APPLICATION NOTE

AT07902: SAM4L Watchdog Timer

ASF PROGRAMMERS MANUAL

SAM4L Watchdog Timer

This document describes Watchdog Timer (WDT) driver. This driver provides access to the main features of the WDT controller. The Watchdog Timer can be used to prevent system lock-up if the software becomes trapped in a deadlock. It can generate a general reset or a processor reset only.

The outline of this documentation is as follows:

Table of Contents

Software License

1. Prerequisites

The WDT module depends on the following modules:

- None

2. Module Overview

The Watchdog Timer (WDT) will reset the device unless it is periodically serviced by the software. This allows the device to recover from a condition that has caused the system to be unstable.

The WDT has an internal counter clocked from the system RC oscillator or one of the 32kHz oscillators.

The WDT counter must be periodically cleared by software to avoid a watchdog reset. If the WDT timer is not cleared correctly, the device will reset and start executing from the boot vector.

If the WDT is configured in interrupt mode an interrupt request will be generated on the first timeout and a reset on the second timeout if the interrupt has not been cleared.

3. Special Considerations

- The watchdog timer is disabled by default unless you enable it in the conf_board.h file
- When the WDT is enabled, it remains clocked in all sleep modes. It is not possible to enter sleep modes where the source clock of CLK_CNT is stopped. Attempting to do so will result in the chip entering the lowest sleep mode where the source clock is running, leaving the WDT operational. After a watchdog reset the WDT bit in the Reset Cause Register (RCAUSE) in the Power Manager will be set.
- The clock for the WDT bus interface (CLK_WDT) is generated by the Power Manager. This clock is enabled at reset, and can be disabled in the Power Manager. It is recommended to disable the WDT before disabling the clock, to avoid freezing the WDT in an undefined state. There are two possible clock sources for the Watchdog Timer clock, CLK_CNT:

  1. System RC oscillator (RCSYS): This oscillator is always enabled when selected as clock source for the WDT.

  2. 32kHz crystal oscillator or RC oscillator (OSC32 or RC32): This oscillator has to be enabled in the Backup System Control Interface (BSCIF) before using it as clock source for the WDT. Selection between OSC32 and RC32 should be done in the Backup Power Manager. The WDT will not be able to detect if this clock is stopped.

- The WDT interrupt request line is connected to the NVIC. Using the WDT interrupt requires the NVIC to be programmed first.
- Debug Operation. The WDT counter is not frozen during debug operation, unless the Core is halted and the bit corresponding to the WDT is set in the Peripheral Debug Register (PDBG). If the WDT counter is not frozen during debug operation it will need periodically clearing to avoid a watchdog reset.
- Fuses. The WDT can be enabled at reset. This is controlled by the WDTAUTO fuse

4. Extra Information

For extra information, see Extra Information for SAM4L Watchdog Timer. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

5. Examples

For a list of examples related to this driver, see Quickstart guide for SAM4L watchdog driver.

6. API Overview

6.1 Structure Definitions

6.1.1 Struct wdt_config

Configuration structure for a Watchdog Timer instance. This structure should be initialized by the wdt_get_config_defaults() function before being modified by the user application.

Table 6-1. Members

Type	Name	Description
bool	always_on	If set, the watchdog will be enabled and locked to the current configuration (SFV bit).[1]
uint32_t	clk_src	Clock source select.[2]
bool	disable_flash_cali	Disable flash calibration redone after a watchdog reset.
bool	disable_wdt_after_reset	Disable the watchdog after a watchdog reset.
enum wdt_period	timeout_period	Number of CLK_CNT until the WDT expires. [3]
uint32_t	wdt_mode	WDT mode. /** [4]
enum wdt_period	window_period	Number of CLK_CNT until the reset window opens. [5]

Notes:
[1]Only a reset unlocks the SFV bit.
[2]Valid values are WDT_MODE_BASIC and WDT_MODE_WINDOW.
[3]See wdt_period for a list of valid values.
[4]Valid values are WDT_CLK_SRC_RCSYS and WDT_CLK_SRC_32K.
[5]See wdt_period for a list of valid values.

6.1.2 Struct wdt_dev_inst

Device instance structure for a Watchdog Timer driver instance. This structure should be initialized by the wdt_init() function to associate the instance with a particular hardware module of the device.

Table 6-2. Members

Type	Name	Description
Wdt *	hw_dev	Base address of the WDT module.
struct wdt_config *	wdt_cfg	Pointer to WDT configuration structure.

6.2 Macro Definitions

6.2.1 Macro WDT_CLK_SRC_32K

```
#define WDT_CLK_SRC_32K
```

Select the 32kHz oscillator as clock source.

6.2.2 Macro WDT_CLK_SRC_RCSYS

```
#define WDT_CLK_SRC_RCSYS
```

Select the system RC oscillator (RCSYS) as clock source.

6.2.3 Macro WDT_MODE_BASIC

```
#define WDT_MODE_BASIC
```

Basic mode.

6.2.4 Macro WDT_MODE_WINDOW

```
#define WDT_MODE_WINDOW
```

Window mode.

6.3 Function Definitions

6.3.1 Function wdt_clear()

Restart the watchdog timer.

```
void wdt_clear(
  struct wdt_dev_inst *const dev_inst)
```

Table 6-3. Parameters

Data direction	Parameter name	Description
[in]	dev_inst	Device structure pointer

6.3.2 Function wdt_disable()

Disable the WDT module.

```
void wdt_disable(
  struct wdt_dev_inst *const dev_inst)
```

Table 6-4. Parameters

Data direction	Parameter name	Description
[in]	dev_inst	Device structure pointer

6.3.3 Function wdt_enable()

Enable the WDT module.

```
void wdt_enable(
  struct wdt_dev_inst *const dev_inst)
```

Table 6-5. Parameters

Data direction	Parameter name	Description
[in]	dev_inst	Device structure pointer

6.3.4 Function wdt_get_config_defaults()

Initializes a Watchdog Timer configuration structure to defaults.

```
void wdt_get_config_defaults(
  struct wdt_config *const cfg)
```

Initializes a given Watchdog Timer configuration structure to a set of known default values. This function should be called on all new instances of these configuration structures before being modified by the user application.

The default configuration is as follows:

- Select the RCSYS oscillator as clock source
- Not locked, to allow for further (re)configuration
- A timeout period of 131072 clocks of the Watchdog module clock (about 1s)
- No window period, so that the watchdog count can be reset at any time

Table 6-6. Parameters

Data direction	Parameter name	Description
[out]	cfg	Configuration structure to initialize to default values

6.3.5 Function wdt_get_status()

Get the watchdog timer status.

```
uint32_t wdt_get_status(
  struct wdt_dev_inst *const dev_inst)
```

Returns Bitmask of watchdog timer status.

6.3.6 Function wdt_init()

Initialize the WDT module.

```
bool wdt_init(
  struct wdt_dev_inst *const dev_inst,
  Wdt *const wdt,
```

```
    struct wdt_config *const cfg)
```

Table 6-7. Parameters

Data direction	Parameter name	Description
[in]	dev_inst	Device structure pointer
[in]	wdt	Base address of the WDT instance
[in]	cfg	Pointer to WDT configuration

Table 6-8. Return Values

Return value	Description
true	if the initialization was successful
false	if initialization failed

6.3.7 Function wdt_reset_mcu()

Reset MCU by generating a WDT reset as soon as possible.

```
bool wdt_reset_mcu(void)
```

Note This function will not work if the SFV bit was previously set (The WDT configuration has previously been locked).

Returns **false** Cannot reset MCU with a WDT reset, or the MCU will be reset.

6.4 Enumeration Definitions

6.4.1 Enum wdt_period

Enum for the possible period settings of the Watchdog timer module, for values requiring a period as a number of Watchdog timer clock ticks.

Formula for Ttimeout = 2pow(PSEL+1) / Fclk_cnt

Formula for Ttimeban = 2pow(TBAN+1) / Fclk_cnt

Table 6-9. Members

Enum value	Description
WDT_PERIOD_NONE	
WDT_PERIOD_MIN_CLK	PSEL = TBAN = 7
WDT_PERIOD_256_CLK	PSEL = TBAN = 7
WDT_PERIOD_512_CLK	PSEL = TBAN = 8
WDT_PERIOD_1024_CLK	PSEL = TBAN = 9
WDT_PERIOD_2048_CLK	PSEL = TBAN = 10
WDT_PERIOD_4096_CLK	PSEL = TBAN = 11
WDT_PERIOD_8192_CLK	PSEL = TBAN = 12

Enum value	Description
WDT_PERIOD_16384_CLK	PSEL = TBAN = 13
WDT_PERIOD_32768_CLK	PSEL = TBAN = 14
WDT_PERIOD_65536_CLK	PSEL = TBAN = 15
WDT_PERIOD_131072_CLK	PSEL = TBAN = 16
WDT_PERIOD_262144_CLK	PSEL = TBAN = 17
WDT_PERIOD_524288_CLK	PSEL = TBAN = 18
WDT_PERIOD_1048576_CLK	PSEL = TBAN = 19
WDT_PERIOD_2097152_CLK	PSEL = TBAN = 20
WDT_PERIOD_4194304_CLK	PSEL = TBAN = 21
WDT_PERIOD_8388608_CLK	PSEL = TBAN = 22
WDT_PERIOD_16777216_CLK	PSEL = TBAN = 23
WDT_PERIOD_33554432_CLK	PSEL = TBAN = 24
WDT_PERIOD_67108864_CLK	PSEL = TBAN = 25
WDT_PERIOD_134217728_CLK	PSEL = TBAN = 26
WDT_PERIOD_268435456_CLK	PSEL = TBAN = 27
WDT_PERIOD_536870912_CLK	PSEL = TBAN = 28
WDT_PERIOD_1073741824_CLK	PSEL = TBAN = 29
WDT_PERIOD_2147483648_CLK	PSEL = TBAN = 30
WDT_PERIOD_4294967296_CLK	PSEL = TBAN = 31
WDT_PERIOD_MAX_CLK	PSEL = TBAN = 31

7. Extra Information for SAM4L Watchdog Timer

7.1 Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

Acronym	Definition
API	Application Programming Interface
BSCIF	Backup System Control Interface
MCU	Microcontroller
NVIC	Nested Vectored Interrupt Controller
OSC32	32kHz crystal oscillator
PDBG	Peripheral Debug Register
PSEL	Timeout Prescale Select
RC32	32kHz RC oscillator
RCSYS	System RC oscillator
SFV	Store Final Value
TBAN	Time Ban Prescale Select
WDT	Watchdog Timer

7.2 Dependencies

In order to use this module, other parts of the system must be configured correctly.

- Power Management
- Clocks
- Interrupts
- Debug Operation
- Fuses

The WDT can be enabled at reset. This is controlled by the WDTAUTO fuse.

7.3 Errata

There are no errata related to this driver.

7.4 Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

Changelog
Initial document release

8. Quickstart guide for SAM4L watchdog driver

This is the quickstart guide for the SAM4L watchdog driver, with step-by-step instructions on how to configure and use the driver in a selection of use cases.

The use cases contain several code fragments. The code fragments in the steps for setup can be copied into a custom initialization function, while the steps for usage can be copied into, e.g., the main application function.

8.1 Use Cases

- Basic Use Case
- Reset MCU by the WDT

8.2 Basic Use Case

In this basic use case, the watchdog will use the RCSYS as the source clock and the timeout period will be configured as 0.57 seconds. Then the watchdog will be kicked every 100ms.

8.2.1 Prerequisites

- System Clock Management (Sysclock) for clock control

8.3 Setup Steps

Note The watchdog clock (CLK_WDT) is enabled at reset and the default source is system RC oscillator(RCSYS). If you want to use 32kHz oscillator as watchdog clock, make sure it is enabled first:

```
/* Enable WDT clock source if need */
if ((wdt_ut_stage & WDT_UT_STAGE_MASK) == WDT_UT_STAGE_WM) {
    /* Enable WDT clock source if need */
    if (BPM->BPM_PMCON & BPM_PMCON_CK32S) {
        /* Enable 32K RC oscillator */
        if (!osc_is_ready(OSC_ID_RC32K)) {
            osc_enable(OSC_ID_RC32K);
            osc_wait_ready(OSC_ID_RC32K);
        }
    } else {
        /* Enable external OSC32 oscillator */
        if (!osc_is_ready(OSC_ID_OSC32)) {
            osc_enable(OSC_ID_OSC32);
            osc_wait_ready(OSC_ID_OSC32);
        }
    }
```

8.3.1 Setup Example Code

Add the following code in the application C-file to setup watchdog:

```
// WDT instance
struct wdt_dev_inst wdt_inst;
// WDT configuration
struct wdt_config   wdt_cfg;
```

```
/* Initialize and enable the watchdog */
wdt_get_config_defaults(&wdt_cfg);
```

```
wdt_cfg.timeout_period = WDT_PERIOD_65536_CLK;
wdt_init(&wdt_inst, WDT, &wdt_cfg);
```

8.3.2 Setup Workflow

1. Create variables to store the watchdog instance and configuration:

```
// WDT instance
struct wdt_dev_inst wdt_inst;
// WDT configuration
struct wdt_config   wdt_cfg;
```

2. Get default configuration but change timeout period to 0.57s. (Ttimeout = 2pow(PSEL+1) / Fclk_cnt = 65535 / 115000).

```
wdt_get_config_defaults(&wdt_cfg);
wdt_cfg.timeout_period = WDT_PERIOD_65536_CLK;
```

3. Initialize the watchdog:

```
wdt_init(&wdt_inst, WDT, &wdt_cfg);
```

8.4 Usage Steps

8.4.1 Usage Example Code

Add to, e.g., main loop in application C-file:

```
wdt_enable(&g_wdt_inst);

while (1) {
     wdt_clear(&g_wdt_inst);
     // delay 100ms
}
```

8.4.2 Usage Workflow

1. Enable the watchdog:

```
wdt_enable(&g_wdt_inst);
```

2. Kick the watchdog in every 100ms:

```
while (1) {
     wdt_clear(&g_wdt_inst);
     // delay 100ms
}
```

3. A daemon program is created now. The MCU tasks added after wdt_clear(). are monitored by the watchdog. If the tasks last more than 0.57 seconds, the options of the watchdog timeout period should be adjusted accordingly.

8.5 Reset MCU by the WDT

The MCU can be reset by generating a WDT reset as soon as possible.

```
bool ret;

ret = wdt_reset_mcu();
```

Index

Document Revision History

Doc. Rev.	Date	Comments
42314A	05/2014	Initial document release

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | www.atmel.com