23. Interval Timer Core

23.1. Core Overview

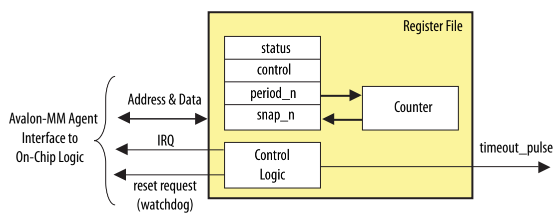
The Interval Timer core with Avalon interface is an interval timer for Avalon-based processor systems, such as a Nios II processor system. The core provides the following features:

- 32-bit and 64-bit counters.
- Controls to start, stop, and reset the timer.
- Two count modes: count down once and continuous count-down.
- Count-down period register.
- Option to enable or disable the interrupt request (IRQ) when timer reaches zero.
- Optional watchdog timer feature that resets the system if timer ever reaches zero.
- Optional periodic pulse generator feature that outputs a pulse when timer reaches zero.
- Compatible with 32-bit and 16-bit processors.

  Device drivers are provided in the HAL system library for the Nios II processor.

23.2. Functional Description

Figure 72. Interval Timer Core Block Diagram



The interval timer core has two user-visible features:

- The Avalon Memory-Mapped (Avalon-MM) interface that provides access to six 16-bit registers
- An optional pulse output that can be used as a periodic pulse generator

  All registers are 16-bits wide, making the core compatible with both 16-bit and 32-bit processors. Certain registers only exist in hardware for a given configuration. For example, if the core is configured with a fixed period, the period registers do not exist in hardware.

  The following sequence describes the basic behavior of the interval timer core:

- An Avalon-MM host peripheral, such as a Nios II processor, writes the core's `control` register to perform the following tasks:
  - Start and stop the timer
  - Enable/disable the IRQ
  - Specify count-down once or continuous count-down mode
- A processor reads the `status` register for information about current timer activity.
- A processor can specify the timer period by writing a value to the period registers.
- An internal counter counts down to zero, and whenever it reaches zero, it is immediately reloaded from the period registers.
- A processor can read the current counter value by first writing to one of the snap registers to request a coherent snapshot of the counter, and then reading the snap registers for the full value.
- When the count reaches zero, one or more of the following events are triggered:
  - If IRQs are enabled, an IRQ is generated.
  - The optional pulse-generator output is asserted for one clock period.
  - The optional watchdog output resets the system.

23.2.1. Avalon-MM Agent Interface

The interval timer core implements a simple Avalon-MM agent interface to provide access to the register file. The Avalon-MM agent port uses the `resetrequest` signal to implement watchdog timer behavior. This signal is a non-maskable reset signal, and it drives the reset input of all Avalon-MM peripherals. When the `resetrequest` signal is asserted, it forces any processor connected to the system to reboot. For more information, refer to **Configuring the Timer as a Watchdog Timer**.

23.3. Configuration

This section describes the options available in the Platform Designer.

23.3.1. Timeout Period

The **Timeout Period** setting determines the initial value of the period registers. When the **Writeable period** option is on, a processor can change the value of the period by writing to the period registers. When the **Writeable period** option is off, the period is fixed and cannot be updated at runtime. See the **Hardware Options** section for information on register options.

The **Timeout Period** is an integer multiple of the **Timer Frequency**. The **Timer Frequency** is fixed at the frequency setting of the system clock associated with the timer. The **Timeout Period** setting can be specified in units of **µs** (microseconds), **ms** (milliseconds), **seconds** , or **clocks** (number of cycles of the system clock associated with the timer). The actual period depends on the frequency of the system clock associated with the timer. If the period is specified in **µs**, **ms**, or **seconds**, the true period will be the smallest number of clock cycles that is greater or equal to the specified **Timeout Period** value. For example, if the associated system clock has a frequency of 30 **ns**, and the specified **Timeout Period** value is 1 **µs**, the true timeout period will be 1.020 microseconds.

23.3.2. Counter Size

The **Counter Size** setting determines the timer's width, which can be set to either 32 or 64 bits. A 32-bit timer has two 16-bit period registers, whereas a 64-bit timer has four 16-bit period registers. This option applies to the snap registers as well.

23.3.3. Hardware Options

The following options affect the hardware structure of the interval timer core. As a convenience, the **Preset Configurations** list offers several pre-defined hardware configurations, such as:

- **Simple periodic interrupt**—This configuration is useful for systems that require only a periodic IRQ generator. The period is fixed and the timer cannot be stopped, but the IRQ can be disabled.
- **Full-featured**—This configuration is useful for embedded processor systems that require a timer with variable period that can be started and stopped under processor control.
- **Watchdog**—This configuration is useful for systems that require watchdog timer to reset the system in the event that the system has stopped responding. Refer to the **Configuring the Timer as a Watchdog Timer** section.

Register Options

Table 253. Register Options

Option	Description
Writeable period	When this option is enabled, a host peripheral can change the count-down period by writing to the period registers. When disabled, the count-down period is fixed at the specified **Timeout Period**, and the period registers do not exist in hardware.
Readable snapshot	When this option is enabled, a host peripheral can read a snapshot of the current count-down. When disabled, the status of the counter is detectable only via other indicators, such as the `status` register or the IRQ signal. In this case, the snap registers do not exist in hardware, and reading these registers produces an undefined value.
Start/Stop control bits	When this option is enabled, a host peripheral can start and stop the timer by writing the START and STOP bits in the `control` register. When disabled, the timer runs continuously. When the **System reset on timeout (watchdog)** option is enabled, the START bit is also present, regardless of the **Start/Stop control bits** option.

Output Signal Options

Table 254. Output Signal Options

Option	Description
Timeout pulse (1 clock wide)	When this option is on, the core outputs a signal `timeout_pulse`. This signal pulses high for one clock cycle whenever the timer reaches zero. When this option is off, the `timeout_pulse` signal does not exist.
System reset on timeout (watchdog)	When this option is on, the core's Avalon-MM agent port includes the `resetrequest` signal. This signal pulses high for one clock cycle whenever the timer reaches zero resulting in a system-wide reset. The internal timer is stopped at reset. Explicitly writing the START bit of the `control` register starts the timer. When this option is off, the `resetrequest` signal does not exist. Refer to the **Configuring the Timer as a Watchdog Timer** section.

23.3.4. Configuring the Timer as a Watchdog Timer

To configure the core for use as a watchdog, in the MegaWizard Interface select **Watchdog** in the **Preset Configurations** list, or choose the following settings:

- Set the **Timeout Period** to the desired "watchdog" period.
- Turn off **Writeable period**.
- Turn off **Readable snapshot**.
- Turn off **Start/Stop control bits**.
- Turn off **Timeout pulse**.
- Turn on **System reset on timeout (watchdog)**.

  A watchdog timer wakes up (comes out of reset) stopped. A processor later starts the timer by writing a 1 to the `control` register's START bit. Once started, the timer can never be stopped. If the internal counter ever reaches zero, the watchdog timer resets the system by generating a pulse on its `resetrequest` output. The `resetrequest` pulse will last for two cycles before the incoming reset signal deasserts the pulse. To prevent an indefinite `resetrequest` pulse, you are required to connect the `resetrequest` signal back to the reset input of the timer.

  To prevent the system from resetting, the processor must periodically reset the timer's count-down value by writing one of the period registers (the written value is ignored). If the processor fails to access the timer because, for example, software stopped executing normally, the watchdog timer resets the system and returns the system to a defined state.

23.4. Software Programming Model

The following sections describe the software programming model for the interval timer core, including the register map and software declarations to access the hardware. For Nios II processor users, Intel provides hardware abstraction layer (HAL) system library drivers that enable you to access the interval timer core using the HAL application programming interface (API) functions.

23.4.1. HAL System Library Support

The Intel-provided drivers integrate into the HAL system library for Nios II systems. When possible, HAL users should access the core via the HAL API, rather than accessing the core's registers directly.

Intel provides a driver for both the HAL timer device models: system clock timer, and timestamp timer.

System Clock Driver

When configured as the system clock, the interval timer core runs continuously in periodic mode, using the default period set. The system clock services are then run as a part of the interrupt service routine for this timer. The driver is interrupt-driven, and therefore must have its interrupt signal connected in the system hardware.

The Nios II integrated development environment (IDE) allows you to specify system library properties that determine which timer device will be used as the system clock timer.

Timestamp Driver

The interval timer core may be used as a timestamp device if it meets the following conditions:

- The timer has a writeable `period` register, as configured in Platform Designer.
- The timer is not selected as the system clock.

  The Nios II IDE allows you to specify system library properties that determine which timer device will be used as the timestamp timer.

  If the timer hardware is not configured with writeable `period` registers, calls to the `alt_timestamp_start()` API function will not reset the timestamp counter. All other HAL API calls will perform as expected.

  For more information about using the system clock and timestamp features that use these drivers, refer to the **Nios II Software Developer’s Handbook**. The Nios II Embedded Design Suite (EDS) also provides several example designs that use the interval timer core.

Limitations

The HAL driver for the interval timer core does not support the watchdog reset feature of the core.

23.4.2. Software Files

The interval timer core is accompanied by the following software files. These files define the low-level interface to the hardware, and provide the HAL drivers. Application developers should not modify these files.

- **altera_avalon_timer_regs.h—**This file defines the core's register map, providing symbolic constants to access the low-level hardware.
- `altera_avalon_timer.h`, **altera_avalon_timer_sc.c, altera_avalon_timer_ts.c, altera_avalon_timer_vars.c—**These files implement the timer device drivers for the HAL system library.

23.4.3. Register Map

You do not need to access the interval timer core directly via its registers if using the standard features provided in the HAL system library for the Nios II processor. In general, the register map is only useful to programmers writing a device driver.

The Intel-provided HAL device driver accesses the device registers directly. If you are writing a device driver, and the HAL driver is active for the same device, your driver will conflict and fail to operate correctly.

The table below shows the register map for the 32-bit timer. The interval timer core uses native address alignment. For example, to access the `control` register value, use offset 0x4.

Table 255. Register Map—32-bit Timer

Offset	Name	R/W	Description of Bits						
			15	...	4	3	2	1	0
0	`status`	RW	(1)					RUN	TO
1	`control`	RW	(1)			STOP	START	CONT	ITO
2	`periodl`	RW	Timeout Period – 1 (bits [15:0])						
3	`periodh`	RW	Timeout Period – 1 (bits [31:16])						
4	`snapl`	RW	Counter Snapshot (bits [15:0])						
5	`snaph`	RW	Counter Snapshot (bits [31:16])						

Notes :
1. Reserved. Read values are undefined. Write zero.

For more information about native address alignment, refer to the System Interconnect Fabric for Memory-Mapped Interfaces.

Table 256. Register Map—64-bit Timer

Offset	Name	R/W	Description of Bits						
			15	...	4	3	2	1	0
0	`status`	RW	(1)					RUN	TO
1	`control`	RW	(1)			STOP	START	CONT	ITO
2	`period_0`	RW	Timeout Period – 1 (bits [15:0])						
3	`period_1`	RW	Timeout Period – 1 (bits [31:16])						
4	`period_2`	RW	Timeout Period – 1 (bits [47:32])						
5	`period_3`	RW	Timeout Period – 1 (bits [63:48])						
6	`snap_0`	RW	Counter Snapshot (bits [15:0])						
7	`snap_1`	RW	Counter Snapshot (bits [31:16])						
8	`snap_2`	RW	Counter Snapshot (bits [47:32])						
9	`snap_3`	RW	Counter Snapshot (bits [63:48])						

Notes :
1. Reserved. Read values are undefined. Write zero.

status Register

The status register has two defined bits.

Table 257. status Register Bits

Bit	Name	R/W/C	Description
0	TO	R/WC	The TO (timeout) bit is set to 1 when the internal counter reaches zero. Once set by a timeout event, the TO bit stays set until explicitly cleared by a host peripheral. Write 0 or 1 to the status register to clear the TO bit.
1	RUN	R	The RUN bit reads as 1 when the internal counter is running; otherwise this bit reads as 0. The RUN bit is not changed by a write operation to the status register.

control Register

The control register has four defined bits.

Table 258. control Register Bits

Bit	Name	R/W/C	Description
0	ITO	RW	If the ITO bit is 1, the interval timer core generates an IRQ when the status register's TO bit is 1. When the ITO bit is 0, the timer does not generate IRQs.
1	CONT	RW	The CONT (continuous) bit determines how the internal counter behaves when it reaches zero. If the CONT bit is 1, the counter runs continuously until it is stopped by the STOP bit. If CONT is 0, the counter stops after it reaches zero. When the counter reaches zero, it reloads with the value stored in the period registers, regardless of the CONT bit.
2	START (1)	W	Writing a 1 to the START bit starts the internal counter running (counting down). The START bit is an event bit that enables the counter when a write operation is performed. If the timer is stopped, writing a 1 to the START bit causes the timer to restart counting from the number currently stored in its counter. If the timer is already running, writing a 1 to START has no effect. Writing 0 to the START bit has no effect.
3	STOP (1)	W	Writing a 1 to the STOP bit stops the internal counter. The STOP bit is an event bit that causes the counter to stop when a write operation is performed. If the timer is already stopped, writing a 1 to STOP has no effect. Writing a 0 to the stop bit has no effect. If the timer hardware is configured with **Start/Stop control bits** off, writing the STOP bit has no effect.

Notes :
1. Writing 1 to both START and STOP bits simultaneously produces an undefined result.

period_n Registers

The period_n registers together store the timeout period value. The internal counter is loaded with the value stored in these registers whenever one of the following occurs:

- A write operation to one of the period_n register
- The internal counter reaches 0

  The timer's actual period is one cycle greater than the value stored in the period_n registers because the counter assumes the value zero for one clock cycle.

  Writing to one of the period_n registers stops the internal counter, except when the hardware is configured with **Start/Stop control bits** off. If **Start/Stop control bits** is off, writing either register does not stop the counter. When the hardware is configured with **Writeable period** disabled, writing to one of the period_n registers causes the counter to reset to the fixed **Timeout Period** specified at system generation time.

Note: A timeout period value of 0 is not a supported use case. Software configures timeout period values greater than 0.

snap_n Registers

A host peripheral may request a coherent snapshot of the current internal counter by performing a write operation (write-data ignored) to one of the snap_n registers. When a write occurs, the value of the counter is copied to snap_n registers. The snapshot occurs whether or not the counter is running. Requesting a snapshot does not change the internal counter's operation.

23.4.4. Interrupt Behavior

The interval timer core generates an IRQ whenever the internal counter reaches zero and the ITO bit of the control register is set to 1. Acknowledge the IRQ in one of two ways:

- Clear the TO bit of the status register
- Disable interrupts by clearing the ITO bit of the control register

  Failure to acknowledge the IRQ produces an undefined result.

23.5. Interval Time Core API

Table 259. alt_timestamp_start

Prototype	int alt_timestamp_start(void)
Include	<alt_timestamp.h>
Parameters	-
Returns	The return value is 0 upon success and -1 if timestamp device has not been registered
Description	Initialize the timestamp feature.

Table 260. alt_timestamp

Prototype	alt_timestamp_type alt_timestamp (void)
Include	<alt_timestamp.h>
Parameters	-
Returns	This function returns the current timestamp count. It returns -1 if the timer has run full period, or there is no timestamp available.
Description	Returns the current timestamp count.

Table 261. alt_timestamp_freq

Prototype	alt_u32 alt_timestamp_freq(void)
Include	<alt_timestamp.h>
Parameters	-
Returns	This function returns the number of timestamp ticks per second. This will be 0 if no timestamp device has been registered.
Description	Returns the number of timestamp ticks per second.

23.6. Interval Timer Core Revision History

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1	Added a new section: *Interval Time Core API*.
2018.05.07	18.0	Implemented editorial enhancements.

Date	Version	Changes
June 2015	2015.06.12	Updated "status Register Bits" table.
July 2014	2014.07.24	Removed mention of SOPC Builder, updated to Platform Designer
December 2013	v13.1.0	Updated the reset pulse description in the **Configuring the Timer as a Watchdog Timer** section.
December 2010	v10.1.0	Removed the "Device Support", "Instantiating the Core in SOPC Builder", and "Referenced Documents" sections.
July 2010	v10.0.0	No change from previous release.
November 2009	v9.1.0	Revised descriptions of register fields and bits.
March 2009	v9.0.0	No change from previous release.
November 2008	v8.1.0	Changed to 8-1/2 x 11 page size. Updated the core's name to reflect the name used in SOPC Builder.
May 2008	v8.0.0	Added a new parameter and register map for the 64-bit timer.