Vcilcu Functional Description

This VCI target is a memory mapped peripheral implementing a vectorized interrupt controller. It can concentrate up to 32 independent interrupt lines **p_irq_in[i]** to a single **p_irq** interrupt line.

The active state is high, and the output interrupt is the logical OR of all input interrupts. Each input interrupt can be individually masked through an internal register.

This component can be addressed to return the index of the highest priority active interrupt **p_irq_[i]**. The priority scheme is fixed : The lower indexes have the highest priority.

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

Memory region layout

This component contains 5 memory mapped registers:

- `ICU_INT`: ADDRESS[4:0] = 0x0 Each bit in this register reflects the state of the corresponding interrupt line. This is read-only.
- `ICU_MASK`: ADDRESS[4:0] = 0x4 Each bit in this register reflects the state of the enable for the corresponding interrupt line. This is read-only.
- `ICU_MASK_SET`: ADDRESS[4:0] = 0x8 Each bit set in the written word will be set in the ICU MASK. (ICU_MASK = ICU_MASK | written_data). This is write-only.
- `ICU_MASK_CLEAR`: ADDRESS[4:0] = 0xc Each bit set in the written word will be reset in the ICU MASK. (ICU_MASK = ICU_MASK & ~written_data). This is write-only.
- `ICU_IT_VECTOR`: ADDRESS[4:0] = 0x10 This register gives the number of the highest-priority active interrupt. If no interrupt is active, (-1) is returned. This is read-only.

Component usage

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

You should include file source:trunk/soclib/include/soclib/icu.h from your software, it defines `ICU_INT`, `ICU_MASK`, `ICU_MASK_SET`, `ICU_MASK_CLEAR`, `ICU_IT_VECTOR`.

Sample code:

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

static const volatile void* timer_address = 0xc0000000;

static icu_test(const size_t icu_no)
{
    volatile int *icu = ((int*)icu_address) + timer_no*ICU_SPAN;

    // Getting / setting interrupt mask
    uint32_t current_interrupt_mask = icu[ICU_MASK];
```

```
    // Enabling IRQ #5
    icu[ICU_MASK_SET] = 0x20;
    // Disabling IRQ #0
    icu[ICU_MASK_CLEAR] = 0x1;

    // When interrupt is raised, you may do:
    int irq_to_serve = icu[ICU_IT_VECTOR];
    // This should be equivalent to (see man 3 ffs)
    int irq_to_serve = ffs(icu[ICU_IT_VECTOR] & icu[ICU_MASK]);
}
```

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

Component definition

Available in source:trunk/soclib/desc/soclib/vci_icu.sd

Usage

VciIcu has no other parameter than VCI ones, it may be used like others, see SoclibCc/VciParameters

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

VciIcu CABA Implementation

The caba implementation is in

- source:trunk/soclib/systemc/include/caba/target/vci_icu.h
- source:trunk/soclib/systemc/src/caba/target/vci_icu.cc

Template parameters:

- The VCI parameters

Constructor parameters

```
VciIcu(
     sc_module_name name,  // Component Name
     const soclib::common::InTab &index, // Target index
     const soclib::common::MappingTable &mt,  // Mapping Table
     size_t nirq);  // Number of input interrupts
```

Ports

- sc_in<bool> **p_resetn** : Global system reset
- sc_in<bool> **p_clk** : Global system clock
- soclib::caba::VciTarget<vci_param> **p_vci** : VCI port
- sc_out<bool> **p_irq** : Output interrupt port
- sc_in<bool> **p_irq_in[]** : Input interrupts ports array