**[Overview](https://opentitan.org/book/hw/ip/aon_timer/index.html" \l "overview)**

This document specifies the Always-On (“AON”) Timer IP functionality. This module conforms to the [Comportable guideline for peripheral functionality.](https://opentitan.org/book/doc/contributing/hw/comportability/index.html) See that document for an overview of how it is integrated into the top level system.

[**Features**](https://opentitan.org/book/hw/ip/aon_timer/index.html#features)

* Two 32-bit upcounting timers: one timer functions as a wakeup timer, one as a watchdog timer
* The watchdog timer has two thresholds: bark (generates an interrupt) and bite (resets core)
* There is 12 bit pre-scaler for the wakeup timer to enable very long timeouts

[**Description**](https://opentitan.org/book/hw/ip/aon_timer/index.html#description)

[**AON Wakeup timer**](https://opentitan.org/book/hw/ip/aon_timer/index.html#aon-wakeup-timer)

The always-on wakeup timer operation is straightforward. A count starts at 0 and slowly ticks upwards (one tick every N + 1 clock cycles, where N is the pre-scaler value). When it reaches / exceeds the wake threshold, a level wakeup signal is sent to the power manager and a level IRQ is sent to the processor. This wakeup signal stays high until it is explicitly acknowledged by software. To clear the wakeup write 0 to the [WKUP\_CAUSE](https://opentitan.org/book/hw/ip/aon_timer/data/aon_timer.hjson#wkup_cause) register. To clear the interrupt write 1 to [INTR\_STATE.wkup\_timer\_expired](https://opentitan.org/book/hw/ip/aon_timer/data/aon_timer.hjson#intr_state). Note that if [WKUP\_COUNT](https://opentitan.org/book/hw/ip/aon_timer/data/aon_timer.hjson#wkup_count) is not zeroed and remains at or above the wake threshold and the wakeup timer isn’t disabled, the wakeup and interrupt will trigger again at the next clock tick. The wakeup timer can be used like a real-time clock for long periods in a low-power mode (though it does not give any guarantees of time-accuracy). **TODO: specify accuracy**

[**AON Watchdog timer**](https://opentitan.org/book/hw/ip/aon_timer/index.html#aon-watchdog-timer)

The always-on watchdog timer behaves similarly to the wakeup timer. It has an independent count starting at 0 which slowly ticks upwards. When the first threshold is met or exceeded, a level wakeup signal (if enabled) is sent to the power manager. Simultaneously, a level IRQ signal is also generated to the processor.

If the system is in a low power state, the wakeup signal asks the power manager to wake the system so that the IRQ can be serviced. If the system is not in a low power mode, the IRQ is immediately serviced. Both the wakeup and the IRQ signals remain asserted until system reset or explicit acknowledgement by software. This first threshold is known as the watchdog bark.

An extra interrupt output is available to connect the watchdog bark output to a non-maskable interrupt pin if required.

When the second threshold is met (this is known as the watchdog bite), a reset request is sent to the power manager which will trigger a system reset. This is independent of the IRQ sent as part of the watchdog bark. The system reset also resets the always-on timer, so software is not required to directly acknowledge anything after a watchdog reset.

To prevent the watchdog bark or bite, software is expected to periodically reset the count when operating normally. This is referred to as petting the watchdog, and is achieved by resetting the count to zero.

Since this timer functions as a watchdog, it has three additional functions not present in the always-on wakeup timer:

* Watchdog configuration lock
* Watchdog pause in sleep
* Watchdog pause during system escalation

Unlike the wakeup timer, the watchdog timer configuration can be locked by firmware until the next system reset. This allows the option of preventing firmware from accidentally or maliciously disabling the watchdog.

The “pause in sleep” option controls whether the watchdog timer continues to count in low-power modes. This allows configurations where the watchdog timer can remain programmed and locked while the device is put to sleep for relatively long periods, controlled by the wakeup timer. Without this feature, the watchdog timer might wake up the core prematurely by triggering a watchdog bark.

The “pause during escalation” feature ensures that watchdog bites and barks do not interfere with system escalation behavior. If during escalation software configures the system to hang instead of reset, the watchdog bite cannot supersede that decision.

**Theory of Operation**

The timer interacts with the CPU core and the power manager and reset manager to drive wakeup / reset events and interrupts. There is also an extra input to tell the counter whether to run ("counter-run"). This is used to stop the watchdog timer running when in debugging mode or when the alert handler has put the system in a "killed" state.

**Design Details**

The always-on timer will run on a ~200KHz clock. The wakeup timer is 64b wide and the watchdog timer 32b wide. This gives a maximum timeout window of roughly ~6 hours for the watchdog and almost 3 million years for the wakeup timer. For the wakeup timer, the pre-scaler can be used to slow the count down so the counter value matches some desired time unit (e.g. milliseconds).

Register reads via the TLUL interface are synchronized to the slow clock using the "async" register generation feature. This means that writes can complete before the data has reached its underlying register in the slow clock domain. If software needs to guarantee completion of a register write, it can read back the register value (which will guarantee the completion of all previous writes to the peripheral).

The wakeup count and threshold are both 64-bit values accessed through two 32-bit registers. It is not possible to do a single atomic read or write of the 64-bit values. The programmer's guide suggests some ways to access the wakeup count and threshold to avoid issues from race conditions caused by this.

**Initialization**

1. Set the timer values [WKUP\_COUNT\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_lo), [WKUP\_COUNT\_HI](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_hi) and [WDOG\_COUNT](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wdog_count) to zero.
2. Program the desired wakeup pre-scaler value in [WKUP\_CTRL](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_ctrl).
3. Program the desired thresholds in [WKUP\_THOLD\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_thold_lo), [WKUP\_THOLD\_HI](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_thold_hi), [WDOG\_BARK\_THOLD](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wdog_bark_thold) and [WDOG\_BITE\_THOLD](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wdog_bite_thold).
4. Set the enable bit to 1 in the [WKUP\_CTRL](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_ctrl) / [WDOG\_CTRL](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wdog_ctrl) registers.
5. If desired, lock the watchdog configuration by writing 0 to the regwen bit in [WDOG\_REGWEN](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wdog_regwen).

**Watchdog pet**

Pet the watchdog by writing zero to the [WDOG\_COUNT](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wdog_count) register.

**Wakeup count and threshold access**

The wakeup counter and threshold are both 64-bit values accessed via two 32-bit hi and lo registers. It is not possible to read or modify the 64-bit values in a single atomic access. Care must be taken to avoid issues due to race conditions caused by this. Below are some recommendations on how to access the counter and threshold to avoid problems.

**Reading the counter**

The counter might increment between the read of [WKUP\_COUNT\_HI](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_hi) and the read of [WKUP\_COUNT\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_lo). If the [WKUP\_COUNT\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_lo) value overflows between the two register reads the combined 64-bit value may be incorrect. Consider the scenario where the 64-bit counter value is 0x1\_ffff\_ffff. A read of the [WKUP\_COUNT\_HI](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_hi) value gives 0x1. If the counter then increments to 0x2\_0000\_0000 then a read of [WKUP\_COUNT\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_lo) gives 0x0000\_00000. The final 64-bit value of 0x1\_0000\_0000 is incorrect. The pseudo code below provides a method to avoid this issue:

counter\_hi = REG\_READ(WKUP\_COUNT\_HI);

counter\_lo = REG\_READ(WKUP\_COUNT\_LO);

counter\_hi\_2 = REG\_READ(WKUP\_COUNT\_HI);

// If WKUP\_COUNT\_LO overflowed between first and second read WKUP\_COUNT\_HI will

// have changed

if counter\_hi != counter\_hi\_2 {

// Read new WKUP\_COUNT\_LO value and use second WKUP\_COUNT\_HI read as top 32 bits

counter\_lo = REG\_READ(WKUP\_COUNT\_LO);

counter\_hi = counter\_hi\_2;

}

counter\_full = counter\_hi << 32 | counter\_lo;

**Writing the counter**

Between the two count register ([WKUP\_COUNT\_HI](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_hi) and [WKUP\_COUNT\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_lo)) writes the counter may increment. If the [WKUP\_COUNT\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_lo) value overflows between a [WKUP\_COUNT\_HI](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_hi) and [WKUP\_COUNT\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_lo) write the intended counter value may be incorrect. For example an attempt to clear the counter to 0 could result in a counter value of 0x1\_0000\_0000. It is recommended the wakeup timer is disabled with [WKUP\_CTRL](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_ctrl) before writing to the [WKUP\_COUNT\_HI](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_hi) and [WKUP\_COUNT\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_lo) registers to avoid this problem.

**Reading the threshold**

The hardware does not alter the value of the [WKUP\_THOLD\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_thold_lo) and [WKUP\_THOLD\_HI](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_thold_hi) registers so there are no race conditions in reading them.

**Writing the threshold**

When writing to [WKUP\_THOLD\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_thold_lo) and [WKUP\_THOLD\_HI](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_thold_hi) between the two writes the 64-bit threshold is effectively an interim value that's not intended to be the real threshold. It is possible the interim threshold is lower than the previous threshold triggering a spurious wakeup. Use the method in the pseudo code below to avoid this issue:

disable\_wakeup\_interrupt();

// Guaranteed 64-bit threshold greater than or equal to old threshold. This

// prevents an interrupt caused by the threshold decreasing.

REG\_WRITE(WKUP\_THOLD\_LO, UINT32\_MAX);

// Guaranteed 64-bit threshold greater than or equal to intended threshold. If

// the counter reaches this value before we've completing the final write then

// the interrupt would have happened with the intended threshold as well.

REG\_WRITE(WKUP\_THOLD\_HI, new\_thold >> 32);

// 64-bit threshold now intended value

REG\_WRITE(WKUP\_THOLD\_LO, new\_thold & 0xffff\_ffff);

enable\_wakeup\_interrupt();

**Interrupt Handling**

If either timer reaches the programmed threshold, interrupts are generated from the AON\_TIMER module. Disable the wakeup timer by clearing the enable bit in [WKUP\_CTRL](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_ctrl). Reset the timer if desired by clearing [WKUP\_COUNT\_HI](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_hi) and [WKUP\_COUNT\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_lo) and renable by setting the enable bit in [WKUP\_CTRL](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_ctrl). Clear the interrupt by writing 1 into the Interrupt Status Register [INTR\_STATE](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#intr_state).

If the timer has caused a wakeup event ([WKUP\_CAUSE](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_cause) is set) then clear the wakeup request by writing 0 to [WKUP\_CAUSE](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_cause).

If {[WKUP\_COUNT\_HI](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_hi), [WKUP\_COUNT\_LO](https://github.com/lowRISC/opentitan/blob/master/hw/ip/aon_timer/doc/registers.md#wkup_count_lo)} remains above the threshold after clearing the interrupt or wakeup event and the timer remains enabled, the interrupt and wakeup event will trigger again at the next clock tick.