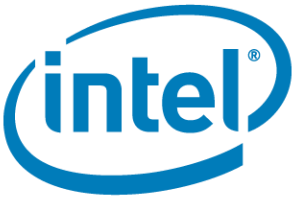


Chassis JTAG BFM

USER GUIDE

Revision 4.1
April 2020

Intel Top Secret

Contents

About This Template

How to Use This Template

Do not remove any headings from this document. If you do not need the headings to describe your IP, enter "Not applicable" under the heading. This lets the reader know that you did not overlook this topic.

In the main document that follows, add new headings that you need at the end of the appropriate chapters.

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; instead just replace the "Gap" text with "Not applicable" and apply the BodyText style.

Goal of This Document

This document should contain all information an team would need to accomplish the 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. This should be a self-contained user guide.

1 Introduction

1.1 Audience

The information in this document is intended for an SoC or an IP Verification team that instantiates this Verification IP or a Bus Functional Model (BFM).

1.2 References

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

Document Name	Link / Location
IP_Name Integration Guide	Included in the 'doc' directory for this release
IP_Name Verification Plan Reference	Included in the 'doc' directory for this release
IP_Name Product Brief	Included in the 'doc' directory for this release
IP_Name Release Notes	Included in the 'doc' directory for this release
Other document used to support this IP	Included in the 'doc' directory for this release

1.3 Contact Information

If you need additional help, use the contact information below.

Function	Name	Email
Validation	Bulusu, Shivaprashant Bandana, Sudheer V Chelli, Vijaya Sharma, Pankaj (Section 3.7) Veeresha Bevinamatti	shivaprashant.bulusu@intel.com sudheer.v.bandana@intel.com vijaya.chelli@intel.com pankaj.sharma@intel.com veeresha.bevinamatti@intel.com
Doc Template Owner	Susann Flowers	susann.flowers@intel.com

1.4 Terminology

The table below defines uncommon terms used in this document.

Term	Definition

1.5 Document Revision History

Author	Rev Number	Description	Revision Date
Bulusu Shivaprashant	0.5	Initial release.	29 May 2011

Author	Rev Number	Description	Revision Date
Bulusu Shivaprashant	1.0	Added information on reusing Ace setup of this IP. Added Code snippet for Test Island usage.	07 Sep 2011
Bulusu Shivaprashant	1.1	Corrected the examples in Section 3.5. Dropped Saola support	27 Mar 2012
Bulusu Shivaprashant	1.2	Added support for SoC TAP Network element	11 May 2012
Bulusu Shivaprashant	1.3	Enhanced SoC TAP Network capabilities	26 Jun 2012
Bulusu Shivaprashant	1.4	Not published on IRR Provided Hierarchy, Hybrid Secondary Port support	17 Aug 2012
Bulusu Shivaprashant	1.5	Added support to use TAPACCESS at IP level and SoC levels Added APIs: Tap Remove, Read all IDCODEs	16 Sep 2012
Bulusu Shivaprashant	1.6	Added new algorithm for dynamic access of TAPs on a hierarchy Removed examples of Hybrid mode	02 Nov 2012
Bulusu Shivaprashant	1.7	Added Tertiary Port, MultiTAP Access, TapAccessRMW, Optimized IR scans. Renamed tick define. Removed "_CTT" from generated file names.	18 Jan 2013
Bulusu Shivaprashant	1.8	Documented known issues Added Shadow Mode capability	22 Mar 2013
Bulusu Shivaprashant	1.9	Bug fixes Major Tracker Enhancement	04 Jul 2013
Bulusu Shivaprashant	2.0	Bug fixes Updates to Table-3, Section 2-10. Added API ReturnTDO_ExpData_MultipleTapRegisterAccessRuti Table-4. Added back primary, secondary, tertiary tracker variable names. Table-5. Added CHASSIS_JTAG_WORD_LENGTH as a new `define. Table-8. Added comment on DisableTap API. Code-2. Contents of enhanced runtime tracker with active TAP are shown. Document the usage of primary, secondary, tertiary tracker variable names and show them being used in Env file. Tracker enhanced from a formatting perspective to suit moat. Active TAP name displayed in the tracker.	14 Sep 2013
Bulusu Shivaprashant	2.1	Bug fixes Updates to Code4, Code-5 updated to reflect a parameterized instantiation of Pin interface, MasterAgent and TestIsland. Addition of new APIs to support ITPP. Tracker enhancement for E2DR.	10 Jun 2014

Author	Rev Number	Description	Revision Date
Bulusu Shivaprashant	2.2	Bug fixes	02 Jan 2015
Badana, V, Sudheer	2.3	But Fixes Simultaneous Primary and Secondary Access for MultiTap. Added a function SetCltapcNetworkSelOpcode. Added CTV_ReturnTDO_ExpData_MultipleTapRegisterAccess for PCR https://hsdes.intel.com/home/default.html#article?id=1205378989	20 Mar 2015
Badana, V, Sudheer	2.4	Converged BFM Code description added Added configuration variable "sample_tdo_on_negedge" to sample tdo at negedge of tck. https://hsdes.intel.com/home/default.html#article?id=1503984885 Added define SNPS201412BC for VCS(<F-2011) to avoid export package compilation Error https://hsdes.intel.com/home/default.html#article?id=1503985317	15 Sep 2015
Badana V Sudheer	2.5	Added RTDR support Added UVM support	22 July 2016
Badana V Sudheer	2.6	Added EnDebug support	26 December 2016
Badana V Sudheer	2.7	Fixed few issues in BFM related to TAPNETWORK for ENDEBUG	01 February 2017
Badana V Sudheer	2.8	Fixed few issues in BFM related to ENDEBUG	26 March 2017
Veeresha Bevinamatti	2.9	Added basic TAPLink support. Documented the steps to get the BFM topology files for Taplink.	31 May 2017
Badana V Sudheer	3.0	Fixed few issues in BFM related to ENDEBUG	25 July 2017
Badana V Sudheer	3.1	Updates of EnDebug for TGP customer	30 Dec 2017
Badana V Sudheer	3.2	Updates of EnDebug for TGP customer	09 Feb 2018
Badana V Sudheer	3.3	Added new APIs to support no of RUTI cycles after connecting a new STAP	24 Mar 2018
Badana V Sudheer	3.4	Added new APIs to support ARC TAP for ENDEBUG. https://hsdes.intel.com/appstore/article/#/1606140990 Added configuration variables to flush tdi_shift_reg and tdo_shift_reg in UPIR/UPDR. https://hsdes.intel.com/appstore/article/#/1505079846 Added configuration variables to configure initial value of trstb https://hsdes.intel.com/appstore/article/#/1406939499	06 Aug 2018

Author	Rev Number	Description	Revision Date
Badana V Sudheer	3.5	JTAG BFM to support TapLink Network Programming for EnDebug https://hsdes.intel.com/appstore/article/#/1606682224 Support needed for enDebug to handle fscan_mode https://hsdes.intel.com/appstore/article/#/1606878794 Need new APIs in JtagBfm for ENDEBUG_DEBUG_*_GRN and FSM_MODE=2'b01 for TAPLINK https://hsdes.intel.com/appstore/article/#/1606882025 EnDebug APIs modification for checking negative Test cases in EnDebug https://hsdes.intel.com/appstore/article/#/1606884607 Add SVA to check TDO value during non-shift TAP FSM states https://hsdes.intel.com/resource/1604289527 Generating an event at rising edge and falling edge of tck https://hsdes.intel.com/appstore/article/#/1606693671	29 Dec 2018
Chelli, Vijaya	3.6	[JTAG BFM Input Monitor] For E2IR -> SHIR , current TDI shifted to end of previous index instead of from 1st index of "con_tdi_add" storage field. https://hsdes.intel.com/appstore/article/#/1606919529 Shift register ijff is capturing data in same clock where capture is asserted https://hsdes.intel.com/appstore/article/#/1607013057 Need support to issue TAP RESET while EnDebug transactions happening https://hsdes.intel.com/appstore/article/#/1607078526 [JTAG BM Monitor] Monitor is not able to sample TMS properly after trst_b 0 -> 1 transition https://hsdes.intel.com/appstore/article/#/1607095825 [ADPLP] [JTAGBFM] Text macro (dbg_display) is redefined https://hsdes.intel.com/appstore/article/#/1507140978 Need Expected TDO in JtagBfmPinIf to allow debuggers to see the expected values in a trace. https://hsdes.intel.com/appstore/article/#/1607165075	26 April 2019
Chelli, Vijaya	3.7	Generated of gated TCK inside JtagBfm Driver https://hsdes.intel.com/appstore/article/#/1607193235 Update all Val Tool versions in JtagBfm wrt latest TSA version https://hsdes.intel.com/appstore/article/#/1607516862 7nm FIVR: Issue with sample_tdo_on_negedge in JTAG BFM https://hsdes.intel.com/appstore/article/#/2208079899 TDI and TDO mismatch with sample_tdo_on_negedge=1 https://hsdes.intel.com/appstore/article/#/14010054508 JTAG BFM: Enhancement in JtagBfmEnDebugSetup API to support fuse decryption feature https://hsdes.intel.com/appstore/article/#/1607954639	27 September 2019
Chelli, Vijaya	3.8	JTAGBFM should support DFTB HTAP(TAPNETWORK in IRR) https://hsdes.intel.com/appstore/article/#/1608528851 MAT1.6 migration to VCS2019 DUVE: JTAGBFM normal tracker logs TDO (shifted out) incorrectly https://hsdes.intel.com/appstore/article/#/2209674543	6 December 2019

Author	Rev Number	Description	Revision Date
Chelli, Vijaya	3.9	PCRs addressed: [MTL SOC] IP SOC Handoff : IPSOC_DOC https://hsdes.intel.com/appstore/article/#/1507674191 [ADP-S] The synchronous of jtag_tck_rst of MPPHY IPs take 3 tck to get out of reset https://hsdes.intel.com/appstore/article/#/1507637683 All MTL IPs should deliver XVM-complient OVM or UVM code. Request to enable XVM at IP-level simulations. https://hsdes.intel.com/appstore/article/#/2207873794 MTL TFM alignment request to MAT/HDK 1.6 https://hsdes.intel.com/appstore/article/#/2207577769 SEG SoC/IP WG passdown: Reference testbenches and ability to turn off assertions https://hsdes.intel.com/appstore/article/#/22010064502 HSDs addressed: [JTAGBFM rev3p9] *ArcEndbg API (TLRS) assertion https://hsdes.intel.com/appstore/article/#/1609728614 Future SoCs: [3p9] The synchronous of jtag_tck_rst of MPPHY IPs take 3 tck to get out of reset https://hsdes.intel.com/appstore/article/#/16010011552	22 February 2020
Chelli, Vijaya	4.1	PCRs addressed: SEG SoC/IP WG passdown: 60 CPS https://hsdes.intel.com/appstore/article/#/22010065435 SEG SoC/IP WG passdown: LRM Compliance and no BCID switches should be used. https://hsdes.intel.com/appstore/article/#/22010064299 [MTL SOC] IP SOC Handoff : IPSOC_VAL https://hsdes.intel.com/appstore/article/#/1507674541 Request all IPs to support 'IP always aliveness requirements' https://hsdes.intel.com/appstore/article/#/2209259784 MTL: Request to be compatible to SLES12 https://hsdes.intel.com/appstore/article/#/2208965252 HSDs addressed: Enhancement of JTAGBFM to support 256bit session key in endebug https://hsdes.intel.com/appstore/article/#/16010670341	13 April 2020

1.6 IP Revision History

Refer to section 1.5, Document Revision History.

1.7 Firmware Version

Not applicable to this IP/IPSS

1.8 Finding the Component and its Support

Support Information:

- DTEG HSD Home: https://hsdes.intel.com/appstore/article/#/dft_services.bugeco/create?release=DFx%20JTAG%20BFM_PIC&family=DTEG%20Verif%20IP%20Family
- HSD access to dft_services tenant can be obtained by applying in IEM2 at the link: http://iem2.intel.com/GroupManagement/RequestGroupAccess.aspx?group=dft_services-tenant-users

1.9 Subscribe for Notifications

Follow these steps to subscribe.

1. Goto https://eclists.intel.com/sympa
2. Login with Intel email and windows password.
3. In "List of Lists", search for `sip_dfx`

2 Overview of the BFM

This section provides an overview of the manual as well as important information associated with this IP.

2.1 Introduction

Current SoC designs within Intel have large quantities of Technology and Product Synchronization (TAP) ranging between 70-170 TAPs that are arranged in a hierarchical network topology. Accessing a single TAP from a standard JTAG port requires a combination of Instruction Register (IR) shifts to store the OPCODE that points to the desired register and finally shifting data into and out of the Test Data Register (TDR) that is pointed to by the IR. Extending this principle to access a leaf level tap deep within a multi-level hierarchy from the Chip-Level TAP (CLTAP) requires programming each parent TAP that controls their sub-network. From a pre-silicon validation perspective, if this process was composed of manual test writing then it would be tedious, error prone, and unproductive. The Chassis JTAG BFM reduces TAP network validation complexity while providing the ability to maintain the same validation tests when TAPs move due to an unexpected floorplan or architectural changes.

A format that captures the SoC TAP hierarchy in System Verilog is defined. This forms an input to the BFM. Chassis TAP tool (CTT) and collage output these files for BFM. CTT examples are illustrated to show the capability of BFM. Several new APIs were implemented to provide the validation test writer with a level of abstraction where tests can be written as if a single TAP is addressed. This was possible because the CTT performs topology checking that enforces unique names for each TAP in the entire SoC design. This greatly reduces full chip test development time. The BFM also allows IP level sequences to be directly ported to full chip without any modification. However, the BFM can only utilize the information that is entered or imported into the Chassis TAP tool. Users are encouraged to fully populate all fields of the TAP tool as accurately as possible for a richer set of validation collateral available today and potentially new capabilities developed in the future. This solution is applicable to all SoC designs that are compliant to Chassis DFx Gen1 or Gen2 standard.

When IP blocks containing TAPs are integrated at the SoC level with tools like Collage, the first step for the full chip DFx validation engineer is to integrate a JTAG BFM into their verification environment. The second step is to validate that the TAPs are stitched according to the architectural requirements defined by the SoC integration team's DFx lead. In the past this process has taken weeks for projects to achieve. Chassis JTAG BFM addresses both of these issues. Since the BFM is architected with OVM methodology, the integration of the BFM into the validation environment is modular and independent of any underlying TAP network topology. The second issue is addressed with the help of the Chassis TAP Tool to organize and assemble the topology from the SystemRDL files. The DFx lead constructs the SoC specific fabric topology with the TAP tool that is consistent with Chassis DFx architectural requirements then imports the SystemRDL files from the IP-blocks. Once the overall data set is in place, the tool can output the six System Verilog header files that are directly consumed by the BFM. It is now ready to immediately direct targeted tests to validate the TAPs that are connected according to architectural requirements.

Providing an abstraction of the TAP network to a test writer could have been addressed by C++ APIs and integrating it with VCS through custom PLIs. However, it may require a deeper knowledge about the parser and DPI2SV to debug through custom language and inherent knowledge of integrating multiple flows (C++ compilation, integrating it in VCS, etc.). Modularity of such a solution is also a huge challenge that was faced in previous generation of SoCs. The current approach resolves all those concerns by implementing in SV/OVM and leveraging TAP SystemRDL standardization. In this instance, it is the advent of a standardized TAP machine readable format that did not exist before. Namely, TAP specific properties using the functional register SystemRDL language as a starting point. If all IPs delivered SystemRDL files, then the Chassis TAP Tool (CTT) becomes an aggregator and first line specification

checker to reduce dependencies on BFM for nuisance bug discovery. For example, at the click of a button CTT is performing unique name checking, unique slave ID code checking and security inversion checking then then the validation engineer can focus on corner cases involving network connection issues without spending valuable resource effort on these issues.

Note: The JTAG BFM requires information about the whole SoC network in System Verilog include files. The files contain information on several network attributes such as; unique enumerated TAP names, total number of TAPs, total number of hierarchies, tree structure connection information from root to leaf level. Other TAP specific data like Instruction Register width, number of opcodes, their width and associated data register length adds register access capabilities to the BFM.

2.2 Component Features

All of the BFM files are located in `ip-jtag-bfm/verif/tb/JtagBfm`

Features include:

- Support for SoC TapNetwork element capability; added Taplink Network capability
- Added Hierarchy, Secondary, and Hybrid mode support
- Added additional APIs
- Ability to move a TAP to a Secondary Port
- Ability to remove a TAP from the Network
- Can read all IDCODES in the entire Hierarchy
- Fixed HSDs pertaining to BFM usage in the context of TAP aware functionality
- Compatibility with Chassis Tap Tool Version 0.594 for capturing the Network hierarchy
- OVM compliant

The existing IP release is an OVM agent with the following components:

- Driver
- Sequencer
- InputMonitor
- OutputMonitor
- TrackerMonitor
- Analysis ports from monitors available
- No scoreboard provided as part of BFM
- Parameterized interface that contains Clock generator and mux for choosing the source of powergood_rst_b
- All APIs are same as in ip-tap-network
- Standard IEEE 1149.1 for the 5 pins are added as assertions in the JTAGBFM Pin Interface
- Except for the interface, no need of setting any parameters for making the BFM work
- Follows Test Island methodology
- Ace setup from this IP is reusable at integration level
- RTDR support is added
- Added UVM Support

- Added enDebug support
- Added TAPLINK support

2.3 Applications

For LoadIR*, Address will replace with Position of RTDR when i_JtagBfmCfg.use_rtdr_interface is 1.

Table 1. Summary Table of APIs in the BFM

API	End State
Reset	TLRS
ForceReset	TLRS
Idle	Current State
ForceClockGatingOff	Current State
Goto	Desired State
Tms_tdi_stream	Desired State
LoadDR	PADR
LoadDR_idle	RUTI
LoadDR_Pause	PADR
LoadIR	PAIR
LoadIR_idle	RUTI
MultipleTapRegisterAccess	RUTI
ExpData_MultipleTapRegisterAccess	PADR
ReturnTDO_ExpData_MultipleTapRegisterAccess	PADR
ReturnTDO_ExpData_MultipleTapRegisterAccessRuti	RUTI
LoadDR_E1DR	E1DR
LoadIR_E1IR	E1IR
CTV_ReturnTDO_ExpData_MultipleTapRegisterAccess	PADR

2.4 Features

Refer to section 2.2, Component Features.

2.5 Control

Not applicable to this IP/IPSS

2.6 Requirements

2.6.1 Specifications and Reference

Test case writing is done in SPF and the links to those are provided below. Topology RTL is generated using the DFTNet Fabric Generator tool from the Sandbox team.

Table 2. Reference Documents

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

Document Title	Location
HAS or EAD	$IP_ROOT/doc
Product Brief	$IP_ROOT/doc
Release Notes	$IP_ROOT/doc
Signal List	$IP_ROOT/doc

2.6.2 Compute Environment

The setup shown here is taken from https://intelpedia.intel.com/Ace:_Efficient_IP_Reuse

Figure 1. Ace Integration in SoC or other IP that uses this as a SubIP for OVM

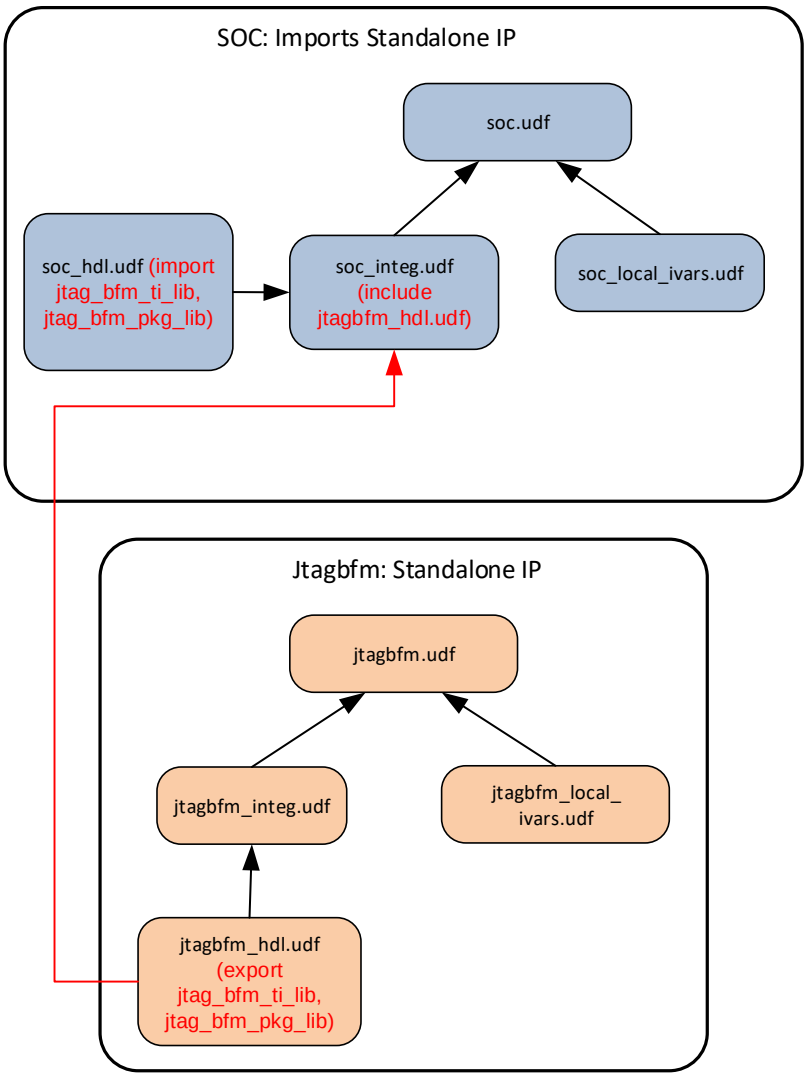
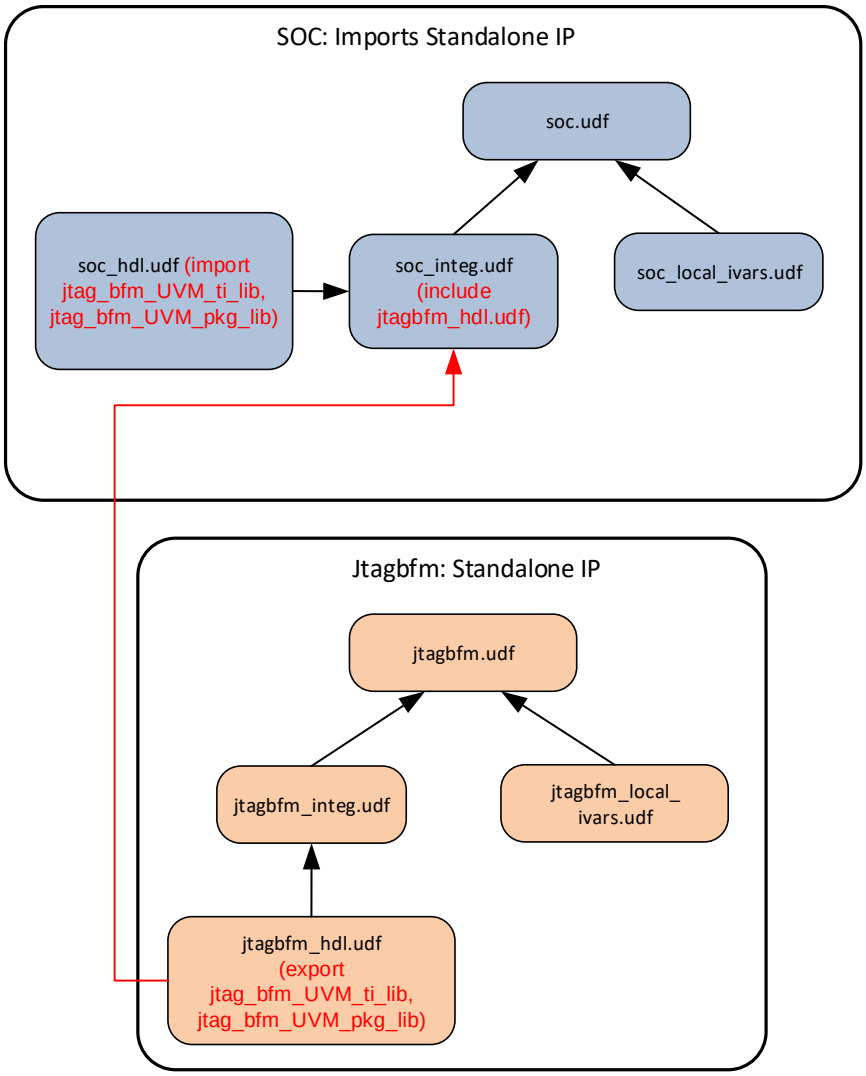


Figure 2. Ace Integration in SoC or other IP that uses this as a SubIP for UVM



Ace directory contains four udf files.

```
ace
|-- ace_test.mak
|-- clean_up.cfg
|-- clean_up_none.cfg
|-- jtagbfm.acerc
|-- jtagbfm.udf
|-- jtagbfm_hdl.udf
|-- jtagbfm_integ.udf
|-- jtagbfm_local_ivars.udf
|-- jtagbfm_postsim.pp
`-- lib
    `-- jtagbfm_model
        `-- RunModes.pm
```

Once the JTAGBFM is downloaded from IRR, SoC will have to import the JTAGBFM in their ace setup. The steps to do this are shown below.

1. Place the JtagBfm in a shared BFM directory available to all groups in the respective site.

   ```
   setenv JTAG_BFM_VER <shared_path>
   ```

2. Add jtagbfm to sip_shared_lib in the `SoC/IP_hdl.udf` file.

3. Define jtagbfm as a subscope in the `soc.acerc` file of the SoC. In this file, add this to the search path of the SOC.

4. In `soc_integ.udf`, add jtagbfm_integ.udf as an included udf.

5. In `soc_hdl.udf`, import and use the jtagbfm_ti_lib and jtagbfm_pkg_lib. Declare these as relevant dependent libs in the higher level testbench libs.

   ```
   run ace –show_udfs
   ```

   command to see if the jtagbfm_hdl.udf shows up.

   ```
   run ace -show_models
   ```

   to see if the model of the soc is clearly shown as the current scope.

2.6.3 System Verilog Packages

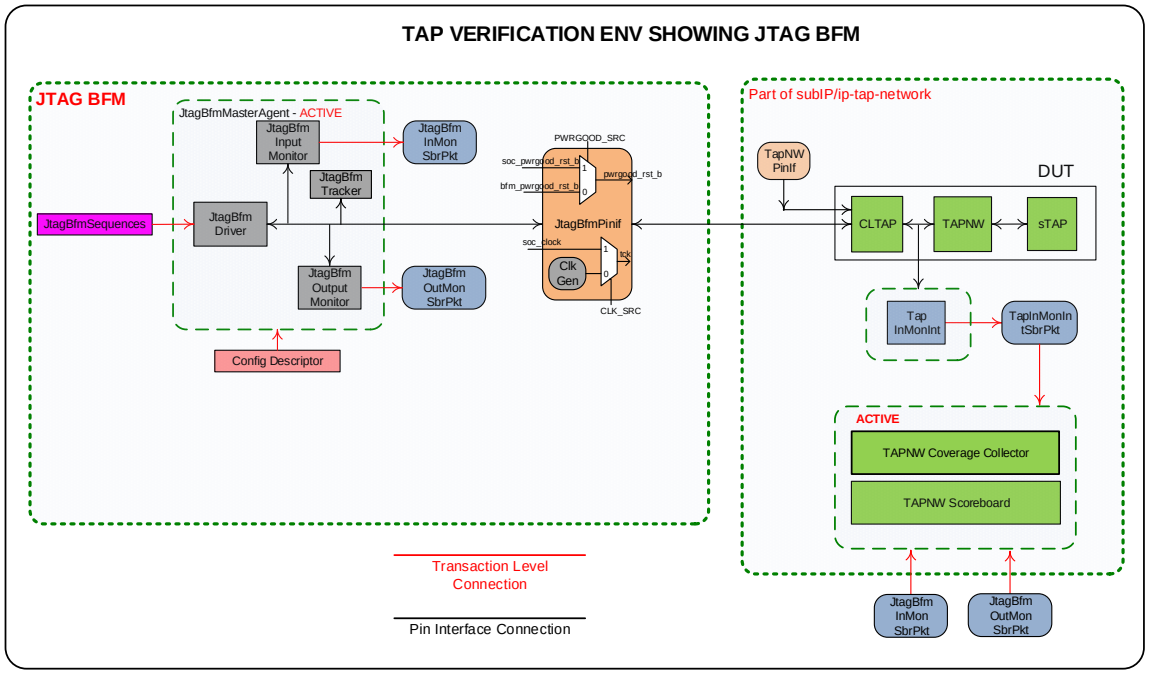
The following SV packages are used.

- ITPP Parser: used from central location
- OVM: used from central location
- Saola: used from central location
- XVM -- used from central location
- UVM -- used from central location.

2.7 Architecture

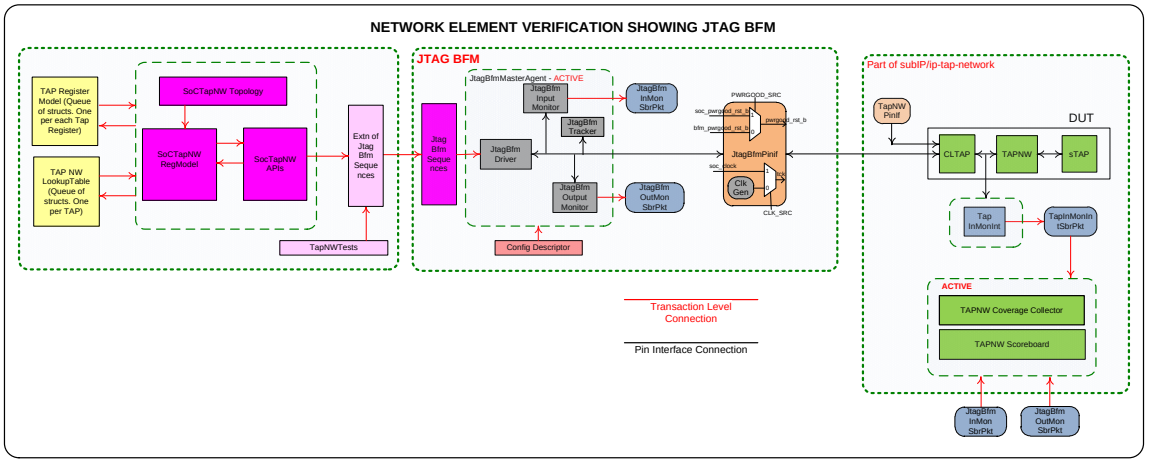
The Verification Env created for this BFM is taken from ip-tap-network (that is already released and available on IRR) and rearranged the components to make a separate Verification IP. The main change was to make OVM driver, monitors into a separate OVM agent and repackaging it as a separate IP. The Test Bench (TB) Architecture is illustrated below in Figure 3.

Figure 3. Testbench Architecture Block Diagram



SoCTapNW element uses a new OVM Object to define required components. This OVM object is used to build TapNW Model Look up Table (LUT) and a Register Model . JtagBfm sequences are extended to add new sequences. These new sequences will make use of Network Topology LUT, Regmodel and APIs for decoding user information. These sequences will make use of existing JtagBFM drivers, tracker and Monitors to drive the sequences and monitor the sequences. The Test Bench (TB) Architecture of SoC TapNetwork capability is illustrated below in Figure 4.

Figure 4. Testbench Architecture Block Diagram for SoC TapNetwork Element



2.7.1 Class/Component Descriptions

The existing IP release is an ovm agent with the following components.

- Driver
- Sequencer
- InputMonitor
- OutputMonitor
- TrackerMonitor

2.7.1.1 JTAG_BFM_Agent

Refer to section 2.7, Architecture, for agent information.

2.8 Usage of the BFM

Not applicable to this IP/IPSS

3 Getting Started

3.1 Setting Up a Testbench Environment

3.1.1 Setting up JTAG BFM

To run any test, replace <test_name> in the following command with one of the other test names listed below:

```
command_prompt> ace -x -t <test_name>
```

The list of tests available to run with this IP are listed.

- TapTestAllPrimary: places all sTAPs on the primary port and exercises each of them with all applicable mode combinations.
- TapTestAllSecondaryNormal: places all sTAPs on the secondary port and tests them in Normal mode.
- TapTestAllSecondaryNormalDecoupled: places all sTAPs on the secondary port. Few of them are in Normal mode while others are in Decoupled or Isolated mode.
- TapTestAllSecondaryNormalShadow: places all sTAPs on the secondary port. All except one will be in Normal mode. The other will be in Shadow mode.
- TapTestAllPriSecInParallel: places a few of the sTAPs on the primary port and a few on the secondary port. Then, all of the TAPs are configured in Normal mode. The primary and secondary ports are driven in parallel with independence.

The following commands show how to run the IP in standalone env.

```
setenv IP_ROOT $cwd
cd $IP_ROOT/scripts
source setup
ace -ccud
ace -x -t TapTestAllPrimary
```

3.1.1.1 Creating Custom Tests in SPF Format

```
# ITPP package poiner
cmd> setenv SPF_ITPP_VER
$IP_RELEASE/verif_misc/spf_itpp_pkg/spf_itpp_pkg_2015ww48
cmd> source scripts/reference/setup
cmd> gmake -f TLINKNetworkGen/Makefile gen_tlink_network
```

3.1.2 Setting up the BFM Tool

BFM tools version is MAT 1.6 compliant which supports following tool versions.

Table 3. Tool Versions in this Release

Tool Name	Version
verdi3	P-2019.06-SP1-1
Ovm	2.1.2_2b_ml
Vcsmx	P-2019.06-SP1-1
Uvm	1.2

Tool Name	Version
Xvm	1.0.5
Ace	2.06.02
Saola	16.2.21p1

3.1.3 Setting up ITPP Parser Package

1. Details about ITPP parser are available at: https://intelpedia.intel.com/SPF/ITPP_Reader
2. ITPP Parser for usage with this BFM should be used from central path. Contact Honcharik, Alexander J for ITPP Parser central location path and Integration support.
3. Read the `README.txt` file located in the central ITPP path for integration of ITPP Reader.
4. ITPP Reader Area: `/p/hdk/rtl/cad/x86-64_linux26/dteg/itpp_reader/<latest>`

Note: From 2016 release onwards, there is no need for assigning bfm_config_obj to spf_itpp_parser. As bfm_config_obj is directly taken with the use of get_config_obj in spf_itpp_sequencer.svh

3.2 Connecting DUT Signals to the Agent

SoC that uses this BFM has to instantiate the JTAGBFM test Island in their SoC TestIsland. Below code snapshot shows how this is done. A parameter in the JtagBfm TI needs to be set. This sets the scope of the BFM in the ovm_test hierarchy. In this example code snippet, the usage of how two instances of the BFM should be used is shown.

Summary: SoC/IP TI should contain the JTAG BFM TI.

IP TI should not have any connections for jtagbfm interface signals. This is due to the fact that these will be connected to SoC TAP NW RTL.

Figure 5. Code 1. Snapshot of BFM TestIsland Instantiation in SoC TI

```
  // Refer to file ip-jtag-bfm/subIP/ip-tap-network/verif/tb/TapTestIsland.sv for
guidance

`ifndef JTAG_IF_PARAMS_INST
 `define PRI_JTAG_IF_PARAMS_INST  .CLOCK_PERIOD(10000), .PWRGOOD_SRC(0), .CLK_SRC(1)
 `define SEC_JTAG_IF_PARAMS_INST  .CLOCK_PERIOD(15000), .PWRGOOD_SRC(0), .CLK_SRC(1)
`endif

module SocTestIsland(
                     JtagBfmIntf Primary_if,
                     JtagBfmIntf Secondary_if
                     );

   // Import the relevant packages
   import ovm_pkg::*;
   import JtagBfmPkg::*;

   // Instantiate the SubIP's TestIslands.
   // Set the hierarchy for the interface string so that it is unique.

   JtagBfmTestIsland #(.TAPVIF("ovm_test_top.Env.PriMasterAgent.*"),
                       `PRI_JTAG_IF_PARAMS_INST)
                        pri_JtagBfmTestIsland(Primary_if);

   JtagBfmTestIsland #(.TAPVIF("ovm_test_top.Env.SecMasterAgent.*")
                       `SEC_JTAG_IF_PARAMS_INST)
```

```
                        sec_JtagBfmTestIsland(Secondary_if);

   // Defparam is being deprecated. Hence avoid using it.
   // Set the hierarchy for the interface string so that it is unique.
   defparam pri_JtagBfmTestIsland.TAPVIF = "ovm_test_top.Env.PriMasterAgent.*";
   defparam sec_JtagBfmTestIsland.TAPVIF = "ovm_test_top.Env.SecMasterAgent.*";


endmodule : SocTestIsland
```

3.3 Instantiation of the BFM in SoCEnv

The BFM is an ovm_agent. Hence it needs to be instantiated in an Env that extends from ovm_env. Complete connections to Scoreboard are not shown in the code snippet below. If the JTAG BFM is the only agent in the env where the clock is used, then you will need to get the interface and assign it to the virtual interface.

Note: The SoCEnv can extend either Saola Env or OVM Env. If it extends Saola Env, then the BFM sequencer has to be registered with Saola Sequencer.

Figure 6. Code 2. Snapshot of BFM Instantiation in Env

```
// Refer to file ip-jtag-bfm/subIP/ip-tap-network/verif/tb/TapEnv.sv for guidance

`ifndef JTAG_IF_PARAMS_INST
 `define PRI_JTAG_IF_PARAMS_INST  .CLOCK_PERIOD(10000), .PWRGOOD_SRC(0), .CLK_SRC(1)
 `define SEC_JTAG_IF_PARAMS_INST  .CLOCK_PERIOD(15000), .PWRGOOD_SRC(0), .CLK_SRC(1)
`endif

class SocEnv extends sla_tb_env OR ovm_env;

    // Components of the environment
    JtagBfmMasterAgent   #(`PRI_JTAG_IF_PARAMS_INST) PriMasterAgent;
    JtagBfmMasterAgent   #(`SEC_JTAG_IF_PARAMS_INST) SecMasterAgent;

    // new Constructor & other variables not shown here

    // OVM build fn
    virtual function void build();
        super.build();
        …
        PriMasterAgent  = JtagBfmMasterAgent#(`PRI_JTAG_IF_PARAMS_INST)::type_id::
                          create("PriMasterAgent",this);
        SecMasterAgent  = JtagBfmMasterAgent#(`SEC_JTAG_IF_PARAMS_INST)::type_id::
                          create("SecMasterAgent",this);
        …
    endfunction : build

    // Connect the various env components here
    virtual function void connect();
        ovm_object temp;
        super.connect ();
        …
    endfunction : connect

endclass : SocEnv
```

3.4 Adding a Test

Follow these steps to add a new test.

1. Create a new file called as MyTapTests.sv. Start by duplicating existing class from the following file and modify it:
   `ip-jtag-bfm/subIP/ip-tap-network/verif/tests/TapTests.sv`

2. Create a new file called as MyTapSequence.sv. Start by duplicating existing class from the following file and modify it:
   `ip-jtag-bfm/subIP/ip-tap-network/verif/tb/TapSeqLib.sv`

3. Enhance your Sequence by using the APIs listed in Table 1 .

4. Code 3 (Figure 7) shows how your new test will look. The pink font signifies the places where you need to make changes.

5. Run your test:d

   ```
   ace -cc -x -t MyNewTest
   ```

Note: The Sequence Class should come first in the compilation order.

Figure 7. Code 3. New Test Sequence Class

```
// Refer to ip-jtag-bfm/subIP/ip-tap-network/verif/tb/TapSeqLib.sv for guidance
class MyNewSequence extends JtagBfmSequences;

    JtagBfmSeqDrvPkt Packet;
    function new(string name = "MyNewSequence");
        super.new(name);
        Packet = new;
    endfunction : new

    `ovm_sequence_utils(MyNewSequence, JtagBfmSequencer)

    virtual task body();

        // following cycle asserts the powergoodrst_b
        ForceReset(RST_PWRGUD, LOW);
        // signal holds the reset for 10 clock
        Idle(10);
        // release the reset
        ForceReset(RST_PWRGUD, HIGH);

        // following cycle pulses a trst_b
        Reset(RST_TRST);

        // following cycle configures none are on secondary
        MultipleTapRegisterAccess(2'b00,8'h10,8'h0,8,8);
        // following cycle configures 8 TAP's on Primary in Normal Mode
        MultipleTapRegisterAccess(2'b00,8'h11,16'h5555,8,16);

        // following cycle configures all 1+8 TAP's in Bypass mode
        // It passes all zeros in the chain
        MultipleTapRegisterAccess(2'b00,72'hFF_FF_FF_FF_FF_FF_FF_FF_FF,
                                        64'h0000_0000_0000_0000,72,64);
        // following cycle reads IDCODE from mTAP and SLVIDCODE from all 8 TAP's.
        MultipleTapRegisterAccess(2'b00,72'h02_0C_0C_0C_0C_0C_0C_0C_0C,
                                        288'h0000_0000_0000_0000,72,288);

        // following cycle reads IDCODE from all 1+8 TAP's
        LoadIR(NO_RST,16'hFF_FF,16'h01_01,16'hFFFF,16);

        // Stay in RUTI for 10 clk cycles.
        Goto(RUTI,10);

        LoadDR(NO_RST,        // Reset Mode
               32'h0123_4567,  // Data into the Register
               32'h048D_159C,  // Expected Data out of the Register
               32'hFFFF_FFFF,  // Mask, here all bits are compared
```

```
            32);            // Length of Data transaction
      LoadIR(NO_RST,        // Reset Mode
            16'hFF_FF,      // Address into the Register
            16'h01_01,      // Expected Address out of the Register
            16'hFFFF,       // Mask, here all bits are compared
            16);            // Length of Address transaction

      // In case of Multiple TAP's, concatenate all IR's & DR's
      // Here Two TAP's are in normal mode
      ExpData_MultipleTapRegisterAccess(NO_RST, // Reset Mode
            16'hFF_FF,      // Concatenated Address of IR
            16,             // Combined Address length of all IR's
            32'h0123_4567,  // Data into the Register
            32'h048b_159C,  // Expected Data out of the Register
            32'hFFF9_FFFF,  // Mask. Here bits 18, 19 are not compared.
            32);            // Length of Data transaction

      tms_tdi_stream(22'hFE0000, // TMS Stream
                     22'h0,      // TDI Stream
                     22);        // Length of Stream

      // In case user needs the TDO back into a string or a local variable
      logic [TOTAL_DATA_REGISTER_WIDTH-1:0] returnredtdo;
      ReturnTDO_ExpData_MultipleTapRegisterAccess(NO_RST, // Reset Mode
            16'hFF_FF,      // Concatenated Address of IR
            16,             // Combined Address length of all IR's
            32'h0123_4567,  // Data into the Register
            32'h048b_159C,  // Expected Data out of the Register
            32'hFFF9_FFFF,  // Mask. Here bits 18, 19 are not compared.
            32,             // Length of Data transaction
            returnredtdo);  // Returned TDO. Lower Dword should be 32'h048b_159C
     CTV_ReturnTDO_ExpData_MultipleTapRegisterAccess(NO_RST, //Reset Mode
            8'hA0,          // Concatenated Address of IR
            8,              // Combined Address length of all IR's
            32'hAAAAAAAA,   // Data into the Register
            32'hCCCCCCCC,   // Expected Data out of the Register
            32'hFFFFFFFF,   // Mask. Here bits 18, 19 are not compared
            32'h000F0000,   // Mask. Here bits 18, 19 are not displayed in log file
            32,             // Length of Data transaction
            returnedtdo);   // Returned TDO. Lower Dword should be 32'hCCCCCCCC

      // Async Remote TDR access whose address = ABCD
      LoadIR(NO_RST,        // Reset Mode
            16'hABCD,       // Address into the Register
            16'h0001,       // Expected Address out of the Register
            16'hFFFF,       // Mask, here all bits are compared
            16);            // Length of Address transaction
      LoadDR_Pause(NO_RST,  // Reset Mode
            32'h0123_4567,  // Data into the Register
            32'h048D_159C,  // Expected Data out of the Register
            32'hFFFF_FFFF,  // Mask, here all bits are compared
            32,             // Length of Data transaction
            10);            // PADR cycle wait time
      Goto(UPDR,10);


   endtask : body

endclass : MyNewSequence
```

Figure 8. Code 4. New Test Class

```
  // Refer to content from file TapTests.sv for guidance

class MyNewTest extends SocBaseTest;

   `ovm_component_utils(MyNewTest) // Factory Registration

    MyNewSequence inst_MyNewSequence;
```

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

    virtual function void build(); // Build Function
        super.build();
    endfunction : build




    virtual task run(); // Run Task
        ovm_report_info("MyNewTest"," MyNewTest Started!!!");
        inst_MyNewSequence = new("My New test");
        inst_MyNewSequence.start(Env.PriMasterAgent.Sequencer);
        ovm_report_info("MyNewTest"," MyNewTest Completed!!!");
        global_stop_request();
    endtask : run

endclass : MyNewTest
```

3.5 Naming and Instantiating Testbench Components

Steps to instantiate the BFM are shown below.

Figure 9. Code 5. Snapshot of BFM PinIF in Testbench Top

```
   module soc_tb_top();
    //-------------------------------------------------------------------
    // In a module where the BFM is instantiated, an option to drive
    // a higher level powergood_rst_b can be used by setting the parameter
    // PWRGOOD_SRC = 1 in JTAG BFM interface along with driving this
    // soc_powergood_rst_b. It is declared here as a placeholder.
    //
    // Also the source for clock can be the JtagBFM or another source.
    // In the latter case, CLK_SRC should be set to 1 and the input clock
    // should be passed.
    //-------------------------------------------------------------------
    reg soc_powergood_rst_b;
    reg soc_clock;

    // Instantiate the BFM's Pin Interface
JtagBfmIntf #(.CLOCK_PERIOD (10000),
              .PWRGOOD_SRC  (0),
              .CLK_SRC      (0))
             i_JtagBfmIntf (soc_powergood_rst_b, soc_clock);

    // Instantiate the SoC Test Island that would contain the JTAGBFM TI
    SocTestIsland i_SocTestIsland( …
                   .Jtag_if (i_JtagBfmIntf),
                   …
                  );
    //Top Module Instantiation
    soc i_soc  (  …
                  …
                  .powergoodrst_b       (Primary_if.powergood_rst_b)
                  .ftap_tck             (Primary_if.tck),
                  .ftap_tms             (Primary_if.tms),
                  .ftap_trst_b          (Primary_if.trst_b),
                  .ftap_tdi             (Primary_if.tdi),
                  .atap_tdo             (Primary_if.tdo),
                  .atap_tdoen           (),
                  .ftap_slvidcode       (32'h1234_5679)
                  …
                  …
```

```
                    );

    // SoC Test (Program Block)
    SoCTest i_SoCTest();

endmodule: soc_tb_top
```

3.6 Extending the Testbench for Test Execution

3.6.1 Defines used in the BFM

Table 4. Test Island Connections

Name of `define	Default State	Description
OVM_MAX_STREAMBITS	Not defined	`ace –cc –vlog_opts "+define+OVM_MAX_STREAMBITS=1000000"` Used when the number of DR shifts more that 4096 is required. Default is 4096. Added in ver 1.1.
JTAG_BFM_DEBUG_MSG	Not defined	`ace –cc –vlog_opts "+define+JTAG_BFM_DEBUG_MSG"` Will print all the debug messages of the BFM. Default is 0. Added in ver 1.6
USE_GENERATED_FILES_FOR_JTAGBFM	Not defined	`ace –cc –vlog_opts "+define+USE_GENERATED_FILES_FOR_JTAGBFM"` Used when files specific to a new configuration of the TAP hierarchy are specified. Default is 0. Added in ver 1.6
CHASSIS_JTAGBFM_TRACKER_WIDTH	Not defined	`ace –cc –vlog_opts "+define+ CHASSIS_JTAGBFM_TRACKER_WIDTH=1024"` User defined width of the tracker can be specified. Default is 128. Added in ver 1.9
CHASSIS_JTAG_WORD_LENGTH	Not defined	`ace –cc –vlog_opts "+define+ CHASSIS_JTAGBFM_TRACKER_WIDTH=1024"` User defined width of the tracker can be specified. Default is 128. Added in ver 2.0
CHASSIS_JTAGBFM_USE_PARAMETERIZED_CLASS	Not defined	`ace –cc –vlog_opts "+define+ CHASSIS_JTAGBFM_USE_PARAMETERIZED_CLASS =1"` Parameterized class to eliminate certain vcs warnings and fullchip errors resulting in JtagBfmPinInterface being parameterized whereas the virtual interface in classes not being so. Default is 0. Added in ver 2.1
SNPS201412BC	Not defined	Pass this switch for Older VCS Versions (<F-2011) to ovoid compilation Error in "export" package.

Name of `define	Default State	Description
JTAG_BFM_TAPLINK_MODE	Not defined	Pass this switch to enable Taplink capability in the JTAG BFM.
JTAGBFM_EN_DFTB_HTAP	Not defined	`ace -cc -vlog_opts "+define+ JTAGBFM_EN_DFTB_HTAP"` Pass this switch to enable DFTB HTAP(Tap Network) capability in the JTAG BFM. Added in ver 3.8
GENERATED_FILES_FOR_JTAGBFM		Set this variable for Taplink/TapNetwork Topology Files `setenv GENERATED_FILES_FOR_JTAGBFM <Path of TAPLINK/TAPNETWORK topology generated by CTT tool>`
JTAGBFM_ENDEBUG_110REV	Not Defined	Pass this switch to get EnDebug 110 Revision compatibility.Jtagbfm supports EnDebug 120 Revision by default.
DTEG_UVM_EN	Not defined	Pass this define along with DTEG_XVM_EN to compile JtagBfm in XVM mode for OVM code. `-vlog_opts "+define+DTEG_OVM_EN+DTEG_XVM_EN"`
DTEG_OVM_EN	Not defined	Pass this define along with DTEG_XVM_EN to compile JtagBfm in XVM mode for UVM code. `-vlog_opts "+define+DTEG_UVM_EN+DTEG_XVM_EN"`
DTEG_XVM_EN	Not defined	Pass this define to compile the JtagBfm in XVM mode.
JTAGBFM_TDO_1_ASSERT_EN	Not defined	Pass this define to get assertion for check TDO is 1 in non-shift states.

INTEL_SVA_OFF	Not defined	Pass this global define to turn off assertions.
		`-vlog_opts "+define+INTEL_SVA_OFF"`
		`Added in ver R3.9`

JTAGBFM_SVA_OFF	Not defined	Pass this bfm specific define to turn off assertions. -vlog_opts "+define+JTAGBFM_SVA_OFF" Added in ver R3.9
DFX_COVERAGE_ON	Not defined	Pass this define to turn on BFM coverpoints. -vlog_opts "+define+DFX_COVERAGE_ON" Added in ver R3.9
JTAG_BFM_AES_256	Not defined	`ace -cc -vlog_opts "+define+ JTAG_BFM_AES_256"` Pass this switch to enable 256-bit AES encryption/decryption endebug capability in the JTAG BFM. Added in ver 4.1.

3.6.2 Description of API Arguments for Standalone BFM

- To reset the TAP:

  This pulses a reset.

  Reset (ResetMode):

  - 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. It will also reset tap_rtdr_irdec[N-1:0]

      Asserts and de-asserts tap_rtdr_prog_rst_b [N-1:0]
    - 2'b10: Enables the TMS for 5 clock cycles. It will also reset tap_rtdr_irdec[N-1:0]
    - 2'b11: Asserts powergood_rst_b. It will also reset tap_rtdr_irdec[N-1:0]

      Asserts and de-asserts tap_rtdr_prog_rst_b [N-1:0]
- To Force a reset to any polarity:

  This does not pulse a reset. This task does not require TCK to be toggling.

  ForceReset(ResetMode, ForceState);

  - 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: Reserved.
    - 2'b11: powergood_rst_b
  - 2nd argument: ForceState
    - 1'b0: Force state to be in 0.
    - 1'b1: Force state to be in 1.
- To reset the TAP:

  This pulses a reset.

  - Reset (ResetMode);
    - 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. It will also reset tap_rtdr_irdec[N-1:0]

      Asserts and de-asserts tap_rtdr_prog_rst_b [N-1:0]
    - 2'b10: Enables the TMS for 5 clock cycles. It will also reset tap_rtdr_irdec[N-1:0]
    - 2'b11: Asserts powergood_rst_b. It will also reset tap_rtdr_irdec[N-1:0]

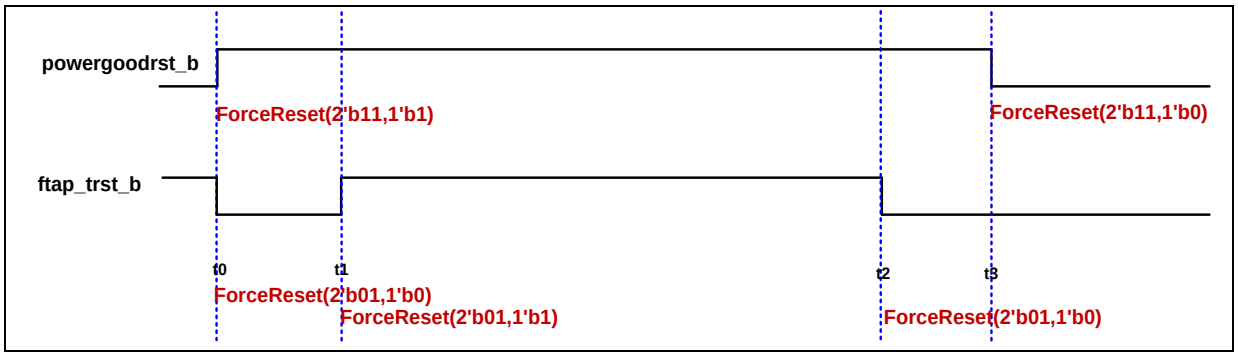
      Asserts and de-asserts tap_rtdr_prog_rst_b [N-1:0]
- To Force a reset to any polarity:

  This does not pulse a reset. This task does not require TCK to be toggling.

- ForceReset(ResetMode, ForceState);
  - 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: Reserved.
  - 2'b11: powergood_rst_b
  - 2nd argument: ForceState
  - 1'b0: Force state to be in 0.
  - 1'b1: Force state to be in 1.

Figure 10. Example waveform showing ForceReset task behavior



To access registers:
This function is used in the Sequence file.

MultipleTapRegisterAccess(ResetMode, Address, Data, Address_length, Data_length);

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.

To access registers:
This function is used in the Sequence file.

ExpData_MultipleTapRegisterAccess(ResetMode, Address, addr_len,
Data, Expected_Data, Mask_Data, Data_len);

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.

To access registers:
These functions are used in the Sequence file.

ReturnTDO_ExpData_MultipleTapRegisterAccess(ResetMode, Address, addr_len,
Data, Expected_Data, Mask_Data, Data_len,
Returnedtdo);

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 3.

To access registers:
These functions are used in the Sequence file.

CTV_ReturnTDO_ExpData_MultipleTapRegisterAccess(ResetMode, Address, addr_len,
Data, Expected_Data, Mask_Data, Mask_Capture,
Data_len, Returnedtdo);

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: Mask_Capture: This field is used to display bits in log file for Returnedtdo. This field has Width as same as Mask_Data.

1 : Display in log

0 : Will not display in log.

8th argument: Data_len: The data width.

9th argument: ReturnedTDO: TDO that comes back. This should be a locally declared variable in the test. Usage is shown in Code 3.

To access registers:
These functions are used in the Sequence file.

ReturnTDO_ExpData_MultipleTapRegisterAccessRuti(ResetMode, Address, addr_len, Data, Expected_Data, Mask_Data, Data_len, Returnedtdo);

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 3.

To go to a specific FSM state of the TAP using the shortest path:

Goto(FSMstate, count);

1st argument: FSMstate: The Tap_state in FSM. The following state table is given for reference.

TLRS = 4'b0000, // Test Logic Reset State

RUTI = 4'b1000, // Run test Idle State

SDRS = 4'b0001, // Select DR Scan State

CADR = 4'b0010, // Capture DR State

SHDR = 4'b0011, // Shift DR State

E1DR = 4'b0100, // Exit1 DR State

PADR = 4'b0101, // Pause DR State

E2DR = 4'b0110, // Exit2 DR State

UPDR = 4'b0111, // Update DR State

SIRS = 4'b1001, // Select IR Scan State

CAIR = 4'b1010, // Capture IR State

SHIR = 4'b1011, // Shift IR State

E1IR = 4'b1100, // Exit1 IR State

PAIR = 4'b1101, // Pause IR State

E2IR = 4'b1110, // Exit2 IR State

UPIR = 4'b1111; // Update IR State

2nd argument: count: This is the number of cycles that it must remain in the state.

To go to a specific FSM state of the TAP using the shortest path:

tms_tdi_stream(tms_stream, tdi_stream, width);

1st argument: tms_stream

This is the stream of TMS to help the user navigate the FSM in desired state.

2nd argument: tdi_stream

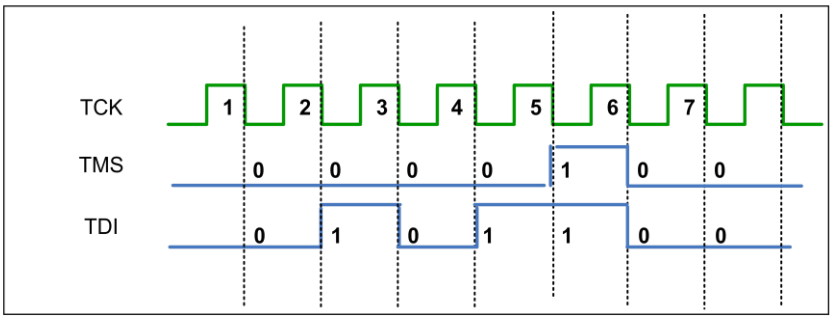
This stream of TDI is driven in parallel to the TMS stream.

3rd argument: width

This argument represents the length of the TMS and TDI streams. The maximum value is 4096.

Figure 11. TDI, TMS string

```
tms_tdi_stream(7'b_0000100, 7'b_0101100, 7); Looks like below…
```



To gate clock for count value number of clock cycles before tuning it on.

Idle(count);

1st argument: count

This is an integer value of the count.

To access TAP registers:

LoadIR(ResetMode, Address, Expected_Address, Mask_Address, addr_len);

- i_jtagbfmcfg.use_rtdr_interface is 0

  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: Expected_ Address: The expected address of IR. The previously stored value of IR will come out.

  4th argument: Mask_Address: The exact bits that you want to compare between Address and Expected_Address.

  5th argument: addr_len: The length of IR address.

- i_jtagbfmcfg.use_rtdr_interface is 1

  1st argument: ResetMode

  This 2-bit mode defines the way the user wants to reset the TAP.

  2'b00: No reset.

  2'b01: Asserts tap_rtdr_prog_rst_b for RTDR register based on irdec position. It will also clears tap_rtdr_irdec for that position.

  2'b10: NA.

  2'b11: NA.

  2nd argument: Address: irdec position of RTDR register.

  3rd argument: Expected_Address: NA.

  4th argument: Mask_Address: NA.

  5th argument: addr_len: NA.

To access TAP registers:

LoadDR(ResetMode, Data, Expected_Data, Mask_Data, data_len);

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. NA for RTDR

2'b10: Enables the TMS for 5 clock cycles. NA for RTDR

2'b11: Asserts powergood_rst_b. NA for RTDR

2nd argument: Data: The data to be loaded into the selected IR.

3rd argument: Expected_Data: The expected data from the selected IR.

4th argument: Mask_Data: The exact bits that you want to compare between Data and Expected_Data.

5th argument: data_len: The length of expected data of IR.

To access TAP registers:

LoadDR_Pause(ResetMode, Data, Expected_Data, Mask_Data, data_len, pause_len);

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. NA for RTDR

2'b10: Enables the TMS for 5 clock cycles. NA for RTDR

2'b11: Asserts powergood_rst_b. NA for RTDR

2nd argument: Data: The data to be loaded into the selected IR.

3rd argument: Expected_Data: The expected data from the selected IR.

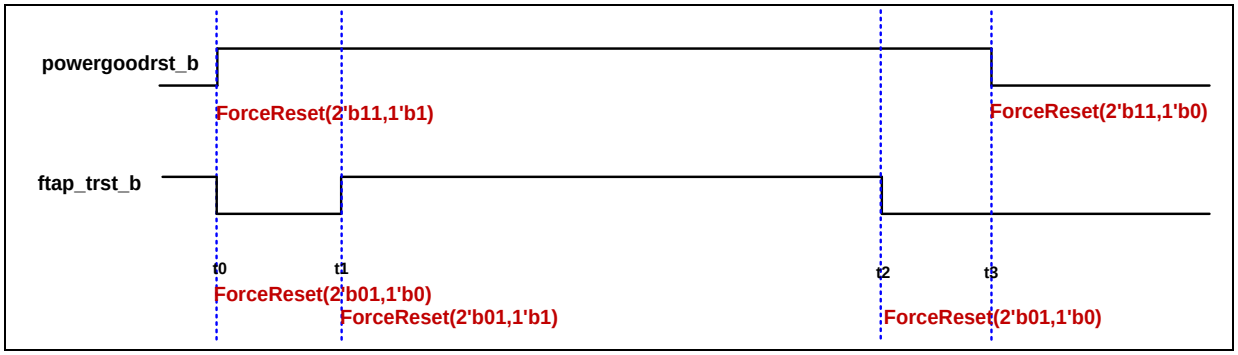
4th argument: Mask_Data: The exact bits that you want to compare between Data and Expected_Data.

5th argument: data_len: The length of expected data of IR.

6th argument: pause_len: The number of cycles to remain in PADR.

For E.g. if pause_len =10 and data_len = 32 ..then following state transition will happen. RUTI(1) → SDRS (1) → CADR (1) → E1DR (1) → PADR (10) → E2DR (1) → SHDR (32) → E1DR (1) →PADR (10). A Goto(UPDR,1) need to be given after this for an update.

Figure 12. Example waveform showing ForceReset task behavior



To access registers:
This function is used in the Sequence file.

MultipleTapRegisterAccess(ResetMode, Address, Data, Address_length, Data_length);

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.

To access registers:
This function is used in the Sequence file.

ExpData_MultipleTapRegisterAccess(ResetMode, Address, addr_len,
Data, Expected_Data, Mask_Data, Data_len);

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.

To access registers:
These functions are used in the Sequence file.

ReturnTDO_ExpData_MultipleTapRegisterAccess(ResetMode, Address, addr_len,
Data, Expected_Data, Mask_Data, Data_len,
Returnedtdo);

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 3.

To access registers:
These functions are used in the Sequence file.

CTV_ReturnTDO_ExpData_MultipleTapRegisterAccess(ResetMode, Address, addr_len, Data, Expected_Data, Mask_Data, Mask_Capture, Data_len, Returnedtdo);

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: Mask_Capture: This field is used to display bits in log file for Returnedtdo. This field has Width as same as Mask_Data.

1 : Display in log

0 : Will not display in log.

8th argument: Data_len: The data width.

9th argument: ReturnedTDO: TDO that comes back. This should be a locally declared variable in the test. Usage is shown in Code 3.

To access registers:
These functions are used in the Sequence file.

ReturnTDO_ExpData_MultipleTapRegisterAccessRuti(ResetMode, Address, addr_len, Data, Expected_Data, Mask_Data, Data_len, Returnedtdo);

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 3.

To go to a specific FSM state of the TAP using the shortest path:

Goto(FSMstate, count);

1st argument: FSMstate: The Tap_state in FSM. The following state table is given for reference.

```
TLRS    = 4'b0000,  // Test Logic Reset State
RUTI    = 4'b1000,  // Run test Idle State
SDRS    = 4'b0001,  // Select DR Scan State
CADR    = 4'b0010,  // Capture DR State
SHDR    = 4'b0011,  // Shift DR State
E1DR    = 4'b0100,  // Exit1 DR State
PADR    = 4'b0101,  // Pause DR State
E2DR    = 4'b0110,  // Exit2 DR State
UPDR    = 4'b0111,  // Update DR State
SIRS    = 4'b1001,  // Select IR Scan State
CAIR    = 4'b1010,  // Capture IR State
SHIR    = 4'b1011,  // Shift IR State
E1IR    = 4'b1100,  // Exit1 IR State
PAIR    = 4'b1101,  // Pause IR State
E2IR    = 4'b1110,  // Exit2 IR State
UPIR    = 4'b1111;  // Update IR State
```

2nd argument: count: This is the number of cycles that it must remain in the state.

To go to a specific FSM state of the TAP using the shortest path:

tms_tdi_stream(tms_stream, tdi_stream, width);

1st argument: tms_stream

This is the stream of TMS to help the user navigate the FSM in desired state.

2nd argument: tdi_stream

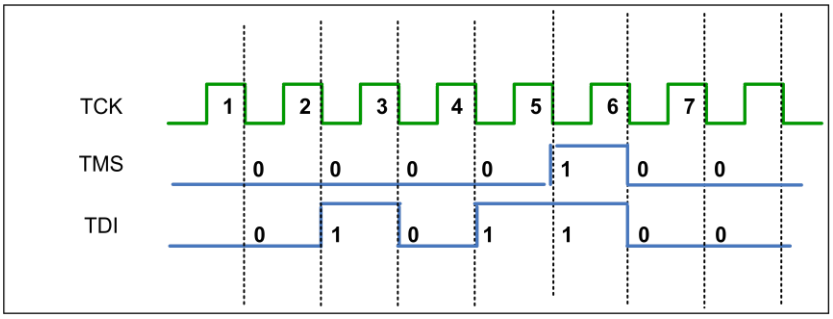
This stream of TDI is driven in parallel to the TMS stream.

3rd argument: width

This argument represents the length of the TMS and TDI streams. The maximum value is 4096.

Figure 13. TDI, TMS string

```
tms_tdi_stream(7'b_0000100, 7'b_0101100, 7); Looks like below…
```



To gate clock for count value number of clock cycles before tuning it on.

Idle(count);

1st argument: count

This is an integer value of the count.

To access TAP registers:

LoadIR(ResetMode, Address, Expected_Address, Mask_Address, addr_len);

- i_jtagbfmcfg.use_rtdr_interface is 0

  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: Expected_ Address: The expected address of IR. The previously stored value of IR will come out.

  4th argument: Mask_Address: The exact bits that you want to compare between Address and Expected_Address.

  5th argument: addr_len: The length of IR address.

- i_jtagbfmcfg.use_rtdr_interface is 1

  1st argument: ResetMode

  This 2-bit mode defines the way the user wants to reset the TAP.

  2'b00: No reset.

  2'b01: Asserts tap_rtdr_prog_rst_b for RTDR register based on irdec position. It will also clears tap_rtdr_irdec for that position.

  2'b10: NA.

  2'b11: NA.

  2nd argument: Address: irdec position of RTDR register.

3rd argument: Expected_Address: NA.

4th argument: Mask_Address: NA.

5th argument: addr_len: NA.

To access TAP registers:

LoadDR(ResetMode, Data, Expected_Data, Mask_Data, data_len);

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. NA for RTDR

2'b10: Enables the TMS for 5 clock cycles. NA for RTDR

2'b11: Asserts powergood_rst_b. NA for RTDR

2nd argument: Data: The data to be loaded into the selected IR.

3rd argument: Expected_Data: The expected data from the selected IR.

4th argument: Mask_Data: The exact bits that you want to compare between Data and Expected_Data.

5th argument: data_len: The length of expected data of IR.

To access TAP registers:

LoadDR_Pause(ResetMode, Data, Expected_Data, Mask_Data, data_len, pause_len);

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. NA for RTDR

2'b10: Enables the TMS for 5 clock cycles. NA for RTDR

2'b11: Asserts powergood_rst_b. NA for RTDR

2nd argument: Data: The data to be loaded into the selected IR.

3rd argument: Expected_Data: The expected data from the selected IR.

4th argument: Mask_Data: The exact bits that you want to compare between Data and Expected_Data.

5th argument: data_len: The length of expected data of IR.

6th argument: pause_len: The number of cycles to remain in PADR.

For E.g. if pause_len =10 and data_len = 32 ..then following state transition will happen. RUTI(1) → SDRS (1) → CADR (1) → E1DR (1) → PADR (10) → E2DR (1) → SHDR (32) → E1DR (1) →PADR (10). A Goto(UPDR,1) need to be given after this for an update.

3.6.3 Sequences

The directory `verif/tb/Converged_JtagBfm/Converged_SampleTests/test_sequences` contains sample test sequences. The test sequences are part of a test package.

In a hierarchical network, a child tap will always be on the same port as its parent with reference to CLTAP. In other words if a Parent is on CLTAP's secondary then the child tap will also be on CLTAP's secondary.

Sequences for Hierarchical Mode: Config Sequence

To configure TAP in hierarchical mode sequences are written as shown below. We need two sequences, one to configure Tap in secondary mode and other to access Tap using secondary interface. Task PutTapOnSecondary is used as part of configuration to place intended Tap in secondary.

Figure 14. Code 6. Secondary Hierarchical Mode: Sample Code for Configuration Sequence

```
2337 class Tap5levelHierSequenceTrySecConfigNew #(parameter EN_REGISTER_PRESENCE_CHECK = 1,
2338                                               parameter Tap_t TAP = STAP0) extends JtagBfmSoCTapNwSequences;
2339
2340    JtagBfmSeqDrvPkt Packet;
2341     function new(string name = "Tap5levelHierSequenceTrySecConfigNew");
2342         super.new(name);
2343         Packet = new;
2344     endfunction : new
2345
2346     `ovm_sequence_utils(Tap5levelHierSequenceTrySecConfigNew, JtagBfmSequencer)
2347
2348     virtual task body();
2349        $display("EN_REGISTER_PRESENCE_CHECK = %0d", EN_REGISTER_PRESENCE_CHECK);
2350        $display("1, TAP = %0s", TAP);
2351        BuildTapDataBase();
2352        Reset(2'b11);
2353        PutTapOnSecondary(TAP);
2354     endtask : body
2355
2356 endclass : Tap5levelHierSequenceTrySecConfigNew
```

Sequences for Hierarchical Mode: Data Sequence

To access any Opcode form Tap placed on secondary, TapAccess task is used the way it is in primary. The following example shows how to access register from Tap which is place on secondary.

Figure 15. Code 7. Secondary Hierarchical Mode: Sample Code for Data Sequence

```
2411 class Tap5levelHierSequenceTrySecondaryNew #(parameter EN_REGISTER_PRESENCE_CHECK = 1,
2412                                               parameter Tap_t TAP = STAP0) extends JtagBfmSoCTapNwSequences;
2413
2414    Tap_t tap = TAP;
2415    JtagBfmSeqDrvPkt Packet;
2416     function new(string name = "Tap5levelHierSequenceTrySecondaryNew");
2417         super.new(name);
2418         Packet = new;
2419     endfunction : new
2420
2421     `ovm_sequence_utils(Tap5levelHierSequenceTrySecondaryNew, JtagBfmSequencer)
2422
2423     virtual task body();
2424     logic [WIDTH_OF_EACH_REGISTER-1:0] ReturnTdo;
2425        TapAccess(.Tap(TAP),
2426                  .Opcode('h0C),
2427                  .ShiftIn(32'h0),
2428                  .ShiftLength(32),
2429                  .EnRegisterPresenceCheck(EN_REGISTER_PRESENCE_CHECK),
2430                  .ReturnTdo(ReturnTdo));
2431     endtask : body
2432
2433 endclass : Tap5levelHierSequenceTrySecondaryNew
```

Sequences for Hierarchical mode: Test

The test is written as shown below. It shows how the Primary and Secondary instance of the BFM are used.

Figure 16. Code 8. Secondary Hierarchical Mode: Sample Code for Test

```
533 class TapTestTry5levelhierSecondary extends TapBaseTest;
534    `ovm_component_utils(TapTestTry5levelhierSecondary)
535
536    Tap5levelHierSequenceTrySecConfig #(1, STAP2) ST_CONFIG;
537    Tap5levelHierSequenceTrySecondary #(1, STAP2) ST_SECONDARY;
538
539    TapSequenceReset RESETPRIMARY, RESETSECONDARY;
540    TapSequenceGoToRuti SECONDARY_RUTI;
541
542    function new (string name = "TapTestTry5levelhierSecondary",
543       ovm_component parent = null);
544       super.new(name,parent);
545    endfunction : new
546
547    virtual function void build();
548       super.build();
549    endfunction : build
550
551    virtual task run();
552       ovm_report_info("TapTestTry5levelhierSecondary","Test Starts!!!");
553       RESETPRIMARY = new();
554       RESETSECONDARY = new();
555       ST_CONFIG = new("ALL Primary Mode");
556       ST_SECONDARY = new("ALL secondary Mode");
557       SECONDARY_RUTI = new();
558
559       RESETPRIMARY.start(Env.PriMasterAgent.Sequencer);
560       RESETSECONDARY.start(Env.SecMasterAgent.Sequencer);
561       SECONDARY_RUTI.start(Env.SecMasterAgent.Sequencer);
562
563       ST_CONFIG.start(Env.PriMasterAgent.Sequencer);
564       ST_SECONDARY.start(Env.SecMasterAgent.Sequencer);
565       ovm_report_info("TapTestTry5levelhierSecondary",
566          "TapTestTry5levelhierSecondary Completed!!!");
567       global_stop_request();
568    endtask : run
569
570 endclass : TapTestTry5levelhierSecondary
```

Figure 17. Illustration of Usage of Multiple Taps being Active

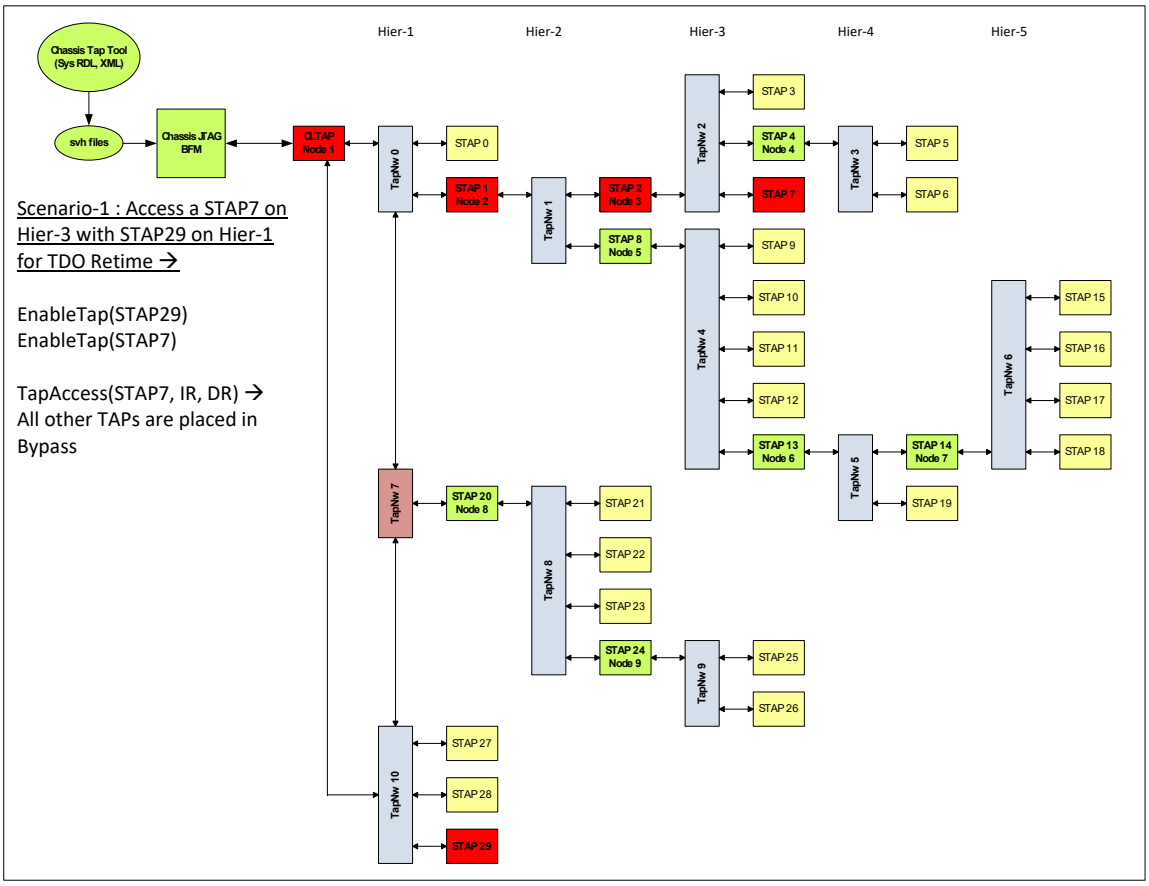
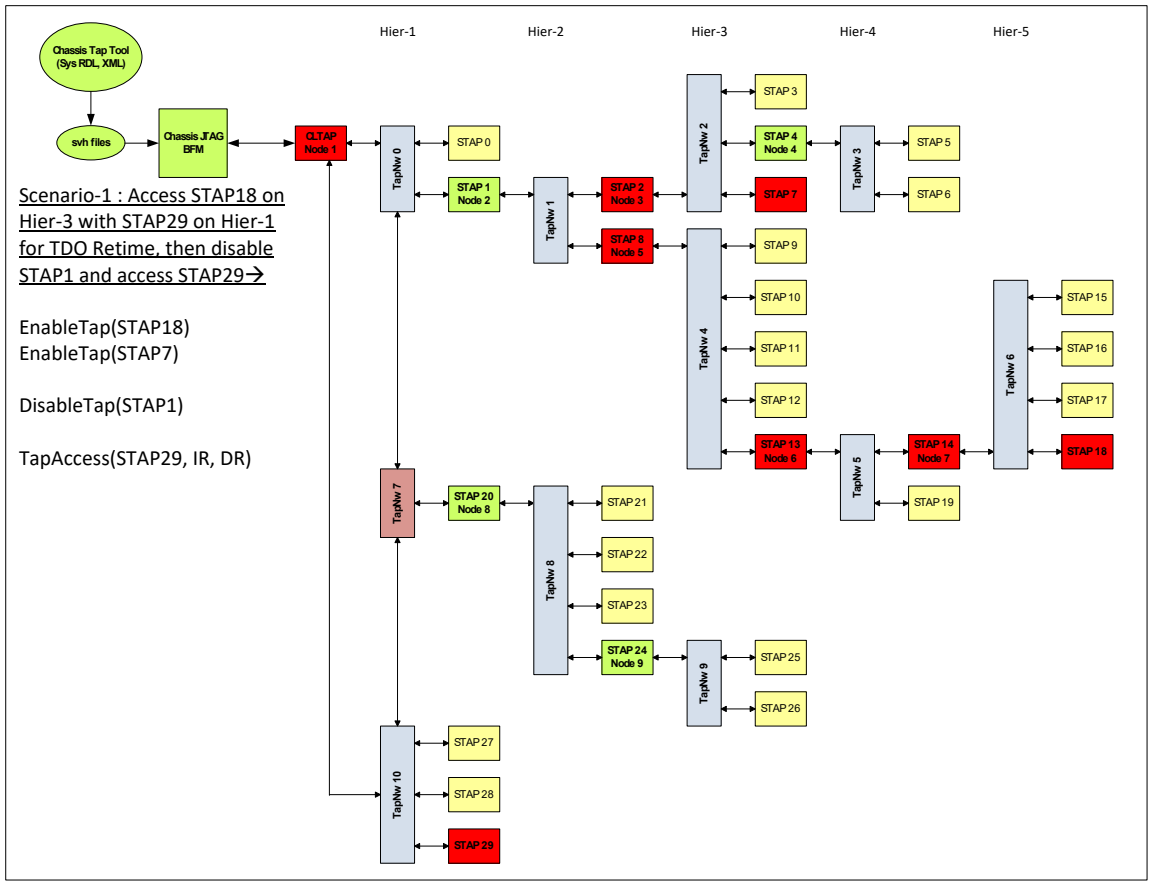


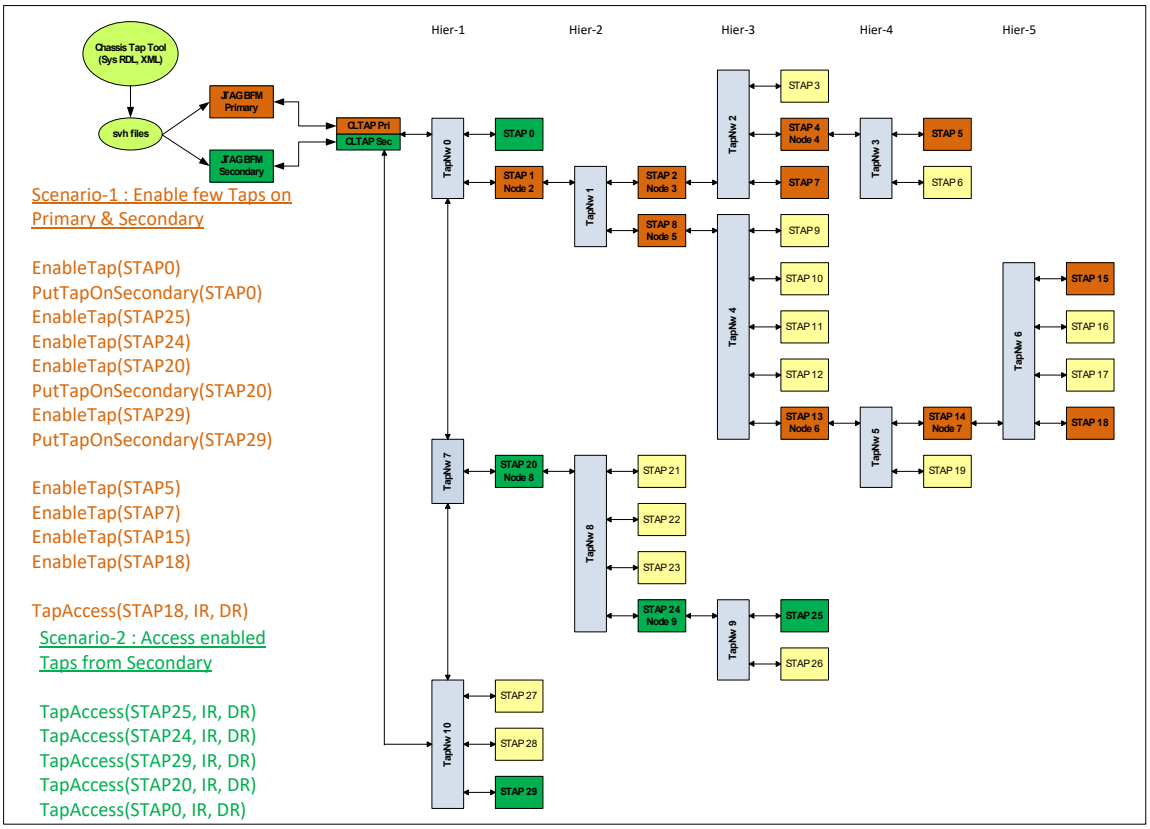
Figure 17 above shows the usage of two targeted TAPs to be active simultaneously. Since these TAPs are on different hierarchies, their parent TAPs should be active. STAP29 at the MSB position of Hier-1 is used as TDO Retime TAP. When an opcode of STAP7 is accessed then all other TAPs are in Bypass mode.

Figure 18. Illustration of the Usage of Multiple Taps along with DisableTap



Usage of TapAccess automatically enables a given TAP along with its parents. However when a Tap has to be disabled then each of the parents has to be individually disabled. In the above example, after enabling STAP18 and STAP7, one of the parents STAP1 is disabled. After this all the other TAPs will be in the enabled state although they can't be reached. After this when STAP29 is enabled then the path from CLTAP will be directly to STAP29. Next, when STAP1 is enabled, then all the other TAPs (STAP8, STAP2, STAP7, STAP13, STAP14, STAP18) will be back in the network without enabling them.

Figure 19. Illustration of the Usage of Secondary Port



This example shows two instances of the BFM. One of them is connected to the Primary Port of CLTAPC and the other to secondary port. Using the Primary port, STAP0, STAP20, STAP29, STAP24, STAP25 are enabled. The STAP0, STAP20, STAP29 are moved onto the Secondary port of the CLTAPC. Now both the instances of BFM can access parallelly the TAPS in the respective color.

3.7 Converged TAP BFM

The Converged TAP BFM treats every TAP as part of a TAP network. A TAP network can consist of any number of TAPs (1, 2, 3…). Any TAP without a master is considered a top-level master TAP (such as CLTAP). Any number of disjoint TAP networks (typically constituting a partition of all the model TAPs) are supported. The BFM supports an arbitrary number of TAPs and JTAG ports in any configuration. A TAP can be designated a "linear TAP" (legacy TAP), a hierarchical TAP or a hybrid TAP.

3.7.1 How to Set Up an OVM TAP Testbench

The directory `verif/tb/Converged_JtagBfm` contains the TAP BFM files. The subdirectory `Converged_SampleTests` contains a full example of a standalone OVM environment using the TAP BFM.

To compile the sample test bench, go to the `verif/tb` directory and run the following command.

```
./Converged_JtagBfm/Converged_SampleTests/compile_and_run _A_TAP_BFM_
dfx_tap_sip_api_sequence
```

3.7.1.1 Project TAP Files

Project TAP files are in the directory `verif/tb/Converged_JtagBfm/Converged_SampleTests/tap_files`. The file `dfx_tap_dut_defines.svh` lists the TAP names (enums used to reference the TAPs). The file `dfx_tap_project_defines.svh` contains the maximum IR size for all TAPs in the TAP network. The other files in this directory define TAP classes, one per TAP. These files contain some essential TAP characteristics. TAPs may share the same class (or TAP classes may extend from the same base class) if their characteristics are identical (or similar).

User needs to create the `dfx_tap_dut_defines.svh` file and should give that path of the file to compilation.

3.7.1.2 Environment Files

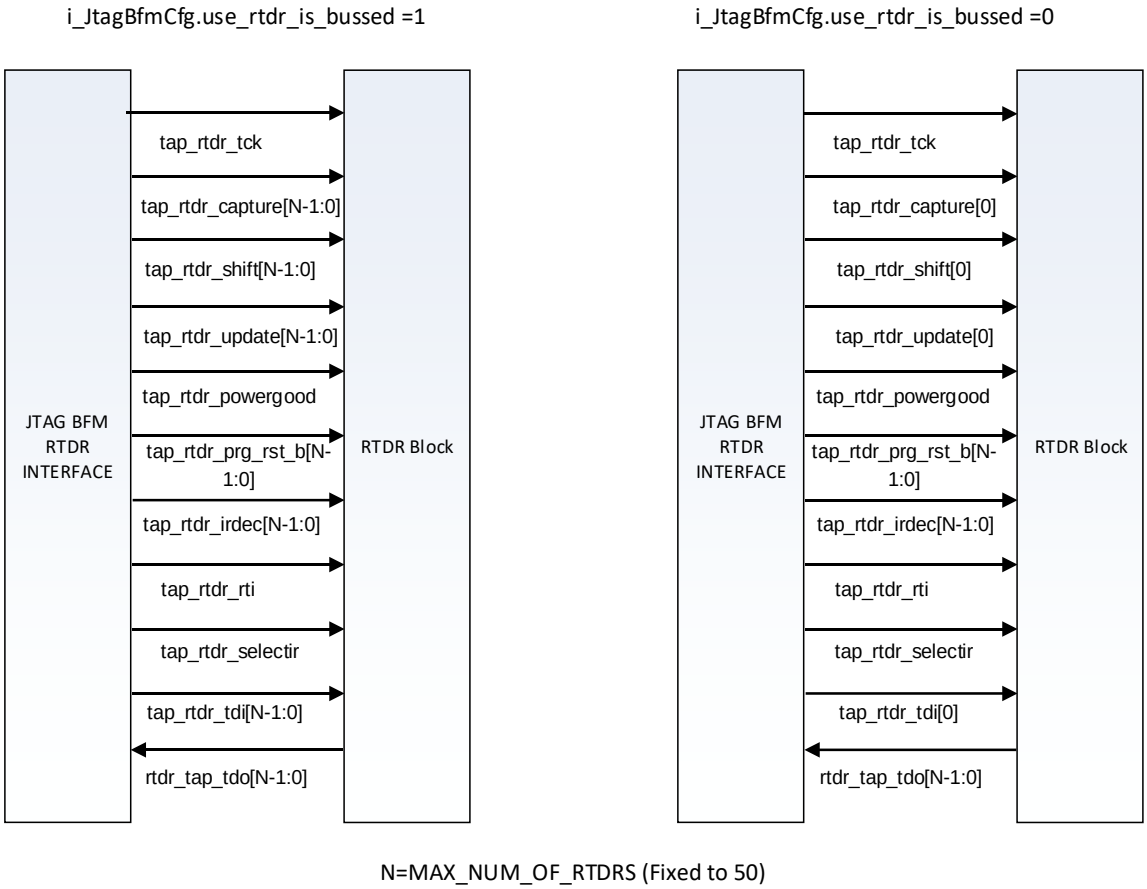
The environment files are as follows:

- `dfx_tb.sv`: the top level module that creates JTAG interfaces, instantiates the sample TAP network model, etc.
- `dfx_test_program.sv`: the program block that includes the OVM test classes (dfx_base_test.sv, dfx_test.sv) and starts a test using run_test().
- `dfx_test.sv`: sets certain configuration parameters and starts a test sequence.
- `tck_generator.sv`: sample TCK generator.
- `dfx_env.sv`: ovm_env class instantiates the top-level TAP environment/agent. The build() function contains a number of set_config_int() calls, one for each TAP in the model that is functional. Non-working TAPs may be skipped or their config table values set to 0 instead of 1.

3.8 How to Use RTDR Interface in JTAG BFM

Connect RTDR Interface of IP to JtagBfmPinIf RTDR Interface, as shown in the following figure.

Figure 20. Port Connection of RTDR with JTAGBFM



Set i_JtagBfmCfg.use_rtdr_interface variable to 1 and i_JtagBfmCfg.rtdr_is_bussed variable as 0/1 as per the requirement.

```
//-------------------------------------------------------------------------------
------------
class RTDRTapTest extends TapBaseTest;

   `ovm_component_utils_begin(RTDRTapTest)
   `ovm_field_object (i_JtagBfmCfg, OVM_ALL_ON|OVM_NOPRINT)
   `ovm_component_utils_end

   TapSeqRtdr SP;
   // Components of the environment
   JtagBfmCfg            i_JtagBfmCfg;

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

   virtual function void build();
      super.build();
```

```
      i_JtagBfmCfg = JtagBfmCfg::type_id::create("i_JtagBfmCfg",this);

      // To register the configuration descriptor
      set_config_object("*", "JtagBfmCfg", i_JtagBfmCfg, 0);
      i_JtagBfmCfg.rtdr_is_bussed =1'b1;
      i_JtagBfmCfg.use_rtdr_interface = 1'b1;
      set_config_int("Env.PriMasterAgent**","rtdr_is_bussed",
i_JtagBfmCfg.rtdr_is_bussed);
      set_config_int("Env.PriMasterAgent**","use_rtdr_interface",
i_JtagBfmCfg.use_rtdr_interface);
      set_config_int("Env.SecMasterAgent**","rtdr_is_bussed",
i_JtagBfmCfg.rtdr_is_bussed);
      set_config_int("Env.SecMasterAgent**","use_rtdr_interface",
i_JtagBfmCfg.use_rtdr_interface);
      set_config_int("Env","has_scoreboard",  0);
   endfunction : build

   virtual task run();
      ovm_report_info("RTDRTapTest", "Test Starts!!!",OVM_NONE);
      SP = new("TapSeqRtdr");
      SP.start(Env.PriMasterAgent.Sequencer);
      ovm_report_info("RTDRTapTest", "RTDRTapTest Completed",OVM_NONE);
      global_stop_request();
   endtask : run
endclass : RTDRTapTest
```

Use enum data type to map tap_rtdr_irdec_positions to corresponding register name.

```
typedef enum int {
                rtdr_reg1   = 0,
                rtdr_reg2   = 1,
                rtdr_reg3   = 2
                } Rtdr_reg;
```

Use Reset task to reset the tap_rtdr_irdec and to assert the tap_rtdr_powergood

```
Reset(RST_PWRGUD); //Will assert the powerGood
Reset(RST_HARD);   //Will resets tap_rtdr_irdec
Reset(RST_SOFT);   //Will resets tap_rtdr_irdec
```

Use Load IR and Load DR APIs to access the register.

```
LoadIR(RST_PWRGUD,    // Reset Mode to assert the tap_rtdr_prog_rst_b
      reg_name1,      // RTDR Register name to be access
      ,
      ,
      );


 LoadDR(NO_RST,         // Reset Mode
      32'h0123_4567,  // Data into the Register
      32'h048D_159C,  // Expected Data out of the Register
      32'hFFFF_FFFF,  // Mask, here all bits are compared
      32);            // Length of Data transaction
```

3.9 File Lists for IRR JTAGBFM and Converged JTAGBFM

Make sure that JTAG_BFM_VER and OVM_HOME variables are set before compiling the file below.

- File list for IRR JTAGBFM: `$IP_ROOT/scripts/filelist.f`
- File list for Converged JTAGBFM: `$IP_ROOT/scripts/converged_filelist.f`

4 Implementing Test Scenarios

The BaseTest is where all the environment components are instantiated. The JTAGBFM configuration object too has to be instantiated here. The values for it can be set here.

Refer to Table 4 for a list of all the configuration variables that can be set from the base test.

Figure 21. Code 9. Snapshot of BFM Instantiation in Env

```
// Refer to file ip-jtag-bfm/subIP/ip-tap-network/verif/tests/TapBaseTest.sv for guidance

class SocBaseTest extends ovm_test;

    // Components of the environment
    JtagBfmCfg    i_JtagBfmCfg;
    SoCEnv        i_SoCEnv;

    // new Constructor & other variables not shown here

    // OVM build fn
    virtual function void build();
        super.build();
        …
        i_JtagBfmCfg = JtagBfmCfg::type_id::create("jtagBfmCfg", this);
        i_SoCEnv     = SoCEnv::type_id::create("Env", this);
        …

        // To register the configuration descriptor
        set_config_object("Env.*", "JtagBfmCfg", jtagBfmCfg, 0);

        // Set each field of the descriptor
        jtagBfmCfg.quit_count        = 2;
        jtagBfmCfg.set_verbosity     = OVM_NONE;
        jtagBfmCfg.enable_clk_gating = 1;
        jtagBfmCfg.park_clk_at       = 0;
        jtagBfmCfg.RESET_DELAY       = 10000;
        jtagBfmCfg.tracker_name                = "STAP";
        jtagBfmCfg.jtag_bfm_tracker_en         = 1;
        jtagBfmCfg.jtag_bfm_runtime_tracker_en = 1;
        jtagBfmCfg.primary_tracker             = 1;
        jtagBfmCfg.secondary_tracker           = 1;
        jtagBfmCfg.sample_tdo_on_negedge       = 1;

        set_config_int("Env.*", "quit_count",        jtagBfmCfg.quit_count);
        set_config_int("Env.*", "set_verbosity",     jtagBfmCfg.set_verbosity);
        set_config_int("Env.*", "enable_clk_gating", jtagBfmCfg.enable_clk_gating);
        set_config_int("Env.*", "park_clk_at",       jtagBfmCfg.park_clk_at);
        set_config_int("Env.*", "RESET_DELAY",       jtagBfmCfg.RESET_DELAY);
set_config_int("Env.*","sample_tdo_on_negedge", jtagBfmCfg.sample_tdo_on_negedge);

            set_config_string("Env.*", "tracker_name", jtagBfmCfg.tracker_name);
        set_config_int("Env.*", "primary_tracker",   jtagBfmCfg.primary_tracker);
        set_config_int("Env.*", "secondary_tracker", jtagBfmCfg.secondary_tracker);
        set_config_int("Env.*", " jtag_bfm_tracker_en", jtagBfmCfg.jtag_bfm_tracker_en);
        set_config_int("Env.*", " jtag_bfm_runtime_tracker_en",
jtagBfmCfg.jtag_bfm_runtime_tracker_en);
    endfunction : build

    // Connect the various env components here
    virtual function void connect();
        …
       super.connect ();
        …
    endfunction : connect

endclass : SocEnv
```

4.1 Configuring the Agents

Parameters for a particular port should be set using the name of the configuration parameter suffixed with "_TAP_PORT_P0" or "_TAP_PORT_P1" or "_TAP_PORT_P2" or so on. These are the enums for the port names defined in the file
`verif/tb/Converged_JtagBfm/testbench/dfx_tap_types.svh`

Table 5. Configuring the Agents table

Configuration Parameter Name (string)	Where Set	Remarks/Examples (All file paths are relative to the directory `verif/tb/Converged_JtagBfm/Converged_SampleTests`)

<TAP enum>	ovm_env class, build() phase	set_config_int("*tap_agent*", "CLTAP", 1);
		set_config_int("*tap_agent*", "STAP0", 1);
		set_config_int("*tap_agent*", "STAP1", 1);
		set_config_int("*tap_agent*", "STAP2", 1);
		set_config_int("*tap_agent*", "STAP3", 1);
		set_config_int("*tap_agent*", "STAP4", 1);
		set_config_int("*tap_agent*", "STAP5", 1);
		set_config_int("*tap_agent*", "STAP6", 1);
		set_config_int("*tap_agent*", "STAP7", 1);
		set_config_int("*tap_agent*", "STAP8", 1);
		set_config_int("*tap_agent*", "STAP9", 1);
		set_config_int("*tap_agent*", "STAP10", 1);
		set_config_int("*tap_agent*", "STAP11", 1);
		set_config_int("*tap_agent*", "STAP12", 1);
		set_config_int("*tap_agent*", "STAP13", 1);
		set_config_int("*tap_agent*", "STAP14", 1);
		set_config_int("*tap_agent*", "STAP15", 1);
		set_config_int("*tap_agent*", "STAP16", 1);
		set_config_int("*tap_agent*", "STAP17", 1);
		set_config_int("*tap_agent*", "STAP18", 1);
		set_config_int("*tap_agent*", "STAP19", 1);
		set_config_int("*tap_agent*", "STAP20", 1);
		set_config_int("*tap_agent*", "STAP21", 1);
		set_config_int("*tap_agent*", "STAP22", 1);
		set_config_int("*tap_agent*", "STAP23", 1);
		set_config_int("*tap_agent*", "STAP24", 1);
		set_config_int("*tap_agent*", "STAP25", 1);
		set_config_int("*tap_agent*", "STAP26", 1);
		set_config_int("*tap_agent*", "STAP27", 1);
		set_config_int("*tap_agent*", "STAP28", 1);
		set_config_int("*tap_agent*", "STAP29", 1);
		If unspecified, the default value is 0.
		If the value is 0, the BFM assumes that the TAP is not present in the model. This means that TAP operations are not permitted for such TAPs.
		If the value is 1, the BFM assumes that the TAP is present in the model.
		See the file `tap_files/dfx_tap_dut_define.svh` for the TAP enums.

Configuration Parameter Name (string)	Where Set	Remarks/Examples (All file paths are relative to the directory `verif/tb/Converged_JtagBfm/Converged_SampleTests`)
count	ovm_test class, build() phase	set_config_int("*.dfx_tap_virt_seqr", "count", 0); Set if desired. The TAP virtual sequencer is a sequencer container and sequences registered with this sequencer must eventually run a sequence on TAP_PORT_P0, TAP_PORT_P1, or so on.
<tap enum>_port_override	ovm_test class, build() phase	set_config_int("*tap_agent*", "STAP14_port_override", TAP_PORT_P1); The BFM assumes that initially every TAP is on TAP_PORT_P0. However, this can be changed in case of a limited TAP network or no TAP network (each TAP's JTAG wires can represent a JTAG port). See the file ../testbench/dfx_tap_types.svh for the TAP port enums.

<tap enum>_state_override	ovm_test class, build() phase	set_config_int("*", "STAP14_state_override", TAP_STATE_NORMAL);
		set_config_int("*", "STAP19_state_override", TAP_STATE_EXCLUDED);
		The BFM assumes that the initial state of any (non-root) TAP (in a disjoint TAP network tree) is TAP_STATE_ISOLATED. The initial state can be changed as shown above.
		See the file ../testbench/dfx_tap_common_defines.svh for the TAP state enums.

dfx_tap_fsm_initial_state_<port enum>	ovm_test class, build() phase	set_config_int("*tap_agent*", "dfx_tap_fsm_initial_state_TAP_PORT_P1", DFX_TAP_TS_TLR);
		set_config_int("*tap_agent*", "dfx_tap_fsm_initial_state_TAP_PORT_P2", DFX_TAP_TS_TLR);
		set_config_int("*tap_agent*", "dfx_tap_fsm_initial_state_TAP_PORT_P3", DFX_TAP_TS_TLR);
		The BFM assumes that the initial state of TAP_PORT_P0 is TLR and that the initial state of every other port is RTI. This determines how the TMS pin is initially driven for each port. The initial state can be changed as shown above.
		See the file `../testbench/dfx_tap_common_defines.svh` for the TAP FSM state enums.

dfx_tap_on_demand_tck_<port enum>	ovm_test class, build() phase	set_config_int("*tap_agent*", "dfx_tap_on_demand_tck_TAP_PORT_P0", 1);
		set_config_int("*tap_agent*", "dfx_tap_on_demand_tck_TAP_PORT_P1", 0);
		If unspecified, the default value is 0.
		Normally, TCK is free-running. However, if the value is set to 1 for a port, TCK runs for that port only when a TAP operation is being performed.

Configuration Parameter Name (string)	Where Set	Remarks/Examples (All file paths are relative to the directory `verif/tb/Converged_JtagBfm/Converged_SampleTests`)

dfx_tap_avoid_race_<port enum>	ovm_test class, build() phase	set_config_int("*tap_agent*", "dfx_tap_avoid_race_TAP_PORT_P0", 0);
		set_config_int("*tap_agent*", "dfx_tap_avoid_race_TAP_PORT_P1", 0);
		If unspecified, the default value is 0.
		If set to 1 for a port, the driver starts performing a TAP operation only on a TCK negedge. This is useful in conjunction with an "on demand" TCK.

dfx_tap_initial_delay_<port enum>	ovm_test class, build() phase	set_config_int("*.tap_driver_*", "dfx_tap_initial_delay_TAP_PORT_P0", 2); If unspecified, the default value is 1 time unit (legacy behavior). The value is the number of time units to wait before starting to drive the JTAG pins. This may be used to avoid contention on the JTAG pins.

4.2 Sending Specific Transaction Sequences

The Stimulus Generator implements custom methods to allow the test to form sequences or templates, and perhaps to make simpler or more reusable tests. There are two steps in this process.

- Connecting a Generator to the Agents
- Writing a Generator or Sequencer

The section after this demonstrates how to use a Stimulus Generator to extend the JTAG_BFM_SeqDrvTxn class, customize constraints, randomize, and send it to the Agents.

4.2.1 Connecting a Sequencer to the Agents

Refer to section 3, Getting Started.

4.2.2 Writing a Sequencer

Refer to section 3, Getting Started.

4.3 Extending Transaction Constraints

The BFM Constraint transaction class extends from the ovm_seqeunce_item. In environments that use the BFM, this class gets inherited to create more constraints.

4.4 Reading and Writing Slave Memory

Not applicable to this IP/IPSS

4.5 Using BFL and Simcon

Not applicable to this IP/IPSS

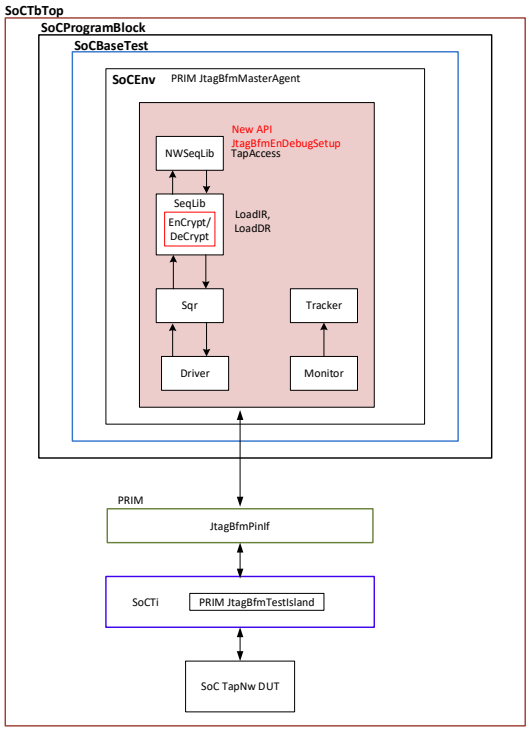
4.6 Test writing using SPF

Not applicable to this IP/IPSS

5 EnDebug Capability

5.1 Strategy for EnDebug Capability in JTAGBFM

Figure 22. Jtagbfm and enDebug Setup



GOAL

- Make Endebug session transparent to users
- Encrypt/Decrypt will use a DPI call to a C program that has `#include <openssl/aes.h>`
- C file will compile with the vcs to generate a simv

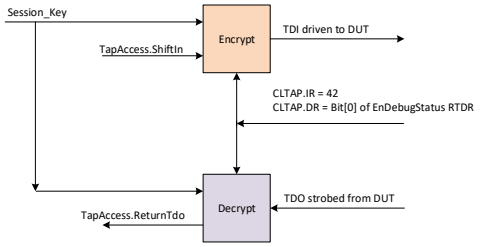
Sequence

- EnDebugSetup API will read the Encrypted key present in the Endebug RTDR 0x43. Decrypt the Encrypted KEY with Fuse Key and will get Session Key.
- Driver will get the tdi vector (ShiftIn) from the TapAccess API i.e Packet.Data and pass it to the EnCrypt function along with the Sesssion key. It will return an encrypted value. This will be the new tdi vector that gets driven in LoadDR.
- Upon the return tdo vector from DUT, the decrypt function will be called to get back the data using the same Session key.

MONITOR/TRACKER

- Monitor will get the Encrypted key present in the Endebug RTDR. Decrypt the Encrypted KEY with Fuse Key and will get Session Key.
- Printing of the Encrypted and Non-Encrypted values will be displayed in the Tracker.

```
1513    task TapAccess(
1514        input Tap_t                                 Tap = 0,
1515        input [API_SIZE_OF_IR_REG-1:0]              Opcode = 0,
1516        input [WIDTH_OF_EACH_REGISTER-1:0]          ShiftIn = 0,
1517        input int                                   ShiftLength = 0,
1518        input [WIDTH_OF_EACH_REGISTER-1:0]          ExpectedData = 0,
1519        input [WIDTH_OF_EACH_REGISTER-1:0]          CompareMask = 0,
1520        input bit                                   ErUserDefinedShiftLength = 0,
1521        input bit                                   ErRegisterPresenceCheck = 1,
1522        output bit [WIDTH_OF_EACH_REGISTER-1:0] ReturnTdo);
```




```
task JtagBfmEnDebugSetup(
                input jtagbfm_encryption_cfg   EndebugEncryptionCfg,
                input int                      CltapIrLength = 16,
        `ifndef JTAG_BFM_AES_256
                input [127:0]                  endbg_fuse_decrypt_key=0);
        `else
                input [255:0]                  endbg_fuse_decrypt_key=0);
        `endif
```

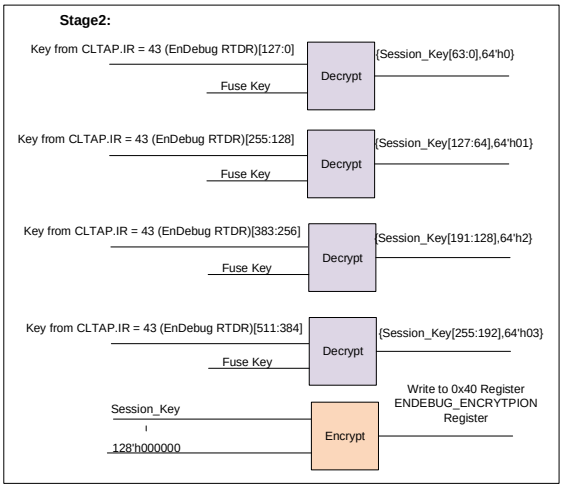
Stage1:

Poll for DRNG Pull Done Status Bit from ENDEBUG_STATUS Register (0x42)

Start the session by programing bit [1:0] in ENDEBUG_CFG (0x41) register

Poll for Key Ready Status Bit from ENDEBUG_STATUS Register (0x42)

Stage2:



Stage3:

Poll for Endebug Ready Status Bit from ENDEBUG_STATUS Register (0x42)

Poll for Endebug Own Status Bit from ENDEBUG_STATUS Register (0x42)

5.2 Compilation of Encryption/Decryption "C" Wrappers

1. Copy the file scripts/aes_c_enc_dec_wrapper.c to scripts folder.
2. Copy the file scripts/gen_aes_c_obj.sh to scripts folder.
3. Need to run the script scripts/gen_aes_c_obj.sh for before Verilog/System Files compilation.
4. Add the option below for elab_opts

```
$ENV{IP_ROOT}/scripts/aes_c_enc_dec_wrapper.o -CFLAGS -lstdc++
/usr/intel/pkgs/openssl/1.0.1j.orig/lib/libcrypto.a
```

5.3 List of enDebug APIs

Table 6. List of enDebug APIs

API Name	Usage
JtagBfmEnDebugRTDRBaseAddressSetup	Used to program the CLTAP RTDR Start Address for EnDebug
JtagBfmEnDebugStatusPoll	Used to poll the EnDebug_STATUS register
JtagBfmEnDebugSetup	Used to setup the EnDebug capability in JTAGBFM
JtagBfmEndebugTapLinkCfg	Used to setup the TAPLINK configuration for EnDebug
JtagBfmEnDebugCfgReg	Used to program the ENDEBUG_CFG register
JtagBfmEnDebug_DebugCfgReg	Used to program the ENDEBUG_DEBUG_CFG Reigster
JtagBfmEnDebugResetPowerGood	Call this API to clear the internal signals in JTAGBFM when endebug is power-down
JtagBfmEnDebugStatusReg	Call this API to get the value of Status Bit in EnDebug Status Register
JtagBfmEnDebug_FSMMODE00_RUTI_TO_TLRS	Call this API to move the FSM from RUTI to TLRS using FSM_MODE=2'b00
JtagBfmEnDebug_DebugCfgRegGrn	Used to program the ENDEBUG_DEBUG_CFG Reigster in green policy
JtagBfmEnDebugEnableLongTAPFSMPath	Update the JtagBfm to get long FSM paths in EnDebug for path from FSM_MODE = 01 -> FSM_MODE=10 -> FSM_MODE=11 (3) When it is 0, path will go from FSM_MODE = 01 and FSM_MODE=10 in same update DR -> FSM_MODE=11 (3)
JtagBfmEnDebug_DebugSessionKeyOvr	Session key override value. This value is used when SessionKeyOvrEn = 1 as the key for this active session. The purpose is to help in debugging endebug.
JtagBfmEndebugTapResetTapAccess	Used to reset the tap during enDebug session

5.4 Description of enDebug APIs

1. To program the CLTAP RTDR Start Address for EnDebug.

```
JtagBfmEnDebugRTDRBaseAddressSetup(
input [API_SIZE_OF_IR_REG-1:0]    EnDebugOpcodeBaseAddress=16'h40
                                                            );
```

   - EnDebugOpcodeBaseAddress: Give the EnDebugOpcodeBaseAddress as per Register Space for EnDebug.

2. To poll the EnDebug status register based on Status bit and Status bit value

```
    JtagBfmEnDebugStatusPoll(
             input int                      CltapIrLength = 16,
             input int                      StatusBit,
             input                          StatusBitValue);
```

   - CltapIrlength : Provide the Instruction Opcode Width.
   - Status Bit : Provide the Status Bit number.

3. Status Bit value: Provide the Bit value want to poll

```
JtagBfmEnDebugSetup(
        input jtagbfm_encryption_cfg   EndebugEncryptionCfg,
        input int                      CltapIrLength = 16,
      `ifndef JTAG_BFM_AES_256
        input [127:0]                  endbg_fuse_decrypt_key=0);
      `else
        input [255:0]                  endbg_fuse_decrypt_key=0);
      `endif
```

   - EndebugEncryptionCfg: Provide the Register value of ENDEBUG_CFG Register.
   - CltapIrlength: Provide the Instruction Opcode Width.
   - endbg_fuse_decrypt_key: Provide the Key used for Fuse encryption.

4. To setup the TAPLINK for EnDebug

```
JtagBfmEndebugTapLinkCfg(
    input [API_SIZE_OF_IR_REG-1:0] CltapEndAddress=16'hFF,
    input                          EndebugEnableTlink = 1'b0 );
```

   - CltapEndAddress: Provide the value of CLTAPC_TDR_END_ADDRESS from CLTAP IP.
   - EndebugEnableTlink: Provide the value of 1 for TAPLINK.

5. To configure the ENDEBUG_CFG Register in between the session.

```
JtagBfmEnDebugCfgReg(
           input jtagbfm_encryption_cfg   EndebugEncryptionCfg,
           input int                      CltapIrLength = 16);
```

   - CltapIrlength : Provide the Instruction Opcode Width.
   - EndebugEncryptionCfg : Provide the Register value of ENDEBUG_CFG Register.

6. To configure the ENDEBUG_DEBUG_CFG Register

```
JtagBfmEnDebug_DebugCfgReg(
  input jtagbfm_encryption_debug_cfg   EndebugEncryptionDebugCfg,
  input int                            CltapIrLength = 16);
```

   - CltapIrlength: Provide the Instruction Opcode Width.
   - EndebugEncryptionDebugCfg: Provide the Register value of ENDEBUG_DEBUG_CFG Register.

7. Call this API when enDebug is powered down.

```
JtagBfmEnDebugResetPowerGood();
```

8. Call this API to move the FSM from RUTI to TLRS in enDebug.

```
task Endebug_FSMMODE00_RUTI_TO_TLRS (input int CLTAPIrWidth = 16);
```

   - CltapIrLength: Provide the Instruction Opcode Width.

9. Call this API to read the value of Status Regsister for particular Bit.

```
task JtagBfmEnDebugStatusReg(
input int CltapIrLength = 16,                                  input int StatusBit,
      output logic   StatusBitValue
      );
```

   - CltapIrLength: Provide the Instruction Opcode Width.
   - StatusBit: Provide the bit number of Status register to read.
   - StatusBitValue: Output of this task to get the Value of particular Status Bit.

10. To Configure the ENDEBUG_DEBUG_CFG_GRN Register.

```
JtagBfmEnDebug_DebugCfgRegGrn(
input jtagbfm_encryption_debug_cfg  EndebugEncryptionDebugCfg, input int
    CltapIrLength = 16);
```

   - CltapIrlength: Provide the Instruction Opcode Width.
   - enDebugEncryptionDebugCfg: Provide the Register value of ENDEBUG_DEBUG_CFG Register.

11. To configure the ENDEBUG_DEBUG_SESSIONKEYOVR Register.

```
task jtagBfmEnDebug_DebugSessionKeyOvr( input
jtagbfm_encryption_debug_sessionkeyovr EndebugEncryptionSessionKeyOvr,input int
    CltapIrLength  = 16);
```

   - CltapIrlength: Provide the Instruction Opcode Width.
   - EndebugEncryptionSessionKeyOvr: Provide the Register value of ENDEBUG_DEBUG_SESSIONKEYOVR Register.

12. To keep FSM in Long path for enBedug.

```
task JtagBfmEnDebugEnableLongTAPFSMPath(bit EnableLongEndebugTAPFSMPath = 1'b0);
```

   - EnableLongEndebugTAPFSMPath: Used to enable the Long FSM path.Update the JtagBfm to get long FSM paths in EnDebug for path from FSM_MODE = 01 -> FSM_MODE=10 -> FSM_MODE=11 (3).When it is 0, path will go from FSM_MODE = 01 and FSM_MODE=10 in same update DR -> FSM_MODE=11 (3).

13. To reset tap during enDebug session.

```
task JtagBfmEndebugTapResetTapAccess (bit EnableTapResetTapAccess = 1'b0);
```

   - EnableTapResetTapAccess: Used to enable the tap reset during enDebug session.

5.5 Use Model of enDebug APIs

Following are the examples for usage of EnDebug APIs.

Give respective value of each argument in all APIs when EnDebug APIs are used.

Figure 23. For TAPNETWORK

```
JtagBfmEnDebugRTDRBaseAddressSetup(
                                .EnDebugOpcodeBaseAddress(8'h50)
                                 );
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h01),
                    .CltapIrLength(8),
                   .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
abcd1234abcd1234abcd1234abcd1234)
                    );
//Can call any TAPNW API here.
TapAccessSlaveIdcode(STAP1);
```

Figure 24. For TAPLINK

```
JtagBfmEnDebugRTDRBaseAddressSetup(
                                .EnDebugOpcodeBaseAddress(8'h50)
                                 );
JtagBfmEndebugTapLinkCfg (.CltapEndAddress(8'hFF),
                          .EndebugEnableTlink(1'b1)
                          );
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h20002),
               .CltapIrLength(12),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));

//Call LoadIR_idle/LoadDR_idle here.
LoadIR_idle( JtagBfmPkg::NO_RST, 12'h120, 12'h000, 12'h000, 12 ); // Load EP
Address to enable IR
LoadDR_idle( JtagBfmPkg::NO_RST, 14'h00C, 14'h000, 14'h000, 14 ); // Load EP DR
```

Figure 25. For TAPNETWORK with BYPASS of ENCRYPTION and DECRYPTION

```
JtagBfmEnDebugRTDRBaseAddressSetup(
                                .EnDebugOpcodeBaseAddress(8'h50)
                                 );
//Bypassing both ENCRYPTION and DECRYPTION
JtagBfmEnDebug_DebugCfgReg(.EndebugEncryptionDebugCfg('h03),
                           .CltapIrLength(8));
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h01),
               .CltapIrLength(8),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));
//Can call any TAPNW API here.
TapAccessSlaveIdcode(STAP1);
```

Figure 26. For TAPLINK with BYPASS of ENCRYPTION and DECRYPTION

```
JtagBfmEnDebugRTDRBaseAddressSetup(
                                .EnDebugOpcodeBaseAddress(8'h50)
                                 );
JtagBfmEndebugTapLinkCfg (.CltapEndAddress(8'hFF),
                          .EndebugEnableTlink(1'b1)
                          );
```

```
//Bypassing both ENCRYPTION and DECRYPTION
JtagBfmEnDebug_DebugCfgReg(.EndebugEncryptionDebugCfg('h03),
                           .CltapIrLength(8));
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h20002),
               .CltapIrLength(12),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));
//Call LoadIR_idle/LoadDR_idle here.
LoadIR_idle( JtagBfmPkg::NO_RST, 12'h120, 12'h000, 12'h000, 12 ); // Load EP
Address to enable IR
LoadDR_idle( JtagBfmPkg::NO_RST, 14'h00C, 14'h000, 14'h000, 14 ); // Load EP DR
```

Figure 27. For TAPLINK with Two Sessions

```
JtagBfmEnDebugRTDRBaseAddressSetup(
                              .EnDebugOpcodeBaseAddress(8'h50)
                               );
JtagBfmEndebugTapLinkCfg (.CltapEndAddress(8'hFF),
                          .EndebugEnableTlink(1'b1)
                          );
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h20002),
               .CltapIrLength(12),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));

//Call LoadIR_idle/LoadDR_idle here.
LoadIR_idle( JtagBfmPkg::NO_RST, 12'h120, 12'h000, 12'h000, 12 ); // Load EP
Address to enable IR
LoadDR_idle( JtagBfmPkg::NO_RST, 14'h00C, 14'h000, 14'h000, 14 ); // Load EP DR
//End the session
JtagBfmEnDebugCfgReg(.EndebugEncryptionCfg('h20004),
                     .CltapIrLength(12));
//Starting the next session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h20002),
               .CltapIrLength(12),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));

//Call LoadIR_idle/LoadDR_idle here.
LoadIR_idle( JtagBfmPkg::NO_RST, 12'h120, 12'h000, 12'h000, 12 ); // Load EP
Address to enable IR
LoadDR_idle( JtagBfmPkg::NO_RST, 14'h00C, 14'h000, 14'h000, 14 ); // Load EP DR
```

Figure 28. For TAPNETWORK with Two Sessions

```
JtagBfmEnDebugRTDRBaseAddressSetup(
                              .EnDebugOpcodeBaseAddress(8'h50)
                               );
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h01),
               .CltapIrLength(8),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234)
//Can call any TAPNW API here.
TapAccessSlaveIdcode(STAP1);
//End the session
JtagBfmEnDebugCfgReg(.EndebugEncryptionCfg('h00004),
                     .CltapIrLength(12));
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h01),
               .CltapIrLength(8),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));
//Can call any TAPNW API here.
```

```
TapAccessSlaveIdcode(STAP1);
```

Figure 29. For TAPNETWORK with MUX_DISABLE Bit

Note: When this bit is configured, direct path from CLTAP to EnDebug can accessed.

```
JtagBfmEnDebugRTDRBaseAddressSetup(
                              .EnDebugOpcodeBaseAddress(8'h50)
                               );
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h01),
               .CltapIrLength(8),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));
//Programming MUXDISABLE bit
JtagBfmEnDebug_DebugCfgReg(.EndebugEncryptionDebugCfg('h20),
                           .CltapIrLength(8));
Reset(2'b01); //Resetting the TOPOLOGY to put all STAPs in TLRS state.
//Can call any TAPNW API here.
TapAccessSlaveIdcode(STAP1);
```

Figure 30. For TAPLINK with MUX_DISABLE Bit

Note: When this bit is configured, direct path from CLTAP to EnDebug can accessed.

```
JtagBfmEnDebugRTDRBaseAddressSetup(
                              .EnDebugOpcodeBaseAddress(8'h50)
                               );
JtagBfmEndebugTapLinkCfg (.CltapEndAddress(8'hFF),
                          .EndebugEnableTlink(1'b1)
                          );
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h20002),
               .CltapIrLength(12),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));

//Programming the MUXDISABLE bit.
JtagBfmEnDebug_DebugCfgReg(.EndebugEncryptionDebugCfg('h20),
                           .CltapIrLength(8));
Reset(2'b01); //Resetting the TOPOLOGY to put all STAPs in TLRS state.
//Call LoadIR_idle/LoadDR_idle here.
LoadIR_idle( JtagBfmPkg::NO_RST, 12'h120, 12'h000, 12'h000, 12 ); // Load EP
Address to enable IR
LoadDR_idle( JtagBfmPkg::NO_RST, 14'h00C, 14'h000, 14'h000, 14 ); // Load EP DR
```

Figure 31. For TAPLINK in ADP SoC

```
 JtagBfmEnDebugRTDRBaseAddressSetup(
                              .EnDebugOpcodeBaseAddress(13'h50)
                               );
JtagBfmEndebugTapLinkCfg (.CltapEndAddress(13'hFF),
                          .EndebugEnableTlink(1'b1)
                          );
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h01),
               .CltapIrLength(13),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));
//Can call any TAPNW API here.
TapAccessSlaveIdcode(STAP1);
```

Figure 32. For TAPLINK in ADP SoC with BYPASS of ENCRYPTION and DECRYPTION

```
JtagBfmEnDebugRTDRBaseAddressSetup(
                                .EnDebugOpcodeBaseAddress(13'h50)
                                 );
                             .CltapIrLength(13));
//Starting the session
JtagBfmEndebugTapLinkCfg (.CltapEndAddress(13'hFF),
                          .EndebugEnableTlink(1'b1)
                          );
//Bypassing both ENCRYPTION and DECRYPTION
JtagBfmEnDebug_DebugCfgReg(.EndebugEncryptionDebugCfg('h03),
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h01),
               .CltapIrLength(13),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));
//Can call any TAPNW API here.
TapAccessSlaveIdcode(STAP1);
```

Figure 33. For TAPLINK in ADP SoC with Two Sessions

```
JtagBfmEnDebugRTDRBaseAddressSetup(
                                .EnDebugOpcodeBaseAddress(13'h50)
                                 );
JtagBfmEndebugTapLinkCfg (.CltapEndAddress(13'hFF),
                          .EndebugEnableTlink(1'b1)
                          );
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h01),
               .CltapIrLength(13),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));
//Can call any TAPNW API here.
TapAccessSlaveIdcode(STAP1);
//End the session
JtagBfmEnDebugCfgReg(.EndebugEncryptionCfg('h00004),
                     .CltapIrLength(12));
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h01),
               .CltapIrLength(13),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));
//Can call any TAPNW API here.
TapAccessSlaveIdcode(STAP1);
```

Figure 34. For TAPLINK in ADP SoC with MUX_DISABLE Bit

When this bit is configured, direct path from CLTAP to EnDebug can accessed.

```
JtagBfmEnDebugRTDRBaseAddressSetup(
                                .EnDebugOpcodeBaseAddress(13'h50)
                                 );
JtagBfmEndebugTapLinkCfg (.CltapEndAddress(13'hFF),
                          .EndebugEnableTlink(1'b1)
                          );
//Starting the session
JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h01),
               .CltapIrLength(13),
               .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
               abcd1234abcd1234abcd1234abcd1234));
//Programming MUXDISABLE bit
JtagBfmEnDebug_DebugCfgReg(.EndebugEncryptionDebugCfg('h20),
                           .CltapIrLength(13));
Reset(2'b01); //Resetting the TOPOLOGY to put all STAPs in TLRS state.
//Can call any TAPNW API here.
TapAccessSlaveIdcode(STAP1);
```

Figure 35. For TAPNETWORK Tap Reset during enDebug session

```
if (ENDBG_POWERGATE_DISABLE == 1)  begin
      JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h20002),
        .CltapIrLength(SIZE_OF_IR_REG_CLTAPC),
        .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
        abcd1234abcd1234abcd1234abcd1234));
end
else begin
      JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h08002),
        .CltapIrLength(SIZE_OF_IR_REG_CLTAPC),
        .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
        abcd1234abcd1234abcd1234abcd1234));

End
JtagBfmEndebugTapResetTapAccess(.EnableTapResetTapAccess(1'b1));
//Can call any TAPNW API here.
TapAccessSlaveIdcode(STAP1);
JtagBfmEndebugTapResetTapAccess(.EnableTapResetTapAccess(1'b0));
```

Figure 36. For TAPLINK Tap Reset during enDebug Session

```
JtagBfmEndebugTapLinkCfg (.CltapEndAddress(13'hFF),
                          .EndebugEnableTlink(1'b1)
                          );
if (ENDBG_POWERGATE_DISABLE == 1)  begin
      JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h20002),
        .CltapIrLength(SIZE_OF_IR_REG_CLTAPC),
        .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
        abcd1234abcd1234abcd1234abcd1234));

end
else begin
      JtagBfmEnDebugSetup(.EndebugEncryptionCfg('h08002),
        .CltapIrLength(SIZE_OF_IR_REG_CLTAPC),
        .endbg_fuse_decrypt_key(256'habcd1234abcd1234abcd1234abcd1234
        abcd1234abcd1234abcd1234abcd1234));
End
JtagBfmEndebugTapResetTapAccess(.EnableTapResetTapAccess(1'b1));
//Can call any TAPNW API here.
TapAccessSlaveIdcode(STAP1);
JtagBfmEndebugTapResetTapAccess(.EnableTapResetTapAccess(1'b0));
```

6 SoC Tap Network Capability

6.1 Chassis TAP Tool (CTT)

Chassis TAP tool is a windows based software application that allows a SoC architect to capture the entire Topology and hierarchy of TAPs. It allows user defined opcodes to be specified on a per TAP basis. An IP provider can use this feature to capture only the TDR info of his SlaveTAP.

Data entered by user to create a topology can be saved to an input XML file. This can be loaded back into the tool for editing the topology. This topology can be saved in different output formats like CSV, XML and TXT. The JTAG BFM uses the output XML file for creating verification collateral that is used for test case writing.

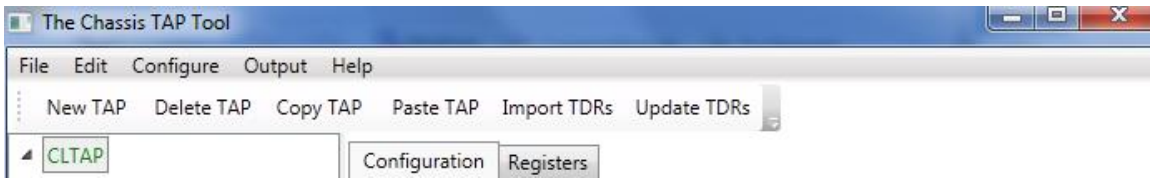
It is available for download from the following link Rev 060- Chassis Tap Tool. The tool is owned by Wiznerowicz, Mike J mike.j.wiznerowicz@intel.com

6.1.1 Creating a Topology

The first step in the usage of CTT is to create an XML file. This file will have the information on the hierarchy of TAPs in SoC.

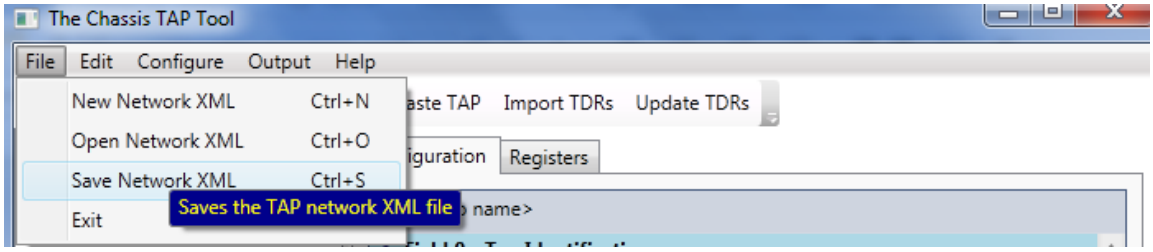
Use the "New TAP" to create the hierarchy. Also enter the desired opcodes information.

Figure 37. CTT Snapshot: Adding a Tap



Save the hierarchy using File -> Save Network XML. This creates an input XML file.

Figure 38. CTT Snapshot: Saving a Topology as Input XML



The XML file with the hierarchy information is normally marinated by the SoC Architect.

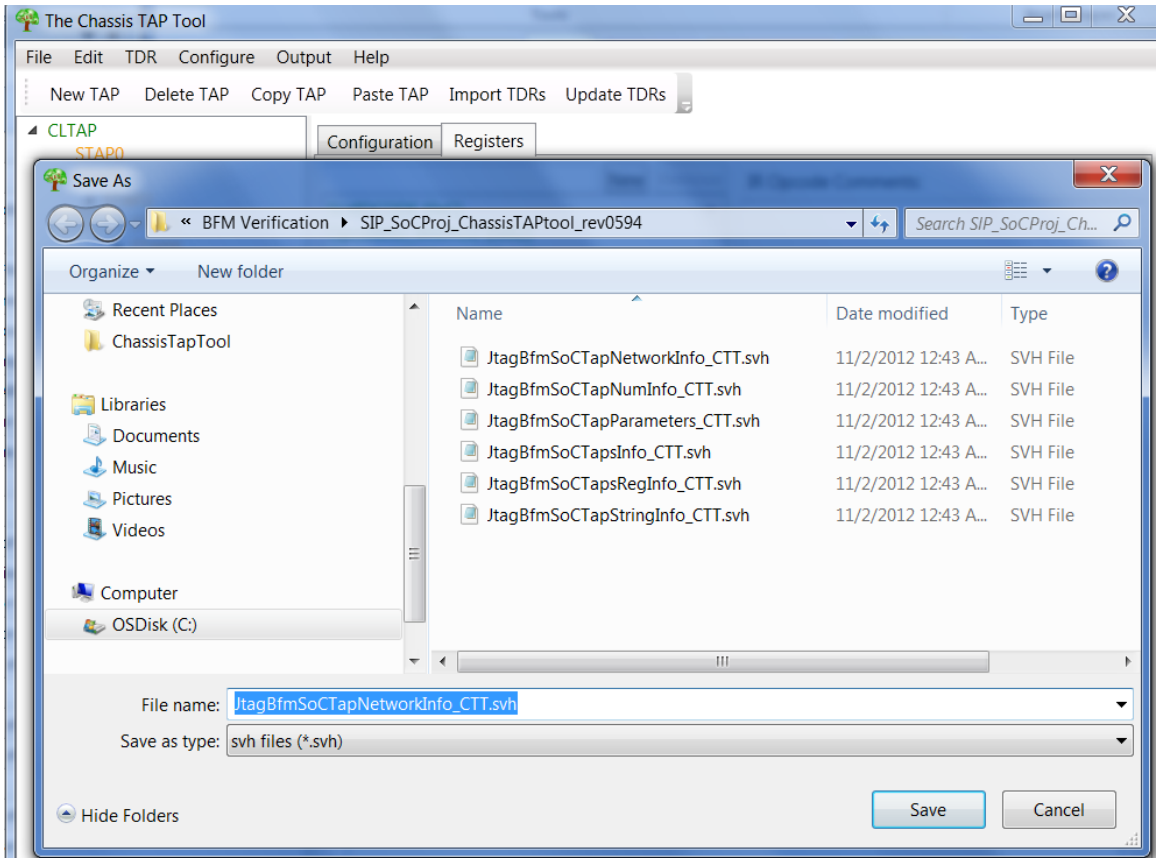
This tool can be used even at the IP level to create the opcodes information and save in XML format that is maintained by the IP provider who uses a TAP.

6.1.2 Generating TAP Aware BFM files with CLTAP (SoC use case model)

Here the BFM talks to the CLTAPC.

Once the XML file is created, there are several output formats that the tool allows to generate. Click on "Output" → "Print to file: TAP BFM verification files" → "Save As"

Figure 39. CTT Snapshot: Generating BFM files for Entire SoC



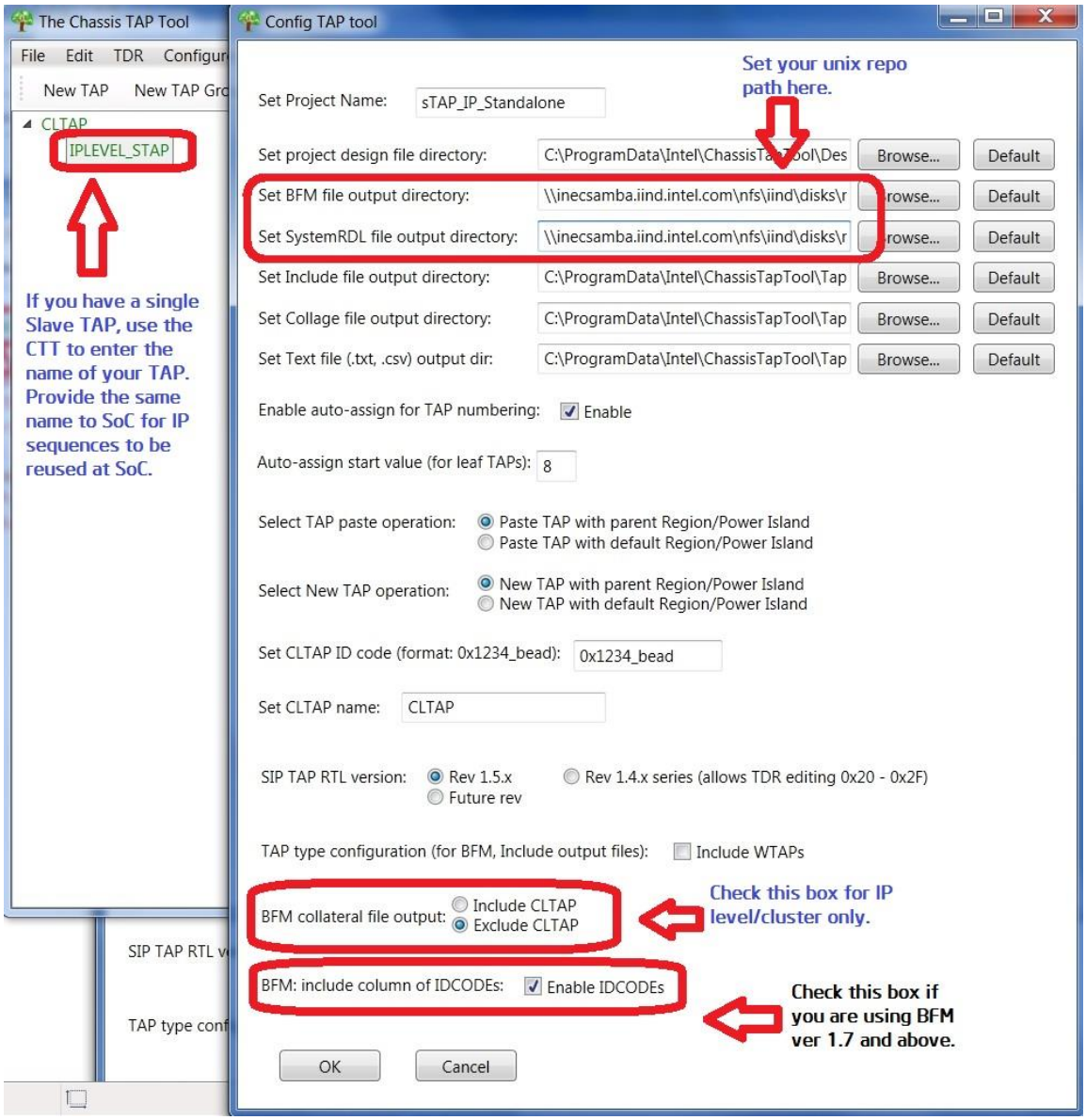
6.1.3 Generating TAP Aware BFM files without CLTAP IP

Here the BFM talks to the SlaveTAP. This could be either at a cluster level (few TAPs) or just at unit level (Single TAP). In either case the CLTAP is not present.

Goto "Configure" → "Config TAP Tool" → "BFM collateral file output" → Check "Exclude CLTAP"

Goto "Output" → "Print to file: TAP BFM verification files" → "Save As"

Figure 40. CTT Snapshot –Generating BFM Files for Standalone TAP



You can store the files in a user defined directory by setting the “Set project directory” path as shown in Figure 40.

6.1.4 Steps to Use the Tap Aware BFM Files (For both SoC, Cluster, and IP level users)

Provide TAPs and Topology information using Chassis TAP Tool. Follow the steps in the section 6.1.1 to section 6.1.3.

1. Make the JTAGBFM part of sip_shared_lib
2. Transfer the following six files to UNIX to a user preferred area outside of the `$JTAG_BFM_VER` directory.
   - `JtagBfmSoCTapsInfo.sv`
   - `JtagBfmSoCTapsRegInfo.sv`
   - `JtagBfmSoCTapParameters.svh`
   - `JtagBfmSoCTapNumInfo.svh`
   - `JtagBfmSoCTapStringInfo.svh`
   - `JtagBfmSoCTapNetworkInfo.svh`

Note: These files had a _CTT in their name until ver 1.6 of the BFM.

These generated files should be part of the compilation of the BFM library. In order to do this, the user has to create an environmental variable that preserves the path where these six generated files are placed.

3. In the soc root project area, create a setenv for the path where the six files are located.

```
cmd> setenv GENERATED_FILES_FOR_JTAGBFM <user_preferred_area>
```

4. The file "jtagbfm_hdl.udf" uses this env variable to search the six files. Add this variable to the project setup process.

Note: Name CTT_GENERATED_FILES is renamed to GENERATED_FILES_FOR_ JTAGBFM in ver 1.7 to be more generic.

5. Now, a tick define macro USE_GENERATED_FILES_FOR_JTAGBFM need to be defined that tells the BFM to pick the six generated files. This could be added at either of two places.
   - In a command prompt of compilation stage as mentioned below
   - In the Local ivars file of ace directory
   - Command:

```
cmd> ace -cc -vlog_opts "+define+USE_GENERATED_FILES_FOR_JTAGBFM"
```

6. If the compilation does not go thru then the integrity of the XML file needs to be inspected as the creation of this XML file is manual sometimes.

Note: Name USE_CTT_GENERATED_FILES is renamed to USE_GENERATED_FILES_ FOR_JTAGBFM in ver 1.7 to be more generic.

Note: If the above steps (1-6) are not followed, then the native BFM commands can still be used and the Network capability cannot be utilized. The compilation will still go through.

6.2 List of SoC TapNetwork BFM APIs

Table 7. Summary Table of all Network APIs in the BFM

API	End State	Description
TapAccess	RUTI	This API first enables the TAP of interest and performs IR, DR scans by padding the appropriate shifts based on the location of a TAP in the hierarchy. When multiple TAPs are enabled, then only the TAP of interest gets the IR that is selected while rest of the enabled TAPs is placed in Bypass. If a TAP is disabled intentionally, then the user will have to intentionally enable it. TapAccess in such cases will not automatically enable the Disabled TAP.
EnableTap	RUTI	Enables a given Tap of Interest in the hierarchy.
DisableTap	RUTI	Disables a given Tap of Interest in the hierarchy. This operation will intentionally disable a TAP that cannot be enabled using TapAccess.
RemoveTap	RUTI	Use only when porting Sequences from IP level to SoC level. This removes a given tap from the network. IR Access: Given TAPs 'h14.
TapAccessSlaveIdcode	RUTI	IR Access: Slave TAPs 'h0C.
TapAccessCltapcIdcode	RUTI	IR Access: CLTAPs 'h02.
PutTapOnSecondary	RUTI	This places a given TAP on the 1st level of hierarchy on the secondary port. (Accesses CLTAP.SEC_SELECT Opcode) IR Access: CLTAPs 'h10.
BuildTapDataBase	NA	Creates all the data structures for the TAP Aware BFM. This should be the first API to be called in the Sequence file.
Added in 1.7 Version		
PutTapOnTertiray	RUTI	Places a TAP on the nearest available Tertiary Port of SoC.
DisableTertiaryPort	NA	This disables a nearest Tertiary port so that the one next Tertiary port up in the hierarchy will be available for Tertiary access.
TapAccessRmw	RUTI	This reads the current contents of a TDR and modifies only the selected bits of the TDR based on mask value. Mask value of 1 intends to change the bit. Mask of zero, does not change the value. FSM states that this API traverses. RUTI → SHIR → UPIR → CADR → SHDR → PADR →SHDR → UPDR
EnableTapForMultiAccess	RUTI	Enables TAP of interest and marks it for MultiAccess.
SetMultiTapCapabilityOnly	NA	If a TAP is already enabled, it marks it for MultiAccess. Other enabled TAPs that are not identified for this will point to the Bypass Opcode during IR operation.
AddIrDrForMultiTapAccess	NA	Build the IR and DR vector for various TAPs of interest.

API	End State	Description
MultiTapAccessLaunchPrimary	RUTI	Initiates an IR/DR operation using the concatenated vector obtained by multiple AddIrDrForMultiTapAccess APIs on CLTAP's Primary port.
MultiTapAccessLaunchSecondary	RUTI	Initiates an IR/DR operation using the concatenated vector obtained by multiple AddIrDrForMultiTapAccess APIs on CLTAP's Secondary port.
ReadIdCodes	RUTI	Reads all the IdCodes and SlaveIdcodes of every TAP.
ShadowTap	RUTI	Places a given Tap of Interest in the hierarchy on shadow mode.
Added in 1.8 Version		
SelectFsmPathToUpdr	NA	Used to configure the FSM during DR chain.
Added in 1.9 version		
dump_history_table	NA	Dumps the contents of history table for a given Tap.
Added in 2.3 Version		
SetCltapcNetworkSelOpcode	NA	'h12: Use CLTAPC_SEL_OVR as the network select register. This is sticky and gets cleared only by fdfx_powergood. 'h11: Use CLTAPC_SEL as the network select register. This is not sticky and gets cleared by fdfx_powergood and trst_b. Other: Assertion error.

Description of the API arguments for SoC TapNw BFM are provided in the subsequent sections.

6.2.1 TapAccess

```
1513    task TapAccess(
1514        input Tap_t                                 Tap = 0,
1515        input [API_SIZE_OF_IR_REG-1:0]              Opcode = 0,
1516        input [WIDTH_OF_EACH_REGISTER-1:0]          ShiftIn = 0,
1517        input int                                   ShiftLength = 0,
1518        input [WIDTH_OF_EACH_REGISTER-1:0]          ExpectedData = 0,
1519        input [WIDTH_OF_EACH_REGISTER-1:0]          CompareMask = 0,
1520        input bit                                   EnUserDefinedShiftLength = 0,
1521        input bit                                   EnRegisterPresenceCheck = 1,
1522       output bit [WIDTH_OF_EACH_REGISTER-1:0]      ReturnTdo);
```

- Tap: Tap defines the Tap ID (enum value) to be accessed.
- Opcode: The Opcode is address or instruction.
- ShiftIn: The data that needs to be loaded in selected register.
- ShiftLength: The length of the data to be shifted into the register. This is valid only when EnRegisterPresenceCheck = 0.
- ExpectedData: The data that is expected to come out.
- CompareMask: The field of ExpectedData that needs to be compared with the field of ShiftIn.
- EnUserDefinedShiftLength: This field enables user to provide a value of Shift length for DR Scan.
- EnRegisterPresenceCheck: The field enables to check for presence of register in the look-up table. When enabled register properties are accessed from the stored locations.

- ReturnTDO: The Actual Collected TDO back from the DUT will be placed into this argument. It is an output of the task.

User has an option not to store register information. For this users need to disable EnRegisterPresenceCheck (set value 0) to access registers which is not part of JTAG BFM. Once EnRegisterPresenceCheck is set to 0, it is the user's responsibility to provide required arguments. Also, when EnRegisterPresenceCheck is set to 0 ShiftLength is a must.

Note: To use the task refer to example in sectuion 6.3, Example Topology supplied with the JTAGBFM.

Table 8. Summary of Usage Scenarios of TapAcess Arguments

Use Cases	Use Case	Argument Usage
SoC does not want to provide the various registers info to the BFM. In this case BFM does not have any info on the length of a specific TDR in a particular TAP. This is the case when none of the RDLs are rolledup to fullchip.	Example: access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide.	EnRegisterPresenceCheck = 0 ShiftLength = 64 EnUserDefinedShiftLength = 1
	Example: access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide. Now you want to access only 32 bits of this register.	EnRegisterPresenceCheck = 0 ShiftLength = 32 EnUserDefinedShiftLength = 1
Say SoC wants to provide all the info of the various registers to the BFM. This is the case when all the IP TAP RDLs are rolled up at fullchip and new set of 6 files for the BFM are generated.	Example: access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide.	EnRegisterPresenceCheck = 1 ShiftLength = 0 EnUserDefinedShiftLength = 0 (BFM already knows it has to shift 64)
	Example: access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide. Now you want to access only 32 bits of this register.	EnRegisterPresenceCheck = 0 ShiftLength = 32 EnUserDefinedShiftLength = 1

Summary: EnRegisterPresenceCheck and ShiftLength go together.

6.2.2 TapAccessRuti

```
task TapAccessRuti(
   input Tap_t                          Tap = Tap_t'(0),
   input [API_SIZE_OF_IR_REG-1:0]       Opcode = 0,
   input [WIDTH_OF_EACH_REGISTER-1:0] ShiftIn = 0,
   input int                            ShiftLength = 0,
   input [WIDTH_OF_EACH_REGISTER-1:0] ExpectedData = 0,
   input [WIDTH_OF_EACH_REGISTER-1:0] CompareMask = 0,
   input bit                            EnUserDefinedShiftLength = 0,
   input bit                            EnRegisterPresenceCheck = 1,
   input [63:0]                         RutiLen = 1,
   output bit [WIDTH_OF_EACH_REGISTER-1:0] ReturnTdo);
```

- Tap: Tap defines the Tap ID (enum value) to be accessed.
- Opcode: The Opcode is address or instruction.
- ShiftIn: The data that needs to be loaded in selected register.

- ShiftLength: The length of the data to be shifted into the register. This is valid only when EnRegisterPresenceCheck = 0.
- ExpectedData: The data that is expected to come out.
- CompareMask: The field of ExpectedData that needs to be compared with the field of ShiftIn.
- EnUserDefinedShiftLength: This field enables user to provide a value of Shift length for DR Scan.
- EnRegisterPresenceCheck: The field enables to check for presence of register in the look-up table. When enabled register properties are accessed from the stored locations.
- RutiLen: This argument used to keep FSM in required no of RUTI cycles based on its value after connecting a new STAP.
- ReturnTDO: The Actual Collected TDO back from the DUT will be placed into this argument. It is an output of the task.

User has an option not to store register information. For this users need to disable EnRegisterPresenceCheck (set value 0) to access registers which is not part of JTAG BFM. Once EnRegisterPresenceCheck is set to 0, it is the user's responsibility to provide required arguments. Also, when EnRegisterPresenceCheck is set to 0 ShiftLength is a must.

Note: To use the task, refer to example section.

Table 9. Summary of Usage Scenarios of TapAcess Arguments

Use Cases	Use Case	Argument Usage
SoC does not want to provide the various registers info to the BFM. In this case BFM does not have any info on the length of a specific TDR in a particular TAP. This is the case when none of the RDLs are rolled up to fullchip.	Example: access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide.	EnRegisterPresenceCheck = 0 ShiftLength = 64 EnUserDefinedShiftLength = 1
	Example: access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide. Now you want to access only 32 bits of this register.	EnRegisterPresenceCheck = 0 ShiftLength = 32 EnUserDefinedShiftLength = 1
Say SoC wants to provide all the info of the various registers to the BFM. This is the case when all the IP TAP RDLs are rolled up at fullchip and new set of 6 files for the BFM are generated.	Example: access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide.	EnRegisterPresenceCheck = 1 ShiftLength = 0 EnUserDefinedShiftLength = 0 (BFM already knows it has to shift 64)
	Example: access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide. Now you want to access only 32 bits of this register.	EnRegisterPresenceCheck = 0 ShiftLength = 32 EnUserDefinedShiftLength = 1

Summary: EnRegisterPresenceCheck & ShiftLength go together.

6.2.3 TapAccessArcEndbg

```
task TapAccessArcEndbg(
   input Tap_t                          Tap = Tap_t'(0),
   input [API_SIZE_OF_IR_REG-1:0]       Opcode = 0,
   input [WIDTH_OF_EACH_REGISTER-1:0] ShiftIn = 0,
   input int                            ShiftLength = 0,
   input [WIDTH_OF_EACH_REGISTER-1:0] ExpectedData = 0,
   input [WIDTH_OF_EACH_REGISTER-1:0] CompareMask = 0,
   input bit                            EnUserDefinedShiftLength = 0,
   input bit                            EnRegisterPresenceCheck = 1,
   output bit [WIDTH_OF_EACH_REGISTER-1:0] ReturnTdo);
```

This API will applies to access ARC TAPs for ENDEBUG.

- Tap: Tap defines the Tap ID (enum value) to be accessed.
- Opcode: The Opcode is address or instruction.
- ShiftIn: The data that needs to be loaded in selected register.
- ShiftLength: The length of the data to be shifted into the register. This is valid only when EnRegisterPresenceCheck = 0.
- ExpectedData: The data that is expected to come out.
- CompareMask: The field of ExpectedData that needs to be compared with the field of ShiftIn.
- EnUserDefinedShiftLength: This field enables user to provide a value of Shift length for DR Scan.
- EnRegisterPresenceCheck: The field enables to check for presence of register in the look-up table. When enabled register properties are accessed from the stored locations.
- ReturnTDO: The Actual Collected TDO back from the DUT will be placed into this argument. It is an output of the task.

User has an option not to store register information. For this users need to disable EnRegisterPresenceCheck (set value 0) to access registers which is not part of JTAG BFM. Once EnRegisterPresenceCheck is set to 0, it is the user's responsibility to provide required arguments. Also, when EnRegisterPresenceCheck is set to 0 ShiftLength is a must.

Note: To use the task, refer to example section.

Table 10. Summary of Usage Scenarios of TapAcess Arguments

Use Cases	Use Case	Argument Usage
SoC does not want to provide the various registers info to the BFM. In this case BFM does not have any info on the length of a specific TDR in a particular TAP. This is the case when none of the RDLs are rolledup to fullchip.	Let say, we want to access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide.	EnRegisterPresenceCheck = 0 ShiftLength = 64 EnUserDefinedShiftLength = 1
	Let say, we want to access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide. Now you want to access only 32 bits of this register.	EnRegisterPresenceCheck = 0 ShiftLength = 32 EnUserDefinedShiftLength = 1

Use Cases	Use Case	Argument Usage
Say SoC wants to provide all the info of the various registers to the BFM. This is the case when all the IP TAP RDLs are rolled up at fullchip and new set of 6 files for the BFM are generated.	Let say, we want to access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide.	EnRegisterPresenceCheck = 1 ShiftLength = 0 EnUserDefinedShiftLength = 0 (BFM already knows it has to shift 64)
	Let say, we want to access Opcode 8'hAB in STAP8 whose DR length is 64 bits wide. Now you want to access only 32 bits of this register.	EnRegisterPresenceCheck = 0 ShiftLength = 32 EnUserDefinedShiftLength = 1

Summary: EnRegisterPresenceCheck & ShiftLength go together.

6.2.4 EnableTap

```
750     task EnableTap (input Tap_t TapOfInterest = Tap_t'(0));
```

- Tap: Tap defines the Tap ID of Slave TAP (enum value) to be enabled on the network.

Note: To use the task, refer to the example in section 6.3, Example Topology supplied with the JTAGBFM.

```
770     task DisableTap (input Tap_t TapOfInterest = Tap_t'(0));
```

- Tap: Tap defines the Tap ID of Slave TAP (enum value) to be disabled on the network.

Note: To use the task, refer to the example in section 6.3, Example Topology supplied with the JTAGBFM.

6.2.5 TapAccessSlaveIdcode

```
327     task TapAccessSlaveIdcode (input Tap_t Tap);
```

- Tap: Tap defines the Tap ID of Slave TAP (enum value) to be accessed. TapAccess_slvidcode read slave idcode, compares with slave TAP idcode value provided in Create_TAP_LUT and generates error for any mismatches.

Note: To use the task, refer to the example in section 6.3, Example Topology supplied with the JTAGBFM.

6.2.6 TapAccessSlaveIdcodeArcEndbg

```
task TapAccessSlaveIdcodeArcEndbg (input Tap_t Tap);
```

Tap: Tap defines the Tap ID of Slave TAP (enum value) to be accessed. TapAccess_slvidcode read slave idcode, compares with slave TAP idcode value provided in Create_TAP_LUT and generates error for any mismatches.

Note: To use the task, refer to the example in section 6.3, Example Topology supplied with the JTAGBFM.

6.2.7 TapAccesssSlaveIdcodeRuti

```
task TapAccessSlaveIdcodeRuti (input Tap_t Tap, input [63:0] RutiLen = 1);
```

- Tap: Tap defines the Tap ID of Slave TAP (enum value) to be accessed.
- RutiLen: This argument used to keep FSM in required no of RUTI cycles based on its value after connecting a new STAP.

TapAccess_slvidcode read slave idcode, compares with slave TAP idcode value provided in Create_TAP_LUT and generates error for any mismatches.

6.2.8 TapAccessCltapcIdcode

```
460     task TapAccessCltapcIdcode (input Tap_t Tap);
```

- Tap: Tap defines the Tap ID of Master TAP (enum value) to be accessed. TapAccess_ cltapidcode read cltapc idcode, compares with master TAP idcode value provided in Create_TAP_LUT and generates error for any mismatches.

Note: To use the task, refer to the example in section 6.3, Example Topology supplied with the JTAGBFM.

6.2.9 TapAccessSlaveIdcodeGetStatus

```
    task TapAccessSlaveIdcodeGetStatus (input Tap_t Tap, output bit Status);
```

- Tap: Tap defines the Tap ID of Slave TAP (enum value) to be accessed. TapAccess_slvidcode read slave idcode, compares with slave TAP idcode value provided in Create_TAP_LUT and generates error for any mismatches.
- Status : Denotes PASS/FAIL status of Data comparison. 1- PASS, 0-FAIL

6.2.10 TapAccessSlaveIdcodeGetStatusRuti

```
task TapAccessSlaveIdcodeGetStatusRuti (input Tap_t Tap, input [63:0] RutiLen = 1, output bit Status);
```

- Tap: Tap defines the Tap ID of Slave TAP (enum value) to be accessed. TapAccess_slvidcode read slave idcode, compares with slave TAP idcode value provided in Create_TAP_LUT and generates error for any mismatches.
- RutiLen: This argument used to keep FSM in required no of RUTI cycles based on its value after connecting a new STAP.
- Status: Denotes PASS/FAIL status of Data comparison. 1- PASS, 0-FAIL

6.2.11 TapAccessCltapcIdcodeGetStatus

```
    task TapAccessCltapcIdcodeGetStatus (input Tap_t Tap, output bit Status);
```

- Tap: Tap defines the Tap ID of Master TAP (enum value) to be accessed. TapAccess_ cltapidcode read cltapc idcode, compares with master TAP idcode value provided in Create_TAP_LUT and generates error for any mismatches.
- Status: Denotes PASS/FAIL status of Data comparison. 1- PASS, 0-FAIL

6.2.12 PutTapOnSecondary

```
1801      task PutTapOnSecondary (input Tap_t Tap);
```

- Tap: Tap defines the Tap ID of Slave TAP (enum value) to be accessed using secondary.

Note: To use the task, refer to the example in section 6.3, Example Topology supplied with the JTAGBFM.

6.2.13 RemoveTap

```
2104      task RemoveTap (input Tap_t Tap);
```

- Tap: Tap defines the Tap ID of Slave TAP (enum value) to be removed from the network using Tap Remove opcode.

Note: To use the task, refer to the example in section 6.3, Example Topology supplied with the JTAGBFM.

6.2.14 BuildTapDataBase

```
104      task BuildTapDataBase();
```

The task creates Taps info database. This API is must before we do any access. Usually this is called only once per test.

6.2.15 PutTapOnTertiary

```
task PutTapOnTertiary (input Tap_t Tap);
```

This task places a TAP on the nearest available Tertiary Port of SoC. It will not place a TAP on a hierarchy one above it.

6.2.16 DisableTertiaryPort

```
task DisableTertiaryPort (input Tap_t Tap);
```

This disables a nearest Tertiary port so that the one next Tertiary port up in the hierarchy will be available for Tertiary access. This physically does not block access to a Tertiary port.

6.2.17 TapAccessRmw

```
3011
3012     //TapAccess Read-Modify-Write
3013     task TapAccessRmw(
3014        input Tap_t                          Tap = 0,
3015        input [API_SIZE_OF_IR_REG-1:0]       Opcode = 0,
3016        input [WIDTH_OF_EACH_REGISTER-1:0] ShiftIn = 0,
3017        input int                            ShiftLength = 0,
3018        input [WIDTH_OF_EACH_REGISTER-1:0] ExpectedData = 0,
3019        input [WIDTH_OF_EACH_REGISTER-1:0] WriteMask = 0,
3020        input bit                            EnUserDefinedShiftLength = 0,
3021        input bit                            EnRegisterPresenceCheck = 1,
3022        output bit [WIDTH_OF_EACH_REGISTER-1:0] ReturnTdo);
3023
```

This reads the current contents of a TDR and modifies only the selected bits of the TDR based on mask value. Mask value of 1 intends to change the bit. Mask of zero, does not change the value. FSM states that this API traverses.

RUTI → SHIR → UPIR → CADR → SHDR → PADR →SHDR → UPDR

6.2.18 EnableTapForMultiAccess

```
task EnableTapForMultiAccess (input Tap_t TapOfInterest = Tap_t'(0));
```

Enables TAP of interest and marks it for MultiAccess.

6.2.19 SetMultiTapCapabilityOnly

```
task SetMultiTapCapabilityOnly(input Tap_t TapOfInterest = Tap_t'(0));
```

If a TAP is already enabled, it marks it for MultiAccess. Other enabled TAPs that are not identified for this will point to the Bypass Opcode during IR operation.

6.2.20 AddIrDrForMultiTapAccess

```
1662     task AddIrDrForMultiTapAccess(
1663        input Tap_t                                 Tap = 0,
1664        input [API_SIZE_OF_IR_REG-1:0]              Opcode = 0,
1665        input [WIDTH_OF_EACH_REGISTER-1:0]          ShiftIn = 0,
1666        input int                                   ShiftLength = 0,
1667        input [WIDTH_OF_EACH_REGISTER-1:0]          ExpectedData = 0,
1668        input [WIDTH_OF_EACH_REGISTER-1:0]          CompareMask = 0,
1669        input bit                                   EnUserDefinedShiftLength = 0,
1670        input bit                                   EnRegisterPresenceCheck = 1,
1671       output bit [WIDTH_OF_EACH_REGISTER-1:0]      ReturnTdo);
1672
```

Build the IR and DR vector for various TAPs of interest.

6.2.21 MultiTapAccessLaunchPrimary

```
task MultiTapAccessLaunchPrimary();
```

Initiates an IR/DR operation using the concatenated vector obtained by multiple AddIrDrForMultiTapAccess APIs on CLTAP's Primary port.

6.2.22 MultiTapAccessLaunchSecondary

```
task MultiTapAccessLaunchSecondary();
```

Initiates an IR/DR operation using the concatenated vector obtained by multiple AddIrDrForMultiTapAccess APIs on CLTAP's Secondary port.

6.2.23 ReadIdCodes

```
task ReadIdCodes ();
```

Reads all the IdCodes and SlaveIdcodes of every TAP that the BFM is aware of.

6.2.24 ShadowTap

```
task ShadowTap (input Tap_t TapOfInterest = Tap_t'(0));
```

- TapOfInterest: Defines the Tap ID of Slave TAP (enum value) to be placed on Shadow mode on the TAPNetwork.

Note: To use the task, refer to example section.

6.2.25 SelectFsmPathToUpdr

```
task SelectFsmPathToUpdr(input int ShortLongPath = 0);
```

- ShortLongPath: For value of '0', TAP FSM traverses thru shortest path on DR Chain i.e., CADR->SHDR->E1DR->UPDR. And for value of '1', FSM path is long i.e., CADR->SHDR->E1DR->PADR -> E2DR -> UPDR.

6.2.26 dump_history_table

```
function dump_history_table(input Tap_t TapOfInterest);
```

Dumps the history table for a given TAP. This includes info such as:

- TapOfInterest
- ParentTap
- HierarchyLevel
- IsVendorTap
- NodeQueue.size
- Tap_Node_q
- NodeArray
- TapBeforeTapOfInterestArr
- TapAfterTapOfInterestArra
- IsTertiaryPortEnabled
- TertiaryParentTap
- ChildTapsArray

6.3 Example Topology supplied with the JTAGBFM

Default JTAGBFM IRR download contains example network configurations for 5 levels of hierarchy as shown in Figure 41. The remaining sections of this Integration Guide will illustrate the various features of TAP Aware BFM using this topology.

Figure 41. CTT Snapshot: Sample Topology Supplied with JTAGBFM

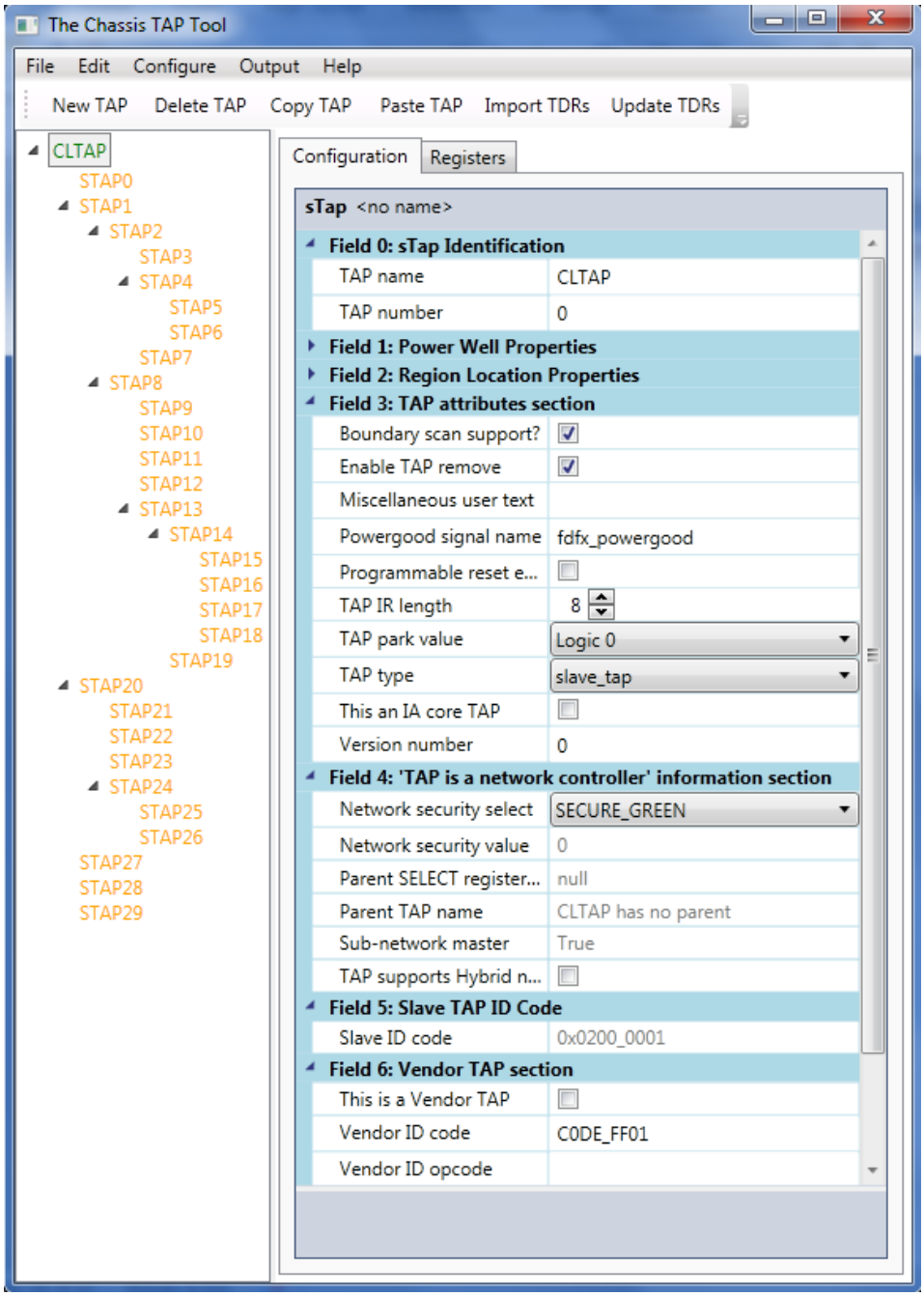
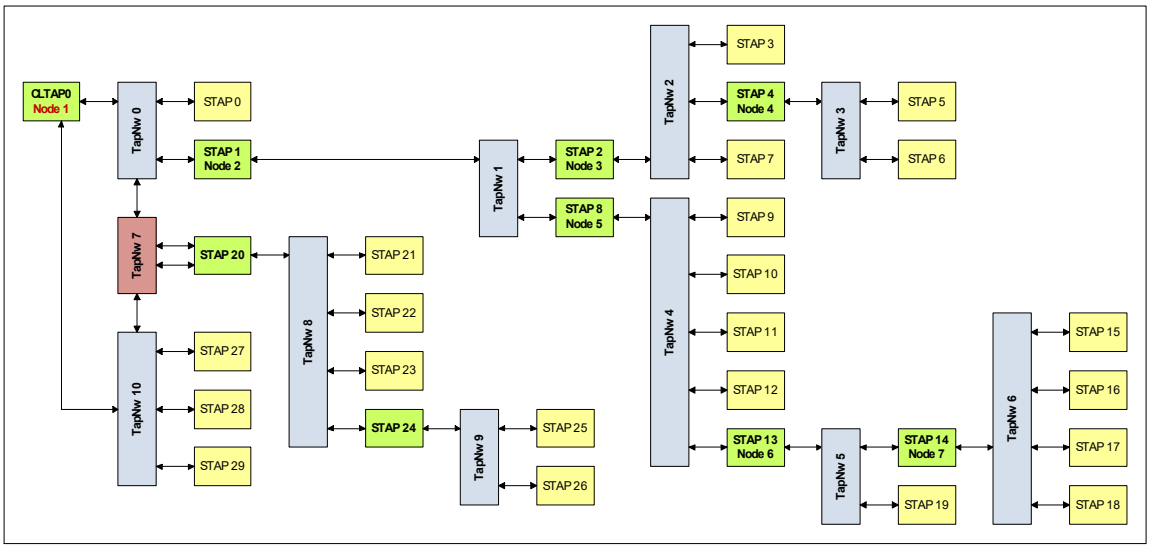


Figure 42. Ace Integration in SoC or other IP that uses this as a SubIP



6.3.1 Add Network Topology Information

Create_TAP_LUT function creates internal queue to store network information. Code example is shown in Code 8. This information is captured in file JtagBfmSoCTapsInfo.svh

Figure 43. Code 10. Contents of Network Topology File

```
//          TAP          SlvIDcode       IDcode      IR_Width  Node  Sec_connections  Hybrid_en  Dfx_Security  Hierarchy_Level  PositionOfTap  IsVendorTap   VendorIdOpcode
Create_TAP_LUT (CLTAP,   32'h0000_0001, 32'hCODE_FF01, 'd8,     'd1,      'd1,           'd0,       GREEN,        0,               0,             1,            'h02);
Create_TAP_LUT (STAP0,   32'h0400_0001, 32'hCODE_0001, 'd8,     'd0,      'd0,           'd0,       ORANGE,       1,               0,             1,            'h0C);
Create_TAP_LUT (STAP1,   32'h0600_0003, 32'hCODE_0101, 'd8,     'd2,      'd1,           'd0,       ORANGE,       1,               1,             1,            'h0C);
Create_TAP_LUT (STAP2,   32'h0600_0105, 32'hCODE_0201, 'd8,     'd3,      'd0,           'd0,       ORANGE,       2,               0,             1,            'h0C);
Create_TAP_LUT (STAP3,   32'h0400_0207, 32'hCODE_0301, 'd8,     'd0,      'd0,           'd0,       ORANGE,       3,               0,             1,            'h0C);
Create_TAP_LUT (STAP4,   32'h0600_0209, 32'hCODE_0401, 'd8,     'd4,      'd0,           'd0,       ORANGE,       3,               1,             1,            'h0C);
Create_TAP_LUT (STAP5,   32'h0400_040B, 32'hCODE_0501, 'd8,     'd0,      'd0,           'd0,       ORANGE,       4,               0,             1,            'h0C);
Create_TAP_LUT (STAP6,   32'h0400_040D, 32'hCODE_0601, 'd8,     'd0,      'd0,           'd0,       ORANGE,       4,               1,             1,            'h0C);
```

Figure 44. Code 11. Create_TAP_LUT Arguments

```
virtual function Create_TAP_LUT(
    input Tap_t         Tap,
    input int           Tap_SlvIDcode,
    input int           Tap_VendorIDcode,
    input int           IR_Width,
    input int           Node,
    input int           SecondaryConnectionEnable,
    input int           SecondaryHybridEnable,
    input DFX_Security  Dfx_Ssecurity,
    input int           HierarchyLevel,
    input int           PostionOfTapInTapcSelReg,
    input int           IsIdcode_eq0C
    );
```

- Tap: Unique alias for identifying the TAP in network.
- Tap_SlvIDcode: Strap value at the input of each Tap RTL Instance. Value pointed by IR opcode 0x0C.
- Tap_VendorIDcode: Value pointed by IR opcode 0x02.
- IR_Width: Width of Instruction in each TAP.
- Node: A non-zero value here represents a TAP having a sub network. It has to be a unique non repeating number.

- SecondaryConnectionEnable: Value of Zero represents secondary connections are not enabled for Tap. Value of One represents secondary connections are enabled.
- SecondaryHybridEnable: Value of Zero represents secondary hybrid connections are not enabled for Tap. Value of One represents secondary hybrid connections are enabled.
- DFx Security: Security level of each TAP on the network.
- HierarchyLevelDFx Security: Security level of each TAP on the network.

6.3.1.1 Add Network Information

One of the fields of Tap_Info_t (used in SoCTapNetwork element) struct stores the info of the network. Code example is shown in Code 9. This information is captured in file JtagBfmSoCTapNetworkInfo.svh.

Figure 45. Code 12. Contents of Network Information file should look like this

```
case (Node)
    'd1 : begin
              Tap_Info_Int.Next_Tap[0] = STAP0;
              Tap_Info_Int.Next_Tap[1] = STAP1;
              Tap_Info_Int.Next_Tap[2] = STAP20;
              Tap_Info_Int.Next_Tap[3] = STAP27;
              Tap_Info_Int.Next_Tap[4] = STAP28;
              Tap_Info_Int.Next_Tap[5] = STAP29;
          end
    'd2 : begin
              Tap_Info_Int.Next_Tap[0] = STAP2;
              Tap_Info_Int.Next_Tap[1] = STAP8;
          end
    'd3 : begin
              Tap_Info_Int.Next_Tap[0] = STAP3;
              Tap_Info_Int.Next_Tap[1] = STAP4;
              Tap_Info_Int.Next_Tap[2] = STAP7;
          end
    'd4 : begin
              Tap_Info_Int.Next_Tap[0] = STAP5;
              Tap_Info_Int.Next_Tap[1] = STAP6;
          end
    'd5 : begin
              Tap_Info_Int.Next_Tap[0] = STAP9;
              Tap_Info_Int.Next_Tap[1] = STAP10;
              Tap_Info_Int.Next_Tap[2] = STAP11;
              Tap_Info_Int.Next_Tap[3] = STAP12;
              Tap_Info_Int.Next_Tap[4] = STAP13;
          end
    'd6 : begin
              Tap_Info_Int.Next_Tap[0] = STAP14;
              Tap_Info_Int.Next_Tap[1] = STAP19;
          end
    'd7 : begin
              Tap_Info_Int.Next_Tap[0] = STAP15;
              Tap_Info_Int.Next_Tap[1] = STAP16;
              Tap_Info_Int.Next_Tap[2] = STAP17;
              Tap_Info_Int.Next_Tap[3] = STAP18;
          end
    'd8 : begin
              Tap_Info_Int.Next_Tap[0] = STAP21;
              Tap_Info_Int.Next_Tap[1] = STAP22;
              Tap_Info_Int.Next_Tap[2] = STAP23;
              Tap_Info_Int.Next_Tap[3] = STAP24;
          end
    'd9 : begin
              Tap_Info_Int.Next_Tap[0] = STAP25;
              Tap_Info_Int.Next_Tap[1] = STAP26;
          end
endcase
```

6.3.2 Add Register Information

Register information is stored in a queue of structs. It will be used to keep a record of all TAP registers accesses in the entire SoC. Include file consists of a function called Create_Reg_Model that will allow a user to fill a struct with various fields. Code 10 shows sample code for Create_Reg_Model.

Figure 46. Code 13. Contents of Register Information File

```
 2 //                TapID,     Opcode,   DR_Width
 3 Create_Reg_Model (CLTAP,     8'hFF,    'd1);
 4 Create_Reg_Model (CLTAP,     8'h02,    'd32);
 5 Create_Reg_Model (CLTAP,     8'h10,    'd6);
 6 Create_Reg_Model (CLTAP,     8'h11,    'd12);
 7 Create_Reg_Model (CLTAP,     8'h12,    'd12);
 8 Create_Reg_Model (CLTAP,     8'h34,    'd32);
 9 Create_Reg_Model (CLTAP,     8'hA0,    'd32);
10
11 Create_Reg_Model (STAP0,     8'hFF,    'd1);
12 Create_Reg_Model (STAP0,     8'h0C,    'd32);
13 Create_Reg_Model (STAP0,     8'h50,    'd10);
14 Create_Reg_Model (STAP0,     8'hA0,    'd32);
15
16 Create_Reg_Model (STAP1,     8'hFF,    'd1);
17 Create_Reg_Model (STAP1,     8'h0C,    'd32);
18 Create_Reg_Model (STAP1,     8'h10,    'd2);
19 Create_Reg_Model (STAP1,     8'h11,    'd4);
20 Create_Reg_Model (STAP1,     8'h12,    'd4);
21 Create_Reg_Model (STAP1,     8'h51,    'd11);
22 Create_Reg_Model (STAP1,     8'hA0,    'd32);
```

Create_Reg_Model arguments are as follows.

- Create_Reg_Model(Tap_ID, Opcode, DR_Width);
- TapID: Retain the same name for TAPID from previous function.
- Opcode: All the TAP opcodes should be mentioned here.
- DR_Width: Width of each Test Data Register (TDR) in each TAP.

User has an option not to provide Opcode information using Create_Reg_Model() and will be able to access register if it is present in RTL. Refer to TapAccess task description to use this feature.

6.3.3 Define TAP Numbers

Tap IDs types are defined with enum type and Tap names are assigned with unique number. Code example is shown in Code 11 (Figure 44). This information is captured in the file `JtagBfmSoCTapNumInfo.svh`

Figure 47. Code 14. Contents of TAP Number File

```
typedef enum int {
  CLTAP             =  'd0,
  STAP0             =  'd1,
  STAP1             =  'd2,
  STAP2             =  'd3,
  STAP3             =  'd4,
  STAP4             =  'd5,
  STAP5             =  'd6,
  STAP6             =  'd7,
  STAP7             =  'd8,
  STAP8             =  'd9,
  STAP9             =  'd10,
  STAP10            =  'd11,
  STAP11            =  'd12,
  STAP12            =  'd13,
  STAP13            =  'd14,
  STAP14            =  'd15,
  STAP15            =  'd16,
  STAP16            =  'd17,
  STAP17            =  'd18,
  STAP18            =  'd19,
  STAP19            =  'd20,
  STAP20            =  'd21,
  STAP21            =  'd22,
  STAP22            =  'd23,
  STAP23            =  'd24,
  STAP24            =  'd25,
  STAP25            =  'd26,
  STAP26            =  'd27,
  STAP27            =  'd28,
  STAP28            =  'd29,
  STAP29            =  'd30,
  NOTAP             =  'hFFFF_FFFF
} Tap_t;
```

6.3.4 Define TAP Strings

Tap Strings are defined and are used for printing logs. Code example is shown in Code 12 (Figure 45). This information is captured in the file `JtagBfmSoCTapStringInfo.svh`

Figure 48. Code 15. Contents of TAP Strings File

```
function string StrTap (input int Tap_Val); begin
    string str;
    case (Tap_Val)
        CLTAP           : begin str = "CLTAP"; end
        STAP0           : begin str = "STAP0"; end
        STAP1           : begin str = "STAP1"; end
        STAP2           : begin str = "STAP2"; end
        STAP3           : begin str = "STAP3"; end
        STAP4           : begin str = "STAP4"; end
        STAP5           : begin str = "STAP5"; end
        STAP6           : begin str = "STAP6"; end
        STAP7           : begin str = "STAP7"; end
        STAP8           : begin str = "STAP8"; end
        STAP9           : begin str = "STAP9"; end
        STAP10          : begin str = "STAP10"; end
        STAP11          : begin str = "STAP11"; end
        STAP12          : begin str = "STAP12"; end
        STAP13          : begin str = "STAP13"; end
        STAP14          : begin str = "STAP14"; end
        STAP15          : begin str = "STAP15"; end
        STAP16          : begin str = "STAP16"; end
        STAP17          : begin str = "STAP17"; end
        STAP18          : begin str = "STAP18"; end
        STAP19          : begin str = "STAP19"; end
        STAP20          : begin str = "STAP20"; end
        STAP21          : begin str = "STAP21"; end
        STAP22          : begin str = "STAP22"; end
        STAP23          : begin str = "STAP23"; end
        STAP24          : begin str = "STAP24"; end
        STAP25          : begin str = "STAP25"; end
        STAP26          : begin str = "STAP26"; end
        STAP27          : begin str = "STAP27"; end
        STAP28          : begin str = "STAP28"; end
        STAP29          : begin str = "STAP29"; end
        NOTAP           : begin str = "NOTAP"; end
    endcase // case
    return str;
end
endfunction : StrTap
```

6.4 How to Add a Test for SoC TAP Network

Follow these steps to add a new test.

1. Create a new file called as MyTapSequence.sv. Start by duplicating existing class from the following file and modify it: `verif/tb/SampleTests/SampleTapSeqLib.sv`

2. Create a new file called as `MyTapTests.sv`. Start by duplicating existing class from the following file and modify it: `verif/tb/SampleTests/SampleTapTests.sv TapSeqLib.sv`

3. Enhance your Sequence by using the APIs listed in section 5.3, List of enDebug APIs.

4. The code shown below describes how the new test will look. The pink font signifies the places where you need to make changes. In the example given below, EN_REGISTER_PRESENCE_CHECK is not used.

5. Run your test:

   ```
   ace -cc -x -t MyNewTest
   ```

Note: The Sequence Class should come first in the compilation order.

Figure 49. Code 16. New Test Sequence Class

```
1413 class Tap5levelHierSequenceTry #(parameter EN_REGISTER_PRESENCE_CHECK = 1) extends JtagBfmSoCTapNwSequences;
1414    JtagBfmSeqDrvPkt Packet;
1415     function new(string name = "Tap5levelHierSequenceTry");
1416         super.new(name);
1417         Packet = new;
1418     endfunction : new
1419
1420     `ovm_sequence_utils(Tap5levelHierSequenceTry, JtagBfmSequencer)
1421     virtual task body();
1422     logic [WIDTH_OF_EACH_REGISTER-1:0] ReturnTdo;
1423        BuildTapDataBase();
1424        Reset(2'b11);
1425
1426        TapAccessCltapcIdcode(CLTAP);
1427        TapAccessSlaveIdcode(STAP0);
1428        TapAccessSlaveIdcode(STAP1);
1429        TapAccessSlaveIdcode(STAP2);
1430
1431        TapAccess(CLTAP,    'h02, DONT_CARE, 0, 32'hCODE_FF01, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1432        TapAccess(STAP0,    'h0C, DONT_CARE, 0, 32'hCODE_0001, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1433        TapAccess(STAP1,    'h0C, DONT_CARE, 0, 32'hCODE_0101, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1434        TapAccess(STAP2,    'h0C, DONT_CARE, 0, 32'hCODE_0201, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1435     endtask : body
1436 endclass : Tap5levelHierSequenceTry
```

Figure 50. Code 17. New Test Class

```
487 class TapTestTry5levelhier extends TapBaseTest;
488
489      `ovm_component_utils(TapTestTry5levelhier)
490      Tap5levelHierSequenceTry #(1) ST;
491
492     function new (string name = "TapTestTry5levelhier", ovm_component parent = null);
493         super.new(name,parent);
494     endfunction : new
495
496     virtual function void build();
497         super.build();
498     endfunction : build
499
500     virtual task run();
501         ovm_report_info("TapTestTry5levelhier","Test Starts!!!");
502         ST = new("ALL Primary Mode");
503         ST.start(Env.PriMasterAgent.Sequencer);
504         ovm_report_info("TapTestTry5levelhier","TapTestTry5levelhier Completed!!!");
505         global_stop_request();
506     endtask : run
507 endclass : TapTestTry5levelhier
```

6.4.1 Positional Mapping of all Arguments for TapAccess

Figure 51. Code 18. Positional Mapping of all Arguments for TapAccess

```
1413 class Tap5levelHierSequenceTry #(parameter EN_REGISTER_PRESENCE_CHECK = 1) extends JtagBfmSoCTapNwSequences;
1414    JtagBfmSeqDrvPkt Packet;
1415     function new(string name = "Tap5levelHierSequenceTry");
1416         super.new(name);
1417         Packet = new;
1418     endfunction : new
1419
1420     `ovm_sequence_utils(Tap5levelHierSequenceTry, JtagBfmSequencer)
1421     virtual task body();
1422     logic [WIDTH_OF_EACH_REGISTER-1:0] ReturnTdo;
1423        BuildTapDataBase();
1424        Reset(2'b11);
1425
1426        TapAccessCltapcIdcode(CLTAP);
1427        TapAccessSlaveIdcode(STAP0);
1428        TapAccessSlaveIdcode(STAP1);
1429        TapAccessSlaveIdcode(STAP2);
1430
1431        TapAccess(CLTAP,    'h02, DONT_CARE, 0, 32'hCODE_FF01, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1432        TapAccess(STAP0,    'h0C, DONT_CARE, 0, 32'hCODE_0001, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1433        TapAccess(STAP1,    'h0C, DONT_CARE, 0, 32'hCODE_0101, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1434        TapAccess(STAP2,    'h0C, DONT_CARE, 0, 32'hCODE_0201, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1435     endtask : body
1436 endclass : Tap5levelHierSequenceTry
```

6.4.2 Positional Mapping of Fewer Arguments for TapAccess

This example shows that TapAccess task can also be used by providing fewer arguments. But user needs to provide at least comma (,) to skip/ignore the argument field.

Figure 52. Code 19. Positional mapping of fewer arguments for TapAccess

```
1413 class Tap5levelHierSequenceTry #(parameter EN_REGISTER_PRESENCE_CHECK = 1) extends JtagBfmSoCTapNwSequences;
1414    JtagBfmSeqDrvPkt Packet;
1415     function new(string name = "Tap5levelHierSequenceTry");
1416         super.new(name);
1417         Packet = new;
1418     endfunction : new
1419
1420     `ovm_sequence_utils(Tap5levelHierSequenceTry, JtagBfmSequencer)
1421     virtual task body();
1422     logic [WIDTH_OF_EACH_REGISTER-1:0] ReturnTdo;
1423        BuildTapDataBase();
1424        Reset(2'b11);
1425
1426        TapAccessCltapcIdcode(CLTAP);
1427        TapAccessSlaveIdcode(STAP0);
1428        TapAccessSlaveIdcode(STAP1);
1429        TapAccessSlaveIdcode(STAP2);
1430
1431        TapAccess(CLTAP,    'h02, DONT_CARE, 0, 32'hCODE_FF01, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1432        TapAccess(STAP0,    'h0C, DONT_CARE, 0, 32'hCODE_0001, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1433        TapAccess(STAP1,    'h0C, DONT_CARE, 0, 32'hCODE_0101, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1434        TapAccess(STAP2,    'h0C, DONT_CARE, 0, 32'hCODE_0201, '1, 0, EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1435     endtask : body
1436 endclass : Tap5levelHierSequenceTry
```

```
1413 class Tap5levelHierSequenceTry #(parameter EN_REGISTER_PRESENCE_CHECK = 1) extends JtagBfmSoCTapNwSequences;
1414    JtagBfmSeqDrvPkt Packet;
1415     function new(string name = "Tap5levelHierSequenceTry");
1416         super.new(name);
1417         Packet = new;
1418     endfunction : new
1419
1420     `ovm_sequence_utils(Tap5levelHierSequenceTry, JtagBfmSequencer)
1421     virtual task body();
1422     logic [WIDTH_OF_EACH_REGISTER-1:0] ReturnTdo;
1423        BuildTapDataBase();
1424        Reset(2'b11);
1425
1426        TapAccessCltapcIdcode(CLTAP);
1427        TapAccessSlaveIdcode(STAP0);
1428        TapAccessSlaveIdcode(STAP1);
1429        TapAccessSlaveIdcode(STAP2);
1430
1431        TapAccess(STAP3,   'hA0, 32'h33, 32, , , , EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1432        TapAccess(STAP2,   'hA0, 32'h32, 32, , , , EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1433        TapAccess(STAP1,   'hA0, 32'h31, 32, , , , EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1434        TapAccess(STAP0,   'hA0, 32'h30, 32, , , , EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1435        TapAccess(CLTAP,   'hA0, 32'h50, 32, , , , EN_REGISTER_PRESENCE_CHECK, ReturnTdo);
1436     endtask : body
1437 endclass : Tap5levelHierSequenceTry
```

6.4.3 Named Mapping of TapAccess with Required Arguments

Named mapping of arguments will provide user flexibility to pass only required fields. SIP DFX recommends using named mapping of TapAccess arguments.

Figure 53. Code 20. Named Mapping of TapAccess with Required Arguments

```
1413 class Tap5levelHierSequenceTry #(parameter EN_REGISTER_PRESENCE_CHECK = 1) extends JtagBfmSoCTapNwSequences;
1414    JtagBfmSeqDrvPkt Packet;
1415     function new(string name = "Tap5levelHierSequenceTry");
1416         super.new(name);
1417         Packet = new;
1418     endfunction : new
1419
1420     `ovm_sequence_utils(Tap5levelHierSequenceTry, JtagBfmSequencer)
1421     virtual task body();
1422     logic [WIDTH_OF_EACH_REGISTER-1:0] ReturnTdo;
1423        BuildTapDataBase();
1424        Reset(2'b11);
1425
1426        TapAccess(.Tap(STAP0),
1427                  .Opcode('hA0),
1428                  .ShiftIn(32'h30),
1429                  .ShiftLength(32),
1430                  .EnRegisterPresenceCheck(EN_REGISTER_PRESENCE_CHECK),
1431                  .ReturnTdo(ReturnTdo));
1432
1433        TapAccess(.Tap(CLTAP),
1434                  .Opcode('hA0),
1435                  .ShiftIn(32'h50),
1436                  .ShiftLength(32),
1437                  .EnRegisterPresenceCheck(EN_REGISTER_PRESENCE_CHECK),
1438                  .ReturnTdo(ReturnTdo));
1439
1440     endtask : body
1441 endclass : Tap5levelHierSequenceTry
```

6.5 Various Topologies

Various topologies are described here with two ports associated with CLTAP. They are Primary JTAG port and Secondary JTAG port. Tertiary ports are not described in this document yet.

6.5.1 Hierarchical Network

In a hierarchical network, a child tap will always be on the same port as its parent with reference to CLTAP. In other words if a Parent is on CLTAP's Secondary port then the child tap will also be on CLTAP's Secondary port.

6.5.1.1 Sequences for Hierarchical Mode: Config Sequence

To configure TAP in hierarchical mode sequences are written as shown below. We need two sequences, one to configure Tap in secondary mode and other to access Tap using secondary interface. Task PutTapOnSecondary is used as part of configuration to place intended Tap in secondary.

Figure 54. Code 21. Secondary Hierarchical Mode: Sample Code for Configuration Sequence

```
2337 class Tap5levelHierSequenceTrySecConfigNew #(parameter EN_REGISTER_PRESENCE_CHECK = 1,
2338                                               parameter Tap_t TAP = STAP0) extends JtagBfmSoCTapNwSequences;
2339
2340    JtagBfmSeqDrvPkt Packet;
2341     function new(string name = "Tap5levelHierSequenceTrySecConfigNew");
2342         super.new(name);
2343         Packet = new;
2344     endfunction : new
2345
2346     `ovm_sequence_utils(Tap5levelHierSequenceTrySecConfigNew, JtagBfmSequencer)
2347
2348     virtual task body();
2349        #display("EN_REGISTER_PRESENCE_CHECK = %0d", EN_REGISTER_PRESENCE_CHECK);
2350        #display("1. TAP = %0s", TAP);
2351        BuildTapDataBase();
2352        Reset(2'b11);
2353        PutTapOnSecondary(TAP);
2354     endtask : body
2355
2356 endclass : Tap5levelHierSequenceTrySecConfigNew
```

6.5.1.2 Sequences for Hierarchical Mode: Data Sequence

To access any Opcode form Tap placed on secondary, TapAccess task is used the way it is in primary. Following example shows how to access register from Tap which is place on secondary.

Figure 55. Code 22. Secondary Hierarchical Mode: Sample Code for Data Sequence

```
2411 class Tap5levelHierSequenceTrySecondaryNew #(parameter EN_REGISTER_PRESENCE_CHECK = 1,
2412                                               parameter Tap_t TAP = STAP0) extends JtagBfmSoCTapNwSequences;
2413
2414    Tap_t tap = TAP;
2415    JtagBfmSeqDrvPkt Packet;
2416     function new(string name = "Tap5levelHierSequenceTrySecondaryNew");
2417         super.new(name);
2418         Packet = new;
2419     endfunction : new
2420
2421     `ovm_sequence_utils(Tap5levelHierSequenceTrySecondaryNew, JtagBfmSequencer)
2422
2423     virtual task body();
2424     logic [WIDTH_OF_EACH_REGISTER-1:0] ReturnTdo;
2425        TapAccess(.Tap(TAP),
2426                  .Opcode('h0C),
2427                  .ShiftIn(32'h0),
2428                  .ShiftLength(32),
2429                  .EnRegisterPresenceCheck(EN_REGISTER_PRESENCE_CHECK),
2430                  .ReturnTdo(ReturnTdo));
2431     endtask : body
2432
2433 endclass : Tap5levelHierSequenceTrySecondaryNew
```

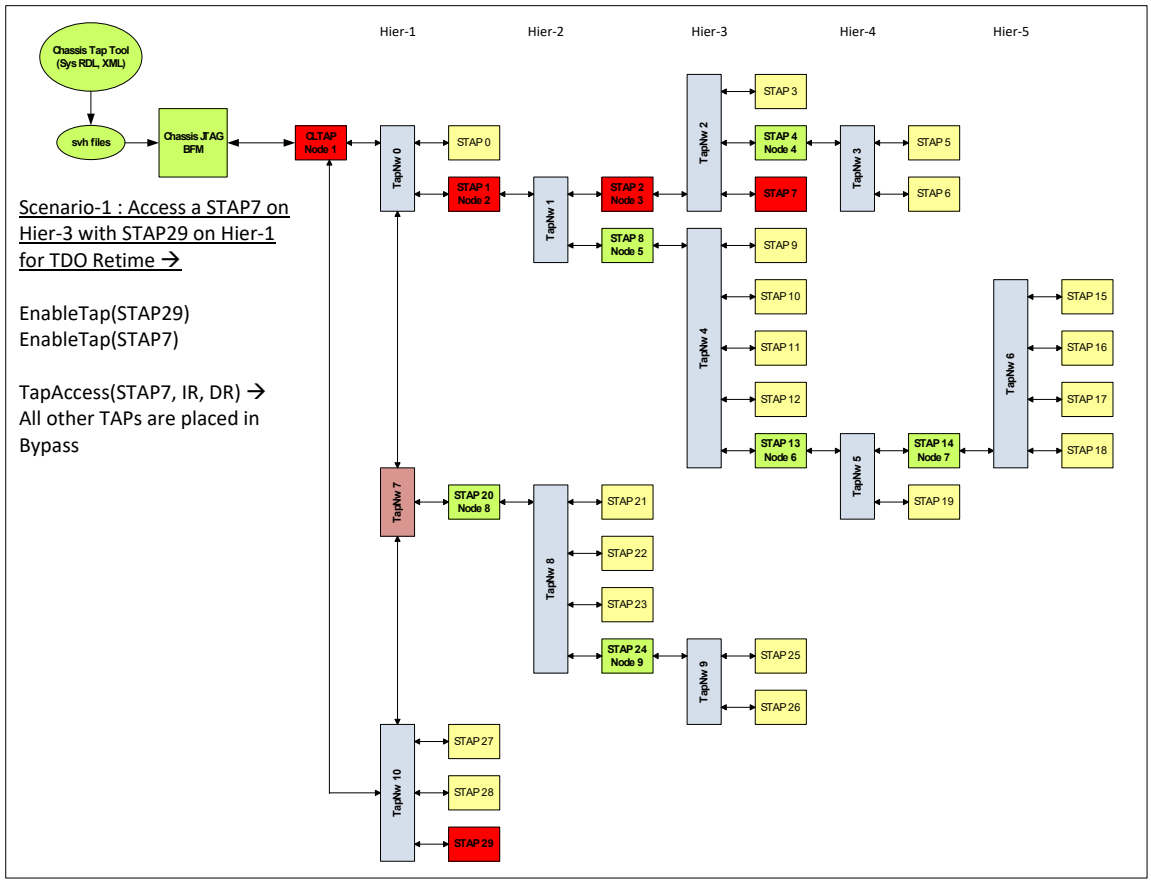
6.5.1.3 Sequences for Hierarchical Mode: Test

Test is written as shown below. It shows how the Primary and Secondary instance of the BFM are used.

Figure 56. Code 23. Secondary Hierarchical Mode: Sample Code for Test

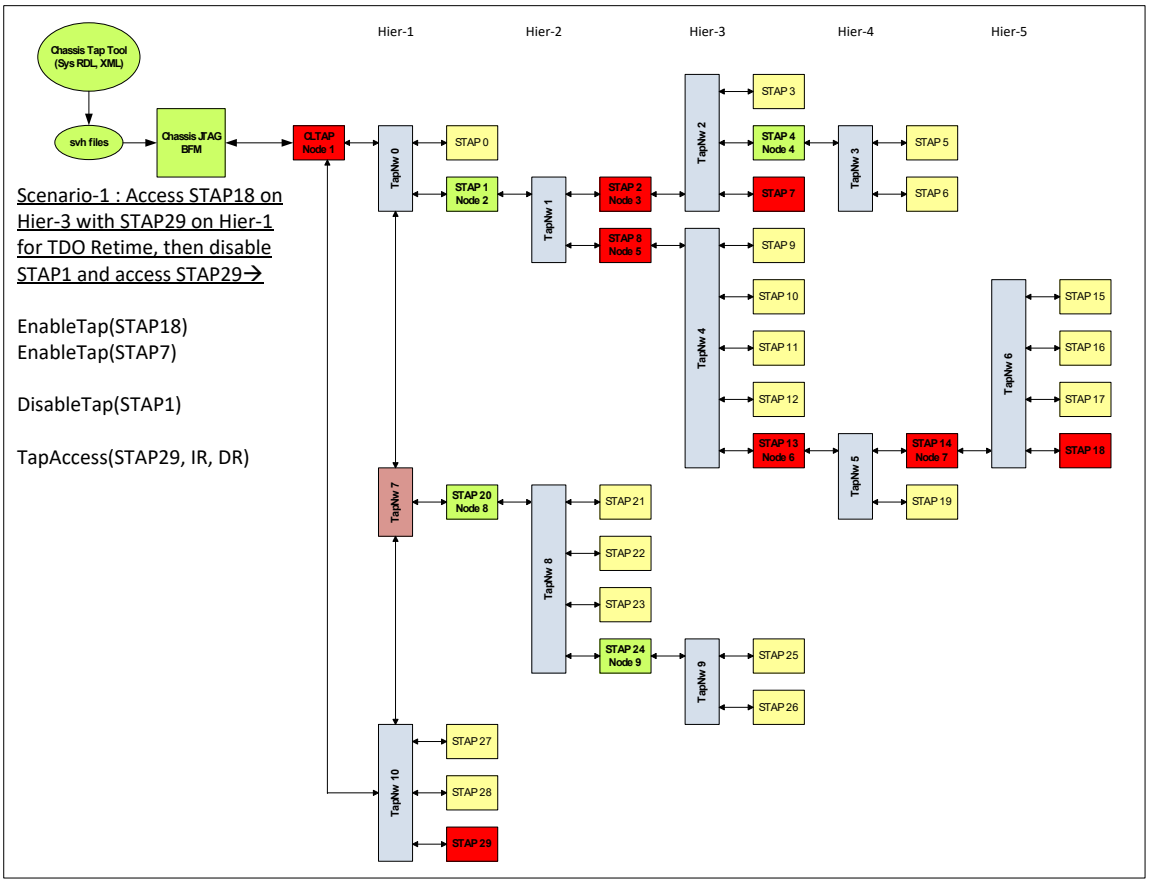
```
533 class TapTestTry5levelhierSecondary extends TapBaseTest;
534    `ovm_component_utils(TapTestTry5levelhierSecondary)
535
536    Tap5levelHierSequenceTrySecConfig #(1, STAP2) ST_CONFIG;
537    Tap5levelHierSequenceTrySecondary #(1, STAP2) ST_SECONDARY;
538
539    TapSequenceReset RESETPRIMARY, RESETSECONDARY;
540    TapSequenceGoToRuti SECONDARY_RUTI;
541
542    function new (string name = "TapTestTry5levelhierSecondary",
543       ovm_component parent = null);
544       super.new(name,parent);
545    endfunction : new
546
547    virtual function void build();
548       super.build();
549    endfunction : build
550
551    virtual task run();
552       ovm_report_info("TapTestTry5levelhierSecondary","Test Starts!!!");
553       RESETPRIMARY = new();
554       RESETSECONDARY = new();
555       ST_CONFIG = new("ALL Primary Mode");
556       ST_SECONDARY = new("ALL secondary Mode");
557       SECONDARY_RUTI = new();
558
559       RESETPRIMARY.start(Env.PriMasterAgent.Sequencer);
560       RESETSECONDARY.start(Env.SecMasterAgent.Sequencer);
561       SECONDARY_RUTI.start(Env.SecMasterAgent.Sequencer);
562
563       ST_CONFIG.start(Env.PriMasterAgent.Sequencer);
564       ST_SECONDARY.start(Env.SecMasterAgent.Sequencer);
565       ovm_report_info("TapTestTry5levelhierSecondary",
566          "TapTestTry5levelhierSecondary Completed!!!");
567       global_stop_request();
568    endtask : run
569
570 endclass : TapTestTry5levelhierSecondary
```

Figure 57. Illustration of the Usage of Multiple Taps being Active



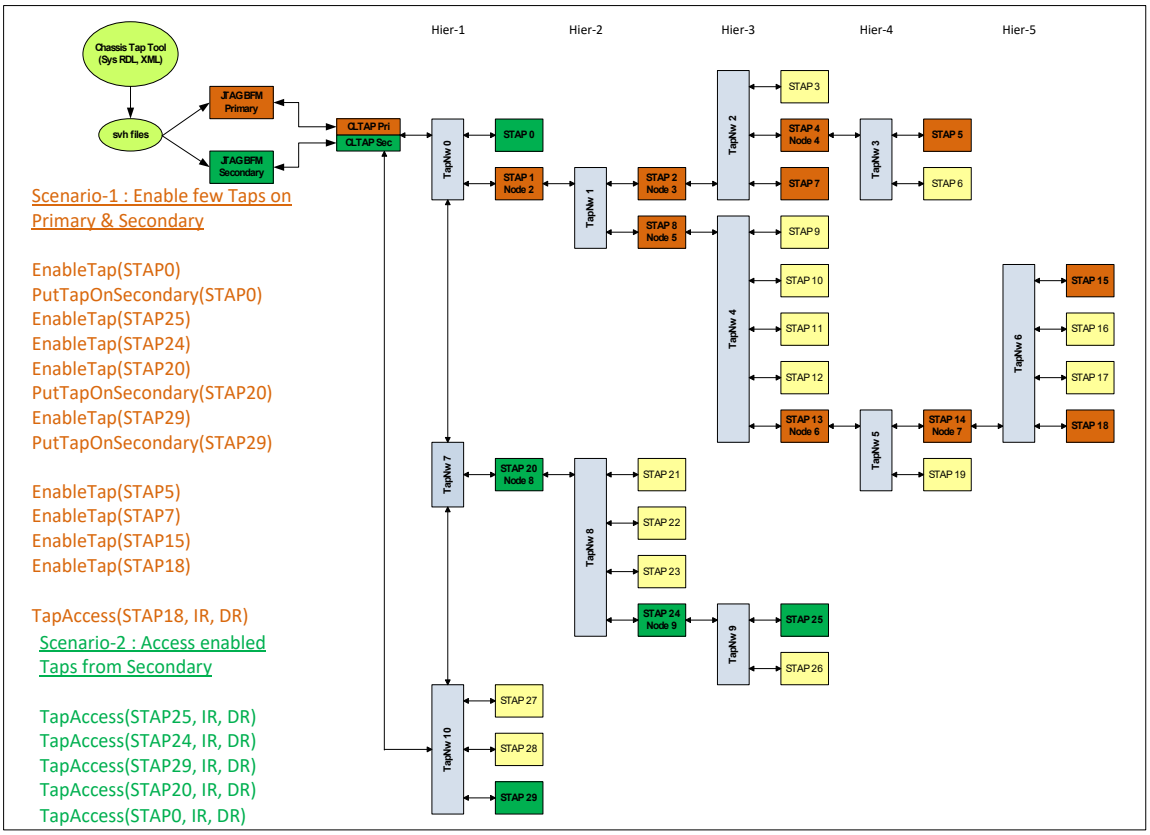
The figure above shows the usage of two targeted TAPs to be active simultaneously. Since these TAPs are on different hierarchies, their parent TAPs should be active. STAP29 at the MSB position of Hier-1 is used as TDO Retime TAP. When an opcode of STAP7 is accessed then all other TAPs are in Bypass mode.

Figure 58. Illustration of the Usage of Multiple Taps along with DisableTap



Usage of TapAccess automatically enables a given TAP along with its parents. However when a Tap has to be disabled then each of the parents has to be individually disabled. In the above example, after enabling STAP18 and STAP7, one of the parents STAP1 is disabled. After this all the other TAPs will be in the enabled state although they cannot be reached. After this when STAP29 is enabled then the path from CLTAP will be directly to STAP29. Next, when STAP1 is enabled, then all the other TAPs (STAP8, STAP2, STAP7, STAP13, STAP14, STAP18) will be back in the network without enabling them.

Figure 59. Illustration of the Usage of Secondary Port



This example shows two instances of the BFM. One of them is connected to the Primary Port of CLTAPC and the other to secondary port. Using the Primary port, STAP0, STAP20, STAP29, STAP24, STAP25 are enabled. The STAP0, STAP20, STAP29 are moved onto the Secondary port of the CLTAPC. Now both the instances of BFM can access parallelly the TAPS in the respective color.

6.6 Porting IP-Level Sequences to SoC level (For IP Providers)

This can be achieved using two methods:

- Use the Tap Aware BFM APIs
- Use the TAPC_Remove feature

Section 6.6.2 (Network Configurations to use BFM at IP-Level) and section 6.6.3 (Usage of TAPC- REMOVE) discuss these two methods.

Note: For reusing the IP level sequences at SoC, make the reusable sequence part of ip_pkg_lib or ip_test_lib (whatever is applicable) and export this thru Ace. SoC in their test case will make an instance of IP sequence and do a .start on the JTAGBFM sequencer declared at SoC level. In theory, this should avoid copying of the IP level sequences in SoC sequence library. Sample sequences and tests can be referred from following files.

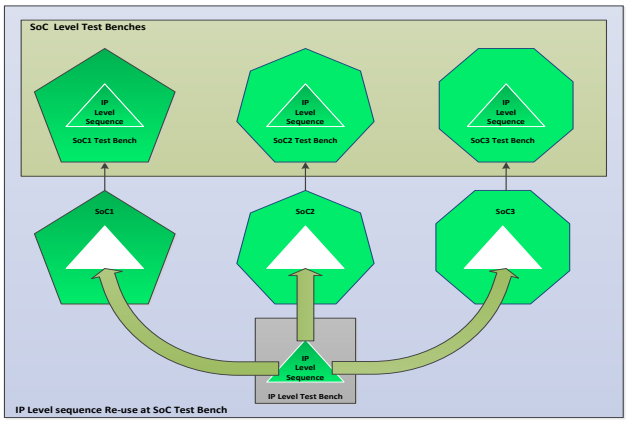
Files:

- `verif/tb/SampleTests/SampleTapSeqLib.s`
- `verif/tb/SampleTests/SampleTapTests.sv`

One of the key features that the BFM enables is the as is reuse of IP level sequences to SoC level. This is possible due to the enumerated TAP names that are used to address each TAP. Therefore, as long as the same enumerated TAP name that is used at the IP level is used at the SoC level, the sequence can be reused.

This BFM is integrated in the Chassis Sandbox verification environment and assuming this continues into the future there will be SoC to SOC porting that can be leveraged due to the architectural consistency and standardization across Regional/Cluster DFx units. For reusing the IP level sequences at SoC, IP provider has to make the reusable sequence part of ip_pkg_lib and export it thru Ace. SoC in their test case will make an instance of the IP sequence and plays it on JTAGBFM sequencer declared at SoC level. First, this avoids copying of the IP level sequences in SoC sequence library. Second, this avoids adding a preamble or a postamble to the IP level sequence.

Figure 60. IP-Level Sequence Re-use at SoC-Level



6.6.1 Recommendation on How to Make IP Tests Visible to SoC

IP sequences should extend from JtagBfmPkg::JtagBfmSoCTapNwSequences

All sequences that are reusable at SoC are preferable to be included in single a file.

This file should be part of IP_TB_LIB. This lib will be part of the ace structure set up. This lib has to be exported through ace. Effectively per IP, there will be at least three libs that will be exported thru ace.

- IP RTL lib
- IP TI Lib
- IP Test Lib → SoC Sequences should be part of this lib/pkg.

TapAccess API should be used with IP name in string format. This string should be reused when SoC Tap Network is constructed. If the SoC appends cluster or region name to a Tap, then SoC has to change the IP level sequence. This needs to be communicated to the SoC.

This API can perform TAP network translation such that IP sequences will be re-usable as FC.

Place the TAP RDL file in the repo and clearly communicate its path to the SoC integrator.

6.6.2 Network Configurations to use BFM at IP-Level

This section showcases the code snippets for six files generated from CTT.

Figure 61. Code 24. IP-Level: Sample Code for JtagBfmSoCTapNetworkInfo.svh

```
case (Node)
   'd0 : begin
            Tap_Info_Int.Next_Tap[0] = NOTAP;
         end
endcase
```

Figure 62. Code 25. IP-Level: Sample Code for JtagBfmSoCTapNumInfo.svh

```
typedef enum int {
  IP_SPECIFIC_TAP =   'd0,
  NOTAP           =   'hFFFF_FFFF
} Tap_t;
```

Figure 63. Code 26. IP level: Sample code for JtagBfmSoCTapParameters.svh

```
parameter NUMBER_OF_HIER  = 1;
parameter NUMBER_OF_STAPS = 0;
```

Figure 64. Code 27. IP level: Sample code for JtagBfmSoCTapStringInfo.svh

```
function string StrTap (input int Tap_Val); begin
    string str;
    case (Tap_Val)
       IP_SPECIFIC_TAP : begin str = "IP_SPECIFIC_TAP"; end
       NOTAP           : begin str = "NOTAP"; end
    endcase // case
    return str;
end
endfunction : StrTap
```

Figure 65. Code 28. IP level: Sample code for JtagBfmSoCTapsInfo.svh

```
//            TAP           SlvIDcode       IR_Width    Node  Sec_connection Hyb_en  Dfx_Security
Create_TAP_LUT (IP_SPECIFIC_TAP, 32'hCODE_1D01,    'h8,       'd0,  'd0,           'd0,     GREEN);
```

Figure 66. Code 29. IP level: Sample code for JtagBfmSoCTapsRegInfo.svh

```
//                TapID,              Opcode,       DR_Width
Create_Reg_Model (IP_SPECIFIC_TAP,  8'hFF,         'd1    );
Create_Reg_Model (IP_SPECIFIC_TAP,  8'h0C,         'd32   );
```

The following code snippet shows how to write sequence at IP level.

Figure 67. Code 30. IP level: Sequence

```
2438 class TapAccessSequence extends JtagBfmSoCTapHwSequences;
2439
2440    JtagBfmSeqDrvPkt Packet;
2441     function new(string name = "TapAccessSequence");
2442         super.new(name);
2443         Packet = new;
2444     endfunction : new
2445
2446     `ovm_sequence_utils(TapAccessSequence, JtagBfmSequencer)
2447
2448     virtual task body();
2449        TapAccessSlaveIdcode(IP_SPECIFIC_TAP);
2450        TapAccess(.Tap(TAP),
2451                  .Opcode('h0C),
2452                  .ShiftIn(32'h0),
2453                  .ShiftLength(32),
2454                  .EnRegisterPresenceCheck(1),
2455                  .ReturnTdo(ReturnTdo));
2456
2457     endtask : body
2458
2459 endclass : TapAccessSequence
```

Figure 68. Code 31. IP level: Test

```
630 class TapAccessSoCTest extends TapBaseTest;
631
632     `ovm_component_utils(TapAccessSoCTest)
633     TapSequenceReset PG;
634     TapAccessSequence TAS;
635
636     function new (string name = "TapAccessSoCTest",
637        ovm_component parent = null);
638        super.new(name,parent);
639     endfunction : new
640
641     virtual function void build();
642        super.build();
643     endfunction : build
644
645     virtual task run();
646        ovm_report_info("TapAccessSoCTest","Test Starts!!!");
647        PG     = new("Powergood reset");
648        TAS    = new("ALL Primary Mode");
649        PG.start(Env.PriMasterAgent.Sequencer);
650        TAS.start(Env.PriMasterAgent.Sequencer);
651        ovm_report_info("TapAccessSoCTest","Test Completed!!!");
652        global_stop_request();
653     endtask : run
654
655 endclass : TapAccessSoCTest
```

6.6.3 Usage of TAPC- REMOVE

Tap remove feature provides an option to access TAP directly to user. Follow these steps to write sequences to use this capability.

1. Identify the TAP you want to access using Remove feature.
2. In Figure 69 (Code 32) shown below, STAP6 is chosen.
3. Create Tap Network information using Build_TAP_DataBase.
4. Enable the Tap that the user wants to access directly using EnableTap.
5. Follow any order to remove the Taps leading to the Tap of interest.
6. Access the desired Tap using API MultipleTapRegisterAccess as if the target Tap is the only available Tap.

In example shown SLVIDCODE is accessed for STAP17.

Note: Refer to the TAPNetwork hierarchy shown in the beginning of the chapter.

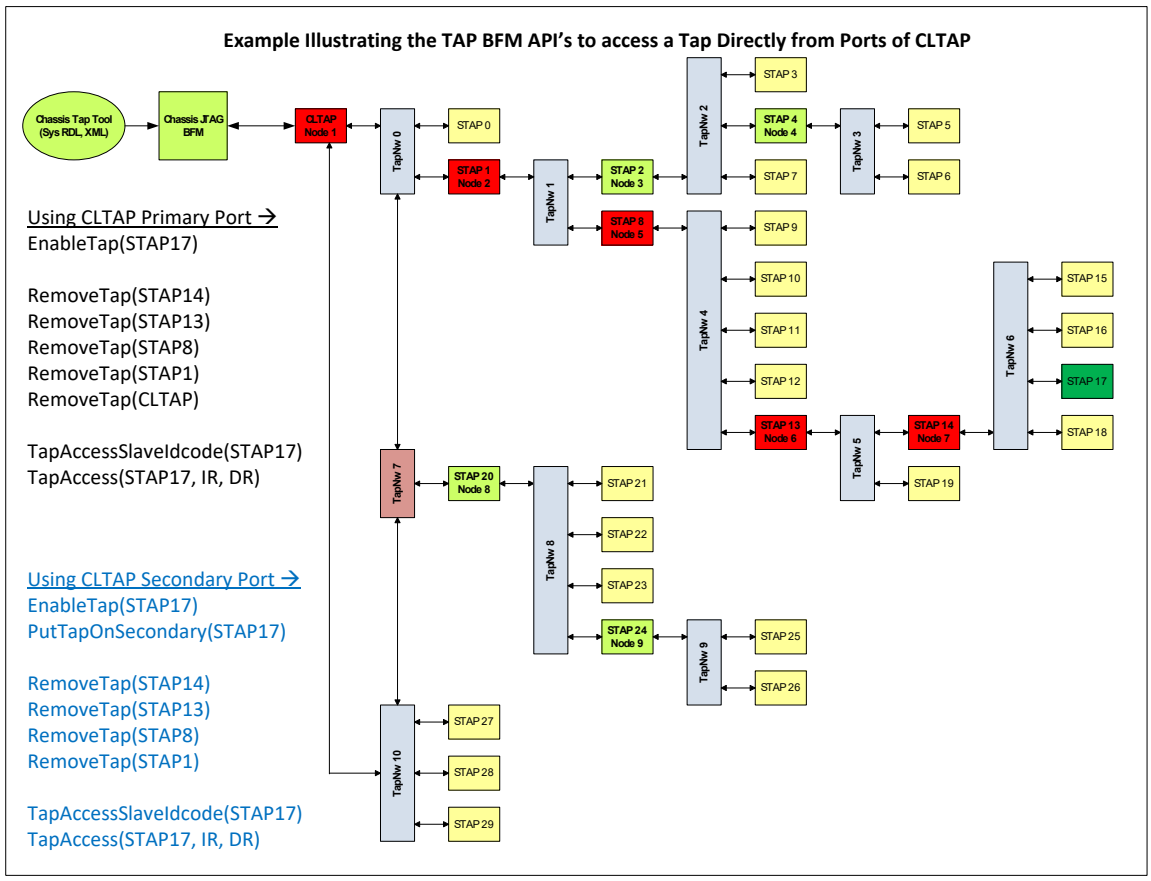
Figure 69. Code 32. SoC-Level Usage of TAPC_Remove Feature

```
2798 class Tap5levelHierSequenceTry_removeNew
2799     #(parameter EN_REGISTER_PRESENCE_CHECK = 1,
2800       parameter Tap_t TAP = 0) extends JtagBfmSoCTapNwSequences;
2801
2802     JtagBfmSeqDrvPkt Packet;
2803     function new(string name = "Tap5levelHierSequenceTry_removeNew");
2804        super.new(name);
2805        Packet = new;
2806     endfunction : new
2807
2808     `ovm_sequence_utils(Tap5levelHierSequenceTry_removeNew, JtagBfmSequencer)
2809
2810      virtual task body();
2811         BuildTapDataBase();
2812         Reset(2'b11);
2813         EnableTap(STAP17);
2814         RemoveTap(STAP14);
2815         RemoveTap(STAP13);
2816         RemoveTap(STAP8);
2817         RemoveTap(STAP1);
2818         RemoveTap(CLTAP);
2819         TapAccessSlaveIdcode (STAP17);
2820      endtask : body
2821
2822 endclass : Tap5levelHierSequenceTry_removeNew
```

Figure 70. Code 33. SoC Level usage of TAPC_Remove test

```
845 class TapTestTry5levelhier_remove_detailedNew extends TapBaseTest;
846
847     `ovm_component_utils(TapTestTry5levelhier_remove_detailedNew)
848
849   function new (string name = "TapTestTry5levelhier_remove_detailedNew", ovm_component parent = null);
850       super.new(name,parent);
851   endfunction : new
852
853   virtual function void build();
854       super.build();
855   endfunction : build
856
857   virtual task run();
858     Tap5levelHierSequenceTry_removeNew #(1, 0) ST;
859     ovm_report_info("TapTestTry5levelhier_remove_detailedNew","Test Starts!!!");
860     ST = new("ALL Primary Mode");
861     ST.start(Env.PriMasterAgent.Sequencer);
862     ovm_report_info("TapTestTry5levelhier_remove_detailedNew",
863                     "TapTestTry5levelhier_remove_detailedNew Completed!!!");
864     global_stop_request();
865   endtask : run
866 endclass : TapTestTry5levelhier_remove_detailedNew
```

Figure 71. Illustration of the Usage of RemoveTap



For example, STAP17 is a HIP TAP. The figure above shows how the sequences written at the HIP level can be ported to SoC. If the HIP sequences are written with the non-Tap aware APIs (E.g LoadIR, LoadDR) then they can be ported to SoC level by the usage of TAPC_REMOVE opcode. The BFM API helps achieve this purpose. The argument to this API is the enumerated TapName. Using the CLTAP's Primary port we remove the various Taps on the network leading to STAP17 in any order. Usage of RemoveAPI is order independent. When you access the IR,

DR of STAP17, then irrespective of the number of Taps in between, the required padding is auto calculated by the BFM. Similarly from the CLTAP's secondary port.

6.6.4 Usage of Tertiary Port

BFM now supports Parametrized number of Tertiary ports. To add this capability to the Environment, sample files located at the following places can be referenced to see how additional instances of the BFM can be created for each Tertiary port.

Files:

- `verif/tb/SampleTests/SampleTapSeqLib.sv`
- `verif/tb/SampleTests/SampleTapTests.sv`

Figure 72. Code 34. SoC-Level Usage of Multiple BFM Interfaces in Testbench Top

File:

- `verif/tb/SampleTests/SampleTapTop.sv`

  Showing multiple instances of JTAGBFM interface.

```
54    JtagBfmIntf #(.CLOCK_PERIOD (10000), .PWRGOOD_SRC (0), .CLK_SRC(0)) Primary_if(soc_powergood_rst_b, soc_clock);
55    JtagBfmIntf #(.CLOCK_PERIOD (33333), .PWRGOOD_SRC (0), .CLK_SRC(0)) Secondary_if(soc_powergood_rst_b, soc_clock);
56    JtagBfmIntf #(.CLOCK_PERIOD (55555), .PWRGOOD_SRC (0), .CLK_SRC(0)) Teritiary0_if(soc_powergood_rst_b, soc_clock);
57    JtagBfmIntf #(.CLOCK_PERIOD (77777), .PWRGOOD_SRC (0), .CLK_SRC(0)) Teritiary1_if(soc_powergood_rst_b, soc_clock);
```

  Showing multiple instances of JTAGBFM interface.

```
101    TapTestIsland i_TapTestIsland(Primary_if,Secondary_if,Teritiary0_if,Teritiary1_if);
102    // Fix for HSD4256873
103    defparam i_TapTestIsland.TAPNWVIF       = "ovm_test_top.Env";
104    defparam i_TapTestIsland.PRI_JTAGBFMVIF = "ovm_test_top.Env.PriMasterAgent.*";
105    defparam i_TapTestIsland.SEC_JTAGBFMVIF = "ovm_test_top.Env.SecMasterAgent.*";
106    defparam i_TapTestIsland.TER0_JTAGBFMVIF = "ovm_test_top.Env.Ter0MasterAgent.*";
107    defparam i_TapTestIsland.TER1_JTAGBFMVIF = "ovm_test_top.Env.Ter1MasterAgent.*";
108
```

Figure 73. Code 35. SoC level usage of Multiple BFM Agent instantiations.

File:

- `verif/tb/SampleTests/SampleTapEnv.sv`

  Showing multiple instances of the JTAGBFM.

```
59 class TapEnv extends sla_tb_env;
60
61     // Components of the environment
62     JtagBfmCfg            jtagBfmCfg;
63
64     JtagBfmMasterAgent    PriMasterAgent;
65     JtagBfmMasterAgent    SecMasterAgent;
66     JtagBfmMasterAgent    Ter0MasterAgent;
67     JtagBfmMasterAgent    Ter1MasterAgent;
68
```

  All Sequencers registered to Saola Virtual Sequencer.

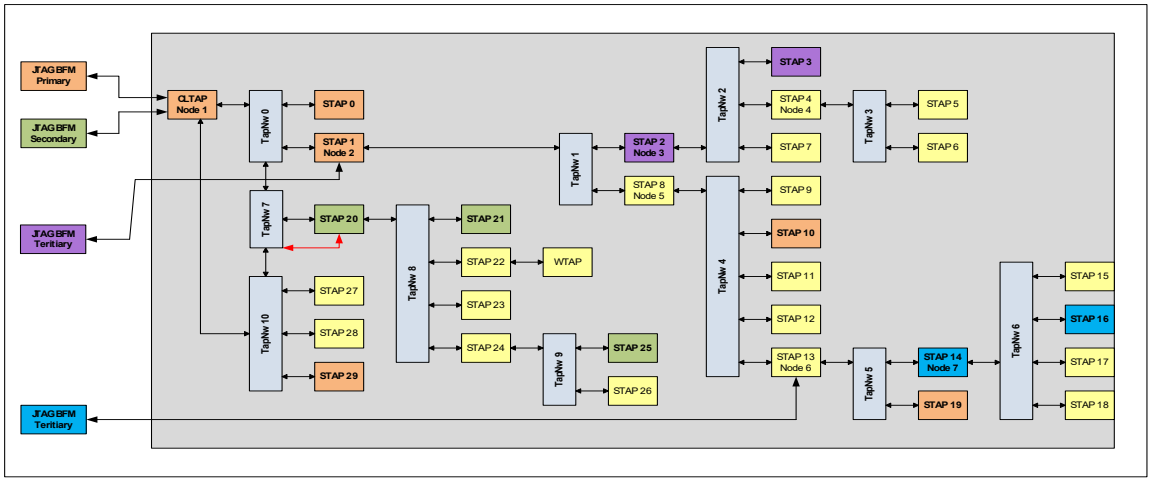
```
227         if (_level == SLA_TOP) begin
228            void'(this.add_sequencer("SLA_SEQUENCER", "jtag_id1", PriMasterAgent.Sequencer));
229            void'(this.add_sequencer("SLA_SEQUENCER", "jtag_id2", SecMasterAgent.Sequencer));
230            void'(this.add_sequencer("SLA_SEQUENCER", "jtag_id3", Ter0MasterAgent.Sequencer));
231            void'(this.add_sequencer("SLA_SEQUENCER", "jtag_id4", Ter1MasterAgent.Sequencer));
232            this.set_test_phase_type(get_name(), "FLUSH_PHASE", "Tap_flush_seq");
233         end
```

File:

- `verif/tb/SampleTests/SampleTapBaseTest.sv`

```
44          set_config_int("Env.Ter0MasterAgent.Sequencer", "count", 0);
45          set_config_int("Env.Ter1MasterAgent.Sequencer", "count", 0);
46          set_config_int("Env.SecMasterAgent.Sequencer", "count", 0);
47          set_config_int("Env.PriMasterAgent.Sequencer", "count", 0);
48          set_config_int("S1","number_of_tap",2);
```

Figure 74. Illustration of the Usage of Multiple Ports



The above diagram has a color coding that shows the various active TAPs on different JTAG ports. They are listed in Table 11. The TAPs in yellow colored boxes are in isolated mode for the purpose of this explanation.

Table 11. JTAG Port Association of Various TAPs

Primary (CLTAP)	Secondary (CLTAP)	Tertiary0 (Secondary of STAP1)	Tertiary1 (Secondary of STAP13)
CLTAP	STAP20	STAP2	STAP14
STAP0	STAP21	STAP3	STAP16
STAP1	STAP25		
STAP10			
STAP19			
STAP29			

Using the Primary instance of the BFM, the configuration of the other ports is set. A TAP that is already accessed by the below APIs will be enabled first before moving it on a different port.

- PutTapOnSecondary(STAP20)
- PutTapOnTertiray(STAP2)
- PutTapOnTertiray(STAP14)

Using the Secondary instance of the BFM, STAP20, STAP21, STAP25 are accessed. Since STAP20 is already enabled, the others can be enabled now.

- EnableTap(STAP21)
- EnableTap(STAP25)

Using the Tertiary0 instance of the BFM, STAP2, STAP3 are accessed.

- EnableTap(STAP3)

Using the Tertiary1 instance of the BFM, STAP14, STAP16 are accessed.

- EnableTap(STAP16)

If for some reason, STAP14, STAP16, have to be used on Tertiray0 port, then the below command can be used.

- DisableTertiaryPort(STAP13)

Figure 74 has a color coding that shows the various active TAPs on different JTAG ports. They are listed in the following table. The TAPs in yellow colored boxes are in isolated mode for the purpose of this explanation.

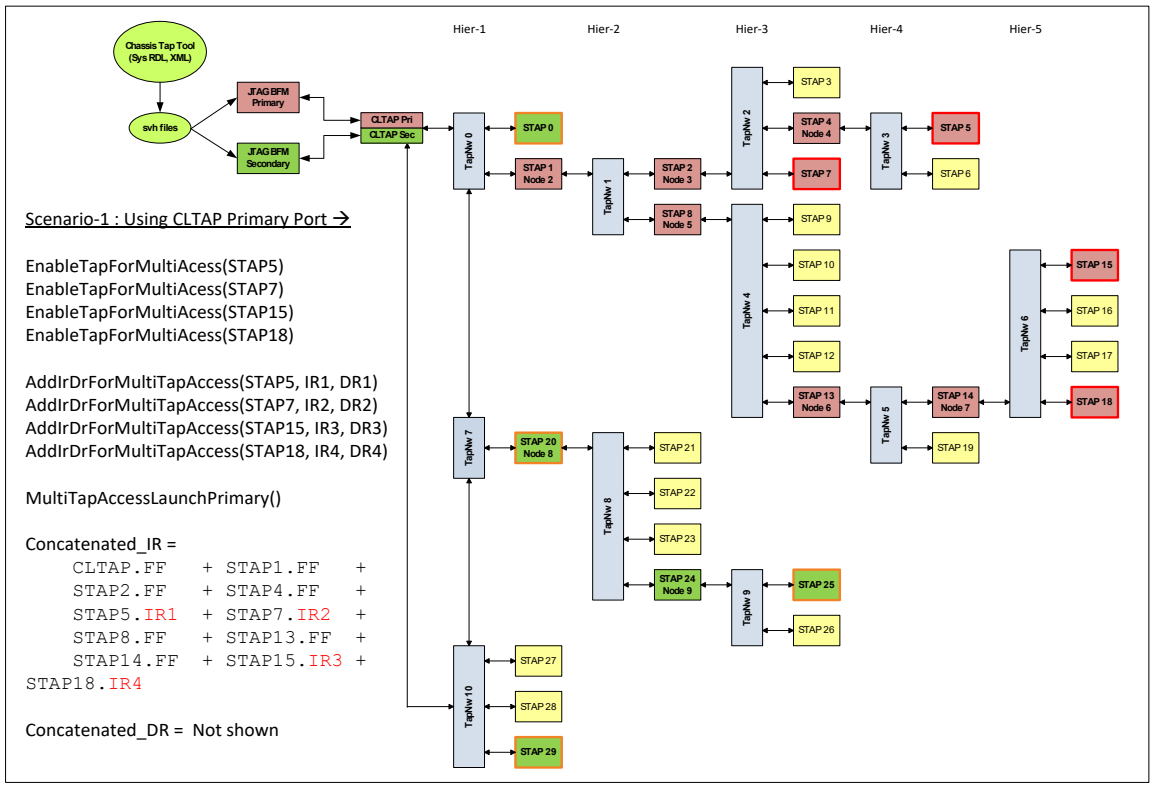
6.6.5 Usage of MultiAccess

MultiAccess is supported from Primary and Secondary ports only. It allows a mechanism to shift different IRs in several TAPs in a single ScanIR/DR operation.

- The TAPs associated with Primary port are marked in Pink color and they are identified by the usage of API EnableTapForMultiAccess.
- The ones that are circled in Red are the ones marked for Multi Access with the IR pointing to the one selected by the API AddIrDrForMultiTapAccess.
- The others that are in Pink will be in Bypass.

After this the concatenated IR and DR vectors is calculated and a IR/DR scan operation is issued when MultiTapAccessLaunchPrimary API is called.

Figure 75. Illustration of the Usage of MultiTap



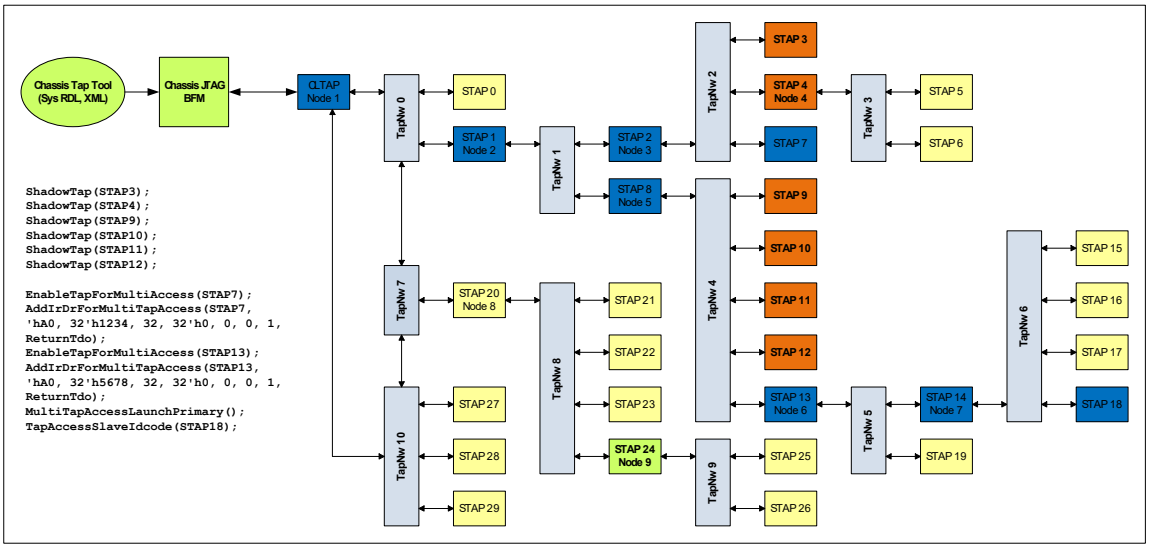
Similar to the code shown in Figure 75, the same can be worked out from Secondary JTAG port as well. After this the concatenarted IR and DR vectors is calculated and a IR/DR scan operation is issued when MultiTapAccessLaunchSecondary API should be called.

6.6.6 Usage of ShadowTap

Shadow mode in TapNetwork can be exercised by using ShadowTap API. Following figure shows the Taps that are placed on Shadow mode. Also, figure has code example of sequence used in test environment as illustration purpose. Here are the sequences of operation in the given hierarchy.

To place STAP3 in Shadow mode, place STAP1and STAP2 on Normal mode. Later place STAP4 on Shadow mode. To place STAP9 on Shadow mode, place STAP8 on Normal mode. After this, place STAP9, STAP10, STAP11 and STAP12 on Shadow mode. This operation is followed by placing STAP13 in Normal mode. We used Multi TapAccess capability to access all enabled TAPs in single IR-DR chain. Add STAP7 and STAP13 to MultiAccess with appropriate IR and DR values followed by MultiTapAccessLaunchPrimary. At the end of operation STAP7, STAP4 and STAP3 will have opcode A0 and data 'h1234. Similarly, STAP13, STAP12, STAP11, STAP10 and STAP9 will have opcodes AO and data 'h5678. To illustrate that even STAP18 can be accessed with the given configuration. This requires enabling STAP14 and STAP18. In the current example, SLVIDCODE is accessed and checked for its correctness using TapAccessSlaveIdcode.

Figure 76. Illustration of the Usage of MultiTap



6.6.7 Taplink Collaterals

Taplink support is added to JTAG BFM. In the current version (2.9), Sandbox-IPGen is used to generate collaterals required for JTAG BFM and this section describes the steps and commands for the generation.

6.6.7.1 Pre-requisites

User need to have sbx group permissions.

```
cmd > groups
users sklall soc socenv cpu1274 socrtl soc73 hdk10nm sbx mpgall cdftsip cdft
```

6.6.7.2 Taplink Architecture Definition

Users are expected to manually write the taplink network architecture using `*.json`. See Figure 77 (Code 36) for the code.

Figure 77. Code 36. Snapshot of `*.json` file

```
{
  "components" : {
     "cltap" : {
       "corekit": "cltapc",
       "ip_type": "MAINTAP",
      "bscan_size": "500",
       "tdo_max_delay_override": "10",
            "params" : {
              "CLTAPC_RTDR_IS_BUSSED": 1,
              "CLTAPC_TDR_END_ADDRESS": 256,
              "CLTAPC_USES_TAPLINK": 1,
              "CLTAPC_ENABLE_CLTAPC_REMOVE": 0,
              "CLTAPC_IR_SIZE": 14
        },
       "rdl_udp_override" : {
         "top_level_udp" : {
           "TapRdlVersion" : "0.8",
           "TapSlvIdCodeInfo" : "8",
           "TapSlvIdRegionNum" : "2",
           "TapRegionName" : "soc_core"
         }
       }
     },

     "sb2tap_top" : {
         "corekit": "sb2tap_wrapper_top",
         "ip_type": "SB2TAP",
         "idcode": "32'h0000_beef",
         "params" : {
              "SB2TAP_TRACKER_TAP_IR_LENGTH_EQUALS_CLTAP": 14
         }
     },

     "stap0" : {
       "corekit": "stap",
       "ip_type": "STAP",
            "params" : {
              "STAP_RTDR_IS_BUSSED": 1,
               "STAP_NUMBER_OF_WTAPS_IN_NETWORK": 1,
               "STAP_WTAP_NETWORK_ONE_FOR_SERIES_ZERO_FOR_PARALLEL": 1,
```

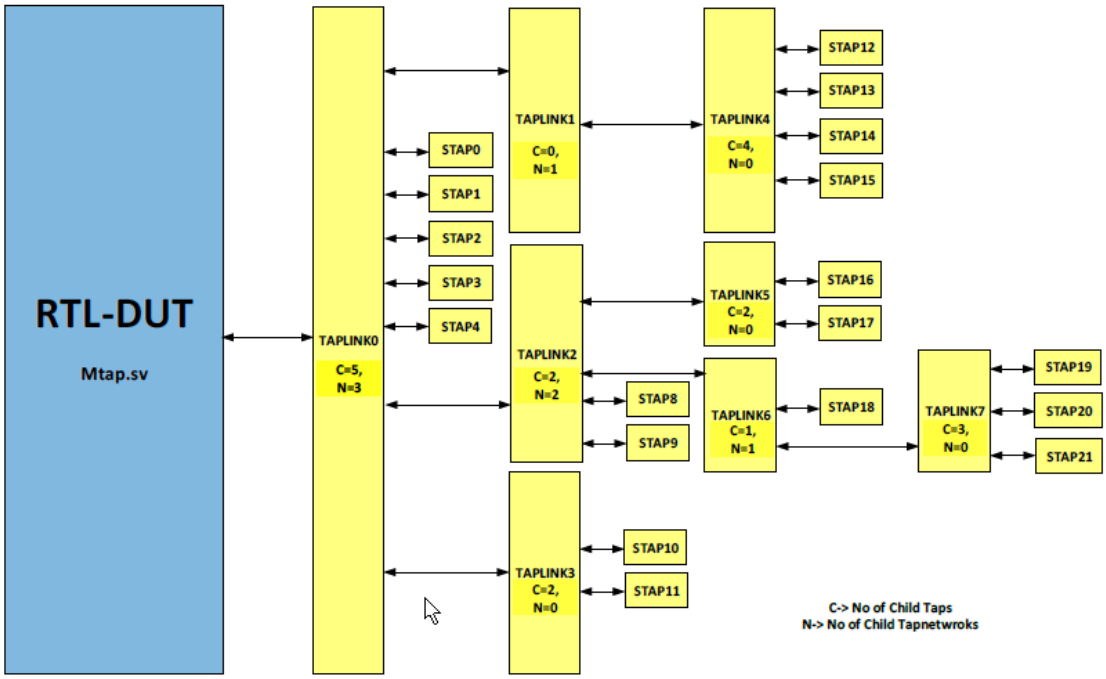
```
102       "taplink_0" : {
103         "ip_type": "TAPLINKNW",
104         "corekit": "taplinknw"
105       },
106       "taplink_1" : {
107         "corekit": "taplinknw",
108         "ip_type": "TAPLINKNW",
109         "cfg_opcode": "14'h80"
110       },
111       "taplink_2" : {
112         "corekit": "taplinknw",
113         "ip_type": "TAPLINKNW",
114         "cfg_opcode": "14'h84"
115       },
116       "taplink_3" : {
117         "corekit": "taplinknw",
118         "ip_type": "TAPLINKNW",
119         "cfg_opcode": "14'h88"
120       },
121       "taplink_4" : {
122         "ip_type": "TAPLINKNW",
123         "corekit": "taplinknw"
124       },
125       "taplink_5" : {
126         "ip_type": "TAPLINKNW",
127         "corekit": "taplinknw",
128        "ir_extend_match_val": "2'b11"
129       },
130       "taplink_6" : {
131         "ip_type": "TAPLINKNW",
132         "corekit": "taplinknw"
133       }
134
```

```
138   "network" : {
139     "properties": {
140       "write_leaf_connections": 0,
141       "network_security": "green",
142       "generate_rtdr_connections": 1,
143       "top_system_name": "sbox",
144       "taplinknw_cfg_opcode_base": "14'h70"
145     },
146
147      "topology": {
148        "cltap" : {
149               "idcode": "32'ha265013",
150               "children": ["taplink_0"]
151        },
152        "taplink_0": {
153               "taplba": "{14'h540,14'h140,14'h144}",
154               "tapgba": "{14'h1A0, 14'h4A0, 14'h4A0}",
155          "children": ["dfx_aggregator", "stap0", "stap1", "Reserved_taplink_0
    ", "taplink_1", "taplink_2"]
156        },
157        "taplink_1": {
158                "taplba": "{14'h148}",
159                "tapgba": "{14'h4A0}",
160           "children": ["stap3"]
161        },
162        "taplink_2": {
163               "taplba": "{14'h164, 14'h14C, 14'h150, 14'h154}",
164               "tapgba": "{14'h4A0, 14'h1A0, 14'h5A0, 14'h5A0}",
165          "children": ["stap4", "Reserved_stap_1", "stap5", "stap6", "taplink_
    3"]
166        },
```

6.6.7.3 Sample Topology

Here is the sample topology of Taplink.

Figure 78. Taplink Topology



For information on the `*.json` file, refer the following links:

- http://chassis.intel.com//tlink_ip_gen_recipe.html
- http://chassis.intel.com//Tlink_Fregently_Ask_Q.html

Also, users can contact sandbox team for more info on `*.json` file.

6.6.7.4 Steps

1. Clone the Taplink IP Gen with the appropriate version.

```
cmd> git clone /p/sbx/release/IP/IP_CONFIG_SOC/dfx/dft/taplinknet_ip-chs-a0-17ww16b
cmd> cd taplinknet_ip-chs-a0-17ww16b/
```

2. Follow the steps below and also in the README file.

```
cmd> setenv TLINK_REPO_ROOT $cwd
cmd> setenv IP_RELEASE /p/sbx/release/IP/IP_CONFIG_SOC
##. Spec
cmd> setenv TLINK_SPEC_DIR $TLINK_REPO_ROOT/ipgen_input/reference/sbox
cmd> setenv TLINK_GEN_DIR $TLINK_REPO_ROOT/ipgen_output/sbox
# ITPP package poiner
cmd> setenv SPF_ITPP_VER
$IP_RELEASE/verif_misc/spf_itpp_pkg/spf_itpp_pkg_2015ww48
cmd> source scripts/reference/setup
cmd> gmake -f TLINKNetworkGen/Makefile gen_tlink_network
```

3. To compile:

   ```
   cmd> ace -cc
   ```

4. To run test to access CLTAP/TAPLINK

   ```
   cmd> ace -x -t "manual_all_idcode_test:seed_1+model_tap_model"
   ```

   Jtag BFM collaterals are generated to the following location.

   ```
   cmd> cd ipgen_output/sbox/tlink_network/verif/jtag_bfm_files/
   ```

   - cmd> ls
   - JtagBfmSoCTlinkTapNetworkInfo.svh
   - JtagBfmSoCTlinkTapNumInfo.svh
   - JtagBfmSoCTlinkTapParameters.svh
   - JtagBfmSoCTlinkTapStringInfo.svh
   - JtagBfmSoCTlinkTapsInfo.svh
   - JtagBfmSoCTlinkTapsRegInfo.svh

5. Copy files generated to project location and set environment variable.

   ```
   cmd> setenv GENERATED_FILES_FOR_JTAGBFM <user_preferred_area>
   ```

   Example:

   ```
   cmd> setenv GENERATED_FILES_FOR_JTAGBFM
   ${IP_ROOT}/verif/ctt_files/taplink
   ```

   Users must run compilation along with JTAG_BFM_TAPLINK_MODE macro. This is project dependent and optionally users can add this to Localivars* file also.

   ```
   cmd> <user compile command> -vlog_opts "+define+JTAG_BFM_TAPLINK_MODE"
   ```

   Example:

   ```
   simbuild -dut cltapc -ace_args ace -cc -vlog_opts
   "+define+JTAG_BFM_DEBUG_MSG+define+JTAG_BFM_TAPLINK_MODE+define+USE_G
   ENERATED_FILES_FOR_JTAGBFM" -ace_args- -1c -CUST TGL
   ```

6.6.7.5 Taplink: Sample Files

Sandbox generates six required files for JTAG BFM. Out of which, four files have same content as in files generated using CTT and other two files are updated to match Taplink architecture.

6.6.7.5.1 Files: Unchanged

The following files are same as the files generated from CTT.

Figure 79. Code 37. Sample Code for JtagBfmSoCTlinkTapNetworkInfo.svh

```
case (Node)
    'd0 : begin
            Tap_Info_Int.Next_Tap[0] = NOTAP;
     end
    'd1 : begin
        Tap_Info_Int.Next_Tap[0] = dfx_aggregator;
        Tap_Info_Int.Next_Tap[1] = stap0;
        Tap_Info_Int.Next_Tap[2] = stap1;
    end
endcase
```

Figure 80. Code 38. Sample Code for JtagBfmSoCTlinkTapNumInfo.svh

```
typedef enum int {
      cltap           = 'd0,
      dfx_aggregator  = 'd1,
      stap0           = 'd2
      stap1           = 'd3,
} Tap_t;
```

Figure 81. Code 39. Sample Code for JtagBfmSoCTlinkTapParameters.svh

```
parameter NUMBER_OF_HIER   = 10;
parameter NUMBER_OF_STAPS  = 14;
parameter NUMBER_OF_TERTIARY_PORTS = 0;
```

Figure 82. Code 40. Sample Code for JtagBfmSoCTlinkTapStringInfo.svh

```
function string StrTap (input int Tap_Val); begin
   string str;
   case (Tap_Val)
       cltap          : begin str = "cltap"; end
       dfx_aggregator : begin str = "dfx_aggregator"; end
       stap0          : begin str = "stap0"; end
       stap1          : begin str = "stap1"; end
       NOTAP          : begin str = "NOTAP"; end
   endcase
   return str;
end
endfunction : StrTap
```

6.6.7.5.2 Files: Changed

Figure 83. Code 41. Sample Code for JtagBfmSoCTlinkTapsInfo.svh

```
//             TAP            SlvIDcode     IDcode      IR_Width   LBA      GBA    Hybrid_en Dfx_Security Hierarchy_Level PositionOfTap IsVendorTap IsIdcode_eq0C
Create_TAP_LUT (cltap,         32'ha265013,  32'ha265013,  14,   0,       0,       0,      GREEN,        0,               0,           0,         'h02);
Create_TAP_LUT (dfx_aggregator, 32'h055_dead, 32'h055_dead,  8,   14'h540, 14'h1A0,  0,      GREEN,        1,               0,           0,         'h0C);
Create_TAP_LUT (stap0,         32'h001_dead, 32'h001_dead,  8,   14'h140, 14'h1A0,  0,      GREEN,        1,               1,           0,         'h0C);
Create_TAP_LUT (stap1,         32'h003_dead, 32'h003_dead,  8,   14'h144, 14'h1A0,  0,      GREEN,        1,               2,           0,         'h0C);
```

Figure 84. Code 42. Sample Code for JtagBfmSoCTlinkTapsRegInfo.svh

```
//               TapID,    Opcode,  DR_Width, Dummy_Hi (int), Dummy_lo (int), IsTapLinkOp (bit)
Create_Reg_Model (cltap,   14'h31,  24,       10,             0,             0);
Create_Reg_Model (cltap,   14'h32,  24,       10,             0,             0);
Create_Reg_Model (cltap,   14'h34,  24,       10,             0,             0);
Create_Reg_Model (cltap,   14'h36,  24,       10,             0,             0);
Create_Reg_Model (cltap,   14'h44,  8,        10,             0,             0);
```

Dummy_Hi is post delay and Dummy_lo is pre delay in the arguments shown above.

7 Monitoring and Checking the Protocol

7.1 Using the Monitor and Tracker

The Tracker, Input Monitor and Output Monitor are the same as for the SIP BFM. The tracker files, by default, are named JTAG_TAP_PORT_P<n>_jtag_tracker_normal.out and JTAG_TAP_PORT_P<n>_jtag_tracker_runtime.out, where <n> is the port number.

7.2 Connecting a Scoreboard to the Testbench

Not applicable to this IP/IPSS

7.3 Controlling Messages

All display messages in the test are controlled using OVM_FLAGS. (OVM_NONE, OVM_MEDIUM, OVM_LOW, OVM_HIGH).

Define +OVM_VERBOSITY in the command line to control display messages.

7.4 Controlling Assertions

Standard IEEE assertions for 1149.1 spec are added here. These are part of JtagBfmPinIf.sv.

Table 12. Description of Assertions in Pin Interface

#	Assertion Title	Description
1	bfmintf_tap_assert_tms_during_posedge_clk	Change detector on posedge of tck
2	bfmintf_tap_assert_tdi_during_posedge_clk	Change detector on posedge of tck
3	bfmintf_tap_assert_tdo_during_posedge_clk	Change detector on posedge of tck when sample_tdo_on_negedge is set to 1
4	bfmintf_tap_assert_tdo_during_negedge_clk	Change detector on negedge of tck when sample_tdo_on_negedge is set to 1
5	bfmintf_tap_assert_check_tms_change_wrt_clk	Check whether tms is asserting with respect to tck
6	bfmintf_tap_assert_check_tdi_change_wrt_clk	Check whether tdi is asserting with respect to tck
7	bfmintf_tap_assert_tdo_1_during_non_shift_state s	Check whether TDO is 1 in non-shift states. Need to compile model with JTAGBFM_TDO_1_ASSERT_ EN to get this assertion

7.5 Extending Coverage

The BFM does not provide any coverage classes. The transactions that get broadcasted on the analysis ports of the monitors in the BFM are used for any coverage collection tasks.

Note: This section lists the functional coverage targets and shows how to extend the Coverage Collector to add or remove System Verilog cover points. It also shows how to ensure that assertion properties described in a previous section were actually covered. You can refer the below listed items:

- Listing the Existing Coverage Targets

- Extending the Coverage Collector
- Interpreting a Coverage Group Report
- Interpreting a Coverage Property Report

8 Agent Packages

Not applicable to this IP/IPSS

9 Agent Parameters

Refer to section 4.1, Configuring the Agents.

10 Agent Interface

All of the variables in the interface are declared as reg.

Table 13. Description of Variable and Parameters in Pin Interface

Variable/Parameter	Default Value	Description
Standard IEEE pins		
Tdi	X	Test Data In.
Tdo	X	Test Data Out.
Tms	X	Test Mode Select.
Tck	X	Test Clock. This is gated based on enable_clk_gating configuration variable. 1: Clock present only when transaction happening. 0: Clock always present.
trst_b	X	Test Reset
Powergood Resets		
PWRGOOD_SRC	0	Parameter to choose the source of Powergood_rst_b reset to be applied to TAP. This needs to be provided when instantiating the interface. 1: Use Powergood_rst_b generated at SoC level. 0: Use BFM Generated Powergood_rst_b
soc_powergood_rst_b	X	This is an input to the interface. At cluster level, if there is a powergood_rst_b then that has to be provided here along with setting PWRGOOD_SRC = 1.
bfm_powergood_rst_b	X	If there is no powergood_rst_b at the SoC level then this bfm_powergood_rst_b generated in this IP should be used by setting PWRGOOD_SRC = 0.
powergood_rst_b	X	Choosen powergood_rst_b that goes to the RTL. This is also consumed in the Tap FSM task that the Driver code maintains.
Clock		
CLK_SRC	0	Parameter for choosing the source of clock generation. 1: Use the clock provided to the BFM. 0: Use the internal clock source within the BFM..
CLOCK_PERIOD	10000	Parameter for the clock period. This needs to be provided when instantiating the interface.
jtagbfm_clk	0	Always running clock source.
BFM_MON_CLK_DIS	0	Parameter for disabling tck generation logic inside the BFM 1: tck generation logic is present in JtagBfmIntf 0: tck generation logic is removed in JtagBfmIntf

Variable/Parameter	Default Value	Description
Other		
tap_tdo_strobe	X	This follows the compare mask argument provided to ExpData_MultipleTapRegisterAccess API.
bfm_shifts_states	X	During Shift IR, Shift DR, this goes high. It is used for certain assertions in the Interface.
exp_tdo	z	During Shift IR, Shift DR, exp_tdo is generated when mask is 1.
RTDR Interface (Added in Version 2.5)		
MAX_NUM_OF_RTDRS = N	50	Local Parameter to define number of RTDRs. This value is fixed.
tap_rtdr_tck	0	Test clock for RTDR interface
tap_rtdr_tdi[N-1:0]	0	Test Data In for RTDR interface
tap_rtdr_tdo[N-1:0]	0	Test Data Out for RTDR interface
tap_rtdr_capture[N-1:0]	0	Capture control signal for RTDR interface
tap_rtdr_shift[N-1:0]	0	Shift control signal for RTDR interface
tap_rtdr_update[N-1:0]	0	Update control signal for RTDR interface
tap_rtdr_irdec[N-1:0]	0	Control signal to indicate this particular TDR's IR was decoded for RTDR interface
tap_rtdr_powergood	0	Powergood reset for RTDR interface
tap_rtdr_selectir	0	Select IR control signal for RTDR interface. 0: Indicates the DR from the TAP FSM. 1: Indicates the IR from the TAP FSM.
tap_rtdr_rti	0	control signal to indicate the value of the Run/Test Idle FSM state. This may be used by the RTDR logic to enable functions once entering into the RTI state for synchronization
tap_rtdr_prog_rst_b[N-1:0]	0	programmable reset pin that allows a software clear, a TSRT_b reset or a powergood reset
Events		
tap_clk_r	NA	Will generate this event at posedge of tck
tap_clk_f	NA	Will generate this event at negedge of tck

11 Configuration Methods

11.1 Configuring the BFM

Table 14. Configuration Variables in BFM Config Object

Variable	Default State	Description
quit_count	20	Controlling after how many error messages should the simulation stop.
Set_verbosity	OVM_DEBUG	Controlling verbosity level.
Enable_clk_gating	1	1: Clock present only when transaction happening. 0: Clock always present.
Park_clk_at	0	When there is no transaction, the clock is parked at the value specified here.
RESET_DELAY	10000 ps	To assert asynchronous resets from clock edge, the BFM uses a random fraction of this number.
Is_active	OVM_ACTIVE	To turn off driver, this can be set to passive
primary_tracker	1	This has to be set to get the active TAP name printed in the tracker file. This names that are on the Primary TAP port will be displayed.
Secondary_tracker	0	This has to be set to get the active TAP name printed in the tracker file. This names that are on the Secondary TAP port will be displayed.
Tertiary_tracker, width = [NUMBER_OF_ TERTIARY_PORTS]	`0	This has to be set to get the active TAP name printed in the tracker file. This names that are on the Tertiary TAP port will be displayed.
Jtag_bfm_tracker_en	1	This variable name replaces the old tracker variables for primary, secondary, tertiary that were present till ver 1.8 providing the same functionality. Tracker file is dumped only in the OVM report phase.
Jtag_bfm_runtime_tracker_en	1	During every UPIR & UPDR states, the tracker file gets updated. When a simulation abruptly ends, the log till that time is preserved in the tracker file. This is a separate file than the above.
Tracker_name	"STAP"	This can be any string that will form the file name of the tracker.
Sample_tdo_on_negedge	0	Set this configuration variable as 1 to sample tdo at negedge of tck. For RTDR mode, sample_tdo_on_negedge = 1 --> BFM will sample tdo on posedge. For RTDR mode, sample_tdo_on_negedge = 0 --> BFM will sample tdo on nedgede. For JTAG mode, sample_tdo_on_negedge = 1 --> BFM will sample tdo on negedge. For JTAG mode, sample_tdo_on_negedge = 0 --> BFM will sample tdo on posedge.

Variable	Default State	Description
Added in Version 2.5		
use_rtdr_interfae	0	JTAGFM will be used as RTDR BFM when this variable is set as 1.
Rtdr_is_bussed	0	RTDR Interface will be bussed when this variable is set to 1.
Added in Version 3.4		
config_trstb_value_en	0	Used to enable the programming of trst_b in JtagBfmPinIf.

config_trstb_value	0	This value applies to trst_b in JtagBfmPinIf when config_trstb_value_en is 1.
		set_config_int("<Object of JtagMasterAgent>", "config_trstb_value_en", 1);
		set_config_int("<Object of JtagMasterAgent>", "config_trstb_value", 0);

override_tdo_data	0	Used to flush the internal register value with `0 for every UPIR/UPDR.
override_tdi_data	0	Used to flush the internal register value with `0 for every UPIR/UPDR.
Added in Version 3.6		
unsplit_ir_dr_data	0	Used to split data between every pause IR/DR. unsplit_ir_dr_data = 1/0 concatenates/splits the data.

These variables are available in a configuration class that needs to be instantiated in the environment class. In order to use the configuration class, you can create a new class that extends this and then override the values in the new class and then use it in the Env where the JTAGBFM will be instantiated.

12 Transaction Sequence Item

This section describes the transaction classes/OVM Sequence Items and tasks available to program the Agent.

12.1 Sequencer to Driver Transaction Class

This is the transaction class between Sequencer and Driver. The test case writer uses this to inject the stimulus into the DUT.Members of JtagBfmSeqDrvPkt

The randomized variable names of this class are listed below.

Table 15. Members of JtagBfmSeqDrvPkt.sv

Variable Name and Hierarchy	Type	Description
Address	Bit vector	The address or instruction Opcode.
Data	Bit vector	The data that needs to be loaded in the selected register
ResetMode	Bit vector	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
addr_len	Bit vector	The address width.
data_len	Bit vector	The data width
pause_len	Bit vector	The number of cycles to remain in PADR.
TMS_Stream	Bit vector	This is the stream of TMS to help the user navigate the FSM in desired state.
TDI_Stream	Bit vector	This stream of TDI is driven in parallel to the TMS stream.
Expected_Data	Bit vector	The expected data from the selected IR.
Mask_Address	Bit vector	The exact bits that you want to compare between Address and Expected_Address.
Mask_Data	Bit vector	The exact bits that you want to compare between Data and Expected_Data
Mask_Capture	Bit vector	This field is used to display bits in log file for Returnedtdo. This field has Width as same as Mask_Data. 1 : Display in log 0 : Will not display in log.
actual_tdo_collected	Bit vector	TDO that comes back.

12.1.1 Parameters Set By the Slave

Not applicable to this IP/IPSS

12.1.2 Methods

Not applicable to this IP/IPSS

12.1.3 Constraints

Not applicable to this IP/IPSS

12.2 Monitor Transaction Classes

These are the transaction classes used by the Input and Output monitors to advertise activity on analysis ports. A Tracker, Coverage Collector, or a Scoreboard can subscribe to this analysis port to get meaningful transactions.

JtagBfmInputMonitor.sv is the input Monitor for the DUT. It Monitors the pin that drive the DUT and sends the relevant information to the Scoreboard. The Monitor has the FSM State Machine to replicate the behavior of RTL.

JtagBfmInMonSbrPkt can be used as packet between the Input monitor and Scoreboard. It contains the defination for all the fields that needs to be passed from the Input monitor to scoreboard. The feilds are: FSM STATE, ADDRESS, DATA, VERCODE, IDCODE, Parallel data in and the Power Good Reset.

JtagBfmOutputMonitor.sv is the output Monitor for the DUT. It monitors output of the DUT and sends the relevent information to the Scoreboard. The Monitor has the FSM State Machine to replicate the behaviour of RTL.

JtagBfmOutMonSbrPkt can be used as the packet between the Output monitor and Scoreboard. It contains the defination for all the fields that needs to be passed from the Output monitor to scoreboard. The feilds are: Parallel Data Out, TDO Data Register, TDO Address Register.

13 Tracker

The BFM now provides an enhanced tracker file output that contains the value right after any of the update states namely UPIR & UPDR. The width of the display is set to 128 as default bits wide in the tracker. It can be widened with a new `define CHASSIS_JTAGBFM_TRACKER_WIDTH added. A later version of the BFM will enhance this to be provided as user input thru the configuration descriptor object.

Figure 85. Code 43. Standard Tracker Contents Describing Value in UPDR and UPIR States with Timestamp

```
CHASSIS JTAG BFM Tracker ver 1.9
File Name:- STAP_stap_JtagMasterAgent_jtag_tracker_normal.out
-------------------------------------------------------------------------------------------------------------------------------
        TIME      STATE   No_of_Shifts              TDI_ShiftedIn                  TDO_ShiftedOut
-------------------------------------------------------------------------------------------------------------------------------
        1535000    UPDR        4          00000000000000000000000000000000     0000000000000000000000000000000f
        1835000    UPDR        8          00000000000000000000000000000000     00000000000000000000000000000087
        2175000    UPIR        8          000000000000000000000000000000ff     xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx01
        2555000    UPDR       32          00000000000000000000000ffffffff     00000000000000000000000ffffffe
```

Figure 86. Code 44. Contents of Enhanced Runtime Rracker

```
-------------------------------------------------------------------------------------------------------------------------------
CHASSIS JTAG BFM Runtime Tracker ver 2.0
File Name:- SOC_TAPNW_PRI_PriMasterAgent_jtag_tracker_runtime.out

-------------------------------------------------------------------------------------------------------------------------------
IRWidth Opcode TapName            TIME      STATE    No_of_Shifts    TDI_ShiftedIn          TDO_ShiftedOut
-------------------------------------------------------------------------------------------------------------------------------
 8        12    CLTAP

                                 225000      UPIR        8            00000012          xxxxxx01    DWord  [01/04]
                                                                      00000000          xxxxxxxx    DWord  [02/04]
                                                                      00000000          xxxxxxxx    DWord  [03/04]
                                                                      00000000          xxxxxxxx    DWord  [04/04]

                                 405000      UPDR        12           00000004          00000000    DWord  [01/04]
                                                                      00000000          00000000    DWord  [02/04]
                                                                      00000000          00000000    DWord  [03/04]
                                                                      00000000          00000000    DWord  [04/04]

-------------------------------------------------------------------------------------------------------------------------------
IRWidth Opcode TapName            TIME      STATE    No_of_Shifts    TDI_ShiftedIn          TDO_ShiftedOut
-------------------------------------------------------------------------------------------------------------------------------
 8        ff    CLTAP
 8        11    STAP1

                                 635000      UPIR        16           0000ff11          xxxx0101    DWord  [01/04]
                                                                      00000000          xxxxxxxx    DWord  [02/04]
                                                                      00000000          xxxxxxxx    DWord  [03/04]
                                                                      00000000          xxxxxxxx    DWord  [04/04]

                                 745000      UPDR        5            00000004          00000000    DWord  [01/04]
                                                                      00000000          00000000    DWord  [02/04]
                                                                      00000000          00000000    DWord  [03/04]
                                                                      00000000          00000000    DWord  [04/04]

-------------------------------------------------------------------------------------------------------------------------------
IRWidth Opcode TapName            TIME      STATE    No_of_Shifts    TDI_ShiftedIn          TDO_ShiftedOut
-------------------------------------------------------------------------------------------------------------------------------
 8        ff    CLTAP
 8        ff    STAP1
 8        11    STAP8

                                1055000      UPIR        24           00ffff11          xx010101    DWord  [01/04]
                                                                      00000000          xxxxxxxx    DWord  [02/04]
                                                                      00000000          xxxxxxxx    DWord  [03/04]
                                                                      00000000          xxxxxxxx    DWord  [04/04]

                                1235000      UPDR        12           00000100          00000000    DWord  [01/04]
                                                                      00000000          00000000    DWord  [02/04]
                                                                      00000000          00000000    DWord  [03/04]
                                                                      00000000          00000000    DWord  [04/04]
```

The names of the trackers are in the following format.

- <tracker_name>_<instance_name_of_JtagBfm>_jtag_tracker.out
- <tracker_name>_<instance_name_of_JtagBfm>_jtag_tracker_runtime.out

The files will be located at:
`$IP_ROOT/pwa/results/tests/<test_name>/*_jtag_tracker*.out`

14 File Descriptions

Here is a short description of the Agent files. As a general rule, each class has its own file and each file is named the same as the class it contains.

Table 16. File List

File Name	Description
JtagBfmMasterAgent.sv	It instantiates Driver, Sequencer, Input/Output Monitor, and Tracker.
JtagBfmCfg.svh	Various controls to configure BFM are bundled in this class.
JtagBfmDriver.sv	This Block Drives the DUT through the pin interface. It decodes the Sequencer tasks and drives the DUT accordingly.
JtagBfmInMonSbrPkt.sv	Packet between the Input monitor and Scoreboard
JtagBfmInputMonitor.sv	Input Monitor for the DUT
JtagBfmOutMonSbrPkt.sv	Packet between the Output monitor and Scoreboard
JtagBfmOutputMonitor.sv	This has two analysis ports where it advertises reset conditions and JTAG bfm output packets.
JtagBfmPinIf.sv	This has all the wires that the driver drives. The DUT is connected to these wires of the interface.
JtagBfmSeqDrvPkt.sv	Packet Between the Sequencer and the Driver
JtagBfmSequencer.sv	This conveys the transaction from Test case to Driver.
JtagBfmTestIsland.sv	This contains the setting up of the interface container.
JtagBfmTracker.sv	This prints the tracker file.
JtagBfmPkg.sv	This holds all the above files.

15 Package Release Procedure

This agent package is released on IRR after undergoing IPDS checks.

Once downloaded, the directory structure will look like below.

> tree -L 2

```
|-- README.txt
|-- ace
|   |-- ace_test.mak
|   |-- clean_up.cfg
|   |-- clean_up_none.cfg
|   |-- jtagbfm.acerc
|   |-- jtagbfm.udf
|   |-- jtagbfm_hdl.udf
|   |-- jtagbfm_integ.udf
|   |-- jtagbfm_local_ivars.udf
|   |-- jtagbfm_postsim.pp
|   `-- lib
|-- cfg
|   `-- ToolData.pm
|-- doc
|   |-- Chassis_JTAGBFM_Rev3.9_Integration_Guide.pdf
|   |-- DFx_JTAG_BFM_PIC_Release_Note.html
|   |-- DFx_JTAG_BFM_PIC_irrupload.xml
|   |-- Release_Note.html
|   |-- SIP_Chassis_JTAGBFM_DTTC_2013.pdf
|   |-- ip_release.2020_02_23_23:15.log
|   |-- ip_release.log
|   |-- ip_release.user_config.xml
|   |-- pbj.txt
|   `-- toolver.xml
|-- scripts
|   |-- SFM_readme.txt
|   |-- aes_c_enc_dec_wrapper.c
|   |-- aes_c_enc_dec_wrapper.o
|   |-- command_help.txt
|   |-- converged_filelist.f
|   |-- custom_post.setup
|   |-- custom_pre.setup
|   |-- default_config.dat
|   |-- filelist.f
|   |-- gen_aes_c_obj.sh
|   |-- intel_ovm_to_uvm.pl
|   |-- name_changes.txt
|   |-- parse_covfile.pl
|   |-- qa
|   |-- rc
|   |-- run_SFM.sh
|   |-- run_all_configs
|   |-- run_all_configs2
|   |-- run_all_hier
|   |-- run_all_hier_hybrid
|   |-- run_all_hier_nb
|   |-- run_all_linear
|   |-- run_all_test_1S
|   |-- run_all_test_1S_nb
|   |-- run_all_test_2S
|   |-- run_all_test_default
|   |-- setup
|   |-- substitute.sh
|   |-- toolfile.All.dat
|   |-- uvm_dpi.so
|   `-- variation_config.dat
|-- subIP
|   |-- ip-master-tap
|   |-- ip-stap
```

```
|   `-- ip-tap-network
|-- tools
|   |-- ctt
|   |-- reports
|
`-- verif
    |-- lib
    |-- scripts
```

`-- tb

Note: ITPP parser is removed from SubIP from the current version. User has to refer to itpp parser from the central HDK location.

16 Compliance Checklist

This IP confirms to the below checklists:

- OVM System Verilog Compliance Checklist
- UVM System Verilog Compliance Checklist
- Saola Checklist
- Test Island Checklist
- IPDS compliance
- Zircon rule checks

17 Compatibility with Other BFMs

This BFM is compliant to OVM/UVM and Saola methodology. It is also VCS2017 compatible. Hence, it should be compatible with other BFMs.

18 Open Issues and To Do List

18.1 Opens List

1. After placing a TAP on secondary or on Tertiary, it has not been validated using the Tap Aware capabilities to place it back on primary.

2. Network mode Excluded is not been implemented and tested. Expectation is to use the TapAccess API for this as well.

3. Minimal testing on Shadow mode has been completed.