

## VciMultiTimer Functional Description

This VCI target is a memory mapped peripheral that can control up to 256 software controlled timers. Each timer can optionally generate an independent periodic interrupt. The memory segment allocated to this component must be aligned on 4K bytes boundary.

This hardware component checks for segmentation violation, and can be used as a default target.

## Memory region layout

The timer index  $i$  is defined by the ADDRESS[12:4] bits.

Each timer contains 4 memory mapped registers:

- **TIMER\_VALUE:** ADDRESS[3:0] = 0x0

This 32 bits register is unconditionally incremented at each cycle. A read request returns the current time contained in this register. A write request sets a new value in this register.

- **TIMER\_MODE:** ADDRESS[3:0] = 0x4 This register contains two flags:
  - ◆ Bit 0: **TIMER\_RUNNING.** When 1, the associated timer will decrease on each cycle
  - ◆ Bit 1: **TIMER\_IRQ\_ENABLED:** When 1, the associated IRQ line will be activated if the timer underflows.
- **TIMER\_PERIOD:** ADDRESS[3:0] = 0x8

This 32 bits register defines the period between two successive interrupts. It may be read or written to.

- **TIMER\_RESETIRQ:** ADDRESS[3:0] = 0xC

Any write request in this Boolean register will reset the pending IRQ. A read request returns the zero value when there is no pending interrupt, and returns a non zero value if there is a pending interrupt.

## Component usage

For extensibility issues, you should access your terminal using globally-defined offsets.

You should include file source:trunk/soclib/include/soclib/timer.h from your software, it defines **TIMER\_VALUE**, **TIMER\_MODE**, **TIMER\_PERIOD**, **TIMER\_RESETIRQ**, **TIMER\_SPAN**, **TIMER\_RUNNING**, **TIMER\_IRQ\_ENABLED**.

Sample code:

```
#include "soclib/timer.h"

static const volatile void* timer_address = 0xc0000000;

static timer_test(const size_t timer_no)
{
    volatile int *timer = ((int*)timer_address) + timer_no*TIMER_SPAN;
```

```

// Getting / setting timer current value
timer[TIMER_VALUE] = 0x2a00;
uint32_t foo = timer[TIMER_VALUE];

// Enabling timer and interrupt
timer[TIMER_MODE] = TIMER_RUNNING | TIMER_IRQ_ENABLED;

// Getting IRQ status, and resetting IRQ
if ( timer[TIMER_RESETIrq] )
    timer[TIMER_RESETIrq] = 0;
}

```

(add -I/path/to/soclib/include to your compilation command-line)

## Component definition

Available in source:trunk/soclib/desc/soclib/vci\_timer.sd

## Usage

VciTimer has no other parameter than VCI ones, it may be used like others, see [SoclibCc/VciParameters](#)

```
Use( 'vci_timer', **vci_parameters )
```

## VciMultiTimer CABA Implementation

The caba implementation is in

- source:trunk/soclib/systemc/include/caba/target/vci\_timer.h
- source:trunk/soclib/systemc/src/caba/target/vci\_timer.cc

## Template parameters:

- The VCI parameters

## Constructor parameters

```

VciMultiTimer(
    sc_module_name name,    // Component Name
    const soclib::common::IntTab & index, // Target index
    const soclib::common::MappingTable &mt, // MappingTable
    size_t nirq;           // Number of available timers

```

## Ports

- sc\_in<bool> **p\_resetn** : Global system reset
- sc\_in<bool> **p\_clk** : Global system clock
- soclib::caba::VciTarget<vci\_param> **p\_vci** : The VCI port
- sc\_out<bool> **p\_irq[]** : Interrupts ports array