NetSpeed Orion AMBA IP Integration Specification: UVM Sanity Bench Addendum

Version: ORION-16.04

April 15, 2016

NetSpeed Orion AMBA IP Integration Specification: UVM Addendum

About This Document

This document describes the architecture of NetSpeed Orion AMBA UVM Sanity bench. This includes details of the IP components and instructions on how to generate and use the UVM bench for customer NoC. The IPXACT file generated by NocStudio has all the information regarding the NoC generated. The registers are generated using iregGen tool with generated IPXACT file as input. The iregGen tool generates the register model (uvm_reg_block) which is used to perform register accesses.

Audience

This document is intended for users of NocStudio:

- NoC Architects
- NoC Designers
- NoC Verification Engineers
- SoC Architects
- SoC Designers
- SoC Verification Engineers

Prerequisite

Before proceeding, you should generally understand:

- Basics of Network on Chip technology
- AMBA 4 interconnect standard
- UVM Methodology

Related Documents

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

- NetSpeed NocStudio Orion AMBA User Manual

- NetSpeed Orion AMBA Physical Design Guidelines
- NetSpeed Register Bus Protocol
- NetSpeed Orion AMBA Protocol Support

Customer Support

For technical support about this product, please contact support@netspeedsystems.com

For general information about NetSpeed products refer to: www.netspeedsystems.com

Contents

List of Figures

List of Tables

1 UVM Sanity Bench Overview

NocStudio generates a UVM verification testbench which 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) 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 (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) IPXACT 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) Included basicTest will generate traffic stimulus with appropriate constraints based on address configuration. If regbus 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) Knobs provided to the script (run_test_incisiv.sh) can be used to control the number of transactions to be run per each host. Enabling trackers (Cadence VIP) and waveform dump can also be controlled by command line knobs.
10) Transaction Coverage can be obtained by using the inbuilt Cadence VIP coverage classes.

Generation of UVM bench is controlled by FLEXLM license. Once the license is provided, NocStudio property (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, IPXACT generation (prop_default ipxact_enable) needs to be enabled as well. This IPXACT 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

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

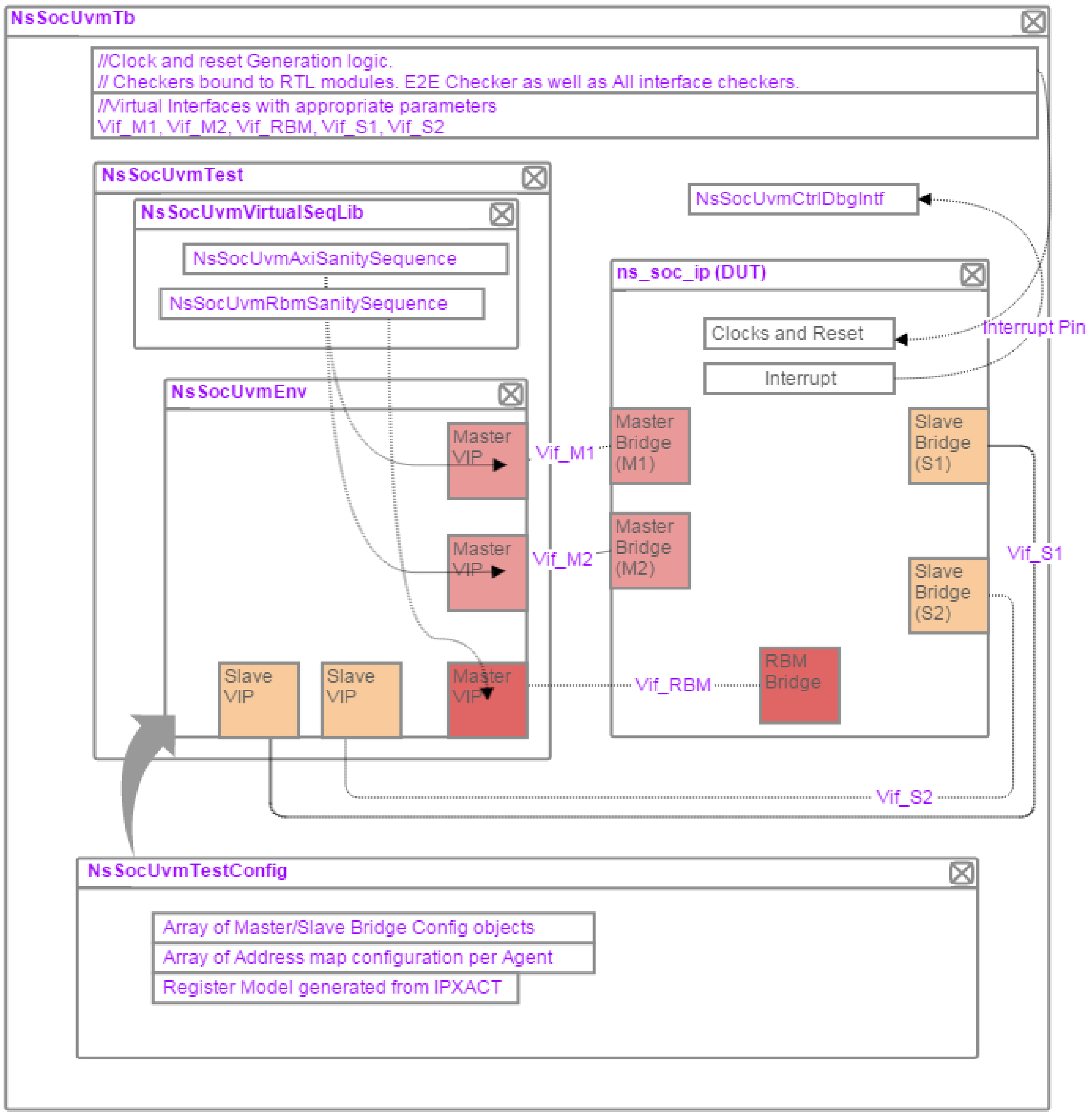
NsSocUvmTb
//Clock and reset Generation logic.
// Checkers bound to RTL modules. E2E Checker as well as All interface checkers.
//Virtual Interfaces with appropriate parameters
Vif_M1, Vif_M2, Vif_RBM, Vif_S1, Vif_S2
NsSocUvmTest
NsSocUvmVirtualSeqLib
NsSocUvmAxiSanitySequence
NsSocUvmRbmSanitySequence
NsSocUvmCtrlDbgIntf
ns_soc_ip (DUT)
Clocks and Reset
Interrupt
Interrupt Pin
NsSocUvmEnv
Master VIP
Vif_M1
Master Bridge (M1)
Slave Bridge (S1)
Master VIP
Vif_M2
Master Bridge (M2)
Vif_S1
Slave Bridge (S2)
Slave VIP
Slave VIP
Master VIP
Vif_RBM
RBM Bridge
Vif_S2
NsSocUvmTestConfig
Array of Master/Slave Bridge Config objects
Array of Address map configuration per Agent
Register Model generated from IPXACT

Figure 1: UVM Testbench Architecture

1.1.1 Testbench top

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

DUT (ns_soc_ip) is instantiated in this file 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, assertion fires to stop the test. This interface also holds the pin which indicates end of simulation for the test run.

UVM Test package (NsSocUvmTest.sv) is instantiated here which in turn instantiates the environment (NsSocUvmEnv) and Test Configuration (NsSocUvmTestConfig) objects.

Checkers (all interface checkers as well as e2e checker) are instantiated and bound to the respective RTL modules.

1.1.2 NoC Env

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

The config object contains all the information about the host agent (i.e., mode (AXI4/Lite/3/AHB/APB), pin widths and address range configuration)

1.1.3 NoC Test Config

NsSocUvmTestConfig class stores information about the created NoC. It keeps track of the number of agents as well as each of their 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 is passed around for environment/tests to figure out the NoC Configuration.

1.1.4 NoC Test Sequences

NsSocUvmVirtualSequence is a virtual sequence which instantiates the sequencers needed for controlling stimulus to all the agents. The test sequences in the per agent sequence library (NsSocUvmAxiMasterSeqLib.sv and NsSocUvmAhbLiteMasterSeqLib.sv) can be invoked on each of the sequencers.

1.1.5 Register Bus Sequence

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

1.2 DIRECTORY STRUCTURE

When UVM bench is generated, all the components of the 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/uvc	Static UVM files: Cadence VIP interfaces, VIP Configuration classes, address map related classes, regbus adapter and sequence library
verif/sim	Script to compile and simulate the testbench.

Table 1: NoC UVM testbench directory structure

2 NocStudio Flow to Generate 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.

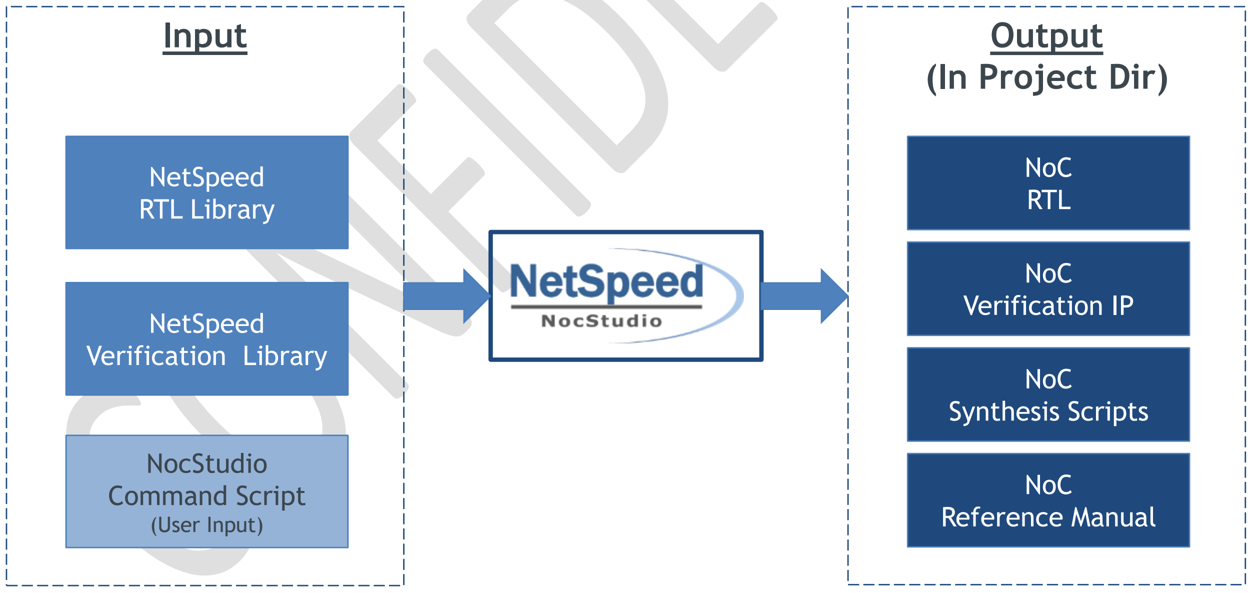

Figure 2: NoC IP generation flow

2.1.1 Example for Generating UVM Sanity Bench.

To generate NoC RTL, include “prop_default uvm_bench_enable” command along with “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. Table below shows a list of key files related to UVM Sanity testbench.

Name	Description	Type
verif/env/NsSocUvmTb.sv	Testbench module which instantiates DUT (ns_soc_ip), UVM test, and checker binds.	Verification
verif/env/NsSocUvmEnv.svh	Environment class which instantiates and configures the Cadence VIP agents.	Verification
verif/env/NsSocUvmTest.sv	Test library	Verification
verif/env/NsSocUvmTestConfig.svh	Test configuration class with information about the NoC.	Verification
verif/env/NsSocUvmVirtualSeqLib.sv	Virtual sequence (library) with all the sequencers instantiated and Sanity sequences invoked.	Verification
verif/env/NsSocUvmConfigPkg.sv	Package with Cadence configuration classes and NetSpeed-specific classes	Verification

verif/env/NsSocUvmPkg.sv	Package for all the UVM Sanity bench classes.	Verification
verif/uvc/*	Static classes for UVM Sanity bench.	Verification
verif/scripts/run_test_incisiv.sh	Wrapper to run the uvm compile and simulation script in project_directory/scripts directory.	Script

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 NetSpeed 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.

Using Cadence Incisive simulator, run:

```
run_test_incisiv.sh
```

This builds the required Cadence VIP libraries, 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 IPXACT.

On 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.

After the completion of simulation run, the presence of a file named SIM_FAILED indicates a failure in the simulation run.

Presence of file named SIM_PASSED in the sim directory after the completion of the simulation run 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.

In order to generate waveforms for incisive simulations, use command line argument "-waves=1" to run_test_incisiv.sh script.

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 Incisiv 13.20.036
- Cadence VIP : VIPCAT11.30.035-uvm

3.2 Tool setup before Compiling the Bench

Before running the uvm compile and test, the following setup needs to be done.

1) The environment variable specifying the path to Cadence VIP root must be set. The following commands are 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 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 regmodel from the generated IPXACT file. The IPXACT file is present in the project directory and named “noc_ipxact.xml”

<++> Cadence iregGen version 13.20.s004

<++> XML parsed. Started decoding

……
………………………………………………………………………………

<++> Decoding done

<++> Input File: ../..//noc_ipxact.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
```

File named "NsSocUvmRbm_rdb.sv" (and quickTest.sv) is generated by iregGen tool which holds the regmodel class. This class helps abstract 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.
- Register Read/Write tests: There is no model for register transactions.
- 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.

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 IPXACT).

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: 0x0000000000000100

- 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 will be part of Cadence's new VIP release (version after VIPCAT11.030.034)

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 of 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 LPI_REQ OUTPUT OF CADENCE LPI VIP UNDRIVEN

(Cadence Bug ID: 45914101) – 'lpi_req' output of Cadence AMBA LPI VIP undriven until first falling edge of 'aresetn', causing fatal errors upon first rising clock edge whilst reset is not asserted

5.5.1 Solution

Temporarily lowered severity of 'bad signal value' error in AMBA LPI VIP monitor from 'fatal' to 'warning', pending resolution of Cadence case:

AMBA_LPI_FATAL_ERR_CDN_AMBA_LPI300_LPI_CONTROLLER_BAD_SIGNAL_VALUE

5.6 IMG2 SLAVE VIP UNIQUE TAG IDS

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

5.6.1 Solution

This check should be disabled.

5.7 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.

5.7.1 Solution

Use address widths of at most 32 bits.

6 DOCUMENT CHANGES/REVISIONS

Documentation Changes include additions, deletions, and modifications made to this document. This section identifies the changes made in each release of the document.

6.1 DOCUMENT REVISION A

Initial Version

6.2 DOCUMENT REVISION B

- IMG2 protocol now supported
- Building of required Cadence VIP libraries now built into run script
- Low power is no longer an unsupported feature
- Added 5 new known limitations and 2 new known VIP issues

6.3 DOCUMENT REVISION C

- Separate interrupts now supported
- Removed 3 known limitations with IMG2 bridges that have been resolved
- Removed 2 known VIP issues that do not apply to currently supported tool and VIP versions
- Added 1 new known limitation, post-test quiesce period

2670 Seely Avenue
Building 11
San Jose CA 95134
www.netspeedsytems.com