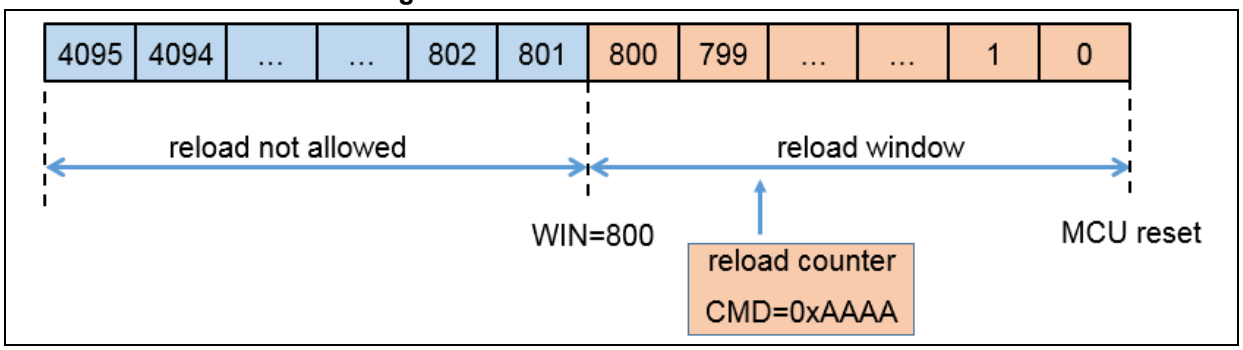
Reload WDT counter (kicking the dog) function:

```
void wdt_counter_reload(void);
```

2.4 Window value

The window value feature is enabled by setting the WIN[11:0] to 0xFF. When the counter value is greater than the window value, the reload counter will perform a system reset. For example, when WIN=800, the time window for reload is shown as below:

Figure 6. Window value feature of WDT



Window value set function:

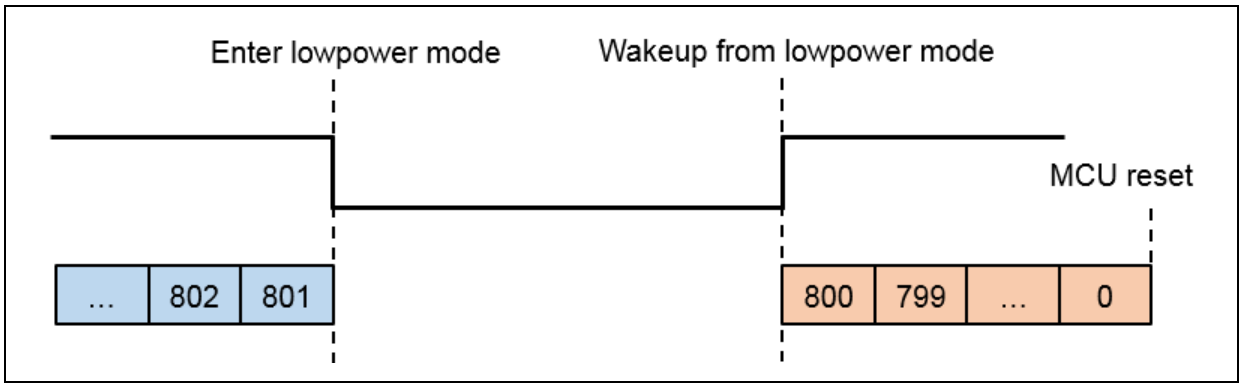
```
void wdt_window_counter_set(uint16_t window_cnt);
```

2.5 WDT low power counting mode

WDT can work in Sleep, Deepsleep and Standby modes. It is possible to stop counting in Deepsleep and Standby modes by setting the nWDT_DEPSLP and nWDT_STDBY bits in the User System Data area.

If the counter is disabled, it will stop decrementing as soon as the Deepsleep and Standby modes are entered. This means that the WDT would not perform a system reset in both low power modes. After waking up from these two modes, it continues downcounting from the value at the time of the entry of these modes.

Figure 7. Stop counting in low-power mode



User system data erase function:

```
flash_status_type flash_user_system_data_erase(void);
```

User system data set function:

```
flash_status_type flash_ssb_set(uint8_t usd_ssb);
```

Example of stop counting in low-power mode:

```
/* User system data erase */
flash_user_system_data_erase();
/* Stop counting in DEEPSLEEP and STANDBY modes */
flash_ssb_set(USD_WDT_ATO_DISABLE | USD_DEPSLP_NO_RST | USD_STDBY_NO_RST |
```

```
FLASH_BOOT_FROM_BANK1 | USD_WDT_DEPSLP_STOP | USD_WDT_STDBY_STOP);
```

2.6 WDT enable

WDT can be enabled by both hardware and software operations. Once enabled, WDT cannot be disabled unless a reset occurs.

Enable by software operation

Write 0xCCCC to the command register to enable WDT.

WDT enable function:

```
void wdt_enable(void);
```

Enable by hardware operation

Set the nWDT_ATO_EN bit in the User System Data area to enable WDT. Once enabled, the WDT will run automatically after power-on reset.

Example of enabling WDT by hardware operation:

```
/* User system data erase */
flash_user_system_data_erase();
/* Enable WDT by hardware operation */
flash_ssb_set(USD_WDT_ATO_ENABLE | USD_DEPSLP_NO_RST | USD_STDBY_NO_RST |
FLASH_BOOT_FROM_BANK1 | USD_WDT_DEPSLP_CONTINUE |
USD_WDT_STDBY_CONTINUE);
```

2.7 Application method

The WDT is mainly used to check whether program crash occurs or program enters an infinite loop. For example, if the program takes 10ms to complete operation, set the WDT timeout period as 20ms. A reset is not generated in case of kicking the dog immediately after the program operation is complete. If kicking the dog is not performed after 20ms, it indicates that an error occurs, and MCU reset is generated at this time.

For example:

To set WDT timeout period = 20ms, set the prescaler divider = 4, and counter value = 200, as shown below:

$$\text{Timeout period} = \text{RLD} \times \frac{\text{Prescaler divider}}{f_{LICK}} = 200 \times \frac{4}{40000} = 20ms$$

Configuration steps:

1. Disable register write protection

```
wdt_register_write_enable(TRUE);
```

2. Set prescaler divider = 4

```
wdt_divider_set(WDT_CLK_DIV_4);
```

3. Set reload value = 200

```
wdt_reload_value_set(200 - 1);
```

4. Enable WDT

```
wdt_enable();
```

5. Reload counter in program

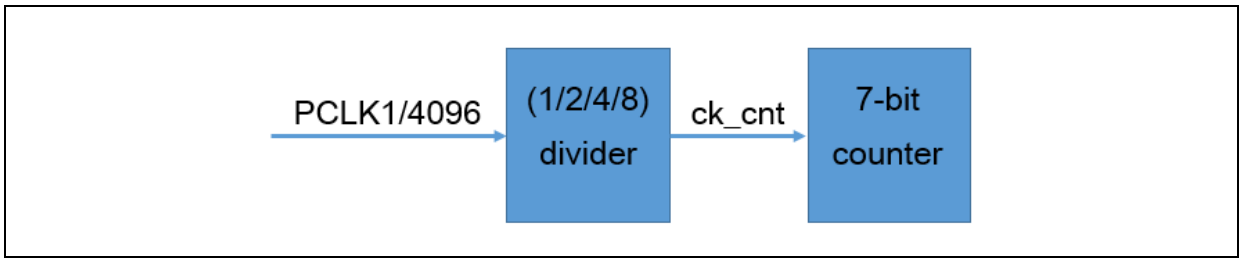
```
wdt_counter_reload();
```

3 Window watchdog timer (WWDT)

A WWDT is mainly used to check detect whether software logic is executed as expected. Set the related registers to set the upper and lower time limits of kicking the dog (a reset is generated when the downcounter value is smaller than 0x40 or the downcounter is refreshed outside the time window).

3.1 Clock structure

Figure 8. WWDT clock



The window watchdog timer is clocked by a divided APB1_CLK. The precision of the APB1_CLK enables the window watchdog to take accurate control of the limited window.

The APB1_CLK is divided by 4096, and is sent to the prescaler and finally to a 7-bit downcounter CNT[6:0]. The prescaler divider can be defined (1, 2, 4, or 8) by the DIV[1:0] bit.

$$f_{ck\_cnt} = \frac{f_{PCLK}}{4096 \times 2^{DIV[1:0]}}$$

Divider set function:

```
void wwdt_divider_set(wwdt_division_type division);
```

3.2 Counter

The WWDT is featured with a 7-bit downcounter (maximum value: 0x7F). Once the WWDT is enabled, the counter starts counting down, and a system reset is generated when the counter reaches 0x3F.

$$\text{Timeout period} = (\text{CNT}[5:0] + 1) \times \frac{4096 \times 2^{DIV[1:0]}}{f_{PCLK}}$$

Table 4. WWDT timeout period (PCLK = 72 MHz)

Division value	Minimum timeout (ms) CNT [6:0] = 0x40	Maximum timeout (ms) CNT [6:0] = 0x7F
1	56.5μs	3.64ms
2	113.5μs	7.28ms
4	227.5μs	14.56ms
8	455μs	29.12ms

Counter value set function:

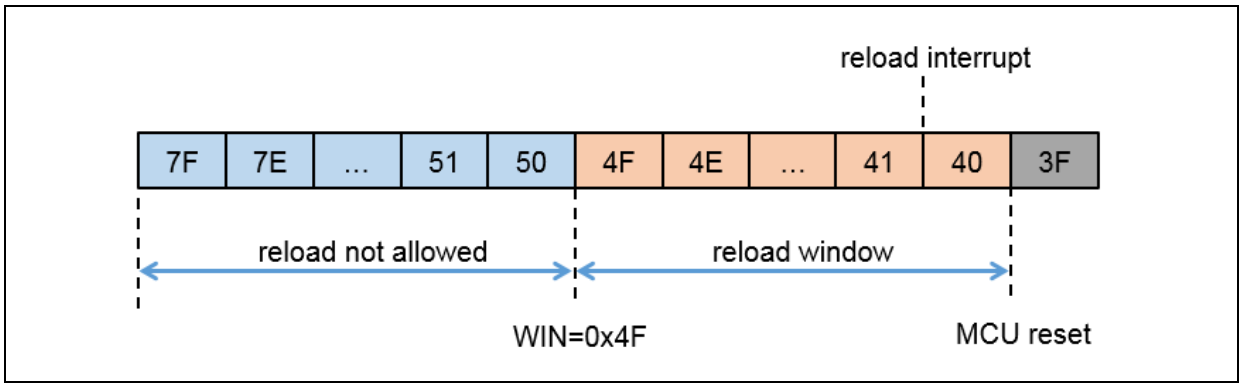
```
void wwdt_counter_set(uint8_t wwdt_cnt);
```

3.3 Window value

The window value (WIN[6:0]) is configurable, with the maximum value of 0x7F and the minimum value being larger than 0x40 (overall range: 64~127, i.e., 0x40~0x7F). When the downcounter value is smaller than the window value, the downcounter is refreshed; otherwise, a reset is generated.

For the sake of kicking the dog, the reload counter interrupt (RLDIEN bit) can be used. When the downcounter reaches 0x40, an interrupt is generated, and the counter is reloaded in the interrupt service routine.

Figure 9. Window value of WWDT



As shown in Figure 9, when the window value is set to 0x4F, it is not allowed to refresh 0x7F~0x50, and 0x4F~0x40 can be refreshed.

Reload flag clear function:

```
void wwdt_flag_clear(void);
```

Get reload flag function:

```
flag_status wwdt_flag_get(void);
```

Reload interrupt enable function:

```
void wwdt_interrupt_enable(void);
```

Window value set function:

```
void wwdt_window_counter_set(uint8_t window_cnt);
```

3.4 WWDT enable

Set WWDTEN=1 to enable WWDT. Once enabled, the WWDT cannot be disabled until a reset is generated. Set WWDT counter value when enabling the WWDT to avoid resetting immediately after the watchdog is enabled.

WWDT enable function:

```
void wwdt_enable(uint8_t wwdt_cnt);
```

3.5 Application method

A WWDT is mainly used to check whether the logic runs properly. For example, a program takes

10ms to complete operation, and a logic error occurs if the program takes less than 10ms. In this case, set the window value to be 9ms. Kicking the dog before this window value (9ms) indicates that the program is faulty, and a reset is generated at this time.

For example:

When PCLK1=36 MHz, set the WWDT timeout=9ms, the prescaler divider=4 and therefore the total division value is 4x4096=16384. The counter value is 127 and window value is 108, and the time for decrementing from the counter value to window value is about 9.1ms.

$$\text{Window value}= \ (\text{CNT}-\text{WIN})\times\frac{4096\times 2^{DIV[1:0]}}{f_{PCLK1}}=((127-108)+1)\times\frac{4096\times 2^{2}}{36MHz}=9.1ms$$

$$\text{Reset time}= \ (\text{CNT}-0\text{x3F})\times\frac{4096\times 2^{DIV[1:0]}}{f_{PCLK1}}=(127-63)\times\frac{4096\times 2^{2}}{36MHz}=29.1ms$$

Therefore, kicking the dog is allowed within 9.1~29.1ms and not allowed within 0~9.1ms.

Configuration steps:

1. Enable WWDT APB1_CLK

```
crm_periph_clock_enable(CRM_WWDT_PERIPH_CLOCK, TRUE);
```

2. Set prescaler divider=4, and total division value is 4096x4=16384

```
wwdt_divider_set(WWDT_PCLK1_DIV_16384);
```

3. Set window value=108

```
wwdt_window_counter_set(108);
```

4. Enable WWDT

```
wwdt_enable(127);
```

5. Reload counter in program

```
wwdt_counter_set(127);
```

Note: It is executed with 0x3F<downcounter value<=window value.

4 Example of WDT application

4.1 Purpose

It is used to demonstrate the function and application of WDT.

4.2 Resources

1) Hardware:

   AT-START BOARD of the corresponding model

2) Software:

   project\at_start_f4xx\examples\wdt\wdt_reset

Note: All projects are built around Keil 5. If users want to use them in other compiling environments, please refer to AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates (such as IAR6/7, keil 4/5) for a simple change.

4.3 Software design

1) Configuration
   - Initialize WDT
   - Kicking the dog in the main program

2) Codes
   - Main function code

```
int main(void)
{
  /* Initialize system clock */
  system_clock_config();

  /* Initialize AT-START board */
  at32_board_init();

  /* Get WDT reset flag */
  if(crm_flag_get(CRM_WDT_RESET_FLAG) != RESET)
  {
    /* WDT reset */
    crm_flag_clear(CRM_WDT_RESET_FLAG);

    at32_led_on(LED4);
  }
  else
  {
    /* Reset from other modes */
    at32_led_off(LED4);
  }

  /* Unlock write protection */
  wdt_register_write_enable(TRUE);
```

```
  /* Set WDT divider */
  wdt_divider_set(WDT_CLK_DIV_4);

  /* Set WDT reload value */
  wdt_reload_value_set(3000 - 1);

  /* Enable WDT */
  wdt_enable();

  while(1)
  {
    /* Reload WDT counter */
    wdt_counter_reload();

    at32_led_toggle(LED3);

    delay_ms(200);

    if(at32_button_press() == USER_BUTTON)
    {
      while(1);
    }
  }
}
```

4.4 Test result

- WDT does not reset during normal operation. Press the USER button and stop kicking the dog, MCU reset will occur.
- After reset, if WDT reset is detected, LED4 will be ON; otherwise, LED4 is OFF.

5 Example of WWDT application

5.1 Purpose

It is used to demonstrate the function and application of WWDT.

5.2 Resources

1) Hardware

   AT-START BOARD of the corresponding model

2) Software

   project\at_start_f4xx\examples\wwdt\wwdt_reset

Note: All projects are built around Keil 5. If users want to use them in other compiling environments, please refer to AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates (such as IAR6/7, keil 4/5) for a simple change.

5.3 Software design

1) Configuration

   - Initialize WWDT
   - Kicking the dog in the main program

2) Codes

   - Main function code

```
int main(void)
{
  /* Initialize system clock */
  system_clock_config();

  /* Initialize AT-START board */
  at32_board_init();

  /* Get WWDT reset flag */
  if(crm_flag_get(CRM_WWDT_RESET_FLAG) != RESET)
  {
    /* WWDT reset */
    crm_flag_clear(CRM_WWDT_RESET_FLAG);

    at32_led_on(LED4);
  }
  else
  {
    /* Reset from other modes */
    at32_led_off(LED4);
  }

  /* Enable WWDT */
  crm_periph_clock_enable(CRM_WWDT_PERIPH_CLOCK, TRUE);
```

```
  /* Set WWDT divider */
  wwdt_divider_set(WWDT_PCLK1_DIV_16384);

  /* Set window value */
  wwdt_window_counter_set(0x6F);

  /* Enable WWDT */
  wwdt_enable(0x7F);

  while(1)
  {
    at32_led_toggle(LED3);

    delay_ms(6);

    /* Reload WWDT */
    wwdt_counter_set(0x7F);

    if(at32_button_press() == USER_BUTTON)
    {
      while(1);
    }
  }
}
```

5.4 Test result

- WWDT does not reset during normal operation. Press the USER button and stop kicking the dog, MCU reset will occur.
- After reset, if WWDT reset is detected, LED4 will be ON; otherwise, LED4 is OFF.

6 Revision history

Table 5. Document revision history

Date	Version	Revision note
2021.12.15	2.0.0	Initial release.
2022.06.06	2.0.1	Modified the *flash_ssb_set* function.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers are solely responsible for the selection and use of ARTERY's products and services; ARTERY assumes no liability for purchasers' selection or use of the products and the relevant services.

No license, express or implied, to any intellectual property right is granted by ARTERY herein regardless of the existence of any previous representation in any forms. If any part of this document involves third party's products or services, it does NOT imply that ARTERY authorizes the use of the third party's products or services, or permits any of the intellectual property, or guarantees any uses of the third party's products or services or intellectual property in any way.

Except as provided in ARTERY's terms and conditions of sale for such products, ARTERY disclaims any express or implied warranty, relating to use and/or sale of the products, including but not restricted to liability or warranties relating to merchantability, fitness for a particular purpose (based on the corresponding legal situation in any unjudicial districts), or infringement of any patent, copyright, or other intellectual property right.

ARTERY's products are not designed for the following purposes, and thus not intended for the following uses: (A) Applications that have specific requirements on safety, for example: life-support applications, active implant devices, or systems that have specific requirements on product function safety; (B) Aviation applications; (C) Aerospace applications or environment; (D) Weapons, and/or (E) Other applications that may cause injuries, deaths or property damages. Since ARTERY products are not intended for the above-mentioned purposes, if purchasers apply ARTERY products to these purposes, purchasers are solely responsible for any consequences or risks caused, even if any written notice is sent to ARTERY by purchasers; in addition, purchasers are solely responsible for the compliance with all statutory and regulatory requirements regarding these uses.

Any inconsistency of the sold ARTERY products with the statement and/or technical features specification described in this document will immediately cause the invalidity of any warranty granted by ARTERY products or services stated in this document by ARTERY, and ARTERY disclaims any responsibility in any form.

© 2022 ARTERY Technology - All Rights Reserved