DTEG DUVE-M

USER GUIDE

Tool Rev. 1.0
January 2021

Contents

List of Figures

List of Tables

1 Introduction

1.1 Audience

The document is targeting IP/SoC designers and validators who need to develop and validate ICL/PDL (IEEE 1687) collateral for their projects.

1.2 Purpose

The User Guide describes capabilities and usage of DTEG DUVE-M (DTEG Universal Validation Environment (Mentor)) infrastructure. DUVE-M is a validation IP for verification of IP/SOC ICL/PDL vs. RTL. In addition, the tool allows extraction of architecture and design information from the provided ICL/PDL and uses it for implementing project-specific checks, tests, or collateral generation. The infrastructure also includes TAP RDL2ICL flow which allows to generate ICL from the existing TAP RDL.

1.3 Scope

The document covers input collateral requirements, tool configuration, main use models, and included tests, sequences, and procedures.

1.4 Terminology

The table below defines the terms used in this document.

Term	Definition
Access Type	Access type of register bits. At Intel, for TAP (TAP RDL), the following types are used: ▪ RW (read-write) ▪ WO (write-only) ▪ RO (read-only) ▪ RW/V (RW volatile) ▪ RO/V (RO volatile) Register or bits, used for control purposes, should be one of the writable types (RW, RW/V, WO). Status registers/bits should have RO/V type. Registers, like IDCODE or SLVIDCODE, which allow to read some internal constants (permanent identifiers), should have RO type. Finally, if a value of read-write register can be changed by some other independent process/master/logic, rather than TAP access, it must have RW/V type. Read values are predictable only for RW and RO registers/bits - only those can be covered by auto-checks of returned TDO. Other register/bit types require expected values from the test writer to check returned TDO data.
BFM	Bus Functional Model - verification component for generating and monitoring/checking transaction level traffic for the given interface/protocol. In the context of this document, it is applicable to JTAG and IJTAG TAP protocols.
Client ScanInterface	Interface/port bundle which is controlled by the external TAP logic. Examples: chip level CLTAP JTAG interface, interface of RTDR register. The client-TAP ScanInterface usually includes TDI, TDO, TMS ports. The client-scan (RTDR) ScanInterface usually includes si, so, select and capture/shift/update signals.
DFT	Design For Test
DFx	Design For *, which includes Test (DFT), Debug (DFD), Manufacturing (DFM), etc.
DR	Same as TDR
DTEG	Design for Test Engineering Group

Term	Definition
DUT	Device Under Test (in the given context) - RTL design, used to generate patterns and then instantiated in the generated Verilog Test Bench for simulation of these patterns.
DUVE	DTEG Unified Validation Environment. It is based on SVTB/OVM/UVM.
DUVE-M	DTEG Unified Validation Environment (Mentor) - it is based on ICL/PDL.
Host ScanInterface	Interface/port bundle which is controlled by the internal TAP logic. Examples: CLTAP interface to the child TAP, TAP interface to connect external RTDR register. The host-TAP ScanInterface usually includes TDI, TDO, TMS (ToTMSPort port in ICL) ports. The host-scan (RTDR) ScanInterface usually includes si, so, select (ToSelectPort port in ICL) and capture/shift/update (ToCaptureEnPort/ToShiftEnPort/ToUpdateEnPort ports in ICL) signals.
HTAP	Hierarchical TAP network - Intel-specific implementation of hierarchical TAP network, aligned with IEEE1149.7 T0. It is slightly different than Intel legacy DFx Chassis 2.0 hierarchical TAP network.
ICL	Instrument Connectivity Language - part of IEEE 1687 which defines how to describe test instrument registers and how to access them
ICL module	The container for a collection of primitive building blocks and possibly instances of other modules that comprise a network and/or instrument
ICL register	ICL allows specifying of instrument serial registers (TAP/IJTAG TDR) using ScanRegister primitives and instrument parallel data registers using DataRegister primitive.
IEEE 1149.1	IEEE Standard Test Access Port and Boundary-Scan Architecture
IEEE 1687	IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device. Current standard version is 2014.
IJTAG	Internal JTAG - TAP fabric architecture, defined as a part of IEEE 1687. IJTAG uses Segment Insertion Bits (SIB) to control inclusion of instrument register or next level of SIB hierarchy into the TDI-TDO (SI-SO) chain.
IPX	Intel IP eXchange system for distributing IP inside the company. Website: goto/ipx
IR	TAP Instruction Register
JTAG	Industry standard for verifying designs and testing printed circuit boards after manufacture based on TAP (name is based on Joint Test Action Group which created it). It became the IEEE 1149.1 standard.
Nebulon	Intel tool to process RDL specifications for Control Registers, Fuses, and TAP
OVM	Open Verification Methodology - standardized methodology, layer and components for verifying semiconductor designs. It was a base for UVM.
PDL	Procedural Description Language - part of IEEE which defines procedures to use test instruments
RDL	Register Description Language - same as SystemRDL
RTDR	Remote TDR - a TDR that is located outside of TAP module
Saola TAL	Saola Tap Access Layer - Intel validation infrastructure for validation of design registers, including TAP (TAL layer). It is based on SVTB/OVM/UVM.
SI	Serial In - same as TDI
SIB	Segment Insertion Bit - control element of IJTAG chain
SO	Serial Out - same as TDO
SPF	Structural Pattern Framework - Intel infrastructure for development of test and validation content for DFT (including TAP)
SVF	Serial Vector Format - Industry Standard specification, used for definition of TAP and Parallel IO sequences (owned by ASSET InterTech, Inc.)
SVTB	System Verilog Test Bench - IEEE 1800 standard, defining hardware description and hardware verification language used to specify, develop, model, simulate and test semiconductor designs

Term	Definition
SystemRDL	The standard language, which is designed to describe and implement a wide variety of registers and memories. At Intel, it was used (and is still in use) for specification of Control Registers, Fuses, and TAP registers. Intel uses version 1.0 of the specification.
TAP	Test Access Port - the device interface pins and controller logic, defined in IEEE Std 1149.1. The interface contains file pins: TDI, TMS, TCK, TDO, and optionally TRST.
TapLink	Intel TAP network architecture used by Client and Server products. In general, it follows IEEE 1149.1 protocol. The architecture is based on broadcast and different than conventional serial or serial-hierarchical TAP network implementations.
Tcl	High-level, general-purpose, interpreted, dynamic programming language. It is used in many EDA tools to implement command interface. It is a base of Tessent Shell command interface.
TDI	Test Data In - Serial data in device interface pin defined in IEEE Std 1149.1
TDO	Test Data Out - Serial data out device interface pin defined in IEEE Std 1149.1
TDR	Test Data Register - TAP Data register (test instrument)
TLR	Test Logic Reset - state of TAP FSM, corresponding to TAP reset. TAP can be enforced to the reset state by asserting TRST (TRSTb) port or by advancing TAP FSM into the TLR state by the TMS sequence (TMS='1' during five consecutive TCLK cycles).
TRST	Test ReSeT - TAP reset pin or event
UDP	User-Defined Property - custom (non-standard) property to define some characteristic of the registers in SystemRDL specification
USC	UltiScan Controller - DTEG Scan Controller IP
UVM	Universal Verification Methodology - standardized methodology, layer, and components for verifying semiconductor designs. It is an evolution of the OVM.
VC	Verification Component - BFM, Scoreboard, etc.
Verilog TB	Verilog Test Bench
VIP	Validation IP - similar to VC

1.5 Related Documents

If you need more information on this tool and collateral, you may find these documents helpful.

Document Title	Location
IEEE Std 1687-2014	IEEE Xplore (available through Intel Library online)
SystemRDL v1.0	Accellera website (https://www.accellera.org/)
Tessent® Shell Reference Manual	Siemens (Mentor Graphics) website (https://support.sw.siemens.com/)
Tessent IJTAG User's Manual	Siemens (Mentor Graphics) website (https://support.sw.siemens.com/)
Serial Vector Format Specification	Asset website (https://www.asset-intertech.com/)
Intel IPx wiki	goto/ipxwiki
Release Notes	Release ‘doc’ directory/release_notes.txt

1.6 Contact Information

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

Function	Name	Email
Tool Architect	Igor Molchanov	igor.v.molchanov@intel.com
Tool Programmer	Igor Molchanov	igor.v.molchanov@intel.com
Tool Validation	Igor Molchanov	igor.v.molchanov@intel.com
Spec Template Owner	Susann Flowers	susann.flowers@intel.com

1.7 Document Revision History

Revision Number	Description of Change	Date	Revised By
0.1	Initial Release	09/2020	
0.2	Update for DUVE-M release 0.8	12/2020	
0.3	Update for DUVE-M release 1.0	01/2021	

2 Overview

2.1 Why ICL/PDL?

IEEE1687 is the industry standard which defines hardware architecture and methodology for accessing of embedded instrumentation via IEEE 1149.1 Test Access Port (TAP).

The standard includes three components:

- Architecture: IJTAG (internal JTAG)
- Design spec: ICL (Instrument Connectivity Language)
- Procedures: PDL (Procedural Description Language

There are no restrictions on the TAP network implementation if it follows IEEE1149.1 protocol. That is why ICL and PDL can be used for Intel HTAP & TapLink TAP fabrics.

Advantages of IEEE 1687 ICL/PDL

- IEEE1687 ICL/PDL is the industry standard and it is supported by EDA tools
    - Good support of the tools & methodology, quickly evolving and improving capabilities
- Native retargeting support of IP level sequences and recipes at SoC level
    - Allows sequences to be defined, validated, and delivered by IP owner
- Native support of variable length registers/chains (IOV, BSCAN, IJTAG, etc.)
- Ease of integration and collateral reuse for 3rd party IP’s
- Support of “not usual” TAP implementations and mixed networks
- Ease of IP TAP/DFT logic and ICL/PDL collateral validation
- PDL provides similar test writing capabilities and experience as Intel SPF or Saola TAL
- ICL data modeling in the EDA tools and extendable set of APIs to introspect design/ICL/test metadata
    - Ease of development and customization for Intel TFM needs
- New capabilities, for example supporting of clock specification and clock validation

2.2 DUVE-M

DTEG DUVE-M is a generic infrastructure and tests for DFx/TAP validation and test content development based on IEEE1687 ICL/PDL. The tool requires Tessent Shell (a tool from Siemens/Mentor Graphics) to process input collateral and generate output validation or test patterns.

Because it uses ICL/PDL, DUVE-M is TAP architecture and design implementation agnostic. It works with any IEEE 1687/Tessent compliant ICL/PDL. Input ICL specification can be created manually, or it can be received from IP provider or it can be generated by some tool (for example, by DTEG DFT builder tool, or by Tessent). The infrastructure includes TAP RDL2ICL flow which allows generation of IP ICL from the existing TAP RDL.

Generated patterns feature auto-checking of readable TAP TDR bits (of RW and RO access types). For validation purposes, patterns can be saved in Verilog Test Bench (Verilog TB) format or in SVF (Serial Vector Format) and then be simulated in the project validation environment.

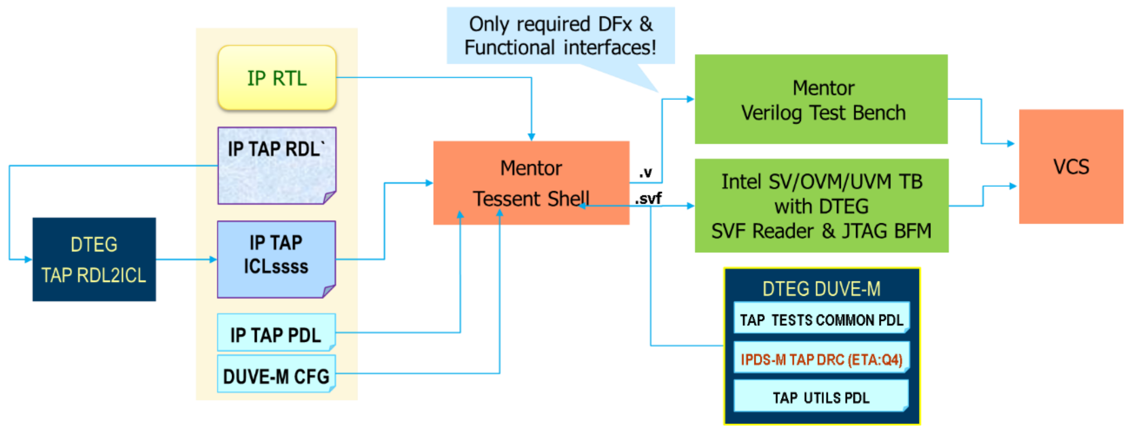
Note: SVF format requires special SVF reader, which is available from DTEG as a standalone VC or as a part of DUVE validation infrastructure.

The tool is flexible – it can be tuned to match design configuration and validation scope. It is achieved by using local configuration files per DUT/targeted scope/initialization preamble.

The tool supports Intel-specific Reset and Security architectures and register bit Access Types.

Typical use of DUVE-M for IP level validation is shown in Figure 1.

Figure 1. Typical Use of DUVE-M for IP Validation



3 Getting Started

3.1 Prerequisites

DUVE-M requires Tessent Shell software from Siemens/Mentor Graphics to generate verification patterns.

Note: Always use the latest available Tessent version!

For pattern simulation, any available Verilog simulator can be used (for example, VCS from Synopsys).

Tessent command interface is based on Tcl. Writing of generic tests, which use design, ICL, or PDL metadata, requires use of Tcl, too. It is recommended that the DUVE-M user is familiar with that programming language.

3.2 First-time Setup

DUVE-M is available for downloading from Intel IPX IP distribution infrastructure.

The tool relies on a few environment variables:

- $DUVE_M_HOME : a pointer to the released DUVE-M repo
- $ IP_ROOT : IP_ROOT of current IP/SOC (a base pointer for all collateral)
- $ IP_TESSENT_CFG : a pointer to the main DUVE-M IP configuration file (ip_cfg.do)

To setup the infrastructure, a set of configuration files should be copied to the project repo and updated to reflect design configuration and validation scope.

The recommended file structure to store collateral in the design repo (similar to the one used in the DUVE-M repo):

$IP_ROOT/verif/cfg	: for DUVE-M configuration files
$IP_ROOT/verif/flow	: for optional, IP specific, customized flow dofiles
$IP_ROOT/verif/pdl	: for new, IP specific, PDL content

The details about customization of configuration and flow files will be provided in 4.1 Defining Design and Test Configurations.

4 DUVE-M Use Flow

4.1 Defining Design and Test Configurations

Description of available configuration files is provided in the next sections.

Note: Configuration file names can be changed as required - the corresponding mapping is a part of main configuration file (with default name ip_cfg.do). Pointer to the main configuration file is specified through $IP_TESSENT_CFG environment variable.

4.1.1 ip_cfg.do

The file contains pointers to all DUVE-M configuration and flow/PDL files.

Note: You can create local custom versions of DUVE-M files and reference them in the main configuration file.

It also includes pointers to:

- Folder to write generated patterns to
- User-defined PDL test(s)
- Tessent log file

The file specifies the following parameters:

- Top design name
  - parameter: $current_design
    - value: string
- Tessent context to load design RTL
  - parameter: $tessent_context_to_read_verilog
    - supported values: dft_rtl | patterns_ijtag
- Verilog type of design (System Verilog vs. Verilog)
  - parameter: $sv_mode
    - supported values: 0 (default) | 1 (for System Verilog)

4.1.2 ip_read_icl.do

The file specifies how to load all required ICL collateral of the project.

4.1.3 ip_read_verilog.do

The file specifies how to load all required Verilog collateral of the project.

Note: Pattern generation requires only TOP level RTL with interfaces (and included files if any).

4.1.4 ip_current_design_cfg.do

The file can be used to enable generation of additional functional or DFT clocks (on top of the available by default TAP clock) or to add other similar custom configuration data which should be applied after 'set_current_design' and before 'check_design_rules' commands of the Tessent Shell processing flow.

4.1.5 ip_pattern_cfg.do

The file specifies parameters and scope for generated patterns. It covers:

- Base name of generated patterns
    - parameter: $pattern_set
        - value: string
- Enabling of generation separate patterns per test group (groups: continuity, reset, read-write access, opcodes/security)
    - parameter: $separate_test_per_group
        - supported values: 0 (default) | 1
- Enabling/disabling of individual tests ('1' enables the test, '0' disables it).
    - parameter: $enable_continuity_tests (default: 1)
    - parameter: $enable_powergood_reset_tests (default: 1)
    - parameter: $enable_trst_reset_tests (default: 0)
    - parameter: $enable_tlr_reset_tests (default: 0)
    - parameter: $enable_rw_tests (default: 1)
    - parameter: $enable_rw_dr_field_tests (default: 0)
    - parameter: $enable_all_opcodes_security_tests (default: 1)
    - parameter: $enable_user_tests (default: 0)
- Configuring covered security levels for security tests
    - parameter: $enable_security_tests_all_levels (default: 0)
        - By default, security tests are performed for GREEN, RED, RED4, ORANGE Security Policies (levels 0,2,4,5 correspondently). Setting $enable_security_tests_all_levels to 1 will enable execution of the tests for all other levels, not excluded using parameter $excluded_security_levels.
    - parameter: $excluded_security_levels (default: {})
        - supported values: Tcl list of numeric values, for example {7 9}
- Tester and TAP clock parameters for Tessent open_pattern_set command and retargeting options
    - parameter: $tester_period (default: 10ns)
    - parameter: $tck_ratio (default: 1)
    - parameter: $tck_period (default: 10ns)

Note: Make sure that the following is true: tck_period = tester_period * tck_ratio

4.1.6 ip_test_cfg.do

The file specifies test configuration parameters, target design scope, and design-specific PDL procedures.

4.1.6.1 Defining Target Validation Scope

User has a complete control over the scope of the registers which verification tests will target. He/she can selectively add and remove specific instances or modules from testing, using explicit paths/names or wildcards.

To include/exclude all instances of the specific register(s), module-based inclusion/exclusion should be used. The corresponding parameters have 'module' as part of the name.

To include/exclude specific instances of modules or registers, use per-instance inclusion/exclusion. The corresponding parameters have 'registers' or 'instances' in the name.

Note: Tessent can uniquify parameterized modules when necessary. Consider this and add '*' in the end of the matching expression for such parameterized modules. For example, use intel_tap_ir* for parameterized module intel_tap_ir which represents TAP IR.

To use generic tests, which are provided with DUVE-M, exclusion of registers with some special behaviors may be required. In some cases, behavior, which is impacting ICL modeling, cannot be completely described in ICL or it is just not accurately modeled by Tessent. So, it can result in TDO mismatches during simulation.

An example of such special case is a register, asserting reset for some TDR(s) when IR is loaded with the corresponding opcode.

Another usual registers to exclude are TAP network and IJTAG chain control (TAPSELECT and SIB registers). These registers will be indirectly validated when tests exercise targeted TAPs or registers.

Note: Some Tessent commands have similar exclusion/inclusion options (for example, create_icl_verification_patterns). These options can work slightly differently and can have different requirements for implementing name matching patterns with wildcards.

Base scope of the registers to target

- parameter: $included_instances
  - value: Tcl list of module instances in the current design (wildcards are supported)
  - value examples:

    (1) {*} - included full scope of the registers (default)

    (2) {par1_dfx_mem_wrapper} - include only registers inside par1_dfx_mem_wrapper instance
- parameter: $included_registers
  - value: Tcl list of register instances (wildcards are supported)
  - value examples:

    (1) {} - no additional register instances are included (default)

Scope exclusion parameters

- parameter: $excluded_modules (module-based exclusions)
  - value: Tcl list of modules (wildcards are supported)
  - value examples (Intel-specific, should be aligned with the content of the design):

    (1) {*intel_tap_ir* *intel_bypass_rsvd_reg} - excluding TAP IR and BYPASS_RSVD registers (Intel-specific)

    (2) {*intel_htap_sel …} - additional exclusion of HTAP TAPSELECT registers (Intel-specific)

(3) {ijtag_sib sib …} - additional exclusions of SIB modules (Intel-specific)

Note: Never use wildcards like *sib* - it can accidently exclude from the validation many components which have that specified short string ('sib') in the module definition name!

(4) {*iov_ctrl* … } - additional exclusion of IOV controller logic (Intel-specific)

(5) {*__SCANDUMP_CHAIN …} - additional exclusion of DTEG USC controller scan dump chains

(6) {… *dfx_mem_wrapper* …} - additional exclusion of MBIST wrappers (Intel-specific)

- parameter: $excluded_registers (instance-based exclusions)
  - value: Tcl list of instances (wildcards are supported)
  - value examples (Intel-specific, should be aligned with the content of the design):

    (1) {… *IOVRESET* …} - exclusion of the IOVRESET* instances

    (2) {… *fake_data_bus_reg* …} - exclusion of the fake (ICL-only) register in SSN

    (3) {… *streaming_through_ijtag_en* …} - exclusion of the register with special behavior in SSN

Note: If some component is excluded from the validation scope, make sure to cover it by dedicated directed tests!

Note: Use sets of main/test/pattern configuration files with different inclusion and exclusion settings to split IP validation scope into the smaller sub-scopes. For example, it can be Intel-specific scope, MBIST scope, SSN scope, etc.

4.1.6.2 Resource Mapping

This group of parameters allows to identify registers of the specific types in the design. It is done based on user-provided paths/names/wildcards. Both module-based and instance-based mapping modes are supported.

Mapping parameters

- parameter: $ir_module_names (module-based mapping)
  - Used to find TAP IR registers in the given design based on provided patterns for module names
  - value: Tcl list of modules (wildcards are supported)
  - value examples (Intel-specific):

    (1) {*intel_tap_ir*} - pattern to find IR registers (Intel-specific, the default value)
- parameter: $ir_reg_names (instance-based mapping)
  - Similar to ir_module_names above but based on instance path/name patterns
  - value: Tcl list of instances (wildcards are supported)
- parameter: $bypass_module_names (module-based mapping)
  - Used to find TAP BYPASS registers in the given design based on provided patterns for module names (not used by included DUVE-M tests at the moment)
  - value: Tcl list of modules (wildcards are supported)
  - value examples (Intel-specific):

    (1) {*intel_bypass_reg}

- parameter: $bypass_reg_names (instance-based mapping)
  - Similar to bypass_module_names above but based on instance path/name patterns
  - value: Tcl list of instances (wildcards are supported)
- parameter: $network_control_modules (module-based mapping)
  - Used to map TAP network/chain control registers. This can include HTAP TAPSELECT registers, IJTAG SIB registers and any other registers which control inclusion/exclusion of TDR instruments in to/from TDI-TDO chain. The registers, scoped with that parameter, are auto-excluded from DUVE-M TLR/TRST reset tests.
  - value: Tcl list of modules (wildcards are supported)
  - value examples (Intel-specific):

    (1) {*intel_htap_sel ijtag_sib sib ip74xodit05_tctrl ip74xvdmt05_tctrl} - mapping of HTAP TAPSELECT, IJTAG SIB, ODI and VDM control (Intel-specific)
- parameter: $network_control_registers (instance-based mapping)
  - Similar to network_control_modules above but based on instance path/name patterns
  - value: Tcl list of modules (wildcards are supported)

4.1.6.3 Test-specific Configuration Parameters

Continuity Test/All Tests

- parameter: $continuity_test_only_registers (instance-based exclusions)
  - The parameter keeps specified registers for the continuity test but excludes them from all other tests. In certain situations, update part of the TDR can be not functioning (for example, if it depends on the functional clock which is not available in the given implementation of the Test Bench). Because of shift part of the register is still expected to be working then it can be exercised for continuity.
  - value: Tcl list of instances (wildcards are supported)
  - value examples (Intel-specific, should be aligned with the content of the design):

    (1) {*BRKPTEN* *BRKPTCTL* *DEBUGCOUNTER?_*} - exclusion of the registers which require a functional clock, not available in the given Test Bench (Intel-specific)

TRST/TLR Tests

- parameter: $trst_tlr_test_excluded_modules (module-based exclusions)
  - The parameter excludes registers in the specified modules from the DUVE-M TRST/TLR reset tests.
  - value: Tcl list of modules (wildcards are supported)
- parameter: $trst_tlr_test_excluded_registers (instance-based exclusions)
  - The parameter excludes specified instances (modules or registers) from the DUVE-M TRST/TLR reset tests.
  - value: Tcl list of instances (wildcards are supported)

        - value examples (Intel-specific):

          (1) {*IOVCONFIG*} - exclusion of IOVCONFIG register which reconfigures TDO path of the IOV chain (Intel-specific)

The next parameters allow specifying the register scopes, which are expected to be reset by TRST and TLR types of resets. Both instance-based and module-based specification types are supported. This allows to partially overcome IEEE 1687 and Tessent limitations with modeling of the non-TAP resets.

DUVE-M reset tests can use the provided data instead of data in the ICL.

- parameter: $dr_modules_trst (module-based exclusion)
    - Used to specify modules with TDR registers which are reset by TRST (excluding network/chain control registers)
    - values: Tcl list of modules (wildcards are supported), default: {}
- parameter: $dr_modules_tlr (module-based exclusion)
    - Used to specify modules with TDR registers which are reset by TLR (excluding network/chain control registers)
    - values: Tcl list of modules (wildcards are supported), default: {}
- parameter: $dr_registers_trst (instance -based exclusion)
    - Used to specify TDR registers (or module instances with registers) which are reset by TRST (excluding network/chain control registers)
    - values: Tcl list of instances (wildcards are supported), default: {}
- parameter: $dr_registers_tlr (instance -based exclusion)
    - Used to specify TDR registers (or module instances with registers) which are reset by TLR (excluding network/chain control registers)
    - values: Tcl list of modules (wildcards are supported), default: {}

All Opcodes/Security Tests

- parameter: $tap_opcodes_tap_instances (instance-based selection)
    - Used to specify TAP instances to run the opcode/security tests for
    - values: Tcl list of TAP module instances (wildcards are supported)
    - value examples:

      (1) {*} - run opcode/security tests for all TAPs in the design

      (2) {htap2} - run opcode/security tests for TAP instance htap2
- parameter: tap_opcodes_to_exclude (type: Tcl array, instance/opcode-based exclusion)
    - Used to specify IR opcode values to exclude when running opcode/security tests for the specific TAP instance. The parameter can be used to exclude opcodes with some special behavior, not modeled properly in ICL or by Tessent (for example, it can be due to specific implementation of the IOVRESET).
    - value: combination of '<instance:opcode>' string and register instance name. For IP with single TAP, use '.' as the TAP instance name

- use examples (Intel-specific)

  (1) set tap_opcodes_to_exclude(htap2:0x32) *IOVRESET*

  (2) set tap_opcodes_to_exclude(.:0x32) *IOVRESET* - IP with single TAP

Note: Parameters $enable_security_tests_all_levels and $excluded_security_levels in the ip_pattern_cfg.do define a set of Intel Security Level Policies the security tests will be run for.

4.1.6.4 Reset Port Specification

Parameters, which specify DFx and Functional resets in the design, allow to create complex reset/power up scenarios to match IP requirements or validation needs.

This is an optional specification. If not provided, the tool will try to identify reset ports from the ICL using pattern *p*w*rgood* (case insensitive). If found, all these ports will be pulsed simultaneously. For powergood ports, which have ResetPort type in ICL, ActivePolarity property will be used to determine the required pulse polarity. For powergood ports, which have DataInPort type, active low polarity will be used.

Powergood Reset Port Specification

The specification format is:

set reset_name_list(<tag>:<polarity>) {<port list>};

- <tag> sets the unique ID to the group of POWERGOOD ports, for example, it can be DFX and FUNC (default). User can add new tags to define desired port groups.
- <polarity> defines an active reset level and can be 0 or 1
- <port list> Tcl list of reset port names in ICL (of ResetPort or DataInPort types)

Default spec in the configuration file:

```
set reset_name_list(DFX:0)  {fdfx_powergood}; #reset_b
set reset_name_list(DFX:1)  {}; #reset
set reset_name_list(FUNC:0) {}; #reset_b
set reset_name_list(FUNC:1) {}; #reset
```

It defines powergood reset group DFX with single active low reset fdfx_powergood.

4.1.6.5 Simulation Parameters

User can set a seed value to control randomization.

- parameter: $seed
  - values: integer number (default: 1)

4.1.6.6 IP-specific PDL Procedures

User can customize some standard procedures, which will be called during execution of the specific flows and tests.

Global Reset Preamble

It is executed once in the beginning of the default DUVE-M test flow.

Default sequence:

```
iTopProc tap_reset_preamble {} {
   set secure_level RED
   iCall tap_set_secure_policy
   iCall tap_reset POWERGOOD:DFX 10 reset_name_list
   iCall tap_set_secure_policy $secure_level
   iCall tap_init_ports slvidcode
   #iForcePort ftap_slvidcode[31:0]  0x12345679
   #iApply
}
```

The preamble uses generic PDL iProcs, available as a part of the DUVE-M sequence library.

The above flow initializes security policy ports (to policy 0), asserts POWERGOOD reset(s) for 10 TCLK cycles, then sets specified Secure policy (RED) and initializes SLVIDCODE strap port.

Pre-/Post- Reset Sequences

User can define custom sequences, which will be called before and/or after assertion/de-assertion of the specific resets. For example, if some register requires a special sequence to become accessible after powergood reset, that special recipe can be specified in the body of the reset_powergood_post procedures of the configuration file:

```
iTopProc reset_powergood_post  {} {
   iWrite IOVRESET_0 0b0
   iApply
}
```

Available pre-/post- reset sequences:

reset_powergood_pre {} {}	# pre- sequence for POWERGOOD resets
reset_powergood_post {} {}	# post- sequence for POWERGOOD resets
reset_tlr_pre {} {}	# pre- sequence for TLR resets
reset_tlr_post {} {}	# post- sequence for TLR resets
reset_trst_pre {} {}	# pre- sequence for TRST resets
reset_trst_post {} {}	# post- sequence for TRST resets
reset_default_pre {} {}	# pre- sequence for DEFAULT resets
reset_default_post {} {}	# post- sequence for DEFAULT resets

4.1.7 ip_test_hotfix.do

The file contains temporary exclusions and overrides which validator wants to apply due to the found bugs, tool limitations or other similar issues. Such settings shouldn't be a part of the "permanent" test/validation configuration of the ip_test_cfg.do. In the end, when all issues are resolved, these "hotfix" overrides should be completely removed. Functions of the parameters in that file are similar to the corresponding options in the ip_test_cfg.do (with no 'hotfix_' prefix in the names).

- parameter: $hotfix_excluded_modules (default: {})
- parameter: $hotfix_excluded_registers (default: {})
- parameter: $hotfix_trst_tlr_test_excluded_modules (default: {})
- parameter: $hotfix_trst_tlr_test_excluded_registers (default: {})
- parameter: tap_opcodes_to_exclude (the original Tcl array from the ip_test_cfg.do)

4.1.8 ip_user_cfg.do

The file is a placeholder for user-defined variables and procedures.

It is executed just before ip_test_cfg.do, making those user-defined variables and procedures available for use in the ip_test_cfg.do file.

4.1.9 write_patterns_v.param

This is a parameter file for write_patterns command of Tessent Shell. The file is provided as an argument of '-param' option when 'write_patterns -verilog' is executed to write out the Verilog Test Bench for generated patterns.

'SIM_*' parameters, specified as a part of that file, provide user with knobs to control some aspects of Tessent Verilog Test Bench generation, including adding of user-provided code (for example, a custom configuration for FSDB dump).

Detailed information about parameter file and supported keywords/options can be found in the Tessent Shell Reference Manual, chapter "Parameter File Format and Keywords."

4.2 Pattern Generation

DUVE-M includes a complete flow to generate test or validation patterns based on the provided design RTL/ICL/PDL collateral and DUVE-M test configuration files.

The flow is defined in the $DUVE_M_HOME/verif/flow/gen_val_tests.do file.

To run it at it is, use the following command line:

```
> tessent -shell -dofile $DUVE_M_ROOT/verif/flow/gen_val_tests.do
```

Note: Review Tessent Shell documentation for additional command line options.

As a result of DUVE-M flow execution, `<pattern_name>.v` (top level Verilog TB), `<pattern_name>.v.*.vec` (vector file with stimuli information), and a few other files will be generated. Together with the design RTL, this is a complete scope for compilation and simulation.

The flow also generates flattened PDL file `<pattern_name>.pdl`. That file contains a complete PDL sequence of the generated pattern. It includes PDL commands from user's PDL sources as well as the commands, added by Tessent to enable access to the targeted registers and to perform TDO checks based on data from the Tessent ICL Data Model. User can review all TAP IR and TDR writes/reads, including network/chain configuration, port initialization/stimuli and informational messages (iNotes).

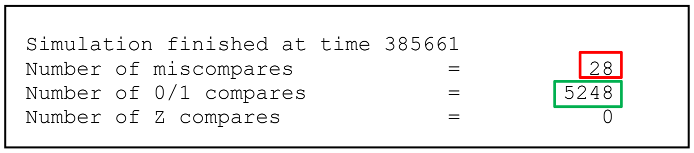
If flow changes are required, user can create and use local customized copy of that file.

4.3 Pattern Simulation

Any available Verilog simulator (e.g., VCS from Synopsys) can be used to simulate the generated pattern. The compilation/simulation scope should include generated `<pattern_name>.v` file of the top-level Verilog TB and RTL files of the design, that Test Bench was generated for.

The simulation status is reported in the end of the simulation log file (Figure 2).

Figure 2. Simulation Status in the Log File



```
Simulation finished at time 385661
Number of miscompares           =          28
Number of 0/1 compares          =        5248
Number of Z compares            =           0
```

If test fails, then it reports a number of mismatches (in the red box on the snippet above).

Note: Pay attention to the number of compares too. If it is smaller than expected (or even zero) then review the test and test setup - it can be that the test is not doing any real validation!

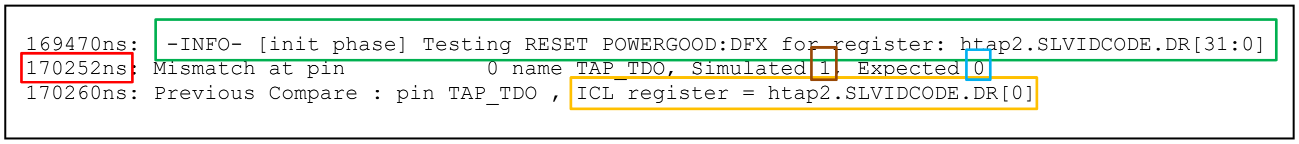
4.4 Debugging of the Simulation Failures

The following technique is recommended for debugging of the simulation mismatches of the Tessent-generated TAP tests.

4.4.1 STEP1: Start from the simulation log

Find the FIRST mismatch failure in the simulation log file. Typical example of such a message is shown in Figure 3.

Figure 3. Typical Mismatch Failure Message in the Log File



```
169470ns:  -INFO- [init phase] Testing RESET POWERGOOD:DFX for register: htap2.SLVIDCODE.DR[31:0]
170252ns: Mismatch at pin           0 name TAP_TDO, Simulated 1, Expected 0
170260ns: Previous Compare : pin TAP_TDO , ICL register = htap2.SLVIDCODE.DR[0]
```

Important data about the mismatch to notice (using snippet above as an example):

- Failure time (in the red box)
- Info message, related to the failing TAP access, which is usually just above the mismatch line (in the green box)
- Expected and simulated TDO data (in the blue and brown boxes correspondently)
- Do not yet consider information about failing register in the mismatch line (orange box) - it can be not accurate if TDI-TDO connectivity in hardware is different than what Tessent expected based on ICL model (due to some design or test issue)

Note: Always debug only the FIRST mismatch failure! When it is understood, eliminate that failure (by fixing RTL or other problematic collateral, by excluding failing component from the test, by test/configuration changes, etc.), regenerate the pattern (if necessary), simulate it and continue working on that new simulation result, the new FIRST failure. Continue this process until all failures are debugged/resolved.

4.4.2 STEP2: Review pattern vector file

Now, review Tessent-generated pattern vector file (`<pattern_name>.v.*.vec`).

Using reported failure time from the log file, search for the corresponding TimeStamp in the vector file. TimeStamp statements are printed every 100ns, so you need to round the time and search for 'Timestamp 1702' for the failure example above.

To double-check location in the vector file, locate the message, printed just before the failure in the log file.

Simplified snippet from the vector file for the provided example is shown in Figure 4.

The figure shows where different pieces of information from the log file can be found - info message, time stamp, target register name and expected data.

Based on that information, review the test flow and understand what is expected from the hardware. It might be needed to look at the previous operations/accesses to better understand initial conditions for the failing instruction.

When the expected behavior and TAP network/TDI-TDO chain layout is understood, it is time to start looking at the FSDB dump and RTL.

4.4.2.1 Anatomy of the Tessent Vector file - TAP Access

The file contains vectors, applied to the ports of the DUT. The vectors are driven/captured every tester cycle based on the defined timeplates.

Note: Tessent allows customization of time when signals are driven and response values are captured. Review the tool documentation if non-default setting is required.

The snippet of vector file in Figure 4 highlights the most important information about TAP operations:

- First, it is a header with data about target register and write/read data
- Then it is information about loaded opcodes into the TAP IR's in the TDI-TDO chain.
- And then, finally, information about access of the TDR registers in the chain.

A very important aspect, which should be noticed, is what is included into the TDI-TDO chain for both IR and DR accesses, including MSB:LSB ranges of each component.

Also, pay attention to what is covered by TDO checker. To find this out, look at the expected TDO data, shown in the "// Unloading:" line. Symbols '0', '1' and 'x' represents expected values for the TDO checker. If it is 'x', then the checker will ignore that bit.

The left-most symbol of the bitstream corresponds to MSB (TDI) side of the chain. Respectively, the right-most symbol corresponds to LSB (TDO) side. Underscores ('_') separate out data for different components in the chain.

Based on the provided example, unloaded data for the first access to htap2 SLVIDCODE will be not checked at all (purple box). TDO checking will be performed during second access, when All'0s are expected (blue box).

4.4.3 STEP3: RTL/FSDB debug

When the test flow, test assumptions and expectations are understood, it's time to look at the RTL and waveforms. All following recommendations are based on features and capabilities of Synopsys Verdi Debug system.

Note: If Synopsys EDA tools are used, for failure debug, it is recommended to use up-to-date version of the Verdi tool.

The first task to accomplish - to go to the time of the failure and trace failing top level TDO port back to the TDI port, comparing the layout of the active TDI-TDO chain with the expected one from the vector file. If the failure happened during TDR access, it should be during Shift-

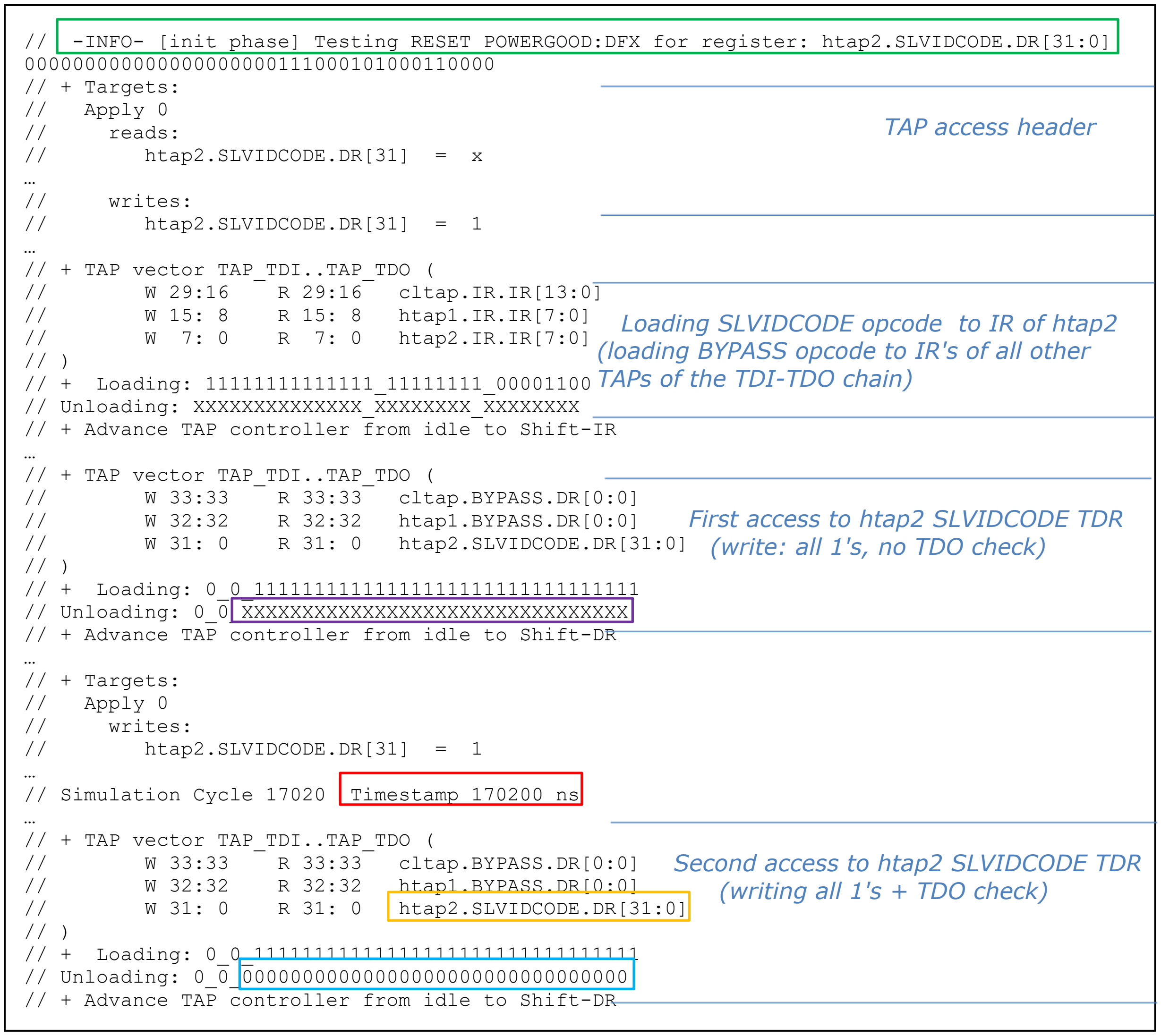
DR state of TAP FSM. In Verdi, turn on active annotation for ease of selection of the active directions of muxes or active path through combinatorial logic.

If layout of the active TDI-TDO chain does not match expected by the test, then look at the previous TAP operations to detect when the mismatch happened and why.

If chain topology is correct, then check what is loaded into the shift chain of the TDR under test during Capture-DR state of the TAP FSM.

This should allow to root cause the issue and help to resolve it - fix RTL of the design, test configuration, test sequence or some other related collateral.

Figure 4. Information about TAP Accesses in the Tessent Vector File


```
//  -INFO- [init phase] Testing RESET POWERGOOD:DFX for register: htap2.SLVIDCODE.DR[31:0]
000000000000000000000111000101000110000
// + Targets:
//   Apply 0
//     reads:
//        htap2.SLVIDCODE.DR[31]  =  x
…
//     writes:
//        htap2.SLVIDCODE.DR[31]  =  1
…
// + TAP vector TAP_TDI..TAP_TDO (
//        W 29:16    R 29:16   cltap.IR.IR[13:0]
//        W 15: 8    R 15: 8   htap1.IR.IR[7:0]
//        W  7: 0    R  7: 0   htap2.IR.IR[7:0]
// )
// +  Loading: 11111111111111_11111111_00001100
// Unloading: XXXXXXXXXXXXXX_XXXXXXXX_XXXXXXXX
// + Advance TAP controller from idle to Shift-IR
…
// + TAP vector TAP_TDI..TAP_TDO (
//        W 33:33    R 33:33   cltap.BYPASS.DR[0:0]
//        W 32:32    R 32:32   htap1.BYPASS.DR[0:0]
//        W 31: 0    R 31: 0   htap2.SLVIDCODE.DR[31:0]
// )
// +  Loading: 0_0_11111111111111111111111111111111
// Unloading: 0_0_XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// + Advance TAP controller from idle to Shift-DR
…
// + Targets:
//   Apply 0
//     writes:
//        htap2.SLVIDCODE.DR[31]  =  1
…
// Simulation Cycle 17020  Timestamp 170200 ns
…
// + TAP vector TAP_TDI..TAP_TDO (
//        W 33:33    R 33:33   cltap.BYPASS.DR[0:0]
//        W 32:32    R 32:32   htap1.BYPASS.DR[0:0]
//        W 31: 0    R 31: 0   htap2.SLVIDCODE.DR[31:0]
// )
// +  Loading: 0_0_11111111111111111111111111111111
// Unloading: 0_0_00000000000000000000000000000000
// + Advance TAP controller from idle to Shift-DR
```
TAP access header
Loading SLVIDCODE opcode to IR of htap2 (loading BYPASS opcode to IR's of all other TAPs of the TDI-TDO chain)
First access to htap2 SLVIDCODE TDR (write: all 1's, no TDO check)
Second access to htap2 SLVIDCODE TDR (writing all 1's + TDO check)

5 Tests

5.1 DUVE-M Tests

These are the TAP validation tests, which are available as a part of DUVE-M:

- TDI-TDO continuity test
- TAP IR/TDR reset test
- TDR Read-Write access tests

TDR Security test

- Reserved Opcodes test

The tests use ICL metadata to implement generic verification algorithms.

All test procedures are defined in the following file:

- $DUVE_M_HOME/verif/pdl/common/tap_tests_common.pdl

5.1.1 TDR TDI-TDO Continuity Test

The test checks TDI-TDO continuity while accessing all TDR registers scoped based on parameters in the ip_test_cfg.do configuration file.

iTopProc name: tap_dr_continuity_test

Target Register Scope

- {$included_instances + $included_registers - $excluded_modules - $excluded_registers - $hotfix_excluded_modules - $hotfix_excluded_registers}
  - Based on the default flow defined in the $DUVE_M_HOME/verif/pdl/tap_tests_all.pdl

Arguments

- Collection of ICL register objects to test

Algorithm

- The described sequence is applied to each specified register
- The sequence measures a length of TDI-TDO chain when accessing the TDR by its overshifting and checking TDO. The overshift is achieved by doing a few shift cycles which all end up in the Pause-DR state and then return back to the Shift-DR for the next shift cycle.
- Single shift cycle uses the TDI patterns with markers on the MSB and LSB sides of TDI data to shift into the target register. For wide registers, it will look like 0100..0010 or its inversion 1011..1101 (length of each pattern is equal to the size of the target register). For short registers, due to not sufficient number of overshifted bits per shift cycle, additional overshift cycles will be performed using both original and inverted patterns to guarantee reliable measuring of the chain length.
- The last shift cycle, which will update the state of the register (TAP FSM will pass through the Update-DR), loads safe value into the register (register reset value).

Debug Help

- If chain layout in RTL matches the expected layout based on ICL model, then after each shift cycle (when TAP FSM is in Pause-DR), input TDI pattern with markers should exactly "fit" the shift register of the target TDR. If the sequence fails, any misalignment should be easily noticeable.

5.1.2 TDR Reset Test

The test can be used for validation of reset behavior of TDR during powergood resets (with the tags, specified in the ip_test_cfg.do) and TLR reset. Unfortunately, TRST reset is considered as a Global reset (iReset) by IEEE 1687 and its validation cannot be done accurately without enhancing of the Tessent tool (WIP with the tool provider).

Note: The test is checking both cases: 1) that the specific reset assertion resets values of the expected scope of the registers and 2) that it does not change values of the remaining registers.

Note: Do not use current test for validation of the TRST!

iTopProc name: tap_dr_reset_test

Target Register Scope

- POWERGOOD* tests: {$included_instances + $included_registers - $excluded_modules - $excluded_registers - $hotfix_excluded_modules - $hotfix_excluded_registers - $continuity_test_only_registers}
    - Based on the default flow defined in the $DUVE_M_HOME/verif/pdl/tap_tests_all.pdl
- TLR/TRST tests: { $included_instances + $included_registers - $excluded_modules - $excluded_registers - $hotfix_excluded_modules - $hotfix_excluded_registers - $continuity_test_only_registers - $trst_tlr_test_excluded_modules - $trst_tlr_test_excluded_registers - $hotfix_trst_tlr_test_excluded_modules - $hotfix_trst_tlr_test_excluded_registers }
    - Based on the default flow defined in the $DUVE_M_HOME/verif/pdl/tap_tests_all.pdl

Arguments

- reset_type (string, default: iReset). Supported values: iReset, POWERGOOD, POWERGOOD:<tag>, TLR, TRST. Currently defined powergood tags are DFX and FUNC (based on the ip_test_cfg.do)
- strict (0|1, default 0, used for powergood tests only). When set '0' (default), Tessent will enable TDO check based on its ICL data model when doing iWrite to the target register after the reset event. When set '1', the test will override Tessent prediction and will calculate expected value based on the ICL data using get_exp_rst_tdo proc of DUVE-M tap_utils package and then will issue explicit iRead check with that calculated value.
- Collection of ICL register objects to test
- Collection of register objects which are expected to be reset by TLR
    - Based on parameters $dr_modules_tlr and $dr_registers_tlr in the ip_test_cfg.do
- Collection of register objects which are expected to be reset by TRST
    - Based on parameters $dr_modules_trst and $dr_registers_trst in the ip_test_cfg.do
- Collection of ICL register objects, representing TAP network/chain control registers (based on the parameters $network_control_modules and $network_control_registers in the ip_test_cfg.do)
    - These registers will be auto excluded from DUVE-M TRST/TLR tests

Algorithm

- The described sequence is applied to each specified register

- Initialize target TDR register using inverted reset value
- Apply the specified reset(s)
- Read register value and compare it with the expected value (with the reset value, if the applied reset expects to reset the register or with the previously written one, if not)
    - For powergood tests, when $strict=0 (default), Tessent adds TDO checking based on its ICL modeling. If $strict=1, DUVE-M will be using its own calculation to check TDO.

5.1.3 TDR Read-Write Test with Configurable Pattern

The test verifies read-write access to the specified TDR registers. It uses the algorithm, similar to the MATS+ for array testing.

With the default pattern, the test is capable to screen repeaters which use a free-running TAP clock.

iTopProc name: tap_dr_rw_test

Target Register Scope

- {$included_instances + $included_registers - $excluded_modules - $excluded_registers - $hotfix_excluded_modules - $hotfix_excluded_registers - $continuity_test_only_registers}
- Based on the default flow defined in the $DUVE_M_HOME/verif/pdl/tap_tests_all.pdl

Arguments

- Collection of ICL register objects to test
- Pattern (type: string, default: "01"). The provided pattern will be concatenated multiple number of times to fit full register width. For example, for 5-bit register, the final pattern will be "10101".

Algorithm

The described Pass1, Pass2, and Pass3 loops are performed for all specified registers.

- Pass1: Initialize all target registers using the specified pattern. For each register, use double shift with intermediate Pause-DR between. TDO check is not performed for the first shift. For the second shift, TDO is checked for the overshifted value (expected to be the specified input pattern).
- Pass2: For all registers, read previously written data, check TDO, and write inverted pattern. Use double shift with intermediate Pause-DR.
- Pass3: Perform one more loop to read last written data, check TDO, and to write safe value to each register (its reset value).

5.1.4 TDR Read-Write Test with Randomization

The test verifies read-write accesses to the specified registers using random patterns. Randomization sequence is controlled by the $seed parameter in the ip_test_cfg.do file.

iTopProc name: tap_dr_rw_random_test

Target Register Scope

- {$included_instances + $included_registers - $excluded_modules - $excluded_registers - $hotfix_excluded_modules - $hotfix_excluded_registers - $continuity_test_only_registers}

- Based on the default flow defined in the $DUVE_M_HOME/verif/pdl/tap_tests_all.pdl

Arguments

- Collection of ICL register objects to test
- num_test_cycles (type: integer, default: 1). Number of loops to run the test (with different random generated patterns). DUVE-M flow uses $num_test_cycles=2.

Algorithm

- The test writes random data to each specified register and performs TDO check for all accesses after the first one. Number of write/read cycles is controlled by the num_test_cycles argument. In the last access, each register will be written with the safe value (its reset value).

5.1.5 TDR Field Read-Write Test with Randomization

Similar to the previously described read-write test with randomization (tap_dr_rw_random_test), but the test generates and applies random value individually to each field of the target register (register field is defined using Alias statement in the ICL).

iTopProc name: tap_dr_field_rw_random_test

Target Register Scope

- Same as tap_dr_rw_random_test

Arguments

- Collection of ICL register objects to test
- num_test_cycles (type: integer, default: 1). Number of loops to run the test (with different random generated patterns). DUVE-M flow uses $num_test_cycles=2.

Algorithm

- Same as for tap_dr_rw_random_test, but random patterns are generated and applied to the register fields (Aliases).

5.1.6 Reserved Opcodes Test

This test is Intel-specific and relies on the naming convention to find instances of reserved bypass registers in ICL.

The test performs accesses and TDI-TDO chain length/returned TDO value checks for all IR opcodes with not assigned "functional" TDR in the ICL of the specified TAPs.

Note: The test must be performed on the unlocked TAPs - Security Policy level should be set to RED (level 2 or 4).

iTopProc name: tap_reserved_opcodes_test

Target Register Scope

- TAPs: based on parameter $tap_opcodes_tap_instances in the ip_test_cfg.do
- Registers: for each specified TAP, all IR opcodes excluding the opcodes, specified with parameter tap_opcodes_to_exclude(<current tap instance>:<opcode>) in the ip_test_cfg.do. For example, for htap2 TAP with 8-bit IR and single exclusion tap_opcodes_to_exclude(htap2:0x32), the test will be exercising 255 opcodes with values (0..0x31, 0x33…0xFF).

Arguments

- List of TAPs to perform the test for (parameter $tap_opcodes_tap_instances in the ip_test.cfg.do)
- Information about opcode exclusions per TAP (parameter tap_opcodes_to_exclude in the ip_test_cfg.do)

Algorithm

- The described sequence is applied to each specified TAP instance
- For each TAP, access all not excluded IR opcodes
- If opcode is reserved (it is associated with *bypass_rsvd_reg module in ICL), then perform multiple shift accesses with intermediate Pause-DR to overshift the bypass register and check returned TDO (the test uses easy identifiable TDI pattern 101100111000).

5.1.7 Opcode Security Test

This test is Intel-specific and relies on the naming convention to find instances of the reserved bypass registers in the ICL.

The test performs accesses and TDI-TDO chain length/returned TDO value checks for all IR opcodes with not assigned "functional" TDR in the ICL of the specified TAPs.

Note: Test must be performed for all Security levels from 0 to 15!

iTopProc name: tap_all_opcodes_security_test

Target Scope

- TAPs: based on parameter $tap_opcodes_tap_instances in the ip_test_cfg.do
- Registers: for each specified TAP, all IR opcodes excluding the opcodes, specified with parameter tap_opcodes_to_exclude(<current tap instance>:<opcode>) in the ip_test_cfg.do. For example, for htap2 TAP with 8-bit IR and single exclusion tap_opcodes_to_exclude(htap2:0x32), the test will be exercising 255 opcodes with values (0..0x31, 0x33…0xFF).
- Security Levels: defined by parameters $enable_security_tests_all_levels and $excluded_security_levels (in the ip_patterns_cfg.do). By default, the test will be performed for GREEN, RED, RED4, ORANGE Security Policies (levels 0,2,4,5 correspondently). Setting $enable_security_tests_all_levels to 1 will execute that test for all other security levels, excluding the levels, excluded using parameter $excluded_security_levels.

Arguments

- List of TAPs to perform the test for (parameter $tap_opcodes_tap_instances in the ip_test.cfg.do)
- Information about opcode exclusions per TAP (parameter tap_opcodes_to_exclude in the ip_test_cfg.do)
- security_level (type: string). Supported values: RED, RED4, ORANGE, GREEN, and numeric values, representing security levels (from 0 to 15 in any Tcl-compliant number format)

Algorithm

- The described sequence is applied to each specified TAP instance
- For each TAP, access all not excluded IR opcodes. Perform three loops - Pass1, Pass2 and Pass3.

- Pass1: Unlock TAP using RED level and initialize all defined TDR and registers, associated with the reserved opcodes, to 0. Perform the second access to check if initialization was successful.
- Pass2: Change security level to the specified for validation. Perform write access with "corrupting" value 0b1 to all opcodes, associated with the bypass reserved registers in the ICL for that security level. If RTL mismatches the ICL spec and some writable functional TAP register, which is not supposed to be accessible at the given security level, is actually not protected, it will be updated (corrupted) by that access.
- Pass3: Unlock the TAP using RED level and access all functional TDRs, read and check TDO. If some register was unexpectedly updated (corrupted) during Pass2, it will be flagged in simulation. All registers will be written with the value 0 in that pass.

5.1.8 TDR Readability Check

This is a very important check which reports all registers which have not readable bits. Not readable means that no TDO check is performed for those bits and no any validation coverage is gained by the basic TAP tests.

Note: User is responsible for creating of directed tests for such registers/bits to cover validation items in the project Test Plan.

This check is called in the end of the default DUVE-M pattern generation/test flow. It does not require simulation - the results are immediately available in the Tessent log file.

Note: Always review the report of that check!

The check is a part of DUVE_M tap_utils package.

proc name: tap_utils::print_all_capture_source_x_info

Target Register Scope

- {$included_instances + $included_registers - $excluded_modules - $excluded_registers - $hotfix_excluded_modules - $hotfix_excluded_registers - $continuity_test_only_registers}
    - Based on the default flow defined in the $DUVE_M_HOME/verif/pdl/tap_tests_all.pdl

Report Example

```
//  sub-command: iNote "-REG_INFO- htap2.DEBUGCOUNTERREAD_LCLK.DR[29:0] has 30 UNREADABLE
bits  out of 30 total"
```

5.2 User-Defined Tests

User can create their own PDL tests, using (or not using) DUVE-M infrastructure/utilities and then use DUVE-M to generate the corresponding patterns.

The pointer to the user PDL file can be specified in the ip_cfg.do using parameter $ip_user_tests_pdl. By default, the parameter references $IP_ROOT/verif/pdl/ip_user_tests.pdl

To enable that test during pattern generation, use $enable_user_tests parameter in the ip_patterns_cfg.do configuration file.

6 Tessent Flow

Default DUVE-M pattern generation flow is defined in the

- $DUVE_M_HOME/verif/flow/gen_val_tests.do

If necessary, the user can create their own version using that dofile as a reference.

The simplified structure of that flow (gen_val_tests.do) is shown in Figure 5.

Figure 5. Default DUVE-M dofile Flow

```
// Load IP Tessent cfg
source $env(IP_TESSENT_CFG)

// Set context
set_context patterns -ijtag

// Load ICL and design RTL
dofile $ip_read_icl_dofile
dofile $ip_read_verilog_dofile

// Set top level of the design to work with
set_current_design $current_design

add_black_boxes -auto
report_clocks

// Transition from setup mode to analysis mode
check_design_rules

// Load common DUVE-M test procedures
source $env(DUVE_M_HOME)/verif/pdl/common/tap_utils.pdl
source $env(DUVE_M_HOME)/verif/pdl/common/tap_tests_common.pdl

// Pattern-specific setup
source $ip_pattern_cfg_dofile

// Open pattern set
set_ijtag_retargeting_options -tck_period $tck_period
open_pattern_set $pattern_set -tester_period $tester_period -tck_ratio $tck_ratio

// User-defined setup
source $ip_user_cfg_dofile

// test-specific setup and temporary hotfixes
source $ip_test_cfg_dofile
source $ip_test_hotfix_dofile

///////// Test flow ////////
source $ip_test_flow_pdl
if {$enable_user_tests} {
   source $ip_user_tests_pdl
}
///////// End of test flow ////////

close_pattern_set

write_patterns <…>.pdl -pdl -replace
write_patterns <…>.v -verilog -replace -param $ip_write_patterns_v_param_file

report_pattern_set

exit
```

Use Tessent Shell documentation to find more details about Tessent flow and commands.

As for the DUVE-M test flow, it is defined in the

- $DUVE_M_HOME/verif/pdl/tap_tests_all.pdl

The simplified test flow is shown in Figure 6.

Figure 6. Default DUVE-M Test Flow

```
tap_utils::set_seed $seed

// Calculate reister scopes for tests (not shown)

// Each test below can be individually enabled and disabled

iCall tap_dr_continuity_test $test_reg_collection

iCall tap_dr_reset_test $test_reg_collection POWERGOOD:DFX 0 …

iCall tap_dr_reset_test $trst_tlr_test_collection TRST 0 …

iCall tap_dr_reset_test $trst_tlr_test_collection TLR  0 …

iCall tap_dr_rw_test $test_reg_collection 01

iCall tap_dr_rw_random_test $test_reg_collection 2

iCall tap_dr_field_rw_random_test $test_reg_collection 2

iCall tap_reserved_opcodes_test $tap_opcodes_tap_instances tap_opcodes_to_exclude …

iCall tap_all_opcodes_security_test $tap_opcodes_tap_instances <level>
tap_opcodes_to_exclude …

tap_utils::print_all_capture_source_x_info $test_reg_collection
```

7 Additional Procedures and Utilities

The procedures, available as a part of DUVE-M, allow writing generic PDL tests and accessing RTL/ICL/PDL metadata in the Tessent Data Models (for example, to generate some project collateral).

These procedures and utilities serve the following functions:

- Generation of resets with configurable type/target signals/duration
- Security level control (by forcing of the *secure* ports)
- Auto-initialization of strap values (based on the Intel-specific ICL attribute)
- Register scoping
- IR/DR<->TAP mapping
- TAP->Parent TAP mapping
- Custom iRead/iWrite procedures
- Calculation of the expected TDO, TDR RW mask and TDO mask
- Access of TDR metadata: register size, reset value, security, alias info, etc.
- Numeric/bit stream data processing

7.1 Test Sequences

Reset Generation

- iTopProc tap_reset

  A procedure to generate custom resets

  Location:

  $DUVE_M_HOME/verif/pdl/common/tap_tests_common.pdl

  Arguments:

  - reset_type (type: string, default: iReset). Supported values: iReset, POWERGOOD, POWERGOOD:<tag>, TLR, TRST. Currently defined powergood tags are DFX and FUNC (based on ip_test_cfg.do)
  - duration (type: integer, default 10) Reset duration in TAP clocks

  Usage examples can be found in the $DUVE_M_HOME/verif/pdl/tap_tests_all.pdl

- iTopProc tap_set_secure_policy

  - A procedure to enforce Intel Secure Policy for the given design. It generates a sequence for SECURE_POLICY, EARLYBOOT_EXIT and POLICY_UPDATE top-level ports to apply user-specified security level. Actual port names in RTL are auto detected based on typical naming patterns.

  Location:

  $DUVE_M_HOME/verif/pdl/common/tap_tests_common.pdl

  Arguments:

  - type (type: string, default: GREEN). Supported values: GREEN, RED, RED4, ORANGE and numeric values from 0 to 15 (using Tcl-compliant format)

Usage examples can be found in the $DUVE_M_HOME/verif/pdl/tap_tests_all.pdl

- iTopProc tap_init_ports
- A procedure to initialize input ports, specified in the ICL using DataInPort type with DefaultLoadValue property. That DefaultLoadValue will be used for port initialization.

  Location:

  $DUVE_M_HOME/verif/pdl/common/tap_tests_common.pdl

  Arguments:

  - signal_type (type: string, default: ""). If argument is the empty string, all DataInPorts with some specified DefaultLoadValue will be initialized. If the value is provided, only the ports, which have ICL Attribute 'intel_signal_type' with the matching value, will be considered for initialization.
  - use_force (values: 0 | 1, default: 1). If '0', the procedure uses iWrite method. If '1', iForcePort is used instead.

  Usage examples can be found in the $DUVE_M_HOME/verif/pdl/tap_tests_all.pdl

7.2 Utilities and Procedures

These procedures are a part of DUVE-M tap_utils package, defined in the $DUVE_M_HOME/verif/pdl/common/tap_utils.pdl.

7.2.1 TAP Metadata Access and Processing

- tap_utils::get_tap_reg_size

  The procedure returns size of the specified tap register.

  Arguments:

  - reg (type: object spec): Tcl list of one or more ScanRegister object names or a collection of one or more ScanRegister objects

- tap_utils::get_reg_reset_value_bin

  The procedure returns register reset value (X's are substituted with 0's).

  Arguments:

  - reg (type: object spec): Tcl list of one or more ScanRegister object names or a collection of one or more ScanRegister objects

- tap_utils:: get_rw_mask

  The procedure returns register-wide read-write mask for the specified register based on the value/sources of the CaptureSource property in ICL. In the returned string, value '1' corresponds to the read-writable bit type (RW), value '0' - to all other types (RW/V, RO/V, WO, RO).

  Arguments:

  - reg (type: object spec): Tcl list of one or more ScanRegister object names or a collection of one or more ScanRegister objects

- tap_utils:: get_tdo_mask

  The procedure returns register-wide TDO mask for the specified register based on the value/sources of the CaptureSource property in ICL. In the returned string, value 1 corresponds to the readable bit type (RW, RO access types), value '0' - to the not readable bit type (RW/V, RO/V, WO).

  Arguments:

  - reg (type: object spec): Tcl list of one or more ScanRegister object names or a collection of one or more ScanRegister objects

- tap_utils::get_reg_prev_value_cmp

  The procedure returns final expected value for (iRead) comparison of the read register value.

  Arguments:

  - reg (type: object spec): Tcl list of one or more ScanRegister object names or a collection of one or more ScanRegister objects
  - prev_value (type: string): string, representing initial expected value in binary format (without 0b)

- tap_utils::get_exp_rst_tdo

  The procedure returns TDO value, which will be observed when reading specified register after applying the reset which resets that register.

  Arguments:

  - reg (type: object spec): Tcl list of one or more ScanRegister object names or a collection of one or more ScanRegister objects

- tap_utils::print_capture_source_x_info

  The procedure prints a total width of X bits of the CaptureSource property for the specified register instance if that X width is not 0.

  Arguments:

  - Register object in ICL

- tap_utils::get_reg_alias_list

  The procedure returns list of the field Aliases of the specified register.

  Arguments:

  - reg (type: object spec): Tcl list of one or more TDR object names or a collection of one or more TDR objects

- tap_utils::get_tap_dr_field_size

  The procedure returns size of the specified register field (Alias).

  Arguments:

  - alias (type: object spec): Tcl list of one or more Alias object names or a collection of one or more Alias objects

7.2.2 TAP Register Scoping

- tap_utils::get_all_regs

  The procedure returns a collection of ICL objects of all registers in the design, below the specified instances, or based on the path matching patterns.

  Arguments:

  - inst_path (type: list of strings, default: *): allows to find ScanRegister instances in the top scope ('*' or '.' to find all registers) or below the specified instances (wildcards are supported)
  - reg_names (type: list of strings, default: {}): allows to find ScanRegister instances based on provided set of patterns (wildcards are supported)

- tap_utils::find_regs

  The procedure returns a collection of ICL objects of registers in the specified modules or based on the path matching patterns.

  Arguments:

  - module_names (type: list of strings, default: *): allows to find ScanRegister instances inside the specified module (wildcards are supported)
  - reg_names (type: list of strings, default: {}): allows to find ScanRegister instances based on the provided set of patterns (wildcards are supported)

- tap_utils::remove_regs_from_collection

  The procedure removes ICL registers from the specified collection of objects based on the provided module/instance/pattern list.

  Arguments:

  - reg_collection - a collection of ICL objects to be processed
  - excluded_modules (type: list of strings, default: {}): a list of modules with the registers to be excluded from the input collection (wildcards are supported)
  - excluded_registers (type; list of strings, default: {}): a list of register instances to be excluded from the input collection (wildcards are supported)

7.2.3 TAP Register Mapping

- tap_utils::map_tap_ir_regs

  The procedure finds TAP instance for the specified IR registers and populates the hash with that information, allowing quick finding of the related IR register for the given TAP instance (TAP -> IR mapping).

  Arguments:

  - ir_list (type: collection of ICL objects): a collection of TAP IR register objects
  - tap_ir_map (type: array): a hash with IR registers to populate

- tap_utils::map_tap_regs

  The procedure finds TAP instances for the specified IR and TDR registers and populates hashes with that information, allowing quick finding of the related TAP for the given TDR and then related IR register (TDR -> TAP and TAP -> IR mapping).

  Arguments:

  - reg_list (type: collection of ICL objects): a collection of TAP TDR register objects
  - ir_list (type: collection of ICL objects): a collection of TAP IR register objects
  - tap_dr_map (type: array): a hash with TDR registers to populate.
  - tap_ir_map (type: array): a hash with IR registers to populate.

- tap_utils::get_tap_parent

  The procedure finds direct parent TAPs for the specified TAP instances.

  Prerequisites: the procedure uses populated TAP IR hash.

  Arguments:

  - tap_inst (type: list of strings of collection of ICL objects): a list of TAPs to find parents for.
  - tap_ir_map (type: array): a populated hash with TAP->IR mapping

    Note: Temporary argument, will be eliminated after code refactoring

- tap_utils::map_tap_parents

  The procedure populates a hash with TAP child-> TAP parent mapping.

  Prerequisites: the procedure uses populated TAP IR hash.

  Arguments:

  - tap_parent_map (type: array): a hash with TAP mapping to populate.
  - tap_ir_map (type: array): a populated hash with TAP->IR mapping

    Note: Temporary argument, will be eliminated after code refactoring

- tap_utils::tap_get_parents

  The procedure returns ordered list of parent TAPs above the specified one, starting from the direct parent (tap1:tap_inst_parent tap2:tap1_parent …)

  Prerequisites: the procedure uses populated TAP parents hash.

  Arguments:

  - tap_inst (type: string): a child TAP instance of the interest (instance path)
  - tap_parent_map (type: array): a populated hash with child TAP -> parent TAP mapping.

- tap_utils::tap_get_parents_all

  The procedure returns ordered list of ALL TAPs above the specified one.

  Prerequisites: the procedure uses populated TAP parents hash.

Arguments:

- tap_inst (type: string): a child TAP instance of the interest (instance path)
- tap_parent_map (type: array): a populated hash with child TAP -> parent TAP mapping.

7.2.4 TAP Special iWrite/iRead

- tap_utils::iwrite_tap_bypass_ir

  The procedure writes BYPASS opcode (All 1's) to the IR register of the TAP with the specified TDR.

  Prerequisites: the procedure uses populated TAP IR and DR hashes.

  Arguments:

  - reg_name (type: string): initial TDR register (instance path)
  - tap_dr_map (type: array): a populated hash with TDR -> TAP mapping.
  - tap_ir_map (type: array): a populated hash with TAP -> IR mapping.

- tap_utils::iwrite_tap_bypass_dr

  The procedure writes to BYPASS TDR of the TAP of the specified TDR (Intel-specific implementation, assuming BYPASS register name).

  Prerequisites: the procedure uses populated TAP IR and DR hashes.

  Arguments:

  - reg_name (type: string): initial TDR register (instance path)
  - tap_dr_map (type: array): a populated hash with TDR -> TAP mapping.
  - tap_ir_map (type: array): a populated hash with TAP -> IR mapping.

7.2.5 Randomization

- tap_utils::set_seed

  The procedure sets a seed for a generator of the random numbers.

  Arguments:

  - seed (type: integer): input seed

- tap_utils::get_random_value_bin

  The procedure generates a random binary value with the required size.

  Arguments:

  - size (type: integer): required size of the returned binary number

7.2.6 Data Processing and Transformation

- tap_utils::num2bin

  The procedure converts numeric value into the binary representation (with no '0b'). The result can be padded to return the string of the required size if specified.

Arguments:

- value (type: Tcl number): input numeric value
- size (type: integer): required size of the binary string (optional)

- tap_utils::bin2dec

  The procedure converts binary value (with no '0b') into the decimal representation.

  Arguments:

  - value (type: 0/1 string): input binary string

- tap_utils::get_inverted_sting_value_bin

  The procedure returns bitwise-inverted value of the binary string.

  Arguments:

  - value (type: 0/1 string): input binary string

- tap_utils::get_expanded_pattern_bin

  The procedure expends the provided binary pattern to the specified size by repeating it required number of times and truncating the result if necessary.

  Arguments:

  - pattern (type: 0/1 string): input binary string
  - reg_size (type: integer): target pattern size

8 Design DRC

[Placeholder]

In the future, DUVE-M will be enhanced to perform Intel-specific quality checks of ICL collateral.

9 Creating IP ICL

9.1 TAP RDL2ICL Flow

SystemRDL is the industry standard language to describe characteristics of the registers.

Intel adopted SystemRDL language for specification of Control Registers (CR), Fuses and TAP registers. Today, TAP RDL is a required deliverable for every IP.

DTEG created a tool (ICLGEN or tap_icl_gen.pl) which allows generation of IP ICL from the existing TAP RDL.

The tool supports a complete scope of the RDL syntax which can be used in TAP RDL. Input RDL can have multiple-instantiated instances (MI), dynamic assignments, default values and use parameterization (include embedded Perl code).

9.1.1 Intel TAP RDL: Introduction

This is how SystemRDL objects are mapped to the TAP components:

addrmap	: TAP definition and integration levels above
regfile	: TAP chain definition (IJTAG, etc.)
reg	: TAP register definition
field	: TAP register field definition

Intel defined an extension of SystemRDL properties (UDP) to describe characteristics of TAP networks and TAP registers. That collection of TAP-related properties was defined as a part of collateral of Nebulon tool, which was created to process RDL specs of Intel designs and generate the required collateral for validation flows and Post-Si tools.

DUVE-M uses a copy of the Nebulon TAP UDP file. It is located in the $DUVE_M_HOME/icl_gen/udp/ folder. The tap_udp.rdl file provides definitions of all Intel-specific TAP RDL UDP.

For ICL generation, only a limited subset of UDP properties is used, which is shown in Table 1.

For the shown UDP's, if not specifically noticed, the default values for string properties is "", for number properties - 0, and for Boolean properties - "false".

Note: In RDL, property values can be specified as a part of component definition and using overrides (dynamic assignments) when a component is instantiated at the integration level.

Table 1. ICLGEN: Used TAP RDL UDP

UDP	Description
All RDL components (addrmap | regfile | reg | field)	
name	verbose name
desc	description
TapCollateralVisibility	property to control generation of external TAP documentation
Component: addrmap	
TapInstName	TAP name (single-word ID)

UDP	Description
TapType	type of TAP ("", "<tap_type>", "reg_only")
TapIrLength	TAP IR size
TapIrCaptureValue	read value of TAP IR (usually 0x1)
TapSecurityLevel	Intel Security Level, assigned to TAP at the integration level
Component: reg	
TapTotalNumRegBits	TDR update register width
TapShiftRegLength	TDR shift register length (usually the same value as TapTotalNumRegBits)
RegOpcode	IR opcode, assigned to the specific TDR
TapOpcodeSecurityLevel	Intel Security Level, assigned to the TAP opcode
TapRegResetType	Register reset type ("PWRGOOD", "PWRGOOD_<id>", "TLR", "TRST", "PWRGOOD, TRST, TLR", "PWRGOOD_<id>, TRST, TLR"
TapDrIsFixedSize	Indication of TDR with fixed or variable (configurable) size. Supported values: "true", "false". Default (if not specified): "true". **Use 'false' value to indicate that the register spec does not capture complete length dependencies of the TDR chain and the register should be blackboxed in the generated ICL**.
TapSibRef	parent SIB reference (used to describe IJTAG chains)
Component: field	
AccessType	Access type of the register field ("RW", "RW/V", "WO", "RO", "RO/V")
TapFieldIsNoInit	not initialized register field (with no reset)

Examples of TAP RDL for different IP configurations can be found in the $DUVE_M_HOME/icl_gen/examples/ folder.

Refer to the SystemRDL specification for details about RDL spec definition.

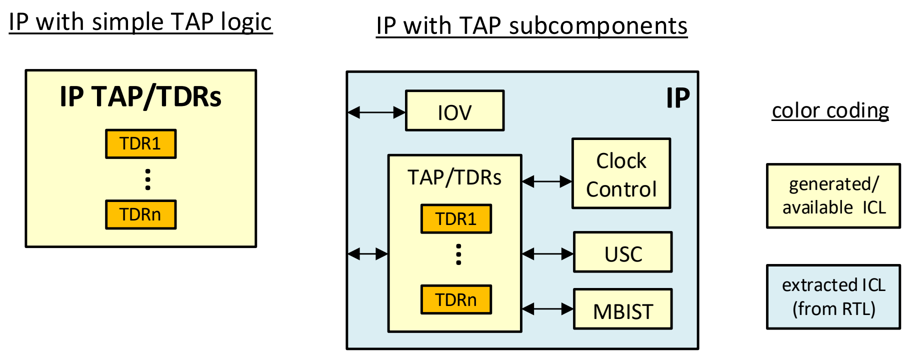
9.1.2 Generation of ICL from TAP RDL

9.1.2.1 IP ICL Creation Strategy

Depending on IP TAP logic type, two different approaches can be used to create IP ICL.

If IP has standard TAP logic and no external subcomponents with provided ICL, then complete IP ICL can be generated from IP TAP RDL using RDL2ICL flow (Figure 7).

Figure 7. Generation vs. Extraction of IP ICL



If IP has integrated external subcomponents with its own ICL (e.g., IOV, USC, MBIST), then ICL should be generated for the internal IP DFx logic wrapper, encapsulating IP-owned TAP logic/registers and providing required interfaces to the integrated subcomponents. In that case, top level IP ICL is created using ICL Extraction from RTL (by Mentor Tessent Shell).

If IP consists of multiple Corekits, then ICL should be created per Corekit, using the methods mentioned above.

9.1.2.2 Design and Collateral Examples

Refer to the provided examples in the $DUVE_M_HOME/icl_gen/examples/ folder for creating input collateral for ICL generation.

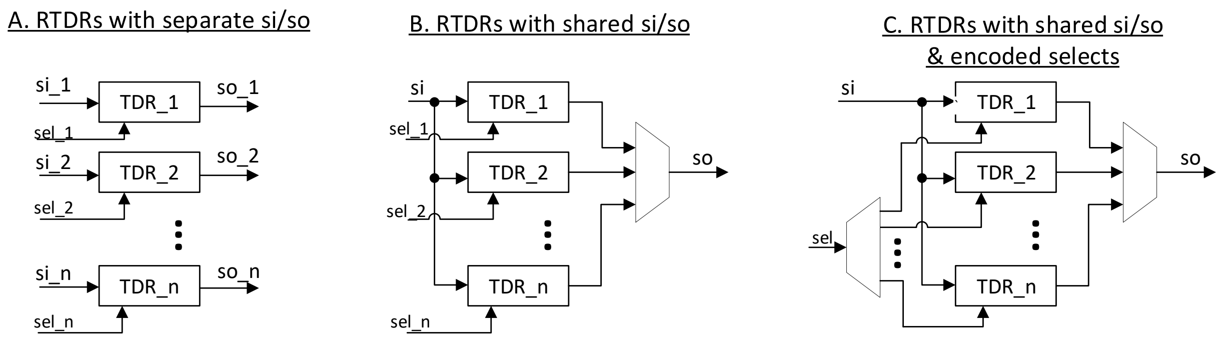
The following design configurations of IP are available for reference in the examples/ folder:

▪ IP with single TAP	: single_tap/
▪ IP with single TAP and parameterization in RDL/ICL	: single_tap_param/
▪ IP with no TAP, separate SI/SO ports of TDR registers	: rtdr_separate/
▪ IP with no TAP, separate ports and parameterization	: rtdr_separate_param/
▪ IP with no TAP, shared SI/SO ports of TDR registers	: rtdr_shared/
▪ IP with no TAP, shared SI/SO, encoded TDR selects	: rtdr_shared_encoded_sel/
▪ IP with no TAP, single IJTAG chain	: ijtag/
▪ IP with multiple TAPs and individual TDR registers	: multi_tap_rtdr/
▪ IP with TAPs, TDR registers and parameterization	: multi_tap_rtdr_param/
▪ IP with data in/out ports exported from TDR registers	: data_port_export/
▪ IP with TAPs/RTDRs and multiple power/reset domains	: multiple_tdr_resets/
▪ IP with TAP and blackboxed TDR	: tap_with_blackbox_tdr/
▪ IP with TAP and host RTDR interface (external RTDR)	: tap_with_rtdr_interface/
▪ IP with TAP and feature_en security interface	: tap_with_feature_en_security/

Each example includes TAP RDL file, ICL header file with IP port definitions, generated ICL and Tessent-generated pattern in the flattened PDL format.

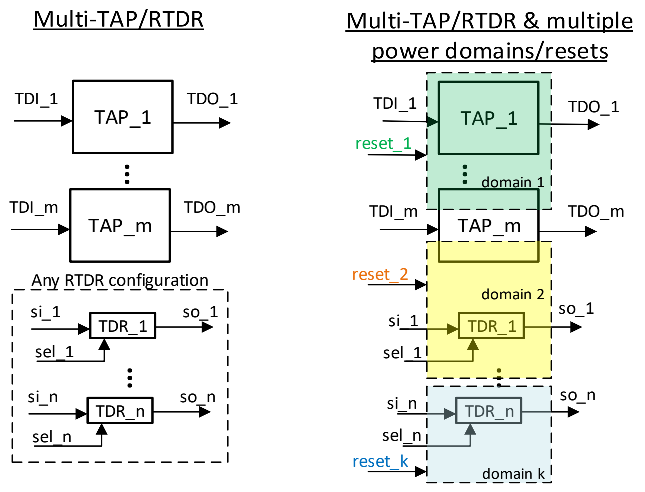
Supported design configurations with RTDRs are shown in Figure 8.

Figure 8. Supported RTDR-only IP Configurations



Design configurations with multiple TAPs and RTDRs are shown in Figure 9. It is possible to specify power domain/reset per TDR if required as illustrated on the second diagram.

Figure 9. Multi-TAP/RTDR Configurations and Multiple Power/Reset Domains

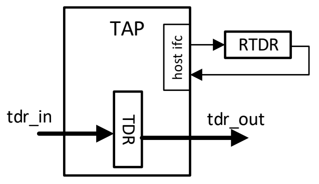


The IP designer can specify input ports, which are inputs of TDRs and/or output ports, driven by TDRs. It is referenced as "exporting" of data ports in the documentation and examples.

If the register, controlled by internal TAP, is outside of the current design level for which ICL is generated, the register can be marked as "external" RTDR by associating with the host scan interface in the ICL header file.

Exporting of TDR in/out and implementing of Host Scan interface are illustrated in Figure 10.

Figure 10. Export of TDR in/out Ports and Implementing Host Scan Interfaces



For all included examples, for the purpose of pattern generation, the following additional collateral was used:

- Design - Verilog module with port definitions only : examples/rtl
- Tessent flow and test (create_icl_verification_patterns) : examples/tessent

9.1.2.3 ICL Header File

ICL header file provides information about TAP ports - this data is not available in TAP RDL.

Refer to the header file examples in the $DUVE_M_HOME/icl_gen/examples/ folder.

The header files use *_hdr.icl pattern for the names.

The format of ICL header files is aligned with the ICL standard, defined in IEEE 1687.

Note: Name of ICL module in the header file must match name of the design module in RTL!

Note: Header file needs to include port specification only, the rest of the ICL constructs, if exists, will be ignored.

The following port-related information is consumed by the RDL2ICL flow:

- Port type
  - Example: ScanInPort
- Port name
  - Example: TdiT731L
- Port properties
  - Examples:
    - ActivePolarity 0;
    - DefaultLoadValue 32'h0a1;
- Port attributes
  - Examples:
    - Attribute intel_signal_type = "powergood";
    - Attribute intel_tdr_dest = "tap1.SLVIDCODE";

Note: Names of ICL ports needs to match the names of ports in RTL.

9.1.2.3.1 Port Properties

The properties, which can be used to provide information about behavior of design ports, are shown in Table 2.

Table 2. ICL Header File: Port Properties

Port Type	Property/Example	Description
ResetPort	ActivePolarity <value>;	Specifies active polarity of the reset port. Supported values: 0 | 1

ResetPort	Attribute intel_signal_type = "<reset_type>";
	Attribute intel_signal_type = "powergood";
	Attribute intel_signal_type = "powergood_d1";
	Attribute intel_signal_type = "tlr";
		Specifies reset type (identifier).
		Supported values: powergood, powergood_<id>, tlr
		The <id> indicates a subtype of the powergood reset. For example, it can be 'aon' or 'd1' (power domain1 reset). If matching tag is used for register reset specification (e.g., register property TapRegResetType = "PWRGOOD_D1"in the TAP RDL), then the corresponding reset port will drive the reset of that register in the generated ICL.

DataInPort	DefaultLoadValue 32'h0a1;	DUVE-M procedure 'tap_init_ports' uses that information to initialize input design ports.

DataInPort	Attribute intel_signal_type = "secure_policy";
	Attribute intel_signal_type = "secure_red";
	Attribute intel_signal_type = "secure_orange";
	Attribute intel_signal_type = "secure_green";
		Specifies port type as Intel Secure Policy or Intel Secure Feature Enable (green/orange/red). It will be used to implement modeling of Intel Security logic in the generated ICL. The attribute is required only if design implements the corresponding security logic in RTL.

DataInPort	Attribute intel_tdr_dest = "tap1.SLVIDCODE";	The attribute specifies parallel data input port as CaptureSource of the specified register or register field (field is defined using Alias to the bit range of the register). The provided instance path must exist in the RDL. Shown example connects data port with capture input of the SLVIDCODE register of TAP tap1.

Port Type	Property/Example	Description
DataOutPort	Attribute intel_tdr_dest = "regA.f1";	The attribute specifies parallel data port as output of the specified register or register field (field is defined using Alias to the bit range of the register). The provided instance path must exist in the RDL. Shown example connects data port with output of the field f1 of the register regA (the example is for IP with no TAP - TapType = "reg_only").

9.1.2.4 Linking TAP Data in RDL and ICL Header Files

In many cases ports and interfaces in the ICL header file should be linked with the corresponding registers or TAPs in the RDL spec.

It can be achieved using a few methods, described below.

9.1.2.4.1 TAP/RTDR with Individual Interface

The approach is used when TAP or RTDR has individual interface/ports on the IP boundary. It works for client ScanInterface of TAP and both client/host interfaces of RTDR (Figure 8-A, Figure 10). The mapping is done based on name of the ScanInterface in the ICL header file.

Client ScanInterface uses naming convention 'c_<tap_instance>' or 'c_<tdr_instance>', where <tap_instance> and <tdr_instance> are the instance names of the TAP/TDR objects in the RDL spec. Similarly, host RTDR ScanInterface uses naming convention 'h_<tdr_instance>'.

Note: There is no need to follow that approach if there is just a single TAP in the RDL and there is a single TAP ScanInterface in the ICL header file - the tool will automatically link them.

Note: In RDL, in the case of IP with individual RTDRs and multiple TAPs, the individual RTDRs will be specified in a separate addrmap with property TapType="reg_only" and that "reg_only" addrmap will be instantiated in the top level addrmap with the specific instance name. When mapping ScanInterfaces for these RTDRs, that instance name of the "reg_only" addrmap should be omitted and only register instance names are used to create ScanInterface names ("reg_only" addrmap does not create additional level of hierarchy in the generated ICL - these individual registers will appear in the top level module).

RDL and ICL header file examples of client-TAP and client-scan ScanInterface mapping are shown in Figure 11.

Figure 11. Individual Client-TAP and Client-scan ScanInterface Mapping

```
RDL - TAP instance:
ip_tap1_def tap1;

ICL header:
ScanInterface c_tap1 {
    Port TdiT731L;
    Port TdoT731H;
    Port TmsT731L;
}
```

```
RDL - TDR instance:
regA_def regA;

ICL header:
ScanInterface c_regA {
    Port si;
    Port so;
    Port sel;
    Port capture;
    Port shift;
    Port update;
}
```

9.1.2.4.2 RTDRs with Shared Interface

When working on IP with individual RTDRs (no TAP controller), designers can implement shared scan-in/scan-out ports (si/so) and/or use encoded selects to optimize IP interface.

Both cases are supported by the tool and can be specified in the ICL header file. The corresponding RDL and ICL header file examples are shown in Figure 12.

Figure 12. Mapping of RTDRs with Shared Interface and Encoded Selects

```
RDL - TDR instances:
regA_def regA;
regB_def regB;

ICL header - shared si/so:
SelectPort      sel_A {
   Attribute intel_tdr_dest = "regA";
}
SelectPort      sel_B {
   Attribute intel_tdr_dest = "regB";
}
ScanInterface c_ip {
   Port si_AB;
   Port so_AB;
   Port shift;
}
```

```
RDL - TDR instances:
regA_def regA;
regB_def regB;
regC_def regC;

ICL header - encoded selects:
DataInPort      sel_ABC[1:0] {
   RefEnum IP_REGISTERS;
}
ScanInterface c_ip {
   Port si_ABC;
   Port so_ABC;
   Port shift;
}
Enum IP_REGISTERS {
   regA    = 'b01;
   regB    = 'b10;
   regC    = 'b11;
}
```

In the first case of shared si/so ports, individual RTDR selects are marked by special attribute which links the selects with the corresponding register instances in the RDL. The required attribute syntax is 'Attribute intel_tdr_dest = "<reg_instance>"'.

Note: Individual selects cannot be included into the shared ScanInterface! Shift Enable signal is required to indicate the scan type of the interface.

In the second example, RTDR select bus is encoded and IP has a decoding logic to produce one-hot selects for individual registers. In the ICL header file, bus encoding is specified using enumeration which is referenced in the RefEnum statement of the select port declaration.

Note: Encoded select bus port should be defined using DataInPort type.

9.1.2.4.3 Exporting of Parallel Inputs and/or Outputs of TDRs

There are the situations when ICL spec should have information about IP parallel data input ports which values are captured by TDRs or IP parallel output ports which are driven by TDRs.

For input ports, captured by TDRs, typical examples include:

- Straps (the ports which will be tied to specific constant values when IP is integrated, for example, SLVIDCODE port).
- Chip level input ports (for example, A0 debug strap).
- Ports, expected to be driven by external TDRs if it is desirable to model that behavior in ICL (otherwise, it would be modeled as unknow/X).

For output ports, driven by TDRs, typical examples include:

- Ports which control TAP network topology.

- Ports which drive inputs of other TDRs.
- Ports which drive chip level output ports.

The diagram of parallel TDR in/out export is shown in Figure 10. RDL/ICL header file examples are shown in Figure 13.

Figure 13. Parallel TDR In/Out Export

```
RDL - Single TAP
slvid_def SLVIDCODE;// 32b TDR
regA_def  regA; // TDR with 4b field 'en'

ICL header:
DataInPort   ftap_slvidcode[31:0] {
   Attribute intel_tdr_dest = "SLVIDCODE";
   DefaultLoadValue 32'h0a1;
}
DataOutPort  enable_out[3:0] {
   Attribute intel_tdr_dest = "regA.en";
}
```

```
RDL - IP with multiple TAP's
tap1_def tap1; // With SLVIDCODE and regA

ICL header:
DataInPort   ftap_slvidcode_1[31:0] {
   Attribute intel_tdr_dest = "tap1.SLVIDCODE";
   DefaultLoadValue 32'h0a1;
}
DataOutPort  enable_out_1[3:0] {
   Attribute intel_tdr_dest = "tap1.regA.en";
}
```

TDR port export in the ICL header file is specified by adding Attribute **intel_tdr_dest** into the definition of DataInPort or DataOutPort objects. The provided attribute value should reflect a correct path to the targeted register instance or register field instance in RDL.

Supported path formats (depending on the RDL structure and target instance):

- "<tap_inst>.<reg_inst>"
- "<tap_inst>.<reg_inst>.<field_inst>"
- "<reg_inst>"
- "<reg_inst>.<field_inst>"

9.1.2.4.4 TAP Security Interfaces

The flow supports two types of Intel Security interfaces:

- DFx Secure Policy interface (4-bit secure bus)
- Decoded GREEN/ORANGE/RED interface (3-bit, based on feature_en outputs of the DFx Secure Plugin)

Both types of interfaces can be specified in the ICL header file as shown in Figure 14.

Figure 14. ICL Header File--Security Interfaces

```
DataInPort  fdfx_secure_policy[3:0] {
   Attribute intel_signal_type = "secure_policy";
}
DataInPort  feature_en[2:2] {
   Attribute intel_signal_type = "secure_red";
}
DataInPort  feature_en[1:1] {
   Attribute intel_signal_type = "secure_orange";
}
DataInPort  feature_en[0:0] {
   Attribute intel_signal_type = "secure_green";
}
```

Note: For 3rd party TAP, which has no Intel TAP Security, **no register** level security properties must be specified in the RDL (**UDP TapOpcodeSecurityLevel**)! Only Intel TAPs with implemented Security logic must have these attributes!

Note: 3rd party TAP, which has no Intel TAP Security, **can** have TAP level security specified in RDL (**UDP TapSecurityLevel**). The attribute will be used to check a security level, assigned to the TAP when it is integrated into the TAP network.

9.1.2.4.5 TAP/RTDR Resets

The tool provides full flexibility in definition of resets for TAP registers:

- Powergood and TRST/TLR (asynchronous/synchronous TAP resets) reset types are supported

Note: Only single reset type can be specified per register.

Note: Specifying of TLR reset means including of powergood reset for the power domain of TAP logic and TRST reset

- Multiple powergood resets can be specified per IP, targeting different TAP registers

In RDL, TDR reset is specified using UDP **TapRegResetType**. The recommended values to use:

  - "PWRGOOD"
  - "PWRGOOD_<id>"
  - "TLR"

Legacy value "PWRGOOD, TRST, TLR" is treated as "TLR".

Tag <id> represents the unique power domain ID, which is used to link register reset with the corresponding reset port in the ICL header file. Tag can use any alpha-numeric combinations, for example 'aon' or 'd1'. The tag is case insensitive.

Although the tool will try to auto-map resets if no additional information is provided, it is always better to explicitly specify reset types in the ICL header files with the Attribute **intel_signal_type**. To map reset ports for different power domains, use the same tags in the RDL and ICL header file. The examples are shown in Figure 15.

Figure 15. TAP/TDR Reset Mapping

```
RDL:
TapRegResetType = "PWRGOOD";
TapRegResetType = "PWRGOOD_D1";
TapRegResetType = "TLR";

ICL header:
ResetPort   fdfx_powergood {
   ActivePolarity 0;
   Attribute intel_signal_type = "powergood";
}
ResetPort   fdfx_powergood_d1 {
   ActivePolarity 0 ;
   Attribute intel_signal_type = "powergood_d1";
}
ResetPort   ijtag_reset_b  {
   ActivePolarity 0;
   Attribute intel_TapRegResetType = "tlr";
}
```

9.1.2.4.6 Host Interface for External RTDR

There are the situations where TAP register is located outside of the IP with TAP controller. In this case, ICL needs a host scan ScanInterface to connect to that external RTDR.

To specify external RTDR, the corresponding host ScanInterface should be added into the ICL header file and linked with the register in the RDL based on ScanInterface name (naming convention: 'h_<reg_inst>') as shown in Figure 16.

Figure 16. Host RTDR Interface

```
RDL:
regA_def  regA;

ICL header:
ScanInterface h_regA {
    Port regA_si;
    Port regA_so;
    Port regA_sel;
    Port regA_capture;
    Port regA_shift;
    Port regA_update;
}
```

Note: If register is linked with host scan interface, then a complete definition of that register in RDL is ignored. Only "integration" properties like opcode security (UDP TapOpcodeSecurityLevel) or documentation properties (for example, instance level attribute "desc") will be used for ICL generation.

9.1.2.5 TAP RDL - Special Situations

9.1.2.5.1 Blackboxed TDR

Some designs can have TDRs with complex access protocol when TDR chain length cannot be modeled using ICL. Such registers should be blackboxed in ICL and then iScan PDL instruction with explicit bit streams/sizes should be used to create validation and test content.

The tool will blackbox the register if the register has property/value **'TapDrIsFixedSize = false'** (by default, the value of that attribute is 'true'). When blackboxing, most of register spec will be ignored (for example, it will be no information kept about fields and associated properties - access type, description, etc.). ICL generation will use properties for opcode value and security when integrating the register and size/reset values to specify DefaultLoadValue of the ScanInterface of the register Module in ICL.

9.1.2.5.2 IJTAG Chain

IJTAG chain in RDL should be defined using Intel TAP RDL 2.0 specification. ICL generation considers **TapRegType** and **TapSibRef** properties in RDL to detect and construct IJTAG chains.

The example of IJTAG chain definition in RDL is shown in Figure 17.

Figure 17. Definition of TAP with IJTAG Chain in RDL

```
addrmap ip { // IP with TAP
   …
   reg regA_def {
     …
   };
   reg regB_def {
     …
   };
   reg sib_reg_def {
      TapRegType = "SIB";
      …
   };

   //---------------------------
   // IJTAG chain definition
   // IMPORTANT: TDR's and SIB's must be instantiated in the following order:
   //    from TDO/lsb (top) to TDI/msb (bottom)
   // Flat chain layout:
   //    TDI -> regB -> sibB -> regA -> sibA -> sib_top -> TDO

   regfile ijtag_chain_def {
      TapRegType = "IJTAG";
      TapOpcodeSecurityLevel = "SECURE_RED";
      …
      sib_reg_def sib_top;
      sib_reg_def sibA;
         sibA -> TapSibRef = "sib_top";
      regA_def regA;
         regA -> TapSibRef = "sibA";
      sib_reg_def sibB;
         sibB -> TapSibRef = "sib_top";
      regB_def regB;
         regB -> TapSibRef = "sibB";
   };

   // Register/chain instantiations
   ijtag_chain_def chain_1;
      chain_1 -> RegOpcode = 0x30;
};
```

TapSibRef values link the register or SIB, which they are assigned to, with the corresponding parent SIB's.

9.1.2.5.3 Parameterized TAP RDL

Some IPs have parameterized TAP RTL/RDL, so IP TAP configuration can be changed when the IP is integrated into SoC or Super-IP.

If IP module is instantiated multiple times with different parameter values in RTL, then the corresponding IP ICL with matching parameters is necessary.

There are two approaches to address that need.

If parameterization is required for TAP register/filed sizes only, meaning that sets of registers/fields and their interfaces are always the same, then the parameterized ICL can be generated from the parameterized RDL using RDL2ICL flow with added '-icl_param' options to specify the RDL parameters which should be kept in the generated ICL.

Note: Parameters in RTL and TAP RDL must have the same names

Note: Parameterization in ICL is supported for register/field size parameters ONLY!

Note: Register/field sizes and field LSB/MSB positions should be linear functions of the parameters (e.g., a0 + a1*$PARAM1 + a2*$PARAM2)

Note: Parameterization of numeric value of reset is not supported, only width of reset value can be parameterized. It means that numeric value of reset should be the same across all applicable parameter values.

A typical example of supported parameterized TAP RDL is shown in Figure 18. Embedded Perl code includes:

- Variables/parameters $F1_WIDTH and $F2_WIDTH for width of fields f1 and f2
- calc_regwidth() function to calculate value of regwidth properties in RDL
- TapShiftRegLength and TapTotalNumRegBits UDP with formulas for total register width
- Field width specification (two examples: for [<msb>:<lsb>] and [<width>] styles)
- Parameterization of reset value width (e.g., <%=$F1_WIDTH%>'h0)

Figure 18. Parameterized TAP RDL Example

```
<%
  # Default parameter values - should be overridden based on the real values used
  $F1_WIDTH  = 1;
  $F2_WIDTH  = 4;
%>
<%
sub calc_regwidth {
  $dr_size = shift;
  return (($dr_size > 8) ? (2**length(sprintf("%b",$dr_size - 1))) : 8 );
}
%>

addrmap ip {
   …
   reg regA_def {
      name = "regA register";
      …
      TapShiftRegLength    = <%=$F1_WIDTH+$F2_WIDTH%>;
      TapTotalNumRegBits   = <%=$F1_WIDTH+$F2_WIDTH%>;
      regwidth             = <%=calc_regwidth($F1_WIDTH+$F2_WIDTH)%>;

      field {
         name = "f1 field";
         …
      } f1 [<%=$F1_WIDTH-1%>:0] = <%=$F1_WIDTH%>'h0; // [msb:lsb] spec style

      field {
         name = "f2 field";
         …
      } f2 [<%=$F2_WIDTH%>] = <%=$F2_WIDTH%>'h0;     // [width] spec style

   }; // end of reg regA_def
};
```

Collateral examples for design with single TAP, design with separate RTDRs, and design with multiple TAPs/RTDRs can be found in the $DUVE_M_HOME/icl_gen/examples/*_param/ folder.

If required IP parameterization controls availability of registers/fields or access interfaces, changes integration properties of TDRs (opcode, security, resets), or specific type of parameterization is not supported by the current implementation of RDL2ICL flow, then the simplest solution is to integrate the IP in RTL using non-parameterized uniquified wrappers, instantiating of IP with the necessary parameter values and provided with the corresponding ICL (created using generation or extraction flows).

9.1.2.6 Running TAP ICLGEN Script

The script is located in the $DUVE_M_HOME/icl_gen/ folder.

This is a typical command line to generate ICL from TAP RDL:

```
> tap_icl_gen.pl -inp <ip>.rdl -icl -hdr <ip>_hdr.icl -out_dir <dir> > <log>
```

Description of the available command line options is provided in Table 3.

Table 3. Command Line Arguments of tap_icl_gen.pl

Option	Type	Description
-input <file>.rdl	Required	Input RDL file
-icl	Required	To generate ICL output
-hdr <file>.rdl	Required	ICL header file
-rdl	Optional	To generate RDL output
-out_dir <dir>	Optional	Output directory. By default, output will be directed to the ./output folder.
-out_file <file>	Optional	Output file name (without file extension). By default, ICL file name will be based on module name in the ICL header file.
-I <path>	Optional	Search path for included RDL files (if RDL has `include statements for design collateral). Inclusion of the tap_udp.rdl file is not considered (the flow uses its own UDP file).
-param "name=value"	Optional	User-specified RDL parameter override. Can be used multiple times as needed, e.g., -param "IR_SIZE=8" -param "CHAIN_NUM=4".
-icl_param <name>	Optional	RDL parameter which should be kept as a parameter in ICL. Can be used multiple times as needed, e.g., -icl_param PWIDTH -icl_param CHAIN_NUM. Only parameters for register width and field width are supported.
-prefix <string>	Optional	Uniquification prefix for design component definitions in ICL (if additional uniquification is required)
-suffix <string>	Optional	Uniquification suffix for design component definitions in ICL (if additional uniquification is required). For example, it can be IP version.

9.2 ICLGEN Front-End

The feature has Beta status.

10 Reference

10.1 DUVE-M Quick Reference

Steps

1. Download DUVE-M VIP from IPX.
2. Create local copies of the DUVE-M configuration files (from the $DUVE_M_HOME/verif/cfg) and customize them according to the IP implementation and requirements.
3. Use provided dofile (or its customized version) with Tessent Shell tool to generate tests (Verilog TB).
4. Simulate generated tests.
5. Debug any test mismatches.

10.2 TAP RDL2ICL Quick Reference

Steps

1. Create IP RDL (if it does not exist).
2. Create ICL header file.
3. Run tap_icl_gen.pl to generate IP ICL.
4. Use DUVE-M to validate ICL versus RTL.

11 Troubleshooting

11.1 Errors - DUVE-M/Tessent

Table 4. Typical Tessent Errors

Tessent Error	Description / How to resolve
// Error: Register '<…>' is not accessible. The iWrite is not scheduled.	Reason: Due to some issue with design ICL, the targeted register is not accessible for iWrite operation. To resolve: Design ICL must be fixed.

// Error: Unable to find a solution for the current PDL constraints.
// Note: The predefined limit of iApply scan loads and/or parallel I/O operations (64) has been reached while resolving the current PDL constraints.
// Note: The following commands were part of the last iApply and are now removed from schedule:
//
// iWrite <…> 0b1
	Reason: due to some issue with design ICL or test configuration/flow, Tessent solver cannot find a sequence to access target register (which is reported in the end of the error message).
	To resolve: Review design ICL and test flow to understand the issue and fix a problematic collateral (ICL, test, test configuration).

// Error: 'sib' is not a known icl_module object.	Reason: Test configuration (e.g., ip_test_cfg.do) or test reference an object which does not exist in the design ICL ('sib' module in the given example). To resolve: Eliminate reference to the non-existing object.

11.2 Errors and Warnings - TAP RDL/ICL (ICLGEN)

Table 5. Typical TAP RDL2ICL (ICLGEN) Errors

Error	Description / How to resolve
RDL related error	Fix input RDL
ICL header related error	Fix ICL header file

Table 6. Typical TAP RDL2ICL (ICLGEN) Warnings

Warning	Description / How to resolve
-W- Input RDL not loaded feature-specific udp.rdl file, using tap_udp.rdl by default	IP RDL has no include of tap_udp.rdl. For information only, no changes are required - tool uses its own UDP definition file.
-W- Mismatch between RDL definition name and module name in ICL header file - using <…> from the ICL file	Name of top addrmap definition in RDL is different than the name provided for top ICL in the ICL header file. For information only, no changes are required.

11.3 Debug Support

11.4 FAQs

Placeholder

11.5 Special Considerations