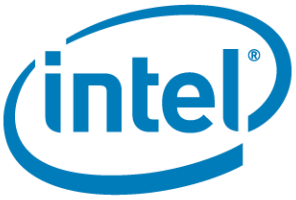


Slave TAP (sTAP)

High-Level Architecture Specification

IP Rev. 2020WW05-PIC6-V1
January 2020

Intel Top Secret

This document contains information on products in the design phase of development.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your Intel account manager or distributor to obtain the latest specifications and before placing your product order.

Copies of documents that have an order number and are referenced in this document or in other Intel literature can be obtained from your Intel account manager or distributor.

Contents

List of Figures

List of Tables

1 About this Document

1.1 Audience

The information in this document is intended for an SoC design team that is using this IP.

1.2 References

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

Document Name	Link / Location
sTAP Integration Guide	Release directory
sTAP Verification Plan Reference	Release directory
sTAP Product Brief	Release directory
sTAP Release Notes	Release directory
HTAP HAS	https://sharepoint.amr.ith.intel.com/sites/docs_dft1/TAP/TAP_PDFs/HTAP_HAS.pdf
TAPLink HAS	https://sharepoint.amr.ith.intel.com/sites/docs_dft1/Tap_Link/Tap_LinkWork/DTEG_TapLink_HAS.doc
IEEE 1149.1 Standard	https://standards.ieee.org/standard/1149_1-2013.html

1.3 Contact Information

Function	Name	Email
Architecture	Oz Yanir	yanir.oz@intel.com
Verification	Bulusu, Shivaprashant	shivaprashant.bulusu@intel.com
Design	Pemula, Latha	latha.pemula@intel.com
Doc Template Owner	Susann Flowers	susann.flowers@intel.com

1.4 Terminology

Term	Definition
CLTAPC	Chip Level TAP Controller
DFT	Design For Test
DR	Data Register
DUT	Design Under Test
Env	Validation/Test Environment
FPV	Formal Property Verification
FSM	Finite State Machine
FUB	Functional Unit Block
GLS	Gate Level Simulation
IEEE	Institute of Electrical and Electronics Engineers

Term	Definition
IP	Intellectual Property
IR	Instruction Register
MDA	Multi Dimensional Array
mTAP	Master TAP
OVM	Open Verification Methodology
RTL	Register Transfer Level
rTDR	
SoC	System on Chip
sTAP	Slave TAP
SV	SystemVerilog
SVA	SystemVerilog Assertions
TAP	Test Access Port
TAPC	TAP Controller
TAPNW	0.7 TAP network
TB	Test Bench
TLM	Transaction Level Modeling
WSP	Wrapper Serial Port
WTAP	Wrapper TAP
WTAPNW	WTAP network

1.5 Document Revision History

Author	Revision No.	Revision Description	Revision Date
	0.2		dd Month yy
	0.5		dd Month yy
	0.6		dd Month yy
	0.7		dd Month yy
	0.8		dd Month yy
	0.9		dd Month yy
	1.0		dd Month yy
	1.0vx		dd Month yy

1.6 IP Revision History

Author	Revision No.	Description *Do not remove or change the text that is here.* *Add rows or text to describe your IP revision*	Corresponding RTL	Revision Date Required
	Rev 0.1	Initial draft Must use current standard HAS content template	N/A	dd Month yy

Author	Revision No.	Description *Do not remove or change the text that is here.* *Add rows or text to describe your IP revision*	Corresponding RTL	Revision Date Required
	Rev 0.2	HAS TR: Identify list of product features for POR and their priority for implementation, and critical architectural changes from previous products Provide "Enough" information for other groups to make judgments about resourcing, timelines, and work efforts Deltas in expected changes; Boot Flow and Interfaces defined	N/A	dd Month yy
	Rev 0.5	Enable all features for Rev 0.5 RTL; full definition of an IP block including performance capabilities, allowing FED to begin Outlines an high-level description of power, signals, interfaces, and security Full Review Prior to Rev 0.5 complete DFx features/requirements are complete Summary outline available for RAS/SER & PerfMon Enables Rev 0.5 RTL Change Control start at Rev. 0.5; no new features added; Issues and BugECOs managed in HSD-ES	RTL0.5	dd Month yy
	Rev 0.8	Spec Complete to enable Rev 0.8 RTL PCR Control (Product Change Request - IP Only) RAS/SER & PerfMon features detailed and specified	RTL0.8	dd Month yy
	Rev 1.0	DCCB Control prior to TO, RTL Rev 0.8 (Post review) + Rev 0.8 HAS + Issues from HSD-ES + Full Review ECO Control (Engineering Change Order)	RTL1.0	dd Month yy
	Rev 1.1	HAS Complete		dd Month yy
	Rev 1.x	HAS Updates		dd Month yy

1.7 Firmware Version

Not applicable to this IP

2 STAP2.0 Introduction

This is a feature HAS, which for IPs describes what the feature is and how it works. It is to be consumed primarily by the design and validation teams.

2.1 IP Description

STAP2.0 is the successor of the legacy IRR STAP component and supports all the basic functionality of the legacy STAP IRR version with additional advanced capabilities that can only be achieved when using the auto-generated RTL methods. STAP2.0 is compliant with IEEE 1149.1.

Note: STAP1.0 is a parametrized RTL component but STAP2.0 will be an auto-generated RTL.

STAP2.0 is being developed for HTAP SoCs. The RTL for STAP2.0 is auto-generated using DFT Build.

2.1.1 I/O Pins

The TAP logic is accessed serially through five dedicated pins on each component as shown in Table 1.

Table 1. TAP Interface Signals

Pin name	Definition	Direction	Description
TCK	Test Clock	input	
TMS	Test Mode Select	input	This signal controls the transitions of the TAP FSM
TDI	Test Data Input	input	The serial input for test instruction and data
TRSTB	Test Reset input	input	'B' means active at low
TDO	Test Data Output	output	The serial output for test data

TMS, TDI, and TDO operate synchronously with TCK (which is independent of all other functional clocks in the system) and TRSTB is an asynchronous reset input signal.

2.1.2 TDO Control

All TAPs are compliant to IEEE1149.1 where the TDO changes on the falling edge of TCK.

2.1.3 TDO Output

The Least Significant Bit (LSB) of any TDR register is shifted out on TDO first. Also, by corollary, the LSB from the TAP host controller will shift into TDI of the CLTAP and through connectivity within the network the LSB will be shifted into the TDI of any slave TAP.

2.1.4 TDO Enable

A TDO enable signal is available to control a general-purpose IO pad, if necessary, to be compliant to the 1149.1 spec. According to the spec, the TDO package pin is only driven when shifting the DR or IR registers otherwise it is tri-stated. Therefore, this implies an open drain driven configuration.

SOCs can use two types of TDO IO buffers:

- Open drain IO buffer that drives the IO pin to 0 when TDO=0 and drives HighZ when TDO=1.
- CMOS IO buffer that drives the IO out pin to its data pin when the enable signal is set and to HighZ when enable signal is clear.

CLTAP TDO is design both for CMOS IO and Open drain IO buffers, where the TDO is driving the Data pin and tdoen signal is driving the Enable signal. If user wishes to use Open drain buffer, he needs to use the tdoen signal too and for CMOS IO buffer he need to use the TDO signal only.

Figure 1.

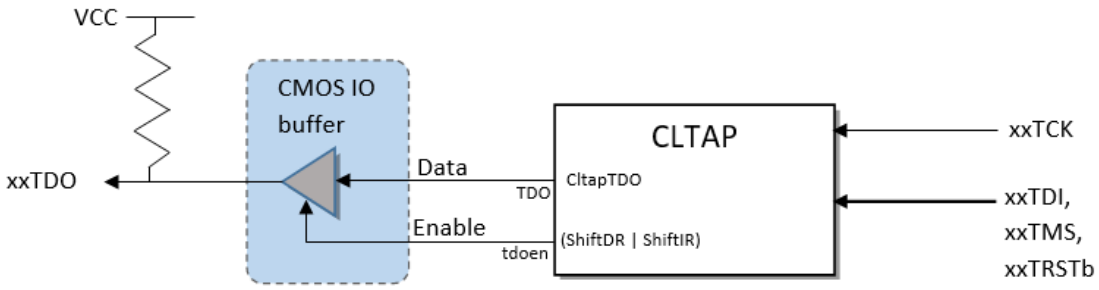
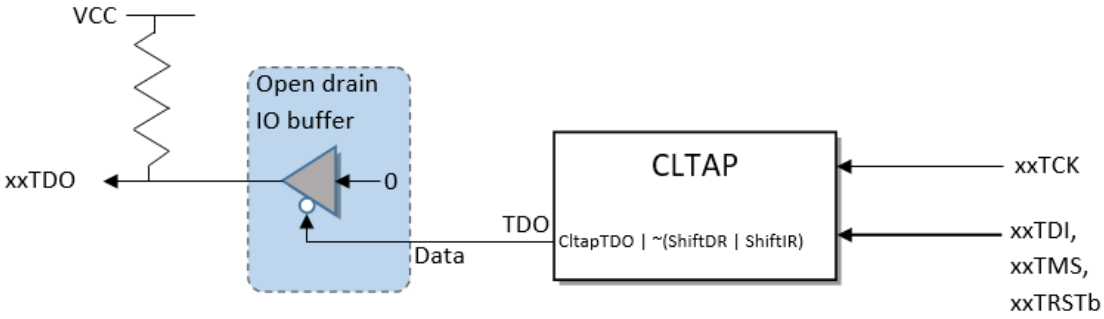


Figure 2.



2.1.5 Building Blocks

The IEEE 1149.1 TAP is built from the TAP Finite State Machine (FSM), a single instruction register (IR), and N number of data registers (DRs).

Figure 3 shows a schematic of the TAP building blocks. The TAP FSM consumes the TMS signal and generates the TAP FSM state control signals:

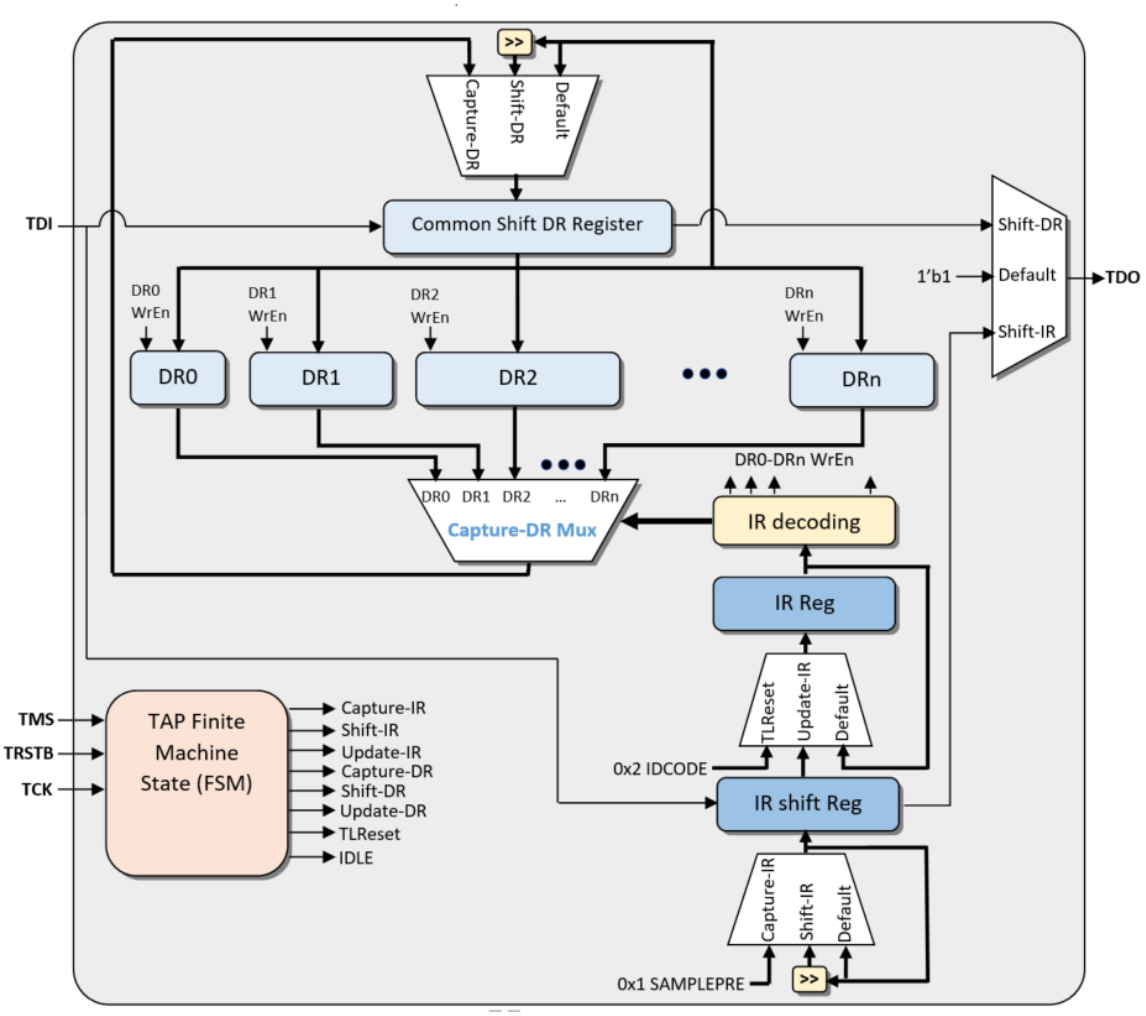
- TLRESET
- IDLE
- Capture-IR/DR
- Shift-IR/DR

- Update-IR/DR

The IR decoding logic generates the write enable (WrEn) signal to each DR based on the IR opcode. WrEn selects the DR to be updated when the TAP FSM is in the update-DR state.

In addition, the IR decoding logic selects the DR output to be sampled by the common shift DR register at capture-DR state. The common shift DR register is a variable length shift register, used for shifting TDI to TDO. The size of the common shift register is determined by the IR opcode, which matches the selected DR size. The capture-DR MUX selects the DR to be captured into the common shift DR register, based on the IR opcode.

Figure 3. TAP Building Blocks



2.1.6 TAP Finite State Machine (TAP FSM)

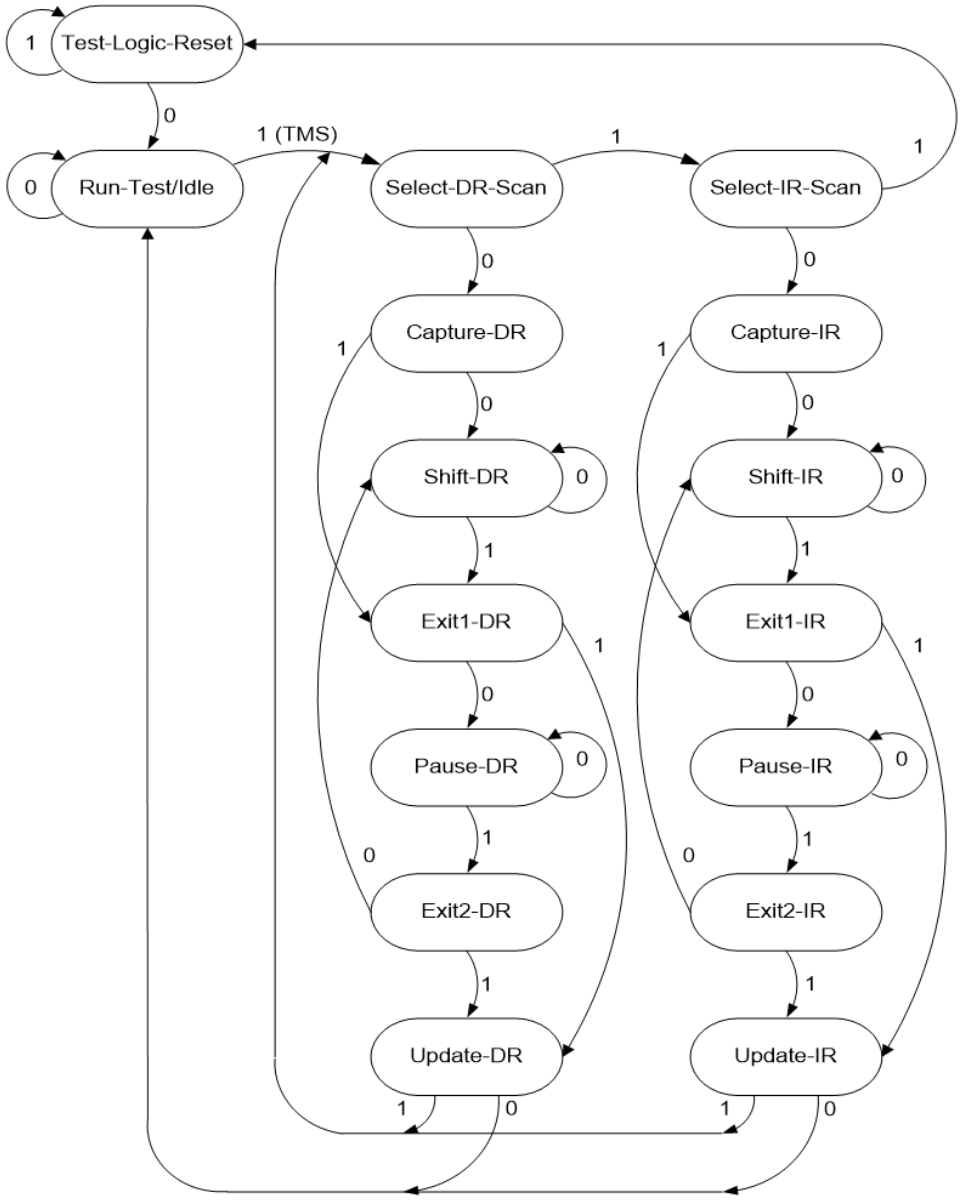
The TAP FSM, shown in Figure 4, contains a reset state, a run-test/idle state, and two major branches for IR and DR. These branches allow access to either the TAP Instruction Register (TAP IR) or to one of the DRs implemented inside or outside the TAP, internal TAP DR (ITDR) or remote TAP DR (RTDR), respectively.

The value of the TMS signal (at the falling edge of TCK) controls the state transition of the FSM. The labels on the transition arcs enable the corresponding transition. TAP instructions

and test data are loaded serially into the IR register and DR register, respectively, in the Shift-IR state and Shift-DR state from the TDI pin.

Normally, the TAP instruction is selected first through the IR branch. If a DR is associated with that instruction, it will automatically be connected between TDI and TDO when the DR branch of the state machine is traversed.

Figure 4. TAP FSM



2.1.6.1 State Descriptions

The TAP controller is a 16-state FSM. Descriptions of the TAP controller states are given below.

- **Test-Logic-Reset:**

  In this state, the test logic is disabled so that normal operation of the SoC can continue unhindered. The TAP Instruction Register is forced to IDCODE. The controller enters the

Test-Logic-Reset state when the TMS input signal is held active for at least five clocks, or when the asynchronous TRST# is pulled active at SoC power up. When TRST# is pulled active (low) the controller enters this state immediately no matter what the state of the controller is, and the controller is held in this state if TRST# is held active.

- **Run-Test/Idle:**

  This is the idle state of the TAP controller. In this state, the contents of all test DRs retain their previous values.

- **Select-IR-Scan:**

  This is a temporary controller state. All registers retain their previous values.

- **Capture-IR:**

  In this state, the primary IR loads a fixed value (of which the two least significant bits are "01") on the rising edge of TCK. The parallel, latched output of the IR ("current instruction") does not change in this state.

- **Shift-IR:**

  The primary IR is connected between TDI and TDO, where TDI is connected to the MSB and TDO is connected to the LSB. On each rising edge of TCK the primary instruction register is shifted one stage right (MSB → LSB). The output TDO is valid at the falling edge of TCK. The current instruction does not change in this state.

- **Exit1-IR:**

  This is a temporary state. The current instruction does not change in this state.

- **Pause-IR:**

  IR shifting is temporarily halted. The current instruction does not change in this state (no Update IR).

- **Exit2-IR:**

  This is a temporary state. The current instruction does not change in this state.

- **Update-IR:**

  The instruction which has been shifted into the primary IR is latched onto the shadow IR (IR parallel output) on the falling edge of TCK. Once the new instruction has been latched, it remains the current instruction until the next Update-IR (or until the TAP controller state machine goes to reset).

- **Select-DR-Scan:**

  This is a temporary controller state. All registers retain their previous values.

- **Capture-DR:**

  In this state, the DR selected by the current instruction may capture data at its parallel inputs.

- **Shift-DR:**

  The DR connected between TDI and TDO as a result of selection by the current instruction is shifted one stage toward its serial output on each rising edge of TCK. The output arrives at TDO on the falling edge of TCK. The parallel, latched output of the selected DR does not change while new data is being shifted in.

- **Exit1-DR:**

  This is a temporary state. All registers retain their previous values.

- **Pause-DR:**

  When pause shifting the select DR without stopping TCK, a user can return to Shift-DR and continue the shifting. All registers retain their previous values.

- **Exit2-DR:**

  This is a temporary state. All registers retain their previous values.

- **Update-DR:**

  Data from the shift register path is loaded into the latched parallel outputs of the selected DR (if applicable) on the raising edge of TCK.

  Note: Update-DR and Test-Logic-Reset are the only states in which the latched paralleled data output can be changed.

2.1.7 Instruction Register (IR)

The TAP IR holds the current instruction. The TAP IR contains the address of the DR that a user wants to access. The TAP Instruction Register (TAP IR), shown in Figure 5, conforms to the IEEE 1149.1 specification and consists of a pair of registers:

- **Shift Register**

  The Shift Register, also known as the Primary Register, connects between TDI and TDO of the TAP controller.

- **Actual IR**

  The Actual IR, also called the Shadow Register, has a parallel load path from the Shift Register. The parallel output of the Actual IR feeds into the TAP instruction decoder.

Note: The size of the TAP IR is greater than or equal to 2.

Figure 5. 8-bit TAP IR

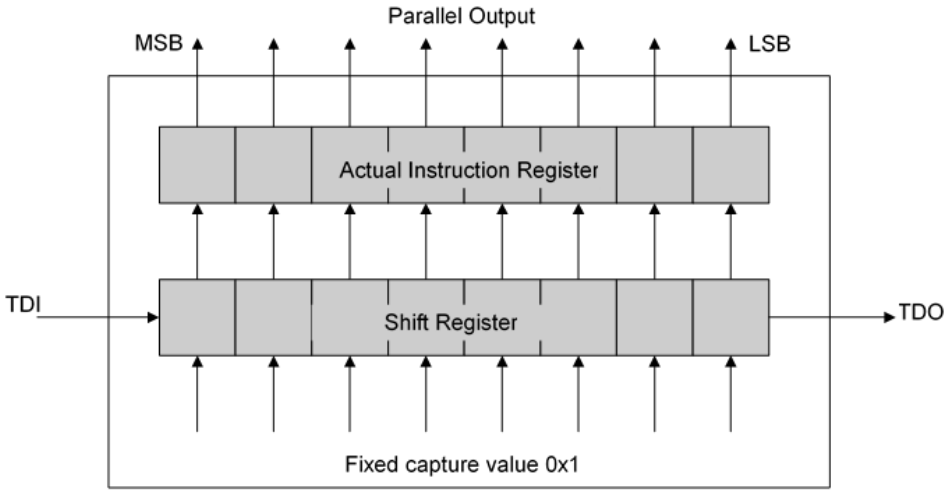


Figure 6 explains the operation of the TAP IR during the Capture-IR, Shift-IR, and Update-IR states of the TAP FSM. The shaded flops in those figures are the flops that are updated for the corresponding operation.

- **Capture-IR**

  In Capture-IR, the Shift Register is loaded in parallel with the fixed value "0…01b".

- **Shift-IR**

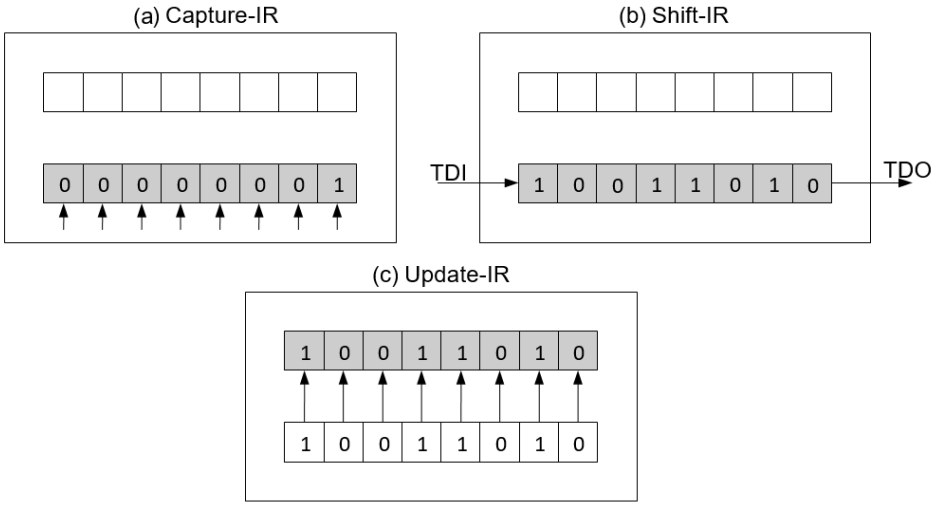
  In Shift-IR, the Shift Register forms a serial data path between TDI and TDO.

- **Update-IR**

  In Update-IR, the contents of the Shift Register are transferred in parallel into the Actual IR.

  Note: The only time the outputs of the Actual IR change is when the TAP FSM enters the Update-IR state.

Figure 6. Operation of the TAP IR



2.1.8 Data Register (TAP DR)

The TAP DR architecture is the same as the TAP IR architecture. Components supporting either the "capture" or "update" functionality are removed from the basic structure if the DR is read only or write only. DRs are similarly accessed as the IR but only using the "select-DR-scan" branch of the TAP FSM. A specific data register (or the default Bypass register) is selected prior to its use by loading the IR with the appropriate opcode.

Note: DR contents can change for the following TAP controller states: Capture-DR, Shift-DR, Update-DR, and Run-Test/Idle.

The TAP DR is implemented as two registers:

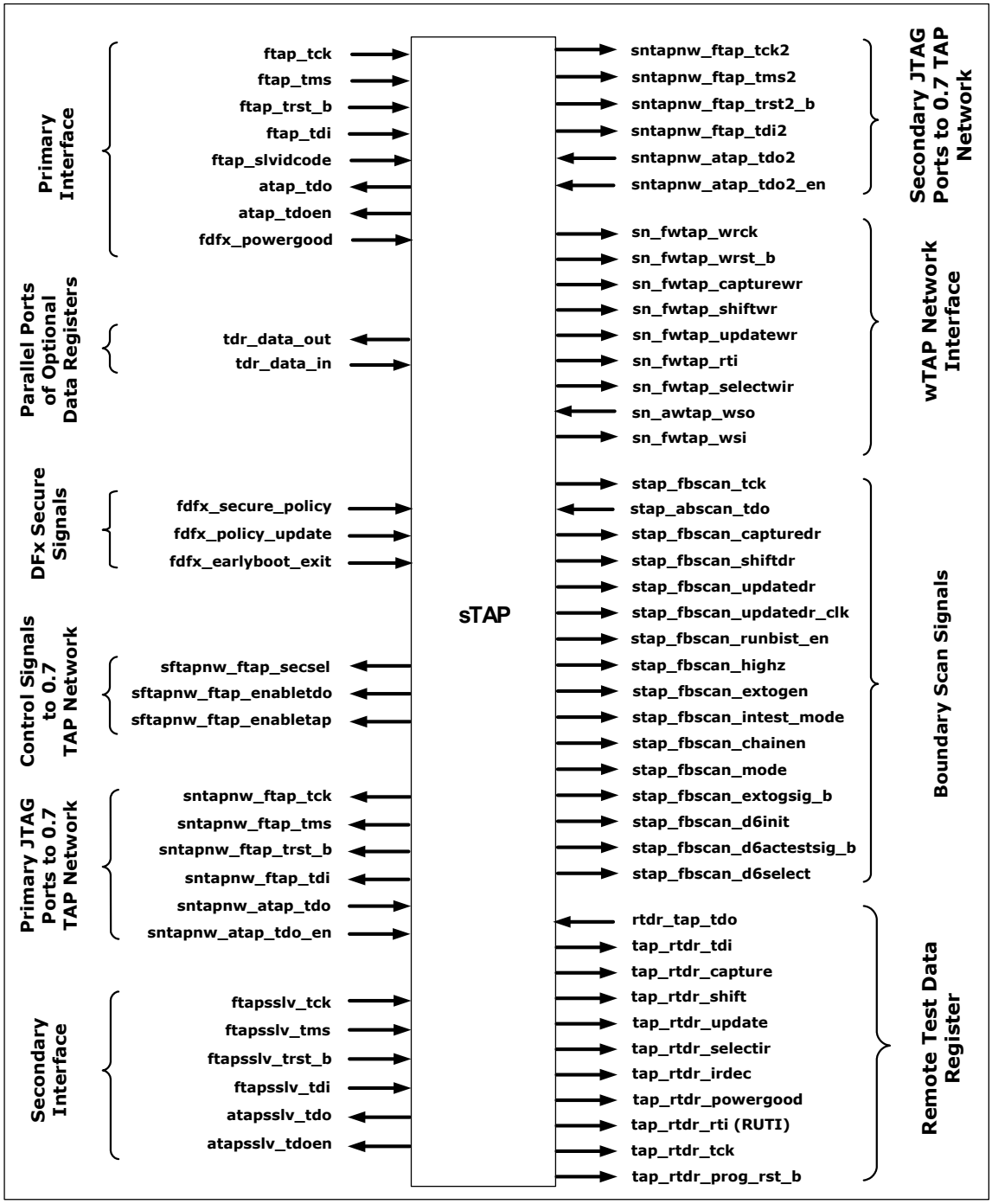
- **Primary Register**

  The primary register, also known as the Shift Register, is connected directly to the shift chain. In the capture-DR state, the contents of the shadow register are transferred to the primary register. The primary register is shifted out of TDO during shift-DR.

  Shadow Register (the real register)

  In the Update-DR state the contents of the primary register are loaded into the shadow register for use by the test circuit. This architecture prevents the inputs from affecting the test circuit while data is shifted in or shifted out of the DR.

Figure 7. Legacy STAP1.0 Top-Level Block Diagram



2.1.9 Accessing the Data Registers

The data registers in the component or agents are accessed in the same way as the instruction register. Depending on the function of the data register, a TAP control signal may or may not be used. For example, a read-only data register would only need the Capture-DR and Shift-DR state signals to provide the necessary control events to capture the data and extract it out of the TDO pin.

Data registers have their own specific IR addresses that need to be written to the TAP before access is allowed. Writing of the IR register enables the TDI and TDO pins to connect to the data register of interest. Note that the only controller states in which data register contents actually change are Capture-DR, Shift-DR, Update-DR and Test-Logic-Reset.

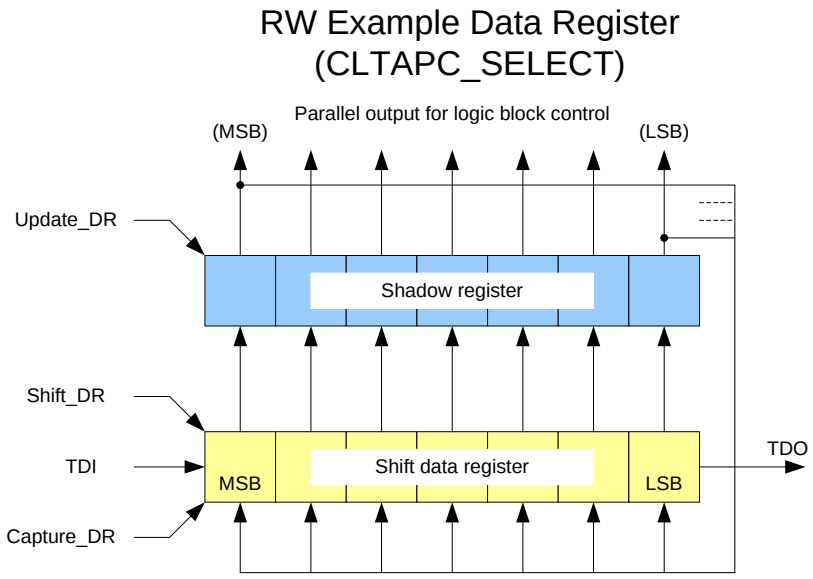
2.1.10 Public TAP Instructions

These instructions select from among three different public data registers – the boundary scan, device ID, and bypass registers. The public instructions can be executed in isolation with only the standard connection of the JTAG port pins. This means only the TCK clock is required for public data register access. The instruction length is implementation dependant and not specified here. Table 3-3 in section 3.2 contains the assigned instruction addresses of all the public TAP instructions outlined in the IEEE1149.1 spec and the SoC TAP required instructions.

2.1.11 Public Test Data Register Descriptions

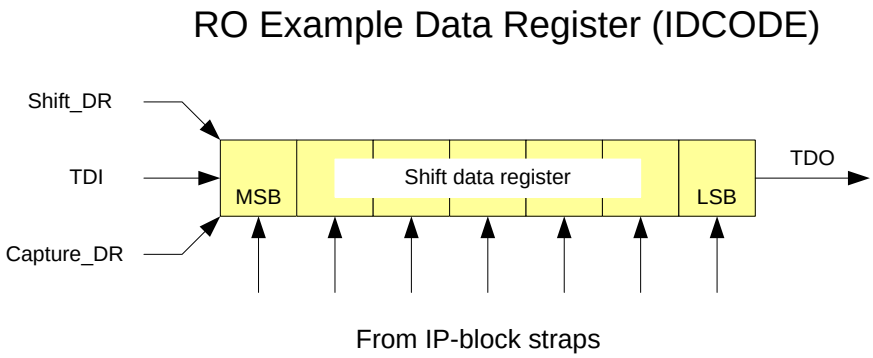
This section describes the test data registers (TDR) that are accessed by the public instructions. In general, there are two primary attribute styles associated with the publically accessible registers; a read/write register and a read-only register. A private test/debug data register may have other attribute types including a mixture of both within the same register (refer to section 2.1.14). Shown in Figure 8 is an example read/write register using the CLTAPC_SELECT as a reference. When the TAP FSM is in the Capture-DR state the shift register capture the output of the shadow register so the user can read what has been written to verify a successful write to the register. In the Shift-DR state the bits are transferred into the shift register from the MSB to the LSB of the register. The final step of the process is to transfer the data from the shift register into the shadow which occurs in the Update-DR state. Of course, all of this is explained in greater detail in the IEEE 1149.1 specification.

Figure 8. A Read-Write Attribute Example for a Test Data Register



For completeness, a read-only register such as the IDCODE register, as shown in Figure 9, only requires a capture and shift state since it is capturing the strapped values assigned to it and shifting them out for observation.

Figure 9. A Read-Only Attribute Example Test Data Register



2.1.12 Public Data Register Control

Table 1-3 defines the actions that occur with a selected set of instructions data register in each of the primary controller states. If a TAP state does not affect the selected data register, then the corresponding table entry will be blank. Not all data registers have a parallel output flop. All data registers have a parallel input latch. Several table entries are still under investigation.

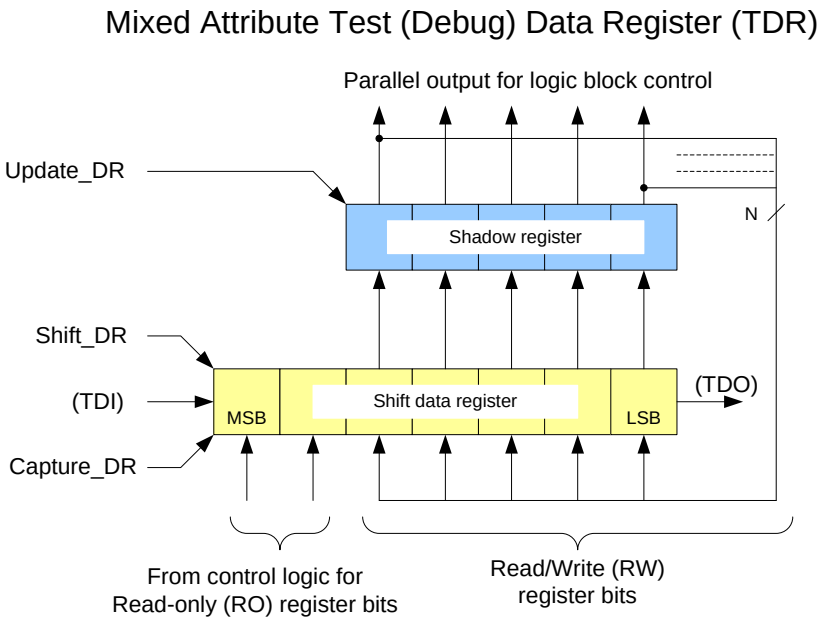
2.1.13 Private TAP Instructions

Each SoC integration team and SEG agents must define their set of test and debug private data registers and associated instructions to suit the requirements of the component or individual agent. Common opcodes are under investigation for those functions that are expected between multiple IP-blocks.

2.1.14 Private Data Register Descriptions

The private test data register (TDR, also referred to as a test/debug register since it supports DFT and DFV capabilities) is functionally the same as the public data register, however, it may support a more complex attribute set. A graphical view is shown Figure 10. This TDR representation has both read-only and read/write bit fields. The read-only bits are the MSB and MSB-1 bit positions of the shift register. During the Capture-DR state the values on the input to the shift register is captured so that they can read externally after the shift operation. Since the bit field is a read-only attributed it does not require shadow register which also reduces unnecessary gates. The read/write bits (from MSB-2 to LSB as shown in the figure) will capture the output of the shadow register in the Capture-DR state and shift out the data in the Shift-DR state. The contents of the shift register are transferred to the shadow register in the Update-DR state.

Figure 10. Private Test Data Register Example



2.1.15 Private DR Reset Control

The private DR registers should only be cleared with the following methods.

All private test data registers are cleared with a power good (fdfx_powergood) assertion.

This revision allows for a programmable reset to clear a private test data register with TRST_b or a software write to a TDR register bit. These register methods are in addition to the power good de-assertion.

2.1.16 Internal TDR Support (iTDR)

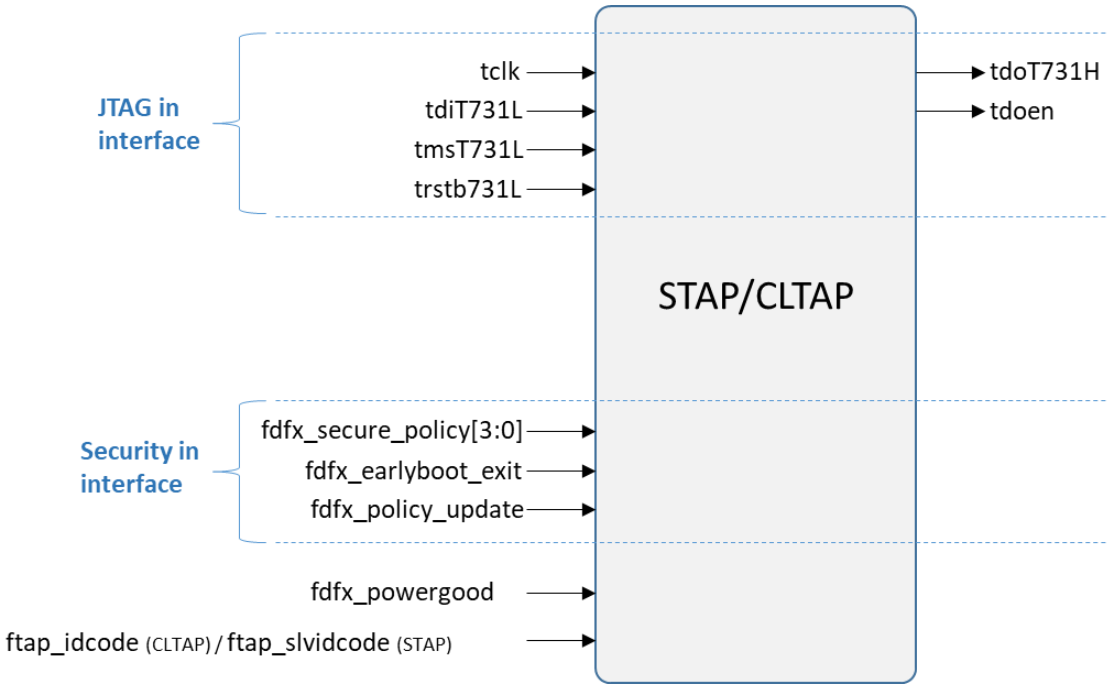
An internal test data register is a parameter developed basic read-write register. It provides an input capture bus and an output data bus. The register is composed of a shift register and shadow (update) register of equal bits. It may be used for any application.

2.1.17 Remote TDR Support (rTDR)

A remote Test/Debug Register (rTDR) is register that resides outside of the TAP controller but within the IP-block or agent. The purpose of this capability is for those sub-units within an IP-block that have embedded TDRs. In most cases, the registers within the TAP controller provide the TDRs. A remote TDR may be driven by TCK or in the IP-block's clock domain. The rTDR can be reset by powergood or by TLReset state, where by default it is reset by powergood.

2.2 Block Diagram

Figure 11. Basic STAP/CLTAP Interfaces



STAP and CLTAP interfaces can be extended for RTDR registers, IOV interface, IJTAG interface, etc.

2.2.1 STAP 2.0 Components

STAP2.0 will support the basic functionality of TAP, the same as in the STAP IRR version. The default STAP2.0 configuration will be fully compliant with the IEEE1149.1 specification and provide the basic TAP commands (BYPASS and IDCODE) by default.

This TAP can be extended on demand according to the TAP network connectivity. Instructions for extending the STAP2.0 will not be part of this HAS.

2.3 Landing Zone

2.3.1 STAP2.0 Features

Table 2. Goals

Goal	POR	Stretch
Goal ABC	X	
Goal XYZ		X
Goal 123	X	

Default

- TAP FSM

- 8-bit IRs (users can extend the IR size but the default is 8 bits)
- Integrated Secure Plugin agent
- BYPASS TAP instruction
- IDCODE TAP instruction
- Common shift register (ComShiftReg) for Capture/Shift/Update, see section 3.1.

On Demand

- ITDR/RTDR registers
- iJTAG (same as RTDR) registers
- CLTAP Boundary scan agent
- TAP register write enables
- Sticky bits with auto clearance at capture-DR
- Integrated micro break point controller
- Integrated IEEE1149.7 network component
- Full TAP clock crossing support

Note: STAP2.0 includes additional features which are not listed in this HAS.

2.4 Voltage and Frequency Operating Points

Not applicable to this IP

2.5 Standards and Specification Compliance

The sTAP has one deviation from the IEEE Std. 1149.1, which is that when the sTAP implements the required SLVIDCODE, the sTAP's instruction register resets to opcode 0xC and points to the SLVIDCODE register.

2.6 Difference from Previous Project

Not applicable to this IP

2.7 Use Cases

Not applicable to this IP

2.8 Derivative Strategy

Not applicable to this IP to this IP

2.9 Scalability, Configurability, and Customizations

Not applicable to this IP to this IP

2.9.1 Hardware Scalability, Configurability, and Customizations

Not applicable to this IP

2.9.2 Software Scalability, Configurability, and Customizations

Not applicable to this IP

2.10 Design Assumptions

Not applicable to this IP

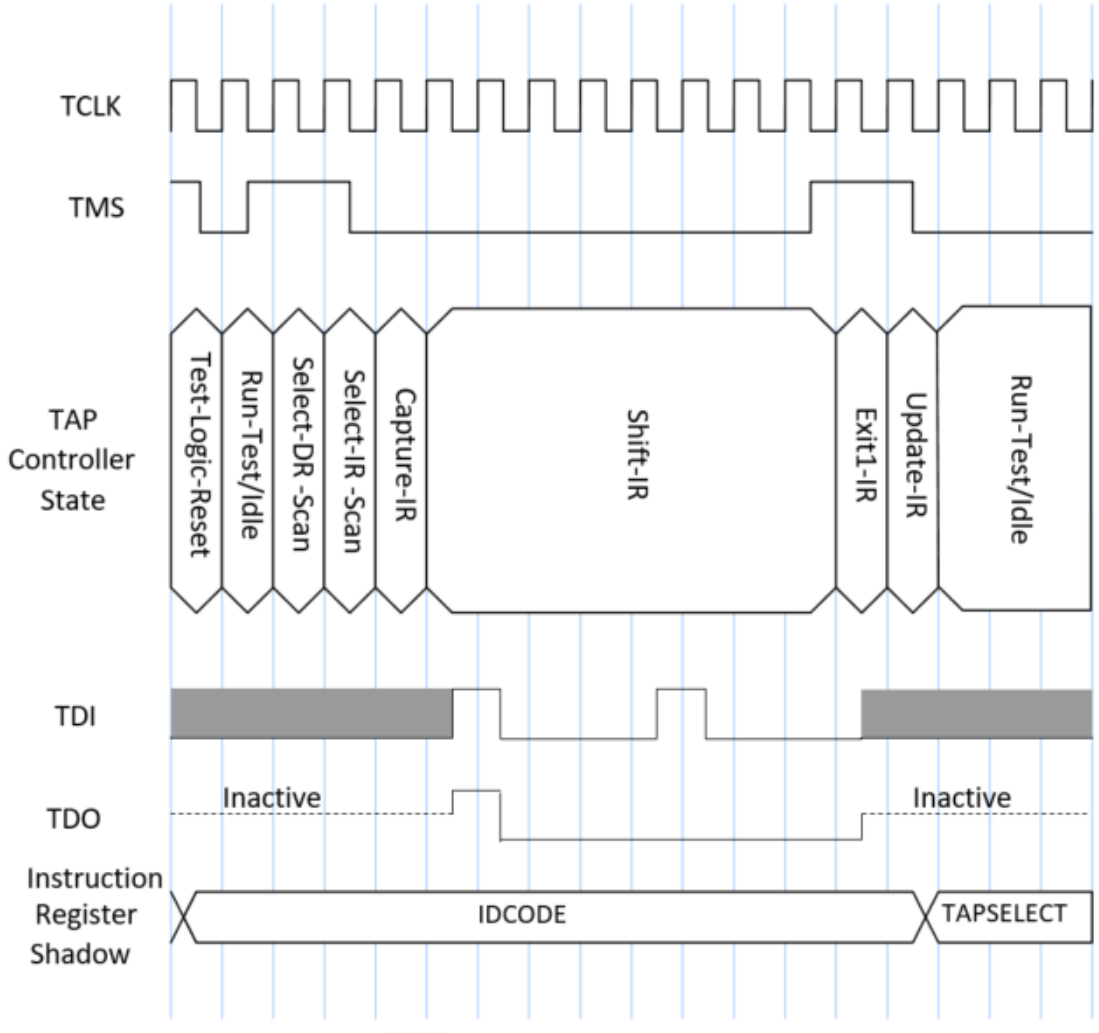
2.11 Example Access Timing Diagram

Figure 12 shows a timing diagram for loading the TAPSELECT instruction (0x11) into the TAP. The vertical labels on the figure show the specific clock edges where the Capture-IR, Shift-IR, and Update-IR actions occur.

Capture-IR (which pre-loads the instruction shift register with “0000 0001b”) and Shift-IR operate on rising edges of TCK, whereas Update-IR (which updates the actual instruction register) takes place on the falling edge of TCK.

Note: The LSB of the TAP instruction must be shifted in first.

Figure 12. TAP IR Access



3 Functional Description

3.1 TAP Common Shift Register (ComShiftReg)

The Common Shift Register (ComShiftReg) is used to save area and provide better timing on the TDO final MUX. The ComShiftReg is used for iTDRs. However, on the remote side, you can use the ComShiftReg to implement the RTDR itself.

Figure 13 shows how a ComShiftReg is implemented. In this example there are four TAP commands with four DRs: DR0 is of size 1, DR1 is of size 2, DR2 is of size 4, and DR3 is of size 5.

The ComShiftReg is the maximum size of these four DRs, which is 5. The 5-bit shift register can be configured as either a 1-, 2-, 4-, or 5-bit shift register based on the DR that is selected using the opcode in the IR. For that, bypass MUXes are added between the flops. A set of MUXes have also been added to select the output of the correct DR to be loaded into the ComShiftReg.

- **@ CaptureDR State**

  Data from the selected DR, according to the current TAP command (capture MUX) is latched into ComShiftReg FFs. The four DRs in this example have different sizes:

  - First capture MUX: selects between DR0-DR3 (5)
  - Second capture MUX :selects between DR1-DR3 (4), no TDI MUX is needed between ComShiftReg[2], ComShiftReg[3], and DR0 size of 1
  - Last capture MUX: default is DR3 (1)

- **@ ShiftDR state**

  Data from the previous ComShiftReg FF or TDI is latched into the next ComShiftReg FF shifted right by one bit. The TAP IR opcode selects a specific TDI MUX corresponding to the size of the DR selected by the IR opcode.

Note: DR3 is the default TAP command (TDI is always driven into ComShiftReg[4] (6) in ShiftDR state).

- **@ UpdateDR state**

  Updates the appropriate DR from using the parallel input from ComShiftReg based on the size of the DR selected by the TAP IR opcode.

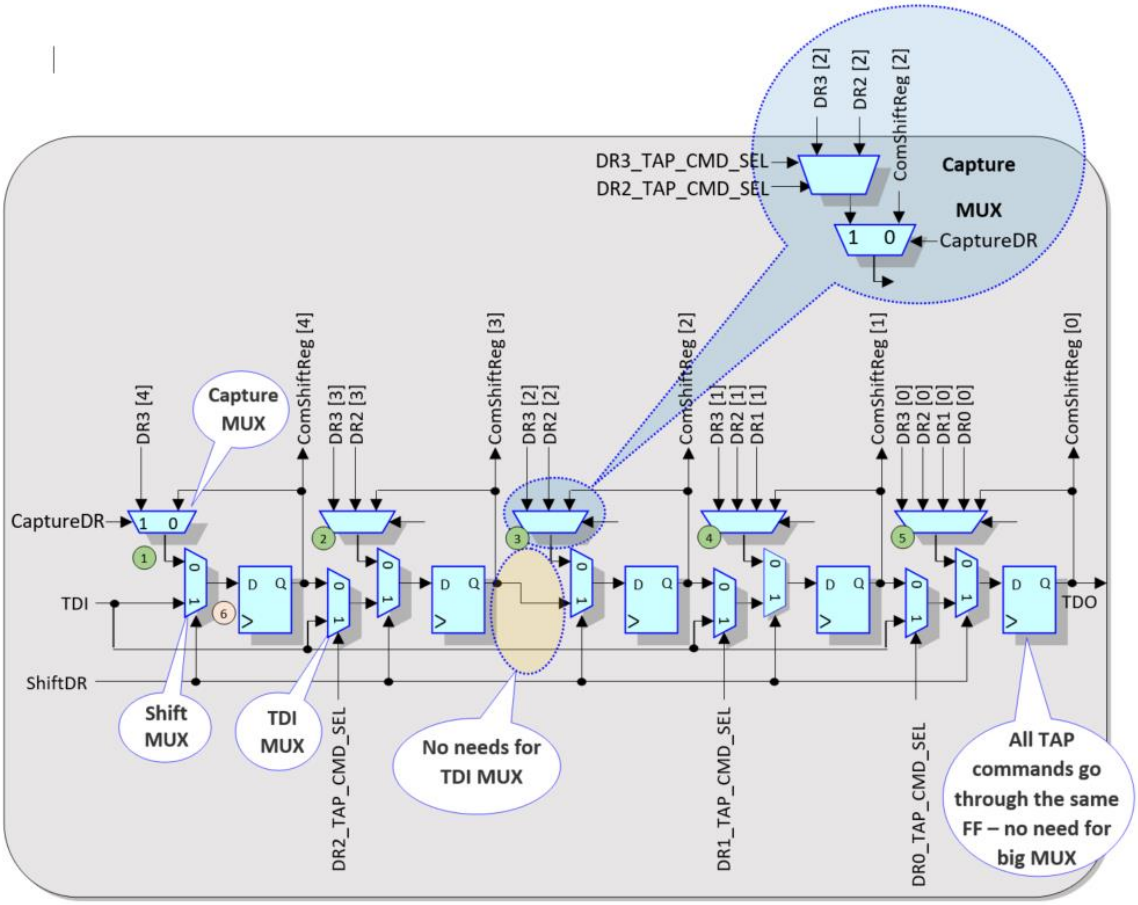
  For example: if (UpdateDR), then DR2[2:0]=ComShiftReg[2:0], else DR2[2:0] retains its value.

Note: The UpdateDR logic is not shown in Figure 13.

- **Default**

  The default for all other states. In all other FSM states, the data in the ComShiftReg is retained.

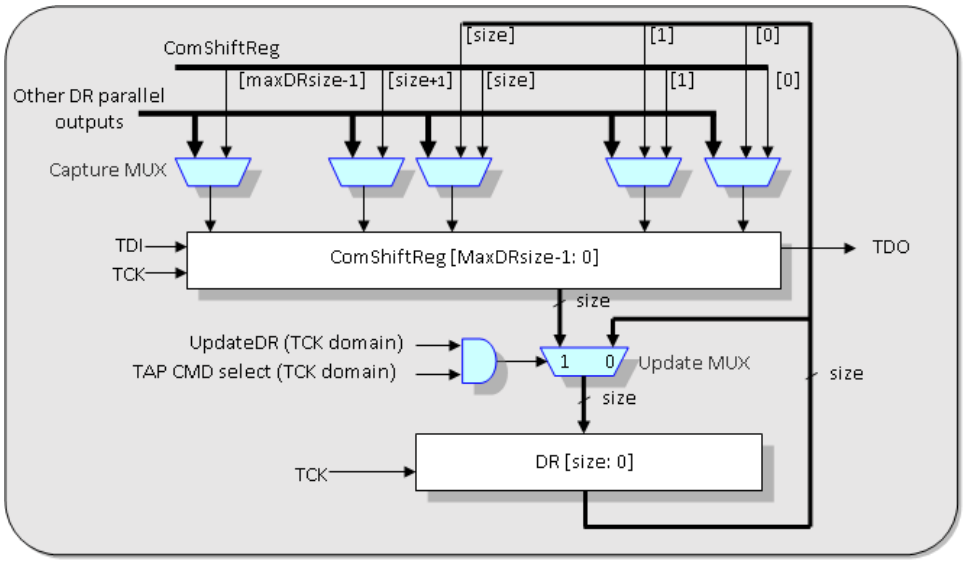
Figure 13. TAP Common Shift Register Implementation Details



3.2 TAP Clock Primary DR with TAP Access Only

DRs in the TCK clock domain are connected to the ComShiftReg, shown in Figure 14. Because both ComShiftReg and DR are in the TCK clock domain, there is no need to synchronize data for the update-DR and capture-DR operations.

Figure 14. Primary in TCK Clock Domain



3.3 Local Clock Primary DR (with TAP access only)

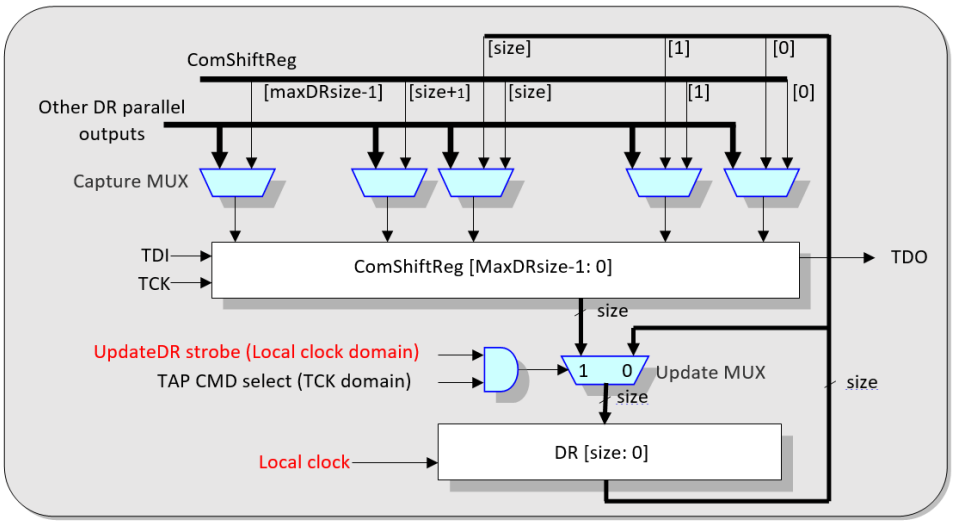
There are two options to implement the local clock domain register with TAP access only:

1. Synchronize the data with the Update-DR strobe signal

Note: Data inside the common shift register is already stable for one TCK clock cycle.

2. Use Clk sample TCK high, this clock is a qualifier for TCK rising edge running in local clock.

Figure 15. Local Clock Primary DR (with TAP access only) – Option 2

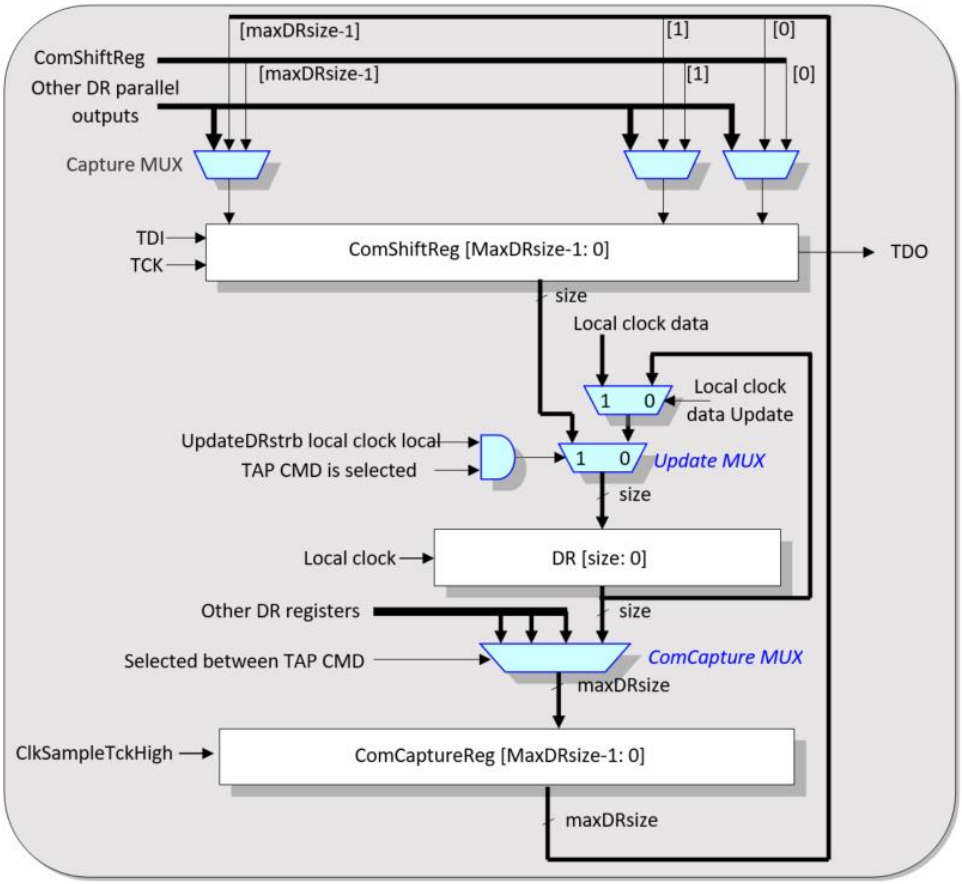


3.4 Local Clock Primary DR (with TCK and local clock access)

A primary DR that's implemented in the local clock domain with TCK and local clock access needs additional logic for the capture DR path. Data from local clock DR is sampled first with

ClkSampleTckHigh (local clock TAP qualifier clock) and then it samples the data again with TCK clock into ComShiftReg.

Figure 16. Primary DR in the Local Clock Domain



3.5 BYPASS (0xFF)

This register provides the minimal length path between TDI and TDO. It is loaded with a logical 0 during the Capture-DR state. The Bypass Register is a single bit register and is used to provide a minimum-length serial path through the device. This allows more rapid movement of test data to and from other components in the system. When in Bypass Mode, the operation of the test logic shall have no effect on the operation of the devices normal logic.

Register Name: BYPASS				
Bits	Access	Default	Label	Bit Description
0	RW	0	Bypass	A one-bit register used to bypass this TAP in a series stitched topology.

3.6 SLVIDCODE (0xC)

The SLVIDCODE provides a unique Slave/ID code and is used to manage the number of TAPs within the SoC. This format applies to any TAP within the SoC no matter what the IP block is, whether it is internal or external. The only exceptions are the CLTAP and the IA-cores. The CLTAP gets an assigned IDCODE to identify the component.

As of this publication, there is no change in how we treat IA-cores. It is important to re-emphasize the use of this format for all TAPs including external IP-blocks. This includes the MBIST TAP that requires an IDCODE. Since the SLVIDCODE follows the IEEE1149.1 convention, namely, the least significant bit is logic '1', then any derived code is suitable.

Using a list of TAP codes, each assigned to a unique value with 2^30 possible combinations, you can satisfy the requirement of uniquification. However, we can use those bits to get important information obtained by reading the SLVIDCODE.

This is especially true when the number of TAPs increases because the process technology allows for more IP-blocks. Table 3 lists the sub-fields and their associated bit range.

Table 3. Legacy SLVIDCODE Format

Sub-field	Bit(s)	Description
Version	[31:28]	
Special TAP information [27:24]	[27]	This TAP is an IA-core
	[26]	This TAP is a vendor TAP
	[25]	This sTAP controls a sub-TAP network
	[24]	This sTAP controls a WTAP network
Power island	[23:19]	
Region ID	[18:13]	
ID of sTAP network controller	[12:8]	
sTAP ID	[7:1]	
Logic 1	[0]	

Using the legacy SLVIDCODE bits format as described in Table 3 ends up with non-unique SLVIDCODE values across the project. HTAP products are requested to assign any 32bit number to the SLVIDCODE register if it is a unique number across the project. Any detailed about the TAP instance can be specified using external specification files instead of assigning it in the RTL logic.

Table 4. HTAP SLVIDCODE Format

Sub-field	Bit(s)	Description
ID	[31:0]	Unique ID number

3.7 Remote TAP Data Register (rTDR)

A Remote Test/Debug Register (rTDR) resides outside of the TAP controller but within the IP block or agent. The rTDR control signals are not available on the IOSF or Pondicherry DFx interface. A SIP TAP controller can implement a TDR in two ways:

- The TAP controller can designate a group of TDRs within the TAP controller where the output is a bus of all bits from all registers concatenated together.
- The TAP controller can provide the controls necessary to drive the rTDR outside of the TAP that is embedded in an IP block. A remote TDR will be driven by TCK.

3.8 TAP Network Control Logic

Refer to the TAPLink HAS / HTAP HAS according to the network type. See section 1.2 for location.

3.9 Boundary Scan Description

Table 5. Chapter Revision History

Revision Number	Description	Revision Date
0.90_rc2a	No updates to this section.	February 2012
0.90_rc4	Updated figure, table and section references after moving to the new HAS template.	March 2012

This chapter is intended to provide a sufficient level of information for the development of a common set of boundary-scan capabilities for SoCs. The information presented here will be more detailed information than a HAS but not enough to be considered a MAS. Although the expectation is that this will be implemented in the CLTAP it may be implemented in a local hard-IP TAP for direct boundary-scan operations if the IP-block developer requires such a capability. It should be noted that a hard-IP block may control the local boundary-scan control signals directly for to satisfy HVM test requirements such as force all 0's, all 1's or high-Z. Such an implementation assumes a local mux control when these capabilities are enabled and the normal boundary-scan operation is not inhibited by such an override.

3.9.1 Boundary-Scan Register

The boundary-scan (also referred to as bscan) register consists of all boundary-scan data cells and control cells for those device pins supporting boundary-scan. Boundary-scan control cells (if any are required) drive the output-enables of one or more bidirectional or output pins. The boundary-scan register is distributed around the I/O pads of the chip.

The parallel outputs of the boundary-scan register change only on the falling edge of TCK in the Update-DR TAP state, and are loaded with the values contained in the serial-shift path of the boundary-scan register. The reset value of the Boundary-scan register is undefined.

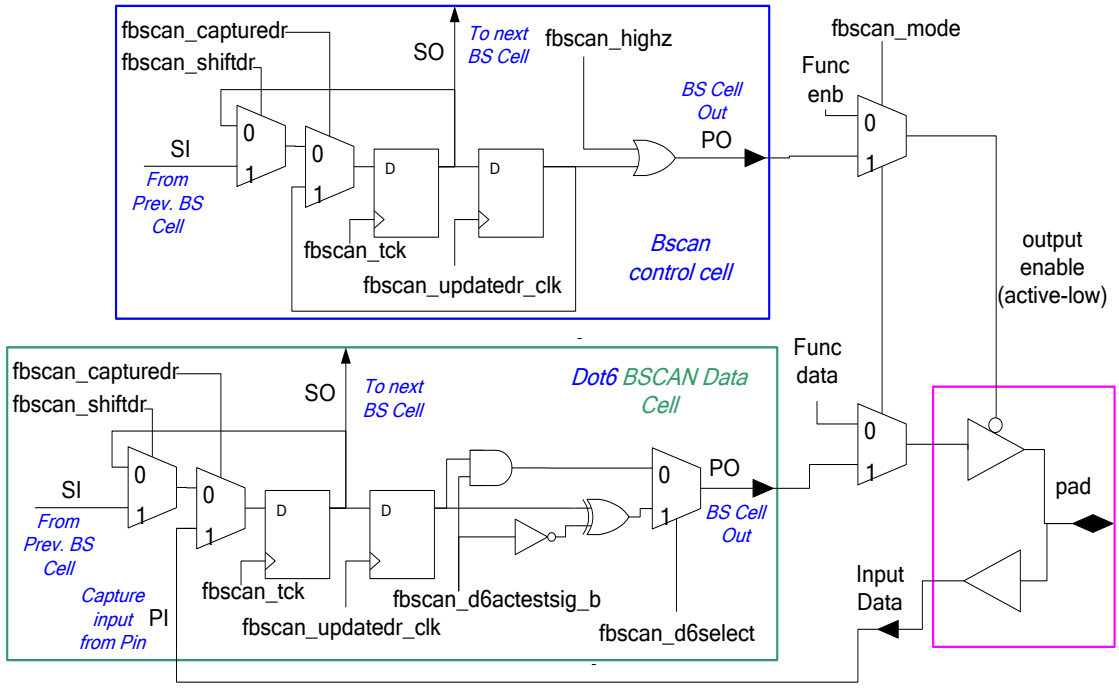
The structure of boundary-scan data and control cells typically vary somewhat across interfaces and pin types. This document explains a general boundary-scan cell then will move into the details of the cells for each interface.

3.9.2 Typical Boundary-scan Register Cell

Within the IEEE 1149.1-2001 specification, Figure 17 shows an example boundary-scan cell for a bidirectional pin where INTEST is not supported. This cell uses a "BC_2" control cell for the pin's output-enable, and a "BC_8" combined input/output data cell. Modifying this to account for active-low enables, to add HIGHZ support, as well as the AND gate for EXTEST_TOGGLE support and the logic for 1149.6 support, results in the configuration as shown in Figure 17. This forms the basic boundary-scan cell for bidirectional pins. In some cases, the boundary-scan control cell is shared to control the output-enables for multiple pins. The boundary-scan data and control cells are serially stitched together around the chip, with the first fed by TDI, and the last feeding TDO.

Note: Figure 17 and Figure 18 are reference designs based on 1149.1 spec and not necessary those used in SoCs. Refer to section 3.9.3 for information on the converged bscan cell architecture.

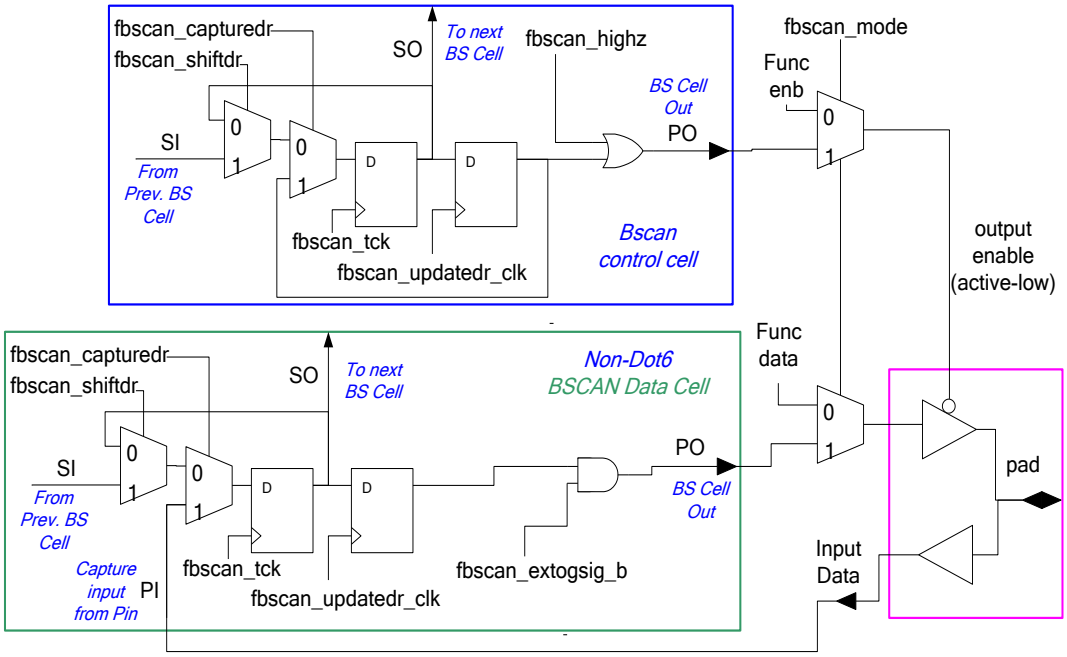
Figure 17. Basic Bscan Cell Configuration (extest_toggle + 1149.6)



NOTES:

1. Figure 17 shows the general structure of boundary-scan cells. Please refer to subsequent sections for details on the boundary-scan registers for each interface.
2. The Flops within the boundary-scan cells do NOT require any reset.
3. The fbscan_d6select mux, XOR gate, and inverter are only required for pins that support 1149.6.
4. The inverter feeding the XOR gate inverts the fbscan_d6actestsig_b which was inverted at the source when used for 1149.6 instructions so that a single AC test signal can be used for EXTEST_TOGGLE and the 1149.6 EXTEST_TRAIN and EXTEST_PULSE instructions.
5. Connections to 1149.6 Test receivers (if any) are not shown here.
6. Refer to IEEE 1149.1 specification for cell details if INTEST support is required.
7. For pins that support EXTEST_TOGGLE but NOT 1149.6, the basic boundary-scan cell configuration is as shown in Figure 18.

Figure 18. Basic Bscan Cell for Pins Supporting EXTEST_TOGGLE Only

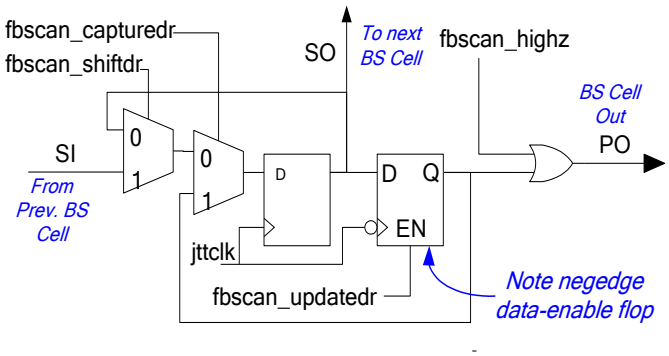


Note: For those pins supporting neither EXTEST_TOGGLE nor 1149.6, the boundary-scan cell is the same as shown in Figure 5-2 except it lacks the AND gate and the fbscan_actestsig_b input (or this input is simply tied high).

3.9.3 HIP Bscan Cell Design - Negedge Data-enable Flop

The current hard-IP design implements a parallel stage negedge flop clocked by TCK, with the fbscan_updatedr used as an enable (instead of using the fbscan_updatedr_clk to directly clock a posedge flop) as shown in Figure 19. This variation will be substituted for any of the types of bscan cells shown in other figures.

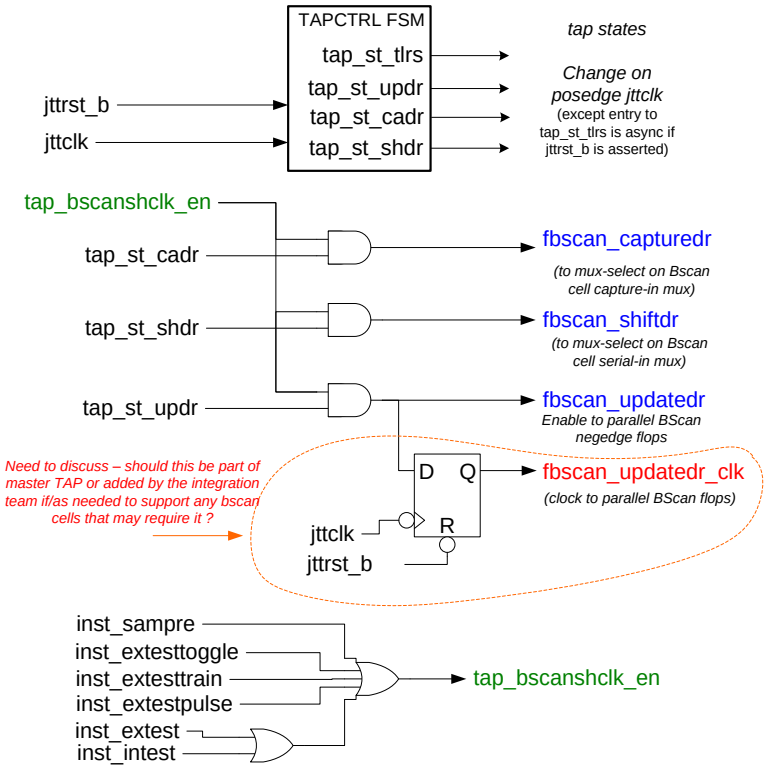
Figure 19. Hard-IP bscan Cell Design using neg-edge data-enable Flop



3.9.4 Boundary-scan Control Signal Generation

The CLTAP (or potentially a slave TAP) generates the control signals necessary for the boundary-scan chain. Figure 20 and Figure 21 are detailed diagrams that are explained throughout this section.

Figure 20. Bscan Chain Control Signal Generation



Note: In the descriptions and logic diagrams, "inst_sampre" refers to a combined SAMPLE/PRELOAD instruction. If separate SAMPLE and PRELOAD instructions are used, "inst_sampre" would be replaced by the Logical OR'ing of inst_sample and inst_preload.

The fbscan_capturedr signal controls the mux(es) that select the source of the data feeding the serial-stage flop of the boundary-scan register. It is asserted to select the parallel-input capture path.

The fbscan_shiftdr signal controls the mux(es) that select the source of the data feeding the serial-stage flop of the boundary-scan register, and it is asserted to select the serial-input shift path.

The fbscan_capturedr and fbscan_shiftdr signals change on the posedge of TCK, giving them nominally a full cycle of setup relative to the rising edge of the TCK clock to the bscan register. Hold time is generally not a problem due to propagation delays from the TAP to the bscan cells, but if needed, hold buffers are added to meet hold time.

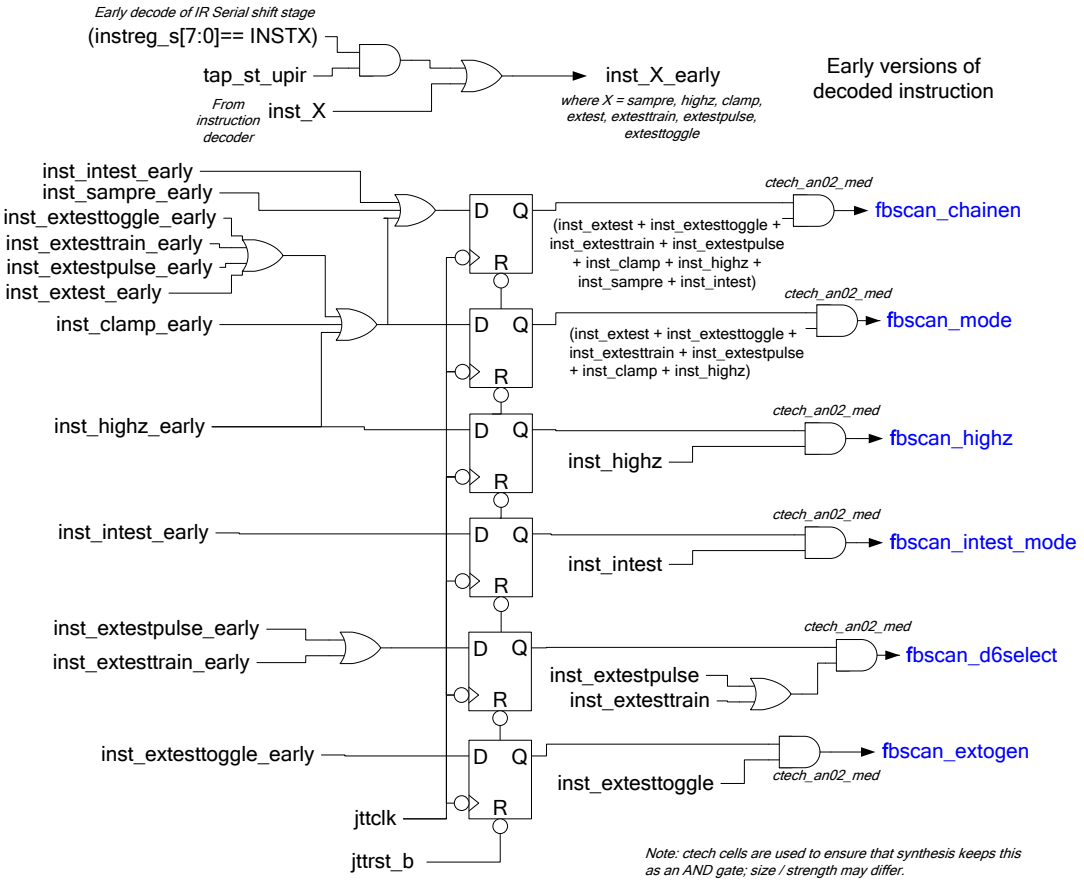
The boundary-scan update-DR has some special considerations, depending on the type of bscan cell used, and considering the requirement to meet IEEE 1149.1 Rule 8.8.1(d), which requires:

> "When the EXTEST instruction is selected, the state of all signals driven from system output pins shall be completely defined by the data held in the boundary-scan register and change only on the falling edge of TCK in the Update-DR controller state."

A given SOC design may require either or both of the following cases:

- Case 1: For boundary-scan cell designs which use negedge TCK flops as parallel-output stage (refer to Figure 5-3): The fbscan_updatedr signal enables loading data into the parallel (output) stage of the boundary-scan register negedge flops, from the serial shift stage. This signal changes on the posedge of TCK, giving ½ cycle of setup to the next falling edge of the clock, which launches the data out onto the pin, thus meeting 1149.1 Rule 8.8.1(d).
- Case 2: For boundary-scan cell designs which use the update-dr signal to directly clock the parallel output stage (such as Figure 5-1 and Figure 5-2): The signal fbscan_updatedr_clk is a negedge version, intended to directly clock the parallel stage of the boundary-scan register. This clock to the bscan register is driven by a neg-edge flop so that the parallel outputs of the bscan register change on the falling edge of the TCK during the Update-DR state (as required by IEEE 1149.1). The fbscan_updatedr_clk signal is one TCK wide instead of ½ TCK wide (as the updateDR signal is shown in IEEE 1149.1), but this is not important since fbscan_updatedr_clk clocks posedge flops in the bscan register.
- The tap_bscanshclk_en signal (which remains internal to the TAP) is asserted during any of the EXTEST instructions or SAMPLE/PRELOAD (or INTEST, if supported) and indicates that the boundary-scan register is to be clocked during Capture-DR or during Shift-DR. It is used within the TAP to prevent updating (pulsing fbscan_updatedr) the bscan register except for those instructions where the boundary-scan register is connected between TDI and TDO. It also prevents Capture and Shift of the boundary-scan register when it's not the selected test data register. The IEEE 1149.1 spec (in the NOTE in Figure 6.5 of -2001 version) states that ... "UpdateDR control signal will be controlled ... such that the signal is only fed to the test data register that is selected as the serial path between TDI and TDO". Thus, 1149.1 only requires that the register is not Updated in such cases.
- Note that glitches may be present on any of these three signals (fbscan_capturedr, fbscan_shiftdr and fbscan_updatedr) due to the way they are driven off combinational logic (e.g. tap_bscanshclk_en. Glitches on these signals are acceptable due to the way these signals are used by the boundary-scan cells, and that the glitches will have settled by the time the bscan cells are clocked.
- It is important that the TAP operation does not interfere with normal operation of the chip, especially in cases such as debug where the TAP is used for other purposes besides boundary-scan. It is critical that key boundary-scan control signals do NOT glitch, to prevent disturbing the normal operation of the I/O interfaces. Combinational logic such as decoders can easily introduce glitches, so the circuits in Figure 21 are recommended for the generation these boundary-scan control signals.

Figure 21. Bscan Control Signal Generation



Early versions of the decoded instructions (inst_X_early) are generated, based on the serial instruction register matching the corresponding instruction and during the tap_st_upir (update-IR) state, OR when the parallel instruction register matches the corresponding opcode. These "early" versions are used to prevent delaying the generation of the signals (such as fbscan_mode), which would introduce delays at the pins upon entry into boundary-scan modes (e.g. moving from SAMPRE to EXTEST). (Refer to IEEE 1149.1-2001 8.8.1(d) which requires that Device pins must change at the falling edge of the jtag clock after entering the Update-DR state, during specific boundary-scan instructions. The case of Update-DR is not generally a problem, but upon switching from SAMPRE to EXTEST, any delays on the boundary-scan controls signals such as fbscan_mode would directly translate to delays on pin edges.)

The "early" instructions run through the necessary combinational logic, and the result is run through falling-edge flops, to prevent falsely asserting the boundary-scan control outputs due to any glitches that occur on the decoded outputs. ANDing the flop outputs with the instruction decodes allows the signals to deassert without incurring additional delay. Note: glitches may happen here (at the deasserting edge, which only can cause them to falsely deassert briefly, but the glitches must not cause signals to falsely assert, since they could cause false entry into boundary-scan). Refer to Figure 5-6 for a timing diagram, using the HIGHZ instruction as example.

The signal fbscan_chainen is asserted during those JTAG instructions that require the boundary-scan register to be enabled for operation. This signal is intended for use to set up

muxes and other logic that allows the boundary-scan register to operate, including during Sample/Preload operation, but not necessarily to actually control the pins.

The fbscan_mode signal is asserted during those JTAG instructions that actually require the boundary-scan register to take control of the pins as output; i.e. this is the traditional "mode" control signal to the boundary-scan register muxes that select between functional and boundary-scan operation. It specifically includes CLAMP and HIGHZ and excludes SAMPLE/PRELOAD.

The fbscan_intest_mode signal is asserted during INTEST. This signal controls muxes in the boundary-scan cells, to enable the boundary-scan cells to control the value going to the core, instead coming from the input buffers.

The fbscan_highz signal is asserted during the HIGHZ instruction, used to place the buffers into high-impedance state. This is used to control the output-enable of the buffers through the boundary-scan cell and to control AFE blocks appropriately during HIGHZ

The fbscan_d6select signal is asserted during the EXTEST_TRAIN or EXTEST_PULSE instruction. It is used to place boundary-scan cells into the 1149.6 AC mode which allows them to be toggled under control of the AC test signal (fbscan_actestsig_b) which takes on 1149.6 functionality during the Dot6 instructions. This signal is deglitched by running it through a flop, to prevent glitching any control signals (to the custom circuits) that are switched based on being in one of the 1149.6 instructions.

IEEE 1149.1 requires the "when the HIGHZ instruction is selected, all system logic outputs ... shall immediately be placed into an inactive-drive state". Thus, it is preferred NOT to incur any delay on the fbscan_highz signal by flopping it. To prevent delays upon entry into HIGHZ, an early version of the HIGHZ instruction is generated based of the contents of the instruction register serial stage, qualified with the Update-IR state, and ORed with inst_highz, then run through a negedge flop to prevent glitches. The result is that assertion of fbscan_highz coincides with inst_highz (falling edge of TCK) and the deassertion of fbscan_highz occurs after exiting from HIGHZ instruction (refer to Figure 22). This case occurs if the user immediately enters HIGHZ or switches from HIGHZ immediately to EXTEST instruction.

Figure 22. Timing Diagram for the Generation of fbscan_highz

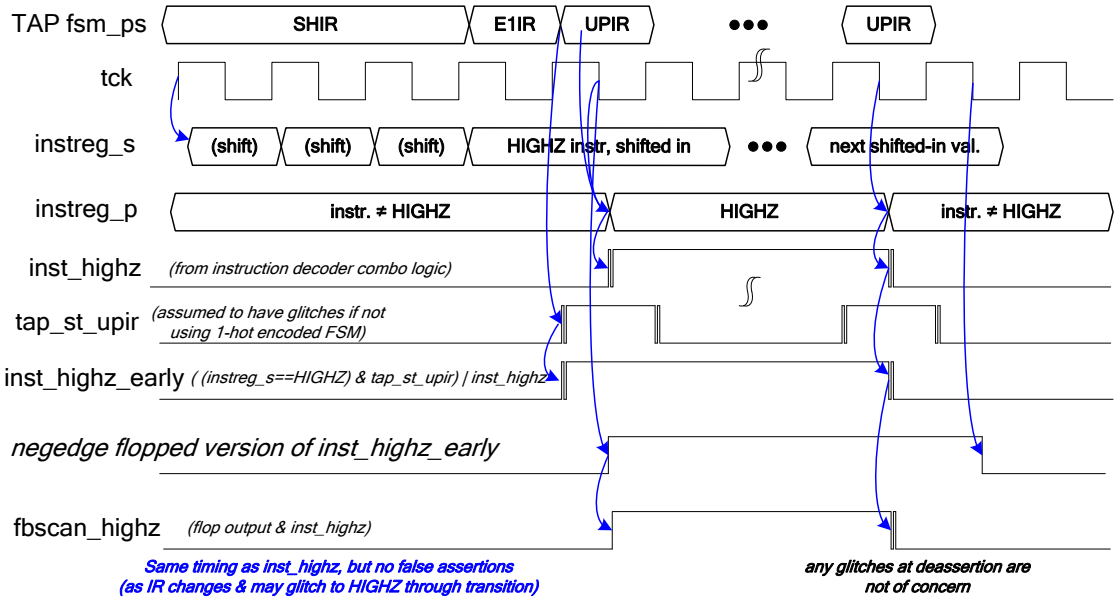


Table 6 summarizes the control signal generation for each boundary-scan instruction.

Table 6. Boundary Scan Test Mode Control Signals

	1Boundary Scan Test Mode Signals								
Instruction	mode	intest_mode	highz	chainen	extogen	extogsig_b	d6select	d6init	d6actestsig_b
SAMPLE/ PRELOAD	0	0	0	1	0	1	0	0	1
EXTEST	1	0	0	1	0	1	0	See note 4	1
EXTEST_ TOGGLE	1	0	0	1	1	toggling3	0	0	toggling3
EXTEST_ TRAIN	1	0	0	1	0	1	1	pulse2	toggle
EXTEST_ PULSE	1	0	0	1	0	1	1	pulse2	pulse
CLAMP	1	0	0	1	0	1	0	0	1
HIGHZ	1	0	1	1	0	1	0	0	1
INTEST	0	1	0	1	0	1	0	0	1
BYPASS	0	0	0	0	0	1	0	0	1
Others	0	0	0	0	0	1	0	0	1

NOTES:

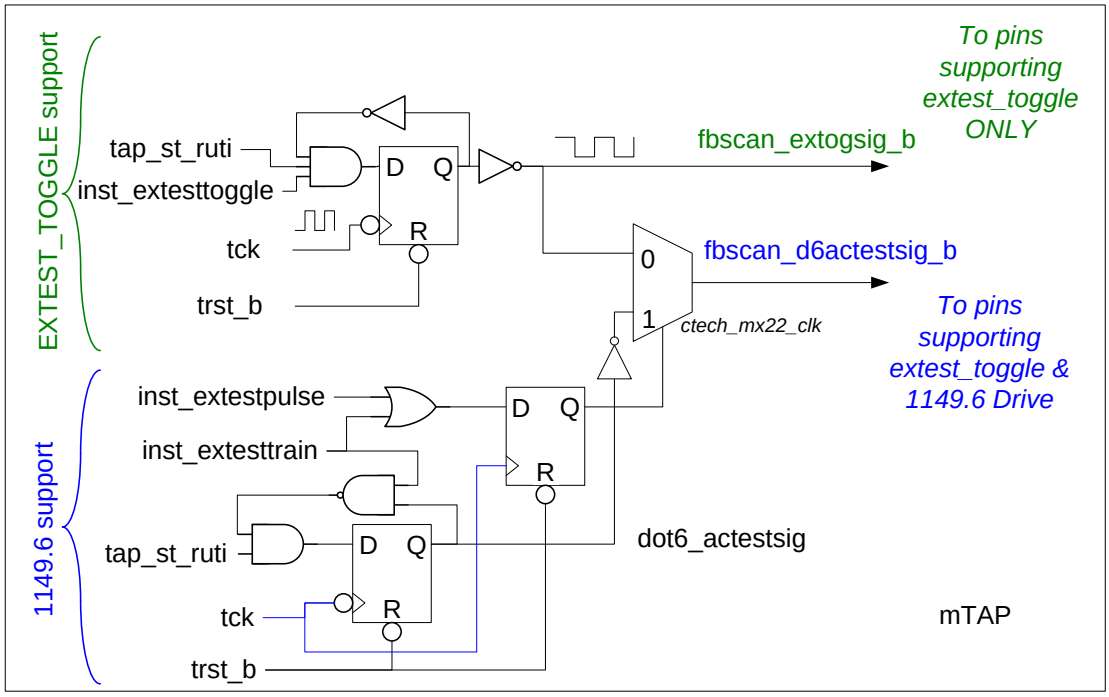
1. The "fbscan_" was removed to reduce the number of characters in the table header cells. The mode, highz, chainen and extogen must be deglitched in the CLTAP (or controlling sTAP). The reader should refer to Figure 1-7 for detailed information about the boundary-scan implementation.
2. The fbscan_d6init is pulsed with TCK in the Exit1-DR or Exit2-DR states
3. Toggles at the falling edge of TCK in Run-Test/Idle (refer to Figure 1-9).
4. It is product dependent whether the d6init signal is asserted for EXTEST. The 1149.6 (page 59) states "This standard only mandates the initialization of the hysteresis for EXTEST, EXTEST_PULSE, or EXTEST_TRAIN instructions". Rule 6.2.2.1 (d). Whenever a test receiver is operating in the level-detection mode on an AC input pin, the test receiver output shall be cleared of prior history on the falling edge of TCK in the Capture-DR TAP Controller state.

3.9.5 Generation of EXTEST_TOGGLE and AC Test Signals

The circuit shown in Figure 23 is used to generate the AC test signals for use during the EXTEST_TOGGLE and 1149.6 EXTEST_TRAIN and EXTEST_PULSE instructions. This logic is located within the designated TAP controller, usually the CLTAP.

The circuit includes two components, one which is used during the EXTEST_TOGGLE instruction, and the other which is used during the 1149.6 EXTEST_TRAIN and EXTEST_PULSE instructions. For pins supporting both EXTEST_TOGGLE and 1149.6, the outputs of these two circuits are muxed onto one signal for distribution to the boundary-scan cells. This eliminates the need to route two separate AC test signals to each boundary-scan cell (i.e. one for the Dot6 instructions, and another for EXTEST_TOGGLE). For pins supporting EXTEST_TOGGLE but not the 1149.6 instructions, only the EXTEST_TOGGLE component is sent to the boundary-scan cells.

Figure 23. Generation of Extest_toggle and 1149.6 AC Test Signal



The fbscan_extogsig_b signal changes on the falling edge of the fbscan_tck (or ftap_tck may be used) and it is toggled at TCK/2 frequency when in Run-Test-Idle during the EXTEST_TOGGLE instruction. It is specifically set "high" to enable traditional boundary-scan operation when EXTEST_TOGGLE is not active. This signal is sent to the bscan cells that support EXTEST_TOGGLE and not the 1149.6 instructions. This signal remains high (inactive) during the 1149.6 instructions so that the corresponding pins are not toggled during the 1149.6 instructions, but rather, they get their value from the boundary-scan data cell.

The dot6_actestsig signal will toggle continually at each falling edge of TCK when in Run-Test-Idle during EXTEST_TRAIN. During the EXTEST_PULSE instruction, dot6_actestsig will go high at the first falling edge of TCK after entering the Run-Test-Idle state, and remain high until the first falling edge after exiting Run-Test-Idle.

The fbscan_d6actestsig_b signal is the combined AC test waveform is connected to those pins supporting both EXTEST_TOGGLE and 1149.6. If any of the 1149.6 instructions are active, then fbscan_d6actestsig_b drives the inverted output of the 1149.6 signal (dot6_actestsig). The combined signal will be inverted at each boundary-scan cell during 1149.6 instructions (EXTEST_TOGGLE or EXTEST_TRAIN) when fbscan_d6select = 1.

3.9.6 Generation of 1149.6 Test Receiver Initialization Signal

The circuit shown in Figure 24 is presented in IEEE 1149.6-2003, to initialize the test receivers. The 1149.6 circuit is not used as shown, due to possible issues with glitches on clocks generated by combinational logic, and race conditions between the last shift (stable output of capture flop) and the resulting memory initialization pulse. Instead, the circuit shown in Figure 25 is used to generate the signal used to initialize the 1149.6 test receivers. This logic is located in the boundary-scan controlling TAP (usually the CLTAP).

The clock gate flop on the state/test mode combinational logic is implemented with a falling edge (TCK) enable flop. This eliminates the effort needed by the backend design teams to resolve warnings from the timing analysis tool. A warning would be flagged because of the

combinational logic to the enable flop originates from the positive edge FSM state transitions (Exit1-DR OR Exit2-DR) and the clock gate (as shown in the figure) also occurs on the positive edge of TCK. The timing tools suggests this signal should be pushed out to the neg-edge which results in an implementation of an unnecessary number of hold buffers on this signal path.

Figure 24. Initialization Circuit for a Test Receiver – 1149.6 Version [NOT USED]

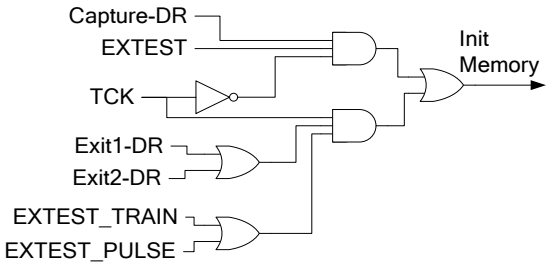
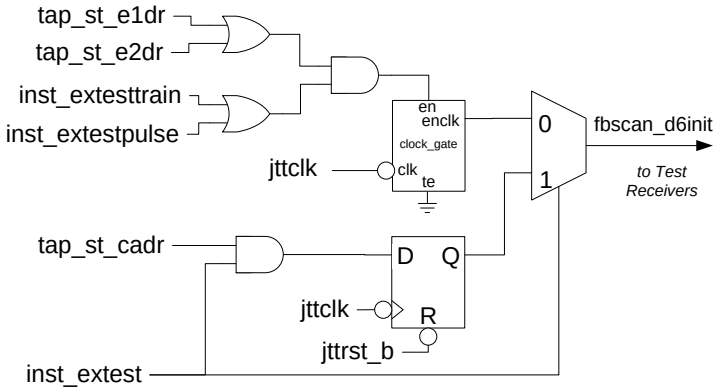


Figure 25. Generation of fbscan_d6init Test Receiver Init Signal [Intel Version]



During AC test instructions (EXTEST_TRAIN, EXTEST_PULSE), fbscan_d6init provides a rising edge to clock initial data into the 1149.6 test receiver D-FlipFlops, during the Exit1-DR or Exit2-DR states. These single-cycle states occur between the Shift-DR and Update-DR states, so that the test receiver initialization occurs according to the 1149.6-2003 Rule 6.2.3.1(d). Based on the TAP FSM state diagram, the Exit1-DR and Exit2-DR states are active for only one JTAG clock, thus, provide one rising edge for fbscan_d6init.

During DC boundary-scan operation (e.g. EXTEST), fbscan_d6init will go high on the falling edge of the clock during the Capture-DR state. The intention is initialize the test receiver memory with the current pin data prior to the Capture-DR state, so that, at the next rising edge of the clock, the pin data is captured into the Capture-DR flop. Rule 6.2.2.1 (d). Whenever a test receiver is operating in the level-detection mode on an AC input pin, the test receiver output shall be cleared of prior history on the falling edge of TCK in the Capture-DR TAP Controller state.

3.9.7 Fabric-facing Boundary-scan Signals

A graphical view of the boundary-scan signals that are connected to the fabric is shown in Figure 26. This is a reduced view from the IOSF spec Rev 1.2 which contains other

miscellaneous signals that are outside of the scope of this document. The detailed signal descriptions are listed in Table 7.

Figure 26. Boundary-scan Signal Diagram

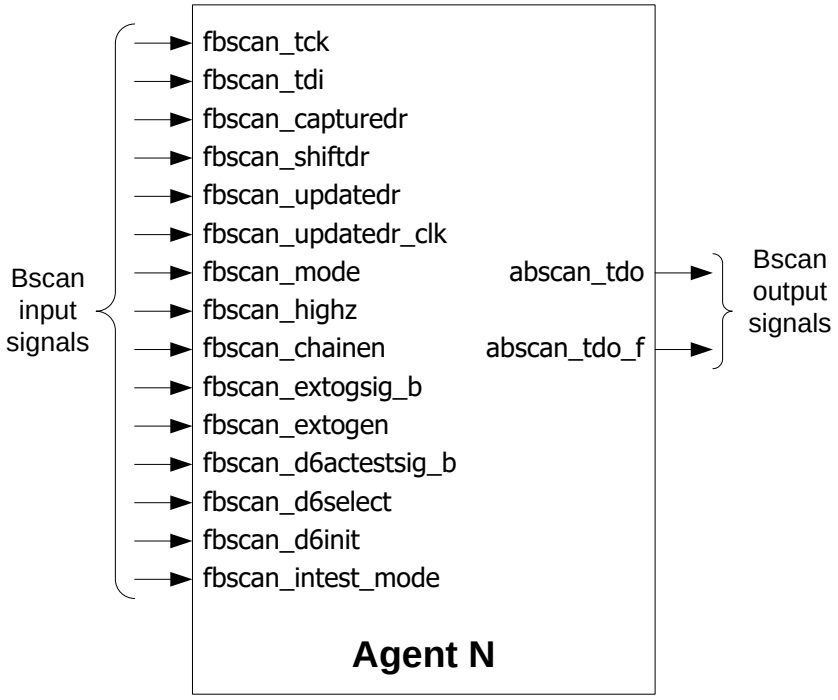


Table 7. IOSF-based Bscan DFT Signal List

Signal	I/O	1R/O	Description
Boundary Scan Support Signals			
2fbscan_tck	I	O	Fabric Boundary Scan Test Clock Input. This signal is the test clock input for this agent's section of the boundary scan chain stitched through the physical layer pins in the hard-IP macro. Generally, this version of the TCK signal is routed opposite (based on floorplan geography) with respect to the data (fbscan_tdi) signal. This fbscan_tck signal is for spec completeness and for hard-IP blocks that may not have a local slave TAP. It is implementation dependent on its use. Note: Use of this signal is optional even for hard-IP blocks that have boundary scan. This is signal phase and timing aligned with ftap_tck and they are expected to be connected to the same package pin source.
fbscan_tdi	I	O	Fabric Boundary Scan Test Data Input. This signal is the test data input for this agent's section of the boundary scan chain stitched through the physical layer pins in the hard-IP macro.
fbscan_capturedr	I	O	Fabric Boundary Scan Capture-DR Input. This signal is the boundary scan decoded Capture-DR control signal for this agent's physical layer pins in the hard-IP macro.
fbscan_shiftdr	I	O	Fabric Boundary Scan Shift-DR Input. This signal is the boundary scan decoded Shift-DR control signal for this agent's physical layer pins in the hard-IP macro.

Signal	I/O	1R/O	Description
fbscan_updatedr	I	O	Fabric Boundary Scan Update-DR Input. This signal is the boundary scan decoded Update-DR control signal for this agent's physical layer pins in the hard-IP macro.
fbscan_updatedr_clk	I	O	Fabric Boundary Scan Update-DR Clock. This signal is the boundary scan decoded Update-DR control signal for this agent's physical layer pins in the hard-IP macro. There are two potential bscan cell designs; this signal supports the cell with the updatedr directly connected to the bscan cell's clock input.
fbscan_mode	I	O	Fabric Boundary Scan Mode Input. This signal is the boundary scan decoded Mode control signal for this agent's physical layer pins in the hard-IP macro.
fbscan_highz	I	O	Fabric Boundary Scan High-Z control. This signal is the test data output for this agent's section of the boundary scan chain stitched through the physical layer pins in the hard-IP macro.
fbscan_chainen	I	O	Fabric Boundary Scan Chain Enable. This signal enables circuits in the physical layer to allow for boundary scan sample/preload operations to occur. It may be used to enable the proper pull ups/downs or turn on sense amps. This signal is active when any boundary scan instruction is decoded in the CLTAP.
fbscan_extogen	I	O	Fabric Boundary Scan Extest Toggle Enable. This signal enables the EXTEST_TOGGLE function when the instruction is valid. EXTEST_TOGGLE is a modified EXTEST boundary scan function that toggles bscan cell as an output at a period equal to the TCK frequency. This may be used for any IO type. For high speed serial IOs, this signal enables both the transmit (Tx) and receive (Rx) paths as an output for toggling logic values at the IO voltage level from the component to a test card.
fbscan_extogsig_b	I	O	Fabric Boundary Scan Extest Toggle Signal bar. This signal provides the toggling signal source when the EXTEST_TOGGLE instruction is enabled.
fbscan_d6init	I	O	Fabric Boundary Scan "dot6" Initialization Signal. This signal will initialize or clear the hysteresis memory (depending on the implement choice) in the IO circuit that captures detected values on the pins.
fbscan_d6select	I	O	Fabric Boundary Scan "dot 6" Select: This signal enables the test circuitry for dot6 test mode and indicates when either of the two train or pulse 1149.6 modes are active.
fbscan_d6actestsig_b	I	O	Fabric Boundary Scan "dot 6" AC Test Signal bar: This signal enables the 1149.6 test mode for use with IR instructions EXTEST_TRAIN and EXTEST_PULSE to perform the boundary scan test function for AC-coupled differential IOs.
fbscan_intest_mode	I	O	Fabric Boundary Scan Intest Mode: This signal enables an INTEST boundary-scan test mode. This is rarely used but available to SoCs.
abscan_tdo	O	O	Agent Boundary Scan Test Data Output. This signal is the test data output for this agent's section of the boundary scan chain stitched through the physical layer pins in the hard-IP macro.
abscan_tdo_f	O	O	Agent Boundary Scan Test Data Output on Falling edge. This signal is the test data output that occurs on the falling edge of TCK for this agent's section of the boundary scan chain stitched through the physical layer pins in the hard-IP macro.

3.10 Microbreak Point

3.11 Micro Break Point controller (uBP)

3.11.1 Background

The Micro break point controller (uBP) is a hardware machine which provides the capability to do debug functionality by giving an indication to stop at some point, wait for some delay and then take an action.

In Intel a lot of functional debug is needed, it is very hard to gain full coverage by using scan content only, thus all the flops have to be covered by a lot of tests.

By doing both functional tests and Scan tests, high coverage can be reached with less tests content, resulting in reduced test time.

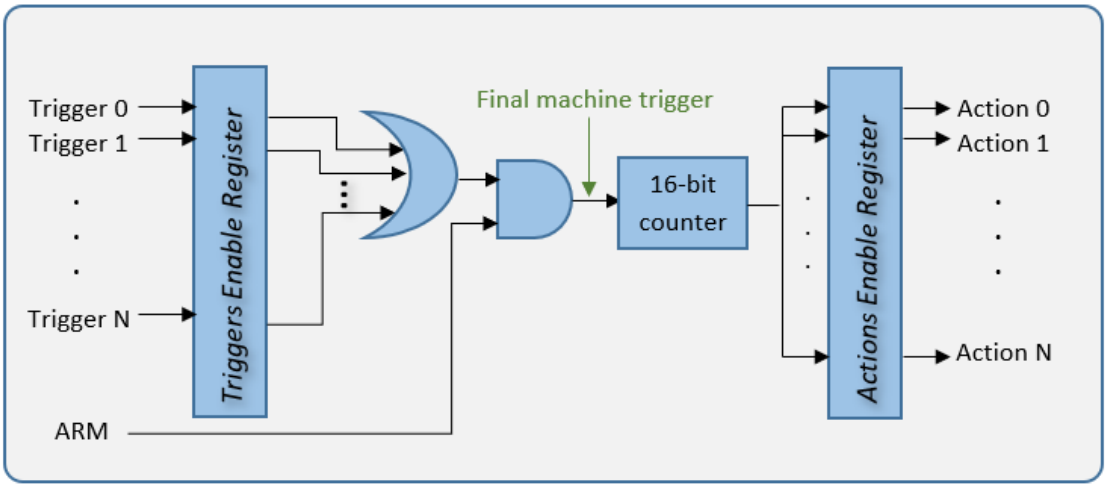
Usually three UBP machines are places per TAP.

3.11.2 UBP Architectural Overview

Each micro break point controller is built from a trigger enables register, a 16-bit counter, an action enables register and ARM/enable bit.

Each controller can be trigger by other controllers in the same TAP by using a dedicate action or by other TAP controllers by using the MBP ring.

Figure 27. uBP Building Block



- Triggers:
  - Each uBP machine includes triggers register.
  - The controller can get N triggers where each trigger must be connected to a critical signal (or any other signal with a specific meaning inside the design).
  - For example, a trigger could be functional reset, PLL lock, etc…
  - From the triggers side everything is ORed, and a single trigger will trigger the machine according to the triggers state.
- Actions:
  - Each uBP machine includes an action register.

- Actions signals are part of the design, it means each action signal must be connected to a critical\functional signal.
- For example, an action signal could be used for stopping some clock or forcing reset.
- Both triggers and actions are part of the design, it means each trigger/action must be connected by the design team in advance.

- Counter:
  - Each uBP machine includes 16-bit counter, which is a programable delay between trigger and action.
  - The counter is 16 bit which means that delay can be ran up to 16K cycles.
- ARM:
- Controller enable bit.
- The ARM feature can be also used for triggering another machine within the same TAP.

3.11.3 Break Point Controllers TAP Commands

Each TAP includes several commands which are used to setup and configure the micro break point controller.

In the following commands, "Y" stands for machine number 0,1, …

3.11.3.1 BRKPTLENY TAP Command

Each BRKPTCL register has an enable bit that can be accessed with BRKPTEN TAP command (shift DR of one bit). Note that the controller can be enabled on the fly by preprograming the BRKPTCTL Armed bits and trigger it to be enable with MBP.

Table 8. TAP Micro Break Point Enable Register

Bit	Width	Label	Comment	Reset Value at CaptureDR
0	1	ENABLE	Disable the machine Enable the machine	0

3.11.3.2 BRKPTCY TAP Command

With BRKPTCTL TAP command, each UBP controller can be programmed. The controller register includes controller behavior controls bits, triggers, actions, armed/disarm MBP triggers and counter configuration. To enable the controller programming, the user must enable the uBP controller with the appropriate X_BRKPTENX TAP command.

Table 9. TAP Micro Break Point Control Register

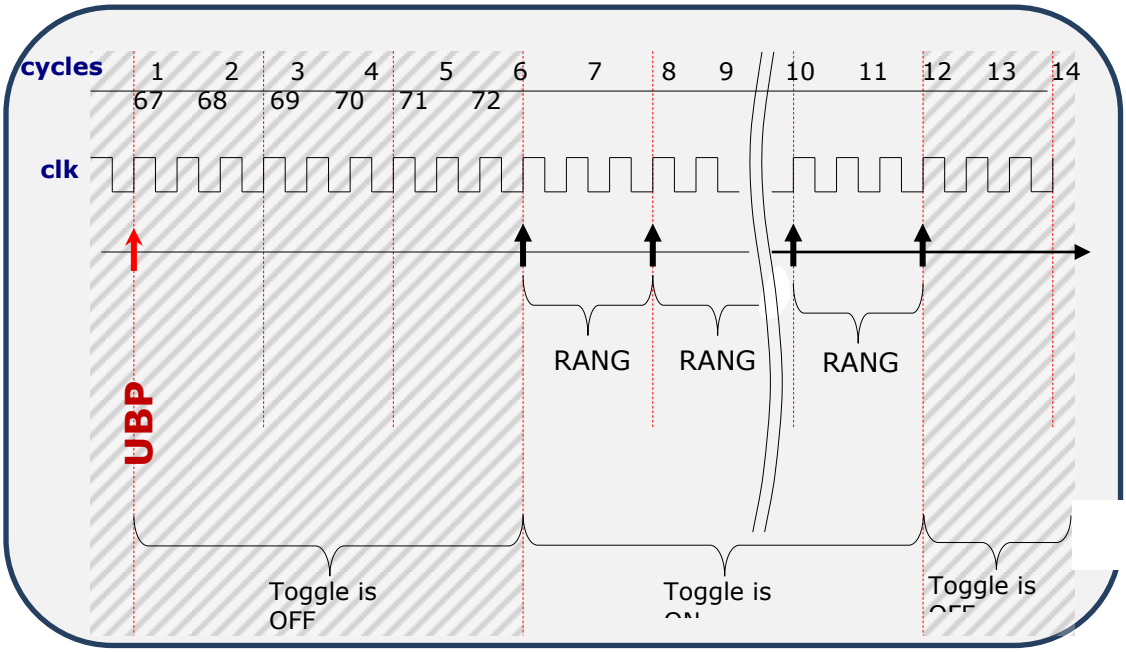
Bit	Width	Label	Reset Value at CaptureDR
0:0	1	CONTROLS.LOOP_MODE	0
1:1	1	CONTROLS.COUNTING_MODE	0
2:2	1	CONTROLS.RELOAD	0
3:3	1	CONTROLS.TRIGGER_DETECT_EDGE	0
4:4	1	CONTROLS.TRIGGER_DETECT_MODE	0
5:5	1	CONTROLS.TRIGGER_ LOGIC_SELECT	0
6:6	1	CONTROLS.TRIGGERS_NOW	0

Bit	Width	Label	Reset Value at CaptureDR
7:7	1	CONTROLS.TRIGGERS_NOW_STICKY	0
8:8	1	CONTROLS.PULSE_MODE	0
9:9	1	AUTOINC_EN	0
25:10	16	COUNTER.VALUE	0
29:26	4	ACTIONS.MBP	0
32:30	3	ACTIONS.TOGGLE_CONTROLLER_ARM	0
33:33	1	ACTIONS.DEBUGCOUNTER_TOGGLE	0
37:34	4	TRIGGERS.MBP	0
39:38	2	TRIGGERS.DEBUGCOUNTER_MATCH	0
40:40	1	BRK_ARM.MBP_0	0
41:41	1	BRK_ARM.MBP_1	0
42:42	1	BRK_ARM.MBP_2	0
43:43	1	BRK_ARM.MBP_3	0
44:44	1	BRK_DISARM.MBP_0	0
45:45	1	BRK_DISARM.MBP_1	0
46:46	1	BRK_DISARM.MBP_2	0
47:47	1	BRK_DISARM.MBP_3	0

3.11.3.3 Micro Break Point Controls Description

- LOOP_MODE:
  - LOOP_MODE = 1: loop mode. the first incoming trigger will start the counting, and at counter equal to zero the uBP will drive an action and reload the counter to the LOAD_COUNTER value and Counting to zero over and over until the uBP is disarmed.
  - LOOP_MODE = 0: normal mode (one action mode). the incoming trigger will start the counting, and at counter equal to zero the uBP will drive an action, reload the counter with the LOAD_COUNTER value and disarm the controller.

Figure 28. ReTrigger uBP Machine(itself). Example: setting uBP action toggle every 4 cycles



- COUNTING_MODE:
  - COUNTING_MODE = 1: count events, each incoming trigger will cause the counter to decrease by one. When counter reaches zero (After LOAD_COUNTER triggers), the uBP will drive an action.
  - COUNTING_MODE = 0: count local clock cycles, first incoming trigger will start the counting at the local clock (each local clock cycle counter is decrease by one). When counter reaches zero it will drive an action.
- RELOAD:
  - What to do in case of a second trigger while UBP controller is counting (already triggered):
  - RELOAD = 1: next counting value will be equal to LOAD_COUNTER value.
  - RELOAD = 0: next counting value will be equal to 'counter - 1' - uBP is ignoring the 2nd trigger.
- DETECT_EDGE and DETECT_MODE:
  - DETECT_EDGE & DETECT_MODE bits are selecting whether way incoming trigger signal will be treated as a trigger or not. DETECT_MODE selects between edge detection and level detection.
- DETECT_EDGE, DETECT_MODE Values:
  - 00 - Level low - triggers the controller while trigger signal value is equal 0.
  - 01 - Falling edge - triggers the controller at the falling edge of trigger signal.
  - 10 - Level high - triggers the controller while trigger signal value is equal 1.
  - 11 - Rising edge - triggers the controller at the rising edge of trigger signal.

- TRIGGER_LOGIG_SEL:
  - 0 - OR mode: UBP will be triggered in case one of the enabled triggers is identified as a trigger
  - 1 - AND mode: UBP will be triggered in case all of the enabled triggers are identified on the same cycle.
- TRIGGERS_NOW:
  - Trigger UBP controller once in every 2-3 cycle (not deterministic) after BRKPTCTL UpdateDR in case controller is armed, in case the controller status is disarm triggers now will be executed one cycle after the controller has been armed.
  - TRIGGERS_NOW = 1: trigger the controller as describe in the paragraph.
  - TRIGGERS_NOW = 0: do nothing.
- TRIGGERS_NOW_STICKY:
  - Same as TRIGGERS_NOW but it is sticky - meaning we can use it more than one time, every time the micro break point controller is armed it will automatically be triggered.
- PULSE_MODE:
  - PULSE_MODE = 0: drive action when UBP counter = 0- action width is one clock cycle
  - PULSE_MODE = 1: drive action when UBP counter >=0 - action width is equal to counter value + 1.
- AUTOINC_EN:
  - In Component Debug automations flows we are using micro break point counter to scan parts of the test (Scan out, LCP …) by running the test X times with deferent sequential counter value (test 0 - counter =0, test 1 - counter 1, …., test X - counter = X). In order to do it we are modifying the test (changing counter value in BRKPTCTL TAP command) each time we are running the test. Tester memory modifications are very slow compare to the test execution, meaning we are spending more time on modifying the pattern than we are spending on the test execution.
  - AUTOINC_EN = 1: at BRKPTCTL UpdateDR LOAD COUNTER bits will be updated with counter + 1 that latched at TAP reset.
  - AUTOINC_EN = 0: at BRKPTCTL UpdateDR LOAD COUNTER bits will be updated with user value that shifted into the common data shift register.
  - At TAP reset the current BRKPTCTL (LOAD COUNTER value – 1) is latched. Note that at the first reset it will latch unknown value and therefore user should always use AUTOINC_EN = 0 at the first time.
- Usage mode:
  - Run first test with the following pattern configuration:
  - BRKPTCTL LOAD COUNTER = desired value, AUTOINC_EN = 0.
  - Modify the pattern to BRKPTCTL LOAD COUNTER = desired value, AUTOINC_EN = 1.
  - Run the test X+1 times.

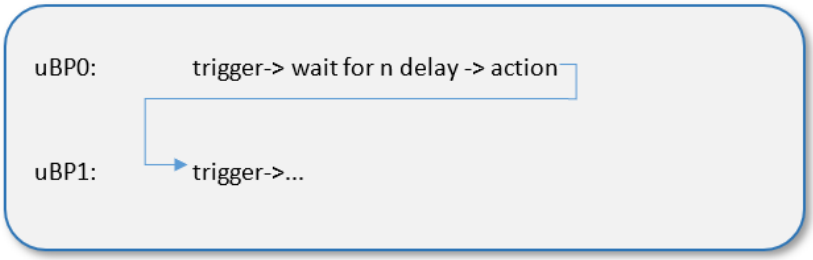
Note: Between each two test executions, VCC is hold on and TRST is set on the beginning of the test, therefore FF holds the previous values (unless FFs are reset) and TRST is latching previous test LOAD COUNTER value + 1.

- ACTIONS:
  - Actions are list of enable bits, which each of these bits are ANDed with UBP action signal (action signal is pulsed for one local clock cycle every time the counter reached

to zero). Each ANDed result signals is used for a specific task, for example LCP pulse action or scan out clock pulse action. ACTIONS range field is dedicate for each tap.

- TOGGLE_CONTROLLER_ARM:
  - ACTIONS enable bits include a common action ACTIONS_TOGGLE_CONTROLER_ARM[X] where X=0, 1, …, Number of controller - 1. With this action a specific controller can arm or disarm the neighbors UBP controller located in the same TAP or itself.

Figure 29. ARM Second Machine. Example: An Action from uBP0 can Drive a Trigger to uBP1



Note: uBP0 and uBP1 must be located in the same TAP.

- TRIGGERS:
  - Triggers are a list of enable bits, which each one of these bits are ANDed with a specific trigger signal. The result of all ANDed signal are ORed, the ORed result is the UBP controller trigger signal that goes to the trigger detect unit (edge, level, … modes detection as describe previously in DETECT_MODE & DETECT_EDGE control bit). Example for trigger are MBPIN [7:0] signals.
- ARM and DISARM TRIGGER:
  - Arm & DisArm trigger gives an option to Arm/Disarm a controller through MBP pins.
  - For each MBP[x] pin there are two enable bits named ARMX and DISARMX.
  - ARM[x] and DISARM[x] bits work as flowed
  - 00 - save previous controller Arm state
  - 01 - DisArm the controller
  - 10 - Arm the controller
  - 11 - Toggle previous controller Arm state (Arm → DisArm / DisArm → Arm)

3.11.3.4 BRKPTSTATE TAP Command

Read the controller state - current counter value, triggers sticky bits and action accrued.

Once BRKPTSTATE register is read, trigger history is erased. To get the correct trigger/action accrued history, a dummy BRKPTSTATE tap command should be used before the test range.

The state register enables us to determine what the current state of the break point controller is. This includes: a set of triggers sticky bits indicating which triggers have been occurred, a bit for action occurs and 16bits for current counter value.

BRKPTSTATE CaptureDR TAP FSM state captures the data from the BRKPTSTATE to the TAP common shift register and clear it, meaning triggers and action occur history is restarting. Note that this register has no TAP reset or powergood reset which means that after power

good part of the bits are X's and can be used and therefore user must read the BRKPTSTATE register before using it.

Table 10. TAP Micro Break Point Status Register

Bit	Width	Label	Comment	Reset Value at CaptureDR
15:0	16	COUNTER.VALUE	UBP counter current value	No reset
16	1	ACTION_OCCURED	UBP action occurred sticky bit, clean after read	0
20:17	4	TRIGGERS.MBP	MBP trigger	0
22:21	2	TRIGGERS.DEBUGCOUNTER_MATCH	UBP debug counter match	0

Note: This table is not completed, once an external trigger occurs; it will be added as a new field to this command DRs. For example

3.12 Debug Counter

32-bit counter running on the local clock, usually starts running when functional reset occurs. On tester there is no capability to view internal signals or do specific propping, thus this counter is a hardware feature which provides the capability to relate between simulation and tester cases.

By configuring the debug counter with a MATCH VALUE, a trigger will be sent once the counter reach this value. The counter value when a trigger occurs indicates in which local clock cycle this trigger happened.

Table 11. Debug Counter TAP Command Description

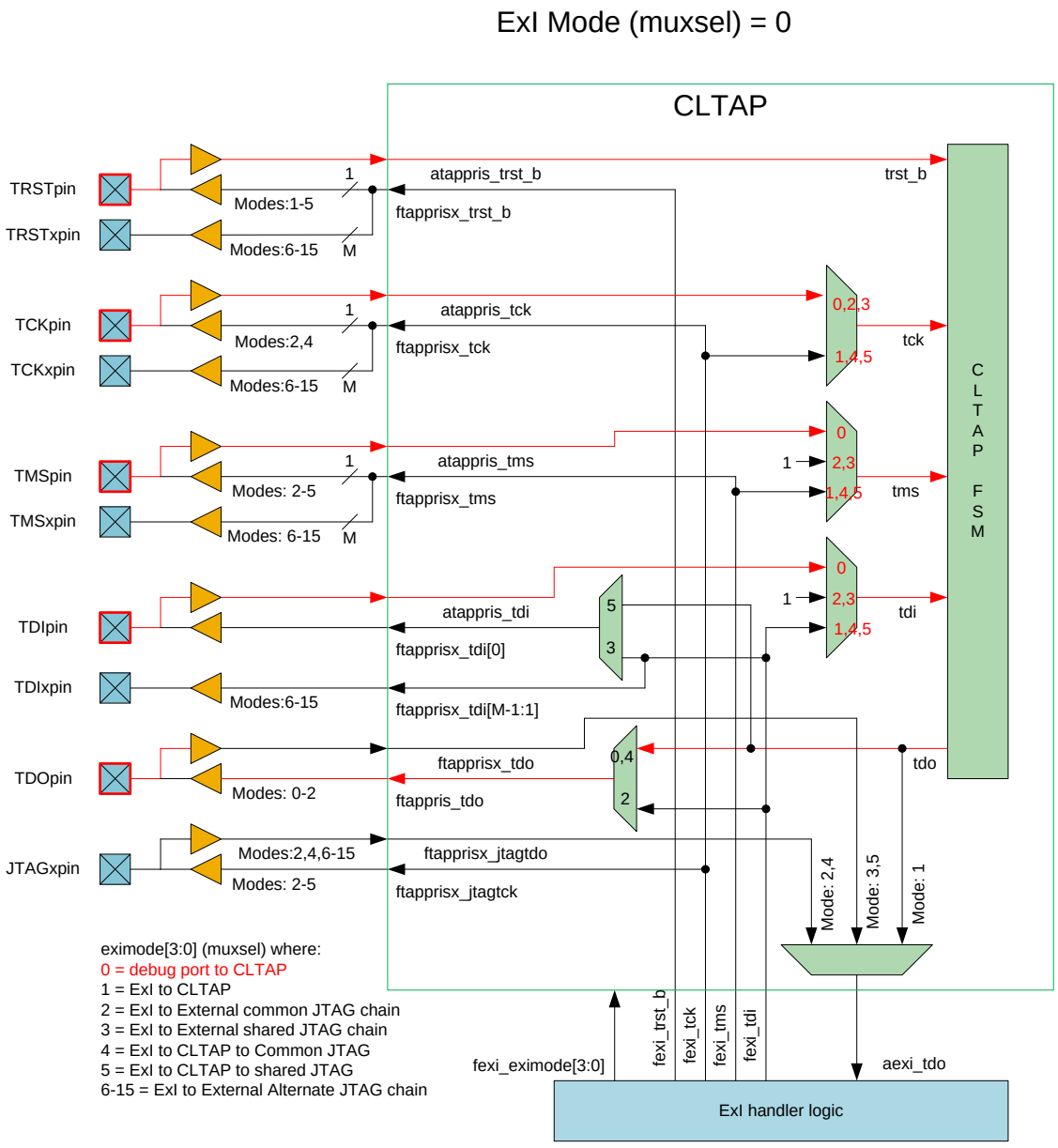
Width	Label	Comment	Reset Value
30	DR. COUNTER_MATCH	Write: debug counter match value. Read: current debug counter match value.	0
1	DR. COUNTING_MODE	0 – start counting immediately 1 – wait for uBP trigger	0
1	DR. RESET_MODE	Reset the counter at UpdateDR	0
1	DR.FUNCTIONAL_RESET_DIS	0 – start to count when functional reset=0 1 – disable functional reset.	0
1	DR.DEBUG_COUNTER_LOOP	Restart counter on match	0

3.13 CLTAP ExI Use Models

3.13.1 Use Model for ExI mode = 0

This mode is defined as the expected normal TAP operation for an SoC. The assign CLTAP package pins will communicate directly with the CLTAP. The red highlight shows the active path from the pins to the CLTAP.

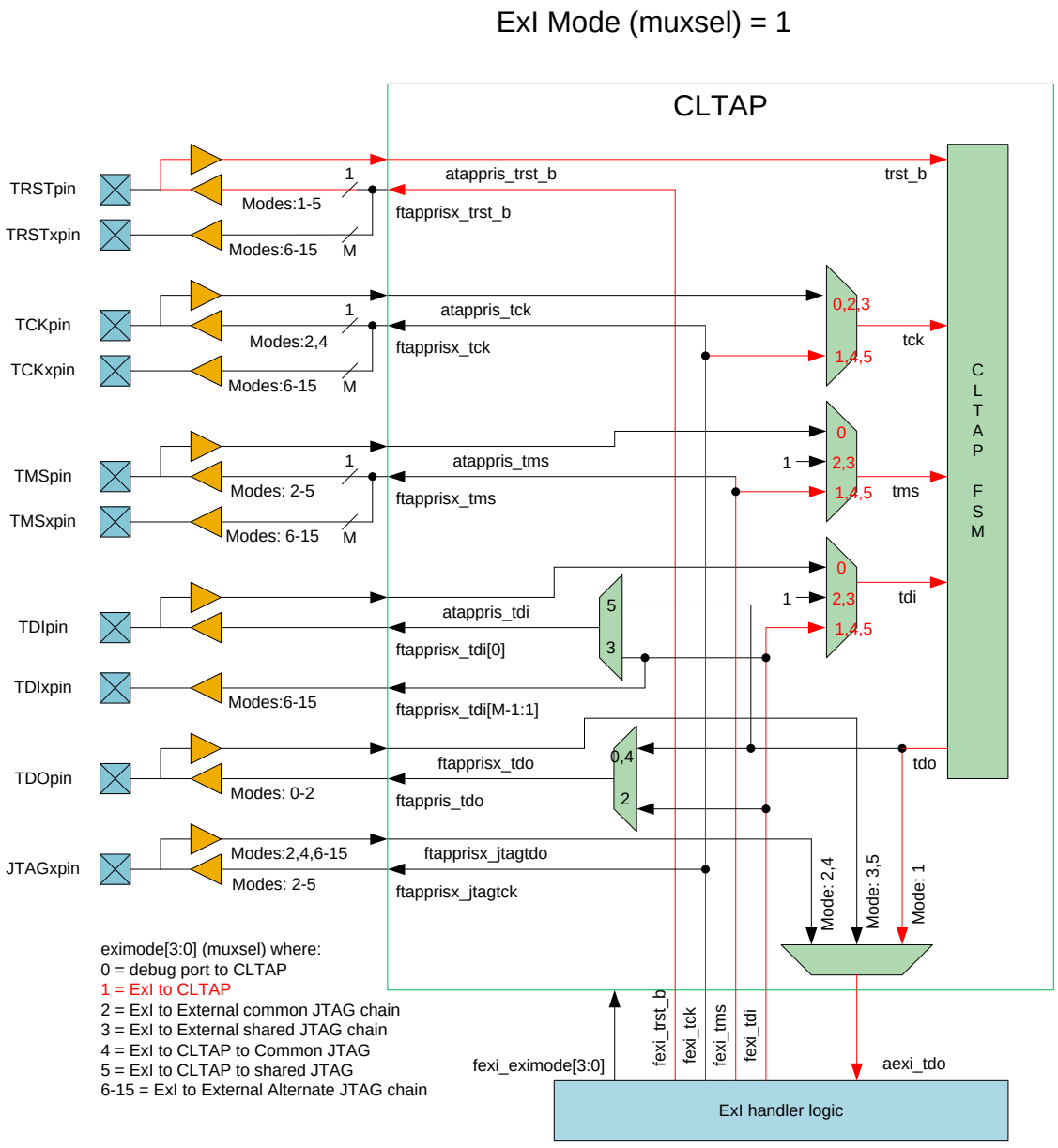
Figure 30. Use Model for ExI mode = 0



3.13.2 Use Model for ExI Mode = 1

This mode provides a path from the BSSB through the ExI handler to communicate with the CLTAP.

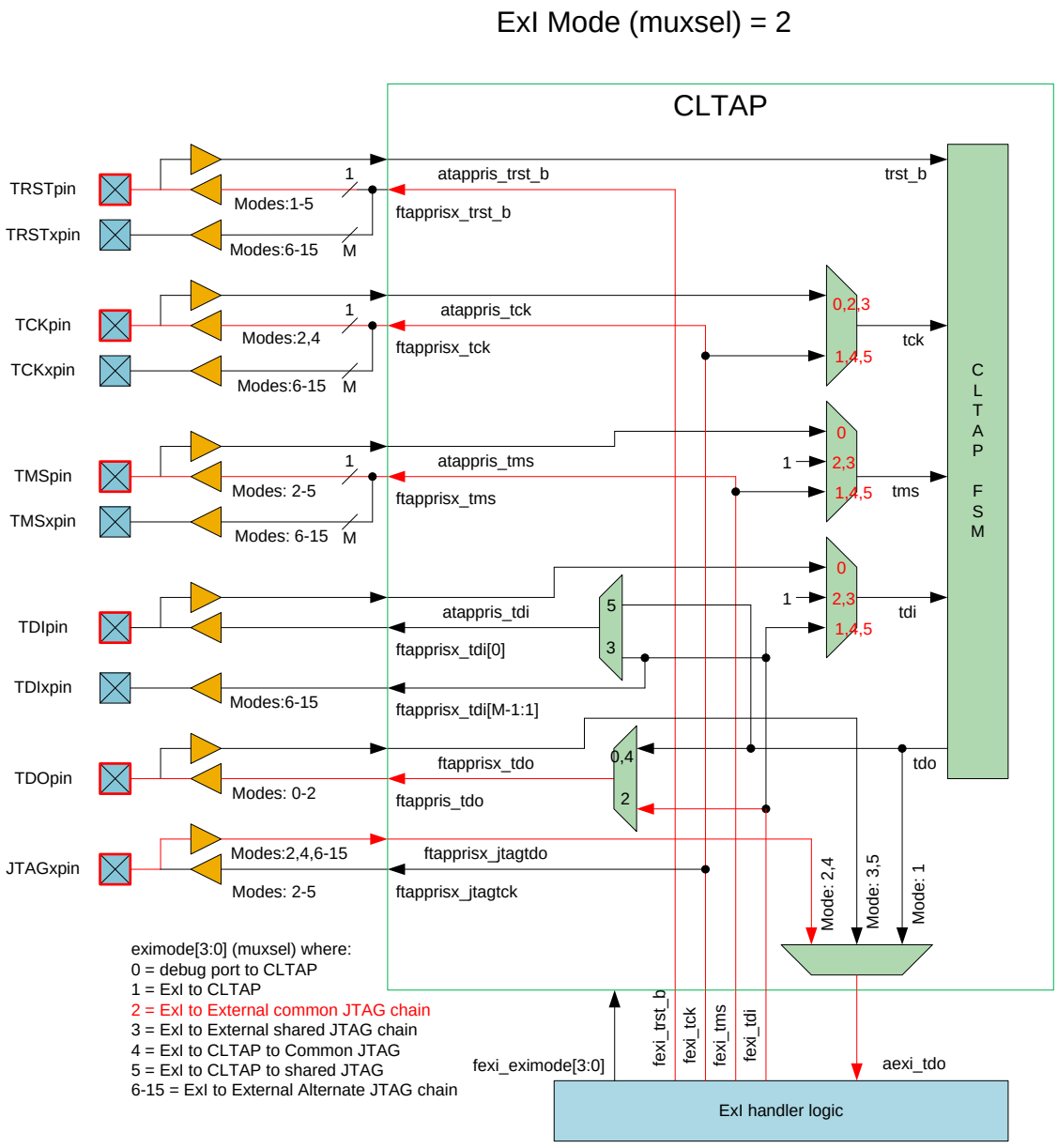
Figure 31. Use Model for ExI Mode = 1



3.13.3 Use Model for ExI Mode = 2

The BSSB or USB device communicates with a TAP on the platform outside of the SoC. This is a common JTAG use model where there is one chain on the platform. The ftapprisx_tdo will drive the TDI of a component on the platform. The chain acts as Normal in-series with TAP connectivity from the SoC to a component on the platform.

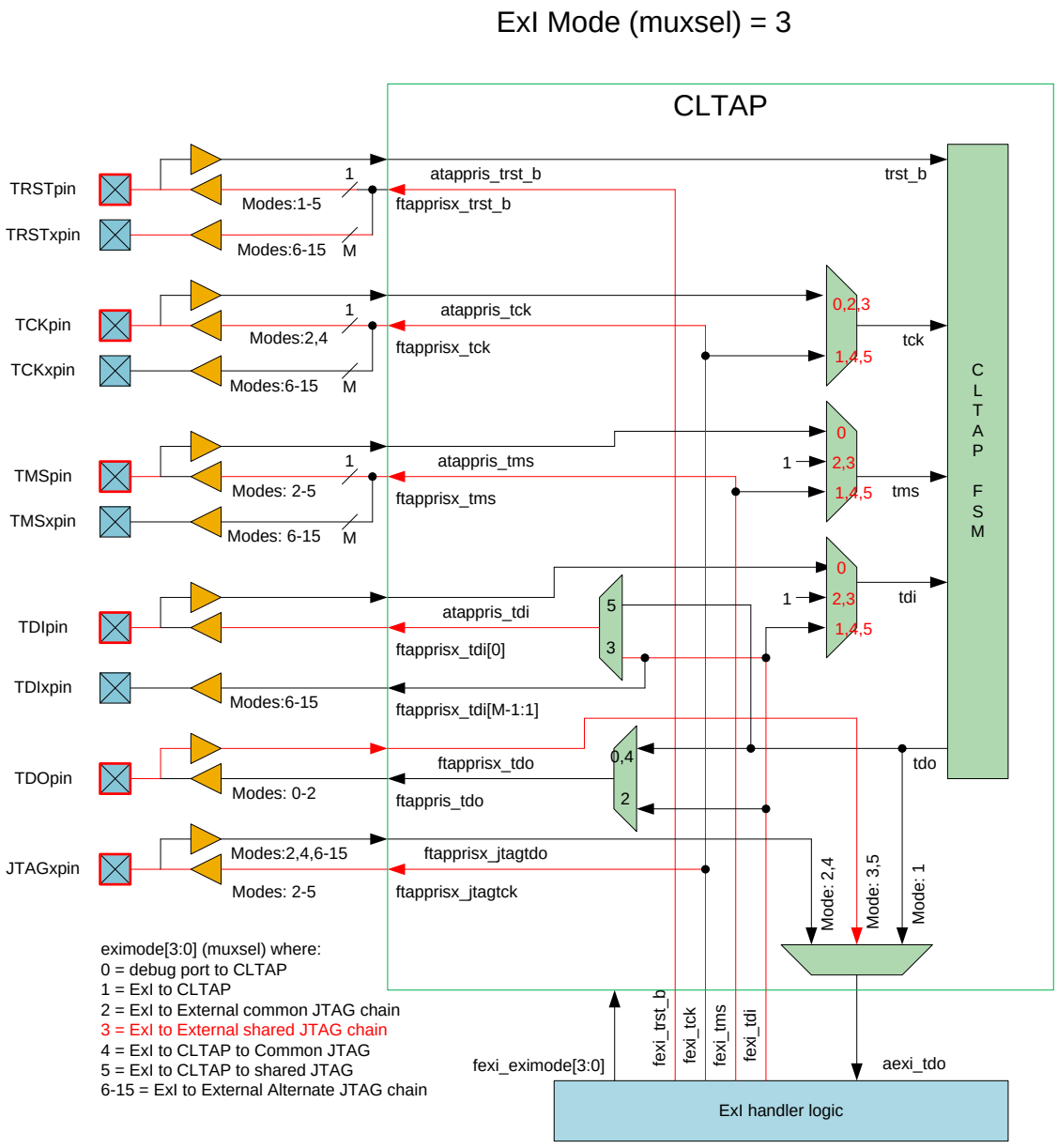
Figure 32. Use Model for ExI Mode = 2



3.13.4 Use Model for ExI Mode = 3

This mode supports more than one chain on the platform. There are two separate TCK outputs to drive two chains.

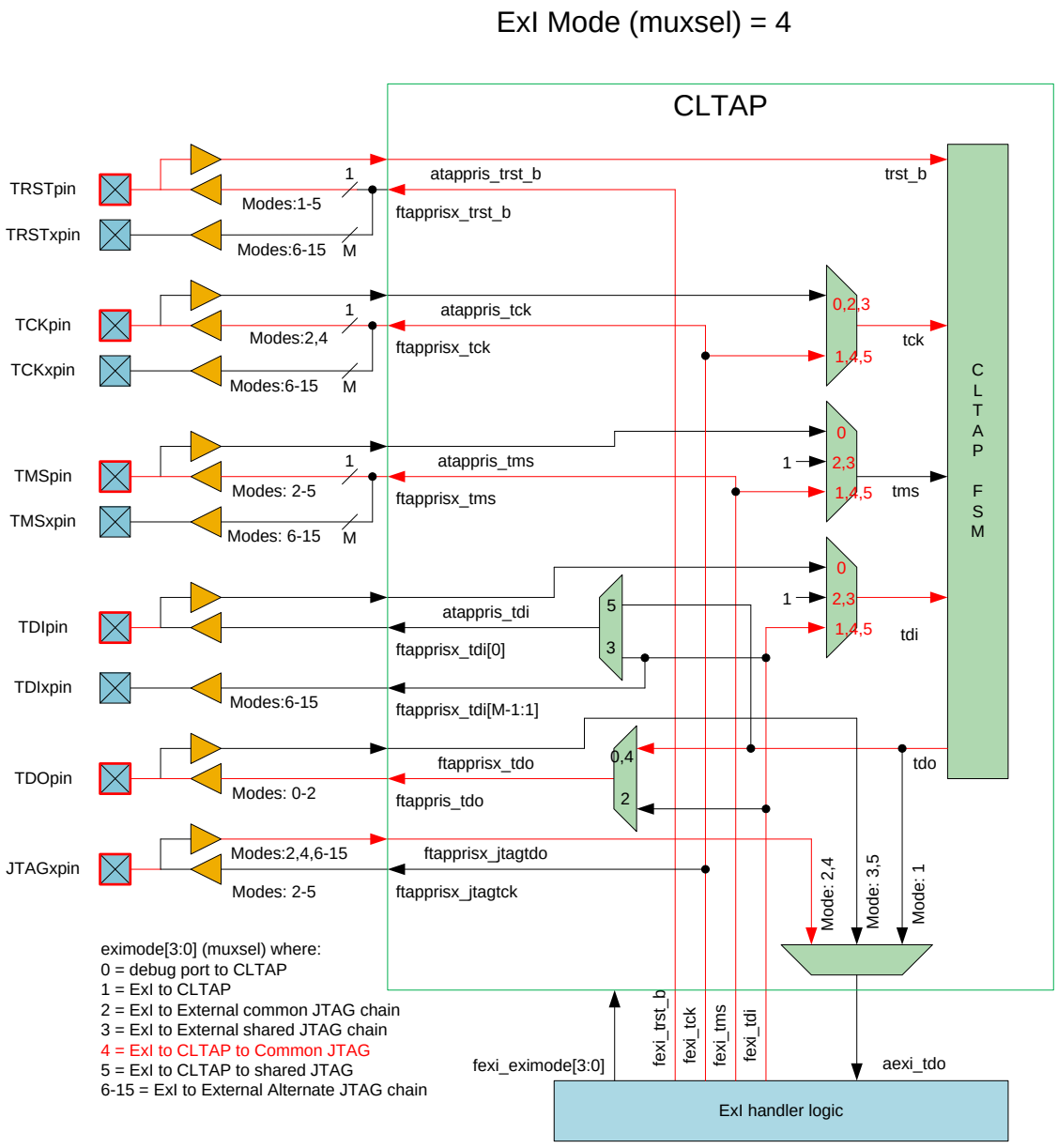
Figure 33. Use Model for ExI Mode = 3



3.13.5 Use Model for ExI Mode = 4

This is the same as mode = 2 but it includes the CLTAP of the SoC.

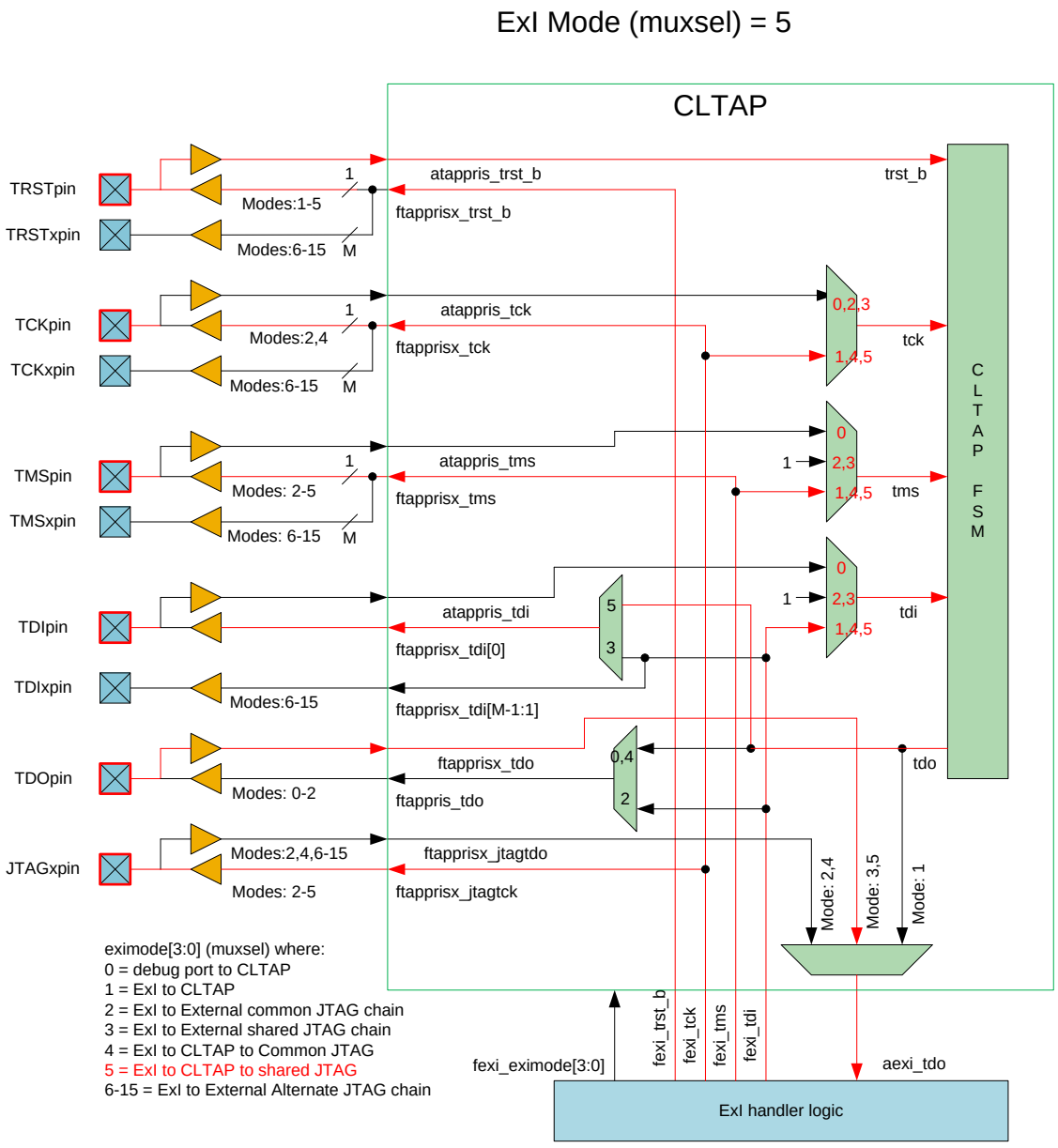
Figure 34. Use Model for ExI Mode = 4



3.13.6 Use Model for ExI Mode = 5

This is the same as mode = 3 but it includes the CLTAP of the SoC.

Figure 35. Use Model for ExI Mode = 5



3.14 TAP to Sideband DFx IP

Table 12. Chapter Revision History

Revision Number	Description	Revision Date
0.90_rc2a	Initial release	February 2012
0.90_rc4	Created this chapter for this feature.	March 2012
rev0.90_rc5	Updated diagram with NTI	May2012

Revision Number	Description	Revision Date
rev0.90_rc7	Updated the security section Added a TAP opcode and TDR for the parallel function ID. Updated the parallel message section with the data transfer format. Updated the parameters, strap and pin interface section of the parallel message bus.	August 2012
Rev0.90.1	Update the security feature for limiting the designation to a fixed number of sideband endpoints Added root space field for SoCs compliant to IOSF rev1.1 Added side_pok signal which is control by a TAP bit or fixed to logic 1.	February 2012

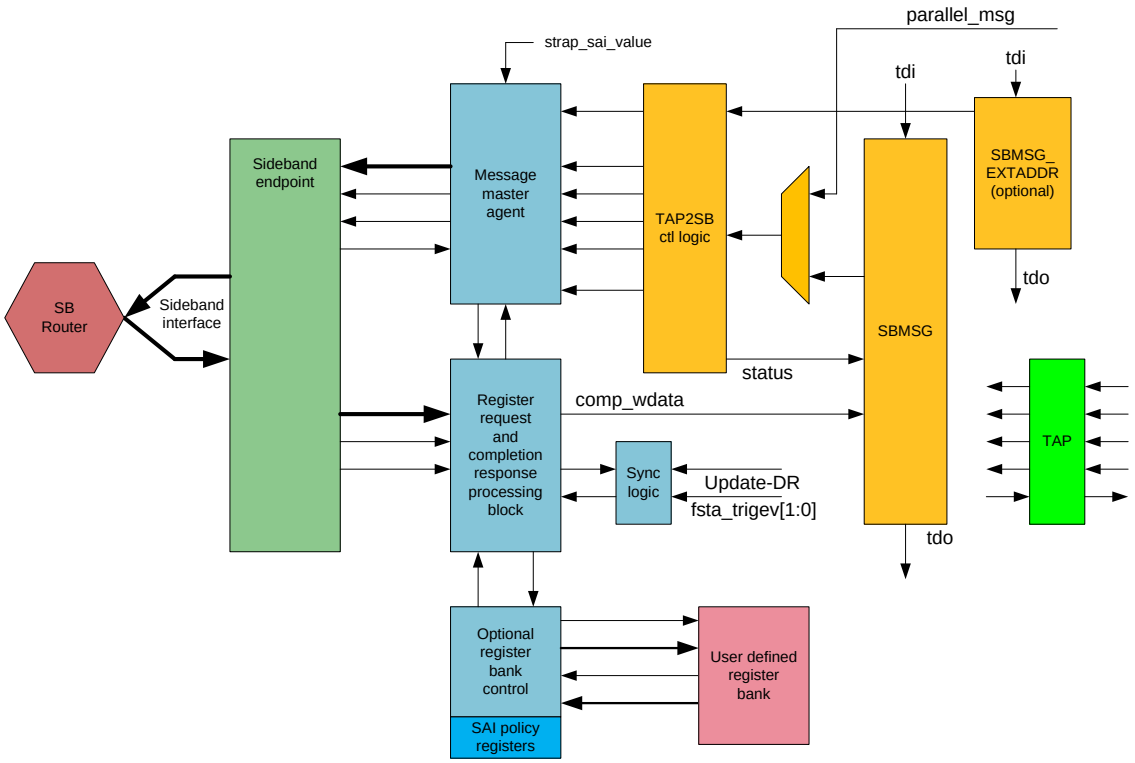
This feature is an out-of-band direct access to the IOSF Sideband message interface for debug and validation purposes. The TAP register definition is agnostic to the message type that the user sends over the Sideband router network but not all message types may be implemented. The primary purpose of this definition is provide the TAP host controller (ITP) with a known programming model to enable any message type such that the definition of the TDR remains constant.

The Sideband message test/debug register (SBMSG) is support by three opcodes. The first opcode is a Sideband message "Go" instruction (SBMSGGO) that enables a shift operation to load a desired request into the TDR then an Update launches the operation. The second opcode is a response instruction to capture the completion response and status bit then shift them out for reading or checking conditions. There is no update with this sequence to prevent an unnecessary initiation of another transaction. A third opcode only shifts the data into the register and waits for a trigger event to launch the transaction. This is a mechanism to allow an internal event or trigger source to coordinate the action of when to send the request rather than an external software operation.

Figure 36 is a block diagram describing the pieces of the TAP to IOSF SB (abbreviated as TAP2SB) transaction generator. The standard test/debug register (SBMSG) is general use case where the address is only 16bits. An extended address is allowed but it must be implementation dependent with its opcode in the user-defined space outside of the 0x0-0x1f opcode space defined in this spec. The message master agent packetizes the request and issues a 4-byte transfer when the end point indicates when it can accept one. A completion response block decodes the target message bus from the Sideband endpoint and holds the data until a SBMSGRSP instruction is executed where the data can be captured and shifted out. This block also contains some miscellaneous logic to detect some basic status information on the condition of the transaction generator. Basic synchronization ensures that the update from the TCK domain or the trigger event from the VISA clock domain will properly launch the request in the Sideband endpoint's clock domain.

An optional register control bank allows the user to connect their project defined registers into this module. This feature is due to the construction of the TAP-to-SB block. This block must come with a sideband endpoint. There is no convenient way of separating it from the unit. To reduce the number of endpoints in a cluster level DFx unit, this TAP-to-SB IP-block provides a register access port for the integration team to include their user defined set of registers.

Figure 36. TAP to IOSF Sideband Access Block Diagram



The TAP register definition can be found in section 3.14.10.1 for the slave TAP. Listed here is an explanation of the byte fields that are loaded into the register. When the SBMSGGO instruction is enabled the data shifted into the SBMSGGO is loaded into an update register and a request is made to the sideband endpoint. If the desired transaction was a read then the SBMSGRSP instruction is loaded which enables capturing the data completion response into the shift register which can then be shifted out. More detailed information about this feature can be found in section 3.14.7 and in this IP-block's MAS document from the SEG repository.

- Reg Byte 0: Miscellaneous control and status bits [7:0].
- Message trigger select[7:6]: If the SBMSGNOGO opcode is used then the request is shifted in the SBMSG TDR and waits for a trigger event to assert which launches the transaction. The triggering feature is only active when one of the trigger events is selected and the SBMSGNOGO instruction is active.
- Two trigger outputs from Lakemore's SoC Trigger Processor (STP)
- Unused bit[5]: reserved for future use.
- 64-bit data enable: This bit enables a 64-bit data payload transaction for a read or write message.
- Non-posted write[3]: This bit indicates if the transaction is a posted or non-posted for a write transaction. The transaction may be of any message type; simple message with or without data or a configuration access message.
- Broadcast enable[2]: The broadcasten bit sets the value of 0xFE in the source ID field of the protocol. Refer to the IOSF rev1.0+ spec for more information.
- Status[1:0]: Two bits to indicate the current status of the current transaction. The conditions for setting the bits are based on what the TAP2SB sees as the transaction is being processed.

- Reg byte1: sbmsgbyte0: Destination ID of the endpoint on the router network for which this transaction is intended to go.
- Reg byte2: sbmsgbyte1: Opcode: This determines the message type and if it is a global or implementation specific code. Refer to the IOSF spec rev1.1 or later for more information.
  - TAP2SB allows opcodes 0x20 and 0x21 to be written in the sbmsgbyte1 field. This has two implications, one is restriction due to stringent security requirements and another more important scenario is the potential to hang the Sideband network due to a receiving a completion from a non-existent non-posted request. This revision (SoC TAP HAS rev0.90.2) will disallow these opcodes and not initiate a transaction with these values set. A future version of the SIP IP will detect and drop the transaction before the TAP2SB endpoint communicates with the router.
- Reg byte3: sbmsgbyte2:
- Tag: This is a unique value that is used to match the data completion response. Since the TAP to SB feature can only handle a single outstanding transaction at a time only 000b is used.
  - The tag should be stored and matched against a completion response for transaction with data completions. Since there are only one outstanding request for the TAP to SB feature this should always match, if it doesn't then it is an error.
- Bar: This is only applicable to memory or IO mapped endpoints. The user writes the appropriate bar for which the target endpoint has a set of registers that this feature is requesting access to. A single endpoint may have more than one bar.
- Addrlen: This is the address length of the request. It is optional to implement the SBEXTADDR
  - 0: 16-bit address
  - 1: 48-bit address
- EH: Expanded header bit. If this bit is set then an expanded header with SAI bits will be sent.
  - If the SAI_AWARE parameter is logic 1 then the logic block sets the EH bit for all transactions.
  - The SAI bits are available as straps to the IP-block so that it can be set by the integration team.
- Reg byte4: sbmsgbyte3: SB message byte 3:
  - For messages with data and for completion with data this will be data[7:0]
  - For register read and write accesses only the first byte enables are used. The last byte enable field is cleared to zero.
- Reg byte5: sbmsgbyte4:
  - For messages with data and for completion with data this will be data[15:8]
  - For register read and write accesses this will be the function ID (FID). An endpoint may have multiple functions each with its own ID for a given endpoint (dest ID).
- Reg byte6: sbmsgbyte5:
  - For messages with data and for completion with data this will be data[23:16].
  - For register read and write accesses this will be address[7:0].
- Reg byte7: sbmsgbyte6:
  - For messages with data and for completion with data this will be data[31:24].
  - For register read and write accesses this will be address[15:8].

- Reg byte8: sbmsgbyte7:
  - For register write accesses this will be the write data[7:0].
  - For 64-bit write messages only (64-bit data enabled), this byte is for data[39:32].
- Reg byte9: sbmsgbyte8:
  - For register write accesses this will be the write data[15:8].
  - For 64-bit write messages only (64-bit data enabled), this byte is for data[47:40].
- Reg byte10: sbmsgbyte9:
  - For register write accesses this will be the write data[23:16].
  - For 64-bit write messages only (64-bit data enabled), this byte is for data[55:48].
- Reg byte11: sbmsgbyte10:
  - For register write accesses this will be the write data[31:24].
  - For 64-bit write messages only (64-bit data enabled), this byte is for data[63:56].

3.14.1 TAP2SB Feature List

The following feature list describes the high level capabilities of this IP-block.

- Primarily it provides configuration access from TAP through the Sideband network to the soft and hard IP-blocks to read and write registers for test and debug.
- Although it is primarily used for configuration, it also do message reads and writes with limitations.
- Configuration reads and writes are limited to a 32-bit data value.
- A 64-bit write message is supported. However, this is the only message type that will handle this quantity of data.
- There are three opcodes that control one message test data register.
- The SBMSGGO instruction allows the user to shift in a transaction and execute the request after the Update_DR state. This is the normal mode of operation.
- The SBMSGRSP is the instruction to shift out the data completion from the SBMSG register.
- The SBMSGNOGO will allow a transaction to be shifted into the SBMSG register without executing a transaction request on the Sideband. This use model expects a hardware trigger event to launch the transaction. This assumes that the trigger event is configured in a Cluster Trigger Block elsewhere in the SoC. The user can select from one of two trigger inputs.
- The TAP2SB IP has the option to support a register bank for DFx or other reasons. The register bank is outside of the TAP2SB and the SoC team integrates the register bank to the TAP2SB IP with a standard parallel register interface.
- The register bank is protected with SAI policy registers.
- The TAP2SB is a master on the Sideband network and optionally supports an SAI policy value as an extended header.
- Up to four SAI values may be strapped on the interface to provide the equivalent of a security locked SAI value, an OEM security unlocked value, an Intel unlocked SAI value and a security unlocked SAI value.
- A parallel message bus to deliver configuration content from the tester through the NTI to the Sideband network.

- A green level security option provides access to a fixed number of sideband endpoints.
- For more information, refer to the TAP2SB integration guideline document.

3.14.2 TAP opcode support

The TAP2SB comes with a slave TAP as part of the IP and the opcodes in Table 13 are used with the SBMSG test data register as the interface between the TAP protocol and the Sideband protocol.

Table 13. TAP2SB Opcode Table

Instruction	Encoding N-bit IR (hex)	Data Register Selected	Default Security Color Level	Description
SBMSGGO	0x30	SBMSGGO	Parameterized (SECURE_ ORANGE)	This instruction provides access to IOSF Sideband (SB) interface from this TAP controller. It will launch a Sideband transaction after an Update_DR and synchronization is completed.
SBMSGRSP	0x31	SBMSGRSP	Parameterized (SECURE_ ORANGE)	This instruction performs a capture and shift operation on the SBMSG register to obtain the contents of the pervious read operation.
SBMSGNOGO	0x32	SBMSGNOGO	Parameterized (SECURE_ ORANGE)	This instruction performs only a shift operation to fill the SBMSG register without initiating a transaction request. This message is executed when the selected input trigger asserts. It is expected that the trigger inputs on the TAP2SB is connected to the SoC Trigger Architecture fabric from a Cluster Trigger Block.
SBEXTADDR	0x33	SBEXTADDR	Parameterized (SECURE_ ORANGE)	This instruction performs a shift and update on the SBEXTHADDR data register. If the extAddrEn bit is logic 1 then when either the SBMSG DR-Update control signal or the selected trigger event is asserted, then a new command request to the sideband endpoint will include the contents of the extended address data register.
POVRSBFID	0x34	POVRSBFID	Fixed: SECURE_RED	This instruction is for enabling the parallel message feature and configuring FID (Sideband Function ID).
SBROOTSPC	0x35	SBROOTSPC	Parameterized (SECURE_ ORANGE)	This instruction is for providing a root space value to support IOSF rev1.1. Also, this register contains an optional bit to manually control side_pok.

3.14.3 Security Features for TAP2SB

The TAP2SB block has several security features depending on which DFx features are enabled. The Security Attributes of Initiator (SAI) value is parameterized to allow security support on transactions sent by the TAP2SB as a master on the sideband network. The user-defined register bank will use SAI as an endpoint to grant or deny access depending on the appropriate SAI value. The TAP opcodes are protected with the slave TAP's embedded DFx secure plug-in.

3.14.3.1 Security for TAP2SB as a Master

There are customer use models requiring the TAP2SB to send transactions to specific sideband endpoints. Since the destination ID field is programmable there must filtering logic to prevent unauthorized access to nodes that require a higher level of security. The logic in the Figure 37 shows the logic necessary to provide TAP to Sideband transactions a fixed set of destination IDs under Security Locked (green access) conditions. This solution requires a DFx secure plug-in that is instantiated with only four DFx features to enable. A list of the parameter values for the plug-in are listed in section 3.14.4.1. For controlling access to any given destination, the SoC team sets the destination ID parameter to identify which Sideband endpoint IDs are allowed by the external customer under security locked policy value. Once the values are set they are synthesized into the design and cannot change at run time. Any ID value should be considered a valid destination value and therefore, the user must set unused IDs with the destEn[n] bit to logic zero. Each destination ID tuple is compared against the destination ID from the output of the TAP update register. The output for the logic compare is then gated by the destination enable bit to determine if this value should be used or not. There is one more qualification which is dependent on the current state of the DFx secure plug-in. If policy is 0x3, 0x6 of 0xF then the TAP2SB is disabled from a security point of view and no transactions are permitted.

If destID[n][7:0] matches and this value is enabled (destEn[n]=1) and the current secure policy is Security Locked (policy = 0000) then this transaction is allowed to continue by setting the posted or non-posted outbound message available signal. If the destination ID doesn’t match or it isn’t enabled or if it is not the Security Locked policy then the TAP2SB logic simply does nothing. The status bits in the SBMSG register will report a value of 2’b00. An example of destination IDs is listed in Table 14.

If the security policy is orange or red (0x7-0xE, 0x2 or 0x4) then the destination comparison logic is bypassed and the user has free access to any Sideband endpoint if the correct policy is set. Meaning if TAP2SB is enabled for Orange access then when policy 0x5 is active then access is permitted.

Note: This only applies to TAP-based requested transactions. The parallel message bus is assumed to be protected in the NTI IP-block also the functional ID field in the TAP register is protected with a red opcode.

TAP2SB provides with a set of four SAI value straps that the SoC integration team can set to the appropriate value based the ID assignment for the component. When SAI is enabled each transaction will send the assign ID value dependent on the corresponding policy. The TAP2SB will select the value based on a fixed policy table. Encoding logic interprets the DFx features enable outputs from the TAP2SB DFx secure plug-in as shown at the bottom of the diagram. A detailed list of which policy enables which SAI value is shown in Table 16.

The IOSD spec defined side_pok signal is parameterized to be either driven from a TAP bit or tied to logic 1. The Sideband root space TAP register contains a bit sbPokEn that is a dummy read/write register bit if the parameter is set to 1 (TPSB_SIDEPOK_IS_TIED_TO_LOGIC1 = 1). Otherwise, it is used to drive the side_pok output pin directly.

If the compile time parameter labeled as SAI_AWARE is set to logic 1 then the expanded header is used on every transaction. The SAI value used is dependent on the current policy and the SAI strap value applied to one of four levels defined. There are only four levels of SAI to reduce complexity.

Figure 37. TAP2SB Security and Misc Control Logic

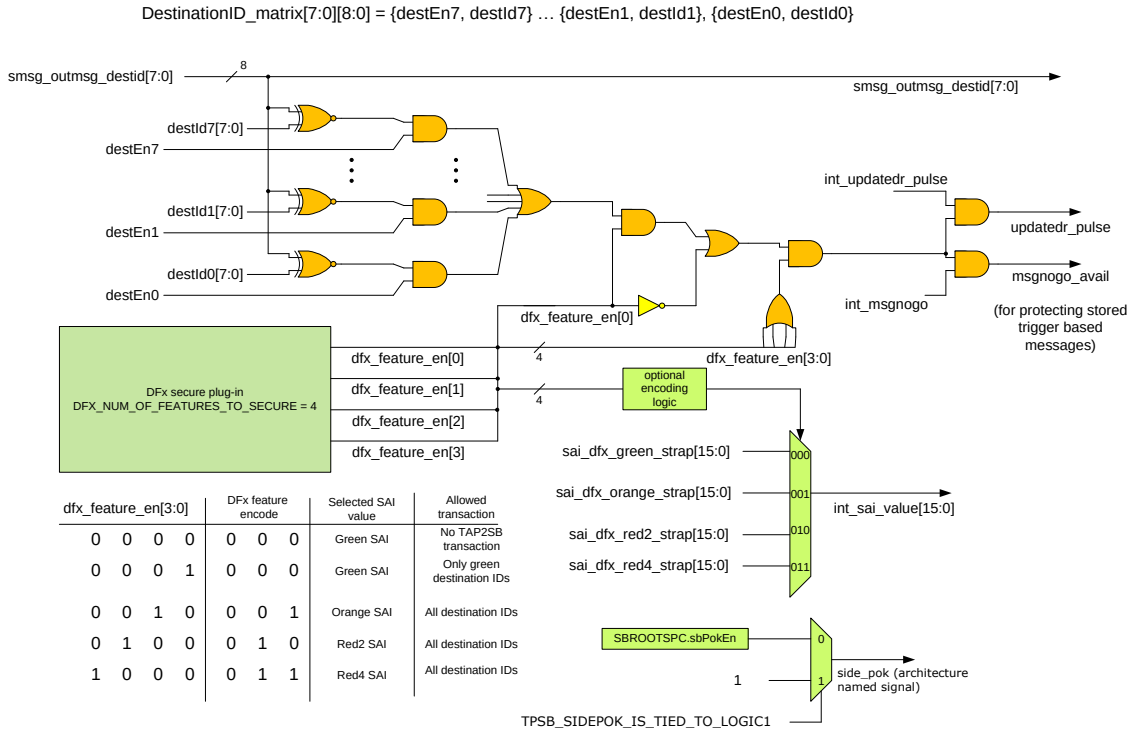


Table 14. Example of Setting Destination IDs for Green SAI Operation

Desired Sideband ID (destID)	Allow Access (destEn[n])	Tuple Value for Destination_matrix	Comments
North Peak = 18h	1	Dest_matrix[0] = {1, 0x18}	Includes VISA register controller
GPIO family 1 = 32h	1	Dest_matrix[0] = {1, 0x32}	
GPIO family 2 = 33h	1	Dest_matrix[0] = {1, 0x33}	
USB3 DeBug Class device (DBC) = 45h	1	Dest_matrix[0] = {1, 0x45}	
Hotham mailbox = 50h	1	Dest_matrix[0] = {1, 0x50}	
Not used	0	Dest_matrix[0] = {0, 0x0}	The destination ID = 0x0 may be a valid ID and therefore any value not used must clear the destEn[n] bit portion of the destination matix tuple
Not used	0	Dest_matrix[0] = {0, 0x0}	
Not used	0	Dest_matrix[0] = {0, 0x0}	

3.14.3.2 Security Features for User Defined Register Bank

The DFx registers are protected by the SAI policy registers. Policy registers determine which secure agent can read or write the DFx registers in the user defined register bank. The TAP2SB feature implements the control logic necessary to manage the register interface but doesn't include the registers themselves. It is up to the integration team to supply the

registers. For the early adapters of the SAI security feature the width of the SAI value is 5-bits. An SAI value represents a unique SAI source. Any Sideband endpoint that can initiate a config transaction on to the Sideband bus is a candidate for checking its access policy. A few examples of a source are; this TAP2SB feature, an IA-core, or a security engine IP. This 5-bit SAI value translates into a 32-bit register for each control, read and write policy register. A logic 1 in a bit position indicates that the binary value representing the SAI source is allowed access to that function. If an SAI value of 0x5 and a write policy register bit[4] = 1 then that source can write any register in the TAP2SB register bank. If it was logic 0 then the write packet would be dropped. The same is true for the read policy register. If a value is logic 0 for a particular bit position then that read transaction would be terminated with an unsupported completion response. A control policy register determines if the read or write policy registers can be updated. Usually, a known secure agent sets these values. A flow chart that describes the actions taken for each value is shown in Figure 38. The address of the policy registers are listed in Table 15.

Figure 38. SAI Flow Chart for TAP2SB Access

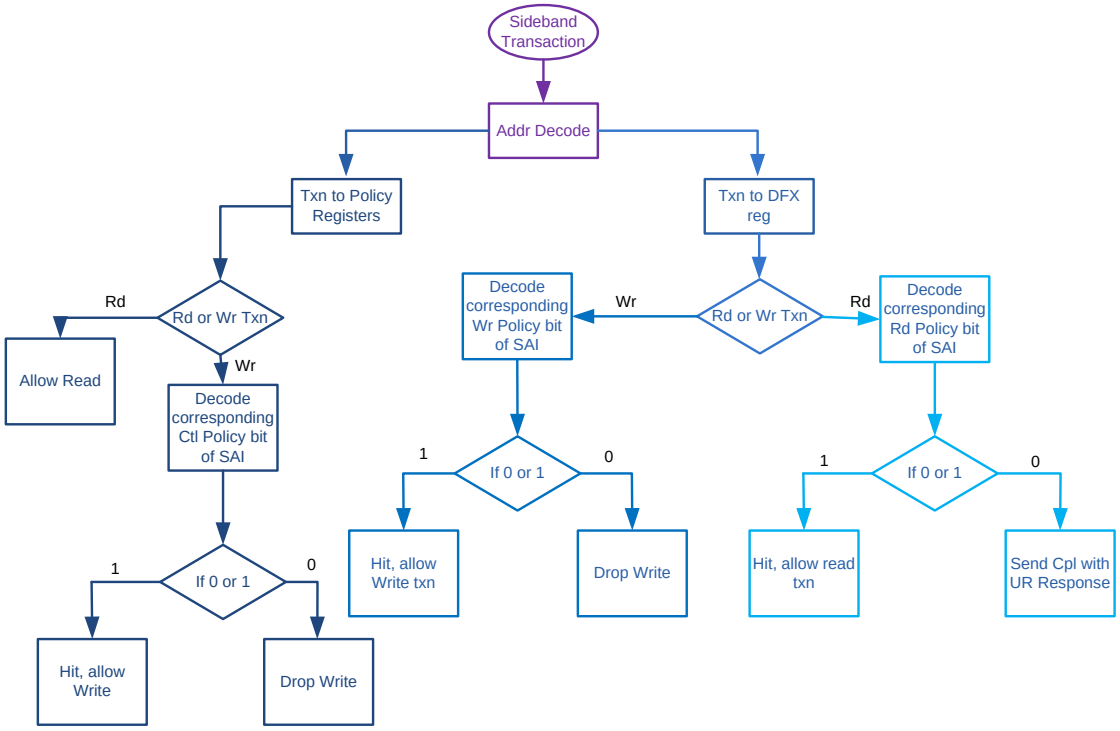


Table 15. TAP2SB SAI Policy Register Reset and Address Assignments

Policy Register Name	Address	Reset Value	Register Type
Control Policy Register	0x0	strap_tpsb_ctl_ply_reg_init_val	Read-Write
Read Policy Register	0x4	strap_tpsb_rd_ply_reg_init_val	Read-Write
Write Policy Register	0x8	strap_tpsb_wr_ply_reg_init_val	Read-Write
User defined register address space	0x0010 – 0xFFFF	NA	Any type

3.14.3.3 Security access with TAP

The TAP2SB block uses the DFx security plug-in that is embedded in the TAP. A parameter provides the level of access to the opcodes used in standard sideband transactions. The TDR to enable the parallel message bus for HVM testing has a red level access.

3.14.4 Parameter and Strap Values

Refer to Table 16 for the parameter and strap value definitions.

Table 16. Parameter Values for TAP2SB

Parameter Name	Range	Default	Typical SoC Settings	Description (Including Interdependencies)
TPSB_SAI_AWARE	0 to 1	1	1	Specifies whether or not it is SAI aware when IOSF Spec 0.90 is supported.
TPSB_PMSG_SUPPORT	0 to 1	1	1	Specifies whether or not PMSG is supported.
TPSB_DFX_REG_SUPPORT	0 to 1	1	1	Specifies whether External Register Unit support is enabled.
TPSB_SAI_START_BIT	0 to 19	0	0	SAI is 5 consecutive bits of Expanded Header. This parameter specifies the start of SAI. Example: if start bit = 0 then sai[4:0] if start bit = 1 then sai[5:1] if start bit = 2 then sai[6:2] etc
TPSB_SAI_CHECK_ENABLED	0 to 1	0	1	Specifies whether or not SAI Enforced Check is enabled: A value of 1 means SAI Enforced Check is enabled A value of 0 means there will be no SAI enforced check for register access.
TPSB_BASE_ADDR_PLY_REG	0 to 0xFFF8	0x0	0x0	The starting address for the policy registers. This parameter should have been removed and the starting address fix but it was missed so the design team should set this value to 0x0 to prevent address collisions.
TPSB_PMSG_EXTN_ADDR_SUPPORT	0 to 1	1	0	Support of 48-bit addressing from Parallel Message Bus. 0: There is only 16-bit addressing supported 1: Both 16-bit and 48-bit addressing are supported
TPSB_BROADCAST_EN	0 TO 1	0	0	Specifies whether the TAP2SB supports the broadcasten bit. If TPSB_BROADCAST_EN = 0 then the broadcasten bit is masked and has no effect on the transaction. Internally, the bit is logic 0 regardless of what the user writes. If TPSB_BROADCAST_EN = 1 then the broadcasten bit is active and the user can set or clear the bit as needed.

Parameter Name	Range	Default	Typical SoC Settings	Description (Including Interdependencies)
TPSB_SBMSG_0_GREEN_1_ ORANGE_2_RED	0 to 2	1	1	Security level for JTAG Opcodes 0x30, 0x31, 0x32, 0x33 and 0x35 0: Security level is GREEN 1: Security level is ORANGE 2: Security level is RED (RED2 and RED4 are the same for the opcodes)
DESTINATIONID_MATRIX		[7:0] [8:0] = 0	implementation dependent	An array of 8 destination IDs with an enable bit in the MSB position. DESTINATIONID_MATRIX[7:0][8:0] = {destEn7, destId7[7:0], destEn6, destId6[7:0], destEn5, destId5[7:0], destEn4, destId4[7:0], destEn3, destId3[7:0], destEn2, destId2[7:0], destEn1, destId1[7:0], destEn0, destId0[7:0] }

Table 17. TAP2SB Strap Inputs

Strap Name	Signal Name	Description
SAI_dfx_green	strap_tpsb_sai_green_id_i [15:0]	When the DFx secure policy is 0x0 the DFx secure plug-in informs the TAP2SB logic to send this SAI value for Sideband transactions.
SAI_dfx_orange	strap_tpsb_sai__orange_id_i [15:0]	When the DFx secure policy is 0x5 and 0x7 – 0xE, the DFx secure plug-in informs the TAP2SB logic to send this SAI value for Sideband transactions.
SAI_dfx_red2	strap_tpsb_sai__red2_id_i [15:0]	When the DFx secure policy is 0x2, the DFx secure plug-in informs the TAP2SB logic to send this SAI value for Sideband transactions.
SAI_dfx_red4	strap_tpsb_sai__red4_id_i [15:0]	When the DFx secure policy is 0x4, the DFx secure plug-in informs the TAP2SB logic to send this SAI value for Sideband transactions.
IOSF SB source port ID	strap_tpsb_port_id_i[7:0]	For Sideband Port ID for TAP2IOSFSB
Read policy initial register value	strap_tpsb_ctl_ply_reg _val_i[31:0]	Initial value of Control Policy Register gets programmed immediately after Reset with this parameter value. Value should NOT be 0xFFFF_FFFF but rather a specific set of valid authenticate agents.
Write policy initial register value	strap_tpsb_rd_ply_reg _val_i[31:0]	Initial value of Read Policy Register gets programmed immediately after Reset with this parameter value. Value should NOT be 0xFFFF_FFFF but rather a specific set of valid authenticate agents.
Control policy initial register value	strap_tpsb_wr_ply_reg _val_i[31:0]	Initial value of Write Policy Register gets programmed immediately after Reset with this parameter value. Value should NOT be 0xFFFF_FFFF but rather a specific set of valid authenticate agents.

3.14.4.1 DFx Secure Plugin Parameter Values

Table 18 lists all of the policies and the associated settings for the DFx feature enables for the TAP2SB. DFx feature enables are used to determine which SAI value is sent as part of the transaction. These feature enables are fixed based on the broadest use models for green (Security Locked), orange (OEM unlocked), and two versions of red. It is referred to as red2 and red4 which is associated with policy value, red2 is Security Unlocked and red4 is Intel Unlocked. Based on this table the policy matrix is hardcoded with the following. The reader should note that TAP2SB does not have a VISA ULM and therefore the DFx secure plug-in encoding is for completeness only.

- STAP_DFX_SECURE_ POLICY_MATRIX[0] = {4'b0001, DFX_VISA_GREEN}
- STAP_DFX_SECURE_ POLICY_MATRIX[1] = {4'b0000, DFX_VISA_BLACK}
- STAP_DFX_SECURE_ POLICY_MATRIX[2] = {4'b0100, DFX_VISA_RED}
- STAP_DFX_SECURE_ POLICY_MATRIX[3] = {4'b0000, DFX_VISA_BLACK}
- STAP_DFX_SECURE_ POLICY_MATRIX[4] = {4'b1000, DFX_VISA_RED}
- STAP_DFX_SECURE_ POLICY_MATRIX[5] = {4'b0010, DFX_VISA_ORANGE}
- STAP_DFX_SECURE_ POLICY_MATRIX[6] = {4'b0000, DFX_VISA_BLACK}
- STAP_DFX_SECURE_ POLICY_MATRIX[14:7] = {4'b0010, DFX_VISA_ORANGE}
- STAP_DFX_SECURE_ POLICY_MATRIX[15] = {4'b0000, DFX_VISA_BLACK}

Table 18. Policy Matrix Table for TAP2SB

Policy Name	DFx Secure Policy Bus Encode	DFx Feature en[3] red4 Access	DFx Feature en[2] red2 Access	DFx Feature en[1] Orange Access	DFx Feature en[0] Green Access	VISA not used	Description with TAP Examples
Security Locked	0000	0	0	0	1	11	Allow TAP to Sideband transactions that enabled based on the destination ID matrix Green destination IDs.
Functionality Locked	0001	0	0	0	0	11	Functionality Locked is not permitted for any level of access
Security Unlocked	0010	0	1	0	0	11	Allow TAP to Sideband with red2 SAI value.
Reserved	0011	0	0	0	0	11	
Intel Unlocked	0100	1	0	0	0	11	Allow TAP to Sideband access with red4 SAI value.
OEM Unlocked	0101	0	0	1	0	11	Allow TAP to Sideband access with OEM orange SAI value.
Revoked (Reserved)	0110	0	0	0	0	11	Reserved for revoking of privileges
"User 1" Unlocked	0111	0	0	1	0	11	Mapped to normal OEM orange access.
"User 2" Unlocked	1000	0	0	1	0	11	Mapped to normal OEM orange access.

Policy Name	DFx Secure Policy Bus Encode	DFx Feature en[3] red4 Access	DFx Feature en[2] red2 Access	DFx Feature en[1] Orange Access	DFx Feature en[0] Green Access	VISA not used	Description with TAP Examples
"User 3" Unlocked	1001	0	0	1	0	11	Mapped to normal OEM orange access.
"User 4" Unlocked	1010	0	0	1	0	11	Mapped to normal OEM orange access.
"User 5" Unlocked	1011	0	0	1	0	11	Mapped to normal OEM orange access.
"User 6" Unlocked	1100	0	0	1	0	11	Mapped to normal OEM orange access.
"User 7" Unlocked	1101	0	0	1	0	11	Mapped to normal OEM orange access.
"User 8" Unlocked	1110	0	0	1	0	11	Mapped to normal OEM orange access.
Part Disabled	1111	0	0	0	0	11	Disable access

Table 19. DFx Secure Plugin IP Parameters

Parameter name	Parameter value(s)	Comments
Local TAP parameters		
STP_DFX_SECURE_WIDTH	4	This parameter is fixed at 4 for this revision of the spec. (Defined for this SoC HAS revision, the IOSF DFx rev1.2 and in the Chassis DFx Gen2 HAS)
STP_DFX_NUM_OF_FEATURES _TO_SECURE	4	Fixed for TAP2SB.
STAP_DFX_SECURE_ POLICY_MATRIX	See Table 18	This parameter determines the lookup table necessary to assign the policy with the DFx feature(s) including VISA access. This parameter is fixed for TAP2SB refer to Table 33.
STAP_DFX_EARLYBOOT_FEATURE_ENABLE	[STAP_DFX_NUM_OF_FEATURES _TO _SECURE+1:0]	This parameter sets the hard coded value for the early debug window for this agent/IP-block. DFX_EARLYBOOT_FEATURE_ENABLE[1:0] = {0001, VISA_BLACK}
USE_SB_OVR	0	The TAP does not use the Sideband override feature.

3.14.5 Support for Root Space

The root space field was added to the IOSF rev1.1 spec. TAP2SB will support this field however, it will be fixed in its width (RS_WIDTH=3). The Sideband rules for masters are listed here for convience but the reader should refer to IOSF rev1.1 for more complete information. The root space field is located in the register named, SBROOTSPC at opcode 0x35 (refer to section 3.14.10.6).

Root Space Rules for Sideband Masters:

- Rule 1. A Root Space aware IOSF agent is allowed to put any legal Root Space Value in the rs field of the SAIRS expanded header. [RA385]
- Rule 2. If the agent's RS_WIDTH strap is less than 3, then rs[3:RS_WIDTH+1] must be treated as reserved and driven to 0. [RA386]
- Rule 3. A Non-Root Space aware IOSF agent must set the rs field in an SAIRS expanded header to 0.[RA387]

3.14.6 Support for side_pok Signal

A parameter option allows the user to permanently set the side_pok signal to logic 1 or manually write a TAP register bit to assert or clear the signal assertion.

The user must assert sbPokEn bit in the SBROOTSPC register before sending a Sideband transaction. The user must deassert when no more transactions are being requested through the IP. This implies only manual control over the signal and the TAP2SB IP-block has no override control of the signal. Manual control reduces complexity of the implementation based for this release of the IP-block

3.14.7 TAP-to-IOSF Sideband Implementation Details

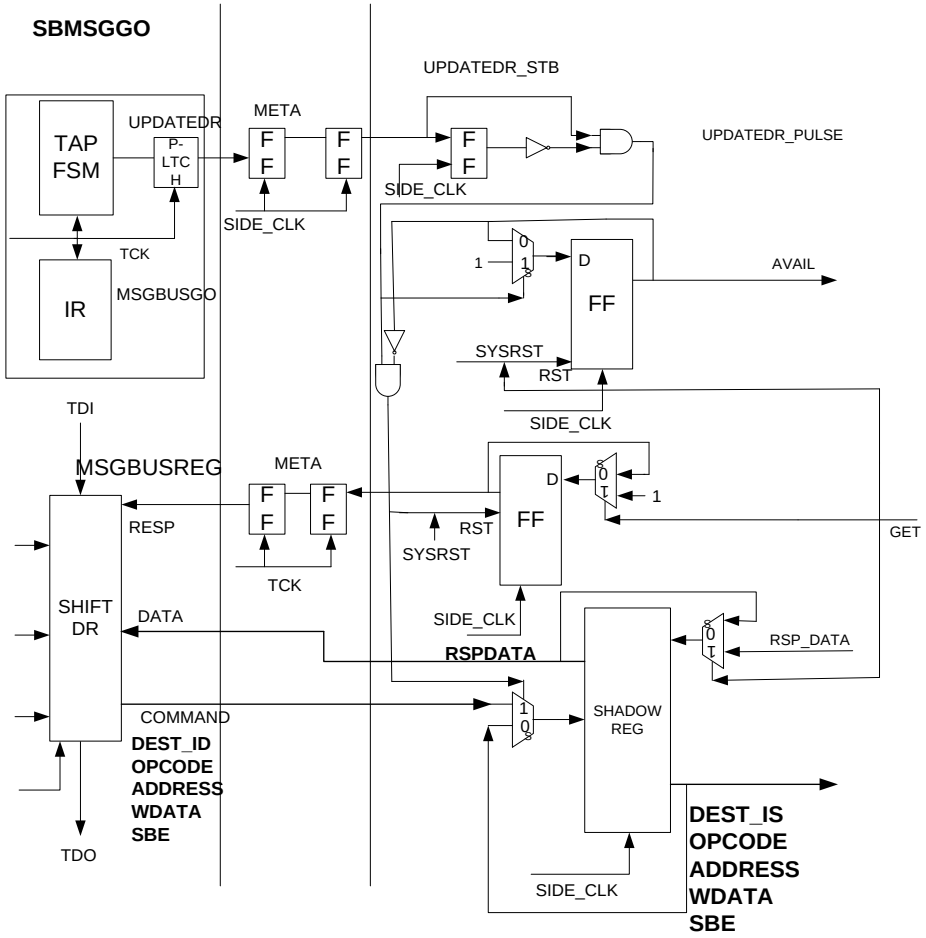
The purpose of this section is to provide the reader with more detailed information about the TAP2IOSF control logic. It is only intended to be informative; the reader is encouraged to read the TAP2IOSFSB MAS documentation as the definitive source of accurate information. This section will only show the essential logic diagrams. Figure 39 shows a detailed block diagram describes how the logic is requesting access on the endpoint to send and receive transactions.

For a standard (non-triggered) posted message (configuration write transaction), the user writes the instruction registers with IR=0x30. If the Sideband Endpoint (SBEP) is currently clock gated then a signal to wake the endpoint will be asserted when the instruction is decoded in the Update-IR state (tap_rtdr_irdec_update_ir[N]. In the Scan-DR phase, the data shifted into the register is command and address of where the write will be sent. When the Update-DR is asserted the signal re-sync'd through a pair of meta-stable flops then edge detected and latched in the sideband's clock domain. The "AVAIL" signal shown in the diagram is the command execution that informs the sideband to begin requesting access to the network. The "GET" signal indicates that the posted message was sent. The endpoint is ready for another transaction. The TAP2SB can send posted or non-posted write messages.

If the request was a non-posted then the sequence of events listed in the previous paragraph are the same except that when a response with data is delivered by the endpoint the "GET" signal is asserted and data appears on the "RSP_DATA" bus. The user will write the instruction IR=0x31 (TAPC_SBMSGRSP) to shift out the data.

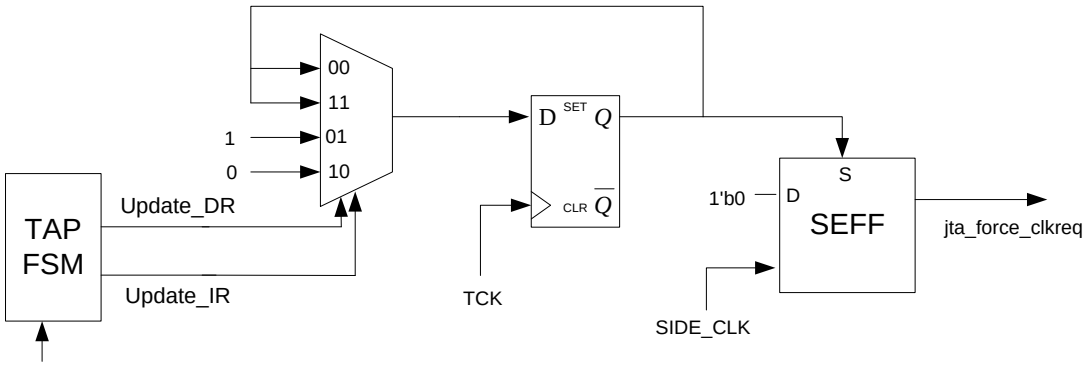
Another use model is to shift data into the register with IR=0x32 (TAPC_SBMSGNOGO) and use a hardware trigger to launch the message. This can be useful for debug to align a message with an internal event to root cause an issue.

Figure 39. TAP2SB Detailed Block Diagram



The IOSF sideband endpoint will be clock gated when there is no activity. To wake the endpoint a jta_force signal is asserted as shown in Figure 40. This signal is generated by the detection of an Update_IR for 0x30, 0x31 or 0x32. It will remain asserted until the Update_DR is asserted. Once a request to the sideband endpoint is made then the endpoint point will keep the logic in an active state until the transaction is complete.

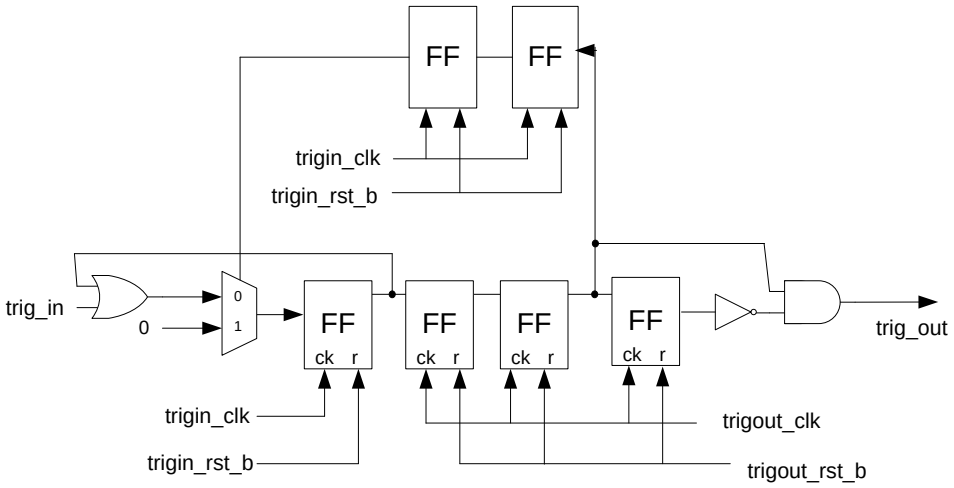
Figure 40. TAP2SB Wake Event to Endpoint Logic



The TAP2SB provides a mechanism to launch a message previous loaded in the TAPC_SBMSG register. An SoC trigger source from either a micro breakpoint controller or the Lakemore SoC Trigger Processor can be used as the event to launch a message. This allows hardware control over the initiation of the message instead of a direct software write. The synchronization logic is shown in Figure 41. The trigin_clk is from the trigger clock domain and the trigout_clk is the IOSF sideband endpoint clock frequency. The frequency of the trigger may be faster or slower than the frequency of the endpoint. Since we are only interested in detecting the edge this logic is sufficient to handle either fast-to-slow or slow-to-fast crossings.

Use of the resets is optional but encouraged. It is assumed these are connected to the warm reset of the domain on where it is used.

Figure 41. Trigger Synchronizer Logic for SoC Trigger Launch of Message



3.14.8 Parallel Message Bus

A higher bandwidth configuration access port is required in HVM testing. This allows full chip functional tests to consume a smaller portion of the overall test time when compared to using the TAP interface. To accomplish this, the SoC uses the Narrow Test Interface (NTI) which is usually derived from DDR port pins. The parallel port width still requires multiple transfers because the internal staging flop is equivalent to the width of the TAP shift register output. A simple protocol to collect packet transaction will upload the staging flops before requesting access to the Sideband network. The parallel message bus is muxed into the same path that already exists between the TAP and the SB management logic. The logic is optimized for writes which are a majority of the transactions for configuration.

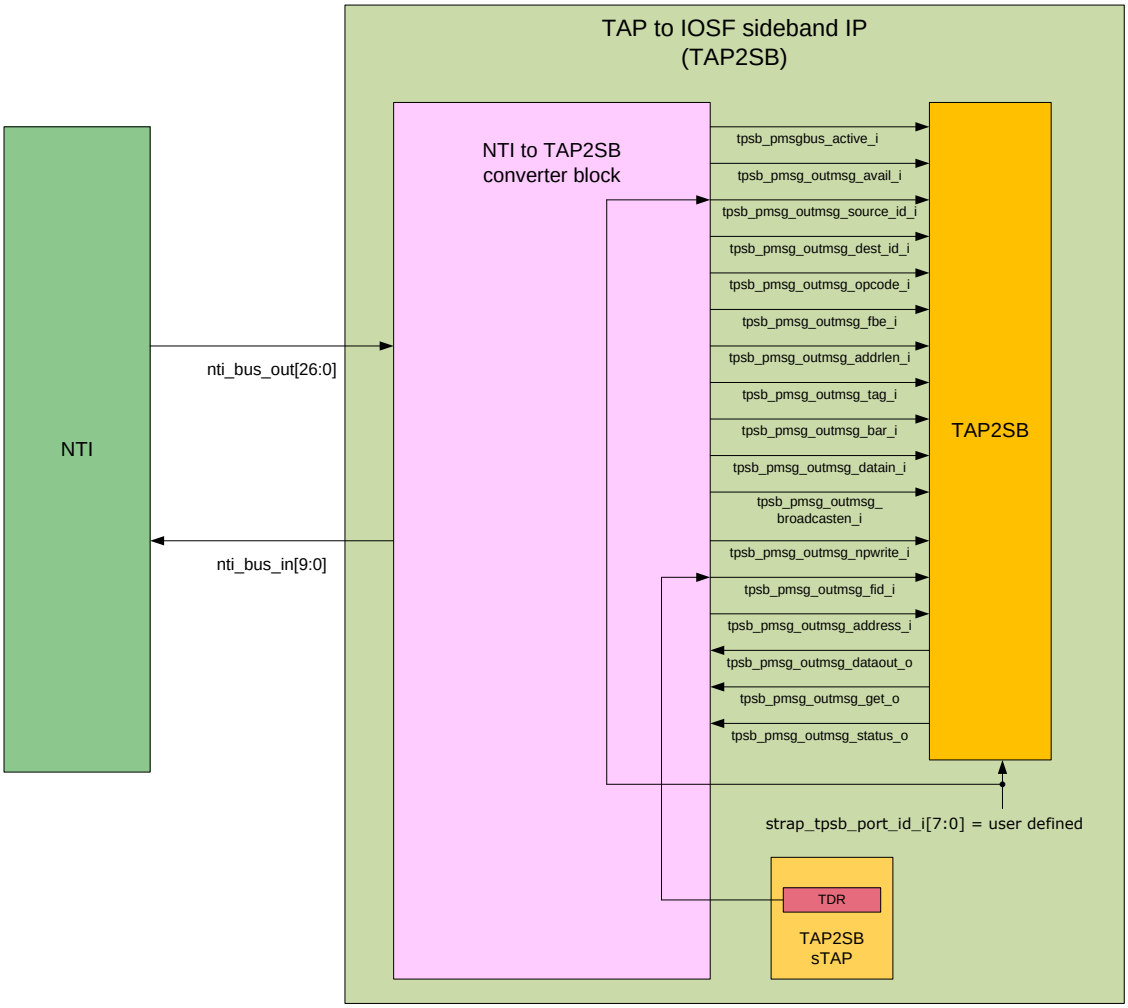
There is not an enable bit for this feature. The logic simply accepts a transaction from one source and waits until completion. It is generally accepted that this feature is only for HVM use models and the user will know to send unique transactions on either the TAP or the parallel port but not both simultaneously.

It is SoC dependent to provide the connectivity from the DDR IO DFx port and associated enabling logic to the TAP2SB logic block.

A parallel message bus bypasses the TAP-to-IOSF Sideband block to directly access the endpoint. This provides a higher bandwidth to config the SoC for HVM testing and debug. A high level block diagram is shown in Figure 42. The external module is SoC dependent and expected to be in the top level SoC DFx unit. It will collect the parallel signals, most likely from the DDR interface in a test bypass mode and present them as an input to the PMSGBUS block. Timing and the appropriate staging flops are part of the external module and the responsibility

of the SoC. When tsb_pmsgbus_active_i is asserted, internally the PMSGBUS block enables the parallel path to the endpoint for HVM test config accesses. The TAP serial register access port to the endpoint is disabled. The pmsgbus_active signal is controlled by a TAP bit and it will usually be the SoC DFx unit's sTAP. It is SoC dependent on where this module and the TAP override is placed.

Figure 42. Parallel Message Bus Block Diagram



The top level signal description for the parallel message to sideband converter block is listed in Table 20.

Table 20. Parallel Message Bus Signal List

Signal Name	Input/Output	Width	Description
tpsb_pmsg_clk_i	Input	1	Parallel Message Bus clock; It may come from NTI interface or directly from Tester
tpsb_pmsg_fdfx_rst_b_i	Input	1	Parallel Message Bus Reset. It can be Powergood reset or Soft-reset; It should be better to be soft-reset.
tpsb_pmsg_outmsg_datain_i	Input	[25:0]	Data for Sideband transaction from the Parallel Message Bus

Signal Name	Input/Output	Width	Description
tpsb_pmsg_outmsg_datain_valid_i	Input	1	Input data in valid
tpsb_pmsg_outmsg_dataout_o	Output	[7:0]	Completion data for Sideband Transaction
tpsb_pmsg_outmsg_dataout_valid_o	Output	1	Output data is valid
tpsb_pmsg_outmsg_get_o	Output	1	Conveys the End of PBUS-Sideband transaction.

3.14.9 Transaction Cycle/Data Flows

Data from the Narrow Test Interface (NTI) has a specific protocol for the data flow from the NTI to the TAP2SB. The data is packetized over NTI and decompressed in the TAP2SB IP. Table 21 is the format to communicate to Sideband endpoint with a standard 16-bit address format. Table 22 is the 48-bit address format, which requires two more cycles to deliver the information to the IP.

Table 21. NTI to Parallel Message Format for 16-bit Standard Address Model

Parallel Message Bus Pin (pm = tpsb_pmsg _outmsg_*)	I/O	PmsgBus input cycle0	PmsgBus Input cycle1	PmsgBus Input cycle2	PmsgBus Input cycle3	PmsgBus Input cycle4	PmsgBus Output cycle0	PmsgBus Output cycle1	PmsgBus Output cycle2	PmsgBus Output cycle3
pm_datain_i[0]	I	dest_id_0	data_0	data_26						
pm_datain_i[1]	I	dest_id_1	data_1	data_27						
pm_datain_i[2]	I	dest_id_2	data_2	data_28						
pm_datain_i[3]	I	dest_id_3	data_3	data_29						
pm_datain_i[4]	I	dest_id_4	data_4	data_30						
pm_datain_i[5]	I	dest_id_5	data_5	data_31						
pm_datain_i[6]	I	dest_id_6	data_6	addr_0						
pm_datain_i[7]	I	dest_id_7	data_7	addr_1						
pm_datain_i[8]	I	opcode_0	data_8	addr_2						
pm_datain_i[9]	I	opcode_1	data_9	addr_3						
pm_datain_i[10]	I	opcode_2	data_10	addr_4						
pm_datain_i[11]	I	opcode_3	data_11	addr_5						
pm_datain_i[12]	I	opcode_4	data_12	addr_6						
pm_datain_i[13]	I	opcode_5	data_13	addr_7						
pm_datain_i[14]	I	opcode_6	data_14	addr_8						
pm_datain_i[15]	I	opcode_7	data_15	addr_9						
pm_datain_i[16]	I	be_0	data_16	addr_10						
pm_datain_i[17]	I	be_1	data_17	addr_11						
pm_datain_i[18]	I	be_2	data_18	addr_12						
pm_datain_i[19]	I	be_3	data_19	addr_13						
pm_datain_i[20]	I	tag_0	data_20	addr_14						
pm_datain_i[21]	I	tag_1	data_21	addr_15						
pm_datain_i[22]	I	tag_2	data_22	np_write						
pm_datain_i[23]	I	bar_0	data_23	ext_address						
pm_datain_i[24]	I	bar_1	data_24	outmsg_avail						
pm_datain_i[25]	I	bar_2	data_25	broadcat_en						
pm_datain_valid_i	I	cycle_ valid	cycle_ valid	cycle_valid						
pm_dataout_valid_o	O						valid_out	valid_out	valid_out	valid_out

Parallel Message Bus Pin (pm = tpsb_pmsg _outmsg_*)	I/O	PmsgBus input cycle0	PmsgBus Input cycle1	PmsgBus Input cycle2	PmsgBus Input cycle3	PmsgBus Input cycle4	PmsgBus Output cycle0	PmsgBus Output cycle1	PmsgBus Output cycle2	PmsgBus Output cycle3
pm_dataout_o[0]	O						data_0	data_8	data_16	data_24
pm_dataout_o[1]	O						data_1	data_9	data_17	data_25
pm_dataout_o[2]	O						data_2	data_10	data_18	data_26
pm_dataout_o[3]	O						data_3	data_11	data_19	data_27
pm_dataout_o[4]	O						data_4	data_12	data_20	data_28
pm_dataout_o[5]	O						data_5	data_13	data_21	data_29
pm_dataout_o[6]	O						data_6	data_14	data_22	data_30
pm_dataout_o[7]	O						data_7	data_15	data_23	data_31
pm_outmsg_get_o	O						0	0	0	1

Table 22. NTI to Parallel Message Format for 48-bit Extended Address Model

Parallel Message Bus Pin (pm = tpsb_pmsg _outmsg_*)	I/O	PmsgBus Input cycle0	PmsgBus Input cycle1	PmsgBus Input cycle2	PmsgBus Input cycle3	PmsgBus Input cycle4	PmsgBus Output cycle0	PmsgBus Output cycle1	PmsgBus Output cycle2	PmsgBus Output cycle3
pm_datain_i[0]	I	dest_id_0	data_0	data_26	addr_16	addr_42				
pm_datain_i[1]	I	dest_id_1	data_1	data_27	addr_17	addr_43				
pm_datain_i[2]	I	dest_id_2	data_2	data_28	addr_18	addr_44				
pm_datain_i[3]	I	dest_id_3	data_3	data_29	addr_19	addr_45				
pm_datain_i[4]	I	dest_id_4	data_4	data_30	addr_20	addr_46				
pm_datain_i[5]	I	dest_id_5	data_5	data_31	addr_21	addr_47				
pm_datain_i[6]	I	dest_id_6	data_6	addr_0	addr_22	0				
pm_datain_i[7]	I	dest_id_7	data_7	addr_1	addr_23	0				
pm_datain_i[8]	I	opcode_0	data_8	addr_2	addr_24	0				
pm_datain_i[9]	I	opcode_1	data_9	addr_3	addr_25	0				
pm_datain_i[10]	I	opcode_2	data_10	addr_4	addr_26	0				
pm_datain_i[11]	I	opcode_3	data_11	addr_5	addr_27	0				
pm_datain_i[12]	I	opcode_4	data_12	addr_6	addr_28	0				
pm_datain_i[13]	I	opcode_5	data_13	addr_7	addr_29	0				
pm_datain_i[14]	I	opcode_6	data_14	addr_8	addr_30	0				
pm_datain_i[15]	I	opcode_7	data_15	addr_9	addr_31	0				
pm_datain_i[16]	I	be_0	data_16	addr_10	addr_32	0				
pm_datain_i[17]	I	be_1	data_17	addr_11	addr_33	0				
pm_datain_i[18]	I	be_2	data_18	addr_12	addr_34	0				
pm_datain_i[19]	I	be_3	data_19	addr_13	addr_35	0				
pm_datain_i[20]	I	tag_0	data_20	addr_14	addr_36	0				
pm_datain_i[21]	I	tag_1	data_21	addr_15	addr_37	0				
pm_datain_i[22]	I	tag_2	data_22	np_write	addr_38	0				
pm_datain_i[23]	I	bar_0	data_23	ext_address	addr_39	0				
pm_datain_i[24]	I	bar_1	data_24	outmsg_avail	addr_40	0				
pm_datain_i[25]	I	bar_2	data_25	broadcat_en	addr_41	0				
pm_datain_valid_i	I	cycle_ valid	cycle_ valid	cycle_valid	cycle_valid	cycle_valid				
pm_dataout_valid_o	O						valid_out	valid_out	valid_out	valid_out
pm_dataout_o[0]	O						data_0	data_8	data_16	data_24
pm_dataout_o[1]	O						data_1	data_9	data_17	data_25

Parallel Message Bus Pin (pm = tpsb_pmsg _outmsg_*)	I/O	PmsgBus Input cycle0	PmsgBus Input cycle1	PmsgBus Input cycle2	PmsgBus Input cycle3	PmsgBus Input cycle4	PmsgBus Output cycle0	PmsgBus Output cycle1	PmsgBus Output cycle2	PmsgBus Output cycle3
pm_dataout_o[2]	O						data_2	data_10	data_18	data_26
pm_dataout_o[3]	O						data_3	data_11	data_19	data_27
pm_dataout_o[4]	O						data_4	data_12	data_20	data_28
pm_dataout_o[5]	O						data_5	data_13	data_21	data_29
pm_dataout_o[6]	O						data_6	data_14	data_22	data_30
pm_dataout_o[7]	O						data_7	data_15	data_23	data_31
pm_outmsg_get_o	O						0	0	0	1

3.14.10 TAP2SB Transaction Generator Registers

Table 23 lists the test/debug registers and the expected behavior in reset and each primary action state of the TAP's FSM.

Table 23. Slave TAP DR State Behavior

TAP Instruction	Data Register Name (type)	Reset Behavior		Action During			
		Powergood _rst_b	TRST_b	RT/Idle	Capture-DR	Shift-DR	Update-DR
SBMSGGO (0x30)	SBMSGGO (remote)	SBMSGGO = 0	Previous state	-	Capture shadow register	Shift	Update shadow register and execute
SBMSGRSP (0x31)	SBMSGRSP (remote)	SBMSGRSP = 0	Previous state	-	Capture data response	Shift	NA
SBMSGNOGO (0x32)	SBMSGNOGO (remote)	SBMSGNOGO = 0	Previous state	-	Capture shadow register	Shift	Update shadow register ONLY Do not execute
SBEXTADDR (0x33)	SBEXTADDR (internal)	SBEXTADDR=0	Previous state	-	Capture shadow register	Shift	Update shadow register
POVRSBFID (0x34)	POVRSBFID (internal)	POVRSBFID=0	Previous state	-	Capture shadow register	Shift	Update shadow register
SBROOTSPC (0x35)	SBROOTSPC (internal)	SBROOTSPC = 0	Previous state	-	Capture output of update register	Shift	Update shadow register

3.14.10.1 SBMSGGO: Slave TAP to IOSF SB transaction execution

This TDR remote register is implemented as a single TDR with glue logic to manage three opcodes. This opcode and TDR name description association is for the benefit of TAP-based SystemRDL and post-silicon DFx users.

The user must use this opcode/TDR for executing a read or write to any Sideband endpoint. If it is a write then only this opcode is necessary. If the transaction is a read then the user must write the appropriate message attributes and poll the SBMSGRSP TDR for the data completion response. The SBMSG* contains the same protocol fields as a message transaction would appear on the interface. This allows different messages to be sent from the TAP to an agent.

This feature supports simple messages, messages with data, register access reads and write and completions with and without data

JTAG Register	
Register name	SBMSGGO
JTAG IR	0x30
Default	0
JTAG DR length	96
DR shift size	96
Power well	User defined

Register Name: SBMSGGO				
Bits	Access	Default	Label	Bit Description
95:88	WO	0	sbmsgbyte10	Sideband message byte 10: SimpMsg: NA MsgD: data[63:56] RegAccRd: NA RegAccWr: data[31:24]
87:80	WO	0	sbmsgbyte9	Sideband message byte 9: SimpMsg: NA MsgD: data[55:48] RegAccRd: NA RegAccWr: data[23:16]
79:72	WO	0	sbmsgbyte8	Sideband message byte 8: SimpMsg: NA MsgD: data[47:40] RegAccRd: NA RegAccWr: data[15:8]
71:64	WO	0	sbmsgbyte7	Sideband message byte 7: SimpMsg: NA MsgD: data[39:32] RegAccRd: NA RegAccWr: data[7:0]
63:56	RW_V	0	sbmsgbyte6	Sideband message byte 6: SimpMsg: NA MsgD: data[31:24] RegAccRd: address[15:8] RegAccWr: address[15:8]
55:48	RW_V	0	sbmsgbyte5	Sideband message byte 5: SimpMsg: NA MsgD: data[23:16] RegAccRd: address[7:0] RegAccWr: address[7:0]

Register Name: SBMSGGO				
47:40	RW_V	0	sbmsgbyte4	Sideband message byte 4: SimpMsg: NA MsgD: data[15:8] RegAccRd: fid RegAccWr: fid
39:32	RW_V	0	sbmsgbyte3	Sideband message byte 3: SimpMsg: NA MsgD: data[7:0] RegAccRd: {0000, fbe[3:0]} RegAccWr: {0000, fbe[3:0]}
31:24	RW_V	0	sbmsgbyte2	Sideband message byte 2: SimpMsg: {EH, rsvd, tag} MsgD: {EH, rsvd, tag} RegAccRd: {EH, addrlen, bar, tag} RegAccWr: {EH, addrlen, bar, tag} Note1: EH bit is automatically set by the TAP2SB logic block if the parameter SAI_AWARE = 1.
23:16	WO	0	sbmsgbyte1	Sideband message byte 1: All message types are supported for the Opcode field except the completions with and without data (0x21, 0x20). TAP2SB will not initiate a transaction with these values written in this field.
15:8	WO	0	sbmsgbyte0	Sideband message byte 0: All message types: Dest (destination)
7:6	WO	0	msgtrigsel	Sideband message trigger select 00: Trigger disable 01: Select Trigger source 0 10: Select Trigger source 1 11: Reserved
5	WO	0	unused	Unused bit reserved for future use.
4	WO	0	data64en	Data with 64-bit enable. This mode is for debugging Vendor Defined Messages primarily to/from the Power Management Unit (PMU). 0: Use 32-bit data 1: Use 64-bit data
3	WO	0	npwrite	Indicates if the write message is non-posted (npwrite=1) or posted (npwrite=0). This may apply to either register access write messages, simple messages or messages with data. Read requests are always sent as a non-posted message for any message type (register access or message with data). 0: Write messages with be posted 1: Write messages with be non-posted.
2	WO	0	broadcasten	Enable a broadcast/multi-cast transaction. 0: Use endpoint defined source ID code. This the strap value applied to this endpoint. 1: Use 0xFE in the source ID code field.

Register Name: SBMSGGO				
1:0	RO_V	0	status	This bit field indicates the current status of pending transactions with the sideband endpoint. 00: No transaction launched. TAP/sideband logic block has not received update_dr for MSGGO(0x30), or TAP/sideband logic has received update_dr for MSGNOGO (0x32) but external trigger has not come, or TAP/sideband logic has not received update_dr for MSGGO(0x18), or 01:Transaction has started. TAP/sideband logic has started launching a transaction at sideband upstream interface but launch as not completed yet (busy) A request was accepted by the IOSF sideband logic and the transaction is in progress; EOM has not come yet, or Trigger condition has been met for launching sideband transaction. 10: A non-posted is pending Sideband upstream non-posted transaction has launched to Sideband fabric; waiting for completion. IOSF Sideband logic is waiting for Sideband Downstream Completion from Sideband Fabric for Non-posted transaction Note: For an sideband upstream posted transaction, this status value will be not be set 11: Transaction completed Completion was received from sideband fabric for non-posted transaction Sideband upstream posted transaction was launched and completed successfully.

3.14.10.2 SBMSGRSP: Slave TAP to IOSF SB Response

This TDR remote register is implemented as a single TDR with glue logic to manage three opcodes. This opcode and TDR name description association is for the benefit of TAP-based SystemRDL and post-silicon DFx users.

The user must use this opcode/TDR for extracting the data completion response from a previously executed read transaction. The SBMSG* contains the same protocol fields as a message transaction would appear on the interface. This allows different messages to be sent from the TAP to an agent. This feature supports simple messages, messages with data, register access reads and write and completions with and without data.

JTAG Register	
Register name	SBMSGRSP
JTAG IR	0x31
Default	0
JTAG DR length	96
DR shift size	96
Power well	User defined

Register Name: SBMSG				
Bits	Access	Default	Label	Bit Description
95:88	RO_V	0	sbmsgbyte10	Sideband message byte 10: Not used by this opcode.
87:80	RO_V	0	sbmsgbyte9	Sideband message byte 9: Not used by this opcode.
79:72	RO_V	0	sbmsgbyte8	Sideband message byte 8: Not used by this opcode.
71:64	RO_V	0	sbmsgbyte7	Sideband message byte 7: Not used by this opcode.
63:56	RO_V	0	sbmsgbyte6	Sideband message byte 6: CompD: data[31:24] CompNoD: NA, there is no data to read
55:48	RO_V	0	sbmsgbyte5	Sideband message byte 5: CompD: data[23:16] CompNoD: NA, there is no data to read
47:40	RO_V	0	sbmsgbyte4	Sideband message byte 4: CompD: data[15:8] CompNoD: NA, there is no data to read
39:32	RO_V	0	sbmsgbyte3	Sideband message byte 3: CompD: data[7:0] CompNoD: NA, there is no data to read
31:24	RO_V	0	sbmsgbyte2	Sideband message byte 2: CompD: {EH, rsvd, rsp, tag} CompNoD: {EH, rsvd, rsp, tag} Note1: EH bit is automatically set by the TAP2SB logic block if the parameter SAI_AWARE = 1.
23:16	RO_V	0	sbmsgbyte1	Sideband message byte 1: Not used by this opcode.
15:8	RO_V	0	sbmsgbyte0	Sideband message byte 0: Not used by this opcode.
7:6	RO_V	0	msgtrigsel	Not used by this opcode.
5	RO_V	0	unused	Not used by this opcode.
4	RO_V	0	data64en	Not used by this opcode.
3	RO_V	0	npwrite	Not used by this opcode.
2	RO_V	0	broadcasten	Not used by this opcode.

Register Name: SBMSG				
1:0	RO_V	0	status	This bit field indicates the current status of pending transactions with the sideband endpoint. 00: No transaction launched. TAP/sideband logic block has not received update_dr for MSGGO(0x30), or TAP/sideband logic has received update_dr for MSGNOGO (0x32) but external trigger has not come, or TAP/sideband logic has not received update_dr for MSGGO(0x18), or 01:Transaction has started. TAP/sideband logic has started launching a transaction at sideband upstream interface but launch as not completed yet (busy) A request was accepted by the IOSF sideband logic and the transaction is in progress; EOM has not come yet, or Trigger condition has been met for launching sideband transaction. 10: A non-posted is pending Sideband upstream non-posted transaction has launched to Sideband fabric; waiting for completion. IOSF Sideband logic is waiting for Sideband Downstream Completion from Sideband Fabric for Non-posted transaction Note: For an sideband upstream posted transaction, this status value will be not be set 11: Transaction completed Completion was received from sideband fabric for non-posted transaction Sideband upstream posted transaction was launched and completed successfully.

3.14.10.3 SBMSGNOGO: Slave TAP to IOSF SB triggered execution

This TDR remote register is implemented as a single TDR with glue logic to manage three opcodes. This opcode and TDR name description association is for the benefit of TAP-based SystemRDL and post-silicon DFx users.

This opcode allows a user to store a pending transaction that will be launched with a trigger event on either (programmable) the tpsb_trigin[1:0] inputs. The transaction may a read or write to any Sideband endpoint. If it is a write then only this opcode is necessary. If the transaction is a read then the user must write the appropriate message attributes and poll the SBMSGRSP TDR for the data completion response. The SBMSG* contains the same protocol fields as a message transaction would appear on the interface. This allows different messages to be sent from the TAP to an agent. This feature supports simple messages, messages with data, register access reads and write and completions with and without data.

JTAG Register	
Register name	SBMSGNOGO
JTAG IR	0x32
Default	0
JTAG DR length	96

JTAG Register	
DR shift size	96
Power well	User defined

Register Name: SBMSGNOGO				
Bits	Access	Default	Label	Bit Description
95:88	WO	0	sbmsgbyte10	Sideband message byte 10: SimpMsg: NA MsgD: data[63:56] RegAccRd: NA RegAccWr: data[31:24]
87:80	WO	0	sbmsgbyte9	Sideband message byte 9: SimpMsg: NA MsgD: data[55:48] RegAccRd: NA RegAccWr: data[23:16]
79:72	WO	0	sbmsgbyte8	Sideband message byte 8: SimpMsg: NA MsgD: data[47:40] RegAccRd: NA RegAccWr: data[15:8]
71:64	WO	0	sbmsgbyte7	Sideband message byte 7: SimpMsg: NA MsgD: data[39:32] RegAccRd: NA RegAccWr: data[7:0]
63:56	RW_V	0	sbmsgbyte6	Sideband message byte 6: SimpMsg: NA MsgD: data[31:24] RegAccRd: address[15:8] RegAccWr: address[15:8]
55:48	RW_V	0	sbmsgbyte5	Sideband message byte 5: SimpMsg: NA MsgD: data[23:16] RegAccRd: address[7:0] RegAccWr: address[7:0]
47:40	RW_V	0	sbmsgbyte4	Sideband message byte 4: SimpMsg: NA MsgD: data[15:8] RegAccRd: fid RegAccWr: fid
39:32	RW_V	0	sbmsgbyte3	Sideband message byte 3: SimpMsg: NA MsgD: data[7:0] RegAccRd: {0000, fbe[3:0]} RegAccWr: {0000, fbe[3:0]}

Register Name: SBMSGNOGO				
31:24	RW_V	0	sbmsgbyte2	Sideband message byte 2: SimpMsg: {EH, rsvd, tag} MsgD: {EH, rsvd, tag} RegAccRd: {EH, addrlen, bar, tag} RegAccWr: {EH, addrlen, bar, tag} Note1: EH bit is automatically set by the TAP2SB logic block if the parameter SAI_AWARE = 1.
23:16	WO	0	sbmsgbyte1	Sideband message byte 1: All message types are supported for the Opcode field except the completions with and without data (0x21, 0x20). TAP2SB will not initiate a transaction with these values written in this field.
15:8	WO	0	sbmsgbyte0	Sideband message byte 0: All message types: Dest (destination)
7:6	WO	0	msgtrigsel	Sideband message trigger select 00: Trigger disable 01: Select Trigger source 0 10: Select Trigger source 1 11: Reserved
5	WO	0	unused	Unused bit reserved for future use.
4	WO	0	data64en	Data with 64-bit enable. This mode is for debugging Vendor Defined Messages primarily to/from the Power Management Unit (PMU). 0: Use 32-bit data 1: Use 64-bit data
3	WO	0	npwrite	Indicates if the write message is non-posted (npwrite=1) or posted (npwrite=0). This may apply to either register access write messages, simple messages or messages with data. Read requests are always sent as a non-posted message for any message type (register access or message with data). 0: Write messages with be posted 1: Write messages with be non-posted.
2	WO	0	broadcasten	Enable a broadcast/multi-cast transaction. 0: Use endpoint defined source ID code. This the strap value applied to this endpoint. 1: Use 0xFE in the source ID code field.

Register Name: SBMSGNOGO				
1:0	RO_V	0	status	This bit field indicates the current status of pending transactions with the sideband endpoint. 00: No transaction launched. TAP/sideband logic block has not received update_dr for MSGGO(0x30), or TAP/sideband logic has received update_dr for MSGNOGO (0x32) but external trigger has not come, or TAP/sideband logic has not received update_dr for MSGGO(0x18), or 01:Transaction has started. TAP/sideband logic has started launching a transaction at sideband upstream interface but launch as not completed yet (busy) A request was accepted by the IOSF sideband logic and the transaction is in progress; EOM has not come yet, or Trigger condition has been met for launching sideband transaction. 10: A non-posted is pending Sideband upstream non-posted transaction has launched to Sideband fabric; waiting for completion. IOSF Sideband logic is waiting for Sideband Downstream Completion from Sideband Fabric for Non-posted transaction Note: For an sideband upstream posted transaction, this status value will be not be set 11: Transaction completed Completion was received from sideband fabric for non-posted transaction Sideband upstream posted transaction was launched and completed successfully.

3.14.10.4 SBEXTADDR: TAP Extended Address

This is the extended address for SoCs that require a 48-bit config register access mechanism over sideband. This is an optional register.

JTAG Register	
Register name	SBEXTADDR
JTAG IR	0x33
Default	0
JTAG DR length	32
DR shift size	32
Power well	User defined

Register Name: SBEXTADDR				
Bits	Access	Default	Label	Bit Description

Register Name: SBEXTADDR				
31:0	W	0	extAddr	Extended Address[31:0] The address information written in this register will be applied to overall SB address as [47:16].

3.14.10.5 POVRSBFID: Parallel Message Function ID

This is the function ID value that is used for the complete parallel message function ID. The function ID field is not included in the parallel message sent from the Narrow Test Interface (NTI). An enable bit allows the parallel message bus to be available for HVM test purposes.

JTAG Register	
Register name	POVRSBFID
JTAG IR	0x34
Default	0
JTAG DR length	9
DR shift size	9
Power well	User defined

Register Name: TAPC_PARLLFID				
Bits	Access	Default	Label	Bit Description
8	RW	0x0	pmsgactive	Override access to from Parallel Message Bus from NTI 0: No Sideband access from the parallel message bus 1: Sideband access from the parallel message bus
7:0	RW	0x0	pmsgfid	TAP access to set the FID (Function ID at Sideband interface) for Parallel Message Bus

3.14.10.6 SBROOTSPC: Sideband Root Space

This register holds the value for the root space that is sent with each transaction that has the expanded header enabled. The expanded header will send both the SAI and the root space. If the root space is not used than register should be cleared to logic 0. The default value is logic 0 so the user does not have to write this register if it is not intended to be used. Security access to this opcode is set by the parameter that affects the other opcodes 0x30-0x33.

Note: This register supports the root space parameter being set to the maximum value defined in IOSF rev1.1 (RS_WIDTH = 3). It has a fixed width because the shift register length will change if the RS_WIDTH strap changes and therefore it will change the programming model which is not desirable. Validation test writers should be aware of this and only program the value that is supported by the SoC.

JTAG Register	
Register name	SBROOTSPC
JTAG IR	0x35
Default	0
JTAG DR length	5
DR shift size	5

JTAG Register	
Power well	User defined

Register Name: SBROOTSPC				
Bits	Access	Default	Label	Bit Description
4	RW	0x0	sbPokEn	Power ok enable. This bit will force the assertion of side_pok signal on the TAP2SB top level interface. The user must assert this signal before uploading the SBMSG for a new transaction. Note: this bit is only active if the parameter TPSB_SIDEPOK_IS_TIED_TO_LOGIC1 is set to 0.
3:0	RW	0x0	rootspace	Sideband root space. This register is mapped to the rs[3:0] field in the SAIRS expanded header. This register is fixed and equivalent to RS_WIDTH = 3. The SoC may support smaller widths which means a value programmed in this register may not be supported by the endpoint. Only valid values are programmed into this register which match the maximum width support by the SoC.

groupingula bits will select a VISA register for this ULM/PLM/CLM. The ents of visadata.d by the visaaddr bit field. ut and ou

4 Implementation Details

Table 24. Chapter Revision History

Revision Number	Description	Revision Date
0.90_rc2a	No updates to this section.	February 2012
0.90_rc4	Updated figure, table and section references after moving to the new HAS template.	March 2012

4.1 Implementation Size Estimates

The reader should refer to the SEG-SIP MAS for this information. A basic sTAP with no addition parameter features and no internal TDRs will be 751 gates.

4.2 ITP TAP IO Specification (Rev1.7)

This section is the ITP-XDP TAP IO specification that is copy/pasted here for reference. This documentation is from the latest known revision 1.7 and a revision history is shown in Table 25.

Table 25. TAP IO Spec Revision Table

Revision	Date	Summary Review Release
0.01	06/07/2001	First Draft
1.10	06/05/2001	Initial Release
1.20	06/13/2001	Assign Intel Document Number
1.30	11/08/2001	Removed ITP requirement for 1.5V tolerance. Systems with 1.5V VTAP voltage will still require 1.5V tolerance to be backward compatible.
1.40	11/30/2001	Corrected a documentation error in the tap dc specification example, and clarified VTAP definition.
1.50	02/11/03	Revised for XDP support.
1.60	02/11/03	Revised for XDP only support.
1.70	02/16/03	Added definition for Tck recovery validation.

4.2.1 SoC TAP IO Specification

This section describes the SoC IO specifications that are required to provide a robust electrical interface with the Validation Tools Group ITP-XDP. A summary is shown in Table 26.

The JTAG TAP chain must be designed to support a 75MHz JTAG TCK.

A 50MHz TCK operation is the minimum with a stretch goal of 100MHz.

VTAP is the processor/system JTAG IO termination voltage.

All the JTAG TAP IO signals shall experience no damage from either maximum continuous voltages of VTAP + 0.3V or from short duration (<5ns) overshoot transients to VTAP + 0.4V.

The TDO driver must be designed as an open drain style driver (ie only sink current), or tri-state when the scan engine is not in the shift state. It is strongly recommended that the TDO

driver includes a circuit to provide a 50 ohm assist, active from the rising edge of TDO till the next rising edge of TCK. This will make the TDO path less susceptible to Transmission Line effects.

The TDO Ron impedance should not exceed 12 Ohms. The TDO net is expected to be terminated to 50 ohms (or less) to VTAP.

The TMS, TDI, and TRST# inputs are not required to have hysteresis. These signals should have a nominal threshold of ½ VTAP. The ITP-XDP is guaranteed to drive these signals below 1/3 * VTAP (low) and above 2/3 * VTAP (high).

The TCK input must have hysteresis centered on ½ VTAP. TCK hysteresis should be designed to be between 10% and 33% of the nominal expected VTAP. This specification is only a guideline, meant to help strike a compromise between noise immunity and design complexity.

For example:

> If VTAP is expected to range between 0.7V and 1.3V, then the TCK input should be designed to have at least 100mV of hysteresis centered on ½ VTAP, where 100mV is 1/10th of the nominal 1.0V VTAP.

Table 26. TAP IO Parameters Spec

Example of a Processor TAP Signals DC Specification (VTAP (VTT) is expected to vary from 0.7V to 1.3V, nominal =1.0V)				
Symbol	Parameter	Min	Max	Units
Vil	TMS, TDI, TRST# input low voltage	-0.5	1/3* VTAP	V
Vih	TMS, TDI, TRST# input high voltage	2/3* VTAP	VTAP + 0.3	V
Vhys	TCK input hysteresis	100	330	mV
Vt+	TCK input low to high threshold voltage	½* VTAP + 0.05	½* VTAP + 0.165	V
Vt-	TCK input high to low threshold voltage	½* VTAP - 0.165	½* VTAP – 0.05	V
Ron	TDO Ron impedance	TBD	12	Ohms

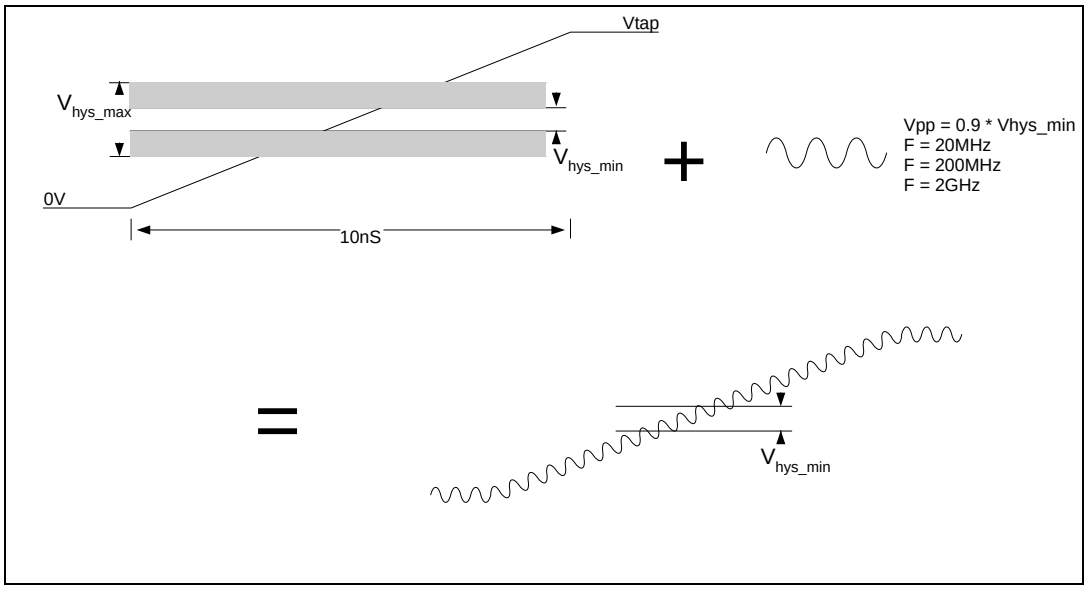
A blanking circuit should be used to mask unwanted transitions (glitches) at the TCK input until the hysteresis circuit has equilibrated after a valid edge detect.

TCK recovery should be validated using the stimulus waveform shown in Figure 43.

Trapezoidal waveform from 0V to VTAP with a 10nS rise and fall time.

Superimposed sine wave Vpp = 0.9 * Vhys_min, F = 20MHz, F = 200MHz, F = 2GHz

Figure 43. TCK Recovery Waveforms



5 Interfaces

Not applicable to this IP

5.1 IP Interface Signal List

Not applicable to this IP

5.2 Functional Signals

Not applicable to this IP

5.3 Power, Resets, and Firewalls

Not applicable to this IP

5.4 Clocks

Not applicable to this IP

5.5 DFx

Not applicable to this IP

6 Interrupts

Not applicable to this IP

7 Clocking

STAP operates the state machine on a single clock (ftap_tck). The frequency can vary from 1 to 100 MHz. An optional clock can be used for any other functional clock such as the microbreak point feature.

Table 27. Clock Domains

Domain	Scope	Range of Frequencies
ftap_tck		0-10ns /0-100Mhz
Optional functional clock		Varies based on the clock you have chosen

7.1 Clocking Requirements

Not applicable to this IP

7.2 Clock Gating

Not applicable to this IP

7.3 PLLs

Not applicable to this IP

7.4 Clocking Assumptions

Not applicable to this IP

8 Power Management

Not applicable to this IP

8.1 Isolation Gates and Power Gating

Not applicable to this IP

8.2 Behavior in Various Power States

Not applicable to this IP

9 Reset Transactions

9.1 Reset Controls

Table 28. sTAP Resets

Reset	Description	Use	Active
fdfx_powergood	This signal indicates that the voltage level is good.	Used to reset the following while the physical voltage is good, and the signal is still zero: ▪ sTAP FSM ▪ All TAP registers	Low
ftap_trst_b	This signal propagates from the IO pins to each sTAP. This reset is a pure delay path from the IO.	Used to reset the following: ▪ sTAP FSM ▪ TAP network configuration registers	Low
ftap_tms	This signal controls the TAP FSM	Asserting the TMS active for five cycles resets the TAP state machine as described in the JTAG protocol	High

9.2 Reset Initialization Flows

sTAP has no power-up sequence. sTAP becomes available once the power is good ("powergood" signal equals 1). After the power is good, the TAP FSM will be in the TLRESET state. The TAP FSM follows the JTAG interface pins accordingly. Refer to section 2.1.6 for more information about the TAP FSM.

9.3 Reset Assumptions

Not applicable to this IP

10 Transaction Flows

Not applicable to this IP

10.1 Transactional Interface

Not applicable to this IP

10.2 Ordering/Coherency Rules

Not applicable to this IP

11 Register Descriptions

Table 29. List of Internal Registers

No.	Register Description	Offset
1	Reserved	0x00
2	SAMPLE/PRELOAD	0x01
3	IDCODE	0x02
4	PRELOAD	0x03
5	CLAMP	0x04
6	USERCODE	0x05
7	INTEST	0x06
8	RUNBIST	0x07
9	HIGHZ	0x08
10	EXTEST	0x09
11	TOGGLE_SETUP	0x0A
12	SELECTIVE_TOGGLE	0x0B
13	SLVIDCODE	0x0C
14	EXTEST_TOGGLE	0x0D
15	EXTEST_PULSE	0x0E
16	EXTEST_TRAIN	0x0F
17	Reserved	0x10
18	TAP_SELECT	0x11
19	TAP_SELECT_OVR	0x12
20	Reserved	0x13
21	Reserved	0x14
22	Reserved	0x15
23	Reserved	0x16
24	Reserved	0x17
25	Reserved	0x18
26	Reserved	0x19
27	CLAMP_HOLD	0x1A
28	CLAMP_RELEASE	0x1B
29	IC_RESET	0x1C
30	TAPC reserved opcode space for future TAP HAS spec use.	0x1D-0x2F
31	TAPC SoC defined opcodes	0x20-0xFE
32	BYPASS	0xFF

12 Area and Gate Count Goals

Not applicable to this IP

13 Power Goals

Not applicable to this IP

14 Performance Goals

Not applicable to this IP

14.1 Maximum Performance Targets

Not applicable to this IP

14.2 Performance Environment

Not applicable to this IP

15 Functional and Performance Monitoring

Not applicable to this IP

16 Reliability, Availability, and Serviceability (RAS)

Not applicable to this IP

17 Security and Access Control

17.1 Security Feature Requirements

Applying security to DFx is critical in protecting internally stored keys and other intellectual property of our SoC designs. However, we must enable our customers with tools through third-party vendors and provide on-die DFx features necessary so they can perform debug and validation of their platforms using our silicon. It must be controlled in such a way as to only allow the level of access necessary for the customer to bring the product to market. The overall security solution involving customers to obtain keys that is authenticated by the security engine is beyond the scope of this specification. This section describes how a policy is broadcasted from the centralized security aggregator which is interpreted by the IP-blocks to provide the access customers they need while protecting Intel's and the customer's assets.

This DFx secure policy interface is a group of signals composed of one bus and two control signals that distributes a uniform policy value to all IP-blocks in the SoC. IP-blocks instantiate a security plug-in IP form the Intel Re-use Repository (IRR) that translates the current policy and enables or disables access to DFx features within the agent or IP-block. The DFx secure policy signal group is composed of a binary encoded policy bus, a policy update and an early boot exit signals. This signal group is distributed from a centralized DFx security aggregator. The aggregator encodes the policy based on the Debug Life Cycle State (DLCS) from the fuse block and who is authenticated to access what level of DFx features. In other words, it is a combination of DLCS and the type of key a user had to gain access. No other source can drive this signal group other than the aggregator. Polices are defined in detail in the Chassis DFx security framework specification (reference listed in section 19.4) and it not re-iterated here. This section will define the DFx security plug-in IP and describe its functionality with examples.

To help describe the plug-in IP we need to review the policy table listed in Table 30. There are eight policies defined that are mandatory which makes them common across all SoCs. Other policies may be enforced for a particular SoC for their market segment. All policy values must by fully decoded and unused policy values are decoded as OEM unlocked. The DFx secure policy is defined as a parameter but it is fixed with a width of four (4) bits for this version of the spec which is passed down from Chassis DFx spec as a requirement. It is mandatory that all IPs be the same width to eliminate security holes due to aliasing of the most significant bits. In the description field we use VISA as an example and a color assignment for each policy to help the reader interpret what level of access is being interpreted by the policy. A security described as green means it is public and anyone has access. From a VISA perspective the debug signals are benign. An orange level of access requires a key. We don't want anyone accessing these signals, they are useful for customers to get their platforms debug and validated but should be used once the product is being sold. A DFx security level that is labeled as red is only meant for Intel. Therefore, a color is meant to convey a how open the DFx features are from the customer's point of view. Red means is the most "wide-open" level of access and should be viewed similar to a red colored document (labeled as Intel Top Secret). The use of colors as applied to VISA signals or DFx features are useful for illustration purposes and employed throughout this document.

Table 30. DFx Secure Policy Bus Encoding

Policy Name	DFx Secure Policy Bus Encode	Description Examples	
		Locked	Color Code
Security Locked	0000	Locked TAP	Green TAP
Functionality Locked	0001	Locked TAP	Green TAP
Security Unlocked	0010	Unlocked TAP	Red TAP
Reserved	0011	Locked TAP	Green TAP

Policy	DFx Secure Policy	Description Examples	
Intel Unlocked	0100	Unlocked TAP	Red TAP
OEM Unlocked	0101	Partial unlocked TAP	Orange TAP
Revoked (Reserved)	0110	Locked TAP	Green TAP
"User 1" Unlocked	0111	Partial unlocked TAP	Orange TAP
"User 2" Unlocked	1000	Partial unlocked TAP	Orange TAP
"User 3" Unlocked	1001	Partial unlocked TAP	Orange TAP
"User 4" Unlocked	1010	Partial unlocked TAP	Orange TAP
"User 5" Unlocked	1011	Partial unlocked TAP	Orange TAP
"User 6" Unlocked	1100	Partial unlocked TAP	Orange TAP
"User 7" Unlocked	1101	Partial unlocked TAP	Orange TAP
"User 8" Unlocked	1110	Partial unlocked TAP	Orange TAP
Part Disabled	1111	Locked TAP	Green TAP

17.2 DFx Secure Policy Plugin IP-block

The DFx security plug-in IP-block is shown in Figure 44. There are three defined IOSF DFx fabric facing interface signals groups and two supporting IP-facing signals buses. Several parameters allow the IP-block to provide the security enabling coverage required by each agent or IP-block. The left side of the block shows the IOSF DFx defined signals are listed in section 12.1 and in the IOSF DFx HAS rev1.2.They are labeled as as fdfx_earlyboot_exit, fdfx_secure_policy[3:0] and fdfx_policy_update in the diagram. The policy is broadcasted to all IP-blocks on the secure policy bus and latched with the policy update signal. The policy value is an input to a lookup table that is defined by the policy matrix parameter. A latch is used rather than a flop for two reasons, one is to latch the value and hold the policy constant after the boot window closes when the level of DFx access is known. A second reason is to latch the static policy value after the IP-block comes out of a low-power state. More information is available in the IOSF DFx rev1.2 and the Chassis DFx security framework HAS.

The DFx secure plug-in IP is re-used for the TAP but it doesn't use all of its capabilities. The oem_secure_policy, sb_policy_ovr_value and VISA signals are not used for the TAP security plug-in. The signals are listed in Table 31 for how they are used.

Figure 44. TAP DFx Security Plugin Block Diagram

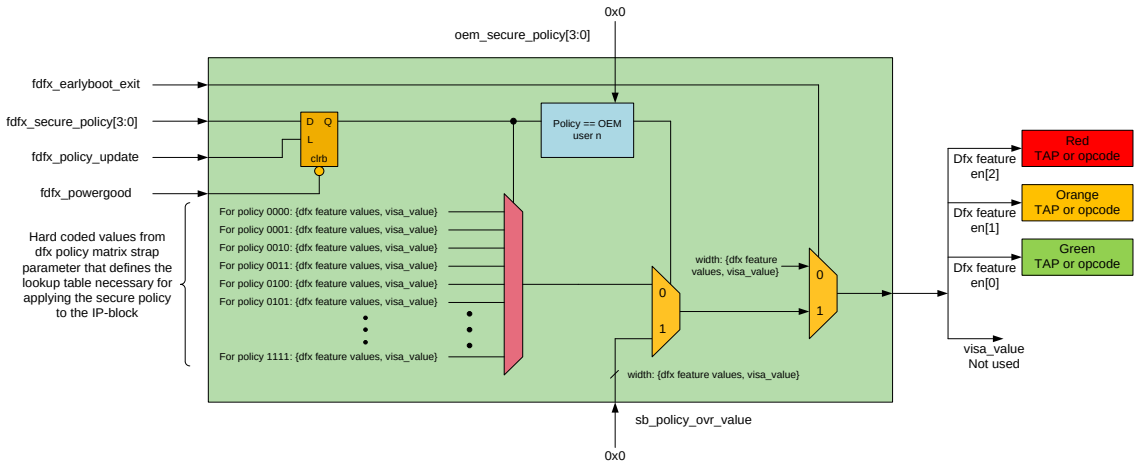


Table 31. TAP's use of DFx Secure Policy Plugin Signal List

Signal	I/O	Description
fdfx_secure_policy [N-1:0]	I	Fabric DFx secure policy: Refer to Table 12-2 for the signal definition. Where N = DFXSECURE_WIDTH = 4 for this revision.
fdfx_policy_update	I	Fabric DFx policy update. Refer to Table 12-2 for the signal definition.
fdfx_earlyboot_exit	I	Fabric DFx early boot exit. Refer to Table 12-2 for the signal definition.
dfxsecure_feature_en[0]	O	DFx secure feature enable [0]. This signal is used for enabling green TAP access with the select register and green opcode access within this TAP.
dfxsecure_feature_en[1]	O	DFx secure feature enable [1]. This signal is used for enabling orange TAP access with the select register and orange opcode access within this TAP.
dfxsecure_feature_en[2]	O	DFx secure feature enable [2]. This signal is used for enabling red TAP access with the select register and red opcode access within this TAP.
visa_all_dis	O	VISA all disable: Not used, not connected
visa_customer_dis	O	VISA customer disable. Not used, not connected
sb_policy_ovr_value[M+1:0]	I	Sideband policy override value. Not used, connect to 0x0
oem_secure_policy[N-1:0]	I	OEM specific secure policy value. Not used, connect to 0x0.

17.2.1 DFx Secure Plug-in Parameters for TAP

Table 32 shows the policy plug-in parameters used by the TAPs. The secure width is set to 4 for this revision of spec which is compliant to IOSF DFx rev1.2 and the number of features to secure is fixed to three (3). TAPs will only support a green, orange and red level of security so we assign one DFx feature enable per color code. The policy matrix is configured such that each color code is accessible based on the policy definition. For example, a red opcode or red TAP access (via the Select register) should only be enabled with policy 2 or 4. The policy select register parameter assigns an enumerated color code for each TAP that defines if it accessible under the current policy value. The last parameter is the secure policy opcode which assigns a color code to each opcode within a TAP. This protects test data registers within the TAP. If an orange TAP is active, it may still contain registers that are only accessible with red access. This provides a second level of protection for the SoC so that it doesn't have to duplicate TAPs to segregate sensitive registers.

Table 32. DFx Secure Plugin IP Parameters

Parameter Name	Parameter Value(s)	Comments
User defined Parameters		
STAP_DFX_SECURE_ POLICY_SELECTREG	colorcode(enum)	The order of the security codes are the order of the child TAPs that this parent TAP controls.
STAP_DFX_SECURE_ POLICY_OPCODE	opcode(hex), colorcode(enum)	A two dimensional array of values. opcode: a hex value per opcode definition. Unused opcode point to the bypass register and are defaulted to green.

Parameter Name	Parameter Value(s)	Comments
Local TAP Parameters		
STP_DFX_SECURE_WIDTH	4	This parameter is fixed at 4 for this revision of the spec. (Defined for this SoC HAS revision, the IOSF DFx rev1.2 and in the Chassis mod-dfx Gen3 HAS)
STP_DFX_NUM_OF_FEATURES _TO_SECURE	3	Fixed for CLTAP, sTAP and WTAP.
STAP_DFX_SECURE_ POLICY_MATRIX	[STAP_DFX_SECURE_WIDTH-1:0][STAP_DFX_NUM_OF_FEATURES_TO _SECURE+1:0]	This parameter determines the lookup table necessary to assign the policy with the DFx feature(s) including VISA access. This parameter is fixed for TAPs refer to Table 33.
STAP_DFX_EARLYBOOT_FEATURE_ENABLE	[STAP_DFX_NUM_OF_FEATURES_TO _SECURE+1:0]	This parameter sets the hard coded value for the early debug window for this agent/IP-block. Early boot DFx feature enable for the TAPs must be SECURE_GREEN. STAP_DFX_EARLYBOOT_FEATURE_ENABLE [4:2] = {3'b001 , STAP_DFX_VISA_BLACK}
USE_SB_OVR	0	The TAP does not use the Sideband override feature.
SECURE_GREEN	0	Assigned local parameter value for enumerating the color codes.
SECURE_ORANGE	1	Assigned local parameter value for enumerating the color codes.
SECURE_RED	2	Assigned local parameter value for enumerating the color codes.
USER1_ORANGE	7	User 1 defined unlocked
USER2_ORANGE	8	User 2 defined unlocked
USER3_ORANGE	9	User 3 defined unlocked
USER4_ORANGE	10	User 4 defined unlocked
USER5_ORANGE	11	User 5 defined unlocked
USER6_ORANGE	12	User 6 defined unlocked
USER7_ORANGE	13	User 7 defined unlocked
USER8_ORANGE	14	User 8 defined unlocked

17.2.2 Policy Matrix for TAP

Table 33 list all of the policies and the associated settings for the DFx feature enables for each color code. The table in general should be fixed for the TAPs because it is known set of features, namely, TAP enabling and opcode enabling. Another reason we have a fixed table is because the TAP will only support three values of green, orange, and red. Based on this table the policy matrix is assigned as the following:

- STAP_DFX_SECURE_ POLICY_MATRIX[6:0]= {00111, 01011, 10011, 00111, 10011, 00111, 00111}
- STAP_DFX_SECURE_ POLICY_MATRIX[14:7] = {01011}

- STAP_DFX_SECURE_ POLICY_MATRIX[15] = {00111}

Table 33. Policy Matrix Table for TAPs

Policy Name	DFx Secure Policy Bus Encode	DFx Feature en[2] Red	DFx Feature en[1] Orange	DFx Feature en[0] Green	VISA not used	Description with TAP Examples
Security Locked	0000	0	0	1	11	Public/Locked, Green TAP
Functionality Locked	0001	0	0	1	11	Public/Locked, Green TAP
Security Unlocked	0010	1	0	0	11	Intel-Only/ Private/Unlocked, Red TAP
Reserved	0011	0	0	1	11	Public/Locked, Green TAP
Intel Unlocked	0100	1	0	0	11	Intel-Only/Private/Unlocked, Red TAP
OEM Unlocked	0101	0	1	0	11	OEM/Partial Unlock, Orange TAP
Revoked (Reserved)	0110	0	0	1	11	Public/Locked, Green TAP
"User 1" Unlocked	0111	0	1	0	11	OEM/Partial Unlock, Orange TAP
"User 2" Unlocked	1000	0	1	0	11	OEM/Partial Unlock, Orange TAP
"User 3" Unlocked	1001	0	1	0	11	OEM/Partial Unlock, Orange TAP
"User 4" Unlocked	1010	0	1	0	11	OEM/Partial Unlock, Orange TAP
"User 5" Unlocked	1011	0	1	0	11	OEM/Partial Unlock, Orange TAP
"User 6" Unlocked	1100	0	1	0	11	OEM/Partial Unlock, Orange TAP
"User 7" Unlocked	1101	0	1	0	11	OEM/Partial Unlock, Orange TAP
"User 8" Unlocked	1110	0	1	0	11	OEM/Partial Unlock, Orange TAP
Part Disabled	1111	0	0	1	11	Public/Locked, Green TAP

17.3 Security Mitigations

Not applicable to this IP

17.4 Security Threats and Attack Points

Not applicable to this IP

17.5 Architecture Review

Not applicable to this IP

17.5.1 Assets

Not applicable to this IP

17.5.2 Security Objectives

Not applicable to this IP

17.6 Security Risk Assessment

Not applicable to this IP

17.7 Additional Security Information

Not applicable to this IP

17.8 Corner Cases

Not applicable to this IP

18 Safety

Not applicable to this IP

18.1 Safety Feature Traceability Mapping

Not applicable to this IP

18.2 Safety Risk Assessment

Not applicable to this IP

18.3 Additional Safety Information

Not applicable to this IP

19 Validation Guidance

Not applicable to this IP

20 Test Requirements

Not applicable to this IP

20.1 MBIST and PBIST

Not applicable to this IP

20.2 Array Freeze

Not applicable to this IP

20.3 Scan

Not applicable to this IP

20.4 MISR

Not applicable to this IP

20.5 DFT Feature Definition

Not applicable to this IP

20.6 DFT PLC Implementation Timeline Checklist

Not applicable to this IP

20.7 SCAN System Connectivity and Features

Not applicable to this IP

20.8 Validation Test & Environment Requirements

Not applicable to this IP

20.9 HVM Tooling and Flows Requirements

Not applicable to this IP

20.10 HVM Reset Requirements

Not applicable to this IP

20.11 HVM Clocking and Determinism Requirements

Not applicable to this IP

20.12 DFT Chassis Topology and Connectivity

Not applicable to this IP

20.13 DFT implementation and Usage Details

Not applicable to this IP

21 Debug Requirements

Not applicable to this IP

21.1 Visualization of Internal Signals Architecture (VISA)

Not applicable to this IP

21.2 IP Firmware Support for Messaging to North Peak

Not applicable to this IP

21.3 Trigger Events and/or Responses

Not applicable to this IP

21.4 Debug Trace Fabric (DTF)

Not applicable to this IP

21.5 Debug Power Domains and Clocks

Not applicable to this IP

21.6 Other

Not applicable to this IP

22 Fuses

Not applicable to this IP

23 Pin List/Ball Map

The only strap in the design is ftap_slvidcode[31:0].

Table 34. Top-Level Primary Interface Pins

Pin Name	Direction	Source/Destination	Width
ftap_tck	Input	TAP network	1
ftap_tms	Input	TAP network	1
ftap_trst_b	Input	TAP network	1
ftap_tdi	Input	TAP network	1
ftap_slvidcode	Input	TAP network	32
atap_tdo	Output	TAP network	1
atap_tdo_en	Output	TAP network	1
fdfx_powergood	Input	Clock and Reset	1

Table 35. DFX Secure Pins

Pin Name	Direction	Source/Destination	Width
fdfx_secure_policy	Input	Security Engine Fuse Bus	STAP_DFX_SECURE_WIDTH
fdfx_policy_update	Input	Security Engine	1
fdfx_earlyboot_exit	Input	Security Engine	1

Table 36. Control Signals to 0.7 TAP Network

Pin Name	Direction	Source/Destination	Width
sftapnw_ftap_secsel	Output	0.7 TAP network	STAP_NUMBER_OF_TAPS
sftapnw_ftap_enabletdo	Output	0.7 TAP network	STAP_NUMBER_OF_TAPS
sftapnw_ftap_enabletap	Output	0.7 TAP network	STAP_NUMBER_OF_TAPS

Table 37. Primary JTAG Ports to 0.7 TAP Network

Pin Name	Direction	Source/Destination	Width
sntapnw_ftap_tck	Output	0.7 TAP network	1
sntapnw_ftap_tms	Output	0.7 TAP network	1
sntapnw_ftap_trst_b	Output	0.7 TAP network	1
sntapnw_ftap_tdi	Output	0.7 TAP network	1
sntapnw_atap_tdo	Input	0.7 TAP network	1

Table 38. Boundary Scan Pins

Pin Name	Direction	Source/Destination	Width
stap_fbscan_tck	Output	BScan Cell	1
stap_abscan_tdo	Input	BScan Cell	1
stap_fbscan_capturedr	Output	BScan Cell	1
stap_fbscan_shiftdr	Output	BScan Cell	1
stap_fbscan_updatedr	Output	BScan Cell	1

Pin Name	Direction	Source/Destination	Width
stap_fbscan_updatedr_clk	Output	BScan Cell	1
stap_fbscan_runbist_en	Output	BScan Cell	1
stap_fbscan_highz	Output	BScan Cell	1
stap_fbscan_extogen	Output	BScan Cell	1
stap_fbscan_intest_mode	Output	BScan Cell	1
stap_fbscan_chainen	Output	BScan Cell	1
stap_fbscan_mode	Output	BScan Cell	1
stap_fbscan_extogsig_b	Output	BScan Cell	1
stap_fbscan_d6init	Output	BScan Cell	1
stap_fbscan_d6actestsig_b	Output	BScan Cell	1
stap_fbscan_d6select	Output	BScan Cell	1

Table 39. Remote Test Data Register Pins

Pin Name	Direction	Source/Destination	Width
<RTDR name>_rtdr_tdo	Input	Remote TDR	Per RTDR register
rtdr_tdi	Output	Remote TDR	1
rtdr_capture	Output	Remote TDR	1
rtdr_shift	Output	Remote TDR	1
<RTDR_name>_rtdr_irdec	Output	Remote TDR	per RTDR register
rtdr_rti	Output	Remote TDR	Optional 1
rtdr_tck	Output	Remote TDR	1
tap_rtdr_rst_b	Output	Remote TDR	Optional 1

24 Wish List and Recommendations

Not applicable to this IP

25 Opens

This section is a list of opens specifically for items that are marked as **Opens** throughout the HAS. This list functions like a table of contents.

To mark an item as **Open,** highlight the text that explains the open, and then select the Opens style from the Styles pane (if your Styles pane is not visible, press Ctrl + Alt + Shift + S).

To update the list of open items in section, right-click anywhere in the list, and then select Update Field > Update entire table. The list updates with all of the items.

25.1 Opens List

The following is a list of any open items throughout this document:

Note: Be sure to ***right click*** on the list and update fields when adding Opens to this document.

List of Opens