

VciMwmrController

1) Functional Description

This VCI component is the hardware part of the MWMR communication middleware. It allows an hardware coprocessor to communicate with one or several MWMR channels. The coprocessor communicates with the MWMR controller through several FIFO interfaces (one FIFO interface per MWMR channel). An internal FSM implements the five steps MWMR communication protocol (5 VCI transactions for one MWMR transaction). This component contains as many hardware FiFOs as the number of supported MWMR channels. An MWMR transaction starts when a Write FIFO is FULL, or when a Read FIFO is empty. The priority policy between the supported channels is Round Robin.

This component is both a VCI target and a VCI initiator.

- It is addressed as a target to be configured.
- It is acting as an initiator to do the MWMR transfers

Besides the communication channels, this MWMR controller provides a variable number of unidirectionnal 32-bits signals going from/to the coprocessor.

- from the coprocessor, they are used to read the value stored in *status* registers
- to the coprocessor, they are used to write values into *configuration* registers

As a target this component contains the following memory mapped registers:

- Status & Configuration Registers (indexed from 0 to MWMR_IOREG_MAX-1)

A read access returns the value stored in the corresponding coprocessor status register. A write access changes the value stored in the corresponding coprocessor configuration registers.

- MWMR_RESET

Writing into this register resets the current state of the controller, flushing all hardware FIFOs and all MWMR channels configuration.

- MWMR_CONFIG_FIFO_WAY and MWMR_CONFIG_FIFO_NO

Used to designate the currently configured MWMR channel. WAY may be MWMR_TO_COPROC or MWMR_FROM_COPROC, NO is the MWMR channel index, for the selected way.

- MWMR_CONFIG_STATUS_ADDR

Sets the address of the status descriptor structure for the selected MWMR channel.

- MWMR_CONFIG_DEPTH

Sets the total depth of the selected MWMR channel (in bytes).

- MWMR_CONFIG_BUFFER_ADDR

Sets the address of the data buffer for the selected MWMR channel.

- `MWMR_CONFIG_RUNNING`

A boolean enabling the selected MWMR channel.

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

For extensibility issues, you should access the `MwmrController` using globally-defined offsets. You should include the `soclib/MwmrController.h` file in your software, that defines all useful offsets and constants.

Sample code:

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

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

2) Component definition & usage

[source:trunk/soclib/soclib/module/internal_component/vci_mwmr_controller/caba/metadata/vci_mwmr_controller.sd?](#)

See [SoclibCc/VciParameters](#)

```
Uses( 'vci_mwmr_controller' )
```

3) CABA Implementation

CABA sources

- interface :
[source:trunk/soclib/soclib/module/internal_component/vci_mwmr_controller/caba/source/include/vci_mwmr_controller.h](#)
- implementation :
[source:trunk/soclib/soclib/module/internal_component/vci_mwmr_controller/caba/source/src/vci_mwmr_controller.c](#)

CABA Constructor

```
VciMwmrController(  
    sc_module_name name, // instance name  
    const MappingTable &mt, // mapping table  
    const IntTab &srcid, // VCI initiator index  
    const IntTab &tgtid, // VCI target index  
    const size_t plaps, // time between two access to a given channel  
    const size_t fifo_to_coproc_depth, // hardware FIFOs depth (in words)  
    const size_t fifo_from_coproc_depth, // hardware FIFOs depth (in words)  
    const size_t n_to_coproc, // number of read MWMR channels  
    const size_t n_from_coproc, // number of write MWMR channels  
    const size_t n_config, // number of configuration registers  
    const size_t n_status) // number of status registers
```

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_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

4) TLM-T Implementation

TLM-T sources

- interface :
[source:trunk/soclib/soclib/module/internal_component/vci_mwmmr_controller/tlmt/source/include/vci_mwmmr_controller.h](#)
- implementation :
[source:trunk/soclib/soclib/module/internal_component/vci_mwmmr_controller/tlmt/source/src/vci_mwmmr_controller.cpp](#)

TLM-T Constructor

```
VciMwmmrController(
    sc_module_name name,                // instance name
    const MappingTable &mt,             // mapping table
    const IntTab &srcid,                 // VCI initiator index
    const IntTab &tgtid,                 // VCI target index
    const size_t fifo_to_coproc_depth,  // hardware FIFOs depth (in words)
    const size_t fifo_from_coproc_depth, // hardware FIFOs depth (in words)
    const size_t n_to_coproc,           // number of read MWMMR channels
    const size_t n_from_coproc,         // number of write MWMMR channels
    const size_t n_config,              // number of configuration registers
    const size_t n_status)              // number of status registers
```

TLM-T Ports

- soclib::tlmt::VciInitiator<vci_param> **p_vci_initiator** : The VCI initiator port
- soclib::tlmt::VciTarget<vci_param> **p_vci_target** : The VCI target port
- std::vector<tlmt_core::tlmt_out<typename vci_param::data_t>*> **p_config** : Configuration ports
- std::vector<tlmt_core::tlmt_out<typename vci_param::data_t*>*> **p_status** : Status ports
- std::vector<soclib::tlmt::!FifoTarget<vci_param>*> **p_read_fifo** : Fifos from coprocessor
- std::vector<soclib::tlmt::!FifoTarget<vci_param>*> **p_write_fifo** : Fifos to coprocessor