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

DDCM with electrical PHY design and verification database

Deliverable no.:	D5.2
Due date:	10/31/2012
Actual Submission date:	10/31/2012
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 data base containing the design and the verification environment of the Dual Die Communication Module (DDCM). The file system structure as well as the tools used for the design and the verification of the block is described.

Change Records

Version	Date	Changes	Author
0.1 (draft)	2012-11-15	Start	Alberto Scandurra
1 (submission)	2012-11-15	Final version	Alberto Scandurra

1. *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.

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

- the DDCM **controller**, responsible for managing incoming/outgoing STNoC/SBus/AMBA-AXI traffic, generating IDN segments through 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.

As shown in figure 1.1, the DDCM top level in each die consists of a transmitter (DDCM Tx) and a receiver (DDCM Rx).

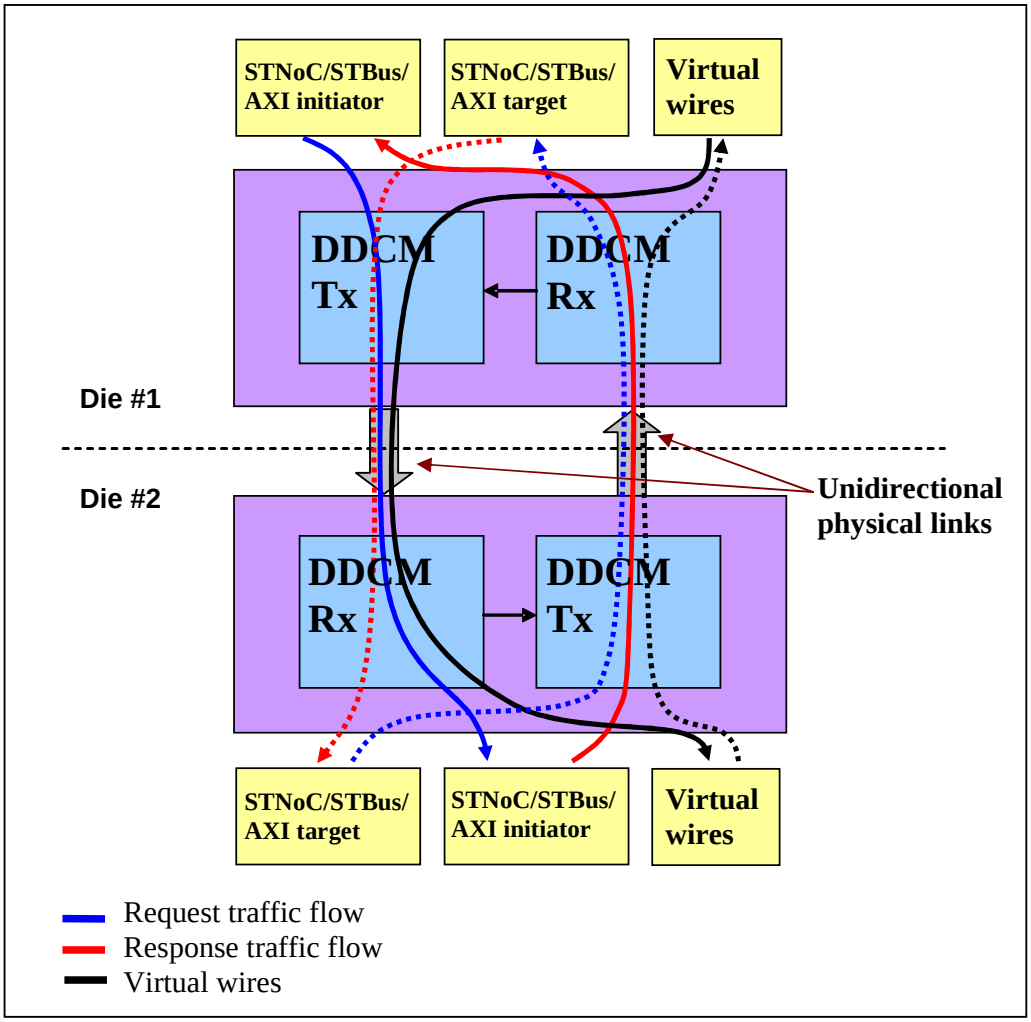

Figure 1-1: DDCM top level architecture and information flow

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 1-2 shows a full architectural view of an DDCM, highlighting the separation between an DDCM transmitter and an DDCM receiver.

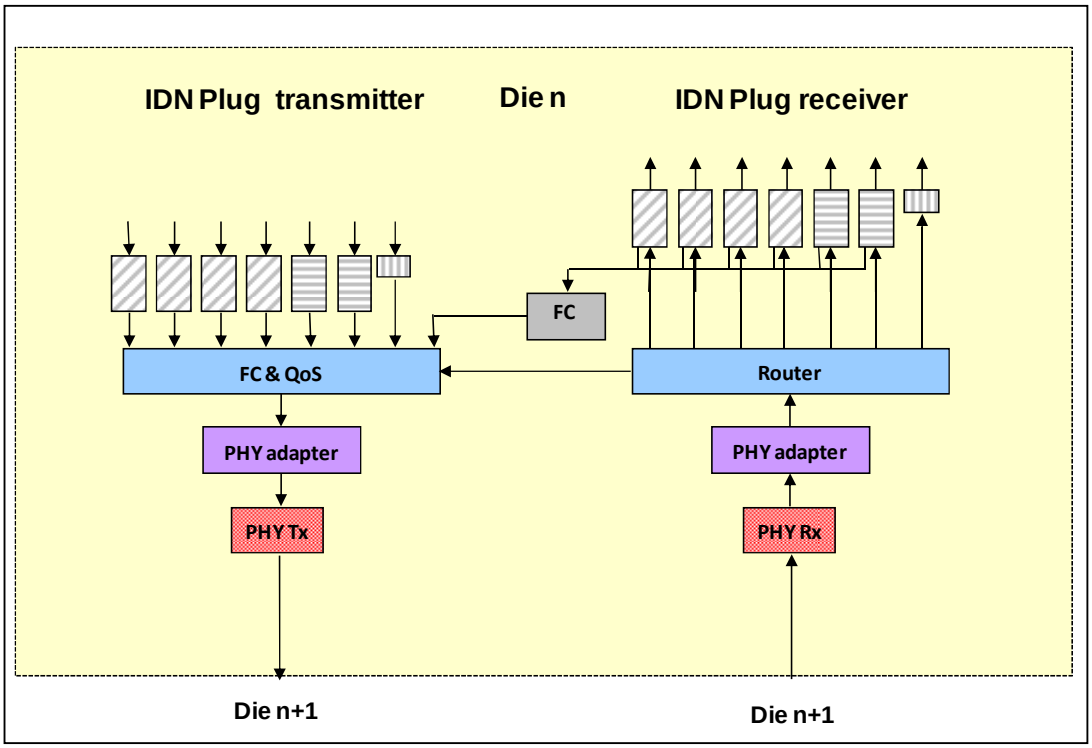

Figure 1-2: DDCM detailed architecture

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 next section describes the environment where such a block has been designed and verified, highlighting the data base structure, the methodologies adopted to manage the mentioned parametric approach and the tools used in the different phases of the design flow.

2. Data base structure

The DDCM data base is located in the ST Interconnect System Group server design area under the directory **ddcm_lib** identifying the design library.
The ddcm_lib directory contains the following two subdirectories:

- **dev** (development) containing the generic design and the generic verification environment;
- **run** containing the simulation area for a set of specific configurations of the design.

The directory dev contains the following subdirectories:

- **doc** containing the functional specification of the block (deliverable D5.1 in NAVOLCHI project context);
- **rtl_vhdl** containing the VHDL files representing the rtl description of the DDCM top level and all its building-blocks;
- **corekit** containing the generic view of the DDCM;
- **verif_env** containing the generic verification environment, i.e. testbench and stimuli sources;
- **verif_run** containing the tests to verify the different functionality of the DDCM;
- **synth** containing the area for the logic synthesis of the DDCM.

2.1 Design environment

The design environment consists of the directory **rtl_vhdl** and **corekit**.

In the rtl_vhdl directory there are the files describing VHDL entity and architecture of the DDCM top level and all its building-blocks (i.e. transmitter, receiver, FIFOs, etc.)
All these blocks are described following a parametric approach, so that after setting a proper set of parameters to the required values, the generic design gets configured accordingly and becomes specific for a well defined application. As described in the DDCM functional specification (deliverable D5.1) the design parameters allow to characterize the DDCM in terms of interfaces size, FIFO depth, traffic management policy, clock frequencies, etc.).
The VHDL description is *technology independent*, that is to say the VHDL files describe the structure and the functionality of the DDCM, with no links with the CMOS technology with which the DDCM itself will be implemented.

The corekit directory contains a set of scripts allowing to build the so called *corekit*, a file containing all the information about the generic design and allowing by means of a GUI (Graphic User Interface) the user to assign the required values to the design parameters and getting a specific configuration of the DDCM, moving from the generic description.

While the VHDL description is *tool independent*, i.e. the VHDL files can be written with any text editor, the corekit generation is *tool dependent*, and is based on the so called *core tools* developed by **Synopsys**.
Specifically, the tool moving from the generic VHDL description and building the corekit is called **coreBuilder** (see fig. 2-1), while the tool allowing to use the corekit and generating a specific configuration of the DDCM after having assigned proper values to the parameters is called **coreConsultant** (see fig 2-2).

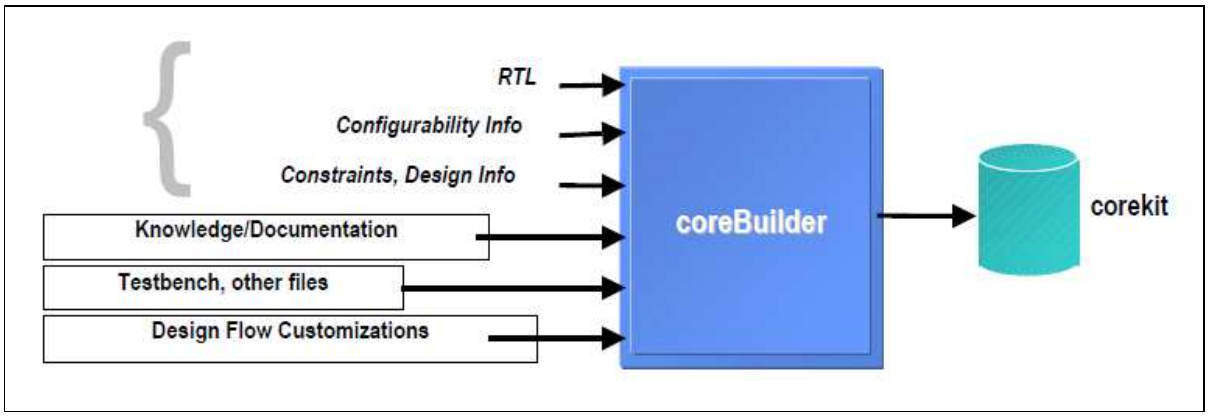

Figure 2-1: corekit generation flow through coreBuilder

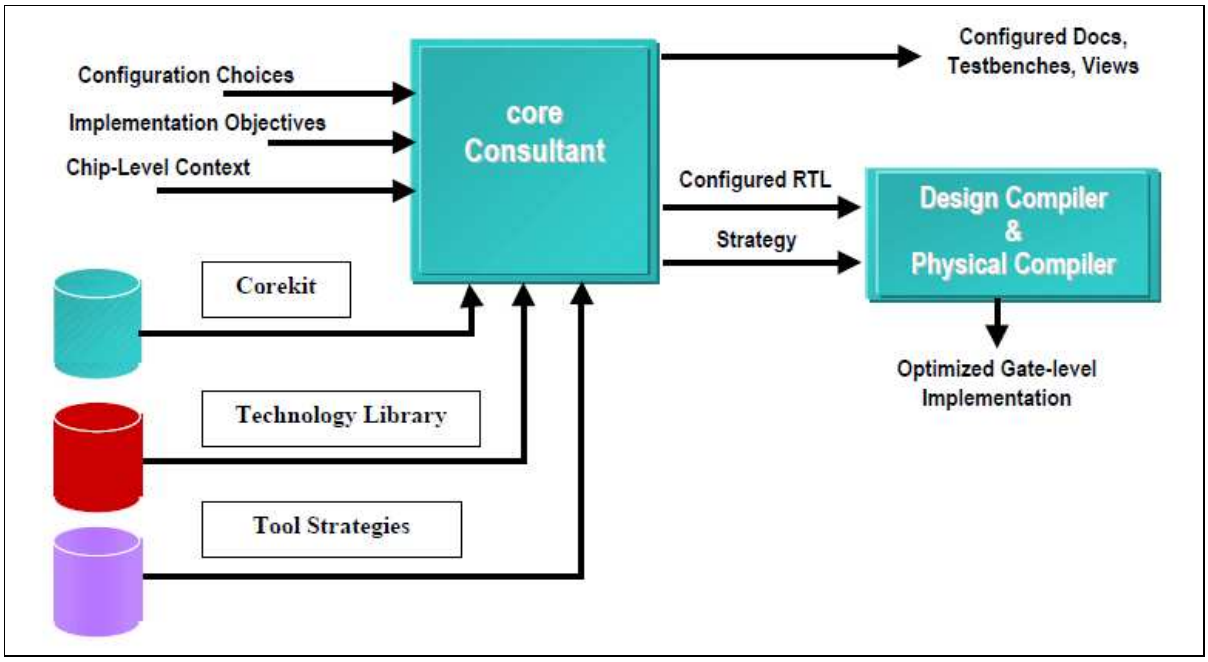

Figure 2-2: corekit utilization flow through coreConsultant

2.2 Verification environment

The verification environment consists of the directories **verif_env** and **verif_run**.
The verif_env directory in turn contains the following main subdirectories:

- **rtl_tb** containing the VHDL testbench, i.e. the structure instantiating the DDCM, considered the **DUT** (Device Under Test) and the traffic generators for stimulating the design and verify its behaviour accordingly;
- **e** containing the functional description of the traffic generators in *e* language, an object oriented high level language specific for verification suites;
- **config_files** containing a variety of files with different set of design parameters so to configure the DDCM in different ways in order to verify as many different specific implementations as possible;

- **tests** containing the description of different tests, aiming at stimulating the different DDCM functionalities.

The verif_env directory contains generic descriptions of all the structures described in it (testbench, stimuli generators, tests); the verif_run directory contains a replication of the verif_env subdirectories but configured according to the design parameters, and the specific for a well defined application or product employing the DDCM.

Figure 2-3 shows the DDCM verification environment structure.

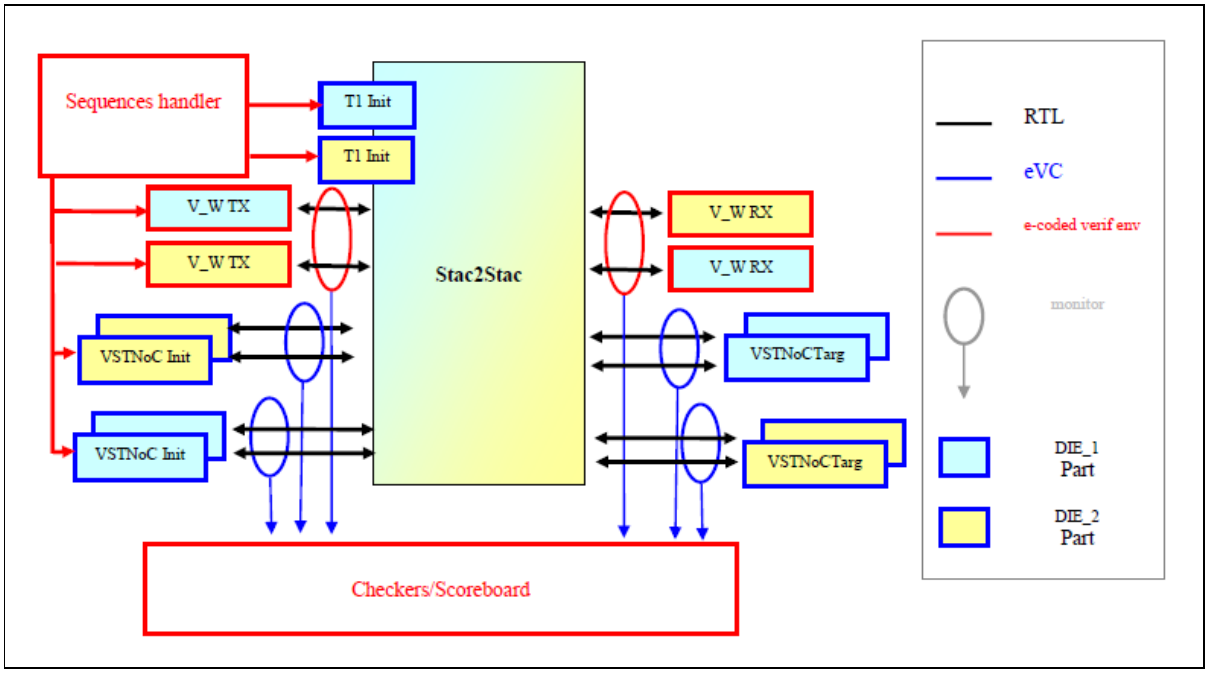

Figure 2-3: DDCM verification environment

In this figure the central block called Stac2Stac represents the DUT composed of two DDCM connected to each other, implementing the communication between two dice.
The top left red block called Sequences handler is the module responsible for the traffic generation in order to stimulate the DUT. The bottom red block called Checkers/Scoreboard is the module responsible for the actual verification of the functionality of the DUT, performing specific checks on peculiar functionalities and checking the correct transfer of information from one die to another across the two DDCM modules.
The whole verification environment is based on the functional verification tools developed by **Cadence**.

Figure 2-4 shows the list of tests implemented (right column) and the DDCM functionalities verified by each of them (left column).

Feature Name	Test
Reset	All (through test_common.e import)
Static/Dynamic Configuration	prog_test.e mixed_pin_prog_test.e reset_prog_vstnoc_test.e routing_check_test.e global_test.e
VSTNoC INIT to TARG communication	reset_prog_vstnoc_test.e routing_check_test.e general_test.e test fast_init_slow_targ.e fast_init_targ.e testmaster_test.e resp_opt_test.e slow_init.e test unalign_addr_test.e single_packets.e general_test_adv.e general_AXI_STBUS_test.e global_test.e
VC-ID based routing	routing_check_test.e global_test.e
Virtual Wires TX to RX communication	Virtual_wires_test.e vw_single_bundle_toggling_test.e global_test.e
Synchronization	reset_prog_vstnoc_test.e routing_check_test.e
Power control	All
Credit-based flow control	All
VC priority setting	global_test.e
Store and Forward	saf_test.e
Variable segment size	global_test.e security_handler_test.e
QoS optimization	All
VCs connection dynamic programming	vc_connection_prog_test.e
SRC-based routing	src_routing_test.e
Phyt encryption	security_handler_test.e

Figure 2-4: DDCM test list

2.3 Synthesis environment

The synthesis environment consists of the directory **synth**.
This in turn contains the following subdirectories:

- **input** containing the configured VHDL code for a specific DDCM implementation;
- **scripts** containing the commands for the synthesis tool;
- **run**, the directory where the synthesis tool is invoked and log files are recorded;
- **reports** containing the characterization of the synthesized design in terms of area, timing (speed) and power consumption
- **output** containing the synthesized DDCM design in terms of Front-End netlist.

The DDCM synthesis environment is fully based on **Synopsys** tools, i.e. **Design Compiler** as synthesis engine and **Design Vision** as interactive GUI.

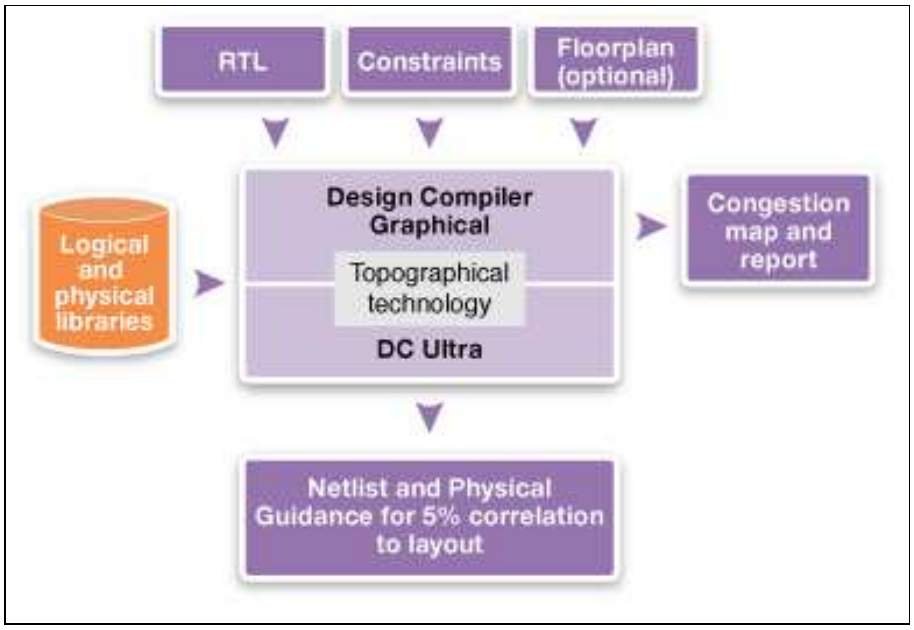

Figure 2-5: Synthesis flow based on Synopsys Design Compiler

The synthesis design phase is strongly technology dependent, since the results of the synthesis process, i.e. the gate level netlist, is an assembly of technological standard cells implementing the structure and the functionality of the DDCM in the required technology.
In its first version the DDCM has been synthesized using 65nm and 40nm CMOS technologies.

Based on the Synthesis environment, a flow for the characterization of the DDCM in terms of power consumption has been developed, as shown in figure 2-6.
According to this flow a specific configuration of the DDCM is simulated many times, and for each simulation the switching activity at each node and across each wire of the design is recorded; then this switching activity is back-annotated on the netlist in Design Compiler environment, and the power analysis tool is invoked so to calculate the power consumed by the DDCM block taking into account the switching activity determined by the traffic injected in different scenarios.
Relying on the obtained data, average and peak power consumption is determined.

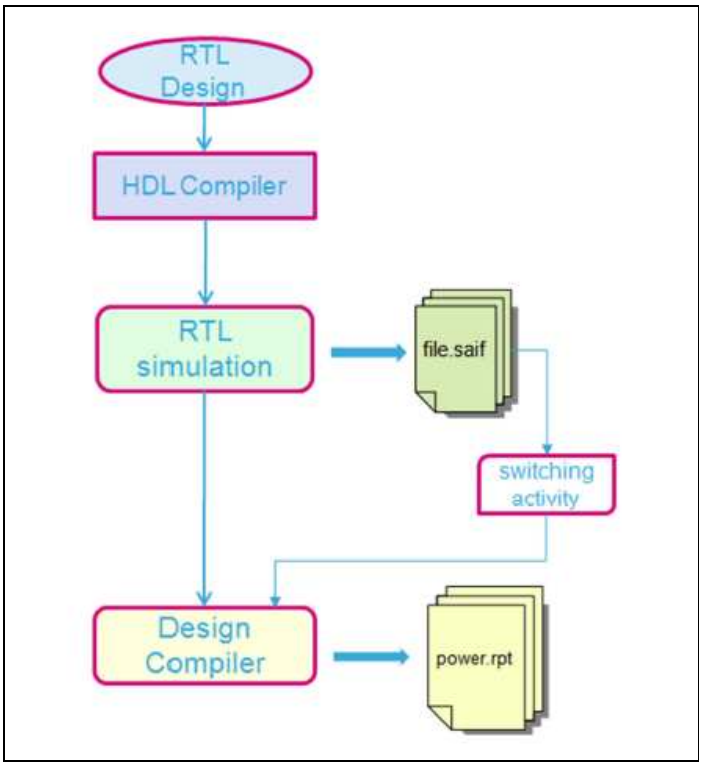

Figure 2-6: Power consumption characterization flow

3. Conclusion

This document describes the DDCM design and verification data base (deliverable D5.2) where the DDCM block is implemented and functionally verified according to its functional specification (deliverable D5.1).