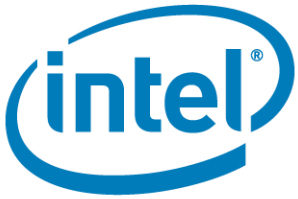

CFG AMBA IP Integration Specification – UVM Addendum

Version: 1907L

Revision: 0.0

Intel Confidential

CFG AMBA IP Integration Specification: UVM Addendum

About this document

This document describes the architecture of the CFG AMBA UVM bench. This includes details of the IP components and instructions on how to generate and use the UVM bench for customer NoCs. The IP-XACT file generated by NocStudio contains information on the generated NoC. The Cadence `iregGen` tool takes the IP-XACT file as input to generate the register model (`uvm_reg_block`), which is used to perform register accesses.

Audience

This document is intended for users of NocStudio:

- NoC architects, designers and verification engineers
- SoC architects, designers and verification engineers

Prerequisites

Before proceeding, you should generally understand:

- Basics of Network on Chip technology
- AMBA4 specification and UVM methodology

Related documents

The following documents can be used as a reference to this document.

- CFG NocStudio Orion AMBA User Manual
- CFG Orion IP Integration Specification
- CFG Orion Technical Reference Manual

Customer support

For technical support about this product and general information, contact CFG Support.

Revision History

Revision	Date	Updates
0.0	Jul 19, 2019	Initial Version

Contents

List of Figures

List of Tables

1 UVM SANITY BENCH OVERVIEW

NocStudio generates a UVM verification testbench that can be used to sanity-test the generated NoC. Following are the salient features of the automatically generated testbench.

1) Testbench is auto-generated using Cadence VIP. Configuration of all the VIP instances is done automatically.
2) ACE, ACE-Lite, AXI4, AXI4-Lite, AXI3, AHB-Lite, APB, and IMG2 are the protocols supported.
3) Interface checkers as well as the end-to-end checker are instantiated to monitor the NoC traffic. They are bound appropriately to the RTL modules.
4) The address range configuration of the NoC is stored in objects (in the `NsSocUvmTestConfig` class) which can be used by the traffic generation sequences.
5) Test sequences can be changed to generate directed stimulus. Sequences are extended from Cadence base sequences.
6) IP-XACT output generated by NocStudio is used to generate the register model for the NoC.
7) Simulation script is generated which helps compile and run the test.
8) The included `basicTest` test will generate traffic stimulus with appropriate constraints based on address configuration. If a register bus is present in the NoC, a register sanity sequence which verifies the POR values of all the registers is run automatically. If the NoC is low-power enabled, the test will toggle each power domain before driving traffic.
9) Command line arguments provided to the `run_test_incisiv.sh` script (see Running UVM sanity testbench) can be used to control knobs such as:
    a. Number of transactions to be generated from each host
    b. Cadence VIP tracker enables
    c. Waveform dumping
    d. Transaction coverage via in-built Cadence VIP coverage classes

Generation of the UVM bench is controlled by FLEXLM licensing. Once the license is provided, the NocStudio property for UVM bench generation (`prop_default uvm_bench_enable`) needs to be enabled in the configuration file to generate all the components of the testbench.

If regbus is enabled in the configuration, IP-XACT generation (`prop_default ipxact_enable`) needs to be enabled as well. This IP-XACT output is used in the auto-generation of `uvm_reg` based register model. This model can be used to access registers by name.

1.1 UVM BENCH ARCHITECTURE

The following figure depicts the UVM testbench that is auto-generated by NocStudio. Each of the components is described in more detail in the following sections.

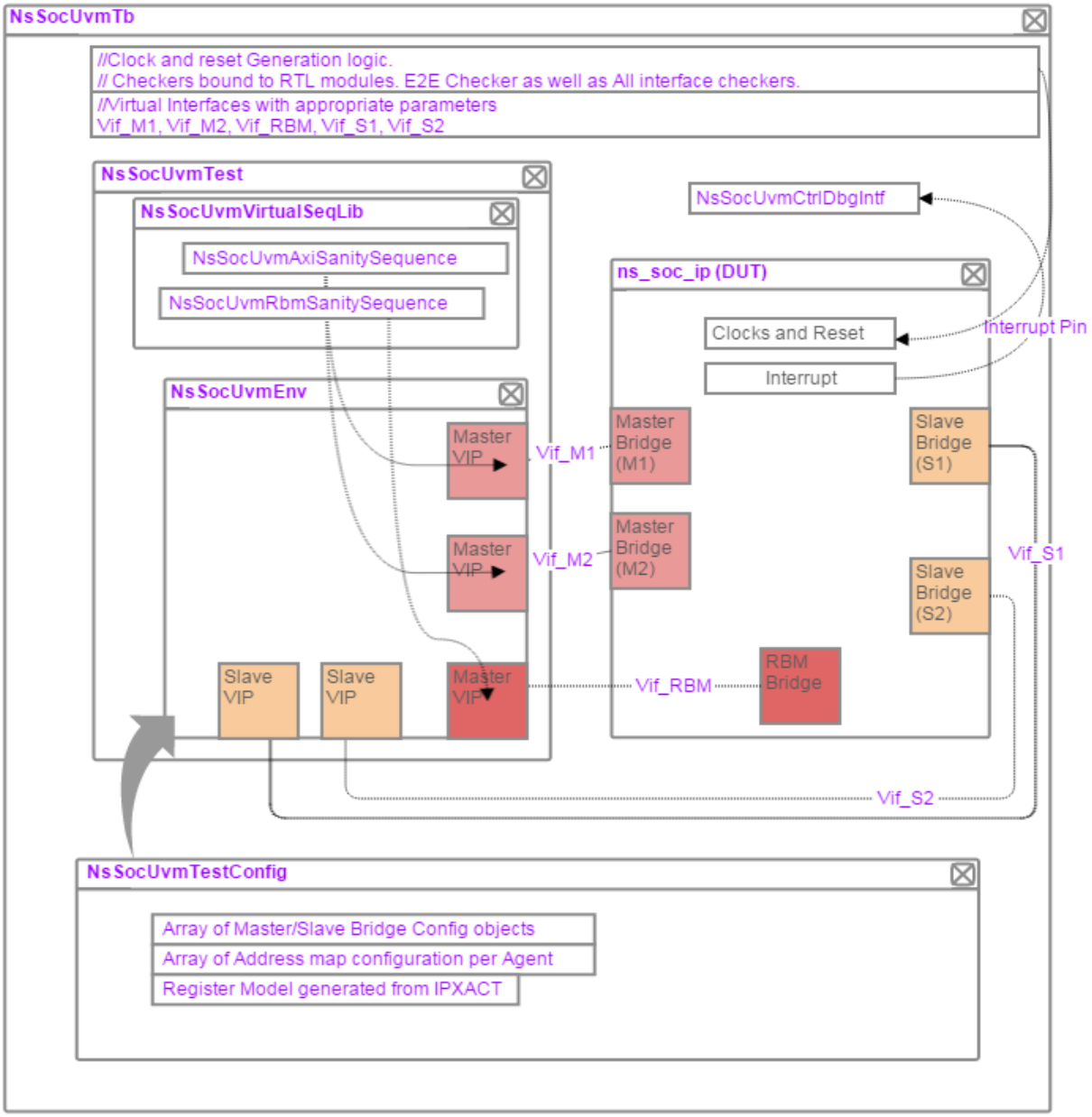

Figure 1: UVM Testbench Architecture

1.1.1 Testbench top

The top-level file for the testbench module is named `NsSocUvmTb.sv`.

In this file, the DUT (`ns_soc_ip`) is instantiated and all the necessary connections to Cadence VIP agents is done through the use of SystemVerilog interfaces.

There is a debug interface which monitors the interrupt pin(s) of the DUT. If any interrupts are generated during the test, an assertion fires to stop the test. This interface also holds the pin which indicates end of simulation for the test run.

The `NsSocUvmPkg.sv` package file is included here, which includes the base virtual sequence (`NsSocUvmVirtSeq_base`), dynamically-generated virtual sequence library (`NsSocUvmVirtualSeqLib.sv`), base test class (`NsSocUvmTest_base`), and dynamically-generated test library (`NsSocUvmTest.sv`). `NsSocUvmTest_base` in turn instantiates the environment (`NsSocUvmEnv`) and test configuration (`NsSocUvmTestConfig`) components.

Checkers (all interface checkers as well as the end-to-end checker) are instantiated and bound to the respective RTL modules.

1.1.2 NoC environment

The `NsSocUvmEnv` class handles the instantiation and configuration of Cadence VIPs. It makes use of the configuration class (`NsSocUvmTestConfig`) auto-generated by NocStudio.

The configuration object contains information about the host agents, such as type (i.e. ACE/AXI4/Lite/3/AHB/APB), pin widths, and address range configuration.

1.1.3 NoC test configuration

The `NsSocUvmTestConfig` class stores information about the created NoC, such as the number of agents as well as their instance-specific interface properties. The system address map configuration is also put into data structures so that they can be accessed by other components in the testbench. A pointer to it is passed around, enabling the environment and test to retrieve the NoC configuration.

1.1.4 NoC test sequences

The `NsSocUvmVirtualSequence` is a virtual sequence that instantiates the sequencers needed for controlling stimulus to all the agents. The test sequences in the per-agent sequence libraries `NsSocUvmAceMasterSeqLib.sv`, `NsSocUvmAxiMasterSeqLib.sv`, and `NsSocUvmAhbLiteMasterSeqLib.sv` can be invoked on each of the sequencers.

1.1.5 Register bus sequence

A regbus master (AXI4-Lite) transactor is instantiated when a NoC has register bus enabled. The register model is built by running the `iregGen` tool on the NocStudio-generated IP-XACT file. An adapter takes the transactions and converts them into AXI4 protocol.

1.2 DIRECTORY STRUCTURE

All the components of the UVM testbench are generated under a directory named `verif` inside the project directory.

Name	Description
verif/env	All dynamically-generated UVM classes and testbench files, based on the NoC configuration file.
verif/uvm	Static sequence classes, static test classes.
verif/uvc	Static protocol-specific UVM files: Cadence VIP interfaces, VIP configuration classes, address map related classes, regbus adapter, and protocol-specific sequence libraries.
verif/sim	Script to compile and simulate the testbench.

Table 1: NoC UVM testbench directory structure

2 NOCSTUDIO FLOW TO GENERATE A UVM SANITY BENCH

Figure 2 describes the NoC IP generation flow using NocStudio. The user specifies a NocStudio command script that describes the user system requirements. NocStudio processes this script to construct a deadlock-free NoC that meets all the system requirements. The following files are generated by NocStudio for the NoC:

- NoC RTL
- NoC verification
- Sanity testbench
- UVM Sanity testbench
- Synthesis scripts
- HTML spec for the generated NoC

All the generated files are output to the project directory, whose name corresponds to the project name specified in the `new_mesh` command in the NocStudio command script.

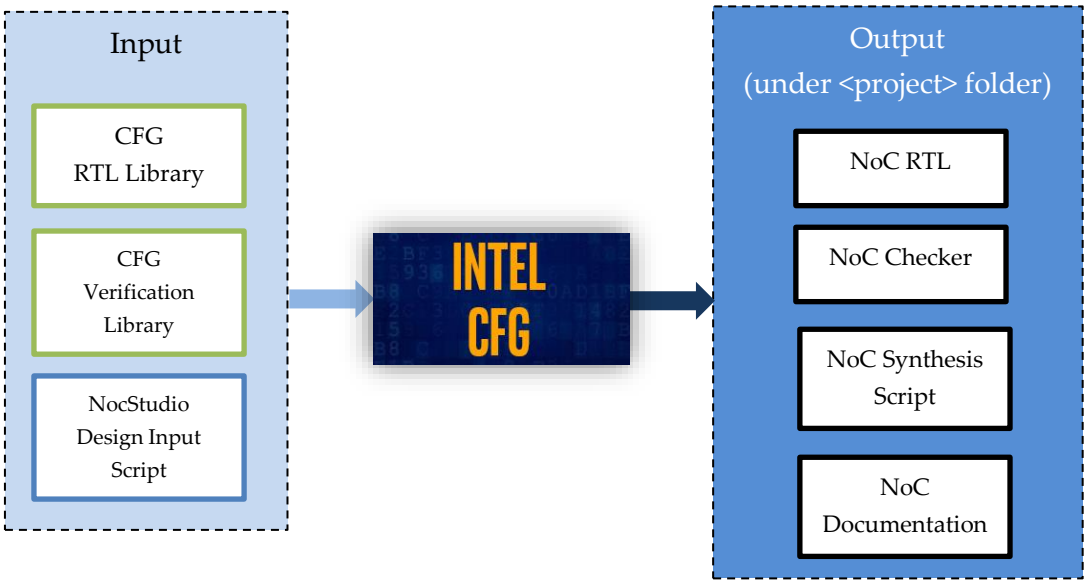

Figure 2: NoC IP generation flow

2.1.1 Example for generating a UVM sanity bench

To generate NoC RTL, include a `prop_default uvm_bench_enable yes` command along with a `gen_ip` command in the NocStudio command script, and then process the script with NocStudio.

For example, from the IP root directory, run the following command for GUI mode:

```
./NocStudio examples/example_cache1.txt
```

Or the following command for batch mode:

```
./NocStudio examples/example_cache1.txt -nogui
```

The last command in the above example script is `gen_ip`. Once the command executes, a project directory called `example_cache1/` is created which contains all the files and directories generated by NocStudio. The table below shows a list of key files related to the UVM sanity testbench.

Files listed as "static" are identical across all NoCs, while those listed as "dynamic" are NoC-specific and therefore dynamically generated by NocStudio.

Name	Description	Type
verif/env/NsSocUvmTb.sv	Testbench module which instantiates DUT (`ns_soc_ip`), UVM test, and checker binds.	Dynamic
verif/env/NsSocUvmEnv.svh	Environment class which instantiates and configures the Cadence VIP agents.	Dynamic
verif/env/NsSocUvmTest.sv	Library of specialized tests.	Dynamic
verif/env/NsSocUvmTestConfig.svh	Test configuration class with information about the NoC.	Dynamic
verif/env/NsSocUvmVirtualSeqLib.sv	Library of specialized virtual sequences that execute lower-level protocol-specific sequences.	Dynamic
verif/env/NsSocUvmConfigPkg.sv	Package with Cadence configuration classes and CFG-specific classes.	Dynamic

verif/uvm/NsSocUvmVirtSeq_base.sv	Base virtual sequence class extended by more specialized virtual sequences.	Static
verif/uvm/NsSocUvmTest_base.sv	Base test class extended by more specialized test classes.	Static
verif/uvc/NsSocUvmPkg.sv	Package containing all UVM bench classes.	Static
verif/uvc/*	Protocol-specific VIP files.	Static
verif/sim/run_test_incisiv.sh	Wrapper to run the UVM compile and simulation script in the `project_directory/scripts` directory.	Static

Table 2: Key files generated by NocStudio for UVM bench in project directory

3 RUNNING UVM SANITY TESTBENCH

Running the sanity testbench performs a sanity check on the NoC RTL in simulation. This is a push-button method of instantiating the NoC RTL in the provided testbench along with CFG Verification IP and running a sanity traffic pattern on the NoC RTL to validate basic operation of the NoC in simulation.

To run, change to the `project_directory/verif/sim` directory, and then run:

```
./run_test_incisiv.sh
```

This builds the required Cadence VIP libraries (if needed), compiles the sanity testbench, and launches the simulation. If regbus is present in the config, an `NsSocUvmRbm_rdb.sv` file is generated which holds the regmodel generated from IP-XACT.

Upon a successful compile and simulation, the following will appear at the prompt:

```
**********************************************************************
***PASSED***
**********************************************************************
```

If the NoC has register bus enabled, there will be an additional sanity test which is run as part of the same script. It will perform register transactions to verify the connectivity of the register bus before starting NoC traffic sequence.

The presence of a file named `SIM_FAILED` in the `verif/sim` directory after the completion of a simulation run indicates a failure, while the presence of a file named `SIM_PASSED` indicates a successful simulation run.

With a successful simulation from the sanity testbench, the generated NoC RTL and verification IP are ready to be integrated.

Some example command line arguments to the `run_test_incisiv.sh` script are listed below:

- Custom default transaction count: `-sim_opts="+count=10"`
- Per-master custom transaction count: `-sim_opts="+count_axi4m_m1_p1=2"`
- Default tracker enable: `-sim_opts="+tracker_enable=1"`
- Per-master tracker enable: `-sim_opts="+tracker_axi4m_m1_p1=1"`
- Disable Advanced Trace Bus checker: `-sim_opts="+ns_atb_checker_en=0"`
- Waveform dumping: `-waves=1`
- Coverage: `-coverage`

For the full list of command line arguments, refer to the help output of the run script:

```
./run_test_incisiv.sh -help
```

For the full list of run-time plusargs (generated according to which agents are present), refer to the lines containing `$value$plusargs` in `verif/env/NsSocUvmTb.sv`.

3.1 TOOL REQUIREMENTS

Supported versions of tools and languages:

- NoC RTL uses Verilog-2005 (IEEE Std 1364™-2005) syntax and its support must be enabled in the tool flow.
- NoC simulation environment uses SystemVerilog IEEE Std 1800-2009
- Simulator: Cadence Incisive 15.22.012
- Cadence VIP: VIPCAT 11.30.039

3.2 TOOL SETUP BEFORE COMPILING THE BENCH

Before running the UVM compile and test, the environment variable specifying the path to Cadence VIP root must be set. The following command is used to set the environment variable:

```
csh:  setenv CDN_VIP_ROOT “Path to VIP root”
bash: export CDN_VIP_ROOT=“Path to VIP root”
```

Check the tool requirements above to see the supported/tested tool versions.

3.3 BUILDING REGISTER MODEL

When regbus is present in a NocStudio configuration file, the script automatically builds the register model from the generated IP-XACT file. The IP-XACT file resides in the project directory and is named `ns_soc_ip.xml`.

```
<++> Cadence iregGen version 15.22.s012
<++> XML parsed. Started decoding
.....................................................................
.....................................................................
.....................................................................
.........................................
<++> Decoding done
<++> Input File: ../../ns_soc_ip.xml
<++> Number of AddrMaps = 3
```

```
<++> Number of RegFiles = 1
<++> Number of Registers = 258
<++> Number of Memories = 0
<++> Files Created: ./NsSocUvmRbm_rdb.sv
<++> Files Created: ./quickTest.sv
```

The `iregGen` tool generates a file in the `verif/sim/` directory named `NsSocUvmRbm_rdb.sv` that holds the register model class. This class abstracts the addresses of the registers and provides a name to be used by the test sequences.

4 FEATURES NOT SUPPORTED IN THIS RELEASE

Following are the features which are not yet supported in the UVM testbench generation.

- Ratio-synchronous bridge clock crossings: Suggest replacing "`bridge_prop ... clock_cross ratio_*`" with "`bridge_prop ... clock_cross async`".
- Tunnel regbus stimulus using register model.
- Error transactions handling. Error transactions can be injected by the stimulus and in that case the checkers will assert.
- Sweep test to run through all the master and slave connections. Currently, the number of transactions can be increased so that the randomly generated stimulus will eventually generate transactions for all the master and slave connectivity paths.
- ACE and ACE-Lite agents are supported, but coherent traffic is not. Therefore, we intentionally restrict:
    - Address map population to make all ranges non-shareable
    - The transactions generated for ACE and ACE-Lite bridges to be non-snooping

4.1 KNOWN LIMITATIONS

4.1.1 "/" in address range names

There is a known limitation with "/" in address range naming. Having a "/" in the address range name breaks the compile of the generated register model classes (from IP-XACT).

It is therefore suggested to remove the "/" from the address range name to make use of the auto generation of register model.

4.1.2 IMG2 bridge limitations

The following are not supported for IMG2 configs:

- User width per byte = 0
- Trans width = 0
- AID width = 0

4.1.3 "FIFO not empty" end-of-test check failure

Although our end-to-end checkers ensure that all traffic has completely and correctly flowed through the NoC, there may be some spurious `NS_ASSERT_EXIT_CHECK_NS_A_FIFO_EMPTY` assertion firings observed.

4.1.4 Post-test quiesce period

Our internal testing has uncovered that in certain unusual cases, tests ended prematurely due to all objections having been dropped, causing our end-of-test checkers to flag false positives. We have added a quiesce wait at the end of the virtual sequence body to allow all traffic to finish flowing through the DUT before the test ends. It is possible that in certain scenarios, this wait may need to be lengthened. This can be done by increasing the value of the `quiesce_wait_ns` field of the `NsSocUvmVirtualSequence` virtual sequence.

5 KNOWN VIP ISSUES

There are a few issues with the VIP which were encountered as part of our development. Those are listed below for reference.

5.1 SEGMENT SIZE 1K ALIGNMENT

```
Segment size is not a multiple of 1K, or segment is not aligned to 1K
boundary. segment size is: 0x00000000000000100
See user guide section 5.1.6 Setting Slave Address Space and AMBA spec. 3-19
- Failed condition: ((seg.size % 1k == 0) and (seg.start_ad % 1k == 0))
```

5.1.1 Solution

(Cadence Bug ID: 45811031) – Cadence has added a new register to the VIP,

`DENALI_CDN_AHB_REG_EnableNon1KAlignedSeg`, which can be used to configure the VIP to disable this check and allow regions to be non 1k aligned.

This register is available in Cadence VIPCAT version 11.030.035 and later.

5.2 RID WITH NO MATCHING ARID

`RID_WITH_NO_MATCHING_ARID – Fatal Error.`
When run with no specific timescale configured for VIP, there is a fatal error during read transaction.

5.2.1 Solution

(Cadence Bug ID: 42812446) – Cadence provided a solution to add a knob to command line

Add `-defineall CDN_VIP_VM_TIMESCALE=1ps/1ps` to command line to force a timescale for all VIP components.

This has been added to the script by default.

5.3 ADDRESS CONSTRAINT

Address constraint limited to half the possible address range

5.3.1 Solution

Disable default FirstAddress constraint.

5.4 USERBITS

Userbits should be less than 256 bits – Cadence VIP limits to this range.

5.4.1 Solution

Cadence thinks that the upper limit of 256 bits for userbits is workable for most customers.

5.5 IMG2 SLAVE VIP UNIQUE TAG IDS

The IMG2 slave VIP should not be expecting unique tag IDs.

5.5.1 Solution

This check should be disabled.

5.6 IMG2 SLAVE VIP >32-BIT ADDRESS WIDTH

The IMG2 slave VIP throws an assertion error if the slave interface address width is configured to be greater than 32 bits.

5.6.1 Solution

Use address widths of at most 32 bits.

2200 Mission College Blvd
Santa Clara, CA 95054
www.intel.com