VciMwmrDma

1) Functional Description

This VCI component can be used to connect an hardware coprocessor to a VCI interconnect.

On the coprocessor side, it provides a variable number of TO_COPROC and FROM_COPROC ports, defining a fifo-like interface, without addresses.

- on a TO_COPROC port the coprocessor can request to read a vector of 32 bits words.
- on a FROM_COPROC port the coprocessor can request to write a vector of 32 bits words.

Each TO_COPROC or FROM_COPROC port define a communication channel to a memory buffer. The number of TO_COPROC and FROM_COPROC channels, are constructor parameters. The total number of channels cannot be larger than 16.

On the VCI side, 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 requested data transfers

It makes the assumption that the VCI RDATA & WDATA fields have 32 bits, but the VCI address field can have up to 64 bits. The VCI TRDID and PKTID fields must have at least 4 bits.

WARNING : This DMA controller uses bursts to transfer the data, and a constructor parameter define the burst size (typically a cache line). This introduce the following constraints:

- The memory buffer address and size must be multiple of the burst size.
- The number of bytes requested by the coprocessor on a TO_COPROC or FROM_COPROC port must be an integer number of bursts.

Each channel FSM implements two main operating modes that can be defined by software:

- In **MODE_DMA_IRQ** or **MODE_DMA_NO_IRQ** modes the channel FSM transfer a single buffer between the memory and the coprocessor port. The number of VCI burst depends on both the memory buffer size, and the burst size. In this mode the software must define the channel configuration by writing the data buffer address and size in the channel configuration registers. When the transfer is completed, the channel FSM wait in the success or ERROR state, until it is reset to IDLE state by writing a zero value in the CHANNEL_RUN register. An optional IRQ can be activated when the requested transfer is completed (only in MODE_DMA_IRQ).

- In **MWMR_MODE**, the channel FSM transfer an "infinite" data stream, between the coprocessor port and a MWMR channel (software FIFO in memory). In this mode the software must write in the channel configuration registers the data buffer address and size, but also the MWMR FIFO descriptor address and the lock address, as the channel FSM implements the 7 steps MWMR protocol. 1 - Read the ticket for queuing lock (1 flit VCI READ) 2 - Increment atomically the ticket (VCI CAS) 3 - Read the lock current value (1 flit VCI READ) 4 - Read the channel status (3 flits VCI READ) 5 - Transfer the data (N flits VCI READ or WRITE) 6 - Upate the status (3 flits VCI WRITE) 7 - Release the lock (1 flit VCI WRITE)

For an "infinite" data stream, the IRQ is not used in normal operation, and is only asserted if a VCI error is reported, ant the channel FSM is waiting in one ERROR state.

Several channels can simultaneously run in different modes, and the various VCI transactions corresponding to different channels are interleaved and parallelized on the VCI network. The maximum number of simultaneous VCI transactions is equal to the number of channels.

Besides the communication channels, this MWMR controller provides a variable number of coprocessor specific *configuration* or *status* 32 bits registers:

- **CONFIG** registers are Read/Write
- **STATUS** registers are Read-Only

2) Addressable registers

For each communication channel, the software addressable registers are the following

- **CHANNEL_BUFFER_LSB[k]** data buffer physical address 32 LSB bits (MWMR or DMA)
- **CHANNEL_BUFFER_MSB[k]** data buffer physical address extend bits (MWMR or DMA)
- **CHANNEL_MWMR_LSB[k]** channel status physical address 32 LSB bits (MWMR only)
- **CHANNEL_MWMR_MSB[k]** channel status physical address extend bits (MWMR only)
- **CHANNEL_LOCK_LSB[k]** channel lock physical address 32 LSB bits (MWMR only)
- **CHANNEL_LOCK_MSB[k]** channel lock physical address extend bits (MWMR only)
- **CHANNEL_WAY[k]** channel direction (TO_COPROC / FROM_COPROC) (MWMR or DMA)
- **CHANNEL_MODE[k]** MWMR / DMA_IRQ / DMA_NO_IRQ (MWMR or DMA)
- **CHANNEL_SIZE[k]** data buffer size (bytes) (MWMR or DMA)
- **CHANNEL_RUN[k]** channel activation/deativation (MWMR or DMA)
- **CHANNEL_STATUS[k]** channel FSM state (MWMR or DMA)

The relevant values for the CHANNEL_STATUS register are the following:

*

For extensibility issues, you should access these registers using these globally-defined offsets. The mwmr_dma.h file? defines all useful offsets and constants.

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

3) Component definition & usage

source:trunk/soclib/soclib/module/infrastructure_component/dma_infrastructure/vci_mwmr_dma/caba/metadata/vci_mwmr_d

```
Uses( 'vci_mwmr_dma' )
```

4) CABA Implementation

CABA sources

- interface : source:trunk/soclib/soclib/module/infrastructure_component/dma_infrastructure/vci_mwmr_dma/caba/source/include
- implementation : source:trunk/soclib/soclib/module/infrastructure_component/dma_infrastructure/vci_mwmr_dma/caba/source/src/vci

CABA Constructor

```
VciMwmrDma(
        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 n_to_coproc,  // number of TO_COPROC channels
        const size_t n_from_coproc, // number of FROM_COPROC channels
        const size_t n_config,     // number of configuration registers
        const size_t n_status,   // number of status registers
        const size_t burst_size )  // number of bytes for VCI bursts
```

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::ToCoprocOutput<uint32_t,uint8_t> **p_to_coproc[]** : to coprocessor ports
- soclib::caba::FromCoprocInput<uint32_t,uint8_t> **p_from_coproc[]** : from coprocessor ports
- sc_out<uint32_t> **p_config[]** : Configuration ports
- sc_in<uint32_t> **p_status[]** : Status ports
- sc_out<bool> **p_irq** : IRQ output port

5) TLM-DT Implementation

Not available yet.