# VciMultiDma

## 1) Functional Description

As the VciDma component, this component moves data from a source memory buffer to a destination memory buffer. It is both a target and an initiator.

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

This hardware component supports up to 16 simultaneous DMA transfers, corresponding to 16 independant DMA channels.

As there is only one VCI port, the general arbitration policy between the active channels is round-robin.

Each channet has its own set of memory mapped registers, and for each channel a specific IRQ can be is optionally asserted when transfer is completed.

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

This component has 5 memory-mapped registers :

- **DMA_SRC** (Read / Write)

It defines the physical address of the source buffer.

- **DMA_DST** (Read / Write)

It defines the physical address of the destination buffer.

- **DMA_LEN** (Read / Write)

It defines the length of transfer, in bytes. This register must be written after writing into registers DMA_SRC & DMA_DST, as the writing into the DMA_LEN register starts the transfer. This register gets back to 0 when transfer is finished. This register can be used to test the DMA coprocessor status.

- **DMA_RESET** (Write-only)

Writing any value into this pseudo-register makes a clean re-initialisation of the DMA coprocessor: The on-going VCI transaction is completed before the coprocessor returns the IDLE state. This write access must be used by the software ISR to aknowledge the DMA IRQ.

- **DMA_IRQ_DISABLED** (Read / Write)

A non zero value disables the IRQ line. The RESET value is zero.

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

You should include file `soclib/dma.h` from your software, it defines `DMA_SRC`, `DMA_DST`, `DMA_LEN`, `DMA_RESET`, `DMA_IRQ_DISABLED`.

Sample code:

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

static const volatile void* dma = 0xc0000000;

void * memcpy(void *dst, const void *src, const size_t len)
{
    soclib_io_set( dma, DMA_DST, dst );
    soclib_io_set( dma, DMA_SRC, src );
    soclib_io_set( dma, DMA_LEN, len );
    while( soclib_io_get( dma, DMA_LEN ) )
        ;
    return dst;
}
```

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

# 2) Component definition & usage

source:trunk/soclib/soclib/module/infrastructure_component/dma_infrastructure/vci_dma/caba/metadata/vci_dma.sd?

See SoclibCc/VciParameters

```
Uses( 'vci_dma' )
```

# 3) CABA Implementation

## CABA sources

- interface : source:trunk/soclib/soclib/module/infrastructure_component/dma_infrastructure/vci_dma/caba/source/include/vci_dm
- implementation : source:trunk/soclib/soclib/module/infrastructure_component/dma_infrastructure/vci_dma/caba/source/src/vci_dma.cp

## CABA Constructor parameters

```
VciDma(
     sc_module_name name,   //  Component Name
     const soclib::common::MappingTable &mt,   // MappingTable
     const soclib::common::IntTab &srcid,  // Initiator index
     const soclib::common::IntTab &tgtid,  // Target index
     const size_t burst_size );   //  Number of bytes transfered in a burst
```

## 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
- sc_out<bool> **p_irq** : Interrupt port

# 4) TLM-DT implementation

The TLM-DT implementation is not available yet.