VciMultiTimer

1) 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 cheks for segmentation violation, and can be used as a default target.

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

Each timer contains 4 memory mapped registers:

- `TIMER_VALUE`

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` 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`

This 32 bits register defines the period between two successive interrupts. It may be read or written to. When written to, the new period is immediatly taken into account.

- `TIMER_RESETIRQ`

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.

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

You should include file `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)
{
    // Getting / setting timer current value
    soclib_io_set( timer_address, TIMER_SPAN*timer_no + TIMER_VALUE, 0x2a00 );
    uint32_t foo = soclib_io_get( timer_address, TIMER_SPAN*timer_no + TIMER_VALUE );

    // Enabling timer and interrupt
    soclib_io_set( timer_address, TIMER_SPAN*timer_no + TIMER_MODE, TIMER_RUNNING | TIMER_IRQ_EN
```

```
        // Getting IRQ status, and resetting IRQ
        if ( soclib_io_get( timer_address, TIMER_SPAN*timer_no + TIMER_RESETIRQ ) )
            soclib_io_set( timer_address, TIMER_SPAN*timer_no + TIMER_RESETIRQ, 0 );
    }
```

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

2) Component definition & usage

source:trunk/soclib/soclib/module/internal_component/vci_timer/caba/metadata/vci_timer.sd?

See SoclibCc/VciParameters

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

3) CABA Implementation

CABA sources

- interface : source:trunk/soclib/soclib/module/internal_component/vci_timer/caba/source/include/vci_timer.h?
- implementation : source:trunk/soclib/soclib/module/internal_component/vci_timer/caba/source/src/vci_timer.cpp?

CABA 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
```

CABA 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

4) TLM-T Iplementation

TLM-T sources

- interface : source:trunk/soclib/soclib/module/internal_component/vci_timer/tlmt/source/include/vci_timer.h?
- implementation : source:trunk/soclib/soclib/module/internal_component/vci_timer/tlmt/source/src/vci_timer.cpp?

TLM-T Constructor parameters

```
VciTimer(
     sc_module_name name,   //  Component Name
```

```
const soclib::common::IntTab & index,  // Target index
const soclib::common::MappingTable &mt,   // MappingTable
size_t ntimer);   //  Number of available timers
```

TLM-T Ports

- soclib::tlmt::VciTarget<vci_param> **p_vci** : The VCI port
- std::vector<tlmt_core::tlmt_out<bool> *> **p_irq** : Interrupts ports array