Nios Timer

July 2003, Version 3.2 Data Sheet

General Description

The Nios® Timer module is an Altera® SOPC Builder library component included in the Nios development kit. This SOPC Builder library component has available system choices to define device logic and interface signals on the Nios development board. The Timer module can be used as a periodic pulse generator or system watchdog timer. The Timer's Verilog HDL or VHDL source code is available for development and includes the necessary software subroutines for easy system integration.

Functional Description

The Nios Timer module is a simple 32-bit interval timer. Software controls the Timer by writing to several module registers and can request and then read coherent snapshots of the internal counter value. The Timer module generates a single interrupt-request output that can be masked by an internal control bit.

Software controls the Timer by:

- Setting the internal countdown preload value (timer period) by writing values to the `periodl` and `periodh` registers
- Starting and stopping the internal counter by writing 1s to the `start` and `stop` bits in the `control` register
- Enabling or disabling interrupts on timeout by writing to the interrupt-enable timeout (`ito`) bit in the `control` register
- Setting the operating mode (continuous or count only once) by writing to the continuous-run (`cont`) bit in the `control` register

Because the Timer works with 16-bit systems, all embedded processor-accessible registers are 16 bits wide. A 32-bit Nios CPU performs two separate write operations on two 16-bit registers (`periodl` and `periodh`) to set a 32-bit downcount value.

Software can request a coherent snapshot of the current internal counter value by writing to either the `snapl` or `snaph` registers. A write to either of these registers causes both registers to be simultaneously loaded with the internal counter's current value.

The Nios Timer runs off a single master clock input (`clk`), the same clock provided to the Nios CPU and other peripherals. This clock runs both the software interface registers and the internal counter.

The Timer may optionally be configured as a system watchdog. A watchdog Timer produces a `reset_output` signal that is automatically recognized by the SOPC Builder and used as an input to the system's reset logic.Table 1 lists and describes the Timer register map.

Timer Registers

Table 1. Timer Register Map

A2..A0	Register Name	R/W	Description/Register Bits					
			15	. . .	3	2	1	0
0	status	RW					run	to
1	control	RW			stop	start	cont	ito
2	periodl	RW	Timeout Period – 1 (bits 15..0)					
3	periodh	RW	Timeout Period – 1 (bits 31..16)					
4	snapl(1)	RW	Timeout Counter Snapshot (bits 15..0)					
5	snaph(1)	RW	Timeout Counter Snapshot (bits 31..16)					

Notes

(1) A write operation to either the `snapl` or `snaph` registers updates both registers with a coherent snapshot of the current internal counter value.

status Register

Table 2. status Register Bits

Bit Number	Bit Name	Description
0	to	Timer timed out
1	run	Timer is running

to Bit

The to 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 software. Software clears the `to` bit by performing a write operation to the `status` register.

run Bit

The `run` bit is set to 1 when the internal counter is running; otherwise this bit is 0. Software starts and stops the internal counter by writing to the `stop` and `start` bits in the `control` register.

The `run` bit is not changed by a write operation to the `status` register.

control Register

Table 3. control Register Bits

Bit Number	Bit Name	Description
0	ito	Enable interrupt for a timeout
1	cont	Continuous mode
2	start	Start Timer
3	stop	Stop Timer

ito Bit

If the `ito` bit is 1, the Timer generates an interrupt request (sets its IRQ output to 1) when the `status` register's `to` bit is 1. If the `ito` bit is 0, the IRQ output is always 0.

cont Bit

The `cont` (continuous) bit determines how the internal counter behaves when it reaches zero. When the internal counter reaches zero, it reloads with the 32-bit value stored in the `periodl` and `periodh` registers, regardless of the `cont` bit. If the `cont` bit is 1, the internal counter runs until it is stopped by the `stop` bit. If `cont` is 0, the internal counter stops after it reaches zero and reloads with the period value.

start Bit

Software starts the internal counter running (counting down) by writing a 1 to the `start` bit. The `start` bit is an event bit—the counter is started when the write operation is performed. The value stored in the `start` bit has no subsequent effect on the internal counter. Writing a 0 to the `start` bit has no effect on the Timer's operation.

Any 1 written to the `start` bit starts the counter, regardless of the stored value. 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.

stop Bit

Software stops the internal counter by writing a 1 to the `stop` bit. The `stop` bit is an event bit that causes the counter to stop when a write operation is performed. The value stored in the `stop` bit has no subsequent effect on the internal counter. Writing a 0 to the `stop` bit has no effect on the Timer's operation.

When the PTF parameter `always_run` is set to 1, writing to the `stop` bit has no effect on the internal counter.

If 1 is written to the start and stop bits simultaneously, the result is undefined.

periodl Register

The `periodl` register holds the 16 LSBs of the 32-bit down count preload value (the 16 MSBs are held in the `periodh` register). The Timer's actual period (that is, interrupt rate) is one greater than the value stored in the `periodh` and `periodl` register because the internal counter assumes the value zero (0x00000000) for one clock cycle.

The internal counter is loaded with the 32-bit value stored in `periodh` and `periodl` when one of the following occurs:

- Software performs a write operation to either the `periodh` or `periodl` registers
- The internal counter reaches 0

The internal counter is automatically stopped when software writes to either the `periodh` or `periodl` registers.

When the PTF parameter `fixed_period` is set to 1, a write operation to the `periodl` register restarts the internal counter at its initial (fixed) value.

When the PTF parameter `always_run` is set to 1, a write operation to the `periodl` register does not stop the internal counter.

periodh Register

The `periodh` register holds the 16 MSBs of the 32-bit downcount preload value (the 16 LSBs are held in the `periodl` register). The Timer's actual period (that is, interrupt rate) is one greater than the value stored in the `periodh` and `periodl` registers because the internal counter assumes the value zero (0x00000000) for one clock cycle.

When the PTF parameter `fixed_period` is set to 1, a write operation to the `periodh` register restarts the internal counter at its initial (fixed) value.

When the PTF parameter `always_run` is set to 1, a write operation to the `periodh` register does not stop the internal counter.

snapl Register

The `snapl` register holds the 16 LSBs of the most recently sampled 32-bit counter snapshot (the 16 MSBs are held in the `snaph` register). Software 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. Software may request a snapshot whether or not the internal counter is running. Requesting a snapshot does not change the internal counter's operation.

snaph Register

The `snaph` register holds the 16 MSBs of the most recently sampled 32-bit counter snapshot. The 16 LSBs are held in the `snapl` register.

Watchdog Timer Operation

The Timer is configured as a system watchdog by these PTF assignments:

```
reset_output = "1";
always_run = "1";
fixed_period = "1";
```

A watchdog timer "wakes up" (comes out of reset) stopped. Software can start the Timer at any time by writing a 1 to the `control` register's `start` bit. Once started, the Timer can never be stopped. The Timer generates a pulse on its `reset_out` output when the internal count reaches zero.

If the Timer is used as a module in an SOPC Builder system, the system resets. Software can restart the internal counter at its initial value by performing a write operation to either the `periodl` or `periodh` registers (any data value written to these registers is ignored). Software must continuously restart the watchdog timer (continuously write one or both of the `periodh`/`periodl` registers) to prevent the system from being reset.

For an example, see the file **../sdk/src/hello_watchdog.c**.

Software Data Structure

```
typedef volatile struct
{
    int np_timerstatus;  // read only, 2 bits (any write to clear TO)
    int np_timercontrol; // write/readable, 4 bits
    int np_timerperiodl; // write/readable, 16 bits
    int np_timerperiodh; // write/readable, 16 bits
    int np_timersnapl;   // read only, 16 bits
    int np_timersnaph;   // read only, 16 bits
} np_timer;
```

Figure 1 shows an example of direct access to the Timer.

Example 1. Direct Access to Timer

```
#include "nios.h"

int main(void)
        {
        int t = 0;

        // Set timer for 1 second
        na_timer1->np_timerperiodl = (short)(nasys_clock_freq & 0x0000ffff);
        na_timer1->np_timerperiodh = (short)((nasys_clock_freq >> 16) & 0x0000ffff);

        // Set timer running, looping, no interrupts
        na_timer1->np_timercontrol = np_timercontrol_start_mask + np_timercontrol_cont_mask;

        // Poll timer forever, print once per second
        while(1)
                {
                if(na_timer1->np_timerstatus & np_timerstatus_to_mask)
                        {
                        printf("A second passed! (%d)\n",t++);

                        // Clear the to (timeout) bit
                        na_timer1->np_timerstatus = 0; // (any value)
                        }
                }
        }
```

Software Subroutine

The software subroutine `nr_timer_milliseconds` is available in the Nios library (**lib** folder in the custom SDK) if one or more Timer peripherals are present in the Nios system. This function is declared in the include file **nios.h**.

nr_timer_milliseconds

This subroutine requires the existence of a Timer called `timer1`, with a base address defined by na_timer1 and an interrupt number defined by na_timer1_irq. The first time this subroutine is called, it installs an interrupt service subroutine for the Timer and returns zero. For each subsequent call, the number of milliseconds elapsed since the first call is returned.

Syntax

```
int nr_timer_milliseconds(void);
```

PTF Assignments

Table 4 lists the Timer's PTF parameters. Detailed descriptions follow the table.

Table 4. Timer PTF Parameters

Parameter	Section[1]	Type	Allowed values	Default
clock_freq	S/WSA	Integer	≥ 1 (Hz)	33333000
fixed_period	M/WSA	Boolean	1, 0	0
snapshot	M/WSA	Boolean	1, 0	1
always_run	M/WSA	Boolean	1, 0	0
timeout_pulse_output	M/WSA	Boolean	1, 0	0
reset_output	M/WSA	Boolean	1, 0	0
period	M/WSA	String	> 0	0
period_units	M/WSA	String	"ns", "us", "ms", "s", "sec", "clocks"	"sec"
mult[2]	M/WSA	Decimal	> 0	.001

Note

(1) The **Section** column describes the parameter's location in the PTF:
S/WSA = `SYSTEM/WIZARD_SCRIPT_ARGUMENTS`
M/WSA = `MODULE/WIZARD_SCRIPT_ARGUMENTS`

(2) The `mult` parameter is strictly a MegaWizard setting; resetting this value in the PTF has no effect on system generation.

clock_freq

The `clock_freq` assignment is the global system clock frequency. It affects the generation of the Timer's achieved period. This setting is derived from the system PTF, and is not set by SOPC Builder.

fixed_period

When `fixed_period` is set to 1, the `periodl` and `periodh` registers cannot be written to in the Timer. In this case, the `periodl` and `periodh` registers' contents are determined by the `period` and `period_units` parameters. When `fixed_period` is set to 0, the `periodl` and `periodh` registers are writable and any downcount can be set.

snapshot

When `snapshot` is set to 0, the `snapl` and `snaph` registers do not exist. In this case, reading from the `snapl` and `snaph` registers produces an undefined result.

always_run

When `always_run` is set to 1, the `control` register's `start` and `stop` bits do not exist—the internal counter runs continuously. When `reset_output` is set to 1, the `control` registers' `start` bit does exist and the Timer can be started but not stopped (for watchdog timer functionality).

timeout_pulse_output

When `timeout_pulse_output` is set to 1, a system output port, `timeout_pulse`, is generated. This port pulses high for every instance of a timeout. This output pulses true for exactly one clock when the Timer reaches 0.

reset_output

When `reset_output` is set to 1, an output port, `resetrequest`, is generated. This port pulses high for every instance of a timeout. `resetrequest` pulses true for exactly one clock when the Timer reaches 0.

When `reset_output` is set to 1, the internal counter is stopped at reset. It starts when the `control` register's `start` bit is set to 1.

period

When the `fixed_period` parameter is set to 1, the `period` assignment, along with the `period_units` and `clock_freq` assignments, specifies the downcount value in the `periodl` and `periodh` registers, according to the formula:

```
scale =  if  (period_units = "s" or "sec")   then 1
         else if (period_units = "ns")       then 1E-9
         else if (period_units = "us")       then 1E-6
         else if (period_units = "ms")       then 1E-3

if (period_units = "clocks") then
  Down-Count = period - 1
else
  Down-Count = period * scale * clock_freq;
```

period_units

The `period_units` assignment specifies the unit of measure to be used for the `period` assignment. The allowed values are shown in Table 5:

Table 5. period_units Allowed Values

Value	Description
ns	nanoseconds
us	microseconds
ms	milliseconds
s	seconds
sec	
clock	use the assigned clock frequency

The default value is “sec”.

mult

The `mult` assignment is strictly an SOPC Builder setting used by the MegaWizard. Changing the value of `mult` in the PTF has no effect on the system generation.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
lit_req@altera.com