14. Timer Core

NII51008-7.1.0

Core Overview

The timer core with Avalon® interface is a 32-bit interval timer for Avalon-based processor systems, such as a Nios® II processor system. The timer provides the following features:

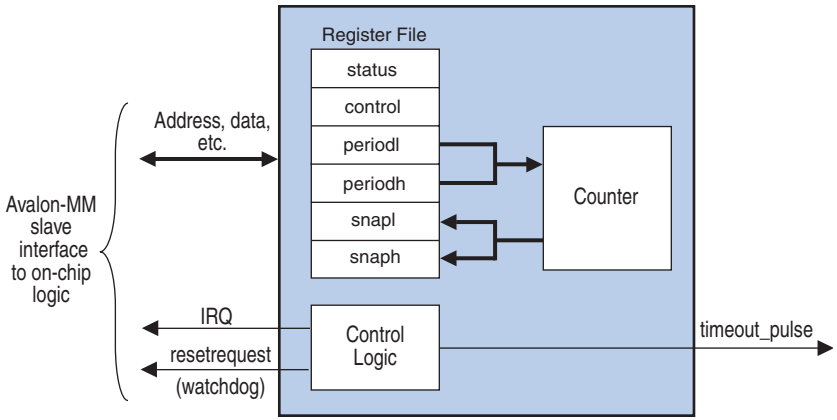
- Controls to start, stop, and reset the timer
- Two count modes: count down once and continuous count-down
- Count-down period register
- Maskable interrupt request (IRQ) upon reaching zero
- Optional watchdog timer feature that resets the system if timer ever reaches zero
- Optional periodic pulse 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. The timer core is SOPC Builder-ready and integrates easily into any SOPC Builder-generated system. This chapter contains the following sections:

Functional Description

Figure 14–1 shows a block diagram of the timer core.

Figure 14–1. Timer Core Block Diagram



The 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 timer compatible with both 16-bit and 32-bit processors. Certain registers only exist in hardware for a given configuration. For example, if the timer is configured with a fixed period, the period registers do not exist in hardware.

The basic behavior of the timer is described below:

- An Avalon-MM master peripheral, such as a Nios II processor, writes the timer core's `control` register to:
  - 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, `periodl` and `periodh`.
- 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 either `snapl` or `snaph` to request a coherent snapshot of the counter, and then reading `snapl` and `snaph` for the full 32-bit value.
- When the count reaches zero:
  - 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

Avalon-MM Slave Interface

The timer core implements a simple Avalon-MM slave interface to provide access to the register file. The Avalon-MM slave 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 in the SOPC Builder system. When the `resetrequest` signal is asserted, it forces any processor connected to the system to reboot. See "Configuring the Timer as a Watchdog Timer" on page 14–5 for further details.

Device and Tools Support

The timer core supports all Altera® FPGA families.

Instantiating the Core in SOPC Builder

Designers use the MegaWizard® interface for the timer core in SOPC Builder to specify the hardware features. This section describes the options available in the MegaWizard interface.

Timeout Period

The **Timeout Period** setting determines the initial value of the `periodl` and `periodh` registers. When the **Writeable period** setting is enabled, a processor can change the value of the period by writing `periodl` and `periodh`. When **Writeable period** (see below) is off, the period is fixed and cannot be updated at runtime.

The **Timeout Period** setting can be specified in units of **usec**, **msec**, **sec**, or **clocks** (number of clock cycles). The actual period achieved depends on the system clock. If the period is specified in usec, msec or sec, the true period will be the smallest number of clock cycles that is greater than or equal to the specified **Timeout Period**.

Hardware Options

The following options affect the hardware structure of the 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. See "Configuring the Timer as a Watchdog Timer" on page 14–5.

Register Options

Table 14–1 shows the settings that affect the timer core's registers.

Table 14–1. Register Options

Option	Description
Writeable period	When this option is enabled, a master peripheral can change the count-down period by writing `periodl` and `periodh`. When disabled, the count-down period is fixed at the specified **Timeout Period**, and the `periodl` and `periodh` registers do not exist in hardware.
Readable snapshot	When this option is enabled, a master 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 `snapl` and `snaph` registers do not exist in hardware, and reading these registers produces an undefined value.
Start/Stop control bits	When this option is enabled, a master 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 14–2 shows the settings that affect the timer core's output signals.

Table 14–2. Output Signal Options

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

Configuring the Timer as a Watchdog Timer

To configure the timer 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 (i.e., 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. To prevent the system from resetting, the processor must periodically reset the timer's count-down value by writing either the `periodl` or `periodh` registers (the written value is ignored). If the processor fails to access the timer because, for example, software stopped executing normally, then the watchdog timer resets the system and returns the system to a defined state.

Software Programming Model

The following sections describe the software programming model for the timer core, including the register map and software declarations to access the hardware. For Nios II processor users, Altera provides hardware

abstraction layer (HAL) system library drivers that enable you to access the timer core using the HAL application programming interface (API) functions.

HAL System Library Support

The Altera-provided drivers integrate into the HAL system library for Nios II systems. When possible, HAL users should access the timer via the HAL API, rather than accessing the timer registers.

Altera 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 timer runs continuously in periodic mode, using the default period set in SOPC builder. 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 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 SOPC Builder.
- 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, then calls to the `alt_timestamp_start()` API function will not reset the timestamp counter. All other HAL API calls will perform as expected.

See the *Nios II Software Developer's Handbook* for details about using the system clock and timestamp features that use these drivers. The Nios II Embedded Design Suite (EDS) also provides several example designs that use the timer core.

Limitations

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

Software Files

The 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.

Register Map

A programmer should never have to directly access the timer 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 Altera-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.

Table 14–3 shows the register map for the timer.

Table 14–3. Register Map

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 (31..16)						

Note to Table 14–3:
(1) Reserved. Read values are undefined. Write zero.

status Register

The `status` register has two defined bits, as shown in Table 14–4.

Table 14–4. status Register Bits

Bit	Name	Read/ Write/ Clear	Description
0	TO	RC	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 master peripheral. Write zero 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, as shown in Table 14–5.

Table 14–5. control Register Bits (Part 1 of 2)

Bit	Name	Read/ Write/ Clear	Description
0	ITO	RW	If the ITO bit is 1, the 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 32-bit value stored in the `periodl` and `periodh` 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 held 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.

Table 14–5. control Register Bits (Part 2 of 2)

Bit	Name	Read/ Write/ Clear	Description
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. Writing 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.

Note Table 14–5:
(1) Writing 1 to both START and STOP bits simultaneously produces an undefined result.

periodl and periodh Registers

The `periodl` and `periodh` registers together store the timeout period value. `periodl` holds the least-significant 16 bits, and `periodh` holds the most-significant 16 bits. The internal counter is loaded with the 32-bit value stored in `periodh` and `periodl` whenever one of the following occurs:

- A write operation to either the `periodh` or `periodl` register
- The internal counter reaches 0

The timer's actual period is one cycle greater than the value stored in `periodh` and `periodl`, because the counter assumes the value zero (0x00000000) for one clock cycle.

Writing to either `periodh` or `periodl` 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 either `periodh` or `periodl` causes the counter to reset to the fixed **Timeout Period** specified at system generation time.

snapl and snaph Registers

A master peripheral may request a coherent snapshot of the current 32-bit internal counter by performing a write operation (write-data ignored) to either the `snapl` or `snaph` registers. When a write occurs, the value of the counter is copied to `snapl` and `snaph`. `snapl` holds the least-significant 16 bits of the snapshot and `snaph` holds the most-significant 16 bits. The snapshot occurs whether or not the counter is running. Requesting a snapshot does not change the internal counter's operation.

Interrupt Behavior

The 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.

Referenced Documents

This chapter references the following document:

- *Nios II Software Developer's Handbook*

Document Revision History

Table 14–6 shows the revision history for this chapter.

Table 14–6. Document Revision History

Date and Document Version	Changes Made	Summary of Changes
May 2007 v7.1.0	• Corrected an error: The timer can be used as a timestamp device if it has a writeable *period* register. • Added table of contents to Overview section. • Added Referenced Documents section.	—
March 2007 v7.0.0	No change from previous release.	—
November 2006 v6.1.0	• Updated Avalon terminology because of changes to Avalon technologies. Changed old "Avalon switch fabric" term to "system interconnect fabric." Changed old "Avalon interface" terms to "Avalon Memory-Mapped interface." • Added statement that failure to acknowledge an IRQ results in an undefined result in section "Interrupt Behavior" on page 12–9.	For the 6.1 release, Altera released the Avalon Streaming interface, which necessitated some re-phrasing of existing Avalon terminology.
May 2006 v6.0.0	No change from previous release.	—
October 2005 v5.1.0	No change from previous release.	—
May 2005 v5.0.0	No change from previous release. Previously in the Nios II Processor Reference Handbook.	—
September 2004 v1.1	Updates for Nios II 1.01 release.	—
May 2004 v1.0	Initial release.	—