

VciMwMrController Functional Description

This VCI component is both a target and an initiator.

- Addressing as a target allows to configure it for a transfer.
- Initiator will do the transfer

This component may handle interfaces to a Fifo-accessed coprocessor. There may be any number of fifos from/to the coprocessor.

In conjunction with the fifos, there are multiple unidirectionnal 32-bits signals going from/to the coprocessor.

- from the coprocessor, they are *status* registers
- to the coprocessor, they are *config* registers

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

Memory region layout

- Registers 0 to MWMR_IOREG_MAX

When read from, they reflect status registers, when written to, they reflect the control registers.

- MWMR_RESET

When written to, this register resets the current state of the controller, flushing all fifos and configuration.

- MWMR_CONFIG_FIFO_WAY and MWMR_CONFIG_FIFO_NO

Used to designate the currently configured fifo. WAY may be MWMR_TO_COPROC or MWMR_FROM_COPROC, NO may be any of available fifos in the selected way.

- MWMR_CONFIG_STATE_ADDR

Sets the address of state field for the selected fifo's state block.

- MWMR_CONFIG_OFFSET_ADDR

Sets the address of read/write pointer field for the selected fifo's state block.

- MWMR_CONFIG_LOCK_ADDR

Sets the address of lock for the selected fifo's state block.

- MWMR_CONFIG_DEPTH

Sets the depth of the selected fifo.

- MWMR_CONFIG_WIDTH

Sets the width of the selected fifo. This will determine the atomic transfer block size. This must be multiple of 4.

- `MWMR_CONFIG_BASE_ADDR`

Sets the address of data for the selected fifo.

- `MWMR_CONFIG_RUNNING`

A boolean enabling the selected fifo.

Component usage

For extensibility issues, you should access the `MwmrController` using globally-defined offsets.

You should include `soclib/MwmrController.h` from your software, it defines all useful offsets and constants.

Sample code:

Please see `source:trunk/soclib/platform/runtime_netlist/mwmr/soft/mwmr.h` and `source:trunk/soclib/platform/runtime_netlist/mwmr/soft/mwmr.c` for reference implementation.

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

Component definition

Available in

`source:trunk/soclib/module/internal_component/vci_mwmr_controller/caba/metadata/vci_mwmr_controller.sd`

Usage

`VciMwmrController` has two template parameters: VCI ones, and internal fifo maximum depth. It may be used like others, see [SoclibCc/VciParameters](#)

```
Uses( 'vci_MwmrController', fifo_depth = 32, **vci_parameters )
```

VciMwmrController CABA Implementation

The caba implementation is in

- `source:trunk/soclib/module/internal_component/vci_mwmr_controller/caba/source/include/vci_mwmr_controller.h`
- `source:trunk/soclib/module/internal_component/vci_mwmr_controller/caba/source/src/vci_mwmr_controller.cpp`

Template parameters:

- The VCI parameters
- The fifo depth

Constructor parameters

```
VciMwmrController(  
    sc_module_name name,  
    const IntTab &index,  
    const MappingTable &mt,  
    const size_t plaps,          # Default timeout between two access retries to a given MWMR  
    const size_t n_to_coproc,   # \ Cardinal of fifos  
    const size_t n_from_coproc, # /   interfacing the coprocessor  
    const size_t n_config,      # \ Cardinal of registers  
    const size_t n_status );    # /   (sideband signals)
```

Ports

- `sc_in<bool> p_resetn` : Global system reset
- `sc_in<bool> p_clk` : Global system clock
- `soclib::caba::VciTarget<vci_param> p_vci_target` : The VCI target port
- `soclib::caba::VciInitiator<vci_param> p_vci_initiator` : The VCI initiator port
- `soclib::caba::FifoOutput<uint32_t> p_to_coproc[]` : Fifos to coprocessor
- `soclib::caba::FifoInput<uint32_t> p_from_coproc[]` : Fifos from coprocessor
- `sc_out<uint32_t> p_config[]` : Configuration ports
- `sc_in<uint32_t> p_status[]` : Status ports