Nano Scale Disruptive Silicon-Plasmonic Platform for Chip-to-Chip Interconnection

Dual Die Communication Module functional specification

Deliverable no.:	D5.1
Due date:	04/30/2012
Actual Submission date:	04/30/2012
Authors:	ST
Work package(s):	WP5
Distribution level:	CO[1] (NAVOLCHI Consortium)
Nature:	Document, available online in the restricted area of the NAVOLCHI webpage

List of Partners concerned

Partner number	Partner name	Partner short name	Country	Date enter project	Date exit project
1	Karlsruher Institut für Technologie	KIT	Germany	M1	M36
2	INTERUNIVERSITAIR MICRO-ELECTRONICA CENTRUM VZW	IMCV	Belgium	M1	M36
3	TECHNISCHE UNIVERSITEIT EINDHOVEN	TU/e	Netherlands	M1	M36
4	RESEARCH AND EDUCATION LABORATORY IN INFORMATION TECHNOLOGIES	AIT	Greece	M1	M36
5	UNIVERSITAT DE VALENCIA	UVEG	Spain	M1	M36
6	STMICROELECTRONICS SRL	ST	Italy	M1	M36
7	UNIVERSITEIT GENT	UGent	Belgium	M1	M36

[1] **PU** = Public
PP = Restricted to other programme participants (including the Commission Services)
RE = Restricted to a group specified by the consortium (including the Commission Services)
CO = Confidential, only for members of the consortium (including the Commission Services)

Deliverable Responsible

Organization: STMicroelectronics
Contact Person: Alberto Scandurra
Address: Stradale Primosole, 50 – 95121 Catania
Italy
Phone: +39 095 740 4432
Fax: +39 095 740 4008
E-mail: alberto.scandurra@st.com

Executive Summary

This document describes the Dual Die Communication Module (DDCM) architecture and functionality. The DDCM is an Intellectual Property (IP) allowing two dice within a System in Package (SiP) to communicate to each other. Its functionality covers all the layers foreseen by the protocol stack, from transport to data link, but the physical layer (PHY), that can be of different nature, i.e. electrical or optical. This document addresses a first DDCM implementation supporting electrical PHY; in a subsequent phase of the project the DDCM will be modified so to support an optical PHY based on plasmonics.

Change Records

Version	Date	Changes	Author
0.1 (draft)	2012-04-19	Start	Alberto Scandurra
1 (submission)	2012-04-30		Alberto Scandurra

Contents

1 Introduction

The **Dual Die Communication Module** (abbreviated **DDCM**) is the building-block responsible for the interconnection of different dice within a so called Network in Package (NiP), the communication system enabling inter dice data transmission in the context of Systems in Package (SiP) technology.
From an architectural point of view it represents the evolution of the STAC (STNoC Advanced off-chip Communication module).
The main features of the DDCM, differing from STAC, are:

- support of **Spidergon STNoC** initiator and target interfaces;
- IP **flit size and frequency conversion**
- integrated **src-remapper**;
- integrated encoder/decoder for **dynamic power saving**;
- registers and logic for on-chip debugging and performance metrics calculation.

The differences between DDCM and STAC are summarized in the following table.

Feature	**DDCM**	**STAC**
Initiator/target interface	STNoC	VSTNoC
Initiator/target interface size	Any	72 bits
Flit size conversion	Yes	No
Frequency conversion	Yes	No
Src remapper	Yes	No
Dynamic power saving	Yes	No
On-chip debugging and monitoring	Yes	No

Table 0.1 – Differences between DDCM and STAC

In the future the DDCM will support the following additional features:

- support of **STBus** initiator and target interfaces;
- support of **AMBA-AXI** initiator and target interfaces;
- integrated encryption/decryption module for **transactions security**;
- support of bidirectional communication with two neighbourhood dice.

The differences between future DDCM and STAC are summarized in the following table.

Feature	**DDCM**	**STAC**
Initiator/target interface	STNoC, STBus, AMBA-AXI	VSTNoC
External dice interfaces	2	1
Encryption/decryption module	Internal	External

Table 0.2 – Differences between future DDCM and STAC

According to a widely used approach, the DDCM is considered composed of two main building blocks:

- the DDCM **controller**, responsible for managing incoming/outgoing STNoC traffic and IDN segments, generating them through STNoC flits encapsulation and preparing them to be sent to the PHY transmitter, as well as collecting them from the PHY receiver;

- the DDCM **PHY**, responsible for transmitting output phyts across the physical link and collecting inputs phyts from the physical link.

The next two figures show the DDCM structure in terms of top level building-blocks.
Figure 1.1 shows the current structure with external security encoder/decoder, while figure 1.2 shows the future structure with the security encoder/decoder embedded within the DDCM top level.

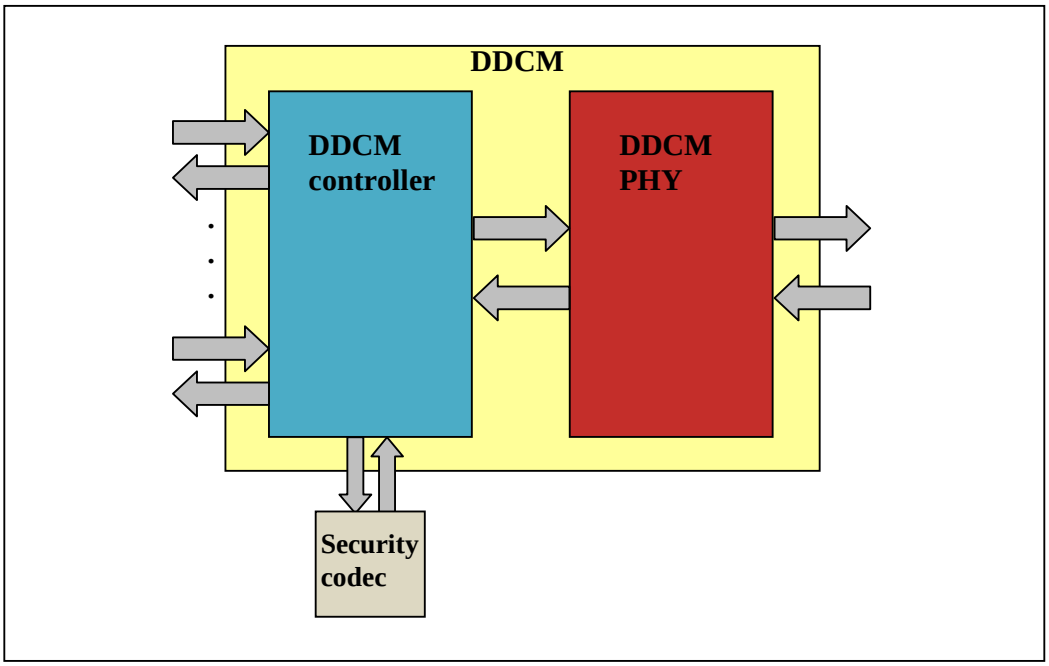


Figure 0-1: DDCM top level structure with external security codec

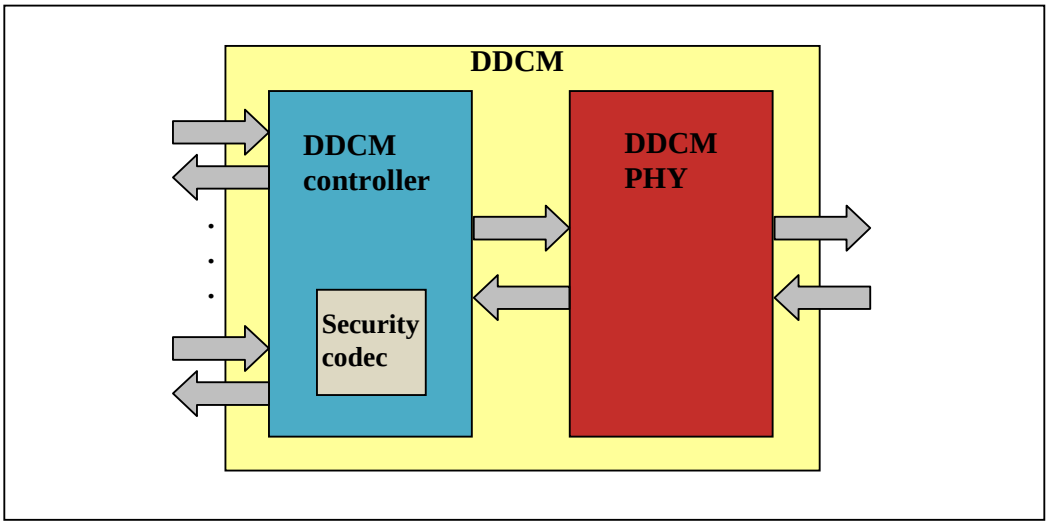


Figure 0-2: DDCM top level structure with internal security codec

2 Parameters

The DDCM is a parametric design that, depending on the SoC where it is used, can be configured properly in order to meet system requirements and needs in terms of interfaces, FIFOs sizes, clock domains synchronization and functionality.

The tables in next subsections list and describe the the DDCM parameters.

Top level

Name	Description	Range	Default
init_port_nb	Number of STNoC initiators.	0 to 16	4
targ_port_nb	Number of STNoC targets.	0 to 16	4
virtual_wires_on	Specifies if to support virtual wires.	true/false	true
src_remapper_on	Specifies if to instantiate the src remapper	true/false	false
power_saving_on	Specifies if to instantiate the codec for dynamic power saving.	true/false	false

Table 2.1: Top level parameters

Initiators

Name	Description	Range	Default
init_i_ds_vn1_on	Specifies if the downstream interface is present in virtual network #1.	true/false	true
init_i_ds_vn2_on	Specifies if the downstream interface is present in virtual network #2.	true/false	false
init_i_ds_flit_size	Downstream interface flit size.	16,18,32,36,64,72,128,144	72
init_i_ds_flit_extra_bits_size	Downstream interface extra bits number.	0 to 144	0
init_i_ds_flit_id_atomic_on	Specifies if the flit_id_atomic port is present in downstream interface.	true/false	false
init_i_ds_flit_id_3_on	Specifies if the flid_id_3 port is present in downstream interface.	true/false	false
init_i_ds_flit_id_err_on	Specifies if the flid_id_err port is present in downstream interface.	true/false	false
init_i_ds_four_be_on	Specifies if the four_be port is present in downstream interface.	true/false	false
init_i_ds_four_be_size	Downstream interface four_be port size.	0 to 4	0
init_i_us_vn1_on	Specifies if the upstream interface is present in virtual network #1.	true/false	true
init_i_us_vn2_on	Specifies if the upstream interface is present in virtual network #2.	true/false	false
init_i_us_flit_size	Upstream interface flit size.	16,18,32,36,64,72,128,144	72

init_i_us_flit_extra_bits_size	Upstream interface extra bits number.	0 to 144	0
init_i_us_flit_id_3_on	Specifies if the flid_id_3 port is present in upstream interface.	true/false	false
init_i_us_flit_id_err_on	Specifies if the flid_id_err port is present in upstream interface.	true/false	false
init_i_us_four_be_on	Specifies if the four_be port is present in upstream interface.	true/false	false
init_i_us_four_be_size	Upstream interface four_be port size.	0 to 4	0

Table 2.2: Initiators parameters

Targets

Name	Description	Range	Default
targ_i_ds_vn1_on	Specifies if the downstream interface is present in virtual network #1.	true/false	true
targ_i_ds_vn2_on	Specifies if the downstream interface is present in virtual network #2.	true/false	false
targ_i_ds_flit_size	Downstream interface flit size.	16,18,32,36,64,72,128,144	72
targ_i_ds_flit_extra_bits_size	Downstream interface extra bits number.	0 to 144	0
targ_i_ds_flit_id_atomic_on	Specifies if the flit_id_atomic port is present in downstream interface.	true/false	false
targ_i_ds_flit_id_3_on	Specifies if the flid_id_3 port is present in downstream interface.	true/false	false
targ_i_ds_flit_id_err_on	Specifies if the flid_id_err port is present in downstream interface.	true/false	false
targ_i_ds_four_be_on	Specifies if the four_be port is present in downstream interface.	true/false	false
targ_i_ds_four_be_size	Downstream interface four_be port size.	0 to 4	0
targ_i_us_vn1_on	Specifies if the upstream interface is present in virtual network #1.	true/false	true
targ_i_us_vn2_on	Specifies if the upstream interface is present in virtual network #2.	true/false	false
targ_i_us_flit_size	Upstream interface flit size.	16,18,32,36,64,72,128,144	72
targ_i_us_flit_extra_bits_size	Upstream interface extra bits number.	0 to 144	0
targ_i_us_flit_id_3_on	Specifies if the flid_id_3 port is present in upstream interface.	true/false	false
targ_i_us_flit_id_err_on	Specifies if the flid_id_err port is present in upstream interface.	true/false	false
targ_i_us_four_be_on	Specifies if the four_be port is present in upstream interface.	true/false	false
targ_i_us_four_be_size	Upstream interface four_be port size.	0 to 4	0

Table 2.3: Targets parameters

Virtual wires

Name	Description	Range	Default
bundle_i_tx_size	Size, in terms of number of wires, of virtual wires bundle #i in DDCM transmitter.	0 to 80	80
bundle_i_sam_rate	Sampling rate of the bundle #i of the virtual wires DDCM transmitter port.	0 to 10	1
bundle_i_rx_size	Size, in terms of number of wires, of virtual wires bundle #i in DDCM receiver.	0 to 80	80

Table 2.4: Virtual wires parameters

Clock domains synchronization

Name	Description	Range	Default
idn_plug_synch_dff_nb	Number of synchronization flip-flops in DDCM clock domain.	1 to 8	2
prog_synch_dff_nb	Number of synchronization flip-flops in programming clock domain.	1 to 8	2
phy_tx_synch_dff_nb	Number of synchronization flip-flops in DDCM PHY transmitter clock domain.	1 to 8	2
phy_rx_synch_dff_nb	Number of synchronization flip-flops in DDCM PHY receiver clock domain.	1 to 8	2

Table 2.5: Clock domains synchronization parameters

Retiming

Name	Description	Range	Default
init_i_ds_retiming	Specifies whether a retiming stage has to be istantiated at initiator #i downstream interface.	true/false	false
targ_i_ds_retiming	Specifies whether a retiming stage has to be istantiated at target #i downstream interface.	true/false	false
init _i_us_retiming	Specifies whether a retiming stage has to be istantiated at initiator #i upstream interface.	true/false	false
targ_i_us_retiming	Specifies whether a retiming stage has to be istantiated at target #i upstream interface.	true/false	false

Table 2.6: Retiming parameters

FIFOs

Name	Description	Range	Default
init_i_tx_fifo_size	Size of the FIFO related to initiator port #i in DDCM transmitter.	2 to 128	8
targ_i_tx_fifo_size	Size of the FIFO related to target port #i in DDCM transmitter.	2 to 128	8
init_i_rx_fifo_ size	Size of IDN segment FIFO related to initiator port #i in DDCM receiver.	8 to 128	8
targ_i_rx_fifo_ size	Size of IDN segment FIFO related to target port #i in DDCM receiver.	8 to 128	8
targ_i_rx_fifo_saf_reset	Value after reset of the register specifying if store & forward mechanism has to be applied by the different FIFOs in DDCM receiver.	0 to 1	1

Table 2.7: FIFOs parameters

Credit-based flow control

Name	Description	Range	Default
targ_i_rx_fifo_ threshold	Number of freed locations the IDN segment FIFO associated to target port #i of DDCM receiver must have in order to send a credit information.	0 to 7 n : 2^n locations (0<= n <= 5) 6 : half FIFO 7 : whole FIFO	3
init_i_rx_fifo_ threshold	Number of freed locations the IDN segment FIFO #i associated to an initiator port of DDCM receiver must have in order to send a credit information.	0 to 7 n : 2^n locations (0<= n <= 5) 6 : half FIFO 7 : whole FIFO	3
targ_i_rx_fifo_credit_period	Value after reset of the frequency with which the credits information has to be sent for target IDN segment FIFO #i from the QoS module of the DDCM receiver.	0 to 7 0 : 4 cycles 1 : 8 cycles 2 : 16 cycles 3 : 32 cycles 4 : 64 cycles 5 : 128 cycles 6 : 256 cycles 7 : 512 cycles	7
init_i_rx_fifo_credit_period	Value after reset of the frequency with which the credits information has to be sent for initiator IDN segment FIFO #i from the QoS module of the DDCM receiver.	0 to 7 0 : 4 cycles 1 : 8 cycles	7

		2 : 16 cycles 3 : 32 cycles 4 : 64 cycles 5 : 128 cycles 6 : 256 7 : 512	

Table 2.8 : Credit-based flow control parameters

3 Interfaces

System

The system interface consists of the clocks on which the operation of the DDCM is based, and the asynchronous reset used for the initialization of the block.
Notice that the reset signal is managed as an asynchronous signal, and its synchronization with respect to the related clock is performed by proper modules inside the DDCM.

Signal name	I/O	Timing	Description
rst_n	I	N/A	Asynchronous active low reset
clk_plug	I	N/A	DDCM clock
clk_prog	I	N/A	Programming clock
clk_phy_tx	I	N/A	DDCM PHY transmitter clock
clk_phy_rx	I	N/A	DDCM PHY receiver clock

Table 3.1 –System interface

Test

The test interface consists of a set of ports allowing to test the DDCM digital modules (scan test) and the physical channel and its controller (PHY) after manufacturing.
Besides the usual scan test signals, i.e. tst_scanenable, tst_scanin, tst_scanout, a tst_mode port is required in order to bypass synchronizers and any synchronization logic during test, because of the need of a unique reset and a unique clock.

Signal name	I/O	Timing	Description
tst_scanenable	I	N/A	Scan test enable
tst_scanin	I	N/A	Scan test input
tst_scanout	O	N/A	Scan test output
tst_mode	I	N/A	Test mode enable
tst_phy_sce_sel	I	N/A	PHY test source selector
tst_pg_hi<15:0>	I	N/A	Pattern generator high data input
tst_pg_lo<15:0>	I	N/A	Pattern generator low data input
tst_pg_vld	I	N/A	Pattern generator data valid

Table 3.2 –Test interface

Notice that the width of scan test input and output signals will depend on the number of scan chains created within the DDCM, according to synthesis results.

Configuration

The configuration interface consists of a set of inputs (mode pins) allowing to configure the DDCM functionality after reset, so to be adapted to different contexts. Notice that the same functionality can be re-programmed through the correspondent registers accessible via the DDCM programming interface.

It's important to point out that this set of configuration signals is related to the DDCM implementation working with the electrical PHY able to operate in both DCE/SCE modes. Using a different PHY and related PHY adapter these parameters realistically will change.
The configuration interface is synchronous with the clk_plug clock.

Signal name	I/O	Timing	Description
phy_tx_width	I	Late	When '0' all the 16 bits of the PHY transmitter interface are used, when '1' only 8 are used
phy_rx_width	I	Late	When '0' all the 16 bits of the PHY receiver interface are used, when '1' only 8 are used
phy_mode	I	Late	When '0' the PHY works in DCE mode, when '1' in SCE mode
lpe_tx_bypass	I	Late	When '0' the bus inverter transmitter for dynamic power optimization is bypassed, when '1' it's used

Table 3.3 –Configuration interface

Initiator

The initiator interface consists of standard STNoC initiator ports, i.e. a set of signals replicated a number of times according to how many initiators are connected to the DDCM.
If an initiator interface is synchronous with a clock differing from the DDCM main clock, the required frequency conversion is performed inside the DDCM itself.
The following tables report the downstream and upstream interfaces for a generic STNoC initiator, identified as initiator #i (1 <= i <= 16).

Signal name	I/O	Timing	Description
init_i_ds_flit<init_i_ds_flit_size+init_i_ds_flit_extra_bits_size-1:0>	I	Early	STNoC flit
init_i_ds_flit_id<1:0>	I	Early	Flit identifier
init_i_ds_flit_id_3	I	Early	Flit identifier bit 3
init_i_ds_flit_id_err<1:0>	I	Early	Error marker
init_i_ds_flit_id_atomic	I	Early	Atomic transaction flag
init_i_ds_four_be<init_i_ds_four_be_size-1:0>	I	Early	Four byte-enables
init_i_ds_vn1_val	I	Early	Virtual network #1 valid
init_i_ds_vn1_credit	O	Early	Virtual network #1 credit
init_i_ds_vn2_val	I	Early	Virtual network #2

			valid
init_i_ds_vn2_credit	O	Early	Virtual network #2 credit

Table 3.4 – STNoC initiator downstream interface

Signal name	I/O	Timing	Description
init_i_us_flit<init_i_us_flit_size+init_i_us_flit_extra_bits_size-1:0>	O	Early	STNoC flit
init_i_us_flit_id<1:0>	O	Early	Flit identifier
init_i_us_flit_id_3	O	Early	Flit identifier bit 3
init_i_us_flit_id_err<1:0>	O	Early	Error marker
init_i_us_flit_id_atomic	O	Early	Atomic transaction flag
init_i_us_four_be<init_i_us_four_be_size-1:0>	O	Early	Four byte-enables
init_i_us_vn1_val	O	Early	Virtual network #1 valid
init_i_us_vn1_credit	I	Early	Virtual network #1 credit
init_i_us_vn2_val	O	Early	Virtual network #2 valid
init_i_us_vn2_credit	I	Early	Virtual network #2 credit

Table 3.5 – STNoC initiator upstream interface

Target

The target interface consists of standard STNoC target ports, i.e. a set of signals replicated a number of times according to how many targets are connected to the DDCM.
If a target interface is synchronous with a clock differing from the DDCM main clock, the required frequency conversion is performed inside the DDCM itself.
The following tables report the downstream and upstream interfaces for a generic STNoC target, identified as target #i (1 <= i <= 16).

Signal name	I/O	Timing	Description
targ_i_ds_flit<targ_i_ds_flit_size+targ_i_ds_flit_extra_bits_size-1:0>	I	Early	STNoC flit
targ_i_ds_flit_id<1:0>	I	Early	Flit identifier

targ_i_ds_flit_id_3	I	Early	Flit identifier bit 3
targ_i_ds_flit_id_err<1:0>	I	Early	Error marker
targ_i_ds_flit_id_atomic	I	Early	Atomic transaction flag
targ_i_ds_four_be<targ_i_ds_four_be_size-1:0>	I	Early	Four byte-enables
targ_i_ds_vn1_val	I	Early	Virtual network #1 valid
targ_i_ds_vn1_credit	O	Early	Virtual network #1 credit
targ_i_ds_vn2_val	I	Early	Virtual network #2 valid
targ_i_ds_vn2_credit	O	Early	Virtual network #2 credit

Table 3.6 –STNoC target downstream interface

Signal name	**I/O**	**Timing**	**Description**
targ_i_us_flit<targ_i_us_flit_size+targ_i_us_flit_extra_bits_size-1:0>	O	Early	STNoC flit
targ_i_us_flit_id<1:0>	O	Early	Flit identifier
targ_i_us_flit_id_3	O	Early	Flit identifier bit 3
targ_i_us_flit_id_err<1:0>	O	Early	Error marker
targ_i_us_flit_id_atomic	O	Early	Atomic transaction flag
targ_i_us_four_be<targ_i_us_four_be_size-1:0>	O	Early	Four byte-enables
targ_i_us_vn1_val	O	Early	Virtual network #1 valid
targ_i_us_vn1_credit	I	Early	Virtual network #1 credit
targ_i_us_vn2_val	O	Early	Virtual

			network #2 valid
targ_i_us_vn2_credit	I	Early	Virtual network #2 credit

Table 3.7 –STNoC target upstream interface

Virtual wires

The virtual routing interface consists of a set of bundles configurable in terms of size, whose individual bits represent specific signals carrying specific information, such as interrupts, power down control, asynchronous events. Such bundles are sampled at given rates, and depending on them their content is transmitted across the DDCM to the second die.

Signal name	**I/O**	**Timing**	**Description**
bundle_1_tx< bundle_1_tx_size-1:0>	I	N/A	Virtual wires input bundle #1
bundle_2_tx< bundle_2_tx_size-1:0>	I	N/A	Virtual wires input bundle #2
bundle_3_tx< bundle_3_tx_size-1:0>	I	N/A	Virtual wires input bundle #3
bundle_4_tx< bundle_4_tx_size-1:0>	I	N/A	Virtual wires input bundle #4
bundle_5_tx< bundle_5_tx_size-1:0>	I	N/A	Virtual wires input bundle #5

Table 3.8 –DDCM transmitter virtual wires interface

Signal name	I/O	Timing	Description
bundle_1_rx< bundle_1_rx_size-1:0>	O	Early	Virtual wires output bundle #1
bundle_2_rx< bundle_2_rx_size-1:0>	O	Early	Virtual wires output bundle #2
bundle_3_rx< bundle_3_rx_size-1:0>	O	Early	Virtual wires output bundle #3
bundle_4_rx< bundle_4_rx_size-1:0>	O	Early	Virtual wires output bundle #4
bundle_5_rx< bundle_5_rx_size-1:0>	O	Early	Virtual wires output bundle #5

Table 3.9 –DDCM receiver virtual wires interface

Notice that virtual wires inputs are asynchronous, and they are synchronized internally to the DDCM.

Programming

The programming interface consists of a standard STBus type 1 interface allowing to access the internal registers in order to configure the operation of the DDCM, mainly in terms of QoS policy. The configuration of the DDCM registers can be done either during the initialization phase or on-fly, in the sense that the registers specifying the QoS policy can be modified even during the normal operation of the block.
The programming interface is synchronous with the clk_prog clock.

Signal name	**I/O**	**Timing**	**Description**
prog_req	I	Late	Request
prog_eop	I	Late	End of packet

prog_opc<3:0>	I	Late	Operation code
prog_add<7:2>	I	Late	Address
prog_data<31:0>	I	Late	Write data
prog_be<3:0>	I	Late	Byteenables
prog_r_req	O	Early	Response request
prog_r_opc	O	Early	Operation status
prog_r_data<31:0>	O	Early	Read data

Table 3.10 – Programming interface

Security encoder

The security encoder interface consists of a set of inputs representing the masks to be used for the encoding/decoding of the outgoing/incoming flits from/to the DDCM PHY adapter, in order to encrypt them for protection against any hacking.

Signal name	I/O	Timing	Description
phyt_hi_enc_tx<15:0>	I	N/A	Transmitter phyt_hi mask
phyt_lo_enc_tx<15:0>	I	N/A	Transmitter phyt_lo mask
phyt_hi_enc_rx<15:0>	I	N/A	Receiver phyt_hi mask
phyt_lo_enc_rx<15:0>	I	N/A	Receiver phyt_lo mask

Table 3.11 – Security encoder interface

Physical channel

The physical channel interface (PHY) is responsible for the actual transmission of data between dice.

According to the layered approach followed by DDCM implementation, the PHY can change case by case according to specific system requirements and technology availability. The PHY interface will then change accordingly.

This section reports the interface of the PHY in case of 16 bits wide electrical physical channel following the DCE/SCE approach. Differently PHY structure realistically will have different interfaces.

For any detail refer to the specific PHY documentation.

When a data is ready to be transmitted a data valid signal is issued; moreover, the clock is required to be transmitted as well, since the DDCM transmitter and receiver are physically located in two different chips and then are clocked by different clocks, so that the communication between them is asynchronous.

DDCM transmitter PHY is synchronous with the clk_tx clock.

Signal name	I/O	Timing	Description
tx_phyt<15:0>	O	late	Data to be transmitted
tx_valid	O	late	Data valid signal
tx_clock	O	late	Transmission clock for synchronization

Table 3.12 – Physical channel transmitter interface

DDCM receiver PHY is instead synchronous with the clk_rx clock.

Signal name	I/O	Timing	Description
rx_phyt<15:0>	I	late	Data to be transmitted
rx_valid	I	late	Data valid signal
rx_clock	I	late	Transmission clock for synchronization

Table 3.13 – Physical channel receiver interface

DDCM controller PHY adapter interface

Since, thanks to the layered protocol approach, the DDCM can be implemented with different PHY structures (DCE, SCE, parallel, serial, optical, RF), it's convenient to specify also the PHY adapter interface, even if it is internal to the DDCM.
The interfaces described in the following tables are related to an electrical PHY working in DCE/SCE mode. In case of different PHY, the PHY adapter interfaces with the DDCM digital parts will not change, while the interfaces with the PHY will change according to the specific PHY structure.

Signal name	I/O	Timing	Description
seg<89:0>	I	Early	DDCM segment coming from layer B
seg_val	I	Early	DDCM segment valid flag (active high)
seg_ack	O	Early	DDCM segment acknowledge (active high)
tx_phy_mode	O	Early	PHY operation mode (DCE/SCE) control flag
tx_phyt_hi<15:0>	O	Early	Phyt to be sent during clock rising edge
tx_phyt_lo<15:0>	O	Early	Phyt to be sent during clock falling edge
tx_phyt_valid	O	Early	Specifies the phyt is ready to be transmitted
tx_phy_tx_clk_enable	O	Early	PHY transmitter clock enable (active high)

Table 3.14 – PHY adapter transmitter interface

Notice that, if the phy_mode configuration input is set to '1', and then the PHY works in SCE mode, only phyt_hi output is meaningful for the PHY adapter transmitter, and only phyt_hi input is meaningful for the PHY adapter receiver.

Signal name	I/O	Timing	Description
rx_phyt_hi<15:0>	I	Early	Phyt received by the PHYduring clock rising edge
rx_phyt_lo<15:0>	I	Early	Phyt received by the PHY during clock falling edge
rx_phyt_valid	I	Early	Specifies the phyt is ready to be kept
seg<89:0>	O	Early	DDCM segment going to layer B
seg_val	O	Early	DDCM segment valid flag (active high)

Table 3.15 – PHY adapter receiver interface

Timing

The timing of all the DDCM ports depends on the technology used to synthesize the design; as example, the timing to be adopted when using the CMOS technology at 28 nm is defined as follows:

- **Early** means within the 30% of the clock cycle; an early input refers to a signal coming from a register located into a module very close to the DDCM; an early output refers to a signal leaving a register of the DDCM.
- **Late** means within the 60% of the clock cycle; a late input refers to a signal coming from a module placed quite far from the DDCM, so that the delay of the wire crossed by such a signal has an impact on the arrival time to the DDCM input; a late output refers to a signal crossing some combinational logic before leaving the DDCM.
- **Mid** means within the 40% of the clock cycle; a mid input refers to a signal coming from a module placed not far from the DDCM, so that the delay of the wire crossed by such a signal has not a big impact on the arrival time to the DDCM input; a mid output refers to a signal crossing some small combinational logic before leaving the DDCM.
- **N/A** means an input is asynchronous with respect to DDCM clock period.

4 Registers

The DDCM is programmable in terms of some functionalities, in particular layer A (PHY adapter, PHY) operation and routing (virtual channels), trough a set of registers. QoS management is also planned to be configurable via registers.

DDCM registers are memory-mapped, and all of them are 32-bits wide and 32 bits-aligned.

The list of registers contained within the optional configuration module of the DDCM is shown in next table in case of a DDCM with N virtual channels.

Address	Name	Description
Base+0x00	PHY_WIDTH	PHY width actually used for transmission
Base+0x04	PHY_MODE	PHY transmission mode (DCE, SCE)
Base+0x08	INIT_1_8_TX_VC_ID	Tx virtual channel – initiator port association (set 1)
Base+0x0C	INIT_9_16_TX_VC_ID	Tx virtual channel – initiator port association (set 2)
Base+0x10	TARG_1_8_TX_VC_ID	Tx virtual channel – target port association (set 1)
Base+0x14	TARG_9_16_TX_VC_ID	Tx virtual channel – target port association (set 2)
Base+0x18	INIT_1_8_RX_FIFO_ID	Rx FIFO – initiator port association (set 1)
Base+0x1C	INIT_9_16_RX_FIFO_ID[2]	Rx FIFO – initiator port association (set 2)
Base+0x20	TARG_1_8_RX_FIFO_THRESHOLD	Target rx FIFOs threshold for credits transmission (set 1)
Base+0x24	TARG_9_16_RX_FIFO_THRESHOLD	Target rx FIFOs threshold for credits transmission (set 2)
Base+0x28	INIT_1_8_RX_FIFO_THRESHOLD	Initiator rx FIFOs threshold for credits transmission (set 1)
Base+0x2C	INIT_9_16_RX_FIFO_THRESHOLD	Initiator rx FIFOs threshold for credits transmission (set 2)
Base+0x30	TARG_1_8_CREDIT_TIMEOUT	Target rx FIFOs credit transmission timeout (set 1)
Base+0x34	TARG_9_16_CREDIT_TIMEOUT	Target rx FIFOs credit transmission timeout (set 2)
Base+0x38	INIT_1_8_CREDIT_TIMEOUT	Initiator rx FIFOs credit transmission timeout (set 1)
Base+0x3C	INIT_9_16_CREDIT_TIMEOUT	Initiator rx FIFOs credit transmission timeout (set 2)
Base+0x40	TARG_1_8_RX_FIFO_PRI	Target FIFO priorities in DDCM rx for FC arbiter (set 1)
Base+0x44	TARG_9_16_RX_FIFO_PRI	Target FIFO priorities in DDCM rx

[2] Registers specifying ports to FIFOs association are not supported by DDCM v1.0

		for FC arbiter (set 2)
Base+0x48	INIT_1_8_RX_FIFO_PRI	Initiator FIFO priorities in DDCM rx for FC arbiter (set 1)
Base+0x4C	INIT_9_16_RX_FIFO_PRI	Initiator FIFO priorities in DDCM rx for FC arbiter (set 2)
Base+0x50	QOS	Enables specific QoS algorithms
Base+0x54	BUNDLE_SIZE	Virtual wires bundles size
Base+0x58	ADCK_EN	Activity driven clock enable for each clock domain
Base+0x5C	BI_TX_BYPASS	BI transmitter bypass enable
Base+0x60	TARG_RX_FIFO_SAF	Target FIFO store and forward enable
Base+0x64	WIRES_SAM_RATE	Virtual wires sampling rate
Base+0x68	DDCM_PHY_FREQ_RATIO	Info on frequency ratio between DDCM and DDCM PHY
Base+0x6C	IPORT_1_BWL	Bandwidth limiter parameters for initiator port #1
Base+0x70	IPORT_2_BWL	Bandwidth limiter parameters for initiator port #2
. . .	. . .	. . .
Base+0xA4	IPORT_15_BWL	Bandwidth limiter parameters for initiator port #15
Base+0xA8	IPORT_16_BWL	Bandwidth limiter parameters for initiator port #16
Base+0xAC	PHY_DEBUG_MODE	PHY input selector in debug mode
Base+0xB0	INIT_1_8_TX_VC_PRI	Initiator tx virtual channel priorities (set 1)
Base+0xB4	INIT_9_16_TX_VC_PRI	Initiator tx virtual channel priorities (set 2)
Base+0xB8	TARG_1_8_TX_VC_PRI	Target tx virtual channel priorities (set 1)
Base+0xBC	TARG_9_16_TX_VC_PRI	Target tx virtual channel priorities (set 2)

Table 4-1 – DDCM registers

The PHY_WIDTH, PHY_MODE, and BI_TX_BYPASS registers are set after reset through the DDCM input ports having the same names; such ports are sampled after reset and their values are loaded into the correspondent DDCM registers; subsequently they can be re-programmed dynamically through the DDCM programming interface. This is shown in figure 4.1.

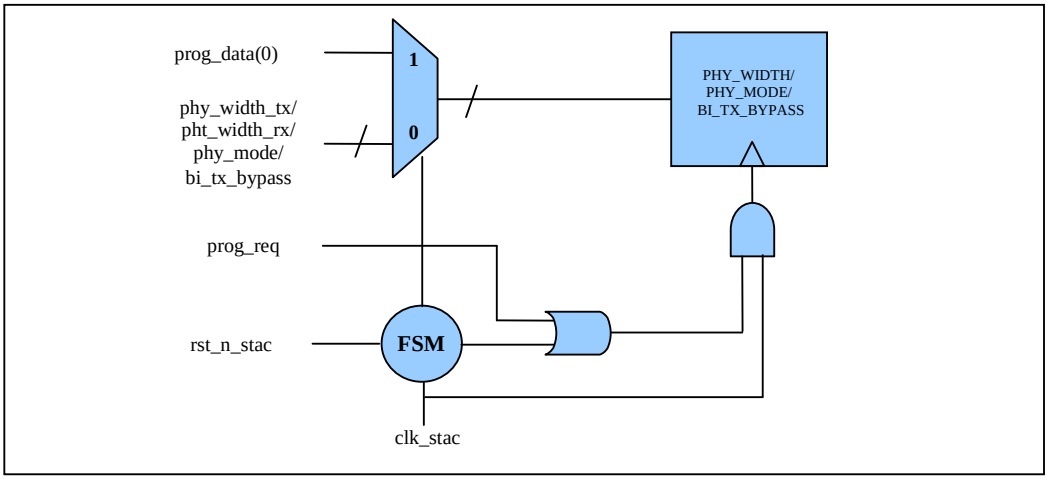


Figure 4-1 – DDCM programming logic

The following tables show in detail the structure and the meaning of each register. Notice that "*not used*" means that FFs are not physically present, while "*reserved*" means the FFs are present but their meaning is not defined at the moment. In both cases, write operations have no effect while read operations return '0'.

Name	Address	Bits	Description
PHY_WIDTH	Base+0x00	<0>	Specify the DDCM transmitter PHY interface data (phyt) size 0 : 16 bits 1 : 8 bits
		<1>	Specify the DDCM receiver PHY interface data (phyt) size 0 : 16 bits 1 : 8 bits
		<31:2>	Not used

Table 4-2 – PHY_WIDTH register structure

Name	Address	Bits	Description
PHY_MODE	Base+0x04	<0>	Specify the PHY operation mode 0 : Dual Clock Edge (DCE) 1 : Single Clock Edge (SCE)
		<31:1>	Not used

Table 4-3 – PHY_MODE register structure

Name	Address	Bits	Description
INIT_1_8_TX_VC_ID	Base+0x08	<3:0>	Specify the ID of the VC connected to initiator port #1
		<7:4>	Specify the ID of the VC connected to initiator port #2
		. . .	. . .
		<31:28>	Specify the ID of the VC connected to initiator port #8

Table 4-4–INIT_1_8_TX_VC_ID register structure

Name	Address	Bits	Description
INIT_9_16_TX_VC_ID	Base+0x0C	<3:0>	Specify the ID of the VC connected to initiator port #9
		<7:4>	Specify the ID of the VC connected to initiator port #10
		. . .	. . .
		<31:28>	Specify the ID of the VC connected to initiator port #16

Table 4-5– INIT_9_16_TX_VC_ID register structure

Name	Address	Bits	Description
TARG_1_8_TX_VC_ID	Base+0x10	<3:0>	Specify the ID of the VC connected to target port #1
		<7:4>	Specify the ID of the VC connected to target port #2
		. . .	. . .
		<31:28>	Specify the ID of the VC connected to target port #8

Table 4-6–TARG_1_8_TX_VC_ID register structure

Name	Address	Bits	Description
TARG_9_16_TX_VC_ID	Base+0x14	<3:0>	Specify the ID of the VC connected to target port #9
		<7:4>	Specify the ID of the VC connected to target port #10
		. . .	. . .
		<31:28>	Specify the ID of the VC connected to target port #16

Table 4-7– TARG_9_16_TX_VC_ID register structure

Name	Address	Bits	Description
INIT_1_8_RX_FIFO_ID	Base+0x18	<3:0>	Specify the ID of the FIFO connected to initiator port #1
		<7:4>	Specify the ID of the FIFO connected to initiator port

			#2
		. . .	. . .
		<31:28>	Specify the ID of the FIFO connected to initiator port #8

Table 4-8–INIT_1_8_RX_FIFO_ID register structure

Name	Address	Bits	Description
INIT_9_16_RX_FIFO_ID	Base+0x1C	<3:0>	Specify the ID of the FIFO connected to initiator port #9
		<7:4>	Specify the ID of the FIFO connected to initiator port #10
		. . .	. . .
		<31:28>	Specify the ID of the FIFO connected to initiator port #16

Table 4-9– INIT_9_16_RX_FIFO_ID register structure

Name	Address	Bits	Description
TARG_1_8_RX_FIFO_THRESHOLD	Base+0x20	<2:0>	Threshold for credit information transmission from target FIFO #1 in DDCM receiver 0 : 1 cell n : 2^n cells (0 < n < 5) 6 : half FIFO 7 : whole FIFO
		<5:3>	Threshold for credit information transmission from target FIFO #2 in DDCM receiver
		<8:6>	Threshold for credit information transmission from target FIFO #3 in DDCM receiver
		<11:9>	Threshold for credit information transmission from target FIFO #4 in DDCM receiver
		<14:12>	Threshold for credit information transmission from target FIFO #5 in DDCM receiver
		<17:15>	Threshold for credit information transmission from target FIFO #6 in DDCM receiver
		<20:18>	Threshold for credit information transmission from target FIFO #7 in DDCM receiver

		<23:21>	Threshold for credit information transmission from target FIFO #8 in DDCM receiver
		<31:24>	Not used

Table 4-10– TARG_1_8_RX_FIFO_THRESHOLD register structure

Name	Address	Bits	Description
TARG_2_16_RX_FIFO_THRESHOLD	Base+0x24	<2:0>	Threshold for credit information transmission from target FIFO #9 in DDCM receiver 0 : 1 cell n : 2^n cells (0 < n < 5) 6 : half FIFO 7 : whole FIFO
		<5:3>	Threshold for credit information transmission from target FIFO #10 in DDCM receiver
		<8:6>	Threshold for credit information transmission from target FIFO #11 in DDCM receiver
		<11:9>	Threshold for credit information transmission from target FIFO #12 in DDCM receiver
		<14:12>	Threshold for credit information transmission from target FIFO #13 in DDCM receiver
		<17:15>	Threshold for credit information transmission from target FIFO #14 in DDCM receiver
		<20:18>	Threshold for credit information transmission from target FIFO #15 in DDCM receiver
		<23:21>	Threshold for credit information transmission from target FIFO #16 in DDCM receiver
		<31:24>	Not used

Table 4-11– TARG_9_16_RX_FIFO_THRESHOLD register structure

Name	Address	Bits	Description
INIT_1_8_RX_FIFO_THRESHOLD	Base+0x28	<2:0>	Threshold for credit information transmission from initiator FIFO #1 in DDCM receiver 0 : 1 cell

			n : 2^n cells (0 < n < 5) 6 : half FIFO 7 : whole FIFO
		<5:3>	Threshold for credit information transmission from initiator FIFO #2 in DDCM receiver
		<8:6>	Threshold for credit information transmission from initiator FIFO #3 in DDCM receiver
		<11:9>	Threshold for credit information transmission from initiator FIFO #4 in DDCM receiver
		<14:12>	Threshold for credit information transmission from initiator FIFO #5 in DDCM receiver
		<17:15>	Threshold for credit information transmission from initiator FIFO #6 in DDCM receiver
		<20:18>	Threshold for credit information transmission from initiator FIFO #7 in DDCM receiver
		<23:21>	Threshold for credit information transmission from initiator FIFO #8 in DDCM receiver
		<31:24>	Not used

Table 4-12– INIT_1_8_RX_FIFO_THRESHOLD register structure

Name	Address	Bits	Description
INIT_9_16_RX_FIFO_THRESHOLD	Base+0x2C	<2:0>	Threshold for credit information transmission from initiator FIFO #9 in DDCM receiver 0 : 1 cell n : 2^n cells (0 < n < 5) 6 : half FIFO 7 : whole FIFO
		<5:3>	Threshold for credit information transmission from initiator FIFO #10 in DDCM receiver
		<8:6>	Threshold for credit information transmission from initiator FIFO #11 in DDCM receiver
		<11:9>	Threshold for credit information transmission from initiator FIFO #12 in DDCM receiver

		<14:12>	Threshold for credit information transmission from initiator FIFO #13 in DDCM receiver
		<17:15>	Threshold for credit information transmission from initiator FIFO #14 in DDCM receiver
		<20:18>	Threshold for credit information transmission from initiator FIFO #15 in DDCM receiver
		<23:21>	Threshold for credit information transmission from initiator FIFO #16 in DDCM receiver
		<31:24>	Not used

Table 4-13– INIT_9_16_RX_FIFO_THRESHOLD register structure

Name	Address	Bits	Description
TARG_1_8_CREDIT_TIMEOUT	Base+0x30	<2:0>	Specifies the credits timeout for target #1 FIFO "000" : 4 cycles "001" : 8 cycles "010" : 16 cycles "011" : 32 cycles "100" : 64 cycles "101" : 128 cycles "110" : 256 cycles "111" : 512 cycles
		<5:3>	Specifies the credits timeout for target #2 FIFO
		<8:6>	Specifies the credits timeout for target #3 FIFO
		<11:9>	Specifies the credits timeout for target #4 FIFO
		<14:12>	Specifies the credits timeout for target #5 FIFO
		<17:15>	Specifies the credits timeout for target #6 FIFO
		<20:18>	Specifies the credits timeout for target #7 FIFO
		<23:21>	Specifies the credits timeout for target #8 FIFO
		<31:24>	Not used

Table 4-14– TARG_1_8_CREDIT_TIMEOUT register structure

Name	Address	Bits	Description
TARG_9_16_CREDIT_TIMEOUT	Base+0x34	<2:0>	Specifies the credits timeout for target #9 FIFO "000" : 4 cycles "001" : 8 cycles "010" : 16 cycles "011" : 32 cycles "100" : 64 cycles "101" : 128 cycles "110" : 256 cycles "111" : 512 cycles
		<5:3>	Specifies the credits timeout for target #10 FIFO
		<8:6>	Specifies the credits timeout for target #11 FIFO
		<11:9>	Specifies the credits timeout for target #12 FIFO
		<14:12>	Specifies the credits timeout for target #13 FIFO
		<17:15>	Specifies the credits timeout for target #14 FIFO
		<20:18>	Specifies the credits timeout for target #15 FIFO
		<23:21>	Specifies the credits timeout for target #16 FIFO
		<31:24>	Not used

Table 4-15– TARG_9_16_CREDIT_TIMEOUT register structure

Name	Address	Bits	Description
INIT_1_8_CREDIT_TIMEOUT	Base+0x38	<2:0>	Specifies the credits timeout for initiator FIFO #1 "000" : 4 cycles "001" : 8 cycles "010" : 16 cycles "011" : 32 cycles "100" : 64 cycles "101" : 128 cycles "110" : 256 cycles

			“111“ : 512 cycles
		<5:3>	Specifies the credits timeout for initiator FIFO #2
		<8:6>	Specifies the credits timeout for initiator FIFO #3
		<11:9>	Specifies the credits timeout for initiator FIFO #4
		<14:12>	Specifies the credits timeout for initiator FIFO #5
		<17:15>	Specifies the credits timeout for initiator FIFO #6
		<20:18>	Specifies the credits timeout for initiator FIFO #7
		<23:21>	Specifies the credits timeout for initiator FIFO #8
		<31:24>	Not used

Table 4-16– INIT_1_8_CREDIT_TIMEOUT register structure

Name	Address	Bits	Description
INIT_9_16_CREDIT_TIMEOUT	Base+0x3C	<2:0>	Specifies the credits timeout for initiator FIFO #9 “000” : 4 cycles “001” : 8 cycles “010” : 16 cycles “011” : 32 cycles “100” : 64 cycles “101” : 128 cycles “110“ : 256 cycles “111“ : 512 cycles
		<5:3>	Specifies the credits timeout for initiator FIFO #10
		<8:6>	Specifies the credits timeout for initiator FIFO #11
		<11:9>	Specifies the credits timeout for initiator FIFO #12
		<14:12>	Specifies the credits timeout for initiator FIFO #13
		<17:15>	Specifies the credits timeout for initiator FIFO #14
		<20:18>	Specifies the credits timeout for initiator FIFO #15

		<23:21>	Specifies the credits timeout for initiator FIFO #16
		<31:24>	Not used

Table 4-17– INIT_9_16_CREDIT_TIMEOUT register structure

Name	Address	Bits	Description
TARG_1_8_RX_FIFO_PRI	Base+0x40	<3:0>	Specifies the priority of target #1 FIFO for flow control arbiter (0 : lowest priority, 15 : highest priority)
		<7:4>	Specifies the priority of target #2 FIFO for flow control arbiter
		<11:8>	Specifies the priority of target #3 FIFO for flow control arbiter
		<15:12>	Specifies the priority of target #4 FIFO for flow control arbiter
		<19:16>	Specifies the priority of target #5 FIFO for flow control arbiter
		<23:20>	Specifies the priority of target #6 FIFO for flow control arbiter
		<27:24>	Specifies the priority of target #7 FIFO for flow control arbiter
		<31:28>	Specifies the priority of target #8 FIFO for flow control arbiter

Table 4-18–TARG_1_8_ PRI register structure

Name	Address	Bits	Description
TARG_9_16_RX_FIFO_PRI	Base+0x44	<3:0>	Specifies the priority of target #9 FIFO for flow control arbiter (0 : lowest priority, 15 : highest priority)
		<7:4>	Specifies the priority of target #10 FIFO for flow control arbiter
		<11:8>	Specifies the priority of target #11 FIFO for flow control arbiter
		<15:12>	Specifies the priority of target #12 FIFO for flow control arbiter
		<19:16>	Specifies the priority of target #13 FIFO for flow control arbiter
		<23:20>	Specifies the priority of target #14 FIFO for flow control arbiter
		<27:24>	Specifies the priority of target #15 FIFO for flow control arbiter
		<31:28>	Specifies the priority of target #16 FIFO for flow control arbiter

Table 4-19–TARG_9_16_ PRI register structure

Name	Address	Bits	Description
INIT_1_8_RX_FIFO_PRI	Base+0x48	<3:0>	Specifies the priority of target #1 FIFO for flow control arbiter (0 : lowest priority, 15 : highest priority)
		<7:4>	Specifies the priority of target #2 FIFO for flow control arbiter
		<11:8>	Specifies the priority of target #3 FIFO for flow control arbiter
		<15:12>	Specifies the priority of target #4 FIFO for flow control arbiter
		<19:16>	Specifies the priority of target #5 FIFO for flow control arbiter
		<23:20>	Specifies the priority of target #6 FIFO for flow control arbiter
		<27:24>	Specifies the priority of target #7 FIFO for flow control arbiter
		<31:28>	Specifies the priority of target #8 FIFO for flow control arbiter

Table 4-20–INIT_1_8_ PRI register structure

Name	Address	Bits	Description
INIT_9_16_RX_FIFO_PRI	Base+0x4C	<3:0>	Specifies the priority of target #9 FIFO for flow control arbiter (0 : lowest priority, 15 : highest priority)
		<7:4>	Specifies the priority of target #10 FIFO for flow control arbiter
		<11:8>	Specifies the priority of target #11 FIFO for flow control arbiter
		<15:12>	Specifies the priority of target #12 FIFO for flow control arbiter
		<19:16>	Specifies the priority of target #13 FIFO for flow control arbiter
		<23:20>	Specifies the priority of target #14 FIFO for flow control arbiter
		<27:24>	Specifies the priority of target #15 FIFO for flow control arbiter
		<31:28>	Specifies the priority of target #16 FIFO for flow control arbiter

Table 4-21–INIT_9_16_ PRI register structure

Name	Address	Bits	Description
QOS	Base+0x50	<0>	Specifies if LRA arbitration scheme is used '0' => Bandwidth limiters are used '1' => LRA scheme is used
		<31:1>	Reserved

Table 4-22– QOS register structure

Name	Address	Bits	Description
BUNDLE_SIZE	Base+0x54	<5:0>	Size of virtual wires bundle #1 "000000" : 0 (no wires) "000001" : 4 wires "000010" : 5 wires "000011" : 11 wires "000100" : 18 wires "000101" : 20 wires "000110" : 25 wires "000111" : 32 wires "001000" : 35 wires "001001" : 39 wires "001010" : 46 wires "001011" : 50 wires "001100" : 53 wires "001101" : 60 wires "001110" : 65 wires "001111" : 67 wires "010000" : 74 wires "010001" : 80 wires "111111" : all the existing wires (bundle_1 size)
		<11:6>	Size of virtual wires bundle #2
		<17:12>	Size of virtual wires bundle #3
		<23:18>	Size of virtual wires bundle #4
		<29:24>	Size of virtual wires bundle #5
		<31:30>	Not used

Table 4-23– BUNDLE_SIZE register structure

Name	Address	Bits	Description
ADCKEN	Base+0x58	<5:0>	Specifies how many clock cycles (1 to 63) have to elaps in the PHY clock domain after the last phyt has been sent by the PHY adapater before issuing the command to deassert the PHY clock. When 0 it means Activity Driven Clock Gating is not enabled.
		<31:6>	Not used

Table 4-24– ADCKEN register structure

Name	Address	Bits	Description
BI_TX_BYPASS	Base+0x5C	<0>	Specifies whether the BI transmitter has to be bypassed for debugging reasons or not 0 : BI Tx is active 1 : BI Tx is bypassed
		<31:1>	Not used

Table 4-25– BI_TX_BYPASS register structure

Name	Address	Bits	Description
TARG_FIFO_SAF	Base+0x60	<0>	Enables store and forward policy for target FIFO #1 0 : store and forward policy inactive 1 : store and forward policy active
		<1>	Enables store and forward policy for target FIFO #2
		<i>	Enables store and forward policy for target FIFO #i (2 < i < 15)
		<15>	Enables store and forward policy for target FIFO #16
		<31:16>	Not used

Table 4-26– ADCKEN register structure

Name	Address	Bits	Description
WIRES_SAM_RATE	Base+0x64	<5:0>	Virtual wires bundle 1 sample rate 0 : every cycle 1 : every 2 cycles 2 : every 4 cycles

			3 : every 8 cycles . . . 10 : every 1024 cycles 11 - 63 : reserved
		<11:6>	Virtual wires bundle 2 sample rate
		<17:12>	Virtual wires bundle 3 sample rate
		<23:18>	Virtual wires bundle 4 sample rate
		<29:24>	Virtual wires bundle 5 sample rate
		<31:30>	Not used

Table 4-27– WIRES_SAM_RATE register structure

Name	Address	Bits	Description
DDCM_PHY_FREQ_RATIO	Base+0x68	<0>	Specifies the frequency ratio between DDCM clock and PHY clock 0 : f(DDCM) < f(PHY) 266 MHz vs 400/450 MHz 300 MHz vs 400/450 MHz 333 MHz vs 400/450 MHz 400 MHz vs 450 MHz 1 : f(DDCM) >= f(PHY) 450 MHz vs 400/450 MHz 400 MHz vs 400 MHz
		<31:1>	Not used

Table 4-28– DDCM_PHY_FREQ_RATIO register structure

Name	Address	Bits	Description
IPORT_1_BWL	Base+0x6C	<0>	Enables bandwidth limiter
		<4:1>	Low priority (when the initiator has to be limited)
		<12:5>	Time window where the bandwidth has to be consumed
		<16:13>	Fixed value for segment counter decrease
		<20:17>	Thresholds (expressed in number of DDCM segments)
		<28:21>	Maximum segment counter value
		<31:29>	Not used

Table 4-29– IPORT_1_BWL register structure

Name	Address	Bits	Description
IPORT_16_BWL	Base+0xA8	<0>	Enables bandwidth limiter
		<4:1>	Low priority (when the initiator has to be limited)
		<12:5>	Time window where the bandwidth has to be consumed
		<16:13>	Fixed value for segment counter decrease
		<20:17>	Thresholds (expressed in number of DDCM segments)
		<28:21>	Maximum segment counter value
		<31:29>	Not used

Table 4-30– IPORT_16_BWL register structure

Name	Address	Bits	Description
PHY_DEBUG_MODE	Base+0xAC	<1:0>	Specify the PHY data source in debug mode "00" : DDCM "01" : reserved "10" : pattern generator "11" : loopback FIFO
		<31:2>	Not used

Table 4-31– PHY_DEBUG_MODE register structure

Name	Address	Bits	Description
INIT_1_8_TX_VC_PRI	Base+0xB0	<3:0>	Specifies the priority of initiator #1 VC for QoS arbiter (0 : lowest priority, 15 : highest priority)
		<7:4>	Specifies the priority of initiator #2 VC for QoS arbiter
		<11:8>	Specifies the priority of initiator #3 VC for QoS arbiter
		<15:12>	Specifies the priority of initiator #4 VC for QoS arbiter
		<19:16>	Specifies the priority of initiator #5 VC for QoS arbiter
		<23:20>	Specifies the priority of initiator #6 VC for QoS arbiter
		<27:24>	Specifies the priority of initiator #7 VC for QoS arbiter
		<31:28>	Specifies the priority of initiator #8 VC for QoS arbiter

Table 4-32–INIT_1_8_ TX_VC_PRI register structure

Name	Address	Bits	Description
INIT_9_16_TX_VC_PRI	Base+0xB4	<3:0>	Specifies the priority of initiator #9 VC for QoS arbiter (0 : lowest priority, 15 : highest priority)
		<7:4>	Specifies the priority of initiator #10 VC for QoS arbiter
		<11:8>	Specifies the priority of initiator #11 VC for QoS arbiter
		<15:12>	Specifies the priority of initiator #12 VC for QoS arbiter
		<19:16>	Specifies the priority of initiator #13 VC for QoS arbiter
		<23:20>	Specifies the priority of initiator #14 VC for QoS arbiter
		<27:24>	Specifies the priority of initiator #15 VC for QoS arbiter
		<31:28>	Specifies the priority of initiator #16 VC for QoS arbiter

Table 4-33–INIT_9_16_ TX_VC_PRI register structure

Name	Address	Bits	Description
TARG_1_8_TX_VC_PRI	Base+0xB8	<3:0>	Specifies the priority of target #1 VC for QoS arbiter (0 : lowest priority, 15 : highest priority)
		<7:4>	Specifies the priority of target #2 VC for QoS arbiter
		<11:8>	Specifies the priority of target #3 VC for QoS arbiter
		<15:12>	Specifies the priority of target #4 VC for QoS arbiter
		<19:16>	Specifies the priority of target #5 VC for QoS arbiter
		<23:20>	Specifies the priority of target #6 VC for QoS arbiter
		<27:24>	Specifies the priority of target #7 VC for QoS arbiter
		<31:28>	Specifies the priority of target #8 VC for QoS arbiter

Table 4-34–TARG_1_8_ TX_VC_PRI register structure

Name	Address	Bits	Description
TARG_9_16_TX_VC_PRI	Base+0xBC	<3:0>	Specifies the priority of target #9 VC for QoS arbiter (0 : lowest priority, 15 : highest priority)

		<7:4>	Specifies the priority of target #10 VC for QoS arbiter
		<11:8>	Specifies the priority of target #11 VC for QoS arbiter
		<15:12>	Specifies the priority of target #12 VC for QoS arbiter
		<19:16>	Specifies the priority of target #13 VC for QoS arbiter
		<23:20>	Specifies the priority of target #14 VC for QoS arbiter
		<27:24>	Specifies the priority of target #15 VC for QoS arbiter
		<31:28>	Specifies the priority of target #16 VC for QoS arbiter

Table 4-35–TARG_9_16_ TX_VC_PRI register structure

Registers access path

This subsection highlights the path followed by programming traffic to access registers of the different DDCM modules of a SiP.

Two different contexts can be individuated:

- the registers to be programmed are within the DDCM module in the same die where the CPU is;
- the registers to be programmed are within the DDCM Module in the other die, where there is no CPU.

Figure 4.2 shows the two different situations.

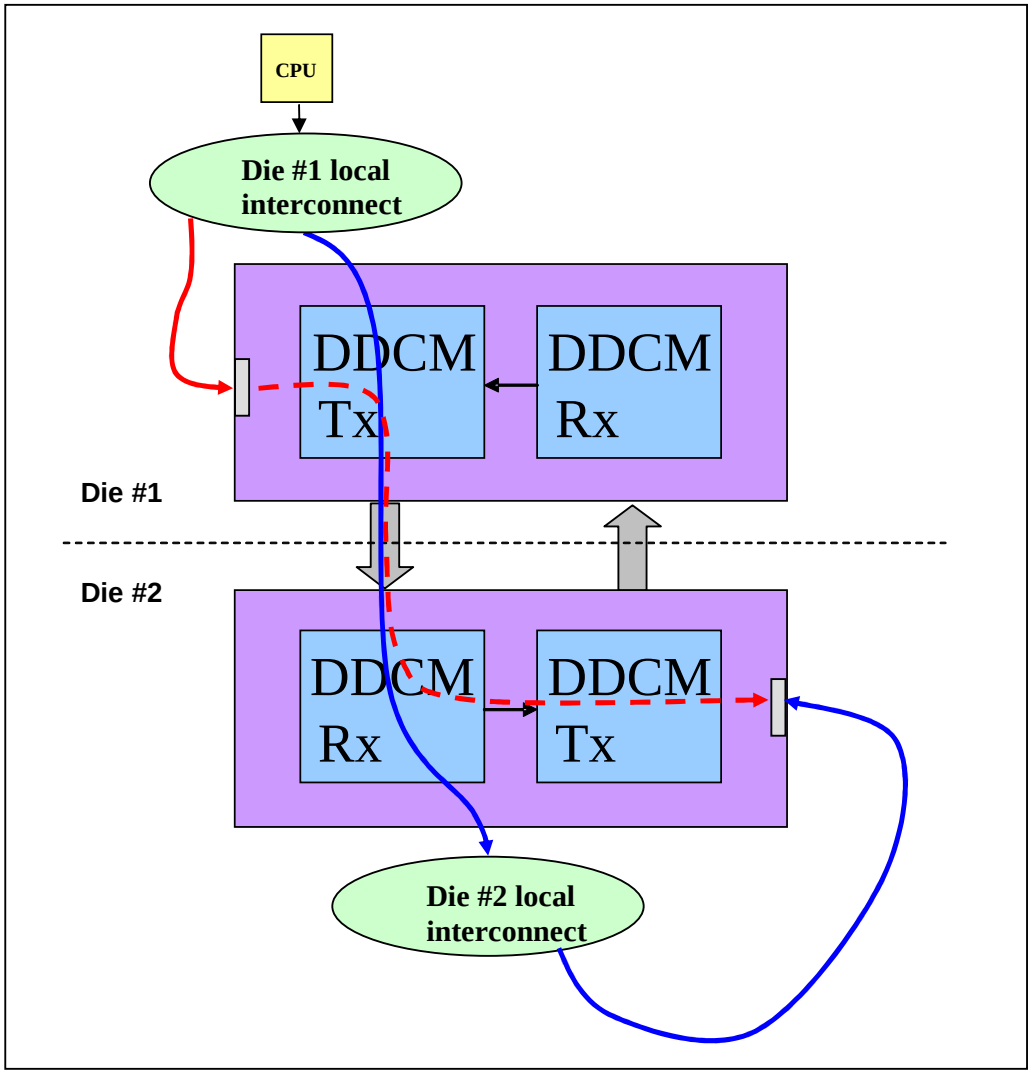


Figure 4-2: DDCM registers access paths

If registers to be programmed are in the same die of the CPU, the CPU programming traffic crosses the local interconnect, through the type 1 peripheral subsystem, and reaches the DDCM type 1 programming port (continuous red arrow). Such a port implements full STBus type 1 protocol, including support for byteenables and 1/2/4/8 bytes operations.

If registers to be programmed are in the other die, the CPU traffic crosses the DDCM module of the first die, reaches the DDCM module of the second die, from which it's routed to the local interconnect of the second die, and after crossing the local peripheral subsystem, it reaches the type 1 port of the DDCM module in the second die (blue arrow).

Registers in the second die could be accessed also through an internal path, i.e. CPU reaches the programming port of the DDCM module in die #1, then if registers addresses are related to registers of the DDCM module in die #2, this can be detected internally and registers configuration commands can be sent directly to the DDCM module in die #2 (dashed red arrow). This second programming option has been deeply evaluated and because of its complexity it will not be implemented.

Notice that the programming logic will implement a mechanism allowing to program DDCM registers in a safe way, meaning that the actual writing of a registers will be prevented if there are transactions in progress across the DDCM, and the programming of the register can impact the safe completion of the operations in progress. Typical registers that can lead to such an issue are the ones containing the threshold values for the credit-based flow control.

5 Architecture

As shown in figure 5.1, the DDCM top level in each die consists of a transmitter (DDCM Tx) and a receiver (DDCM Rx).
In such a figure it's possible to see the two information flows supported by a complete DDCM architecture, i.e.

- requests from STNoC/STBus/AMBA-AXI initiators in chip 1 to STNoC/STBus/AMBA-AXI targets in chip 2, responses from STNoC/STBus/AMBA-AXI targets in chip 2 to STNoC/STBus/AMBA-AXI initiators in chip 1, virtual wires from chip 1 to chip 2 (continuous lines);
- requests from STNoC/STBus/AMBA-AXI initiators in chip 2 to STNoC/STBus/AMBA-AXI targets in chip 1, responses from STNoC/STBus/AMBA-AXI targets in chip 1 to STNoC/STBus/AMBA-AXI initiators in chip 2, virtual wires from chip 2 to chip 1 (dotted lines).

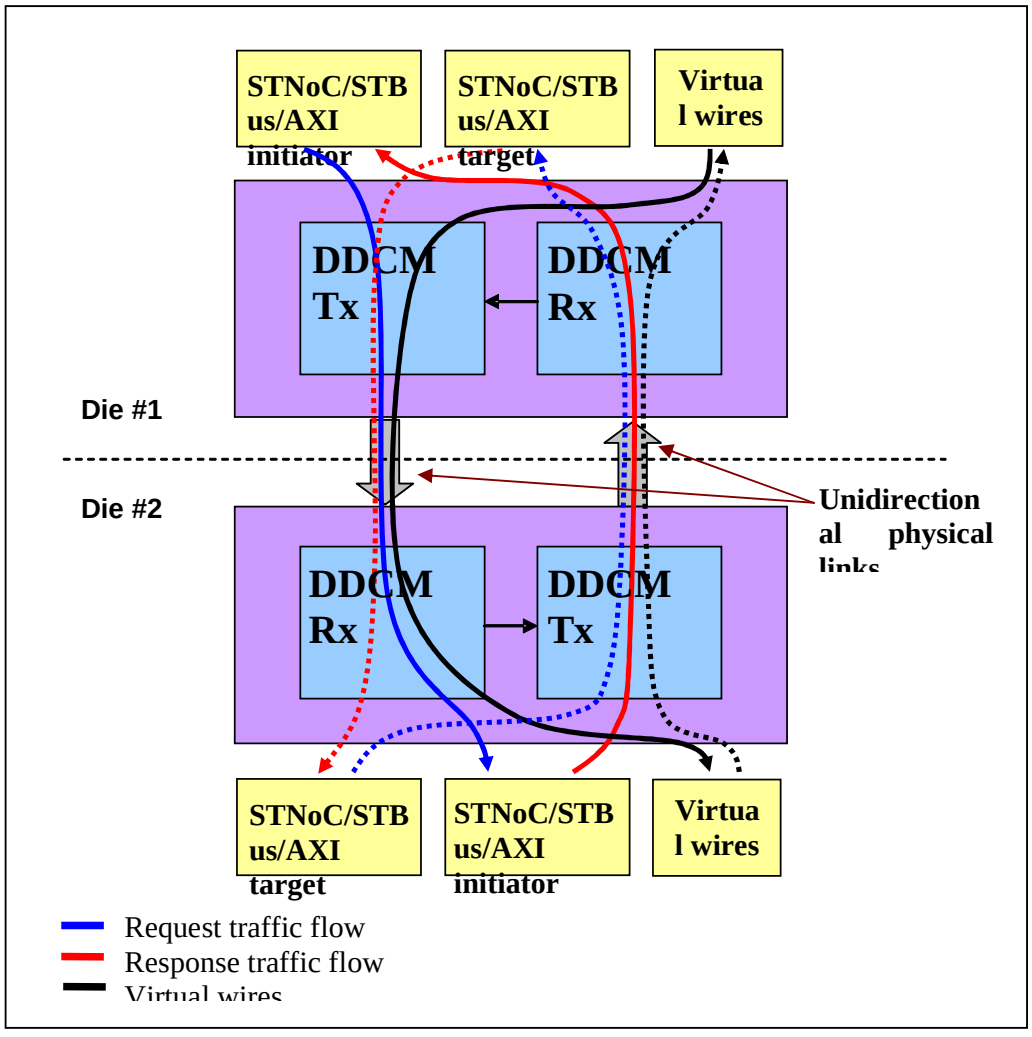


Figure 0-1: DDCM top level architecture and information flow

The DDCM transmitter (DDCM Tx) is responsible for

- receiving requests from STNoC/STBus/AMBA-AXI initiators in the same die and sending them to STNoC/STBus/AMBA-AXI targets in the other die;

- receiving responses from STNoC/STBus/AMBA-AXI targets in the same die and sending them to STNoC/STBus/AMBA-AXI initiators in the other die;
- sampling ancillary signals (virtual wires) generated in the same die at a specified rate and sending samples to the other die.

The DDCM receiver (DDCM Rx) is responsible for

- receiving requests from STNoC/STBus/AMBA-AXI initiators in the other die and sending them to STNoC/STBus/AMBA-AXI targets in the same die;
- receiving responses from STNoC/STBus/AMBA-AXI targets in the other die and sending them to STNoC/STBus/AMBA-AXI initiators in the same die;
- receiving ancillary signals samples generated in the other die and sending them to the proper destination in the same die.

Figure 5-2 shows a full architectural view of an DDCM, highlighting the separation between an DDCM transmitter and an DDCM receiver.

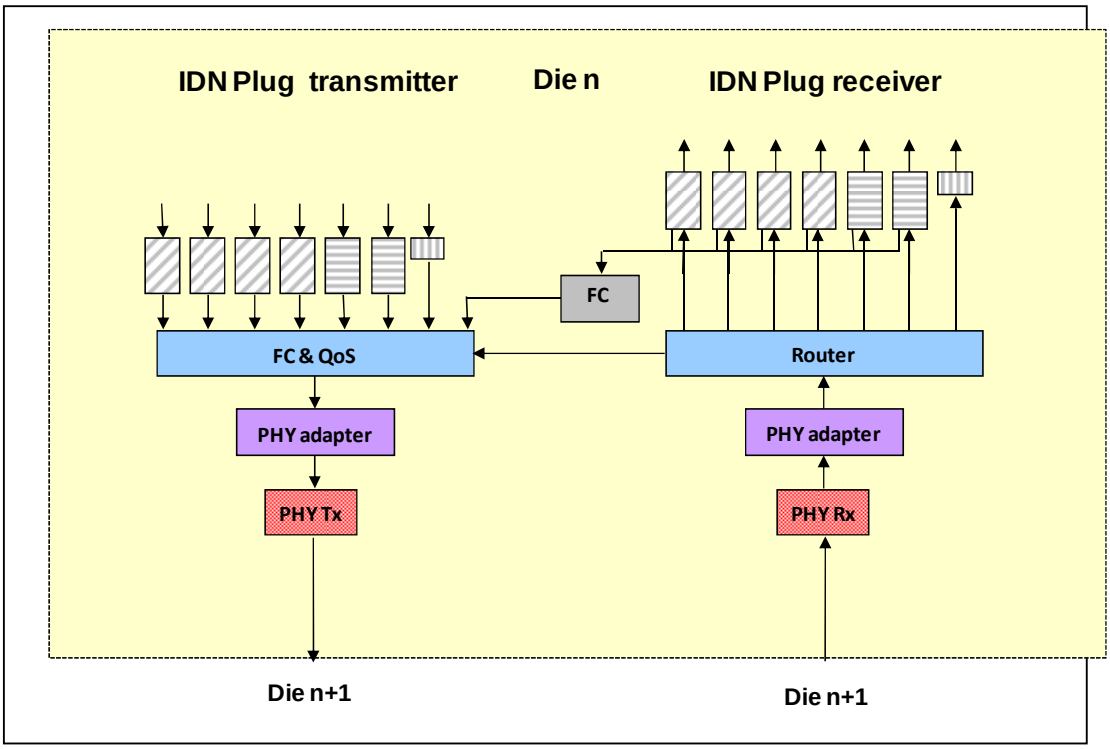


Figure 0-2: DDCM detailed architecture

Figure 5-3 shows the architecture of the DDCM highlighting the connections with initiators and targets across an STNoC interconnect. In this picture it's possible to see clearly how request and response traffic streams flow.

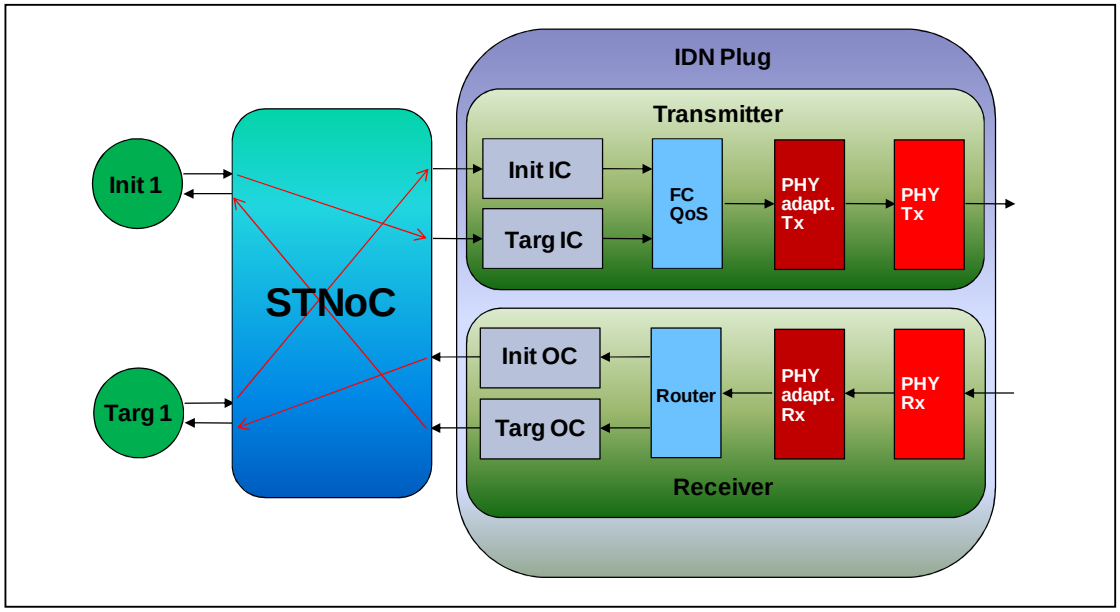


Figure 0-3: DDCM detailed architecture highlighting traffic streams flows

Figure 5-4 shows the connection and the traffic streams flows between two dice, highlighting the two DDCMs architectures and their crossing. Specifically, the orange line represents the request traffic stream flowring from initiator 1 in die #1 towards target 2 in die #2, while the yellow line represents the response traffic stream flowing from target 2 in die #2 towards initiator 1 in die #1.

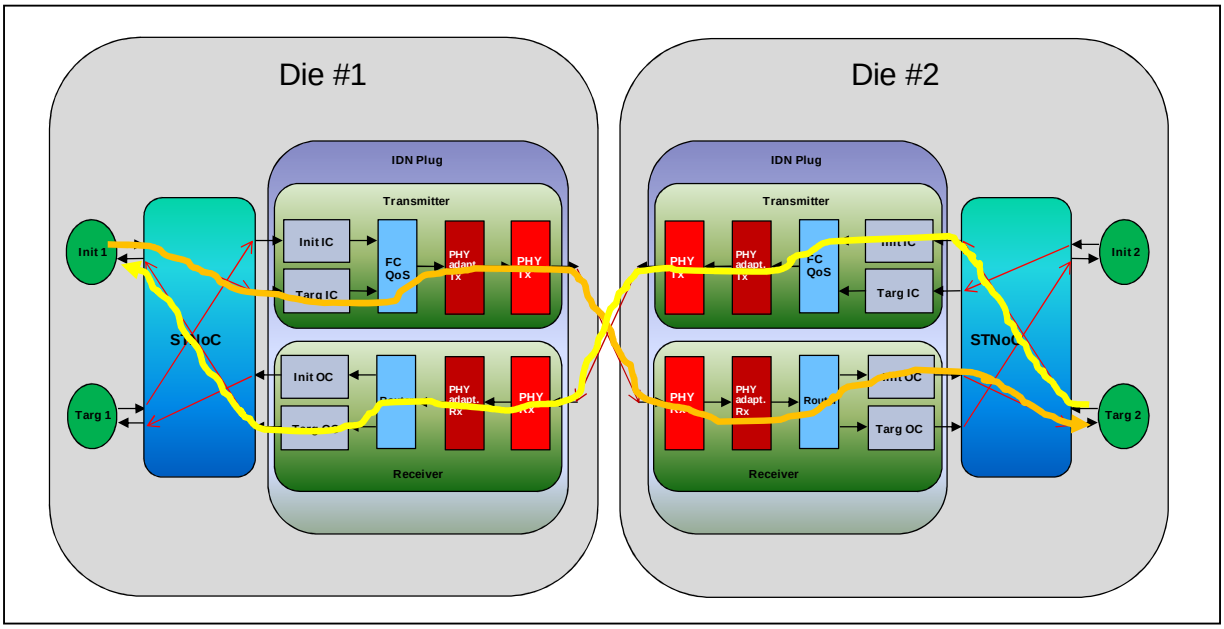


Figure 0-4 : Traffic streams flows between two dice

Next section describes in detail all the DDCM building-blocks.

6 Building-blocks

In this section all the DDCM building-blocks are described.

Transmitter

The DDCM transmitter performs the following functions:

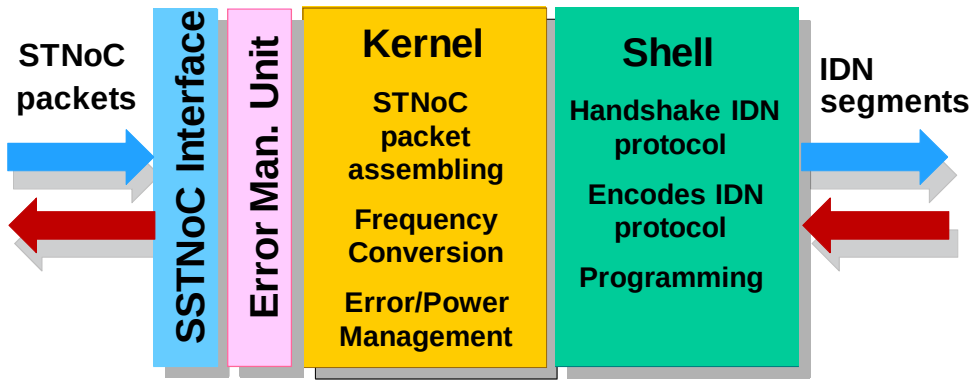
- Buffering of STNoC, STBus, AMBA AXI traffic;
- Sampling of virtual wires
- STNoC, STBus, AMBA AXI traffic size conversion when required
- Frequency conversion when required
- STNoC, STBus, AMBA AXI traffic and virtual wires encapsulation within IDN segments
- Credit-based flow control
- IDN segments QoS management
- IDN segments serialization
- Phyts encryption when enabled
- Phyts transmission in SCE or DCE mode

Request Input Channel

The Request Input Channel (ReqIC) deals with STNoC request traffic generated either by an STNoC upstream interface, or by an STBus or an AMBA AXI initiator Network Interface.

It is divided in three main parts:

- **Kernel**, responsible for buffering the incoming STNoC request traffic and performing flit size conversion when required;
- **FIFOs** (header FIFO and payload FIFO), where STNoC request information is stored and performing frequency conversion when required;
- **Shell**, responsible for encapsulating the STNoC requests into IDN segments by generating a proper IDN header.



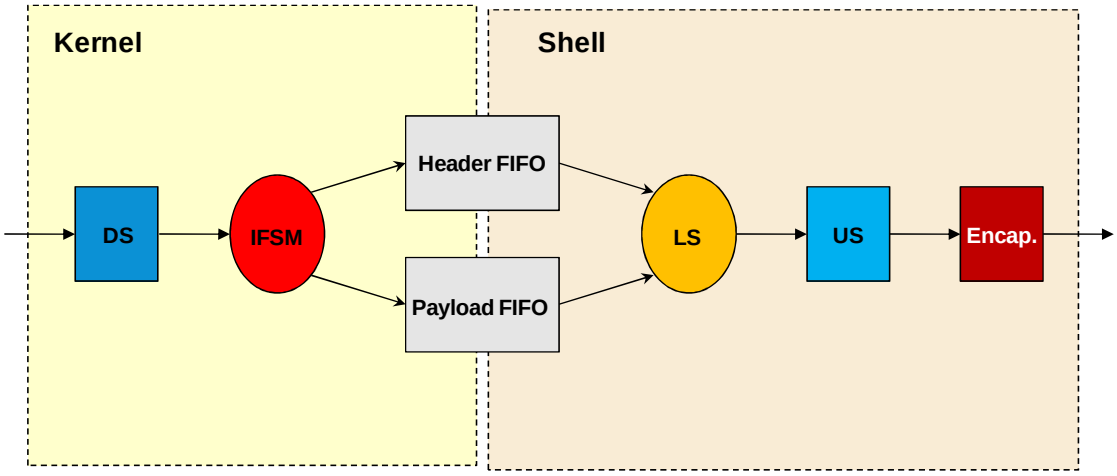
0-1: Input channel generic structure

The IC kernel in turn is composed of the following building-blocks:

- **downstream interface** (**DS**) responsible for collecting STNoC flits and auxiliary signals from STNoC interface;
- **input FSM** (**IFSM**) responsible for discriminating between header and payload flits and storing them in the respective FIFOs.

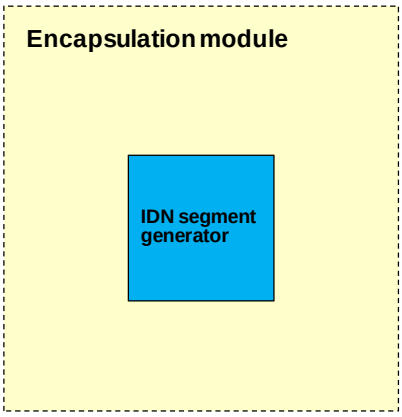
The IC shell in turn is composed of the following building-blocks:

- **link scheduler** (**LS**) responsible for reading header or payload FIFO depending on incoming traffic shape;
- **upstream interface** (**US**) responsible for propagating the proper flit and its associated signals;
- **encapsulation module** (**Encap**) responsible for generating the IDN header and encapsulating the STNoC flits and auxiliary signals within IDN segments; here, since the network layer header is related to the local network topology, only the STNoC transport layer header is encapsulated and propagated across the physical channel, while the network layer header is cut.



0-2: Input channel micro-architecture

The encapsulation module of the Request Input Channel, dealing with STNoC requests, has the function of generating the IDN header and to add it to the STNoC flit and auxiliary signals, so to build the IDN segment to be serialized and propagated across the physical channel.



0-3: Request IC encapsulation module function

Table 6.1 shows the IDN header structure.

Field name	Size	Bits	Description
IC ID	6	<5:0>	Input Channel identifier
Type	2	<7:6>	IDN segment type (STNoC, virtual wires, credit)
Segment ID	2	<9:8>	IDN segment identifier (first, last, intermediate)

Table 6.1 – IDN header structure

The meaning of the header fields is detailed in the following.

- **IC ID** is the identifier of the input channel where the information to be transmitted (both STNoC transactions and virtual wires) comes from; marking segments with the IC ID is key for allowing segments interleaving. If the segment is related to virtual wires (type = "01"), bits <2:0> of the IC ID field represents the number of phyts required to transport the virtual wires information ("---000" = 1 phyt, "---001" = 2 phyts, "---010" = 3 phyts, "---011" = 4 phyts, "---100" = 5 phyts, "---101" = 6 phyts), while bit <5> tells whether the transmitted bundle is the fifth one.
- **Type** allows the DDCM receiver to understand if the segment belongs to an STNoC transaction ("00") or to virtual wires ("01"), in which case it is forwarded to the corresponding OC, or if it carries credit information ("10"), in which case it is sent to the associated DDCM transmitter for computing the new credits value.
- **Segment ID** specifies if the segment is the first ("01"), the last ("10") or an intermediate one ("00") for the transmitted transaction; this information is important for the correct reconstruction of the transaction at destination. If the segment is related to virtual wires (type = "01"), the segment ID field assumes the meaning of the virtual wires bundle identifier ("00" = bundle 0, "01" = bundle 1, "10" = bundle 2, "11" = bundle 3). If the segment is related to credits information, the segment ID field represents the number of phyts required to transport the credit information ("00" = 1 phyt, "01" = 2 phyts, "10" = 3 phyts, "11" = 4 phyts).

Response Input Channel

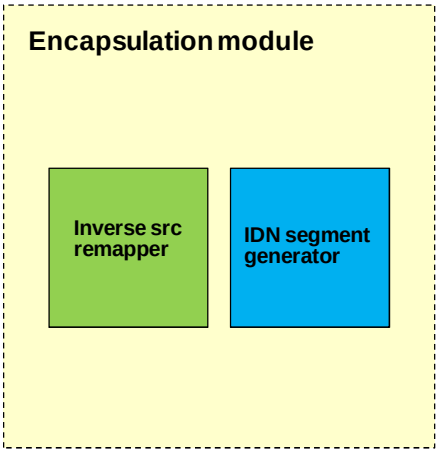
The Response Input Channel (ResIC) deals with STNoC response traffic generated either by an STNoC upstream interface, or by an STBus or an AMBA AXI target Network Interface.

It is divided in three main parts:

- **Kernel**, responsible for buffering the incoming STNoC response traffic and performing flit size conversion when required;
- **FIFOs** (header FIFO and payload FIFO), where STNoC response information is stored and performing frequency conversion when required;
- **Shell**, responsible for encapsulating the STNoC responses into IDN segments by generating a proper IDN header.

Kernel and Shell structure of the ResIC are the same of the ReqIC.

The encapsulation module of the Response Input Channel, dealing with STNoC responses, has the function of performing the inverse src remapping, generating the IDN header and to add it to the STNoC flit and auxiliary signals, so to build the IDN segment to be serialized and propagated across the physical channel.



0-4: Response IC encapsulation module function

Virtual Wires Input Channel

The Virtual Wires Input Channel (VWIC) deals with asynchronous signals not following any standard protocol, such as interrupts, power down handshake, etc.

The Virtual Wires IC interface can be up to 400 bits wide and is organized as a set of 5 bundles, each up to 80-bits wide. However for the input port it's possible to specify how many wires out of the existing ones are meaningful, through a dedicated register; this possibility allows to reuse the same DDCM VWIC block in different systems, where the number of virtual wires is different.

In order to transmit virtual wires information virtual wires bundles are sampled periodically, at a rate specified in the related configuration register, and the sampled values are stored into the proper section of the VWIC, whose elements are also up to 400 bits wide and are split into 5 bundles up to 80-bits wide, in order to be transmitted across the die-to-die channel as a set of 5 segments; each bundle is marked by a proper identifier to allow the correct reconstruction of virtual wires information at destination.

Notice that, due to their intrinsic asynchronous nature, virtual wires are properly synchronized in DDCM clock domain by a proper number of synchronization FFs.

In order to avoid to transmit twice or more the same information, if two back-to-back virtual wires bundles samples are equal, the second one is not transmitted again, since this means no new events to be transmitted have occurred. According to that, when the sampling rate is chosen equal to the DDCM clock frequency, the transmission of virtual wires information follows actually a on-event approach, i.e. as soon as at least one wire changes its state from '0' to '1' the port configuration is stored into the FIFO.

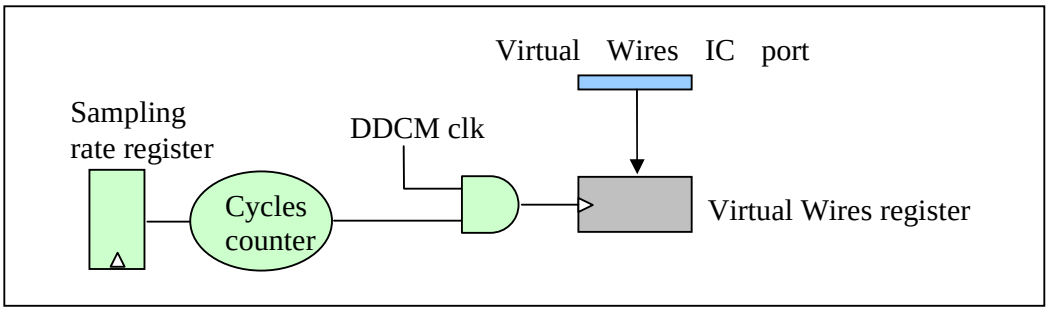


Figure 0-5 –Virtual Wires IC portbundle sampling

If programmed bundles sampling rates are such that more bundles have to be transmitted simultaneously, a proper arbitration is performed in order to select the bundle that can be transmitted, the others waiting for their turn, as

shown in figure 7-6. Bundles priorities are simply determined by bundle index, i.e. bundle 1 has the highest priority, bundle 5 has the lowest.

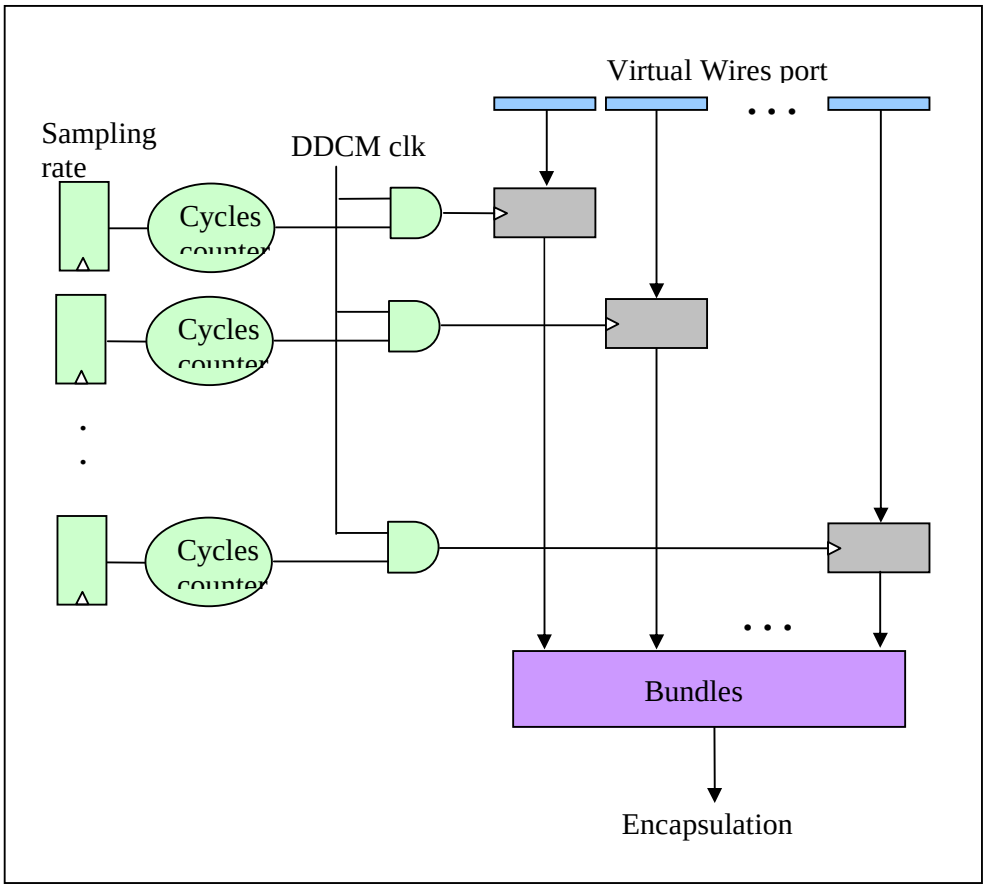


Figure 0-6 - Virtual wires bundles arbitration

The number of phyts required to transmit the virtual wires bundle information is transported in the IC ID field of the IDN segment header; this is required because, while all the STNoC segments have a fixed size, the virtual wires bundles segments have a different size, linked to the number of bits used in each bundle. With this approach the same information can be transported in case of physical channels of different width.

Finally, it's important to highlight that only **level signals** are supported as virtual wires, while **pulses** are not supported, since they would be lost either if their period was lower than the virtual wires sampling period, or if, even having a period greater the sampling period, the related virtual wires bundles lost the arbitration for a time long enough to make the pulse to disappear.

Credits Input Channel

The Credits Input Channel (CIC) deals with the credit information coming from the DDCM receiver, related to the segment FIFOs of the OCs.

Flow Control and QoS

The Flow Control and QoS modules performs arbitration between the IDN segments generated by the different Input Channels, according to the selected QoS policy, taking into account the numer of credits available for each IC. If an IC has no credits available, it won't be arbitrated, so

that at the end the winner of the arbitration will be an IC sure to see its segment propagated to the other die across the physical channel.

Arbitration schemes

The QoS module supports three different arbitration schemes:

- Priority-based
- Priority-based with bandwidth limitation
- LRA (Less Recently Arrived)

The required arbitration scheme can be programmed via software properly setting the dedicated registers (see section 8).

Priority-based

The simplest arbitration scheme supported by the QoS module is based on priority, expressed through a 4 bits value. Provided that credits have always the highest priority, followed by virtual wires, for all the STNoC ICs it's possible to program their priorities so to follow a specific criterion for segment arbitration; STNoC ICs priorities are stored in dedicated DDCM registers and are propagated to the arbiter in the QoS module. Notice that in case of equal priority values, the winner of the arbitration will be determined according to a *positional* approach so as in STBus node arbiters.
Priority-based arbitration scheme is the default one in DDCM QoS module.

Priority-based with bandwidth limitation

With this arbitration scheme the initiators are arbitrated according to their priority, but when they consume the bandwidth programmed for them within a specific time window, their priority is lowered, so to allow other initiators normally having lower priorities to win the arbitration.
In order to enable bandwidth limitation in DDCM QoS module, bandwidth limiters have to be activated and configured via the proper registers.

LRA (Less Recently Arrived)

This arbitration scheme allows to take into account the time at which an initiator has issued its request, so to be priviledged in case of arbitration with other initiators issuing their requests later.

PHY Adapter

The PHY adapter transforms DDCM segments into a format suitable for being propagated across the physical channel; in particular this block is responsible for segment serialization for exploiting the narrower physical channel, and channel encoding for reducing dynamic power consumption. This second block can be bypassed by properly setting the input configuration pin or the correspondent register; such a bypass can be useful in case of system debugging when errors occur.
The output of the PHY adapter is the input of the PHY, responsible for actual transmission across the physical channel.
PHY adapter output is endowed with two 16-bits ports, named phyt_hi and phyt_low, representing two *phyts* (PHYsical uniTS) that can be delivered to the PHY in one clock cycle.
If the PHY works in dual clock edge (DCE) mode the PHY adapter has to deliver the PHY two phyt of 16 bits each, one to be transmitted during the clock rising edge and the other one to be transmitted during the clock falling edge; if the PHY works in single clock edge (SCE) mode, the PHY adapter sends only one phyt over the phyt_hi port.

The PHY adapter has also to deliver the PHY the clock enable signal; this signal is active high, i.e. when high the PHY transmitter clock is on (it must be on during the system boot), when low PHY transmitter clock is off.
Notice that, during the PHY test phase, the input of the PHY does not come from the PHY adapter anymore, but rather from external test sources, i.e. the pattern generator, according to the test interface described in table 6.2.

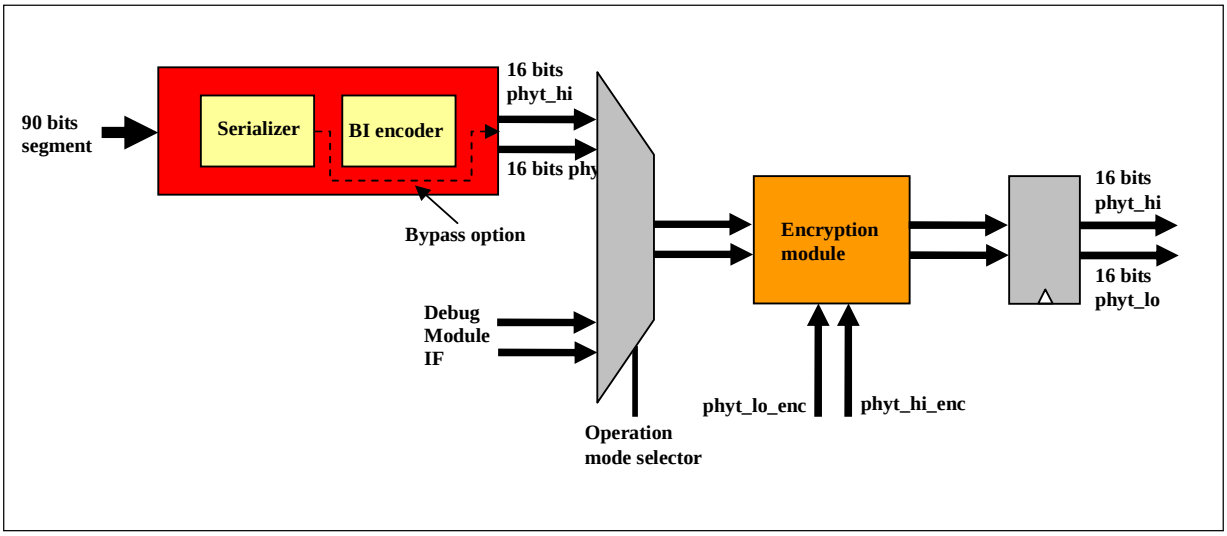


Figure 6-7 – DDCM layer A building-blocks

Serializer

This module has the task to split 90 bits STNoC segments generated by the DDCM modules into smaller units.

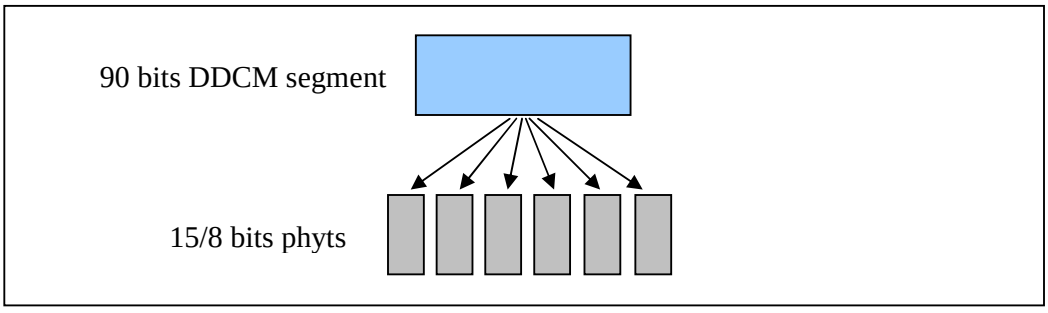


Figure 6-8 – Serializer function

Depending on the physical channel size (16 or 8 bits), the serializer will split an STNoC segment in 6 or 13 smaller units 15-bits or 7-bits wide, and will split the credit and virtual wires segments into the proper number of 15-bits or 7-bits wide units depending on the required number of phyts stored in the segment header, according to the variable segment size policy.
The following table shows the number of phyts generated depending on the incoming segment in case of 16-bits physical channel.

Segment	16 bits phyts
STNoC	6
Credit (< 32)	1
Credit (> 32, < 128)	2
Credit (= 128)	1
Virtual wires (< 4)	1
Virtual wires (>=6, < 21)	2
Virtual wires (<= 21, < 36)	3
Virtual wires (<= 36, < 51)	4
Virtual wires (<= 51, < 66)	5
Virtual wires (<= 66)	6

Table 6-1 – Number of transmitted phyts in case of 16-bits physical channel

In order to compensate the delay introduced by this serialization, the serializer output should run at a higher speed. In any case, whatever is the speed of the PHY, since the DDCM clock and the PHY clock have to be considered asynchronous, the serializer will take care of the frequency conversion, relying on the frequency bridge component.

Bus Inverter (BI) encoder

This module implements a source encoding technique allowing minimizing the Hamming distance between two consecutive phyts, so to reduce as much as possible the switching activity over the physical channel during the transmission of information from one chip to the other.

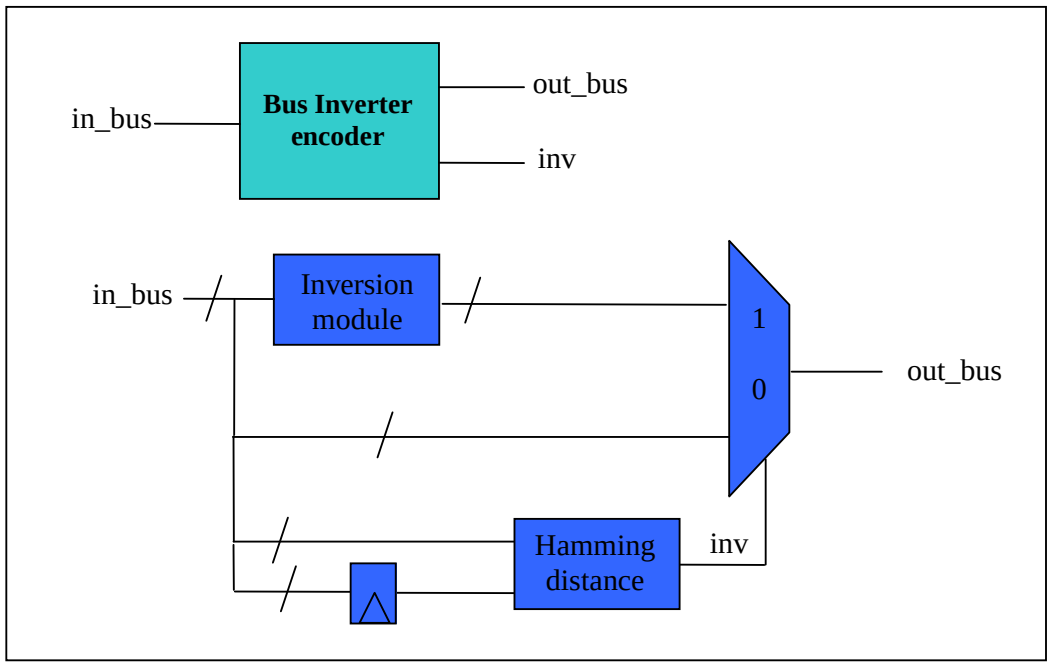


Figure 6-9 – Bus Inverter encoder

The operation of the BI encoder, according to the schematic depicted in figure 7-13, is the following:

- the phyt to be transmitted is inverted by a proper inversion module (inverted means that every '0' is replaced by a '1' and every '1' is replaced by a '0');
- a comparator determines whether the Hamming distance between the last transmitted phyt and the new phyt to be transmitted is greater than half phyt size;
- if yes, the inverted new phyt is transmitted; if not, the original new phyt is transmitted.

With this approach the new phyt to be transmitted will determine the minimum switching activity over the physical link.

Previous phyt	New phyt	H	Encoded phyt	inv	H	Gain
000000000000000	000000000000000	0	000000000000000	0	0	-
000000000000000	111111111111111	15	000000000000000	1	0	100%
111111111111111	111100000000000	11	000011111111111	1	4	64%
000111111111111	010101010101010	7	010101010101010	0	7	-
101010101010101	010101010101010	15	101010101010101	1	0	100%
101010101010101	111111110000000	8	000000001111111	0	7	12.5%
111111100000000	000011110000000	5	000011110000000	0	5	-
000111100000000	111111001100110	8	000000110011001	1	6	25%
000001100110011	111111110000110	10	000000001111001	1	5	50%

Table 6-2 – Examples of Bus Inverter encoding

The table above shows some examples of bus encoding related to a 15 bits bus; H is the Hamming distance between new and previous phyt, and the gain is expressed as the number of removed switching using the encoded new phyt, with respect to the number of original switching using the original new phyt.
Since the BI input data can have either 15 or just 7 meaningful bits, the generation of its output will change accordingly, depending on the value of the PHY_WIDTH register (see section 8). The following table shows the format of the encoded phyts generated by the BI encoder for the two possible sizes.

Phyt size	Bits	Field
16 bits	<0>	*inv* flag
	<1:15>	Phyt data
8 bits	<0>	*inv* flag
	<1:7>	Phyt data
	<8:15>	Not used

Table 6-3 – BI encoder outputs for different phyt sizes

Notice that independently on the phyt size, the *inv* flag is located always in bit 0 of the phyt.

Debug interface

As shown in figure 6-7, the output of the PHY adapter can be driven by an external debug module, for test reasons; according to that, the output of the BI transmitter is multiplexed with the debug input, under the control of a proper selector. The selector is the *tst_phy_sce_sel* input

signal, and allows to choose the PHY adapter output between the BI transmitter output and the *pattern generator* input.

Encryption module

Whatever is the output of the PHY adapter, it can be encrypted simply performing a XOR with the input key coming from the external security encoder module. A key exists for each phyt (hi/lo), and the same keys are used in the receiver for the proper decoding.

PHY

The PHY implements the DDCM transmitter physical layer, responsible for the transmission at physical level of the phyts across the physical channel.

Receiver

The DDCM receiver performs the following functions:

- phyts acquisition in SCE or DCE mode;
- phyts decryption when required;
- IDN segments assembly (deserialization);
- IDN segments routing;
- STNoC, STBus, AMBA AXI traffic and virtual wires reconstruction from IDN segments;
- frequency conversion when required;
- STNoC, STBus, AMBA AXI traffic size conversion when required;
- generation of STNoC, STBus, AMBA AXI traffic;
- generation of virtual wires traffic.
- credit information generation

PHY

The PHY implements the DDCM receiver physical layer, responsible for the acquisition at physical level of the phyts transmitted across the physical channel.
It transforms phyts received from the PHY into DDCM segments; in particular this block is responsible for channel decoding for reconstructing the actual data previously encoded for reducing dynamic power consumption, and DDCM segments assembly through deserialization. The channel decoder does not need the bypass option as its counterpart, since if BI transmitter is bypassed, the information will arrive always not encoded.
The input of the PHY adapter is the output of the PHY, responsible for actual transmission across the physical channel.
PHY adapter input is endowed with two 16-bits ports, named phyt_hi and phyt_low, representing two *phyts*) that can be delivered from the PHY in one clock cycle.
If the PHY works in dual clock edge (DCE) mode the PHY adapter has to take from the PHY two phyt of 16 bits each; if the PHY works in single clock edge (SCE) mode, the PHY adapter takes only one phyt from the phyt_hi port.
Both ports are retimed to remove timing constraints.

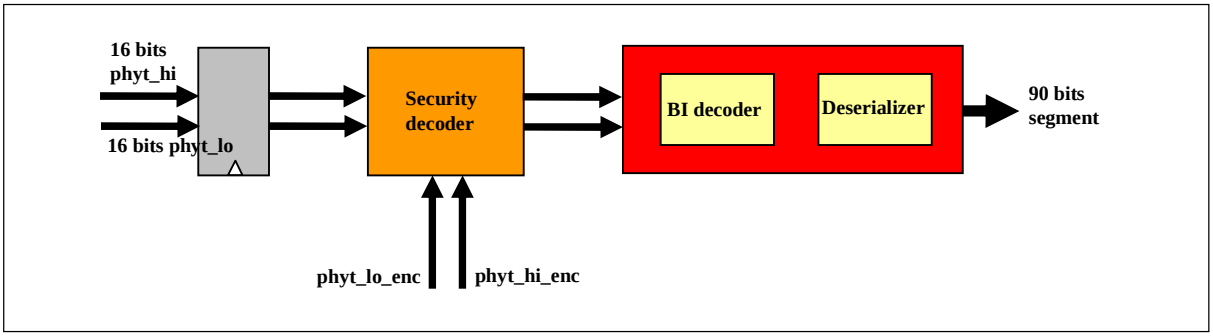


Figure 6-10 – DDCM layer A building-blocks

Decryption module

The input of the PHY adapter can be decrypted, if previously encrypted during transmission, simply performing a XOR with the input key coming from the external security decoder module. A key exists for each phyt (hi/lo), and the same keys ate used in the transmitter for the proper encoding.

Bus Inverter (BI) decoder

This module implements a source decoding technique allowing recovering the original phyt after the source encoding performed by the BI transmitter in order to minimize the Hamming distance between two phyts transmitted back to back.

The operation of the BI decoder, according to the schematic depicted in figure 7-16, is the following:

- the received phyt is inverted by a proper inversion module, following the same inversion algorithm described in when dealing with the BI transmitter;
- depending on the value of the inv signal, either the incoming phyt or the inverted one is propagated.

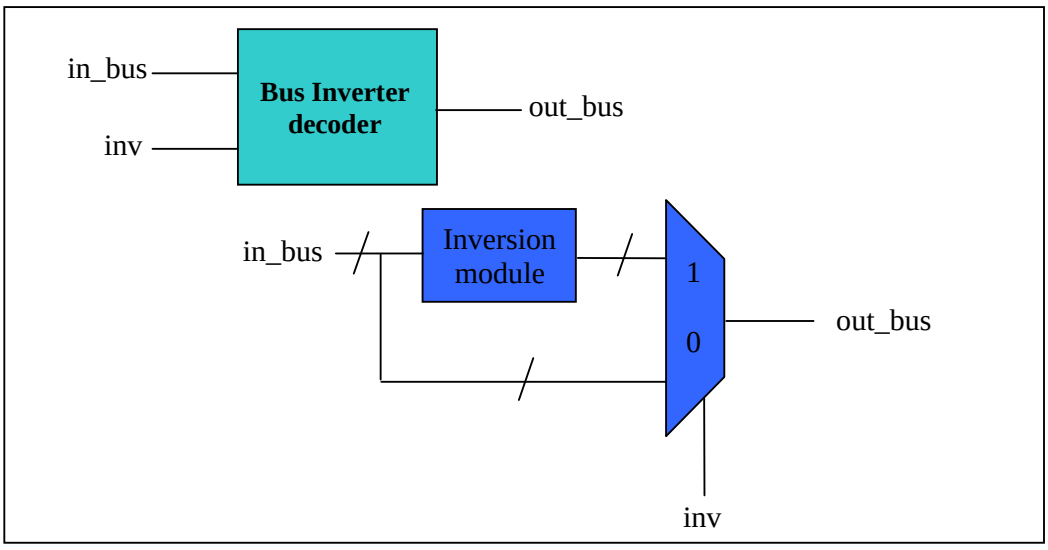


Figure 6-11 –Bus Inverter decoder

The following table shows some examples of bus decoding related to a 16-bits bus.

Bus value	inv	Decoding
000000000000000	0	000000000000000
000000000000000	1	111111111111111
000011111111111	1	111100000000000
010101010101010	0	010101010101010
010101010101010	1	101010101010101
111111110000000	0	111111110000000
000011110000000	0	000011110000000
000000110011001	1	111111001100110
000000001111001	1	111111110000110

Table 6-4 – Examples of Bus Inverter decoding

Since the BI decoder input data can have either 16 or just 8 meaningful bits, the generation of its output will change accordingly, and depending on the value of the PHY_WIDTH register (see section 8).
The BI decoder input format is equal to the BI encoder output format so as described in table 6-4.

Deserializer

This module has the task to build 90 bits segments starting from 6 15-bits wide units coming from the channel decoder.

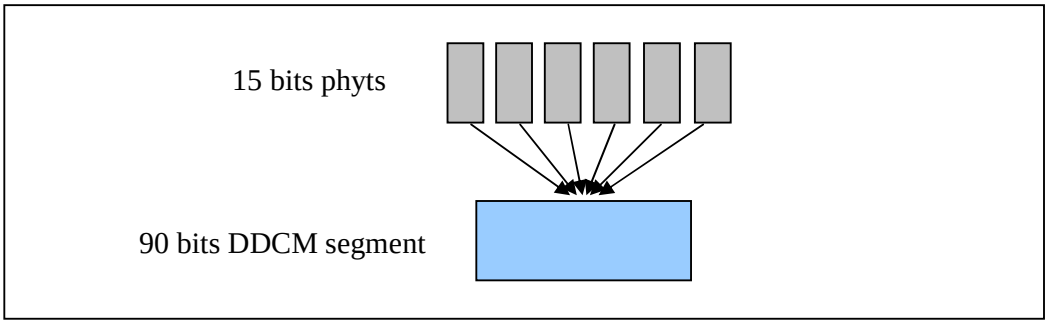


Figure 6-12 – Deserializer function

It's also possible to configure the PHY adapter so that the deserializer uses 13 7-bits wide units to build the 90 bits segment. This in the case in which only 8 bits of the physical channel are used, for example because the transmitter or the receiver has an 8-bits wide interface.
The deserializer gets data at the frequency of the PHY and generates data at the frequency of the DDCM controller, and it works correctly whatever is the frequency ratio between DDCM controller and PHY.

Router

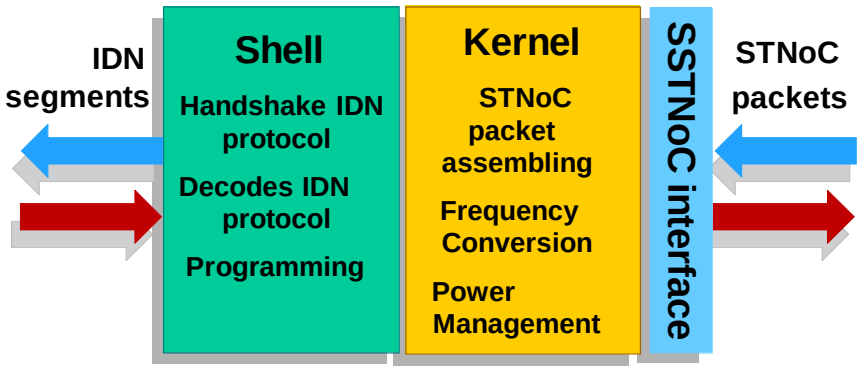
The router sends the re-generated IDN segment towards the proper Output Channel.

Request Output Channel

The Request Output Channel (ReqOC) generates STNoC request traffic either towards an STNoC dowstream interface, or towards an STBus or an AMBA AXI target Network Interface.

It is divided in three main parts:

- **Shell**, responsible for the reconstruction of the STNoC request traffic from the IDN segments regenerated by the PHY adapter.
- **FIFOs** (header FIFO and payload FIFO), where the reconstructed STNoC request information is stored and performing frequency conversion when required;
- **Kernel**, responsible for performing flit size conversion when required, performing the src-remapping and generating the local routing information according to the incoming address and the local (local to the die) network topology.



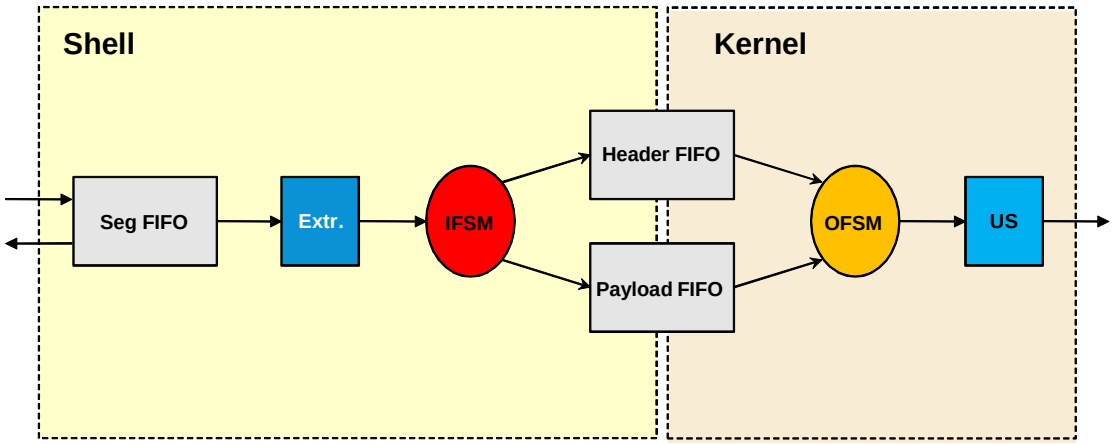
0-13: Output channel generic structure

The IC shell in turn is composed of the following building-blocks:

- **IDN segment FIFO** (**Seg FIFO**) responsible for storing IDN segments after the deserialization performed by the PHY adapter and generating the credit information to be sent to the other die;
- **extraction module** (**Extract**) responsible for removing the IDN header and re-generating the original STNoC flits and related auxiliary signals; at this point only the STNoC transport layer header is re-generated, while the network layer header, depending on the local network topology, is built from scratch relying on network structure awareness and some programming information (QoS, routing).
- **input FSM** (**IFSM**) responsible for discriminating between header and payload flits and storing them in the respective FIFOs.

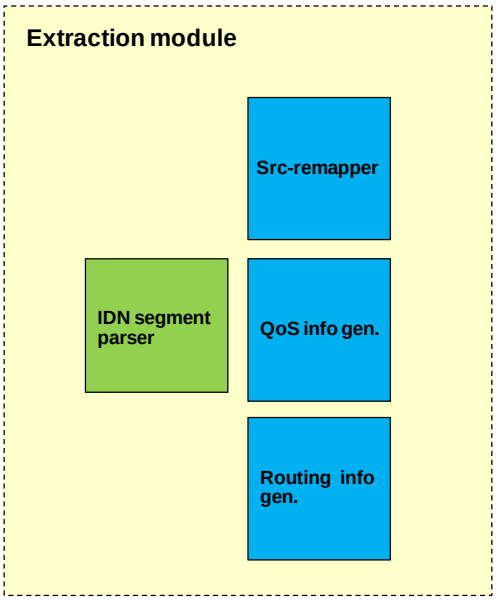
The IC kernel in turn is composed of the following building-blocks:

- **output FSM** (**IFSM**) responsible for reading header or payload FIFO depending on incoming traffic shape;
- **upstream interface** (**US**) responsible for propagating the proper flit and its associated signals.



0-14: Output channel micro-architecture

The extraction module of the Request Output Channel, dealing with STNoC requests, has the function of regenerating the STNoC flits and auxiliary signals, performing the src remapping, generating the QoS information and generating the routing informationaccording to the topology of the network of the local die.



0-15 Request OC encapsulation module function

Response Output Channel

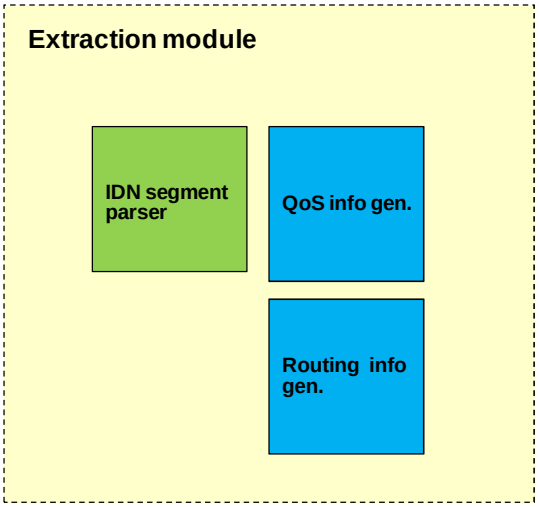
The Response Output Channel (ReqOC) generates STNoC response traffic either towards an STNoC dowstream interface, or towards an STBus or an AMBA AXI initiator Network Interface.

It is divided in three main parts:

- **Shell**, responsible for the reconstruction of the STNoC response traffic from the IDN segments regenerated by the PHY adapter.

- **FIFOs** (header FIFO and payload FIFO), where the reconstructed STNoC response information is stored and performing frequency conversion when required;
- **Kernel**, responsible for performing flit size conversion when required and generating the local routing information according to the incoming src and the local (local to the die) network topology.

The extraction module of the Response Output Channel, dealing with STNoC responses, has the function of regenerating the STNoC flits and auxiliary signals, generating the QoS information and generating the routing informationaccording to the topology of the network of the local die.



0-6 Response OC encapsulation module function

Virtual wires Output Channel

The Virtual Wires Output Channel (VWOC) generates asynchronous signals not following any standard protocol, such as interrupts, power down handshake, etc.

7 Reset strategy

When a system is partitioned among different dice, it's possible that the memory from which the boot is executed is located in a die different from the one where the CPU executing the boot is located; a strategy ensuring the die containing the memory (die #2) exits from reset before the die containing the CPU (die #1) is then required, so to prevent requests from CPU to be lost during the system initialization phase.

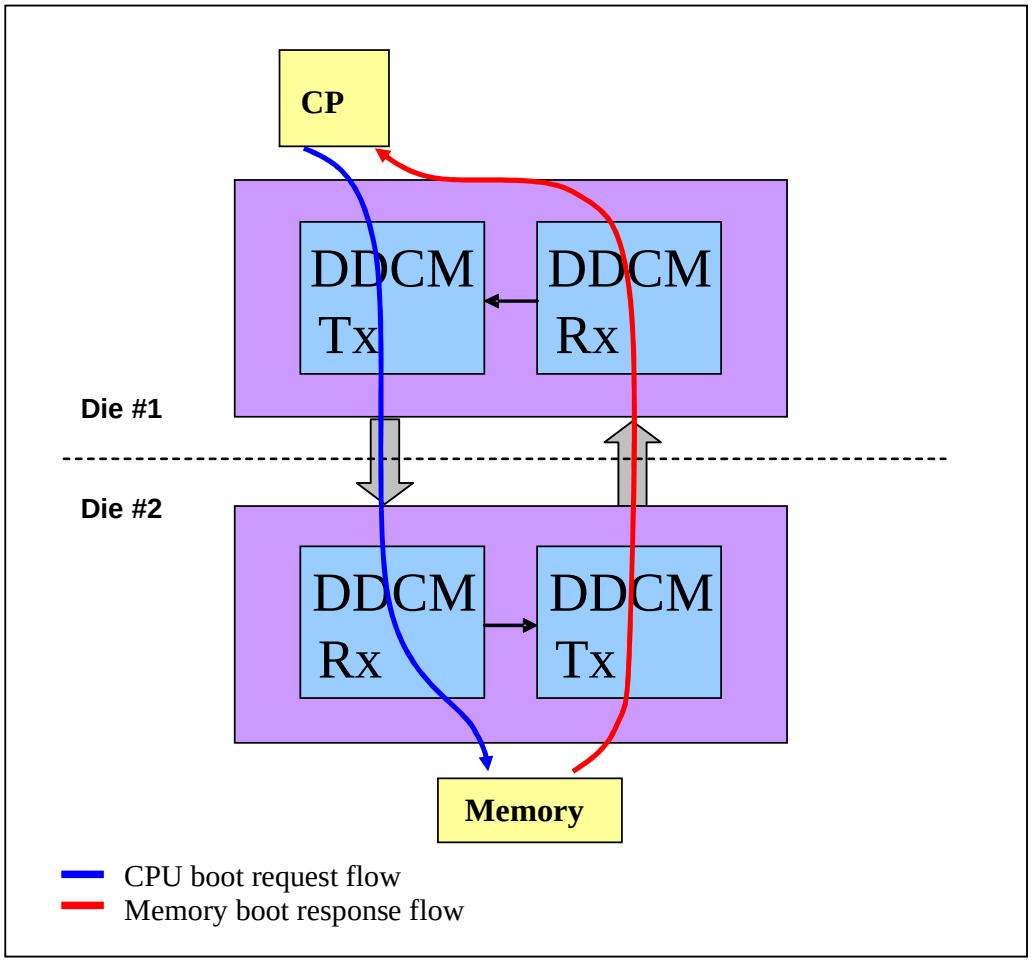


Figure 7-1: DDCM operation during system boot phase

This can be obtained in four different ways at least:

a) the reset controller takes care of reset sequence, guaranteeing that die #2 exits from reset before die #1;

b) the reset controller resets die #2, die #2 signals die #1 it has left the reset phase, and die #1 can proceed with its reset procedure;

c) each die informs continuously the adjacent dice with whom it can talk about its status (available or not available), so that a DDCM transmitter won't assume a data to be sent if

the destination status is not available. This can be achieved for example through a loopback strategy, exploiting a specific *magic bit* of virtual wires.

d) DDCM transmitters have 0 credits after reset, so they can't transmit anything until they receive credit information from adjacent dice.

Solution b) requires additional wires to inform the different DDCM transmitters about the status of the different DDCM receivers.

Solution c) requires additional logic to implement the loopback strategy (i.e. a Tx in the first die sends a virtual wires bundle and waits for the Rx in the second die to send back a copy).

Solution d) looks not practical since, having DDCM transmitter 0 credits after reset, the credit information can't be transmitted either.

8 Power control

This section describes the strategy adopted in the DDCM for reducing as much as possible the power consumption.

Two main mechanisms are used for achieving this objective:

- Activity-driven clock gating
- Source encoding

The **activity driven clock gating** is a technique relying on the request-activated clock (RACK) philosophy, aiming at controlling the PHY clock so to reduce the power consumption at pads level.

Actually when no activity is detected at PHY adapter output for a well determined number of clock cycles, a command is issued telling the PHY to switch-off its clock. As soon as activity starts again to be detected, a command to resume the clock is issued to the PHY.

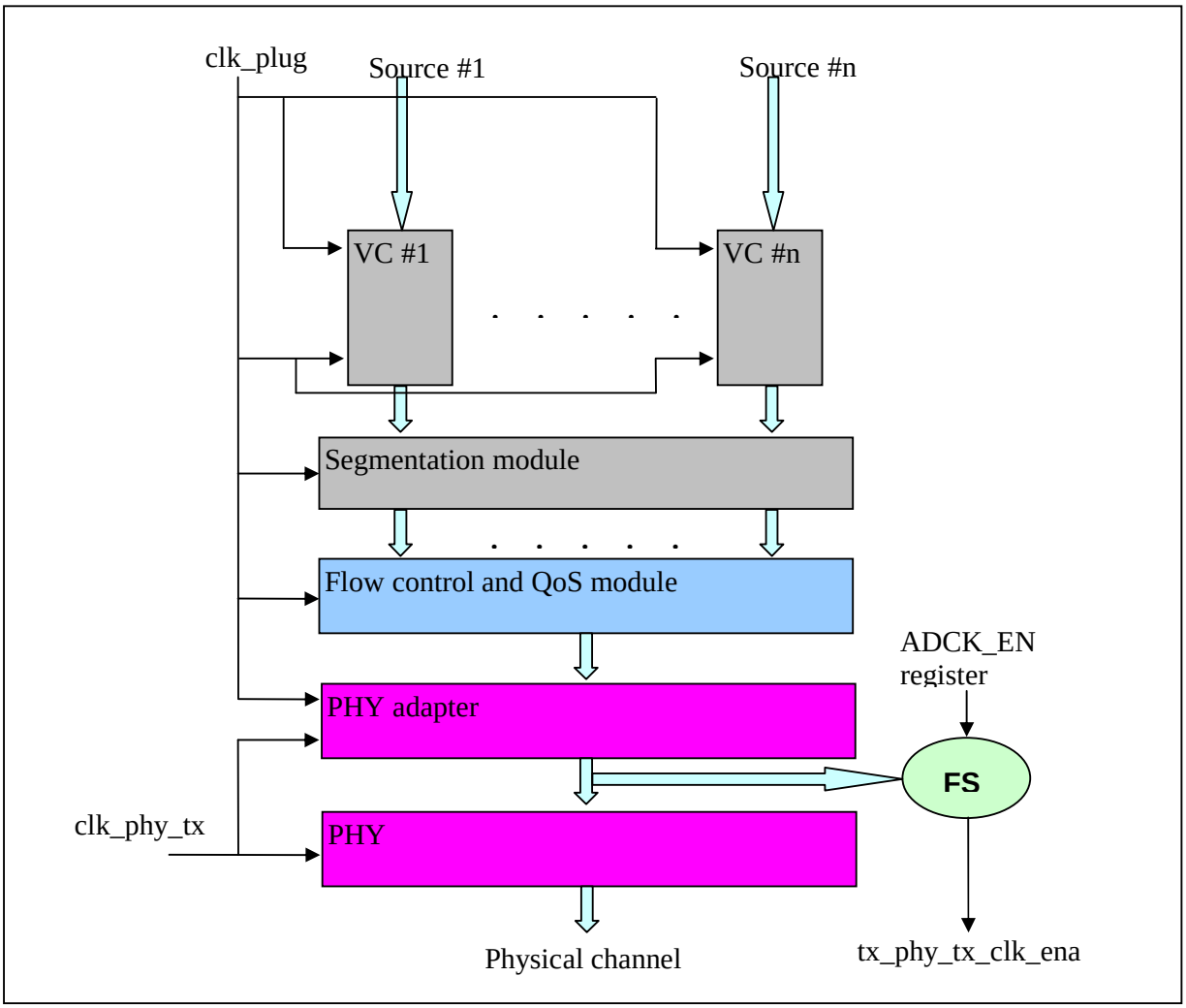


Figure 8-1: Activity-controlled clock gating in DDCM transmitter

As highlighted in figure 8.1 the activity driven clock control is performed by a block at PHY adapter hierarchy level, according to the following operation:

- the ADCG_EN register contains the number of clock cycles in the PHY clock domain that have to elaps with no phyts sent to PHY interface before switching-off the PHY clock; if the value stored in this register is 0, Activity Driven Clock Gating is not enabled and the PHY clock is always on;
- when no phyts are transmitted from PHY adapter serializer interface, a counter starts counting;
- until there are no phyts transmitted from PHY adapter serializer interface and the counter value is lower than the value stored in the ADCG_EN register, the counter is incremented;
- if before reaching the value stored in the ADCG register a phyt is transmitted from PHY adapter serializer interface, the count is interrupted;
- when the counter reaches the value stored in the ADCG register the tx_phy_tx_clk_enable signal is set to ‘0’; this is a command to the PHY telling that the PHY clock has to be switched–off;
- as long as no phyts are transmitted from PHY adapter serializer interface, the tx_phy_tx_clk_enable signal is kept to ‘0’;
- as soon as a new phyt is transmitted from the PHY adapter serializer interface, the tx_phy_tx_clk_enable signal is set again to ‘1’; this is a command to the PHY telling the PHY clock has to be resumed.

Notice that the functionality described above is implemented in the PHY transmitter clock domain, and the ADCG_EN register, coming from the DDCM clock domain, is properly synchronized in order to avoid metastability issues.

Normally when a command to switch-off the clock is issued, the actual clock switch-off occurs after some cycles of latency required to empty the PHY pipeline; when a command to resume the clock is issued, the clock should be resumed immediately.

The **source encoding** is a technique implemented in the PHY adapter and aiming at minimizing the Hamming distance between two consecutive phyts transmitted over the physical channel, in order to minimize the switching activity.

Bus Inverter (BI) encoder

This module implements a source encoding technique allowing minimizing the Hamming distance between two consecutive phyts, so to reduce as much as possible the switching activity over the physical channel during the transmission of information from one chip to the other.