

# VciMasterNic

## 1) Functional Description

The VciMasterNic component, is a GMII compliant, network controller for Gigabit Ethernet network, with a built-in DMA capability.

It can support a throughput of 1 Gigabit/s, as long as the system clock frequency is larger or equal to the GMII clock frequency (ie 125 MHz).

To improve the throughput, this component supports up to 8 channels. These channels are indexed by a *key* derived from the (source) remote IP address and port for the received (RX) packets, and from the (destination) remote IP address and port for the sent (TX) packets:

```
uint32_t key = ( ((addr      ) & 0xFF) +  
                 ((addr > 8 ) & 0xFF) +  
                 ((addr > 16) & 0xFF) +  
                 ((addr > 24) & 0xFF) +  
                 ((port     ) & 0xFF) +  
                 ((port > 8 ) & 0xFF) ) % nb_channels;
```

The actual number of channels is an hardware parameter. The Ethernet packet length can have any value, in the range [42 to 2040] bytes.

The data transfer unit between software and the NIC is a 2K bytes **container**, containing one single Ethernet packet.

The received packets (RX) and the sent packets (TX) are stored in two memory mapped software queues, called *chained buffer*, and defined by the **nic\_chbuf\_s** C structure. **Each slot in the queue is a container. The number of containers, defining the queue depth, is a software defined parameter. The physical addresses are used by the hardware NIC DMA engines. The virtual addresses are used by the software NIC drivers.**

```
struct nic_chbuf_s  
{  
    uint32_t    wid;                               /*! current container write index    */  
    uint32_t    rid;                               /*! current container read index     */  
    uint64_t    cont_pad[SOC_LIB_NIC_CHBUF_DEPTH]; /*! containers physical base addresses */  
    uint32_t *  cont_ptr[SOC_LIB_NIC_CHBUF_DEPTH]; /*! containers virtual base addresses */  
}
```

The **nic\_cont\_s** C structure contains a 2040 bytes data buffer, the actual ethernet packet length (in bytes), and the container state : full (owned by the reader) / empty (owned by the writer). This state variable is used as a SET/RESET flip-flop to synchronize the software server thread, and the hardware NIC DMA engine.

```
struct nic_cont_s  
{  
    uint8_t     buf[2040];                         /*! Ethernet packet (42 to 2040 bytes) */  
    uint32_t     length;                            /*! actual packet length in bytes      */  
    uint32_t     state;                             /*! zero == empty / non zero == full   */  
}
```

Inside the NIC controller, each channel implements a 2 slots chained buffer (two containers) for RX, and another 2 slots chained buffer (two containers) for TX. For each channel, the build-in RX\_DMA engine moves the RX containers from the internal 2 slots chained buffer to the external chained buffer implementing the RX queue in

memory. Another build-in TX-DMA engine moves the TX containers from the external chained buffer implementing the TX queue in memory, to the internal TX 2 slots chained buffer.

To improve the throughput for one specific channel, the DMA engines use *pipelined bursts*: The burst length cannot be larger than 64 bytes, but each channel send 4 pipelined VCI transactions to mask the round-trip latency. Therefore, this NIC controller can control up to 64 parallel VCI transactions (8 channels \* 4 bursts \* 2 directions). The CMD/RSP matching uses both the VCI TRDID and PKTID fields:

- the channel index is sent in SRCID
- the burst index is sent in TRDID[1:0]
- the is\_rx bit is sent in TRDID[2]

Regarding the GMII physical interface, the systemC simulation model supports three modes of operation, defined by a constructor parameter:

- **NIC\_MODE\_FILE**: Both the RX packets stream and the TX packets stream are read/written from/to dedicated files "nic\_rx\_file.txt" and "nic\_tx\_file.txt", stored in the same directory as the top.cpp file.
- **NIC\_MODE\_SYNTHESIS**: The TX packet stream is still written to the "nic\_tx\_file.txt" file, but the RX packet stream is synthesised. The packet length (between 42 and 1538 bytes) and the source MAC address (8 possible values) are pseudo-random numbers.
- **NIC\_MODE\_TAP**: The TX and RX packet streams are sent and received to and from the physical network controller of the workstation running the simulation.

## 2) Addressable registers

The addressable registers can be split in two classes: *global* registers, and *channel* registers.

### 2.1) global registers

These registers are used for global NIC configuration or status, and are not linked to a specific channel.

<b>NIC_G_CHANNELS</b>	Read Only	returns actual number of channels
<b>NIC_G_NPKT_RESET</b>	Write Only	reset all packets counters
<b>NIC_G_NPKT_RX_G2S_RECEIVED</b>	Read Only	packets received on GMII RX port
<b>NIC_G_NPKT_RX_G2S_DISCARDED</b>	Read Only	RX packets discarded by RX_G2S FSM
<b>NIC_G_NPKT_RX_DES_SUCCESS</b>	Read Only	RX packets transmitted by RX_DES FSM
<b>NIC_G_NPKT_RX_DES_TOO_SMALL</b>	Read Only	discarded too small RX packets (<60B)
<b>NIC_G_NPKT_RX_DES_TOO_BIG</b>	Read Only	discarded too big RX packets (>1514B)
<b>NIC_G_NPKT_RX_DES_MFIFO_FULL</b>	Read Only	discarded RX packets if fifo full
<b>NIC_G_NPKT_RX_DES_CRC_FAIL</b>	Read Only	discarded RX packets if CRC32 failure
<b>NIC_G_NPKT_RX_DISP_RECEIVED</b>	Read Only	packets received by RX_DISPATCH FSM
<b>NIC_G_NPKT_RX_DISP_BROADCAST</b>	Read Only	broadcast RX packets received
<b>NIC_G_NPKT_RX_DISP_CH_FULL</b>	Read Only	discarded RX packets if channel full
<b>NIC_G_NPKT_TX_DISP_RECEIVED</b>	Read Only	packets received by TX_DISPATCH FSM
<b>NIC_G_NPKT_TX_DISP_TOO_SMALL</b>	Read Only	discarded too small TX packets (<60B)
<b>NIC_G_NPKT_TX_DISP_TOO_BIG</b>	Read Only	discarded too big TX packets (>1514B)
<b>NIC_G_NPKT_TX_DISP_TRANSMIT</b>	Read Only	transmitted TX packets

## 2.2) Channel registers

These registers are replicated for each channel.

<b>NIC_RX_CHANNEL_RUN</b>	Write Only	channel activation
<b>NIC_RX_CHBUF_DESC_LO</b>	Read/Write	RX chbuf descriptor low word
<b>NIC_RX_CHBUF_DESC_HI</b>	Read/Write	RX chbuf descriptor high word
<b>NIC_RX_CHBUF_NBUFS</b>	Read/WRITE	RX chbuf depth (buffers)
<b>NIC_RX_CHANNEL_STATE</b>	Read Only	RX channel status
<b>NIC_TX_CHANNEL_RUN</b>	Write Only	TX channel activation
<b>NIC_TX_CHBUF_DESC_LO</b>	Read/Write	TX chbuf descriptor low word
<b>NIC_TX_CHBUF_DESC_HI</b>	Read/Write	TX chbuf descriptor high word
<b>NIC_TX_CHBUF_NBUFS</b>	Read/Write	TX chbuf depth (buffers)
<b>NIC_TX_CHANNEL_STATE</b>	Read Only	TX channel status

For extensibility issues, you should access all these registers using the globally-defined offsets in file

[source:trunk/soclib/soclib/module/connectivity\\_component/vci\\_master\\_nic/include/soclib/master\\_nic.h?](#)

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

## 3) Component definition & usage

[source:trunk/soclib/soclib/module/connectivity\\_component/vci\\_master\\_nic/caba/metadata/vci\\_master\\_nic.sd?](#)

```
Uses( 'vci_master_nic' )
```

## 4) CABA Implementation

### CABA sources

- interface :  
[source:trunk/soclib/soclib/module/connectivity\\_component/vci\\_master\\_nic/caba/source/include/vci\\_master\\_nic.h?](#)
- implementation :  
[source:trunk/soclib/soclib/module/connectivity\\_component/vci\\_master\\_nic/caba/source/src/vci\\_master\\_nic.cpp?](#)

### CABA Constructor parameters

```
VciMasterNic(  
    sc_core::sc_module_name name,    // Component Name  
    const soclib::common::MappingTable &mt,    // MappingTable  
    const soclib::common::IntTab &rx_srcid,    // RX DMA initiator index  
    const soclib::common::IntTab &tx_srcid,    // TX DMA initiator index  
    const soclib::common::IntTab &tgtid,    // target index  
    const size_t channels,    // Number of channels  
    const uint32_t burst_order,    // ln2( dma_burst_size )  
    const int mode,    // GMII physical interface modeling  
    const uint32_t inter_frame_gap);    // delay between two packets
```

## CABA Ports

- **p\_resetn** : Global system reset
- **p\_clk** : Global system clock
- **p\_vci** : The VCI target port
- **p\_rx\_irq[k]** : As many RX IRQ ports as the number of channels
- **p\_tx\_irq[k]** : As many TX IRQ ports as the number of channels