

VciMwmrController

1) Functional Description

This VCI component is the hardware part of the MWMMR communication middleware. It allows an hardware coprocessor to communicate to one or several MWMMR channels. The coprocessor communicates with the MWMMR controller through several FIFO interfaces (one FIFO interface per MWMMR channel). An internal FSM implements the five steps MWMMR communication protocol (5 VCI transactions for one MWMMR transaction). This component contains as many hardware FiFOs as the number of supported MWMMR channels. An MWMMR 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 target and an initiator.

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

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

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

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

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

- Registers 0 to MWMMR_IOREG_MAX

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

- MWMMR_RESET

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

- MWMMR_CONFIG_FIFO_WAY and MWMMR_CONFIG_FIFO_NO

Used to designate the currently configured MWMMR channel. WAY may be MWMMR_TO_COPROC or MWMMR_FROM_COPROC, NO may be any MWMMR channel in the selected way.

- MWMMR_CONFIG_STATE_ADDR

Sets the address of state field for the selected MWMMR channel.

- MWMMR_CONFIG_OFFSET_ADDR

Sets the address of read/write pointer field for the selected MWMMR channel.

- MWMMR_CONFIG_LOCK_ADDR

Sets the address of the lock protecting the selected MWMR channel.

- `MWMR_CONFIG_DEPTH`

Sets the depth of the selected MWMR channel.

- `MWMR_CONFIG_WIDTH`

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

- `MWMR_CONFIG_BASE_ADDR`

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

- `MWMR_CONFIG_RUNNING`

A boolean enabling the selected MWMR channel.

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)

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_MwmrController', **vci_parameters )
```

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 IntTab &index, // VCI target index  
    const MappingTable &mt, // mapping table  
    const size_t plaps, // Default timeout between two access retries to a given channel  
    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
const size_t fifo_depth);  // hardware FIFOs depth

```

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

The TLM-T implementation is not available yet.