Slave TAP (sTAP)

VERIFICATION PLAN

IP Rev. PIC6_V1
January 2020

Intel Top Secret

Deleted:

Deleted: 5

Deleted: 5

Deleted: {x.x}

Deleted: August

Deleted: March

Deleted: 19

Contents

1 About This Document

1.1 How to Use This Template

Please do not remove any headings from this document. If you do not need the headings to describe your IP or subsystem, enter “Not applicable” under the heading. This lets the reader know that we did not overlook this topic.

In the main document that follows, add new headings that you need to fully describe the architecture of this IP or subsystem. (Please add them in the appropriate chapters.)

Note: Most red text in this document contains instructions for filling out the section where it appears. The tag for most of this red text is called “Gaps.” You should replace this text with the content appropriate for that section, ensuring that the text is tagged appropriately (for example, with the BodyText or List Bullet style). If a section is not relevant, do not remove it, but just replace the Gap text with “Not applicable” and apply the BodyText style.

1.2 Goal of This Document

This document should contain all information a verification team needs to accomplish the IP or subsystem verification task without needing to seek help from another source. Try not to refer to other documents for required information; do so only if you include specific instructions for obtaining those documents, and only if you are sure your audience has access to them. Verify all links.

1.3 Audience

The information in this document is intended to describe the verification strategy and plans for this IP.

It is written as a Test Plan for the IP verification team, describing how the IP *is going to be* tested. The same document is published as a Verification Reference for an SoC team to see how the IP *was* tested.

1.4 Supported Projects

Modify this table as needed.

This document supports the following projects at the listed RTL maturity level. Type “N/A” if this IP or subsystem is not included in a specific project, or remove those project names from the table.

Table 1. RTL Support Level for SoC Projects

Project Name	**IP Maturity Level**
TGL	
MCC	
SPR	
RYF	
ADL	

1.5 Document Revision History

Author	Revision No.	Description	Revision Date
	Rev0.1		
	Rev0.5		
	Rev0.8		
Shivaprashant Bulusu	Rev1.0	1st Revision	
Sudheer V Bandana	1.0 v7	Revision with HDK update	

1.6 Contact Information

Table 2.

Name	Function	Email
Bulusu, Shivaprashant	Verification	shivaprashant.bulusu@intel.com
Bandana, V, Sudheer	Verification	sudheer.v.bandana@intel.com

1.7 Related Documents

If you need more information on this IP, you may find these documents helpful:

- sTAP Architecture Description
- sTAP Reference Manual
- sTAP Integration Guide

2 Overview

2.1 IP Description

The JTAG 1149.1 specification is an industry standard test interface originally intended for boundary scan testing of the interconnect board traces between integrated circuits on a motherboard. It continues to provide IO pin connectivity testing between components and it there several additional 1149.x standards to address more diverse IO topologies and test techniques. Another important use for the 1149.1 TAP is the serial communication protocol to access private registers for testing and debugging SoC components.

The sTAP is partitioned into many FUBs in order to separate out the parts that to be configurable for implementation in an SoC or other IPs and the parts that will remain fixed.

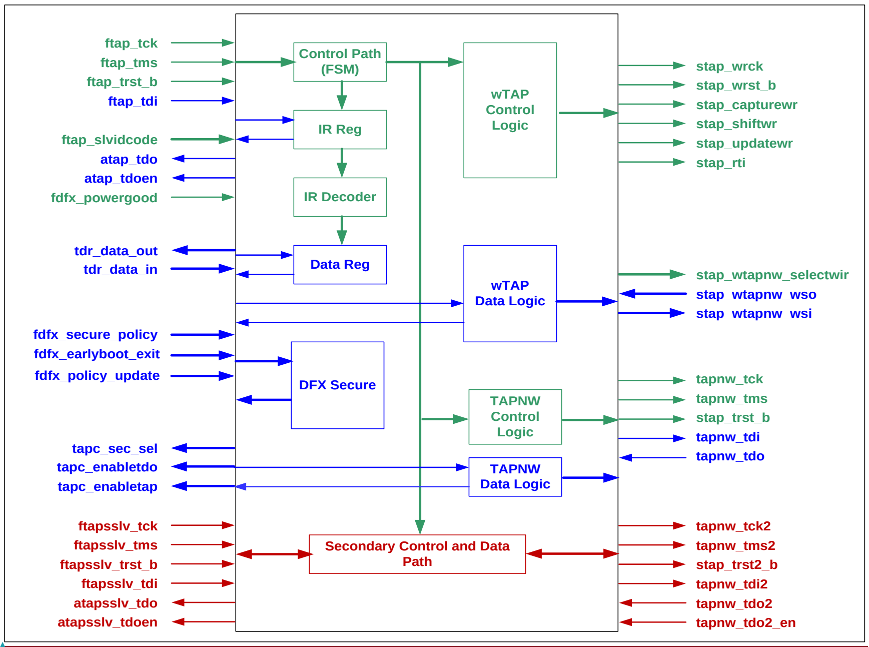
The sTAP consists of following modules.

- sTAP: the top module, which instantiates the following nine modules:
  - stap_irreg: generates parallel instruction output from TDI
  - stap_irdecoder: instantiates the following sub module:
    - stap_decoder: a generic configurable decoder module
  - stap_drreg: instantiates the following sub modules:
    - stap_data_reg: a generic configurable register module
    - stap_remote_data_reg: This module provides control signals for data registers that are placed outside the sTAP module.
  - stap_tdomux: controls the data flow from the data and instruction registers
  - stap_fsm: controls the operation of the TAP
  - stap_tapnw: generates the required control signals for 0.7 networks
  - stap_wtapnw: generates required control signals for wTAP networks
  - stap_glue: contains out-of-band control and data signals logic that connects various modules
  - stap_bscan: generates boundary scan signals

Deleted: Briefly describe the IP or subsystem to be verified and any notable features. The description should be limited to 1-3 paragraphs. Include a high-level block diagram showing major functional blocks and/or demonstrating the relationship of the design to adjacent SoC-level design blocks, if applicable. Any supporting IP sub-components that are delivered for integration into this IP for verification should be clearly identified.¶

2.2 Block Diagram

Figure 1. Module sTAP block diagram



2.3 Testbench Description

The Verification ENV is in OVM methodology. The Testbench architecture is illustrated below. The exact files for each block are provided in the diagram. The Env class is extended from the ovm_env. It instantiates all the OVM components: master agent, slave agent, Scoreboard, Coverage, and OVM report.

2.4 Verification Scope

The usage of this verification environment is primarily to validate the IP. When used at the SoC level, the JTAGBFM was used to configure all the registers. This document outlines how to use the IP at SoC level.

2.5 Dependencies/Assumptions

It is assumed that all subIPs comes as a fully validated subIP. Thus, the testbench does not specifically target coverage of all subIPs.

Formatted: Heading 2

Field Code Changed

Deleted: Briefly describe the testbench topology at a high level. The description should be limited to between 1 to 3 paragraphs. A block diagram of the testbench and IP should be included, demonstrating how components connect or interact. The diagram or description should also clearly identify all interfaces by which the IP or subsystem RTL is stimulated, which interfaces are monitored, and how the checkers are connected to the monitors. Any components that are reused from external sources for IP or subsystem verification should be clearly identified.¶

Deleted: Describe which features of the IP or subsystem will be verified.¶

Deleted: ~~Document any dependencies or assumptions used when determining the verification scope. This may include dependencies on the availability or health of leveraged collateral or assumptions/agreements about verification responsibilities between teams regarding parts or features of the design.¶~~

3 Verification Environment

3.1 SoC-Specific Validation

Verification areas that cannot be tested by the IP/IPSS and have to be tested by the SoC. Call out high-risks and assumptions.

Deleted: Use these sections to describe the verification testbench.¶
¶

Deleted: Verification areas that cannot be tested by the IP/IPSS and have to be tested by the SoC. Call out high-risks and assumptions.¶

3.2 Validation Parameters

This section is documented in the Integration Guide (located in the 'doc' directory for this release).

3.3 Verification Libraries

This section is documented in the Integration Guide (located in the 'doc' directory for this release).

3.4 Testbench Components and Connectivity

This section discusses major validation structures included in drop, how they connect to the DUT, and how they're used.

The following subsections are documented in the Integration Guide (located in the 'doc' directory for this release):

- Testbench Directory Structure
- Test Islands
- Test Island Interfaces
- Checkers and Trackers
- Monitors
- Scoreboards
- BFMs
- Collage or Sandbox Files

3.5 IP/IPSS Environment

Refer to Integration Guide section Testbench Overview and section TestIsland. There is no Usage of Saola (included in the 'doc' directory for this release).

3.5.1 Environment Files

Refer to Verif/tb/ and Verif/tests/ for Environment files and description.

Deleted: sections x-x

3.5.2 Configuring the IP/IPSS Environment

Not applicable to this IP/IPSS

3.5.3 Saola Environment Walkthrough

Not applicable to this IP/IPSS

Deleted: [example] The Saola environment in PCIe2 consists only of RAL (Register Abstraction Layer). RAL, as its name suggests, consists of PCIe2 configuration registers and their attributes.

Formatted: No bullets or numbering

Deleted: Following are the components of RAL:¶
File ...

3.5.4 Saola/RAL Components

Not applicable to this IP/IPSS; none of the RAL components are used for this IP.

3.5.5 System Manager

Not applicable to this IP/IPSS

3.5.6 Fuse

Not applicable to this IP/IPSS

3.5.7 Safety Designer / Verifier

Not applicable to this IP/IPSS

Deleted: Example: Safety Designer.Verifier environments are available in verif.tb.safety

Safety Designer Compnents	Description	SoC Recommendations	Required?

Safety Verifier Compnents	Description	SoC Recommendations	Required?

3.6 Sequences

Sequences are located at: `$IP_ROOT/verif/tb/STapSeqeunces.sv`.

For stimulus-related sequences, see section 7.7, Sequencers / Sequence Drivers—Example.

3.6.1 Sequence for Bringing up the IP/IPSS

SlvIdcode and Bypass sequences are used for Bringing up the IP.

3.6.2 BFM Sequences

Sequence Name	Description	Parameters	Saola Phase
MultipleTapRegisterAccess	1st argument: ResetMode This 2-bit mode defines the way the user wants to reset the TAP. 2’b00: No reset. 2’b01: Asserts the reset_b 2’b10: Enables the TMS for 5 clock cycles. 2’b11: Asserts powergood_rst_b 2nd argument: Address: The address or instruction Opcode. 3th argument: Data: The data that needs to be loaded in the selected register. 4th argument: Address_Length: The address width. 5th argument: Data_Length: The data width.	ResetMode, Address, Data, Address_length, Data_length	Not used
ExpData_ MultipleTapRegisterAccess	1st argument: ResetMode This 2-bit mode defines the way the user wants to reset the TAP. 2’b00: No reset. 2’b01: Asserts the reset_b 2’b10: Enables the TMS for 5 clock cycles. 2’b11: Asserts powergood_rst_b 2nd argument: Address: The address or instruction Opcode. 3rd argument: addr_len: The address width. 4th argument: Data: The data that needs to be loaded in selected register. 5th argument: Expected_Data: The data that is expected to come out. 6th argument: Mask_Data: The field of Expected_data that needs to be compared with the field of Data. 7th argument: Data_len: The data width.	ResetMode, Address, addr_len, Data, Expected_Data, Mask_Data, Data_len	Not used

Sequence Name	Description	Parameters	Saola Phase
ReturnTDO_ExpData_ MultipleTapRegisterAccess	1st argument: ResetMode This 2-bit mode defines the way the user wants to reset the TAP. 2'b00: No reset. 2'b01: Asserts the reset_b 2'b10: Enables the TMS for 5 clock cycles. 2'b11: Asserts powergood_rst_b 2nd argument: Address: The address or instruction Opcode. 3rd argument: addr_len: The address width. 4th argument: Data: The data that needs to be loaded in selected register. 5th argument: Expected_Data: The data that is expected to come out. 6th argument: Mask_Data: The field of Expected_data that needs to be compared with the field of Data. 7th argument: Data_len: The data width. 8th argument: ReturnedTDO: TDO that comes back. This should be a locally declared variable in the test. Usage is shown in Code 7.	ResetMode, Address, addr_len, Data, Expected_Data, Mask_Data, Data_len, Returnedtdo	Not used

3.6.3 IOSF Primary/Sideband BFM Sequences

Not applicable to this IP/IPSS

3.6.4 Other Reusable Sequences

Not applicable to this IP/IPSS; sequences are not re-usable as the SoC Hierarchy determines the test content.

3.6.5 IP/IPSS Test Sequences

Refer to 1 for IP Test Sequences.

Deleted: Table 5

3.6.6 SoC Requirements for Sequence Reuse

Not applicable to this IP/IPSS; this is the parameterized IP.

3.6.7 Sequence File Dependencies

Not applicable to this IP/IPSS; all sequences are used from the JTAG BFM.

3.6.8 Sequence Writing

Table 3. sTAP Test Sequences

Test Sequence Name	Test Sequence Function
TapSequenceTry	A spare sequence for development teams to try out various customer requested features.
TapSequenceBypass	Bypass Sequence: places the sTAP in bypass mode and transfers the data.
TapSequenceMultipleTapRegisterAccess	Register Access: accesses all the registers present in the sTAP.
TapSequenceNetworkRegisterAccess	Accesses the wTAP network and 0.7 TAP NW registers and then accesses the data registers.
TapSequenceBoundaryRegisterAccess	Accesses the Boundary Scan registers.
TapSequenceFsmStates	Transits all the FSM states and checks that the register has the right value.
TapSequenceTmsGlitch	Produces a glitch on the ftap_tms port in TLRS (Test Logic Reset) state and then asserts ftap_tms high for three cycles.
TapSequenceRandom	Randomly chooses various opcodes in the design, writes random data, and then reads it back.
TapSequenceRemoteTDR	Performs a write followed by a read to different remote TDRs.
TapSequenceBoundaryRegisterAccessWoReset	Due to a known limitation in Scoreboard when testing Boundary Scan, in order to consecutively access a register twice without a reset in between, you must disable Scoreboard. In this situation, disable Scoreboard and use the Expect_Data API.

3.7 Using the Runtime or Post-Processing Checkers

Assertions in the run phase are complaint with IEEE1149.1/.7. At the end of the simulation, the checker writes a `JtagBfmTracketPri.out` file containing the log of the various transactions executed on the DUT to the `$IP_ROOT/pwa/results/tests/<test_name>` directory.

3.8 Environment Settings and Files

Defines are available in the below path:

```
$IP_ROOT/source/rtl/include
```

The recommended approach is to instantiate the sTAP Env using type-based factory method, as shown below:

```
// Instantiating the TAP Env: Type-Based Factory Approach
// sTAP ENV TapBaseTest
// From file TapBaseTest.sv

class TapBaseTest extends ovm_test;

   // Factory Registration
   `ovm_component_utils(TapBaseTest)

   TapEnv Env;
      TapReportComponent ReportObject;
   ovm_report_handler ReportComponent;
   // Using type-based factory method
   Env = Env::type_id::create("Env", this);

   ...
endclass
```

Here is a code snippet of the actual test. First, in order to emulate the Debug Life Cycles States, the sequencer of DFx Secure Plugin Agent is used to drive the SECURITY_LOCKED policy on the policy bus. Next, the fdfx_powergood is asserted using the JTAG BFM sequencer to remove the TAP out of reset. Then the SECURITY_UNLOCKED sequence is run to set the correct set of features that are visible either to Intel, selective customer or to all. Then the actual test case sequence is executed.

```
class TapTestBypass extends STapBaseTest;

    `ovm_component_utils(TapTestBypass)
    SecurityLocked    SL;
    PowergoodReset    PG;
    SecurityUnlocked  SUL;
    TapSequenceBypass SP;

    function new (string name = "TapTestBypass", ovm_component parent = null);
        super.new(name,parent);
    endfunction : new

    virtual function void build();
        super.build();
    endfunction : build

    virtual task run();
        ovm_report_info("TapTestBypass","Test Starts!!!");
        SL  = new("Security Locked Mode");
        PG  = new("Powergood reset");
        SUL = new("Security Unlocked Mode");
        SP  = new("ALL Primary Mode");

        //PG.start(Env.stap_DfxSecurePlugin_Agent.i_DfxSecurePlugin_Seqr);
        SL.start(Env.stap_DfxSecurePlugin_Agent.i_DfxSecurePlugin_Seqr);
        PG.start(Env.stap_JtagMasterAgent.Sequencer);
        SUL.start(Env.stap_DfxSecurePlugin_Agent.i_DfxSecurePlugin_Seqr);
        SP.start(Env.stap_JtagMasterAgent.Sequencer);
        ovm_report_info("TapTestBypass","Test Completed!!!");
        global_stop_request();
    endtask : run
endclass : TapTapBypassTest
```

By default, all assertions are turned on. Pass INTEL_SVA_OFF switch to turn it off .

3.8.1 Base Test

Top-Level Environment "STapEnv" is instantiated in Base Test Class. Also contains Configuration Parameters to configure using the "set_config_int" method.

```
class STapBaseTest #(`STAP_DSP_TB_PARAMS_DECL) extends ovm_test;

    STapEnv #(`STAP_DSP_TB_PARAMS_INST) Env;
    STapReportComponent ReportObject;
    ovm_report_handler  ReportComponent;


    // Constructor
    function new (string name = "STapBaseTest", ovm_component parent = null);
        super.new(name,parent);
        ReportComponent = new;
        ReportComponent.set_max_quit_count(2);
        ReportObject = new("ReportObject" , this);
        ReportObject.set_report_verbosity_level_hier(OVM_NONE);
    endfunction : new

    // Register component with Factory
    `ovm_component_param_utils(STapBaseTest #(`STAP_DSP_TB_PARAMS_INST))

  ////  Virtual pin Interface for connection to the ENV
  //virtual stap_pin_if pin_if;

    // Build
    virtual function void build();
         super.build();
        set_config_int("Env", "has_scoreboard",    1);
        set_config_int("Env", "has_cov_collector", 1);
        set_config_int("Env", "quit_count", 2);
        set_config_int("Env", "set_verbosity", OVM_DEBUG);
        set_config_int("Env", "enable_clk_gating", OVM_ACTIVE);
        set_config_int("Env", "park_clk_at", OVM_PASSIVE);
        // Hassan This will stop random sequences from starting at their own
        set_config_int("Env.stap_JtagMasterAgent.Sequencer", "count", 0);
        set_config_int("Env.stap_DfxSecurePlugin_Agent.i_DfxSecurePlugin_Seqr",
"count", 0);
        Env = STapEnv#(`STAP_DSP_TB_PARAMS_INST)::type_id::create("Env", this);
        // Enabling runtime tracker
        set_config_int("*", "jtag_bfm_tracker_en",        1);
        set_config_string("*", "tracker_name",         "sTAP");
        set_config_int("*", "jtag_bfm_runtime_tracker_en",1);

    endfunction : build

    //Connect Pin Interface to The ENV
    //function void connect();
    //     Env.assign_vi(PinIf);
    //endfunction: connect

    // Run Task
    virtual task run();
         ovm_report_info("STapBaseTest","Test Starts!!!");
         ovm_report_info("STapBaseTest","TapBase Test Completed!!!");
         global_stop_request();
    endtask : run

endclass : STapBaseTest
```

3.8.2 Configuration Object

Not applicable to this IP/IPSS

3.8.3 API

Not applicable to this IP/IPSS

3.9 Description of Reusable Tests

Not applicable to this IP/IPSS

3.10 Description of Reusable Automation Scripts

Refer to the Integration Guide, Generating COnfigutaion Parameters, for more information (included in the 'doc' directory for this release).

3.11 Supported Compiler Options for Simulation

The table below summarizes the supported options.

Deleted: Describe any aspects of the simulation of this IP/IPSS that can be controlled using compiler options. Note debug switches, etc.¶

Argument	Input	Example

The supported options follow standard compiler options available. (+FSDB, +verbosity etc) so it is not specifically called out here.

3.12 Reusable Simulation RUNMODEs

Not applicable to this IP/IPSS

Formatted: Font: 9 pt, Not Expanded by / Condensed by

Formatted: BodyText

Deleted: Describe reusable RUNMODEs here. In Ace, RUNMODEs can be set up to configure elements of the simulation environment. RUNMODEs can be anything -- specifying a model, setting testbench parameters that can impact configuration, traffic/checker enables, etc. RUNMODEs for IP/IPSS-level testing that can be used at the SoC level should be provided and described here.

RUNMODE	Description

3.13 Testbench Tools

Deleted: Describe tools and versions used by the verification testbench. Notes such as tool sources, or forward/backward compatibility in tool versions may be included. Testbench components such as BFMs may have versions that should also be documented.

Formatted Table

Table 4. Testbench Tools

Tool	Version
Operating System	CPU=x86_64
Simulator	VCSMX P-2019.06-SP1-1
Verification Language	OVM 2.1.1_2
RTL Language	System Verilog
ESL Language	NA
Bug Reporting system	HSDES
Source Control Tool	Git
User Simulation Area	IRR Download Path
Run/Build Script	make run_vcs CUST=<CUST>
Regression Script	scripts/run_all_configs

Deleted: J-2014.12-SP3-2-B-2*

Tool	Version
Results Reporting Script	Observe Log file at the end of simulation
Coverage Report Script	Scripts/get_coverage

```
All the tools are sourced from customer toolfile.dat
```

3.14 Testbench Utilities for Address, IP State, and Memory

Not applicable to this IP/IPSS

3.15 Simulation Stages

The testbench has all of the sequences in OVM; there is no phasing explicitly called out for 0p5.

3.16 Environment Setup and Test Run

Describe the commands and steps needed to set up the environment and run a test.

1.1.1 Downloading the IP

To setup release flow:

```
       kinit  (then enter password)
                wash -n -g soc socrtl hdk22nm soc73 sbx socenv <*for IP Release usage add Unix group for your IP Library in IPX*> <*also as needed add unix group protecting you directory/WARD*>
       /p/hdk/bin/setproj -p siphdk -cfg SBXA0P05
       setenv ipxclient_ALT0 prod; setenv IPX_PH1_ALT0 1
       setenv ship_ALT0 19.02.020; setenv ship_proj_ALT0 IPXPilot.p003; setenv ip_release_ALT0 19.02.019; setenv NOBLE_HIP_FLOW_ALT0 IPXPilot.p023;setenv indicator_ALT0 IPXPilot.p002
       setup -d -t ip_release -b none -ath ship -at ipx_services -ov <path to your stod>/<work area name e.g. ipx-sbx-ww07>
       setenv PATH "$IPXPATH":"$PATH"; pi login $USER; p4 login -a
```

To download the IP:

pi ip load dteg_tap.stap@1.PIC56

1.1.2 Environment Setup

1. Wash unwanted groups:

   ```
   wash -n dfxsip sip soc siphdk dk10nm hdk10nm hdk10nmproc soc71proc dk1273 soc73 soc73proc cdftsip cdft
   ```

2. Source the HDK env:

   ```
   source /p/hdk/rtl/hdk.rc -cfg sip
   ```

3. This is the command if want to resource the env on the same shell:

   ```
   source /p/hdk/rtl/hdk.rc -cfg sip -reentrant
   ```

Formatted Table

Deleted: Tool ...

Formatted: Code Snippet

Deleted: Describe any address, state, or global memory management implemented in the testbench, if applicable. These can include testbench components that contribute to address generation for stimulus, mechanisms to track state (such as current configuration) for stimulus or checking, and/or memory images used to check for corrections or generate appropriate responses to transactions. If independent reference models of the IP are simulated in conjunction with the IP or subsystem for stimulus or checking, describe details here. If memory models are maintained, describe backdoor mechanisms used to preload or access memory images.¶

Deleted: Describe the simulation stages implemented in the testbench. The description can include high-level documentation of test-run phases, control mechanisms and hooks for advancing between phases, how test end is determined, and how run status is determined and communicated. Where applicable, use tables to document testbench control hooks related to simulation stages.¶
¶

Formatted: Code Snippet

Deleted: 2

Formatted: Default Paragraph Font

1.1.3 Commands for Test Simulation, Verdi, Regression

Defines are available in the path below:

```
$IP_ROOT/source/rtl/include
```

The recommended approach is to instantiate the STAP Env using type-based factory method, as shown below:

```
// Instantiating the TAP Env: Type-Based Factory Approach
// STAP ENV TapBaseTest
// From file TapBaseTest.sv

class TapBaseTest extends ovm_test;

   // Factory Registration
   `ovm_component_utils(TapBaseTest)

   TapEnv Env;
      TapReportComponent ReportObject;
   ovm_report_handler ReportComponent;

   // Using type-based factory method
   Env = Env::type_id::create("Env", this);

   ...
endclass
```

Here is a code snippet of the actual test. First, in order to emulate the Debug Life Cycles States, the sequencer of DFx Secure Plugin Agent is used to drive the SECURITY_LOCKED policy on the policy bus. Next, `fdfx_powergood` is asserted using the JTAG BFM sequencer to remove the TAP out of reset. Then the SECURITY_UNLOCKED sequence is run to set the correct set of features that are visible either to Intel, selective customer or to all. Then the actual test case sequence is executed.

```
class TapTestBypass extends stapBaseTest;

    `ovm_component_utils(TapTestBypass)
    SecurityLocked    SL;
    PowergoodReset    PG;
    SecurityUnlocked  SUL;
    TapSequenceBypass SP;

    function new (string name = "TapTestBypass", ovm_component parent = null);
        super.new(name,parent);
    endfunction : new

    virtual function void build();
        super.build();
    endfunction : build

    virtual task run();
        ovm_report_info("TapTestBypass","Test Starts!!!");
        SL  = new("Security Locked Mode");
        PG  = new("Powergood reset");
        SUL = new("Security Unlocked Mode");
        SP  = new("ALL Primary Mode");

        //PG.start(Env.stap_DfxSecurePlugin_Agent.i_DfxSecurePlugin_Seqr);
        SL.start(Env.stap_DfxSecurePlugin_Agent.i_DfxSecurePlugin_Seqr);
        PG.start(Env.stap_JtagMasterAgent.Sequencer);
        SUL.start(Env.stap_DfxSecurePlugin_Agent.i_DfxSecurePlugin_Seqr);
        SP.start(Env.stap_JtagMasterAgent.Sequencer);
        ovm_report_info("TapTestBypass","Test Completed!!!");
        global_stop_request();
    endtask : run
endclass : TapTapBypassTest
```

By default, all assertions are turned on. You can turn off the assertions by passing the switch INTEL_SVA_OFF

4. Navigate to the directory where you downloaded the IP package.

5. To compile the model:

   Please do setenv `JTAG_BFM_VER <Download path of JTAGBFM>` in `cfg/ace/templates/custom_post.env.template`

   ```
   make run_vcs CUST=<CUST>
   ```

6. To run simple tests:

   a. To run a basic test:

      ```
      make run_vcs_test CUST=<CUST>
      ```

   b. To run the sample IP-level test (included in the release):

      ```
      make run_vcs_test CUST=<CUST> TESTCASENAME=<testcasename>
      ```

   c. To run the test interactively:

      ```
      make run_vcs_test_GUI CUST=<CUST> TESTCASENAME=<testcasename>
      ```

Deleted: Describe the commands and steps needed to set up the environment and run a test.¶

3.17 Testbench Output

Simregress is used to run test content and the output of the runs is placed in folder: `$IP_ROOT/regression/default/<regression_list>/<test_name>/acerun.log`

Full OVM verbosity is enabled with:

```
+OVM_VERBOSITY=OVM_FULL
```

Simregress is used to run test content and the output of the runs is placed in folder: `$IP_ROOT/regression/defaault/<regression_list>/`

Deleted: Describe the kinds of output the testbench produces. Include details such as where output is created (like run directories and logfiles), how the output is enabled/disabled, and details on verbosity control, if applicable. Provide information on how to interpret messages if the output format might be unclear to some users.

4 Verification Strategy

The IP verification strategy focuses on IP features which are easily testable standalone at the IP level with the minimal testbench capabilities within that environment. It is intended to be supplemented with SoC level use model validation tests. The IP level tests are design to cover 100% of the features and achieve full feature, line, toggle, branch and FSM coverage of the IP RTL.

4.1 High-Level Verification Strategy

Deleted: Describe the pre-silicon strategy and high-level methodologies used to verify the IP or subsystem. Keep the entire section brief and, if possible, limited to between 2 and 5 pages. Describe lower-level details in the later sections.¶
¶

4.1.1 Methodology

The high level strategy is to target read write tests of each register in the deisgn, target features of the design, analyze coverage and target any missing areas of coverage to achieve 100% toggle, line, branch and FSM coverage. All content is writen using OVM sequences following OVM-Saola methodology. Cases are self checking and include the expected response.

Deleted: Describe the high-level verification methodology. Include type(s) of verification environment(s) that will be used to verify the IP or subsystem (such as n-val simulation, emulation, FPV, and the like.). Identify the primary verification testbench methodology used in verification (such as OVM-Saola).¶

Formatted: Font: 9 pt, Not Expanded by / Condensed by

4.1.2 Stimulus Strategy

All stimuli are applied from the OVM Driver from JTAG BFM, IOSF BFM, and Fuse VC. Test content is captured in sequences extend from OVM sequence class and are passed to the OVM driver as a transaction item.

- Fuse Puller key is currently random stimulus; it can also be applied in a directed stimulus to target corner cases.
- DRNG stimulus is currently directed but there is a provision to apply either Directed or Random Stimulus.
- TAP Stimulus is directed
- IOSF Stimulus is directed stimlus from Standard IOSF Primary VC which includes full compliance checker for IOSF spec, which takes are of most of the validation checks as fas as IOSF protocol is conserned.

There is no other means of generating stimulus other than the Driver which receives the commands from the test case.

Deleted: Describe the RTL stimulus strategy at a high level. If applicable, include answers to questions such as:¶
Are sequences, sequence libraries, test templates, or test lists used for verification, and if so, how do they work together? ¶
Is stimulus directed or random? ¶
Is stimulus manually coded or auto-generated by a tool? ¶
Are there other mechanisms for providing stimulus such as RTL injectors or CR access tools? ¶
Is any other collateral required to generate stimulus? ¶
Document all major mechanisms for generating stimulus, but not in great detail (that is, actual sequences are not to be listed here.)¶

4.1.3 Coverage Strategy

Code Coverage is measured with Line Coverage, Conditional Coverage, Branch Coverage, and FSM Coverage. Functional coverage is measured with data bins. Both Functional coverage and Code coverage are generated with using "urg" command.

Run command to get Coverage:

```
make run_script SCRIPT='source scripts/get_coverage' CUST=<CUST_NAME>
```

4.1.4 Checking Strategy

The primary checking for the various tests is self-checking tests.

Scoreboard is enabled for IP checking. Testbench is also leveraging checking from BFMs to complement the checking solution. Many tests are self checking having expected output in test sequnece.

Deleted: Describe the high-level checking strategy used to determine the functional correctness of the IP or subsystem. If applicable, include answers to questions such as:¶
Is dynamic/runtime checking done or is it a post process? ¶
Are scoreboards or other testbench checking components used to verify correctness? ¶
Are assertions used in the RTL to detect and flag illegal behavior? ¶
Will external checking collateral be leveraged to complement the checking solution (that is, BFM checking or compliance monitors)? ¶
Are any other mechanisms used for checking? ¶
Document all major mechanisms for IP verification checking of the IP, but not in great detail (you will give more details on checking in a subsequent section).¶

4.1.5 Formal Verification Strategy

Not applicable to this IP/IPSS; formal verification is not set up for the IP.

4.1.6 Debug Strategy

Not applicable to this IP/IPSS; formal verification is not set up for the IP.

4.1.7 Security Strategy

The Scoreboard subscribes to the analysis ports of the JTAGBFM (Input and Output Monitors) as well as stap (Input and Output Monitors) to get transactions for processing. Various scenarios in the test environment are checked to see if they are covered.

4.1.8 Safety Verification Strategy

4.1.8.1 Safety HW Feature Verification Strategy

Refer to stap HAS Chapter 17 (included in the 'doc' directory for this release).

4.1.8.2 Safety SW Feature Verification Strategy

Refer to stap HAS Chapter 17 (included in the 'doc' directory for this release).

4.2 Verification Strategy for Areas of Special Emphasis

Not applicable to this IP/IPSS

4.2.1 Reset Verification

The IP has two resets: powergood reset and sideband reset. Resets are driven through Initial Block in TOP testbench. There is no verification for Reset Mechanism. There is one reset for CCU BFM present which is also driven from TOP Testbench.

4.2.2 Control Register and Fuse Verification

Not applicable to this IP/IPSS

4.2.3 Power Management Verification

Power management verification Validated in UPF test case.

4.2.4 Mixed Signal Verification

Not applicable to this IP/IPSS

4.2.5 Performance Verification

Not applicable to this IP/IPSS

Deleted: Describe the use of formal verification and the role it plays in the overall IP or subsystem verification strategy, if applicable. Identify which portions of the design are to be verified, and the tools or methods that are used.¶
¶

Deleted: Describe the verification debug strategy. Describe any debug tools, testbench components, failure management tools, workaround methods, or other collateral that is used or will be available to contribute to pre-silicon debug of the IP or subsystem RTL.¶
¶

Deleted: Describe the verification security strategy. Identify which portions of the design that will be verified by negative, focused, and/or fuzz testing, and which tools are to be used. Also specifiy any specific attack scenarios that will be used to validate a mitigation.¶
¶

Deleted: Describe Safety verification strategy in IP Hardware design. Identify which portions of the design that will be verified by negative, focused, and/or fuzz testing, and which tools are to be used/ Also specify any specific attack scenarios that will be used to validate a mitigation.¶

Deleted: Describe Safety verification strategy in IP Sotware design. Identify which portions of the SW that will be verified by negative, focused, and/or fuzz testing, and which tools are to be used/ Also specify any specific attack scenarios that will be used to validate a mtigation.

Deleted: <#>¶

Deleted: Describe the verification strategy for areas of special emphasis, where applicable. Although these areas may be verified as part of the overall functional verification strategy, list them separately so that unique requirements are clearly identified and the verification strategy is documented and communicated. These areas may include, but are not limited to, the subsections listed below. Each section should describe the applicable verification requirements of the IP or subsystem and how pre-silicon stimulus, checking, and coverage (and the like) will be used to address those requirements.¶

Formatted: Font: 9 pt, Font color: Auto, Not Expanded by / Condensed by

4.2.6 Security Verification

Registers are secured with tap color.

4.2.7 Safety Verification

Not applicable to this IP/IPSS

4.2.8 Error Scenario Verification

Not applicable to this IP/IPSS

4.2.9 Design for Test (DFT) Verification

Not applicable to this IP/IPSS

4.2.10 Design for Validation (DFV) Verification

Not applicable to this IP/IPSS

4.2.11 Firmware Verification

Not applicable to this IP/IPSS

4.2.12 Software / Driver Verification

Not applicable to this IP/IPSS

4.2.13 Timer / Counter Verification

Deleted: Ensure all timers and counters have a cover point for the overflow/expire condition. List the timers/counters and their validation strategy. List any dependent speedup modes, etc., required to perform this validation.¶

4.3 Reuse Strategy

Not applicable to this IP/IPSS

4.3.1 Local Reuse for IP Verification

Deleted: Briefly describe the strategy for reuse. This includes both the reuse of sub-IPs or verification collateral within this IP's or subsystem's verification environment, as well as the reuse strategy of the verification collateral as it is integrated into the SoC.¶
¶

4.3.2 Reuse of IP or Subsystem Verification Collateral for SoC

At the SoC level for chipset, which is in ACE environment, the IP verification can be used directly.

In the SoC level for server, test development will be in SPF format which needs support for DTS/DTEG automation group.

For more information, refer

to section 6.5 in Integration Guide (included in the 'doc' directory for this release).

Deleted: ¶
IP-Level Information Required for Sequence Writing¶
sTAP Test Sequences¶
Test Sequence Name ...

5 Flows

5.1 Bring-up Flow Details

Refer to the STAP HAS section 3 (included in the ‘doc’ directory for this release).

5.2 Linkup/Down Flow Details

Not applicable to this IP/IPSS

5.3 Reset Flow Details

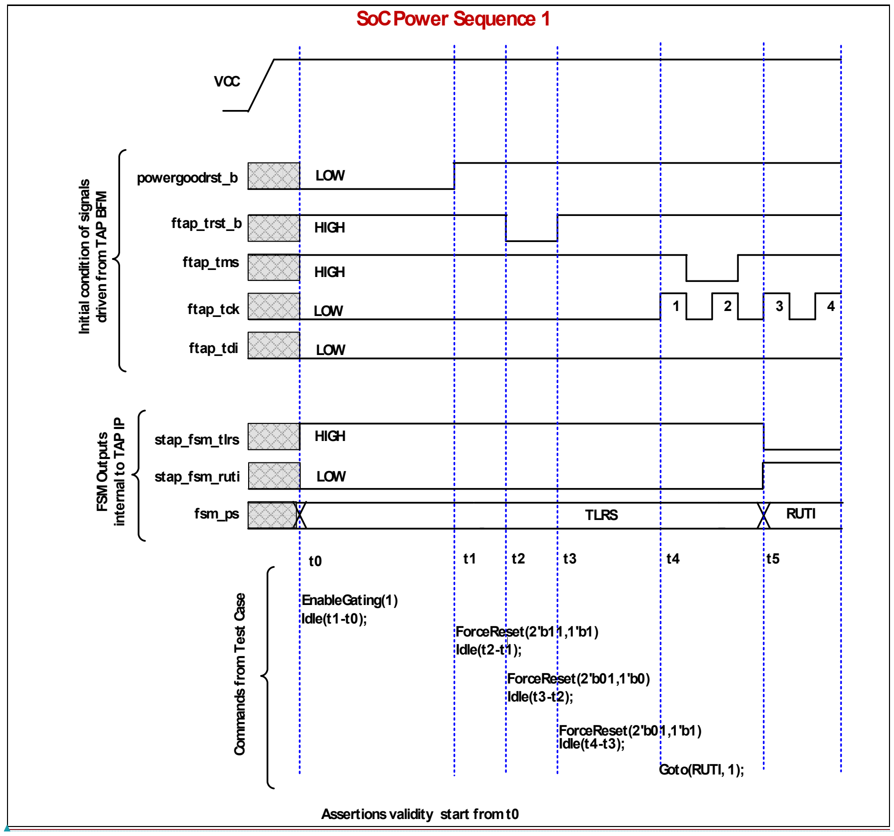
The BFM provides tasks that let you create a custom power sequence. In the figure shown below, a power sequence for an SoC is outlined. A separate unit senses the power bumps and generates powergoorst_b signal to all internal logic. For DHG products, this is the P-unit in the SoC. TAP IPs are tested with this power sequence. The commands from the BFM that let you do this are also shown in the bottom of the timing diagram.

Deleted: ~~This chapter is primarily for an IP or subsystem to describe usage flows for an SoC to use. It is not intended to describe co-validation efforts shared between IP and subsystem or subsystem and SoC.¶~~
~~Describe the main IP and co-IPs together.¶~~
~~Include details for enabling key flows (for example, reset, configuration, PM entry/exit, linkup/down, upstream/downstream traffic etc).¶~~
¶

Deleted: ~~Complete.¶~~
¶

Deleted: ~~Complete.¶~~

Figure 2. SoC Power-Up Sequence #1

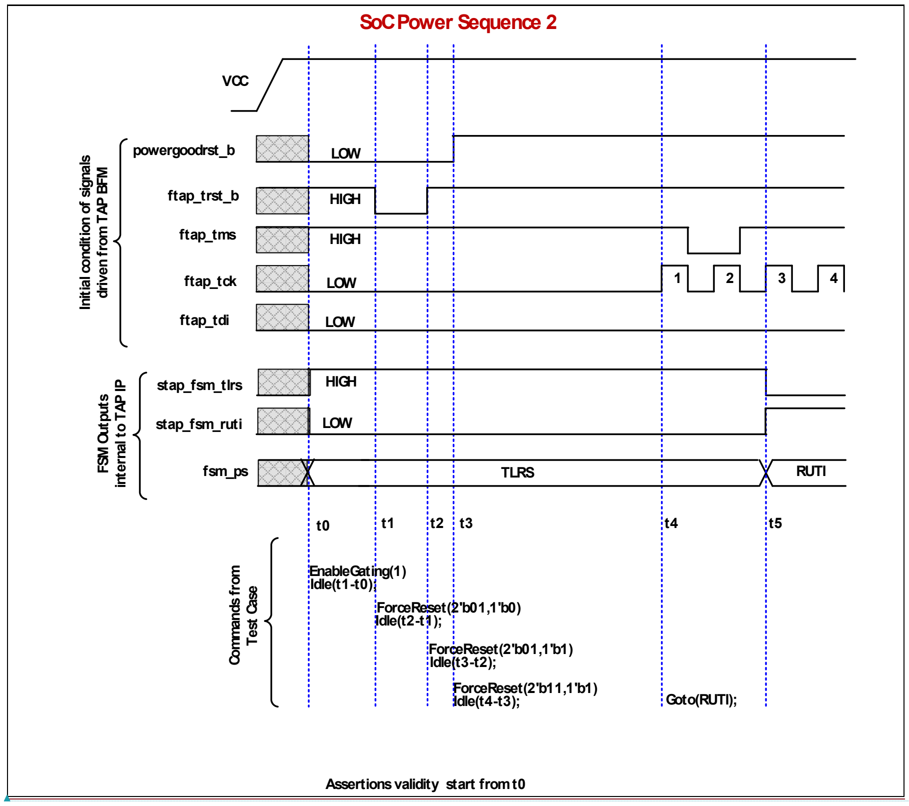


Field Code Changed

As shown in the following figures, the ForceReset task lets you drive reset ports as per the polarity given in the argument for the task call. The fdfx_powergood and trst_b are logically AND'ed inside TAP IPs and used.

As illustrated in the figure below, issuing a TRST_B while fdfx_powergood is asserted will not cause any access to the TAP. The default state of the TAP state machine in any reset condition is TLRS.

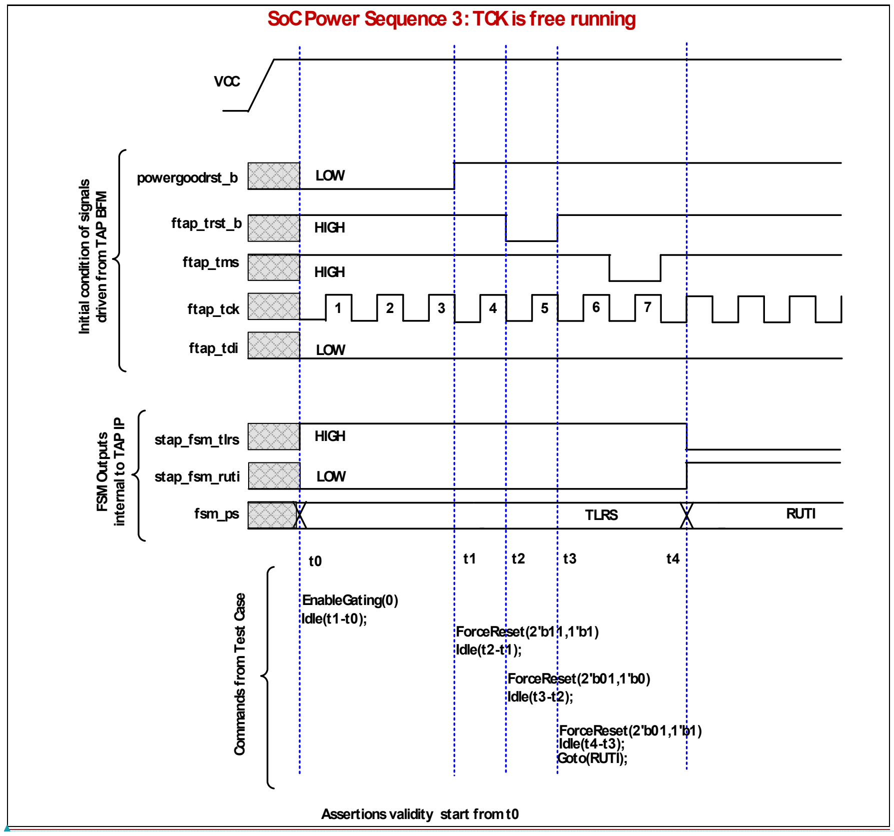
Figure 3. SoC Power-Up Sequence #2



Field Code Changed

The figure below shows the case of a free running TCK condition, as TMS is pulled high externally on a board by a weak pull-up, the TAP will remain in TLRS condition until all the resets are deasserted and a proper TMS is driven to start a valid transition.

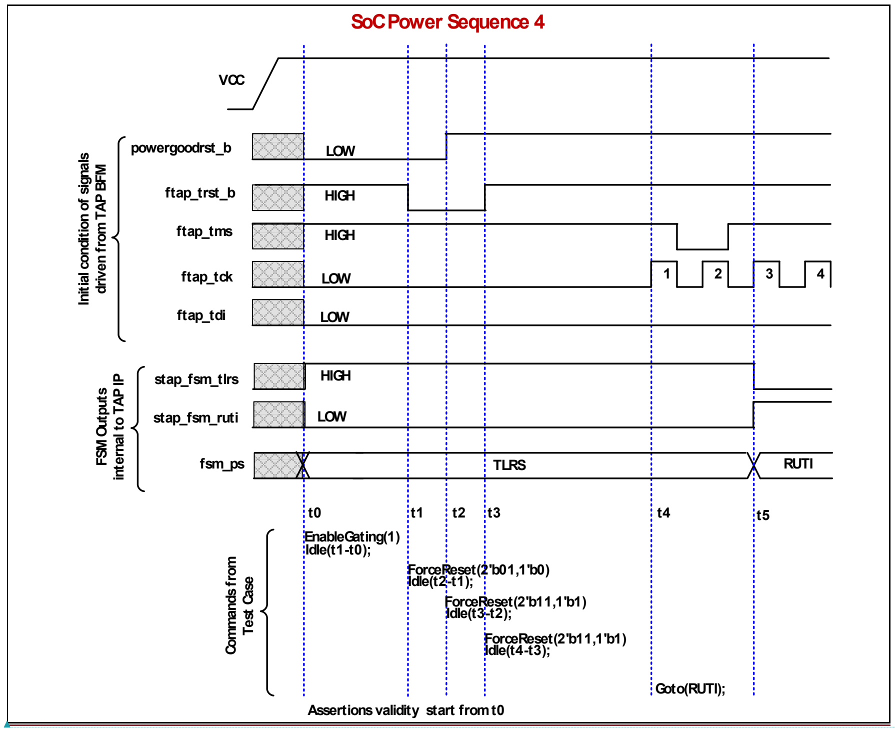
Figure 4.   SoC Power-Up Sequence #3



Field Code Changed

In the figure below, trst_b is asserted right in the middle of powergoodrst_b assertion window. Again, the TAP will remain in TLRS until a valid transaction occurs.

Figure 5. SoC Power-Up Sequence #4



Field Code Changed

Deleted: Complete.¶

5.4 Upstream/Downstream Traffic Flow Details

Not applicable to this IP/IPSS

Deleted: Complete.¶

5.5 PM Entry/Exit Flow Details

Not applicable to this IP/IPSS

Deleted: Complete.¶

5.6 Safety Entry/Exit Flow Details

Not applicable to this IP/IPSS

Deleted: Complete.¶

5.7 Other Flow Details

Not applicable to this IP/IPSS

Deleted: Include details relevant to the enablement and SoC integration of the design (main IP + co-IPs together).

6 Test Scenarios

Refer to section 3.6.

6.1 IP Integration "First Bring-up/Debug" Test

PowergoodReset & SecurityUnlocked are started in every test which will be used as Bring-up test.

6.2 Register Access through RAL

Not applicable to this IP/IPSS

6.3 PCIE Configuration Space

Not applicable to this IP/IPSS

6.4 Register Access Policies and Attributes

Not applicable to this IP/IPSS

6.5 Security Features

Secured through DFX Secutre policy. Refer to HAS section 16 (included in the 'doc' directory for this release).

6.6 Safety Features

Not applicable to this IP/IPSS

6.7 Datapaths

Not applicable to this IP/IPSS

6.7.1 Upstream/Downstream Traffic

Not applicable to this IP/IPSS

6.7.2 Various Transfer Rates

Not applicable to this IP/IPSS

6.7.3 Other

Not applicable to this IP/IPSS

Deleted: This chapter is primarily for an IP or subsystem to describe test scenarios for an SoC to use. It is not intended to describe co-validation efforts shared between IP and subsystem or subsystem and SoC.¶
Include details for enabling key flows (for example, reset, configuration, PM entry/exit, linkup/down, upstream/downstream traffic, etc.).¶
Each of the test scenario descriptions need to contain the following information for main IP + co-IPs together:¶
Location of the reusable IP sequences¶
Location of an example IP test that utilizes this reusable IP sequence ¶
Description of each reusable test and sequence used in the specific test scenario¶
Information on how to fine-tune sequences to hit certain desired scenarios (certain data rates, cock frequencies, etc.)¶
Information on how to configure the BFM(s), if any, for the tests¶
Information on the checkers/trackers relevant for the test case (enablement, significant data points, etc.)¶
Information on any forces, hacks needed¶
Information on how the PASS/FAIL criteria is determined¶
BKMs for enabling, debugging the test scenario ¶
Any other information that is not in this list but relevant for the enablement and SoC integration of the test scenario¶

Formatted: Font: 9 pt, Not Expanded by / Condensed by

Formatted: BodyText

Deleted: Complete.¶

Deleted: Include primary, sideband.¶

Deleted: Include DEVID and Vendor ID.¶

Deleted: Include SAI validation, i.e. through CREST tests.¶

Deleted: Include any other security features other than SAI, i.e. register locks, etc. ¶

Formatted: Font: 9 pt, Not Expanded by / Condensed by

Deleted: Include any other safety features other than safety feature, i.e. fault injection, configuration parameters setting, calibration parameter setting, etc.¶

Deleted: Include any simple and complex tests for validating data paths.¶

Deleted: Complete.¶

Formatted: Font: 9 pt, Not Expanded by / Condensed by

Deleted: Complete.¶

Deleted: Complete.¶

6.8 Interrupt Verifications

Not applicable to this IP/IPSS

Deleted: Complete.¶

6.9 Straps

Not applicable to this IP/IPSS

Deleted: Discuss tests to cover straps through functional tests and to cover simple connectivity (for example, after strap pulling, verify that strap values are propagating to their endpoints regardless of functional correctness).¶

6.10 Fuses

Not applicable to this IP/IPSS

Deleted: Include tests to cover fuses through functional tests and through simple connectivity, including HIP fuses that are pulled by the controller (for example, after fuse pulling, verify that fuse values are propagating to their endpoints regardless of functional correctness).¶
¶

6.11 IP-specific Clocks and Reset Tests

Most of tests will cover programming Clocks and Resets.

Deleted: Complete.¶

6.12 Device and Function Disablement/Enablement

Not applicable to this IP/IPSS.

Deleted: Complete.¶

6.13 Sequences to Support SoC Power Gating and Flows

Not applicable to this IP/IPSS.

Deleted: Include Idle, PM entry/exit, VNN removal, save-restore, etc.).¶

6.14 Sequences to Support SoC Reset Flows

Not applicable to this IP/IPSS.

Deleted: Include cold and warm reset related sequences.¶

6.15 Sequences to Support SoC Performance

Not applicable to this IP/IPSS

Deleted: Include test sequences that allow tweaking performance parameters, transfer , block sizes, etc.¶

6.16 Linkup/Down

Not applicable to this IP/IPSS

Deleted: Complete.¶

6.17 Co-IP Validation Test Scenarios

Not applicable to this IP/IPSS

Deleted: For complex IPs consisting of any other sub-IPs that are included in the main IP's cluster level as part of a co-validation effort. (For example, PCIe, USB, UFS controller with PHY and FIA).¶

6.17.1 Register Access through RAL to Co-IPs

Not applicable to this IP/IPSS

Deleted: Complete.¶

6.17.2 Security Features and SAI Validation of Co-IPs

Not applicable to this IP/IPSS

6.17.3 Safety Features and Validation of Co-IPs

Not applicable to this IP/IPSS

6.17.4 Straps, Fuses, and Configuration of Co-IPs

Not applicable to this IP/IPSS

6.17.5 Power Gating Verification of Co-IPs

Not applicable to this IP/IPSS

6.17.6 Other Test Scenarios

Not applicable to this IP/IPSS

6.18 Other Specific Functional Scenarios

Not applicable to this IP/IPSS

Deleted: Complete.¶

Deleted: Complete.¶

Deleted: Complete.¶

Deleted: Complete.¶

Deleted: Include other relevant integration verification of the co-IPs.¶

Deleted: Discuss main IP +co-IP specific functional scenarios that are not listed in the previous sections but are needed to ensure proper design (main IP +co-IP) integration in the SoC.¶

7 Stimulus Details

7.1 IP Power Up and Reset—Example

PowergoodReset describes the reset and powergood sequence.

7.2 Initial IP Configuration—Example

Stap has the following configurations defined.

- ADL
- ADP

7.3 Dynamic IP Configuration—Example

Once a test starts then there is no configuration that needs to be changed until the test gets completed.

7.4 Dynamic Injectors—Example

The injectors are TAP Driver from JTAG BFM. This takes in the command from the test sequences and then drives the pin interface with the intended stimulus for exercising the DUT.

7.5 Stimulus Generation—Example

The only way for generating stimuli is through standard OVM sequences.

7.6 Transaction Classes / Sequence Items—Example

Table 5. Transaction Classes

Transaction	Characteristics
TapInMonSbrPkt.sv, JtagBfmInMonSbrPkt.sv	This class captures the snapshot of input ports of the DUT that needs to be verified in Scoreboard.
TapOutMonSbrPkt.sv, JtagBfmOutMonSbrPkt.sv	This class captures the snapshot of output ports of the DUT that needs to be verified in Scoreboard.
STAPCLinkAdapterCfgPkt.sv	This class defines packet class which listed Fields of Adapter CFG Register.

`JTAG_BFM_Path : /p/cdft/dteg/bfms/jtag_bfm/CHASSIS_JTAGBFM_2019WW17_R3.6`

7.7 Sequencers / Sequence Drivers—Example

The sequencer STAP_TAP_Seqr is extended from JTAG Sequencer to drive JTAG traffic. stapSbRdSeq is extended from ovm_sequence for SB read. It connects iosfsbm sequences through the sequence library (STAP_SeqLib) to drive IOSF traffic.

Details about sequence item and more for these sub-IPs can be found in IOSF and JTAG documentation.

Deleted: Describe all relevant details about the stimulus used to verify the IP or subsystem. Details on stimulus will likely vary between IPs, and the contents of this section are provided as examples. Sections can be deleted, appended, or customized as needed.¶
¶

Deleted: Describe the stimulus mechanisms used to achieve IP or subsystem power up and reset for simulation, including warm and cold resets. This can include the modeling of reset inputs, boot flows for CR initialization, fuse download, memory initialization, firmware download and/or modeling activity associated with ramping power domains. ¶

Formatted: Font: 9 pt, Not Expanded by / Condensed by

Deleted: Describe the stimulus mechanisms used to achieve initial IP or subsystem configuration that is not associated with power up or reset flows, but that can occur any time prior to the point at which meaningful stages of test activity begins. This can include mechanisms for state initialization (such as random init), non-reset CR, or fuse configurations, non-reset memory initialization or test-specific modeling of software for configuring or contouring the IP. List user-defined control knobs for these, if applicable.¶
¶

Deleted: Describe any stimulus mechanisms used to alter IP or subsystem configuration during the primary test run phase. This can include dynamic CR accesses. Note that the contents of this section may be already covered in the sequence or injector sections. Combine or exclude sections as needed to eliminate duplication. ¶

Formatted: Font: 9 pt, Not Expanded by / Condensed by

Deleted: Describe any additional mechanisms used to provide dynamic stimulus. This can include tasks, events, error injections, or other methods used to intervene during the simulation of the IP or subsystem to exercise IP features. Each injector should be described, as well as information on how the injection is triggered and/or handled.¶

Formatted: Font: 9 pt, Not Expanded by / Condensed by

Formatted: Normal

Deleted: Describe the mechanisms or flows for generating stimulus. These can include random test generators, test list generators, custom command files, high-level modeling, or custom flows to convert or process collateral into a format suitable for stimulus in the IP verification environment. If all stimuli are hand coded, you can specify that here.¶

Deleted: Describe the primary transaction classes or sequence items used for DUT stimulus. Describe relevant fields of the transaction class as well as any mechanisms for manipulating elements of the transaction class. If elements of the transaction cla ...

Deleted: Describe any sequencers or sequence drivers that are available for driving stimulus into the IP or subsystem, as well as the corresponding interfaces and transaction classes or sequence items used. ...

7.8 Sequences / Sequence Libraries—Example

Table 6. <Interface X> Sequences—Example

Name	Description	MS	Status
SINGLE_READ	Single read sent from <Interface X>	0.5	Coded
MULTI_RANDOM_RW	Multiple random reads/writes from <Interface X>	0.8	Planned

Table 7. <Interface Y> Sequences—Example

Name	Description	MS	Status
CR_WRITE	Single CR write sent from <Interface Y>	0.5	Coded
RAND_IO_CFG_MEM	Single random IO/Cfg/Mem access from <Interface Y>	0.8	Planned

Table 8. Virtual Sequences—Example

Name	Description	MS	Status
ADDR_CONFLICT	Simultanneous mem access to same address from <Interface X> and <Interface Y>. Uses result data compares to perform self-checking	0.8	Coded
RECOVER_FLOW	Multi-part sequence used to exercise the recover flow.	1.0	Planned

Deleted: Indicate which sequences or sequence libraries are used for stimulus. You can use separate tables to describe the sequences for each interface or virtual sequence, as illustrated in the examples below. You can also document sequence categories instead of actual sequences, for brevity. If a table or list of sequences is much longer than one page, include it as an appendix to this document instead. The target milestone for implementation and current status can also be tracked in such tables. Include the location of sequences and/or sequence libraries, and the mechanism for including these in tests.¶

Formatted: Font: 10 pt

Formatted: Font: 10 pt

Formatted: Font: 10 pt

7.9 Tests and Test Templates—Example

Table 9. Directed Tests—Example

Name	Description	MS	Status
basic_init	Directed test for basic initialization	0.5	Passing
offset_calibration	Directed test for offset calibration. Configuration is random.	0.8	Coded

Table 10. Random Tests—Example

Name	Description	MS	Status
randmem	Random memory access test	0.5	Coded
ten_random_sequences	Ten random sequences from sequence library	0.8	Planned

Deleted: Describe the tests or test templates that are used for stimulus. As illustrated in the examples below, you can use tables to describe random or directed test templates. The target milestone for implementation and status can also be tracked here. If a table or list of tests is much larger than one page, you can provide it as anppendix to this document. If preferred, you can describe test categories instead of actual tests, for brevity. Include the location of tests or test templates.¶

Formatted: Font: 10 pt

Formatted: Font: 10 pt

7.10 Test Lists and Regressions—Example

Table 11. Regression and Other Test Lists—Example

Name	Description	MS	Status
doa.list	Basic set of tests used to determine dead-or-alive health. Used during initial bring-up stages of new collateral	0.5	Coded
level0.list	Gating regression for turnins. ~50 tests with fixed seeds. Appended as needed.	0.5+	Coded
level1.list	Regression list of ~300 tests with random configurations and random seeds. Appended as needed. Run ~3x/week.	0.5+	Coded, Passing 85%
soc_int_0.5.list	SoC integration list that is delivered to SoC for MS 0.5. Static IP configs. 5 tests (subset of L1). Verify passing for 0.5 drop.	0.5	Coded, Passing

Deleted: Describe any test lists or regressions that are used to verify the IP or subsystem. As illustrated in the examples below, you can use tables to describe lists, their function, and their status. If a table of test lists is much longer than one page, you can provide it as an appendix to this document. Be sure to include the location of test lists.¶

Formatted: Font: 10 pt

Name	Description	MS	Status
soc_int_0.8.list	SoC integration list that is delivered to SoC for MS 0.5. Static IP configs. 85 tests (subset of L1). Verify passing for IP drop.	0.8	Planned
power.list	Special list of ~25 tests to stress power flows. Updated as needed, run weekly.	0.8	Planned
perf_bw.list	List of 10 high bandwidth tests developed for use by the performance team.	N/A	Planned

8 Coverage Details

8.1 Coverage Tools —Example

Information on how to setup and run the IP level regressions are contained in
`$IP_ROOT/README_cov.txt`

VCS coverage tool will be used for getting the functional, code, toggle, assert coverage. To run coverage, below command is used.

```
$IP_ROOT/scripts/get_coverage
```

This invokes VCS simulation of each of the regression cases with coverage analysis turned on and collects the aggregate coverage using URG, displaying the results in HTML.

To invoke the DVE interactive viewer, use the following command after running get_coverage:

```
$IP_ROOT/scripts/run_dve
```

8.2 Functional Coverage Conditions —Example

Table 12. IP Functional Coverage Conditions—Example

Group	Name	Cases	Description	MS	Status
input_unit	all_us_txns	40	All upstream input unit transaction types	0.5	Coded, 80%
input_unit	no_credits	4	No credits available after init, per channel	0.5	Coded, 100%
input_unit	us_ds_conflict	9	Address conflicts in us/ds transactions	0.8	Planned
input_unit	errrors	6	Post-reset recoverable input errors	1.0	Planned
fifo	fifo_full	2	Primary Fifo full, full-1	0.8	Coded, 0%
power	clk_gate	4	Post-reset clock gating duration	0.8	Planned
Current Coverage Summary	Current MS: Val 0.5	44 (/80)	0.5 Goal: 40-50% of all planned coverage - 36/46 (78%) of coded coverage - 36/80 (45%) of all est. planned coverage	0.5	78% / 45%

8.3 Code Coverage—Example

Code cverage is analyzed for the entire stap module excluding all sub-IPs.

```
COVERAGE REPORT:
SCORE    LINE   COND    TOGGLE   FSM    BRANCH ASSERT GROUP
99.80  100.00 100.00  100.00  100.00 100.00  98.83
```

8.4 Coverage Indicators—Example

After running the coverage, the coverage database that is located at the below location can be viewed to see if the coverage is hit or not and the total is reported here.

```
firefox tools/cov_rpt/AllCov/dashboard.txt
```

Deleted: Describe all relevant details involving coverage and its use in verifying the IP or subsystem, or in providing test feedback. Details on coverage will likely vary between IPs and subsystems, and so the contents of this section are provided only as examples. Delete or append sections as needed.¶
¶

Deleted: Document the tools and mechanisms for gathering or analyzing coverage. The type and location of the coverage database should be identified, as well as any tools or scripts used to roll-up, combine, track, or analyze coverage.¶
¶

Deleted: <#>Describe the functional coverage conditions targeted for IP or subsystem verification. You can use tables to describe coverage conditions and other data such as the number of conditions, coverage groups (including security), target milestones for implementation, and current status. For brevity, you can describe coverage groups instead of detailed conditions. If a table or list of conditions is much longer than one page, you can provide it as a link or an appendix to this document.¶
General statistics on coverage can be included here; however, a detailed analysis of coverage holes should not be included. These holes are to be reviewed separately as part of IP or Subsystem Verification exit reviews.¶

Formatted: Font color: Custom Color(RGB(8,96,168))

Formatted: Font color: Custom Color(RGB(8,96,168))

Formatted: Font: 10 pt

Deleted: Describe planning and execution details of code coverage if they are used as part of the IP or subsystem verification strategy.¶
¶

Formatted: Command Line 1

Deleted: Describe coverage indicators, if applicable. Add a link to indicators or embed a graph or chart of project coverage indicators, if appropriate. ¶
A detailed analysis of coverage holes should not to be included here. These holes should be reviewed separately as part of IP or Subsystem Verification exit reviews.

9 Checking Details

Table 13. Checkers—Example

Name	Description	MS	Status
IOSF Compliance	Passive verification module delivered with IOSF BFM that does dynamic checking of various IOSF interface rules. Will be enabled by default. May only be disabled during compile with +NO_IOSF_CHK. Checker assertions may be ignored if specific assertions are added to $IP_ROOT/sva_control/ignore.sva	0.5	Enabled
Input Scoreboard	Main dynamic scoreboard checker used to detect dropped or corrupted transactions. Will be enabled by default. Can be disabled from test run commandline using -NO_INP_SB_CHK	0.5	Enabled
Rcomp Checker	Dynamic checks for a variety of rcomp rules / protocols. Will be enabled by default. Can be disabled from test run commandline using -NO_RCOMP_CHK	0.8	Planned
Data Checker	End-of-test dynamic check to ensure contents of local memory match expected values from the specified test's mem.out file	0.8	Coded
Power Flow Checker	Post-process power flow checker (inherited) that operates on power.log. Will only be enabled on power regression, disabled by default. Enable using =PWR_CHK on test commandline.	0.8	Coded
Performance Checks	Performance checks will be owned by the performance team.	N/A	N/A
Misc SVAs	Embedded SVAs used throughout the design to detect illegal conditions. Will be enabled by default. Failing assertions can be ignored if specific assertions are added to $IP_ROOT/sva_control/ignore.sva	0.5+	Ongoing
Test Self-Check	Note to highlight that many directed tests do self-checks as part of the test.	0.5+	N/A

Deleted: Describe the checkers used to verify the IP or subsystem. Document all major checkers, along with details on what is being checked and how to enable/disable the checking. As seen in the example table below, descriptions can also include information on the implementation target milestone and status. If there are checking holes or any out-of-scope checking that is not expected to be performed as part of this IP's or subsystem's verification plan, highlight it here. Details on checking will likely vary between IPs and subsystems, and therefore the contents of this section are provided only as examples. Customize this section as needed.

9.1 Scoreboard and Checker

- Downstream path (IOSF SB and JTAG):

  There is a full scoreboard on this path, which includes correct checking of ordering as well as the packet bytes integrity checking. Possibility of out of order dispatch of requests (P & NP) by stap inorder to pull the Fuses and DRNG Key. It is assumed for queuing checking purposes. As soon as we drives a packet into stap, it will push the expected packet into the downstream queue. If there is no out of order grant from the IOSF SB BFM, all transactions will be "in-order" comparing the order at IOSF interface in downstream path. When there is an out of order grant from the IOSF bfm, there can be a swap between the order of certain P and NP requests coming out of stap IOSF master interface.

  In case of JTAG packet injection, expected packets are injected by the environment layer into the downstream.

- Upstream path:

  Scoreboard Calculates the Expected Output. If there is any mismatch between Expected Output and Actual output (From DUT), Scoreboard throws an error Message. On the other side assertions will fail if there are any mismatches.

  Downstream checker should be able to take care of out of order checking due to the possibility of reordering between P and NP from stap hardware.

- Sideband path:

All stap register accesses are checked for attributes using Sideband and TAP interface.

Check Point	Objective	MS	Status
IOSF Compliance	Passive verification module delivered with IOSF SB BFM that does dynamic checking of various IOSF interface rules. Will be enabled by default.	1.0	Enabled
Input Scoreboard	Main dynamic scoreboard checker used to detect dropped or corrupted transactions. Will be enabled by default.	1.0	Enabled
Data Checker	End-of-test dynamic check to ensure contents of local memory match expected values from the specified test's mem.out file	1.0	Enabled
Power Flow Checker	Post-process power flow checker (inherited) that operates on power.log. Will only be enabled on power regression, disabled by default.	1.0	Enabled
Performance Checks	Performance checks will be owned by the performance team.	N/A	N/A
Test Self-Check	Note to highlight that many directed tests do self-checks as part of the test.	1.0	Enabled

Scoreboard is extended from the `ovm_scoreboard` class. This class does the data integrity check between the address and data sent by the input monitor with the accumulated TDO data sent by the output monitor. The TDO data from the output monitor is verified against the address from the input data when the state is Exit 1 IR. Similarly, the accumulated data is verified with data from the input monitor when the state is Exit 1 DR. When accessing the RW registers, this verification happens twice` for bypass and read-only registers it happens when the first data packet is received from the output monitor. While the transaction packet from the input monitor come at each clock cycle, the transaction packet from the output monitor only comes once the state comes out of the Shift IR or Shift DR state. Scoreboard also checks for the parallel data out captured by monitor.

Both stap interface and SB2TAP interface will use the same scoreboard.

1.1.4 Transaction Packet from the JTAGBFM Input Monitor to Scoreboard

The transaction packet from the JTAGBFM Input Monitor to Scoreboard is extended from the `ovm_sequence_item` class. The table below lists the transaction packet fields.

Table 14. JTAGBFM Input Monitor to Scoreboard Transaction Packet

Field	Description
FsmState	Present state of Input Monitor FSM
ShiftRegisterAddress	Value of the IR opcode that the JTAGBFM sent to the DUT (current IP under test)
ShiftRegisterData	Data to be shifted in by the JTAGBFM into the selected DR in the DUT (current IP under test)
Parallel_Data_in	Parallel data that is input to the DUT
Idcode	IDCODE that is strapped at input to the DUT
Power_Gud_Reset	Powergood reset that is issued to the DUT
tdi_shift_cnt	Provides a counter value that counts the number of shifts in SHDR. This counter resets upon entry into SHDR from CADR or E2DR and holds the value in E1DR, PADR, and E2DR.

1.1.5 Transaction Packet from Input Monitor to Scoreboard

The transaction packet from the Input Monitor to Scoreboard is extended from the ovm_transaction class. The table below lists the transaction packet fields.

Table 15. Input Monitor to Scoreboard Transaction Packet

Field	Description
Idcode	IDCODE that is strapped at input to the DUT
Parallel_Data_in	Parallel data that is input to the DUT
fdfx_secure_policy	DFx Secure that is input to DUT
atapsecs_tck	Secondary JTAG clk pass-throughs that are input to the DUT
atapsecs_tms	Secondary JTAG tms pass-throughs that are input to the DUT
atapsecs_trst_b	Secondary JTAG trst_b pass-throughs that are input to the DUT
atapsecs_tdi	Secondary JTAG tdi pass-throughs that are input to the DUT
cntapnw_atap_tdo2	Secondary JTAG tdo2 pass-throughs that are input to the DUT
cntapnw_atap_tdo2_en	Secondary JTAG tdo2_en pass-throughs that are input to the DUT

1.1.6 Transaction Packet from the JTAGBFM Output Monitor to Scoreboard

The transaction packet from the JTAGBFM Output Monitor to Scoreboard is extended from the ovm_sequence_item class. The table below lists the transaction packet fields.

Table 16. JTAGBFM Output Monitor to Scoreboard Transaction Packet

Field	Description
ParallelDataOut	Parallel data from parallel out pins at UPDR
ShiftRegisterAddress	Accumulated TDO data at SHIR
ShiftRegisterData	Accumulated TDO data at SHDR
tdo_shift_cnt	Provides a counter value that counts the number of shifts in SHDR. This counter resets upon entry into SHDR from CADR or E2DR and this counter holds the value in E1DR, PADR, and E2DR.

1.1.7 Transaction Packet from the Output Monitor to Scoreboard

The transaction packet from the Output Monitor to Scoreboard is extended from the ovm_transaction class. The table below lists the transaction packet fields.

Table 17. Output Monitor to Scoreboard Transaction Packet

Field	Description
ParallelDataOut	Parallel data from parallel out pins at UPDR
dfxsecure_feature_en	Enable particular dfx secure feature
ftapsecs_tdo	Secondary JTAG port output from DUT
ftapsecs_tdoen:	Secondary JTAG port output from DUT
cntapnw_ftap_tck2	Secondary JTAG port output from DUT
cntapnw_ftap_tms2	Secondary JTAG port output from DUT

Field	Description
cntapnw_ftap_trst2_b	Secondary JTAG port output from DUT
cntapnw_ftap_tdi2	Secondary JTAG port output from DUT

1.1.8 Using the Scoreboard

The Scoreboard pipes the OVM report messages into the following file:

```
$IP_ROOT/pwa/results/tests/<test_name>/JtagBfmTrackerPri.out
```

1.1.9 API

This is not applicable for this IP. Please refer to the JTAG BFM Integration Guide for a list of APIs.

1.1.10 Examples for Using the Scoreboard

The Scoreboard does not have a separate instantiation.

1.1.11 Error Messages

When there are failures in assertions or mismatched data, the error messages are written to the log. A sample mismatched data condition is shown below.

```
OVM_ERROR @ 10065000: ovm_test_top.Env.ScoreBoard [**Data Mis Match ]
Actual Data:   0000000000000000000000000000000000000000000001
Expected Data: 0000000000000000000000000000000000000000000000
```

Formatted: Default Paragraph Font, Font color: Auto

Formatted: BodyText

Deleted: Give details of scoreboard/MOAT checks here.¶
Check Point

...

10 Debug Details

In order to debug the IP, verbosity can be controlled by enabling FSDB in the list file.

```
+defaults -ace_args -simv_args +FSDB=ALL -ace_args-
```

For opening in verdi, the following commands can be used.

```
cd $IP_ROOT

simbuild -dut stap -ace_args ace -ccod -ice -view_debussy -verdi -ace_args- -1c
-CUST ADP -1c-
```

Once Verdi is loaded, open the FSDB from the directory:
`$IP_ROOT/regression/default/<test.list>/<test>/novas.fsdb`

Deleted: Describe debug collateral, or special debug methods or flows for the IP or subsystem. This can include log files, trackers, or scripts to interpret tracker output, watch windows, Verdi utilities, RC files, IP defeature modes used to isolate issues, or other debug-related tools. Describe common workarounds here.¶

11 Formal Verification Details—Optional if no FV/FPV

Not applicable to this IP/IPSS

Deleted: Provide details of formal verification methods used to validate the IP or subsystem. Include details of FV boundaries of the IP or subsystem, the properties and proofs that were targeted, and information on the assumptions and constraints used. If no formal methods were used in verification, leave this section and type "Not applicable."¶

12 IP or Subsystem Verification Milestones

12.1 Milestones

Table 18. Development Verification Milestones (with example Status)

Milestone	Verification Tasks	Status
IP Val 0.0	(Prerequisite: HAS 0.0)	Received
	Planning effort complete for testbench and infrastructure changes per HAS 0.0 definition, Stragety sections complete in v0.0 Plan	Complete
	Coverage enabled	Complete
	Infrastructure enabling coded	Complete
IP Val 0.5 (Current MS)	(Prerequisite: HAS 0.5)	Received
	Testbench and infrastructure support for HAS 0.5 features	Complete
	Tests coded and passing for HAS 0.5 features	Complete
	Overall regression pass rate 50%-60% (SoC Val 0.5 reuse tests pass)	Complete
	Coverage coded for HAS 0.5 features	Complete
	Coverage at 40%-50% of estimated IP Val 1.0 targets	Complete
IP Val 0.8	(Prerequisite: HAS 0.8)	Received
	Testbench and infrastructure support for HAS0.8 features	
	Tests coded and passing for HAS 0.8 features (all features)	
	Overall regression pass rate of 75%-85% (SoC Val 0.8 reuse tests pass)	
	Coverage coded for HAS 0.8 features	
	Coverage at 70%-80% of estimated IP Val 1.0 targets	
IP Val 1.0	(Prerequisite: HAS 1.0)	
	Overall regression pass rate of 98% with all failures explained	
	Coverage at 95%	

12.2 Other Indicators

Not applicable to this IP/IPSS

Deleted: Describe high-level verification tasks and corresponding status for each IP or subsystem development milestone. The example table provided below should be used (especially for IPs or subsystems that are currently under development), but it can be modified to include relevant notes.¶
The example table below contains IP development verification milestone definitions and high-level verification tasks that have been officially prescribed for IP developers. Any deviations from these definitions and tasks must be coordinated between the IP delivery and SoC teams during the project Map Day.¶
Table 10.

Formatted: TableCaption

Formatted: Font color: Custom Color(RGB(8,96,168))

Deleted: Include any indicators or exit criteria used to track or communicate the progress of IP or subsystem verification or related tasks. For example, include staging plans or indicators that track the history or status of bugs and/or ECOs.¶
If this section is not relevant, leave the heading and replace the red explanatory text with "Not applicable." Apply the BodyText style.¶

13 Validation Risks

Not applicable to this IP/IPSS

Deleted: Describe the main validation risks, specifically, any waivers given in the IP-level or subsystem-level validation reviews and any needed information for SoC validation attention. For example, specify if a particular function must be validated to qualify other functions.¶

14 Glossary

This is the last chapter in the document.

This chapter lists the definition of key terms used in this document.

Term	Definition
Assertion	A property that is true at all times.
Assertion coverage	A measure of how thoroughly an assertion has been exercised.
TAP	Test Access Port
sTAP	Slave TAP
mTAP	Master TAP
TAPNW	0.7 TAP Network
OVM	Open Verification Methodology
IP	Intellectual Property
SoC	System on Chip
FSM	Finite State Machine
WSP	Wrapper Serial Port
WTAP	Wrapper TAP
WTAPNW	WTAP Network
FUB	Functional Unit Block
CLTAPC	Chip Level TAP Controller
TAPC	TAP Controller
Env	Validation/Test Environment
HAS	High Level Architectural Specification
MAS	Micro level Architectural Specification
IEEE	Institute of Electrical and Electronics Engineers
SV	SystemVerilog
SVA	SystemVerilog Assertions
TB	Test Bench
GLS	Gate Level Simulation
RTL	Register Transfer Level
TLM	Transaction Level Modeling
TBD	To Be Done
Rev	Revision
DFT	Design For Test
IR	Instruction Register
DR	Data Register
DUT	Design Under Test
IRR	Intel Reuse Repository
FPV	Formal Property Verification

Term	Definition
MDA	Multi Dimensional Array

Deleted: <#>Goals¶
To describe the verification architecture for STAP.¶
Scope¶
The usage of this verification environment is primarily to validate the IP. When used at the SoC level, the JTAGBFM was used to configure all the registers. This document outlines how to use the IP at SoC level.¶
Completion Criteria¶
To meet all the OVM and Saola compliance guidelines for a Verification Component IP.¶
Strategy¶
Milestone Definitions¶
¶
Milestone