Timers

August 21, 2024

Software Watchdog Timer (SWT)

1 Overview

The Software Watchdog Timer (SWT) is a 32-bit window watchdog timer that enables the system to recover from situations such as:

- Software trapped in a loop
- A bus transaction failing to terminate

In regular operation, SWT requires periodic execution of a watchdog servicing operation. The servicing operation resets the timer to a specified timeout period. If this servicing action does not occur before the timer expires, SWT generates an interrupt or a hardware reset request. You can configure SWT to generate a reset request or an interrupt on the initial timeout. SWT always generates a reset request on a second consecutive timeout

2 Functions

Function Name	SWTControl_Set
Parameter	(const watchdog_register_t* config)
return	void
Description	SWTContains fields for configuring and controlling SWT. The register is read-only if either hard lock or soft lock is enabled (either HLK or SLK is 1).

Function Name	SWTControl_Get
Parameter	(watchdog_register_t* config)
return	void
Description	SWTContains fields for configuring and controlling SWT. The register is read-only if either hard lock or soft lock is enabled (either HLK or SLK is 1).

Function Name	set_Interrupt
Parameter	(uint8_t TIF)
return	void
Description	setTimeout Interrupt Flag

Function Name	get_Interrupt
Parameter	void
return	uint8_t
Description	getTimeout Interrupt Flag

Function Name	set_Timeout
Parameter	(uint32_t WTO)
return	void

Description	setWatchdog Timeout .Watchdog timeout period in clock cycles. When software writes a service sequence or enables SWT, SWT loads the internal 32-bit countdown timer with this value or 3h, whichever is greater.

Function Name	get_Timeout
Parameter	void
return	uint32_t
Description	getWatchdog Timeout .Watchdog timeout period in clock cycles. When software writes a service sequence or enables SWT, SWT loads the internal 32-bit countdown timer with this value or 3h, whichever is greater.

Function Name	set_Window
Parameter	(uint32_t WST)
return	void
Description	setWindow Start Value When you enable window mode (CR[WND]), you can write the service sequence only when the internal timer is less than this value

Function Name	get_Window
Parameter	void
return	uint32_t
Description	getWindow Start Value When you enable window mode (CR[WND]), you can write the service sequence only when the internal timer is less than this value

Function Name	set_Service
Parameter	(uint16_t WSC)
return	void
Description	setInitiates the service operation and resets the watchdog timer. Watchdog Service Code Use this field to service the watchdog and to unlock the Soft Lock (CR[SLK]). To service the watchdog: If SWT is in keyed service mode (CR[SMD]), write two pseudorandom key values to WSC (see Service key generation for details). Otherwise, write the following values to WSC, in the order shown: 1. A602h 2. B480h To unlock the Soft Lock (CR[SLK]), write the following values to WSC, in the order shown: 1. C520h 2. D928h When read, WSC always returns zero.

Function Name	get_Counter_Output
Parameter	void
return	uint32_t
Description	getShows the value of the internal timer when SWT is disabled.Watchdog Count When SWT is disabled (CR[WEN] is 0), CNT shows the value of the internal timer. When SWT is enabled (CR[WEN] is 1), it writes 0 to CNT. Values in this field can lag behind the internal timer value up to six system clock cycles plus eight counter clock cycles. Therefore, the CNT value that is read immediately after disabling SWT may be higher than the actual value of the internal timer.

Function Name	set_Service_Key
Parameter	(uint16_t sk)
return	void
Description	setService Key Holds the previous (or initial) service key value used in Initiate a keyed service sequence. If CR[SMD] is 01b, the next key value to write to SR is (17 * SK + 3) mod 216.

Function Name	get_Service_Key
Parameter	void
return	uint16_t
Description	getService Key Holds the previous (or initial) service key value used in Initiate a keyed service sequence. If CR[SMD] is 01b, the next key value to write to SR is (17 * SK + 3) mod 216.

Function Name	get_Event_Request
Parameter	void
return	uint8_t
Description	getContains the timeout reset request flag.

Function Name	set_Event_Request
Parameter	(uint8_t rrr)
return	void
Description	setReset Request Flag Write 1 to clear the flag and request. Writing 0 has no effect. 0b - No reset request 1b - Any reset request initiated

System Timer Module (STM)

3 Overview

STM supports commonly required system and application software timing functions. STM includes a 32-bit count-up timer and four 32-bit compare channels with a separate interrupt source for each channel. The timer is driven by the STM module clock divided by an 8-bit prescale value (1 to 256).

4 Functions

Function Name	CR_SWT_Set
Parameter	(const STM_Control* config)
return	void
Description	CR_fields for the prescale value, freeze control, and timer enable.

Function Name	CR_SWT_Get
Parameter	(STM_Control* config)
return	void
Description	CR_fields for the prescale value, freeze control, and timer enable.

Function Name	get_Count
Parameter	void
return	uint32_t
Description	getTimer Count The time base for all compare channels. When enabled, the timer count increments at the rate of the module clock divided by the prescale value.

Function Name	set_Count
Parameter	(uint32_t cnt)
return	void
Description	setTimer Count The time base for all compare channels. When enabled, the timer count increments at the rate of the module clock divided by the prescale value.

Function Name	get_Channel_Control
Parameter	(uint8_t chanel)
return	uint8_t
Description	getchannel n of the timer.

Function Name	set_Channel_Control
Parameter	(uint8_t chanel,uint8_t CEN)
return	void
Description	setEnables channel n of the timer.

Function Name	set_Channel_Interrupt
Parameter	(uint8_t CIF,uint8_t chanel)
return	void
Description	setIndicates and the interrupt flag for channel n of the timer.

Function Name	get_Channel_Interrupt
Parameter	(uint8_t chanel)
return	uint8_t
Description	getclears the interrupt flag for channel n of the timer.

Function Name	set_Channel_Compare
Parameter	(uint32_t cmp,uint8_t chanel)
return	void
Description	setChannel Compare If the channel is enabled (CCR n[CEN]), when the timer count (CNT) matches this value, STM asserts the channel IRQ and sets the channel interrupt flag (CIR n[CIF])

Function Name	get_Channel_Compare
Parameter	(uint8_t chanel)
return	uint32_t
Description	getChannel Compare If the channel is enabled (CCR n[CEN]), when the timer count (CNT) matches this value, STM asserts the channel IRQ and sets the channel interrupt flag (CIR n[CIF])

Function Name	get_PIT_MCR
Parameter	void
return	PIT_MCR_t
Description	getthe PIT timer clocks and specifies the behavior of the timers when PIT enters Debug mode.

Function Name	set_PIT_MCR
Parameter	(PIT_MCR_t mcr)
return	void
Description	set the PIT timer clocks and specifies the behavior of the timers when PIT enters Debug mode.

Function Name	get_PIT_Upper_Lifetimer
Parameter	void
return	uint32_t

Description	getLifetimer Value Indicates the timer value of timer 1. This value is the upper 32 bits of the 64-bit lifetimer value. When you read this register at t1, the value of timer 0 at t1 is latched into LTMR64L.

Function Name	get_PIT_Lower_Lifetimer
Parameter	void
return	uint32_t
Description	getLifetimer Value Indicates the timer value of timer 0 at the moment LTMR64H was last read. This value is the lower 32 bits of the 64-bit lifetimer value. This field updates only when LTMR64H is read.

Function Name	get_RTI_Timer_Load_Value_Sync_Status
Parameter	void
return	uint8_t
Description	get**Sync Status** indicates whether the RTI start value is loaded. • **When reading:** – `0b` - Not loaded – `1b` - Loaded • **When writing:** – `0b` - Clears status – `1b` - Clears status

Function Name	set_RTI_Timer_Load_Value_Sync_Status
Parameter	(uint8_t RT_STAT)
return	void
Description	set**Sync Status** indicates whether the RTI start value is loaded. • **When reading:** – `0b` - Not loaded – `1b` - Loaded • **When writing:** – `0b` - Clears status – `1b` - Clears status

Function Name	get_RTI_Timer_Load_Value
Parameter	void
return	uint32_t

Description	getTimer Start Value Specifies the starting RTI timer value. The timer counts down until it reaches 0, then sets the interrupt flag and reloads this value. When you write a new value to this register, the timer does not restart with the new value until the current timing period expires. To terminate the current period and start a new period with the new value, you must disable the timer (write 0 to RTI_TCTRL n[TEN]) and then enable it again (write 1 to RTI_TCTRL n[TEN]).

Function Name	set_RTI_Timer_Load_Value
Parameter	(uint32_t TSV)
return	void
Description	setTimer Start Value Specifies the starting RTI timer value. The timer counts down until it reaches 0, then sets the interrupt flag and reloads this value. When you write a new value to this register, the timer does not restart with the new value until the current timing period expires. To terminate the current period and start a new period with the new value, you must disable the timer (write 0 to RTI_TCTRL n[TEN]) and then enable it again (write 1 to RTI_TCTRL n[TEN]).

Function Name	get_Current_RTI_Timer_Value
Parameter	void
return	uint32_t
Description	getCurrent Timer Value Indicates the current RTI timer value

Function Name	get_RTI_Timer_Control
Parameter	void
return	RTI_control
Description	getControls RTI timer behavior. The RTI may take several RTI clock cycles to enable or update. Therefore, you must wait for at least three RTI clock cycles after RTI configuration before relying on the RTI timer.

Function Name	set_RTI_Timer_Control
Parameter	(RTI_control cfg)
return	void

Description	setControls RTI timer behavior. The RTI may take several RTI clock cycles to enable or update. Therefore, you must wait for at least three RTI clock cycles after RTI configuration before relying on the RTI timer.

Function Name	get_RTI_Timer_Interrupt_Flag
Parameter	void
return	uint8_t
Description	**Timer Interrupt Flag (TIF)** :indicates that the Real-Time Interrupt (RTI) timer period has expired (CVALn[TVL] = 0). If interrupts are enabled , TIF triggers an interrupt request. • **0b** - Timer still counting down. • **1b** - Timer has expired.

Function Name	clear_RTI_Timer_Interrupt_Flag
Parameter	(uint8_t TIF)
return	void
Description	clear the **Timer Interrupt Flag (TIF):** Indicates that the RTI timer period has expired). If interrupts are enabled TIF flag triggers an interrupt request. The flag can be cleared by writing the following values: • **0b** - No effect • **1b** - Clears the flag

Function Name	get_Timer_Load_Value
Parameter	(uint8_t n)
return	uint32_t
Description	getSpecifies the length of the timeout period in clock cycles. The value change is visible immediately. The synchronization mechanism allows 0 wait states in this case. Specifies the starting timer value. The timer counts down until it reaches 0, then sets the interrupt flag and reloads this value. When you write a new value to this register, the timer does not restart with the new value until the current timing period expires. To terminate the current period and start a new period with the new value, you must disable the timer (write 0 to TCTRL n[TEN]) and then enable it again (write 1 to TCTRL n[TEN]).

Function Name	set_Timer_Load_Value
Parameter	(uint32_t tsv,uint8_t n)
return	void
Description	setSpecifies the length of the time-out period in clock cycles. The value change is visible immediately. The synchronization mechanism allows 0 wait states in this case. Specifies the starting timer value. The timer counts down until it reaches 0, then sets the interrupt flag and reloads this value. When you write a new value to this register, the timer does not restart with the new value until the current timing period expires. To terminate the current period and start a new period with the new value, you must disable the timer (write 0 to TCTRL n[TEN]) and then enable it again (write 1 to TCTRL n[TEN]).

Function Name	get_Current_Timer_Value
Parameter	(uint8_t n)
return	uint32_t
Description	getTimer Value Indicates the current timer value.

Function Name	get_Control_timer
Parameter	(uint8_t n)
return	Control_timer
Description	getControls timer behavior.

Function Name	set_Control_timer
Parameter	(Control_timer ctrl)
return	void
Description	set Controls timer behavior.

Function Name	get_Timer_Flag
Parameter	(uint8_t n)
return	uint8_t
Description	get**Timer Interrupt Flag (TIF)** Indicates the timer period has expired (`CVAL`$_n$`[TVL] = 0`). If interrupts are enabled (`TCTRL`$_n$`[TIE] = 1`), `TIF` triggers an interrupt request. This field behaves differently for register reads and writes. • **When reading:** – `0b` - Timer has not expired – `1b` - Timer expired

Function Name	clear_Timer_Flag
Parameter	(uint8_t TIF,uint8_t n)
return	void
Description	cle**Timer Interrupt Flag (TIF)** Indicates the timer period has expired (`CVAL`$_n$`[TVL] = 0`). If interrupts are enabled (`TCTRL`$_n$`[TIE] = 1`), `TIF` triggers an interrupt request. This field behaves differently for register reads and writes. • **When writing:** – `0b` - No effect – `1b` - Clears flag

Real Time Clock (RTC)

5 Overview

The Real-Time Clock (RTC) is a free-running counter used for time keeping applications. The RTC can be configured to generate an interrupt at a pre-defined interval independent of the mode of operation (run mode or low power mode). If in a low power mode, the RTC interval is reached, the RTC first generates a wakeup and then asserts the interrupt request. The RTC also supports an API function used to generate a periodic wakeup request to exit a low-power mode or an interrupt request.

6 Functions

Function Name	get_PIT_MCR
Parameter	void
return	PIT_MCR_t
Description	getRTC Supervisor Bit 0b - All registers are accessible in both user as well as supervisor mode 1b - All other registers are accessible in the supervisor mode only

Function Name	set_PIT_MCR
Parameter	(PIT_MCR_t mcr)
return	void
Description	setRTC Supervisor Bit 0b - All registers are accessible in both user as well as supervisor mode 1b - All other registers are accessible in the supervisor mode only

Function Name	get_PIT_Upper_Lifetimer
Parameter	void
return	uint32_t

Description	get **CNTEN - Counter Enable** The CNTEN bit enables the RTC counter. Setting CNTEN to 0b asynchronously resets all RTC and API logic, allowing RTC configuration and clock source selection without synchronization issues. CNTEN should be disabled when INV_RTC and INV_API are cleared. **RTCIE - RTC Interrupt Enable** The RTCIE bit enables interrupt requests to the system if RTCF is asserted. **FRZEN - Freeze Enable Bit** The FRZEN bit causes the counter to freeze at the last valid count value upon entering debug mode. When debug mode ends, the counter resumes from the frozen value. This bit should not be changed while in debug mode. **ROVREN - Counter Roll Over Wakeup/Interrupt Enable** The ROVREN bit enables wakeup and interrupt requests when the RTC rolls over from 0xFFFF_FFFF to 0x0000_0000. The RTCIE bit must also be set to generate an interrupt from a counter rollover. **APIEN - Autonomous Periodic Interrupt Enable** The APIEN bit enables the autonomous periodic interrupt function. Disabling APIEN also disables API wakeup output from the RTC. **APIIE - API Interrupt Enable** The APIIE bit enables interrupt requests to the system if APIF is asserted. **CLKSEL - Clock Select** The CLKSEL[1:0] bits select the clock source for the RTC. CLKSEL can only be updated when CNTEN is 0. Ensure that the oscillator is enabled before selecting it as a clock source for the RTC. **DIV512EN - Divide by 512 Enable** The DIV512EN bit enables the 512 clock divider. DIV512EN can only be updated when CNTEN is 0. **DIV32EN - Divide by 32 Enable** The DIV32EN bit enables the 32 clock divider. DIV32EN can only be updated when CNTEN is 0. **TRIG_EN - Trigger Enable for Analog Comparator** The TRIG_EN bit enables the trigger function for the analog comparator.

Function Name	get_PIT_Lower_Lifetimer
Parameter	void
return	uint32_t
Description	setsame as get_RTCC

Function Name	get_RTI_Timer_Load_Value_Sync_Status
Parameter	void
return	uint8_t
Description	get**RTC Interrupt Flag** The `RTCF` bit indicates that the RTC counter has reached the counter value matching the RTC Compare Value register (`RTCVAL`). `RTCF` is cleared by writing a `1` to `RTCF`. Writing a `0` to `RTCF` has no effect. • **0b** - RTC counter is not equal to `RTCVAL` • **1b** - RTC counter matches `RTCVAL` **Invalid RTC Write** This bit returns value `1` after a value is written to the `RTCVAL` register and the synchronization process is in progress. During this synchronization period, any attempt to write to the `RTCVAL` register again is ignored. Synchronization will complete only when `CNTEN` is set. **Invalid APIVAL Write** This bit returns value `1` after a value is written to the `APIVAL` register and the synchronization process is in progress. During this synchronization period, any attempt to write to the `APIVAL` register again is ignored. Synchronization will complete only when `CNTEN` is set. **API Interrupt Flag** The `APIF` bit indicates that the RTC counter has reached the counter value matching the API offset value. `APIF` is cleared by writing a `1` to `APIF`. Writing a `0` to `APIF` has no effect. • **0b** - Counter is not equal to API offset value • **1b** - Counter matches the API offset value **Counter Roll Over Interrupt Flag** The `ROVRF` bit indicates that the RTC has rolled over from `0xFFFF_FFFF` to `0x0000_0000`. `ROVRF` is cleared by writing a `1` to `ROVRF`. • **0b** - RTC has not rolled over • **1b** - RTC has rolled over

Function Name	set_RTI_Timer_Load_Value_Sync_Status
Parameter	(uint8_t RT_STAT)
return	void
Description	set same as get_RTC_STATUS

Function Name	get_RTI_Timer_Load_Value
Parameter	void
return	uint32_t
Description	getRTC Counter Value Because of clock synchronization, the RTCCNT value may represent a previous counter value

Function Name	set_RTI_Timer_Load_Value
Parameter	(uint32_t TSV)
return	void
Description	getAPI Compare Value APIVAL bits are added to the current count to calculate an offset. The APIVAL offset bits are compared to the RTC counter bits and if a match occurs, an interrupt/wakeup request is asserted.

Function Name	get_Current_RTI_Timer_Value
Parameter	void
return	uint32_t
Description	setAPI Compare Value APIVAL bits are added to the current count to calculate an offset. The APIVAL offset bits are compared to the RTC counter bits and if a match occurs, an interrupt/wakeup request is asserted.

Function Name	get_RTI_Timer_Control
Parameter	void
return	RTI_control
Description	getRTC Compare Value The RTCVAL bits are compared to the RTC counter bits and if a match occurs, RTCF is set.

Function Name	set_RTI_Timer_Control
Parameter	(RTI_control cfg)
return	void
Description	setRTC Compare Value The RTCVAL bits are compared to the RTC counter bits and if a match occurs, RTCF is set.