VC VIP for AMBA CHI FAQ

Version W-2024.09, September 2024

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Feedback

Contents

Preface

About This Guide

This guide contains the frequently asked questions about VC Verification IP for AMBA CHI, and is for design or verification engineers who want to verify CHI operation using an UVM testbench written in SystemVerilog.

Web Resources

- Documentation through SolvNetPlus: https://solvnetplus.synopsys.com (Synopsys password required)
- Synopsys Common Licensing (SCL): http://www.synopsys.com/keys

Customer Support

To obtain support for your product, choose one of the following:

1. Go to https://solvnetplus.synopsys.comand open a case.

   Enter the information according to your environment and your issue.

2. Send an e-mail message tosupport_center@synopsys.com.

   Include the Product name, Sub Product name, and Tool Version in your e-mail so it can be routed correctly.

3. Telephone your local support center.
   - North America:

     Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
   - All other countries:

     https://www.synopsys.com/support/global-support-centers.html

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working

environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

1

CHI Frequently Asked Questions (FAQ)

What are the AXI arguments in svt_amba_system_configuration::create_sub_cfgs() and What Should I Pass to them?

Note:

Based on the AMBA Progressive Terminology updates, you must interpret the term Master as Manager and Slave as Subordinate in the VIP documentation and messages.

`svt_amba_system_configuration::create_sub_cfgs` allows user to allocate system configurations for AXI, AHB, APB, and CHI System Envs.

The Prototype of the method is:

```
void create_sub_cfgs (int num_axi_systems, int num_ahb_systems, int num_apb_systems,int num_chi_systems )
```

For example, To allocate one CHI System configuration, and three AXI System configurations within the AMBA System configuration, use as follows:

```
create_sub_cfgs(3,0,0,1);
```

How to Enable the Following?

- Snoop generation from RN

  You must run a snoop response sequence on the snoop_sequencer of RN-F. `svt_chi_rn_snoop_response_sequence` is part of the VIP sequence collection, that can be used.

  For examle, do the following in the `build_phase` of your base test

  ```
  uvm_config_db#(uvm_object_wrapper)::set(this,
  "env.amba_system_env.chi_system[0].rn*.rn_snp_xact_seqr.run_phase",
  "default_sequence",
  svt_chi_rn_snoop_response_sequence::type_id::get());
  ```

Feedback

- Response from SN

  You must run a slave response sequence on SN sequencer. `svt_chi_sn_transaction_memory_sequence` is part of the VIP sequence collection, that can be used.

  For example, do the following in the `build_phase` of your base test

```
uvm_config_db#(uvm_object_wrapper)::set(this,
"env.amba_system_env.chi_system[0].sn[0].sn_xact_seqr.run_phase",
"default_sequence",
svt_chi_sn_transaction_memory_sequence::type_id::get());
```

- *Memory in SN*

  Memory in SN is modeled using `svt_chi_memory` (extended from `svt_mem`). The SN VIP creates the memory by default. The SN response sequence must use the memory APIs to access the memory. The memory APIs that are used in the sequence are `get_read_data_from_mem_to_transaction`(), `put_write_transaction_data_to_mem()`. `svt_chi_sn_transaction_memory_sequence` already uses these memory APIs. The memory can also be created in user TB, and then passed to the SN `agent using uvm_config_db::set()`.

- Cache in SN

  SN does not have a cache. Cache is only part of RN agent, but not part of SN agent.

Flit Delay Programming

The delay is controlled through the following transaction class members:

- `svt_chi_common_transaction::txdatflitv_delay []`: Applicable for RN transaction, RN snoop transaction, and SN transaction.
- `svt_chi_common_transaction::txreqflitv_delay`: Applicable for RN transaction.
- `svt_chi_common_transaction::txrspflitv_delay`: Applicable for RN transaction, RN snoop transaction, and SN transaction.
- `svt_chi_flit::tx_flitpend_flitv_delay`
- `svt_chi_flit::tx_flit_delay`

  For more information, you can refer the HTML class reference documentation.

Feedback

Protocol Analyzer Enabling Steps for Native FSDB on VCS and IUS

- *With VCS*:

  Compile time options:

  ```
  -lca -kdb +define+SVT_FSDB_ENABLE -P
   ${VERDI_HOME}/share/PLI/VCS/${verdi_platform}/verdi.tab
  -debug_access
  ```

  For more information on how to set the FSDB dumping libraries, see Appendix B section in *Linking Novas Files with Simulators and Enabling FSDB Dumping* guide available at: `$VERDI_HOME/doc/linking_dumping.pdf`.

- *With IUS*:

  Compile time options:

  ```
  -define SVT_FSDB_ENABLE -debug_access+r
  ```

  ENV variable setting:

  ```
  setenv LD_LIBRARY_PATH ${VERDI_HOME}/share/PLI/IUS/${verdi_platform}
  ```

- *VIP Configuration*:

  ```
  svt_chi_node_configuration::enable_xact_xml_gen=1
  svt_chi_node_configuration::enable_fsm_xml_gen=1
  svt_chi_system_configuration::enable_xml_gen=1
  svt_chi_system_configuration::pa_format_type=svt_xml_writer::FSDB
  ```

Coverage Availability and Enabling

There are four kinds of coverage at port level:

- signal state coverage. This can be enabled using `svt_chi_node_configuration::state_coverage_enable`
- signal toggle coverage. This can be enabled using `svt_chi_node_configuration::toggle_coverage_enable`
- Transaction coverage (functional covergroups to cover scenarios). This can be enabled using `svt_chi_node_configuration::transaction_coverage_enable`.

  For details, check the "covergroups" tab in the HTML class reference documentation.

- protocol checks coverage (functional covergroups to cover protocol checks). This can be enabled using:
  - `svt_chi_node_configuration::pl_protocol_checks_coverage_enable`
  - `svt_chi_node_configuration::ll_protocol_checks_coverage_enable`

There is one kind of coverage at system level:

- system level protocol checks coverage (functional covergroups to cover system level protocol checks). This can be enabled using `svt_chi_system_configuration::system_checks_coverage_enable`

Cache Backdoor Example

Cache is present in RN-F agents. Cache is modeled using "`svt_axi_cache`". The following are the APIs in `svt_axi_cache`, which can be used in the testbench to do backdoor access:

```
function bit   get_addr_at_index ( input int index , output addr_t addr )
function bit   get_age ( addr_t addr , output longint age )
function int   get_any_index ( int is_unique , int is_clean , int
 low_index , int high_index )
function svt_axi_cache_line   get_cache_line ( addr_t addr )
function bit   get_cache_type ( addr_t addr , output bit [3:0]
 cache_type )
function int   get_index_for_addr ( input addr_t addr )
function int   get_least_recently_used ( int low_index , int high_index ,
 bit is_not_reserved = 1 )
function bit   get_prot_type ( addr_t addr , output bit is_privileged ,
 output bit is_secure , output bit is_instruction )
function bit   get_status ( addr_t addr , output bit is_unique , output
 bit is_clean )
function bit   invalidate_addr ( addr_t addr )
function void   invalidate_all ( )
function bit   invalidate_index ( int index )
function bit   is_partial_dirty_line ( input addr_t addr , input bit
 is_aligned_addr = 1 )
function bit   read_by_addr ( input addr_t addr , output int index ,
 output bit [7:0] data [], output bit is_unique , output bit is_clean ,
 output longint age )
function bit   read_by_index ( input int index , output addr_t addr ,
 output bit [7:0] data [], output bit is_unique , output bit is_clean ,
 output longint age )
function bit   set_cache_type ( addr_t addr , bit [3:0] cache_type )
function bit   set_prot_type ( addr_t addr , int is_privileged = -1, int
 is_secure = -1, int is_instruction = -1 )
function bit   update_status ( addr_t addr , int is_unique , int
 is_clean )
```

```
function bit   write ( int index , addr_t addr = 0, bit [7:0] data [],
 bit byteen [], int is_unique = -1, int is_clean = -1, longint age = -1,
 bit retain_reservation = 0 )
```

For example,

```
    bit [7:0]  my_data[];
    bit is_unique = 0;
    bit is_clean = 0;
    bit [43:0]   addr;
    svt_chi_rn_transaction _data;
    bit  byteen[];
    my_data = new[`SVT_CHI_CACHE_LINE_SIZE];
    byteen = new[`SVT_CHI_CACHE_LINE_SIZE];
    for (int j= 0; j < my_data.size(); j++) begin
    byteen[j] = 1'b1;
    my_data[j] = _data.data[j*8+:8];
    end
    addr = 'h100000;
    chi_system_env.rn[0].rn_cache. write(-1, addr, my_data, byteen,
 is_unique, is_clean);
```

What is Default RN-F Cache Size and How to Modify?

The default cache size is 1024 lines. This can be modified using `svt_chi_node_configuration::num_cache_lines` for that RN-F.

The default max value of cache size is 1024 lines, based on the value of macro SVT_CHI_MAX_NUM_CACHE_LINES. To increase the cache size beyond 1024, you must redefine the macro SVT_CHI_MAX_NUM_CACHE_LINES to a larger value.

Tracing/Reporting Capabilities from RN, SN, and System Monitor

For RN, SN, the tracing and reporting capabilities can be enabled using the following configurations. By default, these are disabled:

```
svt_chi_node_configuration::enable_ll_reporting
svt_chi_node_configuration::enable_ll_tracing
svt_chi_node_configuration::enable_pl_reporting
svt_chi_node_configuration::enable_pl_tracing
```

The tracing and reporting from system monitor can be enabled using the following configurations. By default, these are disabled:

```
svt_chi_system_configuration::display_summary_report
svt_chi_system_configuration::enable_summary_reporting
svt_chi_system_configuration::enable_summary_tracing
```

What Does Following Error Mean?

```
UVM_ERROR @ 41525 ns:
uvm_test_top.axitb_env.chi_master.driver.chi_system[0].chi_system_monitor
[get_mem_contents_as_byte_stream] Address 40000 of transaction {SYS_ID(0)
OBJ_NUM(0) START_TIME(1325 ns) NODE_ID(1) TYPE(WRITENOSNPPTL) TXN_ID(0)
ADDR(40000) SIZE(SIZE_4BYTE) MEM_TYPE(NORMAL)} does not fall in any
slave's range. Skipping check
```

You might have missed to specify SN to HN mapping. It can be done by calling `svt_chi_system_configuration::set_sn_to_hn_map()`. Refer the HTML class reference documentation for more information.

How to Disable Node and System Monitor Protocol Checks?

To disable all the link layer checks of an RN, SN, do:

```
svt_chi_node_configuration::ll_protocol_checks_enable=0;
```

To disable all the protocol layer checks of an RN, SN, do:

```
svt_chi_node_configuration::pl_protocol_checks_enable=0;
```

To disable a specific check of RN/SN node, you need to get the handle of that check from the VIP agent's `err_check` and then call `disable_check()` on that handle. For example, to disable the check "`signal_valid_rxsnpflitv_during_reset`" on RN[0] of a `chi_system_env`, do the following in the `end_of_elaboration_phase`:

```
svt_err_check_stats l_signal_valid_rxsnpflitv_during_reset;
l_signal_valid_rxsnpflitv_during_reset =
 env.amba_system_env.chi_system[0].rn[0].err_check.find("signal_valid_rxs
npflitv_during_reset");
if(l_signal_valid_rxsnpflitv_during_reset!=null)
 env.amba_system_env.chi_system[0].rn[0].err_check.disable_check(l_signal
_valid_rxsnpflitv_during_reset);
else
`uvm_error("end_of_elaboration_phase","could not get handle of
 signal_valid_rxsnpflitv_during_reset")
```

To disable all the system monitor checks do:

```
svt_chi_system_configuration::system_checks_enable=0;
```

To disable a specific check of chi system monitor, you need to get the handle of that check from the `chi_system_env's system_checker` and then call `disable_check()` on that handle.

For example, to disable the check "`coherent_snoop_type_match_check`", do the following in the `end_of_elaboration_phase`:

```
svt_err_check_stats l_coherent_snoop_type_match_check;
l_coherent_snoop_type_match_check =
 env.amba_system_env.chi_system[0].system_checker.find("coherent_snoop_ty
pe_match_check");
if(l_coherent_snoop_type_match_check!=null)
env.amba_system_env.chi_system[0].system_checker.disable_check(l_coherent
_snoop_type_match_check);
else
`uvm_error("end_of_elaboration_phase","could not get handle of
 coherent_snoop_type_match_check")
```

How to Generate Back to Back Transactions from CHI RN VIP to Generate Maximum Throughput for Performance Testing?

1. In RN port configuration, set the following members:

   ```
   num_outstanding_xact = 27; // You need to set this to a large value as
    per the DUT requirement
   delays_enable  = 0;
   rx_rsp_vc_flit_buffer_size  = 12; // You need to set this to a large
    value as per the DUT requirement
   rx_dat_vc_flit_buffer_size  = 12; // You need to set this to a large
    value as per the DUT requirement
   rx_snp_vc_flit_buffer_size =  12; // You need to set this to a large
    value as per the DUT requirement
   rx_req_vc_flit_buffer_size  = 12; // You need to set this to a large
    value as per the DUT requirement
   ```

2. From the RN sequence, make sure you send the consecutive transactions in a non-blocking manner. Avoid call to "`get_response()`" method. If required, call "`get_response`" in a parallel running thread (inside a fork - join_none)

3. In the transaction object that is sent from the RN sequence, follow the following conditions:

   '`txn_id`' should be unique for every transaction. Otherwise, the VIP will not drive a new transaction until the previous transaction with same `txn_id` is completed.

   '`order_type`' should be NO_ORDERING_REQUIRED.

How can the x/z checks be enabled and disabled in CHI VIP dynamically during the simulation?

CHI VIP has several checks to report `UVM_ERRORs` when it observes x/z on the CHI interface signals. These checks are generally performed after two clocks from the time when the initial reset is asserted. It is possible that some designs might take longer time to drive the signals to valid state (0/1).

Hence, you can dynamically disable and enable the checks during simulation to assist in such situations.

For example:

In base test:

```
virtual task run_phase(uvm_phase phase);
svt_err_check_stats l_signal_valid_rxdatflitpend_check,
 l_signal_valid_rxsnpflitpend_check, l_signal_valid_rxrspflitpend_check;
super.run_phase(phase);
l_signal_valid_rxdatflitpend_check =
 env.amba_system_env.chi_system[0].rn[0].err_check.find("signal_valid_rxd
atflitpend_check");
l_signal_valid_rxsnpflitpend_check =
 env.amba_system_env.chi_system[0].rn[0].err_check.find("signal_valid_rxs
npflitpend_check");
l_signal_valid_rxrspflitpend_check =
 env.amba_system_env.chi_system[0].rn[0].err_check.find("signal_valid_rxr
spflitpend_check");

if(l_signal_valid_rxdatflitpend_check == null ||
 l_signal_valid_rxsnpflitpend_check == null ||
 l_signal_valid_rxrspflitpend_check == null) begin
`uvm_error("run_phase", "l_signal_valid_rxdatflitpend_check /
 l_signal_valid_rxsnpflitpend_check /l_signal_valid_rxrspflitpend_check
 is null. Please review the err check handle retrieving mechanism")
end

//wait for one clock
@(posedge vcatb_top.vcatb_dut_wrapper.chi_if.rn_if[0].clk); //use
 appropriate xmr
`uvm_info("run_phase", $sformatf("printing the
 is_reset_active (%0b) from chi rn status object",
 env.amba_system_env.chi_system[0].rn[0].shared_status.is_reset_active),
 UVM_LOW)

//disable the checks
`uvm_info("run_phase", $sformatf("Disabling the checks
 signal_valid_rxdatflitpend_check, signal_valid_rxsnpflitpend_check,
 signal_valid_rxrspflitpend_check"), UVM_LOW)
env.amba_system_env.chi_system[0].rn[0].err_check.disable_check(l_signal_
valid_rxdatflitpend_check);
```

```
env.amba_system_env.chi_system[0].rn[0].err_check.disable_check(l_signal_
valid_rxsnpflitpend_check);
env.amba_system_env.chi_system[0].rn[0].err_check.disable_check(l_signal_
valid_rxrspflitpend_check);

//wait until reset is over
`uvm_info("run_phase", $sformatf("waiting for is_reset_active to become
 0"), UVM_LOW)
wait(env.amba_system_env.chi_system[0].rn[0].shared_status.is_reset_act
ive == 0);
`uvm_info("run_phase", $sformatf("waiting for is_reset_active to become 0
 is over.."), UVM_LOW)

//enable the checks
`uvm_info("run_phase", $sformatf("Disabling the checks
 signal_valid_rxdatflitpend_check, signal_valid_rxsnpflitpend_check,
 signal_valid_rxrspflitpend_check"), UVM_LOW)
env.amba_system_env.chi_system[0].rn[0].err_check.enable_check(l_signal_v
alid_rxdatflitpend_check);
env.amba_system_env.chi_system[0].rn[0].err_check.enable_check(l_signal_v
alid_rxsnpflitpend_check);
env.amba_system_env.chi_system[0].rn[0].err_check.enable_check(l_signal_v
alid_rxrspflitpend_check);

endtask
```