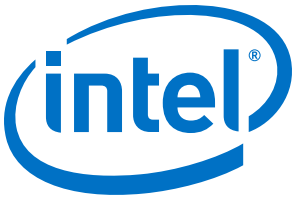
Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China



Virtual JTAG (altera_virtual_jtag) IP Core User Guide

Updated for Intel® Quartus® Prime Design Suite: **16.1**

Subscribe
Send Feedback

UG-SLDVRTL | 2018.07.19
Latest document on the web: PDF | HTML

Contents

Altera Virtual JTAG (altera_virtual_jtag) IP Core User Guide

The Altera Virtual JTAG (altera_virtual_jtag) IP core provides access to the PLD source through the JTAG interface. This IP core is optimized for Intel® device architectures. Using IP cores in place of coding your own logic saves valuable design time, and offers more efficient logic synthesis and device implementation. You can scale the IP core's size by setting parameters.

Related Information

Introduction to Intel FPGA IP Cores

Introduction

The Virtual JTAG IP core allows you to create your own software solution for monitoring, updating, and debugging designs through the JTAG port without using I/O pins on the device, and is one feature in the On-Chip Debugging Tool Suite. The Intel Quartus® Prime software or JTAG control host identifies each instance of this IP core by a unique index. Each IP core instance functions in a flow that resembles the JTAG operation of a device. The logic that uses this interface must maintain the continuity of the JTAG chain on behalf the PLD device when this instance becomes active.

With the Virtual JTAG IP core you can build your design for efficient, fast, and productive debugging solutions. Debugging solutions can be part of an evaluation test where you use other logic analyzers to debug your design, or as part of a production test where you do not have a host running an embedded logic analyzer. In addition to debugging features, you can use the Virtual JTAG IP core to provide a single channel or multiple serial channels through the JTAG port of the device. You can use serial channels in applications to capture data or to force data to various parts of your logic.

Each feature in the On-Chip Debugging Tool Suite leverages on-chip resources to achieve real time visibility to the logic under test. During runtime, each tool shares the JTAG connection to transmit collected test data to the Intel Quartus Prime software for analysis. The tool set consists of a set of GUIs, IP core intellectual property (IP) cores, and Tcl application programming interfaces (APIs). The GUIs provide the configuration of test signals and the visualization of data captured during debugging. The Tcl scripting interface provides automation during runtime.

The Virtual JTAG IP core provides you direct access to the JTAG control signals routed to the FPGA core logic, which gives you a fine granularity of control over the JTAG resource and opens up the JTAG resource as a general-purpose serial communication interface. A complete Tcl API is available for sending and receiving transactions into your device during runtime. Because the JTAG pins are readily accessible during runtime, this IP core enables an easy way to customize a JTAG scan chain internal to the device, which you can then use to create debugging applications.

Examples of debugging applications include induced trigger conditions evaluated by a Signal Tap logic analyzer by exercising test signals connected to the analyzer instance, a replacement for a front panel interface during the prototyping phase of the design, or inserted test vectors for exercising the design under test.

The infrastructure is an extension of the JTAG protocol for use with Intel-specific applications and user applications, such as the Signal Tap logic analyzer.

Installing and Licensing Intel FPGA IP Cores

The Intel Quartus Prime software installation includes the Intel FPGA IP library. This library provides many useful IP cores for your production use without the need for an additional license. Some Intel FPGA IP cores require purchase of a separate license for production use. The Intel FPGA IP Evaluation Mode allows you to evaluate these licensed Intel FPGA IP cores in simulation and hardware, before deciding to purchase a full production IP core license. You only need to purchase a full production license for licensed Intel IP cores after you complete hardware testing and are ready to use the IP in production.

The Intel Quartus Prime software installs IP cores in the following locations by default:

Figure 1. IP Core Installation Path

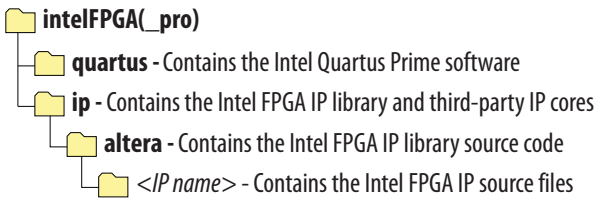


Table 1. IP Core Installation Locations

Location	Software	Platform
<drive>:\intelFPGA_pro\quartus\ip\altera	Intel Quartus Prime Pro Edition	Windows*
<drive>:\intelFPGA\quartus\ip\altera	Intel Quartus Prime Standard Edition	Windows
<home directory>:/intelFPGA_pro/quartus/ip/altera	Intel Quartus Prime Pro Edition	Linux*
<home directory>:/intelFPGA/quartus/ip/altera	Intel Quartus Prime Standard Edition	Linux

On-Chip Debugging Tool Suite

The On-Chip Debugging Tool Suite enables real time verification of a design and includes the following tools:

Table 2. On-Chip Debugging Tool Suite

Tool	Description	Typical Circumstances for Use
Signal Tap Logic Analyzer	Uses FPGA resources to sample tests nodes and outputs the information to the Intel Quartus Prime software for display and analysis.	You have spare on-chip memory and want functional verification of your design running in hardware.
Signal Probe	Incrementally routes internal signals to I/O pins while preserving the results from your last place-and-route.	You have spare I/O pins and want to check the operation of a small set of control pins using either an external logic analyzer or an oscilloscope.
Logic Analyzer Interface (LAI)	Multiplexes a larger set of signals to a smaller number of spare I/O pins. LAI allows you to select which signals are switched onto the I/O pins over a JTAG connection.	You have limited on-chip memory and have a large set of internal data buses that you want to verify using an external logic analyzer. Logic analyzer vendors, such as Tektronics and Agilent, provide integration with the tool to improve usability.
In-System Memory Content Editor	Displays and allows you to edit on-chip memory.	You want to view and edit the contents of either the instruction cache or data cache of a Nios® II processor application.
In-System Sources and Probes	Provides a way to drive and sample logic values to and from internal nodes using the JTAG interface.	You want to prototype a front panel with virtual buttons for your FPGA design.
Virtual JTAG Interface	Opens the JTAG interface so that you can develop your own custom applications.	You want to generate a large set of test vectors and send them to your device over the JTAG port to functionally verify your design running in hardware.

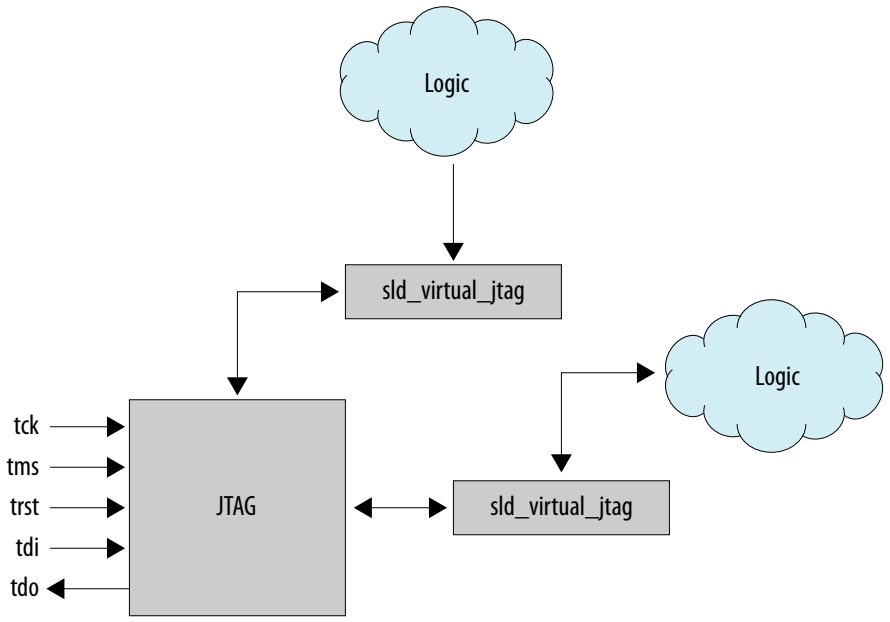
Related Information

System Debugging Tools Overview

Applications of the Virtual JTAG IP Core

You can instantiate single or multiple instances of the Virtual JTAG IP core in your HDL code. During synthesis, the Intel Quartus Prime software assigns unique IDs to each instance, so that each instance is accessed individually. You can instantiate up to 128 instances of the Virtual JTAG IP core. The figure below shows a typical application in a design with multiple instances of the IP core.

Figure 2. Application Example



The hub automatically arbitrates between multiple applications that share a single JTAG resource. Therefore, you can use the IP core in tandem with other on-chip debugging applications, such as the Signal Tap logic analyzer, to increase debugging visibility. You can also use the IP core to provide simple stimulus patterns to solicit a response from the design under test during run-time, including the following applications:

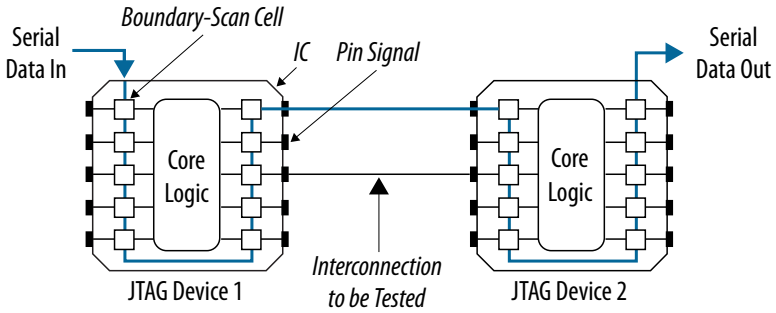
- To diagnose, sample, and update the values of internal parts of your logic. With this IP core, you can easily sample and update the values of the internal counters and state machines in your hardware device.
- To build your own custom software debugging IP using the Tcl commands to debug your hardware. This IP communicates with the instances of the Virtual JTAG IP core inside your design.
- To construct your design to achieve virtual inputs and outputs.
- If you are building a debugging solution for a system in which a microprocessor controls the JTAG chain, you cannot use the Signal Tap logic analyzer because the JTAG control must be with the microprocessor. You can use low-level controls for the JTAG port from the Tcl commands to direct microprocessors to communicate with the Virtual JTAG IP core inside the device core.

JTAG Protocol

The original intent of the JTAG protocol (standardized as IEEE 1149.1) was to simplify PCB interconnectivity testing during the manufacturing stage. As access to integrated circuit (IC) pins became more limited due to tighter lead spacing and FPGA packages, testing through traditional probing techniques, such as “bed-of-nails” test fixtures, became infeasible. The JTAG protocol alleviates the need for physical access to IC pins via a shift register chain placed near the I/O ring. This set of registers near the I/O ring, also known as boundary scan cells (BSCs), samples and forces values out onto the I/O pins. The BSCs from JTAG-compliant ICs are daisy-chained into a serial-shift chain and driven via a serial interface.

During boundary scan testing, software shifts out test data over the serial interface to the BSCs of select ICs. This test data forces a known pattern to the pins connected to the affected BSCs. If the adjacent IC at the other end of the PCB trace is JTAG-compliant, the BSC of the adjacent IC samples the test pattern and feeds the BSCs back to the software for analysis. The figure below illustrates the boundary-scan testing concept.

Figure 3. IEEE Std. 1149.1 Boundary-Scan Testing



Because the JTAG interface shifts in any information to the device, leaves a low footprint, and is available on all Intel devices, it is considered a general purpose communication interface. In addition to boundary scan applications, Intel devices use the JTAG port for other applications, such as device configuration and on-chip debugging features available in the Intel Quartus Prime software.

Related Information

IEEE 1149.1 JTAG Boundary-Scan Testing

JTAG Circuitry Architecture

The basic architecture of the JTAG circuitry consists of the following components:

- A set of Data Registers (`DRs`)
- An Instruction Register (`IR`)
- A state machine to arbitrate data (known as the Test Access Port (TAP) controller)
- A four- or five-pin serial interface, consisting of the following pins:
  - Test data in (`TDI`), used to shift data into the IR and DR shift register chains
  - Test data out (`TDO`), used to shift data out of the IR and DR shift register chains
  - Test mode select (`TMS`), used as an input into the TAP controller
  - `TCK`, used as the clock source for the JTAG circuitry
  - `TRST` resets the TAP controller. This is an optional input pin defined by the 1149.1 standard.

Note: The `TRST` pin is not present in the Cyclone device family.

The bank of DRs is the primary data path of the JTAG circuitry. It carries the payload data for all JTAG transactions. Each DR chain is dedicated to serving a specific function. Boundary scan cells form the primary DR chain. The other DR chains are used for identification, bypassing the IC during boundary scan tests, or a custom set

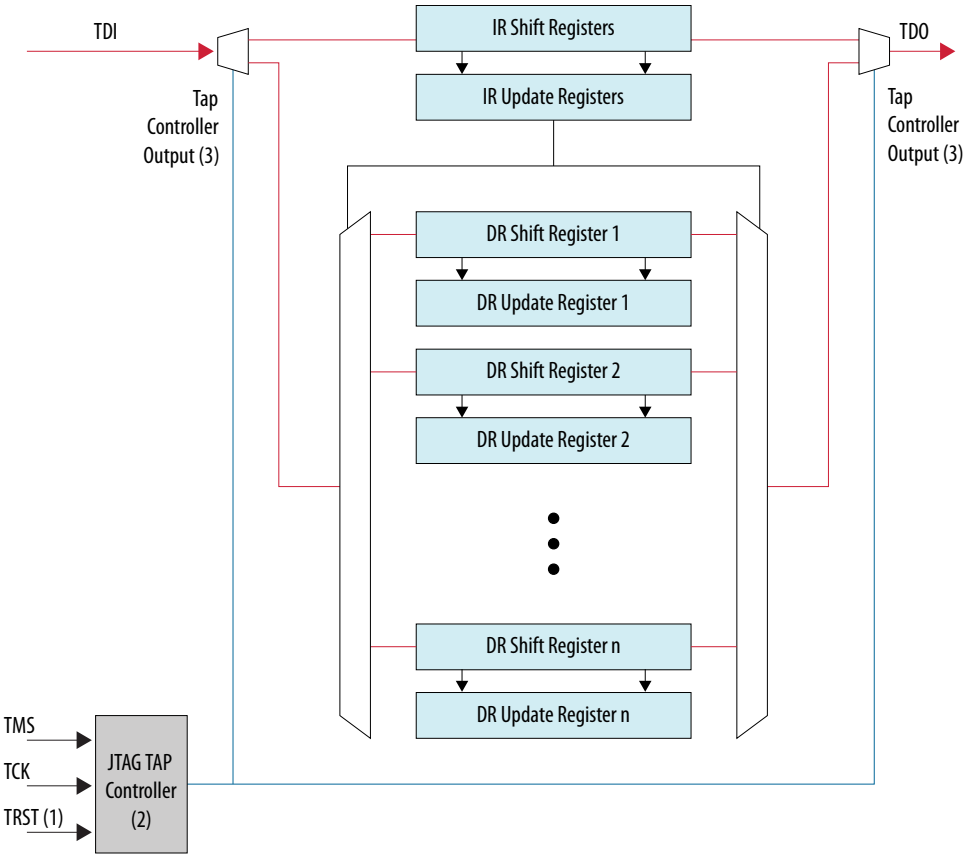
of register chains with functions defined by the IC vendor. Intel uses two of the DR chains with user-defined IP that requires the JTAG chain as a communication resource, such as the on-chip debugging applications. The Virtual JTAG IP core, in particular, allows you to extend the two DR chains to a user-defined custom application.

You use the instruction register to select the bank of Data Registers to which the TDI and TDO must connect. It functions as an address register for the bank of Data Registers. Each IR instruction maps to a specific DR chain.

All shift registers that are a part of the JTAG circuitry (IR and DR register chains) are composed of two kinds of registers: shift registers, which capture new serial shift input from the TDI pin, and parallel hold registers, which connect to each shift register to hold the current input in place when shifting. The parallel hold registers ensure stability in the output when new data is shifted.

The figure below shows a functional model of the JTAG circuitry. The TRST pin is an optional pin in the 1149.1 standard and not available in Cyclone devices. The TAP controller is a hard controller; it is not created using programmable resources. The major function of the TAP controller is to route test data between the IR and DR register chains.

Figure 4. Functional Model of the JTAG Circuitry

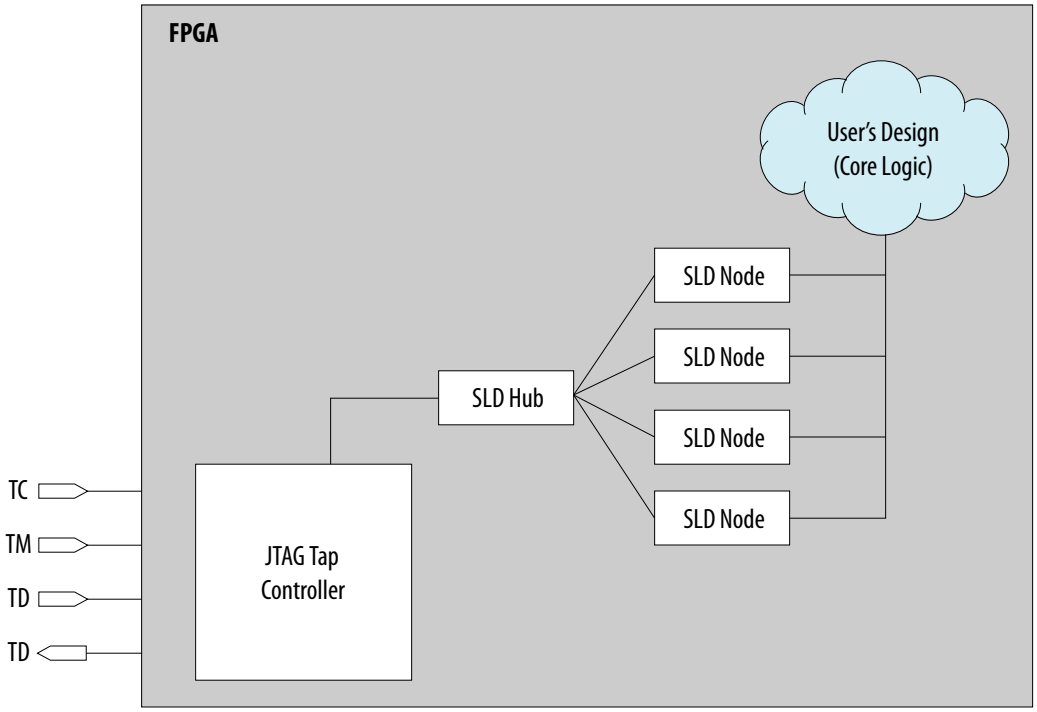


System-Level Debugging Infrastructure

On-chip debugging tools that require the JTAG resources share two Data Register chain paths; `USER1` and `USER0` instructions select the Data Register chain paths. The datapaths are an extension of the JTAG circuitry for use with the programmable logic elements in Intel devices.

Because the JTAG resource is shared among multiple on-chip applications, an arbitration scheme must define how the `USER0` and `USER1` scan chains are allocated between the different applications. The system-level debugging (SLD) infrastructure defines the signaling convention and the arbitration logic for all programmable logic applications using a JTAG resource. The figure below shows the SLD infrastructure architecture.

Figure 5. System Level Debugging Infrastructure Functional Model



Transaction Model of the SLD Infrastructure

In the presence of an application that requires the JTAG resource, the Intel Quartus Prime software automatically implements the SLD infrastructure to handle the arbitration of the JTAG resource. The communication interface between JTAG and any IP cores is transparent to the designer. All components of the SLD infrastructure, except for the JTAG TAP controller, are built using programmable logic resources.

The SLD infrastructure mimics the IR/DR paradigm defined by the JTAG protocol. Each application implements an Instruction Register, and a set of Data Registers that operate similarly to the Instruction Register and Data Registers in the JTAG standard. Note that the Instruction Register and the Data Register banks implemented by each

application are a subset of the USER1 and USER0 Data Register chains. The SLD infrastructure consists of three subsystems: the JTAG TAP controller, the SLD hub, and the SLD nodes.

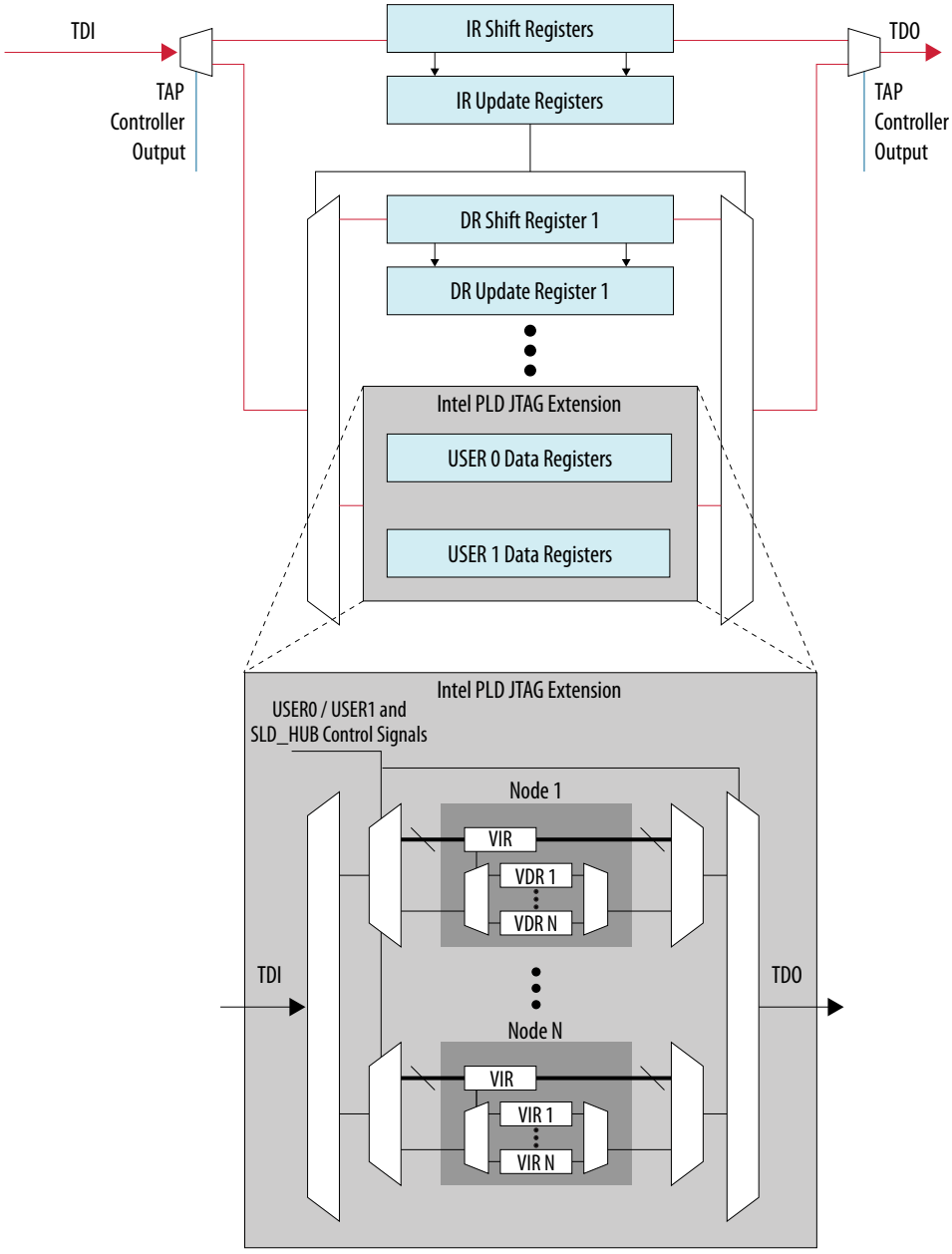
The SLD hub acts as the arbiter that routes the TDI pin connection between each SLD node, and is a state machine that mirrors the JTAG TAP controller state machine.

The SLD nodes represent the communication channels for the end applications. Each instance of IP requiring a JTAG communication resource, such as the Signal Tap logic analyzer, would have its own communication channel in the form of a SLD node interface to the SLD hub. Each SLD node instance has its own Instruction Register and bank of DR chains. Up to 255 SLD nodes can be instantiated, depending on resources available in your device.

Together, the sld_hub and the SLD nodes form a virtual JTAG scan chain within the JTAG protocol. It is virtual in the sense that both the Instruction Register and DR transactions for each SLD node instance are encapsulated within a standard DR scan shift of the JTAG protocol.

The Instruction Register and Data Registers for the SLD nodes are a subset of the USER1 and USER0 Data Registers. Because the SLD Node IR/DR register set is not directly part of the IR/DR register set of the JTAG protocol, the SLD node Instruction Register and Data Register chains are known as Virtual IR (VIR) and Virtual DR (VDR) chains. The figure below shows the transaction model of the SLD infrastructure.

Figure 6. Extension of the JTAG Protocol for PLD Applications



SLD Hub Finite State Machine

The SLD hub decodes `TMS` independently from the hard JTAG TAP controller state machine and implements an equivalent state machine (called the "SLD hub finite state machine") for the internal JTAG path. The SLD hub performs a similar function for the VIR and VDR chains that the TAP controller performs for the JTAG IR and DR chains. It enables an SLD node as the active path for the `TDI` pin, selects the TDI data between

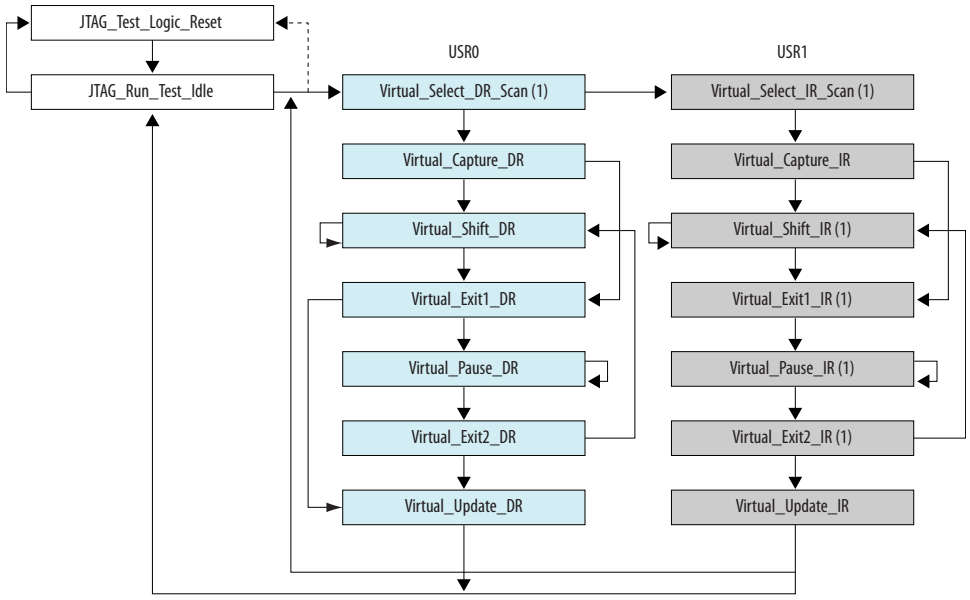
the VIR and VDR registers, controls the start and stop of any shift transactions, and controls the data flow between the parallel hold registers and the parallel shift registers of the VIR and VDR.

Because all shifts to VIR and VDR are encapsulated within a DR shift transaction, an additional control signal is necessary to select between the VIR and VDR data paths. The SLD hub uses the `USER1` command to select the VIR data path and the `USER0` command to select the VDR data path.

This state information, including a bank of enable signals, is forwarded to each of the SLD nodes. The SLD nodes perform the updates to the VIR and VDR according to the control states provided by the `sld_hub`. The SLD nodes are responsible for maintaining continuity between the `TDI` and `TDO` pins.

The figure below shows the SLD hub finite state machine. There is no direct state signal available to use for application design.

Figure 7. sld_hub Finite State Machine



Virtual JTAG Interface Description

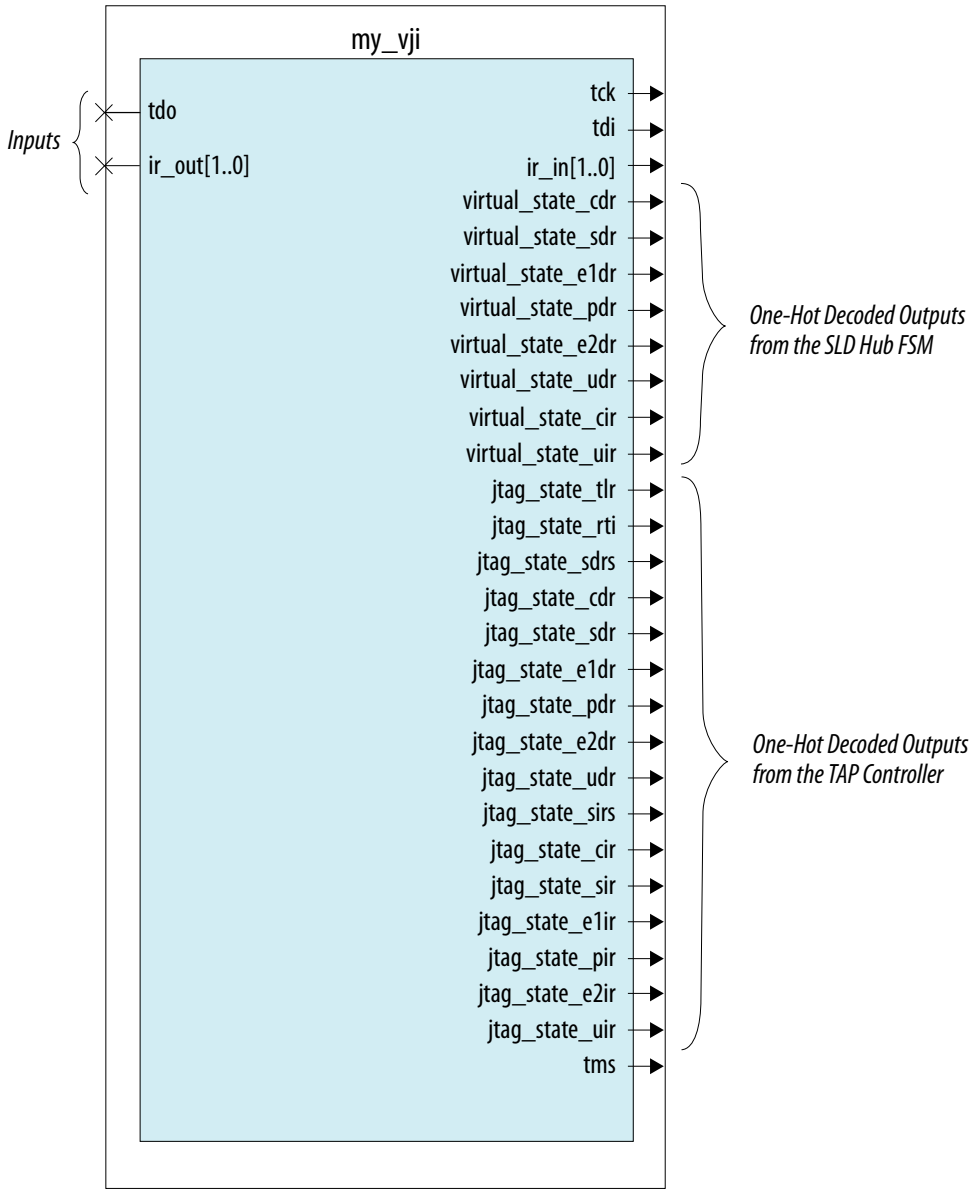
The Virtual JTAG Interface implements an SLD node interface, which provides a communication interface to the JTAG port. The IP core exposes control signals that are part of the SLD hub; namely, JTAG port signals, all finite state machine controller states of the TAP controller, and the SLD hub finite state machine. Additionally, each instance of the Virtual JTAG IP cores contain the virtual Instruction Register for the SLD node. Instantiation of this IP core automatically infers the SLD infrastructure, and one SLD node is added for each instantiation.

The Virtual JTAG IP core provides a port interface that mirrors the actual JTAG ports. The interface contains the JTAG port pins, a one-hot decoded output of all JTAG states, and a one-hot decoded output of all the virtual JTAG states. Virtual JTAG states are the states decoded from the SLD hub finite state machine. The `ir_in` and `ir_out` ports are the parallel input and output to and from the VIR. The VIR ports are used to

select the active VDR datapath. The JTAG states and `TMS` output ports are provided for debugging purposes only. Only the virtual JTAG, `TDI`, `TDO`, and the `IR` signals are functional elements of the IP core. When configuring this IP core using the parameter editor, you can hide `TMS` and the decoded JTAG states.

The figure below shows the input and output ports of the virtual JTAG IP core. The JTAG TAP controller outputs and TMS signals are used for informational purposes only. These signals can be exposed using the **Create primitive JTAG state signal ports** option in the parameter editor.

Figure 8. Input and Output Ports of the Virtual JTAG IP Core



Input Ports

Table 3. Input Ports for the Virtual JTAG IP Core

Port name	Required	Description	Comments
tdo	Yes	Writes to the TDO pin on the device.	
ir_out[]	No	Virtual JTAG instruction register output. The value is captured whenever virtual_state_cir is high.	Input port [SLD_IR_WIDTH-1..0] wide. Specify the width of this bus with the SLD_IR_WIDTH parameter.

Output Ports

Table 4. Output Ports for the Virtual JTAG IP Core

Port Name	Required	Description	Comments
tck	Yes	JTAG test clock.	Connected directly to the TCK device pin. Shared among all virtual JTAG instances.
tdi	Yes	TDI input data on the device. Used when virtual_state_sdr is high.	Shared among all virtual JTAG instances.
ir_in[]	No	Virtual JTAG instruction register data. The value is available and latched when virtual_state_uir is high.	Output port [SLD_IR_WIDTH-1..0] wide. Specify the width of this bus with the SLD_IR_WIDTH parameter.

Table 5. High-Level Virtual JTAG State Signals

Port Name	Required	Description	Comments
virtual_state_cdr	No	Indicates that virtual JTAG is in Capture_DR state.	
virtual_state_sdr	Yes	Indicates that virtual JTAG is in Shift_DR state.	In this state, this instance is required to establish the JTAG chain for this device.
virtual_state_e1dr	No	Indicates that virtual JTAG is in Exit1_DR state.	
virtual_state pdr	No	Indicates that virtual JTAG is in Pause_DR state.	The Intel Quartus Prime software does not cycle through this state using the Tcl command.
virtual_state_e2dr	No	Indicates that virtual JTAG is in Exit2_DR state.	The Intel Quartus Prime software does not cycle through this state using the Tcl command.
virtual_state_udr	No	Indicates that virtual JTAG is in Update_DR state.	
virtual_state_cir	No	Indicates that virtual JTAG is in Capture_IR state.	
virtual_state_uir	No	Indicates that virtual JTAG is in Update_IR state.	

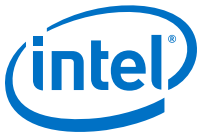


Table 6. Low-Level Virtual JTAG State Signals

Port Name	Required	Description	Comments
jtag_state_tlr	No	Indicates that the device JTAG controller is in the Test_Logic_Reset state.	Shared among all virtual JTAG instances.
jtag_state_rti	No	Indicates that the device JTAG controller is in the Run_Test/Idle state.	Shared among all virtual JTAG instances.
jtag_state_sdrs	No	Indicates that the device JTAG controller is in the Select_DR_Scan state.	Shared among all virtual JTAG instances.
jtag_state_cdr	No	Indicates that the device JTAG controller is in the Capture_DR state.	Shared among all virtual JTAG instances.
jtag_state_sdr	No	Indicates that the device JTAG controller is in the Shift_DR state.	Shared among all virtual JTAG instances.
jtag_state_e1dr	No	Indicates that the device JTAG controller is in the Exit1_DR state.	Shared among all virtual JTAG instances.
jtag_state_pdr	No	Indicates that the device JTAG controller is in the Pause_DR state.	Shared among all virtual JTAG instances.
jtag_state_e2dr	No	Indicates that the device JTAG controller is in the Exit2_DR state.	Shared among all virtual JTAG instances.
jtag_state_udr	No	Indicates that the device JTAG controller is in the Update_DR state.	Shared among all virtual JTAG instances.
jtag_state_sirs	No	Indicates that the device JTAG controller is in the Select_IR_Scan state.	Shared among all virtual JTAG instances.
jtag_state_cir	No	Indicates that the device JTAG controller is in the Capture_IR state.	Shared among all virtual JTAG instances.
jtag_state_sir	No	Indicates that the device JTAG controller is in the Shift_IR state.	Shared among all virtual JTAG instances.
jtag_state_e1ir	No	Indicates that the device JTAG controller is in the Exit1_IR state.	Shared among all virtual JTAG instances.
jtag_state_pir	No	Indicates that the device JTAG controller is in the Pause_IR state.	Shared among all virtual JTAG instances.
jtag_state_e2ir	No	Indicates that the device JTAG controller is in the Exit2_IR state.	Shared among all virtual JTAG instances.
jtag_state_uir		Indicates that the device JTAG controller is in the Update_IR state.	Shared among all virtual JTAG instances.
tms		TMS input pin on the device.	Shared among all virtual JTAG instances.

Parameters

Table 7. Virtual JTAG Parameters

Parameter	Type	Required	Description
SLD_AUTO_INSTANCE_INDEX	String	Yes	Specifies whether the Compiler automatically assigns an index to the Virtual JTAG instance. Values are **YES** or **NO**. When you specify **NO**, you can find the auto assigned value of INSTANCE_ID in the **quartus_map** file. When you specify **NO**, you must define INSTANCE_INDEX. If the index specified is not unique in a design, the Compiler automatically reassigns an index to the instance. The default value is **YES**.
SLD_INSTANCE_INDEX	Integer	No	Specifies a unique identifier for every instance of alt_virtual_jtag when AUTO_INSTANCE_ID is specified to **YES**. Otherwise, this value is ignored.
SLD_IR_WIDTH	Integer	Yes	Specifies the width of the instruction register ir_in[] of this virtual JTAG between 1 and 24. If omitted, the default is 1.

Design Flow of the Virtual JTAG IP Core

Designing with the Virtual JTAG IP core includes the following processes:

- Configuring the Virtual JTAG IP core with the desired Instruction Register length and instantiating the IP core.
- Building the glue logic for interfacing with your application.
- Communicating with the Virtual JTAG instance during runtime.

In addition to the JTAG datapath and control signals, the Virtual JTAG IP core encompasses the VIR. The Instruction Register size is configured in the parameter editor. The Instruction Register port on the Virtual JTAG IP core is the parallel output of the VIR. Any updated VIR information can be read from this port after the virtual_state_uir signal is asserted.

After instantiating the IP core, you must create the VDR chains that interface with your application. To do this, you use the virtual instruction output to determine which VDR chain is the active datapath, and then create the following:

- Decode logic for the VIR
- VDR chains to which each VIR maps
- Interface logic between your VDR chains and your application logic

Your glue logic uses the decoded one-hot outputs from the IP core to determine when to shift and when to update the VDR. Your application logic interfaces with the VDR chains during any one of the non-shift virtual JTAG states.

For example, your application logic can parallel read an updated value that was shifted in from the JTAG port after the virtual_state_uir signal is asserted. If you load a value to be shifted out of the JTAG port, you would do so when the virtual_state_cdr signal is asserted. Finally, if you enable the shift register to clock out information to TDO, you would do so during the assertion of virtual_state_sdr.

Maintaining TDI-to-TDO connectivity is important. Ensure that all possible instruction codes map to an active register chain to maintain connectivity in the TDI-to-TDO datapath. Intel recommends including a bypass register as the active register for all unmapped IR values.

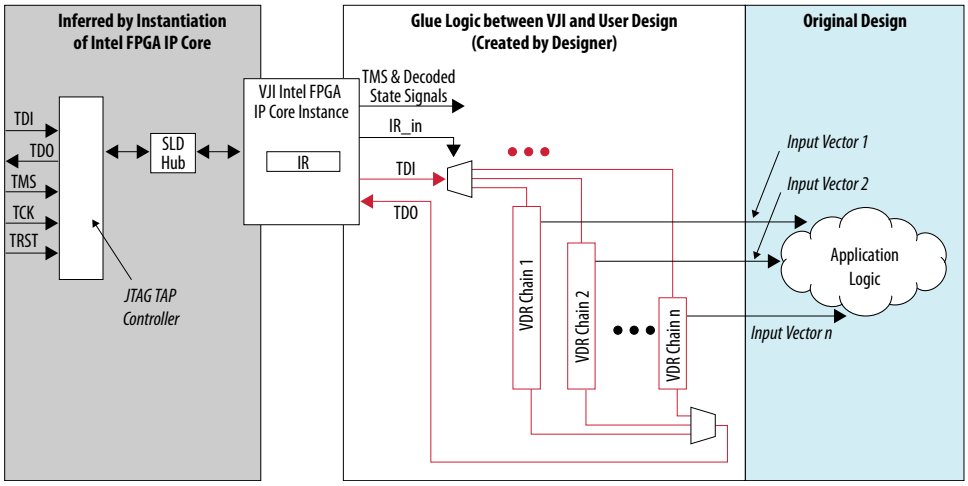
Note that TCK (a maximum 10-MHz clock, if using an Intel programming cable) provides the clock for the entire SLD infrastructure. Be sure to follow best practices for proper clock domain crossing between the JTAG clock domain and the rest of your application logic to avoid metastability issues. The decoded virtual JTAG state signals can help determine a stable output in the VIR and VDR chains.

After compiling and downloading your design into the device, you can perform shift operations directly to the VIR and VDR chains using the Tcl commands from the quartus_stp executable and an Intel programming cable (for example, an Intel FPGA Download Cable, a MasterBlaster™ cable, or an Intel FPGA Parallel Port Cable). The quartus_stp executable is a command-line executable that contains Tcl commands for all on-chip debug features available in the Intel Quartus Prime software.

The figure below shows the components of a design containing one instance of the Virtual JTAG IP core. The TDI–to–TDO datapath for the virtual JTAG chain, shown in red, consists of a bank of DR registers. Input to the application logic is the parallel output of the VDR chains. Decoded state signals are used to signal start and stop of shift transactions and signals when the VDR output is ready.

The IR_out port, not shown, is an optional input port you can use to parallel load the VIR from the FPGA core logic.

Figure 9. Block Diagram of a Design with a Single Virtual JTAG Instantiation



Simulation Model

The virtual JTAG IP core contains a functional simulation model that provides stimuli that mimic VIR and VDR shifts. You can configure the stimuli using the parameter editor. You can use this simulation model for functional verification only, and the operation of the SLD hub and the SLD node-to-hub interface is not provided in this simulation model.

Run-Time Communication

The Tcl API for the Virtual JTAG IP core consists of a set of commands for accessing the VIR and VDR of each virtual JTAG instance.

These commands contain the underlying drivers for accessing an Intel programming cable and for issuing shift transactions to each VIR and VDR. The table below provides the Tcl commands in the `quartus_stp` executable that you can use with the Virtual JTAG IP core, and are intended for designs that use a custom controller to drive the JTAG chain.

Each instantiation of the Virtual JTAG IP core includes an instance index. All instances are sequentially numbered and are automatically provided by the Intel Quartus Prime software. The instance index starts at instance index `0`. The VIR and VDR shift commands described in the table decode the instance index and provide an address to the SLD hub for each IP core instance. You can override the default index provided by the Intel Quartus Prime software during configuration of the IP core.

The table below provides the Tcl commands in the quartus_stp executable that you can use with the Virtual JTAG IP core, and are intended for designs that use a custom controller to drive the JTAG chain.

Table 8. Virtual JTAG IP Core Tcl Commands

Command	Arguments	Description
Device virtual ir shift	`-instance_index` <*instance_index*> `-ir_value` <*numeric_ir_value*> `-no_captured_ir_value`(1) `-show_equivalent_device_ir_dr_shift`(1)	Perform an IR shift operation to the virtual JTAG instance specified by the `instance_index`. Note that `ir_value` takes a numerical argument.
Device virtual dr shift	`-instance_index` <*instance_index*> `-dr_value` <*dr_value*> `-length` <*data_register_length*> `-no_captured_dr_value`(1) `-show_equivalent_device_ir_dr_shift` `-value_in_hex`(1)	Perform a DR shift operation to the virtual JTAG instance.
Get hardware names	NONE	Queries for all available programming cables.
Open device	`-device_name` <*device_name*> `-hardware_name` <*hardware_name*>	Selects the active device on the JTAG chain.
Close device	NONE	Ends communication with the active JTAG device.
Device lock	`-timeout` <*timeout*>	Obtains exclusive communication to the JTAG chain.
Device unlock	NONE	Releases `device_lock`.
Device ir shift	`-ir_value` <*ir_value*> `-no_captured_ir_value`	Performs a IR shift operation.
Device dr shift	`-dr_value` <*dr_value*> `-length` <*data register length*>	Performs a DR shift operation.

continued...

(1) This argument is optional.

Command	Arguments	Description
	-no_captured_dr_value -value_in_hex	

Central to Virtual JTAG IP core are the `device_virtual_ir_shift` and `device_virtual_dr_shift` commands. These commands perform the shift operation to each VIR/VDR and provide the address to the SLD hub for the active JTAG datapath.

Each `device_virtual_ir_shift` command issues a `USER1` instruction to the JTAG Instruction Register followed by a DR shift containing the VIR value provided by the `ir_value` argument prepended by address bits to target the correct SLD node instance.

Note: Use the `-no_captured_ir_value` argument if you do not care about shifting out the contents of the current VIR value. Enabling this argument increases the speed of the VIR shift transaction by eliminating a command cycle within the underlying transaction.

Similarly, each `device_virtual_dr_shift` command issues a `USER0` instruction to the JTAG Instruction Register followed by a DR shift containing the VDR value provided by the `dr_value` argument. These commands return the underlying JTAG transactions with the `show_equivalent_device_ir_dr_shift` option set.

Note: The `device_virtual_ir_shift` takes the `ir_value` argument as a numeric value. The `device_virtual_dr_shift` takes the `dr_value` argument by either a binary string or a hexadecimal string. Do not use numeric values for the `device_virtual_dr_shift`.

Running a DR Shift Operation Through a Virtual JTAG Chain

A simple DR shift operation through a virtual JTAG chain using an Intel download cable consists of the following steps:

1. Query for the Intel programming cable and select the active cable.
2. Target the desired device in the JTAG chain.
3. Obtain a device lock for exclusive communication to the device.
4. Perform a VIR shift.
5. Perform a VDR shift.
6. Release exclusive link with the device with the `device_unlock` command.
7. Close communication with the device with the `close_device` command.

Run-Time Communication

The Virtual JTAG IP core Tcl API requires an Intel programming cable. Designs that use a custom controller to drive the JTAG chain directly must issue the correct JTAG IR/DR transactions to target the Virtual JTAG IP core instances. The address values and register length information for each Virtual JTAG IP core instance are provided in the compilation reports.

The following figure shows the compilation report for a Virtual JTAG IP core Instance. The following table describes each column in the Virtual JTAG Settings compilation report.

Figure 10. Compilation Report

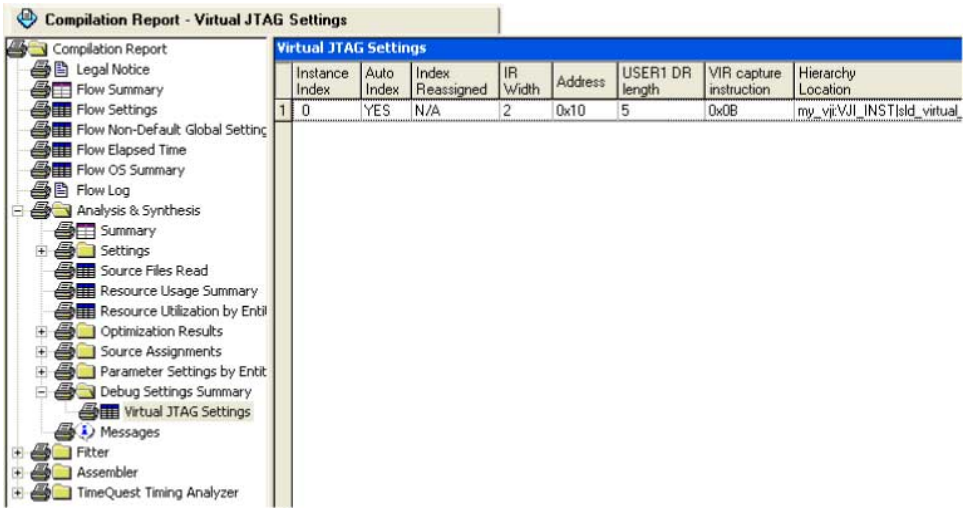


Table 9. Virtual JTAG Settings Description

Setting	Description
Instance Index	Instance index of the virtual JTAG IP core. Assigned at compile time.
Auto Index	Details whether the index was auto-assigned.
Index Re-Assigned	Details whether the index was user-assigned.
IR Width	Length of the Virtual IR register for this IP core instance; defined in the parameter editor.
Address	The address value assigned to the IP core by the compiler.
USER1 DR Length	The length of the USER1 DR register. The USER1 DR register encapsulates the VIR for all SLD nodes.
VIR Capture Instruction	Instruction value to capture the VIR of this IP core instance.

The Tcl API provides a way to return the JTAG IR/DR transactions by using the `show_equivalent_device_ir_dr_shift` argument with the `device_virtual_ir_shift` and `device_virtual_dr_shift` commands. The following examples use returned values of a virtual IR/DR shift to illustrate the format of the underlying transactions.

To use the Tcl API to query for the bit pattern in your design, use the `show_equivalent_device_ir_dr_shift` argument with the `device_virtual_ir_shift` and `device_virtual_dr_shift` commands.

Both examples are from the same design, with a single Virtual JTAG instance. The VIR length for the reference Virtual JTAG instance is configured to 3 bits in length.

Virtual IR/DR Shift Transaction without Returning Captured IR/DR Values

VIR shifts consist of a `USER1` (`0x0E`) IR shift followed by a DR shift to the virtual Instruction Register. The DR Scan shift consists of the value passed by the `–dr_value` argument. The length and value of the DR shift is dependent on the number of SLD nodes in your design. This value consists of address bits to the SLD node instance concatenated with the desired value of the virtual Instruction Register. The addressing scheme is determined by the Intel Quartus Prime software during design compilation.

The Tcl command examples below show a VIR/VDR transaction with the `no_captured_value` option set. The commands return the underlying JTAG shift transactions that occur.

Virtual IR Shift with the no_captured_value Option

```
device_virtual_ir_shift –instance_index 0 –ir_value 1 \
–no_captured_ir_value –show_equivalent_device_ir_dr_shift □
```

Returns:
Info: Equivalent device ir and dr shift commands
Info: device_ir_shift -ir_value 14
Info: device_dr_shift -length 5 -dr_value 11 -value_in_hex

Virtual DR Shift with the no_captured_value Option

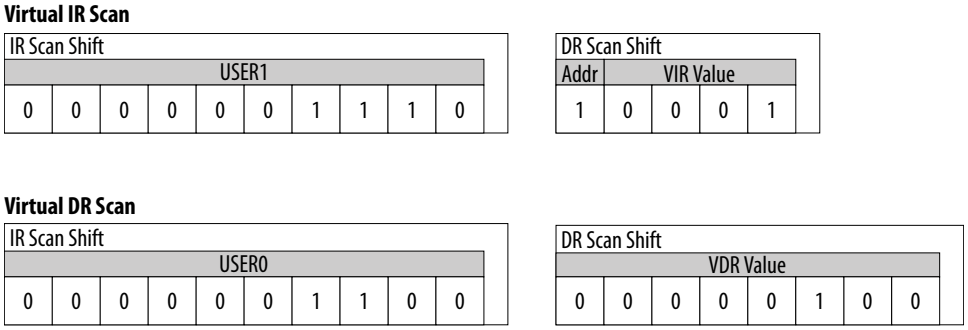
```
device_virtual_dr_shift –instance_index 0 –length 8 –dr_value \
04 –value_in_hex –no_captured_dr_value \
–show_equivalent_device_ir_dr_shift □
```

Returns:
Info: Equivalent device ir and dr shift commands
Info: device_ir_shift -ir_value 12
Info: device_dr_shift -length 8 -dr_value 04 -value_in_hex

The VIR value field in the figure below is four bits long, even though the VIR length is configured to be three bits long, and shows the bit values and fields associated with the VIR/VDR scans. The Instruction Register length for all Intel FPGAs and CPLDs is 10-bits long. The `USER1` value is `0x0E` and `USER0` value is `0x0C` for all Intel FPGAs and CPLDs. The Address bits contained in the DR scan shift of a VIR scan are determined by the Intel Quartus Prime software.

All `USER1` DR chains must be of uniform length. The length of the VIR value field length is determined by length of the longest VIR register for all SLD nodes instantiated in the design. Because the SLD hub VIR is four bits long, the minimum length for the VIR value field for all SLD nodes in the design is at least four bits in length. The Intel Quartus Prime Tcl API automatically sizes the shift transaction to the correct length. The length of the VIR register is provided in the Virtual JTAG settings compilation report. If you are driving the JTAG interface with a custom controller, you must pay attention to size of the `USER1` DR chain.

Figure 11. Equivalent Bit Pattern Shifted into Device by VIR/VDR Shift Commands



Virtual IR/DR Shift Transaction that Captures Current VIR/VDR Values

The Tcl command examples below show that the `no_captured_value` option is not set in the Virtual IR/DR shift commands and the underlying JTAG shift commands associated with each. In the VIR shift command, the command returns two `device_dr_shift` commands.

Virtual IR Shift

```
device_virtual_ir_shift -instance_index 0 -ir_value 1 \
-no_captured_ir_value -show_equivalent_device_ir_dr_shift
```

Returns:

```
Info: Equivalent device ir and dr shift commands
Info: device_ir_shift -ir_value 14
Info: device_dr_shift –length 5 –dr_value 0B –value_in_hex
Info: device_dr_shift -length 5 -dr_value 11 -value_in_hex
```

Virtual DR Shift

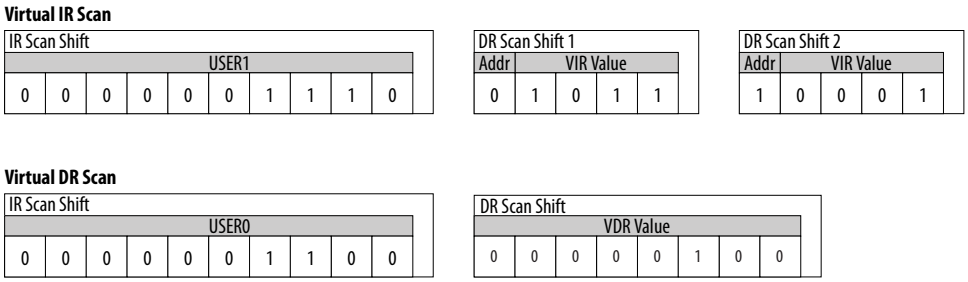
```
device_virtual_dr_shift -instance_index 0 -length 8 -dr_value \
04 -value_in_hex -show_equivalent_device_ir_dr_shift
```

Returns:

```
Info: Equivalent device ir and dr shift commands
Info: device_ir_shift -ir_value 12
Info: device_dr_shift -length 8 -dr_value 04 -value_in_hex
```

The figure below shows an example of VIR/VDR Shift Commands with captured IR values. DR Scan Shift 1 is the `VIR_CAPTURE` command, as shown in the figure below. It targets the VIR of the `sld_hub`. This command is an address cycle to select the active VIR chain to shift after `jtag_state_cir` is asserted. The `HUB_FORCE_IR` capture must be issued whenever you capture the VIR from a target SLD node that is different than the current active node. DR Scan Shift 1 targets the SLD hub VIR to force a captured value from Virtual JTAG instance 1 and is shown as the VIR_CAPTURE command. DR Scan Shift 2 targets the VIR of Virtual JTAG instance.

Figure 12. Equivalent Bit Pattern Shifted into Device by VIR/VDR Shift Commands with Captured IR Values



Note: If you use an embedded processor as a controller for the JTAG chain and your Virtual JTAG IP core instances, consider using the JAM Standard Test and Programming Language (STAPL). JAM STAPL is an industry-standard flow-control-based language that supports JTAG communication transactions. JAM STAPL is open source, with software downloads and source code available from the Intel website.

Related Information

- ISP & the Jam STAPL
- Embedded Programming With Jam STAPL

Reset Considerations when Using a Custom JTAG Controller

The SLD hub decodes `TMS` independently to determine the JTAG controller state. Under normal operation, the SLD hub mirrors all of the JTAG TAP controller states accurately. The JTAG pins (`TCK`, `TMS`, `TDI`, and `TDO`) are accessible to the core programmable logic; however, the JTAG TAP controller outputs are not visible to the core programmable logic. In addition, the hard JTAG TAP controller does not use any reset signals as inputs from the core programmable logic.

This can cause the following two situations in which control states of the SLD hub and the JTAG TAP controller are not in lock-step:

- An assertion of the device wide global reset signal (`DEV_CLRn`)
- An assertion of the `TRST` signal, if available on the device

`DEV_CLRn` resets the SLD hub but does not reset the hard TAP controller block. The TAP controller is meant to be decoupled from `USER` mode device operation to run boundary scan operations. Asserting the global reset signal does not disrupt boundary-scan test (BST) operation.

Conversely, the `TRST` signal, if available, resets the JTAG TAP controller but does not reset the SLD hub. The `TRST` signal does not route into the core programmable logic of the PLD.

To guarantee that the states of the JTAG TAP controller and the SLD hub state machine are properly synchronized, `TMS` should be held high for at least five clock cycles after any intentional reset of either the TAP controller or the system logic. Both the JTAG TAP controller and the sld_hub controller are guaranteed to be in the Test Logic Reset state after five clock cycles of `TMS` held high.

Instantiating the Virtual JTAG IP Core

To create the Virtual JTAG IP core in an Intel Quartus Prime design requires the following system and software requirements:

- The Intel Quartus Prime software
- An Intel download cable, such as an Intel FPGA Download Cable cable

The download cable is required to communicate with the Virtual JTAG IP core from a host running the `quartus_stp` executable.

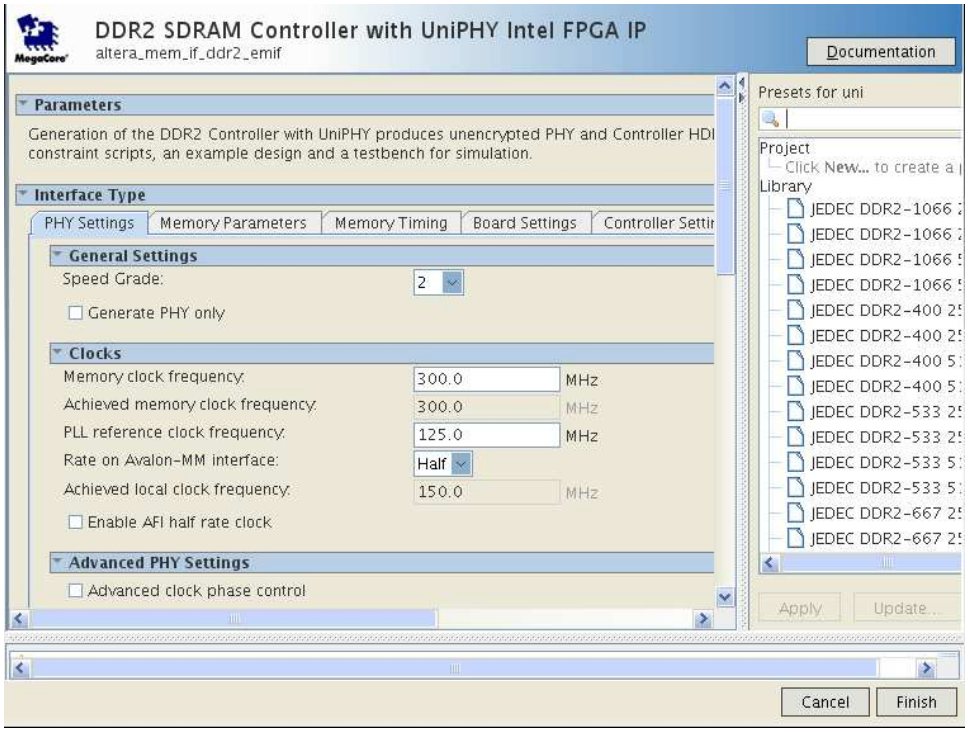
IP Catalog and Parameter Editor

The IP Catalog displays the IP cores available for your project, including Intel FPGA IP and other IP that you add to the IP Catalog search path.. Use the following features of the IP Catalog to locate and customize an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**. If you have no project open, select the **Device Family** in IP Catalog.
- Type in the Search field to locate any full or partial IP core name in IP Catalog.
- Right-click an IP core name in IP Catalog to display details about supported devices, to open the IP core's installation folder, and for links to IP documentation.
- Click **Search for Partner IP** to access partner IP information on the web.

The parameter editor generates a top-level Quartus IP file (`.qip`) for an IP variation in Intel Quartus Prime Standard Edition projects. These files represent the IP variation in the project, and store parameterization information.

Figure 13. IP Parameter Editor (Intel Quartus Prime Standard Edition)



The Parameter Editor

The parameter editor helps you to configure IP core ports, parameters, and output file generation options. The basic parameter editor controls include the following:

- Use the **Presets** window to apply preset parameter values for specific applications (for select cores).
- Use the **Details** window to view port and parameter descriptions, and click links to documentation.
- Click **Generate ➤ Generate Testbench System** to generate a testbench system (for select cores).
- Click **Generate ➤ Generate Example Design** to generate an example design (for select cores).

The IP Catalog is also available in Platform Designer (**View ➤ IP Catalog**). The Platform Designer IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in the Intel Quartus Prime IP Catalog. Refer to *Creating a System with Platform Designer* or *Creating a System with Platform Designer (Standard)* for information on use of IP in Platform Designer (Standard) and Platform Designer, respectively.

Related Information

- Creating a System with Platform Designer
- Creating a System with Platform Designer (Standard) (Standard)