VciMultiTty Functional Description

This VCI target is a TTY terminal controller. This hardware component controls independant terminals. The number of emulated terminals is defined by the arguments in the constructor (one name per terminal). Name list MUST be terminated by NULL.

Each terminal is acting both as a character display, and a keyboard interface. For each terminal, a specific IRQ is activated when a character entered at the keyboard is available in a buffer. IRQ is kept low as long as the buffer is not empty.

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

This component uses a TtyWrapper per terminal in order to abstract different simulator's ttys.

Memory region layout

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

Each TTY controller contains 3 memory mapped registers:

- `TTY_WRITE`: ADDRESS[3:0] = 0x0

This 8 bits pseudo-register is write only. Any write request will interpret the 8 LSB bits of the WDATA field as an ASCII character, and this character will be displayed on the addressed terminal.

- `TTY_STATUS`: ADDRESS[3:0] = 0x4

This Boolean status register is read-only. A read request returns the zero value if there is no pending character. It returns a non zero value if there is a pending character in the keyboard buffer.

- `TTY_READ`: ADDRESS[3:0] = 0x8

This 8 bits register contains one single ASCII character. This register is read-only. A read request returns the ACSII character in the 8 LSB bits of the RDATA field, and reset the status register

Component usage

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

You should include file source:trunk/soclib/include/soclib/tty.h from your software, it defines `TTY_WRITE`, `TTY_STATUS`, `TTY_READ` and `TTY_SPAN`.

A putc/getc implementation could be:

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

static const volatile void* tty_address = 0xc0000000;

static inline void putc(const size_t term_no, const char x)
{
    volatile int *tty = ((int*)tty_address) + term_no*TTY_SPAN;
    tty[TTY_WRITE] = x;
}

static inline char getc(const size_t term_no)
{
    volatile int *tty = ((int*)tty_address) + term_no*TTY_SPAN;
    return tty[TTY_READ];
}
```

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

Component definition

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

Usage

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

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

VciMultiTty CABA Implementation

The caba implementation is in

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

Template parameters:

- The VCI parameters

Constructor parameters

```
VciMultiTty(
     sc_module_name name,   // Instance name
     const soclib::common::IntTab &index,   //  Target index
     const soclib::common::MappingTable &mt,   // Mapping Table
     const char *first_tty_name,   // TTY names (as many names as terminals)
     ...);
```

Example instanciation:

```
VciMultiTty tty("tty_comp", IntTab(2,3), mapping_table, "term0", "term1", "term2", NULL);
```

Ports

- sc_in<bool> **p_resetn** : Global system reset
- sc_in<bool> **p_clk** : Global system clock
- soclib::common::VciTarget<vci_param> **p_vci** : The VCI port
- sc_out<bool> **p_irq[]** : Interrupt ports array.