

# VciDma

## 1) Functional Description

This VCI 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.

There is only one DMA context handled at a time. An IRQ is optionally asserted when transfer is finished. This hardware component checks for segmentation violation, and can be used as a default target.

This component has 4 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 & DMADSR, 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 acknowledge 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', **vci_parameters )

```

## 3) CABA Implementation

### CABA sources

- interface :  
[source:trunk/soclib/soclib/module/infrastructure\\_component/dma\\_infrastructure/vci\\_dma/caba/source/include/vci\\_dma.h](#)
- implementation :  
[source:trunk/soclib/soclib/module/infrastructure\\_component/dma\\_infrastructure/vci\\_dma/caba/source/src/vci\\_dma.cpp](#)

### CABA Constructor parameters

```

VciDma (
    sc_module_name name,    // Component Name
    const soclib::common::IntTab & index, // Target index
    const soclib::common::MappingTable &mt, // MappingTable
    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-T implementation

The TLM-T implementation is not available yet.