



Vivado Design Suite User Guide: Programming and Debugging (UG908)

Introduction

- Navigating Content by Design Process
- Getting Started
- Debug Terminology

Vivado Lab Edition

- Installation
- Using the Vivado Lab Edition
- Vivado Lab Edition Project
- Programming Features
- Debug Features

Generating the Bitstream or Device Image

- Changing the Bitstream File Format Settings
- Changing the Device Image (.pdi) File Format Settings
- Changing Device Configuration Bitstream Settings

Programming the Device

- Opening the Hardware Manager
- Opening Hardware Target Connections
- Connecting to a Hardware Target Using hw_server
- Opening a New Hardware Target
- Troubleshooting a Hardware Target
- Associating a Programming File with the Hardware Device
- Programming the Hardware Device
- Closing the Hardware Target
- Closing a Connection to the Hardware Server
- Reconnecting to a Target Device with a Lower JTAG Clock Frequency
- Connecting to a Server with More Than 32 Devices in a JTAG Chain
- Changing the Default SmartLynq Ports

Remote Debugging in Vivado

- Using Vivado Hardware Server to Debug Over Ethernet
- Xilinx Virtual Cable (XVC)

Programming Configuration Memory Devices

- Changing Device Image Properties
- Creating a Configuration Memory File (for FPGA Devices)
- Creating a Configuration Memory File for SPI Dual Quad (x8) Devices (for FPGA Devices)
- Creating an Initialization PDI (for Versal Devices)
- Connect to the Hardware Target in Vivado
- Adding a Configuration Memory Device
- Programming a Configuration Memory Device
- Programming a Configuration Memory Device (Versal Devices)
- Booting the FPGA Device
- Configuration Failures in Master Mode

Advanced Programming Features

- Readback and Verify
- Generating Encrypted and Authenticated Files for 7 Series Devices
- Generating Encrypted and Authenticated Files for UltraScale and UltraScale+
- Programming the AES Key for 7 Series Devices

Programming the AES Key for UltraScale and UltraScale+ Devices
eFUSE Register Access and Programming
Cable Support for eFUSE Programming
eFUSE Register Access and Programming for 7 Series Devices
eFUSE Register Access and Programming for UltraScale and UltraScale+ Devices
eFUSE NKZ File
System Monitor

Standard Test and Programming Language (STAPL) Programming

Creating an STAPL Target
Adding Devices to a STAPL Target
Operations on the STAPL Chain
Writing STAPL Files
Executing STAPL Files

Serial Vector Format (SVF) File Programming

Creating an SVF Target
Adding Devices to an SVF Target
Adding Configuration Memory Parts to Xilinx Devices
Operations on the SVF Chain
Writing SVF Files
Executing SVF Files

Debugging the Design

RTL-Level Design Simulation
Post-Implemented Design Simulation
In-System Logic Design Debugging
In-System Serial I/O Design Debugging

In-System Logic Design Debugging Flows

Probing the Design for In-System Debugging
Versal In-System Debugging
Using the Netlist Insertion Debug Probing Flow
HDL Instantiation Debug Probing Flow Overview
Using the HDL Instantiation Debug Probing Flow
Debug Flow in IP Integrator
Implementing the Design Containing the Debug Cores
ILA Core and Timing Considerations
Debug Cores Clocking Guidelines
Debug Hub Insertion Guidelines
Adding Vivado Debug Cores to a Dynamic Function eXchange Design

Debugging Logic Designs in Hardware

Using Vivado Logic Analyzer to Debug the Design
Connecting to the Hardware Target and Programming the Device
Vivado Hardware Manager Dashboards
Setting Up the ILA Core to Take a Measurement
Writing ILA Probes Information
Reading ILA Probes Information
Viewing Captured Data from the ILA Core in the Waveform Viewer
Using Waveform ILA Trigger and Export Features
Saving and Restoring Captured Data from the ILA Core

- Enumeration of Probe Values
- Debugging AXI Interfaces in the Hardware Manager
- Setting Up the VIO Core to Take a Measurement
- Viewing the VIO Core Status
- Interacting with VIO Core Output Probes
- Hardware System Communication Using the JTAG-to-AXI Master Debug Core
- Using Vivado Logic Analyzer in a Lab Environment
- Description of Hardware Manager Tcl Objects and Commands
- Using Tcl Commands to Interact with a JTAG-to-AXI Master Core
- Using Tcl Commands to Take an ILA Measurement
- Trigger At Startup
- Memory Calibration Debug
- Debugging Dynamic Function eXchange (DFX) Designs in Vivado Hardware Manager
- High Bandwidth Memory (HBM) Monitor
- PCI Express Link Debug
- ChipScoPy API

Viewing ILA Probe Data in the Waveform Viewer

- ILA Data and Waveform Relationship
- Waveform Viewer Layout
- Waveform Viewer Operation
- Removing Probes from the Waveform
- Adding Probes to the Waveform
- Using Waveform ILA Trigger and Export Features
- Using the Zoom Features
- Waveform Settings
- Customizing the Configuration
- Renaming Objects
- Bus Radices
- Viewing Analog Waveforms
- Bus Plot Viewer
- Zoom Gestures

Debugging Designs Post Implementation

- Using Vivado ECO Flow to Replace Existing Debug Probes
- Replacing Debug Probes on a Placed and Routed Design Checkpoint
- Vivado ECO Tcl Flow to Replace Existing Debug Probes
- Incremental Compile with Debug Core (ILA) Modifications

Serial I/O Hardware Debugging Flows

- Serial I/O Hardware Debugging Flows

Versal Serial I/O Hardware Debugging Flows

Debugging the Serial I/O Design in Hardware

- Using Vivado Serial I/O Analyzer to Debug the Design
- Viewing Slicer Eye, Histogram, and Signal-to-Noise Ratio (GTM Transceivers Only)

Device Configuration Bitstream or PDI Settings

- 7 Series Bitstream Settings
- Artix, Virtex, and Kintex UltraScale+ Bitstream Settings
- UltraScale Bitstream Settings
- Zynq UltraScale+ MPSoC Bitstream Settings

Zynq 7000 Bitstream Settings

Versal Adaptive SoC Programmable Device Image (PDI) Settings

Trigger State Machine Language Description

States

Goto Action

Conditional Branching

Counters

Flags

Conditional Statements

Low Level SVF JTAG Commands

Header Data Register (HDR), Header Instruction Register (HIR)

TDR, TIR (Trailer Data Register, Trailer Instruction Register)

scan_ir_hw

scan_dr_hw

Multi Chain SVF Operation

JTAG Cables and Devices Supported by hw_server

Programming FTDI Devices for Vivado Hardware Manager Support

Configuration Memory Support

Artix 7 Configuration Memory Devices

Kintex 7 Configuration Memory Devices

Spartan 7 Configuration Memory Devices

Virtex 7 Configuration Memory Devices

Artix UltraScale+ Configuration Memory Devices

Kintex UltraScale Configuration Memory Devices

Kintex UltraScale+ Configuration Memory Devices

Virtex UltraScale Configuration Memory Devices

Virtex UltraScale+ Configuration Memory Devices

Zynq 7000 Configuration Memory Devices

Zynq UltraScale+ MPSoC Configuration Memory Devices

Zynq UltraScale+ RFSoC Configuration Memory Devices

Versal Configuration Memory Devices

Vendor Validated Flash Table

Command Line Options for hw_server

Standard hw_server Options

Advanced Options

Additional Resources and Legal Notices

Finding Additional Documentation

Support Resources

References

Training Resources

Revision History

Please Read: Important Legal Notices

Introduction

Navigating Content by Design Process

AMD Adaptive Computing documentation is organized around a set of standard design processes to help you find relevant content for your current development task. You can access the AMD Versal™ adaptive SoC design processes on the [Design Hubs](#) page. You can also use the [Design Flow Assistant](#) to better understand the design flows and find content that is specific to your intended design needs.

Hardware, IP, and Platform Development

Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the AMD Vivado™ timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:

- [Debugging the Design](#)
- [In-System Logic Design Debugging Flows](#)
- [Debugging Logic Designs in Hardware](#)
- [Viewing ILA Probe Data in the Waveform Viewer](#)
- [Debugging Designs Post Implementation](#)

Board System Design

Designing a PCB through schematics and board layout. Also involves power, thermal, and signal integrity considerations. Topics in this document that apply to this design process include:

- [Programming the Device](#)
- [Remote Debugging in Vivado](#)
- [Programming Configuration Memory Devices](#)
- [Advanced Programming Features](#)
- [Serial Vector Format \(SVF\) File Programming](#)
- [Serial I/O Hardware Debugging Flows](#)
- [Debugging the Serial I/O Design in Hardware](#)
- [Device Configuration Bitstream or PDI Settings](#)
- [Low Level SVF JTAG Commands](#)
- [JTAG Cables and Devices Supported by hw_server](#)
- [Configuration Memory Support](#)

Getting Started

After successfully implementing your design, the next step is to run it in hardware by programming the FPGA or adaptive SoC and debugging the design in-system. All of the necessary commands to perform programming of FPGAs and in-system debugging of the design are in the Program and

Debug section of the Flow Navigator in the AMD Vivado™ Integrated Design Environment (IDE) (see the following figure).

Figure: Program and Debug Section of the Flow Navigator Panel



Debug Terminology

ILA

The Integrated Logic Analyzer (ILA) feature allows you to perform in-system debugging of post-implemented designs on an FPGA, SoC, or AMD Versal™ device. This feature should be used when there is a need to monitor signals in the design. You can also use this feature to trigger on hardware events and capture data at system speeds.

The ILA core can be instantiated in your RTL code or inserted post synthesis in the Vivado design flow. Detailed documentation on the ILA core IP can be found in the *Integrated Logic Analyzer LogiCORE IP Product Guide (PG172)*.

Related Information

[In-System Logic Design Debugging Flows](#)

[Debugging Logic Designs in Hardware](#)

VIO

The Virtual Input/Output (VIO) debug feature can both monitor and drive internal FPGA, SoC, or Versal adaptive SoC signals in real time. In the absence of physical access to the target hardware, you can use this debug feature to drive and monitor signals that are present on the real hardware.

This debug core needs to be instantiated in the RTL code, hence you need to know up-front, what nets to drive. The IP catalog lists this core under the Debug category. Detailed documentation on the VIO core IP can be found in the *Virtual Input/Output LogiCORE IP Product Guide* ([PG159](#)).

Related Information

[Debugging Logic Designs in Hardware](#)

IBERT

The Integrated Bit Error Ratio Tester (IBERT) Serial Analyzer design enables in-system serial I/O validation and debug. This allows you to measure and optimize your high-speed serial I/O links in your FPGA-based system. AMD recommends using the IBERT Serial Analyzer design when you are interested in addressing a range of in-system debug and validation problems from simple clocking and connectivity issues to complex margin analysis and channel optimization issues.

AMD recommends using the IBERT Serial Analyzer design when you are interested in measuring the quality of a signal after a receiver equalization has been applied to the received signal. This ensures that you are measuring at the optimal point in the TX-to-RX channel thereby ensuring real and accurate data. Users can access this design by selecting, configuring, and generating the IBERT core from the IP catalog and selecting the Open Example Design feature of this core. See [Serial I/O Hardware Debugging Flows](#) and [Debugging the Serial I/O Design in Hardware](#) for more details on the IBERT core and its usage methodology in the Vivado Design Suite.

Related Information

[Debugging the Serial I/O Design in Hardware](#)

[Serial I/O Hardware Debugging Flows](#)

JTAG-to-AXI Master

 **Note:** The JTAG-to-AXI Master is not supported on Versal adaptive SoC devices as the built-in CIPS AXI Master interfaces can be used in combination with the Debug Packet Controller (DPC) to generate AXI transactions without additional IP.

The JTAG-to-AXI Master debug feature is used to generate AXI transactions that interact with various AXI4 and AXI4-Lite slave cores in a system that is running in hardware. AMD recommends that you use this core to generate AXI transactions and debug/drive AXI signals internal to an FPGA at runtime. This core can be used in designs without processors as well.

The IP catalog lists this core under the Debug category. Debugging Logic Designs in Hardware of this guide has more details about the JTAG-to-AXI Master core and its usage methodology in the Vivado Design Suite. Detailed documentation on the JTAG-to-AXI IP core can be found in the *JTAG to AXI Master LogiCORE IP Product Guide* ([PG174](#)).

Related Information

[Debugging Logic Designs in Hardware](#)

Debug Hub

On 7 series and AMD UltraScale™ architectures, the Vivado Debug Hub core provides an interface between the JTAG Boundary Scan (BSCAN) interface of the FPGA and the Vivado Debug cores including the following types of cores:

- Integrated Logic Analyzer (ILA)
 - Virtual Input/Output (VIO)
 - Integrated Bit Error Ratio Tester (IBERT)
 - JTAG-to-AXI
 - Memory IP
-

!! Important: The Vivado Debug Hub core *cannot* be instantiated into the design. It is inserted by Vivado during the opt_design stage.

AXI4 Debug Hub

On AMD Versal™ adaptive SoC architectures the AXI4 Debug Hub is an IP core that provides an interface between the AXI4 Master interface of the CIPS and the AXI4-Stream interface on the Vivado Hardware Debug cores including the following:

- Integrated Logic Analyzer (ILA)
 - Virtual Input/Output (VIO)
 - Soft Memory IP
-

 **Note:** On Versal devices, the AXI4 Debug Hub can be manually instantiated as an IP or inserted automatically during opt_design, as with previous architectures.

System ILA

The System Integrated Logic Analyzer (System ILA) IP core is a logic analyzer that allows you to perform in-system debugging of post-implemented designs on an FPGA device. Use this IP when you need to monitor interfaces and signals in the IP integrator Block Design. You can also use this feature to trigger interface and signal related hardware events and capture data at system speeds. This ensures the intuitive presentation of interface events in the Hardware Manager when debugging the design on an FPGA or adaptive SoC. This IP offers AXI interface debug and monitoring capability along with AXI4-MM and AXI4-Stream protocol checking.

Because the System ILA core is synchronous to the design being monitored, all design clock constraints that are applied to your design are also applied to the components of the System ILA core. Detailed documentation on the System ILA core IP can be found in the *System Integrated Logic Analyzer LogiCORE IP Product Guide* ([PG261](#)).

 **Note:** On AMD Versal™ devices the System ILA functionality is available using the Versal ILA core.

Debug Bridge

 **Note:** The Debug Bridge IP is not supported on Versal architectures.

The Debug Bridge IP core is a controller that provides multiple options to communicate with the debug cores in the design.

The primary use case for a Debug Bridge is to use an Xilinx Virtual Cable (XVC) to remotely debug designs through Ethernet or other interfaces without the need for a JTAG cable.

The other common use case is for debugging Dynamic Function eXchange and Tandem PCIe with Field Updates designs. For more information on the Tandem PCIe with Field Updates flow and

Debug Bridge refer to *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG213](#)).

The Debug Bridge can also be used with the PCIe® core in systems where JTAG is not the preferred communication and debug mechanism. For more information on the using the XVC flow with the PCIe core and Debug Bridge refer to *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG213](#)).

Detailed documentation on the Debug Bridge core IP can be found in the *Debug Bridge LogiCORE IP Product Guide* ([PG245](#)).

In-System IBERT

 **Note:** In-System IBERT is supported on UltraScale and UltraScale+ only.

The In-System IBERT IP enables you to perform 2-D eye-scans of UltraScale and UltraScale+ transceivers in your design, using the Vivado Serial IO Analyzer. The IP uses data from the design to plot the eye-scan of the transceivers in real time while they interact with the rest of the system. This IP can be integrated with user logic in the design or AMD transceiver-based IPs (for example GT Wizard, or Aurora, etc.).

Detailed documentation on the In-System IBERT IP can be found in the *In-System IBERT LogiCORE IP Product Guide* ([PG246](#)).

IBERT GTR

IBERT UltraScale+ GTR can be used to evaluate and monitor GTR transceivers in Zynq UltraScale+ MPSoCs. With this feature, you can accomplish the following tasks:

- Perform eye scans with user data
- Change GTR settings
- View link status
- Check the "lock" status of all PLLs used by all GTR lanes

However, IBERT GTR does not provide the following capabilities:

- Perform eye scans with raw PRBS data patterns
- Measure Bit Error Ratio (no bit or error counters)

 **Note:** This solution is software based, meaning that no IP or logic is required in the programmable logic of the device.

Vivado Lab Edition

AMD Vivado™ Lab Edition is a standalone installation of the full Vivado Design Suite with all the features and capabilities required to program and debug AMD devices after generating the device image. Typical usage is for programming and debugging in the lab environment where machines have a smaller number of resources in terms of disk space, memory, and connectivity. Vivado Lab Edition has a reduced product footprint of around 2.4 GB after installation and the install package size is 1 GB.

Installation

To install the Vivado Lab Edition, select Lab Edition from the Unified Installer.

Detailed installation, licensing, and release information is available in *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* ([UG973](#)).

Launching Vivado Lab Edition on Windows

To launch Vivado Lab Edition, select the following:

Start > All Programs > Xilinx Design Tools > Vivado Lab 2023.1

Launching the Vivado Lab Edition from the Command Line on Windows or Linux

Enter the following command at the command prompt:

```
vivado_lab
```

★ Tip: To run `vivado_lab` at the command prompt, set up your environment using the following script:

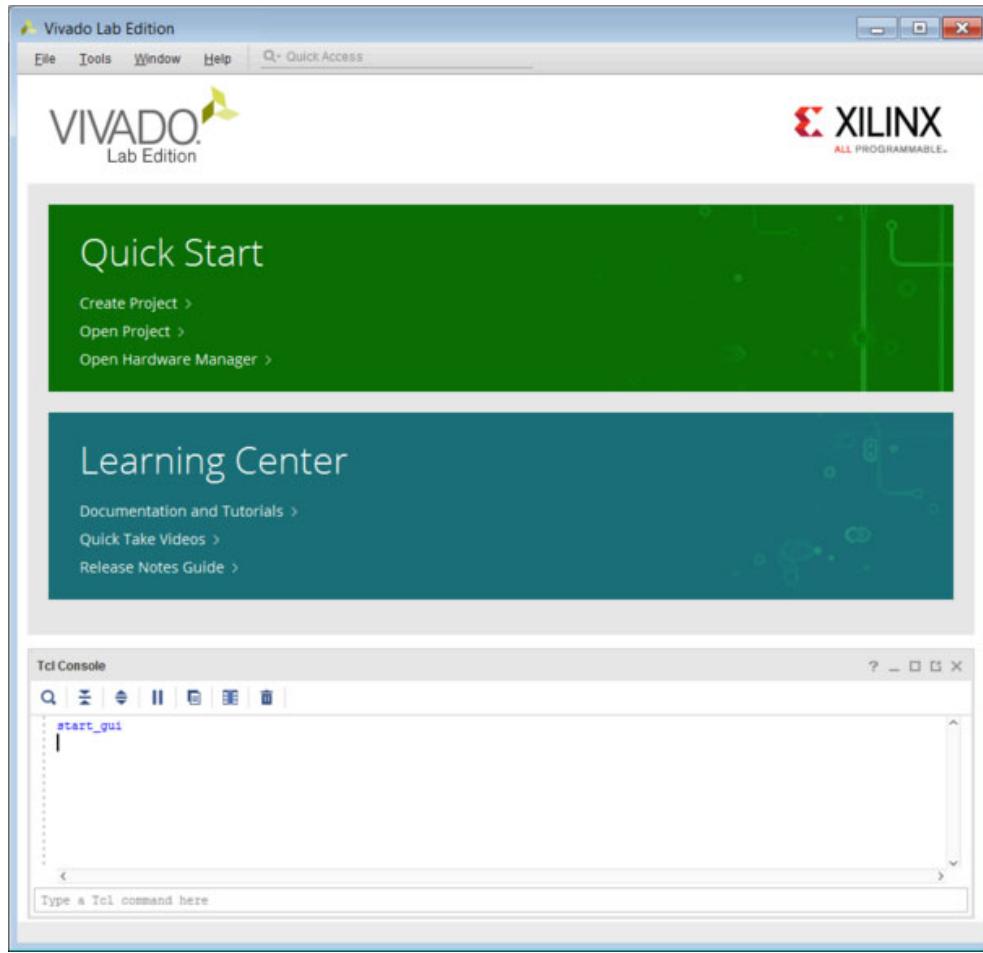
```
C:\Xilinx\Vivado_Lab\2023.x\settings64.(bat|sh)
```

You can open the Vivado Lab Edition from any directory. However, AMD recommends running it from a writable project directory, because the Vivado Lab Edition log and journal files are written to the launch directory. When running from a command prompt, launch the Vivado IDE from the project directory, or use the `vivado_lab -log` and `journal` options to specify a location. When using a Windows shortcut, you must modify the Start in folder, which is a property of the shortcut. Failure to launch from a writable project directory results in warnings and unpredictable behavior from the tool.

Using the Vivado Lab Edition

When you launch the Vivado Lab Edition, the Getting Started page (see the following figure) displays and provides you with different options to help you begin working with the Vivado Lab Edition.

Figure: Vivado Lab Edition Welcome Screen



Starting with a Project

To program or debug your design, you can create or open a project, and connect to a target server and device. The Quick Start section of the Getting Started Page provides links for easy access to the following tasks:

- Create a project.
- Open existing projects

Note: You can also open recently accessed projects from the Recent Projects list.

Opening the Hardware Manager

You can open the AMD Vivado™ Design Suite Hardware Manager to download your design bitstream to a device. Use the Vivado logic analyzer and Vivado serial I/O analyzer features of the Hardware Manager to debug your design. For example, you can add ILA, VIO, and JTAG-to-AXI cores to your design for debugging in the Vivado logic analyzer or use the IBERT example design from the AMD IP catalog to test and configure the GTs in your design with the Vivado serial I/O analyzer.

Reviewing Documentation and Videos

From the Getting Started page, you can use the Xilinx Documentation Navigator to access documentation, including user guides, tutorials, videos, and the release notes.

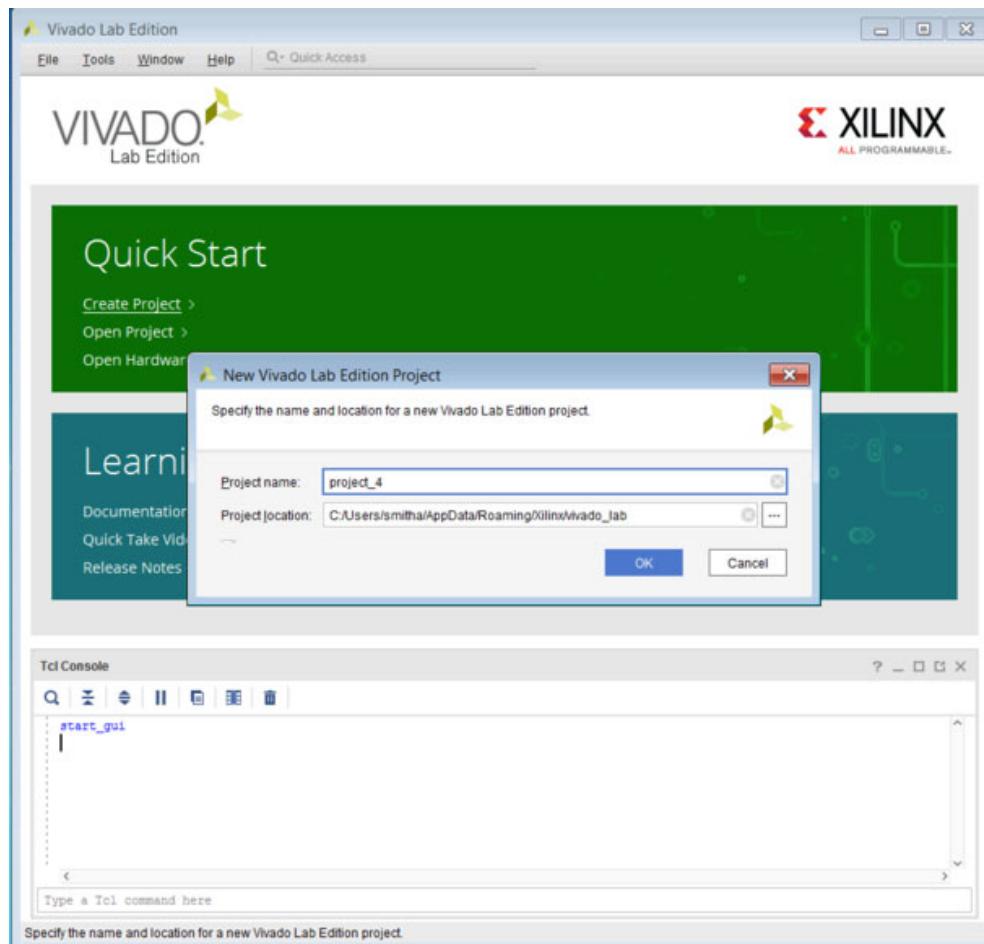
Vivado Lab Edition Project

Vivado Lab Edition allows users in the lab to create projects. All the relevant programming and runtime debug preferences and settings are stored in the project. When the project is reopened, the settings and preferences are restored back into the tool. A Vivado Lab Edition project can be created in both the Vivado Lab Edition tool and in Vivado Design Suite.

Create a New Project

To create a new project in Vivado Lab Edition, click the Create New Project icon as shown in the following figure. Enter the project name and location in the New Vivado Lab Edition Project dialog box. When you create a new project, Vivado Lab Edition creates a project file. The project file has the same name as the project name entered in the New Vivado Lab Edition Project dialog box with the .lpr extension. See the following figure.

Figure: Vivado Lab Edition Creating a New Project



Creating Projects Using Tcl Commands

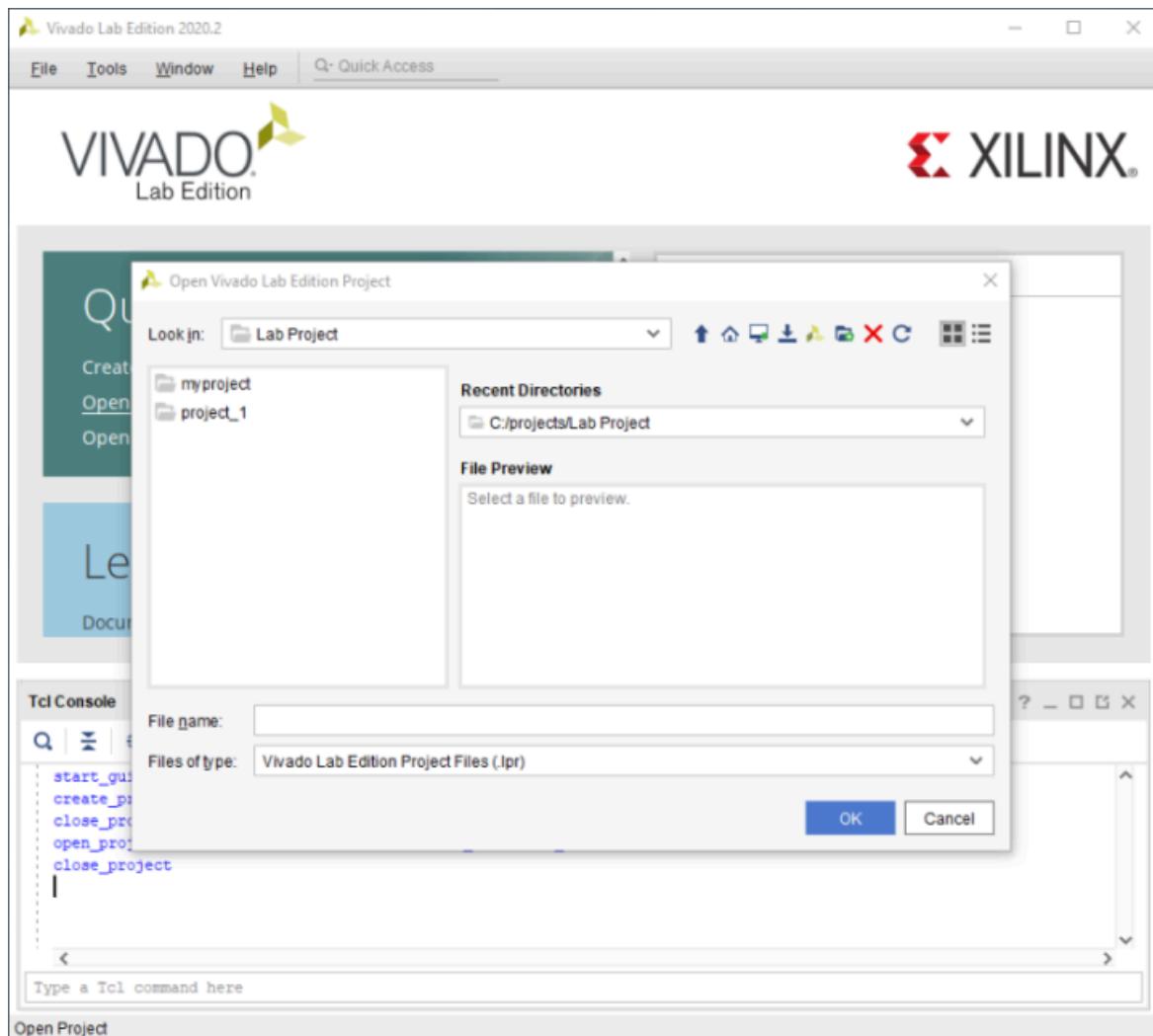
You can also create a project using Tcl commands. Enter the following command in the Tcl Console of Vivado Lab Edition or source them from a Tcl file.

```
create_project project_1 C:/Lab_edition/project_1
```

Opening the Project

To open existing projects, click the open project icon as shown in the following figure or double-click a project in the Recent Projects list. This opens a browser that enables you to open any Vivado Lab Edition project file (.lpr extension). By default, the last ten previously opened projects are listed in the Recent Projects list. To change this number, click Tools > Settings and update the Project options. Vivado Lab Edition checks to ensure that the project data is available before displaying the projects.

Figure: Vivado Lab Edition Open Project Dialog



Opening Projects Using Tcl Commands

You can also open a project using Tcl commands. Enter the following command in the Tcl Console of the Vivado Lab Edition or source them from a .tcl file.

```
open_project C:/Lab_edition/project_1/project_1.lpr
```

Using an Existing Device Image and Debug Probes Files in Vivado Lab Edition

You can use the existing device image (.bit or .pdi) and .ltx file from a previous implementation run in the lab machine where Vivado Lab Edition is installed.

Typical flow would entail the following:

1. Create a new Vivado Lab Edition project.
2. Connect to the board.
3. Specify the .bit or .pdi file and .ltx file for the project.
4. You can either manually copy these files or point to them on a network drive.
5. Program the device.
6. Debug the design in your hardware.
7. Changes are continuously saved to the project.
8. User preferences, runtime manager debug dashboard, and window settings are continuously saved to the project.
9. User preferences, runtime manager debug dashboard, and window settings are restored at project reopen.

Using an Existing .lpr Project from Vivado Design Suite Edition

Vivado Design Suite creates an .lpr file at project startup and populates this file with appropriate details when you use the Hardware Manager to program and/or debug the design in the project. This file is located in the project_name.hw directory and is named project_name.lpr. This project file can be opened in the Vivado Lab Edition.

Typical flow would entail:

1. Click the Open Project icon on the Vivado Lab Edition start page.
2. Traverse to the project_name.hw directory, which is located inside the Vivado IDE project directory.
3. Select the .lpr project file inside the project_name.hw directory and click OK.
4. Connect to your hardware.
5. Program and debug with the correct device image file and .ltx file from the appropriate Vivado run directory.
6. User preferences, runtime manager debug dashboard, and window settings are restored at project open.

Programming Features

After the project is open and the Hardware Manager is connected with a target device, you can use all the programming features that were available in the Vivado Design Suite from the Vivado Lab Edition. All the programming related Tcl commands are supported in Vivado Lab Edition. For more details on the programming features available refer to Programming Configuration Memory Devices.

Related Information

[Programming Configuration Memory Devices](#)

Debug Features

After you open the project and connect the Hardware Manager with a target device, you can use all the debug features that were available in the Vivado Design Suite from the Vivado Lab Edition. All the debug related Tcl commands are supported in Vivado Lab Edition. For more details on the debug features available refer to Debugging Logic Designs in Hardware of this user guide.

Related Information

[Debugging Logic Designs in Hardware](#)

Generating the Bitstream or Device Image

Before generating a bitstream or device image, it is important to review the settings to make sure they are correct for your design.

There are two types of bitstream and device image settings in AMD Vivado™ IDE:

1. Bitstream or Device Image file format settings.
2. Device configuration settings.

Select Settings > Bitstream in the Vivado Flow Navigator or Flow > Settings > Bitstream Settings menu selection to open the Bitstream Settings popup window (see the following figure). Once the settings are correct, the bitstream data file can be generated using the `write_bistream` Tcl command or by using the Generate Bitstream button in the Vivado flow navigator.

If an AMD Versal™ device is targeted, a programmable device image (.pdi) is generated instead of a bitstream. The procedure to change the device image settings is similar to previous architectures however, the menu options, Tcl commands, and available settings differ. To access the device image settings, select Settings > Generate Device Image in the Vivado Flow Navigator or Flow > Settings > Generate Device Image Settings... menu selection to open the Device Image section in the Settings popup window (see the following figure). The device image data file can be generated using the `write_device_image` Tcl command or by using the Write Device Image button in the Vivado flow navigator. For more details about the PDI format, refer to the *Bootgen User Guide* ([UG1283](#)).

Figure: Bitstream Settings Panel

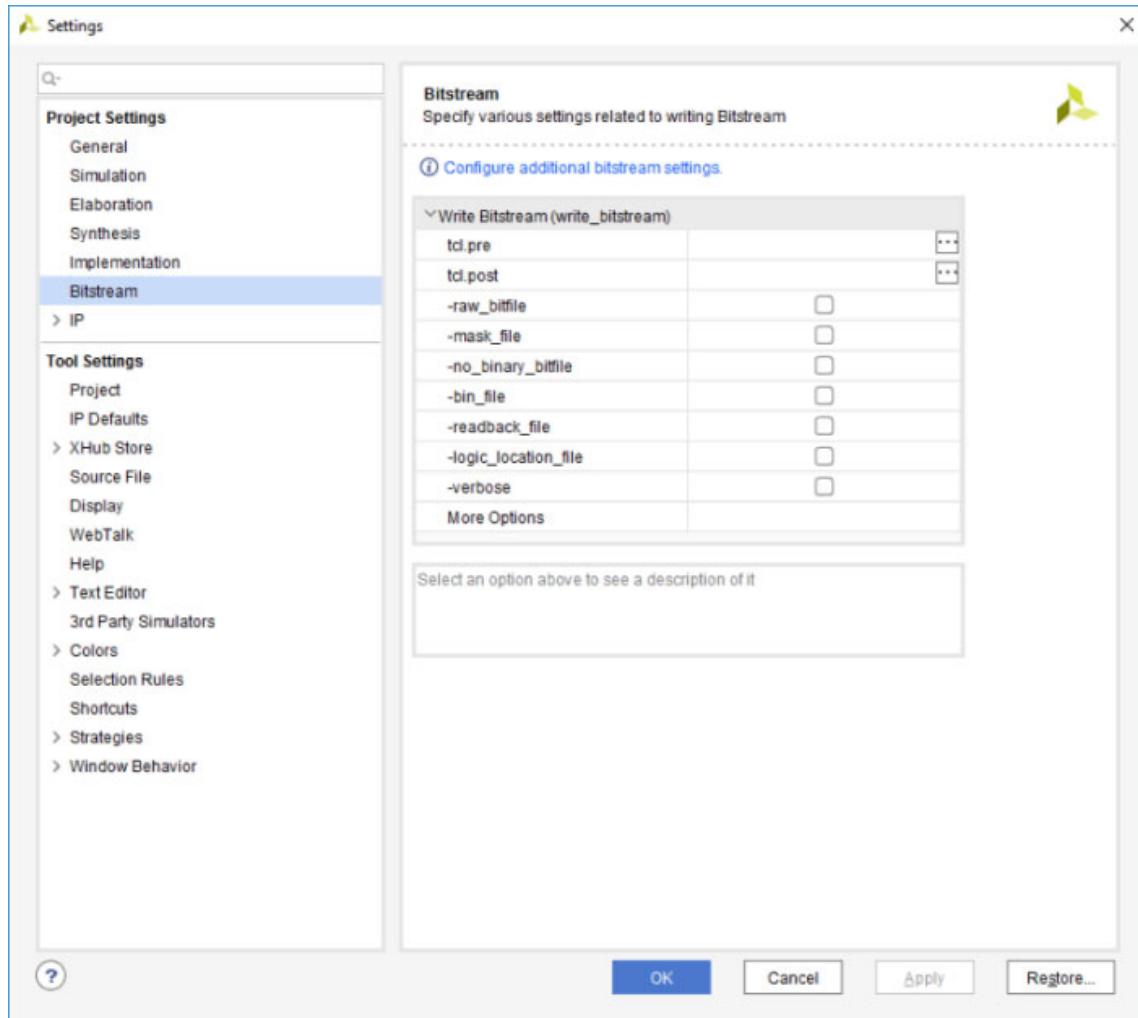
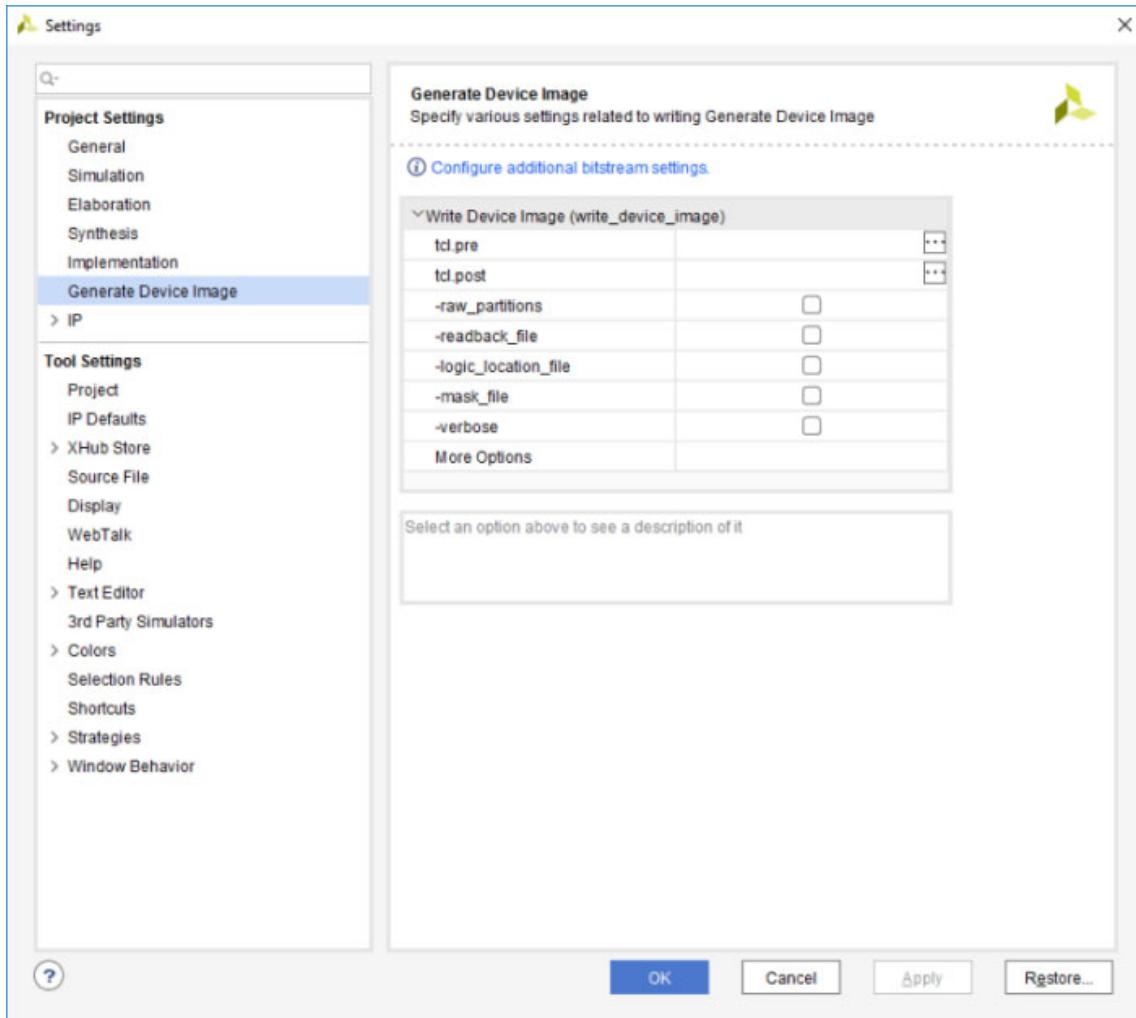


Figure: Generate Device Image Settings Panel



Changing the Bitstream File Format Settings

By default, the `write_bitstream` Tcl command generates a binary bitstream (.bit) file only. You can optionally change the file formats written out by the `write_bitstream` Tcl command by using the following command switches:

- `-raw_bitfile`: (Optional) This switch causes `write_bitstream` to write a raw bit file (.rbt), which contains the same information as the binary bitstream file, but is in ASCII format. The output file is named <filename>.rbt.
- `-mask_file`: (Optional) Write a mask file (.msk), which has mask data where the configuration data is in the bitstream file. This file determines which bits in the bitstream should be compared to readback data for verification purposes. If a mask bit is 0, that bit should be verified against the bitstream data. If a mask bit is 1, that bit should not be verified. The output file is named <file>.msk.
- `-no_binary_bitfile`: (Optional) Do not write the binary bitstream file (.bit). Use this command when you want to generate the ASCII bitstream or mask file, or to generate a bitstream report, without generating the binary bitstream file.
- `-logic_location_file`: (Optional) Creates an ASCII logic location file (.ll) that shows the bitstream position of latches, flip-flops, LUTs, Block RAMs, and I/O block inputs and outputs. Bits are referenced by frame and bit number in the location file to help you observe the contents of FPGA registers.
- `-bin_file`: (Optional) Creates a binary file (.bin) containing only device programming data, without the header information found in the standard bitstream file (.bit).
- `-reference_bitfile <arg>`: (Optional) Read a reference bitstream file and output an incremental bitstream file containing only the differences from the specified reference file. This partial bitstream file can be used for incrementally programming an existing device with an updated design.

Changing the Device Image (.pdi) File Format Settings

By default, the `write_device_image` Tcl command generates a (.pdi) file only. You can optionally change the file formats written out by the `write_device_image` Tcl command by using the following command switches:

- `-force` (Optional): Overwrite existing file.
- `-verbose` (Optional): Print `write_device_image` options.
- `-raw_partitions` (Optional): Write raw CFI and NPI partition files (.rnpi and .rcdo)
- `-mask_file` (Optional): Write a mask file (.msk)
- `-logic_location_file` (Optional): Write logic location file (.ll)
- `-cell <arg>` (Optional): Create only partial device image for the named cell.
- `-no_pdi` Do not generate a pdi file. Stop after generating raw partitions files only.
- `-no_partial_pdifile` (Optional): Do not write partial pdi files for a Dynamic Function eXchange design.
- `-quiet` (Optional): Ignore command errors.
- `<file>` (Required): The name of the .pdi file to write.

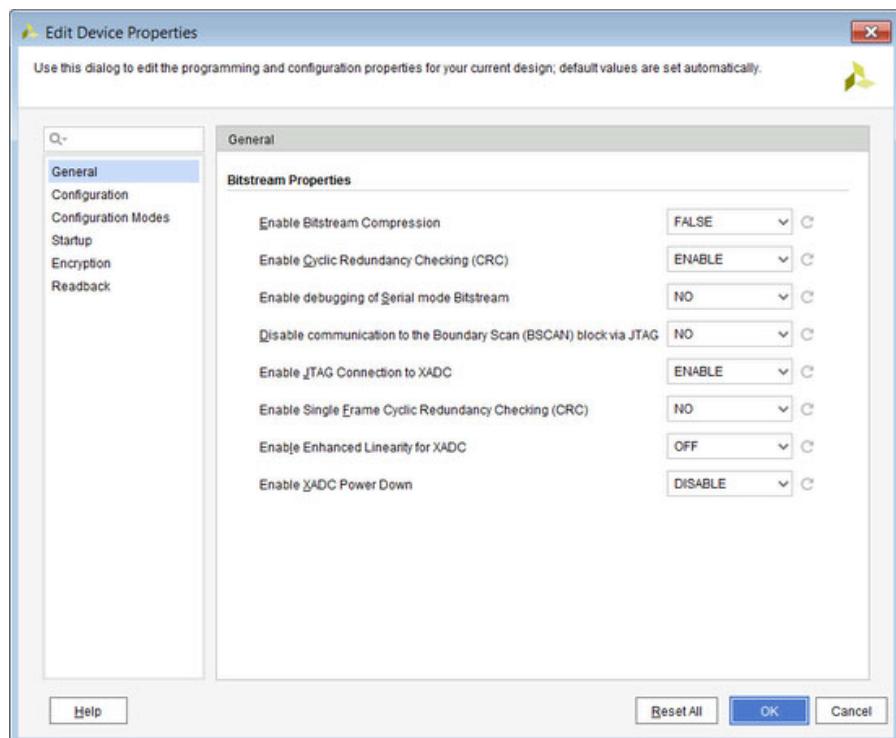
Changing Device Configuration Bitstream Settings

The most common configuration settings that you can change fall into the device configuration settings category. These settings are properties on the device model, and you change them by

using the Edit Device Properties dialog for the selected synthesized or implemented design netlist. The following steps describe how to set various bitstream properties using this method:

1. Select Tools > Edit Device Properties.
2. In the Edit Device Properties dialog, select one of the categories in the left-hand column (see the following figure).

★ Tip: You can type a property in the Search field. For example, type jtag into the Search text field to find and select properties related to JTAG programming.



3. Set the properties to the desired values, and click OK.
4. Select File > Constraints > Save to save the updated properties to the target XDC file.

You can also set the bitstream properties using the `set_property` command in an XDC file. For instance, here is an example on how to change the start-up DONE cycle property:

```
set_property BITSTREAM.STARTUP.DONE_CYCLE 4 [current_design]
```

Additional examples and templates are provided in the Vivado Templates. Device Configuration Bitstream Settings describes all of the device configuration settings.

!! Important: Edit only the Device Configuration Bitstream Settings relevant to the configuration mode being used. Leave the other settings at their default values.

Related Information

[Device Configuration Bitstream or PDI Settings](#)

Programming the Device

The next step after generating the device image is to download it into the target device. Vivado IDE has native in-system device programming capabilities built in to do this.

Vivado Design Suite and Vivado Lab Edition includes functionality that allows you to connect to hardware containing one or more AMD Device(s) to program and interact with those devices.

Connecting to hardware can be done from either the Vivado Lab Edition, or Vivado Design Suite graphical user interface or by using Tcl commands. In either case, the steps to connect to hardware and program the target device are the same:

1. Open the Hardware Manager.
2. Open a hardware target that is managed by a hardware server running on a host computer.
3. Associate the device image with the appropriate device.
4. Program or download the device image into the hardware device.

Opening the Hardware Manager

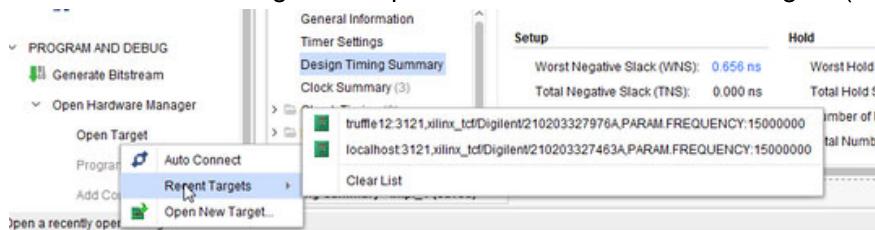
Opening the Hardware Manager is the first step in programming and/or debugging your design in hardware. To open the Hardware Manager, do one of the following:

- If you have a project open, click the Open Hardware Manager button in the Program and Debug section of the Vivado flow navigator.
- Select Flow > Open Hardware Manager.
- In the Tcl Console window, run the `open_hw_manager` command.

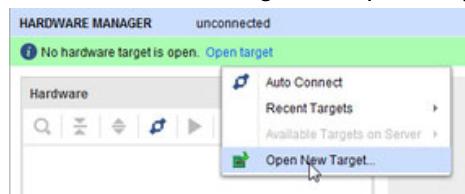
Opening Hardware Target Connections

The next step in opening a hardware target (for instance, a hardware board containing a JTAG chain of one or more FPGAs or adaptive SoCs) is connecting to the hardware server that is managing the connection to the hardware target. You can do this in one of the following three ways:

- Use the Open Target selection under Hardware Manager in the Program and Debug section of the Vivado Flow Navigator to open new or recent hardware targets (see the following figure).



- Use the Open Target > Recent targets or Open Target > Open New Target selection on the green user assistance banner across the top of the Hardware Manager window to open recent or new hardware targets, respectively (see the following figure).



- Use Tcl commands to open a connection to a hardware target.

★ Tip: Use the Auto Connect selection to automatically connect to a local hardware target.

Connecting to a Hardware Target Using hw_server

The `hw_server` is automatically started by Vivado when connecting to targets on the local machine. However, you can also start the `hw_server` manually on either local or remote machines. For instance, in a full Vivado installation on a Windows platform, at a cmd prompt run the following command:

```
C:\Xilinx\Vivado\<Vivado_version>\bin\hw_server.bat
```

If you are using a Hardware Server (Standalone) installation on a Windows platform, at a cmd prompt run the following command:

```
c:\Xilinx\HWSRVR\<Vivado_version>\bin\hw_server.bat
```

Follow the steps in the next section to open a connection to a new hardware target using this agent. For a list of compatible JTAG download cables and devices see [JTAG Cables and Devices Supported by hw_server](#).

For more information on using SmartLynq data cables, see the [SmartLynq Data Cable User Guide \(UG1258\)](#).

!! Important: If Vivado Hardware Manager is connected to the `hw_server`, and the `hw_server` is stopped, the Hardware Manager detects this condition automatically and disconnects from the server.

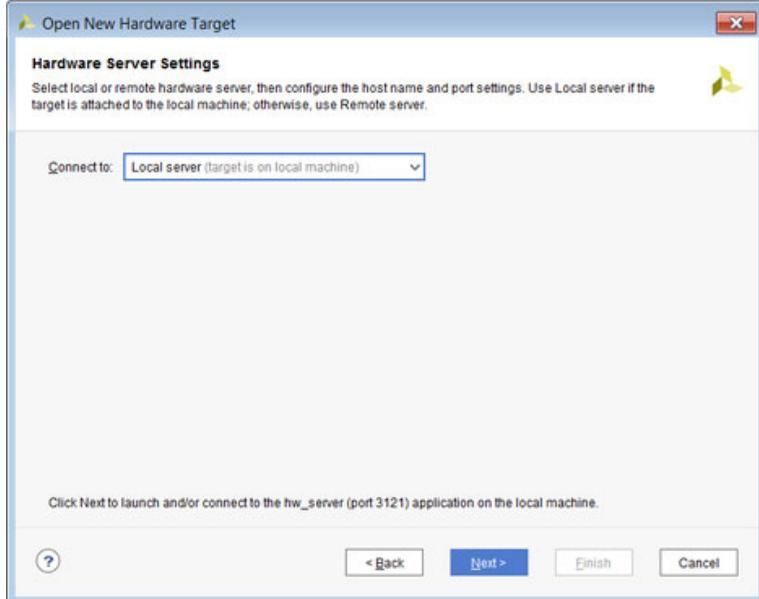
Opening a New Hardware Target

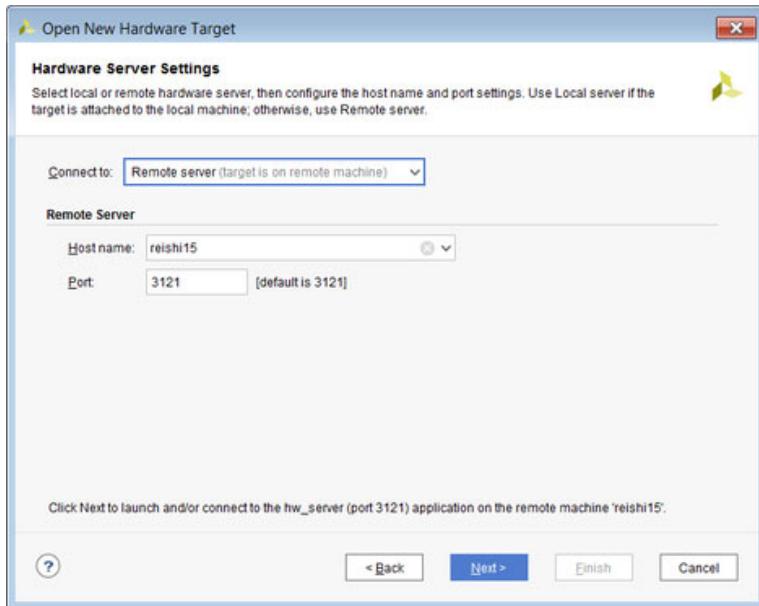
The Open New Hardware Target wizard provides an interactive way for you to connect to a hardware server and target. The wizard process has the following steps:

1. Select a local or remote server, depending on what machine your hardware target is connected to:
 - Local server: Use this setting if your hardware target is connected to the same machine on which you are running the Vivado Lab Edition or Vivado IDE (See the following figure). The Vivado software automatically starts the Vivado hardware server (hw_server) application on the local machine.
 - Remote server: Use this setting if your hardware target is connected to a different machine on which you are running the Vivado Lab Edition or Vivado IDE. Specify the hostname or IP address of the remote machine and the port number for the hardware server (hw_server) application running on that machine (see the following figures). Refer to Connecting to a Remote hw_server Running on a Lab Machine for more details on remote debugging.

!! Important: When using the remote server, you need to manually start the Vivado hardware server (hw_server) application of the same version of Vivado software that you use to connect to the hardware server.

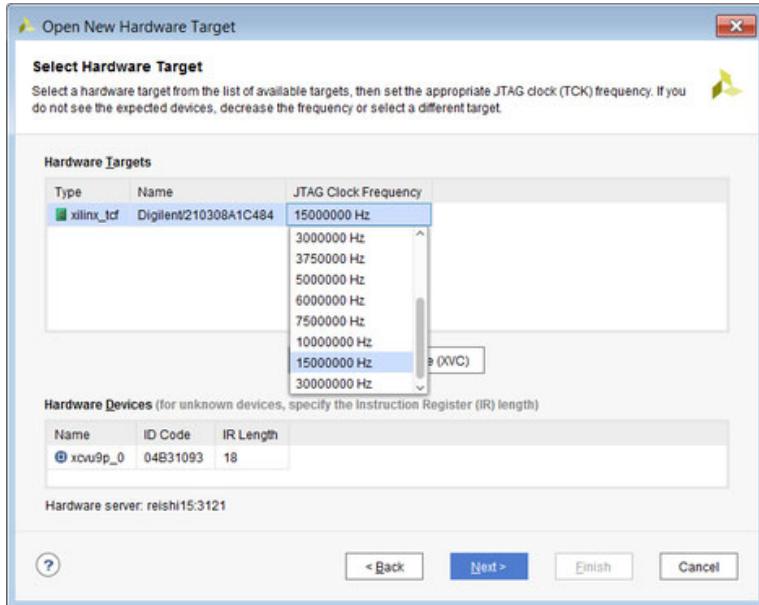
★ Tip: If you only want to connect to your lab machine remotely, you do not need to install the full Vivado design suite on that remote machine. Instead, you can install the lightweight Vivado Hardware Server (Standalone) tool on the remote machine.





2. Select the appropriate hardware target from the list of targets that are managed by the hardware server.

Note: When you select a target, you see the various hardware devices that are available on that hardware target.



!! Important: If there are third-party devices in the JTAG chain, use the instructions in [Answer Record 61312](#) to add IDCODE, IR Length, and name for the unknown devices.

Related Information

[Connecting to a Remote hw_server Running on a Lab Machine](#)

Troubleshooting a Hardware Target

You might run into issues when trying to connect to a hardware target. Here are some common issues and recommendations on how to resolve them:

- If you cannot correctly identify the hardware devices on your target, it might mean that your hardware is not capable of running at the default target frequency. You can adjust the frequency of the TCK pin of the hardware target or cable (see the previous figure).
-
- Note:** Each type of hardware target can have different properties. Refer to the documentation of each hardware target for more information about these properties.
-
- While the Vivado hardware server attempts to automatically determine the instruction register (IR) length of all devices in the JTAG chain, in some rare circumstances, it might not be able to correctly do so. You should check the IR length for each unknown device to ensure it is correct. If you need to specify the IR length, you can do so directly in the Hardware Devices table of the Open New Hardware Target wizard (see Opening a New Hardware Target).

Related Information

[Opening a New Hardware Target](#)

Opening a Recent Hardware Target

The Open New Hardware Target wizard is also what populates a list of previously connected hardware targets. Instead of connecting to a hardware target by going through the wizard, you can re-open a connection to a previously connected hardware target by selecting the Open recent target link in the Hardware Manager window and selecting one of the recently connected hardware server/target combinations in the list. You can also access this list of recently used targets through the Open Target selection under Hardware Manager in the Program and Debug section of the Vivado flow navigator.

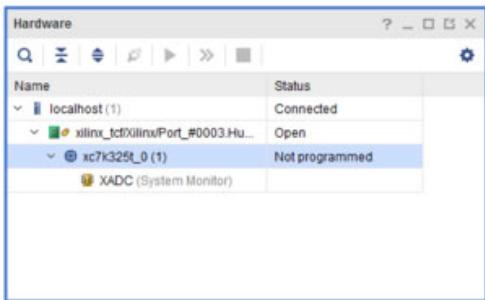
Opening a Hardware Target Using Tcl Commands

You can also use Tcl commands to connect to a hardware server/target combination. For instance, to connect to the digilent_plugin target (serial number 210203339395A) that is managed by the hw_server running on localhost 3121, use the following Tcl commands:

```
connect_hw_server -url localhost:3121
current_hw_target [get_hw_targets */xilinx_tcf/Digilent/210203339395A]
set_property PARAM.FREQUENCY 15000000 [get_hw_targets \
*/xilinx_tcf/Digilent/210203339395A]
open_hw_target
```

Once you finish opening a connection to a hardware target, the Hardware window is populated with the hardware server, hardware target, and various hardware devices for the open target (see the following figure).

Figure: Hardware View after Opening a Connection to the Hardware Target



Associating a Programming File with the Hardware Device

After connecting to the hardware target and before you program the device, you need to associate the bitstream data programming file with the device. Select the hardware device in the Hardware window and make sure the Programming file property in the Properties window is set to the appropriate programming file.

Note: As a convenience, Vivado IDE automatically uses the programming file for the current implemented design as the value for the Programming File property of the first matching device in the open hardware target. This feature is only available when using the Vivado IDE in project mode. When using the Vivado IDE in non-project mode, you need to set this property manually.

You can also use the `set_property` Tcl command to set the `PROGRAM.FILE` property of the hardware device:

```
set_property PROGRAM.FILE {C:/<path_to_programming_file>} [lindex  
[get_hw_devices] 0]
```

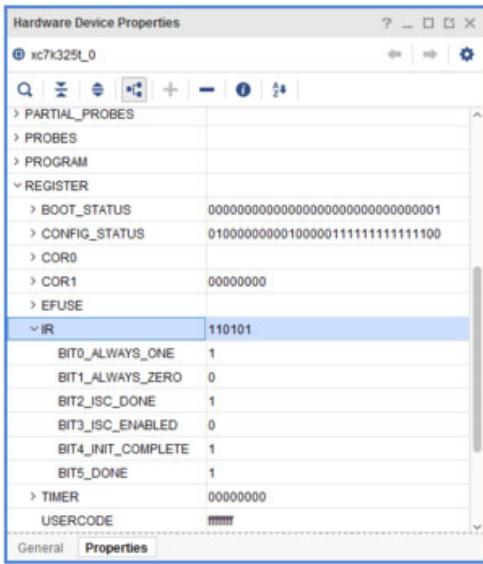
Programming the Hardware Device

Once the programming file is associated with the hardware device, you can program the hardware device by right-clicking on the device in the Hardware window and selecting the Program Device menu option. You can also use the `program_hw_device` Tcl command. For instance, to program the first device in the JTAG chain, use the following Tcl command:

```
program_hw_devices [lindex [get_hw_devices] 0]
```

Once the progress dialog has indicated that the programming is 100% complete, you can check that the hardware device has been programmed successfully by examining the DONE status in the Hardware Device Properties view.

Figure: Checking the DONE Status of a Device



You can also use the `get_property` Tcl command to check the DONE status. For instance, to check the DONE status of an AMD Kintex™ 7 device that is the first device in the JTAG chain, use the following Tcl command:

```
get_property REGISTER.IR.BIT5_DONE [lindex [get_hw_devices] 0]
```

On Versal devices, the command differs slightly because the DONE status register is different. Index 1 must be read as the first device returned by `get_hw_devices` and be the `arm_dap` in a single device use case.

```
get_property REGISTER.JTAG_STATUS.BIT[34]_DONE [lindex [get_hw_devices] 1]
```

If you use another means to program the hardware device (for instance, a flash device or external device programmer), you can also refresh the status of a hardware device by right-clicking the Refresh Device menu option or by running the `refresh_hw_device` Tcl command. This refreshes the various properties for the device, including but not limited to the DONE status.

!! Important: For non-Versal architectures, if your design contains debug cores, ensure that the JTAG clock is 2.5 times slower than the debug hub clock.

!! Important: User SCAN Chain: For non-Versal architectures, Vivado Programmer tries to detect debug cores on the user scan chain specified in the design by default. It does the detection by issuing a `JTAG_CHAIN 1` command to the device. If you have programmed a device with a design that does not have any debug cores or a debug core with a user scan chain of 2, 3, or 4, you see a warning.

To determine the user scan chain setting, for non-Versal architectures, open the implemented design and use:

```
get_property C_USER_SCAN_CHAIN [get_debug_cores dbg_hub]
```

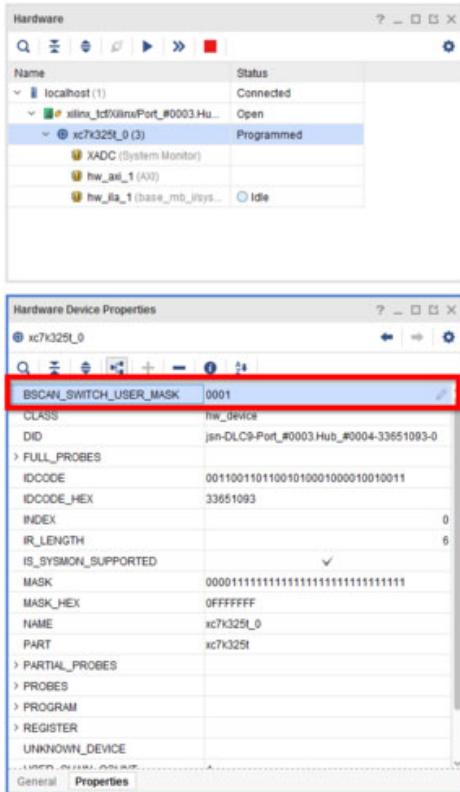
You can change the user scan chain used in the Vivado Hardware Manager.

Note: The `BSCAN_SWITCH_USER_MASK` is a bit mask value. See the following figure.

Alternatively, you can specify the user scan chain value as an option to `hw_server` start-up.

```
hw_server -e "set bscan-switch-user-mask <user-bit-mask>"
```

Figure: BSCAN Switch User Mask



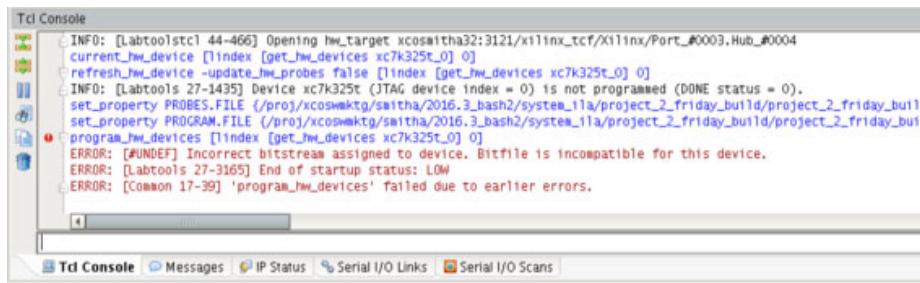
★ **Tip:** For designs prior to Vivado 2016.3 AMD recommends manually launching `hw_server` with -e "set xsdb-user-bscan <C_USER_SCAN_CHAIN scan_chain_number>" to detect the debug hub at User Scan Chain of 2 or 4.

Incorrect Bitstream Assignment Message

Vivado Hardware Manager generates an *incorrect bitstream assignment* message when:

- Attempting to program an image with a bitstream or programmable device image generated for a different FPGA or adaptive SoC.
- For example, the following error message appears when trying to program an XCKU115 with an XCVU190 bitstream.

Figure: Programming an XCKU115 with an XCVU190 Bitstream



The screenshot shows the Tcl Console window of the Vivado IDE. The console output is as follows:

```

Tcl Console
INFO: [Labtoolstc1 44-466] Opening hw_target xcosmaltha32:3121/xilinx_tcf/Xilinx/Port_#0003.Hub_#0004
current_hw_device [lindex [get_hw_devices xc7k325t_0] 0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices xc7k325t_0] 0]
INFO: [Labtools 27-1435] Device xc7k325t (JTAG device index = 0) is not programmed (DONE status = 0).
set_property PROBES.FILE ./proj/xcosmktg/smitha/2016.3_bash2/system_1la/project_2_friday_build/project_2_friday_build
set_property PROGRAM.FILE ./proj/xcosmktg/smitha/2016.3_bash2/system_1la/project_2_friday_build/project_2_friday_build
program_hw_devices [lindex [get_hw_devices xc7k325t_0] 0]
ERROR: [#UNDEF] Incorrect bitstream assigned to device. Bitfile is incompatible for this device.
ERROR: [Labtools 27-3165] End of startup status: LOW
ERROR: [Common 17-39] 'program_hw_devices' failed due to earlier errors.

```

The bottom of the window shows tabs for Tcl Console, Messages, IP Status, Serial I/O Links, and Serial I/O Scans.

The solution is to specify the correct bitstream for the FPGA or adaptive SoC being programmed.

Attempting to Program Configuration Memory Attached to an FPGA Device

To program configuration memory attached to an FPGA device, Vivado Hardware Manager first downloads a flash controller bitstream to the FPGA device. The Hardware Manager sends flash commands and data through the FPGA device's JTAG port to be processed by the controller, sending the processed flash commands/data to the configuration memory interface.

The controller bitstream downloaded by Hardware Manager is generated for the latest silicon revision of the FPGA device. For example, the configuration memory controller bitstream for the XCKU115 in 2016.3 was later generated for XCKU115-es2 silicon.

When programming configuration memory attached to this FPGA, if the user has an XCKU115-es1 device on the board, the error message shown in Attempting to Program an FPGA Device with a Bitstream Generated for a Different Silicon Revision of the FPGA appears. This is because Hardware Manager is attempting to download the -es2 flash controller bitstream into the -es1 device.

Closing the Hardware Target

You can close a hardware target by right-clicking on the hardware target in the Hardware window and selecting Close Target from the pop-up menu. You can also close the hardware target using a Tcl command. For instance, to close the xilinx_platformusb/USB21 target on the local host server, use the following Tcl command:

```
close_hw_target {localhost/xilinx_tcf/Digilent/210203339395A}
```

!! Important: If the board is powered off or cable disconnected, Vivado IDE closes the hardware target in the Hardware Manager. Any Vivado operation in the main Vivado thread is also canceled. If the board is powered back on or the cable is reconnected, the Vivado IDE attempts to re-open the hardware target in the Hardware Manager.

Closing a Connection to the Hardware Server

You can close a hardware server by right-clicking on the hardware server in the Hardware window and selecting Close Server from the popup menu. You can also close the hardware server using a Tcl command. For instance, to close the connection to the localhost server, use the following Tcl command:

```
disconnect_hw_server localhost
```

!! Important: If Vivado Hardware Manager is connected to the hw_server, and the hw_server is stopped, the Hardware Manager detects this condition automatically and disconnects from the server.

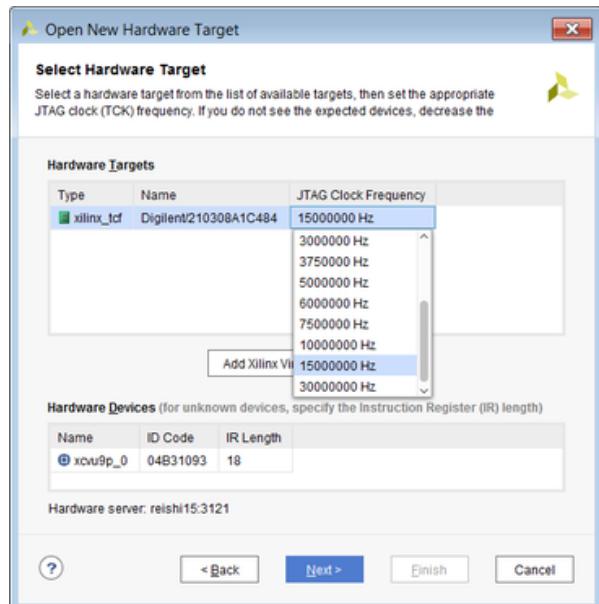
Reconnecting to a Target Device with a Lower JTAG Clock Frequency

The JTAG chain is as fast as the slowest device in the chain. Therefore, to lower the JTAG clock frequency, connect to a target device whose JTAG clock frequency is less than the default JTAG clock frequency.

You should attempt to open with a default JTAG clock frequency that is 15 MHz for the Digilent cable connection and 6 MHz for the USB cable connection. If it is not possible to connect at these speeds, AMD recommends that you lower the default JTAG clock frequency even further as described as follows.

To change the JTAG clock frequency, use the Open New Hardware Target wizard, from AMD Vivado™ Design Suite, as shown in the following figure.

Figure: Vivado Lower JTAG Frequency



Alternately, you can use the following sequence of Tcl commands:

```
open_hw_manager

connect_hw_server -url machinename:3121

current_hw_target [get_hw_targets */xilinx_tcf/Digilent/210203327962A]

set_property PARAM.FREQUENCY 250000 [get_hw_targets
*/xilinx_tcf/Digilent/210203327962A]
open_hw_target
```

Connecting to a Server with More Than 32 Devices in a JTAG Chain

It can connect to a server that has more than 32 devices in its JTAG chain in Vivado. You need to provide option `max-jtag-devices` at the startup of `hw_server` to enable the ability to detect more devices in a scan chain. The default value for this setting is 32.

 **Note:** Increasing this number slows down the device discovery process, which in turn can slow down cable access.

Specify the `max-jtag-devices` option at `hw_server` start-up as follows:

```
hw_server -e "set max-jtag-devices 64"
```

Usage

This option is used to start up the `hw_server` with the ability to enable ir lengths greater than 64 bits. The default value for this setting is 64. You can increase this value for devices in the JTAG chains whose ir lengths are wider (for example, 93).

 **Note:** Increasing this number slows the device discovery process, which can slow cable access. Therefore, you should only increase this value for systems with long ir lengths and device counts.

This is how you specify the option at `hw_server` start-up:

```
hw_server -e "set max-ir-length 93"
```

Init Option

You can also use the `--init=script.txt` option to load this setting through a file. To use the `init` option, create an initialization script as shown in the following example. In the script, specify the `set max-jtag-device` parameter.

```
# Sample script.txt
set max-ir-length 93
```

Start the hw_server as shown in the following example:

```
hw_server --init=script.txt
```

Changing the Default SmartLynq Ports

By default, the SmartLynq module uses the following ports. In some cases, you might want to change the port used. To update the port used, add the commands to the SmartLynq config.ini file and update using the procedure documented in the *SmartLynq Data Cable User Guide* ([UG1258](#)).

Table: Changing SmartLynq Default Ports

Default Port	Description	Add to config.ini to change
80	TCF over HTTP	set http-port <port>
3121	TCF	set tcf-port <port>
10200	Low level JTAG access over XVC	set xvc-port <port>
3000-3003	GNU Debugger ports (for Arm® / MicroBlaze™ processors)	set gdb-port <base port> Setting the base port sets the lowest port for the range of four ports.

Remote Debugging in Vivado

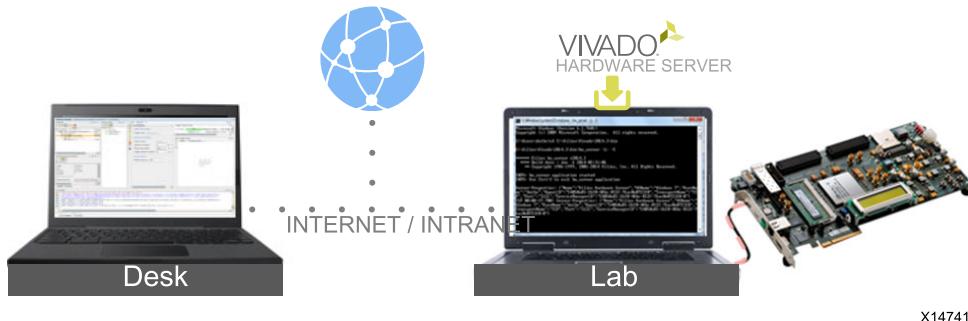
The need for remote debugging can arise in a variety of situations. It could be required in the prototyping phase of a product, where you might want to debug a design in the lab without physical access to the lab, or where you might want to share resources across your organization. Remote debugging could also be required to perform in-the-field debug to diagnose issues or extend product life cycle.

AMD provides multiple solutions to debug your design remotely. This can be done using the AMD Hardware Server product to connect to a remote computer in the lab. You could also implement the Xilinx Virtual Cable (XVC) protocol to connect to a network-connected board. Each of these solutions is explained in detail in the following sections.

Using Vivado Hardware Server to Debug Over Ethernet

You can connect to a remote lab machine using Vivado Hardware Server product. This is a small sized (<100 MB) standalone download available for install on the lab machine. This option requires intranet or internet access and can be used internally within your organization as well.

Figure: Debug via Internet/Intranet Using Hardware Server



X14741-062315

Xilinx Virtual Cable (XVC)

Vivado IDE supports the Xilinx Virtual Cable (XVC) protocol. XVC lets you access and debug an AMD device without using USB or parallel configuration cable. This capability helps facilitate Vivado IDE to debug for designs that:

- Have the FPGA in a hard-to-access location, where a "lab-PC" is not convenient.
- Do not have direct access to the device pins on the board - for example, if the JTAG pins are only accessible with a local microprocessor interface.

XVC is an internet-based (TCP/IP) protocol that acts like a JTAG cable. It has very basic cable commands. This allows XVC to debug a system over an intranet, or even the internet. With this capability you can save on costly or impractical travel and reduce the time it takes to debug a remote system.

Another common use of XVC is for shared systems that are not co-located with teams that need access to them. It can also be used when there are physical constraints to using the system, such as when the JTAG connector is not available or accessible. XVC implementation is programming language and platform independent.

Rather than using a dedicated JTAG header, an existing Ethernet connection can be used to create the appropriate JTAG commands from a processor to a target device. With the XVC v1.0 Protocol, Vivado can communicate the same JTAG commands over an Ethernet connection and still support all of the existing Vivado debug features.

!! Important: If the Vivado Debug Bridge IP is used for XVC, Vivado IDE does not support programming features. The assumption is that the device is programmed before using XVC to debug the design. The Debug Bridge IP is not compatible with Versal adaptive SoC.

Vivado Debug Bridge IP and Xilinx Virtual Cable (XVC) Flow

Note: Vivado Debug Bridge IP is not supported on AMD Versal™ devices.

The Vivado Debug Bridge IP core is a controller that provides multiple options to communicate with the debug cores in the design. This design can be a flat design or a Dynamic Function eXchange design. In addition, the Debug Bridge IP core can also be configured to take advantage of debugging designs using a JTAG cable or remotely through Ethernet, PCIe®, or other interfaces without the need for a JTAG cable.

Different modes in Debug Bridge IP facilitate the support of various use cases.

Debug Bridge in XVC Modes

There are five modes in the Debug Bridge that are used in Xilinx Virtual Cable (XVC) implementations.

From AXI to BSCAN

In this mode, the Debug Bridge receives XVC Commands via AXI4-Lite slave interface.

From JTAG to BSCAN

In this mode, the Debug Bridge receives XVC Commands via JTAG slave interface driven by user logic.

From PCIe to BSCAN

In this mode, the Debug Bridge receives XVC Commands via PCIe Extended Configuration slave interface.

From PCIe to JTAG

In this mode, the Debug Bridge receives XVC Commands via PCIe Extended Configuration interface. This Debug Bridge brings out the JTAG pins out of the FPGA through I/O pins. This mode is mainly used to debug design on another board over XVC.

From AXI to JTAG

In this mode, the Debug Bridge receives XVC commands via AXI4-Lite interface to send over the JTAG pins to a target device.

In all of these modes the Debug Bridge can further communicate with other debug cores/ Debug Bridge instances in the design via the Soft-BSCAN (Boundary Scan) interface. The Soft BSCAN master interface enables extension of the JTAG interface to internal USER defined scan chains/Debug Bridge instances.

Using Debug Bridge IP in Dynamic Function eXchange Designs

The Debug Bridge IP can be used in both flat and Dynamic Function eXchange designs. Following are the details on the Debug Bridge configurations used in the static or Reconfigurable Partition (RP) region of a Dynamic Function eXchange design. Multiple Debug Bridge instances are permitted in a partition depending on the design requirements.

BSCAN Primitive

This mode is used when a Debug Bridge containing a BSCAN primitive is required in the static region. The BSCAN master interface of this Debug Bridge can be connected to another Debug Bridge instance in the static and/or RP region(s) providing one or more communication pathways for debugging those regions.

From BSCAN to Debug Hub

In this mode, the Debug Bridge uses the BSCAN slave interface to communicate to Vivado Hardware Manager. It uses the Debug Hub interface to communicate with the design cores within the relevant static or RP region. You can also optionally add additional BSCAN Masters to the output of this Debug Bridge, which enables debugging other debug cores like MicroBlaze™ Debug Module (MDM) or other Debug Bridge instances.

 **Note:** The tool automatically connects the debug cores in an RP to the Debug Bridge if this is the only Debug Bridge instantiated in the partition.

From AXI to BSCAN

In this mode, the Debug Bridge receives XVC Commands via AXI4-Lite slave interface. This Debug Bridge can further communicate with other debug cores/ Debug Bridge instances in the design via the Soft-BSCAN (Boundary Scan) master interface. The Soft BSCAN interface enables extension of the JTAG interface to internal USER defined scan chains/Debug Bridge instances.

From JTAG to BSCAN

In this mode, the Debug Bridge receives XVC Commands via JTAG slave interface driven by user logic. This Debug Bridge can further communicate with other debug cores/ Debug Bridge instances in the design via the Soft-BSCAN (Boundary Scan) master interface. The Soft BSCAN interface enables extension of the JTAG interface to internal USER defined scan chains/Debug Bridge instances.

From PCIe to BSCAN

In this mode, the Debug Bridge receives XVC Commands via PCIe Extended Configuration slave interface. This Debug Bridge can further communicate with other debug cores/ Debug Bridge instances in the design via the Soft-BSCAN (Boundary Scan) interface. The Soft BSCAN master interface enables extension of the JTAG interface to internal USER defined scan chains/Debug Bridge instances.

 **Note:** This mode is only available for UltraScale+™ and UltraScale™ device architectures

From PCIe to JTAG

In this mode, the Debug Bridge receives XVC Commands via PCIe Extended Configuration interface. This Debug Bridge brings out the JTAG pins out of the FPGA through I/O pins. This mode is mainly used to debug design on another board over XVC.

 **Note:** This mode is only available for UltraScale+ and UltraScale device architectures.

From AXI to JTAG

In this mode, the Debug Bridge receives XVC commands via AXI4-Lite interface to send over the JTAG pins to a target device.

JTAG Fallback Support

The XVC based debug solution can be used with AXI masters such as the PCIe® XDMA IP. If the AXI master is in a hang situation or is otherwise not functioning properly, there are no methods to debug those scenarios. To provide a JTAG-based fall back debug pathway that is parallel to the XVC pathway, AMD recommends using the Debug Bridge in BSCAN Primitive mode. A Debug

Bridge in BSCAN Primitive mode can be instantiated in static region and its BSCAN master interface can be connected to the BSCAN slave interface of a second Debug Bridge that is configured with the JTAG Fallback Support enabled. There are two JTAG Fallback Support types:

1. If the Debug Bridge that you want to provide JTAG Fallback for resides in a RP region, you need to enable the External BSCAN Master JTAG Fallback Support.
2. If the Debug Bridge that you want to provide JTAG Fallback for resides in the static region (or in a flat design), you should enable the Internal BSCAN Master JTAG Fallback Support.

MicroBlaze Debug Module (MDM) Support

Debug access to MicroBlaze Debug Module (MDM) is also supported by the Debug Bridge. The MDM BSCAN slave input can be connected to any Debug Bridge configuration mode that supports multiple BSCAN master interfaces at the output (for example, AXI to BSCAN with its BSCAN Master Count greater than zero).

Multiple Debug Trees

The Debug Bridge IP supports the setup and configuration of multiple independent debug trees in a design. You can use multiple independent debug trees in applications where it is desirable to make specific debug logic visible to certain users (for example, system administrators) while hiding it from other users. This feature supports the setup of independent debug trees both in a standalone and Dynamic Function eXchange design. Each of these independent debug trees can be connected to any of the supported debug cores (for example, ILAs, and VIOs)

To enable this feature, you need to instantiate one Debug Bridge IP in the appropriate mode, either the "From AXI to BSCAN" or "From PCIe to BSCAN" mode, for each of the debug trees you want to enable. For instance, in a data center design where multiple classes of users access the DUT, you can instantiate a "From AXI to BSCAN" Debug Bridge IP in the customer-visible address map while instantiating a second "From AXI to BSCAN" Debug Bridge IP in the administrator-visible address map.

When the administrator and/or the customer are ready to debug the design, they only have to connect to the debug bridge using the Vivado Hardware Manager at the correct device offset depending on how they are communicating with the debug cores (for example, PCIe, or JTAG pins). For more information on using the XVC flow with the PCIe core and Debug Bridge in this mode, and for an example design refer to *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG213](#)).

Following table listing the different Debug Bridge modes and features available in those modes:

Table: Debug Bridge Modes

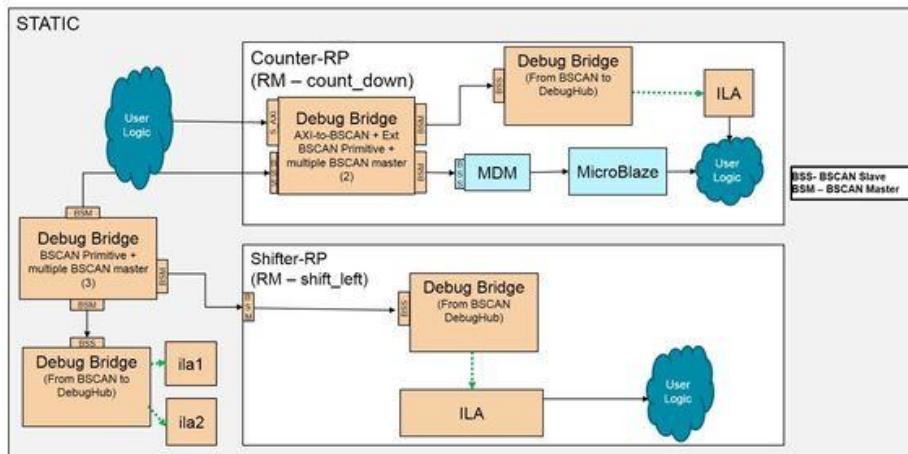
Debug Bridge Mode	XVC Support	Can be Used in Reconfigurable Platform	JTAG Fallback Support	MDM Support
From AXI to BSCAN	Yes	Yes ¹	Yes ²	Yes ³
From JTAG to BSCAN	Yes	Yes ¹	Yes ²	Yes ³

Debug Bridge Mode	XVC Support	Can be Used in Reconfigurable Partition	MDM Support	MDM Support
From PCIe to BSCAN	Yes	Yes ¹	Yes ²	Yes ³
From PCIe to JTAG	Yes	Yes ¹	NA	NA
From BSCAN to DebugHub	No	Yes ¹	NA	Yes ³
BSCAN Primitive	No	No	NA	Yes ³
From AXI to JTAG	Yes	Yes	NA	NA

1. BSCAN Master Count can be greater than 0 and can be connected to other Debug Bridge instances or MicroBlaze/MDM core within the same RP only.
2. Internal BSCAN Mode can only be used when the Debug Bridge is in static partition while External BSCAN Mode can be used when the Debug Bridge is in either the static or RPs.
3. BSCAN Master Count can be greater than 0 and can be connected to other Debug Bridge instances or MicroBlaze/MDM core within the same RP only.

Following illustration of a design with the XVC Debug Bridge in an RP.

Figure: Dynamic Function eXchange Design with the XVC Debug Bridge in an RP



This is an RP design with two reconfigurable partitions, Counter RP and Shifter RPs. This figure illustrates the different Debug Bridge modes used in both static and RP regions.

The static partition of design has two Debug Bridge IPs. The first Debug Bridge IP is in BSCAN Primitive mode and configured to have three BSCAN master interfaces. Two of the BSCAN master interfaces are connected to the Debug Bridge instances in Counter-RP and Shifter-RP partitions providing a parallel path for debugging. The third BSCAN master interface is connected to another Debug Bridge instance within the static partition configured in the From BSCAN to Debug Hub mode. The Debug Bridge configured in From BSCAN to Debug Hub mode can communicate to the various Debug IPs (ILA, VIO, JTAG-to-AXI, etc.) in the design, which in this case is the ILA IP.

In this system the Counter-RP partition contains a Debug Bridge instantiated in the AXI-to-BSCAN mode. You can use this Debug Bridge in XVC mode, the Debug Bridge receives XVC Commands via AXI4-Lite interface. This Debug Bridge can further communicate with other Debug Bridge instances in the design via the Soft-BSCAN (Boundary Scan) interface. Because this Debug Bridge is configured to contain two BSCAN master interfaces, it communicates with the MDM and the Debug Bridge instance configured in From BSCAN to Debug Hub mode. The Debug Bridge configured in From BSCAN to Debug Hub mode can communicate to the various Debug IPs (ILA, VIO, JTAG-to-AXI, etc.) in the design, which in this case is the ILA IP.

On the other hand, the Shifter-RP partition contains only one Debug Bridge instance configured in From BSCAN to Debug Hub mode that can communicate with the various Debug IPs (ILA, VIO, JTAG-to-AXI, etc.) in the design, which in this case is the ILA IP.

For more information see the *Debug Bridge LogiCORE IP Product Guide* ([PG245](#)).

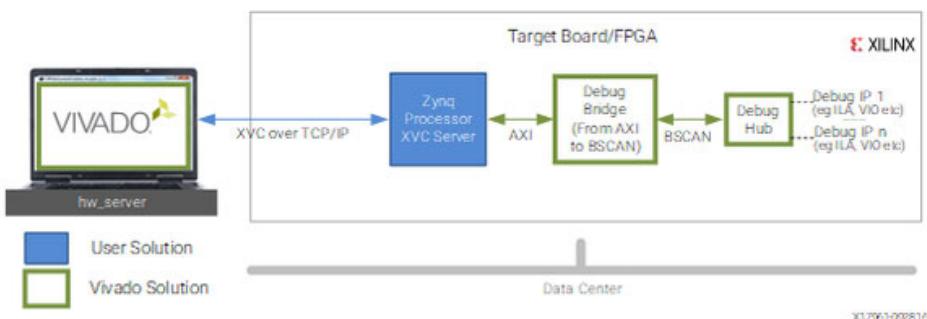
An illustration of some of the Debug Bridge modes is presented as follows.

From AXI to BSCAN

This bridge type is intended for designs using Xilinx Virtual Cable (XVC) to remotely debug an FPGA or SoC device through Ethernet or other interfaces without the need for JTAG cable. In this mode, the Debug Bridge expects to receive XVC commands via AXI4-Lite interface. Use this mode to debug designs on the FPGA over the XVC.

For more information, see the *Debug Bridge LogiCORE IP Product Guide* ([PG245](#)).

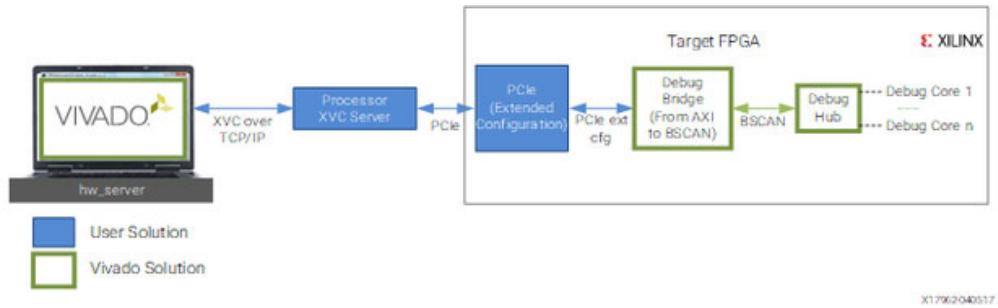
Figure: AXI to BSCAN Debug Bridge



From PCIe to BSCAN

In a typical PCIe setup - you can use the Debug Bridge in the PCIe to BSCAN mode to communicate with the debug cores. In this mode, Debug Bridge connects to the Extended Configuration Interface of the PCIe IP. This is a common data center use case where PCIe is the preferred communication pathway to the Host PC instead of JTAG. For more information on using the XVC flow with the PCIe core and Debug Bridge in this mode, and for an example design refer to *UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG213](#)).

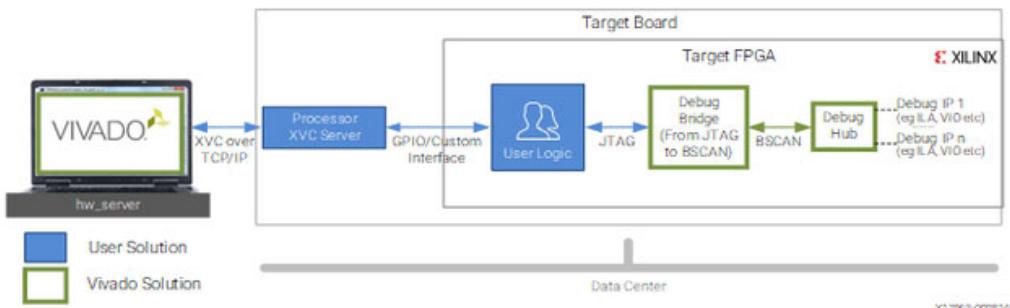
Figure: PCIe to BSCAN Debug Bridge Used with PCIe Extended Configuration Interface



From JTAG to BSCAN

This bridge type is intended for designs that use Xilinx Virtual Cable (XVC) to remotely debug an FPGA or SoC device through Ethernet or other interfaces without the need for JTAG cable. In this mode, the Debug Bridge expects to receive XVC commands via JTAG interface driven by user logic. For more information see the *Debug Bridge LogiCORE IP Product Guide* ([PG245](#)).

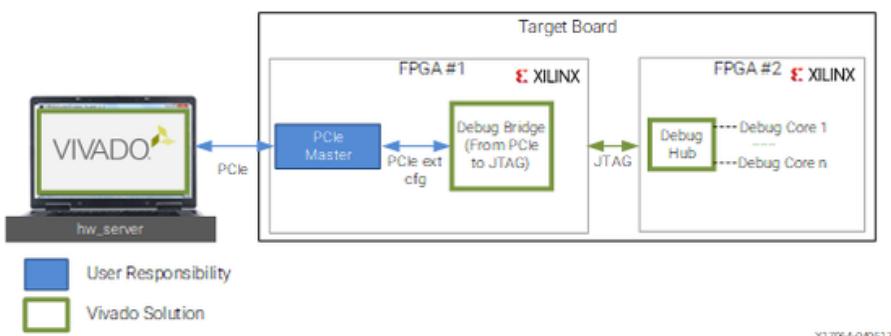
Figure: JTAG to BSCAN Debug Bridge



From PCIe to JTAG

In a PCIe setup, you can use the Debug Bridge in the PCIe to JTAG mode to communicate with the debug cores. In this mode, Debug Bridge connects to the Extended Configuration Interface of the PCIe® IP, which in turn can communicate over JTAG to the debug hub on a different target FPGA.

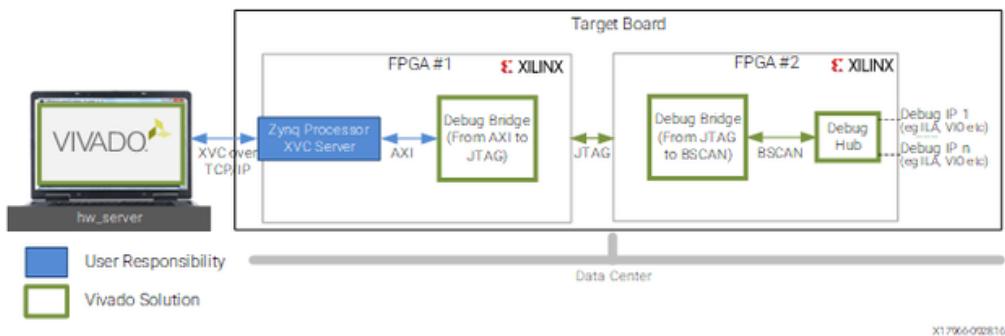
Figure: PCIe® to JTAG Debug Bridge Used with PCIe Extended Configuration Interface



From AXI to JTAG

This bridge type is intended for designs that use Xilinx Virtual Cable (XVC) to remotely debug an FPGA or SoC device through Ethernet or other interfaces. In this mode, the Debug Bridge receives XVC commands via AXI4-Lite interface to send over the JTAG pins to a target device. For more information see the *Debug Bridge LogicCORE IP Product Guide (PG245)*.

Figure: AXI to JTAG Debug Bridge



Xilinx Virtual Cable (XVC) Flow for Versal Devices

XVC is also supported on AMD Versal™ devices as a software-only solution that runs as an application on the APU under Linux and requires no additional IP. For more information, see the [AMDVirtualCable GitHub repository](https://github.com/Xilinx/XilinxVirtualCable) at the following link <https://github.com/Xilinx/XilinxVirtualCable>.

Note: At the current time, only PL debug cores are supported for use with XVC on Versal devices. Examples of PL debug cores are AXIS-ILA and AXIS-VIO. Versal hard-block debug cores such as SYSMON, DDRMC Calibration Debug, PCI® Express Link Debug and IBERT do not currently support remote access over XVC.

XVC Server Implementation

You need to implement the XVC protocol to create an XVC server on the appropriate processor.

XVC Protocol

The XVC protocol allows Vivado IDE to communicate JTAG commands over ethernet to an embedded system so that a target AMD device can be programmed and/or debugged. This enables a vendor agnostic solution for debugging and programming an AMD device. Programming capabilities include the same support as a traditional JTAG connection would provide. Debugging capabilities include operability with Xilinx System Debugger (XSDB) or with Vivado Hardware Debug IP.

The JTAG commands to the device are the same commands that would be transferred to the device if it were natively communicating with a programming cable or using a Digilent module. This ensures functionality between all the existing Vivado Hardware Debug tools.

User XVC 1.0 Commands

The XVC 1.0 Protocol commands are summarized in the following table:

Table: Description of XVC Commands

Command	Description
getinfo	<p>Command format: getinfo: This command gets the XVC Service version. The service returns the following string when it receives "getinfo:" xvcServer_v1.0:<xvc_vector_len>\n<xvc_vector_len> is the max width of the vector that can be shifted into the service.</p>
shift	<p>Command format: shift:[num_bits][tms_vector][tdi_vector] This command shifts in num_bits using the byte vectors tms_vector and tdi_vector num_bits is an integer in little-endian mode. This represents the number of TCK clk toggles needed to shift the vectors out. tms_vector is a byte sized vector with all the TMS shift bits. Bit 0 in Byte 0 of this vector is shifted out first. The vector is num_bits and rounds up to the nearest byte. tdi_vector is like tms_vector but this represents all the tdi vectors to be shifted in. This command returns a byte vector of the same size as tms_vector with the corresponding tdo bits sampled for every bit shifted in. Bit 0 in Byte 0 of this vector is the first tdo value read from the shift</p>
settck	<p>Command format: settck:[period] This command attempts to set the service tck period to [period]. [period] is specified in ns. This is a little-endian integer value This command returns the applied period when it completes settck:. Returned value is specified in ns. This is a little-endian integer value</p>

Initializing Vivado IDE hw_server

When Vivado IDE hw_server is initialized with an XVC connection, Vivado IDE discovers the XVC cable like any USB cable. To do that, start the Vivado IDE hw_server with these arguments:

```
hw_server -e "set auto-open-servers xilinx-xvc:localhost:10200"
```

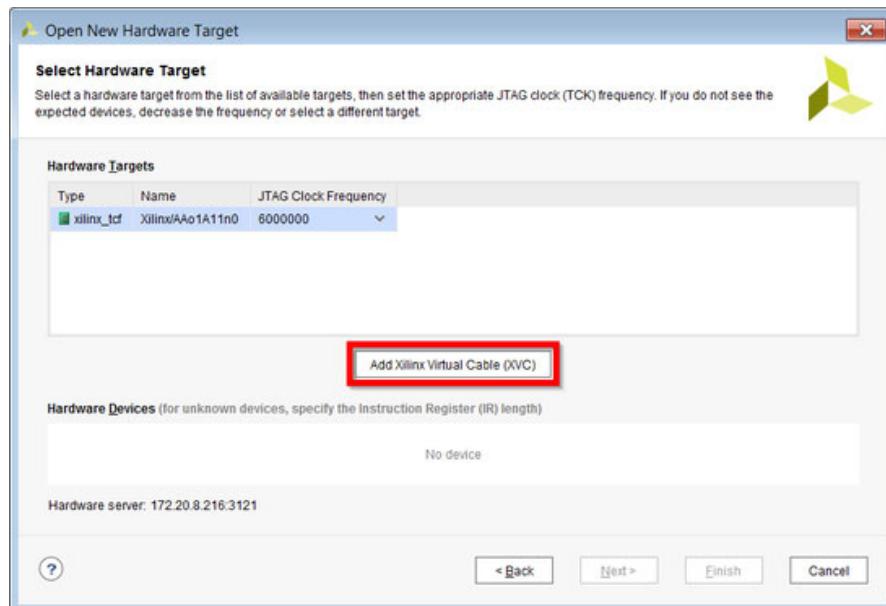
The auto-open-servers option enables the XVC cable to be initialized by hw_server at start up. You can initialize the hardware server to force a connection to an existing XVC cable. The server automatically discovers the XVC cables in future connections.

The argument to auto-open-servers is as follows:

```
xilinx-xvc:<xvc_host_name>:<xvc_port>
```

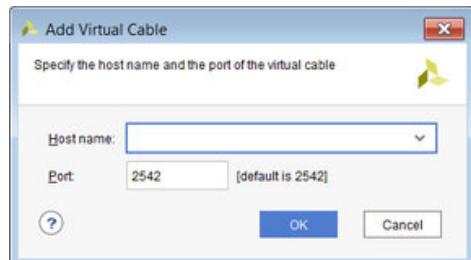
Multiple servers can be specified using comma-separated strings. When the hardware server starts, it attempts to establish connections to the specified XVC servers. Alternatively, you can provide the XVC server details when connecting to the target using the Vivado Hardware Manager Open New Hardware Target wizard, as shown in the following figure.

Figure: Open New Hardware Target Dialog



Click the Add Xilinx Virtual Cable button. This brings up the Add Virtual Cable dialog box as shown in the following figure.

Figure: Add Virtual Cable Dialog



Provide the XVC Hostname and Port number to connect to.

Refer to *Xilinx Virtual Cable Running on Zynq 7000 Using the PetaLinux Tools (XAPP1251)* for an example of this.

This application note shows how to get XVC server running on an AMD Zynq™ 7000 device with a Linux operating system generated with the PetaLinux Tools. A reference design is provided for the Avnet MicroZed board. The target device in this application note is on an AC701 board, programmed, and debugged by the MicroZed board running XVC on Linux.

 **Note:** For an example XVC server implementation over TCP/IP, refer to the following GitHub repository: <https://github.com/Xilinx/XilinxVirtualCable>.

Programming Configuration Memory Devices

The AMD Vivado™ device programmer feature enables you to directly program AMD devices via JTAG. Vivado can also indirectly program select Flash-based configuration memory devices via JTAG. Do this by first programming the AMD FPGA with a special configuration that provides a data path between JTAG and the Flash device interface followed by programming the configuration memory device contents using this data path.

The Vivado device configuration feature enables you to directly configure AMD Devices or Memory Devices using either AMD or Digilent cables. See [Connecting to a Hardware Target Using hw_server](#) for a list of appropriate cables. Operating in Boundary-Scan mode, Vivado can configure or program AMD Devices, and Configuration Memory Devices.

Refer to [JTAG Cables and Devices Supported by hw_server](#) for a complete list of configuration memory devices supported by Vivado.

To program and boot from a Configuration Memory Device in Vivado follow the steps:

1. Generate device images for use with configuration memory devices.
2. Create a Configuration Memory File (.mcs or .bin).
3. Connect to the Hardware target in Vivado.
4. Add the configuration memory device.
5. Program the configuration memory device using the Vivado IDE.
6. Boot the AMD device (optional).

Related Information

[Connecting to a Hardware Target Using hw_server](#)

[Configuration Memory Support](#)

Changing Device Image Properties

On the synthesized or implemented design select Tools > Edit Device Properties to open the Edit Device Properties dialog as follows.

On the synthesized or implemented design, from Flow Navigator, select Settings > Bitstream (Settings → Device Image on AMD Versal™ Devices), and click the Configure additional Bitstream Settings link to open the Edit Device Properties dialog as follows.

Figure: Edit Device Properties: Bitstream Properties for FPGA Devices

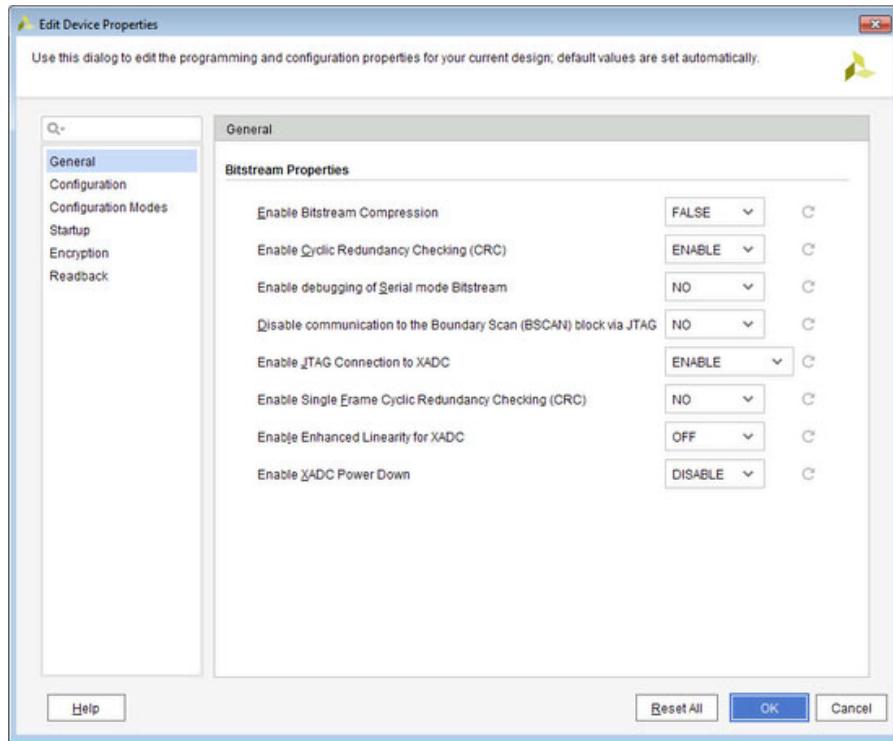
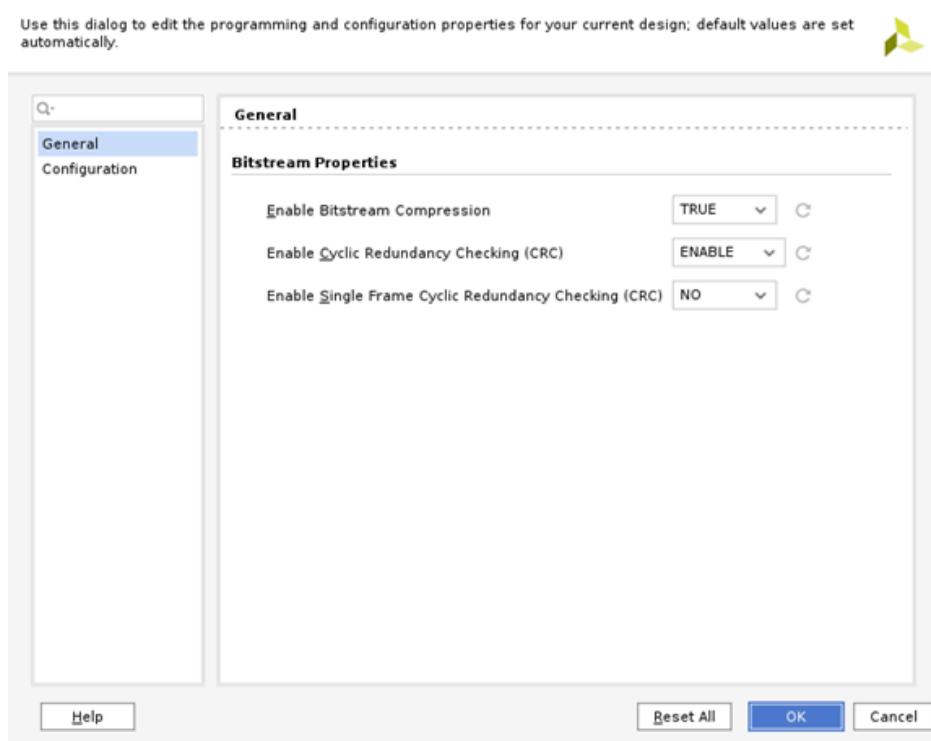


Figure: Edit Device Properties: Bitstream Properties for Versal Devices



Use the search field in the upper left of the dialog box to search for all SPI or BPI related fields and select the appropriate option settings. See Device Configuration Device Image or PDI Settings for the device configuration settings.

Related Information

[Device Configuration Bitstream or PDI Settings](#)

Creating a Configuration Memory File (for FPGA Devices)

Use the `write_cfgmem` Tcl command to create the .mcs or .bin programming file. This file is used in programming the configuration memory device.

For example, to generate an .mcs file to configure an FPGA with a single 1 Gbit BPI configuration memory device:

```
write_cfgmem -format mcs -interface bpix16 -size 128      \
              -loadbit "up 0x0 design.bit"-file design.mcs
```

Note: The `-size` argument to `write_cfgmem` is in megabytes, different from flash device capacity which is based on megabits. Hence, a 1 Gbit sized flash device is provided as 128 megabytes to `write_cfgmem` in the previous example.

Note: `write_cfgmem` automatically sizes the configuration memory file to the size of the bitstream.

Vivado IDE supports the ability to chain multiple .bit files together using the `write_cfgmem` command. To generate an .mcs file for a single 1 Gbit BPI configuration memory device containing multiple bitstreams:

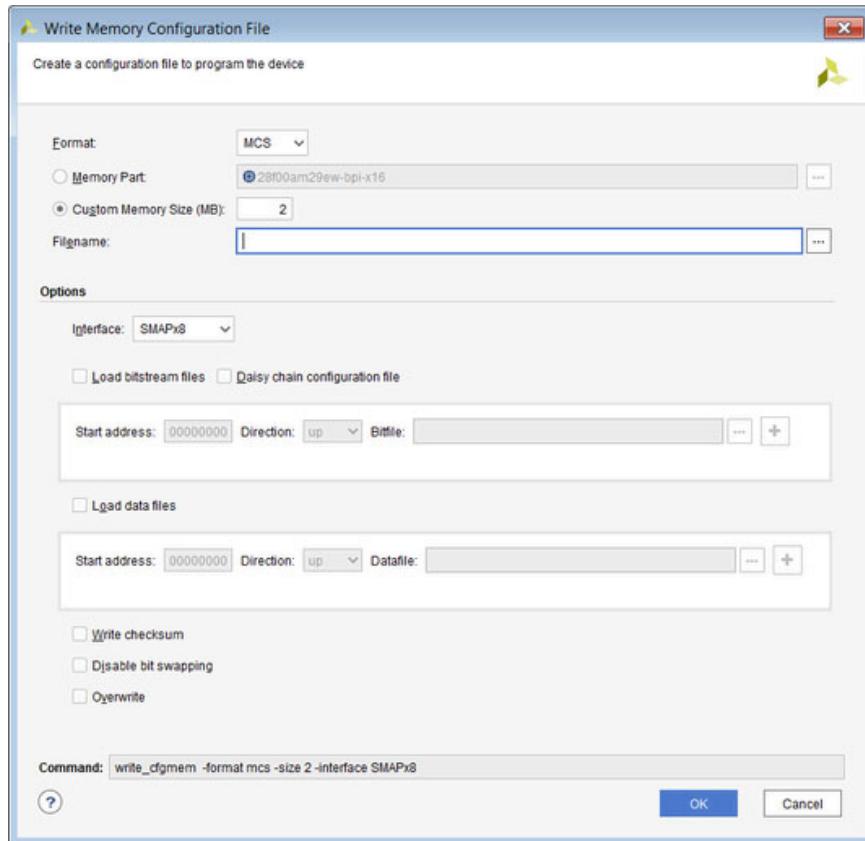
```
write_cfgmem -format mcs -interface bpix16 -size 128      \
              -loadbit "up 0 design1.bit up 0xFFFFF design2.bit"  \
              -file design1_design2.mcs
```

For more information on `write_cfgmem` command refer to the *Vivado Design Suite Tcl Command Reference Guide* ([UG835](#)).

Tip: You can create configuration memory files in Vivado Lab Edition.

You can also create the Configuration Memory file in Vivado IDE. Click on Tools > Generate Memory Configuration File. This brings up the Write Memory Configuration File dialog box as follows:

Figure: Write Memory Configuration File



Select the relevant format and options, and click OK to generate the configuration memory file.

Creating a Configuration Memory File for SPI Dual Quad (x8) Devices (for FPGA Devices)

You can use the `write_cfgmem` Tcl command to generate .mcs images for a dual Quad SPI (x8) device. This command automatically splits the configuration data into two separate .mcs files.

Note: The size specified when generating the .mcs for SPIx8 is the total size of the two Quad Flash devices.

Note: The `write_cfgmem` Tcl command divides the start address by 2 when building .mcs files for dual quad SPI (x8) mode.

Example `write_cfgmem` Usage

This example shows how to generate the .mcs files for a multiboot design with the "golden" bitstream loaded at address 0 and the multiboot bitstream loaded at address 0x0100_0000.

Devices: 2x 256 Mib Quad SPI Flash devices: 256 Mib = 32 MiB

Total storage size: 2 * 32 MiB = 64 MiB

Load addresses:

Golden: 0 * 2 = 0

Multiboot: 0x0100_0000 * 2 = 0x0200_0000

```

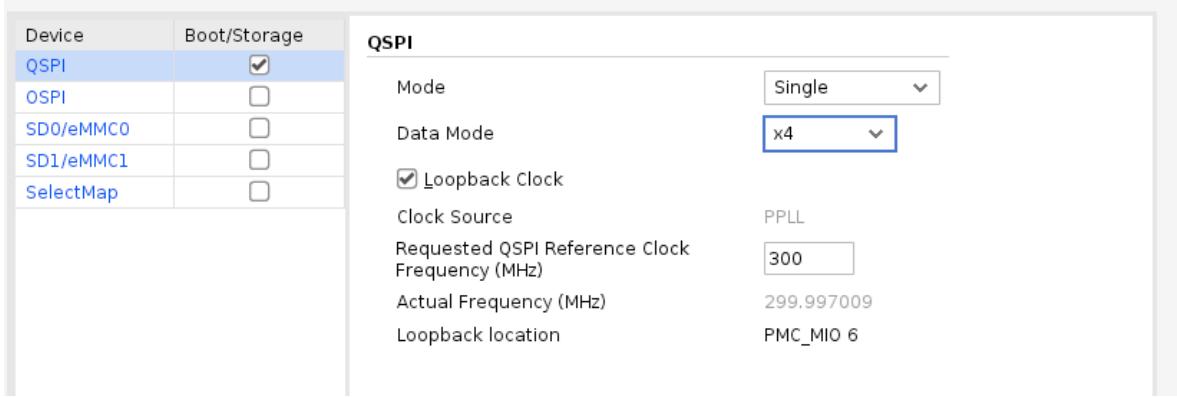
write_cfmem -format mcs -interface spix8 -size 32 \
-loadbit "up 0 ./design1_spix8.bit up 0x02000000 ./design2_spix8.bit" \
-file design1_design2_spix8.mcs

```

Creating an Initialization PDI (for Versal Devices)

AMD Versal™ devices require a user supplied device image that is used to boot the device during the configuration memory programming process. This can be the design PDI but in some cases, such as when a different controller option is desired, a different PDI can be used for this initial boot process.

1. Create a new AMD Vivado™ project targeting the desired Versal device to be used during memory device configuration.
2. Once the project has been created, create a new block design by clicking IP INTEGRATOR → Create Block Design.
3. Click + icon to add a new IP to the IP Integrator Canvas and search for the Control, Interfaces & Processing System. Add an instance of the Control, Interfaces & Processing System IP to the IP Integrator canvas.
4. Double click the Control, Interfaces & Processing System IP to configure options in the PS PMC. At this point, the desired controller options to be used in the Initialization PDI should be configured. For example, to use QSPI x4 during configuration memory device programming, the following could be set.



Note: The options set in the Initialization PDI are used only during the configuration memory device programming steps and do not carry over to the design programmed into the configuration memory device.

5. After configuration is complete in the Control, Interfaces & Processing System IP, run block design validation and save the block design. Return to the Project Navigator by clicking PROJECT MANAGER in the Flow Navigator and right-click on the newly created block design in the sources pane. Select Create HDL Wrapper and select Let Vivado Manage wrapper and auto-update.
6. In the Flow Navigator, select PROGRAM AND DEBUG → Generate Device Image to create the PDI.

Note: The PDI created during this step is the Initialization PDI to be used during the configuration memory programming process.

7. The Initialization PDI has been created and should be saved for use during the configuration memory programming process later in this guide.

Connect to the Hardware Target in Vivado

To connect to a hardware target in AMD Vivado™ , do the following:

1. To boot or configure the AMD device from flash, ensure the mode pins are selected for the target flash type. See the appropriate Technical Reference Manual (for AMD Versal™ devices *Versal Adaptive SoC Technical Reference Manual (AM011)*) or the Configuration User Guide for the device you are targeting.
For more information, see the appropriate Configuration User Guide for the device you are targeting.
2. Follow the steps in Programming the Device to connect to the hardware target.

!! Important: If the board is powered off or cable disconnected, Vivado IDE closes the hardware target. Any Vivado operation in the main Vivado thread is also canceled.

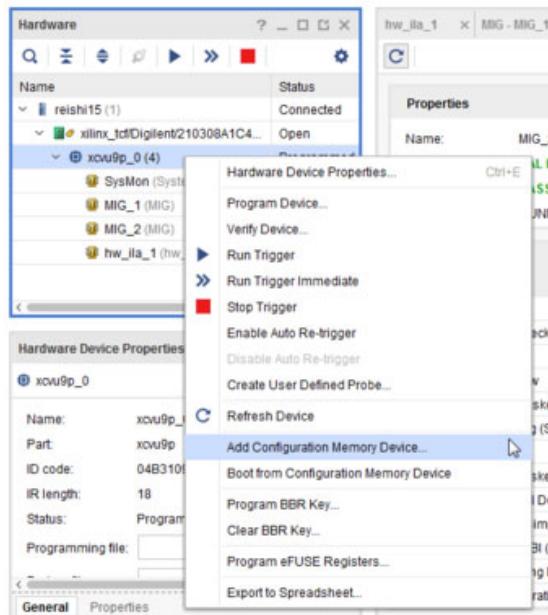
Related Information

[Programming the Device](#)

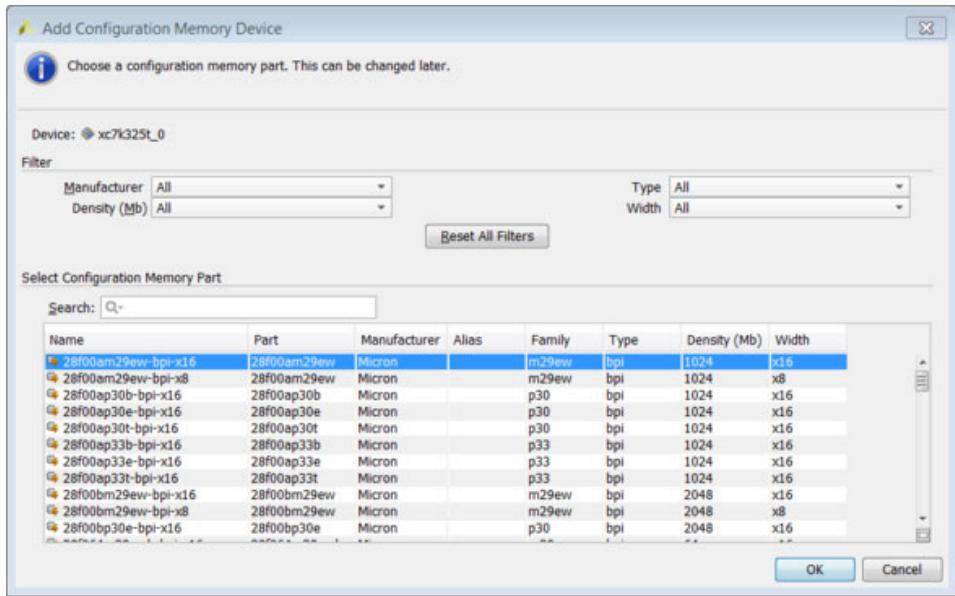
Adding a Configuration Memory Device

To add the configuration memory device to a hardware target in AMD Vivado™ device programmer, do the following:

1. After connecting to the hardware target as outlined, right-click the hardware target and select Add Configuration Memory Device to add the configuration memory device.



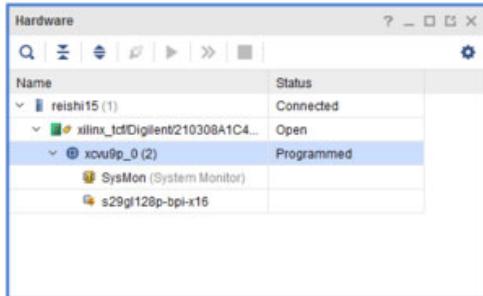
When you click this menu item, the Add Configuration Memory Device dialog box opens:



2. Select the appropriate configuration memory part and click OK.

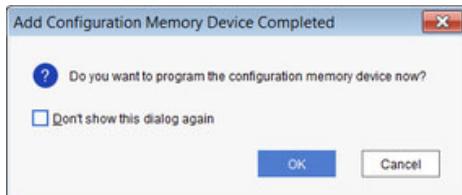
★ Tip: Use the Search field to pare down the list using Manufacturer, Density, or Type information.

The configuration memory device is now added to the hardware target device.

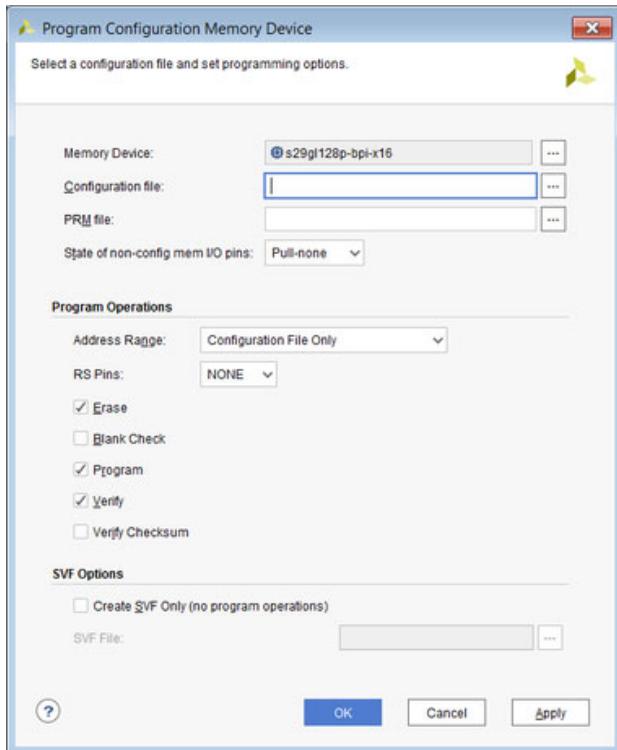


Programming a Configuration Memory Device

- After creating the configuration memory device, AMD Vivado™ device programmer prompts "Do you want to program the configuration memory device now?" as follows:



Click OK to open the Program Configuration Memory Device dialog box.



- Set all the fields in this dialog box appropriately:

Configuration file (.mcs or .bin)

Specifies the file to use for programming the configuration memory device. The memory configuration file is created with the `write_cfgmem` Tcl command. See [Creating a Configuration Memory File](#) for more information.

State of non-config mem I/O pins:

Pull-none

Specifies that the indirect configuration bitstream programmed into the FPGA has the unused I/O pins set to pull-none.

Pull-up

Specifies that the indirect configuration bitstream programmed into the FPGA has the unused I/O pins set to pull-up.

Pull-down

Specifies that the indirect configuration bitstream programmed into the FPGA has the unused I/O pins set to pull-down.

!! Important: Ensure the state of non-config mem I/O pins matches what you set in the `write_bitstream` properties. The default value for this property is pull-down.

Program Operations performed on the configuration memory device:

Address Range

Specifies the address range of the configuration memory device to program. The address range values can be one of the following.

Configuration File Only

Use only the address space required by the memory configuration file to erase, blank check, program, and verify.

Entire Configuration Memory Device

Erase, blank check, program, and verify are performed on the entire device.

RS Pins

Optional. Revision Select Pin Mapping that is used with BPI configuration memory devices only (where the upper two FPGA address pins on the flash are tied to the FPGA RS[1:0]). When the option is enabled, Vivado drives the FPGA RS[1:0] for programming. Refer to the appropriate FPGA Configuration User Guide on application usage.

Erase

Erases the contents of the configuration memory device.

Blank Check

Checks the configuration memory device to make sure the device is void of data prior to programming.

Program

Programs the configuration memory device with the specified Configuration File (.mcs or .bin).

Verify

Verifies that the configuration memory device contents match the Configuration File (.mcs or .bin) after programming.

Verify Checksum

Validates the data programmed in the configuration memory device. The tool calculates the checksum value based on the data programmed in the configuration memory device and compares it to the checksum value specified in the .prm file.

★ Tip: User generates cfgmem file and specifies -checksum write_cftmem option. This step creates the .prm files that contain checksum information about the cfgmem output file.

Create SVF Only

Enabling this option allows for the creation of an .svf file with the program operations that you specified. Other third-party tools can use the .svf file to program configuration memory devices outside of Vivado.

!! Important: When this option is enabled, Vivado generates the .svf file with the relevant program options. It does not actually program the configuration memory device.

3. Click OK to start the Erase, Blank Check, Program, and Verify operations on the configuration memory device per the selections in this dialog box. Vivado notifies you as each operation finishes.

>Note: Pressing Apply, stores the configuration memory settings but does not program the configuration memory device. If you press Cancel after pressing Apply, the configuration memory device is set, and programming can be performed later.

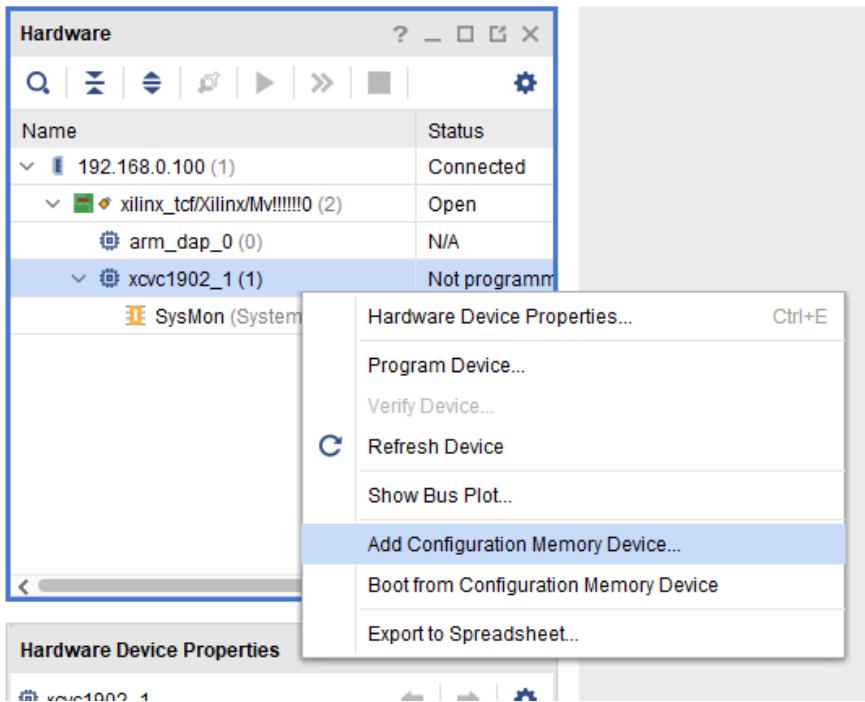
Related Information

[Creating a Configuration Memory File \(for FPGA Devices\)](#)

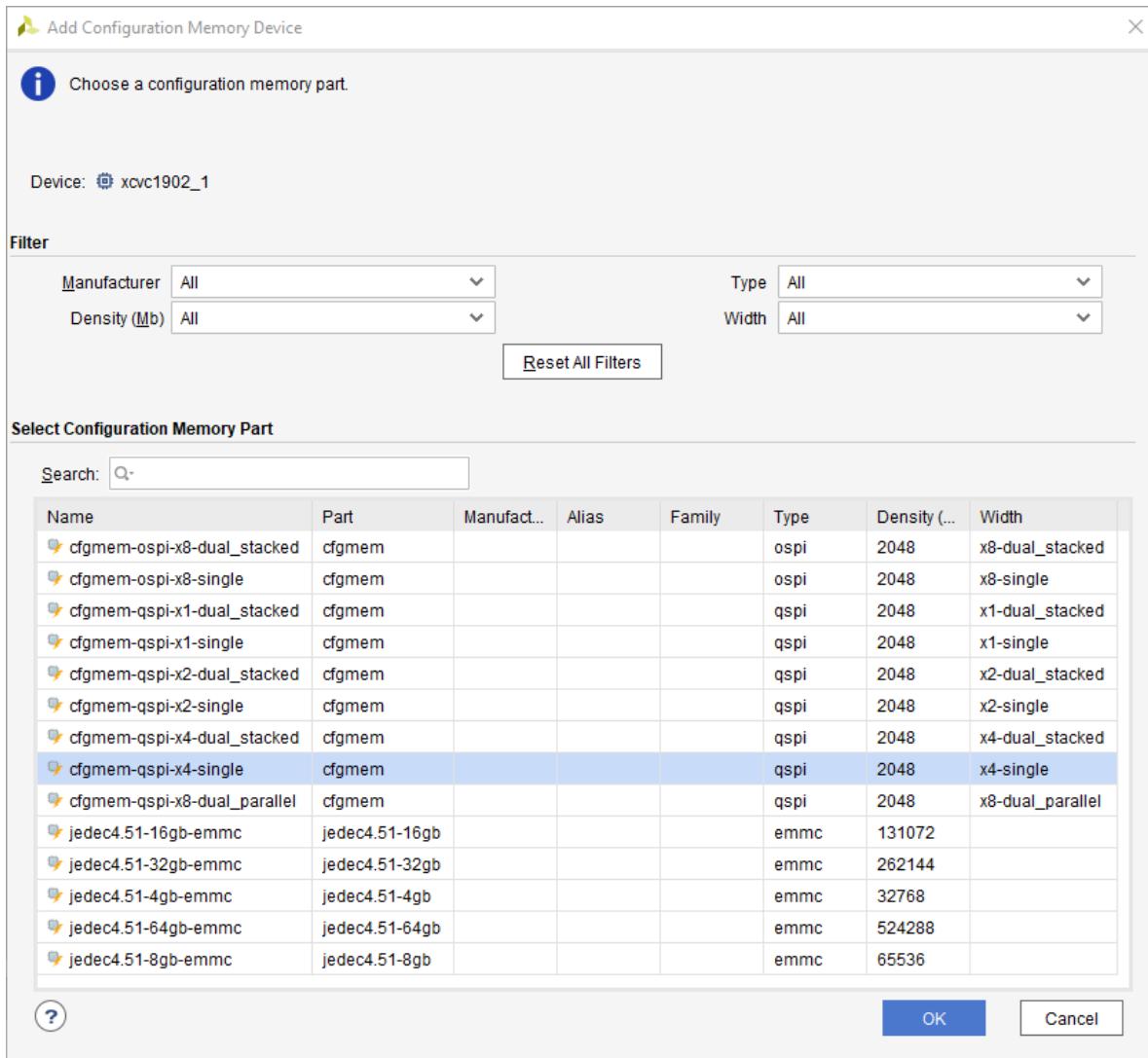
Programming a Configuration Memory Device (Versal Devices)

After creating the initialization PDI, the following steps can be used to program the configuration memory device.

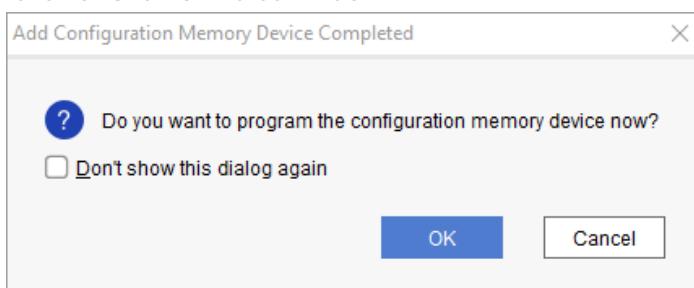
1. Launch Vivado Hardware Manager and connect to a hardware target as described in the previous sections.
2. After connecting to the hardware target, add the configuration memory device by right-clicking the hardware target as follows and selecting Add Configuration Memory Device.

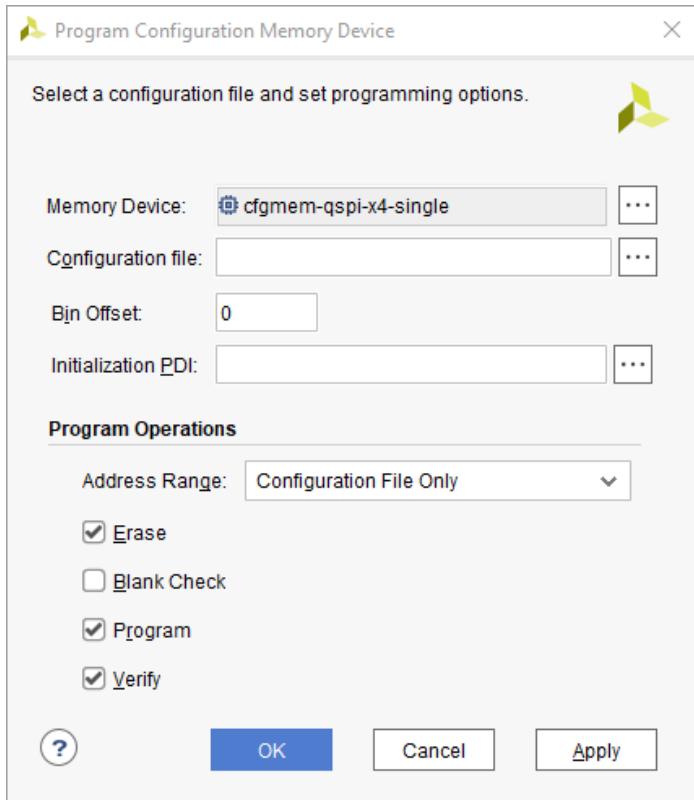


3. When the Add Configuration Memory Device dialog box appears, select the appropriate memory device and click OK.



4. The configuration memory device is now added to the hardware target. To continue, the Vivado device manager prompts "Do you want to program the configuration memory device now?" as follows. Click OK to continue.





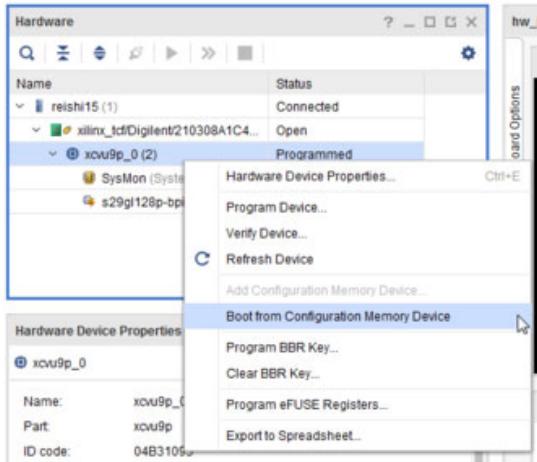
5. In the Program Configuration Memory Device window, set the fields in this dialog box appropriately:
 - o **Configuration file:** The path to the PDI to be programmed into the configuration memory.
 - o **Bin Offset:** The offset to apply when programming the device image into the configuration memory.
 - o **Initialization PDI:** The path to the PDI to be used to boot the device before programming. This can be the same as the **Configuration File** specified previously if there are no differences in the configuration memory controller options selected in the CIPS IP needed to program the device.
 - o **Program Options:** Selecting Configuration File Only performs the selected options on only the address space required by the PDI to be programmed. Selecting Entire Configuration Memory Device performs the selected options including erase, blank check, program, and verify on the entire device.
 - o **Erase:** Erase the contents of the configuration memory device.
 - o **Blank Check:** Checks the configuration memory device to make sure the device is void of data before programming.
 - o **Program:** Programs the configuration memory device with the selected device image (PDI).
 - o **Verify:** Verifies the configuration memory device contents match the selected device image (PDI).
6. Click OK to start the selection operations on the configuration memory device.

 **Note:** Pressing Apply stores the configuration memory settings but does not program the configuration memory device.

Booting the FPGA Device

After programming the configuration memory device, you can issue a soft boot operation (that is, JPROGRAM) to initiate the FPGA configuration from the attached configuration memory device. If you want to perform a Boot operation on the target FPGA select the target device and right-click and select Boot from Configuration Memory Device.

Figure: Boot from Configuration Memory Device



!! Important: There can be situations after booting from configuration memory where the debug cores do not show up immediately due to system boot up considerations. AMD recommends that you wait for the specified time period as appropriate using the boot_hw_device Tcl command in the AMD Vivado™ Hardware Manager Tcl Console, as follows:

```
boot_hw_device after 1000 [refresh_hw_device]
```

Where 1000 can be specified by you as the max "wait_on" value.

Configuration Failures in Master Mode

Note: The following is not supported on MPSOC or Versal architectures.

Configuration failures can occur when a board is in Master BPI or Master SPI mode and the JTAG cable is connected to the Vivado Hardware Manager. When the Hardware Manager polling and recover feature interrupts the Master mode configuration, intermittent configuration failures occur on power up. To avoid this issue, set the following parameter in the Vivado Hardware Manager Tcl console to ensure that the configuration status registers are not updated:

```
set_param xicom.allow_cfgin_commands false
```

Note: This parameter affects all devices in the chain.

Advanced Programming Features

Readback and Verify

Bitstream Verify and Readback for FPGAs and MPSoCs

 **Note:** The following is not applicable to AMD Versal™ architectures.

AMD Vivado™ IDE can verify and/or readback the configuration data (that is, .bit file) downloaded into an FPGA/MPSoC. When using `write_bitstream` to generate the .bit file, use the `-mask_file` option to create a corresponding mask (.msk) file. Run `write_bitstream -help` in the Vivado IDE Tcl Console for details on bitstream generation options.

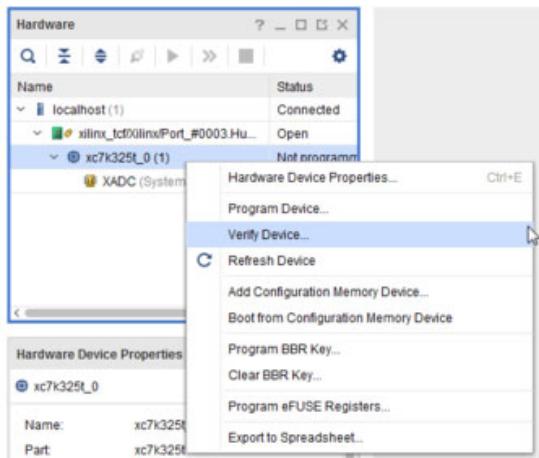
When performing a verify operation, the `verify_hw_devices` Tcl command reads data back from the FPGA/MPSoC and uses the .msk file to determine which readback data bits to skip and which ones to compare against the corresponding bits in the .bit file.

Following is an example of a bitstream verify Tcl command sequence (the .bit and .msk files were generated by a previous call to `write_bitstream`):

```
create_hw_bitstream -hw_device [current_hw_device] \
    -mask kcu105_cnt_ila_uncmpr.msk  kcu105_cnt_ila_uncmpr.bit
verify_hw_devices [current_hw_device]
```

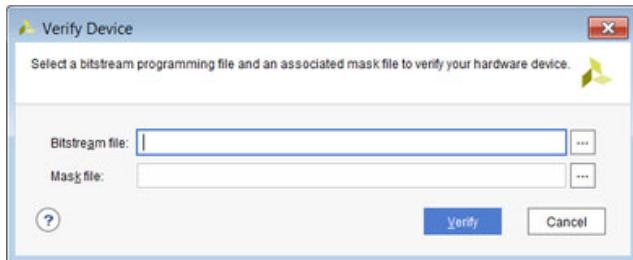
You can use the Vivado Hardware Manager to verify the configuration data. Right-click the device, select Verify Device as follows:

Figure: Verify Device Selection



This opens up the Verify Device dialog box.

Figure: Verify Device Dialog



You need to enter the bit file and corresponding mask (.msk) file. Click Verify to execute the verification.

Use the `readback_hw_device` Tcl command with at least one of the following options to read back the FPGA/MPSOC configuration data:

- To save readback data in ASCII format:

```
-readback_file <filename.rbd>
```

- To saves readback data in binary format:

```
-bin_file <filename.bin>
```

Example: Readback FPGA/MPSOC configuration data in both ASCII and binary formats:

```
readback_hw_device [current_hw_device] \
    -readback_file kcu105_cnt_ila_uncmpr_rb.rbd \
    -bin_file kcu105_cnt_ila_uncmpr_rb.bin
```

1. Bitstream and readback operations are done through the Tcl Console.
2. Verify and readback operations do not work for FPGAs or MPSOCs programmed with encrypted bitstreams. Encrypted bitstreams contain commands that disable readback. Readback is re-enabled by pulsing the device's PROG pin, or if the device/board is powered down and powered back up again.
3. The data readback using `readback_hw_device` contains configuration data only (no configuration commands are included).

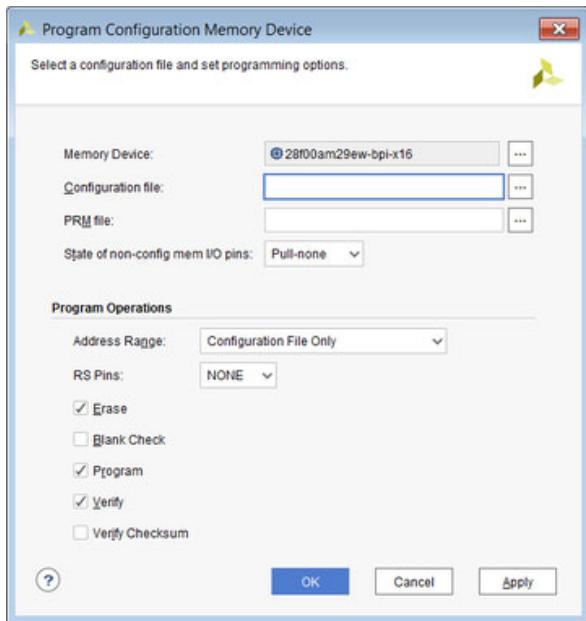
For more information on the readback and mask files, see the "Verifying Readback Data" section in the *UltraScale Architecture Configuration User Guide (UG570)* or the *7 Series FPGAs Configuration User Guide (UG470)*.

Configuration and Boot Memory Verify and Readback for FPGAs or MPSOCs

You can convert a bitstream file (.bit) to an .mcs or .bin file and program it into a configuration memory device, such as serial/SPI or parallel/BPI flash, via the `write_cfm` command. See the *Vivado Design Suite Tcl Command Reference Guide (UG835)* for details.

Verify the configuration memory device through the AMD Vivado™ Design Suite Hardware Manager as shown in the following figure.

Figure: Configuration Memory Verification



You can also verify the configuration memory device by setting the appropriate HW_CFGMEM properties and calling [program_hw_cfgmem](#) as shown in the following code:

```

set_property PROGRAM.ADDRESS_RANGE {use_file} [ get_property PROGRAM.HW_CFGMEM
[lindex [get_hw_devices] 0 ] ]
set_property PROGRAM.FILES [list "H:/projects/k7_led/k7_led_325t_afx_x16_33v.mcs"
] \
[ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0]]
set_property PROGRAM.BPI_RS_PINS {none} [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]
set_property PROGRAM.UNUSED_PIN_TERMINATION {pull-none} [ get_property \
PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.BLANK_CHECK 0 [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]
set_property PROGRAM.ERASE 0 [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]
set_property PROGRAM.CFG_PROGRAM 0 [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]
set_property PROGRAM.VERIFY 1 [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]
startgroup
if {[![string equal [get_property PROGRAM.HW_CFGMEM_TYPE [lindex [get_hw_devices]
0]] [get_property MEM_TYPE
[get_property CFGMEM_PART [get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]]]] } \
{ create_hw_bitstream -hw_device [lindex [get_hw_devices] 0] [get_property
PROGRAM.HW_CFGMEM_BITFILE \
[ lindex [get_hw_devices] 0]]; program_hw_devices [lindex [get_hw_devices] 0]; };

```

```
program_hw_cfgmem -hw_cfgmem [get_property PROGRAM.HW_CFGMEM [lindex  
[get_hw_devices] 0 ]]  
endgroup
```

On FPGA devices, the contents of the configuration memory can be readback through the Vivado Design Suite Tcl Console using the following command sequence (not supported on MPSoC):

```
readback_hw_cfgmem -file test.bin -hw_cfgmem \  
[get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
```

 **Note:** Perform configuration memory readback operations through the Tcl Console only.

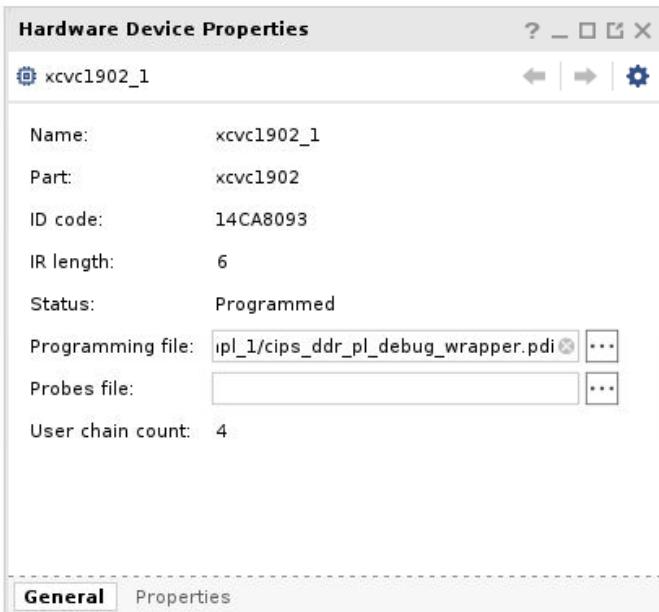
For more information on these features, see the *UltraScale Architecture Configuration User Guide* ([UG570](#)) or the *7 Series FPGAs Configuration User Guide* ([UG470](#)).

PDI Verify for Versal Adaptive SoC Devices

The Vivado IDE does not support configuration readback of .rcco and .rnpi files from a Versal device, however the PDI downloaded into a Versal adaptive SoC device can be verified using the `verify_hw_devices` Tcl command. The `verify_hw_devices` command compares the selected PDI's identification fields (image/node ID, unique ID, parent unique ID, and function ID) to the identification fields found on the programmed device.

To verify a PDI matches the images currently programmed into the Versal device:

1. Launch the Vivado Hardware Manager and connect to the previously programmed target.
2. In the Hardware Device Properties window, click the  icon and select the PDI to verify the programmed device against.



3. At the Vivado Tcl prompt, run the command `verify_hw_devices [current_hw_device]` to verify the Image IDs contained in the selected PDI against those found in the programmed device.

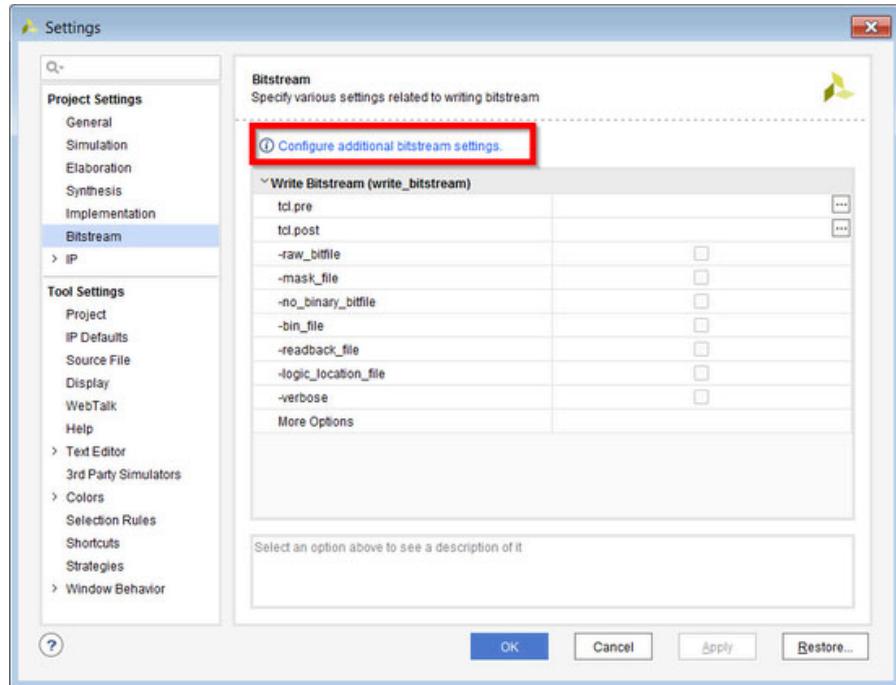
 **Note:** For Versal architectures, the PDI verify feature is currently Tcl only.

Generating Encrypted and Authenticated Files for 7 Series Devices

Note: For additional information, refer to *Using Encryption to Secure a 7 Series FPGA Bitstream* (XAPP1239).

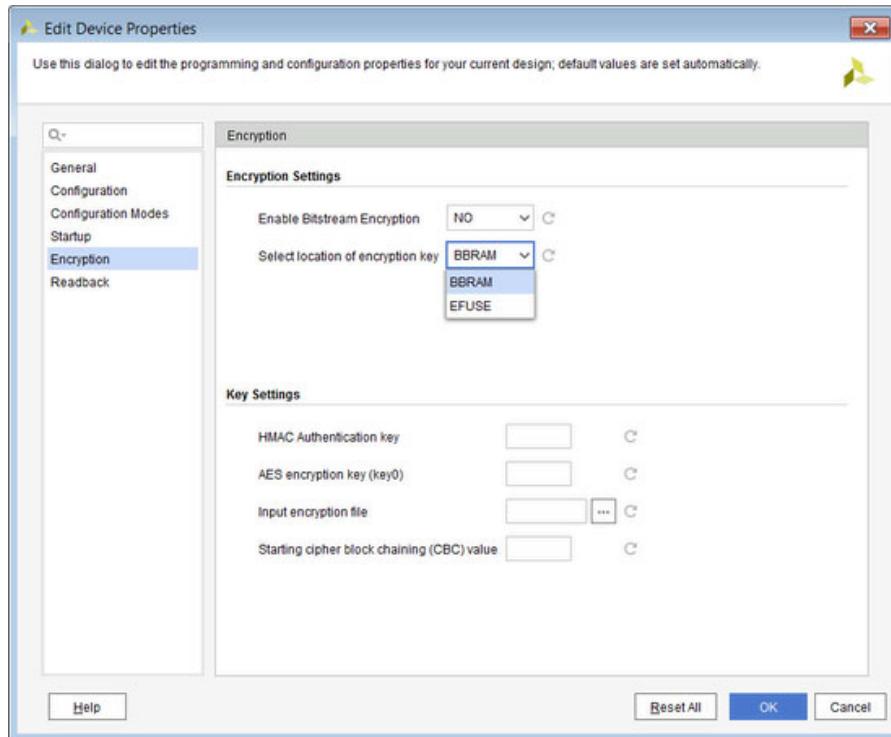
To generate an encrypted bitstream, open an implemented design in Vivado IDE. From the main toolbar Select Flow > Bitstream Settings to make the Settings dialog box appear. At the top of the dialog box click Configure Additional Bitstream Settings.

Figure: 7 Series Settings



This brings up the Edit Device Properties dialog box. Select Encryption in the left-hand pane.

Figure: 7 Series Configure Encryption Settings



In the Edit Device Properties dialog box, specify the encryption and key settings:

- Encryption Settings
 - Set Enable Bitstream Encryption to YES.
 - Set Select location of encryption key to either BBRAM or EFUSE.
 - The key location is embedded in the encrypted bitstream.
 - When the encrypted bitstream is downloaded to the device, it instructs the FPGA to use the key loaded into the BBR or the eFUSE key register to decrypt the encrypted bitstream.
- Key Settings
 - Specify HMAC authentication key and Starting cipher block chaining (CBC) value.
 - If these values are unspecified, Vivado generates a random value for you.
 - These values are embedded in the encrypted bitstream and do not have to be programmed into the FPGA.

 **Note:** These values are stored in the current project constraints file unless an input encryption file is specified. To avoid storing this value in the constraints file, specify the input encryption file.

- Specify the AES encryption key to use when encrypting the bitstream. You can use up to 64 hex characters to specify the 256-bit key.
 - The key is written to a file with the .nky file extension. Use this file when loading the key into the BBR or when programming the key into the eFUSE key register.

 **Note:** These values are stored in the current project constraints file unless an input encryption file is specified. To avoid storing this value in the constraints file, specify the input encryption file.

- Specify Input encryption file.
 - Specify an existing .nky file to obtain the encryption key settings. This field is optional and can be omitted if specifying the AES, HMAC, and CBC manually.

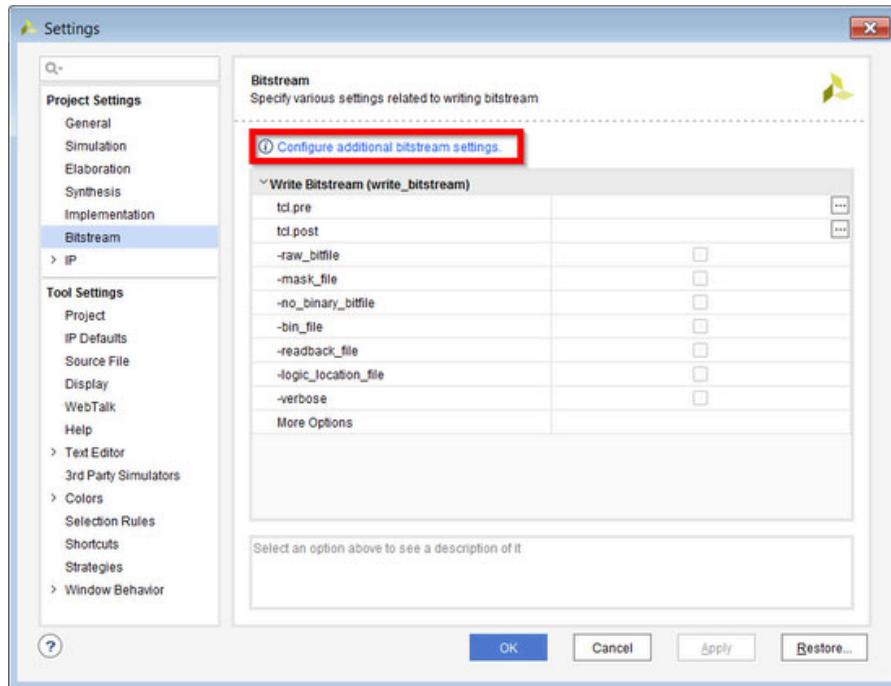
After specifying the encryption settings, click OK to apply the settings to the project, and regenerate your bitstream. Upon completing the `write_bitstream` operation, you get a programming file and a .nky, encryption file.

Generating Encrypted and Authenticated Files for UltraScale and UltraScale+

 **Note:** For additional information refer to *Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream* ([XAPP1267](#)).

To generate an encrypted bitstream, open an implemented design in Vivado IDE. From the main toolbar select Flow > Bitstream Settings to make the Settings dialog box appear. At the top of the dialog box, click Configure Additional Bitstream Settings.

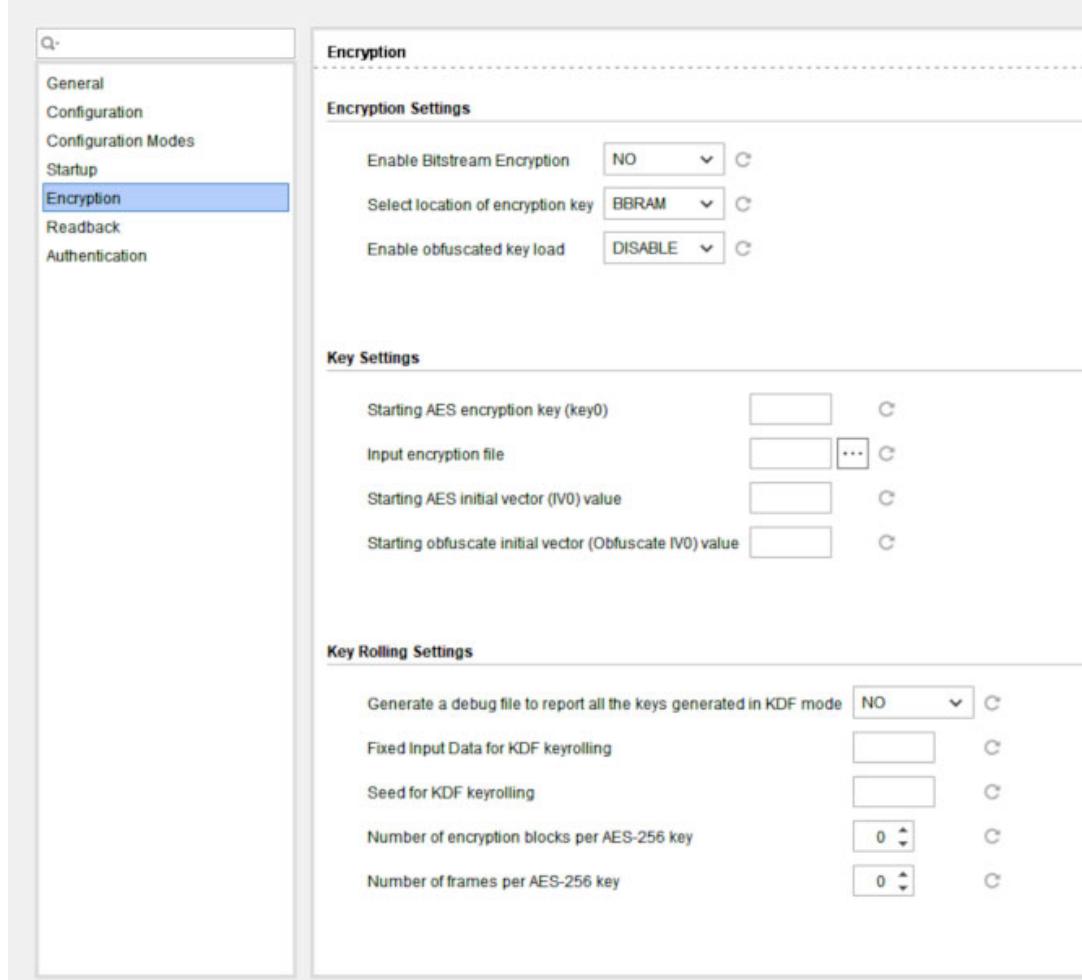
Figure: UltraScale and UltraScale+ Configure Additional Bitstream Settings



This brings up the Edit Device Properties dialog box. Select Encryption in the left-hand pane.

Figure: UltraScale Configure Encryption Settings

Use this dialog to edit the programming and configuration properties for your current design; default values are set automatically.



In the Edit Device Properties dialog box, specify the Encryption Settings and Key Settings as follows.

Encryption Settings

- Set Enable Bitstream Encryption to YES.
- Set Select location of encryption key to either BBRAM or EFUSE.
 - The key location is embedded in the encrypted bitstream.
 - When the encrypted bitstream is downloaded to the device, it instructs the FPGA to use the key loaded into the BBR or the eFUSE key register to decrypt the encrypted bitstream.

Key Settings

- Specify the Starting AES encryption key (key0) to use when encrypting the bitstream. You can use up to 64 hex characters to specify the 256-bit key.
 - The key is written to a file with an .nky file extension. Use this file when loading the key into the BBR or when programming the key into the eFUSE key register.

Note: This value is stored in the current project constraints file unless an input encryption file is specified. To avoid storing this value in the constraints file, specify the input encryption file.

- Specify Input encryption file: Specify an existing .nky file to obtain the encryption key settings. This field is optional and can be omitted if specifying the AES, HMAC, and CBC manually.
- Specify the Starting AES initial vector (IV0) value. Select initialization vector for the first key.

Note: Each key needs a separate initialization vector value that can be supplied through the input encryption file.

Note: This value is stored in the current project constraints file. To avoid storing this value in the constraints file, specify the input encryption file.

- Specify the Starting obfuscate initial vector (Obfuscate IV0) value. Select the initialization vector for the obfuscated key.

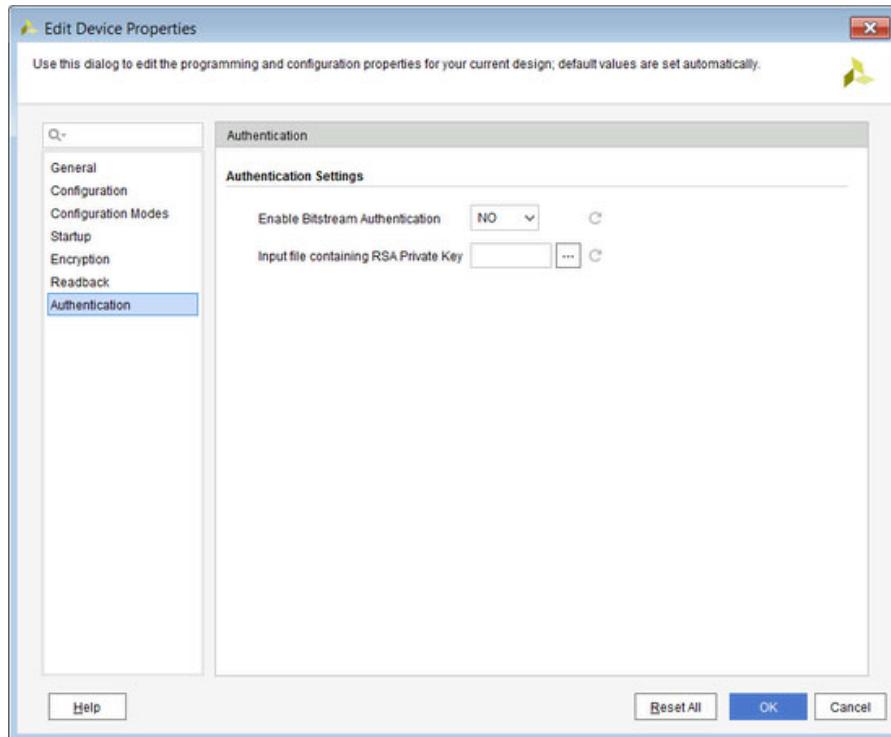
Note: This value is stored in the current project constraints file. To avoid storing this value in the constraints file, specify the input encryption file.

Key Rolling Settings

- Specify if a debug file to report all the keys generated in KDF mode should be generated.
- Specify Fixed Input Data for KDF key rolling. This is an optional 60 byte fixed input value, specified as a 120-digit hexadecimal value. This 60-Byte input along with the 4-Byte counter serves as the 64 byte input to the KDF pseudo-random fixed input value via RAND_bytes.
- Specify Seed for KDF Key rolling. This is an optional 32 byte seed value for the KDF, specified as a 64 digit hexadecimal value.
- Specify Number of encryption blocks per key and Number of frames per AES-256 key. The number of encryption blocks and frames are used to specify how many sections a bitstream is broken into with distinct keys.

For authentication settings select Authentication in the left-hand pane

Figure: Edit Device Properties - Authentication



In the Edit Device Properties-Authentication dialog box, specify the encryption and key settings as follows:

Authentication Settings

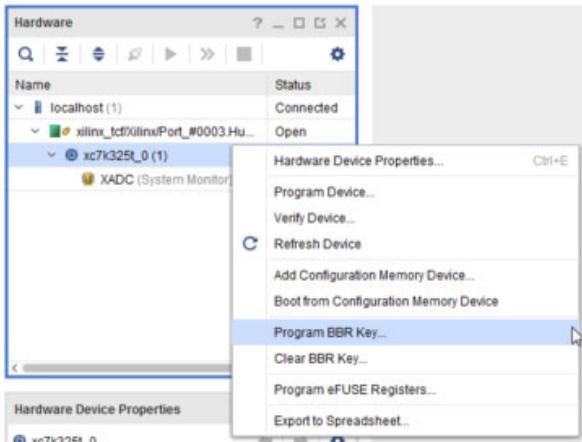
- Set Enable Bitstream Authentication to YES.
- Specify the Input file containing RSA Private Key.
- Provide an RSA private key file after specifying the encryption and authentication settings, Click OK to apply the settings to the project. Re-run Implementation and regenerate the bitstream file. Upon successful completion of the `write_bitstream` operation, the generated .nky encryption key file appears in the same directory as the encrypted bitstream file.

You can protect IP in bitstreams by encrypting the bitstreams with a 256-bit Advanced Encryption Standard (AES) key and downloading the bitstreams to run only on authorized FPGAs. Do this by programming the 256-bit key into the BBR register of the authorized FPGAs before downloading the encrypted bitstream.

Programming the AES Key for 7 Series Devices

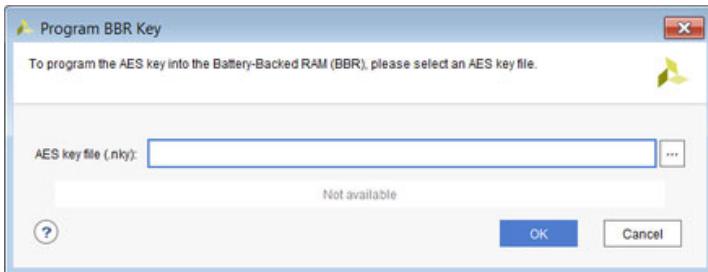
To program the AES key into the BBR, right-click the FPGA in the Hardware window, and select Program BBR Key.

Figure: Program the BBR Key



In the Program BBR Key dialog box, specify the AES key (.nky) file by typing the file name or navigating to the desired file. After specifying a valid .nky file, the AES key field automatically fills in. Click OK, to have the Hardware Manager program/load the key into the BBR.

Figure: Program BBR Key - 7 Series



After programming the key, program the FPGA with an encrypted bitstream that:

- was encrypted using the same AES key as was loaded into BBR.
- had BBRAM selected as the specified encryption key location.

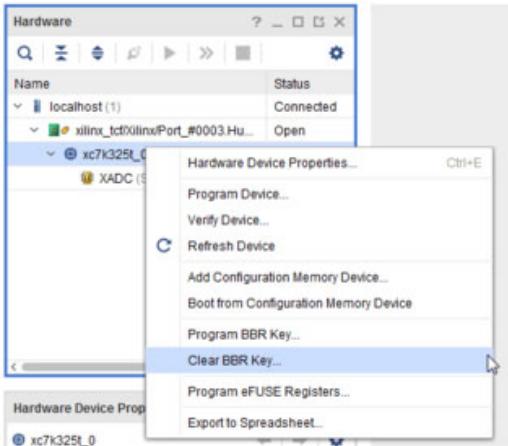
Clearing the AES Key for 7 Series Devices

To clear the AES key manually, disconnect the Vbatt pins and power-cycle the board.

Note: Pressing or pulsing the PROG pin when the board/FPGA is powered up does *not* clear the BBR register.

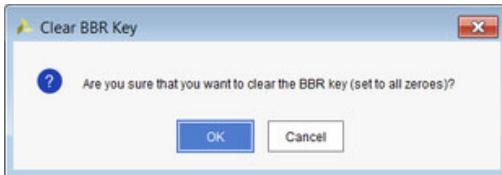
Alternatively, you can clear the AES key in Vivado IDE by right-clicking the FPGA in the Hardware window, selecting Clear BBR Key...

Figure: Clearing the AES Key for 7 Series Devices



When the Clear BBR Key dialog box appears, click OK to clear the key from the device.

Figure: Clear BBR Key Dialog Box

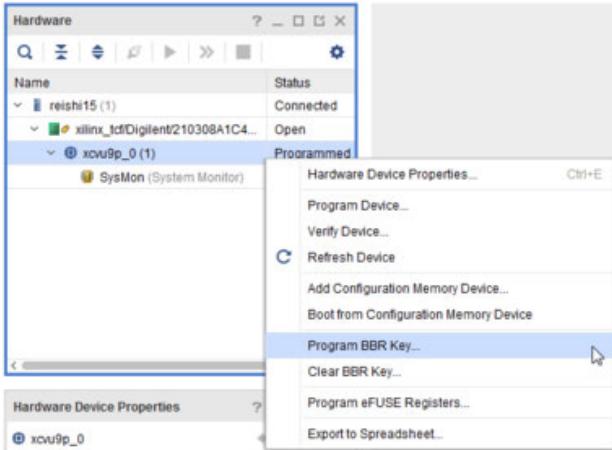


!! Important: When `verify_hw_devices` is performed on the BBR key, an error is displayed. To verify the BBR key, the user should test this by programming the FPGA with a bitstream that has the key. Vivado does not support any post BBR program verify option to verify the programmed BBR key.

Programming the AES Key for UltraScale and UltraScale+ Devices

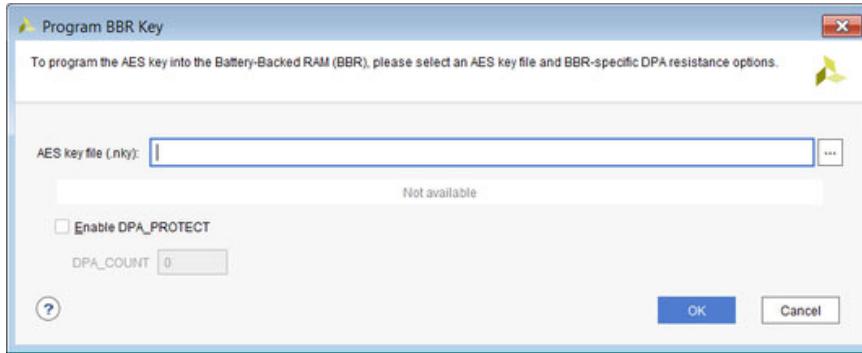
To program the AES key into the BBR, right-click FPGA in the Hardware window, and select Program BBR Key.

Figure: Program the BBR Key from Hardware Window



The Program BBR Key dialog box appears.

Figure: Program BBR Key – UltraScale and UltraScale+ Devices



In the Program BBR Key dialog box, specify the AES key file (.nky) and Enable DPA PROTECT as follows.

1. Specify the AES key file (.nky) by typing the file name or navigating to the desired file. After specifying a valid .nky file, the AES key field automatically fills in.
 2. Check the Enable DPA PROTECT check box.
 3. Specify the DPA_COUNT value. The valid range is 1–256 when enabled.
-
- Note:** For more details on the BBR AES key and DPA_PROTECT feature refer to the *UltraScale Architecture Configuration User Guide (UG570)*.
-
4. Click OK, to have the Hardware Manager program load the key into the BBR.
 5. After programming the key, program the FPGA with an encrypted bitstream that was encrypted using the same AES key as was loaded into BBR and had BBRAM selected as the specified encryption key location.

!! Important: For UltraScale devices, if you download an encrypted bitstream (which uses the BBR as the key source) before programming the key into the BBR register, the FPGA locks up and you are unable to load the BBR key. You can still download unencrypted bitstreams, but you are unable to download encrypted bitstreams because the FPGA prevents you from downloading a key into BBR. You must power-cycle the board to unlock the UltraScale device and then reload the BBR key.

!! Important: When `verify_hw_devices` is performed on the BBR key, an error is shown. To verify the BBR key, the user should test this by programming the FPGA with a bitstream that has the key. Vivado does not support any post BBR program verify option to verify the programmed BBR key.

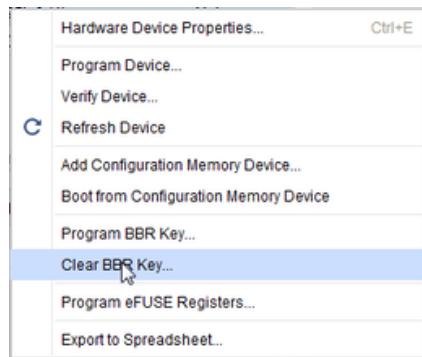
Clearing the AES Key for UltraScale and UltraScale+ Devices

To clear the AES key manually, disconnect the Vbatt pins and power-cycle the board.

Note: Pressing or pulsing the PROG pin when the board/FPGA is powered up does *not* clear the BBR register.

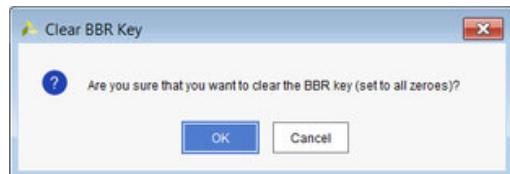
Alternatively, you can clear the AES key in Vivado IDE by right-clicking the FPGA in the Hardware window, selecting Clear BBR Key....

Figure: Clearing the AES Key for UltraScale and UltraScale+ Devices



When the Clear BBR Key dialog box appears, click OK to clear the key from the device.

Figure: Clear BBR Key Dialog Box



eFUSE Register Access and Programming

Note: The following method for eFUSE access and programming is not supported on MPSOC and Versal devices.

7 series, UltraScale, and UltraScale+ devices have one-time programmable bits called eFUSE bits that perform specific functions. The different eFUSE bit types are as follows:

- FUSE_DNA - Stores unique device identifier bits (non-programmable).
 - FUSE_USER - Stores a 32-bit user-defined code.
 - FUSE_KEY - Stores a key for use by the AES bitstream decryptor.
 - FUSE_CNTL - Controls key use and read/write access to eFUSE registers.
 - FUSE_SEC - Controls special device security settings in UltraScale and UltraScale+ devices.
-

!! Important: Programming eFUSE register bits is a one-time-only operation. Once eFUSE register bits are programmed (from unprogrammed state 0 to programmed 1 state), they cannot be reset to 0 and/or programmed again. You should take great care to double-check your settings before programming any eFUSE registers.

⚠ CAUTION! If any eFUSE register bits were previously programmed (from unprogrammed state 0 to programmed state 1) and an attempt is made to program them again, the Vivado hardware manager issues a critical warning indicating that some bits have already been programmed. However, despite this warning, subsequent eFUSE register bits that have not been programmed during the previous operation (in unprogrammed state 0) are programmed.

!! Important: AMD recommends programming the FUSE_USER, FUSE_KEY, and FUSE_RSA registers first, rerunning the eFUSE programming wizard to program the FUSE_SEC bits to control the FPGA security settings, and finally, the FUSE_CNTL bits to control read/write access to these eFUSE bits.

Cable Support for eFUSE Programming

The list of compatible JTAG download cables and devices that support efuse programming are:

- AMD SmartLynq Data Cable (HW-SMARTLYNQ-G/DLC20)
- AMD Platform Cable USB II (DLC10)
- Digilent JTAG-HS1
- Digilent JTAG-HS2
- Digilent JTAG-HS3

eFUSE Register Access and Programming for 7 Series Devices

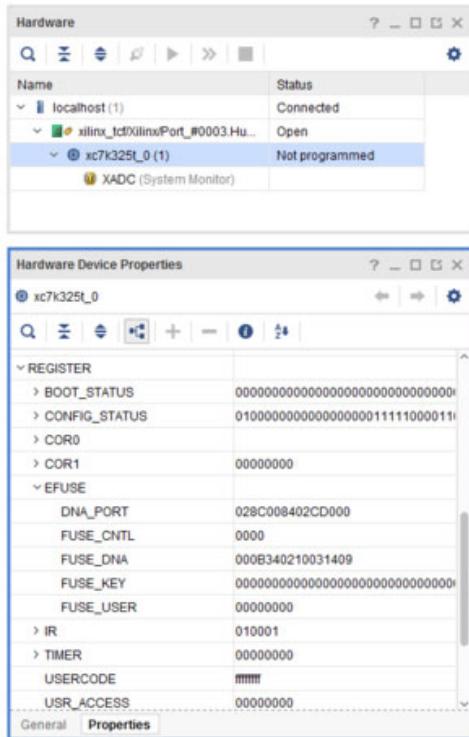
FUSE_DNA: Unique Device DNA

Each 7 series device has a unique device ID called device DNA that has already been programmed into it by AMD. 7 series devices have a 64-bit DNA. You can read these bits by running the following Tcl command in the Vivado Design Suite Tcl Console:

```
get_property [lindex [get_hw_device] 0] REGISTER.EFUSE.FUSE_DNA
```

You can also access the device DNA by viewing the eFUSE registers in the Hardware Device Properties window in Vivado Design Suite as shown in the following figure.

Figure: eFUSE DNA

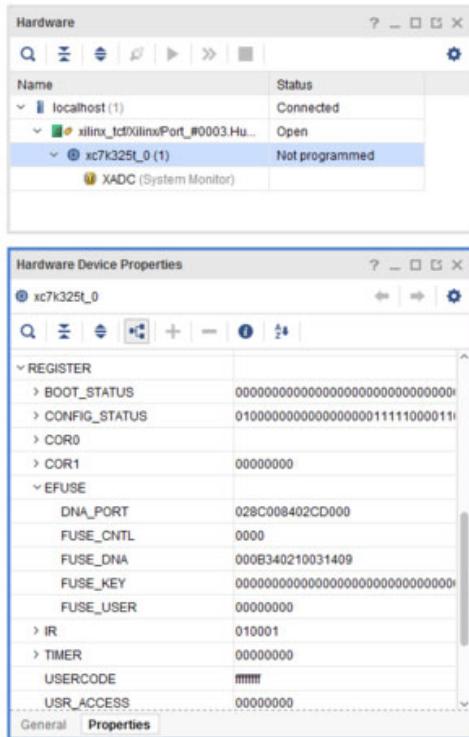


For more information on these features, see the *7 Series FPGAs Configuration User Guide* ([UG470](#)).

Programming the eFUSE Registers

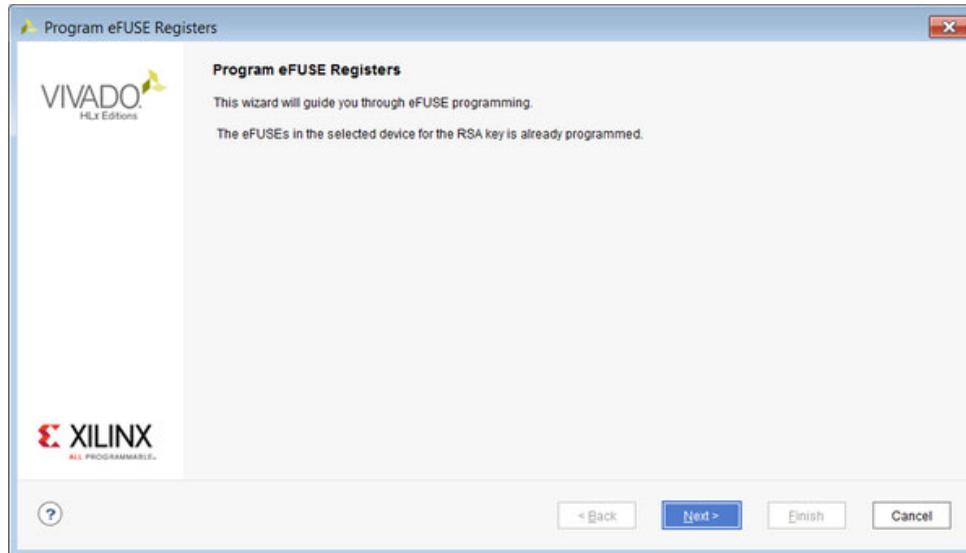
To program the eFUSE registers, right-click the FPGA in the Hardware window, select Program eFUSE Registers.

Figure: Select Program eFUSE Registers



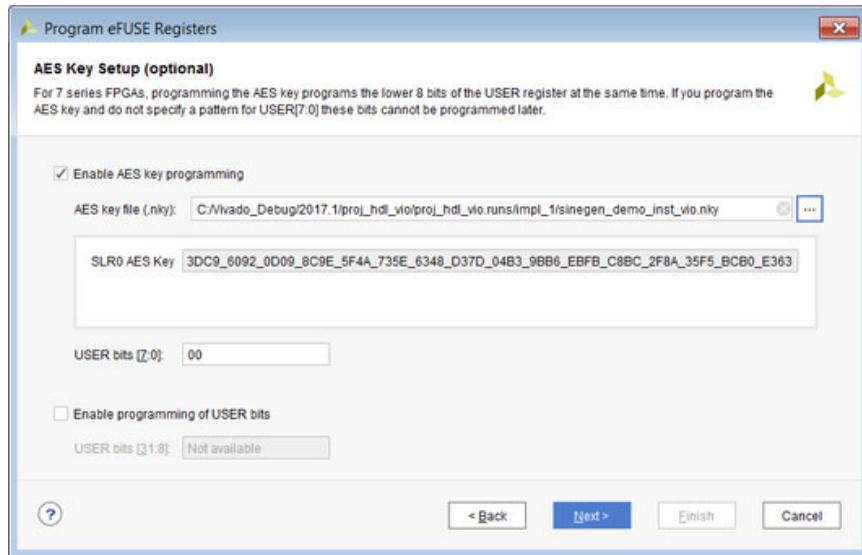
The Program eFUSE Registers wizard appears as shown in the following figure and guides you to set the various options for the eFUSE registers.

Figure: Program eFUSE Registers Wizard



In the AES Key Setup pane, specify the following settings:

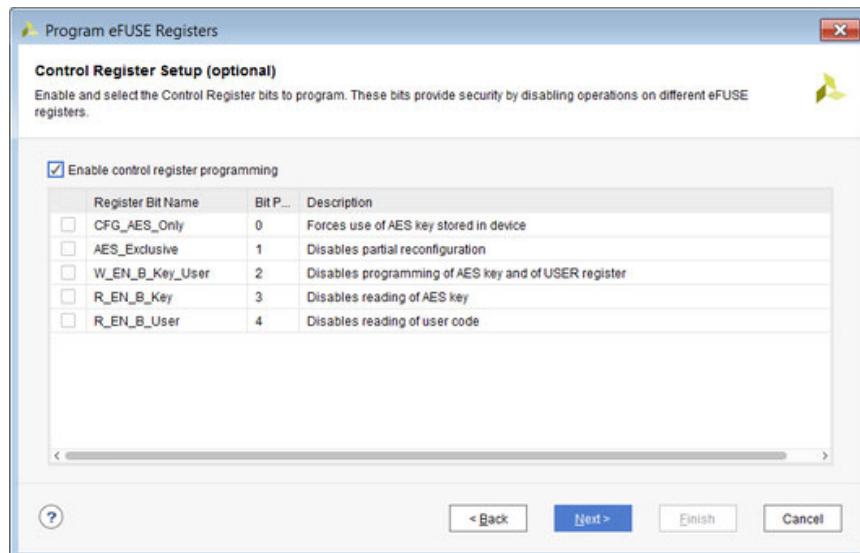
Figure: eFUSE AES Key Setup



- AES Key file
 - Specify the AES key file (.nky) by typing the file name or navigating to the desired file. After specifying a valid .nky file, the AES key field automatically fills in.
- USER bits [7:0] and USER bits [31:8]
 - The USER eFUSES bits are provided to allow users to program their own special 32-bit pattern. The lower eight FUSE_USER bits are programmed at the same time as the 256-bit Advanced Encryption Engine (AES) key. The upper 24 user bits can be programmed concurrently with AES key or at a later time.

In the Control Register Settings pane, specify the following settings:

Figure: Control Register Setup



CFG_AES_Only

When set, forces the use of the stored AES key.

AES_Exclusive

When set, disables use of partial reconfiguration.

W_EN_B_Key_User

When set, disables programing of AES key and User register.

R_EN_B_Key

When set, disables reading of AES key.

R_EN_B_User

When set, disables reading of user code.

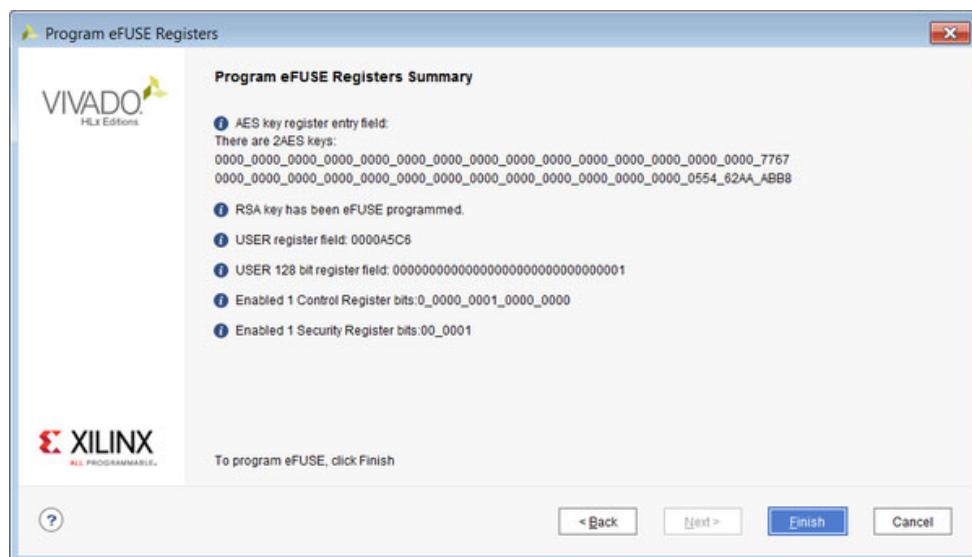
W_EN_B_Cntl

When set, disables programing of this control register.

For more information on these features, see the *7 Series FPGAs Configuration User Guide (UG470)*.

Review the eFUSE settings in the Program eFUSE Registers Summary page.

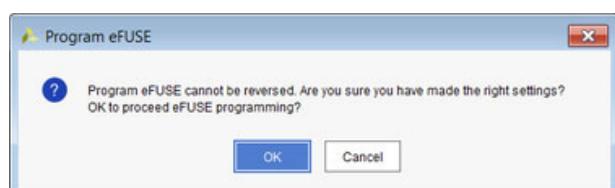
Figure: Program eFUSE Registers Summary



All bits set in the Program eFUSE Registers wizard panels are shown in this pane. In this pane you can see individual bit settings to review the specific programming settings. Review this summary page carefully to ensure every bit that is intended to be programmed is set.

Click Finish to bring up the Program eFUSE confirmation dialog box:

Figure: Program eFUSE Confirmation



Click OK to program the specified fuse bits.

Forcing eFUSE Programming

To force any bit to be set regardless of where it is in the register or whether it has been previously programmed the -force_efuse option to program_hw_devices can be used. When used, only basic register boundary checking occurs.

eFUSE Register Access and Programming for UltraScale and UltraScale+ Devices

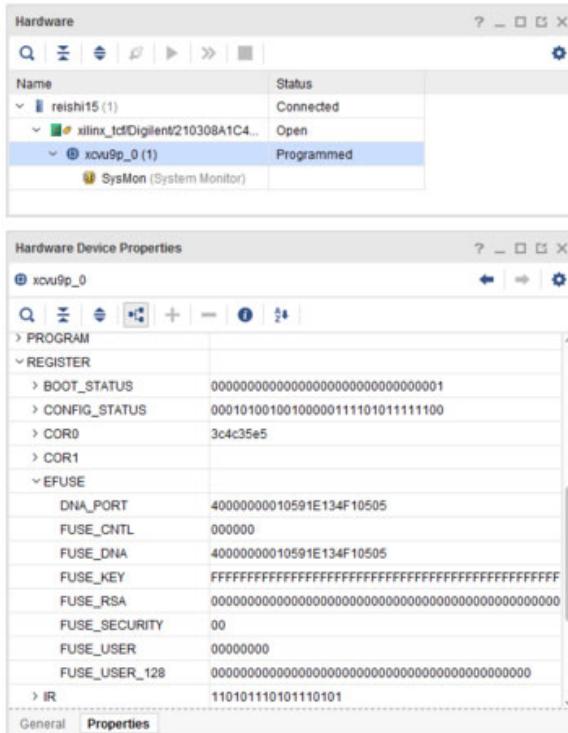
FUSE_DNA: Unique Device DNA

Each UltraScale device has a unique device ID called device DNA that has already been programmed into it by AMD. The FUSE_DNA is not user programmable. UltraScale devices have a 96-bit DNA. You can read the FUSE_DNA by running the following Tcl command in the Vivado Design Suite Tcl Console:

```
get_property [lindex [get_hw_device] 0] REGISTER.EFUSE.FUSE_DNA
```

You can also access the device DNA by viewing the eFUSE registers in the Hardware Device Properties window in Vivado Design Suite as shown in the following figure:

Figure: eFUSE DNA UltraScale and UltraScale+ Devices

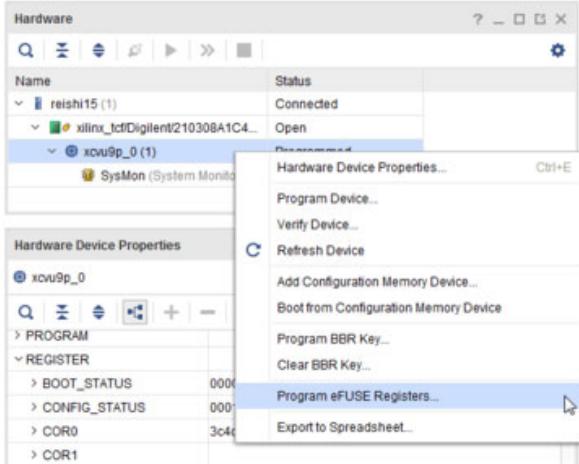


For more information on these features, see the *UltraScale Architecture Configuration User Guide (UG570)*.

Programming the eFUSE Registers

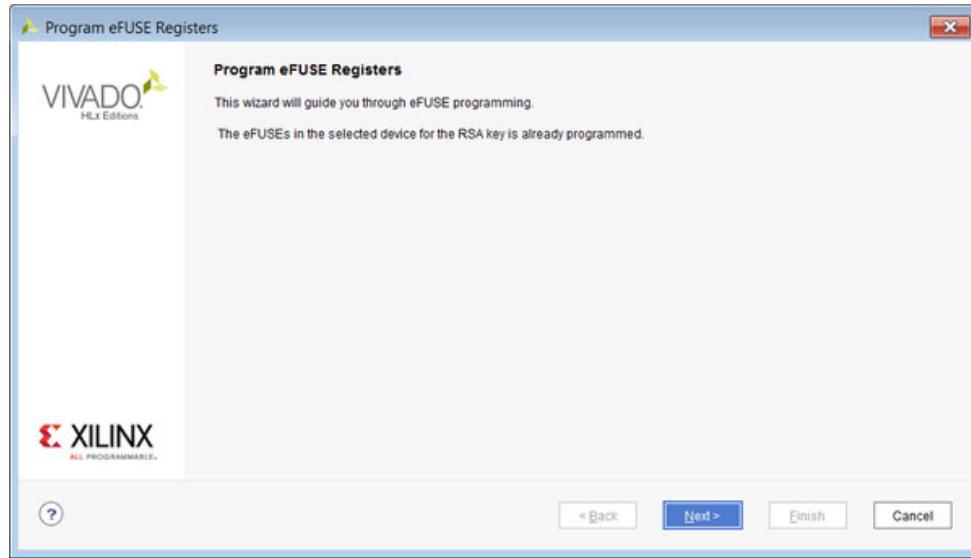
To program the eFUSE registers, right-click the FPGA in the Hardware window and select Program eFUSE Registers.

Figure: Select Program eFUSE Registers UltraScale and UltraScale+ Devices



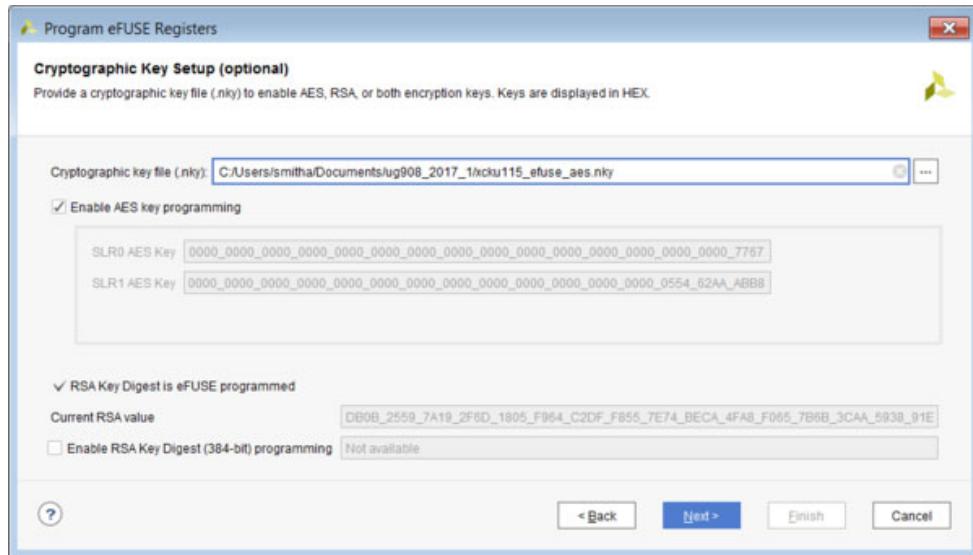
The eFUSE Registers wizard is displayed as shown in the following figure and guides you to set the various options for the eFUSE registers.

Figure: Program eFUSE Registers Wizard



In the AES Key Setup pane, specify the following settings:

Figure: eFUSE Cryptographic Key Setup



In the Cryptographic Key Setup wizard pane, specify these key settings:

Cryptographic file key (.nky)

Specify a .nky file containing eFUSE AES and RSA keys.

AES Key (256-bit)

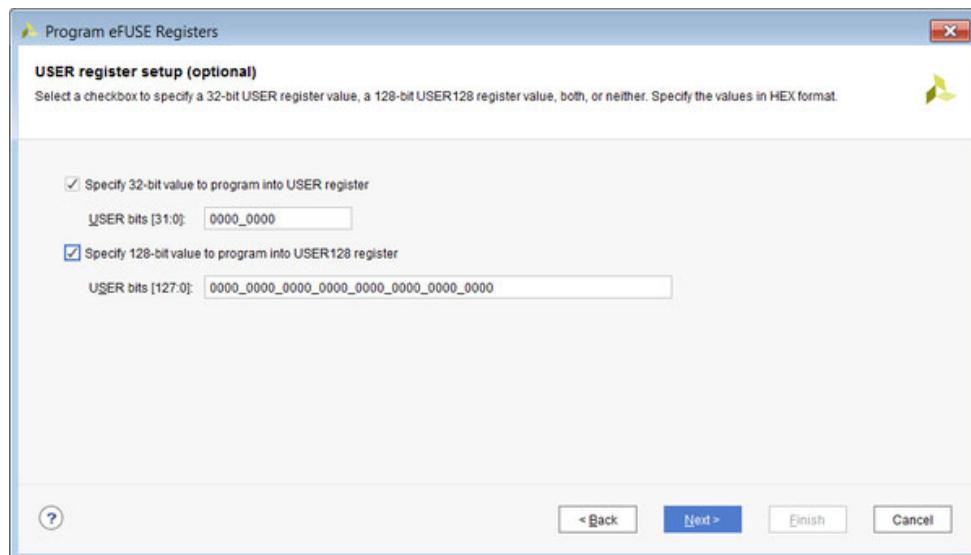
The 256-bit AES eFUSE key read in from specified .nky file used to decrypt loaded encrypted bitstream.

RSA Key Digest (384-bit)

The 384-bit RSA eFUSE key read in from specified .nky file used by RSA.

- In the USER Register Setup wizard pane, specify the 32-bit USER or 128-bit USER register.

Figure: eFUSE USER Register Setup



In the USER register setup pane specify user define register bits. The 32-bit USER (FUSE_USER) and 128-bit USER register (FUSE_USER128) registers are a set of user-defined one-time programmable eFUSE bits. The bits of these registers are cumulatively programmable. This means that if you program only one USER bit in an eFUSE programming session (for example, USER = 0x0000_0001 or bit 0), on subsequent eFUSE programming sessions you can program any of the remaining 0 bits (for example, USER = 0x0000_0002 or bit 1).

After programming the FUSE_USER and FUSE_USER_128 registers, these registers can be read in several ways:

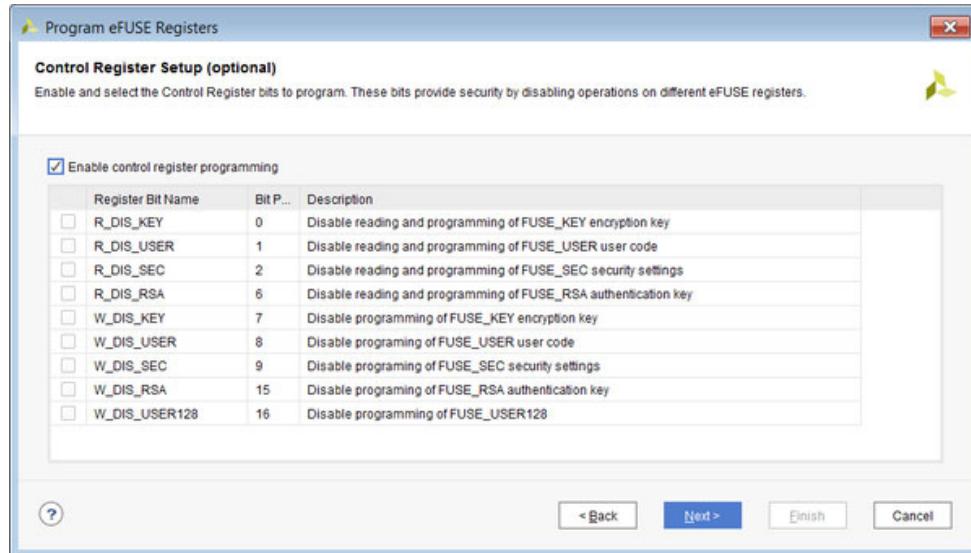
- Using the Tcl command

```
report_property [lindex [get_hw_device] 0] REGISTER.EFUSE.FUSE_USER
report_property [lindex [get_hw_devices] 0] REGISTER.EFUSE.FUSE_USER_128
```

- Through the Vivado Hardware Device Properties window after running a refresh_hw_device operation.

In the Control Register Setup wizard pane, specify the following settings:

Figure: Control Register Setup Pane



In the Control Register Setup pane, specify the eFUSE control settings.

R_DIS_KEY

When set, disables CRC check that verifies the key and programming of the FUSE_KEY encryption key.

R_DIS_USER

When set, disables reading and programming the 32-bit user bits (FUSE_USER).

R_DIS_SEC

When set, disables reading and programming of the security register bits (FUSE_SEC).

R_DIS_RSA

When set, disables reading and programming of the RSA key register (FUSE_RSA).

W_DIS_USER

When set, disables programming of the 32-bit user bits (FUSE_USER).

W_DIS_SEC

When set, disables programming of the security register bits (FUSE_SEC).

W_DIS_RSA

When set, disables programming of the RSA key register (FUSE_RSA).

W_DIS_USER_128

When set, disables programming of the 128-bit user bits (FUSE_USER128).

For more details on the FUSE_SEC register refer to the *UltraScale Architecture Configuration User Guide* ([UG570](#)).

Disabling Control Registers Setup

To disable the Control Register programming, run the following Tcl command:

```
program_hw_devices -control_efuse {20} [lindex [get_hw_devices] $deviceIdx]
```

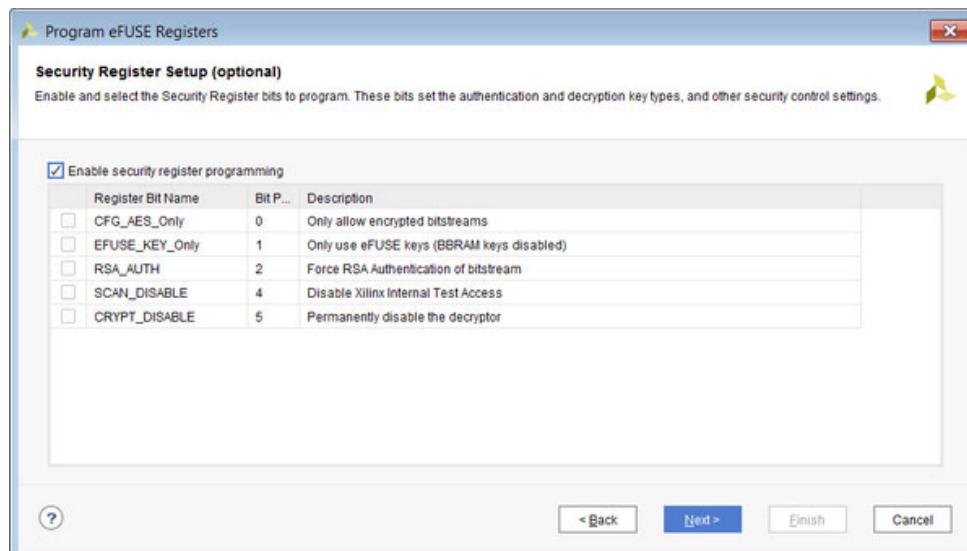
Where \$deviceIdx is set to the index of the UltraScale or UltraScale+ device on which you are disabling the eFUSE Control Register bit programming.

This sets the W_DIS_CNTL bit, which in turn disables further eFUSE Control Register bit programming.

!! Important: If the W_DIS_CNTL bit is programmed, the programming of other eFUSE control register bits is disabled, preventing future edits to the control register of the device.

In the Security Register Setup wizard pane, specify the following settings:

Figure: eFUSE Security Register Setup



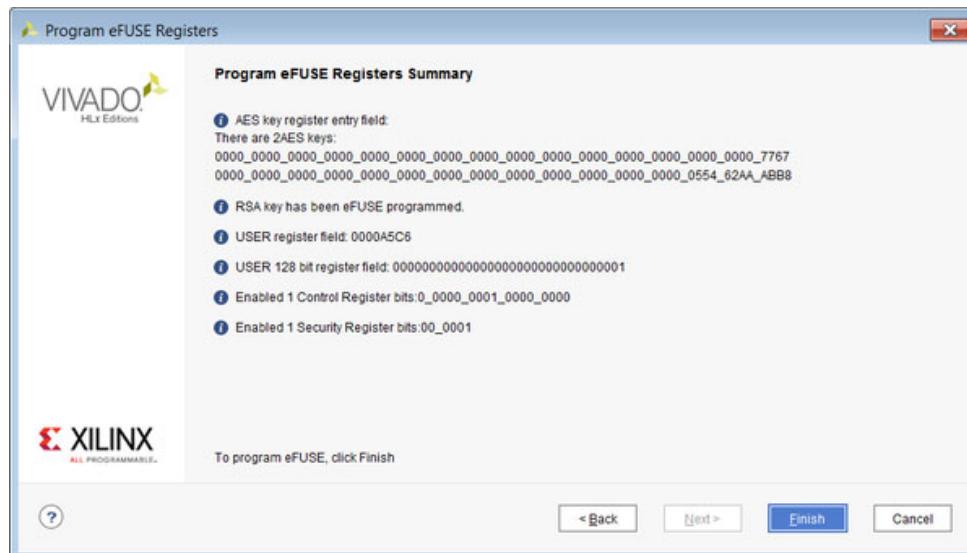
In the Security Register Setup wizard pane specify security control options over the type of bitstreams allowed to load on the FPGA. The FUSE_SEC settings are:

- CFG_AES_Only: When set, only accept encrypted bitstreams.
- EFUSE_KEY_Only: When set, only the eFUSE key can be used for decryption.
- RSA_AUTH: When set, forces RSA Authentication of bitstreams.
- SCAN_DISABLE: When set, disables AMD access to internal test registers.
- CRYPT_DISABLE: When set, permanently disables the decryptor.

For more details on the FUSE_SEC register, refer to the *UltraScale Architecture Configuration User Guide (UG570)*.

Review the eFUSE settings in the Program eFUSE Registers Summary pane.

Figure: Program eFUSE Registers Summary



All bits set in the Program eFUSE Registers wizard panels are shown in this pane. In this pane you see individual bit settings to review the specific programming settings. Carefully review this summary page to ensure every bit that is intended to be programmed is set.

Click Finish to bring up the Program eFUSE confirmation dialog box:

Figure: Program eFUSE Confirmation Dialog



Click OK to program the specified fuse bits.

Disabling the JTAG Interface

To disable the JTAG interface through the eFUSE registers, run the following Tcl command:

```
program_hw_devices -force_efuse -security_efuse {08} [lindex [get_hw_devices]  
$deviceIdx]
```

Where \$deviceIdx is set to the index of the UltraScale or UltraScale+ device that has its JTAG interface disabled.

!! Important: The Tcl command to disable the JTAG interface for 7 series differs from command used previously for UltraScale or UltraScale+ devices. If a 7 series device is used, see the entry for XSK_EFUSEPL_DISABLE_JTAG_CHAIN listed in Answer Record [65110](#).

 **Note:** This programming step should be performed as the last and final step after all desired eFUSE bits are programmed.

!! Important: If the JTAG Disable bit is programmed, the JTAG interface is disabled, preventing future test and configuration access to the device. This bit should only be programmed if JTAG access to the device is no longer required.

Forcing eFUSE Programming

To force any bit to be set regardless of where it is in the register or whether it has been previously programmed the -force_efuse option to program_hw_devices can be used. When used, only basic register boundary checking occurs.

eFUSE NKZ File

In order to capture all eFUSE programming settings in one file, and so make it easier to export eFUSE settings to other eFUSE programmer implementations, and to mass-program the same eFUSE settings into many devices, AMD has defined a file format called NKZ, designated by the file extension .nkz. The NKZ format is a superset of the existing NKY format; that is, NKZ supports all NKY fields and any programmable eFUSE register settings.

eFUSE Export NKZ File

Because you are advised to program eFUSE settings in multiple passes, eFUSE settings are always exported to an externally visible NKZ file which is updated during each eFUSE operation. The intent of this file is to accumulate all eFUSE settings applied to a device, even if they are applied over multiple eFUSE operations. This file is tracked by Vivado so that there is always a single eFUSE export file containing the cumulative eFUSE settings for a device in NKZ format.

If no options are specified, the default name of this file is as follows, where <FUSE_DNA> is the FUSE_DNA register value of the device.

```
export_<FUSE_DNA>.nkz
```

This default file is located in the launch directory of the Vivado IDE. You can change this file by using the -efuse_export_file option on the program_hw_devices Tcl command, as in the

following example:

```
program_hw_devices -user_efuse {1} -efuse_export_file {my_settings.nkz}
```

The Vivado IDE starts using the specified NKZ file for exported eFUSE settings without having to repeatedly specify this file in the Tcl options. Any previously created eFUSE export file is not removed; it contains all the eFUSE settings applied before changing the eFUSE export file name. After all eFUSE operations are performed, all eFUSE settings are stored in the current eFUSE export file.

There is also an option to only export eFUSE settings instead of actually programming the device. This can be useful for creating an NKZ file containing all the desired eFUSE settings without affecting the device. Any eFUSE programming mistakes made in this mode can easily be corrected by removing the eFUSE export file and starting over. This export-only mode can be used with the following Tcl command:

```
program_hw_devices -user_efuse {1} -only_export_efuse
```

This option applies only to each individual command, so the option must be used in every eFUSE operation to keep the device from being programmed. Because eFUSE settings are always exported to an NKZ file regardless of whether or not programming occurs, it is recommended *not* to intermix the programming flow with the export-only flow. Otherwise, the resulting eFUSE export file might contain a mix of eFUSE settings actually programmed into the device and those only exported to the NKZ file, which makes the true state of the eFUSE registers in the device unclear. Using this export-only mode, the same exact eFUSE export file in NKZ format is created as in the programming case, but no programming is performed on the device.

It is also possible to skip programming the encryption keys that are specified in the NKZ file by using the `-skip_program_keys` option on the `program-hw_devices` Tcl command.

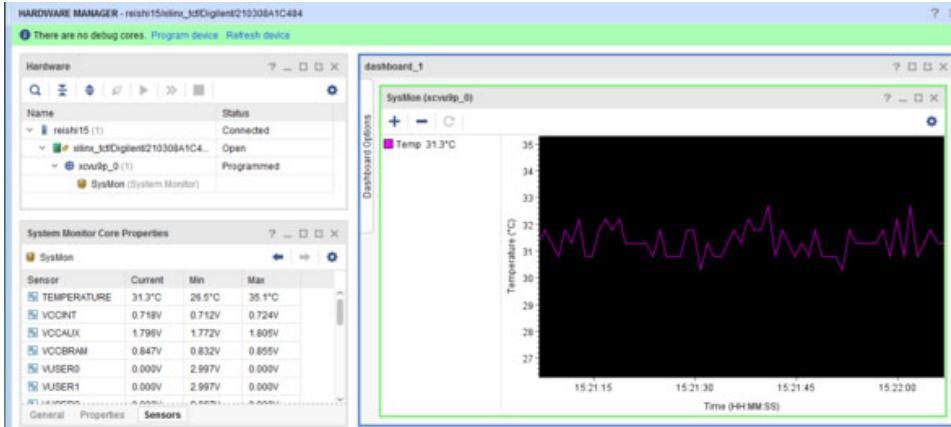
- For 7 series devices, if `-skip_program_keys` is specified, the programming of FUSE_AES is skipped, no check to validate AES keys in the encryption file is performed, and the programming continues.
- For 7 series devices, `-skip_program_keys` must *not* be used if the NKZ also contains a non-zero FUSE_USER setting because FUSE_AES and FUSE_USER values must be programmed together in the 7 series device.
- For UltraScale and UltraScale+ devices, the setting of FUSE_SEC[6] is skipped if `-skip_program_keys` is used when programming obfuscated keys.

System Monitor

The System Monitor (SYSMON) Analog-to-Digital Converter (ADC) measures die temperature and voltage on the hardware device. The SYSMON monitors the physical environment via on-chip temperature and supply sensors. The ADC provides a high-precision analog interface for a range of applications. Refer to the following for more information on specific device architecture.

- UltraScale Architecture System Monitor User Guide ([UG580](#))
- 7 Series FPGAs and Zynq 7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide ([UG480](#))
- Versal Adaptive SoC System Monitor Architecture Manual ([AM006](#))

Figure: System Monitor



The hw_sysmon data is stored in dedicated registers called status registers accessible through the hw_sysmon_reg object. You can get the contents of the System Monitor registers by using the get_hw_sysmon_reg command.

Every device that supports the System Monitor automatically has one or more hw_sysmon objects created when refresh_hw_device is called. When the hw_sysmon object is created, it is assigned a property for all the temperature and voltage registers, and the control registers. On the hw_sysmon object, the values assigned to the temperature and voltage registers are already translated to Celsius/Fahrenheit and Volts.

Although you can use the get_hw_sysmon_reg command to access the hex values stored in registers of a System Monitor, you can also retrieve values of certain registers as formatted properties of the hw_sysmon object. For example, the following code retrieves the TEMPERATURE property of the specified hw_sysmon object rather than directly accessing the hex value of the register:

```
set opTemp [get_property TEMPERATURE [lindex [get_hw_sysmons] 0]]
```

Complete list of all the System Monitor commands can be found in [Description of hw_sysmon Tcl Commands](#).

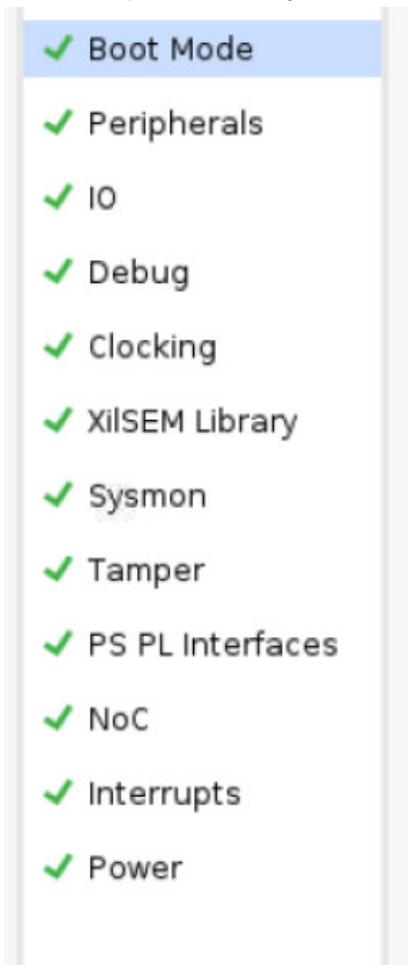
System Monitor for Versal Devices

Unlike previous architectures, the System Monitor used on AMD Versal™ devices can display a larger number of on die sensors. Before using System Monitor on Versal devices, you must select the sensors to be measured in the Control, Interface, and Processing System (CIPS) IP under the PS-PMC settings.

Note: If no sensors are selected, or the CIPS is not configured, only device temperature is accessible.

Configuring System Monitor Sensors in Versal CIPS IP

1. Ensure a CIPS is instantiated in the design. For more information on integrating the CIPS IP, see *Control, Interface and Processing System LogiCORE IP Product Guide (PG352)*. If a CIPS is already present in the design, open the Block Design containing the CIPS by clicking Open Block Design under IP integrator in the Flow Navigator. When the Block Design has opened, double click on the CIPS to bring up the IP customization GUI.
2. Click Next and click on the blue box labeled PS PMC to enter the PS-PMC configuration wizard.
3. In the left pane, click Sysmon.



4. The SysMon Configuration appears and SysMon can now be configured. Basic temperature and/or voltage monitoring can be quickly configured by selecting a Common Configuration Template. To select a specific voltage rail for monitoring, click on the On Chip Supply Monitor tab and check the Enable radio button next to the rail to be monitored.

Figure: Configuring System Monitor Sensors in the CIPS IP Customization GUI

The screenshot shows the AMD STAPL Configuration Tool's configuration interface. The top navigation bar has tabs for 'Basic Configuration', 'On Chip Supply Monitor', 'Temperature', and 'External Supply Monitor'. The 'Basic Configuration' tab is active. Below it, there's a dropdown for 'Common Configuration Template' set to 'Custom'. A dropdown for 'What Interface Do you Want To Use?' is set to 'None'. There's a checkbox for 'De-restrict PMBus Commands'. Under 'PMBUS Address', the value is '0x0'. Under 'I2C IO', the value is 'PS MIO 23 .. 25'. Under 'Reference Used by SYSMON', the value is 'Internal'. A 'Custom' section is expanded, showing a 'Voltage Averaging' subsection with a dropdown for 'Number Of Samples for Voltage Averaging' set to 'None'.

Standard Test and Programming Language (STAPL) Programming

Note: STAPL Programming is supported on AMD UltraScale™ /AMD UltraScale+™ FPGAs and AMD Versal™ devices. STAPL Programming is not supported on AMD Zynq™ devices or 7 series FPGAs.

An alternative way to program FPGAs and configuration memory devices is using a standard test and programming language (STAPL) file. The STAPL file generated through AMD Vivado™ Design Suite and AMD Vivado™ Lab Edition contains low-level JTAG instructions and data required to program these devices. Once the file is generated, boundary scan test tools can be used independently of the Vivado IDE.

The general steps to create an STAPL file are as follows:

1. Create a STAPL target.
2. Add devices to the STAPL target.
3. Add operations to the devices in the STAPL chain.
4. Write STAPL files.
5. Close STAPL target.
6. (Optional) Execute STAPL.

Step 4 records the program operations sequentially and stored as a cached file. The cached file is written out to a target destination in step 5. After the file is created, it can be used by boundary scan tools or executed through Vivado Design Suite or Vivado Lab Edition tools.

Creating an STAPL Target

The STAPL target is similar to a live AMD Platform Cable USB or Digilent JTAG cable hardware target. The properties and Tcl commands are all the same, with the main distinction being that the

STAPL target is not an active live cable. Any operations performed on this target do not affect the hardware until the STAPL is executed.

 **Note:** You do not need a cable connected to your system to create an STAPL.

Using the Command Line

Following are the steps needed to create an STAPL target after initially launching Vivado or Vivado Lab Edition:

```
open_hw_manager
connect_hw_server
create_hw_target -stapl my_stapl_target
if {[string length [get_hw_targets -quiet -filter
{IS_OPENED == TRUE}]] > 0} \
{close_hw_target [get_hw_targets * -filter {IS_OPENED == TRUE}]
}
open_hw_target [get_hw_targets -regexp .*my_stapl_target]
current_hw_target
```

The first two commands can be omitted if already connected to a server. When executed, the `create_hw_target` command defines the `my_stapl_target`.

 **Note:** You cannot have two targets with the same name in a session.

Finally, after closing any open target and opening the STAPL target, the `create_hw_target` command is run. As a result, the final command shows the full hardware target handle name of the created `my_stapl_target`.

All standard operations on the target, such as `get_hw_targets` and `open_hw_target` commands, are supported. You can use the `IS_STAPL` hardware target property to distinguish between a live target and an STAPL target. For instance, the following is a sample command line that reads the `IS_STAPL` property from a target named "my_stapl_target."

```
get_property IS_STAPL [get_hw_targets -regexp .*my_stapl_target]
```

Additionally, all the STAPL `hw_targets` created in this session can be displayed by issuing the following command:

```
get_hw_targets -filter {IS_STAPL}
```

To delete the created target, use the `delete_hw_target` command. For instance, by issuing the following command, the `my_stapl_target` is deleted:

```
delete_hw_target [get_hw_targets -regexp .*my_stapl_target]
```

!! Important: When a target is deleted, all the devices created for the target are also deleted. Moreover, a deleted target is also closed if it was previously opened.

Adding Devices to a STAPL Target

After creating the STAPL target, devices can be added to define the STAPL JTAG device chain configuration. The STAPL JTAG device chain configuration should match the target hardware chain to ensure proper STAPL file execution.

 **Note:** It is not recommended to create a STAPL device chain that contains both UltraScale/UltraScale+ FPGAs and Versal Devices as this may lead to unpredictable results. This limitation will be addressed in a later release.

Using the Command Line

To create the chain using the Vivado Tcl mode or the Tcl Console in the Vivado IDE, perform sequential `create_hw_device` operations on an open STAPL target. For instance, to add an `xcvc1902` part followed by an `xcvu095` part, perform the following steps:

```
current_hw_target my_stapl_target
open_hw_target
create_hw_device -part xcvc1902
create_hw_device -part xcvu095
refresh_hw_target
get_hw_devices
```

The first two steps can be skipped in this example if the STAPL is already created and opened. The `create_hw_device` commands in the example define the devices of the JTAG chain, starting with the first device on the chain and onward.

 **Note:** The `create_hw_device` command only creates devices on an open STAPL hardware target.

 **Note:** Devices can only be appended to an STAPL device chain.

To add user-defined devices to the chain, add the `-idcode`, `-irlength`, and `-mask` values and the part type name using the `-part` options. For instance, if you have a part called "my_part" with a JTAG idcode of 1234567, an ir length of 8, mask of ffffffff, create the device as follows:

```
open_hw_target [current_hw_target]
create_hw_device -idcode 01234567 -irlength 8 -mask ffffffff -part my_part
# print IR length for user defined devices
puts [get_property IR_LENGTH [lindex [get_hw_devices -filter {PART == my_part}]
0]]
puts $idcode_hex
close_hw_target
```

 **Note:** The `idcode` for the `create_hw_device` should be a valid device ID code. Silicon vendors typically provide ID code values and IR lengths through device BSDL files.

To see a report of the target and its devices, run the `report_hw_targets` command. The report shows details for all active targets in the system. Use this report to obtain properties of the server, target, and device as follows:

```
report_hw_targets
INFO: Server Property Information: localhost:3121
    CLASS: hw_server
    HOST: localhost
    NAME: localhost:3121
    PORT: 3121
    SID: TCP:localhost:3121
INFO: Target Property Information:
localhost:3121/xilinx_tcf/Xilinx/my_stapl_target
    CLASS: hw_target
    DEVICE_COUNT: 3
    HW_JTAG: 0
    IS_OPENED: 1
    MAX_DEVICE_COUNT: 32
    NAME: localhost:3121/xilinx_tcf/Xilinx/my_stapl_target
    FREQUENCY: 10000000
    TYPE: xilinx_tcf
    TID: jsn-XNC-my_stapl_target
    UID: Xilinx/my_stapl_target
    SVF: 0
    STAPL: 1
    Device: arm_dap_0
    Device: xcvc1902_1
    Device: xcvu095_2
INFO: Device Property Information: arm_dap_0
    CLASS: hw_device
    PART: arm_dap
    ID CODE: 0110101110100000000010001110111
    IR LENGTH: 4
    MASK: 0
    USER_CHAIN_COUNT: 0
    PROGRAMMING FILE:
    PROBES FILE:
    PROGRAMMING STATUS: 0
INFO: Device Property Information: xcvc1902_1
    CLASS: hw_device
    PART: xcvc1902
    ID CODE: 00010100110010101000000010010011
    IR LENGTH: 6
    MASK: 0
    USER_CHAIN_COUNT: 4
    PROGRAMMING FILE:
    PROBES FILE:
    PROGRAMMING STATUS: 0
INFO: Device Property Information: xcvu095_2
    CLASS: hw_device
    PART: xcvu095
    ID CODE: 01110011100010001000010010011
```

```
IR LENGTH: 6
MASK: 0
USER_CHAIN_COUNT: 4
PROGRAMMING FILE:
PROBES FILE:
PROGRAMMING STATUS: 0
```

 **Note:** Adding configuration memory part to devices is not supported in the STAPL chain.

Operations on the STAPL Chain

Once you have created a STAPL chain that reflects all the devices in the right order, you can start adding programming operations to the devices in the STAPL chain.

Using the Command Line

The following can be used to program device in the STAPL chain:

```
create_hw_target -stapl my_stapl_target
open_hw_target
set device0 [create_hw_device -part xcvc1902]
set_property PROGRAM.FILE {my_xcvc1902.pdi} $device0
program_hw_devices $device0
```

Writing STAPL Files

Using the Command Line

To write the STAPL file using the Vivado Tcl mode or the Tcl Console in the Vivado IDE, use the `write_hw_stapl` command.

The STAPL chain direct FPGA/SoC operations are captured in a temporary file. When the `write_hw_stapl` command is called, the temporary file is moved to the filename passed to the command. After the `write_hw_stapl` command is called, the temporary file is reset, and a subsequent programming operation is added to the beginning of the STAPL file sequence.

The following code segment shows the Tcl commands used to create a file named `my_vc1902.stapl` containing the direct programming of an `xcvc1902` device:

```
create_hw_target -stapl my_stapl_target
open_hw_target
set device0 [create_hw_device -part xcvc1902]
set_property PROGRAM.FILE {my_xcvc1902.pdi} $device0
program_hw_devices $device0
current_hw_device [lindex [get_hw_devices] 1]
write_hw_stapl my_vc1902.stapl
close_hw_target
```

In this sample code, the xcvc1902 device is created using the `create_hw_device` command, whose return value is set to a temporary variable called `device0`. This temporary value is used to reference the object when setting the `PROGRAM.FILE` property to the file `my_xcvc1902.pdi` file. Next, the `program_hw_device` command is called using the `device0` reference. When this `program_hw_device` command runs, it creates a temporary STAPL file with the STAPL operations necessary to program the `my_xcvc1902.pdi` file on the xcvc1902. Lastly, the `write_hw_stapl` command takes the temporary file to the final target destination, `my_xcvc1902.stapl`. At this point, the STAPL file creation process is complete, and the target can be closed.

★ Tip: When writing STAPL files, you should first create all the devices for the JTAG chain and perform the programming operations. If you interleave `create_hw_device` commands between programming commands, you produce an output STAPL file with two different chain sequences.

Example of *Incorrect* STAPL File Creation Steps:

```
create_hw_target -stapl my_stapl_target
open_hw_target
set device0 [create_hw_device -part xcvc1902]
set_property PROGRAM.FILE {my_xcvc1902_1.pdi} $device0
# this program command will produce stapl instructions
# which account for only device0 in chain
program_hw_devices $device0
set device1 [create_hw_device -part xcvc1902]
set_property PROGRAM.FILE {my_xcvc1902_2.pdi} $device1
# this program command will produce stapl instructions
# which account for device0 and device1 in chain
program_hw_devices $device1
write_hw_stapl my_bad_xcvc1902.stapl
close_hw_target
```

The first program command only captures the chain definition containing the first device. The second program command includes both devices in the chain when writing out the STAPL instructions. Therefore, if you attempt to play this STAPL file on a chain with two devices, the first programming operations fail because the live chain gets two devices, not one, as the command expected.

To correct this problem, you run the `create_hw_device` commands first. After the chain is completely defined, perform the program operations as follows:

```
create_hw_target -stapl my_stapl_target
open_hw_target
# create device chain first
set device0 [create_hw_device -part xcvc1902]
set device1 [create_hw_device -part xcvc1902]
# program device0
set_property PROGRAM.FILE {my_xcvc1902_1.bit} $device0
program_hw_devices $device0
# program device1
set_property PROGRAM.FILE {my_xcvc1902_2.bit} $device1
```

```
program_hw_devices $device1
write_hw_stapl my_good_xcvc1902.stapl
close_hw_target
```

Executing STAPL Files

Once the STAPL file is created, you can optionally execute the STAPL file through the `stapl_player` application executable, which is part of the Vivado installation. STAPL Player can execute STAPL files generated through the STAPL generation feature and is intended as the validation test tool. The `stapl_player` application is not intended as a general-purpose STAPL execution command; only use STAPL files created through Vivado IDE.

To run a STAPL file, you can run the application using the command on a live target as follows:

```
stapl_player run --help
Usage: stapl_player run [OPTIONS]

Options:
  -i, --input-file FILE  STAPL file to play  [required]
  -h, --hw-server TEXT   Hardware server URL  [default: TCP:localhost:3121]
  -a, --action TEXT      STAPL action to perform  [required]
  --cable-serial TEXT    Serial number of JTAG cable. This option can be used
                        to choose a specific JTAG cable amongst multiple
                        cables connected to the hardware server. [optional]

  --help                  Show this message and exit.
```

Example

```
$stapl_player run --input-file my_vc1902.stapl --hw-server TCP:192.168.0.100:3121
--action PROGRAM

***** AMD stapl-player v2023.2.0
***** Build date : Aug 18 2023-16:35:08
**** Build number : 2023.2.1692398108
** Copyright 2021-2022 Xilinx, Inc. All Rights Reserved.
** Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.
```

In this example, the `PROGRAM` action from `my_vc1902.stapl` file is executed on the hardware server running on host `192.168.0.100`.

If the host machine running the hardware server has multiple JTAG cables, you can use the `list_cables` command of the STAPL player to list the serial numbers of available cables. You can specify the target serial number using the `--cable-serial` option of the `run` command.

Example

```
$ stapl_player list_cables -h 192.168.0.200:3121

***** Xilinx stapl-player v2023.2.0
***** Build date : Aug 18 2023-16:35:08
**** Build number : 2023.2.1692398108
** Copyright 2021-2022 Xilinx, Inc. All Rights Reserved.
** Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.
```

Connecting to the hardware server at 192.168.0.200:3121

Serial numbers of JTAG cables connected to the hardware server on
192.168.0.200:3121 are -
Cable #1 - 492218147875A
Cable #2 - 0000180fe3cf01

In this example, the STAPL player lists the serial numbers of two cables available on 192.168.0.200.

Serial Vector Format (SVF) File Programming

 **Note:** Serial Vector Format (SVF) programming is not supported on AMD Versal™ devices.

An alternative way to program FPGAs and configuration memory devices is through the use of a serial vector format (SVF) file. The SVF file generated through AMD Vivado™ Design Suite and Vivado Lab Edition contains low level JTAG instructions and data required to program these devices. Once the file is generated it can be used by boundary scan test tools independent of the Vivado IDE.

The general steps to create an SVF file are as follows:

1. Create an SVF offline target.
2. Open the created SVF target.
3. Add devices to the target to define the SVF JTAG scan chain.
4. Program FPGAs or configuration memory devices.
5. Write SVF.
6. Close SVF target.
7. (Optional) Execute SVF.

In step 4, the program operations are recorded in sequential order and stored as a cached file. The cached file is written out to a target destination in step 5. After the file is created, it can be used by boundary scan tools or executed through Vivado Design Suite or Vivado Lab Edition tools.

!! Important: The XSVF file format is not supported in Vivado IDE.

Creating an SVF Target

The SVF target is similar to a live AMD Platform Cable USB or Digilent JTAG cable hardware target. The properties and Tcl commands are all the same, with the main distinction being that the SVF target is not an active live cable. Any operations performed on this target do not affect the hardware until the SVF is executed.

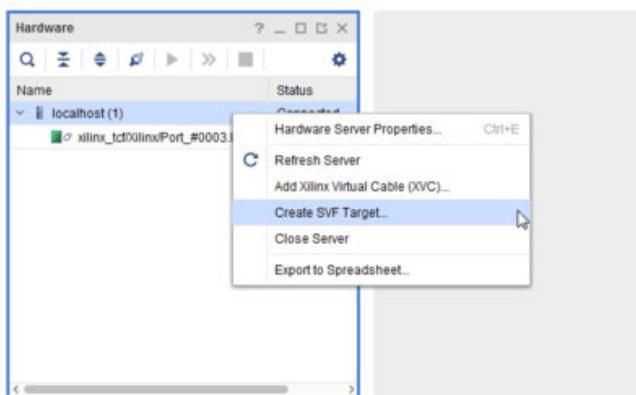
 **Note:** You do not need a cable connected to your system to create an SVF.

Using Vivado IDE

To create an SVF target in the Vivado Hardware Manager, open the Vivado Hardware Manager by either launching Vivado or Vivado Lab Edition. You can create an SVF Target by selecting Tools > Create SVF Target. This automatically opens a server on the local host, and also opens the Create SVF Target dialog as box shown in the following figure.

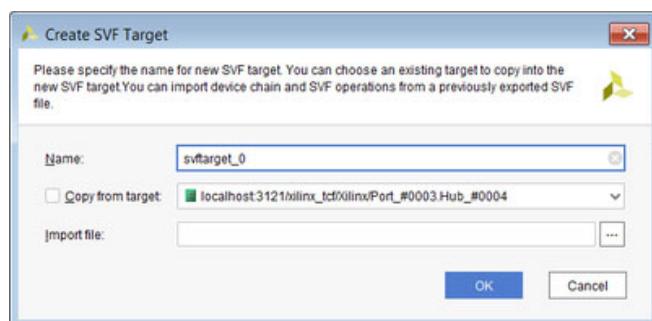
On any available server you can create an offline SVF target as follows:

Figure: Create SVF Target



The Create SVF Target dialog box opens as follows:

Figure: Create SVF Target Dialog Box

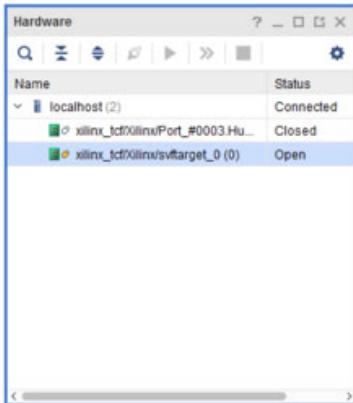


★ Tip: You can copy an existing SVF chain by enabling the Copy from target option. Alternatively, you can specify an SVF file that you created using the Vivado Hardware Manager from an earlier

run of the flow. Vivado IDE saves specifics of the SVF chain, so when it is read back, the SVF chain can be recreated.

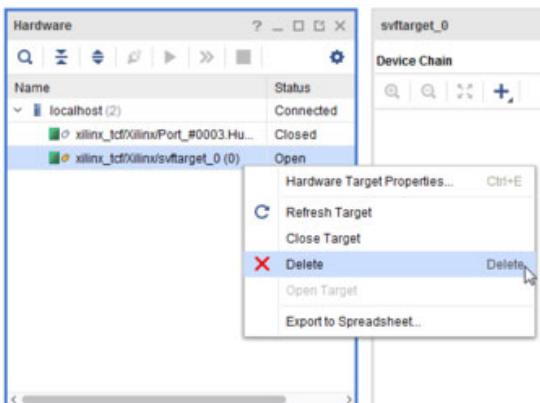
The SVF Target that you created appears Open under your server in the Hardware window in Vivado Hardware Manager.

Figure: SVF Target in Hardware Window



To delete an existing SVF target, right-click the SVF target in the Hardware window and select Delete.

Figure: Delete SVF Target in Hardware Window



!! Important: When you delete a target, all the devices created for that target are also deleted. Moreover, a deleted target is also closed if it was previously opened.

You can also use the Vivado Tcl mode or the Tcl Console in the Vivado IDE to create the SVF Target.

Following are the steps needed to create an SVF target after initially launching Vivado or Vivado Lab Edition:

Using the Command Line

Following are the steps needed to create an SVF target after initially launching Vivado or Vivado Lab Edition:

```
open_hw_manager
connect_hw_server
create_hw_target my_svf_target
if {[string length [get_hw_targets -quiet -filter
{IS_OPENED == TRUE}]] > 0} \
{close_hw_target [get_hw_targets * -filter {IS_OPENED == TRUE}]
} };
open_hw_target [get_hw_targets *my_svf_target]
current_hw_target
```

The first two commands can be omitted if already connected to a server. When executed, the `create_hw_target` command defines the `my_svf_target`.

 **Note:** You cannot have two targets with the same name in a session.

Finally, after closing any open target and opening the SVF target, the `create_hw_target` command is run. As a result, the final command shows the full hardware target handle name of the created `my_svf_target`.

All standard operations on the target, such as `get_hw_targets` and `open_hw_target` commands are supported. You can use the `IS_SVF` hardware target property to distinguish between a live target and an SVF target. For instance, the following is a sample command line that reads the `IS_SVF` property from a target named "my_svf_target".

```
get_property IS_SVF [get_hw_targets -regexp .*my_svf_target]
```

Additionally, all the SVF `hw_targets` created in this session can be displayed by issuing the following command:

```
get_hw_targets -filter {IS_SVF}
```

To delete the created target, use the `delete_hw_target` command. For instance, by issuing the following command, the `my_svf_target` is deleted:

```
delete_hw_target [get_hw_targets -regexp .*my_svf_target]
```

!! Important: When a target is deleted, all the devices created for the target are also deleted.

Moreover, a deleted target is also closed if it was previously opened.

Adding Devices to an SVF Target

After the SVF target is created, devices can be added to it to define the SVF JTAG device chain configuration. The SVF JTAG device chain configuration should match the target hardware chain to ensure proper SVF file execution.

Using Vivado IDE

Click the + button to add Xilinx or Non-Xilinx parts to the SVF chain.

Figure: Add Devices to SVF Target



When you click Add Xilinx Part,... the Add Xilinx Device dialog box opens. You can now choose the appropriate Xilinx device to append to the SVF chain.

Note: Devices can only be appended to an SVF device chain.

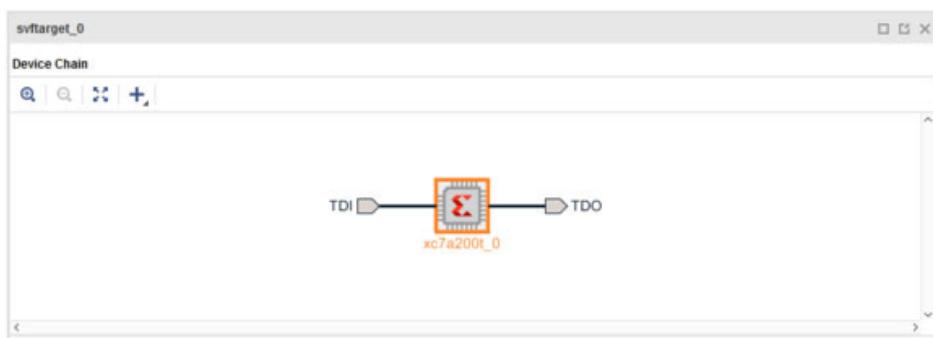
Figure: Add Xilinx Device Dialog Box



Tip: This dialog box is slightly different in Vivado ML Enterprise.

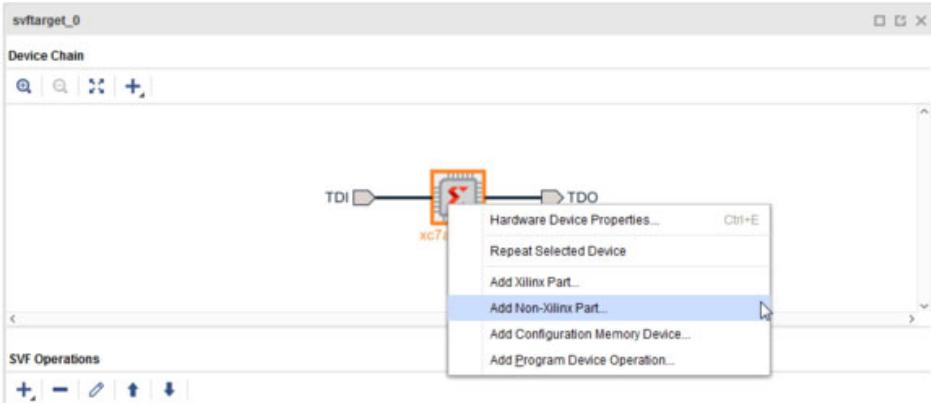
On selecting a Xilinx device and clicking OK, the Xilinx device is added to the SVF chain as shown in the following figure.

Figure: Xilinx Device in SVF Chain



You can also add non-Xilinx parts to the SVF device chain by right-clicking the SVF chain and selecting Add Non-Xilinx Part as follows:

Figure: Adding a Non-Xilinx Part



This opens the Add Non-Xilinx Device dialog box as follows:

Figure: Add Non-Xilinx Device Dialog Box



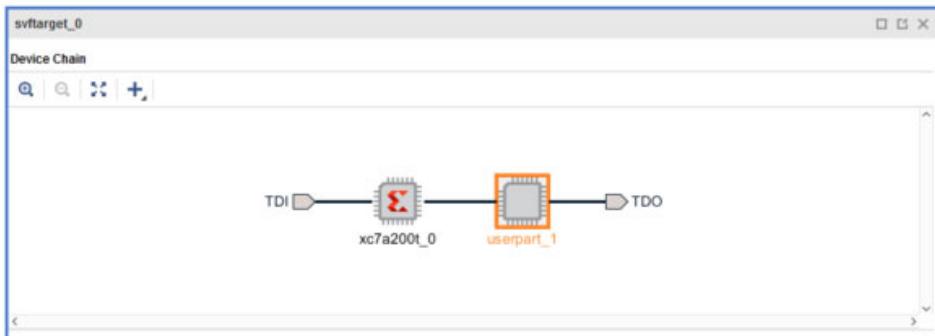
Fill in the dialog box as follows:

- The Part Name can be any part name you choose.
- The ID code is a Hex value that represents a valid device ID code.
- The IR length is a decimal numerical value that represents the Instruction Register length.
- The Mask is a Hex bit mask value.

Note: The ID code, IR Length, and Mask values are typically provided by silicon vendors through device BSDL files.

Click OK, and the non-Xilinx part is added to the SVF device chain.

Figure: Non-Xilinx Device in SVF Chain



Using the Command Line

To create the chain using the Vivado Tcl mode or the Tcl Console in the Vivado IDE, perform sequential `create_hw_device` operations on an open SVF target. For instance, to add an `xcku9p` part followed by an `xcvu095` part perform the following steps:

```
current_hw_target my_svf_target
open_hw_target
create_hw_device -part xcku9p
create_hw_device -part xcvu095
refresh_hw_target
get_hw_devices
```

In this example, the first two steps can be skipped if the SVF is already created and opened. The `create_hw_device` commands in the example define the devices of the JTAG chain starting with the first device on the chain and onward.

Note: The `create_hw_device` command only creates devices on an open SVF hardware target.
To add user defined devices to the chain, add the `-idcode`, `-irlength`, and `-mask` values along with the part type name using the `-part` options. For instance, if you have a part called "my_part" with a JTAG idcode of 1234567, an ir length of 8, mask of ffffffff, create the device as follows:

```
open_hw_target [current_hw_target]
create_hw_device -idcode 01234567 -irlength 8 -mask ffffffff -part my_part
# print IR length for user defined devices
puts [get_property IR_LENGTH [lindex [get_hw_devices -filter {PART == my_part}] 0]]
puts $idcode_hex
close_hw_target
```

Note: The idcode for the `create_hw_device` should be a valid device ID code. ID code values and IR lengths are typically provided by silicon vendors through device BSDL files.

To see a report of the target and its devices, run the `report_hw_targets` command. The report shows details for all active targets in the system. Use this report to obtain properties of the server, target, and device as follows:

```

report_hw_targets
INFO: Server Property Information: localhost:3121
    CLASS: hw_server
    HOST: localhost
    NAME: localhost:3121
    PORT: 3121
    SID: TCP:localhost:3121
INFO: Target Property Information: localhost:3121/xilinx_tcf/Xilinx/my_svf_target
    CLASS: hw_target
    DEVICE_COUNT: 3
    HW_JTAG: 0
    IS_OPENED: 1
    MAX_DEVICE_COUNT: 32
    NAME: localhost:3121/xilinx_tcf/Xilinx/my_svf_target
    FREQUENCY: 10000000
    TYPE: xilinx_tcf
    TID: jsn-XNC-my_svf_target
    UID: Xilinx/my_svf_target
    SVF: 1
    Device: xcku9p_0
    Device: xcuv095_1
    Device: my_part_2

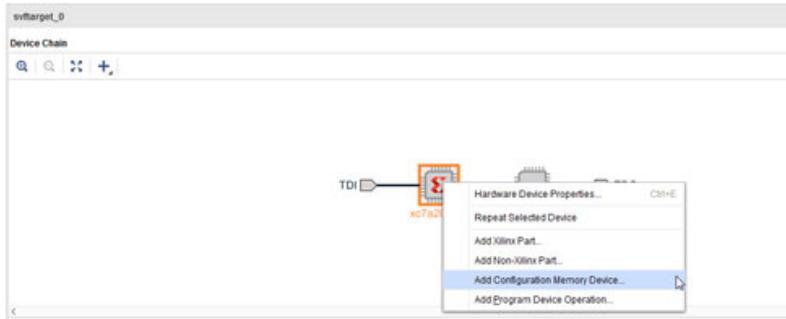
```

Adding Configuration Memory Parts to Xilinx Devices

Using Vivado IDE

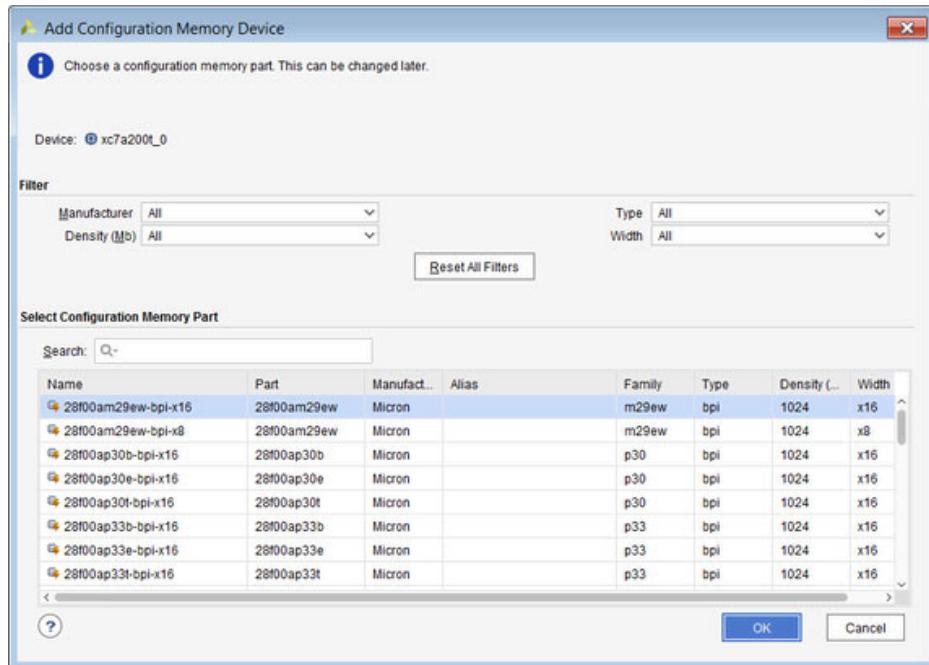
When you right-click on a Xilinx device part in an SVF chain, you have the option of creating and associating a Configuration Memory device to it.

Figure: Adding a Configuration Memory Device



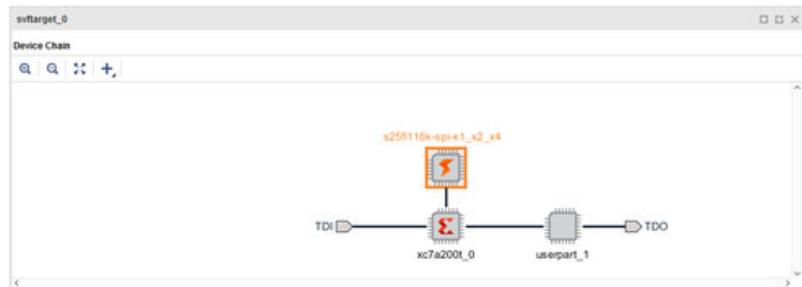
This opens up the Add Configuration Memory Device dialog box as follows:

Figure: Add Configuration Memory Device Dialog Box



Select the appropriate memory device and click OK. The device is associated with the Xilinx device and appears in the SVF device chain as follows:

Figure: Configuration Memory Device in SVF Chain



Using Command Line

To create and associate a configuration memory device using the Vivado Tcl mode or the Tcl Console in the Vivado IDE, use the `create_hw_cfgmem` Tcl command as follows:

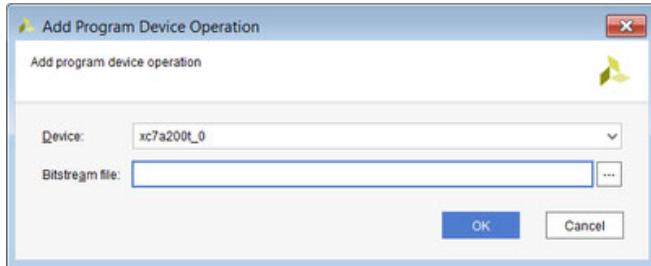
```
create_hw_cfgmem -hw_device [lindex [get_hw_devices xc7a200t_0] 0] [lindex [get_cfgmem_parts {s25fl116k-spi-x1_x2_x4}] 0]
```

Operations on the SVF Chain

Once you have created an SVF chain that reflects all the devices and their configuration memory in the right order, you can start adding programming operations to the devices in the SVF chain. For example, you can right-click on the AMD a200t device in the chain and select Add Program Device Operation which brings up the Add Program Device Operation dialog box as follows. Specify

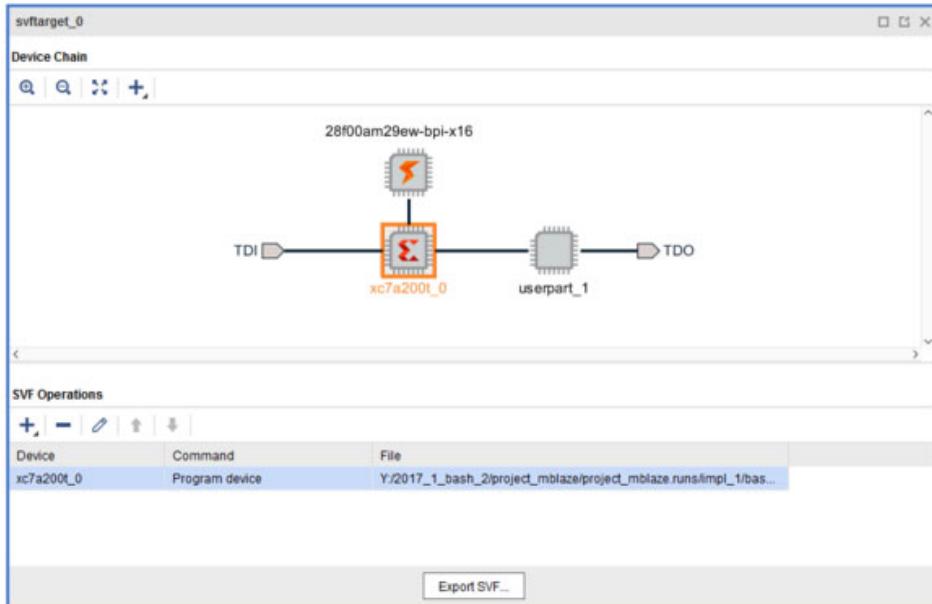
the bitstream file to program the device with.

Figure: Add Program Device Operation Dialog Box



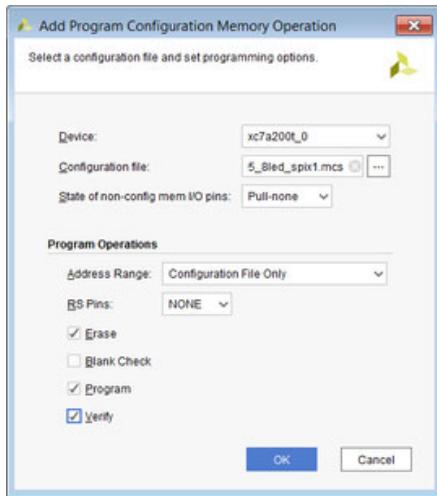
After you click OK, the program device operation is listed at the bottom of the SVF Operations window.

Figure: SVF Operations Window



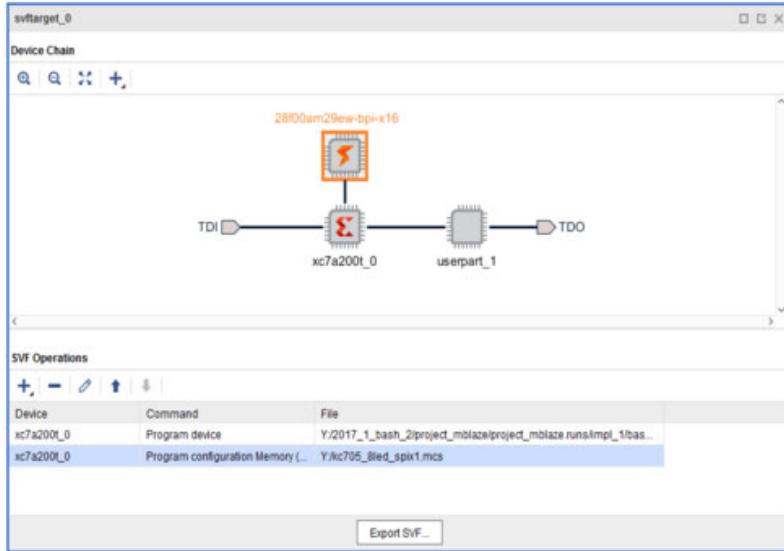
Similarly, you can program the configuration memory device by right-clicking on the memory device and selecting Add Program Configuration Memory which brings up the Add Program Configuration Memory dialog box as follows. Specify the configuration file to program the memory device with. You can also select additional programming options for memory devices like Erase, Blankcheck, and Verify.

Figure: Add Program Configuration Memory Dialog Box



After you click OK, the program configuration memory device operation is listed at the bottom of the SVF Operations window.

Figure: Configuration Memory Device in SVF Operations Window

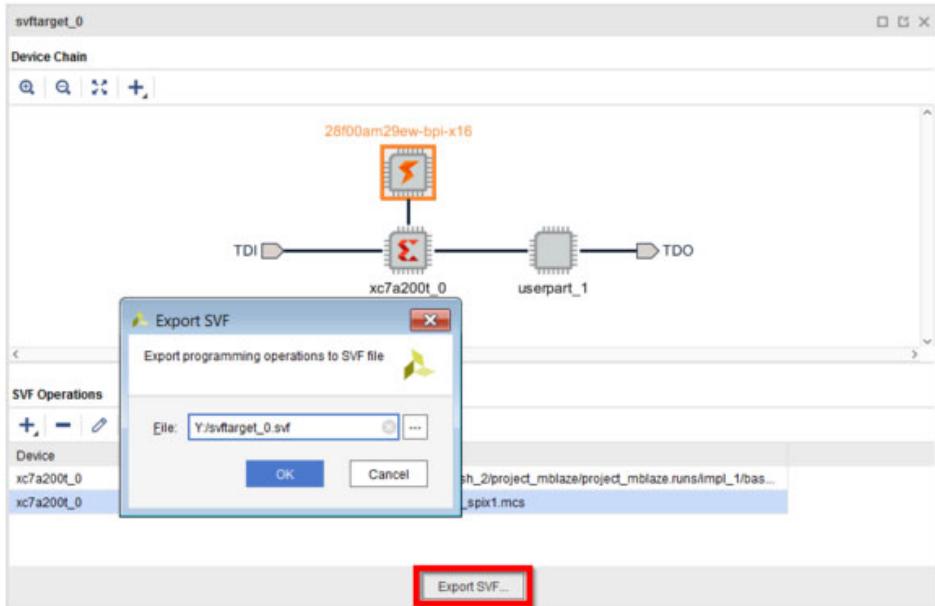


Writing SVF Files

Using the Vivado IDE

The SVF chain setup and its operations can be saved to a file by clicking Export SVF at the bottom of the SVF Operations window as follows.

Figure: Exporting the SVF Chain Setup



!! Important: You can recreate an existing SVF chain by specifying an SVF file that you created with the Vivado Hardware Manager from an earlier run of the flow. Vivado IDE saves specifics of the SVF chain to the file, so that when it is read back, the SVF chain can be recreated.

Using the Command Line

To write the SVF file using the Vivado Tcl mode or the Tcl Console in the Vivado IDE use the `write_hw_svf` command.

The SVF chain, direct FPGA and indirect flash programming operations are captured in a temporary file. When the `write_hw_svf` command is called, the temporary file is moved to the filename passed to the command. After the `write_hw_svf` command is called, the temporary file is reset, and a subsequent programming operation is added to the beginning of the SVF file sequence.

The following code segment shows the Tcl commands used to create a file named `my_xcku9p.svf` containing the direct programming of a `xcku9p` device:

```
create_hw_target my_svf_target
open_hw_target
set device0 [create_hw_device -part xcku9p]
set_property PROGRAM.FILE {my_xcku9p.bit} $device0
program_hw_devices $device0
write_hw_svf my_xcku9p.svf
close_hw_target
```

In this sample code the `xcku9p` device is created using `create_hw_device` command whose return value is set to a temporary variable called `device0`. This temporary value is used to reference the object when setting the `PROGRAM.FILE` property to the file `my_xcku9p.bit` file. Next, the `program_hw_device` command is called using the `device0` reference. When this `program_hw_device` command runs, it creates a temporary SVF file with the SVF operations necessary to program the `my_xcku9p.bit` file on the `xcku9p`. Lastly, the `write_hw_svf` command

takes the temporary file and moves it to the final target destination, `myxcku9p.svf`. At this point, the SVF file creation process is complete and the target can be closed.

★ **Tip:** When writing STAPL files, you should first create all the devices for the JTAG chain and perform the programming operations. If you happen to interleave `create_hw_device` commands in between programming commands, you produce an output SVF file that has two different chain sequences.

Example of *Incorrect* SVF File Creation Steps:

```
create_hw_target my_svf_target
open_hw_target
set device0 [create_hw_device -part xcku9p]
set_property PROGRAM.FILE {my_xcku9p1.bit} $device0
# this program command will produce SVF instructions
# which account for only device0 in chain
program_hw_devices $device0
set device1 [create_hw_device -part xcku9p]
set_property PROGRAM.FILE {my_xcku9p2.bit} $device1
# this program command will produce SVF instructions
# which account for device0 and device1 in chain
program_hw_devices $device1
write_hw_svf my_bad_xcku9p.svf
close_hw_target
```

The first program command only captures the chain definition containing the first device. The second program command includes both devices in the chain when writing out the SVF instructions. Therefore, if you attempt to play this SVF file on a chain with two devices, the first programming operations fail because the live chain gets two devices, not one as the command expected. To correct this problem, you run the `create_hw_device` commands first. After the chain is completely defined, perform the program operations as follows:

Example of *Correct* SVF File Creation Steps

```
create_hw_target my_svf_target
open_hw_target
# create device chain first
set device0 [create_hw_device -part xcku9p]
set device1 [create_hw_device -part xcku9p]
# program device0
set_property PROGRAM.FILE {my_xcku9p1.bit} $device0
program_hw_devices $device0
# program device1
set_property PROGRAM.FILE {my_xcku9p2.bit} $device1
program_hw_devices $device1
write_hw_svf my_good_xcku9p.svf
close_hw_target
```

Executing SVF Files

Once the SVF file is created, you can optionally execute the SVF file through Vivado IDE. Vivado IDE can execute SVF files generated through the SVF generation feature and is intended as the validation test tool. The `execute_hw_svf` command is not intended as a general purpose SVF execution command; take care to only use SVF files created through Vivado IDE.

To run a `svf` command you run the command on an open live target as follows:

```
execute_hw_svf my_file.svf
INFO: [Labtoolstcl 44-548] Creating JTAG TCL script from SVF file
INFO: [Labtoolstcl 44-549] Re-opening target in JTAG mode
INFO: [Labtoolstcl 44-551] Sourcing JTAG TCL script: my_file.tcl
Pass: SVF Execution completed with no errors
INFO: [Labtoolstcl 44-550] Restoring target to original mode
INFO: [Labtoolstcl 44-570] Execute SVF completed successfully
```

In this example, the file `my_file.svf` is executed. As part of the execution flow, the input SVF file is converted via HW_JTAG Tcl operations into a temporary file. After creating this Tcl code, the file is sourced to execute the converted SVF instructions. To see the JTAG_TCL operations, you can run the `execute_hw_svf` command using the `-verbose` option. Once the command completes, you see the error at the instruction where the execution failed or a "Pass" message at the end of the message log.

★ Tip: Vivado supports SVF execution for SVF files under 500 MB. To execute SVF files that exceed 500 MB in size, use a third party SVF player.

Debugging the Design

Debugging an FPGA or adaptive SoC design is a multi-step, iterative process. Like most complex problems, it is best to break the FPGA or adaptive SoC design debugging process down into smaller parts by focusing on getting smaller sections of the design working one at a time rather than trying to get the whole design to work at once. Iterating through the design flow by adding one module at a time and getting it to function properly in the context of the whole design is one example of a proven design and debug methodology. You can use this design and debug methodology in any combination of the following design flow stages:

- RTL-level design simulation
- Post-implemented design simulation
- In-system debugging

RTL-Level Design Simulation

The design can be functionally debugged during the simulation verification process. AMD provides a full design simulation feature in the AMD Vivado™ IDE. The AMD Vivado™ design simulator can be used to perform RTL simulation of your design. The benefits of debugging your design in an RTL-

level simulation environment include full visibility of the entire design and ability to quickly iterate through the design/debug cycle. The limitations of debugging your design using RTL-level simulation includes the difficulty of simulating larger designs in a reasonable amount of time in addition to the difficulty of accurately simulating the actual system environment. For more information about using the AMD Vivado™ simulator, refer to the *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#)).

Post-Implemented Design Simulation

The Vivado simulator can also be used to simulate the post-implemented design. One of the benefits of debugging the post-implemented design using the Vivado simulator includes having access to a timing-accurate model for the design. The limitations of performing post-implemented design simulation include those mentioned in the previous section: long runtimes and system model accuracy.

In-System Logic Design Debugging

The Vivado Design Suite also includes a logic analysis feature that enables you to perform in-system debugging of the post-implemented design in an FPGA or adaptive SoC. The benefits of debugging your design in-system include debugging your timing-accurate, post-implemented design in the actual system environment at system speeds. The limitations of in-system debugging include somewhat lower visibility of debug signals compared to using simulation models and potentially longer design/implementation/debug iterations, depending on the size and complexity of the design. In general, the Vivado tool provides several different ways to debug your design. You can use one or more of these methods to debug your design, depending on your needs. In-System Logic Design Debugging Flows focuses on the in-system logic debugging capabilities of the Vivado Design Suite.

Related Information

[In-System Logic Design Debugging Flows](#)

In-System Serial I/O Design Debugging

To enable in-system serial I/O validation and debugging, the Vivado Design Suite includes a serial I/O analysis feature. This allows you to measure and optimize your high-speed serial I/O links in your FPGA-based system. The Vivado serial I/O analyzer features are designed to help you address a range of in-system debug and validation problems from simple clocking and connectivity issues to complex margin analysis and channel optimization issues. The main benefit of using the Vivado serial I/O analyzer over some other external instrumentation techniques is that you are measuring the quality of the signal after the receiver equalization has been applied to the received signal. This ensures that you are measuring at the optimal point in the TX-to-RX channel thereby ensuring real and accurate data.

The Vivado tool provides the means to generate the design used to exercise the gigabit transceiver endpoints and the runtime software to take measurements and help you optimize your high-speed serial I/O channels. Serial I/O Hardware Debugging Flows guides you through the process of

generating the IBERT design. Debugging the Serial I/O Design in Hardware guides you through the use of the runtime Vivado serial I/O analyzer feature.

Related Information

[Serial I/O Hardware Debugging Flows](#)
[Debugging the Serial I/O Design in Hardware](#)

In-System Logic Design Debugging Flows

The AMD Vivado™ tool provides many features to debug a design in-system in an actual hardware device. The in-system debugging flow has three distinct phases:

1. **Probing phase:** Identifying what signals in your design you want to probe and how you want to probe them.
2. **Implementation phase:** Implementing the design that includes the additional debug IP that is attached to the probed nets.
3. **Analysis phase:** Interacting with the debug IP contained in the design to debug and verify functional issues.

This in-system debug flow is designed to work using the iterative design/debug flow described in the previous section. If you choose to use the in-system debugging flow, it is advisable to get a part of your design working in hardware as early in the design cycle as possible. The rest of this chapter describes the three phases of the in-system debugging flow and how to use the Vivado logic debug feature to get your design working in hardware as quickly as possible.

Probing the Design for In-System Debugging

The probing phase of the in-system debugging flow is split into two steps:

1. Identifying what signals or nets you want to probe.
2. Deciding how you want to add debug cores to your design.

In many cases, the decision you make on what signals to probe or how to probe them can affect one another. It helps to start by deciding if you want to manually add the debug IP component instances to your design source code (called the HDL instantiation probing flow) or if you want the Vivado tool to automatically insert the debug cores into your post-synthesis netlist (called the netlist insertion probing flow). The following table describes some of the advantages and trade-offs of the different debugging approaches.

Table: Debugging Strategies

Debugging Goal	Recommended Debug Programming Flow
Identify debug signals in the HDL source code while retaining flexibility to	Use mark_debug property to tag signals for debugging in HDL. Use the Set up Debug wizard to guide you through the Netlist Insertion probing flow.

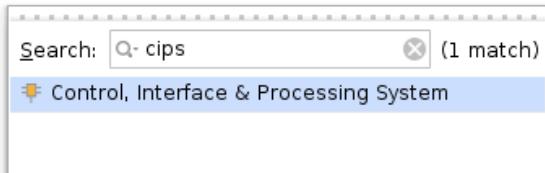
Debugging Goal	Recommended Debug Programming Flow
enable/disable debugging later in the flow.	
Identify debug nets in synthesized design netlist without having to modify the HDL source code.	Use the Mark Debug right-click menu option to select nets for debugging in the synthesized design netlist. Use the Set up Debug wizard to guide you through the Netlist Insertion probing flow.
Automated debug probing flow using Tcl commands.	Use <code>set_property</code> Tcl command to set the <code>mark_debug</code> property on debug nets. Use Netlist Insertion probing flow Tcl commands to create debug cores and connect to them to debug nets.
Explicitly attach signals in the HDL source to an ILA debug core instance.	Identify HDL signals for debugging. Use the HDL Instantiation probing flow to generate and instantiate an Integrated Logic Analyzer (ILA) core and connect it to the debug signals in the design.

Versal In-System Debugging

While the AMD Versal™ adaptive SoC architecture differs from previous FPGA architectures and uses different debug IP and infrastructure to connect the debug cores in-system, the debugging flow is similar in many ways to previous FPGA architectures. There are a few differences to be aware of, as detailed in this section.

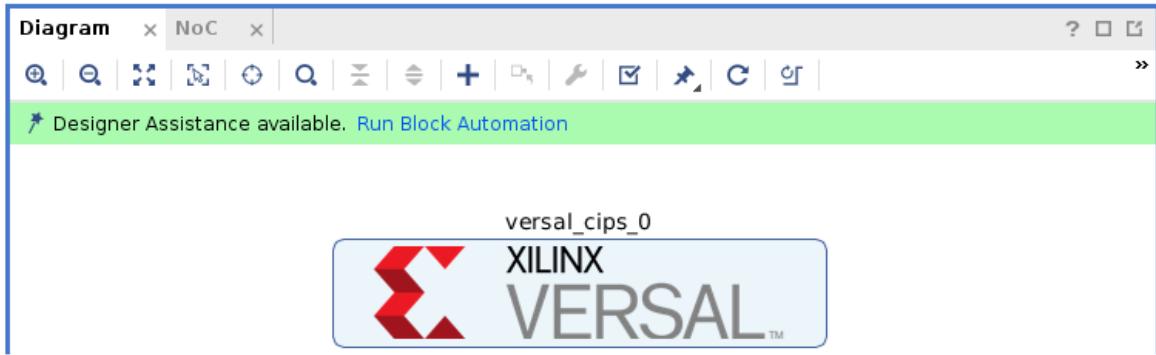
Adding a Control, Interface, and Processing System (CIPS)

1. If the design does not already contain a block design, click Create Block Design under the IP integrator category in the Flow Navigator to create one.
2. Click + to add new IP to the IP integrator Canvas and type `cips` to search for the CIPS IP (as shown in the following figure). Once found, double click to add it to the IP integrator canvas.



3. Once added, a green bar displays near the top of the tool bar indicating that designer assistance is available. It is not necessary to run block automation unless there is a need to

configure additional options.



4. Click the button to validate the block design, click the button to save the block design.
5. Generate the HDL wrapper for the Block Design by returning to the Project Manager, right-clicking on the newly created block design, and selecting Create HDL Wrapper.
6. Once created, ensure this block design is instantiated as part of the design.
7. Proceed using either the Netlist Insertion Debug Probing Flow, HDL Instantiation Debug Probing Flow, or the IP integrator Debug Flow. If the design contains any debug cores, the AXI4 Debug Hub is automatically added to the netlist during opt_design and the debug cores are connected automatically.

Note: By default, this block design does not have any input or output ports unless additional IPs are added requiring I/O.

Note: For most applications, it is not necessary to manually connect instantiated debug cores to the AXI4 Debug Hub. During opt_design an instance of the AXI4 Debug Hub and NoC is inserted into the netlist, and the connections between the debug cores, debug hub, and CIPS core are automatically stitched.

Note: For more information on design creation, simulation, and debug using the CIPS core, see *Control, Interface and Processing System LogiCORE IP Product Guide* ([PG352](#)).

AXI4 Debug Hub

Versal devices use the AXI4 Debug Hub, which is similar to the debug hub used in previous architectures and provides connectivity between the Versal Control, Interface, and Processing (CIPS) IP and the debug cores in the design. In previous architectures Vivado inserts this core automatically without user intervention. It is also possible to manually instantiate the AXI4 Debug Hub. Manual instantiation is only recommended if using DFX, or the design's addressing scheme that requires setting a manual address for the AXI4 Debug Hub.

AXI4-Stream ILA

Versal devices use the AXI4-Stream ILA that includes both ILA and System-ILA functionality. The AXI4-Stream ILA also provides two choices for trace memory: Block RAM (BRAM) and UltraRAM (URAM).

AXI4 Debug Hub Connectivity

To use the AMD Vivado™ debug cores, the design must contain an AXI4 Debug Hub. The AXI4 Debug Hub connects an AXI4 interface of the CIPS with the AXI4-Stream interface. The interface connects to Vivado debug cores, which includes the following types of cores:

- AXI4-Stream Integrated Logic Analyzer (AXIS-ILA)
- AXI4-Stream Virtual Input/Output (AXIS-VIO)
- PCI Express® Link Debugger

Table: AXI4 Debug Hub Auto-Insertion

AXI4 Debug Hub Connectivity	Debug Flow
Automatic AXI4 Debug Hub post-synthesis netlist insertion and connection.	<p>This method is recommended for most use cases as it provides the most flexibility.</p> <ol style="list-style-type: none">1. During opt_design the Vivado debug flow detects if the design contains any debug cores that require AXI4 Debug Hub connectivity. The Vivado debug flow also detects if the design contains an instance of the Control, Interface, and Processing (CIPS) IP.2. The Vivado debug flow inserts an instance of the AXI4 Debug Hub into the synthesized netlist and automatically connects it to the debug cores used in the design. <hr/> <p> Note: This method cannot be used on designs that use Dynamic Function eXchange (DFX).</p>
Manual AXI4 Debug Hub instantiation, automatic post-synthesis netlist debug core connection.	<p>This method should be used when it is desired to manually assign the address used by the AXI4 Debug Hub or when Dynamic Function eXchange (DFX) is used. In this case the design should have a manually instantiated AXI4 Debug Hub with a connection to an AXI4 master from the Control, Interface, and Processing (CIPS) IP.</p> <ol style="list-style-type: none">1. During opt_design the Vivado debug flow detects if the design contains any debug cores that require AXI4 Debug Hub Connectivity. The Vivado debug flow also detects if the design contains an

AXI4 Debug Hub Connectivity	Debug Flow
	<p>instance of the Control, Interface, and Processing (CIPS) IP.</p> <p>2. The Vivado debug flow locates the manually added AXI4 Debug Hub. This instance of the AXI4 Debug Hub is replaced with an AXI4 Debug Hub configured with a suitable number of AXI4-Stream interfaces to connect to each debug core used in the design.</p> <hr/> <p> Note: The AXI4 Debug Hub that is replaced by the Vivado debug flow in the previous steps retains the user specified address and properties.</p> <hr/>
Manual AXI4 Debug Hub instantiation, manual debug core connection.	<p>This method should be used when it is desired to manually define all connectivity between the AXI4 Debug Hub, CIPS, and all debug cores in the design. This method can also be used when the design uses Dynamic Function eXchange (DFX).</p> <ol style="list-style-type: none"> When building the design, one, or more instances of the AXI4 Debug Hub are added to the design with connectivity to an appropriate AXI4 master on the CIPS IP. The AXI4 Debug Hub should be customized to include the <i>exact</i> number of AXI4-Stream interfaces as there are debug cores in the design. Each debug core in the design should have the option to enable AXI4-Stream ports for manual connectivity turned on. It is the user's responsibility to connect each debug core's AXI4-Stream master and slave to a corresponding slave and master on the AXI4 Debug Hub.

Manually Connecting Versal Debug Cores

The Versal AXI4 Debug Hub IP and debug cores (such as AXI4-Stream ILA, AXI4-Stream VIO) offer the option to manually define the connectivity between the debug cores and the AXI4 Debug Hub.

This option is not required for most designs.

To enable manual connectivity to between the AXI4 Debug Hub and a debug core:

1. Generate and instantiate an instance of the AXI4 Debug Hub IP and connect it to the Control, Interface, and Processing (CIPS) IP in the design using the PMC.
 2. Customize the AXI4 Debug Hub IP and set the Number of Debug Cores to the exact number of debug cores to be manually connected. For each debug core an AXI4-Stream master and slave appear on the AXI4 Debug Hub IP.
-
-  **Note:** Unconnected AXI4-Stream ports on the AXI4 Debug Hub can cause errors during opt_design.
-
3. In the IP customization interface for the debug cores to be manually connected, select Enable AXI4-Stream Interfaces for Manual Connection to AXI Debug Hub. AXI4-Stream master and slave ports appear on the debug core.
 4. Connect each AXI4-Stream master and slave port on the debug core to those on the AXI4 Debug Hub. The debug core also includes aclk and aresetn ports, which should be connected to the same clock and reset connected to the AXI4 Debug Hub.

Using the Netlist Insertion Debug Probing Flow

Insertion of debug cores in the Vivado tool is presented in a layered approach to address different needs of the diverse group of Vivado users:

- The highest level is a simple wizard that creates and configures Integrated Logic Analyzer (ILA) cores automatically based on the selected set of nets to debug.
- The next level is the main Debug window allowing control over individual debug cores, ports, and their properties. The Debug window can be displayed when the Synthesized Design is open by selecting the Debug layout from the Layout Selector or the Layout menu or can be opened directly using Window > Debug.
- The lowest level is the set of Tcl XDC debug commands that you can enter manually into an XDC constraints file or replay as a Tcl script.

You can also use a combination of modes to insert and customize debug cores.

Marking HDL Signals for Debug

You can identify signals for debugging at the HDL source level prior to synthesis by using the mark_debug constraint. Nets corresponding to signals marked for debug in HDL are automatically listed in the Debug window under the Unassigned Debug Nets folder.

 **Note:** In the Debug window, the Debug Nets view is a more net-centric view of nets that you have selected for debug. The Debug Cores view is a more core-centric view where you can view and set core properties.

The procedure for marking nets for debug depends on whether you are working with an RTL source-based project or a synthesized netlist-based project. For an RTL netlist-based project:

- Using the Vivado synthesis feature you can optionally mark HDL signals for debug using the `mark_debug` constraint in VHDL and Verilog source files. The valid values for the `mark_debug` constraint are "TRUE" or "FALSE". The Vivado synthesis feature does not support the "SOFT" value.

For a synthesized netlist-based project:

- Using the Synopsys® Synplify® synthesis tool, you can optionally mark nets for debug using the `mark_debug` and `syn_keep` constraints in VHDL or Verilog or using the `mark_debug` constraint alone in the Synopsys Design Constraints (SDC) file. Synplify does not support the "SOFT" value, as this behavior is controlled by the `syn_keep` attribute.
- Using the Mentor Graphics® Precision® synthesis tool, you can optionally mark nets for debug using the `mark_debug` constraint in VHDL or Verilog.

The following subsections provide syntactical examples for Vivado synthesis, XST, Synplify, and Precision source files.

Icons and ILA Core

- The hollow green icon indicates nets with `MARK_DEBUG` property set, but not connected to any ILA core.
- The full green icon indicates nets with `MARK_DEBUG` property set and connected to an ILA core.
- The yellow icon indicates that there is no `MARK_DEBUG` on the net, but it is connected to an ILA core.

Vivado Synthesis `mark_debug` Syntax Examples

The following are examples of VHDL and Verilog syntax when using Vivado synthesis.

- VHDL Syntax Example

```
attribute mark_debug : string;
attribute mark_debug of char_fifo_dout: signal is "true";
```

- Verilog Syntax Example

```
(* mark_debug = "true" *) wire [7:0] char_fifo_dout;
```

Synplify `mark_debug` Syntax Examples

The following are examples of Synplify syntax for VHDL, Verilog, and SDC.

- VHDL Syntax Example

```
attribute syn_keep : boolean;
attribute mark_debug : string;
attribute syn_keep of char_fifo_dout: signal is true;
attribute mark_debug of char_fifo_dout: signal is "true";
```

- Verilog Syntax Example

```
(* syn_keep = "true", mark_debug = "true" *) wire [7:0] char_fifo_dout;
```

- SDC Syntax Example

```
define_attribute {n:char_fifo_din[*]} {mark_debug} {"true"}
define_attribute {n:char_fifo_din[*]} {syn_keep} {"true"}
```

!! Important: Net names in an SDC source must be prefixed with the "n:" qualifier.

Note: Synopsys Design Constraints (SDC) is an accepted industry standard for communicating design intent to tools, particularly for timing analysis. A reference copy of the SDC specification is available from Synopsys by registering for the TAP-in program by clicking this [link](#).

Precision mark_debug Syntax Examples

The following are examples of VHDL and Verilog syntax when using Precision.

- VHDL Syntax Example

```
attribute mark_debug : string;
attribute mark_debug of char_fifo_dout: signal is "true";
```

- Verilog Syntax Example

```
(* mark_debug = "true" *) wire [7:0] char_fifo_dout;
```

Synthesizing the Design

The next step is to synthesize the design containing the debug cores by clicking Run Synthesis in the Vivado Design Suite or by running the following Tcl commands:

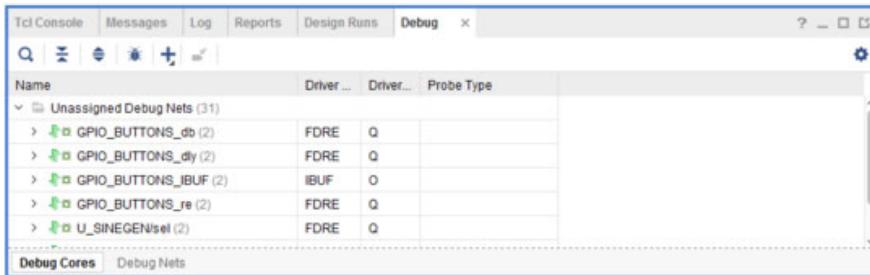
```
launch_runs synth_1
wait_on_run synth_1
```

You can also use the `synth_design` Tcl command to synthesize the design. Refer to the *Vivado Design Suite User Guide: Synthesis* ([UG901](#)) for more details on the various ways you can synthesize your design.

Marking Nets for Debug in the Synthesized Design

Open the synthesized design by clicking Open Synthesized Design in the Flow Navigator and select the Debug window layout to see the Debug window. Any nets that correspond to HDL signals that were marked for debugging are shown in the Unassigned Debug Nets folder in the Debug window.

Figure: Unassigned Debug Nets



- Selecting a net in any of the design views (such as Netlist or Schematic windows), right-click select the Mark Debug option.
- Selecting a net in any design views, drag, and drop the nets into the Unassigned Debug Nets folder.
- Using the net selector in the Set up Debug wizard (see Using the Set Up Debug Wizard to Insert Debug Cores for details).

Related Information

[Using the Set Up Debug Wizard to Insert Debug Cores](#)

Using the Set Up Debug Wizard to Insert Debug Cores

The next step after marking nets for debugging is to assign them to debug cores. The Vivado Design Suite provides an easy to use Set up Debug wizard to help guide you through the process of automatically creating the debug cores and assigning the debug nets to the inputs of the cores.

To use the Set up Debug wizard to insert the debug cores:

1. Optionally, select a set of nets for debugging either using the unassigned nets list or direct net selection.
2. Select Tools > Set up Debug from the Vivado Design Suite main menu, or click Set up Debug in the Flow Navigator under the Synthesized Design section.
3. Click Next to get to the Specify Nets to Debug panel (see the following figure).
4. Optionally, click Find Nets to Add to add more nets or remove existing nets from the table. You can also right-click a debug net and select Remove Nets to remove nets from the table.

!! Important: You can also select nets in the Netlist or other windows, drag them to the list of Nets to Debug.

5. Right-click a debug net and select Select Clock Domain to change the clock domain to be used to sample value on the net.

Note: The Set up Debug wizard attempts to automatically select the appropriate clock domain for the debug net by searching the path for synchronous elements. Use the Select

Clock Domain dialog window to modify this selection as needed but be aware that each clock domain present in the table results in a separate ILA core instance.

★ Tip: Refer to **ILA Core and Timing Considerations** in *UltraFast Design Methodology Guide for FPGAs and SoCs (UG949)* for tips on helping to minimize timing impact of the ILA Core.

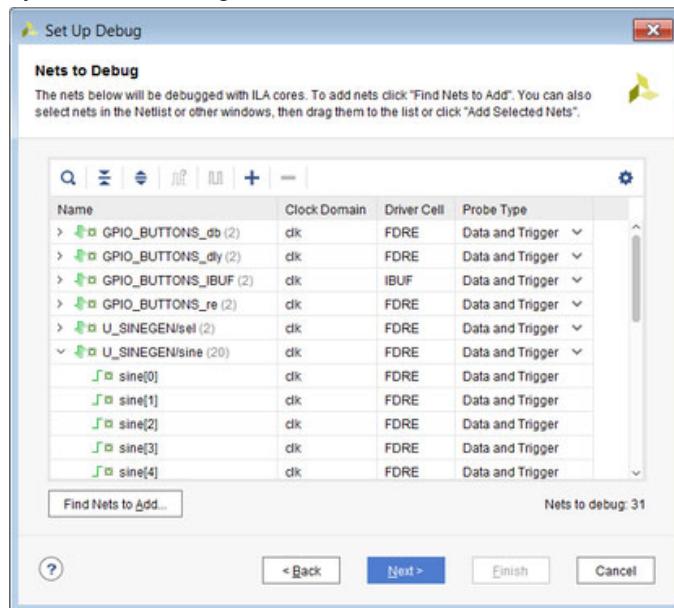
- Once you are satisfied with the debug net selection, click Next.

Note: The Set up Debug wizard inserts one ILA core per clock domain. The nets that were selected for debugging are assigned automatically to the probe ports of the inserted ILA cores. The last wizard screen shows the core creation summary displaying the number of clocks found and ILA cores to be created and/or removed.

- If you want to enable either advanced trigger mode or basic capture mode, use the corresponding check boxes to do so. Click Next to move to the last panel.

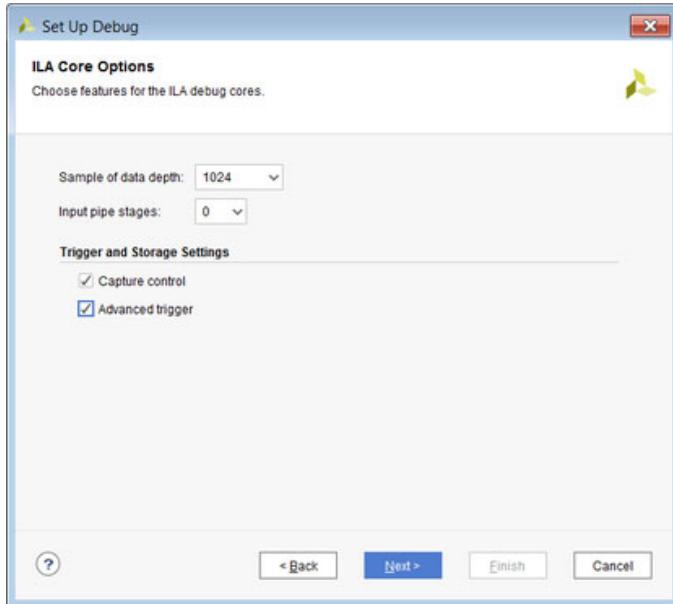
Note: The advanced trigger mode and basic capture mode features, when used in the Vivado Hardware Manager, are described in more detail in Debugging Logic Designs in Hardware.

- If you are satisfied with the results, click Finish to insert and connect the ILA cores in your synthesized design netlist.

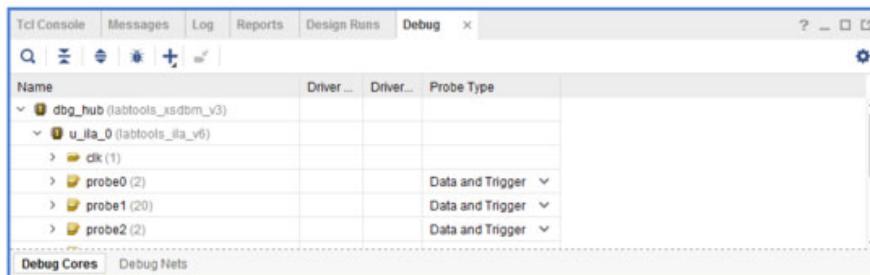


- Configure the ILA core general options such as ILA data depth (C_DATA_DEPTH), number of input pipe stages (C_INPUT_PIPE_STAGES), enabling the capture control feature (C_EN_STRG_QUAL), and enabling the advanced trigger feature (C_ADV_TRIGGER). Refer

to Modifying Properties on the Debug Cores for descriptions of these options.



10. The debug nets are now assigned to the ILA debug core, as shown in the following figure.



Related Information

[Debugging Logic Designs in Hardware](#)

[Modifying Properties on the Debug Cores](#)

[ILA Core and Timing Considerations](#)

Using the Debug Window to Add and Customize Debug Cores

The Debug Cores tab in the Debug window provides more fine-grained control over ILA core and debug core hub insertion than what is available in the Set up Debug wizard. The controls available in this window allow core creation, core deletion, debug net connection, and core parameter changes.

The Debug Cores tab of the Debug window:

- Shows the list of debug cores that are connected to the Debug Hub (dbg_hub) core.
- Maintains the list of unassigned debug nets at the bottom of the window.

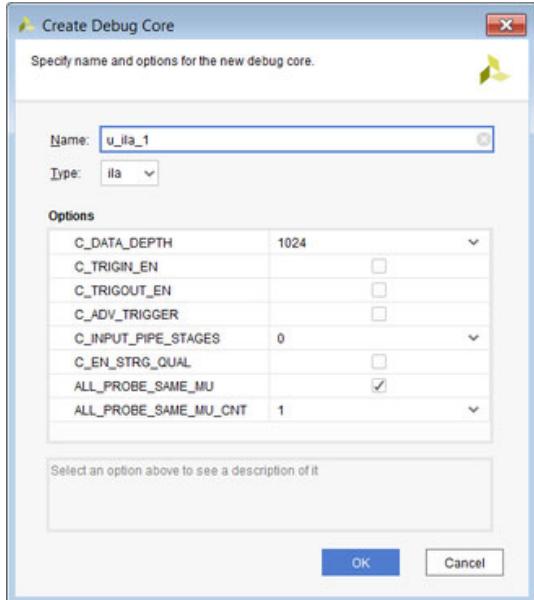
You can manipulate debug cores and ports from the popup menu or the toolbar buttons on the top of the window.

Creating and Removing Debug Cores

To create debug cores in the Debug window, click Create Debug Core. Using this interface, you can change the parent instance, debug core name, and set parameters for the core. To remove an

existing debug core, right-click the core in the Debug window and select Delete. Refer to Modifying Properties on the Debug Cores for a description of the ILA core options found in the Create Debug Core dialog.

Figure: Creating a New Debug Core



Related Information

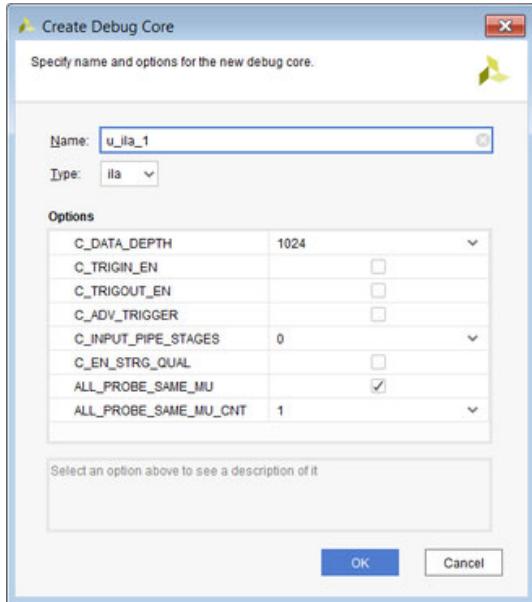
[Modifying Properties on the Debug Cores](#)

Adding, Removing, and Customizing Debug Core Ports

In addition to adding and removing debug cores, you can also add, remove, and customize ports of each debug core to suit your debugging needs. To add a new debug port:

1. Select the debug core in the Debug window.
2. Click Create Debug Port to open the dialog.
3. Select or type in the port width.
4. Click OK.

5. To remove a debug port, first select the port on the core in the Debug window, select Delete.



Connecting and Disconnecting Nets to Debug Cores

You can select, drag, and drop nets and buses (also called bus nets) from the Schematic or Netlist windows onto the debug core ports. This expands the debug port as needed to accommodate the net selection. You can also right-click any net or bus and select Assign to Debug Port.

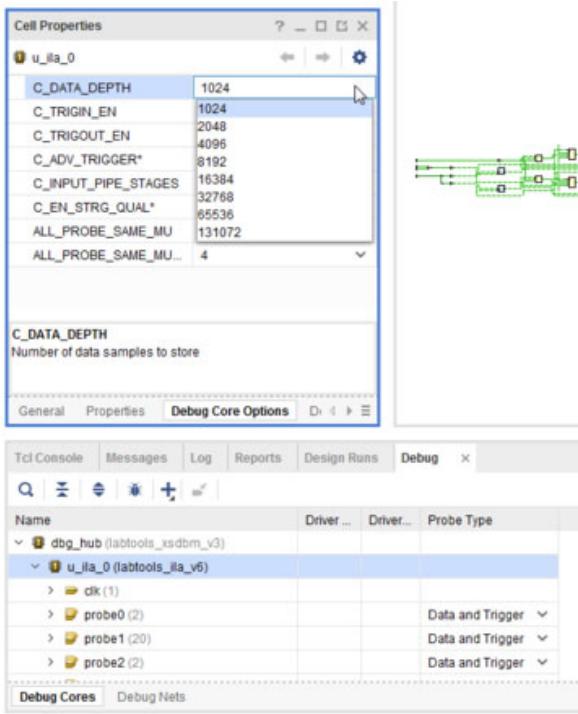
To disconnect nets from the debug core port, select the nets that are connected to the debug core port, and click Disconnect Net.

Modifying Properties on the Debug Cores

Each debug core has properties you can change to customize its behavior. To learn how to change properties on the debug_core_hub debug core, refer to Changing the BSCAN User Scan Chain of the Debug Core Hub.

You can also change properties on the ILA debug core. For example, to change the number of samples captured by the ILA debug core, do the following:

1. In the Debug window, select the desired ILA core (such as u_il_0).
2. In the Cell Properties window, select the Debug Core Options view.
3. Using the C_DATA_DEPTH pull-down list, select the desired number of samples to be captured.



A full description of all ILA core properties can be found in the following table.

Table: ILA Debug Core Properties

Debug Core Property	Description	Possible Values
C_DATA_DEPTH	Maximum number of data samples that can be stored by the ILA core. If you increase this value, it can consume more block RAM in the ILA core and adversely affect design performance.	1024 (Default) 2048 4096 8192 16384 32768 65536 131072
C_TRIGIN_EN	Enables the TRIG_IN and TRIG_IN_ACK ports of the ILA core. Note: You must use the advanced netlist change commands to connect these ports to nets in your design. If you want to use the ILA trigger input or output signals, consider using the HDL instantiation method of adding ILA cores to your design.	false (Default) true
C_TRIGOUT_EN	Enables the TRIG_OUT and TRIG_OUT_ACK ports of the ILA core. Note: You must use the advanced netlist change commands to connect these ports to	false (Default) true

Debug Core Property	Description	Possible Values
	nets in your design. If you want to use the ILA trigger input or output signals, consider using the HDL instantiation method of adding ILA cores to your design.	
C_ADV_TRIGGER	Enables the advanced trigger mode of the ILA core. Refer to Debugging Logic Designs in Hardware for more details about this feature.	false (Default) true
C_MEMORY_TYPE (Versal Only)	Selects the memory primitive (BlockRAM or UltraRAM) to use for the AXIS-ILA trace memory. Targeting UltraRAM can be useful for designs with high Block RAM utilization.	0 (BRAM) 1 (URAM)
C_INPUT_PIPE_STAGES	Enables extra levels of pipe stages (for example, flip-flop registers) on the PROBE inputs of the ILA core. You can use this feature to improve timing performance of your design by allowing the Vivado tools to place the ILA core away from critical sections of the design.	0 (Default) 1 2 3 4 5 6
C_EN_STRG_QUAL	Enables the basic capture control mode of the ILA core. Refer to Debugging Logic Designs in Hardware for more details about this feature.	false (Default) true
C_ALL_PROBE_SAME_MU	Enables all PROBE inputs of the ILA core to have the same number of comparators (also called "match units"). This property should always be set to true.	true (Default) false (not recommended)
C_ALL_PROBE_SAME_MU_CNT	The number of comparators (or match units) per PROBE input of the ILA core. The number of comparators that are required depends on the settings of the C_ADV_TRIGGER and C_EN_STRG_QUAL properties: If C_ADV_TRIGGER is false and C_EN_STRG_QUAL is false, can be set to 1 through 16. If C_ADV_TRIGGER is false and C_EN_STRG_QUAL is true, can be set to 2 through 16. If C_ADV_TRIGGER is true and C_EN_STRG_QUAL is false, can be set to 1 through 16.	1 2 3 4 5 6 7 8 9 10 11 12 13 14

Debug Core Property	Description	Possible Values
	If C_ADV_TRIGGER is true and C_EN_STRG_QUAL is true, can be set to 2 through 16. IMPORTANT: if you do not follow the previous rules, you can encounter an error during implementation when the ILA core is generated.	15 16

Related Information

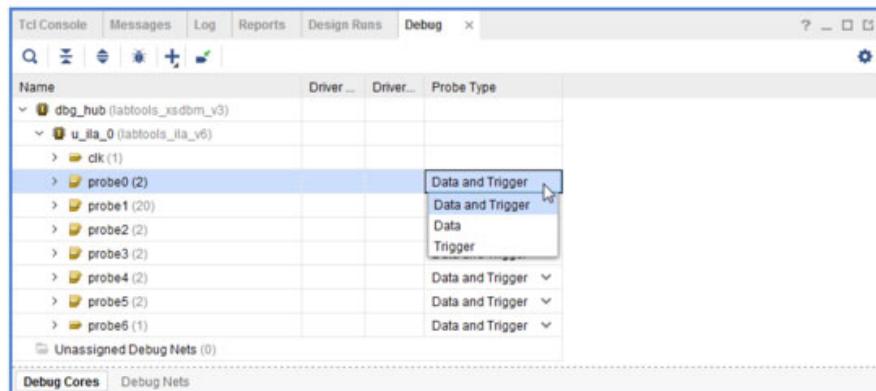
[Changing the BSCAN User Scan Chain of the Debug Core Hub](#)

[Debugging Logic Designs in Hardware](#)

Probe as Data or Trigger or Both

You can customize a probe to be used as data or trigger or both in the Vivado Hardware Manager. Probes that participate in trigger or capture compare values are configured as "trigger" only probes. This optimizes the use of BRAMs by the ILA core. Typically, probes whose data needs to be captured are configured as "data" only probes. Probes that participate in both trigger compare values, and whose data needs to be captured should be configured as "trigger and data". You can configure probes as data or trigger or both using the Set up Debug wizard as shown in the following figure.

Figure: Configuring Probes as Data or Trigger or Both Using the Set Up Debug Wizard

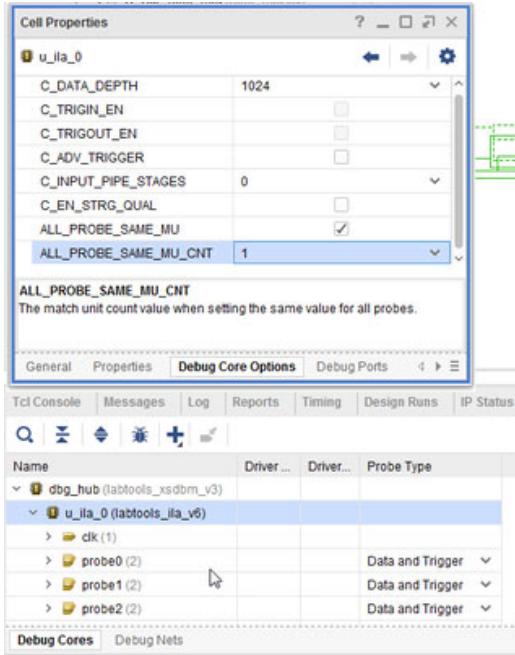


When you program the device at runtime with a design containing probes configured as "data" only, you cannot use these probes to configure trigger or capture setup conditions. Conversely, you cannot use probes configured as "trigger" only in the Waveform window.

Configuring the Number of Comparators Used

After you have inserted the ILA core on a post-synthesized netlist it is possible for you to set the number of comparators used to anywhere from 1 to 16. To do that in the Vivado IDE, go to the Debug Core Options tab of the ILA core and set the ALL_PROBE_SAME_MU_CNT property to the desired number of comparators.

Figure: Debug Core Options



Alternatively, you can set the ALL_PROBE_SAME_MU_CNT property in the Tcl Console as follows:

```
set_property ALL_PROBE_SAME_MU_CNT 10 [get_debug_cores u_il_0]
```

★ Tip: If Capture Control is enabled, you have a choice of using 1 to 15 comparators. One comparator is used by the capture control data filtering mechanism.

!! Important: It is not possible to set different number of comparators for different probes in the ILA using the insertion flow. AMD recommends that you use the HDL instantiation flow to achieve that.

Using XDC Commands to Insert Debug Cores

In addition to using the Set up Debug wizard, you can also use XDC commands to create, connect, and insert debug cores into your synthesized design netlist. Follow these steps by typing the XDC commands in the Tcl Console:

1. Open the synthesized design netlist from the synthesis run called synth_1.

```
open_run synth_1
```

!! Important: The XDC commands in the following steps are only valid when a synthesized design netlist is open.

2. Create the ILA core black box.

```
create_debug_core u_il_0 ila
```

3. Set the various properties of the ILA core.

```
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
```

4. Set the width of the clk port of the ILA core to 1 and connect it to the desired clock net.

```
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk ]]
```

Note: You do not have to create the clk port of the ILA core because it is automatically created by the `create_debug_core` command.

Important: All debug port names of the debug cores are lowercase. Using uppercase or mixed-case debug port names result in an error.

5. Set the width of the probe0 port to the number of nets you plan to connect to the port.

Note: You do not have to create the first probe port (probe0) of the ILA core because it is automatically created by the `create_debug_core` command. `set_property port_width 1 [get_debug_ports u_ila_0/probe0]`

6. Connect the probe0 port to the nets you want to attach to that port.

```
connect_debug_port u_ila_0/probe0 [get_nets [list A_or_B]]
```

7. Optionally, create more probe ports, set their width, and connect them to the nets you want to debug.

```
create_debug_port u_ila_0 probe
set_property port_width 2 [get_debug_ports u_ila_0/probe1]
connect_debug_port u_ila_0/probe1 [get_nets [list {A[0]} {A[1]}]]
```

For more information on these and other related Tcl commands, type `help -category ChipScope` in the Tcl Console of the Vivado Design Suite.

Saving Constraints After Running Debug XDC Commands

You need to save constraints after using the Set up Debug wizard, using Vivado Design Suite to create debug cores or ports, and/or running the following XDC commands:

- `create_debug_core`
- `create_debug_port`
- `connect_debug_port`
- `set_property` (on any `debug_core` or `debug_port` object)

The corresponding XDC commands are saved to a constraints file with the suffix _debug.xdc and are used during implementation to insert and connect the debug cores.

!! Important: Saving constraints while in project mode can cause the synthesis and implementation steps to go out-of-date. However, you do not need to re-synthesize the design because the debug XDC constraints are only used during implementation. You can force the synthesis step up-to-date by selecting the Design Runs window, right-clicking the synthesis run (for example, synth_1), and selecting Force up-to-date.

Implementing the Design

After inserting, connecting, and customizing your debug cores, you are now ready for implementing your design (refer to [Implementing the Design Containing the Debug Cores](#)).

Related Information

[Implementing the Design Containing the Debug Cores](#)

Debug Core Insertion in Non-Project Mode

Debug cores can be inserted in either Project Mode or Non-Project Mode. The following sample Tcl script shows how to create the debug core, set debug core attributes, and connect the debug core probes to the signals in the design being probed. In Non-Project Mode, the insertion of the debug core needs to happen after synthesizing the design, and prior to the opt_design step as follows:

!! Important: Debug core insertion is only supported for ILA cores.

The following Tcl script is an example of using the debug core insertion commands in a Non-Project flow.

```
#read relevant design source files
read_vhdl [glob .//*.vhdl]
read_verilog [ glob ./Sources/*.v ]
read_xdc ./target.xdc
#Synthesize Design
synth_design -top top -part xc7k325tffg900-2
#Create the debug core
create_debug_core u_ila_0 ila
#set debug core properties
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
#connect the probe ports in the debug core to the signals being probed in the
design
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk ]]
set_property port_width 1 [get_debug_ports u_ila_0/probe0]
```

```

connect_debug_port u_ila_0/probe0 [get_nets [list A_or_B]]
create_debug_port u_ila_0 probe
#Optionally, create more probe ports, set their width,
# and connect them to the nets you want to debug
#Implement design
opt_design
place_design
report_drc -file ./placed_drc_rpt.txt
report_timing_summary -file ./placed_timing_rpt.txt
route_design
report_drc -file ./routed_drc_rpt.txt
report_timing_summary -file ./routed_timing_rpt.txt
write_bitstream

```

HDL Instantiation Debug Probing Flow Overview

The HDL instantiation probing flow involves the manual customization, instantiation, and connection of various debug core components directly in the HDL design source. The new debug cores that are supported in this flow in the Vivado tool are shown in table the following table.

Table: Debug Cores in Vivado IP catalog Available for Use in the HDL Instantiation Probing Flow

Debug Core	Version	Description	Runtime Analyzer Tool
ILA (Integrated Logic Analyzer)	v6.2	Debug core that is used to trigger on hardware events and capture data at system speeds.	Vivado logic analyzer
VIO (Virtual Input/Output)	v3.0	Debug core that is used to monitor or control signals in design at JTAG chain scan rates.	Vivado logic analyzer
JTAG-to-AXI Master	v1.2	Debug core that is used to generate AXI transactions to interact with various AXI full and AXI lite slave cores in a system that is running in hardware.	Vivado logic analyzer

The new ILA core has two distinct advantages over the legacy ILA v1.x core:

- Works with the integrated Vivado logic analyzer feature (refer to Debugging Logic Designs in Hardware).
- No ICON core instance or connection is required.

Related Information

[Debugging Logic Designs in Hardware](#)

Using the HDL Instantiation Debug Probing Flow

The steps required to perform the HDL instantiation flow are:

1. Customize and generate the ILA and/or VIO debug cores that have the right number of probe ports for the signals you want to probe.
2. (Optional) Customize and generate the JTAG-to-AXI Master debug core and connect it to an AXI slave interface of an AXI peripheral or interconnect core in your design.
3. Synthesize the design containing the debug cores.
4. (Optional) Modify debug hub core properties.
5. Implement the design containing the debug cores.

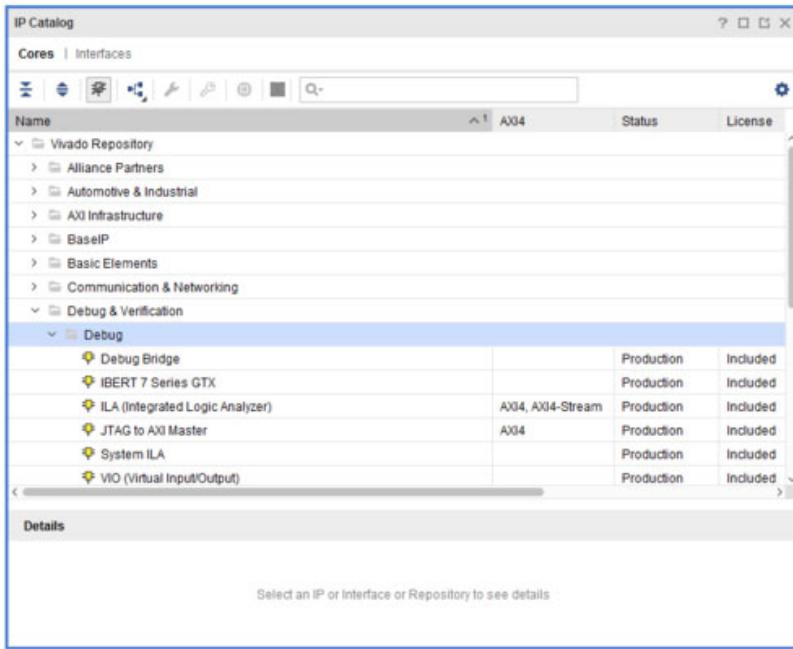
Customizing and Generating the Debug Cores

Use the IP catalog button in the Project Manager to locate, select, and customize the desired debug core. The debug cores are located in the Debug & Verification > Debug category of the IP catalog (see the following figure). You can customize the debug core by double-clicking on the IP core or by right-clicking the Customize IP menu selection.

- For more information on customizing the ILA core, refer to *Integrated Logic Analyzer LogiCORE IP Product Guide* ([PG172](#)).
- For more information on customizing the VIO core, refer to *Virtual Input/Output LogiCORE IP Product Guide* ([PG159](#)).
- For more information on customizing the JTAG-to-AXI Master core, refer to *JTAG to AXI Master LogiCORE IP Product Guide* ([PG174](#)).

After customizing the core, click the Generate button in the IP customization wizard. This generates the customized debug core and add it to the Sources view of your project.

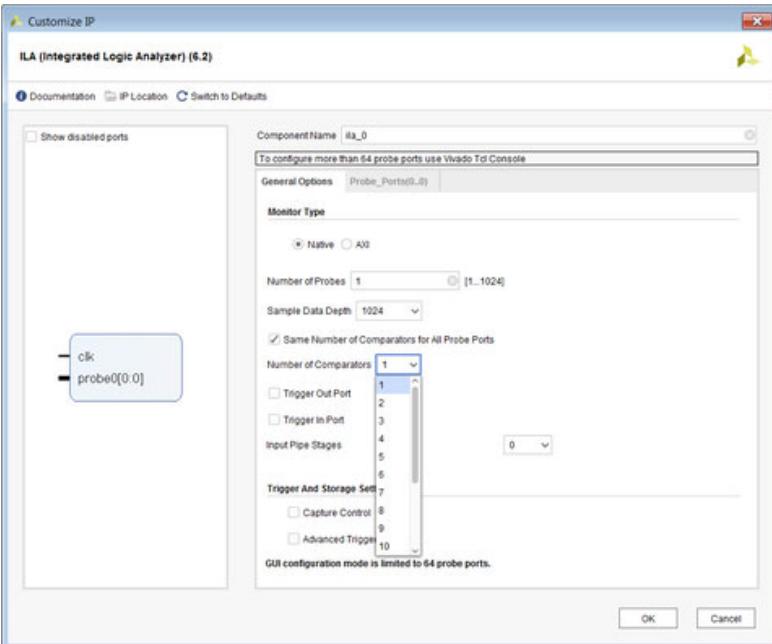
Figure: Debug Cores in the IP catalog



Configuring the Number of Comparators Used

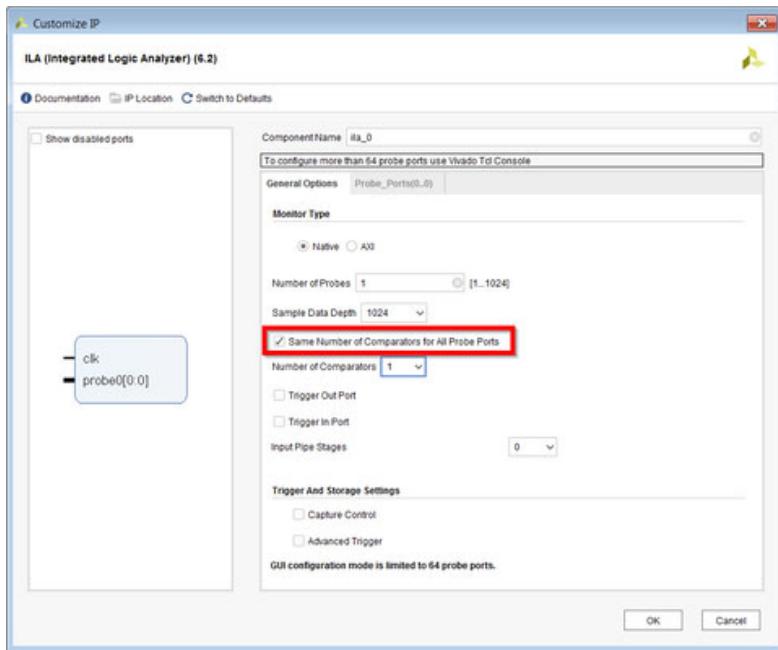
During the process of customizing the ILA IP, it is possible for you to set the number of comparators used. The range allowed is 1 to 16. It is possible to set a common number of comparators for all the probes in the ILA IP.

Figure: ILA IP Comparators in General Options



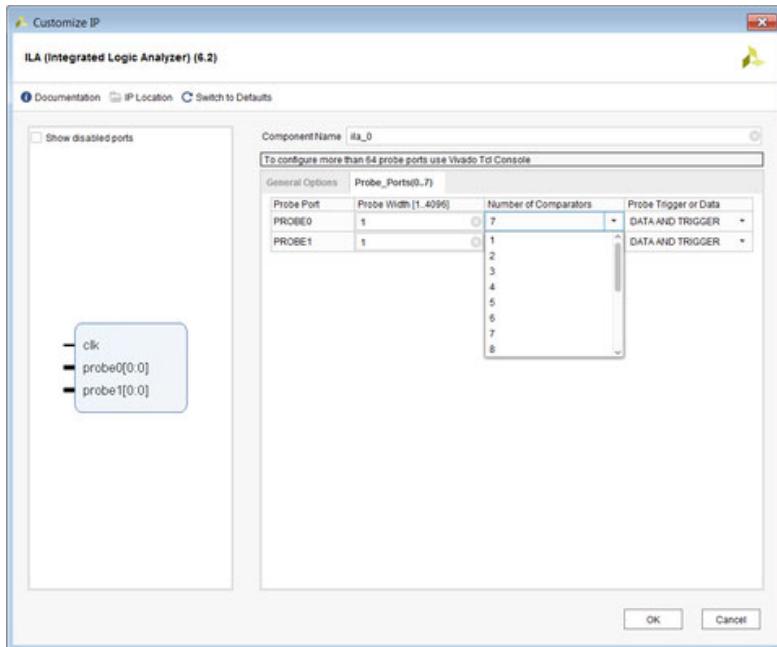
It is also possible to set the comparators for each IP as follows. It is possible to have multiple probes of different width within a single ILA. To do that you need to uncheck the Same Number of Comparators for All Probe Ports field under General Options.

Figure: Same Number of Comparators for All Ports Field



You set the exact number of comparators to be used per probe by selecting the Probe_Ports tab and setting the Number of Comparators field with the desired number of comparators.

Figure: Number of Comparators



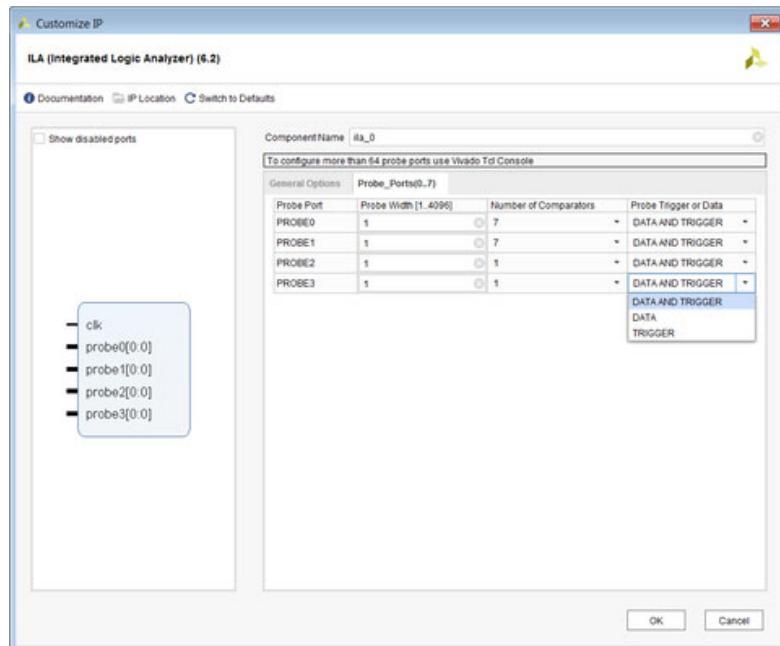
★ Tip: If Capture Control is enabled, you have a choice of using 1 to 15 comparators. One comparator is used by the capture control data filtering mechanism.

★ Tip: Depending on the number of comparators chosen, the tool automatically recalculates the number of probes that you can use in the ILA IP. The maximum number of comparators allowed per ILA is 1024.

Probe as Data or Trigger

Probes can be configured as data or trigger or both in the ILA IP Configuration wizard as shown in the following figure.

Figure: Configuring Probes as Trigger and Data



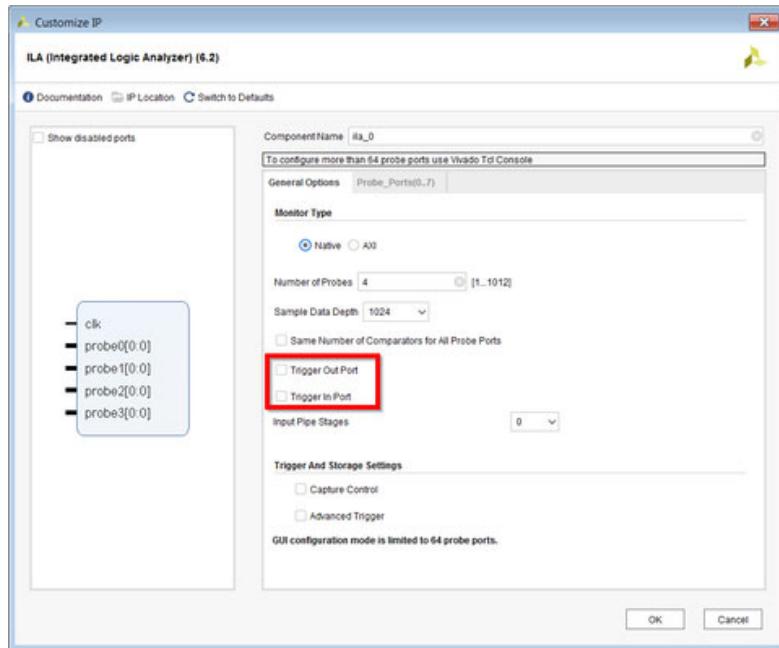
Probes that participate in trigger or capture compare values are configured as "trigger" only probes. This optimizes the use of BRAMs by the ILA core. Typically, probes whose data needs to be captured are configured as "data" only probes. Probes that participate in both trigger compare values, and whose data needs to be captured should be configured as "trigger and data."

ILA Cross Trigger

ILA Cross Triggering feature enables cross triggering between ILA cores, and between ILA cores and a processor for example, AMD Zynq™ 7000 SoC. This feature is useful for when you want to trigger between two ILA cores that are in different clock domains or perform hardware/software cross triggering between a processor and an ILA core.

For using cross trigger feature, at core generation time, you should configure the ILA core to have dedicated trigger input ports (TRIG_IN and TRIG_IN_ACK) and dedicated trigger output ports (TRIG_OUT and TRIG_OUT_ACK). If you want to use the ILA trigger input or output signals, you must use the HDL instantiation method of adding ILA cores to your design.

Figure: ILA Cross Trigger Feature



TRIG_OUT_ACK signal is an indication to the ILA core (another ILA, user design, or processor) that TRIG_OUT is properly received and causes the ILA to lower the TRIG_OUT signal on receiving TRIG_OUT_ACK.

In other words, TRIG_OUT remains HIGH until TRIG_OUT_ACK is available. If TRIG_OUT_ACK signal is tied to LOW, TRIG_OUT remains HIGH until the user re-arms the ILA. Only the TRIG_OUT goes LOW. You can rearm the ILA if TRIG_OUT_ACK is tied to LOW.

A typical cross trigger setup is illustrated in the following image where ILA2 cross triggers into ILA1. The TRIG_OUT signal of ILA2 is connected to the TRIG_IN signal of ILA1. The TRIG_IN_ACK signal of ILA1 is connected to the TRIG_OUT_ACK signal of ILA2.

```
(ILA 2) trig_out -> (ILA 1) trig_in
(ILA 1) trig_in_ack -> (ILA 2) trig_out_ack
```

Figure: Typical Cross Trigger Setup

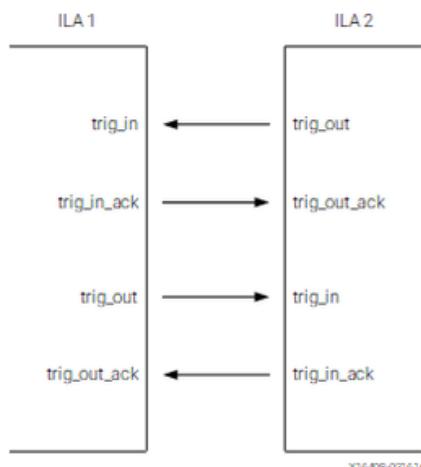
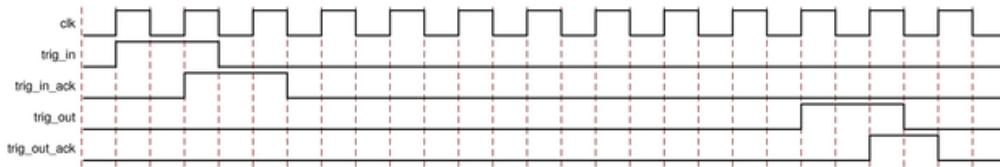


Figure: ILA Cross Trigger Timing



- It is assumed that the logic driving the `trig_in` port is synchronous to the ILA `clk`.
- It takes 1 `clk` cycle for the `trig_in_ack` signal to get asserted after `trig_in` is asserted.
- It takes 9 `clk` cycles for the `trig_out` signal to get asserted when `trig_in` is used or trigger condition is met.
- The `trig_in_ack` and `trig_out_ack` signals go low only when trigger signals are de-asserted.

For a detailed tutorial that covers using the Cross Trigger feature between the FPGA fabric and the Zynq 7000 SoC processor, see the *Vivado Design Suite Tutorial: Embedded Processor Hardware Design* ([UG940](#)).

Instantiating the Debug Cores

After generating the debug core, instantiate it in your HDL source code and connect it to the signals that you wish to probe for debugging purposes. Following is an example of the ILA instance in a Verilog HDL source file:

```
u_ila_0
(
    .clk(clk),
    .probe0(counterA),
    .probe1(counterB),
    .probe2(counterC),
    .probe3(counterD),
    .probe4(A_or_B),
    .probe5(B_or_C),
    .probe6(C_or_D),
    .probe7(D_or_A)
);
```

 **Note:** Unlike the legacy VIO and ILA v1.x cores, the new ILA core instance does not require a connection to an ICON core instance. Instead, a debug core hub (`dbg_hub`) is automatically inserted into the synthesized design netlist to provide connectivity between the new ILA core and the JTAG scan chain.

Synthesizing the Design Containing the Debug Cores

In the next step, synthesize the design containing the debug cores by clicking Run Synthesis in the Vivado Design Suite or by running the following Tcl commands:

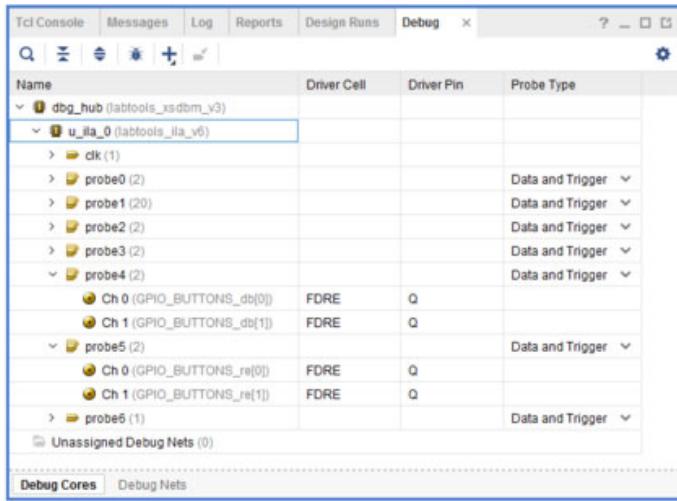
```
launch_runs synth_1  
wait_on_run synth_1
```

You can also use the synth_design Tcl command to synthesize the design. Refer to *Vivado Design Suite User Guide: Synthesis (UG901)* for more details on the various ways you can synthesize your design.

Viewing the Debug Cores in the Synthesized Design

After synthesizing your design, you can open the synthesized design to view the debug cores and modify their properties. Open the synthesized design by clicking Open Synthesized Design in the Flow Navigator and select the Debug window layout to see the Debug window that shows your ILA debug cores connected to the debug hub core (dbg_hub) as shown in the following figure.

Figure: Debug Window Showing ILA Core and Debug Core Hub



Changing the BSCAN User Scan Chain of the Debug Core Hub

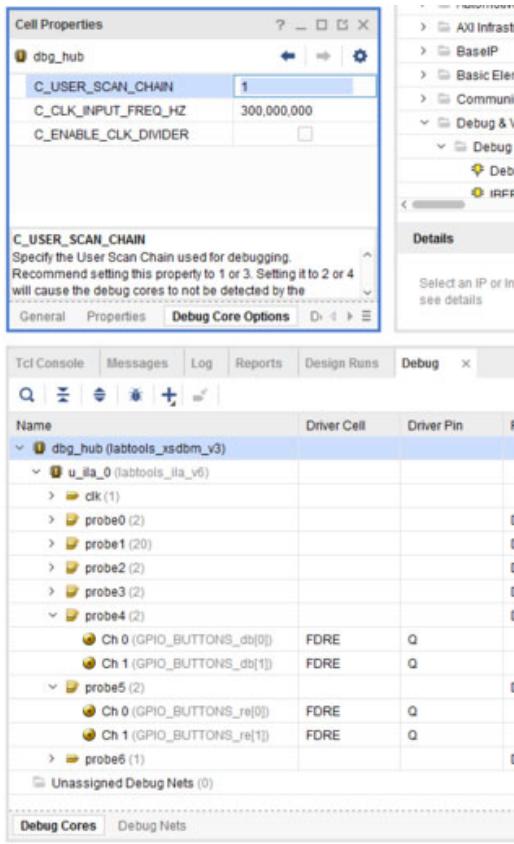
You can view and change the BSCAN user scan chain index of the debug core hub by selecting the dbg_hub in the Debug window, selecting the Debug Core Options view in the Properties window, changing the value of the C_USER_SCAN_CHAIN property (see the following figure).

!! Important: The default values for C_USER_SCAN_CHAIN is 1 for the debug hub core. If using a scan chain value other than 1 for the debug hub core, you must manually change them on the device in the Hardware Manager. Refer to Programming the Hardware Device for more details.

!! Important: If you plan to use the Microprocessor Debug Module (MDM) or other IP that uses the BSCAN primitive with the Vivado logic debug cores, you need to set the C_USER_SCAN_CHAIN

property of the dbg_hub to a user scan chain that does not conflict with the other IPs Boundary Scan Chain setting. Failure to do so results in errors later in the implementation flow.

Figure: Changing the User Scan Chain Property of the Debug Core Hub



Related Information

[Programming the Hardware Device](#)

Debug Flow in IP Integrator

The System ILA IP in Vivado IP integrator allows you to perform in-system debugging of post-implemented designs on an FPGA or adaptive SoC. Use this feature when you need to monitor interfaces and signals in the IP integrator Block Design. This feature enables you to debug AXI Read and Write Transactions in addition to AXI Read and Write, Data, and Address channel events in the Vivado Hardware Manager.

Note: On Versal devices, all System ILA features are now supported using the ILA core and can be selected by changing the ILA Input Type from Net Probes to Interface Monitor.

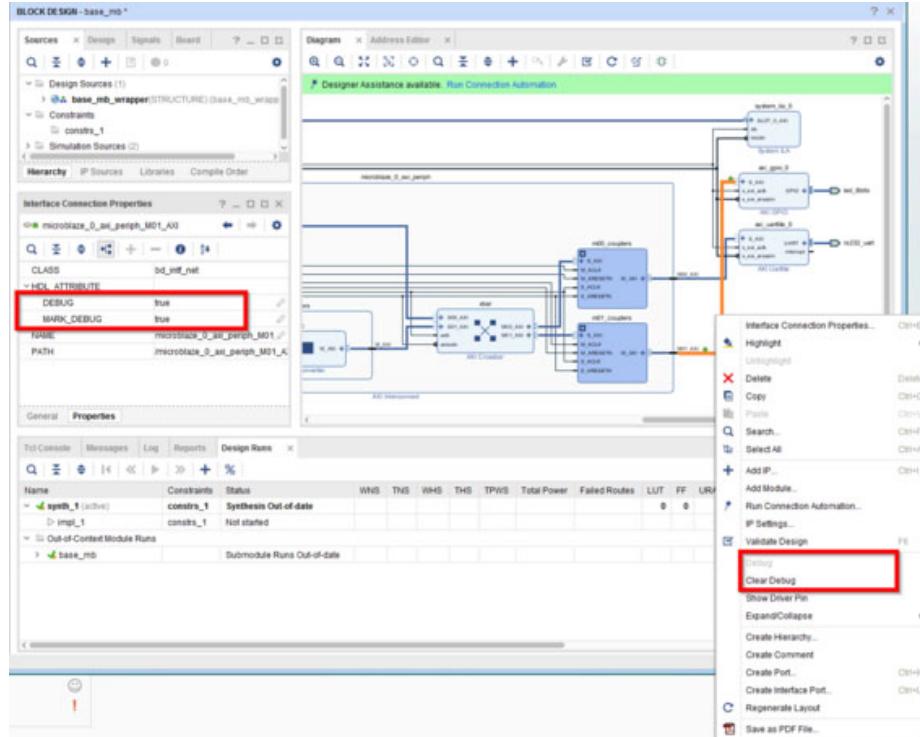
See this [link](#) in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator (UG994)* for the steps to debug interfaces and/or nets in the Block Design.

Debugging Nets and Interfaces in the IP Integrator Block Design

In the IP integrator Block Design Canvas, you can debug both nets and interfaces. You can right-click an interface or net in the Block Design and select Debug as follows. This sets the Debug and

MARK_DEBUG attributes to true. In addition, this also enables the Designer Assistance to run Connection Automation, where you can choose the net and/or interface to a System Interface ILA core, and also customize the various attributes of the debug core.

Figure: IP Integrator Mark Debug

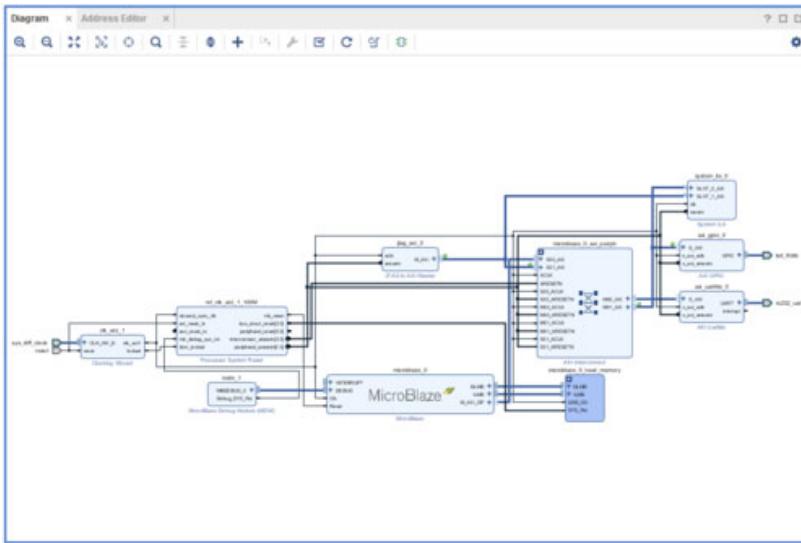


To clear a net and/or interface of debug attributes, right-click the net/interface and click Clear Debug.

Viewing System ILA Debug Cores in the Synthesized Design

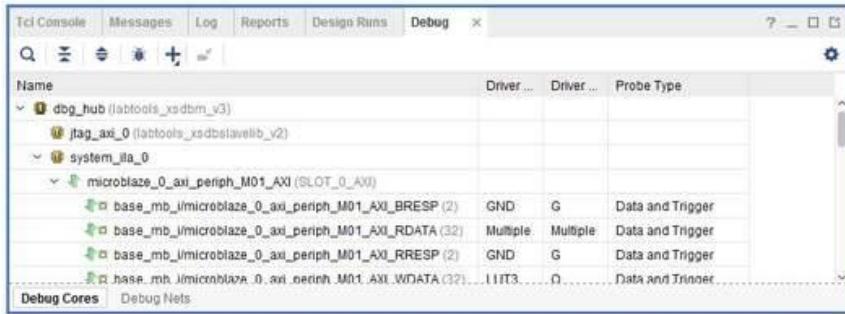
The System ILA IP in the IP integrator Block Design must be instantiated. The following figure is a snapshot of the Block Design with two debug cores instantiated in the design, the System ILA, and the JTAG to AXI Master IP cores.

Figure: Block Design



After this Block Design has been validated and synthesized, you can open the Debug window in the synthesized design to view the debug cores instantiated and inserted into the design. The System ILM and JTAG to AXI Master debug cores are displayed as follows:

Figure: System ILM and JTAG to AXI Master Debug Cores



For more details on how these interfaces can be used for debug in the Hardware Manager and to take advantage of the AXI Event level debug, see [Debugging AXI Interfaces in the Hardware Manager](#).

Related Information

[Debugging AXI Interfaces in the Hardware Manager](#)

Implementing the Design Containing the Debug Cores

The Vivado software creates the debug core hub initially as a black box. This core must be implemented prior to running the placer and router.

Implementing the Design

Implement the design containing the debug core by clicking Run Implementation in the Vivado Design Suite or by running the following Tcl commands:

```
launch_runs impl_1  
wait_on_run impl_1
```

You can also implement the design using the implementation commands `opt_design`, `place_design`, and `route_design`. Refer to the *Vivado Design Suite User Guide: Implementation (UG904)* for more details on the various ways you can implement your design.

ILA Core and Timing Considerations

The configuration of the ILA core has an impact in meeting the overall design timing goals. Follow the recommendations to minimize the impact on timing:

- Choose probe width judiciously. The bigger the probe width the greater the impact on both resource utilization and timing.
- Choose ILA core data depth judiciously. The bigger the data depth, the greater the impact on both block RAM resource utilization and timing.
- Ensure that the clocks chosen for the ILA cores are free-running clocks. Failure to do so could result in an inability to communicate with the debug core when the design is loaded onto the device.
- Close timing on the design prior to adding the debug cores. AMD does not recommend using the debug cores to debug timing related issues.
- If the design does not meet the timing requirements after adding debug cores and the timing failure is in the ILA or AXIS-ILA core, try increasing the number of input pipeline stages (C_INPUT_PIPE_STAGES).
- If the design does not meet the timing requirements after adding debug cores, and the timing failure is in the AXIS-ILA core, try changing the storage target to UltraRAM (URAM), as this can ease the timing requirement for BlockRAM (BRAM) control signals.
- If the design does not meet the timing requirements after adding debug cores and the timing failure is in the ILA or AXIS-ILA core, try a different implementation strategy such as Performance_Explore or Performance_ExtraTimingOp.
- Make sure the clock input to the ILA core is synchronous to the signals being probed. Failure to do so results in timing issues and communication failures with the debug core when the design is programmed into the device.
- For Versal architectures, if a timing failure is observed after adding debug cores try using a clock with a frequency between 100 MHz and 250 MHz for the clock connected to the AXI4-Debug Hub as this eases the timing requirements for the AXI4-Stream connectivity on all debug cores connected to this AXI4-Debug Hub.
- Make sure that the design meets timing before running it on hardware. Failure to do so results in unreliable results.
- *For Non-Versal architectures*, ensure that the clock going to the dbg_hub is a free running clock. Failure to do so could result in an inability to communicate with the debug core when the design is loaded onto the device. You can use the connect_debug_port Tcl command to connect the clk pin of the debug hub to a free-running clock.
- *For Non-Versal architectures*, if you still see that timing has degraded due to adding the ILA debug core, and the critical path is in the dbg_hub, perform the following steps:
 1. Open the synthesized design.
 2. Find the dbg_hub cell in the netlist.
 3. Go to the properties of the dbg_hub.
 4. Find property C_CLK_INPUT_FREQ_HZ.
 5. Set it to frequency (in Hz) of the clock that is connected to the dbg_hub.
 6. Find property C_ENABLE_CLK_DIVIDER and enable it.
 7. Re-implement design.

Debug Cores Clocking Guidelines

 **Note:** The following section applies to 7 series, UltraScale, and UltraScale+ devices. The Versal Debug Cores use AXI based connectivity and are not subject to the clocking guidelines in this section.

The Vivado Hardware Manager uses the JTAG interface to communicate with the Vivado Debug Hub core, which provides an interface between the JTAG Boundary Scan (BSCAN) interface of the FPGA and the Vivado Debug cores.

JTAG Clock

This clock synchronizes the internal state machine operation of the JTAG Boundary Scan (BSCAN) interface. You can choose the JTAG clock frequency in the Vivado Hardware Manager while connecting to the target device. If your design contains debug cores, ensure that the JTAG clock is 2.5x times slower than the debug hub clock.

You can modify the JTAG frequency by using the Open New Hardware Target wizard or the following Tcl command:

```
set_property PARAM.FREQUENCY 250000 [get_hw_targets  
*/xilinx_tcf/Digilent/210203327962A]
```

Debug Hub Clock

The Vivado Debug Hub core, which provides an interface between the JTAG Boundary Scan (BSCAN) interface of the FPGA and the Vivado Debug cores. The Debug Hub core is inserted automatically by the Vivado IDE during the design implementation step if it detects debug cores in the design. The Vivado IDE chooses the clock driving the Debug Hub core during the design implementation step.

AMD recommends that the Debug Hub clock frequency be around 100 MHz or less because the JTAG clock speeds do not require a particularly high frequency.

You can change the Debug Hub Clock using the following Tcl command.

```
connect_debug_port dbg_hub/clk [get_nets <clock net name>]
```

 **Note:** You need to run this command after the design has been synthesized, but before implementation.

You can also reduce the Debug Hub Clock Frequency to 100 MHz using the following Tcl commands.

```
set_property C_CLK_INPUT_FREQ_HZ 200000000 [get_debug_cores dbg_hub]  
set_property C_ENABLE_CLK_DIVIDER true [get_debug_cores dbg_hub]
```

 **Note:** You need to run this command after the design has been synthesized but before implementation. This is recommended for designs that have very high-speed clocks. This command enables the inclusion of an MMCM-based clock divider inside the Debug Hub core to achieve a clock frequency of 100 MHz.

Debug Core Clocks

All of the debug cores available in the Vivado IP catalog require a clock that ensures synchronization with the input probes being monitored or any output signals being driven by the debug cores. During core discovery and debug measurement phase, it is expected that the clock is free running and stable. It is also expected that the clock is synchronous to the signals monitored or driven. Failure to do so could result in a cycle of inaccurate data.

The Debug Hub IP bridges between the host machine (through BSCAN Primitive which supports a serial interface) and debug cores on the chip (through XSDB interface which supports a parallel interface). The BSCAN primitive clock shifts the data in and out of the chip to the Debug Hub IP serially. The Debug Hub IP collects the data and sends it to all the debug cores on parallel interface using the Debug Hub clock and vice versa. If any of the debug core clocks are not free running or not stable, you end up with corrupted data which results in a "Debug Cores not detected" message. To avoid any corruption of data, it is important to ensure that the JTAG clock and Debug Hub clocks are stable and free running during the debug core detection process.

1. The Debug Hub Clock must be free running and stable. AMD recommends that the clock be driven from a clock driver that is properly constrained and whose timing is met.
2. If the clocks are driven from MMCM/PLL, ensure that the MMCM/PLL LOCKED signal is high prior to any debug core measurements. If the clock is connected to the Debug Hub or any of the debug cores and the MMCM/PLL LOCKED signal transitions to a 0 in the middle of debug operations, the clock can have significant jitter that might result in unpredictable behavior of the debug logic.
3. In order to detect the debug cores, take measurements using those cores and capture data. It is required to have all the associated clocks free running and stable.

The following table lists the various debugging phases and the clocks required during the specific phases.

Table: Debugging Phase Clock Requirements

Debugging Phase	JTAG Clock	Debug Hub Clock	Debug Core Clock ²
Connect to Target	Stable ¹	NA	NA
Programming	Stable ¹	NA	NA
Debug Core Discovery	Stable ¹	Stable	NA
Debug Core Measurement ³	Stable ¹	Stable ¹	Stable

1. Stable Clock: A clock that does not pause/stop during the event.
2. Assumes the Debug Core Clock is different from the Debug Hub Clock.
3. A Debug Core Measurement phase includes any step that does a get or set of properties on the debug core.

Vivado Hardware Manager Clocking Related Error Messages

If the JTAG Clock is inactive or unavailable, you are not able to connect to the hardware target.
If the Debug Hub Clock is inactive or unavailable, the Vivado Hardware Manager issues the following error message:

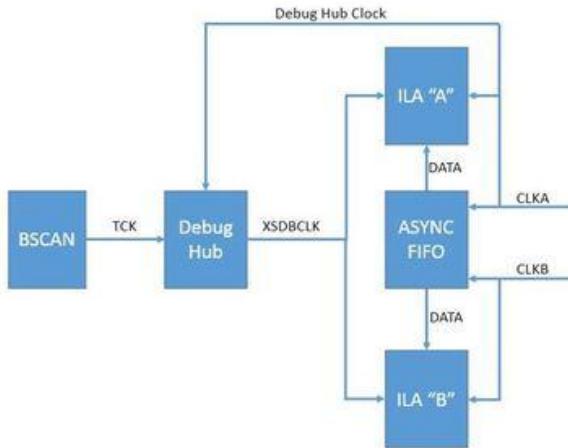
```
INFO: [Labtools 27-1434] Device xxx (JTAG device index = 0) is programmed with a
design that has no supported debug core(s) in it.
WARNING: [Labtools 27-3123] The debug hub core was not detected at User Scan
Chain 1
or 3.
Resolution:
1. Make sure the clock connected to the debug hub (dbg_hub) core is a free
running
clock and is active OR
2. Manually launch hw_server with -e "set xsdb-user-bscan <C_USER_SCAN_CHAIN
scan_chain_number>" to detect the debug hub at User Scan Chain of 2 or 4. To
determine
the user scan chain setting, open the implemented design and use: get_property
C_USER_SCAN_CHAIN [get_debug_cores dbg_hub].
```

If any of the Debug Core Clocks are inactive or unavailable, the Vivado Hardware Manager issues the following error message:

```
INFO: [Labtools 27-2302] Device xxx (JTAG device index = 1) is programmed with a
design that has 1 ILA core(s).
CRITICAL WARNING: [Labtools 27-1433] Device xxx (JTAG device index = 1) is
programmed
with a design that has an unrecognizable debug core (slave type = 17) at user
chain
= 1, index = 0.
Resolution:
1) Ensure that the clock signal connected to the debug core and/or debug hub is
clean
and free-running.
2) Ensure that the clock connected to the debug core and/or debug hub meets all
timing
constraints.
3) Ensure that the clock connected to debug core and/or debug hub is faster than
the
JTAG clock frequency.
```

The following figure is an example of a design with two ILA cores:

Figure: Debug Core Clocking Example



The example design contains two ILA cores, ILA "A" and ILA "B."

The clocking topology of this debug network is as follows. After the design has been programmed into the device, the Vivado Hardware Manager tries to discover the existence of the Debug Hub core in the design. The Debug Hub in turn tries to discover and account for all the debug cores connected to it. In this design the debug cores are ILA "A" and ILA "B."

Note: CLKA drives both the ILA "A" and the Debug Hub core. CLKB drives the ILA "B" debug core.

When you connect to the target and program the device, expect an active JTAG clk. In the Debug Core Discovery Phase, expect a free-running and stable clock driving the Debug Hub core, which in this case is CLKA. During Debug Core Measurement phase (anything that involves getting/setting properties on the debug core), expect an active JTAG, Debug Hub, and Debug Core clocks. If you expect to trigger and capture data on ILA "B", expect free running and stable JTAG, Debug Hub (CLKA), and Debug Core (CLKB) Clocks.

Debug Hub Insertion Guidelines

In some cases, a design might contain an instance of the Debug Bridge IP that appears to prevent the debug flow from successfully inserting a debug hub and the following error might appear:

```
[Chipscope 16-336] Failed to find or create hub core for debug slave <debug core name>. Insertion of debug hub is not supported when there are instantiated debug bridge cores in either master mode or switch enabled in the design. Either remove debug slave core or instantiate a debug bridge master in the region of the debug slave
```

This issue occurs because the debug flow detects the presence of a Debug Bridge IP and does not attempt to automatically insert a Debug Hub IP. However, the instance of the Debug Bridge IP in the design is not in the correct mode to connect to a debug core. To resolve this issue, ensure that the design has at least one instance of a Debug Bridge IP in BSCAN-to-Debug Hub mode.

Adding Vivado Debug Cores to a Dynamic Function eXchange Design

Vivado Debug cores can be instantiated in a Dynamic Function eXchange design including within the Reconfigurable Modules. There are specific requirements and a methodology to adding and connecting these cores. See this [link](#) in the *Vivado Design Suite User Guide: Dynamic Function eXchange (UG909)* for the methodology needed to add and connect these Vivado Debug cores. For an example of instantiating debug cores in a Dynamic Function eXchange design and a description of the functionality within the Vivado Hardware Manager, see this [link](#) in *Vivado Design Suite Tutorial: Dynamic Function eXchange (UG947)*.

Debugging Logic Designs in Hardware

Once you have the debug cores in your design, you can use the run time logic analyzer features to debug the design in hardware.

Using Vivado Logic Analyzer to Debug the Design

The AMD Vivado™ logic analyzer feature is used to interact with new ILA, VIO, and JTAG-to-AXI Master debug cores that are in your design. To access the Vivado logic analyzer feature, click the Open Hardware Manager button in the Program and Debug section of the Flow Navigator.

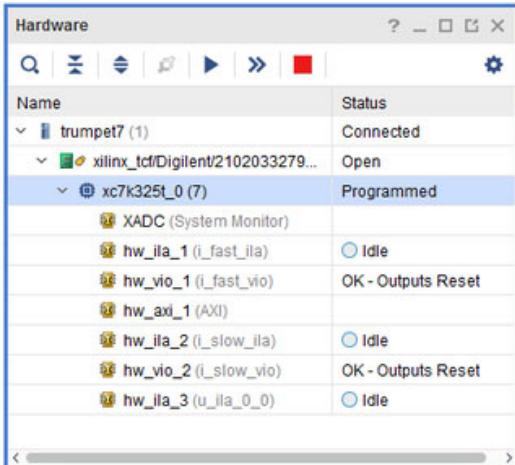
The steps to debug your design in hardware using an ILA debug core are:

1. Connect to the hardware target and program the FPGA or adaptive SoC with the .pdi file.
2. Set up the ILA, debug a core trigger, and capture controls.
3. Arm the ILA debug core trigger.
4. View the captured data from the ILA debug core in the Waveform window.
5. Use the VIO debug core to drive control signals and/or view design status signals.
6. Use the JTAG-to-AXI Master debug core to run transactions to interact with various AXI slave cores in your design.

Connecting to the Hardware Target and Programming the Device

Programming an FPGA or adaptive SoC prior to debugging uses exactly the same steps as described in Programming the FPGA or adaptive SoC. After programming the device with the .pdi file that contains the new ILA, VIO, and JTAG-to-AXI Master debug cores, the Hardware window now shows the debug cores with the RTL instance name shown in parenthesis which are detected when scanning the device.

Figure: Hardware Window Showing Debug Cores



For more information on using the ILA core, refer to [Setting up the ILA Core to take a Measurement](#).
 For more information on using the VIO core, refer to [Setting up the VIO Core to take a Measurement](#).

!! Important: Ensure the JTAG clock is slower than the clocks input to the debug cores. You can modify the JTAG frequency using the Open New Hardware Target wizard or the following Tcl command: `set_property PARAM.FREQUENCY 250000 [get_hw_targets */xilinx_tcf/Digilent/210203327962A]`

Related Information

[Programming the Device](#)

[Setting Up the ILA Core to Take a Measurement](#)

[Setting Up the VIO Core to Take a Measurement](#)

Vivado Hardware Manager Dashboards

The Vivado Hardware Manager Dashboards help you manage the various windows for your System Monitor, ILA, and VIO Debug cores. The dashboards enable you to create, modify, and save the configuration of these windows in your AMD Vivado™ Design Suite project.

Default Dashboards

When debug cores are detected upon refreshing a hardware device, the default dashboard for each debug core is automatically opened.

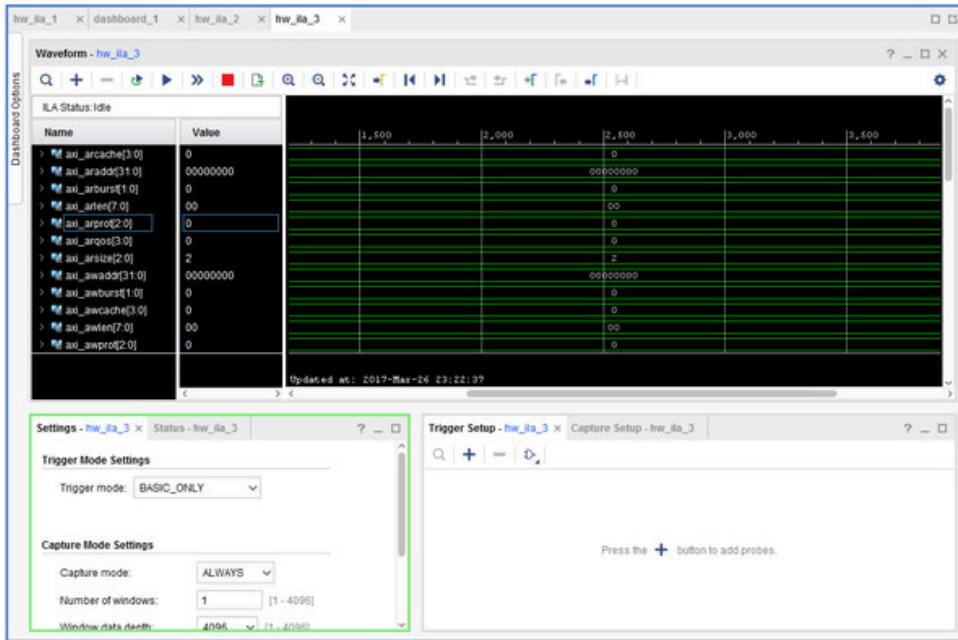
Default Dashboard Windows

Every default dashboard contains windows relevant to the debug core the dashboard is created for. The default dashboard created for the ILA debug core contains five windows:

- Settings window
- Status window
- Trigger Setup window
- Capture Setup window
- Waveform window

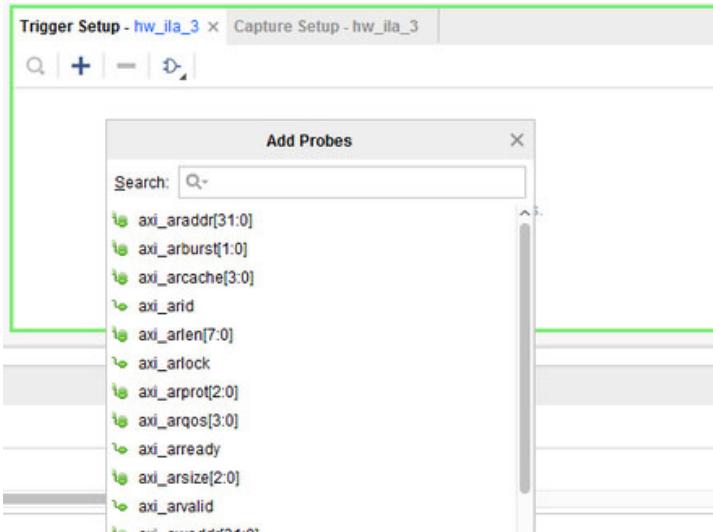
An example of the default ILA Dashboard can be seen as follows:

Figure: Default ILA Dashboard



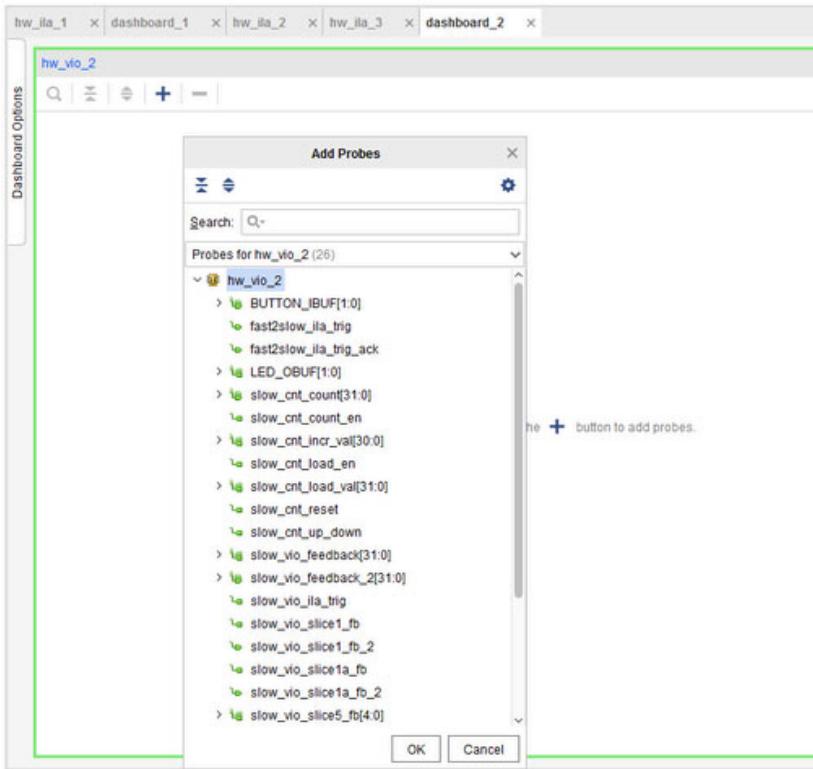
You can start adding probes to the Trigger Setup window by clicking the "+" button in the center of the window and selecting probes from the Add Probes window as seen in the following figure:

Figure: Add Probes Window



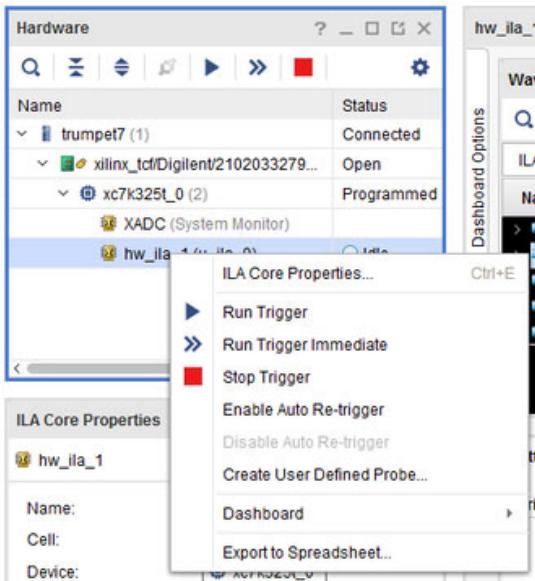
The VIO Default dashboard starts out empty to which you can add VIO probes to as shown in the following figure.

Figure: Adding VIO Probes



To view a dashboard associated with a debug core, right-click the debug core object in the Hardware window, select the Dashboard option, and click the dashboard name. Double-clicking a debug core in the hardware window pops up the dashboards associated with that debug core.

Figure: Associated Dashboards



Window Controls within a Dashboard

Each window has the following title bar controls, which enable you to manipulate the window:

- Minimize
- Maximize
- Close

Moving Windows

To move windows:

1. Select the window tab or title bar and drag the window. A gray outline indicates where the window is located after the move.
2. To commit to the placement, release the mouse button.

 **Note:** Dropping one window onto an existing window places the two window tabs in the same region.

!! Important: You cannot move windows into or out of the workspace. However, you can resize and move the windows within the workspace.

Resizing Windows

- To resize windows, click, and drag the window border.
-
-  **Note:** The mouse cursor changes to a resize cursor when positioned over a window border or drag handle, indicating that you can click and drag the window border to resize the window.
-
- To expand a window to use all of the viewing environment, click the Maximize button in the upper right corner of the window.
 - To restore a window to its original size, double-click the window title bar or tab.

Closing Windows

- To close windows, click the Close button in the upper right corner of the window.
-
-  **Note:** In some cases, this button is also available in the window tab.
-
- Right-click a window tab or title bar and select Close from the popup menu.

Window Tabs

Each window has a tab that you can select to make that the window is active. The tab is at the bottom of some windows, such as the Trigger Setup and Capture Setup windows.

★ Tip: To make the next tab active, press Ctrl+Tab. To make the previous tab active, press Ctrl+Shift+Tab. To maximize or minimize the window, double-click the window tab.

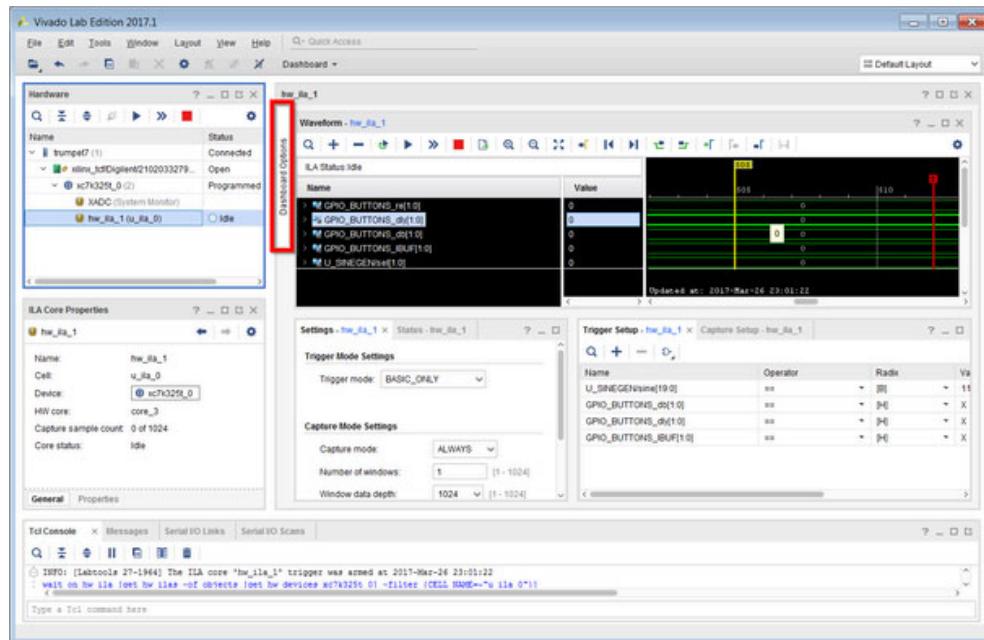
Customizing Dashboards

Typically, the windows in the default dashboards should be enough to debug your design and view the results. However, you might want to move windows around (that is, customize your dashboards). For example, you want to view both the ILA status and Waveform window in addition to controlling VIO probes in the same dashboard. In those situations, AMD recommends customizing the dashboard to fit your needs.

Dashboard Options

Every dashboard has a Dashboard Options slideout on the left. Use the Dashboard Options button on the left side of a dashboard to open its Dashboard Options settings. The Dashboard Options settings allow you to control the windows that appear in a specific dashboard. For example, you want to customize the ILA dashboard to include one of the VIO windows as well. You click on the VIO window to include it in the Dashboard Options and the VIO window shows up in the ILA Dashboard as follows. You can now add the VIO probes you are interested in and trigger the ILA window.

Figure: Adding Dashboard Options

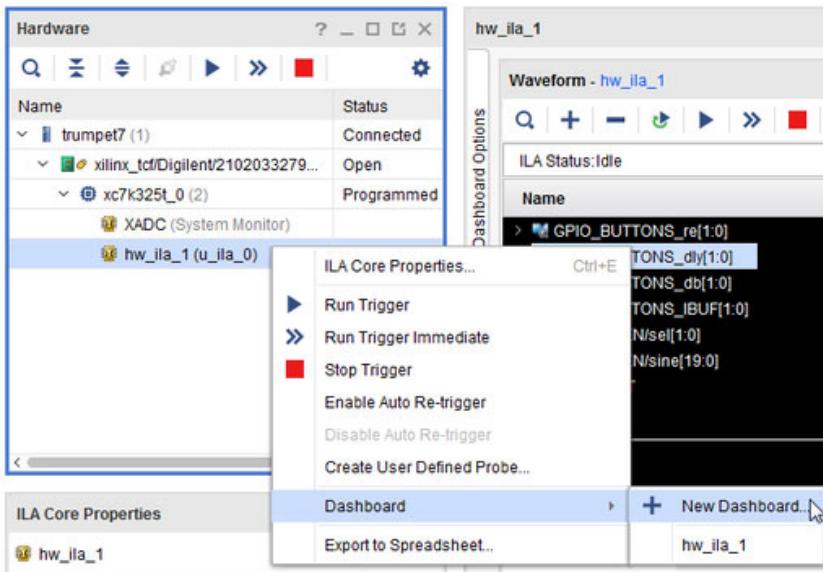


Click the Dashboard Options button on the left side of a dashboard to open and close the Dashboard Options slideout.

Creating New Dashboards

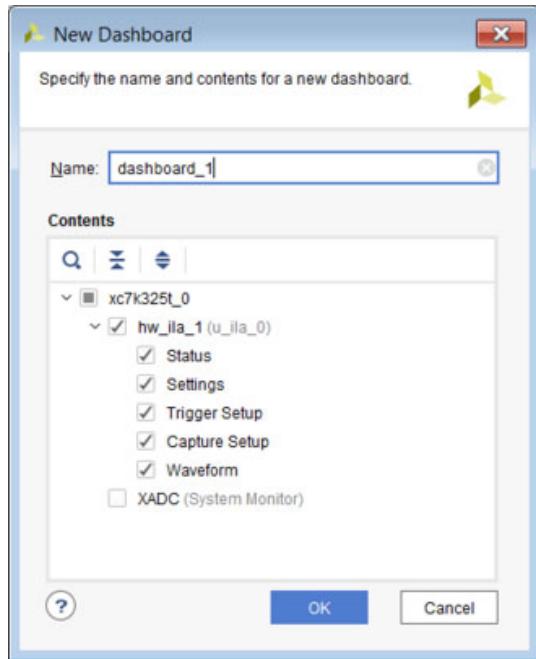
In addition to customizing Default Dashboards using the Dashboard Options, you can also create brand new dashboards. To do that right-click the debug core object in the Hardware window and select the Dashboard > New Dashboard option as shown in the following figure.

Figure: Creating New Dashboard



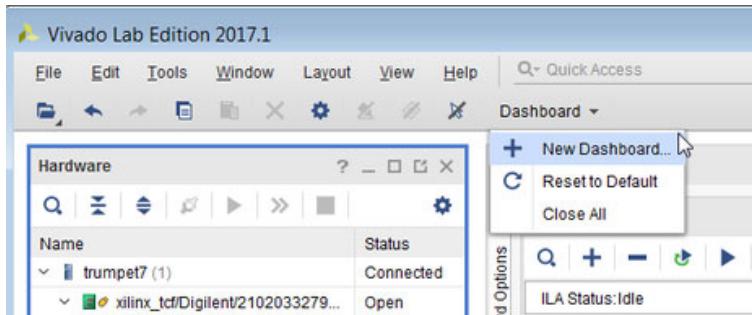
When the New Dashboard dialog appears, you can customize the dashboard as necessary and click OK.

Figure: New Dashboard Dialog



You can also use the Dashboard toolbar button to create new dashboards as follows:

Figure: Dashboard Toolbar Button



★ Tip: To view all the dashboards associated with a debug core, right-click the debug core in the Hardware view and click Dashboard. Alternatively double-click the debug core in the Hardware view and the list of dashboards associated with the debug core pop-up.

★ Tip: To float a single window in a dashboard, AMD recommends that you create a dashboard with that window and float the dashboard.

ILA Waveform Window in Dashboards

Each ILA Waveform window can only appear in a single dashboard. If you click a Waveform window that is located in another dashboard, you are going to get a notification that the window is being relocated as shown in the following figure.

Figure: ILA Waveform Confirmation



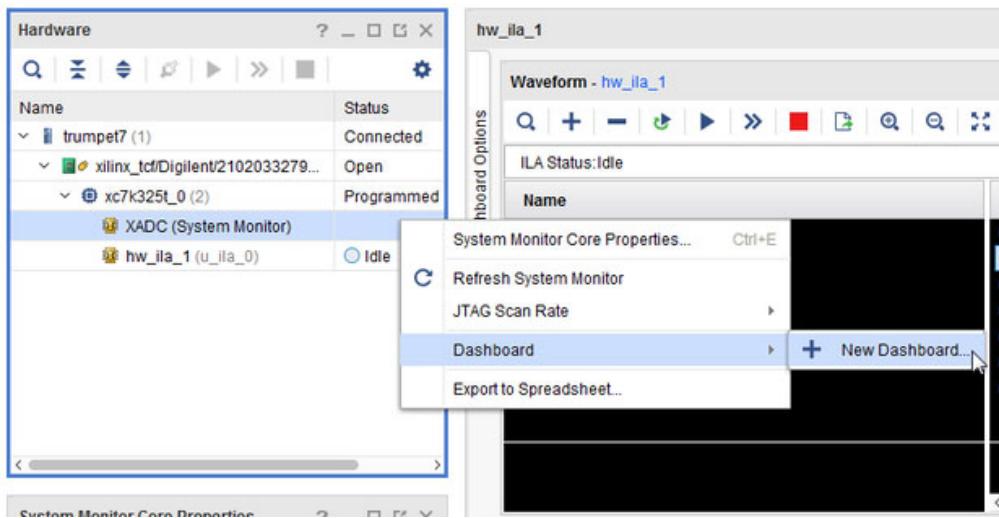
Click OK, and the Waveform window relocates to the specified dashboard.

★ Tip: Save ILA data when closing the Waveform window.

System Monitor Dashboards

You can also include the XADC/System Monitor window into another dashboard or its own dashboard.

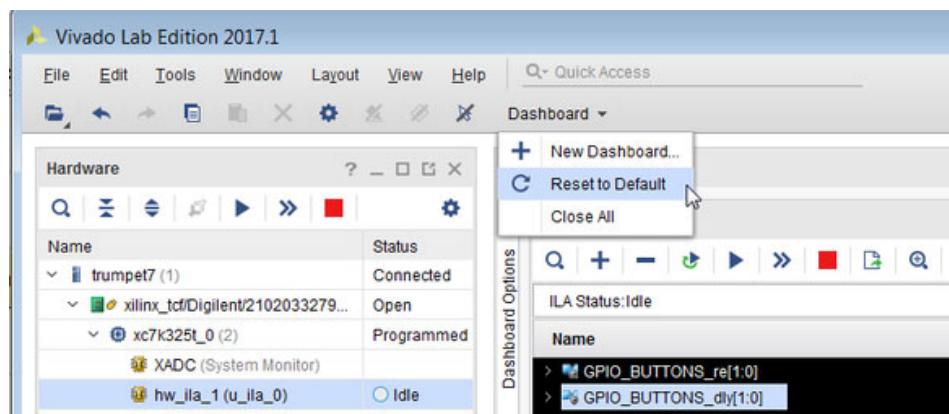
Figure: System Monitor Dashboard



Resetting to Default Dashboards

You can reset the dashboards back to their default state by clicking Dashboard on the toolbar and selecting Reset to Default.

Figure: Resetting to Default Dashboard



Closing Dashboards

You can close all the Dashboards by clicking Dashboard on the toolbar and selecting Close All. This deletes all of the dashboards and the user settings in those dashboards.

You can also close a single dashboard by clicking the "X" button on the upper right-hand corner. This deletes the dashboard and all of the user settings in the dashboard.

Saving User Dashboard Preferences and Settings

User Dashboard settings and preferences are saved automatically by Vivado IDE. Upon closing and reopening the project the user settings and preferences are restored back into the Hardware Manager.

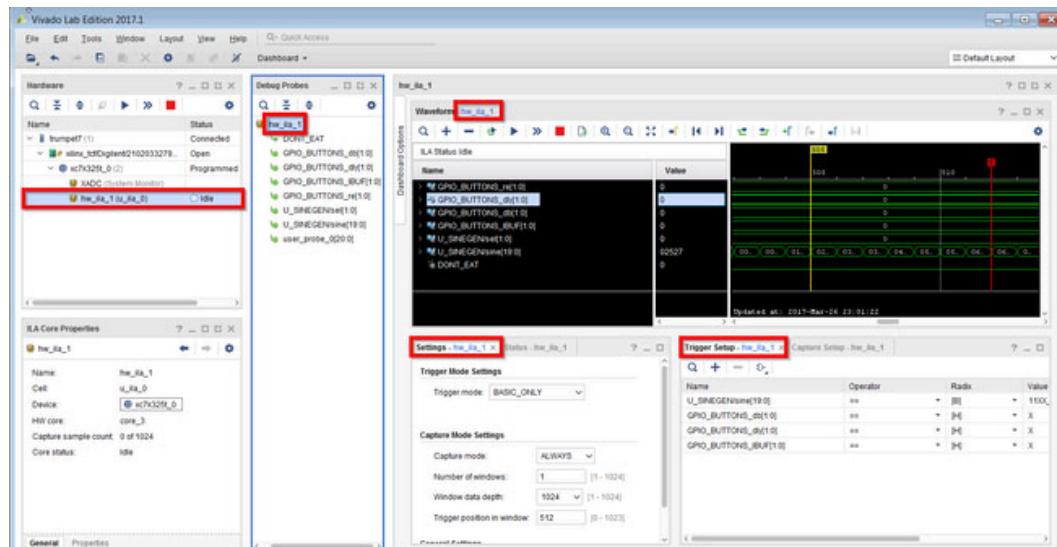
Setting Up the ILA Core to Take a Measurement

The ILA cores that you add to your design appear in the Hardware window under the target device. If you do not see the ILA cores appear, right-click the device and select Refresh Device. This re-scans the FPGA or adaptive SoC and refreshes the Hardware window.

Note: If you still do not see the ILA core after programming and/or refreshing the FPGA or adaptive SoC, check to make sure the device was programmed with the appropriate .bit file and check to make sure the implemented design contains an ILA core. Also, check to make sure the appropriate .ltx probes file that matches the .bit file is associated with the device.

Click the ILA core (called hw_ila_1 in the following figure) to see its properties in the ILA Core Properties window. You can see all the probes corresponding to the ILA core by using the Windows > Debug Probes menu option, which displays the Debug Probes window as shown in the following figure.

Figure: Selection of the ILA Core in Various Views



Adding Probes

You can add relevant probes to specific windows in an ILA Dashboard by clicking on the "+" button on the window toolbar or workspace.

Writing Debug Probes Information

The Debug Probes window contains information about the nets that you probed in your design using the ILA and/or VIO cores. This debug probe information is extracted from your design and is stored in a data file that typically has an .ltx file extension.

Normally, the debug probes file is automatically created during the implementation process. However, you can also use the `write_debug_probes` Tcl command to write out the debug probes information to a file:

1. Open the Synthesized or Netlist Design.
 2. Run the `write_debug_probes filename .ltx` Tcl command.
-

!! Important: If you are using non-project mode, you must manually call the `write_debug_probes` command immediately following the `opt_design` command.

Reading Debug Probes Information

The debug probes file is automatically associated with the hardware device if the Vivado IDE is in project mode and a probes file is called `debug_nets.ltx` is found in the same directory as the bitstream programming (.bit) file that is associated with the device.

You can also specify the location of the probes file:

1. Select the hardware device in the Hardware window.
2. Set the Probes file location in the Hardware Device Properties window.
3. Right-click the hardware device in the Hardware window and select Refresh Device to read the contents of the debug probes file and associate and validate the information with the debug cores found in the design running in the hardware device.

You can also set the location using the following Tcl commands to associate a debug probes file called `C:/myprobes.ltx` with the first device on the target board:

```
% set_property PROBES.FILE {C:/myprobes.ltx} [lindex [get_hw_devices] 0]
% refresh_hw_device [lindex [get_hw_devices] 0]
```

Renaming Debug Probes

You can use the Debug Probes window to rename debug probes that belong to an ILA or VIO core. You can rename the debug probes and add them to an existing Waveform Viewer for the core, or you can add them to the various trigger and/or capture windows of the ILA Dashboard. These names could be the *custom*, *long*, or *short* name associated with the debug probe.

To perform these operations, right-click an ILA/VIO core's debug probe(s) and select one of the following:

Rename

Prompts you to rename the probe to a custom name.

Name

Allows you to select the long, short, or custom name of the debug probe. Subsequent references to the debug probe in the Vivado IDE window uses the name that you selected.

Long

Displays the full hierarchical name of the signal or bus being probed.

Short

Displays the name of the signal or bus being probed.

Custom

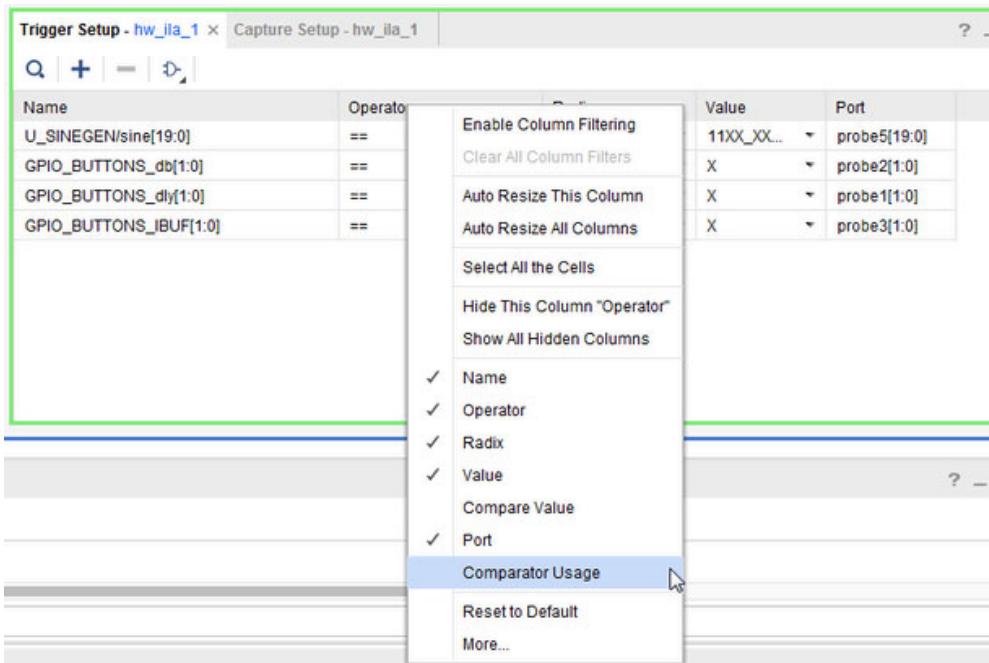
Displays the custom name given to the signal or bus when renamed.

Using Multiple Comparators

If you have customized the probes and/or ILA debug cores to use more than 1 comparator in the Basic/Advanced mode, you can use these comparators both in the Basic and Advanced Trigger Setup window.

You can add the probe into the Basic trigger setup window and set up the trigger conditions. The comparator usage column gives you information on the comparator used within the probe for the specific compare condition out of the total number of comparators associated with the probe.

Figure: Basic Trigger Setup - Comparator Usage



★ Tip: The Comparator Usage column is a hidden column. To enable this column right-click the Trigger Setup column title row as follows and select Comparator Usage.

Figure: Comparator Usage Column

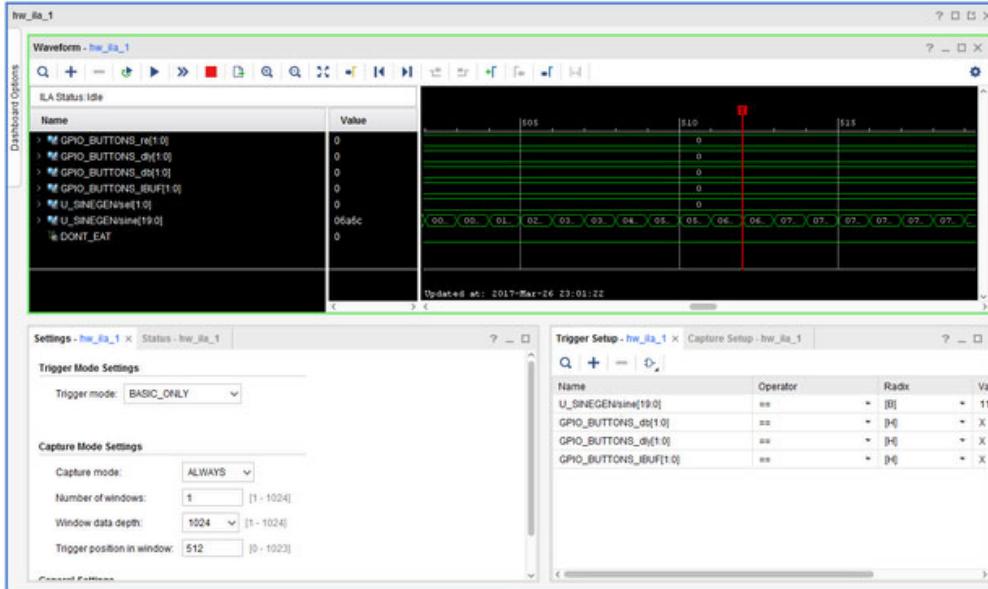
The screenshot shows the 'Trigger Setup - hw_ilia_1' window with the Comparator Usage column enabled. The table now includes the 'Comparator Usage' column, showing values like '1 of 3' for the first row.

Name	Operator	Radix	Value	Port	Comparator Usage
U_SINEGEN/sine[19:0]	==	[B]	11XX_XX...	probe5[19:0]	1 of 3
GPIO_BUTTONS_db[1:0]	==	[H]	X	probe2[1:0]	
GPIO_BUTTONS_dly[1:0]	==	[H]	X	probe1[1:0]	
GPIO_BUTTONS_IBUF[1:0]	==	[H]	X	probe3[1:0]	

Using the ILA Default Dashboard

The ILA Dashboard (see the following figure) is a central location for all status and control information pertaining to a given ILA core. When an ILA core is first detected upon refreshing a hardware device, the Default ILA Dashboard for the core is automatically opened. If you need to manually open or re-open the dashboard, right-click the ILA core object in the Hardware window and select Default Dashboard.

Figure: ILA Dashboard



You can use the ILA Dashboard to interact with the ILA debug core in several ways:

- Set the trigger mode to trigger on various events in hardware:
 - BASIC_ONLY: The ILA Basic Trigger Mode can be used to trigger the ILA core when a basic AND/OR functionality of debug probe comparison result is satisfied.
 - ADVANCED_ONLY: The ILA Advanced Trigger Mode can be used to trigger the ILA core as specified by a user defined state machine.
 - TRIG_IN_ONLY: The ILA TRIG_IN Trigger Mode can be used to trigger the ILA core when the TRIG_IN pin of the ILA core transitions from a low to high.
 - BASIC_OR_TRIG_IN: The ILA BASIC_OR_TRIG_IN Trigger Mode can be used to trigger the ILA core when a logical OR-ing of the TRIG_IN pin of the ILA core and BASIC_ONLY trigger mode is desired.
 - ADVANCED_OR_TRIG_IN: The ILA ADVANCED_OR_TRIG_IN Trigger Mode can be used to trigger the ILA core when a logical OR-ing of the TRIG_IN pin of the ILA core and ADVANCED_ONLY trigger mode is desired.
- Set the trigger output mode.
- Use ALWAYS and BASIC capture modes to control filtering of data to be captured.
- Set the number of ILA capture windows.
- Set the data depth of the ILA capture window.
- Set the trigger position to any sample within the capture window.
- Monitor the trigger and capture status of the ILA debug core.

User-Defined Debug Probes

Using user-defined debug probes (also called hw_probes) in the Hardware Manager allows you the ability to create probes from combinations of physical ILA probe ports and constant values. You can use these probes in the Trigger Setup or Waveform windows in the Hardware Manager. On successful creation of these probes, they are listed in the Debug Probes window as part of the debug core they were associated with during creation.

You can create the following types of user defined probes:

- An ILA probe port.
- One or more constant values of 0 and/or 1.
- A mix of ILA probe port and constant values.

!! Important: User-defined probes that involve constant values can only be used in the Waveform window. They cannot be used in the Trigger Setup window.

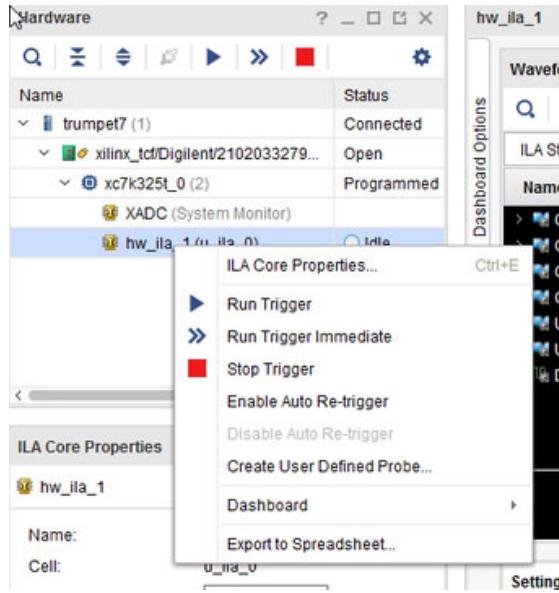
★ Tip: You can only create user-defined probes on ILA debug cores. Creating user-defined debug probes for VIO cores is not currently supported.

Creating a User-Defined Debug Probe

GUI Flow

To create a user-defined debug probe in the Vivado IDE Hardware Manager, in the Hardware window, right-click the ILA core that you want to probe and select Create User Defined Probe.

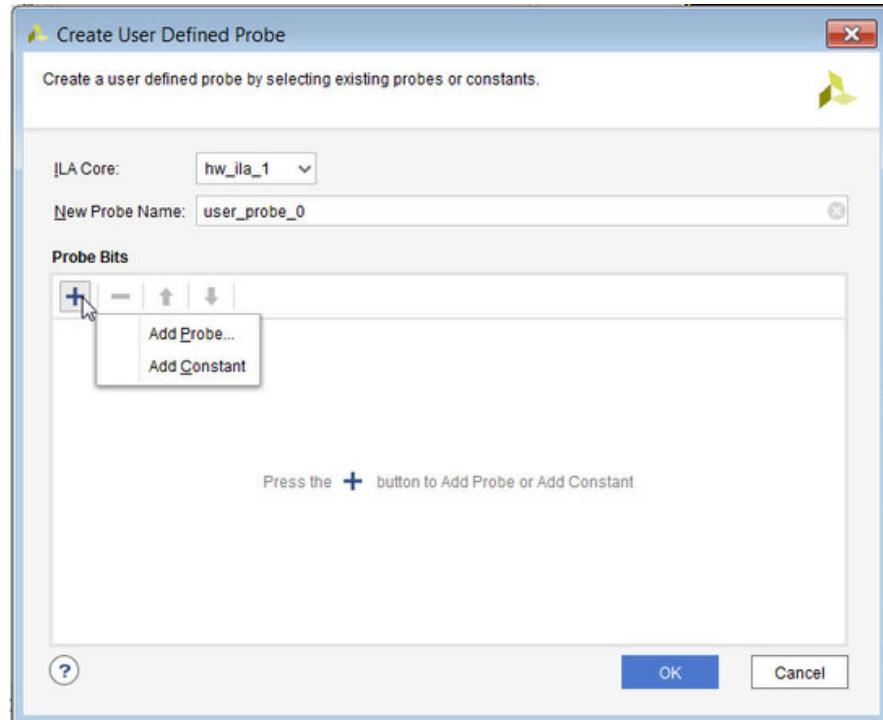
Figure: Selecting Create User Defined Probe



This brings up the Create User Defined Probe dialog box. Select the ILA core you want to create the probes on, the name of the new probe, and finally the probe bits, and/or constants that make up this new probe.

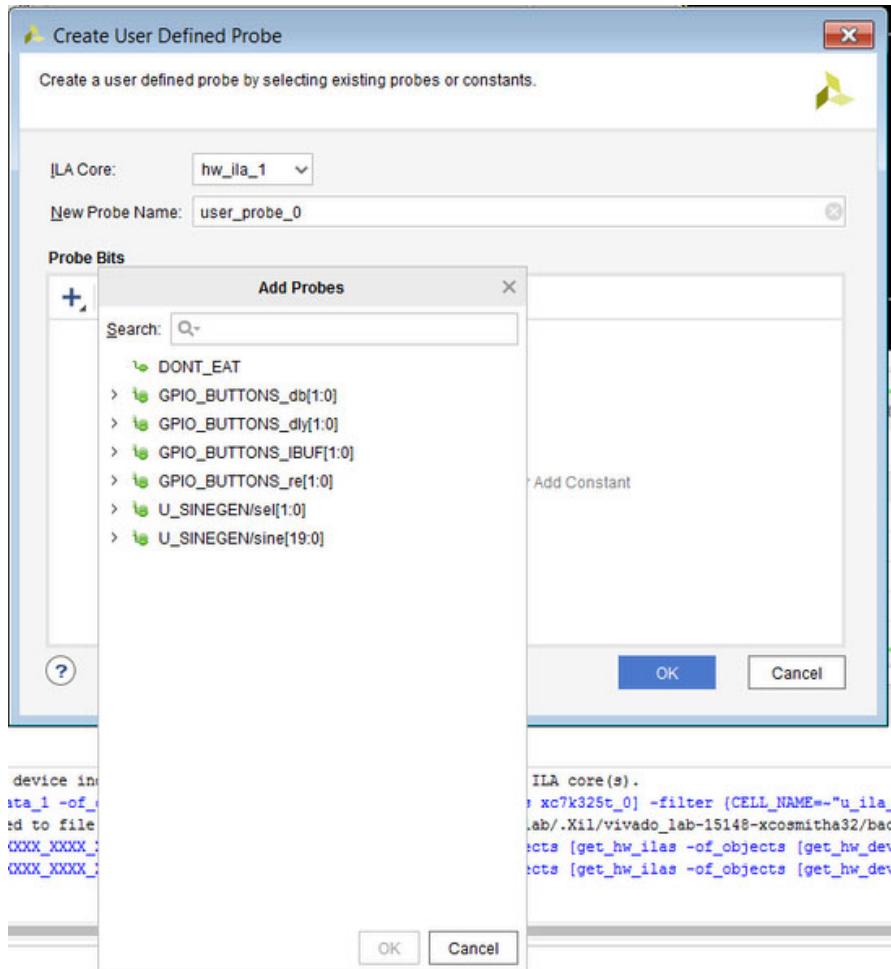
To add specific probe bits to this new probe, click the "+" button and select Add Probe.

Figure: Create User Defined Probe Dialog Box



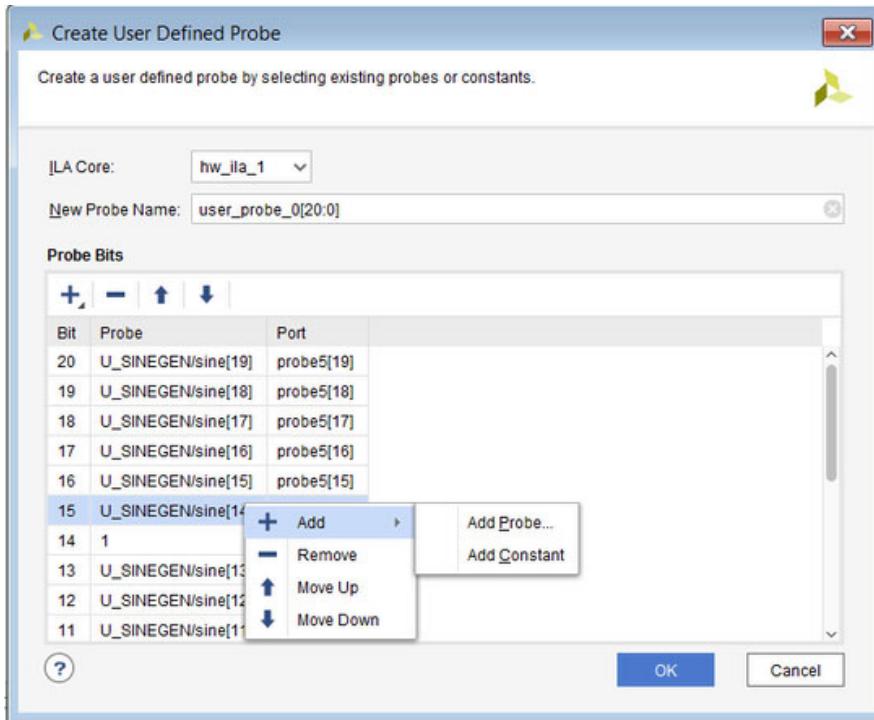
This brings up the Add Probes dialog that allows you to choose an existing probe or specific bits of an existing probe.

Figure: Add Probes Dialog Box



You can also add or remove bits in the Create User Defined dialog box. Move specific bits up or down as shown in the following figure.

Figure: Editing Bits in Create User Defined Probe Dialog Box



Tcl Flow

To create a user-defined debug probe use the `create_hw_probe` Tcl command.

```
create_hw_probe [-verbose] [-map <arg>] <name> <core>
```

Where:

name

is the name of the `hw_probe`. Must be unique for `hw_probes` belonging to the same device.
Bus probes must have their range specified. For example, `myNewProbe[31:0]`.

core

is the `hw_il` to associate the probe with.

-map

is the declaration of bits to base the user-defined probe on.

Examples of creating user-defined debug probes:

```
# Given a 512-bit counter "counterA[511:0)": Connects [255:223] to
#     ILA probe port 0 [31:0]
# Create a user-defined probe called foobar pointing at the
#     ILA buffer specified range [255:223]
create_hw_probe -map {probe0[31:0]} {foobar [255:223]} [get_hw_ilas hw_il_1]
# Constant only probe. NO triggering allowed on constant ONLY probes.
        create_hw_probe -map {0} {my_constant_gnd[0:0]} [get_hw_ilas hw_il_1]
# Create a user-defined probe as a mix of constants and ILA probe ports
```

```

create_hw_probe -map {0000 probe0[31:0] 1010} {my_mixed_probe[47:8]}
[get_hw_ilas
hw_ila_1]
# Creating scalar hw_probe called "foobar" from probe1:
create_hw_probe -map {probe1} foobar [get_hw_ilas hw_ila_1]
# Creating scalar hw_probe called "foobar" from bit 3 of probe0:
create_hw_probe -map {probe0[3]} foobar [get_hw_ilas hw_ila_1]
# Creating vector hw_probe called "foobar[0:0]" from probe1:
create_hw_probe -map {probe1} foobar[0:0] [get_hw_ilas hw_ila_1]
# Creating vector probe called "foobar[3:0]" from probe0:
create_hw_probe -map {probe0} foobar[3:0] [get_hw_ilas hw_ila_1]
# Creating vector probe called "foobar[3:2]" from probe0[1:0]:
create_hw_probe -map {probe0[1:0]} foobar[3:2] [get_hw_ilas hw_ila_1]

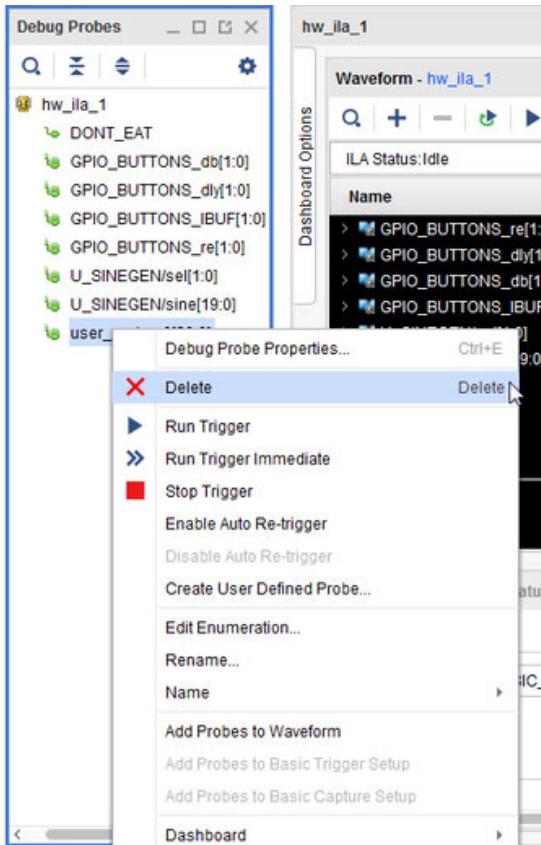
```

Deleting a User-Defined Debug Probe

GUI Flow

To delete a user defined probe in the Vivado IDE Hardware Manager, select Window > Debug Probe. This brings up the Debug Probes window beside the Hardware Manager dashboards. Right-click the appropriate probe in this window and click Delete as follows:

Figure: Deleting a Debug Probe



Tcl Flow

You can delete user-defined debug probes using the `delete_hw_probe` Tcl command.

For example, to delete a probe `foobar` created earlier using `create_hw_probe` do the following:

```
delete_hw_probe [get_hw_probes foobar -of_objects [get_hw_ilas -of_objects  
[get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"i_fast_ilा"}]]
```

Persistence of User-Defined Debug Probes

Any of the user-defined probes created in the Hardware Manager in the project flow are automatically persisted by the tool. The next time the project is opened, and you connect to the hardware target using Vivado Hardware Manager, these user-defined probes are resurrected. If these user-defined debug probes participated in Basic triggering or were added to the Waveform window, on the project open and connecting to the target in the Hardware Manager, you see the probe set up in all the windows exactly as when you closed the project earlier.

Interacting with a User-Defined Probe

Any user-defined debug probes created in the Hardware Manager can participate in Basic triggering, Advanced Triggering, and/or the Waveform window. The only exception is user-defined debug probes that involve constant values. These debug probes can only be used in the Waveform window.

Using Basic Trigger Mode

Use the basic trigger mode to describe a trigger condition that is a global Boolean equation of participating debug probe comparators. Basic trigger mode is enabled when the Trigger Mode is set to either `BASIC_ONLY` or `BASIC_OR_TRIG_IN`. Use the Basic Trigger Setup window (see the following figure) to create this trigger condition and debug probe compare values.

Figure: ILA Basic Trigger Setup Window



You can also use the `set_property` Tcl command to change the trigger mode of the ILA core. For instance, to change the trigger mode of ILA core `hw_ilা_1` to `BASIC_ONLY`, use the following command:

```
set_property CONTROL.TRIGGER_MODE BASIC_ONLY [get_hw_ilas hw_ilা_1]
```

Adding Probes to Basic Trigger Setup Window

The first step in using the basic trigger mode is to decide what ILA debug probes you want to participate in the trigger condition. Do this by selecting the desired ILA debug probes from the Debug Probes window and either right-click selecting Add Probes to Basic Trigger Setup or by dragging and dropping the probes into the Basic Trigger Setup window.

Note: You can drag-and-drop the first probe anywhere in the Basic Trigger Setup window, but you must drop the second and subsequent probes on top of the first probe. The new probe is always added on top of the previously added probe in the table. You can also use drag-and-drop operations in this manner to re-arrange probes in the table.

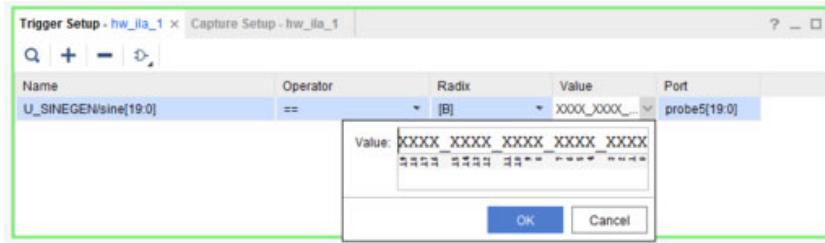
Important: Only probes that are in the Basic Trigger Setup window participate in the trigger condition. Any probes that are not in the window are set to "don't care" values and are not used as part of the trigger condition.

You can remove probes from the Basic Trigger Setup window by selecting the probe and hitting the Delete key or by right-click selecting the Remove option.

Setting Basic Trigger Compare Values

Use the ILA debug probe trigger comparators to detect specific equality or inequality conditions on the probe inputs to the ILA core. The trigger condition is the result of a Boolean "AND", "OR", "NAND", or "NOR" calculation of each of the ILA probe trigger comparator results. To specify the compare values for a given ILA probe, select the Value cell in for a given ILA debug probe in the Basic Trigger Setup window to open the (see the following figure).

Figure: ILA Probe Compare Value Dialog Box



Tip: Prior to changing the Radix ensure that the value is set to a string that applies to the new Radix.

ILA Probe Compare Value Settings

The Basic Trigger Setup window has three cells that you can configure in a specific row corresponding to a probe:

1. Operator: This is the comparison operator that you can set to the following values:

- == (equal)
- != (not equal)
- < (less than)
- <= (less than or equal)
- > (greater than)
- >= (greater than or equal)

2. Radix: This is the radix or base of the Value that you can set to the following values:

- [B] Binary
- [H] Hexadecimal
- [O] Octal
- [U] Unsigned Decimal
- [S] Signed Decimal

3. Value: This is the comparison value that is compared (using the Operator) with the real-time value on the net(s) in the design that are connected to the probe input of the ILA debug core. Depending on the Radix settings, the Value string is as follows:

- Binary
 - 0: logical zero
 - 1: logical one
 - X: don't care
 - R: rising or low-to-high transition
 - F: falling or high-to-low transition
 - B: either low-to-high or high-to-low transitions
 - N: no transition (current sample value is the same as the previous value)
- Hexadecimal
 - X: All bits corresponding to the value string character are "don't care" values
 - 0-9: Values 0 through 9
 - A-F: Values 10 through 15
- Octal
 - X: All bits corresponding to the value string character are "don't care" values
 - 0-7: Values 0 through 7
- Unsigned Decimal
 - Any non-negative integer value
- Signed Decimal
 - Any integer value

Setting Basic Trigger Condition

You can set up the trigger condition using the toolbar button on the left side of the Basic Trigger Setup window that has an icon in the shape of a logic gate on it (see the following figure). You can also use the `set_property` Tcl command to change the trigger condition of the ILA core:

```
set_property CONTROL.TRIGGER_CONDITION AND [get_hw_ilas hw_ila_1]
```

The meaning of the four possible values is shown in the following table.

Figure: Setting the Basic Trigger Condition

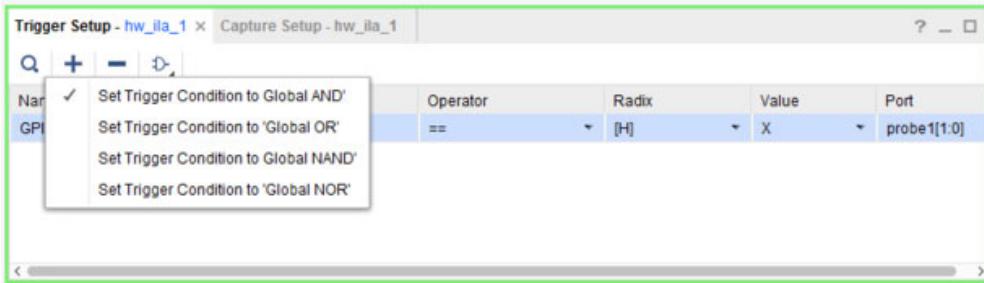


Table: Basic Trigger Condition Setting Descriptions

Trigger Condition Setting	QONTROL.TRIGGER_CONDITION Property Value	Condition Output
Global AND	AND	Trigger condition is "true" if all participating probe comparators evaluate "true", otherwise trigger condition is "false."
Global OR	OR	Trigger condition is "true" if at least one participating probe comparator evaluates "true", otherwise trigger condition is "false."
Global NAND	NAND	Trigger condition is "true" if at least one participating probe comparator evaluates "false", otherwise trigger condition is "false."
Global NOR	NOR	Trigger condition is "true" if all participating probe comparators evaluate "false", otherwise trigger condition is "false."

!! Important: If the ILA core has two or more debug probes that are concatenated together to share a single physical probe port of the ILA core, only the "Global AND" (AND) and "Global NAND" (NAND) trigger condition settings are supported. The "Global OR" (OR) and "Global NOR" (NOR) functions are not supported due to limitations of the probe port comparator logic. If you want to use the "Global OR" (OR) or "Global NOR" (NOR) trigger condition settings, make sure you assign each unique net or bus net to separate probe ports of the ILA core.

Using Advanced Trigger Mode

The ILA core can be configured at core generation or insertion time to have advanced trigger capabilities that include the following features:

- Trigger state machine consisting of up to 16 states.
- Each state can consist of one-, two-, or three-way conditional branching.
- Up to four counters can be used in a trigger state machine program to keep track of multiple events.
- Up to four flags can be used in a trigger state machine program to indicate when certain branches are taken.
- The state machine can execute "goto", "trigger", and various counter- and flag-related actions.

If the ILA core in the design that is running in the hardware device has advanced trigger capabilities, the advanced trigger mode features can be enabled by setting the Trigger mode control in the ILA Properties window of the ILA Dashboard to ADVANCED_ONLY or ADVANCED_OR_TRIG_IN.

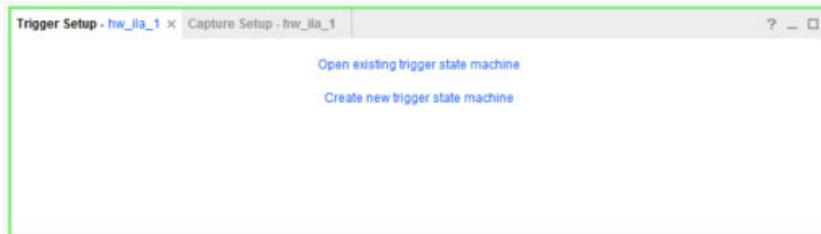
Specifying the Trigger State Machine Program File

When you set the Trigger mode to ADVANCED_ONLY or ADVANCED_OR_TRIG_IN, two changes happen in the ILA Dashboard:

1. A new control called Trigger State Machine appears in the ILA Properties window.
2. The Basic Trigger Setup window is replaced by a Trigger State Machine code editor window.

If you are specifying an ILA trigger state machine program for the first time, the Trigger State Machine code editor window appears as the one shown in the following figure.

Figure: Creating or Opening a Trigger State Machine Program File



To create a new trigger state machine, click the Create new trigger state machine link, otherwise click the Open existing trigger state machine link to open a trigger state machine program file (.tsm extension). You can also open an existing trigger state machine program file using the Trigger state machine text field and/or browse button in the ILA Properties window of the ILA Dashboard.

Editing the Trigger State Machine Program

If you created a new trigger state machine program file, the Trigger State Machine code editor window is populated with a simple trigger state machine by default (see the following figure).

Figure: Simple Default Trigger State Machine Program

Trigger Setup - hw_il_1.x

Trigger Setup - hw_il_1

?

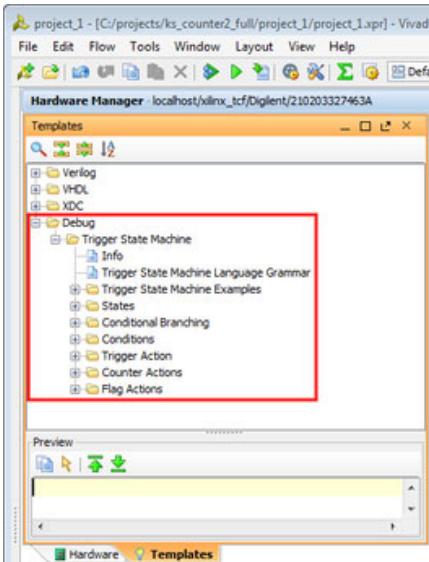
+ | - | X | Capture Setup - hw_il_1

1 #####
2 #
3 # For info on creating trigger state machines:
4 # 1) In the main Vivado menu bar, select
5 # Window > Language Templates
6 # 2) In the Templates window, select
7 # Debug > Trigger State Machine
8 # 3) Refer to the entry 'Info' for an overview
9 # of the trigger state machine language.
10 #
11 # More information can be found in this document:
12 #
13 # Vivado Design Suite User Guide: Programming
14 # and Debugging (UG905)
15 #
16 #####
17 state my_state0:
18 trigger;
19

The simple default trigger state machine program is designed to be valid for any ILA core configuration regardless of debug probe or trigger settings. This means that you can click the Run Trigger for the ILA core without modifying the trigger state machine program.

However, it is likely that you want to modify the trigger state machine program to implement some advanced trigger condition. The comment block at the top of the simple state machine shown in the previous figure gives some instructions on how to use the built-in language templates in the Vivado IDE to construct a trigger state machine program (see the following figure). A full description of the ILA trigger state machine language, including examples, is found in Trigger State Machine Language Description.

Figure: Trigger State Machine Language Templates



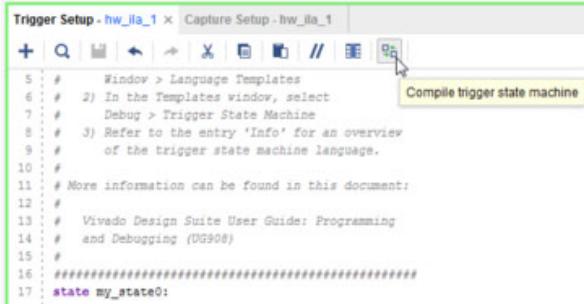
Related Information

Trigger State Machine Language Description

Compiling the Trigger State Machine

The trigger state machine is compiled every time you run the ILA trigger. If you would like to compile the trigger state machine without running or arming the ILA trigger, click the Compile trigger state machine button in the ILA Dashboard (see the following figure).

Figure: Compiling the Trigger State Machine without Arming the Trigger



Enabling Trigger In and Out Ports

The ILA core can be configured at core generation-time to have dedicated trigger input ports (TRIG_IN and TRIG_IN_ACK) and dedicated trigger output ports (TRIG_OUT and TRIG_OUT_ACK). If the ILA core has trigger input ports enabled, you can use the following Trigger Mode settings to trigger on events on the TRIG_IN port:

- BASIC_OR_TRIGGER_IN: used to trigger the ILA core when a logical OR-ing of the TRIG_IN pin of the ILA core and BASIC_ONLY trigger mode is desired.
- ADVANCED_OR_TRIGGER_IN: used to trigger the ILA core when a logical OR-ing of the TRIG_IN pin of the ILA core and ADVANCED_ONLY trigger mode is desired.
- TRIG_IN_ONLY: used to trigger the ILA core when the TRIG_IN pin of the ILA core transitions from a low to high.

If the ILA core has trigger output ports enabled, you can use the following TRIG_OUT Mode to control the propagation of trigger events to the TRIG_OUT port:

- DISABLED: disables the TRIG_OUT port.
- TRIGGER_ONLY: enables the result of the basic/advanced trigger condition to propagate to the TRIG_OUT port.
- TRIG_IN_ONLY: propagates the TRIG_IN port to the TRIG_OUT port.
- TRIGGER_OR_TRIGGER_IN: enables the result of a logical OR-ing of the basic/advanced trigger condition and TRIG_IN port to propagate to the TRIG_OUT port.

Configuring Capture Mode Settings

The ILA core can capture data samples when the core status is Pre-Trigger, Waiting for Trigger, or Post-Trigger (refer to the section called Viewing Trigger and Capture Settings for more details). The Capture mode control is used to select what condition is evaluated before each sample is captured:

ALWAYS

Store a data sample during a given clock cycle regardless of any capture conditions

BASIC

Store a data sample during a given clock cycle only if the capture condition evaluates "true"

You can also use the `set_property` Tcl command to change the capture mode of the ILA core. For instance, to change the capture mode of ILA core `hw_ila_1` to BASIC, use the following command:

```
set_property CONTROL.CAPTURE_MODE BASIC [get_hw_ilas hw_ila_1]
```

Related Information

[Viewing Trigger and Capture Status](#)

Using BASIC Capture Mode

Use the basic capture mode to describe a capture condition that is a global Boolean equation of participating debug probe comparators. Use the Basic Capture Setup window (see the following figure) to create this capture condition and debug probe compare values.

Figure: Setting the Basic Capture Condition



Configuring the Basic Capture Setup Window

The process for configuring debug probes and basic capture condition in the Basic Capture window is very similar to working with debug probes in the Basic Trigger Setup window:

- For information on adding probes to the Basic Capture Setup window, refer to the section called Adding Probes to Basic Trigger Setup Window.
- For information on setting the compare values on each probe in the Basic Capture Setup window, refer to the section called ILA Probe Compare Value Settings.
- For information on setting the basic capture condition in the Basic Capture Setup window, refer to the section called Setting Basic Trigger Condition. One key difference is the ILA core property used to control the capture condition is called `CONTROL.CAPTURE_CONDITION`.

Related Information

[Adding Probes to Basic Trigger Setup Window](#)

[Setting Basic Trigger Condition](#)

[ILA Probe Compare Value Settings](#)

Setting the Number of Capture Windows

The ILA capture data buffer can be subdivided into one or more capture windows. The depth each window is a power of 2 number of samples from 1 to (((buffer size) / (number of windows)) - 1). For example, if the ILA data buffer is 1024 samples deep and is segmented into four capture windows, each window can be up to 256 samples deep. Each capture window has its own trigger mark corresponding to the trigger event that caused the capture window to fill.

 **Note:** An ILA buffer size equal to the number of capture windows is not supported.

★ Tip: Click Stop Trigger before the entire ILA data capture buffer is full to upload and display all capture windows that are filled. For example, if the ILA data buffer is segmented into four windows, and three of them have filled with data, click Stop Trigger to halt the ILA core and upload and display the three filled capture windows.

The following table illustrates the interaction between the Vivado runtime software and hardware when you set the Number of Capture Windows to more than 1 and the Trigger Position to 0.

Table: Number of Capture Windows > 1 and Trigger Position = 0

Software	Hardware
User Runs Trigger on the ILA core	Window 0: ILA is armed Window 0: ILA triggers Window 0: ILA fills capture window 0 Window 1: ILA is rearmed Window 1: ILA triggers Window 1: ILA fills capture window 1 Window n-1: ILA is rearmed Window n-1: ILA triggers Window n-1: ILA fills capture window n Entire ILA Capture Buffer is Full
Data is uploaded and displayed	The ILA core is rearmed such that it is ready to trigger on the clock cycle immediately following the last sample captured of the previous window.

The following table illustrates the interaction between the Vivado runtime software and hardware when you set the Number of Capture Windows to more than 1 and the Trigger Position to greater than 0.

Table: Number of Capture Windows > 1 and Trigger Position > 0

Software	Hardware
----------	----------

Software	Hardware
User Runs Trigger on the ILA core	Window 0: ILA is armed Window 0: ILA fills capture buffer up to trigger position Window 0: ILA triggers Window 0: ILA fills the rest of capture window 0 Window 1: ILA is rearmed Window 1: ILA fills capture buffer up to trigger position Window 1: ILA triggers Window 1: ILA fills capture buffer Window 1: ILA fills window 1 Window n-1: ILA is rearmed Window n-1: ILA fills capture buffer up to trigger position Window n-1: ILA triggers Window n-1: ILA fills capture buffer Window n-1: ILA fills window n Entire ILA Capture Buffer is Full
Data is uploaded and displayed	Triggers could be missed between two windows because the ILA now has to fill the capture data up to the trigger position.

Setting the Trigger Position in the Capture Window

Use the Trigger position control in the Capture Mode Settings window (or the Trigger Position property in the ILA Core Properties window) to set the position of the trigger marker in the captured data window. You can set the trigger position to any sample number in the captured data window. For instance, in the case of a captured data window that is 1024 samples deep:

- Sample number 0 corresponds to the first (left-most) sample in the captured data window.
- Sample number 1023 corresponds to the last (right-most) sample in the captured data window.
- Samples numbers 511 and 512 correspond to the two "center" samples in the captured data window.

You can also use the `set_property` Tcl command to change the ILA core trigger position:

```
set_property CONTROL.TRIGGER_POSITION 512 [get_hw_ilas hw_ila_1]
```

Setting the Data Depth of the Capture Window

Use the Data Depth control in the Capture Mode Settings window (or the Capture data depth property in the ILA Core Properties window) to set the data depth of the ILA core's captured data

window. You can set the data depth to any power of two from 1 to the maximum data depth of the ILA core (set during core generation or insertion time).

Note: Refer to [Setting the Data Depth of the Capture Window](#) for more details on how to set the maximum capture buffer data depth on ILA cores that are added to the design using the Netlist Insertion probing flow.

You can also use the `set_property` Tcl command to change the ILA core data depth:

```
set_property CONTROL.DATA_DEPTH 512 [get_hw_ilas hw_il_1]
```

Related Information

[Modifying Properties on the Debug Cores](#)

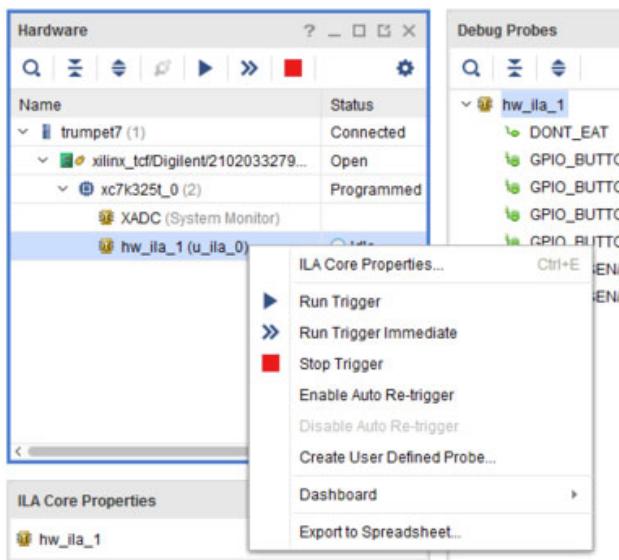
Running the Trigger

You can run or arm the ILA core trigger in two different modes:

- Run Trigger: Selecting the ILA core(s) to be armed followed by clicking the Run Trigger button on the ILA Dashboard or Hardware window toolbar arms the ILA core to detect the trigger event that is defined by the ILA core basic or advanced trigger settings.
- Run Trigger Immediate: Selecting the ILA core(s) to be armed followed by clicking the Run Trigger Immediate button on the ILA Dashboard or Hardware window toolbar arms the ILA core to trigger immediately regardless of the ILA core trigger settings. This command is useful for detecting the "aliveness" of the design by capturing any activity at the probe inputs of the ILA core.

You can also arm the trigger by selecting and right-clicking on the ILA core and selecting Run Trigger or Run Trigger Immediate from the pop-up menu (see the following figure).

Figure: ILA Core Trigger Commands



★ Tip: You can run or stop the triggers of multiple ILA cores by selecting the desired ILA cores, using the Run Trigger, Run Trigger Immediate, or Stop Trigger buttons in the Hardware window

toolbar. You can also run or stop the triggers of all ILA cores in a given device by selecting the device in the Hardware window and click the appropriate button in the Hardware window toolbar.

Stopping the Trigger

You can stop the ILA core trigger by selecting the appropriate ILA core followed by clicking on the Stop Trigger button on the ILA Dashboard or Hardware window toolbar. You can also stop the trigger by selecting and right-clicking on the appropriate ILA core(s) and selecting Stop Trigger from the popup menu (see Running the Trigger).

Related Information

[Running the Trigger](#)

Using Auto Re-Trigger

Select the Enable Auto Re-Trigger right-click menu option (or corresponding button on the ILA Dashboard toolbar) on the ILA core to enable Vivado IDE to automatically re-arm the ILA core trigger after a successful trigger+upload+display operation has completed. The captured data displayed in the waveform viewer corresponding to the ILA core is overwritten upon each successful trigger event. The Auto Re-Trigger option can be used with the Run Trigger and Run Trigger Immediate operations. Click Stop Trigger to stop the trigger currently in progress.

The following table illustrates the interaction between the Vivado IDE runtime software and hardware when you invoke the Auto Re-Trigger option.

Table: Auto Re-Trigger

Software	Hardware
Click the Auto Re-trigger option on the ILA core	ILA is armed ILA triggers ILA fills capture buffer ILA is full
Data is uploaded and displayed	ILA is rearmed ILA triggers ILA fills capture buffer ILA is full Lots of cycles are missed between the ILA "full" event and display of the ILA data

!! Important: As there is a delay between the time the ILA data is full and the data is uploaded and displayed in the GUI, there is a very high probability of missing cycles between these events where the ILA could have triggered.

Viewing Trigger and Capture Status

The ILA debug core trigger and capture status is displayed in two places in Vivado IDE:

- In the Hardware window Status column of the row(s) corresponding to the ILA debug core(s).
- In the Trigger Capture Status window of the ILA Dashboard.

The Status column in the Hardware window indicates the current state or status of each ILA core (see the following table).

Table: ILA Core Status Description

ILA Core Status	Description
Idle	The ILA core is idle and waiting for its trigger to be run. If the trigger position is 0, the ILA core transitions to the Waiting for Trigger status once the trigger is run, otherwise the ILA core transitions to the Pre-Trigger status.
Pre-Trigger	The ILA core is capturing pre-trigger data into its data capture window. Once the pre-trigger data has been captured, the ILA core transitions to the Waiting for Trigger status.
Waiting for Trigger	The ILA core trigger is armed and is waiting for the trigger event to occur as described by the basic or advanced trigger settings. Once the trigger occurs, the ILA core transitions to the Full status if the trigger position is set to the last data sample in the capture window, otherwise it transitions to the Post-Trigger status.
Post-Trigger	The ILA core is capturing post-trigger data into its data capture window. Once the post-trigger data has been captured, the ILA core transitions to the Full status.
Full	The ILA core capture buffer is full and is being uploaded to the host for display. The ILA core transitions to the Idle status once the data has been uploaded and displayed.

The contents of the Trigger Capture Status window in the ILA Dashboard depend on the Trigger Mode setting of the ILA core.

Partial Buffer Capture

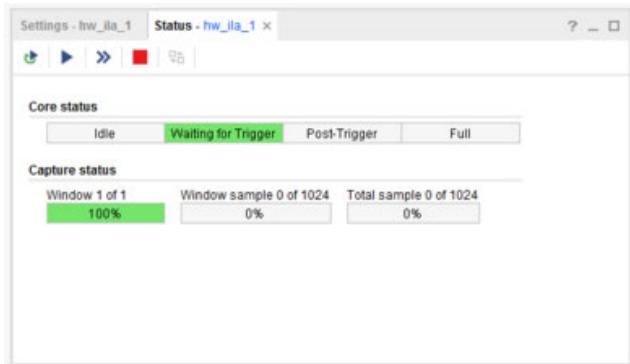
Clicking Stop Trigger before the entire ILA data capture buffer is full uploads and displays all captured windows. For example, if the ILA data buffer is segmented into four windows and three have filled with data, clicking Stop Trigger halts the ILA core and uploads and displays the three filled capture windows. In addition, clicking Stop Trigger halts the ILA core and displays a partially filled capture window so long as the trigger event occurred in that capture window.

Basic Trigger Mode Trigger and Capture Status

The Trigger Capture Status window contains two status indicators when the Trigger Mode is set to BASIC (see the following figure):

- Core status: indicates the status of the ILA core trigger/capture engine (see Viewing Trigger and Capture Status for a description of the status indicators).
- Capture status: indicates the current capture window, the current number of samples captured in the current capture window, and the total number of samples captured by the ILA core. These values are all reset to 0 once the ILA core status is Idle.

Figure: Basic Trigger Mode Trigger Capture Status Window



Related Information

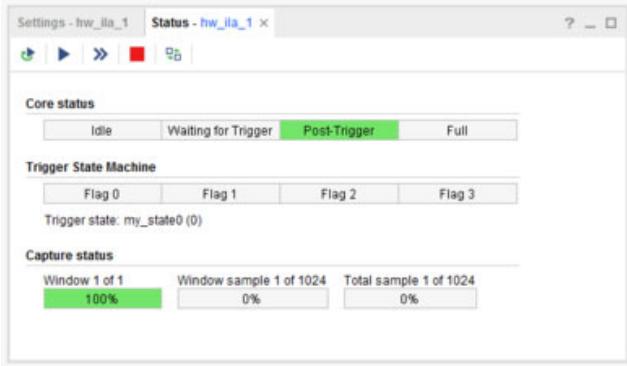
[Viewing Trigger and Capture Status](#)

[Advanced Trigger Mode Trigger and Capture Status](#)

The Trigger Capture Status window contains four status indicators when the Trigger Mode is set to ADVANCED (see the following figure):

- Core status: indicates the status of the ILA core trigger/capture engine (see Viewing Trigger and Capture Status for a description of the status indicators).
- Trigger State Machine Flags: indicates the current state of the four trigger state machine flags.
- Trigger State: when the core status is Waiting for Trigger, this field indicates the current state of the trigger state machine.
- Capture status: indicates the current capture window, the current number of samples captured in the current capture window, and the total number of samples captured by the ILA core. These values are all reset to 0 once the ILA core status is Idle.

Figure: Advanced Trigger Mode Trigger Capture Status Window



Related Information

[Viewing Trigger and Capture Status](#)

Writing ILA Probes Information

The ILA Cores tab view in the Debug Probes window contains information about the nets that you probed in your design using the ILA core. This ILA probe information is extracted from your design and is stored in a data file that typically has a .ltx file extension.

Normally, the ILA probe file is automatically created during bitstream generation. However, you can also use the `write_debug_probes` Tcl command to write out the debug probes information to a file:

1. If you are in project mode, open Implemented Design. If you are in non-project mode, open the implemented design checklist.
2. Run the `write_debug_probes filename.ltx` Tcl command.

Reading ILA Probes Information

The ILA probe file is automatically associated with the FPGA or adaptive SoC hardware device if the probes file is called `debug_nets.ltx` and is found in the same directory as the bitstream programming (.bit) file that is associated with the device.

You can also specify the location of the probes file:

1. Select the FPGA or adaptive SoC in the Hardware window.
2. Set the Probes file location in the Hardware Device Properties window.
3. Click Apply to apply the change.

You can also set the location using the `set_property` Tcl command:

```
set_property PROBES.FILE {C:/myprobes.ltx} [lindex [get_hw_devices] 0]
```

Viewing Captured Data from the ILA Core in the Waveform Viewer

Once the ILA core captured data has been uploaded to the Vivado IDE, it is displayed in the Waveform Viewer. See Viewing ILA Probe Data for details on using the Waveform Viewer to view captured data from the ILA core.

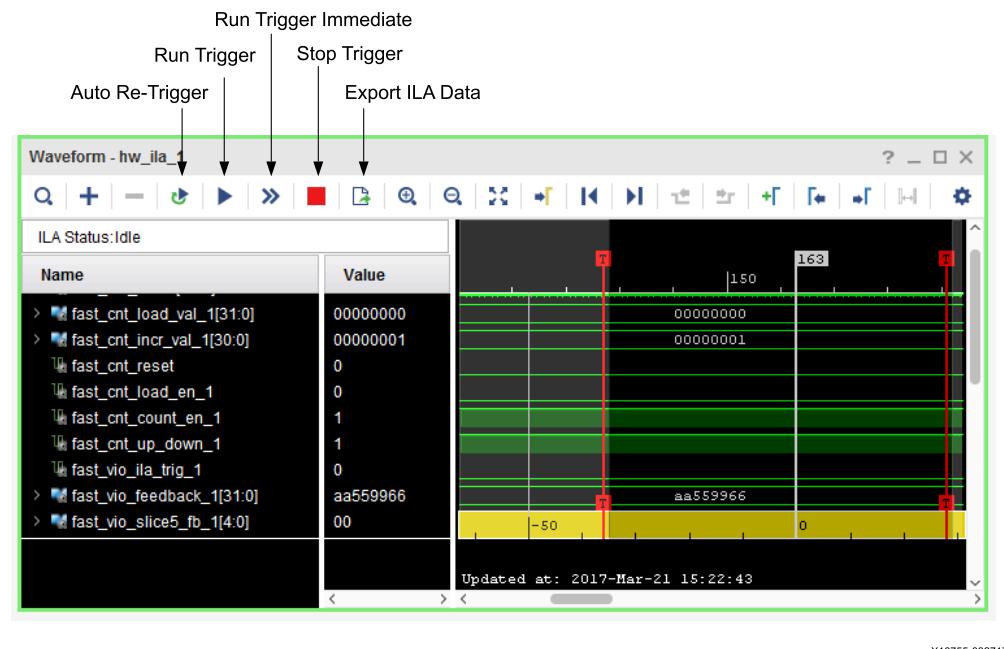
Related Information

[Viewing ILA Probe Data in the Waveform Viewer](#)

Using Waveform ILA Trigger and Export Features

You can use the icons in the waveform window to arm ILAs and run trigger, stop the trigger, and export ILA data as follows:

Figure: Waveform ILA Trigger and Export Features



Enable Auto Re-Trigger: Select the Enable Auto Re-Trigger button on the Waveform window toolbar to enable Vivado IDE to automatically re-arm the ILA core associated with the Waveform window trigger after a successful trigger+upload+display operation has completed.

The captured data displayed in the Waveform window corresponding to the ILA core is overwritten upon each successful trigger event. The Auto Re-Trigger option can be used with the Run Trigger and Run Trigger Immediate operations. Click the Stop Trigger button to stop the trigger currently in progress.

Run Trigger: Arms the ILA core associated with the Waveform window to detect the trigger event that is defined by the ILA core basic or advanced trigger settings.

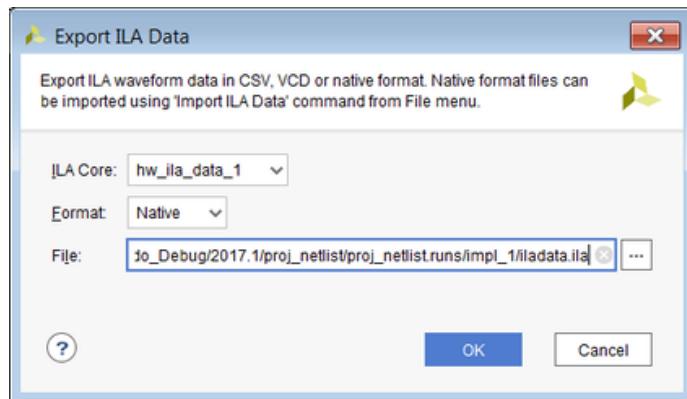
Run Trigger Immediate: Arms the ILA core associated with the Waveform window to trigger immediately regardless of the ILA core trigger settings. This command is useful for detecting the

"aliveness" of the design by capturing any activity at the probe inputs of the ILA core.

Stop Trigger: Stops the ILA core trigger of the ILA associated with the Waveform window.

Export ILA Data: Captures data from an ILA core and saves it to a file. The data can be captured in either native, .csv, or .vcd format. Clicking this icon on the Waveform window toolbar, the following dialog box appears.

Figure: Export ILA Data Dialog Box



The ILA core is the name of the ILA debug core to export data for. The format is a selection among Native, CSV, and VCD formats.

- Native format configures the `write_hw_iladata` command to export the ILA data in native ILA format.

This ILA file can be imported back into Vivado IDE so that you can view previously captured ILA data. The `read_hw_iladata` Tcl command can be used to import the ILA data back into Vivado IDE as shown in the following example:

```
read_hw_iladata {./iladata.ila}
display_hw_iladata
```

- CSV format configures the `write_hw_iladata` command to export the ILA data in the form of a .csv file that can be used to import the data into a spreadsheet or third-party application.
- VCD file format configures the `write_hw_iladata` command to export the ILA data in the form of a .vcd file that can be used to import into a third-party application or viewer.

!! Important: While ILA data can be exported in the CSV, VCD, and native ILA format, only the native ILA format can be imported into Vivado. Also, native ILA data imported into Vivado is supported only for offline viewing of previously captured data. The probe signals cannot be used for other purposes, such as triggering, and so on.

Saving and Restoring Captured Data from the ILA Core

In addition to displaying the captured data directly uploaded from the ILA core, you can also write the captured data to a file, read the data from a file, and display it in the waveform viewer.

Saving Captured ILA Data to a File

Currently, the only way to upload captured data from an ILA core and save it to a file is to use the following Tcl command:

```
write_hw_ila_data my_hw_ila_data_file.ila [upload_hw_ila_data hw_ila_1]
```

This Tcl command sequence uploads the captured data from the ILA core and writes it to an archive file called `my_hw_ila_data_file.ila`. The archive file contains the waveform database file, the waveform configuration file, a waveform comma separated value file, and a debug probes file.

★ **Tip:** Use the `-csv_file` option to generate a comma-separated values (CSV) file. This configures the `write_hw_ila_data` command to export the ILA data in the form of a CSV file that can be used to import into a spreadsheet or third-party application, rather than the default binary ILA file format.

★ **Tip:** Use the `-vcd_file` option to generate a value change dump (VCD) file. This configures the `write_hw_ila_data` command to export the ILA data in the form of a VCD file that can be used to import into a third-party application or viewer, rather than the default binary ILA file format.

Probe Data Radix

Every probe has a `DISPLAY_RADIX` property associated with it. This property is set to HEX for a multi-bit probe and BINARY for a one-bit probe by default. The exported probe data in the .csv files use probe radix.

You can change the `DISPLAY_RADIX` property of all the probes of all ILAs in the design as follows in the Vivado Hardware Manager Tcl Console:

```
foreach probe [get_hw_probes -of [get_hw_ilas]] {  
    set_property DISPLAY_RADIX binary $probe  
    set_property DISPLAY_AS_ENUM false $probe  
}
```

✎ **Note:** Here you are changing the radix of all the probes in all the ILAs to BINARY. To change the radix to HEX, use the following script:

```
foreach probe [get_hw_probes -of [get_hw_ilas]] {  
    set_property DISPLAY_RADIX hex $probe  
    set_property DISPLAY_AS_ENUM false $probe  
}
```

Listing Data Samples Associated with a Single Probe

You can also list data samples associated with individual probes using the `list_hw_samples` Tcl command.

An example of listing the samples associated with probe `fast_cnt_incr_val_1` on ILA named `i_fast_il`a is as follows:

```
list_hw_samples [get_hw_probes fast_cnt_incr_val_1 -of_objects [get_hw_ilas
-of_objects [get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"i_fast_ila"}]]
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001...
00000001...
```

Restoring Captured ILA Data from a File

Currently, the only way to restore captured data from a file and display it in the waveform viewer is to use the following Tcl command:

```
display_hw_ila_data [read_hw_ila_data my_hw_ila_data_file.ila]
```

This Tcl command sequence reads the previously saved captured data from the ILA core and displays it in the Waveform window.

Note: The waveform configuration settings (dividers, markers, colors, probe radices, etc.) for the ILA Data Waveform window is also saved in the ILA captured data archive file. Restoring and displaying any previously saved ILA data uses these stored waveform configuration settings.

Important: Do NOT use the open_wave_config command to open a waveform created from ILA captured data. This simulator-only command does not function correctly with ILA-captured data waveforms.

Enumeration of Probe Values

This feature provides a way to refer to probe values, both during Trigger/Capture Setup to compare values, and in the Waveform window, by symbolic names.

Some common use cases include the following.

- state machine state values
- op codes
- any probe match value, that you want to refer to by name, instead of value.

The process involves entering enumeration name-value pairs and associating them with a probe. The enumeration name-value pair associations are available using Tcl commands and are stored between sessions.

In cases where the mark_debug attribute is used to probe a state register in a state machine with enumerated states, the state enumeration is preserved through implementation, and displayed automatically in the Waveform window.

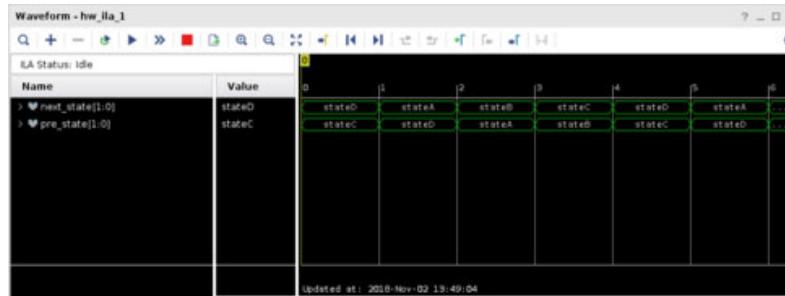
Figure: RTL with State Encoding

```

13 :
14 : (* mark_debug = "true" *) reg [1:0] pre_state;
15 : (* mark_debug = "true" *) reg [1:0] next_state;
16 :
17 : localparam stateA = 2'b00;
18 : localparam stateB = 2'b01;
19 : localparam stateC = 2'b10;
20 : localparam stateD = 2'b11;
21 :
22 :

```

Figure: State Encoding Preserved and Displayed as a Probe Enumeration in the Waveform Viewer



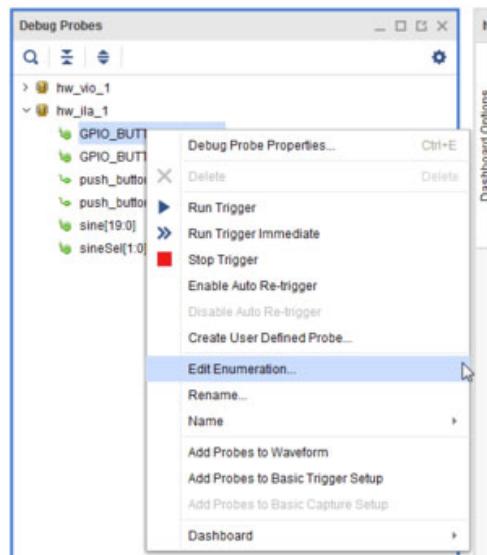
Probe compare values can be set using enumeration names in the Trigger Setup and Capture Control Setup windows. Probes and their enumeration names corresponding to values can be displayed in the Waveform window as well.

Add/Edit Enumerations

Define New Enumerations Using the Hardware Manager

To associate a new enumeration name-value pair to a debug probe, right-click the debug probe either in the Debug Probes window or the Trigger/Capture Setup window and select Edit Enumeration.

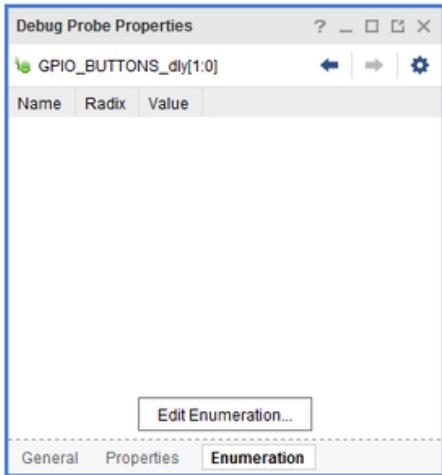
Figure: Edit Enumeration from Trigger Setup Window



Editing Enumeration Associated with a Debug Probe in the Trigger Setup Window

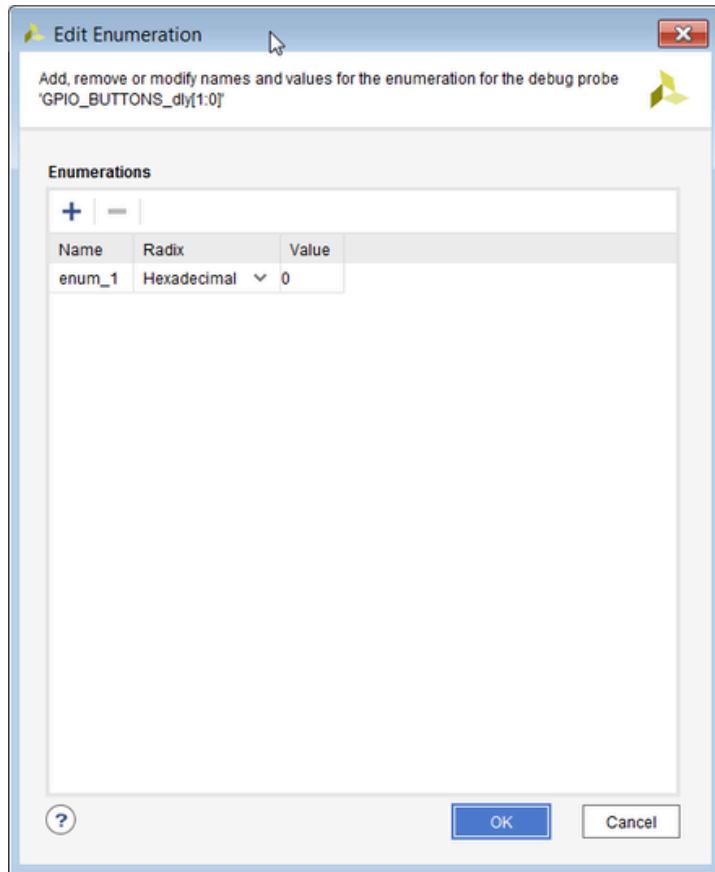
You can also associate a new enumeration name-value pair to a debug probe by selecting the debug probe in the Debug Probes window or Trigger Setup window. The Enumeration tab of the Debug Probe Properties window also allows you to associate a new enumeration name-value pair to the probe.

Figure: Debug Probe Properties Dialog



Click Edit Enumeration to open the Edit Enumeration dialog box.

Figure: Edit Enumeration Dialog



Select the name-value pair and use the "+" and "-" buttons on the left to add or delete enumerations. You can change the name, radix, and values fields in the table.

Add Enumerations Using Tcl Commands

The `add_hw_probe_enum` command, associates an enumeration name-value pair to a debug probe. You can add `add_hw_probe` commands to a Tcl file, to have the definitions appear in a separate file. The enumeration names maintain the case they were entered in, but lookup is case-insensitive.

Example:

```
set probe [get_hw_probes fast_cnt_count -of_objects [get_hw_ilas -of_objects  
[get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"i_fast_ilा"}]]  
add_hw_probe_enum ZERO eq5'h00 $probe  
add_hw_probe_enum TWELVE eq5'u12 $probe  
add_hw_probe_enum THIRTEEN eq5'u13 $probe  
add_hw_probe_enum FOURTEEN eq5'u14 $probe  
add_hw_probe_enum FIFTEEN eq5'u15 $probe  
add_hw_probe_enum SIXTEEN eq5'u16 $probe  
add_hw_probe_enum SEVENTEEN eq5'u17 $probe
```

Delete Enumerations Using Tcl Commands

Use the `remove_hw_probe_enum` command to remove explicitly named enumerations entries, or all enumerations for a `hw_probe`.

Example:

```
remove_hw_probe_enum -list {zero } [get_hw_probes U_SINEGEN/sel -of_objects  
[get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter  
{CELL_NAME=~"u_ila_0"}]]
```

★ Tip: Using the `-remove_all` option on `remove_hw_probe_enum` removes all of the enumerations associated with the probe.

Access Enumeration

Enumerations are stored as `hw_probe` properties, so `set_property`, `get_property`, and `report_property` commands can be used on these properties.

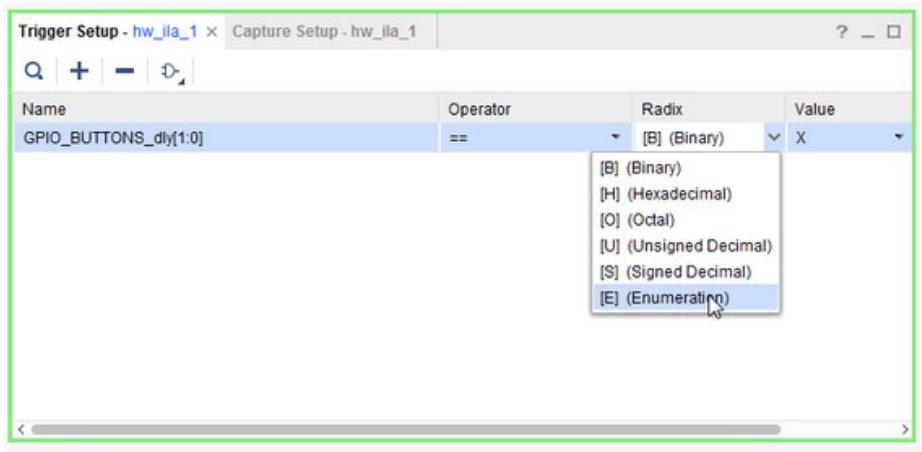
The enumeration properties have a prefix `ENUM`, which needs to be used when using it with `set_property` and `get_property` commands. See the following example.

```
get_property ENUM.FIFTEEN -of_objects [get_hw_ilas -of_objects [get_hw_devices  
xc7k325t_0] -filter {CELL_NAME=~"i_fast_ilas"}]]  
eq5'u15  
report_property [get_hw_probes fast_vio_slice5_fb -of_objects [get_hw_ilas  
hw_ilas_1]] ENUM*  
Property      Type  Read-only Visible Value  
ENUM.FIFTEEN   string true    true   eq5'u15  
ENUM.FOURTEEN   string true    true   eq5'u14  
ENUM.SEVENTEEN   string true    true   eq5'u17  
ENUM.SIXTEEN    string true    true   eq5'u16  
ENUM.THIRTEEN   string true    true   eq5'u13  
ENUM.TWELVE     string true    true   eq5'u12  
ENUM.ZERO       string true    true   eq5'h00  
set_property ENUM.FIFTEEN eq5'h0F [get_hw_probes fast_vio_slice5_fb -of_objects  
[get_hw_ilas hw_ilas_1]]
```

Using Enumerations in Trigger Setup Window

Setting compare values with enumerations in the Trigger Setup window, has similar syntax to numeric compare values. The radix char is 'e'. Only the operators 'eq' and 'neq' are supported for enumeration compare values.

Figure: Enumerations in Trigger Setup Window

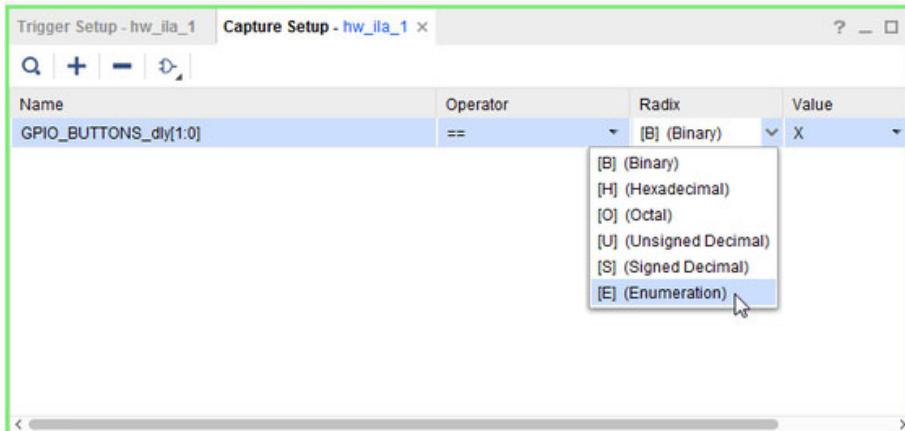


```
set_property TRIGGER_COMPARE_VALUE eq2'ethree [get_hw_probes U_SINEGEN/selected_of_objects [get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"u_il_ila_0"}]]
```

Using Enumerations in Capture Setup Window

You can also compare values using enumeration in the Capture Setup window in the Vivado IDE or using Tcl commands.

Figure: Enumerations in Capture Setup Window



```
set_property CAPTURE_COMPARE_VALUE eq2'eone [get_hw_probes locked -of_objects [get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"u_il_ila_0_0"}]]
```

Advanced Trigger

You can compare values using enumeration in the Advanced Trigger State Machine scripts as in the following example.

```

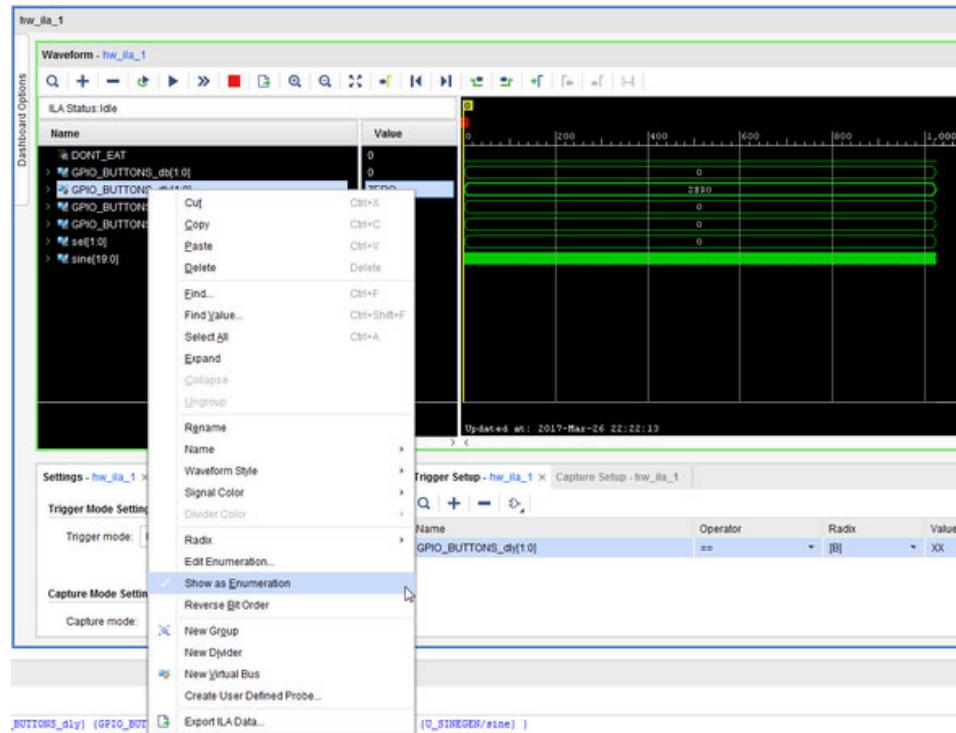
state my_state0:
if (fast_ila_slice5_fb == 5'eFIFTEEN) then
    set_flag $flag1;
    goto my_state0;
elseif (fast_ila_slice5_fb == 5'eTWELVE) then
    trigger;
else
    clear_flag $flag1;
    goto my_state0;
endif

```

Using Enumerations in the Waveform Window

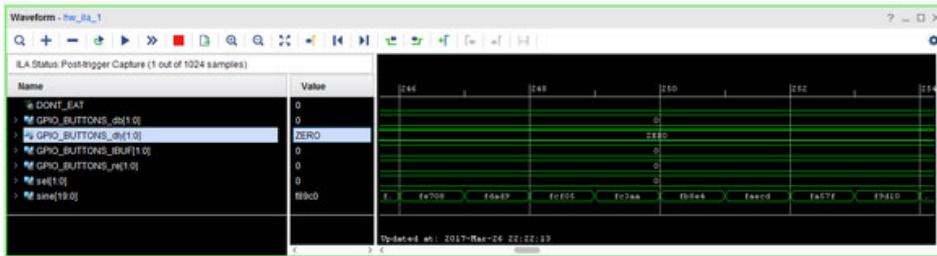
You can show enumerations in the Waveform window by choosing the Show as Enumeration option for each signal. Right-click the signal in the Waveform window and select Show as Enumeration in the menu that appears. When not shown as an enumeration, bus values are displayed according to regular radix selection.

Figure: Show as Enumeration Option in the Waveform Window



Enumeration information is saved to waveform data files and is used in subsequent displays of waveform data. The default for waveform probes that have Enumerations defined is to have the Enumerations displayed.

Figure: Waveforms with Enumerations



When a waveform object has Show as Enumeration selected, enumeration names are displayed. If there is no matching Enumeration for the waveform value, it is displayed according to the selected radix.

!! Important: If the waveform has been created prior to creating the enumerations, you can apply new enumerations to the waveform by saving the waveform ILA data using the Tcl commands as follows:

```
write_hw_iladata -force data_ilad3.ilad [upload_hw_iladata hw_ilad3]
display_hw_iladata [read_hw_iladata ./data_ilad3.ilad]
```

Debugging AXI Interfaces in the Hardware Manager

The System ILA IP in IP integrator allows you to perform in-system debugging of designs on an FPGA. On Versal devices, the System ILA core is obsolete. Interface debugging is now supported in the standard ILA with AXIS interface. Use this feature when there is a need to monitor interfaces and signals in the IP integrator Block Design.

See this [link](#) in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator (UG994)* for the steps to debug interfaces and/or nets in the Block Design.

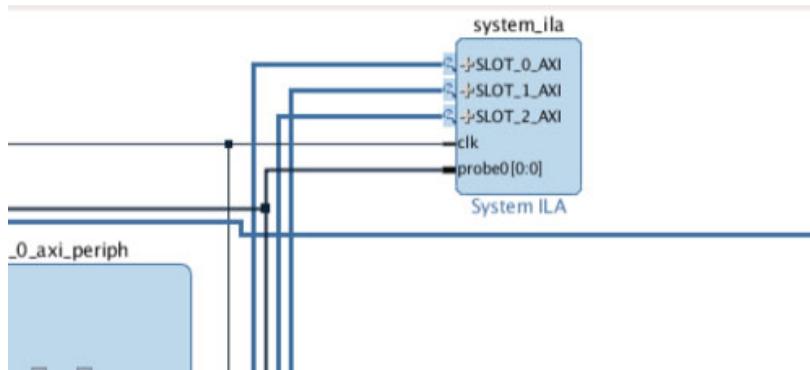
If you have instantiated System ILA debug cores in your IP integrator Block Design, you can debug and monitor AXI transactions and their corresponding read and write events in the waveform window.

 **Note:** For Non-Versal architectures, System ILA must be used for interface debugging.

Waveform and AXI Interfaces

The System ILA debug core enables you to debug and monitor interfaces as slots. Each slot corresponds to an interface being debugged in IP integrator Block Design. In the following figure, there are two AXI Memory Map interfaces being probed by the System ILA IP in slot 0, slot 1.

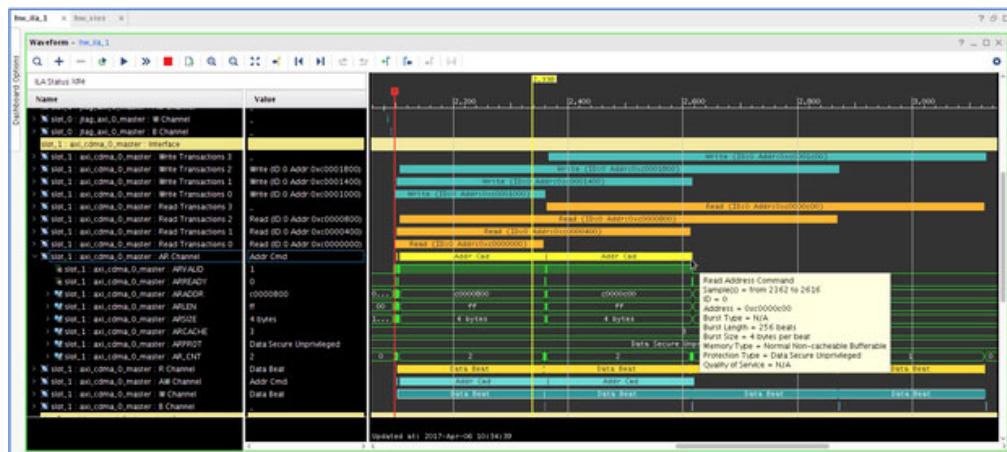
Figure: Probing 2 AXI Memory Map Interfaces



AXI Transactions in the Waveform Viewer

Transactions associated with AXI3, AXI4, and AXI4-Lite interfaces that are being debugged by the System ILA can be viewed in the waveform viewer as shown in the following figure

Figure: AXI Transactions in the Waveform Viewer



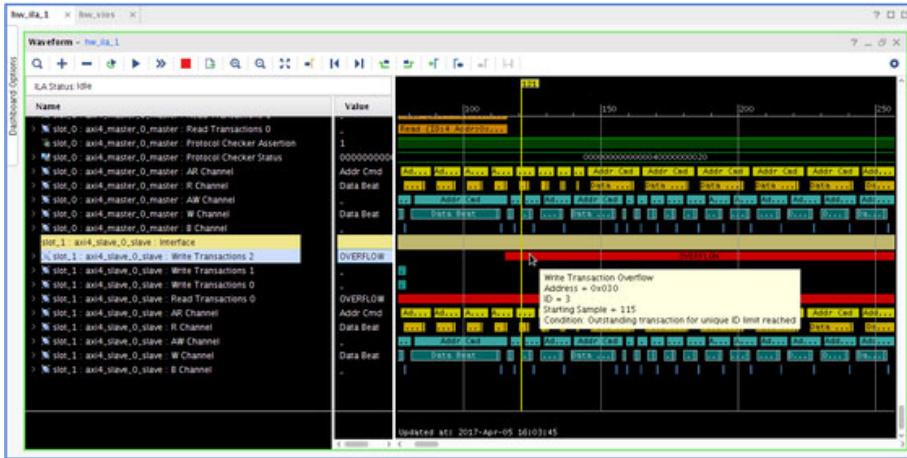
AXI transactions are defined as follows:

- Read transactions start with the beginning of the Address Command event on the AR (Read Address) channel.
- Read transactions end with the Last Read Data event on the R (Read Data) channel.
- Write transactions start with the beginning of the Address Command event on the AW (Write Address) channel.
- Write transactions end with the Write Response event on the B (Write Response) channel.

Transactions are only shown when the address, data, and/or response events have matching IDs. In addition, transactions are only shown in the waveform if both the start and end events occur within the captured data waveform. When multiple outstanding/overlapping transactions are displayed in the Waveform window, multiple transaction rows are used.

It is possible that the transactions on the interface can cause the outstanding transaction tracking logic within the System ILA IP to overflow as shown in the following figure.

Figure: AXI Transaction Counter Overflow Condition

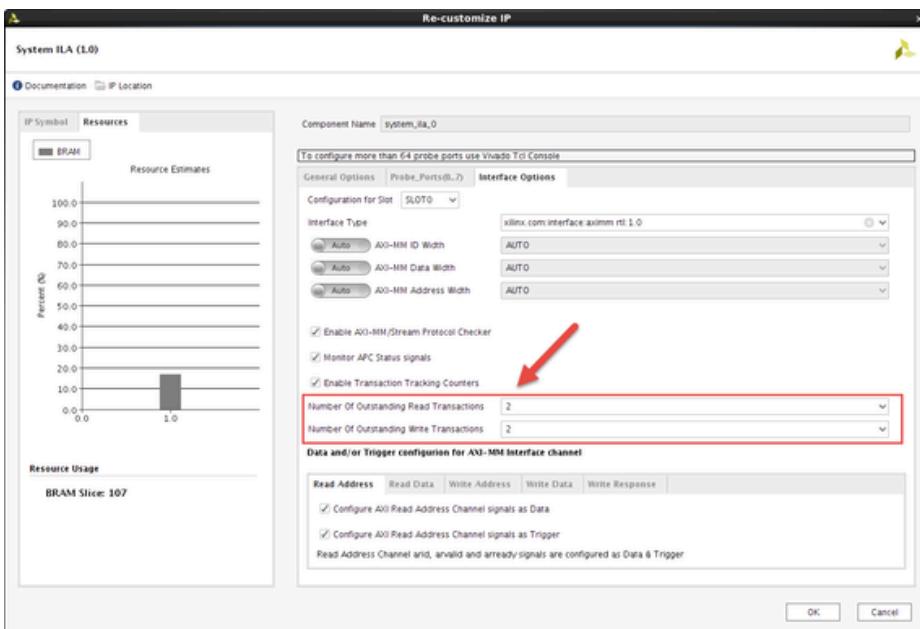


Two overflow conditions can result:

- The number of outstanding transactions for a particular ID overflow the transaction counter capacity.
- The number of IDs that have outstanding transactions overflow the number of available counters.

In either case, an overflow condition can be resolved by re-customizing the System ILA core in the IP integrator block design to increase the number of outstanding read and/or write transactions. See the following figure.

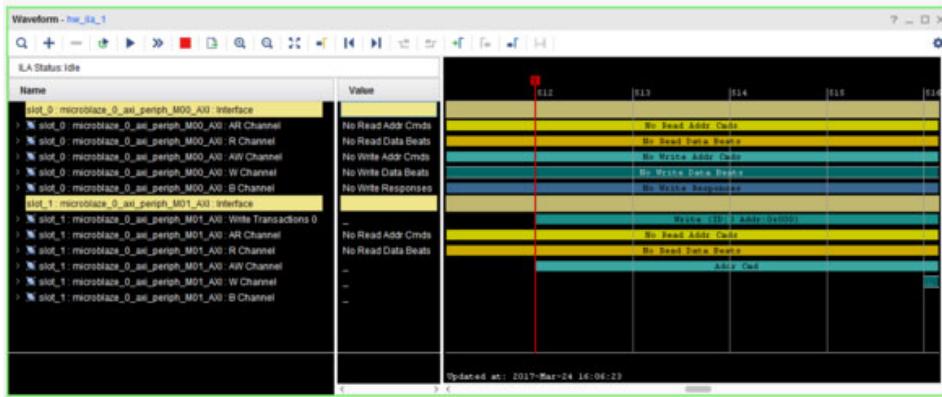
Figure: Increasing the Number of Outstanding Transactions That Can Be Tracked by System ILA



AXI Interface Events

In the Vivado Hardware Manager, if you debug a design AXI interface with a System ILA IP, the Waveform window displays the interface slots, events, and signal groups corresponding to the interfaces being probed by the System ILA. As you can see in the following figure, the Waveform window displays both of the 2 interface slots that were probed by the System ILA IP. You can see the AXI Transactions, Write Address Channel Events, the Write Data Channel Events on slot 1. You can also see the Write Data CAXI interface slots in the Waveform window.

Figure: AXI Interface Transactions and Events



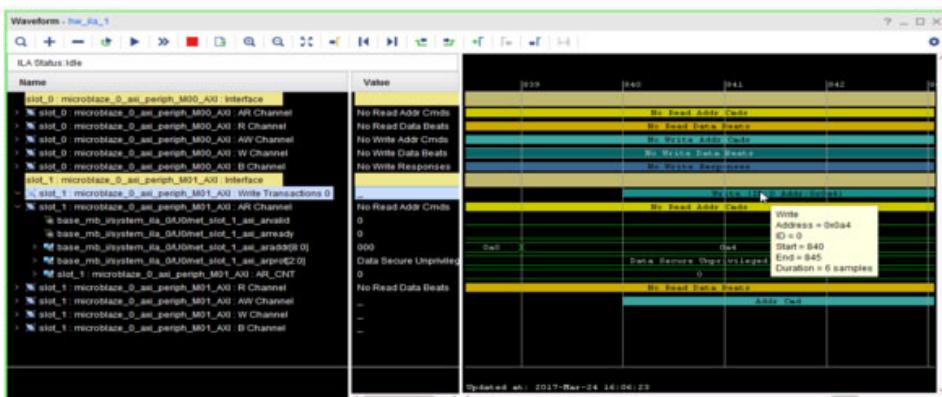
This waveform reports AXI interface related transactions and events on the Read, Write, Address, and Data channel events.

AXI Transactions

The AXI Transaction reports on the Read and Write Transactions of the AXI Read Address, AXI Read Data, Write Address, and Write Data channels.

Hovering over a specific read or write Transaction in the Waveform window brings up a window that highlights the Address, ID, Start, End, and Duration associated with the specific transaction as follows:

Figure: AXI Transactions



AXI Channel Events

The AXI Channel Events group reports on the AXI events in the AXI Read Address (AR), Read Data (R), Write Address (AW), and Write Data (W), and Write Response (B) channels.

Read Address (AR) Channel Events

Name	Description
No Read Addr Cmds	Indicates that no address command events occurred on the Read Address channel.
Addr Cmd	Indicates a valid address command phase of a read transaction. This event starts when ARVALID = '1'. This event ends when both ARVALID = '1' and ARREADY = '1' in the same clock cycle.

Read Address Channel Signal Group

This signal group is composed of all the signals that participate in a Read Address Channel Event. The signals are as follows:

- Net Name
 - ARVALID
 - ARREADY
 - ARID
 - ARADDR
 - ARBURST
 - ARLEN
 - ARSIZE
 - ARCACHE
 - ARPROT
 - ARLOCK
 - ARQOS
 - AR_CNT

Read Data Channel Events

Name	Description
No Read Data Beats	Indicates that no events occurred on the Read Data channel
Data Beat	Indicates a (non-last) data beat of a read transaction. This event occurs when RVALID = '1' and RREADY = '1' in the current clock cycle.
Last Data	Indicates the last data beat of a read transaction. This event occurs when RVALID = '1' and RREADY = '1' and RLAST = '1' in the current

Name	Description
	clock cycle.

Read Data Channel Signal Group

This signal group is composed of all the signals that participate in a Read Data Channel Event. The signals are as follows:

- Net Name
 - RVALID
 - RREADY
 - RLAST
 - RID
 - RDATA
 - RRESP
 - R_CNT

Write Address Channel Events

Name	Description
No Write Addr Cmds	Indicates that no address command events occurred on the Write Address channel.
Addr Cmd	Indicates a valid address command phase of a write transaction. This event starts when AWVALID = '1'. This event ends when both AWVALID = '1' and AWREADY = '1' in the same clock cycle.

Write Address Channel Signal Group

This signal group is composed of all the signals that participate in a Write Address Channel Event. The signals are as follows:

- Net Name
 - AWVALID
 - AWREADY
 - AWID
 - AWADDR
 - AWBURST
 - AWLEN
 - AWSIZE
 - AWCACHE
 - AWPROT
 - AWLOCK
 - AWQOS
 - AW_CNT

Write Data Channel Events

Name	Description
No Write Data Beats	Indicates that no events occurred on the Write Data channel.
Data Beat	Indicates a (non-last) data beat of a write transaction. This event occurs when WVALID = '1' and WREADY = '1' in the current clock cycle, and either of the two signals were '0' in the previous clock cycle.
Last Data	Indicates the last data beat of a write transaction. This event occurs when WVALID = '1' and WREADY = '1' and WLAST = '1' in the current clock cycle.

Write Data Channel Signal Group

This signal group is composed of all the signals that participate in a Write Data Channel Event. The signals are as follows:

- Net Name
 - WVALID
 - WREADY
 - WLAST
 - WDATA
 - WSTRB

Write Response Channel Events

Name	Description
No Write Responses	Indicates that no events occurred on the Write Response channel.

Name	Description
Write Response	Indicates response phase of a write transaction. This event occurs when BVALID = '1' and BREADY = '1' in the current clock cycle.

Write Response Channel Signal Group

This signal group is composed of all the signals that participate in a Write Response Channel Event. The signals are as follows:

- Net Name
 - BVALID
 - BREADY
 - BID
 - BRESP
 - B_CNT

Triggering on AXI Address Command and Data Beats

Debugging AXI interfaces often involves triggering three specific kinds of AXI events: End of the Address Command, End of Data Beat, and Write Response. It is often required to trigger one or more of these events on different interface channels. For instance, to implement the trigger condition of "End of Read Address Command OR End of Write Address Command", the following equation is required:

Trigger Condition = (((ARVALID == 1) && (ARREADY == 1)) || ((AWVALID == 1) && (AWREADY == 1)))

However, this requires a "Sum of Products" or "SOP"-style Boolean equation that is not possible to implement when the required AXI signals (such as ARVALID and ARREADY) reside on different probe ports. To facilitate this type of triggering, the required *VALID, *READY, and *LAST control signals are concatenated together onto a single probe port as shown in the following table.

Table: AXI Channel Control Signal Probes

Description	Probe Name	Bit 2	Bit 1	Bit 0
Read Address channel control signals	*ar_ctrl[1:0]	N/A	ARREADY	ARVALID
Read Data channel control signals	*r_ctrl[2:0]	RLAST	RREADY	RVALID
Write Address channel control signals	*aw_ctrl[1:0]	N/A	AWREADY	AWVALID
Write Data channel control signals	*w_ctrl[2:0]	WLAST	WREADY	WVALID

Description	Probe Name	Bit 2	Bit 1	Bit 0
Write Response channel control signals	*b_ctrl[1:0]	N/A	BREADY	BVALID

The following table shows how to use both the individual AXI control signal probes and the AXI channel control probes to implement useful basic trigger and capture control equations. The following figure shows how to implement the "End of Read Address Command OR End of Write Address Command" event using the basic trigger setup GUI.

Table: Trigger and/or Capture Control Event

AXI Event	Individual AXI Control Signals	Combined AXI Channel C
End of the Read Address Command	((ARVALID == 1) && (ARREADY == 1))	(*ar_ctrl == 2'b11)
End of the Last Read Data Beat	((RVALID == 1) && (RREADY == 1) && (RLAST == 1))	(*r_ctrl == 3'b111)
End of the (Non-Last) Read Data Beat	((RVALID == 1) && (RREADY == 1) && (RLAST == 0))	(*r_ctrl == 3'b011)
End of the Write Address Command	((AWVALID == 1) && (AWREADY == 1))	(*aw_ctrl == 2'b11)
End of the Write Read Data Beat	((WVALID == 1) && (WREADY == 1) && (WLAST == 1))	(*w_ctrl == 3'b111)
End of the (Non-Last) Read Data Beat	((WVALID == 1) && (WREADY == 1) && (WLAST == 0))	(*w_ctrl == 3'b011)

Figure: Basic Trigger Setup for "End of Read Address Command OR End of Write Address Command" Event



Setting Up the VIO Core to Take a Measurement

The VIO cores that you add to your design appear in the Hardware window under the target device. If you do not see the VIO cores appear, right-click the device and select Refresh Hardware. This rescans the FPGA or adaptive SoC and refreshes the Hardware window.

Note: If you still do not see the VIO core after programming and/or refreshing the FPGA or adaptive SoC, check to make sure the device was programmed with the appropriate .pdi file and check to make sure the implemented design contains an VIO core. Also, check to make sure the appropriate .ltx probes file that matches the .bit file is associated with the device.

Click the VIO core (called hw_vio_1 in the following figure) to see its properties in the VIO Core Properties window. By selecting the VIO core, you should also see the probes corresponding to the VIO core in the Debug Probes window and the corresponding VIO Dashboard in the Vivado IDE workspace (see the following figure).

Figure: VIO Core in the Hardware Window

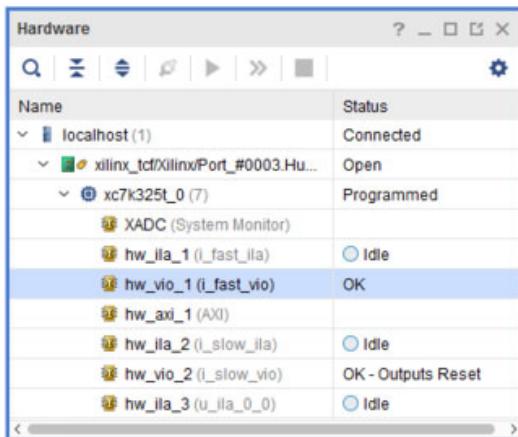
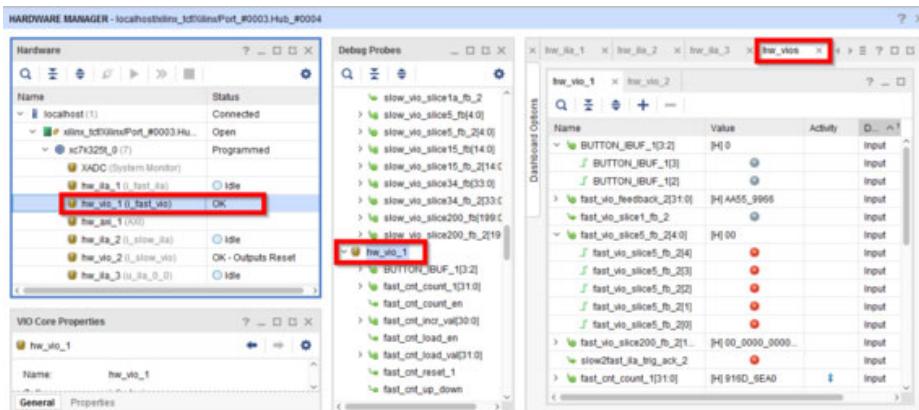


Figure: Selection of the VIO Core in Various Views



The VIO core can become out-of-sync with the Vivado IDE. Refer to Viewing the VIO Core Status for more information on how to interpret the VIO status indicators.

The VIO core operates on an object property-based set/commit and refresh/get model:

- To read VIO input probe values, first refresh the hw_vio object with the VIO core values. Observe the input probe values by getting the property values of the corresponding hw_probe object. Refer to [Interacting with VIO Core Input Probes](#) for more information.
- To write VIO output probe values, first set the desired value as a property on the hw_probe object. These property values are committed to the VIO core in hardware to write these values to the output probe ports of the core. Refer to [Interacting with VIO Core Input Probes](#) for more information.

Related Information

[Interacting with VIO Core Input Probes](#)

[Interacting with VIO Core Output Probes](#)

[Viewing the VIO Core Status](#)

Viewing the VIO Core Status

The VIO core can have zero or more input probes and zero or more output probes.

 **Note:** The VIO core must have at least one input or output probe.

The VIO core status shown in the Hardware window is used to indicate the current state of the VIO core output probes. The possible status values and any action that you need to take are described in the following table.

Table: VIO Core Status and Required User Action

VIO Status	Description	Required User Action
OK – Outputs Reset	The VIO core outputs are in sync with the Vivado IDE and the outputs are in their initial or "reset" state.	None
OK	The VIO core outputs are in sync with the Vivado IDE, however, the outputs are not in their initial or "reset" state.	None
Outputs out-of-sync	The VIO core outputs are not in sync with the Vivado IDE.	<p>You must choose one of two user actions:</p> <p>Write the values from the Vivado IDE to the VIO core by right-clicking the VIO core in the Hardware window and selecting the Commit VIO Core Outputs option.</p> <p>Update the Vivado IDE with the current values of the VIO core output probe ports by right-clicking the VIO core in the Hardware window and selecting the</p>

VIO Status	Description	Required User Action
		Refresh Input and Output Values from VIO Core option.

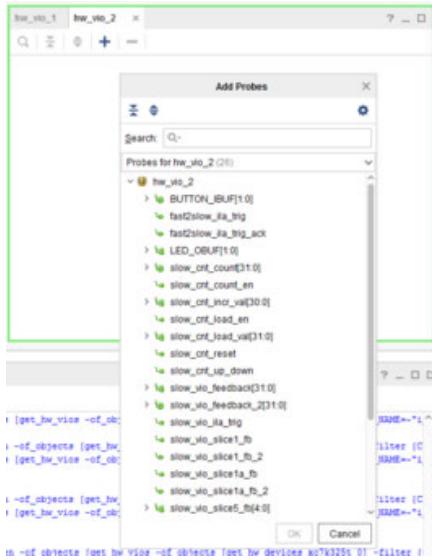
Viewing VIO Cores in the Debug Probes Window

The "+" button in the VIO Dashboard window is used to view, add, and delete the debug probes that belong to VIO core.

Using the VIO Dashboard

The VIO Default Dashboard starts out empty to which you can add VIO probes to as showing the following figure.

Figure: VIO Default Dashboard



The VIO Dashboard is a central location for all status and control information pertaining to a given VIO core. When a VIO core is first detected upon refreshing a hardware device, the VIO Dashboard for the core is automatically opened. If you need to manually open or re-open the dashboard, right-click the VIO core object in either the Hardware or Debug Probes windows and select Open Dashboard.

Interacting with VIO Core Input Probes

The VIO core input probes are used to read values from a design that is running in an FPGA or adaptive SoC in actual hardware. The VIO input probes are typically used as status indicators for a design-under-test. VIO debug probes need to be added manually to the VIO Probes window in the VIO Dashboard. Refer to the section called Viewing VIO Cores in the Debug Probes Window on how to do this. An example of what VIO input probes look like in the VIO Probes window of the VIO Dashboard is shown in the following figure.

Figure: Core Input Probes

Name	Value	Activity	D...	VIO
↳ BUTTON_IBUF_1[3:2]	[H] 0		Input	hw_vio_1
↳ BUTTON_IBUF_1[3]			Input	hw_vio_1
↳ BUTTON_IBUF_1[2]			Input	hw_vio_1
↳ fast_vio_feedback_2[31:0]	[H] AA55_9966		Input	hw_vio_1
↳ fast_vio_slice1_fb_2			Input	hw_vio_1
↳ fast_vio_slice5_fb_2[4:0]	[H] 00		Input	hw_vio_1
↳ fast_vio_slice5_fb_2[4]			Input	hw_vio_1
↳ fast_vio_slice5_fb_2[3]			Input	hw_vio_1
↳ fast_vio_slice5_fb_2[2]			Input	hw_vio_1
↳ fast_vio_slice5_fb_2[1]			Input	hw_vio_1
↳ fast_vio_slice5_fb_2[0]			Input	hw_vio_1
↳ fast_vio_slice200_fb_2[1...]	[H] 00_0000_0000		Input	hw_vio_1
↳ slow2fast_ilia_trig_ack_2			Input	hw_vio_1
↳ fast_cnt_count_1[31:0]	[H] 58C4_BC34		Input	hw_vio_1
↳ fast_vio_slice1a_fb_2	[B] 0		Input	hw_vio_1

Related Information

[Viewing VIO Cores in the Debug Probes Window](#)

Reading VIO Inputs Using the VIO Cores View

The VIO input probes can be viewed using the VIO Probes window of the VIO Dashboard window. Each input probe is viewed as a separate row in the table. The value of the VIO input probes are shown in the Value column of the table (see [Interacting with VIO Core Input Probes](#)). The VIO core input values are periodically updated based on the value of the refresh rate of the VIO core. You can set the refresh rate by changing the Refresh Rate (ms) in the VIO Properties window or by running the following Tcl command:

```
set_property CORE_REFRESH_RATE_MS 1000 [get_hw_vios hw_vio_1]
```

Note: Setting the refresh rate to 0 causes all automatic refreshes from the VIO core to stop.

Note: Very small refresh values can cause your Vivado IDE to become sluggish.

Recommended: AMD recommends a refresh rate of 500 ms or longer.

If you want to manually read a VIO input probe value, you can use Tcl commands to do so. For instance, if you wanted to refresh and get the value of the input probe called BUTTON_IBUF of the VIO core hw_vio_1, run the following Tcl commands:

```
refresh_hw_vio [get_hw_vios {hw_vio_1}]
get_property INPUT_VALUE [get_hw_probes BUTTON_IBUF]
```

Related Information

[Interacting with VIO Core Input Probes](#)

Setting the VIO Input Display Type and Radix

The display type of VIO input probes can be set by right-clicking a VIO input probe in the VIO Probes window of the VIO Dashboard window and selecting:

- Text to display the input as a text field. This is the only display type for VIO input probe vectors (more than one bit wide).
- LED to display the input as a graphical representation of a light-emitting diode (LED). This display type is only applicable to VIO input probe scalars and individual elements of VIO input probe vectors. You can set the high and low values to one of four colors:
 - Gray (off)
 - Red
 - Green
 - Blue

When the display type of the VIO input probe is set to Text, you can change the radix by right-clicking a VIO input probe in the VIO Probes window of the VIO Dashboard window and selecting:

- Radix > Binary to change the radix to binary.
- Radix > Octal to change the radix to octal.
- Radix > Hex to change the radix to hexadecimal.
- Radix > Unsigned to change the radix to unsigned decimal.
- Radix > Signed to change the radix to signed decimal.

You can also set the radix of the VIO input probe using a Tcl command. For instance, to change the radix of a VIO input probe called "BUTTON_IBUF", run the following Tcl command:

```
set_property INPUT_VALUE_RADIX HEX [get_hw_probes BUTTON_IBUF]
```

Observing and Controlling VIO Input Activity

In addition to reading values from the VIO input probes, you can also monitor the activity of the VIO input probes. The activity detectors are used to indicate when the values on the VIO inputs have changed in between periodic updates to the Vivado IDE.

The VIO input probe activity values are shown as arrows in the activity column of the VIO Probes window of the VIO Dashboard window:

- An up arrow indicates that the input probe value has transitioned from a 0 to a 1 during the activity persistence interval.
- A down arrow indicates that the input probe value has transitioned from a 1 to a 0 during the activity persistence interval.
- A double-sided arrow indicates that the input probe value has transitioned from a 1 to a 0 and from a 0 to a 1 at least once during the activity persistence interval.

The persistence of how long the input activity status is displayed can be controlled by right-clicking a VIO input probe in the VIO Probes window of the VIO Dashboard window and selecting:

- Activity Persistence > Infinite to accumulate and retain the activity value until you reset it.
- Activity Persistence > Long (80 samples) to accumulate and retain the activity for a longer period of time.
- Activity Persistence > Short (8 samples) to accumulate and retain the activity for a shorter period of time.

You can also set the activity persistence using a Tcl command. For instance, to change the activity persistence on the VIO input probe called BUTTON_IBUF to a long interval, run the following Tcl command:

```
set_property ACTIVITY_PERSISTENCE LONG [get_hw_probes BUTTON_IBUF]
```

The activity for all input probes for a given core can be reset by right-clicking the VIO core in the Hardware window and selecting Reset All Input Activity. You can also do this by running the following Tcl command:

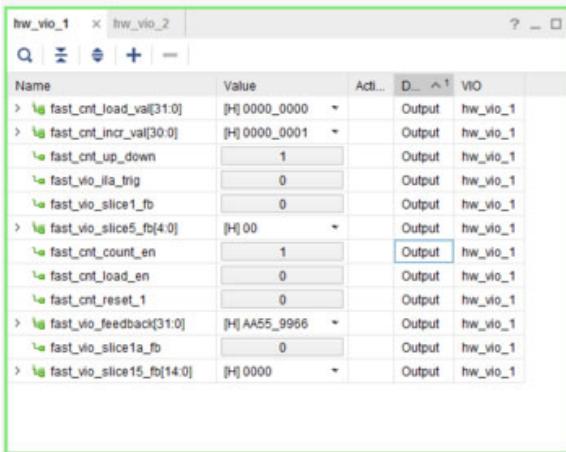
```
reset_hw_vio_activity [get_hw_vios {hw_vio_1}]
```

★ Tip: You can change the type, radix, and/or activity persistence of multiple scalar members of a VIO input probe vector by right-clicking the whole probe or multiple members of the probe, making a menu choice. The menu choice applies to all selected probe scalars.

Interacting with VIO Core Output Probes

The VIO core output probes are used to write values to a design that is running in an FPGA or adaptive SoC in actual hardware. The VIO output probes are typically used as low-bandwidth control signals for a design-under-test. VIO debug probes need to be added manually to the VIO Probes window in the VIO Dashboard. Refer to the section called Viewing VIO Cores in the Debug Probes Window on how to do this. An example of what VIO output probes look like in the VIO Probes window of the VIO Dashboard is shown in the following figure.

Figure: VIO Outputs in the VIO Probes Window of the VIO Dashboard



The screenshot shows a software interface titled "hw_vio_1" with a list of output probes. The columns are labeled "Name", "Value", "Acti...", "D...", "VIO", and "hw_vio_1". The "hw_vio_1" column is highlighted with a blue border. The table contains 15 rows, each representing a different probe name and its corresponding value and activity settings.

Name	Value	Acti...	D...	VIO	hw_vio_1
> fast_cnt_load_val[31:0]	[H] 0000_0000			Output	hw_vio_1
> fast_cnt_incr_val[30:0]	[H] 0000_0001			Output	hw_vio_1
> fast_cnt_up_down	1			Output	hw_vio_1
> fast_vio_ilia_trig	0			Output	hw_vio_1
> fast_vio_slice1_fb	0			Output	hw_vio_1
> fast_vio_slice5_fb[4:0]	[H] 00			Output	hw_vio_1
> fast_vio_slice5_fb[4:0]	[H] 00			Output	hw_vio_1
> fast_vio_slice1_fb	0			Output	hw_vio_1
> fast_vio_slice15_fb[14:0]	[H] 0000			Output	hw_vio_1
> fast_vio_feedback[31:0]	[H] AA55_9966			Output	hw_vio_1
> fast_vio_ilia_en	0			Output	hw_vio_1
> fast_vio_ilia_trig	0			Output	hw_vio_1
> fast_vio_ilia_trig	0			Output	hw_vio_1
> fast_vio_ilia_trig	0			Output	hw_vio_1

Related Information

[Viewing VIO Cores in the Debug Probes Window](#)

Writing VIO Outputs Using the VIO Cores View

The VIO output probes can be set using the VIO Probes window of the VIO Dashboard window. Each output probe is viewed as a separate row in the table. The value of the VIO output probes are shown in the Value column of the table (see [Interacting with VIO Core output Probes](#)). The VIO core output values are updated whenever a new value is entered into the Value column. Clicking on the Value column causes a pull-down dialog to appear. Type the desired value into the Value text field and click OK.

You can also write out a new value to the VIO core using Tcl commands. For instance, if you wanted to write a binary value of "11111" to the VIO output probe called vio_slice5_fb_2 whose radix is already set to BINARY, run the following Tcl commands:

```
set_property OUTPUT_VALUE 11111 [get_hw_probes vio_slice5_fb_2]
commit_hw_vio [get_hw_probes {vio_slice5_fb_2}]
```

Related Information

[Interacting with VIO Core Output Probes](#)

Setting the VIO Output Display Type and Radix

The display type of VIO output probes can be set by right-clicking a VIO output probe in the VIO Probes window of the VIO Dashboard window and selecting one of the following.

Text

Displays the output as a text field. This is the only display type for VIO input probe vectors (more than one bit wide).

Toggle Button

Displays the output as a graphical representation of a toggle button. This display type is only applicable to VIO output probe scalars and individual elements of VIO input probe vectors.

When the display type of the VIO output probe is set to Text, you can change the radix by right-clicking a VIO output probe in the VIO Cores tabbed view of the Debug Probes window and selecting:

- Radix > Binary to change the radix to binary.
- Radix > Octal to change the radix to octal.
- Radix > Hex to change the radix to hexadecimal.
- Radix > Unsigned to change the radix to unsigned decimal.
- Radix > Signed to change the radix to signed decimal.

You can also set the radix of the VIO output probe using a Tcl command. For instance, to change the radix of a VIO output probe called "vio_slice5_fb_2" to hexadecimal, run the following Tcl command:

```
set_property OUTPUT_VALUE_RADIX HEX [get_hw_probes vio_slice5_fb_2]
```

Resetting the VIO Core Output Values

The VIO v2.0 core has a feature that allows you to specify an initial value for each output probe port. You can reset the VIO core output probe ports to these initial values by right-clicking the VIO core in the Hardware window and selecting the Reset VIO Core Outputs option. You can also reset the VIO core outputs using a Tcl command:

```
reset_hw_vio_outputs [get_hw_vios {hw_vio_1}]
```

 **Note:** Resetting the VIO output probes to their initial values cause the output probe values to become out-of-sync with the Vivado IDE. Refer to the section called Synchronizing the VIO Core Output Values to the Vivado IDE on how to handle this situation.

Related Information

[Synchronizing the VIO Core Output Values to the Vivado IDE](#)

Synchronizing the VIO Core Output Values to the Vivado IDE

The output probes of a VIO core can become out-of-sync with the Vivado IDE after resetting the VIO outputs, re-programming the FPGA or adaptive SoC, or by another Vivado tool instance setting output values before the current instance has started. In any case, if the VIO status indicates "Outputs out-of-sync", you need to take one of two actions:

- Write the values from the Vivado IDE to the VIO core by right-clicking the VIO core in the Hardware window and selecting the Commit VIO Core Outputs option. You can also do this by running a Tcl command:

```
commit_hw_vio [get_hw_vios {hw_vio_1}]
```

- Update the Vivado IDE with the current values of the VIO core output probe ports by right-clicking the VIO core in the Hardware window and selecting the Refresh Input and Output Values from VIO Core option. You can also do this by running a Tcl command:

```
refresh_hw_vio -update_output_values 1 [get_hw_vios {hw_vio_1}]
```

Hardware System Communication Using the JTAG-to-AXI Master Debug Core

The JTAG-to-AXI Master debug core is a customizable core that can generate the AXI transactions and drive the AXI signals internal to an FPGA at runtime. The core supports all memory-mapped AXI and AXI-Lite interfaces and can support 32 or 64-bit wide data interfaces.

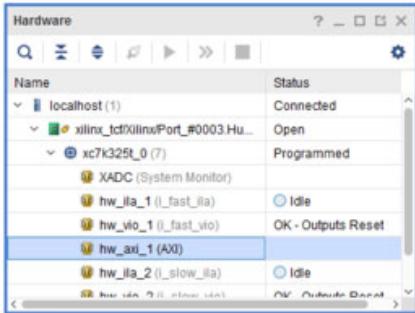
The JTAG-to-AXI Master (JTAG-AXI) cores you add to your design appear in the Hardware window under the target device. If you do not see the JTAG-AXI cores appear, right-click the device and select Refresh Hardware. This re-scans the FPGA and refreshes the Hardware window.

 **Note:** If you still do not see the ILA core after programming and/or refreshing the FPGA device, check to make sure the device was programmed with the appropriate .bit file and check to make

sure the implemented design contains an ILA core.

Click to select the JTAG-AXI core (called hw_axi_1 in the following figure) to see its properties in the AXI Core Properties window.

Figure: JTAG-to-AXI Master Core in the Hardware Window



Interacting with the JTAG-to-AXI Master Debug Core in Hardware

The JTAG-to-AXI Master debug core can only be communicated with using Tcl commands. You can create and run AXI read and write transactions using the `create_hw_axi_txn` and `run_hw_axi` commands, respectively.

Resetting the JTAG-to-AXI Master Debug Core

Before creating and issuing transactions, it is important to reset the JTAG-to-AXI Master core using the following Tcl command:

```
reset_hw_axi [get_hw_axis hw_axi_1]
```

Creating and Running a Read Transaction

The Tcl command used to create an AXI transaction is called `create_hw_axi_txn`. For more information on how to use this command, type "help `create_hw_axi_txn`" at the Tcl Console in the Vivado IDE. Here is an example on how to create a 4-word AXI read burst transaction from address 0:

```
create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -type READ -address 00000000 -  
len 4
```

where:

- `read_txn` is the user-defined name of the transaction
- `[get_hw_axis hw_axi_1]` returns the `hw_axi_1` object
- `-address 00000000` is the start address
- `-len 4` sets the AXI burst length to 4 words

The next step is to run the transaction that was created using the `run_hw_axi` command. Here is an example on how to do this:

```
run_hw_axi [get_hw_axi_txns read_txn]
```

The last step is to get the data that was read due to running the transaction. You can use either the `report_hw_axi_txn` or `report_property` commands to print the data to the screen or you can use the `get_property` command to return the value for use elsewhere.

```
report_hw_axi_txn [get_hw_axi_txns read_txn]
0 00000000 00000000
8 00000000 00000000
report_property [get_hw_axi_txns read_txn]
Property Type Read-only Visible Value
CLASS string true true hw_axi_txn
CMD.ADDR string false true 00000000
CMD.BURST enum false true INCR
CMD.CACHE int false true 3
CMD.ID int false true 0
CMD.LEN int false true 4
CMD.SIZE enum false true 32
DATA string false true 00000000000000000000000000000000
HW_AXI string true true hw_axi_1
NAME string true true read_txn
TYPE enum false true READ
```

Creating and Running a Write Transaction

Here is an example on how to create a 4-word AXI write burst transaction from address 0:

```
create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -type WRITE -address 00000000
\
    -len 4 -data {11111111_22222222_33333333_44444444}
```

where:

- `write_txn` is the user-defined name of the transaction
- `[get_hw_axis hw_axi_1]` returns the `hw_axi_1` object
- `-address 00000000` is the start address
- `-len 4` sets the AXI burst length to 4 words
- `-data {11111111_22222222_33333333_44444444}` - The `-data` direction is LSB to the left (that is, address 0) and MSB to the right (that is, address 3).

The next step is to run the transaction that was created using the `run_hw_axi` command. Here is an example on how to do this:

```
run_hw_axi [get_hw_axi_txns write_txn]
```

!! Important: If you reprogram the device, all the existing `jtag_axi` transactions are deleted. You can recreate these transactions again.

★ Tip: The -queue optional argument to the run_hw_axi Tcl command allows you to specify hw_axi transactions in queue mode. Queued operation allows up to 16 read and 16 write transactions to be queued in the JTAG to AXI Master FIFO and issued back-to-back for low latency and higher performance between the transactions. Non-queued transactions are simply run as submitted.

Using Vivado Logic Analyzer in a Lab Environment

The Vivado logic analyzer feature is integrated into the Vivado IDE and Vivado Lab Edition. To use Vivado logic analyzer feature to debug a design that is running on a target board that is in a lab environment, you need to do one of three things:

- Install and run the Vivado Lab Edition on your lab machine. For more details refer to Vivado Lab Edition of this user guide.
- Install and run the full Vivado IDE on your lab machine.
- Install the latest version of the Vivado Design Suite or Vivado Hardware Server (Standalone) on your remote lab machine and use the Vivado logic analyzer feature on your local machine to connect to a remote instance of the Vivado Hardware Server (hw_server).

Related Information

[Vivado Lab Edition](#)

Connecting to a Remote hw_server Running on a Lab Machine

If you have a network connection to your lab machine, you can also connect to the target board by connecting to a hardware server that is running on that remote lab machine. For security reasons, the hw_server launched within the Vivado IDE only listens on the loop-back adapter (for example, localhost, or 127.0.0.1). For remote access, launch the hw_server manually.

The following steps explain how to use the Vivado logic analyzer feature to connect to a Vivado hardware server (hw_server.bat on Windows platforms or hw_server on Linux platforms) that is running on the lab machine:

1. Install the latest version of the Vivado Design Suite or Vivado hardware server (standalone) on the lab machine.

!! Important: You do *not* need to install the full Vivado Design Suite or Vivado Lab Edition on the lab machine to only use the remote hardware server feature. However, if you want to use the Vivado Hardware Manager features (such as the Vivado logic analyzer or Vivado serial I/O analyzer) on the lab machine, you need to install the Vivado Lab Edition on the lab machine.

You do *not* need any software licenses to run the hardware server, any of the Hardware Manager features, or the Vivado Lab Edition.

2. Start up the hw_server application on the remote lab machine. Assuming you installed the Vivado hardware server (standalone) to the default location and your lab machine is a 64-bit Windows machine, the file path is as follows:

C:\Xilinx\VivadoHWSRV\vivado_release.version\bin\hw_server.bat

3. Start the Vivado IDE in GUI mode on a different machine than your lab machine.
4. Follow the steps in Connecting to the Hardware Target and Programming the Device section to open a connection to the target board that is connected to your lab machine. However, instead of connecting to a Vivado CSE server running on localhost, use the hostname of your lab machine.
5. Follow the steps in Setting up the ILA Core to Take a Measurement section and beyond to debug your design in hardware.

Related Information

[Connecting to the Hardware Target and Programming the Device](#)

[Setting Up the ILA Core to Take a Measurement](#)

Description of Hardware Manager Tcl Objects and Commands

You can use Tcl commands to interact with your hardware under test. The hardware is organized in a set of hierarchical first-class Tcl objects (see the following table).

Table: Hardware Manager Tcl Objects

Tcl Object	Description
hw_server	Object referring to hardware server. Each hw_server can have one or more hw_target objects associated with it.
hw_target	Object referring to JTAG cable or board. Each hw_target can have one or more hw_device objects associated with it.
hw_device	Object referring to a device in the JTAG chain, including AMD FPGAs or adaptive SoCs. Each hw_device can have one or more hw_ila objects associated with it.
hw_ila	Object referring to an ILA core in the AMD FPGA or adaptive SoC. Each hw_ila object can have only one hw_ila_data object associated with it. Each hw_ila object can have one or more hw_probe objects associated with it.
hw_ila_data	Object referring to data uploaded from an ILA debug core.
hw_probe	Object referring to the probe input of an ILA debug core.
hw_vio	Object referring to a VIO core in the AMD FPGA or adaptive SoC.

For more information about the Hardware Manager commands, run the help -category hardware Tcl command in the Tcl Console.

Description of hw_server Tcl Commands

The following table contains descriptions of all Tcl commands used to interact with hardware servers.

Table: Descriptions of hw_server Tcl Commands

Tcl Command	Description
connect_hw_server	Open a connection to a hardware server.
current_hw_server	Get or set the current hardware server.
disconnect_hw_server	Close a connection to a hardware server.
get_hw_servers	Get list of hardware server names for the hardware servers.
refresh_hw_server	Refresh a connection to a hardware server.

Description of hw_target Tcl Commands

The following table contains descriptions of all Tcl commands used to interact with hardware targets.

Table: Descriptions of hw_target Tcl Commands

Tcl Command	Description
close_hw_target	Close a hardware target.
current_hw_target	Get or set the current hardware target.
get_hw_targets	Get list of hardware targets for the hardware servers.
open_hw_target	Open a connection to a hardware target on the hardware server.
refresh_hw_target	Refresh a connection to a hardware target.

Description of hw_device Tcl Commands

The following table contains descriptions of hw_device Tcl Commands used to interact with hardware devices.

Table: Descriptions of hw_device Tcl Commands

Tcl Command	Description
current_hw_device	Get or set the current hardware device.
get_hw_devices	Get list of hardware devices for the target.
program_hw_device	Program AMD FPGA devices.
refresh_hw_device	Refresh a hardware device.

Description of hw_ilab Tcl Commands

The following table contains descriptions of hw_ilab Tcl Commands used to interact with ILA debug cores.

Table: Descriptions of hw_ilab Tcl Commands

Tcl Command	Description
current_hw_ilab	Get or set the current hardware ILA.
get_hw_ilabs	Get list of hardware ILAs for the target.
reset_hw_ilab	Reset hw_ilab control properties to default values.
run_hw_ilab	Arm® hw_ilab triggers.
wait_on_hw_ilab	Wait until all data has been captured.

Description of hw_iladata Tcl Commands

The following table contains descriptions of hw_iladata Tcl Commands used to interact with captured ILA data.

Table: Descriptions of hw_iladata Tcl Commands

Tcl Command	Description
current_hw_iladata	Get or set the current hardware ILA data.
display_hw_iladata	Display hw_iladata in waveform viewer.
get_hw_iladata	Get list of hw_iladata objects.
list_hw_samples	Lists data samples associated with an individual hardware probe.
read_hw_iladata	Read hw_iladata from a file.
upload_hw_iladata	Stop the ILA core from capturing data and upload any captured data.
write_hw_iladata	Write hw_iladata to a file.

Description of hw_probe Tcl Commands

The following table contains descriptions of all Tcl commands used to interact with captured ILA data.

Table: Descriptions of hw_probe Tcl Commands

Tcl Command	Description

Tcl Command	Description
create_hw_probe	Creates a new hardware probe from physical ILA probe ports and/or constant values.
delete_hw_probe	Deletes a user-defined hardware probe creating using the create_hw_probe command
get_hw_probes	Get list of hardware probes.

Description of hw_vio Tcl Commands

The following table contains descriptions of all Tcl commands used to interact with VIO cores.

Table: Descriptions of hw_vio Tcl Commands

Tcl Command	Description
commit_hw_vio	Write hw_probe OUTPUT_VALUE properties values to VIO cores.
get_hw_vios	Get a list of hw_vios
refresh_hw_vio	Update hw_probe INPUT_VALUE and ACTIVITY_VALUE properties with values read from VIO cores.
reset_hw_vio_activity	Reset VIO ACTIVITY_VALUE properties, for hw_probes associated with specified hw_vio objects.
reset_hw_vio_outputs	Reset VIO core outputs to initial values.

Description of hw_axi and hw_axi_txn Tcl Commands

The following table contains descriptions of all Tcl commands used to interact with JTAG-to-AXI Master cores.

Table: Description of hw_axi and hw_axi_txn Tcl Commands

Tcl Command	Description
create_hw_axi_txn	Creates hardware AXI transaction object.
delete_hw_axi_txn	Deletes hardware AXI transaction objects.
get_hw_axi_txns	Gets a list of hardware AXI transaction objects.
get_hw_axis	Gets a list of hardware AXI objects.
refresh_hw_axi	Refreshes hardware AXI object status.
report_hw_axi_txn	Reports formatted hardware AXI transaction data.
reset_hw_axi	Resets hardware AXI core state.

Tcl Command	Description
run_hw_axi	Runs hardware AXI read/write transactions and update transaction status in the corresponding hw_axi object.

Description of hw_sysmon Tcl Commands

The following table contains descriptions of all Tcl commands used to interact with System Monitor core.

Table: Descriptions of hw_sysmon Tcl commands

Tcl Command	Description
commit_hw_sysmon	Commits the current property values defined on a hw_sysmon object to the System Monitor registers on the hardware device.
get_hw_sysmon_reg	Returns the hex value of the System Monitor register defined at the specified address of the specified hw_sysmon object.
get_hw_sysmons	Returns the list of Sysmon debug core objects defined on the current hardware device.
refresh_hw_sysmon	Refreshes the properties of the hw_sysmon object with the values on the System Monitor from the current hw_device.
set_hw_sysmon_reg	Sets the System Monitor register at the specified address to the hex value specified.

 **Note:** Detailed help for each of these commands can be obtained by typing < command name > -help on the Vivado Tcl Console.

Using Tcl Commands to Interact with a JTAG-to-AXI Master Core

Here is an example Tcl command script that interacts with the following example system:

- One KC705 board's Digilent JTAG-SMT1 cable (serial number 12345) accessible via a Vivado hw_server running on localhost:3121.
- Single JTAG-to-AXI Master core in a design running in the XC7K325T device on the KC705 board.
- JTAG-to-AXI Master core is in an AXI-based system that has an AXI BRAM Controller Slave core in it.

Example Tcl Command Script

```
# Connect to the Digilent Cable on localhost:3121
connect_hw_server -url localhost:3121
current_hw_target [get_hw_targets */xilinx_tcf/Digilent/12345]
open_hw_target

# Program and Refresh the XC7K325T Device
current_hw_device [lindex [get_hw_devices] 0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices] 0]
set_property PROGRAM.FILE {C:/design.bit} [lindex [get_hw_devices] 0]
set_property PROBES.FILE {C:/design.ltx} [lindex [get_hw_devices] 0]
program_hw_devices [lindex [get_hw_devices] 0]
refresh_hw_device [lindex [get_hw_devices] 0]

# Reset the JTAG-to-AXI Master core
reset_hw_axi [get_hw_axis hw_axi_1]
# Create a read transaction bursts 128 words starting from address 0
create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -type read \
-address 00000000 -len 128
# Create a write transaction bursts 128 words starting at address 0
# using a repeating fill value of 11111111_22222222_33333333_44444444
# (where LSB is to the left)
create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -type write \
-address 00000000 -len 128 -data {11111111_22222222_33333333_44444444}
# Run the write transaction
run_hw_axi [get_hw_axi_txns wrte_txn]
# Run the read transaction
run_hw_axi [get_hw_axi_txns read_txn]
```

Using Tcl Commands to Take an ILA Measurement

Here is an example Tcl command script that interacts with the following example system:

- One KC705 board's Digilent JTAG-SMT1 cable (serial number 12345) accessible via a Vivado CSE server running on localhost:3121.
- Single ILA core in a design running in the XC7K325T device on the KC705 board.
- ILA core has a probe called counter[3:0].

Example Tcl Command Script

```
# Connect to the Digilent Cable on localhost:3121
connect_hw_server -url localhost:3121
current_hw_target [get_hw_targets */xilinx_tcf/Digilent/12345]
open_hw_target

# Program and Refresh the XC7K325T Device
current_hw_device [lindex [get_hw_devices] 0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices] 0]
set_property PROGRAM.FILE {C:/design.bit} [lindex [get_hw_devices] 0]
```

```

set_property PROBES.FILE {C:/design.ltx} [lindex [get_hw_devices] 0]
program_hw_devices [lindex [get_hw_devices] 0]
refresh_hw_device [lindex [get_hw_devices] 0]
# Set Up ILA Core Trigger Position and Probe Compare Values
set_property CONTROL.TRIGGER_POSITION 512 [get_hw_ilas hw_ila_1]
set_property COMPARE_VALUE.0 eq4'b0000 [get_hw_probes counter]
# Arm the ILA trigger and wait for it to finish capturing data
run_hw_ila hw_ila_1
wait_on_hw_ila hw_ila_1
# Upload the captured ILA data, display it, and write it to a file
current_hw_ila_data [upload_hw_ila_data hw_ila_1]
display_hw_ila_data [current_hw_ila_data]
write_hw_ila_data my_hw_ila_data [current_hw_ila_data]

```

Trigger At Startup

The Trigger at Start up feature is used to configure the trigger settings of an ILA core in a design .bit file so that it is pre-armed to trigger immediately after device start up. You do this by taking the various trigger settings that ordinarily get applied to an ILA core running in a design in hardware, and applying them to the ILA core in the implemented design.

!! Important: The following process for using Trigger at Start up assumes that you have a valid ILA design working in hardware, and that the ILA core has NOT been flattened during the synthesis flow.

To use the Trigger at Start up feature perform the following steps:

1. Run through the first pass of the ILA flow as usual to set up the trigger condition.
 - a. Open the target, configure the device, and bring up the ILA Dashboard.
 - b. Enter the trigger equations for the ILA core in the ILA Dashboard.
2. From the Vivado Tcl command line, export the trigger register map file for the ILA core. This file contains all of the register settings to "stamp" back on to the implemented netlist. The output from this is a single file.

```
% run_hw_ila -file ila_trig.tas [get_hw_ilas hw_ila_1]
```

3. Go back and open the previously implemented routed design in Vivado IDE. There are two ways to do this depending on your project flow.
 - a. Project Mode: Use the Flow Navigator to open the implemented design.
 - b. Non-Project Mode: Open your routed checkpoint: %open_checkpoint <file>.dcp
4. At the Implemented Design Tcl Console, apply the trigger settings to the current design in memory, which is your routed netlist.

```
%apply_hw_ila_trigger ila_trig.tas
```

 **Note:** If you see an ERROR indicating that the ILA core has been flattened during synthesis, you need to regenerate your design and force synthesis to preserve hierarchy for the ILA core. Ensure that you have a valid ILA design working in hardware, and that the ILA core has NOT been flattened during the synthesis flow.

5. At the Implemented Design Tcl Console, write the bitstream with Trigger at Start up settings.

!! Important: To pick up the routed design changes do this at the Tcl command console only:
write_bitstream trig_at_startup.bit

6. Go back to the Hardware Manager and reconfigure with the new .bit file that you generated in the previous step. You have to set the property for the updated .bit file location either through the GUI or through a Tcl command. Make sure you set the new .bit file as the one to use for configuration in the hardware tool.

- Select the device in the hardware tree.
- Assign the .bit file generated in step 5.

7. Program the device using the new .bit file.

Once programmed, the new ILA core should immediately arm at start up. You should see an indication in the Trigger Capture Status for the ILA core. If trigger or capture events have occurred, the ILA core is now populated with captured data samples.

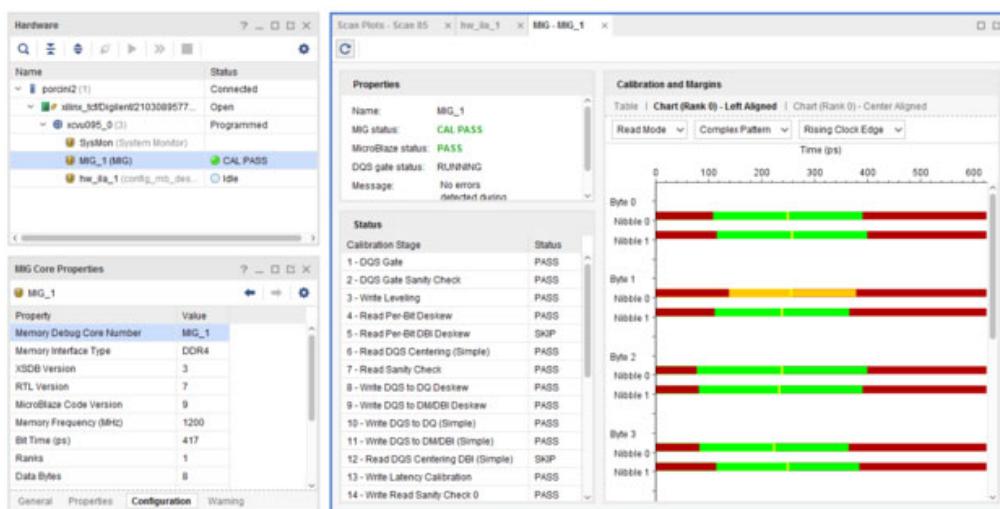
Memory Calibration Debug

Memory Interface IPs in Vivado support calibration debug. They store useful core configuration, calibration, and data window information that is accessible in the Vivado Hardware Manager. The Memory Calibration Debug can be used at any point to read out this information and get valuable statistics and feedback from the memory interface IPs. The information can be viewed through a Memory Calibration Debug GUI in the Vivado Hardware Manager or through available Memory Calibration Debug Tcl commands.

Memory Calibration Debug GUI Usage

Upon configuring the device, the memory interfaces are visible in the Vivado Hardware Manager. Memory calibration content are shown in a debug interface that can be used to very quickly identify calibration status, and read and write window margin. This debug interface is always included in the generated Memory Interface (AMD UltraScale™ and AMD UltraScale+™) designs.

Figure: Memory Calibration Debug Interface



Memory Calibration Debug Tcl Usage

Use the following Tcl commands in the Vivado Tcl Console when connected to the hardware in Vivado Hardware Manager to output all memory calibration debug content that is displayed in the Vivado IDE.

- `get_hw_migs`
 - Displays what memory interfaces exist in the design.
- `refresh_hw_mig [lindex [get_hw_migs] 0]`
 - Refreshes only the memory interfaces denoted by index (index begins with 0).
- `report_property[lindex [get_hw_migs] 0]`
 - Reports all of the parameters available for the memory interface.
 - Where 0 is the index of the memory interface to be reported (index begins with 0).

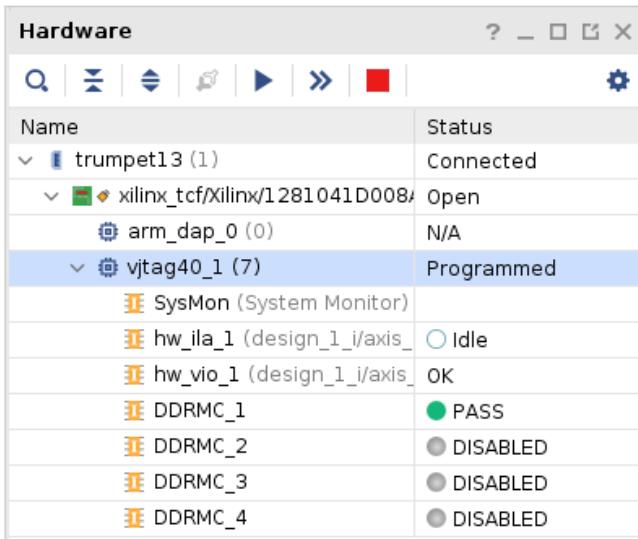
For more specific details see the UltraScale or 7 series Memory Calibration debug commands in the following documents:

- [Answer 43879, MIG 7 Series DDR3/DDR2 - Hardware Debug Guide.](#)
- [UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide \(PG150\)](#).

DDRMC Calibration Debug GUI Usage

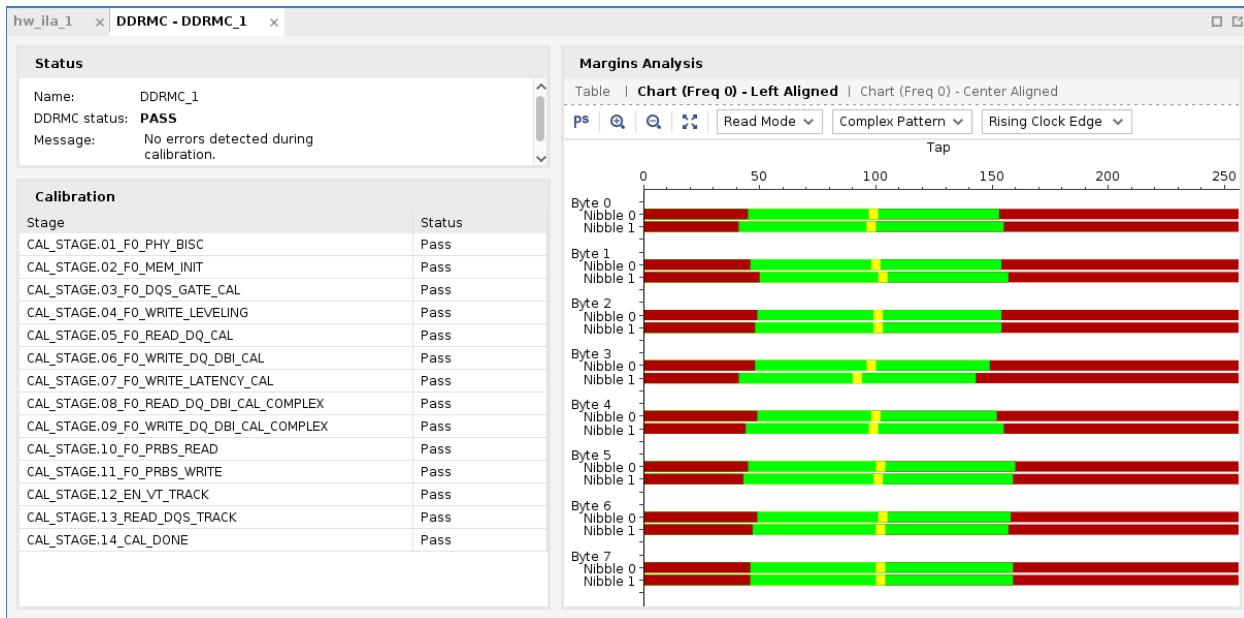
The DDRMC is one of the many integrated blocks included in the Versal architecture. Upon configuring the Versal device, both the enabled and disabled memory interfaces are visible in the Vivado Hardware Manager.

Figure: Hardware Window with DDRMC Calibration Status



Memory calibration content is shown in a debug interface that can be used to very quickly identify calibration status and read/write window margin. This debug interface is always enabled as part of the DDRMC integrated block.

Figure: DDRMC Calibration Debug Interface



DDRMC Calibration Debug Tcl Usage

Use the following Tcl commands in the Vivado Tcl Console when connected to the hardware in Vivado Hardware Manager to output DDRMC calibration debug content that is displayed in the Vivado IDE.

- `get_hw_ddrmcs`
 - Get a list of Versal adaptive SoC integrated and soft DDRMC cores.
- `refresh_hw_ddrmc [lindex [get_hw_ddrmcs] 0]`
 - Refreshes only the DDRMC denoted by index (index begins with 0).
- `report_property [lindex [get_hw_ddrmcs] 0]`
 - Reports all of the parameters available for the DDRMC, where 0 is the index of the DDRMC to be reported (index begins with 0).

For more information on the DDRMC, see the *Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide (PG313)*.

Debugging Dynamic Function eXchange (DFX) Designs in Vivado Hardware Manager

Vivado Hardware Manager supports debugging on DFX designs. To debug such a design successfully, it is necessary to program the full design bitstream before programming the partial bitstream to replace specific reconfigurable modules.

For an example of instantiating debug cores in a DFX design, and functionality within the Vivado Hardware Manager, see this [link](#) in *Vivado Design Suite Tutorial: Dynamic Function eXchange (UG947)*.

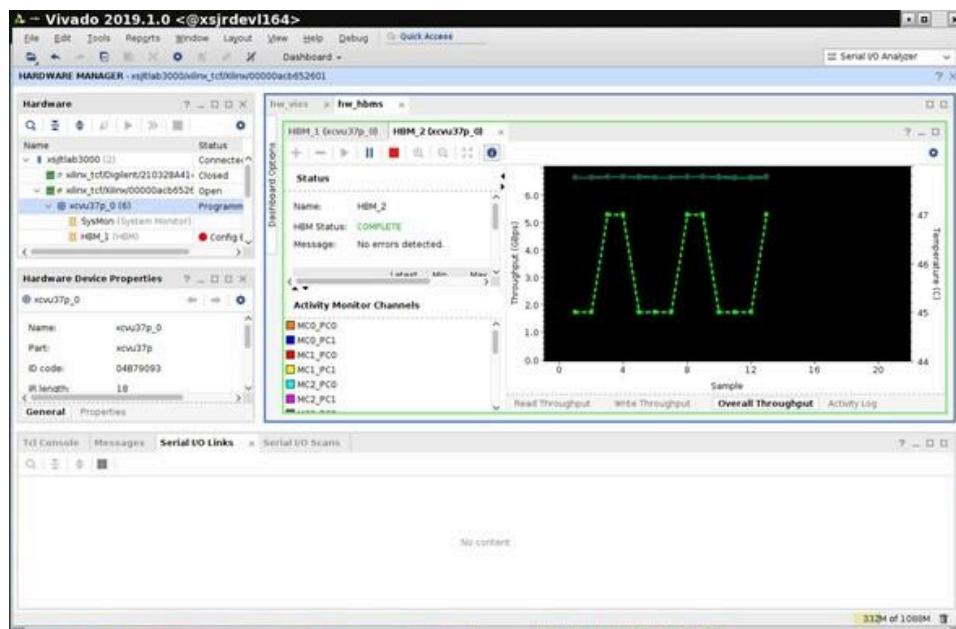
High Bandwidth Memory (HBM) Monitor

Certain AMD Virtex™ UltraScale+™ FPGAs include an integrated High Bandwidth Memory (HBM) controller and memory stacks. The integrated HBM controller and memory stacks contain both performance counters and temperature sensors. The HBM monitor can be used at any time to gain real-time access, capture, and export of performance monitoring and temperature sensors on the HBM die.

HBM Monitor GUI Usage

After configuring an HBM enabled device with a design that contains an instance of the AXI High Bandwidth Memory Controller, you can see the HBM interfaces in the Vivado Hardware Manager. Support for the HBM monitor is always included in the generated High Bandwidth Memory Controller. The HBM monitor displays the stack temperature, read, write, and overall throughput. You can export the captured data to a comma separated value (CSV) formatted text file for further post-processing or analysis.

Figure: Viewing Live Performance



HBM Monitor Tcl Usage

Use the following Tcl commands in the Vivado Tcl Console when connected to the hardware in the Vivado Hardware Manager to interact with the HBM Monitor.

- `get_hw_hbms` - Displays a list of the HBM interfaces that exist in the design.
- `refresh_hw_hbm [lindex [get_hw_hbms] 0]` - Refreshes the status of the specified hardware HBM(s), in this case the HBM denoted by index 0.
- `report_property [lindex [get_hw_hbms] 0]` - Reports all the parameters available for the HBM interface specified, in this case for the HBM interface denoted by index 0.
- `run_hw_hbm_amon [lindex [get_hw_hbms] 0]` - Enables the activity monitor runs for the specified hardware HBM(s).
- `stop_hw_hbm_amon [lindex [get_hw_hbms] 0]` - Disables the Activity Monitor runs for the specified hardware HBM(s).

More specific details and examples can be found in Appendix D of *AXI High Bandwidth Controller LogiCORE IP Product Guide* ([PG276](#)).

PCI Express Link Debug

The Versal PCI Express® Integrated Block in Vivado supports link debug. If enabled, the core stores the Link Training and Status State Machine (LTSSM) state transitions which is accessible in the Vivado Hardware Manager.

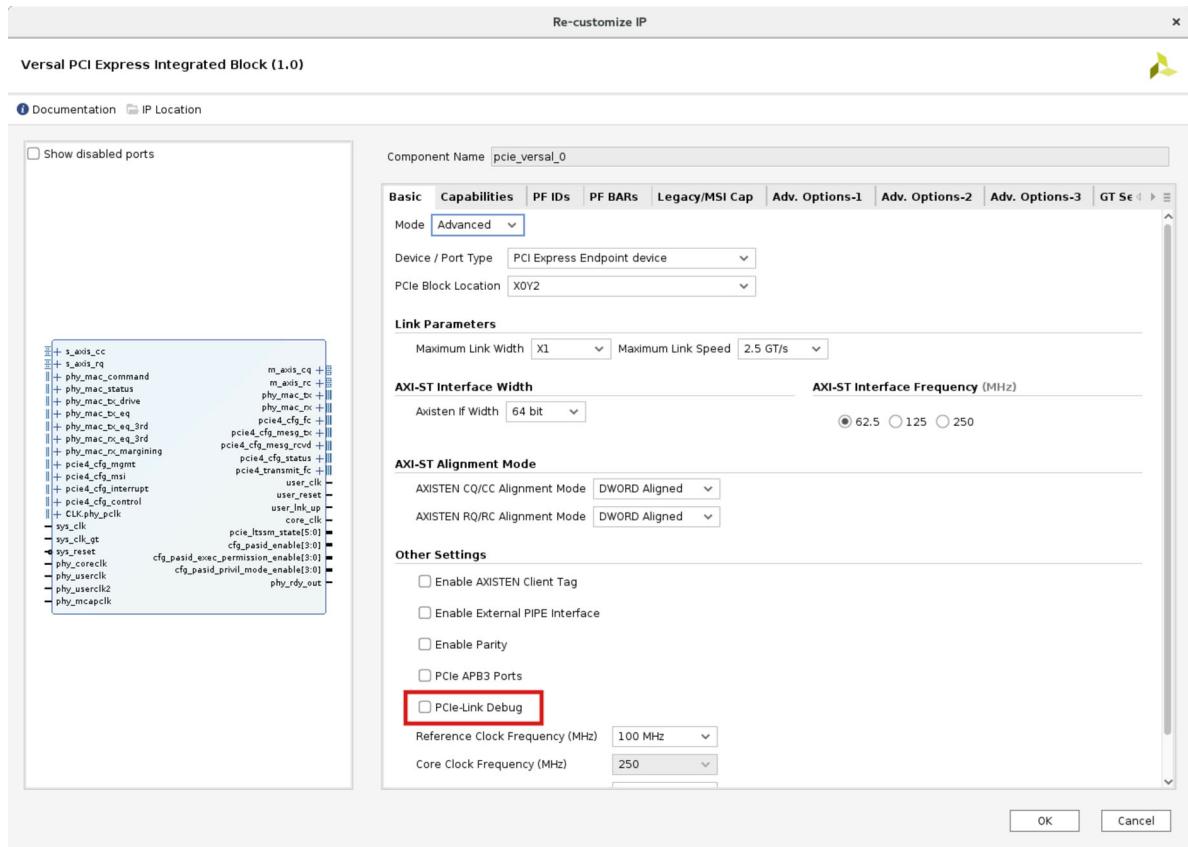
Enabling PCI Express Link Debug

To use PCI Express Link Debug, it must be enabled in the Versal PCI Express Integrated Block IP.

To enable the PCI Express Link Debug feature:

1. Invoke the Versal PCI Express Integrated Block IP configuration GUI.
2. Under the Basic tab, change the Mode to Advanced.
3. Under Other Settings, check PCIe-Link Debug.

Figure: Enabling PCI Express Link Debug in the Versal PCI Express Integrated Block IP Configuration GUI

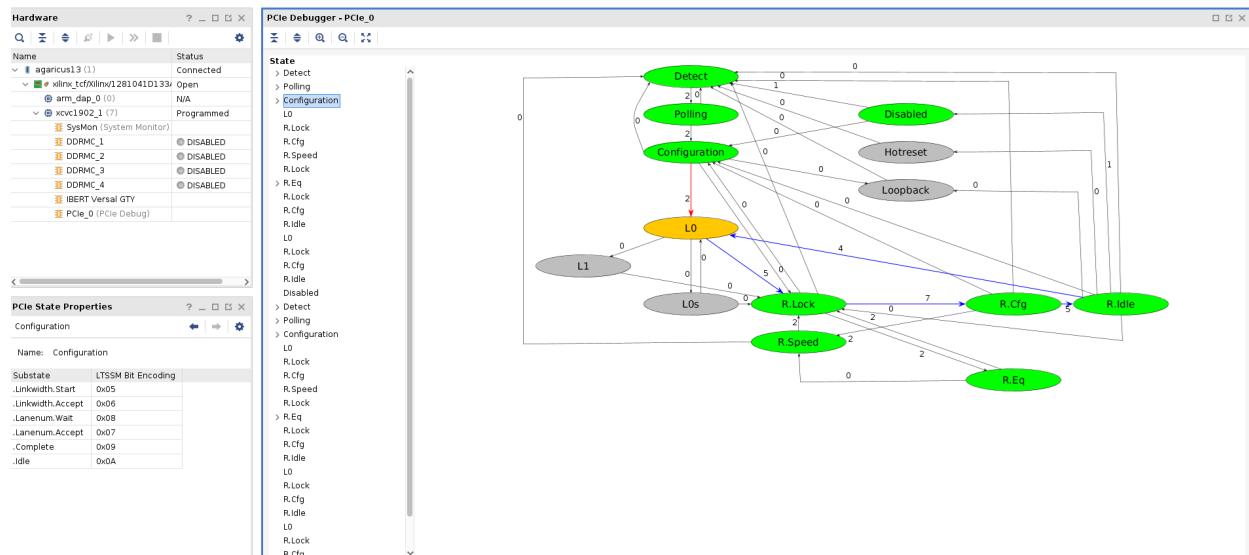


PCI Express Link Debug GUI Usage

Upon configuring the device, if enabled, the PCI Express cores are visible in the Vivado Hardware Manager.

The PCI Express LTSSM debug content is shown in an LTSSM State Transition Diagram. This interface displays an ordered list of the LTSSM state transitions showing which states are visited and a diagram illustrating the visited states and currently occupied states in the LTSSM.

Figure: PCI Express Link Debug Interface



ChipScoPy API

ChipScoPy is an open-source project from AMD that enables high-level control of AMD Versal™ debug IP running in hardware without the need to use the AMD Vivado™ Hardware Manager. Using a simple Python API, developers can develop applications that control and communicate with ChipScope™ debug IP such as the Integrated Logic Analyzer (ILA), Virtual IO (VIO), device memory access, and more.

For more information on the ChipScoPy Python API, refer to <https://github.com/Xilinx/chipscoopy>.

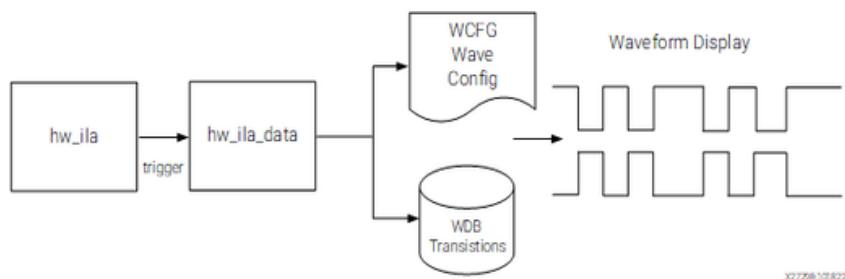
Viewing ILA Probe Data in the Waveform Viewer

The ILA waveform viewer in the AMD Vivado™ Integrated Design Environment (IDE) provides a powerful way to analyze data captured from the ILA Debug Core. After successfully triggering an ILA core and capturing data, Vivado automatically populates a corresponding waveform viewer with data collected from the ILA core. When using Vivado in project mode, configurable waveform settings such as coloring, radix selection, and signal ordering persist and are conveniently remembered between Vivado sessions.

ILA Data and Waveform Relationship

It is useful to understand the relationship between the `hw_ila_data` captured ILA data object and the waveform, as shown in the following figure.

Figure: ILA Data and Waveform Relationship



The `hw_ila` Tcl object represents the ILA core in hardware. Every time an ILA core uploads captured data, it is stored in memory in a corresponding Tcl `hw_ila_data` object. These objects are named predictably so the first ILA core in hardware '`hw_ila_1`' produces data in a corresponding Tcl data object named '`hw_ila_data_1`' after trigger and upload. When working online with hardware, every waveform is backed by the in-memory `hw_ila_data` object and has a 1:1 correspondence with this object illustrated by the diagram in the previous figure. For each Tcl `hw_ila_data` object, a wave database (WDB) file, and wave configuration (WCFG) file are created and automatically tracked in a directory of the Vivado project. The previous figure illustrates the flow of data from the hardware `hw_ila` on the left through to the waveform display on the right.

The combination of the wave configuration, WCFG, file, and wave transition database, WDB, file contain the waveform database and customizations displayed in the Vivado waveform user interface. These waveform files are automatically managed in the Vivado ILA flow and users are not expected to modify the WDB or WCFG files directly. The wave configuration can be modified by changing objects in the waveform viewer (such as signal color, bus radix, signal order, markers, etc). This automatically saves the wave configuration changes to the appropriate WCFG file in the Vivado project.

It is possible to archive waveform configurations and data for later viewing by using the Tcl command `write_hw_ila_data`. This stores the `hw_ila_data`, wave database and wave configuration in an archive for later viewing offline. See the section, Saving and Restoring Captured Data from the ILA Core for details on using `read_hw_ila_data` and `write_hw_ila_data` for offline storage and retrieval of waveforms.

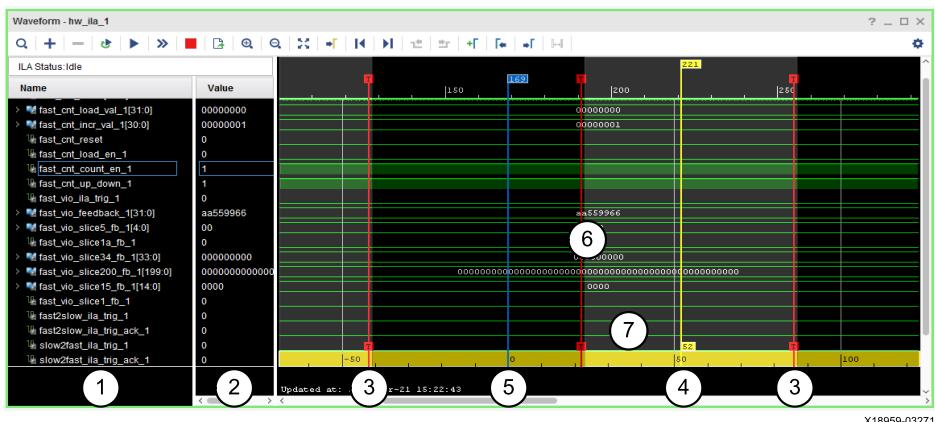
Related Information

[Saving and Restoring Captured Data from the ILA Core](#)

Waveform Viewer Layout

The ILA waveform viewer (sometimes referred to as waveform configuration) is composed of several dynamic objects working together to provide a complete visualization tool for the captured ILA data, as shown in the following figure.

Figure: Waveform Viewer Showing Captured ILA Data



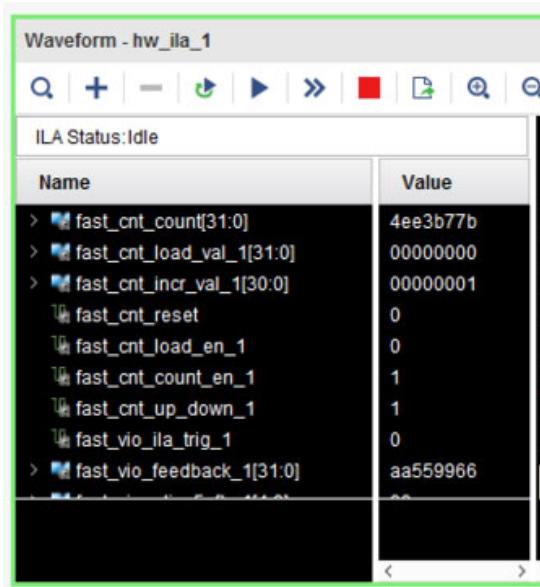
The description for the labeled objects in the previous figure is as follows:

1. Net or Bus Name from the ILA probes file (.ltx)
2. Net or Bus Value at the cursor
3. Trigger Markers (red lines)
4. Cursor (yellow line)
5. Markers (blue line)
6. ILA capture window transitions (alternating clear/grey regions)
7. Floating measurement ruler (yellow bar)

Waveform Viewer Operation

The scalars and buses shown in the Name column of the wave viewer represent the names of the probe design objects in the waveform (see the following figure). These correspond to the hardware probes of the ILA core (see the `get_hw_probes` Tcl command).

Figure: ILA Probe Names and Values Shown in Waveform Viewer



Immediately after triggering and uploading ILA data for the first time, the waveform viewer populates with all probes connected to the ILA core. It is possible to customize probes in the viewer in addition to removing existing probes or adding new probes to the viewer. This section covers the basic operation of the waveform viewer.

Removing Probes from the Waveform

All probes by default are added to the waveform during the first trigger and upload operation. If you do not want the waveform to contain all probes, it is simple to remove probes from the viewer.

To remove a probe from the waveform viewer, right-click the scalar or bus to delete in the Name column and select Delete from the pop-up menu. Alternatively, select the signal or bus to delete and press the Delete key. Probe transition data is not actually deleted from memory it is hidden from view when probes are removed.

Adding Probes to the Waveform

To add probes to the waveform, select the Probes to add for the associated ILA core in the Debug Probes window, right-click, and select Add Probes to Waveform from the pop-up menu.

To add another copy of a signal or bus to the Waveform window, select the signal or bus in the Waveform window. Select Edit > Copy or type Ctrl+C. This copies the object to the clipboard. Select Edit > Paste or type Ctrl+V to paste a copy of the object in the waveform.

You can do the same using the Tcl command add_wave as follows:

```
add_wave -into {hw_ila_data_1.wcfg} -radix hex { {counter1} }
```

In this example, probe counter1 is added to the Waveform Configuration window of hw_ila_1 and its display radix in the Waveform window is set to hex.

Using Waveform ILA Trigger and Export Features

Figure: Waveform ILA Trigger and Export Features



Enable Auto Re-Trigger

Select the Enable Auto Re-Trigger button on the Waveform window toolbar to enable Vivado IDE to automatically re-arm the ILA core associated with the Waveform window trigger after a successful trigger+upload+display operation has completed.

The captured data displayed in the Waveform window corresponding to the ILA core is overwritten upon each successful trigger event. The Auto Re-Trigger option can be used with the Run Trigger and Run Trigger Immediate operations. Click the Stop Trigger button to stop the trigger currently in progress.

Run Trigger

Arms the ILA core associated with the Waveform window to detect the trigger event that is defined by the ILA core basic or advanced trigger settings.

Run Trigger Immediate

Arms the ILA core associated with the Waveform window to trigger immediately regardless of the ILA core trigger settings. This command is useful for detecting the "aliveness" of the design by capturing any activity at the probe inputs of the ILA core.

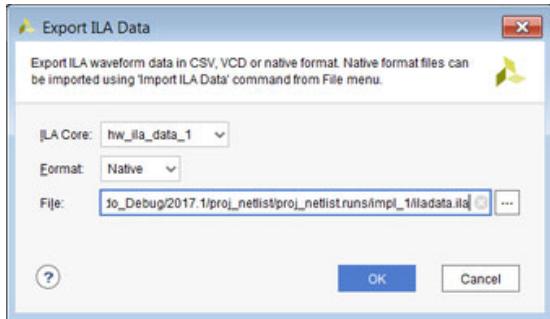
Stop Trigger

Stops the ILA core trigger of the ILA associated with the Waveform window.

Export ILA Data

Captures data from an ILA core and saves it to a file. The data can be captured in either native, .csv, or .vcd format. By clicking this icon on the Waveform window toolbar, the following dialog box appears.

Figure: Export ILA Data Dialog Box



The ILA core is the name of the ILA debug core to export data for. The format is a selection among Native, CSV, and VCD formats.

- Native format configures the `write_hw_iladata` command to export the ILA data in the form of a default ILA file format file that can be used to import into Vivado and back again at another point in time so that you can view previously captured ILA data.
- CSV format configures the `write_hw_iladata` command to export the ILA data in the form of a .csv file that can be used to import the data into a spreadsheet or third-party application.
- VCD file format configures the `write_hw_iladata` command to export the ILA data in the form of a .vcd file that can be used to import into a third-party application or viewer.

!! Important: While ILA data can be exported in the CSV, VCD, and native ILA format, only the native ILA format can be imported into Vivado. Also, native ILA data imported into Vivado is supported only for offline viewing of previously captured data. The probe signals cannot be used for other purposes such as triggering, etc.

Using the Zoom Features

Toolbar buttons provide quick access to waveform zooming features (see the following figure). Alternatively, use the mouse wheel combined with the Ctrl key to zoom in and out of the currently selected waveform.

Note: The zoom level is not persistent and resets between Vivado sessions.

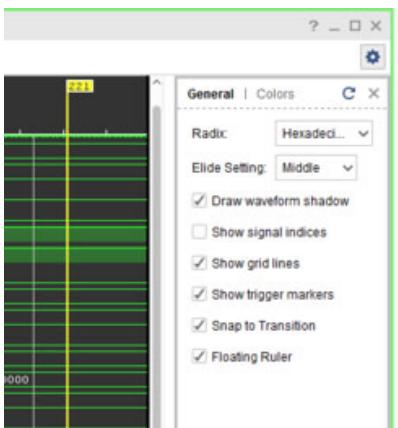
Figure: Waveform Zoom Buttons



Waveform Settings

The waveform viewer allows you to customize the way objects are displayed. When you select the Waveforms Settings button the Waveform Settings window in the following figure opens:

Figure: Waveform Settings



The options are as follows:

Colors tab

Lets you choose custom colors for waveform objects

Radix

Sets the default radix for bus probes

Draw waveform shadow

Displays a light green shadow under scalar '1' to help differentiate between '1' and '0'

Show signal indices

Display index position number to the left side of scalar and bus names

Show trigger markers

Show (or hide) the red trigger markers in the wave viewer

Customizing the Configuration

You can customize the Waveform configuration using the features that are listed and briefly described in the following table; the feature name links to the subsection that fully describes the feature.

Table: Customization Features in the Waveform Configuration

Feature	Description
Cursors	The main cursor and secondary cursor in the Waveform window let you display and measure time, and they form the focal point for various navigation activities.
Markers	You can add markers to navigate through the waveform, and to display the waveform value at a particular time.
Dividers	You can add a divider to create a visual separator of signals.
Using Groups	You can add a group, that is a collection to which signals and buses can be added in the wave configuration as a means of organizing a set of related signals.
Using Virtual Buses	You can add a virtual bus to your wave configuration, to which you can add logic scalars and arrays.
Renaming Objects	You can rename objects, signals, buses, and groups.
Radixes	The default radix controls the bus radix that displays in the wave configuration, Objects panel, and the Console panel.
Bus Bit Order	You can change the Bus bit order from Most Significant Bit (MSB) to Least Significant Bit (LSB) and vice versa.

Related Information

[Cursors](#)

[Markers](#)

[Dividers](#)

[Using Groups](#)

[Using Virtual Buses](#)

[Radixes](#)

[Bus Bit Order](#)

[Renaming Objects](#)

Cursors

Cursors are used primarily for temporary indicators of sample position and are expected to be moved frequently, as in the case when you are measuring the distance (in samples) between two waveform edges.

★ Tip: For more permanent indicators, used in situations such as establishing a time-base for multiple measurements, add markers to the Wave window instead. See [Markers](#) for more information.

You can place the main cursor with a single click in the Waveform window.

To place a secondary cursor, Ctrl+Click and hold the waveform, and drag either left or right. You can see a flag that labels the location at the top of the cursor.

Alternatively, you can hold the Shift key and click a point in the waveform. The main cursor remains in the original position, and the other cursor is at the point in the waveform that you clicked.

 **Note:** To preserve the location of the secondary cursor while positioning the main cursor, hold the Shift key while clicking. When placing the secondary cursor by dragging, you must drag a minimum distance before the secondary cursor appears.

To move a cursor, hover over the cursor until you see the grab symbol and click and drag the cursor to the new location.

As you drag the cursor in the Waveform window, you see a hollow or filled-in circle if the Snap to Transition button is selected, which is the default behavior.

- A hollow circle  indicates that you are between transitions in the waveform of the selected signal.
- A filled-in circle  indicates that you are hovering over the waveform transition of the selected signal. A secondary cursor can be hidden by clicking anywhere in the Waveform window where there is no cursor, marker, or floating ruler.

Related Information

[Markers](#)

Markers

Use a marker when you want to mark a significant event within your waveform in a permanent fashion. Markers allow you to measure distance (in samples) relevant to that marked event.

You can add, move, and delete markers as follows:

- You add markers to the wave configuration at the location of the main cursor.
 1. Place the main cursor at the sample number where you want to add the marker by clicking in the Waveform window at the sample number or on the transition.
 2. Select Edit > Markers > Add Marker or click the Add Marker button. 

A marker is placed at the cursor, or slightly offset if a marker already exists at the location of the cursor. The sample number of the marker is displayed at the top of the line.
- You can move the marker to another location in the waveform using the drag and drop method. Click the marker label (at the top of the marker) and drag it to the location.
 - The drag symbol  indicates that the marker can be moved. As you drag the marker in the Waveform window, you see a hollow or filled-in circle if the Snap to Transition button is selected, which is the default behavior.
 - A filled-in circle  indicates that you are hovering over a transition of the waveform for the selected signal or over another marker.
 - For markers, the filled-in circle is white.
 - A hollow circle  indicates that you are between transitions in the waveform of the selected signal.
 - Release the mouse key to drop the marker to the new location.
- You can delete one or all markers with one command. Right-click over a marker, and do one of the following:
 - Select Delete Marker from the popup menu to delete a single marker.
 - Select Delete All Markers from the popup menu to delete all markers.

 **Note:** You can also use the Delete key to delete a selected marker.

 - Use Edit > Undo to reverse a marker deletion.

Trigger Markers

The red trigger marker (whose label is a red letter 'T') a special marker that indicates the occurrence of the trigger event in the capture buffer. The position of the trigger marker in the buffer directly corresponds to the Trigger Position setting (see Using the ILA Default Dashboard).

 **Note:** The trigger markers are not movable using the same technique as regular markers. Set their position using the ILA core's Trigger Position property setting.

Related Information

[Using the ILA Default Dashboard](#)

Dividers

Dividers create a visual separator between signals. You can add a divider to your wave configuration to create a visual separator of signals, as follows:

1. In the Name column of the Waveform window, click a signal to add a divider below that signal.
2. From the pop-up menu, select Edit > New Divider, or right-click and select New Divider.

The change is visual and nothing is added to the HDL code. The new divider is saved with the wave configuration file when you save the file.

You can move or delete Dividers as follows:

- Move a Divider to another location in the waveform by dragging and dropping the divider name.
- To delete a Divider, highlight the divider, and click the Delete key, or right-click and select Delete from the pop-up menu.

You can also rename Dividers; see Renaming Objects.

Related Information

[Renaming Objects](#)

Using Groups

A Group is a collection of expandable and collapsible categories, to which you can add signals and buses in the wave configuration to organize related sets of signals. The group itself displays no waveform data but can be expanded to show its contents or collapsed to hide them. You can add, change, and remove groups.

To add a Group:

1. In a wave configuration, select one or more signals or buses to add to a group.

 **Note:** A group can include dividers, virtual buses, and other groups.

2. Select Edit > New Group, or right-click and select New Group from the popup menu.

A Group that contains the selected signal or bus is added to the wave configuration.

A Group is represented with the Group button .

The change is visual and nothing is added to the ILA core.

You can move other signals or buses to the group by dragging and dropping the signal or bus name.

You can move or remove Groups as follows:

- Move Groups to another location in the Name column by dragging and dropping the group name.
- Remove a group, by highlighting it and selecting Edit > Wave Objects > Ungroup, or right-click and select Ungroup from the popup menu. Signals or buses formerly in the group are placed at the top-level hierarchy in the wave configuration.

Groups can be renamed also; see Renaming Objects.

⚠ CAUTION! The Delete key removes the group and its nested signals and buses from the wave configuration.

Related Information

[Renaming Objects](#)

Using Virtual Buses

You can add a virtual bus to your wave configuration, which is a grouping to which you can add logic scalars and arrays. The virtual bus displays a bus waveform, which shows the signal waveforms in the vertical order that they appear under the virtual bus, flattened to a one-dimensional array. You can change or remove virtual buses after adding them.

To add a virtual bus:

1. In a wave configuration, select one or more signals or buses you want to add to a virtual bus.
2. Select Edit > New Virtual Bus, or right-click and select New Virtual Bus from the pop-up menu.
The virtual bus is represented with the Virtual Bus button .
The change is visual and nothing is added to the HDL code.

You can move other signals or buses to the virtual bus by dragging and dropping the signal or bus name. The new virtual bus and its nested signals or buses are saved when you save the wave configuration file. You can also move it to another location in the waveform by dragging and dropping the virtual bus name.

You can rename a virtual bus; see [Renaming Objects](#).

To remove a virtual bus, and ungroup its contents, highlight the virtual bus, and select Edit > Wave Objects > Ungroup, or right-click and select Ungroup from the pop-up menu.

⚠ CAUTION! The Delete key removes the virtual bus and its nested signals and buses from the wave configuration.

Related Information

[Renaming Objects](#)

Renaming Objects

You can rename any object in the Waveform window, such as signals, dividers, groups, and virtual buses.

1. Select the object name in the Name column.
2. Select Rename from the popup menu.
3. Replace the name with a new one.
4. Press Enter or click outside the name to make the name change take effect.

You can double-click the object name and type a new name. The change is effective immediately. Object name changes in the wave configuration do not affect the names of the nets attached to the ILA core probe inputs.

Radixes

Understanding the type of data on your bus is important. You need to recognize the relationship between the radix setting and the data type to use the waveform options of Digital and Analog effectively. See [Bus Radixes](#) for more information about the radix setting and its effect on Analog waveform analysis.

You can change the radix of an individual signal (ILA probe) in the Waveform window as follows:

1. Right-click a bus in the Waveform window.
2. Select Radix and the format you want from the drop-down menu:
 - Binary
 - Hexadecimal
 - Unsigned Decimal
 - Signed Decimal
 - Octal

!! Important: Changes to the radix of an item in the Objects window do not apply to values in the Waveform window or the Tcl Console. To change the radix of an individual signal (ILA probe) in the Waveform window, use the Waveform window popup menu.

- Maximum bus width of 64 bits on real. Incorrect values are possible for buses wider than 64 bits.
- Floating point supports only 32 and 64-bit arrays.

Related Information

[Bus Radixes](#)

Using the Floating Ruler

The floating ruler assists with sample measurements using a sample number base other than the absolute sample numbers shown on the standard ruler at the top of the Waveform window.

You can display (or hide) a floating ruler and move it to a location in the Waveform window. The sample base (sample 0) of the floating ruler is the secondary cursor, or, if there is no secondary cursor, the selected marker.

The floating ruler button and the floating ruler itself are visible only when the secondary cursor (or selected marker) is present.

1. Do either of the following to display or hide a floating ruler:
 - Place the secondary cursor.
 - Select a marker.
2. Select View > Floating Ruler.

You only need to follow this procedure the first time. The floating ruler displays each time the secondary cursor is placed or a marker is selected.
Select the command again to hide the floating ruler.

Bus Bit Order

You can reverse the bus bit order in the wave configuration to switch between MSB-first and LSB-first signal representation.

To reverse the bit order:

1. Select a bus.
2. Right-click and select Reverse Bit Order.

The bus bit order is reversed. The Reverse Bit Order command is marked to show that this is the current behavior.

Bus Radixes

Bus values are interpreted as numeric values, which are determined by the radix setting on the bus wave object, as follows:

- Binary, octal, hexadecimal, ASCII, and unsigned decimal radices cause the bus values to be interpreted as unsigned integers. The format of data on the bus must match the radix setting.
- Any non-0 or -1 bits cause the entire value to be interpreted as 0.
- The signed decimal radix causes the bus values to be interpreted as signed integers.

Viewing Analog Waveforms

To convert a digital waveform to analog, do the following:

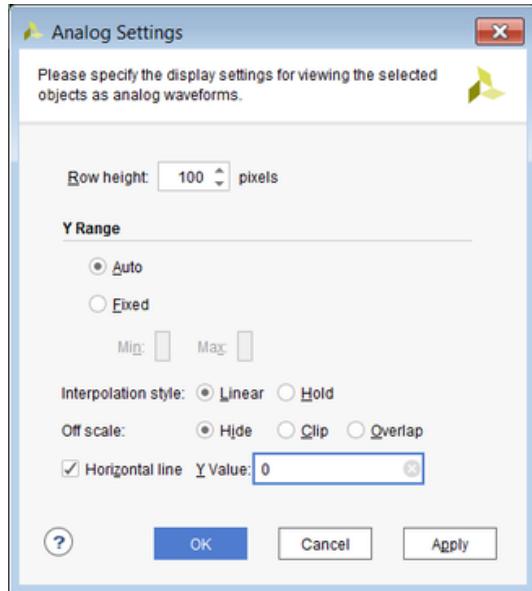
1. In the Name area of a Waveform window, right-click the bus.
2. Select Waveform Style and Analog Settings to choose an appropriate drawing setting.

The digital drawing of the bus converts to an analog format.

You can adjust the height of either an analog waveform or a digital waveform by selecting and dragging the rows.

The following figure shows the Analog Settings dialog box with the settings for analog waveform drawing.

Figure: Analog Settings Dialog Box



The Analog Settings dialog box options are as follows:

Row Height

Specifies how tall to make the select wave objects, in pixels. Changing the row height does not change how much of a waveform is exposed or hidden vertically, but rather stretches or contracts the height of the waveform.

When switching between Analog and Digital waveform styles, the row height is set to an appropriate default for the style (20 for digital, 100 for analog).

Y Range

Specifies the range of numeric values to be shown in the waveform area.

Auto

Specifies that the range should continually expand whenever values in the visible time range of the window are discovered to lie outside the current range.

Fixed

Specifies that the time range is to remain at a constant interval.

Min

Specifies the value displays at the bottom of the waveform area.

Max

Specifies the value displays at the top.

Both values can be specified as floating point; however, if radix of the wave object radix is integral, the values are truncated to integers.

Interpolation Style

Specifies how the line connecting data points is to be drawn.

Linear

Specifies a straight line between two data points.

Hold

Specifies that of two data points, a horizontal line is drawn from the left point to the X-coordinate of the right point, another line is drawn connecting that line to the right data point, in an L shape.

Off Scale

Specifies how to draw waveform values that lie outside the Y range of the waveform area.

Hide

Specifies that outlying values are not shown, such that a waveform that reaches the upper or lower bound of the waveform area disappears until values are again within the range.

Clip

Specifies that outlying values be altered so that they are at the top or bottom of the waveform area, such that a waveform that reaches the upper- or lower-bound of the waveform area follows the bound as a horizontal line until values are again within the range.

Overlap

Specifies that the waveform be drawn wherever its values are, even if they lie outside the bounds of the waveform area and overlap other waveforms, up to the limits of the wave window itself.

Horizontal Line

Specifies whether to draw a horizontal rule at the given value. If the check-box is on, a horizontal grid line is drawn at the vertical position of the specified Y value, if that value is within the Y range of the waveform.

As with Min and Max, the Y value accepts a floating point number but truncates it to an integer if the radix of the selected wave objects is integral.

!! Important: Analog settings are saved in a wave configuration; however, because control of zooming in the Y dimension is highly interactive, unlike other wave object properties such as radix, they do not affect the modification state of the wave configuration. Consequently, zoom settings are not saved with the wave configuration.

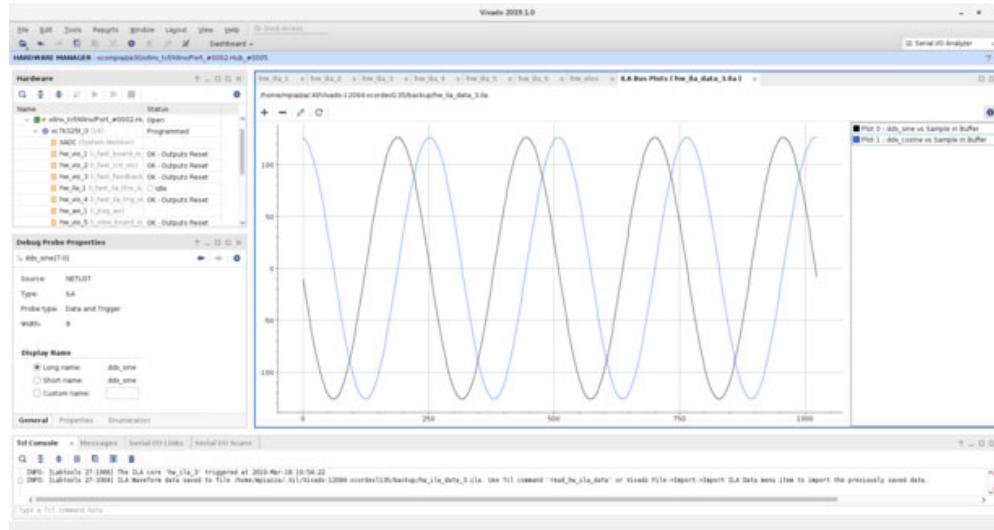
Bus Plot Viewer

In addition to the analog waveform viewer the Vivado Hardware Manager also supports the Bus Plot Viewer, which allows viewing a bus's values over time, or one bus' values vs. another bus' values plotted on an X vs Y axis.

It can be helpful to view a Bus Plot when it is necessary to perform either of the following:

- Plotting analog sample data vs. time
- Plotting analog sample data vs. analog sample data

Figure: Bus Plot Showing Trigger Data vs. the Sample in Buffer



Creating a Bus Plot

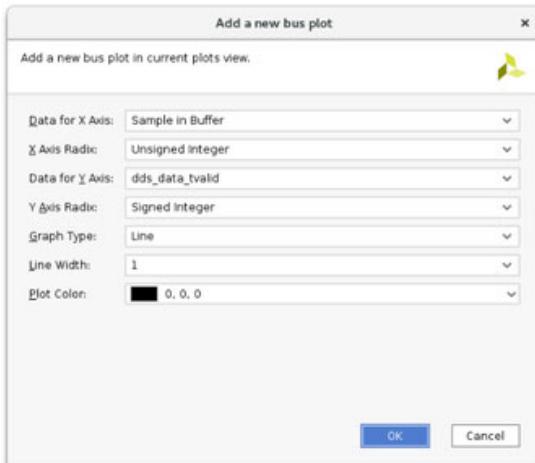
The Bus Plot viewer can be used to plot previously captured ILA trace data. As such, it requires the path to an ILA bus plot data file (either .csv or .ila). By default, the Show Bus Plot dialog selects the last auto-saved ILA trigger data.

Example of Bus Plot Creation

1. To create a Bus Plot, open the Vivado Hardware Manager and select Tools > Show Bus Plot.
2. The Show Bus Plot dialog box appears allowing the selection of a previously saved ILA data file. By default, the auto-saved ILA data file corresponding to the last ILA trace is auto-selected. To select a previously saved ILA data file, enter the path to the corresponding .ila or .csv file.



3. Click OK and a blank Bus Plot window appears. To add a Bus Plot, click the "+" symbol, and a dialog box appears with options to configure the new Bus Plot.

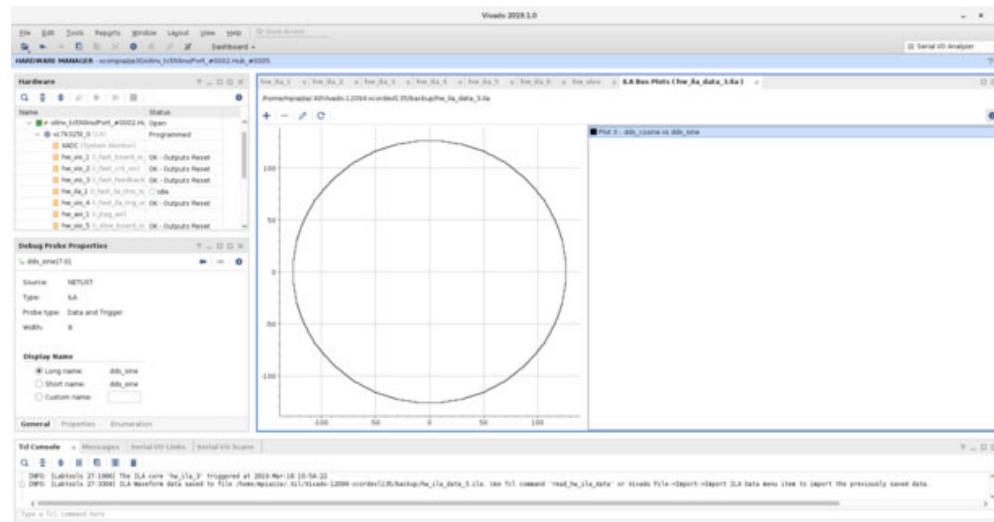


The Add New Bus Plot options are as follows:

- Data for X Axis: Specifies the bus data to use for the X axis.
 - Sample in Buffer: ILA sample number in the ILA capture buffer.
 - Sample in Window: ILA sample number in the capture window. If one capture window is selected, this number is the same as the sample in buffer, however if you use multiple capture windows, this number indicates the sample number for the given capture window.
 - TRIGGER: Trigger position in the capture window.
- X Axis Radix: Specifies the radix to use when plotting the X Axis data.
 - Signed Integer.
 - Unsigned Integer.
- Data for Y Axis: Specifies the bus data to use for the Y axis.
 - Sample in Buffer: ILA sample number in the ILA capture buffer.
 - Sample in Window: ILA sample number in the capture window. If one capture window is selected, this number is the same as the sample in buffer, however if you use multiple capture windows, this number indicates the sample number for the given capture window.
 - TRIGGER: Trigger position in the capture window.
- Y Axis Radix: Specifies the radix to use when plotting the Y Axis data.
 - Signed Integer.
 - Unsigned Integer.
- Graph Type
 - Line: Display bus plot as a continuous line connected between discrete samples.
 - Point: Display bus plot as points representing discrete samples.
- Line Width: Specifies the width to use for drawing the signals in the Bus Plot viewer.
- Plot Color: Allows choosing a different color to draw the Bus Plot.

4. After configuring the Bus Plot, click OK to add the Bus Plot to the Vivado Hardware Manager.

The Bus Plot is displayed.

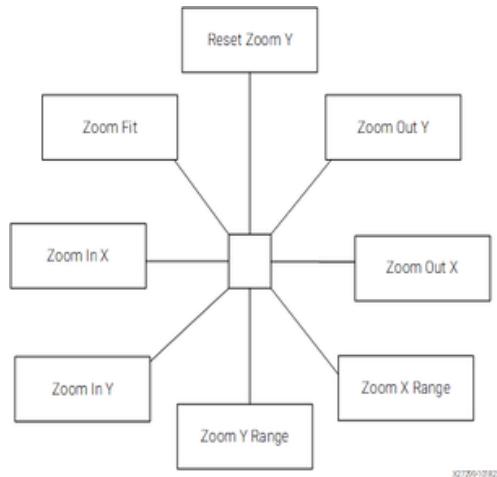


Note: Bus Plot settings are not saved when exiting the Vivado Hardware Manager. Be sure that all desired measurements are completed before closing Vivado IDE.

Zoom Gestures

In addition to the zoom gestures supported for zooming in the X dimension, when over an analog waveform, additional zoom gestures are available, as shown in the following figure.

Figure: Analog Zoom Options



To invoke a zoom gesture, hold down the left mouse button and drag in the direction indicated in the diagram, where the starting mouse position is the center of the diagram.

The additional Zoom gestures are as follows:

Zoom Out Y

Zooms out in the Y dimension by a power of 2 determined by how far away the mouse button is released from the starting point. The zoom is performed such that the Y value of the starting mouse position remains stationary.

Zoom Y Range

Draws a vertical curtain which specifies the Y range to display when the mouse is released.

Zoom In Y

Zooms in toward the Y dimension by a power of 2 determined by how far away the mouse button is released from the starting point.

The zoom is performed such that the Y value of the starting mouse position remains stationary.

Reset Zoom Y

Resets the Y range to that of the values currently displayed in the wave window and sets the Y Range mode to Auto.

All zoom gestures in the Y dimension set the Y Range analog settings. Reset Zoom Y sets the Y Range to Auto, whereas the other gestures set Y Range to Fixed.

Debugging Designs Post Implementation

You might want to modify, add, or delete your debug cores post implementation. There are two ways to do it in AMD Vivado™ Design Suite.

If you want to replace the existing connections to the ILA cores, AMD recommends that you use the ECO flow. The ECO flow operates on an implemented checkpoint (DCP) and could save you time that could otherwise be spent in a complete re-route of the design.

If you want to add new ILA cores, delete existing ILA cores, or modify existing ILA cores (for example, resizing probe width, changing the data depth, etc.), AMD recommends that you use the Incremental Compile flow. The Incremental flow for debug cores operates on a synthesized design or checkpoint (DCP) and uses a reference implemented checkpoint (ideally from a previous run of implementation). This could save you time that could otherwise be spent in a complete re-implementation of the design.

The following sections discuss each of these debug related flows in detail.

Using Vivado ECO Flow to Replace Existing Debug Probes

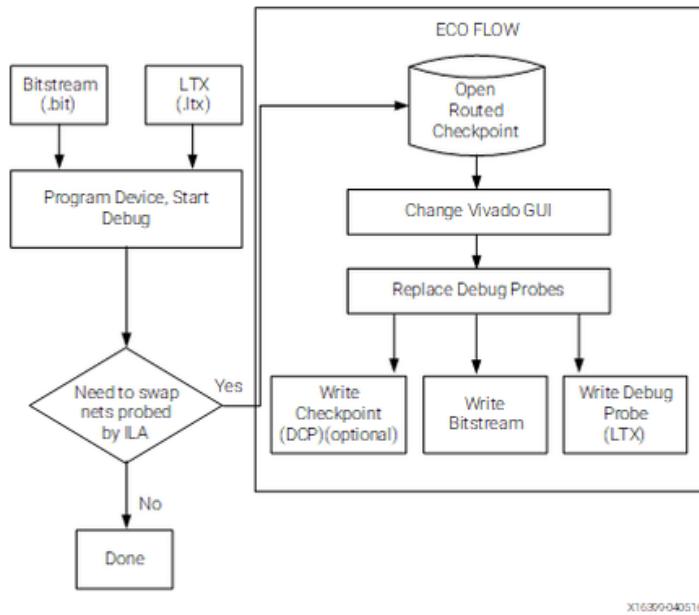
It is possible to replace debug nets connected to an ILA core in a placed and routed design checkpoint. You can do this by using the Engineering Change Order (ECO) flow. This is an advanced design flow used for designs that are nearing completion, where you need to swap nets connected to an ILA probe port. This method serves two purposes:

- **It saves you time.** This feature lets you swap existing debug nets that are being probed for different nets.
- **It is minimally invasive.** After replacing probed nets, it is necessary to route those nets to the inputs of the debug core. The rest of the design remains intact, thereby not only preserving previous implementation results, but also reducing the possibility that a re-implementation hides the bug you are trying to find.

!! Important: This flow is only applicable to designs where ILA cores have already been instantiated or inserted.

The following figure shows the process of replacing debug nets using the ECO design flow.

Figure: Debug ECO Design Flow



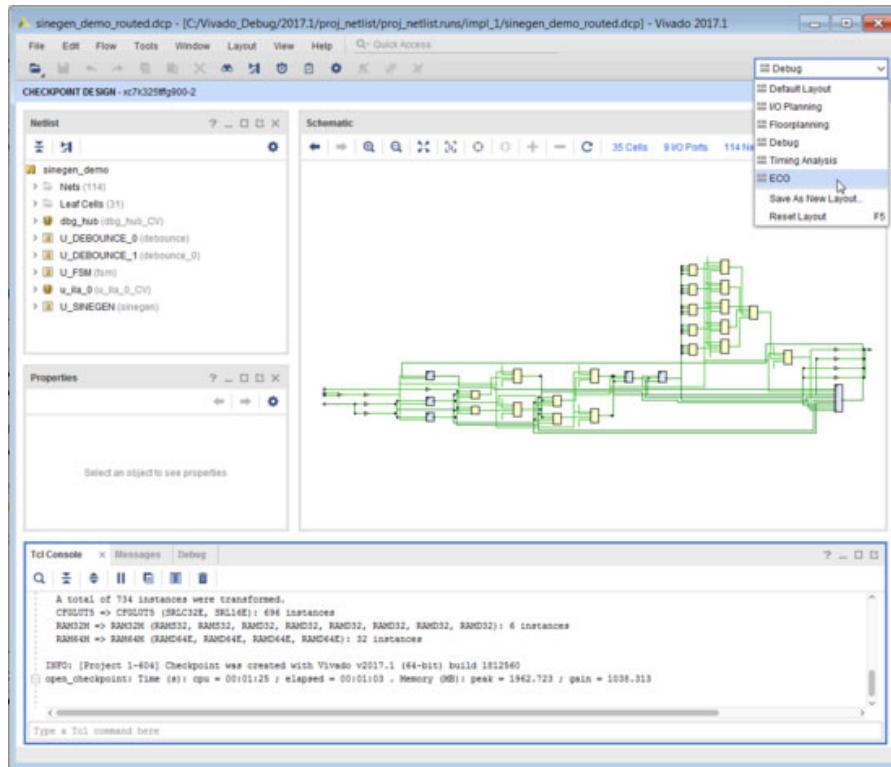
X16399-040516

Replacing Debug Probes on a Placed and Routed Design Checkpoint

When using the Vivado Hardware Manager to debug a design that has been programmed on a device, the nets being probed for debug sometimes need to be swapped for other alternative nets. Instead of going back and changing your RTL code, or changing the nets being probed in the inserted debug cores, you can use the ECO flow to replace the debug nets.

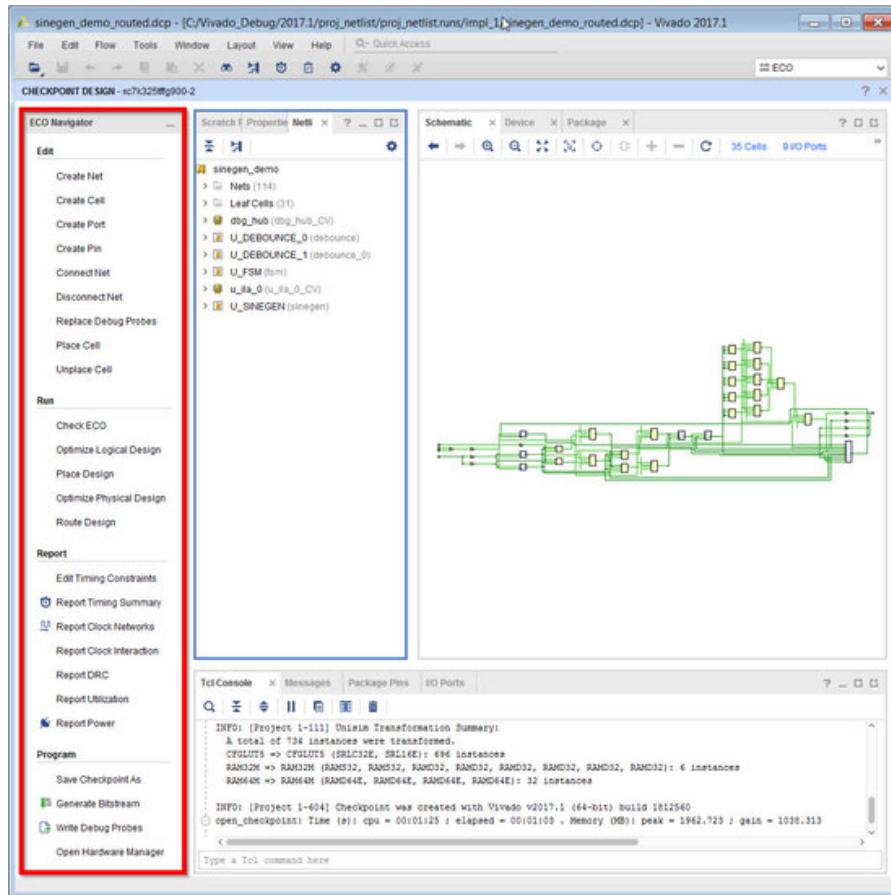
To use the ECO flow, open the placed and routed design checkpoint (DCP), in the Vivado IDE, and change the layout to ECO.

Figure: Selecting the ECO Layout



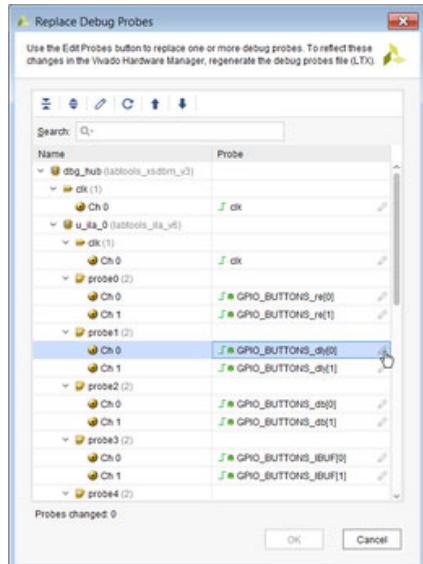
The Flow Navigator now changes to ECO Navigator with a different set of options

Figure: ECO Navigator



In the ECO Navigator, click Replace Debug Probes to bring up the Replace Debug Probes dialog box.

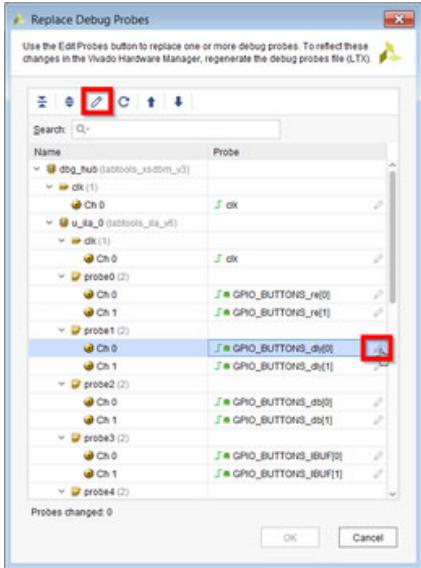
Figure: Replace Debug Probes Dialog Box



In the Replace Debug Probes dialog box, highlight the probes whose nets you want to change, and click the Edit Probes button. Use the Edit Probes button to the right of each probe to change

individual nets. Alternatively, you can use the Edit Probes button on the left edge of the window to change the nets for multiple probes.

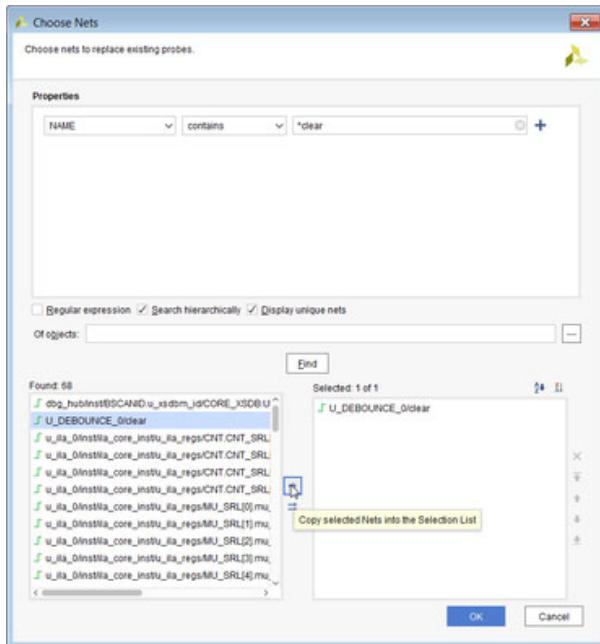
Figure: Edit Probes Button



Click the Edit Probes button to bring up the Choose Nets dialog box, where you can choose nets to replace the existing ones.

Enter the Find criterion to select the nets you want to replace the existing nets. If the Find criterion returns more than 10000 nets, refine your criterion and try again. Select the preferred nets on the Find Results on the left and click the arrow (->) to add those nets to the Selected Names column on the right. Ensure the nets in the Selected Names column on the right correspond to the number of nets being replaced. Click OK to continue.

Figure: Choose Nets Dialog Box



!! Important: Once you have completed replacing all the debug probes necessary, rerouted them, and regenerated the bitstream, you must regenerate the debug probes file (.ltx).

★ Tip: You can also choose multiple nets or a bus by clicking on the Edit Probes button on the left in the Replace Debug Probes dialog box.

After you replace all the desired nets on the debug cores, click OK to bring up a confirmation dialog box to confirm the changes about to be made.

!! Important: Check the Tcl Console to ensure that there are no Warnings/Errors.

Figure: Modify Debug Probe Tcl Messages

```

Tcl Console < Messages | Debug | Package Pins | I/O Ports
modify_debug_ports -probes [list {design_1_1/top_0/system_11a_0/inst/11a_1/lib/probe0 0 s1_sport0_1[16]}]
NetList sorting complete. Time (s): cpu = 00:00:00.02 ; elapsed = 00:00:00.01 . Memory (MB): peak = 9819.992 ; gain = 0.000 ; free physical = 27249 ; free virtual = 62958
INFO: [Vivade_Tcl 4-960] Removed DONT_TOUCH property on net design_1_1/top_0/via_0_probe_out0 to prepare for debug probe changes.
INFO: [Vivade 12-3773] The DONT_TOUCH property on this net is implied by a MARK_DEBUG. Setting the DONT_TOUCH property to FALSE or 0 will enable the implementation tools to q
Starting Physical Synthesis Task
Phase I Physical Synthesis Initialization
INFO: [FPhysical 32-721] Multithreading enabled for phys_opt_design using a maximum of 8 CPUs
INFO: [FPhysical 32-668] Current Timing Summary | MHS=7.320 | TMH=0.000 | WHS=0.031 | THS=0.000 |
Phase I Physical Synthesis Initialization | Checksum: 25304c725
Time (s): cpu = 00:00:26 ; elapsed = 00:00:20 . Memory (MB): peak = 10048.645 ; gain = 228.652 ; free physical = 27122 ; free virtual = 62844
INFO: [FPhysical 32-9611] Swapping nets in group of 12:
<snip>
Type a Tcl command here

```

Deleting any net segment on the path of a net being probed could have an impact on the probe names displayed in the Hardware Manager. Vivado IDE picks the net segment closest to the net being probed with a MARK_DEBUG attribute on it. If there is no net segment with a MARK_DEBUG attribute, the top level net is picked. If there is more than one net segment with a MARK_DEBUG attribute, the tool randomly selects one of those nets.

After you have replaced all the debug probe ports, you can save your modifications to a new checkpoint using the Save Checkpoint As option in the ECO Navigator. The Replace Debug Probes command in the ECO Navigator needs to be run to generate a new .ltx file for the debug probes. You should generate a new bit file to program the device. You can connect to the Vivado Hardware Manager to debug the design with the new changes.

Vivado ECO Tcl Flow to Replace Existing Debug Probes

You can use the Vivado Tcl flow as an alternative to the GUI flow described in the previous sections. Use the following Tcl command to modify nets being probed by the debug cores.

```
modify_debug_ports -probes [list {top/x_ila/probe0 0 top/inst_A/net_0} \
    {top/x_ila/probe1 1 top/inst_A/net_a} {top/x_ila/probe1 2
    top/inst_A/net_b}]
```

This command performs all of the netlist modifications to disconnect existing net connections to the specified probe ports. In this example the existing net connections to probe port 0 at index 0, probe port 1 at index 1 and index 2 of the ILA are disconnected. Each of these probes are connected to the net specified net_0, net_a, and net_b respectively. The modified connections are also routed automatically. Nets that become disconnected during the process are left unconnected.

Incremental Compile with Debug Core (ILA) Modifications

Incremental Compile is an advanced design flow for designs that are nearing completion, where small changes are required. After resynthesizing these small changes, the flow:

- Speeds up place and route runtime.
- Preserves QoR predictability by reusing prior placement and routing from a reference design.
The flow is most effective when synthesis changes result in at least 95 percent similarity to the reference design.

Incremental Debug changes are applied to a placed and routed design by re-implementing the design using the Incremental Compile design flow. This flow is recommended in situations where:

- The debug cores are absent from the existing implemented design or,
- You need to modify existing debug cores by changing probe widths, data depth, and so on.
- You need to delete debug cores from the design.

Incremental Compile Flow Designs

The Incremental Compile flow involves two different designs, the *reference* design and the *current* design with debug core modifications.

Reference Design

The reference design is usually an earlier iteration or variation of the current design that has been synthesized, placed, and routed. However, you can use a checkpoint that has any amount of placement, routing, or both. The reference design checkpoint (DCP) might be the product of many design iterations involving code changes, floorplanning, and revised constraints necessary to close

timing. After the current design is loaded, load the reference design checkpoint using the `read_checkpoint -incremental <dcp>` command. Loading the reference design checkpoint with the `-incremental` option enables the Incremental Compile design flow for subsequent place and route operations.

Current Design

The current design incorporates small debug related design changes or variations from the reference design. These changes or variations can include:

- Debug core RTL instantiation changes
- Debug core insertion changes
- Both debug core related RTL changes and insertion changes

To insert, delete, or modify debug cores to an existing design that has been implemented, open the synthesized DCP or design, and use the debug insertion flow. Details on the debug insertion flow can be found in [Using the Netlist Insertion Debug Probing Flow](#).

You can also modify existing debug cores or instantiate new debug cores into your existing RTL design. The Incremental Compile flow reuses placement and routing from the reference design along with the new debug related modifications. Details on the debug instantiation flow can be found in [HDL Instantiation Debug Probing Flow Overview](#).

Related Information

[Using the Netlist Insertion Debug Probing Flow](#)

[HDL Instantiation Debug Probing Flow Overview](#)

Using Incremental Compile

In both Project Mode and Non-Project Mode, incremental place and route mode is entered when you load the reference design checkpoint using the `read_checkpoint -incremental <reference_dcp_file>` command where `<reference_dcp_file>` specifies the path and file name of the reference design checkpoint. Loading the reference design checkpoint with the `-incremental` option enables the Incremental Compile design flow for subsequent place and route operations. In Non-Project Mode, `read_checkpoint -incremental` should: (1) follow `opt_design` and; (2) precede `place_design`. If using the debug insertion flow the debug core related XDC commands should precede `opt_design`.

Using Incremental Compile in Non-Project Mode

To specify a design checkpoint file (DCP) to use as the reference design, and to run incremental place in Non-Project Mode:

1. Load the current design.
2. Run debug core commands.
3. Run `opt_design`.

!! Important: Make sure the `opt_design` options and directives match those used in the original reference run as closely as possible.

4. Run `read_checkpoint -incremental <reference_dcp_file>`.

5. Run `place_design`.

6. Run `route_design`.

```
# to load the current design
link_design;
#Create the debug core
create_debug_core u_ila_0 ila
#set debug core properties
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
#connect the probe ports in the debug core to the signals being probed in the
design
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk ]]
set_property port_width 1 [get_debug_ports u_ila_0/probe0]
connect_debug_port u_ila_0/probe0 [get_nets [list A_or_B]]
create_debug_port u_ila_0 probe
opt_design
read_checkpoint -incremental <reference_dcp_file>
place_design
route_design
```

!! Important: You must open the synthesized checkpoint to modify the debug cores in the design. Insertion of debug cores by opening a post-routed checkpoint is not supported.

Using Incremental Compile in Project Mode

In Project Mode, you can set the incremental compile option in the Design Runs window.

To set the incremental compile option:

1. Select a run in the Design Runs window.
2. Click Set Incremental Compile from the context menu.
3. In the Set Incremental Compile window, select a reference design checkpoint. This enables incremental compile mode for the run.
4. Open the Synthesized netlist and optionally modify/add the debug cores instantiated in the RTL.
5. Use the Set Up Debug wizard to insert/delete/modify debug cores inserted into the design.
6. Implement Design.

!! Important: You must open the synthesized design to modify the debug cores in the design. Insertion of debug cores by opening a post-routed design is not supported.

For more information, see the Incremental Compile feature in the *Vivado Design Suite User Guide: Implementation* ([UG904](#)).

Examining the Similarity Between the Reference Design and the Current Design

Run `report_incremental_reuse` to examine and report the similarity between a reference design checkpoint file and the current design. The `report_incremental_reuse` command compares the netlist from the reference design checkpoint with the current in-memory design, and reports the percentage of matching of cells, nets, and ports.

A higher degree of design similarity results in more effective reuse of placement and routing from the reference design. The greater the percentage of similarity between the reference design and current design, the greater the opportunity for placement and routing reuse.

Serial I/O Hardware Debugging Flows

Related Information

[Debugging the Serial I/O Design in Hardware](#)

Serial I/O Hardware Debugging Flows

 **Note:** For AMD Versal™ Serial I/O Hardware Debugging Flows, see [Versal Serial I/O Hardware Debugging Flows](#).

The AMD Vivado™ IDE provides a quick and easy way to generate a design that helps you debug and verify your system that uses AMD high-speed gigabit transceiver (GT) technology. The in-system serial I/O debugging flow has three distinct phases:

1. IBERT Core generation phase: Customizing and generating the IBERT core that best meets your hardware high-speed serial I/O requirements.
2. IBERT Example Design Generation and Implementation phase: Generating the example design for the IBERT core generated in the previous step.
3. Serial I/O Analysis phase: Interacting with the IBERT IP contained in the design to debug and verify issues in your high-speed serial I/O links.

The rest of this chapter shows how to complete the first two phases. The third phase is covered in [Debugging the Serial I/O Design in Hardware](#).

Generating an IBERT Core using the Vivado IP Catalog

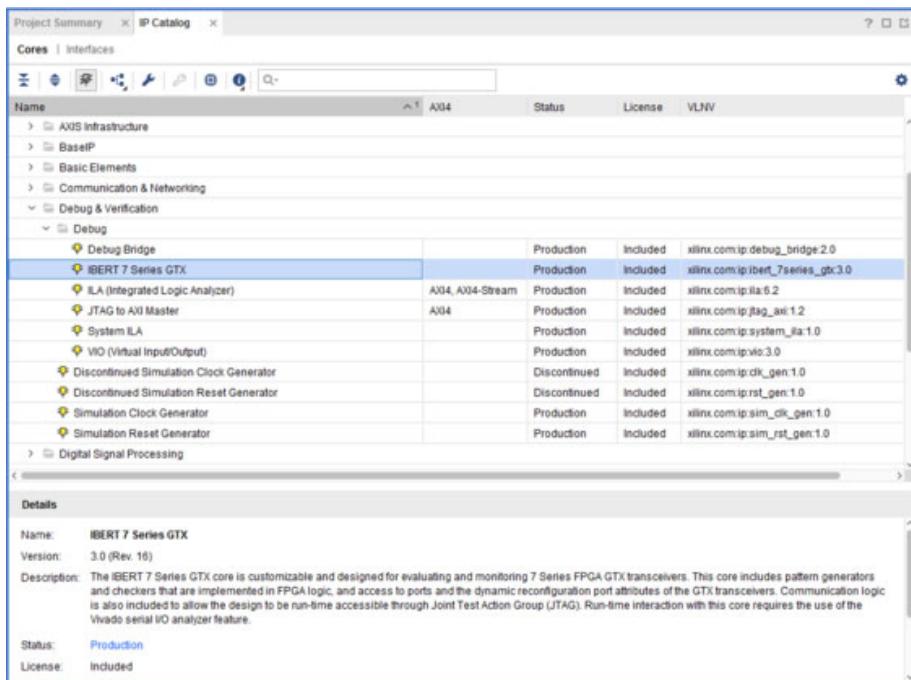
The first phase of getting a suitable hardware design to help debug and validate your system's high-speed serial I/O interfaces is to generate the IBERT core. The following steps outline how to do this:

1. Open the Vivado IDE.
2. On the first panel, choose `Manage IP > New IP Location`, click `Next` when the `Open IP Catalog` wizard opens.
3. Select the desired part, target language, target simulator, and IP location. Click `Finish`.

4. In the IP Catalog under Debug and Verification > Debug, find one or more available IBERT cores as shown in the following figure, depending on the device selected in the previous step.
5. Double-click the desire IBERT architecture to open the Customize IP Wizard for that core.

Customize the IBERT core for your given hardware system requirements. For details on the various IBERT cores available, see the following IP Documents:

- *Integrated Bit Error Ratio Tester 7 Series GTX Transceivers LogiCORE IP Product Guide* ([PG132](#))
- *Integrated Bit Error Ratio Tester 7 Series GTP Transceivers LogiCORE IP Product Guide* ([PG133](#))
- *Integrated Bit Error Ratio Tester 7 Series GTH Transceivers LogiCORE IP Product Guide* ([PG152](#))



Generating and Implementing the IBERT Example Design

After generating the IBERT IP core, it appears in the Sources window as `ibert_7series_gtx` or something similar. To generate the example design, right-click the IBERT IP in the Sources window and select Open IP Example Design, specify the desired location of the example design project in the resulting dialog window. This command opens a new Vivado project window for the example design and adds the proper top-level wrapper and constraints file to the project, as shown in the following figure.

!! Important: Modification of the IBERT IP example design is not recommended and can result in functional issues when interacting with the IBERT IP core in hardware.

Once the example design is generated, you can implement the IBERT example design through bitstream creation core by clicking Generate Bitstream in the Program and Debug section of the Vivado IDE flow navigator or by running the following Tcl commands:

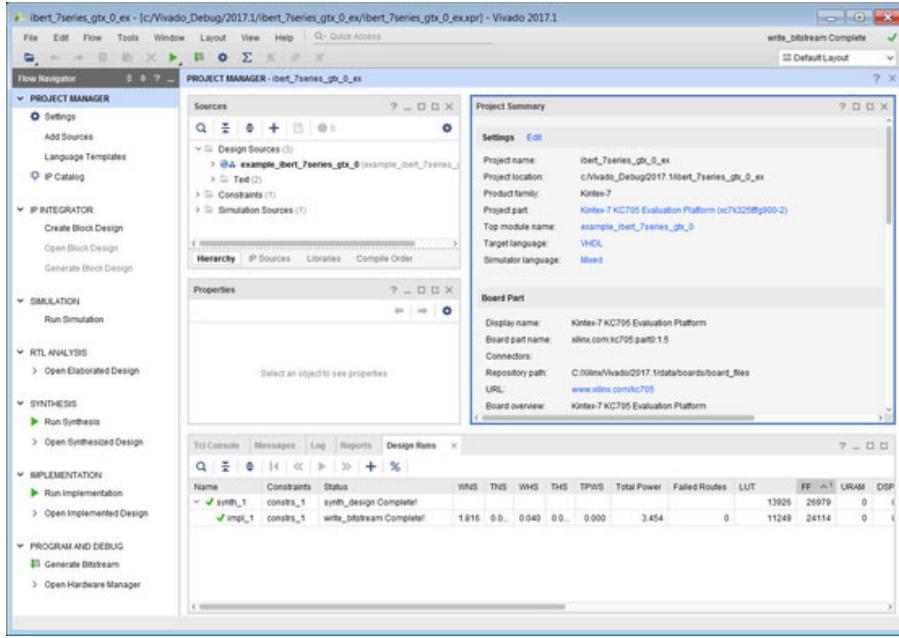
```

launch_runs impl_1 -to_step write_bitstream
wait_on_run impl_1

```

Refer to the *Vivado Design Suite User Guide: Design Flows Overview* ([UG892](#)) for more details on the various ways you can implement your design.

Figure: IBERT Example Design



In-System IBERT System Serial I/O Design Debugging Flows

!! Important: The In-System IBERT core is only available for the UltraScale and UltraScale+ device families and is not supported on Versal device families as In-System IBERT functionality has been integrated into Versal IBERT.

The In-System IBERT IP lets you perform 2-D eye scans of UltraScale and UltraScale+ transceivers using the Vivado Serial IO Analyzer tool. This IP uses data from the design to plot the eye scan of the transceivers in real time while they interact with the rest of the system. This IP can be integrated with user logic in the design or AMD transceiver based IPs (for example GT Wizard or Aurora).

The In System Serial I/O Debugging flow has three distinct phases:

1. In-System IBERT Core Generation phase: Customizing and generating the In-System IBERT core that best meets your hardware high-speed serial I/O requirements.
2. Integration phase: Instantiating the IP and integrating it into your design.
3. Serial I/O Analysis phase: Interacting with the In-System IBERT IP contained in the design to debug and verify issues in your high-speed serial I/O links.

Details about the In-System IBERT Core Generation phase and the Integration phase are covered the remainder of this section. For details of the Serial I/O Analysis phase, see Debugging the Serial I/O Design in Hardware.

Related Information

Debugging the Serial I/O Design in Hardware

Generating an In-System IBERT Core Using the Vivado IP Catalog

The first phase of debugging your design's high-speed serial I/O interfaces is to generate the In-System IBERT core.

To do this, follow these steps:

1. Open the Vivado IDE.
2. On the first panel, choose Manage IP > New IP Location, click Next when the Open IP Catalog wizard opens.
3. Select the desired part, target language, target simulator, and IP location. Click Finish.
4. In the IP Catalog under Debug and Verification > Debug , find one or more available In-System IBERT cores depending on the device selected in the previous step.
5. Double-click the desired In-System IBERT architecture to open the Customize IP Wizard for that core.

Customize the In-System IBERT core for your given hardware system requirements. For details on the In-System IBERT cores, see the *In-System IBERT LogiCORE IP Product Guide* ([PG246](#)).

Instantiating the IP and integrating In-System IBERT IP in the User Design

After generating the In-System IBERT IP core do the following:

1. Open your top level RTL file to edit and add the In-System IBERT core generated in the previous step.
2. Copy the instantiation template of the In-System IBERT core generated by the tool and instantiate it in the RTL file.
3. Connect the ports of your transceiver to the In-System IBERT IP.

For a detailed example of how to integrate In-System IBERT into the user design, see Chapter 5 "Example Designs" of the following IP Document: *In-System IBERT LogiCORE IP Product Guide* ([PG246](#)).

★ Tip: Ensure that you have read the FAQ section of the *In-System IBERT LogiCORE IP Product Guide* ([PG246](#)), which lists some recommendations for issues you could encounter while integrating this IP into your design.

4. Synthesize and implement the design.

Versal Serial I/O Hardware Debugging Flows

AMD Versal™ adaptive SoCs no longer require the generation of IBERT IP as the necessary logic required to use IBERT is now integrated into the GTY transceiver architecture. Any design that uses the GTY transceivers can be used for serial I/O hardware debugging. The Versal serial I/O hardware debugging flow has two distinct phases:

1. Design creation. Customizing and generating a design that uses the GTY transceivers of the device, typically using the Versal adaptive SoC transceivers wizard or the Versal IBERT Configurable Example Design in AMD Vivado™ .
2. Serial I/O analysis. Interacting with the GTY transceivers in the design using the Vivado Hardware Manager to debug and verify issues in your high-speed serial I/O links.

Note: Versal IBERT can use internal pattern generators such as PRBS and user data from the design. For this reason, the In-System IBERT core is not supported for Versal devices. To gain functionality similar to In-System IBERT, change the pattern to user data.

Note: Versal IBERT does not currently support rate change. Furthermore, it is not recommended to drive signals such as the PIN_EN pins on the Transceivers Wizard outside the Vivado serial I/O analyzer because it might lead to unpredictable results. The unpredictable results are because when IBERT is used, it takes control of the transceiver and can modify transceiver settings.

For more information, see the *Versal Adaptive SoC Transceivers Wizard LogiCORE IP Product Guide* ([PG331](#)).

Debugging the Serial I/O Design in Hardware

Once you have IBERT core implemented, you can use the runtime serial I/O analyzer features to debug the design in hardware. Only IBERT cores version v3.0 and later can be accessed using the serial I/O analyzer feature.

Using Vivado Serial I/O Analyzer to Debug the Design

The AMD Vivado™ serial I/O analyzer feature is used to interact with IBERT debug IP cores that are in your design. To access the Vivado serial I/O analyzer feature, click the Open Hardware Manager button in the Program and Debug section of the Flow Navigator.

The steps to debug your design in hardware are:

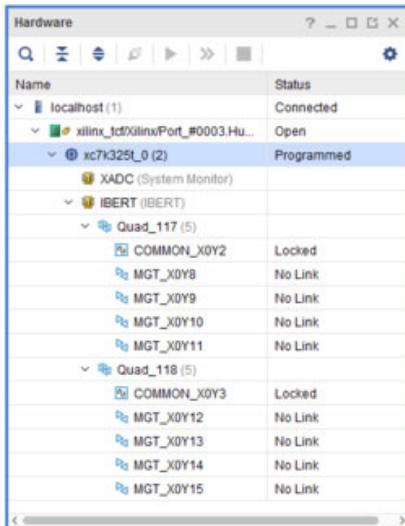
1. Connect to the hardware target and programming the FPGA with the bit file.
2. Create Links.
3. Modify link settings and examine status.
4. Run scans as needed.

Connecting to the Hardware Target and Programming the Device

Programming an FPGA or adaptive SoC prior to debugging involves exactly the same steps described in Programming the FPGA or adaptive SoC. After programming the device with the .pdi file that contains the IBERT core, the Hardware window now shows the components of the IBERT core with the RTL instance name shown in parenthesis, which were detected when scanning the device (see the following figure).

!! Important: In designs using the In-System IBERT IP for UltraScale and UltraScale+ designs, you see the In-System IBERT core being detected in the Hardware window.

Figure: Hardware Window Showing the IBERT Core



Related Information

[Programming the Device](#)

Creating Links and Link Groups

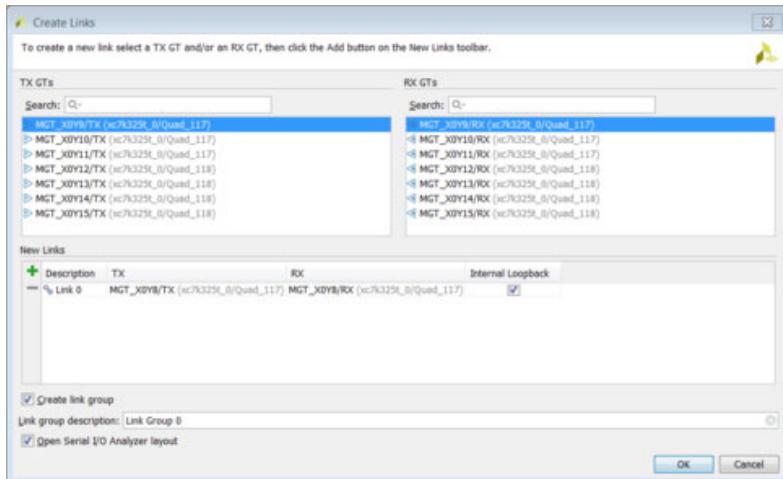
The IBERT core present in the design appears in the Hardware window under the target device. If you do not see the core appear, right-click the device and select the Refresh Hardware command. This re-scans the FPGA and refreshes the Hardware window.

Note: If you still do not see the IBERT core after programming and/or refreshing the FPGA device, check to make sure the device was programmed with the appropriate .bit file. Also check to make sure the implemented design contains an IBERT v3.0 core.

The Vivado serial I/O analyzer feature is built around the concept of links. A link is analogous to a channel on a board, with a transmitter and a receiver. The transmitter and receiver can or cannot be the same GT, on the same device, or be the same architecture. Because a link must be associated with both a transmitter and receiver, connecting an external pattern generator to a single GT receiver is not supported. To create one or more links, go to the Links tab in Vivado, and click either the Create Links button, or right-click and choose Create Links. This causes the Create Links dialog window to appear, as shown in the following figure.

When an IBERT core is detected, the Hardware Manager notes that there are no links present and shows a green banner at the top. Click Create Links to open the dialog box, as shown in the following figure.

Figure: Create Links Dialog Box



Choose a TX and/or an RX from the list available. Or type in a string into the search field to narrow down the list. Then click the Add + button to add the link to the list. Repeat for all links desired.

!! Important: A given TX or RX endpoint can only belong to one link.

Links can also be a part of a link group. By default, all new links are grouped together. You can choose not to add the links to a group by deselecting Create link group. The name of the link group is specified in the Link group description field.

Viewing and Changing Links Settings Using the Links Window

Once links are created, they are added to the Links view (see the following figure) which is the primary and best way to change link settings and view status.

Figure: Links Window

Serial I/O Links									
Name	TX	RX	Status	Bits	Errors	BER	BERT Reset	TX Pattern	R
Ungrouped Links (0)									
Link Group SMA (1)									
Link 0	MGT_XOY8/TX	MGT_XOY8/RX	7.988 Gbps	1.343E12	2.645E11	1.969E-1	Reset	PRBS 7-bit	P
Link Group Internal...									
Link 1	MGT_XOY9/TX	MGT_XOY9/RX	7.987 Gbps	3.805E12	2.079E12	5.465E-1	Reset	PRBS 7-bit	P
Link 2	MGT_XOY10/TX	MGT_XOY10/RX	7.988 Gbps	3.805E12	2.175E12	5.715E-1	Reset	PRBS 7-bit	P

Each row in the Links window represents a link. Common and useful status and controls are enabled by default, so the health of the links can be quickly seen. The various settings that can be viewed in the Links window's table columns are shown in the following table.

Table: Links Window Settings

Link View Column Name	Description
Name	The name of the link
TX	The GT location of the transmitter
RX	The GT location of the receiver

Link View Column Name	Description
Status	If linked (meaning the incoming RX data as expected). Status displays the measured line rate. Otherwise, it displays "No Link".
Bits	The measured number of bits received.
Errors	The measured number of bit errors by the receiver.
BER	Bit Error Ratio = (1 + Errors) / (Bits).
BERT Reset	Resets the received bits and error counters.
RX Pattern	Selects which pattern the receiver is expecting.
TX Pattern	Selects which pattern the transmitter is sending.
TX Pre-Cursor	Selects the pre-cursor emphasis on the transmitter.
TX Post-Cursor	Selects the post-cursor emphasis on the transmitter.
TX Diff Swing	Selects the differential swing values for the transmitter.
DFE Enabled	Selects whether the Decision Feedback Equalizer is enabled on the receiver (not available for all architectures).
Inject Error	Injects a single bit error into the transmit path.
TX Reset	Resets the transmitter.
RX Reset	Resets the receiver and BERT counters (see BERT Reset).
Loopback Mode	Selects the loopback mode on the receiver GT. Warning: Changing this value might affect the link status depending on the system topology.
Termination Voltage	Selects the termination voltage of the receiver.
RX Common Mode	Selects the RX Common Mode setting of the receiver.
TXUSERCLK Freq	Shows the measured TXUSERCLK frequency in MHz.
TXUSERCLK2 Freq	Shows the measured TXUSERCLK2 frequency in MHz.
RXUSERCLK Freq	Shows the measured RXUSERCLK frequency in MHz.
RXUSERCLK2 Freq	Shows the measured RXUSERCLK2 frequency in MHz.
TX Polarity Invert	Inverts the polarity of the transmitted data.
RX Polarity Invert	Inverts the polarity of the received data.

It is possible to change the values of a given property for all links in a link group by changing the setting in the link group row. For instance, changing the TX Pattern to "PRBS 7-bit" in the "Link

Group 0" row changes the TX Pattern of all the links to "PRBS 7-bit". If not all the links in the group have the same setting, "Multiple" appears for that column in the link group row.

When the In System IBERT IP is used in your design, only a subset of the link settings apply. The following table lists the applicable link settings.

Table: In System IBERT Link Window Settings

Link view column name	Description
TX	The GT location of the transmitter
RX	The GT location of the receiver
TX Pre-Curser	Selects the pre-curser emphasis on the transmitter.
TX Post-Cursor	Selects the post-cursor emphasis on the transmitter.
TX Diff Swing	Selects the differential swing values for the transmitter.
DFE Enabled	Selects whether the Decision Feedback Equalizer is enabled on the receiver (not available for all architectures).

Creating and Running Link Scans

To analyze the margin of a given link, it is often helpful to run a scan of the link using the specialized Eye Scan hardware of the AMD 7 series FPGA transceivers. The Vivado serial I/O analyzer feature enables you to define, run, save, and recall link scans.

A scan runs on a link. To create a scan, select a link in the Link window, and either right-click and choose Create Scan, or click the Create Scan button in the Link window toolbar. This brings up the Create Scan dialog (see the following figure). The Create Scan dialog shows the settings for performing a scan, as shown in the following table.

There are two types of scans that can be generated, the 2D Eyescan or the 1D Bathtub Plot. Both these scans use the same settings specified in the Create Scan dialog as follows. The Scan type field in the following dialog determines the type of scan generated.

Figure: Create Scan Dialog

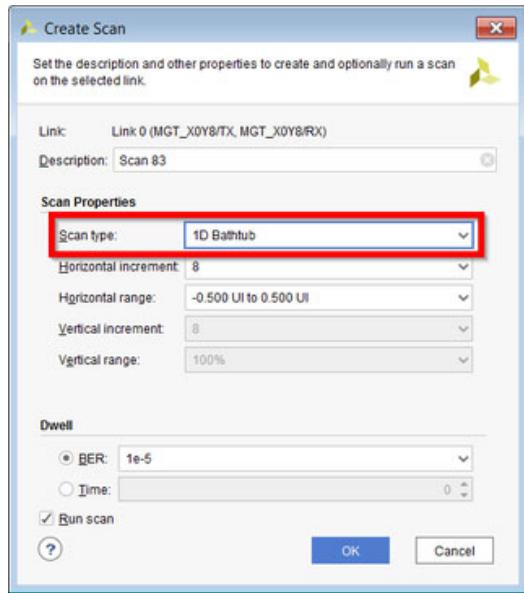


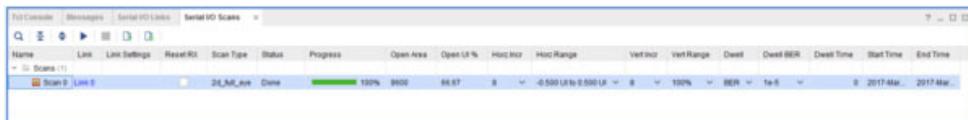
Table: Scan Settings

Scan Setting	Description
Description	A user-defined name for the scan.
Scan Type	The type of scan to run. This can be a 2D Eyescan or a 1D Bathtub plot.
Horizontal Increment	Allows you to choose to scan the eye at a reduced resolution, but at increased speed by skipping horizontal codes.
Horizontal Range	Reducing the horizontal range increases the scan speed. By default, the entire eye is scanned (-1/2 of a unit interval to +1/2 in reference to the center of the eye).
Vertical Increment	Allows you to choose to scan the eye at a reduced resolution, but increased speed by skipping vertical codes.
Vertical Range	Reducing the vertical range increases the scan speed. By default, the entire eye is scanned.
Dwell BER	Each point in the chart is scanned for a certain amount of time. Dwell BER allows you to choose the scan depth by selecting the desired Bit Error Ratio.
Dwell Time	Dwell Time allows you to choose the scan depth by inputting the desired time in seconds. The Dwell time setting is not supported on designs that use In System IBERT IP.

By default, the scan is run after it is created. If you do not want to run the scan, and only define it, uncheck the Run Scan check box.

If a scan is created, but not run, it can be subsequently run or run by right-clicking on a scan in the Scans window and choosing Run Scan (see the following figure). While a scan is running, it can be prematurely stopped by right-clicking on a scan and choosing Stop Scan or clicking the Stop Scan button in the Scans window toolbar.

Figure: Scans Window



Creating and Running Link Sweeps

To analyze the margin of a given link, it can be helpful to run multiple scans of the link with different MGT settings. This helps determine which settings are the best. The Vivado serial I/O analyzer feature enables you to define, run, save, and recall link sweeps, which are a collection of link scans. A sweep runs on a link. To create a sweep, select a link in the Link window, and either right-click and choose Create Sweep, or click the Create Sweep button in the Link window toolbar. This brings up the Create Sweep dialog box, which looks similar to the Create Scan dialog box, except that it has additional options for defining which properties to sweep, and how.

Figure: Create Sweep Dialog Box

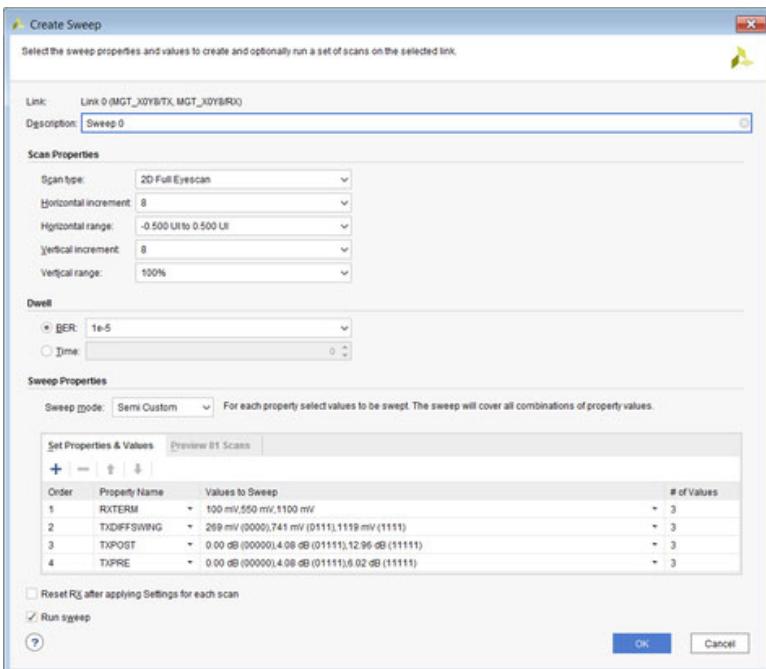


Table: Sweep Settings

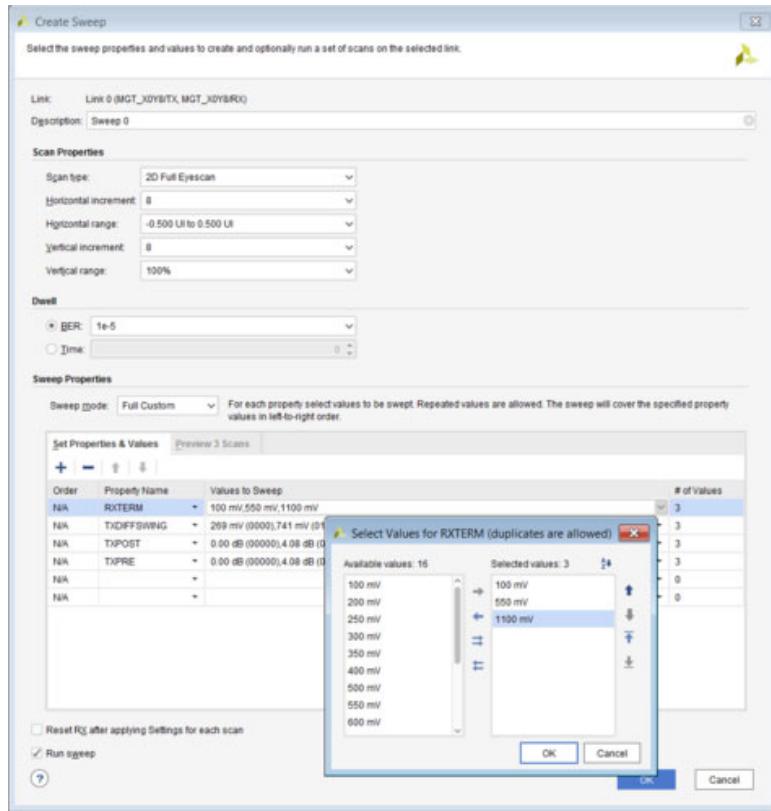
Sweep Setting	Description
---------------	-------------

Sweep Setting	Description
Description	A user-defined name for the sweep.
Scan Type	The type of scan to run. This can be a 2D Eyescan or a 1D Bathtub plot.
Horizontal Increment	Allows you to scan the eye at a reduced resolution, but at increased speed by skipping horizontal codes.
Horizontal Range	Reducing the horizontal range increases the scan speed. By default, the entire eye is scanned (-1/2 of a unit interval to +1/2 in reference to the center of the eye).
Vertical Increment	Allows the user to choose to scan the eye at a reduced resolution, but increased speed by skipping vertical codes.
Vertical Range	Reducing the vertical range increases the scan speed. By default, the entire eye is scanned.
Dwell BER	Each point in the chart is scanned for a certain amount of time. Dwell BER allows you to choose the scan depth by selecting the desired Bit Error Ratio (BER).
Dwell Time	Dwell Time allows you to choose the scan depth by inputting the desired time in seconds.
Sweep Mode	The type of sweep to run. The choices are Semi Custom, Full Custom, and Exhaustive.

After these settings are chosen, the next step is to choose the Sweep Properties. Any writable properties of the link can be swept. To add a property, click the "+" button on the left to add another row to the table. Click the Property Name to choose a property to sweep.

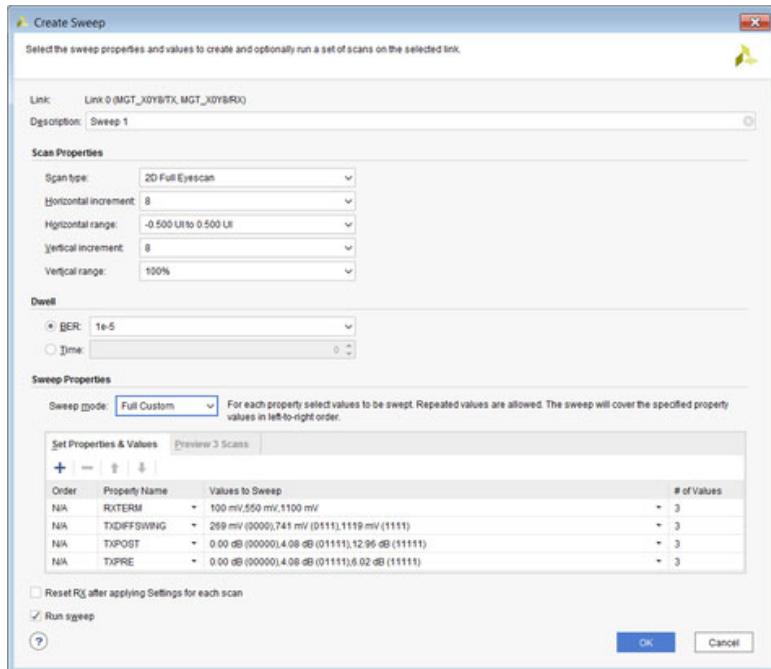
To change the values, click the Values to Sweep Cell, and use the chooser to select which values to sweep. If the property does not have enumerated values, type one hex value on each line of the text area provided.

Figure: Values to Sweep Cell



- In the Semi Custom case shown in the following figure, every combination of the properties chosen is defined for a single scan, and that scan is performed according to the sweep properties. The number of sweeps that are performed, and in what order can be previewed by selecting the Preview & Scans tab.
- In the Full Custom case, the first choice for each of the properties listed is used for the first scan, the second choice for each of the properties is used for the second scan, etc. If one of the properties has fewer choices than other properties, the last choice is used for all subsequent scans. With the same properties choices but Full Custom as the sweep Mode, only three scans would be performed.

Figure: Sweep Properties Dialog Box



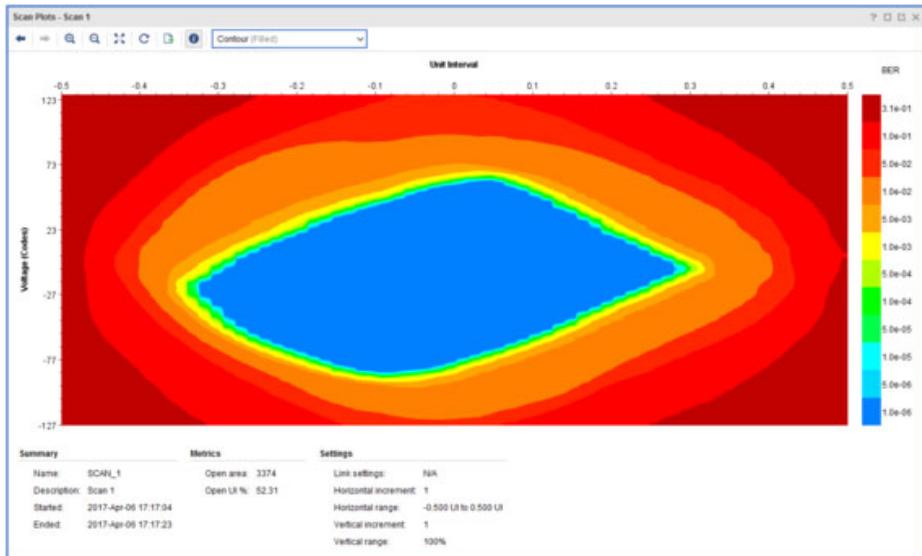
- In the Exhaustive case, the Values to Sweep is no longer editable, as all values are chosen for a given property.

When all the properties are set, to run each of the scans sequentially, keep Run Sweep checked. The list of scans is elaborated in the Scan window once you click OK. During the sweep, the progress is tracked in the Scan window, and the latest Scan result is displayed.

Displaying and Navigating the Scan Plots

After a scan is created, it automatically launches the Scan Plots window for that scan. For 2D Eyescan, the plot is a heat map of the BER value.

Figure: Scan Plot Window



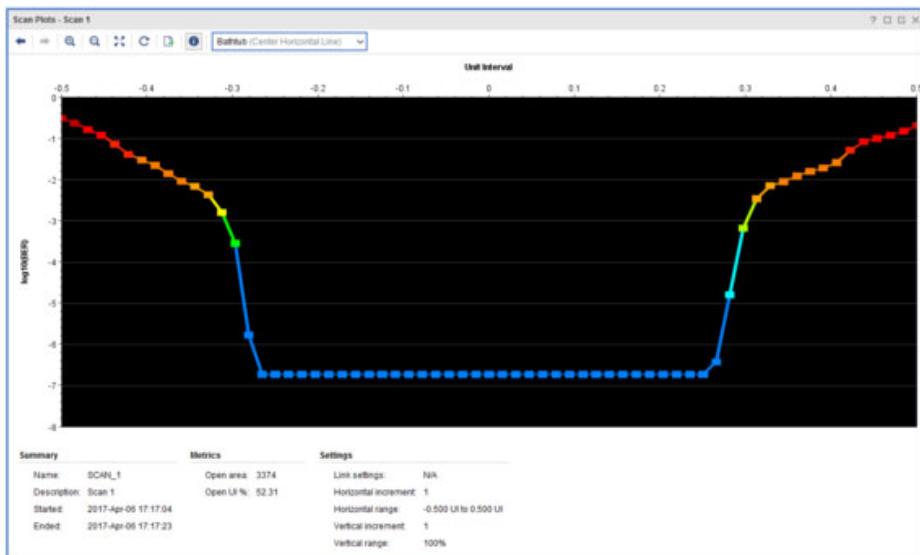
As in other charts and displays within the Vivado IDE, the mouse gestures for zooming in the eye scan plot window are as follows:

- Zoom Area: left-click drag from top-left to bottom-right
- Zoom Fit: left-click drag from bottom-right to top-left
- Zoom In: left-click drag from top-right to bottom-left
- Zoom Out: left-click drag from bottom-left to top-right

Also, when the mouse cursor is over the Plot, the current horizontal and vertical codes, along with the scanned BER value is displayed in the tooltip. You can also change the plot type by clicking the Plot Type button in the plot window, and choosing Show Contour (filled), Show Contour (lines), Bathtub (Center Horizontal Line), and Heat Map.

A summary view is present at the bottom of the scan plot, stating the scan settings, along with basic information like when the scan was performed. During the 2D Eyescan, the number of pixels in the scan with zero errors is calculated (taking into account the horizontal and vertical increments), and this result is displayed as Open Area. The Scan window contents are sorted by Open Area by default, so the scans with the largest open area appear at the top. The following figure is a Bathtub plot for the same scan as shown in the previous figure.

Figure: Bathtub Plot



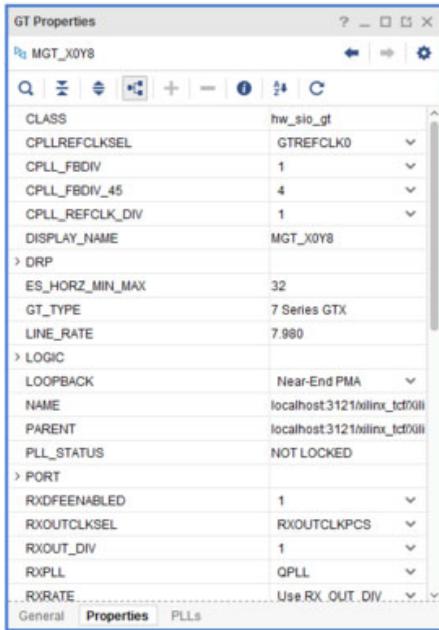
Writing the Scan Results to a File

When scan data exists due to a partial or full 2D Eyescan, these results can be written to a CSV file by clicking the Write Scan button in the Scans window. This saves the scan results to comma-delimited file, with the BER values in a block that replicated the scan plot.

Properties Window

Whenever a GT or a COMMON block in the hardware window, a Link in the Links window, or a scan in the Scans window is selected, the properties of that object shows in the Properties window. For GTs and COMMONs, these include all the attributes, ports, and other settings of those objects. These settings can be changed in the Properties window (see the following figure), or by writing Tcl commands to change and commit the properties. Some properties are read-only and cannot be changed.

Figure: Properties Window



Description of Serial I/O Analyzer Tcl Objects and Commands

You can use Tcl commands to interact with your hardware under test. The hardware is organized in a set of hierarchical first class Tcl objects (see the following table).

Table: Serial I/O Analyzer Tcl Objects

Tcl Object	Description
hw_sio_ibert	Object referring to an IBERT core. Each IBERT object can have one or more hw_sio_gt, or hw_sio_common objects associated with it.
hw_sio_gt	Object referring to a single AMD Gigabit Transceiver (GT).
hw_sio_gtgroups	Object referring to a logical grouping of GTs, could be a Quad or an Octal.
hw_sio_common	Object referring to a COMMON block.
hw_sio_tx	Object referring to the transmitter side of a hw_sio_gt. Only the TX related ports, attributes, and logic properties flows to the hw_sio_tx.
hw_sio_rx	Object referring to the receiver side of a hw_sio_gt. Only the RX related ports, attributes, and logic properties flows to the hw_sio_rx.
hw_sio_pll	Object referring to a PLL in either an hw_sio_gt or an hw_sio_common object. Only the related ports, attributes, and logic properties flow to the hw_sio_pll.
hw_sio_link	Object referring to a link, a TX-RX pair. A link can also consist of a TX only or an RX only.
hw_sio_linkgroup	Object referring to a group of links.

Tcl Object	Description
hw_sio_scan	Object referring to a margin analysis scan.

For more information about the Hardware Manager commands, run the `help -category hardware` Tcl command in the Tcl Console.

Description of Tcl Commands to Access Hardware

The following table contains descriptions of all Tcl commands used to interact with the IBERT core.

!! Important: Using the `get_property` or `set_property` command does not read or write information to/from the IBERT core. You must use the `refresh_hw_sio` and `commit_hw_sio` commands to read and write information from/to the hardware, respectively.

Table: Descriptions of hw_server Tcl Commands

Tcl Command	Description
refresh_hw_sio	Read the property values out of the provided object. Works for any <code>hw_sio</code> object that refers to hardware.
commit_hw_sio	Writes property changes to the hardware. Works for any <code>hw_sio</code> object that refers to hardware.

Description of hw_sio_link Tcl Commands

The following table contains descriptions of all Tcl commands used to interact with links.

Table: Descriptions of hw_sio_link Tcl Commands

Tcl Command	Description
create_hw_sio_link	Create an <code>hw_sio_link</code> object with the given <code>hw_sio_rx</code> and/or <code>hw_sio_tx</code> objects.
remove_hw_sio_link	Deletes the given link.
get_hw_sio_links	Get list of <code>hw_sio_links</code> for the given object.

Description of hw_sio_linkgroup Tcl Commands

The following table contains descriptions of all Tcl commands used to interact with linkgroups.

Table: Descriptions of hw_sio_linkgroup Tcl Commands

Tcl Command	Description
create_hw_sio_linkgroup	Create an <code>hw_sio_linkgroup</code> object with the <code>hw_sio_link</code> objects.

Tcl Command	Description
remove_hw_sio_linkgroup	Deletes the given linkgroup.
get_hw_sio_linkgroups	Get list of hw_sio_linkgroups for the given object.

Description of hw_sio_scan Tcl Commands

The following table contains descriptions of all Tcl commands used to interact with scans.

Table: Descriptions of hw_sio_scan Tcl Commands

Tcl Command	Description
create_hw_sio_scan	Creates a scan object.
remove_hw_sio_scan	Deletes a scan object.
run_hw_sio_scan	Runs a scan.
stop_hw_sio_scan	Stops a scan.
wait_on_hw_sio_scan	Blocks the Tcl console prompt until a given run_hw_sio_scan operation is complete.
display_hw_sio_scan	Displays a partial or complete scan in the Scan Plot.
write_hw_sio_scan	Writes the scan data to a file.
read_hw_sio_scan	Reads scan data from a file into a scan object.
get_hw_sio_scans	Get a list of hw_sio_scan objects.

Description of Tcl Commands to Get Objects

The following table contains descriptions of all Tcl commands used to get serial I/O objects.

Table: Descriptions of Tcl Commands to Get Objects

Tcl Command	Description
get_hw_sio_iberts	Get list of IBERT objects.
get_hw_sio_gts	Get list of GTs.
get_hw_sio_commons	Get list of COMMON blocks.
get_hw_sio_txs	Get list of transmitters.
get_hw_sio_rxes	Get list of receivers.

Tcl Command	Description
get_hw_sio_pll	Get list of PLLs.
get_hw_sio_links	Get list of links.
get_hw_sio_linkgroups	Get list of linkgroups.
get_hw_sio_scans	Get list of scans.

Using Tcl Commands to Take an IBERT Measurement

Here is an example Tcl command script that interacts with the following example system

- One KC705 board's Digilent JTAG-SMT1 cable (serial number 12345) accessible via a hw_server running on localhost:3121
- Single IBERT core in a design running in the XC7K325T device on the KC705 board
- IBERT core has Quad 117 and Quad 118 enabled

Example Tcl Command Script

```
# Connect to the Digilent Cable on localhost:3121
connect_hw_server -url localhost:3121
current_hw_target [get_hw_targets */digilent_plugin/SN:12345]
open_hw_target

# Program and Refresh the XC7K325T Device
current_hw_device [lindex [get_hw_devices] 0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices] 0]
set_property PROGRAM.FILE {C:/design.bit} [lindex [get_hw_devices] 0]
program_hw_devices [lindex [get_hw_devices] 0]
refresh_hw_device [lindex [get_hw_devices] 0]

# Set Up Link on first GT
set tx0 [lindex [get_hw_sio_txs] 0]
set rx0 [lindex [get_hw_sio_rxes] 0]
set link0 [create_hw_sio_link $tx0 $rx0]
set_property DESCRIPTION {Link 0} [get_hw_sio_links $link0]
# Set link to use PCS Loopback, and write to hardware
set_property LOOPBACK "Near-End PCS" $link0
commit_hw_sio $link0

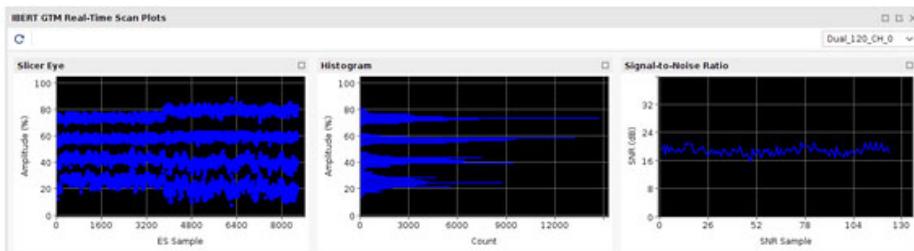
# Create, run, display and save scan
set scan0 [create_hw_sio_scan 2d_full_eye [get_hw_sio_rxes -of $link0]]
run_hw_sio_scan $scan0
display_hw_sio_scan $scan0
write_hw_sio_scan "scan0.csv" $scan0
```

Viewing Slicer Eye, Histogram, and Signal-to-Noise

Ratio (GTM Transceivers Only)

Because the GTM receiver is ADC-based, the conventional eyescan as used in the previous families of transceivers (such as GTH or GTY transceivers) cannot be used. For this reason, the IBERT dashboard for GTM shows three plots: Slicer-Eye, Histogram, and Signal to Noise Ratio (SNR) instead of the traditional Scan Plot Window.

Figure: Slicer Eye, Histogram, and Signal-to-Noise Ratio Plots



Once a link is created, the Slicer-Eye, Histogram, and Signal to Noise Ratio (SNR) plots are displayed for the link. If you created multiple links, the active link can be changed by selecting the desired DUAL and Channel in the upper-right corner.

For more information on the Slicer-Eye and GTM Transceiver Architecture see the *Virtex UltraScale+ FPGAs GTM Transceivers User Guide* ([UG581](#)).

For more information on IBERT GTM, see the *IBERT for UltraScale GTM Transceivers LogiCORE IP Product Guide* ([PG342](#)).

Device Configuration Bitstream or PDI Settings

7 Series Bitstream Settings

The device configuration settings for 7 series devices available for use with the `set_property <Setting> <Value> [current_design]` AMD Vivado™ tool Tcl command are shown in the following table.

Note: Bitstream settings for BPI are not valid for Spartan™ 7 devices.

Table: 7 Series Bitstream Settings

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1, 2, 3, 4	Helps synchronize BPI configuration with the timing of page mode operations in flash devices. It allows you to set the cycle number for a valid read of the first page. The

Setting	Default Value	Possible Values	Description
			BPI_page_size must be set to 4 or 8 for this option to be available.
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1, 4, 8	For BPI configuration, this option lets you specify the page size which corresponds to the number of reads required per page of flash memory.
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable, Type1, Type2	<p>Sets the BPI synchronous configuration mode for different types of BPI flash devices.</p> <p>Disable (the default) disables the synchronous configuration mode.</p> <p>Type1 enables the synchronous configuration mode and settings to support the Micron G18(F) family.</p> <p>Type2 enables the synchronous configuration mode and settings to support the Micron (Numonyx) P30 and P33 families.</p>
BITSTREAM.CONFIG.CCLKPIN	Pullup	Pullup, Pullnone	Adds an internal pull-up to the Cclk pin. The Pullnone setting disables the pullup.
BITSTREAM.CONFIG.CONFIGFallback	Disable	Disable, Enable	<p>Enables or disables the loading of a default bitstream when a configuration attempt fails.</p> <p>If the MultiBoot solution setting BITSTREAM.CONFIG.NEXT_CONFIG_ADDR is used, the BITSTREAM.CONFIG.FALLBACK setting is enabled.</p> <p>Fallback MultiBoot is not supported in Virtex 7 HT devices.</p>
BITSTREAM.CONFIG.CONFIGRATE	3	3, 6, 9, 12, 16, 22, 26, 33, 40, 50, 66	Uses an internal oscillator to generate the configuration clock, Cclk, when configuring in a master mode. Use this option to select the rate for Cclk.
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired, Continuous, Quiet	Controls how often the Digitally Controlled Impedance circuit attempts to update the impedance match for DCI IOSTANDARDS.

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.DONEPIN ¹	Pullup	Pullup, Pullnone	Adds an internal pull-up to the DONE pin. The Pullnone setting disables the pullup. Use DonePin only if you intend to connect an external pull-up resistor to this pin. The internal pull-up resistor is automatically connected if you do not use DonePin.
BITSTREAM.CONFIG.EXTMASTERCCLK_EN	Disable	Disable, Div-1, Div-2, Div-4, Div-8	Allows an external clock to be used as the configuration clock for all master modes. The external clock must be connected to the dual-purpose EMCCLK pin.
BITSTREAM.CONFIG.INITPIN ¹	Pullup	Pullup, Pullnone	Specifies whether you want to add a Pullup resistor to the INIT pin, or leave the INIT pin floating.
BITSTREAM.CONFIG.INITSIGNALSErrorR	Enable	Enable, Disable	When Enabled, the INIT_B pin asserts to '0' when a configuration error is detected.
BITSTREAM.CONFIG.M0PIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M0 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M0 pin.
BITSTREAM.CONFIG.M1PIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M1 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M1 pin.
BITSTREAM.CONFIG.M2PIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M2 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M2 pin.
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	None	<string>	Sets the starting address for the next configuration in a MultiBoot set up, which is stored in the WBSTAR register.
BITSTREAM.CONFIG.NEXT_CONFIG_REBOOT	Enable	Enable, Disable	When set to Disable the IPROG command is removed from the .bit file. This allows the Golden image to load upon power up rather than jumping to the multiboot image in a multiboot setup.

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.PERSIST	No	No, Yes	Maintains the configuration logic access to the multi-function configuration pins after configuration. Primarily used to maintain the SelectMAP port after configuration for readback access, but can be used with any configuration mode. Persist is not needed for JTAG configuration because the JTAG port is dedicated and always available. PERSIST and ICAP cannot be used at the same time. Refer to the user guide for a description. Persist is needed for Readback and Partial Reconfiguration using the SelectMAP configuration pins, and should be used when either SelectMAP or Serial modes are used.
BITSTREAM.CONFIG.REVISIONSELECT	00	00, 01, 10, 11	Specifies the internal value of the RS[1:0] settings in the Warm Boot Start Address (WBSTAR) register for the next warm boot.
BITSTREAM.CONFIG.REVISIONSELECT_TRISTATE	Disable	Disable, Enable	Specifies whether the RS[1:0] 3-state is enabled by setting the option in the Warm Boot Start Address (WBSTAR). 0: Enable RS 3-state 1: Disable RS 3-state
BITSTREAM.CONFIG.SELECTMAPABORT	Enable	Enable, Disable	Enables or disables the SelectMAP mode Abort sequence. If disabled, an Abort sequence on the device pins is ignored.
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No, Yes	Enables SPI 32-bit address style, which is required for SPI devices with storage of 256 Mb and larger.
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE, 1, 2, 4	Sets the SPI bus to Dual (x2) or Quad (x4) mode for Master SPI configuration from third party SPI flash devices.
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No, Yes	Sets the FPGA to use a falling edge clock for SPI data capture. This improves timing margins and can allow faster clock rates for configuration.
BITSTREAM.CONFIG.TCKPIN ¹	Pullup	Pullup, Pulldown,	Adds a pull-up, a pull-down, or neither to the TCK pin, the JTAG test clock. The Pullnone

Setting	Default Value	Possible Values	Description
		Pullnone	setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.TDIPIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to the TDI pin, the serial data input to all JTAG instructions and JTAG registers. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.TDOPIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to the TDO pin, the serial data output for all JTAG instruction and data registers. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.TIMER_CFG	None	<8-digit hex string>	Enables the Watchdog Timer in Configuration mode and sets the value. This option cannot be used at the same time as TIMER_USR.
BITSTREAM.CONFIG.TIMER_USR	0x00000000	<8-digit hex string>	Enables the Watchdog Timer in Configuration mode and sets the value. This option cannot be used at the same time as TIMER_CFG.
BITSTREAM.CONFIG.TMSPIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, pull-down, or neither to the TMS pin, the mode input signal to the TAP controller. The TAP controller provides the control logic for JTAG. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pulldown, Pullup, Pullnone	Adds a pull-up, a pull-down, or neither to unused SelectIO™ pins (IOBs). It has no effect on dedicated configuration pins. The list of dedicated configuration pins varies depending upon the architecture. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	<8-digit hex string>	Used to identify implementation revisions. You can enter up to an 8-digit hexadecimal string in the User ID register.

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.USR_ACCESS	None	None, <8-digit hex string>, TIMESTAMP	Writes an 8-digit hexadecimal string, or a timestamp into the AXSS configuration register. The format of the timestamp value is ddddd MMMM yyyy hhmmss : day, month, year (year 2000 = 00000), hour, minute, seconds. The contents of this register can be directly accessed by the FPGA fabric via the USR_ACCESS primitive.
BITSTREAM.ENCRYPTION No ENCRYPT	No	Yes	Encrypts the bitstream.
BITSTREAM.ENCRYPTION bbram ENCRYPTKEYSELECT	bbram, efuse		Determines the location of the AES encryption key to be used, either from the battery-backed RAM (BBRAM) or the eFUSE register. This property is only available when the Encrypt option is set to True.
BITSTREAM.ENCRYPTION HKEY	<hex string>		HKey sets the HMAC authentication key for bitstream encryption. 7 series devices have an on-chip bitstream-keyed Hash Message Authentication Code (HMAC) algorithm implemented in hardware to provide additional security beyond AES decryption alone. These devices require both AES and HMAC keys to load, modify, intercept, or clone the bitstream. To use this option, you must first set Encrypt to Yes.
BITSTREAM.ENCRYPTION Key0	<hex string>		Key0 sets the AES encryption key for bitstream encryption. To use this option, you must first set Encrypt to Yes.
BITSTREAM.ENCRYPTION KEYFILE	<string>		Specifies the name of the input encryption file (with a .nky file extension). To use this option, you must first set Encrypt to Yes.
BITSTREAM.ENCRYPTION STARTCBC	<32-bit hex string>		Sets the starting cipher block chaining (CBC) value.

Setting	Default Value	Possible Values	Description
BITSTREAM.GENERAL.COMPRESS	False	True, False	Uses the multiple frame write feature in the bitstream to reduce the size of the bitstream, not the Bitstream (.bit) file. Using Compress does not guarantee that the size of the bitstream shrinks.
BITSTREAM.GENERAL.CRC	Enable	Enable, Disable	Controls the generation of a Cyclic Redundancy Check (CRC) value in the bitstream. When enabled, a unique CRC value is calculated based on bitstream contents. If the calculated CRC value does not match the CRC value in the bitstream, the device fails to configure. When CRC is disabled a constant value is inserted in the bitstream in place of the CRC, and the device does not calculate a CRC. The CRC default value is Enable, except when BITSTREAM.ENCRYPTION.ENCRYPT is Yes, the CRC is disabled.
BITSTREAM.GENERAL.DEBUGBITSTREAM	No	No, Yes	Lets you create a debug bitstream. A debug bitstream is significantly larger than a standard bitstream. DebugBitstream can be used only for master and slave serial configurations. DebugBitstream is not valid for Boundary Scan or Slave Parallel>SelectMAP. In addition to a standard bitstream, a debug bitstream offers the following features: Writes 32 0s to the LOUT register after the synchronization word. Loads each frame individually. Performs a Cyclic Redundancy Check (CRC) after each frame. Writes the frame address to the LOUT register after each frame.
BITSTREAM.GENERAL.DISABLE_JTAG	No	No, Yes	Disables communication to the Boundary Scan (BSCAN) block via JTAG after configuration.
BITSTREAM.GENERAL.JTAG_XADC	Enable	Enable, Disable,	Enables or disables the JTAG connection to the XADC.

Setting	Default Value	Possible Values	Description
		StatusOnly	
BITSTREAM.GENERAL. PERFRAMECRC	No	No, Yes	Inserts CRC values at regular intervals within bitstreams. These values validate the integrity of the incoming bitstream and can flag an error (shown on the INIT_B pin and the PRERROR port of the ICAP) prior to loading the configuration data into the device. While most appropriate for partial bitstreams, when set to Yes, this property inserts the CRC values into all bitstreams, including full device bitstreams.
BITSTREAM.GENERAL. XADCENHANCEDLINEARITY	Off	Off, On	Disables some built-in digital calibration features that make INL look worse than the actual analog performance.
BITSTREAM.READBACK. ACTIVERECONFIG	No	No, Yes	Prevents the assertions of GHIGH and GSR during configuration. This is required for the active partial reconfiguration enhancement features.
BITSTREAM.READBACK. ICAP_SELECT	Auto	Auto, Top, Bottom	Selects between the top and bottom ICAP ports.
BITSTREAM.READBACK. READBACK	False	True, False	Lets you perform the Readback function by creating the necessary readback files.
BITSTREAM.READBACK. SECURITY	None	None, Level1, Level2	Specifies whether to disable Readback and Reconfiguration. Specifying Security Level1 disables Readback. Specifying Security Level2 disables Readback and Reconfiguration.
BITSTREAM.READBACK. XADCPARTIALRECONFIG	Disable	Disable, Enable	When Disabled XADC can work continuously during Partial Reconfiguration. When Enabled XADC works in Safe mode during partial reconfiguration.
BITSTREAM.STARTUP. DONEPIPE	Yes	Yes, No	Tells the FPGA to wait on the CFG_DONE (DONE) pin to go High and wait for the first clock edge before moving to the Done state.
BITSTREAM.STARTUP. DONE_CYCLE	4	4, 1, 2, 3, 5, 6,	Selects the Startup phase that activates the FPGA Done signal. Done is delayed when

Setting	Default Value	Possible Values	Description
		Keep	DonePipe=Yes.
BITSTREAM.STARTUP.GTS_CYCLE	5	5, 1, 2, 3, 4, 6, Done, Keep	Selects the Startup phase that releases the internal 3-state control to the I/O buffers.
BITSTREAM.STARTUP.GWE_CYCLE	6	6, 1, 2, 3, 4, 5, Done, Keep	Selects the Startup phase that asserts the internal write enable to flip-flops, LUT RAMs, and shift registers. GWE_cycle also enables the BRAMS. Before the Startup phase, both block RAMs writing and reading are disabled.
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait, 0, 1, 2, 3, 4, 5, 6	Selects the Startup phase to wait until DLLs/DCMs/PLLs lock. If you select NoWait, the Startup sequence does not wait for DLLs/DCMs/PLLs to lock.
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto, NoWait, 0, 1, 2, 3, 4, 5, 6	Specifies a stall in the Startup cycle until digitally controlled impedance (DCI) match signals are asserted. DCI matching does not begin on the Match_cycle. The Startup sequence waits in this cycle until DCI has matched. Given that there are a number of variables in determining how long it takes DCI to match, the number of CCLK cycles required to complete the Startup sequence can vary in any given system. Ideally, the configuration solution should continue driving CCLK until DONE goes high. When the Auto setting is specified, <code>write_bitstream</code> searches the design for any DCI I/O standards. If DCI standards exist, <code>write_bitstream</code> uses <code>BITSTREAM.STARTUP.MATCH_CYCLE=2</code> . Otherwise, <code>write_bitstream</code> uses <code>BITSTREAM.STARTUP.MATCH_CYCLE=NoWait</code> .
BITSTREAM.STARTUP.STARTUPCLK	Cclk	Cclk, UserClk, JtagClk	The StartupClk sequence following the configuration of a device can be synchronized to either Cclk, a User Clock, or the JTAG Clock. The default is Cclk. Cclk lets you synchronize to an internal clock provided in the FPGA device.

Setting	Default Value	Possible Values	Description
			<p>UserClk lets you synchronize to a user-defined signal connected to the CLK pin of the STARTUP symbol.</p> <p>JtagClk lets you synchronize to the clock provided by JTAG. This clock sequences the TAP controller which provides the control logic for JTAG.</p> <p>The Spartan7 7s6 / 7s15 devices, do not support the STARTUPE2.CLK (UserClk) user startup clock pin.</p>
1. For the dedicated configuration pins AMD recommends that you use the bitstream setting default.			

Artix, Virtex, and Kintex UltraScale+ Bitstream Settings

The device configuration settings for the AMD Artix™ UltraScale+™ , AMD Virtex™ UltraScale+™ , and AMD Kintex™ UltraScale+™ devices available for use with the `set_property <Setting> <Value> [current_design]` Vivado tool Tcl command are shown in the following table.

Table: Artix UltraScale+, Virtex UltraScale+, and Kintex UltraScale+ Bitstream Settings

Setting	Default Value	Possible Values	Description
BITSTREAM.AUTHENTICATION.AUTHENTICATE	No	No, Yes	Indicates whether or not to use RSA authentication. If No AES_GCM is used.
BITSTREAM.AUTHENTICATION.RSAPRIVATEKEYFILE			Specifies the OpenSSL .pem file that contains the key pairs that should be used to sign the RSA-2048 authenticated bitstream.
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1, 2, 3, 4	Helps synchronize BPI configuration with the timing of page mode operations in flash devices. It allows you to set the cycle number for a valid read of the first page. The BPI_page_size must be set to 4 or 8 for this option to be available.
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1, 4, 8	For BPI configuration, this sub-option lets you specify the page size which

Setting	Default Value	Possible Values	Description
			corresponds to the number of reads required per page of flash memory.
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable, Type1, Type2	<p>Sets the BPI synchronous configuration mode for different types of BPI flash devices.</p> <p>Disable (the default) disables the synchronous configuration mode.</p> <p>Type1 enables the synchronous configuration mode and settings to support the Micron G18(F) family.</p> <p>Type2 enables the synchronous configuration mode and settings to support the Micron (Numonyx) P30 and P33 families.</p>
BITSTREAM.CONFIG.CCLKPIN	Pullup	Pullup, Pullnone	Adds an internal pull-up to the Cclk pin. The Pullnone setting disables the pullup.
BITSTREAM.CONFIG.PERSIST	No	No, Yes	Prohibit usage of the configuration pins as user I/O and persist after configuration.
BITSTREAM.CONFIG.CONFIGRATE	2.7	2.7, 5.3, 8.0, 10.6, 21.3, 31.9, 36.4, 51.0, 56.7, 63.8, 72.9, 85.0, 102.0, 127.5, 170.0	Bitstream generation uses an internal oscillator to generate the configuration clock, Cclk, when configuring is in a master mode. Use this sub-option to select the rate for Cclk.
BITSTREAM.CONFIG.D00_MOSI	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the D00_MOSI pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the D00_MOSI pin.
BITSTREAM.CONFIG.D01_DIN	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the D01_DIN pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the D01_DIN pin.

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.D02	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the D02 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the D02 pin.
BITSTREAM.CONFIG.D03	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the D03 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the D03 pin.
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired, Quiet, Safe	Controls how often the Digitally Controlled Impedance circuit attempts to update the impedance match for DCI IOSTANDARDs.
BITSTREAM.CONFIG.DONEPIN	Pullup	Pullup, Pullnone	Adds an internal pull-up to the DONE pin. The Pullnone setting disables the pullup. Use DonePin only if you intend to connect an external pull-up resistor to this pin. The internal pull-up resistor is automatically connected if you do not use DonePin.
BITSTREAM.CONFIG.EXTMASTERCCLK_EN	Disable	Disable, Div-1, Div-2, Div-3, Div-4, Div-6, Div-8, Div-12, Div-16, Div-24, Div-48	Allows an external clock to be used as the configuration clock for all master modes. The external clock must be connected to the dual-purpose EMCCLK pin.
BITSTREAM.ENCRYPTION FAMILY_KEY_FILEPATH	None	Path to familyKey.cfg	<p>Specifies the install location of the Family Key. No specific directory is required.</p> <p>AMD does not provide the family key as part of the AMD Tool Suite. Customers must send a request for the family key to secure.solutions@xilinx.com. The family key is distributed to qualified customers through the Product Licensing site on https://www.xilinx.com.</p>
BITSTREAM.CONFIG.INITPIN	Pullup	Pullup, Pullnone	Specifies whether you want to add a Pullup resistor to the INIT pin, or leave the INIT pin floating.

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.M0PIN	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M0 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M0 pin.
BITSTREAM.CONFIG.M1PIN	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M1 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M1 pin.
BITSTREAM.CONFIG.M2PIN	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M2 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M2 pin.
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	None	<string>	Sets the starting address for the next configuration in a MultiBoot set up, which is stored in the WBSTAR register.
BITSTREAM.CONFIG.NEXT_CONFIG_REBOOT	Enable	Enable, Disable	When set to Disable the IPROG command is removed from the .bit file.
BITSTREAM.CONFIG.SELECTMAPABORT	Enable	Enable, Disable	Enables or disables the SelectMAP mode Abort sequence. If disabled, an Abort sequence on the device pins is ignored.
BITSTREAM.CONFIG.CONFIGFallback	Enable	Enable, Disable	Enables or disables the loading of a default bitstream when a configuration attempt fails.
BITSTREAM.CONFIG.PROGPIN	Pullup	Pullup, Pullnone	Adds an internal pull-up to the PROGRAM_B pin. The Pullnone setting disables the pullup. The pullup affects the pin after configuration.
BITSTREAM.CONFIG.PUDC_B	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the PUDC_B pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the PUDC_B pin.
BITSTREAM.CONFIG.RDWR_B_FCS_B	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the RDWR_B_FCS_B pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the RDWR_B_FCS_B pin.

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.REVISIONSELECT	00	00, 01, 10, 11	Specifies the internal value of the RS[1:0] settings in the Warm Boot Start Address (WBSTAR) register for the next warm boot.
BITSTREAM.CONFIG.REVISIONSELECT_TRISTATE	Disable	Disable, Enable	Specifies whether the RS[1:0] 3-state is enabled by setting the option in the Warm Boot Start Address(WBSTAR). RS[1:0] pins 3-state enable 0: Enable RS 3-state 1: Disable RS 3-state
BITSTREAM.CONFIG.OVERTEMPSHUTDOWN	Disable	Disable, Enable	Enables the device to shut down when the System Monitor detects a temperature beyond the acceptable operational maximum. An external circuitry set up for the System Monitor is required to use this option.
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No, Yes	Enables SPI 32-bit address style, which is required for SPI devices with storage of 256 Mb and larger.
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE, 1, 2, 4, 8	Sets the SPI bus to Dual (x2) or Quad (x4) mode for Master SPI configuration from third party SPI flash devices.
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No, Yes	Sets the FPGA to use a falling edge clock for SPI data capture. This improves timing margins and can allow faster clock rates for configuration.
BITSTREAM.CONFIG.TCKPIN	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to the TCK pin, the JTAG test clock. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.TDIPIN	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to the TDI pin, the serial data input to all JTAG instructions and JTAG registers. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.TDOPIN	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to the TDO pin, the serial data output for all JTAG instruction and data registers. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.TIMER_CFG			Enables the Watchdog Timer in Configuration mode and sets the value. This option cannot be used at the same time as TIMER_USR.
BITSTREAM.CONFIG.TIMER_USR			Enables the Watchdog Timer in Configuration mode and sets the value. This option cannot be used at the same time as TIMER_CFG.
BITSTREAM.CONFIG.TMSPIN	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, pull-down, or neither to the TMS pin, the mode input signal to the TAP controller. The TAP controller provides the control logic for JTAG. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to unused SelectIO™ pins (IOBs). It has no effect on dedicated configuration pins. The list of dedicated configuration pins varies depending upon the architecture. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.USERID	0xFFFFFFFF0xFFFFFFFF		Used to identify implementation revisions. You can enter up to an 8-digit hexadecimal string in the User ID register.
BITSTREAM.CONFIG.USR_ACCESS	None	None, <8-digit hex string>, TIMESTAMP	Writes an 8-digit hexadecimal string, or a timestamp into the AXSS configuration register. The format of the timestamp value is dddd MMMM yyyy hh:mm:ss : day, month, year (year 2000 = 00000), hour, minute, seconds. The contents of this register can be directly accessed by the FPGA fabric via the USR_ACCESS primitive.

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.INITSIGNALSErrorR	Enable	Enable, Disable	When Enabled, the INIT_B pin asserts to '0' when a configuration error is detected.
BITSTREAM.ENCRYPTION.ENCRYPT	No	No, Yes	Encrypts the bitstream.
BITSTREAM.ENCRYPTION.NDEBUGKDFKEYS			When enabled, generate a debug file containing all the keys generated in the KDF mode.
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT	bbram	bbram, efuse	Determines the location of the AES encryption key to be used, either from the battery-backed RAM (BBRAM) or the eFUSE register. This property is only available when the Encrypt option is set to True.
BITSTREAM.ENCRYPTION.OBFUSCATEKEY	Disable	Disable, Enable	When the AES key is not read-secured, a read of the key returns the CRC hash of the key instead of the actual key value.
BITSTREAM.ENCRYPTION.KEY0			Key0 sets the 64-bit AES encryption key for bitstream encryption. To get the pick setting, leave this blank generator to select a random number for the value. To use this option, you must first set Encrypt to Yes.
BITSTREAM.ENCRYPTION.STARTIV0			Sets the starting AES initial vector value. Only the first 96 bits of the 128-bit value are used for the initialization vector. To use this option, you must first set Encrypt to Yes.
BITSTREAM.ENCRYPTION.STARTIVOBFUSCATE			Sets the 128-bit starting obfuscate initial vector value. To use this option, you must first set Encrypt to Yes.
BITSTREAM.ENCRYPTION.NKDFFIXEDINPUT			Optional 60-byte fixed input value, specified as a 120-digit hexadecimal value. This 60-byte fixed input along with the 4-Byte counter serves as the 64-Byte fixed input data to the KDF pseudo-random-function (PRF) to generate the 32-byte key output (KO). If not specified, write_bitstream

Setting	Default Value	Possible Values	Description
			generates a 60-byte pseudo-random fixed input value using RAND_bytes.
BITSTREAM.ENCRIPTION.NKYFILESEED	NKYFILE		Optional 32-byte seed value for the KDF, specified as a 64-digit hexadecimal value. If not specified, write_bitstream takes the Key0 value from an input .NKY file as the seed value. If not specified and if no Key0 input from an NKY file exists, write_bitstream generates a 32-byte pseudo-random seed value via RAND_bytes.
BITSTREAM.ENCRIPTION.KEYFILE			Specifies the name of the input encryption file (with a .nky file extension). To use this option, you must first set Encrypt to Yes.
BITSTREAM.ENCRIPTION.KEYLIFE	32	4 up to 2147483647	The number of 128-bit encryption blocks over which a single key should be used for AES-GCM authenticated bitstreams.
BITSTREAM.ENCRIPTION.RSAKEYLIFEFRAMES	8	8 up to 2147483647	Specifies how many configuration frames should be used for any given AES-256 key when RSA Public Key Authentication is specified. A value of 8 configuration frames is equivalent to using the key for 246 encryption blocks.
BITSTREAM.GENERAL.COMPRESS	False	True, False	Uses the multiple frame write feature in the bitstream to reduce the size of the bitstream, not the bit file. Using compress does not guarantee that the size of the bitstream shrinks.
BITSTREAM.GENERAL.CRCENABLE	Enable, Disable		Controls the generation of a Cyclic Redundancy Check (CRC) value in the bitstream. When enabled, a unique CRC value is calculated based on bitstream contents. If the calculated CRC value does not match the CRC value in the bitstream, the device fails to configure. When CRC is disabled a constant value is inserted in the bitstream in place of the CRC, and the device does not calculate a CRC.

Setting	Default Value	Possible Values	Description
			The CRC default value is Enable, except when BITSTREAM.ENCRYPTION.ENCRYPT is Yes, the CRC is disabled.
BITSTREAM.GENERAL.DEBUGBITSTREAM	No	No, Yes	Lets you create a debug bitstream. A debug bitstream is significantly larger than a standard bitstream. DebugBitstream can be used only for master and slave serial configurations. DebugBitstream is not valid for Boundary Scan or Slave Parallel/SelectMAP. In addition to a standard bitstream, a debug bitstream offers the following features: Writes 32 0s to the LOUT register after the synchronization word. Loads each frame individually. Performs a Cyclic Redundancy Check (CRC) after each frame. Writes the frame address to the LOUT register after each frame.
BITSTREAM.GENERAL.PERFRAMECRC	No	No, Yes	Inserts CRC values at regular intervals within bitstreams. These values validate the integrity of the incoming bitstream and can flag an error (shown on the INIT_B pin and the PRERROR port of the ICAP) prior to loading the configuration data into the device. While most appropriate for partial bitstreams, when set to Yes, this property inserts the CRC values into all bitstreams, including full device bitstreams.
BITSTREAM.GENERAL.SYSMONPOWERDOWN	Disable	Disable, Enable	Enables the device to power down SYSMON to save power. Only recommended for permanently powering down SYSMON.
BITSTREAM.GENERAL.DISABLE_JTAG	No	No, Yes	Disables communication to the Boundary Scan (BSCAN) block via JTAG after configuration.
BITSTREAM.GENERAL.JTAG_SYSMON	Enable	Enable, Disable, StatusOnly	Enables or disables the JTAG connection to SYSMON.

Setting	Default Value	Possible Values	Description
BITSTREAM.READBACK. Auto ICAP_SELECT		Auto, Top, Bottom	Selects between the top and bottom ICAP ports.
BITSTREAM.READBACK. No ACTIVERECONFIG		No, Yes	Prevents the assertions of GHIGH and GSR during configuration. This is required for the active partial reconfiguration enhancement features.
BITSTREAM. READBACK. SECURITY	None	None, Level1, Level2	Specifies whether to disable Readback and Reconfiguration. Specifying Security Level1 disables Readback.
BITSTREAM.STARTUP. DONE_CYCLE	4	4, 1, 2, 3, 5, 6	Selects the Startup phase that activates the FPGA Done signal. Done is delayed when DonePipe=Yes.
BITSTREAM.STARTUP. GTS_CYCLE	5	5, 1, 2, 3, 4, 6, Done, Keep	Selects the Startup phase that releases the internal 3-state control to the I/O buffers.
BITSTREAM.STARTUP. GWE_CYCLE	6	6, 1, 2, 3, 4, 5, Done, Keep	Selects the Startup phase that asserts the internal write enable to flip-flops, LUT RAMs, and shift registers. GWE_cycle also enables the BRAMS. Before the Startup phase, both block RAMs writing and reading are disabled.
BITSTREAM.STARTUP. LCK_CYCLE	NoWait	NoWait, 0, 1, 2, 3, 4, 5, 6	Selects the Startup phase to wait until MMCM/PLLs lock. If you select NoWait, the Startup sequence does not wait for MMCM/PLLs to lock.
BITSTREAM.STARTUP. MATCH_CYCLE	Auto	Auto, NoWait, 0, 1, 2, 3, 4, 5, 6	Specifies a stall in the Startup cycle until digitally controlled impedance (DCI) match signals are asserted. DCI matching does not begin on the Match_cycle. The Startup sequence waits in this cycle until DCI has matched. Given that there are a number of variables in determining how long it takes DCI to match, the number of CCLK cycles required to complete the Startup sequence can vary in any given system. Ideally, the configuration solution should continue driving CCLK until DONE goes High.

Setting	Default Value	Possible Values	Description
			When the Auto setting is specified, <code>write_bitstream</code> searches the design for any DCI I/O standards. If DCI standards exist, <code>write_bitstream</code> uses <code>BITSTREAM.STARTUP.MATCH_CYCLE=2</code> . Otherwise, <code>write_bitstream</code> uses <code>BITSTREAM.STARTUP.MATCH_CYCLE=NoWait</code> .

UltraScale Bitstream Settings

The device configuration settings for AMD UltraScale™ devices available for use with the `set_property <Setting> <Value> [current_design]` Vivado tool Tcl command are shown in the following table.

Table: UltraScale Bitstream Settings

Setting	Default Value	Possible Values	Description
BITSTREAM.AUTHENTICATION.AUTHENTICATE	No	Yes, No	Indicates whether or not to use RSA authentication. If No, AES_GCM is used.
BITSTREAM.AUTHENTICATION.RSAPRIVATEKEYFILE	None	<string>	Specifies the OpenSSL .pem file that contains the key pairs that should be used to sign the RSA-2048 authenticated bitstream.
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1, 2, 3, 4	Helps synchronize BPI configuration with the timing of page mode operations in flash devices. It allows you to set the cycle number for a valid read of the first page. The <code>BPI_page_size</code> must be set to 4 or 8 for this option to be available.
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1, 4, 8	For BPI configuration, this option lets you specify the page size which corresponds to the number of reads required per page of flash memory.
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable, Type1, Type2	Sets the BPI synchronous configuration mode for different types of BPI flash devices. Disable (the default) disables the synchronous configuration mode. Type1 enables the synchronous configuration mode and settings to support the Micron G18(F) family.

Setting	Default Value	Possible Values	Description
			Type2 enables the synchronous configuration mode and settings to support the Micron (Numonyx) P30 and P33 families.
BITSTREAM.CONFIG.CCLKPIN ¹	Pullup	Pullup, Pullnone	Adds an internal pull-up to the Cclk pin. The Pullnone setting disables the pullup.
BITSTREAM.CONFIG.CONFIGFallback	Enable	Disable, Enable	Enables or disables the loading of a default bitstream when a configuration attempt fails.
BITSTREAM.CONFIG.CONFIGRATE	3	3, 6, 9, 12, 22, 33, 40, 50, 57, 69, 82, 87, 90, 110, 115, 130, 148	Uses an internal oscillator to generate the configuration clock, Cclk, when configuring in a master mode. Use this option to select the rate for Cclk.
BITSTREAM.CONFIG.D00_MOSI ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the D00_MOSI pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the D00_MOSI pin.
BITSTREAM.CONFIG.D01_DIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the D01_DIN pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the D01_DIN pin.
BITSTREAM.CONFIG.D02 ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the D02 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the D02 pin.
BITSTREAM.CONFIG.D03 ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the D03 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the D03 pin.
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired, Continuous, Quiet	Controls how often the Digitally Controlled Impedance circuit attempts to update the impedance match for DCI IOSTANDARDS.
BITSTREAM.CONFIG.DONEPIN ¹	Pullup	Pullup, Pullnone	Adds an internal pull-up to the DONE pin. The Pullnone setting disables the pullup.

Setting	Default Value	Possible Values	Description
			Use DonePin only if you intend to connect an external pull-up resistor to this pin. The internal pull-up resistor is automatically connected if you do not use DonePin.
BITSTREAM.CONFIG. EXTMASTERCCLK_EN	Disable	Disable, Div-1, Div-2, Div-3, Div-4, Div-6, Div-8, Div-12, Div-16, Div-24, Div-48	Allows an external clock to be used as the configuration clock for all master modes. The external clock must be connected to the dual-purpose EMCCLK pin.
BITSTREAM.CONFIG. INITPIN ¹	Pullup	Pullup, Pullnone	Specifies whether you want to add a Pullup resistor to the INIT pin, or leave the INIT pin floating.
BITSTREAM.CONFIG. INITSIGNALSErrorR	Enable	Enable, Disable	When Enabled, the INIT_B pin asserts to '0' when a configuration error is detected.
BITSTREAM.CONFIG. M0PIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M0 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M0 pin.
BITSTREAM.CONFIG. M1PIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M1 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M1 pin.
BITSTREAM.CONFIG. M2PIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M2 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M2 pin.
BITSTREAM.CONFIG. NEXT_CONFIG_ADDR	none	<string>	Sets the ing address for the next configuration in a MultiBoot set up, which is stored in the WBSTAR register.
BITSTREAM.CONFIG. NEXT_CONFIG_REBOOT	Enable	Enable, Disable	When set to Disable the IPROG command is removed from the .bit file. This allows the Golden image to load upon power up rather than jumping to the multiboot image in a multiboot setup.

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.OVERTEMPSHUTDOWN	Disable	Disable, Enable	Enables the device to shut down when the System Monitor detects a temperature beyond the acceptable operational maximum. An external circuitry set up for the System Monitor is required to use this option.
BITSTREAM.CONFIG.PERSIST	No	No, Yes	Maintains the configuration logic access to the multi-function configuration pins after configuration. Primarily used to maintain the SelectMAP port after configuration for readback access, but can be used with any configuration mode. Persist is not needed for JTAG configuration because the JTAG port is dedicated and always available. Persist and ICAP cannot be used at the same time. Refer to the user guide for a description. Persist is needed for Readback and Partial Reconfiguration using the SelectMAP configuration pins, and should be used when either SelectMAP or Serial modes are used.
BITSTREAM.CONFIG.PROGPIN ¹	Pullup	Pullup, Pullnone	Adds an internal pull-up to the PROGRAM_B pin. The Pullnone setting disables the pullup. The pullup affects the pin after configuration.
BITSTREAM.CONFIG.PUDC_B ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the PUDC_B pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the PUDC_B pin.
BITSTREAM.CONFIG.RDWR_B_FCS_B ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the RDWR_B_FCS_B pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the RDWR_B_FCS_B pin.
BITSTREAM.CONFIG.REVISIONSELECT	00	00, 01, 10, 11	Specifies the internal value of the RS[1:0] settings in the Warm Boot Start Address (WBSTAR) register for the next warm boot.
BITSTREAM.CONFIG.REVISIONSELECT_TRISTATE	Disable	Disable, Enable	Specifies whether the RS[1:0] 3-state is enabled by setting the option in the Warm Boot Start Address (WBSTAR).

Setting	Default Value	Possible Values	Description
			RS[1:0] pins 3-state enable 0: Enable RS 3-state 1: Disable RS 3-state
BITSTREAM.CONFIG.SELECTMAPABORT	Enable	Enable, Disable	Enables or disables the SelectMAP mode Abort sequence. If disabled, an Abort sequence on the device pins is ignored.
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No, Yes	Enables SPI 32-bit address style, which is required for SPI devices with storage of 256 Mb and larger.
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE, 1, 2, 4, 8	Sets the SPI bus to Dual (x2) or Quad (x4) mode for Master SPI configuration from third party SPI flash devices.
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No, Yes	Sets the FPGA to use a falling edge clock for SPI data capture. This improves timing margins and can allow faster clock rates for configuration.
BITSTREAM.CONFIG.TCKPIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to the TCK pin, the JTAG test clock. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.TDIPIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to the TDI pin, the serial data input to all JTAG instructions and JTAG registers. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.TDOPIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to the TDO pin, the serial data output for all JTAG instruction and data registers. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.TIMER_CFG	None	<8-digit hex string>	Enables the Watchdog Timer in Configuration mode and sets the value. This option cannot be used at the same time as TIMER_USR.
BITSTREAM.CONFIG.TIMER_USR	0x00000000	<8-digit hex>	Enables the Watchdog Timer in Configuration mode and sets the value. This

Setting	Default Value	Possible Values	Description
		string>	option cannot be used at the same time as TIMER_CFG.
BITSTREAM.CONFIG.TMSPIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, pull-down, or neither to the TMS pin, the mode input signal to the TAP controller. The TAP controller provides the control logic for JTAG. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pulldown, Pullup, Pullnone	Adds a pull-up, a pull-down, or neither to unused SelectIO pins (IOBs). It has no effect on dedicated configuration pins. The list of dedicated configuration pins varies depending upon the architecture. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	<8-digit hex string>	Used to identify implementation revisions. You can enter up to an 8-digit hexadecimal string in the User ID register.
BITSTREAM.CONFIG.USR_ACCESS	None	<8-digit hex string>, TIMESTAMP	Writes an 8-digit hexadecimal string, or a timestamp into the AXSS configuration register. The format of the timestamp value is dddd MMMM yyyy hhmmss : day, month, year (year 2000 = 00000), hour, minute, seconds. The contents of this register can be directly accessed by the FPGA fabric via the USR_ACCESS primitive.
BITSTREAM.ENCRYPTION ENCRYPT	No	Yes	Encrypts the bitstream.
BITSTREAM.ENCRYPTION DEBUGKEYS	No	Yes	When enabled, generate a debug file containing all the keys generated in the KDF mode.
BITSTREAM.ENCRYPTION ENCRYPTKEYSELECT	bbram	bbram, efuse	Determines the location of the AES encryption key to be used, either from the battery-backed RAM (BBRAM) or the eFUSE register.

Setting	Default Value	Possible Values	Description
			This property is only available when the Encrypt option is set to True.
BITSTREAM.ENCRYPTION FAMILY_KEY_FILEPATH	None	Path to familyKey.cfg	<p>Specifies the install location of the Family Key. No specific directory is required.</p> <p>AMD does not provide the family key as part of the AMD Tool Suite. Customers must send a request for the family key to secure.solutions@xilinx.com. The family key is distributed to qualified customers through the Product Licensing site on https://www.xilinx.com.</p>
BITSTREAM.ENCRYPTION KEY0	None	<hex string>	Key0 sets the AES encryption key for bitstream encryption. To use this option, you must first set Encrypt to Yes.
BITSTREAM.ENCRYPTION KEYFILE	None	<string>	Specifies the name of the input encryption file (with a .nky file extension). To use this option, you must first set Encrypt to Yes.
BITSTREAM.ENCRYPTION KEYLIFE	128	4 up to 2147483647	<p>The number of 128-bit encryption blocks over which a single key should be used for AES-GCM authenticated bitstreams.</p> <p>Setting this to 0 disables these options</p>
BITSTREAM.ENCRYPTION RSAKEYLIFEFRAMES	8	8 up to 2147483647	<p>Specifies how many configuration frames should be used for any given AES-256 key when RSA Public Key Authentication is specified. A value of 8 configuration frames is equivalent to using the key for 246 encryption blocks.</p> <p>Setting this to 0 disables these options.</p>
BITSTREAM.ENCRYPTION OBFUSCATEKEY	Disable	Enable, Disable	Creates a bitstream whereby the key used to encrypt the bitstream is obfuscated before it is written to eFUSE or battery-backed RAM (BBR). This allows the user to provide the device programmer with an obfuscated key rather than the original customer key. The device programmer can write the obfuscated key to the eFUSE or BBR and mark it as obfuscated using the obfuscated-key flag in the selected storage location.

Setting	Default Value	Possible Values	Description
BITSTREAM.ENCRYPTION.STARTIV0			The initialization vector used to specify the initial GCM count value in the first AES-GCM message. 128-bit hex value.
BITSTREAM.ENCRYPTION.STARTIVOBFUSCATE			Sets the starting AES initial vector value. Only the first 96 bits of the 128-bit value are used for the initialization vector. To use this option, you must first set Encrypt to Yes
BITSTREAM.ENCRYPTION.NKYKDFFIXEDINPUT			Optional 60-byte fixed input value, specified as a 120-digit hexadecimal value. This 60-byte fixed input along with the 4-Byte counter serves as the 64-Byte fixed input data to the KDF pseudo-random-function (PRF) to generate the 32-byte key output (KO). If not specified, write_bitstream generates a 60-byte pseudo-random fixed input value via RAND_bytes.
BITSTREAM.ENCRYPTION.NKYKDFSEED			Optional 32-byte seed value for the KDF, specified as a 64-digit hexadecimal value. If not specified, write_bitstream takes the Key0 value from an input .NKY file as the seed value. If not specified and if no Key0 input from an NKY file exists, write_bitstream generates a 32-byte pseudo-random seed value via RAND_bytes.
BITSTREAM.GENERAL.COMPRESS	False	True, False	Uses the multiple frame write feature in the bitstream to reduce the size of the bitstream, not the bitstream (.bit) file. Using Compress does not guarantee that the size of the bitstream shrinks.
BITSTREAM.GENERAL.CRC	Enable	Enable, Disable	Controls the generation of a cyclic redundancy check (CRC) value in the bitstream. When enabled, a unique CRC value is calculated based on bitstream contents. If the calculated CRC value does not match the CRC value in the bitstream, the device fails to configure. When CRC is disabled a constant value is inserted in the bitstream in place of the CRC, and the device does not calculate a CRC.

Setting	Default Value	Possible Values	Description
			The CRC default value is Enable, except when BITSTREAM.ENCRYPTION.ENCRYPT is Yes, the CRC is disabled.
BITSTREAM.GENERAL.DEBUGBITSTREAM	No	No, Yes	Lets you create a debug bitstream. A debug bitstream is significantly larger than a standard bitstream. DebugBitstream can be used only for master and slave serial configurations. DebugBitstream is not valid for Boundary Scan or Slave Parallel>SelectMAP. In addition to a standard bitstream, a debug bitstream offers the following features: Writes 32 0s to the LOUT register after the synchronization word. Loads each frame individually. Performs a Cyclic Redundancy Check (CRC) after each frame. Writes the frame address to the LOUT register after each frame.
BITSTREAM.GENERAL.DISABLE_JTAG	No	No, Yes	Disables communication to the Boundary Scan (BSCAN) block via JTAG after configuration.
BITSTREAM.GENERAL.PERFRAMECRC	No	No, Yes	Inserts CRC values at regular intervals within bitstreams. These values validate the integrity of the incoming bitstream and can flag an error (shown on the INIT_B pin and the PRERROR port of the ICAP) prior to loading the configuration data into the device. While most appropriate for partial bitstreams, when set to Yes, this property inserts the CRC values into all bitstreams, including full device bitstreams.
BITSTREAM.GENERAL.JTAG_SYSMON	Enable	Enable, Disable, StatusOnly	Enables or disables the JTAG connection to SYSMON.
BITSTREAM.GENERAL.SYSMONPOWERDOWN	Disable	Disable, Enable	Enables the device to power down SYSMON to save power. Only recommended for permanently powering down SYSMON.

Setting	Default Value	Possible Values	Description
BITSTREAM.MMCM.BANDWIDTH	OPTIMIZED	POSTCRC	Changes all MMCM(s) with a BANDWIDTH setting of OPTIMIZED to POSTCRC.
BITSTREAM.PLL.BANDWIDTH	OPTIMIZED	POSTCRC	Changes all PLL(s) with a BANDWIDTH setting of OPTIMIZED to POSTCRC.
BITSTREAM.READBACK.No ACTIVERECONFIG		No, Yes	Prevents the assertions of GHIGH and GSR during configuration. This is required for the active partial reconfiguration enhancement features.
BITSTREAM.READBACK.Auto ICAP_SELECT		Auto, Top, Bottom	Selects between the top and bottom ICAP ports.
BITSTREAM.READBACK.False READBACK		True, False	Lets you perform the Readback function by creating the necessary readback files.
BITSTREAM.READBACK.None SECURITY		None, Level1, Level2	Specifies whether to disable Readback and Reconfiguration. Specifying Security Level1 disables Readback. Specifying Security Level2 disables Readback and Reconfiguration.
BITSTREAM.STARTUP.DONE_CYCLE	4	4, 1, 2, 3, 5, 6, Keep	Selects the Startup phase that activates the FPGA Done signal. Done is delayed when DonePipe=Yes.
BITSTREAM.STARTUP.GTS_CYCLE	5	5, 1, 2, 3, 4, 6, Done, Keep	Selects the Startup phase that releases the internal 3-state control to the I/O buffers.
BITSTREAM.STARTUP.GWE_CYCLE	6	6, 1, 2, 3, 4, 5, Done, Keep	Selects the Startup phase that asserts the internal write enable to flip-flops, LUT RAMs, and shift registers. GWE_cycle also enables the BRAMS. Before the Startup phase, both block RAMs writing and reading are disabled.
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait, 0, 1, 2, 3, 4, 5, 6	Selects the Startup phase to wait until DLLs/DCMs/PLLs lock. If you select NoWait, the Startup sequence does not wait for DLLs/DCMs/PLLs to lock.
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto, NoWait, 0,	Specifies a stall in the Startup cycle until digitally controlled impedance (DCI) match

Setting	Default Value	Possible Values	Description
		1, 2, 3, 4, 5, 6	<p>signals are asserted. DCI matching does not begin on the Match_cycle. The Startup sequence waits in this cycle until DCI has matched. Given that there are a number of variables in determining how long it takes DCI to match, the number of CCLK cycles required to complete the Startup sequence can vary in any given system. Ideally, the configuration solution should continue driving CCLK until DONE goes High.</p> <p>When the Auto setting is specified, write_bitstream searches the design for any DCI I/O standards. If DCI standards exist, write_bitstream uses BITSTREAM.STARTUP.MATCH_CYCLE=2. Otherwise, write_bitstream uses BITSTREAM.STARTUP.MATCH_CYCLE=NoWait.</p>
1. For the dedicated configuration pins AMD recommends that you use the default bitstream setting.			

Zynq UltraScale+ MPSoC Bitstream Settings

The device configuration settings for AMD Zynq™ UltraScale+™ MPSoC devices available for use with the `set_property <Setting> <Value> [current_design]` Vivado tool Tcl command are shown in the following table.

Table: Zynq UltraScale+ MPSoC Bitstream Settings

Setting	Default Value	Possible Value	Description
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired, Quiet, Safe	Controls how often the Digitally Controlled Impedance circuit attempts to update the impedance match for DCI IOSTANDARDS.
BITSTREAM.CONFIG.PUDC_B	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the PUDC_B pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the PUDC_B pin.
BITSTREAM.CONFIG.OVERTEMPSHUTDOWN	Disable	Disable, Enable	Enables the device to shut down when the System Monitor detects a temperature beyond the acceptable operational

Setting	Default Value	Possible Value	Description
			maximum. An external circuitry set up for the System Monitor is required to use this option.
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to unused SelectIO pins (IOBs). It has no effect on dedicated configuration pins. The list of dedicated configuration pins varies depending upon the architecture. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	0xFFFFFFFF	Used to identify implementation revisions. You can enter up to an 8-digit hexadecimal string in the User ID register.
BITSTREAM.CONFIG.USR_ACCESS	None	None, <8-digit hex string>, TIMESTAMPssssss : day, month, year (year 2000 = 00000), hour, minute, seconds.	Writes an 8-digit hexadecimal string, or a timestamp into the AXSS configuration register. The format of the timestamp value is dddd MMMM yyyy hhmmmm TIMESTAMPssssss : day, month, year (year 2000 = 00000), hour, minute, seconds. The contents of this register can be directly accessed by the FPGA fabric via the USR_ACCESS primitive.
BITSTREAM.CONFIG.INITSIGNALSErrorR	Enable	Enable, Disable	When Enabled, the INIT_B pin asserts to '0' when a configuration error is detected.
BITSTREAM.GENERAL.COMPRESS	False	True, False	Uses the multiple frame write feature in the bitstream to reduce the size of the bitstream, not the bit file. Using compress does not guarantee that the size of the bitstream shrinks.
BITSTREAM.GENERAL.CRC	Enable	Enable, Disable	Controls the generation of a Cyclic Redundancy Check (CRC) value in the bitstream. When enabled, a unique CRC value is calculated based on bitstream contents. If the calculated CRC value does not match the CRC value in the bitstream, the device fails to configure. When CRC is disabled a constant value is inserted in the

Setting	Default Value	Possible Value	Description
			bitstream in place of the CRC, and the device does not calculate a CRC.
BITSTREAM.GENERAL.PERFRAMECRC	No	No, Yes	Inserts CRC values at regular intervals within bitstreams. These values validate the integrity of the incoming bitstream and can flag an error (shown on the INIT_B pin and the PRERROR port of the ICAP) prior to loading the configuration data into the device. While most appropriate for partial bitstreams, when set to Yes, this property inserts the CRC values into all bitstreams, including full device bitstreams.
BITSTREAM.GENERAL.SYSMONPOWERDOWN	Disable	Disable, Enable	Enables the device to power down SYSMON to save power. Only recommended for permanently powering down SYSMON.
BITSTREAM.GENERAL.DISABLE_JTAG	No	No, Yes	Disables communication to the Boundary Scan (BSCAN) block via JTAG after configuration.
BITSTREAM.GENERAL.JTAG_SYSMON	Enable	Enable, Disable, StatusOnly	Enables or disables the JTAG connection to SYSMON.
BITSTREAM.READBACK.ICAP_SELECT	Auto	Auto, Top, Bottom	Selects between the top and bottom ICAP ports.
BITSTREAM.READBACK.ACTIVERECONFIG	No	No, Yes	Prevents the assertions of GHIGH and GSR during configuration. This is required for the active partial reconfiguration enhancement features.
BITSTREAM.READBACK.SECURITY	None	None, Level1, Level2	Specifies whether to disable Readback and Reconfiguration. Specifying Security Level1 disables Readback.Specifying Security
BITSTREAM.STARTUP.DONE_CYCLE	4	4, 1, 2, 3, 5, 6, Keep	Selects the Startup phase that activates the FPGA Done signal. Done is delayed when DonePipe=Yes

Setting	Default Value	Possible Value	Description
BITSTREAM.STARTUP.GTS_CYCLE	5	5, 1, 2, 3, 4, 6, Done, Keep	Selects the Startup phase that releases the internal 3-state control to the I/O buffers
BITSTREAM.STARTUP.GWE_CYCLE	6	6, 1, 2, 3, 4, 5, Done, Keep	Selects the Startup phase that asserts the internal write enable to flip-flops, LUT RAMs, and shift registers. GWE_cycle also enables the BRAMS. Before the Startup phase, both block RAMs writing and reading are disabled.
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait, 0, 1, 2, 3, 4, 5, 6	Selects the Startup phase to wait until MMCM/PLLs lock. If you select NoWait, the Startup sequence does not wait for MMCM/PLLs to lock.
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto, NoWait, 0, 1, 2, 3, 4, 5, 6	Specifies a stall in the Startup cycle until digitally controlled impedance (DCI) match signals are asserted. DCI matching does not begin on the Match_cycle. The Startup sequence waits in this cycle until DCI has matched. Given that there are a number of variables in determining how long it takes DCI to match, the number of CCLK cycles required to complete the Startup sequence can vary in any given system. Ideally, the configuration solution should continue driving CCLK until DONE goes high. When the Auto setting is specified, write_bitstream searches the design for any DCI I/O standards. If DCI standards exist, write_bitstream uses BITSTREAM.STARTUP.MATCH_CYCLE=2. Otherwise, write_bitstream uses BITSTREAM.STARTUP.MATCH_CYCLE=NoWait.

Zynq 7000 Bitstream Settings

The device configuration settings for AMD Zynq™ 7000 devices available for use with the `set_property <Setting> <Value> [current_design]` Vivado tool Tcl command are shown in the following table.

 **Note:** Bitstream settings for encryption are not valid for Zynq 7000 devices.

Table: Zynq 7000 Bitstream Settings

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1, 2, 3, 4	Helps synchronize BPI configuration with the timing of page mode operations in flash devices. It allows you to set the cycle number for a valid read of the first page. The BPI_page_size must be set to 4 or 8 for this option to be available.
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1, 4, 8	For BPI configuration, this option lets you specify the page size which corresponds to the number of reads required per page of flash memory.
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable, Type1, Type2	<p>Sets the BPI synchronous configuration mode for different types of BPI flash devices.</p> <p>Disable (the default) disables the synchronous configuration mode.</p> <p>Type1 enables the synchronous configuration mode and settings to support the Micron G18(F) family.</p> <p>Type2 enables the synchronous configuration mode and settings to support the Micron (Numonyx) P30 and P33 families.</p>
BITSTREAM.CONFIG.CCLKPIN ¹	Pullup	Pullup, Pullnone	Adds an internal pull-up to the Cclk pin. The Pullnone setting disables the pullup.
BITSTREAM.CONFIG.CONFIGFALLBACK	Enable	Disable, Enable	Enables or disables the loading of a default bitstream when a configuration attempt fails.
BITSTREAM.CONFIG.CONFIGRATE	3	3, 6, 9, 12, 16, 22, 26, 33, 40, 50, 66	Uses an internal oscillator to generate the configuration clock, Cclk, when configuring in a master mode. Use this option to select the rate for Cclk.
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired, Continuous, Quiet	Controls how often the Digitally Controlled Impedance circuit attempts to update the impedance match for DCI IOSTANDARDS.
BITSTREAM.CONFIG.DONEPIN ¹	Pullup	Pullup, Pullnone	Adds an internal pull-up to the DONE pin. The Pullnone setting disables the pullup. Use DonePin only if you intend to connect

Setting	Default Value	Possible Values	Description
			an external pull-up resistor to this pin. The internal pull-up resistor is automatically connected if you do not use DonePin.
BITSTREAM.CONFIG.INITPIN ¹	Pullup	Pullup, Pullnone	Specifies whether you want to add a Pullup resistor to the INIT pin, or leave the INIT pin floating.
BITSTREAM.CONFIG.INITSIGNALSErrorR	Enable	Enable, Disable	When Enabled, the INIT_B pin asserts to '0' when a configuration error is detected.
BITSTREAM.CONFIG.M0PIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M0 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M0 pin.
BITSTREAM.CONFIG.M1PIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M1 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M1 pin.
BITSTREAM.CONFIG.M2PIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds an internal pull-up, pull-down, or neither to the M2 pin. Select Pullnone to disable both the pull-up resistor and the pull-down resistor on the M2 pin.
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	None	<string>	Sets the starting address for the next configuration in a MultiBoot set up, which is stored in the WBSTAR register.
BITSTREAM.CONFIG.NEXT_CONFIG_REBOOT	Enable	Enable, Disable	When set to Disable the IPORG command is removed from the .bit file. This allows the Golden image to load upon power up rather than jumping to the multiboot image in a multiboot setup.
BITSTREAM.CONFIG.OVERTEMPPOWERDOWN	Disable	Disable, Enable	Enables the device to shut down when the XADC detects a temperature beyond the acceptable operational maximum. An external circuitry set up for the XADC is required to use this option.
BITSTREAM.CONFIG.PERSIST	No	No, Yes	Maintains the configuration logic access to the multi-function configuration pins after configuration. Primarily used to maintain the SelectMAP port after configuration for

Setting	Default Value	Possible Values	Description
			<p>readback access, but can be used with any configuration mode. Persist is not needed for JTAG configuration because the JTAG port is dedicated and always available.</p> <p>PERSIST and ICAP cannot be used at the same time.</p> <p>Refer to the user guide for a description.</p> <p>Persist is needed for Readback and Partial Reconfiguration using the SelectMAP configuration pins, and should be used when either SelectMAP or Serial modes are used.</p>
BITSTREAM.CONFIG.REVISIONSELECT	00	00, 01, 10, 11	Specifies the internal value of the RS[1:0] settings in the Warm Boot Start Address (WBSTAR) register for the next warm boot.
BITSTREAM.CONFIG.REVISIONSELECT_TRISTATE	Disable	Disable, Enable	<p>Specifies whether the RS[1:0] 3-state is enabled by setting the option in the Warm Boot Start Address (WBSTAR).</p> <p>RS[1:0] pins 3-state enable</p> <p>0: Enable RS 3-state</p> <p>1: Disable RS 3-state</p>
BITSTREAM.CONFIG.SELECTMAPABORT	Enable	Enable, Disable	Enables or disables the SelectMAP mode Abort sequence. If disabled, an Abort sequence on the device pins is ignored.
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No, Yes	Enables SPI 32-bit address style, which is required for SPI devices with storage of 256 Mb and larger.
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE, 1, 2, 4	Sets the SPI bus to Dual (x2) or Quad (x4) mode for Master SPI configuration from third party SPI flash devices.
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No, Yes	Sets the FPGA to use a falling edge clock for SPI data capture. This improves timing margins and can allow faster clock rates for configuration.
BITSTREAM.CONFIG.TCKPIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to the TCK pin, the JTAG test clock. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.TDIPIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to the TDI pin, the serial data input to all JTAG instructions and JTAG registers. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.TDOPIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, a pull-down, or neither to the TDO pin, the serial data output for all JTAG instruction and data registers. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.TIMER_CFG	None	<8-digit hex string>	Enables the Watchdog Timer in Configuration mode and sets the value. This option cannot be used at the same time as TIMER_USR.
BITSTREAM.CONFIG.TIMER_USR	0x00000000	<8-digit hex string>	Enables the Watchdog Timer in Configuration mode and sets the value. This option cannot be used at the same time as TIMER_CFG.
BITSTREAM.CONFIG.TMSPIN ¹	Pullup	Pullup, Pulldown, Pullnone	Adds a pull-up, pull-down, or neither to the TMS pin, the mode input signal to the TAP controller. The TAP controller provides the control logic for JTAG. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pulldown, Pullup, Pullnone	Adds a pull-up, a pull-down, or neither to unused SelectIO™ pins (IOBs). It has no effect on dedicated configuration pins. The list of dedicated configuration pins varies depending upon the architecture. The Pullnone setting shows that there is no connection to either the pull-up or the pull-down.
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	<8-digit hex string>	Used to identify implementation revisions. You can enter up to an 8-digit hexadecimal string in the User ID register.
BITSTREAM.CONFIG.USR_ACCESS	None	<8-digit hex	Writes an 8-digit hexadecimal string, or a timestamp into the AXSS configuration

Setting	Default Value	Possible Values	Description
		string>, TIMESTAMP	register. The format of the timestamp value is dddd MMMM yyyy hh:mm:ssssss : day, month, year (year 2000 = 0000), hour, minute, seconds. The contents of this register can be directly accessed by the FPGA fabric via the <code>USR_ACCESS</code> primitive.
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT	bbram	bbram, efuse	Determines the location of the AES encryption key to be used, either from the battery-backed RAM (BBRAM) or the eFUSE register(7 series). This property is only available when the Encrypt option is set to True.
BITSTREAM.GENERAL.COMPRESS	False	True, False	Uses the multiple frame write feature in the bitstream to reduce the size of the bitstream, not the Bitstream (.bit) file. Using Compress does not guarantee that the size of the bitstream shrinks.
BITSTREAM.GENERAL.CRC	Enable	Enable, Disable	Controls the generation of a Cyclic Redundancy Check (CRC) value in the bitstream. When enabled, a unique CRC value is calculated based on bitstream contents. If the calculated CRC value does not match the CRC value in the bitstream, the device fails to configure. When CRC is disabled a constant value is inserted in the bitstream in place of the CRC, and the device does not calculate a CRC. The CRC default value is Enable, except when BITSTREAM.ENCRYPTION.ENCRYPT is Yes, the CRC is disabled.
BITSTREAM.GENERAL.DISABLE_JTAG	No	No, Yes	Disables communication to the Boundary Scan (BSCAN) block via JTAG after configuration.
BITSTREAM.GENERAL.JTAG_XADC	Enable	Enable, Disable, StatusOnly	Enables or disables the JTAG connection to the XADC.

Setting	Default Value	Possible Values	Description
BITSTREAM.GENERAL.XADCENHANCEDLINEARITY	Off	Off, On	Disables some built-in digital calibration features that make INL look worse than the actual analog performance.
BITSTREAM.GENERAL.PERFRAMECRC	No	No, Yes	Inserts CRC values at regular intervals within bitstreams. These values validate the integrity of the incoming bitstream and can flag an error (shown on the INIT_B pin and the PRERROR port of the ICAP) prior to loading the configuration data into the device. While most appropriate for partial bitstreams, when set to Yes, this property inserts the CRC values into all bitstreams, including full device bitstreams.
BITSTREAM.READBACK.NoACTIVERECONFIG	No	No, Yes	Prevents the assertions of GHIGH and GSR during configuration. This is required for the active partial reconfiguration enhancement features.
BITSTREAM.READBACK.AutoICAP_SELECT	Auto	Auto, Top, Bottom	Selects between the top and bottom ICAP ports.
BITSTREAM.READBACK.FalseREADBACK	False	True, False	Lets you perform the Readback function by creating the necessary readback files.
BITSTREAM.READBACK.NoneSECURITY	None	None, Level1, Level2	Specifies whether to disable Readback and Reconfiguration. Specifying Security Level1 disables Readback. Specifying Security Level2 disables Readback and Reconfiguration.
BITSTREAM.READBACK.DisableXADCPARTIALRECONFIG	Disable	Disable, Enable	When Disabled XADC can work continuously during Partial Reconfiguration. When Enabled XADC works in Safe mode during partial reconfiguration.
BITSTREAM.STARTUP.DONEPIPE	Yes	Yes, No	Tells the FPGA to wait on the CFG_DONE (DONE) pin to go High and wait for the first clock edge before moving to the Done state.
BITSTREAM.STARTUP.DONE_CYCLE	4	4, 1, 2, 3, 5, 6, Keep	Selects the Startup phase that activates the FPGA Done signal. Done is delayed when DonePipe=Yes.

Setting	Default Value	Possible Values	Description
BITSTREAM.STARTUP.GTS_CYCLE	5	5, 1, 2, 3, 4, 6, Done, Keep	Selects the Startup phase that releases the internal 3-state control to the I/O buffers.
BITSTREAM.STARTUP.GWE_CYCLE	6	6, 1, 2, 3, 4, 5, Done, Keep	Selects the Startup phase that asserts the internal write enable to flip-flops, LUT RAMs, and shift registers. GWE_cycle also enables the BRAMS. Before the Startup phase, both block RAMs writing and reading are disabled.
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait, 0, 1, 2, 3, 4, 5, 6	Selects the Startup phase to wait until DLLs/DCMs/PLLs lock. If you select NoWait, the Startup sequence does not wait for DLLs/DCMs/PLLs to lock.
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto, NoWait, 0, 1, 2, 3, 4, 5, 6	Specifies a stall in the Startup cycle until digitally controlled impedance (DCI) match signals are asserted. DCI matching does not begin on the Match_cycle. The Startup sequence waits in this cycle until DCI has matched. Given that there are a number of variables in determining how long it takes DCI to match, the number of CCLK cycles required to complete the Startup sequence can vary in any given system. Ideally, the configuration solution should continue driving CCLK until DONE goes high. When the Auto setting is specified, write_bitstream searches the design for any DCI I/O standards. If DCI standards exist, write_bitstream uses BITSTREAM.STARTUP.MATCH_CYCLE=2. Otherwise, write_bitstream uses BITSTREAM.STARTUP.MATCH_CYCLE=NoWait.
BITSTREAM.STARTUP.STARTUPCLK	Cclk	Cclk, UserClk, JtagClk	The StartupClk sequence following the configuration of a device can be synchronized to either Cclk, a User Clock, or the JTAG Clock. The default is Cclk. Cclk lets you synchronize to an internal clock provided in the FPGA device.

Setting	Default Value	Possible Values	Description
			<p>UserClk lets you synchronize to a user-defined signal connected to the CLK pin of the STARTUP symbol.</p> <p>JtagClk lets you synchronize to the clock provided by JTAG. This clock sequences the TAP controller which provides the control logic for JTAG.</p>
1. For the dedicated configuration pins AMD recommends that you use the bitstream setting default.			

Versal Adaptive SoC Programmable Device Image (PDI) Settings

The device configuration settings for Versal adaptive SoC devices are available for use with the `set_property <Setting> <Value> [current_design]` Vivado tool Tcl command are shown in the following table.

 **Note:** On the Versal adaptive SoC architecture, most programmable device image settings previously supported as a Bitstream settings are configured in either the Control, Interface, and Processing System (CIPS) IP or as Bootgen settings. See the *Control, Interface and Processing System LogiCORE IP Product Guide* ([PG352](#)) or the *Bootgen User Guide* ([UG1283](#)) for more information. For example, on the UltraScale architecture the USERCODE was set with the `BITSTREAM.CONFIG.USERID` bitstream setting. To set USERCODE on Versal architectures this must be set on the CIPS IP with the following property: `set_property CONFIG.PS_PMC_CONFIG {JTAG_USERCODE 0x<32 bit hex value>} [get_bd_cells /versal_cips_0]`

Table: Versal adaptive SoC Programmable Device Image Settings

Setting	Default Value	Possible Values	Description
BITSTREAM.CONFIG.USR_ACCESS	None	None, <8-digit hex string>, TIMESTAMP	<p>Writes an 8-digit hexadecimal string, or a timestamp into the USR_ACCESS register in the PLM_RTCA module.</p> <p>The format of the timestamp value is dddd MMMM yyyy hhmmss : day, month, year (year 2000 = 00000), hour, minute, seconds. The contents of</p>

Setting	Default Value	Possible Values	Description
			this register can be directly accessed via the PS or an AXI Manager in the PL.
BITSTREAM.GENERAL.COMPRESS	True	True, False	Use the run length encoding algorithm to reduce the size of the PL configuration data. In most cases this can reduce the size of the PL configuration data.
BITSTREAM.GENERAL.CRC	False	True, False	Controls the generation of a Cyclic Redundancy Check (CRC) value in the PL portion of the PDI. When enabled, a unique CRC value is calculated based on the PL portion of the PDI contents. If the calculated CRC value does not match the CRC value in the PDI the device fails to configure.
BITSTREAM.GENERAL.PERFRAMECRC	False	True, False	Inserts CRC values at regular intervals in the PL portion of the PDI. These values validate the integrity of the incoming configuration data and can flag an error prior to loading the configuration data into the device. While most appropriate for partial PDI's, when set to Yes this property inserts the CRC values into all PDI's including full device images.

Trigger State Machine Language Description

The trigger state machine language is used to describe complex trigger conditions that map to the advanced trigger logic of the ILA debug core. The trigger state machine has the following features:

- Up to 16 states.
- One-, two-, and three-way conditional branching used for complex state transitions.
- Four built-in 16-bit counters are used to count events, implement timers, etc.
- Four built-in flags used for monitoring trigger state machine execution status.
- Trigger action.

States

Each state machine program can have up to 16 states declared. Each state is composed of a state declaration and a body:

```
state <state_name>:  
    <state_body>
```

Goto Action

The goto action is used to transition between states. Here is an example of using the goto action to transition from one state to another before triggering:

```
state my_state_0:  
    goto my_state_1;  
state my_state_1:  
    trigger;
```

Conditional Branching

The trigger state machine language supports one-, two-, and three-way conditional branching per state.

- One-way branching involves using goto actions without any if/elseif/else/endif constructs:

```
state my_state_0:
    goto my_state_1;
```

- Two-way conditional branching uses goto actions with if/else/endif constructs:

```
state my_state_0:
    if (<condition1>) then
        goto my_state_1;
    else
        goto my_state_0;
    endif
```

- Three-way conditional branching uses goto actions with if/else/elseif/endif constructs:

```
state my_state_0:
    if (<condition1>) then
        goto my_state_1;
    elseif (<condition2>) then
        goto my_state_2;
    else
        goto my_state_0;
    endif
```

For more information on how to construct conditional statements represented previously with <condition1> and <condition2>, refer to the section [Conditional Statements](#).

Related Information

[Conditional Statements](#)

Counters

The four built-in 16-bit counters have fixed names and are called \$counter0, \$counter1, \$counter2, \$counter3. The counters can be reset, incremented, and used in conditional statements.

- To reset a counter, use the `reset_counter` action:

```
state my_state_0:
    reset_counter $counter0;
    goto my_state_1;
```

- To increment a counter, use the `increment_counter` action:

```
state my_state_0:
    increment_counter $counter3;
    goto my_state_1;
```

For more information on how to use counters in conditional statements, refer to [Conditional Statements](#).

Related Information

[Conditional Statements](#)

Flags

Flags can be used to monitor progress of the trigger state machine program as it executes. The four built-in flags have fixed names and are called \$flag0, \$flag1, \$flag2, and \$flag3. The flags can be set and cleared.

- To set a flag, use the set_flag action:

```
state my_state_0:  
    set_flag $flag0;  
    goto my_state_1;
```

- To clear a flag, use the clear_flag action:

```
state my_state_0:  
    clear_flag $flag2;  
    goto my_state_1;
```

Conditional Statements

Debug Probe Conditions

Debug probe conditions can be used in two-way and three-way branching conditional statements. Each debug probe condition consumes one trigger comparator on the PROBE port of the ILA to which the debug probe is attached.

!! Important: Each PROBE port can have from 1 to 16 trigger comparators as configured at compile time. This means that you can only use a particular debug probe in a debug probe condition up from 1 to 16 times in the entire trigger state machine program, depending on the number of comparators configured on the PROBE port.

The debug probe conditions consist of a comparison operator and a value. The valid debug probe condition comparison operators are:

- == (equal to)
- != (not equal to)
- > (greater than)
- < (less than)
- >= (greater than or equal to)
- <= (less than or equal to)

Valid values are of the form:

```
<bit_width>'<radix><value>
```

Where:

- <bit width> is the width of the probe (in bits)
- <radix> is one of
 - b (binary)
 - h (hexadecimal)
 - u (unsigned decimal)
- <value> is one of
 - 0 (Logical zero)
 - 1 (Logical one)
 - X (dont care)
 - R (0-to-1 transition) – Valid only for 1-bit probes
 - F (1-to-0 transition) – Valid only for 1-bit probes
 - B (both transitions) – Valid only for 1-bit probes
 - N (No transitions) – Valid only for 1-bit probes

Examples of valid debug probe condition values are:

- 1-bit binary value of 0

```
1'b0
```

- 12-bit hex value of 7A

```
12'h07A
```

- 9-bit integer value of 123

```
9'u123
```

Examples of debug probe condition statements are:

- A single-bit debug probe called abc equals 0

```
if (abc == 1'b0) then
```

- A 23-bit debug probe xyz equals 456

```
if (xyz >= 23'u456) then
```

- A 23-bit debug probe klm does not equal hex A5

```
if (klm != 23'h0000A5) then
```

Examples of multiple debug probe condition statements are:

- Two debug probe comparisons combined with an "OR" function:

```
if ((xyz >= 23'u456) || (abc == 1'b0)) then
```

- Two debug probe comparisons combined with an "AND" function:

```
if ((xyz >= 23'u456) && (abc == 1'b0)) then
```

- Three debug probe comparisons combined with an "OR" function:

```
if ((xyz >= 23'u456) || (abc == 1'b0) || (klm != 23'h0000A5)) then
```

- Three debug probe comparisons combined with an "AND" function:

```
if ((xyz >= 23'u456) && (abc == 1'b0) && (klm != 23'h0000A5)) then
```

Counter Conditions

Counter conditions can be used in two-way and three-way branching conditional statements. Each counter condition consumes one counter comparator.

!! Important: Each counter has only one counter comparator. This means that you can only use a particular counter in a counter condition once in the entire trigger state machine program.

The probe port conditions consist of a comparison operator and a value. The valid probe condition comparison operators are:

- == (equals)
- != (not equals)

!! Important: Each counter is always 16 bits wide.

Examples of valid counter condition values are:

- 16-bit binary value of 0

```
16'b0000_0000_0000_0000
16'b0000000000000000
```

- 16-bit hex value of 7A

```
16'h007A
```

- 16-bit integer value of 123

```
16'u123
```

Examples of counter condition statements:

- Counter \$counter0 equals binary 0

```
($counter0 == 16'b0000000000000000)
```

- Counter \$counter2 does not equal decimal 23

```
($counter2 != 16'u23)
```

Combined Debug Probe and Counter Conditions

Debug probe conditions and counter conditions can be combined together to form a single condition using the following rules:

- All debug probe comparisons must be combined together using the same "||" (OR) or "&&" (AND) operators.
- The combined debug probe condition can be combined with the counter condition using either the "||" (OR) or "&&" (AND) operators, regardless of the operator used to combine the debug probe comparisons together.

Examples of multiple debug probe and counter condition statements are:

- Two debug probe comparisons combined with an "OR" function, combined with counter conditional using "AND" function:

```
if (((xyz >= 23'u456) || (abc == 1'b0)) && ($counter0 == 16'u0023)) then
```

- Two debug probe comparisons combined with an "AND" function, combined with counter conditional using "OR" function:

```
if (((xyz >= 23'u456) && (abc == 1'b0)) || ($counter0 == 16'u0023)) then
```

- Three debug probe comparisons combined with an "OR" function, combined with counter conditional using "AND" function:

```
if (((xyz >= 23'u456) || (abc == 1'b0) || (klm != 23'h0000A5)) && ($counter0
== 16'u0023)) then
```

- Three debug probe comparisons combined with an "AND" function, combined with counter conditional using "OR" function:

```
if (((xyz >= 23'u456) && (abc == 1'b0) && (klm != 23'h0000A5)) || ($counter0
== 16'u0023)) then
```

 Note:

- The language is case insensitive
- Comment character is hash '#' character. Anything including and after a # character is ignored.
- 'THING' = THING is a terminal
- {<thing>} = 0 or more thing
- [<thing>] = 0 or 1 thing

```
<program> ::= <state_list>
<state_list> ::= <state_list> <state> | <state>
<state> ::= 'STATE' <state_label> ':' <if_condition> | <action_block>

<action_block> ::= <action_list> 'GOTO' <state_label> ';'
| <action_list> 'TRIGGER' ';'
| 'GOTO' <state_label> ';'
| 'TRIGGER' ';'
<action_list> ::= <action_statement> | <action_list> <action_statement>
<action_statement> ::= 'SET_FLAG' <flag_name> ';'
| 'CLEAR_FLAG' <flag_name> ';'
| 'INCREMENT_COUNTER' <counter_name> ';'
| 'RESET_COUNTER' <counter_name> ';'
<if_condition> ::= 'IF' '(' <condition> ')' 'THEN' <actionblock>
                  ['ELSEIF' '(' <condition> ')' 'THEN' <actionblock>]
                  'ELSE' <actionblock>
                  'ENDIF'
<condition> ::= <probe_match_list>
| <counter_match>
| <probe_counter_match>
<probe_counter_match> ::= '(' <probe_counter_match> ')'
| <probe_match_list> <boolean_logic_op> <counter_match>
| <counter_match> <boolean_logic_op> <probe_match_list>
<probe_match_list> ::= '(' <probe_match> ')'
| <probe_match>
<probe_match> ::= <probe_match_list> <boolean_logic_op> <probe_match_list>
| <probe_name> <compare_op> <constant>
| <constant> <compare_op> <probe_name>
<counter_match> ::= '(' <counter_match> ')'
| <counter_name> <compare_op> <constant>
| <constant> <compare_op> <counter_name>
<constant> ::= <integer_constant>
| <hex_constant>
| <binary_constant>
<compare_op> ::= '==' | '!=' | '>' | '>=' | '<' | '<='
<boolean_logic_op> ::= '&&' | '||'
--- The following are in regular expression format to simplify expressions:
```

```

--- [A-Z0-9] means match any single character in A...Z,0..9
--- [AB]+ means match [AB] one or more times like A, AB, ABAB, AAA, etc
--- [AB]* means match [AB] zero or more times
<probe_name> ::= [A-Z_\[\]\>\<]/[A-Z_0-9\[\]\>\<]+
<state_label> ::= [A-Z_][A-Z_0-9]+
<flag_name> ::= \$FLAG[0-3]
<counter_name> ::= \$COUNTER[0-3]
<hex_constant> ::= <integer>*'h<hex_digit>+
<binary_constant> ::= <integer>*'b<binary_digit>+
<integer_constant> ::= <integer>*'u<integer_digit>+
<integer> ::= <digit>+
<hex_digit> ::= [0-9ABCDEFBN_]
<binary_digit> ::= [01XRFBN_]
<digit> ::= [0-9]

```

Low Level SVF JTAG Commands

 **Note:** SVF is not supported on AMD Versal™ devices.

Low level JTAG commands allow you to scan multiple FPGA JTAG chains. The SVF commands generated for chain operations use these low-level commands to access the FPGAs in the chain. This appendix is an overview of these commands. A detailed explanation can be found in the Serial Vector Format Specification document that can be found at:

http://www.jtagtest.com/pdf/svf_specification.pdf.

Header Data Register (HDR), Header Instruction Register (HIR)

Syntax

```

HDR length [TDI (tdi)] [TDO (tdo)] [MASK (mask)] [SMASK (smask)];
HIR length [TDI (tdi)] [TDO (tdo)] [MASK (mask)] [SMASK (smask)];

```

Purpose

Specifies a default header pattern that is shifted in before every scan operation. The header pattern specifies how to pad the scan statements with a set of leading bits that accommodate the devices located on the scan path beyond the component of interest.

General Information

The Header Data Register (HDR) specifies a default header pattern pre-pending to the beginning of all subsequent SDR commands. The Header Instruction Register (HIR) specifies a default header pattern to the beginning of all subsequent SIR commands. The header commands have a set of counterpart trailer commands (TIR, TDR) described in the following section. A header can be removed by setting its length to 0.

TDR, TIR (Trailer Data Register, Trailer Instruction Register)

Syntax

```
TDR length [TDI (tdi)] [TDO (tdo)][MASK (mask)] [SMASK (smask)];  
TIR length [TDI (tdi)] [TDO (tdo)][MASK (mask)] [SMASK (smask)];
```

Purpose

Specifies a default trailer pattern that is shifted in after all subsequent scan operations. The trailer pattern specifies how to pad the scan statements with a set of trailing bits that accommodate the devices located on the scan path after the component of interest.

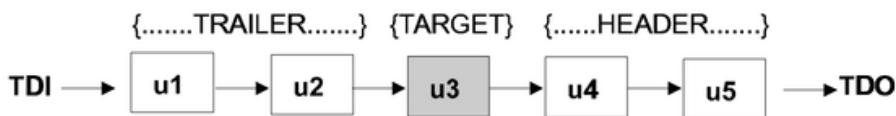
General Information

The Trailer Data Register (TDR) specifies a trailer pattern that is appended to the end of all subsequent SDR commands. Trailer Instruction Register (TIR) specifies a default trailer pattern that is appended to the end of all subsequent SIR commands. A trailer can be removed by setting its length to 0.

Example

In this example an SVF file is developed for an ASIC. The ASIC is placed in a board as u3, as follows:

Figure: TDR Example



The set of SVF statements originally developed for the ASIC can be reused with minimal modification if the appropriate header and trailer statements are defined to accommodate the devices in front of and behind u3. In this example, a header pattern would be defined for devices u4 and u5, and a trailer pattern would be defined for u2 and u1. The optional parameters can be specified in any order. Each optional parameter can only be specified once. Hex strings specified for TDI, TDO, MASK, or SMASK cannot be a value larger than the maximum implied by the length parameter. Leading zeros are assumed for a hex string if not explicitly specified.

scan_ir_hw

Perform shift IR on hw_jtag.

Syntax

```
scan_ir_hw_jtag [-tdi <arg>] [-tdo <arg>] [-mask <arg>] [-smask <arg>] [-quiet]
[-verbose] <length>
```

General Information

The `scan_ir_hw_jtag` command specifies a scan pattern to be scanned into the JTAG interface target instruction register. The command targets a `hw_jtag` object which is created when the `hw_target` is opened in JTAG mode through the use of the `open_hw_target -jtag_mode` command. When targeting the `hw_jtag` object prior to shifting the scan pattern specified in the `scan_ir_hw_jtag` command, the last defined header property (HIR) is pre-pended to the beginning of the specified data pattern. The last defined trailer property (TIR) is appended to the end of the data pattern.

The number of bits represented by the hex strings specified for `-tdi`, `-tdo`, `-mask`, or `-smask` cannot be greater than the maximum specified by `<length>`.

The `scan_ir_hw_jtag` command returns a hex array containing captured TDO data from the `hw_jtag`, or returns an error if it fails.

Example

The following example scans the JTAG instruction register for a 24-bit value:

```
scan_ir_hw_jtag 24
```

The following example sends a 24-bit value `0x00_0010` (LSB first) to TDI, captures the TDO output, applies a mask with `0xF3_FFFF`, and compares the returned TDO value against the specified value `-tdo 0x81_8181`.

```
scan_ir_hw_jtag 24 -tdi 000010 -tdo 818181 -mask F3FFFF -smask 0
```

scan_dr_hw

Perform shift DR on `hw_jtag`.

Syntax

```
scan_dr_hw_jtag [-tdi <arg>] [-tdo <arg>] [-mask <arg>] [-smask <arg>] [-quiet]
[-verbose] <length>
```

General Information

The `scan_dr_hw_jtag` command specifies a scan pattern to be scanned into the JTAG interface target data register. The command targets a `hw_jtag` object which is created when the `hw_target` is opened in JTAG mode through the use of the `open_hw_target -jtag_mode` command. When targeting the `hw_jtag` object prior to shifting the scan pattern specified in the `scan_dr_hw_jtag`

command, the last defined header property (HDR) is pre-pended to the beginning of the specified data pattern. The last defined trailer property (TDR) is appended to the end of the data pattern. The `scan_dr_hw_jtag` command returns a hex array containing captured TDO data from the `hw_jtag`, or returns an error if it fails.

Example

The following example scans the JTAG data register for a 24-bit value:

```
scan_dr_hw_jtag 24
```

The following example sends a 24-bit value `0x00_0010` (LSB first) to TDI, captures the data output, TDO, applies a mask with `0xF3_FFFF`, and compares the returned TDO value against the specified value `-tdo 0x81_8181`.

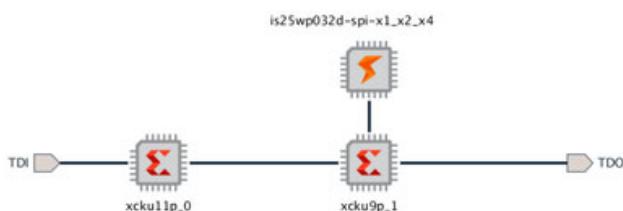
```
scan_dr_hw_jtag 24 -tdi 000010 -tdo 818181 -mask F3FFFF -smask 0
```

Multi Chain SVF Operation

The following examples show how to handle operations on a SVF chain.

Two devices, `xcku11`, and `xcku9`, are connected in a chain. A configuration memory is attached to the second device in the chain (`xcku9`). To access this configuration memory, SVF generates commands using the HIR, HDR, TIR, and TDR commands. The commands generated for flashing this configuration memory take into account the chain length, and incorporate this information in the low level JTAG operations.

Figure: Multi Chain SVF Operation Example



The generated `.svf` file contains the following operations:

```
HIR 0 ;
TIR 6 TDI (3f) SMASK (3f) ;
HDR 0 ;
TDR 1 TDI (00) SMASK (01) ;
// config/idcode
SIR 6 TDI (09) ;
SDR 32 TDI (00000000) TDO (0484a093) MASK (0xffffffff) ;
// config/jprog
STATE RESET;
```

```

STATE IDLE;
SIR 6 TDI (0b) ;
SIR 6 TDI (14) ;
// Modify the below delay for config_init operation (0.100000 sec typical,
0.100000
sec maximum)
RUNTEST 0.100000 SEC;
// config/jprog/poll
RUNTEST 10000 TCK;
SIR 6 TDI (14) TDO (11) MASK (31) ;
// config/slrx
SIR 6 TDI (05) ;

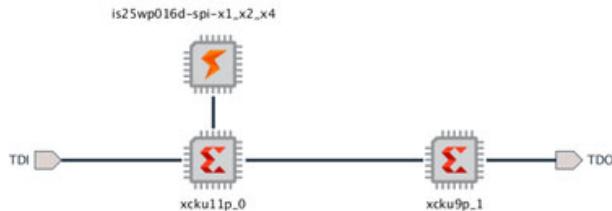
```

Multi Chain SVF Operation with Configuration Memory Attached to First Device

In this example, to access the ku9 device, a SMASK value of 0000 0011 1111 (0x3f) is used with the TIR and TDR instructions. To access the second device in the chain, the mask value is pushed in followed by SIR and SDR instructions. SIR and SDR instructions combine HIR, HDR, TIR, and TDR information.

If the configuration memory attached to the first device (xcku11) needs to be programmed, the SVF generated commands are different:

Figure: Multi Chain SVF Operation Example with Configuration Memory Attached to First Device



```

HIR 6 TDI (3f) SMASK (3f) ;
TIR 0 ;
HDR 1 TDI (00) SMASK (01) ;
TDR 0 ;
// config/idcode
SIR 6 TDI (09) ;
SDR 32 TDI (00000000) TDO (04a4e093) MASK (0xffffffff) ;
// config/jprog
STATE RESET;
STATE IDLE;
SIR 6 TDI (0b) ;
SIR 6 TDI (14) ;
// Modify the below delay for config_init operation (0.100000 sec typical,
0.100000
sec maximum)

```

```

RUNTEST 0.100000 SEC;
// config/jprog/poll
RUNTEST 10000 TCK;
SIR 6 TDI (14) TDO (11) MASK (31) ;
// config/slr
SIR 6 TDI (05) ;

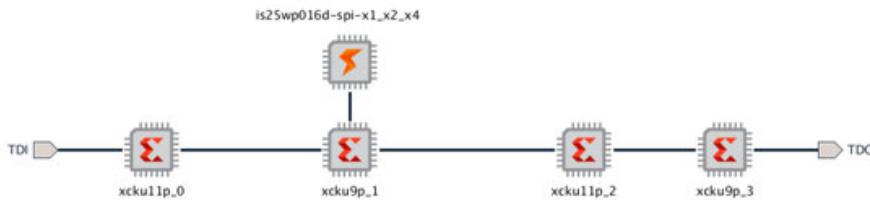
```

Multi Chain SVF Operation with Configuration Memory Attached to Second Device in Chain

In this example, to access the ku11 device, a SMASK value of 0011 1111 (0x3f) is used with the HIR and HDR instructions. To access the first device in the chain, the mask value is pushed first followed by an SIR and SDR instructions. SIR and SDR instructions combine HIR, HDR, TIR, and TDR information.

Now consider four devices, xcku11, xcku9, xcku11, and xcku9, are connected in a chain. A configuration memory is attached to the second device in the chain (xcku9) and if you want to access it, both the HIR and TIR instructions are used in this case:

Figure: Multi-Chain-SVF- Operation- Example-Second Device in Chain



```

HIR 12 TDI (0fff) SMASK (0fff) ;
TIR 6 TDI (3f) SMASK (3f) ;
HDR 2 TDI (00) SMASK (03) ;
TDR 1 TDI (00) SMASK (01) ;
// config/idcode
SIR 6 TDI (09) ;
SDR 32 TDI (00000000) TDO (0484a093) MASK (0xffffffff) ;
// config/jprog
STATE RESET;
STATE IDLE;
SIR 6 TDI (0b) ;
SIR 6 TDI (14) ;
// Modify the below delay for config_init operation (0.100000 sec typical,
0.100000
sec maximum)
RUNTEST 0.100000 SEC;
// config/jprog/poll
RUNTEST 10000 TCK;
SIR 6 TDI (14) TDO (11) MASK (31) ;

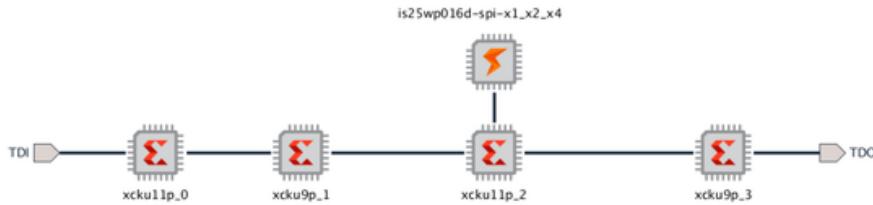
```

```
// config/slr
SIR 6 TDI (05) ;
```

Multi Chain SVF Operation with Configuration Memory Attached to Third Device in Chain

If a configuration memory is attached to the third device in the chain (xcku9).

Figure: Multi-Chain-SVF- Operation- Example-Third Device in Chain



```
HIR 6 TDI (3f) SMASK (3f) ;
TIR 12 TDI (0fff) SMASK (0fff) ;
HDR 1 TDI (00) SMASK (01) ;
TDR 2 TDI (00) SMASK (03) ;
// config/idcode
SIR 6 TDI (09) ;
SDR 32 TDI (00000000) TDO (04a4e093) MASK (0xffffffff) ;
// config/jprog
STATE RESET;
STATE IDLE;
SIR 6 TDI (0b) ;
SIR 6 TDI (14) ;
// Modify the below delay for config_init operation (0.100000 sec typical,
0.100000
sec maximum)
RUNTEST 0.100000 SEC;
// config/jprog/poll
RUNTEST 10000 TCK;
SIR 6 TDI (14) TDO (11) MASK (31) ;
// config/slr
SIR 6 TDI (05) ;
```

JTAG Cables and Devices Supported by hw_server

The list of compatible JTAG download cables and devices that are supported by hw_server are:

- AMD SmartLynq+ Module (HW-SMARTLYNQ-PLUS-G)
- AMD SmartLynq Data Cable (HW-SMARTLYNQ-G/DLC20)
- AMD Platform Cable USB II (DLC10)
- AMD Platform Cable USB (DLC9G, DLC9LP, DLC9)
- Digilent JTAG-HS1
- Digilent JTAG-HS2
- Digilent JTAG-HS3
- Digilent JTAG-SMT1
- Digilent JTAG-SMT2

Programming FTDI Devices for Vivado Hardware Manager Support

For FTDI devices to be recognized as a USB-to-JTAG interface in AMD JTAG software tools such as XSDB or the AMD Vivado™ Hardware Manager the EEPROM on the FTDI device must be programmed. Programming the FTDI is accomplished by using the `program_ftdi` utility included in the Vivado install. Once programmed, the FTDI device is recognized as a valid programming cable in Vivado.

 **Note:** For on-board implementation details including FTDI connectivity, reference the AMD VCK190 Schematics available in XTP610 on <https://www.xilinx.com/products/boards-and-kits/vck190.html>. Ensure that FTDI connectivity (including ADBUS0-7) matches the VCK190 implementation.

 **Note:** The `program_ftdi` utility configures Channel A to be used as a USB-to-JTAG interface. If the device has multiple channels, such as in the case of FT4232H, these channels are configured to the default RS232 UART mode. If the default RS232 UART mode is not desirable, it is possible to use the `FT_PROG` EEPROM utility provided by FTDI to change the port configuration for any of the unused channels without impacting the ability to recognize the FTDI device in AMD JTAG software tools.

The `program_ftdi` utility supports the following FTDI devices:

- FT232H
- FT2232H
- FT4232H

The following is the command reference.

```
***** program_ftdi v2023.1
**** Build date : Apr 16 2023-14:45:22
** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
** Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.
```

Short Description:

Write/Read to FT2I EEPROM for Xilinx JTAG Tools support

Syntax:

```
program_ftdi {-write -ftdi=<ftdi_part> -serial=<serial_number> [options] |  
-write -filein=<cfg_filein> |  
-read [-fileout=<cfg_fileout>] |  
-erase} [-help]
```

options:

Name	Description
------	-------------

-f, -ftdi	Specify the ftdi device to be programmed <FT232H FT2232H FT4232H>
-s, -serial	Serial number to be written into the EEPROM
[-v, --vendor]	Vendor information
[-b, --board]	Name of the board/product being programmed
[-d, -desc, -description]	A short description of the board
-fi, -filein	Input file with all fields to be written
[-fo, -fileout]	File to which the FDI EEPROM should be read back
[-lh, -longhelp]	Get long help description for program_ftdi util

Examples:

```
program_ftdi -write -ftdi FT2232H -serial 0ABC01 -vendor "my vendor co" -board  
"my board" -desc "my product desc"  
program_ftdi -write -filein <path_to_config_file>  
program_ftdi -read  
program_ftdi -read -fileout <path_to_config_file>  
program_ftdi -erase
```

 **Note:** Be sure to verify and update the Vendor, Board, Description, and Serial Number fields in the previous example before programming.

Configuration Memory Support

This section covers the various non-volatile device memories that are supported by AMD Vivado™ software. Use this section as a guide to select the appropriate configuration memory device for your application by AMD family, interface, manufacturer, density, and data width.

Artix 7 Configuration Memory Devices

The Flash devices supported for configuration of AMD Artix™ 7 devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in the following table.

The tables in this Appendix are running lists per AMD family of non-volatile memories which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain components on this list even after they are no longer appropriate for new designs, to support long-term maintenance of end products which can contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life cycle. References to specific devices in the tables are not an assurance of their current or future availability.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Table: Supported Flash Memory Devices for Artix 7 BPI

Manufacturer	Device Alias	MANUFACTURER	TYPE	DEVICE ID	DEVICE ID	Density	Mbit	Data Width Bits
Infineon s29glxxxp	s29gl01gp	1	227e	2228	2201	1024		x16, x8
Infineon s29glxxxp	s29gl128p	1	227e	2221	2201	128		x16, x8
Infineon s29glxxxp	s29gl256p	1	227e	2222	2201	256		x16, x8
Infineon s29glxxxp	s29gl512p	1	227e	2223	2201	512		x16, x8
Infineon s29glxxxp	s70gl02gp	1	227e	2248	2201	2048		x16
Infineon s29glxxxs	s29gl01gs	1	227e	2228	2201	1024		x16
Infineon s29glxxxs	s29gl128s	1	227e	2221	2201	128		x16
Infineon s29glxxxs	s29gl256s	1	227e	2222	2201	256		x16
Infineon s29glxxxs	s29gl512s	1	227e	2223	2201	512		x16
Infineon s29glxxxs	s70gl02gs	1	227e	2248	2201	2048		x16
Infineon s29glxxxt	s29gl01gt	1	227e	2228	2201	1024		x16, x8
Infineon s29glxxxt	s29gl512t	1	227e	2223	2201	512		x16, x8
Infineon s29glxxxt	s70gl02gt	1	227e	2248	2201	2048		x16, x8
Macronixmx29gl	mx29gl128f c2		227e	2221	2201	128		x16, x8
Macronixmx29gl	mx29gl256f c2		227e	2222	2201	256		x16, x8
Micron g18	28f128g18f 89		8900	-	-	128		x16
Micron g18	mt28gu01ga 89 1e [28f00ag18f- bpi-x16]		88b0	-	-	1024		x16

Manufacturer	Device ID	Alias	MANUFACTURE_DATE	MANUFACTURE_MONTH	MANUFACTURE_YEAR	DEVICE_ID	Density	Mbit	Data Width Bits
Micron	g18	mt28gu256a89e [28f256g18f-bpi-x16]	8901	-	-	256	x16		
Micron	g18	mt28gu512a89e [28f512g18f-bpi-x16]	887e	-	-	512	x16		
Micron	m29ew	28f00am29ew9	227e	2228	2201	1024	x16, x8		
Micron	m29ew	28f00bm29ew9	227e	2248	2201	2048	x16, x8		
Micron	m29ew	28f064m29ew9	227e	2210	2200	64	x16, x8		
Micron	m29ew	28f064m29ew9	227e	220c	2201	64	x16, x8		
Micron	m29ew	28f064m29ew9	227e	220c	2201	64	x16, x8		
Micron	m29ew	28f064m29ew9	227e	2210	2201	64	x16, x8		
Micron	m29ew	28f128m29ew9	227e	2221	2201	128	x16, x8		
Micron	m29ew	28f256m29ew9	227e	2222	2201	256	x16, x8		
Micron	m29ew	28f512m29ew9	227e	2223	2201	512	x16, x8		
Micron	m29w	m29w128gh 20	227e	2221	2201	128	x16, x8		
Micron	m29w	m29w128gl 20	227e	2221	2200	128	x16, x8		
Micron	m29w	m29w256gh 20	227e	2222	2201	256	x16, x8		
Micron	m29w	m29w256gl 20	227e	2222	2201	256	x16, x8		
Micron	m29w	m29w640gh 20	227e	220c	2201	64	x16, x8		
Micron	m29w	m29w640gl 20	227e	220c	2200	64	x16, x8		
Micron	mt28ew	mt28ew01ga89	227e	2228	2201	1024	x16, x8		
Micron	mt28ew	mt28ew128a89	227e	2221	2201	128	x16, x8		
Micron	mt28ew	mt28ew256a89	227e	2222	2201	256	x16, x8		
Micron	mt28ew	mt28ew512a89	227e	2223	2201	512	x16, x8		
Micron	mt28fw	mt28fw02gb 89	227e	2248	2201	2048	x16		
Micron	p30	28f00ap30b 89	8963	-	-	1024	x16		
Micron	p30	28f00ap30e 89	899a	-	-	1024	x16		

Manufacturer	Device ID	Memory Alias	MANUFACTURER	MEMORY ID	CAPACITY	Mbit	Data Width Bits
Micron	p30	28f00ap30t 89	8962	-	-	1024	x16
Micron	p30	28f00bp30e 89	899a	-	-	2048	x16
Micron	p30	28f128p30b 89	881b	-	-	128	x16
Micron	p30	28f128p30t 89	8818	-	-	128	x16
Micron	p30	28f256p30b 89	891c	-	-	256	x16
Micron	p30	28f256p30t 89	8919	-	-	256	x16
Micron	p30	28f512p30b 89	8961	-	-	512	x16
Micron	p30	28f512p30e 89	8999	-	-	512	x16
Micron	p30	28f512p30t 89	8960	-	-	512	x16
Micron	p30	28f640p30b 89	881a	-	-	64	x16
Micron	p30	28f640p30t 89	8817	-	-	64	x16
Micron	p33	28f00ap33b 89	8967	-	-	1024	x16
Micron	p33	28f00ap33e 89	899f	-	-	1024	x16
Micron	p33	28f00ap33t 89	8966	-	-	1024	x16
Micron	p33	28f128p33b 89	8821	-	-	128	x16
Micron	p33	28f128p33t 89	881e	-	-	128	x16
Micron	p33	28f256p33b 89	8922	-	-	256	x16
Micron	p33	28f256p33t 89	891f	-	-	256	x16
Micron	p33	28f512p33b 89	8965	-	-	512	x16
Micron	p33	28f512p33e 89	899e	-	-	512	x16
Micron	p33	28f512p33t 89	8964	-	-	512	x16
Micron	p33	28f640p33b 89	8820	-	-	64	x16
Micron	p33	28f640p33t 89	881d	-	-	64	x16

Table: Supported Flash Memory Devices for Artix 7 SPI

Manufacturer	Device ID	Memory Alias	MANUFACTURER	MEMORY ID	CAPACITY	Mbit	Data Width Bits
Infineon	s25fl1	s25fl116k	1	40	15	16	x1, x2, x4

Manufacturer	Manufacturer ID	Device Alias	MANUFACTURER_ID	MEMORY_ID	CAPACITY	DATA WIDTH	BITS
Infineon	s25fl1	s25fl132k	1	40	16	32	x1, x2, x4
Infineon	s25fl1	s25fl164k	1	40	17	64	x1, x2, x4
Infineon	s25flxxxl	s25fl064l	1	60	17	64	x1, x2, x4
Infineon	s25flxxxl	s25fl128l	1	60	18	128	x1, x2, x4
Infineon	s25flxxxl	s25fl256l	1	60	19	256	x1, x2, x4
Infineon	s25flxxxp	s25fl032p	1	2	15	32	x1, x2, x4
Infineon	s25flxxxp	s25fl064p	1	2	16	64	x1, x2, x4
Infineon	s25flxxxs	s25fl128sxxxxx1x0 [s25fl127s- spi- x1_x2_x4]		20	18	128	x1, x2, x4
Infineon	s25flxxxs	s25fl128sxxxxx1x1		20	18	128	x1, x2, x4
Infineon	s25flxxxs	s25fl256sxxxxx1x0		2	19	256	x1, x2, x4
Infineon	s25flxxxs	s25fl256sxxxxx1x1		2	19	256	x1, x2, x4
Infineon	s25flxxxs	s25fl512s	1	2	20	512	x1, x2, x4
Infineon	s25hs	s25hs02gt	-	-	-	2048	x1, x2, x4, x8
ISSI	is25l	is25lp01g	9d	60	1b	1024	x1, x2, x4, x8
ISSI	is25lp	is25lp016d	9d	60	15	16	x1, x2, x4
ISSI	is25lp	is25lp032d	9d	60	16	32	x1, x2, x4
ISSI	is25lp	is25lp064a	9d	60	17	64	x1, x2, x4
ISSI	is25lp	is25lp080d	9d	60	14	8	x1, x2, x4
ISSI	is25lp	is25lp128f	9d	60	18	128	x1, x2, x4
ISSI	is25lp	is25lp256d	9d	60	19	256	x1, x2, x4
ISSI	is25lp	is25lp512m	9d	60	1a	512	x1, x2, x4
ISSI	is25w	is25wp01g	9d	70	1b	1024	x1, x2, x4
ISSI	is25wp	is25wp016d	9d	70	15	16	x1, x2, x4

Manufacturer	Manufacturer ID	Device Alias	MANUFACTURER_ID	MEMORY_ID	CAPACITY	DATA WIDTH	BITS
ISSI	is25wp	is25wp032d	9d	70	16	32	x1, x2, x4
ISSI	is25wp	is25wp064a	9d	70	17	64	x1, x2, x4
ISSI	is25wp	is25wp080d	9d	70	14	8	x1, x2, x4
ISSI	is25wp	is25wp128f	9d	70	18	128	x1, x2, x4
ISSI	is25wp	is25wp256d	9d	70	19	256	x1, x2, x4
ISSI	is25wp	is25wp512m	9d	70	1a	512	x1, x2, x4
Macronix	mx25l	mx25l12872f [mx25l12833f- mx25l12835f- mx25l12845g- spi- x1_x2_x4, mx25l12833f- mx25l12835f- mx25l12845g- spi- x1_x2_x4_x8]	c2	20	18	128	x1, x2, x4, x8
Macronix	mx25l	mx25l25673g [mx25l25635f- mx25l25645g- spi- x1_x2_x4, mx25l25635f- mx25l25645g- spi- x1_x2_x4_x8]	c2	20	19	256	x1, x2, x4, x8
Macronix	mx25l	mx25l3273f [mx25l3233f- spi- x1_x2_x4]	c2	20	16	32	x1, x2, x4
Macronix	mx25l	mx25l51273g [mx25l51245g- mx66l51235f- spi- x1_x2_x4, mx25l51245g- mx66l51235f-	c2	20	1a	512	x1, x2, x4

Manufacturer	Manufacturer Alias	MANUFACTURE	MEMORY	MEMORY ID	CAPACITY	Data Width Bits	
		spi-[x1_x2_x4_x8]					
Macronix mx25l		mx25l6433f [mx25l6473f- spi-[x1_x2_x4]]	c2	20	17	64	x1, x2, x4
Macronix mx25l		mx25v1635f	c2	23	15	16	x1, x2, x4
Macronix mx25l		mx25v8035f	c2	23	14	8	x1, x2, x4
Macronix mx25lu		mx25u8035f [mx25u8033e- spi-[x1_x2_x4, mx25u8033e- spi-[x1_x2_x4_x8]]	c2	25	34	8	x1, x2, x4, x8
Macronix mx25u		mx25u12872f [mx25u12832f- mx25u12835f- mx25u12843g- spi-[x1_x2_x4, mx25u12832f- mx25u12835f- mx25u12843g- spi-[x1_x2_x4_x8]]	c2	25	38	128	x1, x2, x4
Macronix mx25u		mx25u1635f [mx25u1632f- spi-[x1_x2_x4]]	c2	25	35	16	x1, x2, x4
Macronix mx25u		mx25u25673g [mx25u25635f- mx25u25643g- mx25u25645g- spi-[x1_x2_x4, mx25u25635f- mx25u25643g- mx25u25645g-	c2	25	39	256	x1, x2, x4

Manufacturer	Manufacturer	Device Alias	MANUFACTURE	REMARKS	MEMORY IC	CAPACITY	DATA WIDTH	BITS
		spi-[x1_x2_x4_x8]						
Macronix	mx25u	mx25u3235f [mx25u3232f-spi-[x1_x2_x4]]	c2	25	36	32	x1, x2, x4	
Macronix	mx25u	mx25u51245g [mx66u51235f-spi-[x1_x2_x4]]	c2	25	3a	512	x1, x2, x4	
Macronix	mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-[x1_x2_x4, mx25u6435f-mx25u6432f-spi-[x1_x2_x4_x8]]]	c2	25	37	64	x1, x2, x4	
Macronix	mx66l	mx66l1g45g	c2	20	1b	1024	x1, x2, x4	
Macronix	mx66l	mx66l2g45g	c2	20	1c	2048	x1, x2, x4	
Macronix	mx66u	mx66u1g45g	c2	25	3b	1024	x1, x2, x4	
Macronix	mx66u	mx66u2g45g	c2	25	3c	2048	x1, x2, x4	
Micron	mt25ql	mt25ql01g	20	ba	21	1024	x1, x2, x4	
Micron	mt25ql	mt25ql02g	20	ba	22	2048	x1, x2, x4	
Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-[x1_x2_x4]]	20	ba	18	128	x1, x2, x4	
Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-[x1_x2_x4]]	20	ba	19	256	x1, x2, x4	
Micron	mt25ql	mt25ql512	20	ba	20	512	x1, x2, x4	
Micron	mt25qu	mt25qu01g	20	bb	21	1024	x1, x2, x4	

Manufacturer	Manufacturer Device Alias	MANUFACTURER DEVICE ID	MEMORY ID	CAPACITY	DATA WIDTH	BITS	
Micron	mt25qu	mt25qu02g	20	bb	22	2048	x1, x2, x4
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4]	20	bb	18	128	x1, x2, x4
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4]	20	bb	19	256	x1, x2, x4
Micron	mt25qu	mt25qu512	20	bb	20	512	x1, x2, x4
Micron	n25q	n25q32-1.8v	20	bb	16	32	x1, x2, x4
Micron	n25q	n25q32-3.3v	20	bb	16	32	x1, x2, x4
Micron	n25q	n25q64-1.8v	20	bb	17	64	x1, x2, x4
Micron	n25q	n25q64-3.3v	20	bb	17	64	x1, x2, x4

Kintex 7 Configuration Memory Devices

The Flash devices supported for configuration of AMD Kintex™ 7 devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in the following table. The tables in this Appendix are running lists per AMD family of non-volatile memories which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain components on this list even after they are no longer appropriate for new designs, to support long-term maintenance of end products which can contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life cycle. References to specific devices in the tables are not an assurance of their current or future availability.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Table: Supported Flash Memory Devices for Kintex 7 BPI

Manufacture	Manufacturer Device Alias	MANUFACTURER DEVICE ID	MEMORY ID	DEVICE_ID	DENSITY	MB	Data Width Bits

Manufacturer	Device	Device Alias	MANUFACTURE_DATE	MANUFACTURE_DATE	MANUFACTURE_DATE	DEVICE_ID	DDensity	Mb	Data Width Bits
Infineon	s29glxxxxp	s29gl01gp	1	227e	2228	2201	1024	x16, x8	
Infineon	s29glxxxxp	s29gl128p	1	227e	2221	2201	128	x16, x8	
Infineon	s29glxxxxp	s29gl256p	1	227e	2222	2201	256	x16, x8	
Infineon	s29glxxxxp	s29gl512p	1	227e	2223	2201	512	x16, x8	
Infineon	s29glxxxxp	s70gl02gp	1	227e	2248	2201	2048	x16	
Infineon	s29glxxxxs	s29gl01gs	1	227e	2228	2201	1024	x16	
Infineon	s29glxxxxs	s29gl128s	1	227e	2221	2201	128	x16	
Infineon	s29glxxxxs	s29gl256s	1	227e	2222	2201	256	x16	
Infineon	s29glxxxxs	s29gl512s	1	227e	2223	2201	512	x16	
Infineon	s29glxxxxs	s70gl02gs	1	227e	2248	2201	2048	x16	
Infineon	s29glxxxxt	s29gl01gt	1	227e	2228	2201	1024	x16, x8	
Infineon	s29glxxxxt	s29gl512t	1	227e	2223	2201	512	x16, x8	
Infineon	s29glxxxxt	s70gl02gt	1	227e	2248	2201	2048	x16, x8	
Macronix	mx29gl	mx29gl128f c2		227e	2221	2201	128	x16, x8	
Macronix	mx29gl	mx29gl256f c2		227e	2222	2201	256	x16, x8	
Micron	g18	28f128g18f 89		8900	-	-	128	x16	
Micron	g18	mt28gu01ga 89 1e [28f00ag18f-bpi-x16]		88b0	-	-	1024	x16	
Micron	g18	mt28gu256a 89 1e [28f256g18f-bpi-x16]		8901	-	-	256	x16	
Micron	g18	mt28gu512a 89 1e [28f512g18f-bpi-x16]		887e	-	-	512	x16	
Micron	m29ew	28f00am29e 89		227e	2228	2201	1024	x16, x8	
Micron	m29ew	28f00bm29e 89		227e	2248	2201	2048	x16, x8	
Micron	m29ew	28f064m29e 89		227e	2210	2200	64	x16, x8	
Micron	m29ew	28f064m29e 89		227e	220c	2201	64	x16, x8	

Manufacturer	Device Alias	MANUFACTURE_DATE	MANUFACTURE_MONTH	MANUFACTURE_YEAR	DEVICE_IDD	DEVICE_CAPACITY	DATA_WIDTH	BITS
Micron	m29ew	28f064m29ew89	227e	220c	2201	64	x16, x8	
Micron	m29ew	28f064m29ew89	227e	2210	2201	64	x16, x8	
Micron	m29ew	28f128m29ew89	227e	2221	2201	128	x16, x8	
Micron	m29ew	28f256m29ew89	227e	2222	2201	256	x16, x8	
Micron	m29ew	28f512m29ew89	227e	2223	2201	512	x16, x8	
Micron	m29w	m29w128gh 20	227e	2221	2201	128	x16, x8	
Micron	m29w	m29w128gl 20	227e	2221	2200	128	x16, x8	
Micron	m29w	m29w256gh 20	227e	2222	2201	256	x16, x8	
Micron	m29w	m29w256gl 20	227e	2222	2201	256	x16, x8	
Micron	m29w	m29w640gh 20	227e	220c	2201	64	x16, x8	
Micron	m29w	m29w640gl 20	227e	220c	2200	64	x16, x8	
Micron	mt28ew	mt28ew01ga89	227e	2228	2201	1024	x16, x8	
Micron	mt28ew	mt28ew128a89	227e	2221	2201	128	x16, x8	
Micron	mt28ew	mt28ew256a89	227e	2222	2201	256	x16, x8	
Micron	mt28ew	mt28ew512a89	227e	2223	2201	512	x16, x8	
Micron	mt28fw	mt28fw02gb 89	227e	2248	2201	2048	x16	
Micron	p30	28f00ap30b 89	8963	-	-	1024	x16	
Micron	p30	28f00ap30e 89	899a	-	-	1024	x16	
Micron	p30	28f00ap30t 89	8962	-	-	1024	x16	
Micron	p30	28f00bp30e 89	899a	-	-	2048	x16	
Micron	p30	28f128p30b 89	881b	-	-	128	x16	
Micron	p30	28f128p30t 89	8818	-	-	128	x16	
Micron	p30	28f256p30b 89	891c	-	-	256	x16	
Micron	p30	28f256p30t 89	8919	-	-	256	x16	
Micron	p30	28f512p30b 89	8961	-	-	512	x16	
Micron	p30	28f512p30e 89	8999	-	-	512	x16	

Manufacturer	Manufacturer	Device Family	Alias	MANUFACTURER_ID	ROM_ID	RAM_ID	FLASH_ID	DEVICE_ID	IDD	Capacity Mb	Data Width Bits
Micron	p30	28f512p30t	89	8960	-	-	-	512		x16	
Micron	p30	28f640p30b	89	881a	-	-	-	64		x16	
Micron	p30	28f640p30t	89	8817	-	-	-	64		x16	
Micron	p33	28f00ap33b	89	8967	-	-	-	1024		x16	
Micron	p33	28f00ap33e	89	899f	-	-	-	1024		x16	
Micron	p33	28f00ap33t	89	8966	-	-	-	1024		x16	
Micron	p33	28f128p33b	89	8821	-	-	-	128		x16	
Micron	p33	28f128p33t	89	881e	-	-	-	128		x16	
Micron	p33	28f256p33b	89	8922	-	-	-	256		x16	
Micron	p33	28f256p33t	89	891f	-	-	-	256		x16	
Micron	p33	28f512p33b	89	8965	-	-	-	512		x16	
Micron	p33	28f512p33e	89	899e	-	-	-	512		x16	
Micron	p33	28f512p33t	89	8964	-	-	-	512		x16	
Micron	p33	28f640p33b	89	8820	-	-	-	64		x16	
Micron	p33	28f640p33t	89	881d	-	-	-	64		x16	

Table: Supported Flash Memory Devices for Kintex 7 SPI

Manufacturer	Manufacturer	Device Family	Alias	MANUFACTURER_ID	ROM_ID	RAM_ID	FLASH_ID	DEVICE_ID	IDD	Capacity Mb	Data Width Bits
Infineon	s25fl1	s25fl116k	1	40	15			16		x1, x2, x4	
Infineon	s25fl1	s25fl132k	1	40	16			32		x1, x2, x4	
Infineon	s25fl1	s25fl164k	1	40	17			64		x1, x2, x4	
Infineon	s25flxxxl	s25fl064l	1	60	17			64		x1, x2, x4	
Infineon	s25flxxxl	s25fl128l	1	60	18			128		x1, x2, x4	
Infineon	s25flxxxl	s25fl256l	1	60	19			256		x1, x2, x4	
Infineon	s25flxxxp	s25fl032p	1	2	15			32		x1, x2, x4	
Infineon	s25flxxxp	s25fl064p	1	2	16			64		x1, x2, x4	
Infineon	s25flxxxs	s25fl128sxxxxx0 [s25fl127s-		20	18			128		x1, x2, x4	

Manufacturer	Manufacturer Family	Alias	MANUFACTURE_DATE	REVISION	MEMORY_TYPE	DATA_WIDTH	Bits
		spi-[x1_x2_x4]					
Infineon	s25flxxxxs	s25fl128sxxxx[x1		20	18	128	x1, x2, x4
Infineon	s25flxxxxs	s25fl256sxxxx[x0		2	19	256	x1, x2, x4
Infineon	s25flxxxxs	s25fl256sxxxx[x1		2	19	256	x1, x2, x4
Infineon	s25flxxxxs	s25fl512s	1	2	20	512	x1, x2, x4
Infineon	s25hs	s25hs02gt	-	-	-	2048	x1, x2, x4, x8
ISSI	is25l	is25lp01g	9d	60	1b	1024	x1, x2, x4, x8
ISSI	is25lp	is25lp016d	9d	60	15	16	x1, x2, x4
ISSI	is25lp	is25lp032d	9d	60	16	32	x1, x2, x4
ISSI	is25lp	is25lp064a	9d	60	17	64	x1, x2, x4
ISSI	is25lp	is25lp080d	9d	60	14	8	x1, x2, x4
ISSI	is25lp	is25lp128f	9d	60	18	128	x1, x2, x4
ISSI	is25lp	is25lp256d	9d	60	19	256	x1, x2, x4
ISSI	is25lp	is25lp512m	9d	60	1a	512	x1, x2, x4
ISSI	is25w	is25wp01g	9d	70	1b	1024	x1, x2, x4
ISSI	is25wp	is25wp016d	9d	70	15	16	x1, x2, x4
ISSI	is25wp	is25wp032d	9d	70	16	32	x1, x2, x4
ISSI	is25wp	is25wp064a	9d	70	17	64	x1, x2, x4
ISSI	is25wp	is25wp080d	9d	70	14	8	x1, x2, x4
ISSI	is25wp	is25wp128f	9d	70	18	128	x1, x2, x4
ISSI	is25wp	is25wp256d	9d	70	19	256	x1, x2, x4
ISSI	is25wp	is25wp512m	9d	70	1a	512	x1, x2, x4
Macronix	mx25l	mx25l12872f c2 [mx25l12833f- mx25l12835f- mx25l12845g-		20	18	128	x1, x2, x4, x8

Manufacturer	Manufacturer Family	Alias	MANUFACTURE_DATE	MEMORY_TYPE	DATA_CAPACITY	VID	Data Width Bits
		spi-x1_x2_x4, mx25l12833f- mx25l12835f- mx25l12845g- spi-x1_x2_x4_x8]					
Macronix mx25l		mx25l25673g c2 [mx25l25635f- mx25l25645g- spi-x1_x2_x4, mx25l25635f- mx25l25645g- spi-x1_x2_x4_x8]	20	19	256		x1, x2, x4, x8
Macronix mx25l		mx25l3273f c2 [mx25l3233f- spi-x1_x2_x4]	20	16	32		x1, x2, x4
Macronix mx25l		mx25l51273g c2 [mx25l51245g- mx66l51235f- spi-x1_x2_x4, mx25l51245g- mx66l51235f- spi-x1_x2_x4_x8]	20	1a	512		x1, x2, x4
Macronix mx25l		mx25l6433f c2 [mx25l6473f- spi-x1_x2_x4]	20	17	64		x1, x2, x4
Macronix mx25l		mx25v1635f c2	23	15	16		x1, x2, x4
Macronix mx25l		mx25v8035f c2	23	14	8		x1, x2, x4
Macronix mx25lu		mx25u8035f c2 [mx25u8033e- spi-x1_x2_x4,	25	34	8		x1, x2, x4, x8

Manufacturer	Manufacturer Family	Alias	MANUFACTURE_DATE	MEMORY_TYPE	DATA_CAPACITY	VID	Data Width Bits
		mx25u8033e- spi- x1_x2_x4_x8]					
Macronix mx25u		mx25u12872fc2 [mx25u12832f- mx25u12835f- mx25u12843g- spi- x1_x2_x4, mx25u12832f- mx25u12835f- mx25u12843g- spi- x1_x2_x4_x8]	25	38	128		x1, x2, x4
Macronix mx25u		mx25u1635f_c2 [mx25u1632f- spi- x1_x2_x4]	25	35	16		x1, x2, x4
Macronix mx25u		mx25u25673gc2 [mx25u25635f- mx25u25643g- mx25u25645g- spi- x1_x2_x4, mx25u25635f- mx25u25643g- mx25u25645g- spi- x1_x2_x4_x8]	25	39	256		x1, x2, x4
Macronix mx25u		mx25u3235f_c2 [mx25u3232f- spi- x1_x2_x4]	25	36	32		x1, x2, x4
Macronix mx25u		mx25u51245gc2 [mx66u51235f- spi- x1_x2_x4]	25	3a	512		x1, x2, x4
Macronix mx25u		mx25u6472fc2 [mx25u6435f-	25	37	64		x1, x2, x4

Manufacturer	Manufacturer Family	Alias	MANUFACTURE_DATE	REVISION	MEMORY_TYPE	DATA_CAPACITY	VID	Data Width Bits
		mx25u6432f-[spi- x1_x2_x4, mx25u6435f- mx25u6432f- spi- x1_x2_x4_x8]						
Macronix	mx66l	mx66l1g45g	c2	20	1b	1024		x1, x2, x4
Macronix	mx66l	mx66l2g45g	c2	20	1c	2048		x1, x2, x4
Macronix	mx66u	mx66u1g45g	c2	25	3b	1024		x1, x2, x4
Macronix	mx66u	mx66u2g45g	c2	25	3c	2048		x1, x2, x4
Micron	mt25ql	mt25ql01g	20	ba	21	1024		x1, x2, x4
Micron	mt25ql	mt25ql02g	20	ba	22	2048		x1, x2, x4
Micron	mt25ql	mt25ql128 [n25q128- 3.3v-spi- x1_x2_x4]	20	ba	18	128		x1, x2, x4
Micron	mt25ql	mt25ql256 [n25q256- 3.3v-spi- x1_x2_x4]	20	ba	19	256		x1, x2, x4
Micron	mt25ql	mt25ql512	20	ba	20	512		x1, x2, x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024		x1, x2, x4
Micron	mt25qu	mt25qu02g	20	bb	22	2048		x1, x2, x4
Micron	mt25qu	mt25qu128 [n25q128- 1.8v-spi- x1_x2_x4]	20	bb	18	128		x1, x2, x4
Micron	mt25qu	mt25qu256 [n25q256- 1.8v-spi- x1_x2_x4]	20	bb	19	256		x1, x2, x4
Micron	mt25qu	mt25qu512	20	bb	20	512		x1, x2, x4

Manufacturer	Manufacturer Device ID	Device Alias	MANUFACTURER_ID	READ_ID	WRITE_ID	DEVICE_ID	IDDENSITY	MB	Data Width Bits
Micron	n25q	n25q32-1.8v	20	bb	16		32		x1, x2, x4
Micron	n25q	n25q32-3.3v	20	bb	16		32		x1, x2, x4
Micron	n25q	n25q64-1.8v	20	bb	17		64		x1, x2, x4
Micron	n25q	n25q64-3.3v	20	bb	17		64		x1, x2, x4

Spartan 7 Configuration Memory Devices

The Flash devices supported for configuration of Spartan™ 7 devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in the following table. The tables in this Appendix are running lists per AMD family of non-volatile memories which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain components on this list even after they are no longer appropriate for new designs, to support long-term maintenance of end products that can contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life cycle. References to specific devices in the tables are not an assurance of their current or future availability.

 **Note:** Serial peripheral interface (SPI) flash is the supported configuration memory storage for Spartan 7 devices. Byte-wide Peripheral Interface (BPI) flash is not supported for Spartan 7 devices.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Table: Supported Flash Memory Devices for Spartan 7 BPI Device Configuration

Manufacturer	Manufacturer Device ID	Device Alias	MANUFACTURER_ID	READ_ID	WRITE_ID	DEVICE_ID	IDDENSITY	MB	Data Width Bits
Infineon s29glxxxp	s29gl01gp	1	227e	2228		2201	1024		x16, x8
Infineon s29glxxxp	s29gl128p	1	227e	2221		2201	128		x16, x8
Infineon s29glxxxp	s29gl256p	1	227e	2222		2201	256		x16, x8
Infineon s29glxxxp	s29gl512p	1	227e	2223		2201	512		x16, x8
Infineon s29glxxxp	s70gl02gp	1	227e	2248		2201	2048		x16
Infineon s29glxxxs	s29gl01gs	1	227e	2228		2201	1024		x16
Infineon s29glxxxs	s29gl128s	1	227e	2221		2201	128		x16

Manufacturer	Device Alias	MANUFACTURE_DATE	MANUFACTURE_MONTH	MANUFACTURE_YEAR	DEVICE_IDD	DEVICE_IDD_V	DENSITY_Mb	Data Width Bits
Infineon	s29glxxxxs	s29gl256s	1	227e	2222	2201	256	x16
Infineon	s29glxxxxs	s29gl512s	1	227e	2223	2201	512	x16
Infineon	s29glxxxxs	s70gl02gs	1	227e	2248	2201	2048	x16
Infineon	s29glxxxxt	s29gl01gt	1	227e	2228	2201	1024	x16, x8
Infineon	s29glxxxxt	s29gl512t	1	227e	2223	2201	512	x16, x8
Infineon	s29glxxxxt	s70gl02gt	1	227e	2248	2201	2048	x16, x8
Macronix	mx29gl	mx29gl128f	c2	227e	2221	2201	128	x16, x8
Macronix	mx29gl	mx29gl256f	c2	227e	2222	2201	256	x16, x8
Micron	g18	28f128g18f	89	8900	-	-	128	x16
Micron	g18	mt28gu01ga 89 1e [28f00ag18f-bpi-x16]		88b0	-	-	1024	x16
Micron	g18	mt28gu256a 89 1e [28f256g18f-bpi-x16]		8901	-	-	256	x16
Micron	g18	mt28gu512a 89 1e [28f512g18f-bpi-x16]		887e	-	-	512	x16
Micron	m29ew	28f00am29e 89 9		227e	2228	2201	1024	x16, x8
Micron	m29ew	28f00bm29e 89 9		227e	2248	2201	2048	x16, x8
Micron	m29ew	28f064m29e 89 9		227e	2210	2200	64	x16, x8
Micron	m29ew	28f064m29e 89 9		227e	220c	2201	64	x16, x8
Micron	m29ew	28f064m29e 89 9		227e	220c	2201	64	x16, x8
Micron	m29ew	28f064m29e 89 9		227e	2210	2201	64	x16, x8
Micron	m29ew	28f128m29e 89 9		227e	2221	2201	128	x16, x8
Micron	m29ew	28f256m29e 89 9		227e	2222	2201	256	x16, x8
Micron	m29ew	28f512m29e 89 9		227e	2223	2201	512	x16, x8
Micron	m29w	m29w128gh	20	227e	2221	2201	128	x16, x8
Micron	m29w	m29w128gl	20	227e	2221	2200	128	x16, x8

Manufacturer	Device	Device Alias	MANUFACTURE_DATE	MANUFACTURE_MONTH	MANUFACTURE_YEAR	DEVICE_ID	DEVICE_ID2	DEVICE_ID3	DDensity	Mb	Data Width Bits
Micron	m29w	m29w256gh 20	227e	2222		2201		256		x16, x8	
Micron	m29w	m29w256gl 20	227e	2222		2201		256		x16, x8	
Micron	m29w	m29w640gh 20	227e	220c		2201		64		x16, x8	
Micron	m29w	m29w640gl 20	227e	220c		2200		64		x16, x8	
Micron	mt28ew	mt28ew01ga89	227e	2228		2201		1024		x16, x8	
Micron	mt28ew	mt28ew128a89	227e	2221		2201		128		x16, x8	
Micron	mt28ew	mt28ew256a89	227e	2222		2201		256		x16, x8	
Micron	mt28ew	mt28ew512a89	227e	2223		2201		512		x16, x8	
Micron	mt28fw	mt28fw02gb 89	227e	2248		2201		2048		x16	
Micron	p30	28f00ap30b 89	8963	-		-		1024		x16	
Micron	p30	28f00ap30e 89	899a	-		-		1024		x16	
Micron	p30	28f00ap30t 89	8962	-		-		1024		x16	
Micron	p30	28f00bp30e 89	899a	-		-		2048		x16	
Micron	p30	28f128p30b 89	881b	-		-		128		x16	
Micron	p30	28f128p30t 89	8818	-		-		128		x16	
Micron	p30	28f256p30b 89	891c	-		-		256		x16	
Micron	p30	28f256p30t 89	8919	-		-		256		x16	
Micron	p30	28f512p30b 89	8961	-		-		512		x16	
Micron	p30	28f512p30e 89	8999	-		-		512		x16	
Micron	p30	28f512p30t 89	8960	-		-		512		x16	
Micron	p30	28f640p30b 89	881a	-		-		64		x16	
Micron	p30	28f640p30t 89	8817	-		-		64		x16	
Micron	p33	28f00ap33b 89	8967	-		-		1024		x16	
Micron	p33	28f00ap33e 89	899f	-		-		1024		x16	
Micron	p33	28f00ap33t 89	8966	-		-		1024		x16	
Micron	p33	28f128p33b 89	8821	-		-		128		x16	

Manufacturer	Device Family	Device Alias	MANUFACTURER_ID	ROM_ID	FLASH_ID	DEVICE_ID	IDD	Capacity Mb	Data Width Bits
Micron	p33	28f128p33t 89	881e	-	-	-	128	x16	
Micron	p33	28f256p33b 89	8922	-	-	-	256	x16	
Micron	p33	28f256p33t 89	891f	-	-	-	256	x16	
Micron	p33	28f512p33b 89	8965	-	-	-	512	x16	
Micron	p33	28f512p33e 89	899e	-	-	-	512	x16	
Micron	p33	28f512p33t 89	8964	-	-	-	512	x16	
Micron	p33	28f640p33b 89	8820	-	-	-	64	x16	
Micron	p33	28f640p33t 89	881d	-	-	-	64	x16	

Table: Supported Flash Memory Devices for Spartan 7 SPI Device Configuration

Manufacturer	Device Family	Device Alias	MANUFACTURER_ID	ROM_ID	FLASH_ID	DEVICE_ID	IDD	Capacity Mb	Data Width Bits
Infineon	s25fl1	s25fl116k	1	40	15	16		x1, x2, x4	
Infineon	s25fl1	s25fl132k	1	40	16	32		x1, x2, x4	
Infineon	s25fl1	s25fl164k	1	40	17	64		x1, x2, x4	
Infineon	s25flxxxl	s25fl064l	1	60	17	64		x1, x2, x4	
Infineon	s25flxxxl	s25fl128l	1	60	18	128		x1, x2, x4	
Infineon	s25flxxxl	s25fl256l	1	60	19	256		x1, x2, x4	
Infineon	s25flxxxp	s25fl032p	1	2	15	32		x1, x2, x4	
Infineon	s25flxxxp	s25fl064p	1	2	16	64		x1, x2, x4	
Infineon	s25flxxxs	s25fl128sxxxxx0 [s25fl127s- spi- x1_x2_x4]		20	18	128		x1, x2, x4	
Infineon	s25flxxxs	s25fl128sxxxxx1		20	18	128		x1, x2, x4	
Infineon	s25flxxxs	s25fl256sxxxxx0		2	19	256		x1, x2, x4	
Infineon	s25flxxxs	s25fl256sxxxxx1		2	19	256		x1, x2, x4	
Infineon	s25flxxxs	s25fl512s	1	2	20	512		x1, x2, x4	
Infineon	s25hs	s25hs02gt	-	-	-	2048		x1, x2, x4, x8	

Manufacturer	Manufacturer ID	Family	Alias	MANUFACTURE_DATE	REV	MEMORY_TYPE	DATA_CAPACITY	VID	Data Width Bits
ISSI	is25l	is25lp01g	9d	60	1b		1024		x1, x2, x4, x8
ISSI	is25lp	is25lp016d	9d	60	15		16		x1, x2, x4
ISSI	is25lp	is25lp032d	9d	60	16		32		x1, x2, x4
ISSI	is25lp	is25lp064a	9d	60	17		64		x1, x2, x4
ISSI	is25lp	is25lp080d	9d	60	14		8		x1, x2, x4
ISSI	is25lp	is25lp128f	9d	60	18		128		x1, x2, x4
ISSI	is25lp	is25lp256d	9d	60	19		256		x1, x2, x4
ISSI	is25lp	is25lp512m	9d	60	1a		512		x1, x2, x4
ISSI	is25w	is25wp01g	9d	70	1b		1024		x1, x2, x4
ISSI	is25wp	is25wp016d	9d	70	15		16		x1, x2, x4
ISSI	is25wp	is25wp032d	9d	70	16		32		x1, x2, x4
ISSI	is25wp	is25wp064a	9d	70	17		64		x1, x2, x4
ISSI	is25wp	is25wp080d	9d	70	14		8		x1, x2, x4
ISSI	is25wp	is25wp128f	9d	70	18		128		x1, x2, x4
ISSI	is25wp	is25wp256d	9d	70	19		256		x1, x2, x4
ISSI	is25wp	is25wp512m	9d	70	1a		512		x1, x2, x4
Macronix	mx25l	mx25l12872f c2 [mx25l12833f- mx25l12835f- mx25l12845g- spi- x1_x2_x4, mx25l12833f- mx25l12835f- mx25l12845g- spi- x1_x2_x4_x8]		20	18		128		x1, x2, x4, x8
Macronix	mx25l	mx25l25673g c2 [mx25l25635f- mx25l25645g- spi-		20	19		256		x1, x2, x4, x8

Manufacturer	Manufacturer Family	Alias	MANUFACTURE_DATE	MEMORY_TYPE	DATA_CAPACITY	VID	Data Width Bits
		x1_x2_x4, mx25l25635f- mx25l25645g- spi- x1_x2_x4_x8]					
Macronix mx25l	mx25l3273f	c2 [mx25l3233f- spi- x1_x2_x4]	20	16	32		x1, x2, x4
Macronix mx25l	mx25l51273g	c2 [mx25l51245g- mx66l51235f- spi- x1_x2_x4, mx25l51245g- mx66l51235f- spi- x1_x2_x4_x8]	20	1a	512		x1, x2, x4
Macronix mx25l	mx25l6433f	c2 [mx25l6473f- spi- x1_x2_x4]	20	17	64		x1, x2, x4
Macronix mx25l	mx25v1635f	c2	23	15	16		x1, x2, x4
Macronix mx25l	mx25v8035f	c2	23	14	8		x1, x2, x4
Macronix mx25lu	mx25u8035f	c2 [mx25u8033e- spi- x1_x2_x4, mx25u8033e- spi- x1_x2_x4_x8]	25	34	8		x1, x2, x4, x8
Macronix mx25u	mx25u12872fc2		25	38	128		x1, x2, x4
	[mx25u12832f- mx25u12835f- mx25u12843g- spi- x1_x2_x4, mx25u12832f- mx25u12835f- spi- x1_x2_x4_x8]						

Manufacturer	Manufacturer Family	Alias	MANUFACTURE_DATE	REVISION	MEMORY_TYPE	DATA_CAPACITY	VID	Data Width Bits
		mx25u12843g- spi- x1_x2_x4_x8]						
Macronix mx25u		mx25u1635f c2 [mx25u1632f- spi- x1_x2_x4]	25	35	16			x1, x2, x4
Macronix mx25u		mx25u25673gc2 [mx25u25635f- mx25u25643g- mx25u25645g- spi- x1_x2_x4, mx25u25635f- mx25u25643g- mx25u25645g- spi- x1_x2_x4_x8]	25	39	256			x1, x2, x4
Macronix mx25u		mx25u3235f c2 [mx25u3232f- spi- x1_x2_x4]	25	36	32			x1, x2, x4
Macronix mx25u		mx25u51245gc2 [mx66u51235f- spi- x1_x2_x4]	25	3a	512			x1, x2, x4
Macronix mx25u		mx25u6472f c2 [mx25u6435f- mx25u6432f- spi- x1_x2_x4, mx25u6435f- mx25u6432f- spi- x1_x2_x4_x8]	25	37	64			x1, x2, x4
Macronix mx66l		mx66l1g45g c2	20	1b	1024			x1, x2, x4
Macronix mx66l		mx66l2g45g c2	20	1c	2048			x1, x2, x4
Macronix mx66u		mx66u1g45g c2	25	3b	1024			x1, x2, x4

Manufacturer	Manufacturer ID	Device ID	Alias	MANUFACTURE_DATE	REVISION	ROM_BY_ID	DATA_BY_ID	Yield	Data Width Bits
Macronix	mx66u	mx66u2g45g	c2		25	3c	2048		x1, x2, x4
Micron	mt25ql	mt25ql01g	20		ba	21	1024		x1, x2, x4
Micron	mt25ql	mt25ql02g	20		ba	22	2048		x1, x2, x4
Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4]	20		ba	18	128		x1, x2, x4
Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4]	20		ba	19	256		x1, x2, x4
Micron	mt25ql	mt25ql512	20		ba	20	512		x1, x2, x4
Micron	mt25qu	mt25qu01g	20		bb	21	1024		x1, x2, x4
Micron	mt25qu	mt25qu02g	20		bb	22	2048		x1, x2, x4
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4]	20		bb	18	128		x1, x2, x4
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4]	20		bb	19	256		x1, x2, x4
Micron	mt25qu	mt25qu512	20		bb	20	512		x1, x2, x4
Micron	n25q	n25q32-1.8v	20		bb	16	32		x1, x2, x4
Micron	n25q	n25q32-3.3v	20		bb	16	32		x1, x2, x4
Micron	n25q	n25q64-1.8v	20		bb	17	64		x1, x2, x4
Micron	n25q	n25q64-3.3v	20		bb	17	64		x1, x2, x4

Virtex 7 Configuration Memory Devices

The Flash devices supported for configuration of AMD Virtex™ 7 devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in the following table.

The tables in this Appendix are running lists per AMD family of non-volatile memories which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain components on this list even after they are no longer appropriate for new designs, to support long-term maintenance of end products which can contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life cycle. References to specific devices in the tables are not an assurance of their current or future availability.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Table: Supported Flash Memory Devices for Virtex 7 BPI Device Configuration

Manufacturer	Device Alias	MANUFACTURE_ID	TYPE_ID	DEVICE_ID	IDD	Density	Mb	Data Width Bits
Infineon s29glxxxp	s29gl01gp	1	227e	2228	2201	1024		x16, x8
Infineon s29glxxxp	s29gl128p	1	227e	2221	2201	128		x16, x8
Infineon s29glxxxp	s29gl256p	1	227e	2222	2201	256		x16, x8
Infineon s29glxxxp	s29gl512p	1	227e	2223	2201	512		x16, x8
Infineon s29glxxxp	s70gl02gp	1	227e	2248	2201	2048		x16
Infineon s29glxxxs	s29gl01gs	1	227e	2228	2201	1024		x16
Infineon s29glxxxs	s29gl128s	1	227e	2221	2201	128		x16
Infineon s29glxxxs	s29gl256s	1	227e	2222	2201	256		x16
Infineon s29glxxxs	s29gl512s	1	227e	2223	2201	512		x16
Infineon s29glxxxs	s70gl02gs	1	227e	2248	2201	2048		x16
Infineon s29glxxxt	s29gl01gt	1	227e	2228	2201	1024		x16, x8
Infineon s29glxxxt	s29gl512t	1	227e	2223	2201	512		x16, x8
Infineon s29glxxxt	s70gl02gt	1	227e	2248	2201	2048		x16, x8
Macronixmx29gl	mx29gl128f c2		227e	2221	2201	128		x16, x8
Macronixmx29gl	mx29gl256f c2		227e	2222	2201	256		x16, x8
Micron g18	28f128g18f 89	8900	-	-	-	128		x16

Manufacturer	Device	Device Alias	MANUFACTURE_DATE	MANUFACTURE_MONTH	MANUFACTURE_YEAR	DEVICE_ID	DEVICE_IDD	Density	Mb	Data Width Bits
Micron	g18	mt28gu01ga89 [28f00ag18f-bpi-x16]	88b0	-	-	-	-	1024	x16	
Micron	g18	mt28gu256a89 [28f256g18f-bpi-x16]	8901	-	-	-	-	256	x16	
Micron	g18	mt28gu512a89 [28f512g18f-bpi-x16]	887e	-	-	-	-	512	x16	
Micron	m29ew	28f00am29ew89	227e	2228	2201	2201	2201	1024	x16, x8	
Micron	m29ew	28f00bm29ew89	227e	2248	2201	2201	2201	2048	x16, x8	
Micron	m29ew	28f064m29ew89	227e	2210	2200	2200	2200	64	x16, x8	
Micron	m29ew	28f064m29ew89	227e	220c	2201	2201	2201	64	x16, x8	
Micron	m29ew	28f064m29ew89	227e	220c	2201	2201	2201	64	x16, x8	
Micron	m29ew	28f064m29ew89	227e	2210	2201	2201	2201	64	x16, x8	
Micron	m29ew	28f128m29ew89	227e	2221	2201	2201	2201	128	x16, x8	
Micron	m29ew	28f256m29ew89	227e	2222	2201	2201	2201	256	x16, x8	
Micron	m29ew	28f512m29ew89	227e	2223	2201	2201	2201	512	x16, x8	
Micron	m29w	m29w128gh20	227e	2221	2201	2201	2201	128	x16, x8	
Micron	m29w	m29w128gl20	227e	2221	2200	2200	2200	128	x16, x8	
Micron	m29w	m29w256gh20	227e	2222	2201	2201	2201	256	x16, x8	
Micron	m29w	m29w256gl20	227e	2222	2201	2201	2201	256	x16, x8	
Micron	m29w	m29w640gh20	227e	220c	2201	2201	2201	64	x16, x8	
Micron	m29w	m29w640gl20	227e	220c	2200	2200	2200	64	x16, x8	
Micron	mt28ew	mt28ew01ga89	227e	2228	2201	2201	2201	1024	x16, x8	
Micron	mt28ew	mt28ew128a89	227e	2221	2201	2201	2201	128	x16, x8	
Micron	mt28ew	mt28ew256a89	227e	2222	2201	2201	2201	256	x16, x8	
Micron	mt28ew	mt28ew512a89	227e	2223	2201	2201	2201	512	x16, x8	
Micron	mt28fw	mt28fw02gb89	227e	2248	2201	2201	2201	2048	x16	

Manufacturer	Device	Device Alias	MANUFACTURE_ID	PROGRAMMING_ID	DEVICE_ID	IDD	Density	MB	Data Width Bits
Micron	p30	28f00ap30b 89	8963	-	-	1024	x16		
Micron	p30	28f00ap30e 89	899a	-	-	1024	x16		
Micron	p30	28f00ap30t 89	8962	-	-	1024	x16		
Micron	p30	28f00bp30e 89	899a	-	-	2048	x16		
Micron	p30	28f128p30b 89	881b	-	-	128	x16		
Micron	p30	28f128p30t 89	8818	-	-	128	x16		
Micron	p30	28f256p30b 89	891c	-	-	256	x16		
Micron	p30	28f256p30t 89	8919	-	-	256	x16		
Micron	p30	28f512p30b 89	8961	-	-	512	x16		
Micron	p30	28f512p30e 89	8999	-	-	512	x16		
Micron	p30	28f512p30t 89	8960	-	-	512	x16		
Micron	p30	28f640p30b 89	881a	-	-	64	x16		
Micron	p30	28f640p30t 89	8817	-	-	64	x16		
Micron	p33	28f00ap33b 89	8967	-	-	1024	x16		
Micron	p33	28f00ap33e 89	899f	-	-	1024	x16		
Micron	p33	28f00ap33t 89	8966	-	-	1024	x16		
Micron	p33	28f128p33b 89	8821	-	-	128	x16		
Micron	p33	28f128p33t 89	881e	-	-	128	x16		
Micron	p33	28f256p33b 89	8922	-	-	256	x16		
Micron	p33	28f256p33t 89	891f	-	-	256	x16		
Micron	p33	28f512p33b 89	8965	-	-	512	x16		
Micron	p33	28f512p33e 89	899e	-	-	512	x16		
Micron	p33	28f512p33t 89	8964	-	-	512	x16		
Micron	p33	28f640p33b 89	8820	-	-	64	x16		
Micron	p33	28f640p33t 89	881d	-	-	64	x16		

Table: Supported Flash Memory Devices for Virtex 7 SPI Device Configuration

Manufacturer	Manufacturer	Family	Device Alias	Density Mb	Data Width Bits
Infineon	s25fl1	s25fl116k		16	x1, x2, x4
Infineon	s25fl1	s25fl132k		32	x1, x2, x4
Infineon	s25fl1	s25fl164k		64	x1, x2, x4
Infineon	s25flxxxl	s25fl064l		64	x1, x2, x4
Infineon	s25flxxxl	s25fl128l		128	x1, x2, x4
Infineon	s25flxxxl	s25fl256l		256	x1, x2, x4
Infineon	s25flxxxp	s25fl032p		32	x1, x2, x4
Infineon	s25flxxxp	s25fl064p		64	x1, x2, x4
Infineon	s25flxxxs	s25fl128xxxxxxxx0 [s25fl127s-spi-x1_x2_x4]		128	x1, x2, x4
Infineon	s25flxxxs	s25fl128xxxxxxxx1		128	x1, x2, x4
Infineon	s25flxxxs	s25fl256xxxxxxxx0		256	x1, x2, x4
Infineon	s25flxxxs	s25fl256xxxxxxxx1		256	x1, x2, x4
Infineon	s25flxxxs	s25fl512s		512	x1, x2, x4
Infineon	s25hs	s25hs02gt		2048	x1, x2, x4, x8
ISSI	is25l	is25lp01g		1024	x1, x2, x4
ISSI	is25lp	is25lp016d		16	x1, x2, x4

Manufacturer	Manufacturer	Family	Device Alias	Density Mb	Data Width Bits
ISSI	is25lp	is25lp032d		32	x1, x2, x4
ISSI	is25lp	is25lp064a		64	x1, x2, x4
ISSI	is25lp	is25lp080d		8	x1, x2, x4
ISSI	is25lp	is25lp128f		128	x1, x2, x4
ISSI	is25lp	is25lp256d		256	x1, x2, x4
ISSI	is25lp	is25lp512m		512	x1, x2, x4
ISSI	is25w	is25wp01g		1024	x1, x2, x4
ISSI	is25wp	is25wp016d		16	x1, x2, x4
ISSI	is25wp	is25wp032d		32	x1, x2, x4
ISSI	is25wp	is25wp064a		64	x1, x2, x4
ISSI	is25wp	is25wp080d		8	x1, x2, x4
ISSI	is25wp	is25wp128f		128	x1, x2, x4
ISSI	is25wp	is25wp256d		256	x1, x2, x4
ISSI	is25wp	is25wp512m		512	x1, x2, x4
Macronix	mx25l	mx25l12872f [mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4, mx25l12833f-mx25l12835f-mx25l12845g-spi-x1_x2_x4_x8]		128	x1, x2, x4

Manufacturer	Manufacturer	Family	Device Alias	Density	Mb	Data Width Bits
Macronix	mx25l	mx25l25673g [mx25l25635f-mx25l25645g-spi-x1_x2_x4, mx25l25635f-mx25l25645g-spi-x1_x2_x4_x8]	256		x1, x2, x4	
Macronix	mx25l	mx25l3273f [mx25l3233f-spi-x1_x2_x4]	32		x1, x2, x4	
Macronix	mx25l	mx25l51273g [mx25l51245g-mx66l51235f-spi-x1_x2_x4, mx25l51245g-mx66l51235f-spi-x1_x2_x4_x8]	512		x1, x2, x4	
Macronix	mx25l	mx25l6433f [mx25l6473f-spi-x1_x2_x4]	64		x1, x2, x4	
Macronix	mx25l	mx25v1635f	16		x1, x2, x4	
Macronix	mx25l	mx25v8035f	8		x1, x2, x4	
Macronix	mx25lu	mx25u8035f [mx25u8033e-spi-x1_x2_x4, mx25u8033e-spi-x1_x2_x4_x8]	8		x1, x2, x4, x8	
Macronix	mx25u	mx25u12872f [mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4, mx25u12832f-mx25u12835f-mx25u12843g-spi-x1_x2_x4_x8]	128		x1, x2, x4	
Macronix	mx25u	mx25u1635f [mx25u1632f-spi-x1_x2_x4]	16		x1, x2, x4	
Macronix	mx25u	mx25u25673g [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4, mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4_x8]	256		x1, x2, x4	
Macronix	mx25u	mx25u3235f [mx25u3232f-spi-x1_x2_x4]	32		x1, x2, x4	
Macronix	mx25u	mx25u51245g [mx66u51235f-spi-x1_x2_x4]	512		x1, x2, x4	
Macronix	mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-x1_x2_x4, mx25u6435f-mx25u6432f-spi-x1_x2_x4_x8]	64		x1, x2, x4	
Macronix	mx66l	mx66l1g45g	1024		x1, x2,	

Manufacturer	Manufacturer	Family	Device Alias	Density Mb	Data Width Bits
					x4
Macronix	mx66l	mx66l2g45g		2048	x1, x2, x4
Macronix	mx66u	mx66u1g45g		1024	x1, x2, x4
Macronix	mx66u	mx66u2g45g		2048	x1, x2, x4
Micron	mt25ql	mt25ql01g		1024	x1, x2, x4
Micron	mt25ql	mt25ql02g		2048	x1, x2, x4
Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4]		128	x1, x2, x4
Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4]		256	x1, x2, x4
Micron	mt25ql	mt25ql512		512	x1, x2, x4
Micron	mt25qu	mt25qu01g		1024	x1, x2, x4
Micron	mt25qu	mt25qu02g		2048	x1, x2, x4
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4]		128	x1, x2, x4
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4]		256	x1, x2, x4
Micron	mt25qu	mt25qu512		512	x1, x2, x4
Micron	n25q	n25q32-1.8v		32	x1, x2, x4
Micron	n25q	n25q32-3.3v		32	x1, x2, x4
Micron	n25q	n25q64-1.8v		64	x1, x2, x4

Manufacturer	Family	Device Alias	Density Mbit	Data Width Bits
Micron	n25q	n25q64-3.3v	64	x1, x2, x4

Artix UltraScale+ Configuration Memory Devices

The Flash devices supported for configuration of Artix UltraScale+ devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in the following table.

The tables in this Appendix are running lists per AMD family of non-volatile memories which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain components on this list even after they are no longer appropriate for new designs, to support long-term maintenance of end products which contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life cycle. References to specific devices in the tables are not an assurance of their current or future availability.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Table: Supported Flash Memory Devices for Artix UltraScale+ BPI Device Configuration

Manufacturer	Family	Device Alias	Manufacturer	Device ID	Device ID (Hex)	Density Mbits	Data Width Bits
Infineon	s29glxxxp	s29gl01gp	1	2228	2201	1024	x16, x8
Infineon	s29glxxxp	s29gl128p	1	2221	2201	128	x16, x8
Infineon	s29glxxxp	s29gl256p	1	2222	2201	256	x16, x8
Infineon	s29glxxxp	s29gl512p	1	2223	2201	512	x16, x8
Infineon	s29glxxxs	s29gl01gs	1	2228	2201	1024	x16
Infineon	s29glxxxs	s29gl128s	1	2221	2201	128	x16
Infineon	s29glxxxs	s29gl256s	1	2222	2201	256	x16
Infineon	s29glxxxs	s29gl512s	1	2223	2201	512	x16
Infineon	s29glxxxs	s70gl02gs	1	2248	2201	2048	x16
Infineon	s29glxxxt	s29gl01gt	1	2228	2201	1024	x16, x8
Infineon	s29glxxxt	s29gl512t	1	2223	2201	512	x16, x8
Infineon	s29glxxxt	s70gl02gt	1	2248	2201	2048	x16, x8
Macronix	mx29gl	mx29gl128f	c2	2221	2201	128	x16, x8

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Density	Mbits	Data Width	Bits
Macronix	mx29gl	mx29gl256f	c2	2222	2201		256		x16, x8	
Micron	g18	28f128g18f	89	-	-		128		x16	
Micron	g18	mt28gu01gaa89e [28f00ag18f-bpi-x16]		-	-		1024		x16	
Micron	g18	mt28gu256aa89e [28f256g18f-bpi-x16]		-	-		256		x16	
Micron	g18	mt28gu512aa89e [28f512g18f-bpi-x16]		-	-		512		x16	
Micron	m29ew	28f00am29ew89		2228	2201		1024		x16, x8	
Micron	m29ew	28f00bm29ew89		2248	2201		2048		x16, x8	
Micron	m29ew	28f064m29ew89		2210	2200		64		x16, x8	
Micron	m29ew	28f064m29ew89		220c	2201		64		x16, x8	
Micron	m29ew	28f064m29ew89		220c	2201		64		x16, x8	
Micron	m29ew	28f064m29ew89		2210	2201		64		x16, x8	
Micron	m29ew	28f128m29ew89		2221	2201		128		x16, x8	
Micron	m29ew	28f256m29ew89		2222	2201		256		x16, x8	
Micron	m29ew	28f512m29ew89		2223	2201		512		x16, x8	
Micron	m29w	m29w128gh	20	2221	2201		128		x16, x8	
Micron	m29w	m29w128gl	20	2221	2200		128		x16, x8	
Micron	m29w	m29w256gh	20	2222	2201		256		x16, x8	
Micron	m29w	m29w256gl	20	2222	2201		256		x16, x8	
Micron	mt28ew	mt28ew01ga	89	2228	2201		1024		x16, x8	
Micron	mt28ew	mt28ew128a	89	2221	2201		128		x16, x8	
Micron	mt28ew	mt28ew256a	89	2222	2201		256		x16, x8	
Micron	mt28ew	mt28ew512a	89	2223	2201		512		x16, x8	
Micron	mt28fw	mt28fw02gb	89	2248	2201		2048		x16	

Manufacturer	Manufacturer ID	Device ID	Alias	Manufacturer	Device ID	Device Index	Density	Mbits	Data Width	Bits
Micron	p30	28f00ap30b	89		-	-	1024		x16	
Micron	p30	28f00ap30e	89		-	-	1024		x16	
Micron	p30	28f00ap30t	89		-	-	1024		x16	
Micron	p30	28f00bp30e	89		-	-	2048		x16	
Micron	p30	28f128p30b	89		-	-	128		x16	
Micron	p30	28f128p30t	89		-	-	128		x16	
Micron	p30	28f256p30b	89		-	-	256		x16	
Micron	p30	28f256p30t	89		-	-	256		x16	
Micron	p30	28f512p30b	89		-	-	512		x16	
Micron	p30	28f512p30e	89		-	-	512		x16	
Micron	p30	28f512p30t	89		-	-	512		x16	
Micron	p30	28f640p30b	89		-	-	64		x16	
Micron	p30	28f640p30t	89		-	-	64		x16	

Table: Supported Flash Memory Devices for Artix UltraScale+ SPI Device Configuration

Manufacturer	Manufacturer ID	Device ID	Alias	Manufacturer	Device ID	Device Index	Density	Mbits	Data Width	Bits
Infineon	s25fl	s25fl02g			-	-	-	2048		x1, x2, x4, x8
Infineon	s25fl1	s25fl116k	1		40	15	16		x1, x2, x4	
Infineon	s25fl1	s25fl132k	1		40	16	32		x1, x2, x4	
Infineon	s25fl1	s25fl164k	1		40	17	64		x1, x2, x4	
Infineon	s25flxxxl	s25fl064l	1		60	17	64		x1, x2, x4	
Infineon	s25flxxxl	s25fl128l	1		60	18	128		x1, x2, x4	
Infineon	s25flxxxl	s25fl256l	1		60	19	256		x1, x2, x4	
Infineon	s25flxxxp	s25fl032p	1		2	15	32		x1, x2, x4	
Infineon	s25flxxxp	s25fl064p	1		2	16	64		x1, x2, x4	
Infineon	s25flxxxs	s25fl128sxxxxx0 [s25fl127s-			20	18	128		x1, x2, x4, x8	

Manufacturer	Manufacturer Family	Device Alias	Manufacturer	Device ID	Device Index	Density	Mbits	Data Width	Bits
		spi-x1_x2_x4, s25fl127s- spi-x1_x2_x4_x8]							
Infineon	s25flxxxxs	s25fl128sxxxxx1		20	18	128		x1, x2, x4, x8	
Infineon	s25flxxxxs	s25fl256sxxxxx0		2	19	256		x1, x2, x4	
Infineon	s25flxxxxs	s25fl256sxxxxx1		2	19	256		x1, x2, x4, x8	
Infineon	s25flxxxxs	s25fl512s	1	2	20	512		x1, x2, x4, x8	
Infineon	s25hs	s25hs02gt	-	-	-	2048		x1, x2, x4	
Infineon	s25hs	s25hs512t	34	2b	1a	512		x1, x2, x4	
ISSI	is25l	is25lp01g	9d	60	1b	1024		x1, x2, x4	
ISSI	is25lp	is25lp512m	9d	60	1a	512		x1, x2, x4	
ISSI	is25w	is25wp01g	9d	70	1b	1024		x1, x2, x4, x8	
ISSI	is25wp	is25wp016d	9d	70	15	16		x1, x2, x4, x8	
ISSI	is25wp	is25wp032d	9d	70	16	32		x1, x2, x4	
ISSI	is25wp	is25wp064a	9d	70	17	64		x1, x2, x4, x8	
ISSI	is25wp	is25wp080d	9d	70	14	8		x1, x2, x4, x8	
ISSI	is25wp	is25wp128f	9d	70	18	128		x1, x2, x4, x8	
ISSI	is25wp	is25wp256d	9d	70	19	256		x1, x2, x4, x8	
ISSI	is25wp	is25wp512m	9d	70	1a	512		x1, x2, x4	
Macronix	mx25l	mx25l12872fc2 [mx25l12833f-		20	18	128		x1, x2, x4	

Manufacturer	Manufacturer Family	Device Alias	Manufacturer	Device	Device Index	Device Density	Mbits	Data Width	Bits
		mx25l12835f- mx25l12845g- spi- x1_x2_x4, mx25l12833f- mx25l12835f- mx25l12845g- spi- x1_x2_x4_x8]							
Macronix mx25l		mx25l25673g c2 [mx25l25635f- mx25l25645g- spi- x1_x2_x4, mx25l25635f- mx25l25645g- spi- x1_x2_x4_x8]		20	19	256		x1, x2, x4, x8	
Macronix mx25l		mx25l51273g c2 [mx25l51245g- mx66l51235f- spi- x1_x2_x4, mx25l51245g- mx66l51235f- spi- x1_x2_x4_x8]		20	1a	512		x1, x2, x4, x8	
Macronix mx25lu		mx25u8035f c2 [mx25u8033e- spi- x1_x2_x4, mx25u8033e- spi- x1_x2_x4_x8]		25	34	8		x1, x2, x4, x8	
Macronix mx25u		mx25u12872fc2 [mx25u12832f- mx25u12835f- mx25u12843g- spi- x1_x2_x4,		25	38	128		x1, x2, x4	

Manufacturer	Manufacturer Family	Alias	Manufacturer	Device	Device Index	Bank Density	Mbits	Data Width	Bits
		mx25u12832f- mx25u12835f- mx25u12843g- spi- x1_x2_x4_x8]							
Macronix mx25u		mx25u1635f_c2 [mx25u1632f- spi- x1_x2_x4, mx25u1632f- spi- x1_x2_x4_x8]		25	35	16		x1, x2, x4	
Macronix mx25u		mx25u25673gc2 [mx25u25635f- mx25u25643g- mx25u25645g- spi- x1_x2_x4, mx25u25635f- mx25u25643g- mx25u25645g- spi- x1_x2_x4_x8]		25	39	256		x1, x2, x4, x8	
Macronix mx25u		mx25u3235f_c2 [mx25u3232f- spi- x1_x2_x4, mx25u3232f- spi- x1_x2_x4_x8]		25	36	32		x1, x2, x4, x8	
Macronix mx25u		mx25u51245gc2 [mx66u51235f- spi- x1_x2_x4, mx66u51235f- spi- x1_x2_x4_x8]		25	3a	512		x1, x2, x4, x8	
Macronix mx25u		mx25u6472f_c2 [mx25u6435f- mx25u6432f- spi- x1_x2_x4_x8]		25	37	64		x1, x2, x4	

Manufacturer	Manufacturer Family	Device Alias	Manufacturer	Device ID	Device Index	Density	Mbits	Data Width Bits
		spi-x1_x2_x4, mx25u6435f- mx25u6432f- spi-x1_x2_x4_x8]						
Macronix	mx66u	mx66u1g45g c2		25	3b	1024		x1, x2, x4, x8
Macronix	mx66u	mx66u2g45g c2		25	3c	2048		x1, x2, x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024		x1, x2, x4, x8
Micron	mt25qu	mt25qu02g	20	bb	22	2048		x1, x2, x4
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4, n25q128-1.8v-spi-x1_x2_x4_x8]	20	bb	18	128		x1, x2, x4, x8
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4, n25q256-1.8v-spi-x1_x2_x4_x8]	20	bb	19	256		x1, x2, x4
Micron	mt25qu	mt25qu512	20	bb	20	512		x1, x2, x4, x8
Micron	n25q	n25q32-1.8v	20	bb	16	32		x1, x2, x4
Micron	n25q	n25q64-1.8v	20	bb	17	64		x1, x2, x4
Winbond	w25q	w25q256jw	ef	60	19	256		x1, x2, x4, x8

Kintex UltraScale Configuration Memory Devices

The Flash devices supported for configuration of Kintex UltraScale devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in the following table.

The tables in this Appendix are running lists per AMD family of non-volatile memories which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain components on this list even after they are no longer appropriate for new designs, to support long-term maintenance of end products which can contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life cycle. References to specific devices in the tables are not an assurance of their current or future availability.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Table: Supported Flash Memory Devices for Kintex UltraScale BPI Device Configuration

Manufacturer	Device Alias	MANUFACTURE	TYPE	PROGRAMMING	DEVICE_ID	IDD	Density	Mb	Data Width Bits
Infineon s29glxxxp	s29gl01gp	1	227e	2228	2201	1024		x16, x8	
Infineon s29glxxxp	s29gl128p	1	227e	2221	2201	128		x16, x8	
Infineon s29glxxxp	s29gl256p	1	227e	2222	2201	256		x16, x8	
Infineon s29glxxxp	s29gl512p	1	227e	2223	2201	512		x16, x8	
Infineon s29glxxxp	s70gl02gp	1	227e	2248	2201	2048		x16	
Infineon s29glxxxs	s29gl01gs	1	227e	2228	2201	1024		x16	
Infineon s29glxxxs	s29gl128s	1	227e	2221	2201	128		x16	
Infineon s29glxxxs	s29gl256s	1	227e	2222	2201	256		x16	
Infineon s29glxxxs	s29gl512s	1	227e	2223	2201	512		x16	
Infineon s29glxxxs	s70gl02gs	1	227e	2248	2201	2048		x16	
Infineon s29glxxxt	s29gl01gt	1	227e	2228	2201	1024		x16, x8	
Infineon s29glxxxt	s29gl512t	1	227e	2223	2201	512		x16, x8	
Infineon s29glxxxt	s70gl02gt	1	227e	2248	2201	2048		x16, x8	
Macronixmx29gl	mx29gl128f c2		227e	2221	2201	128		x16, x8	
Macronixmx29gl	mx29gl256f c2		227e	2222	2201	256		x16, x8	

Manufacturer	Device Alias	MANUFACTURE_DATE	MANUFACTURE_MONTH	MANUFACTURE_YEAR	DEVICE_IDD	DEVICE_IDD	Density	Mb	Data Width Bits
Micron	g18	28f128g18f	89		8900	-	-	128	x16
Micron	g18	mt28gu01ga89	89	01e	88b0	-	-	1024	x16
Micron	g18	mt28gu256a89	89	01e	8901	-	-	256	x16
Micron	g18	mt28gu512a89	89	01e	887e	-	-	512	x16
Micron	m29ew	28f00am29e	89		227e	2228	2201	1024	x16, x8
Micron	m29ew	28f00bm29e	89		227e	2248	2201	2048	x16, x8
Micron	m29ew	28f064m29e	89		227e	2210	2200	64	x16, x8
Micron	m29ew	28f064m29e	89		227e	220c	2201	64	x16, x8
Micron	m29ew	28f064m29e	89		227e	220c	2201	64	x16, x8
Micron	m29ew	28f064m29e	89		227e	2210	2201	64	x16, x8
Micron	m29ew	28f128m29e	89		227e	2221	2201	128	x16, x8
Micron	m29ew	28f256m29e	89		227e	2222	2201	256	x16, x8
Micron	m29ew	28f512m29e	89		227e	2223	2201	512	x16, x8
Micron	m29w	m29w128gh	20		227e	2221	2201	128	x16, x8
Micron	m29w	m29w128gl	20		227e	2221	2200	128	x16, x8
Micron	m29w	m29w256gh	20		227e	2222	2201	256	x16, x8
Micron	m29w	m29w256gl	20		227e	2222	2201	256	x16, x8
Micron	m29w	m29w640gh	20		227e	220c	2201	64	x16, x8
Micron	m29w	m29w640gl	20		227e	220c	2200	64	x16, x8
Micron	mt28ew	mt28ew01ga	89		227e	2228	2201	1024	x16, x8
Micron	mt28ew	mt28ew128a	89		227e	2221	2201	128	x16, x8
Micron	mt28ew	mt28ew256a	89		227e	2222	2201	256	x16, x8
Micron	mt28ew	mt28ew512a	89		227e	2223	2201	512	x16, x8

Manufacturer	Device	Device Alias	MANUFACTURE_DATE	MANUFACTURE_ID	MANUFACTURE_ID	DEVICE_ID	DEVICE_ID	DDensity	Mb	Data Width Bits
Micron	mt28fw	mt28fw02gb 89		227e	2248	2201		2048	x16	
Micron	p30	28f00ap30b 89		8963	-	-		1024	x16	
Micron	p30	28f00ap30e 89		899a	-	-		1024	x16	
Micron	p30	28f00ap30t 89		8962	-	-		1024	x16	
Micron	p30	28f00bp30e 89		899a	-	-		2048	x16	
Micron	p30	28f128p30b 89		881b	-	-		128	x16	
Micron	p30	28f128p30t 89		8818	-	-		128	x16	
Micron	p30	28f256p30b 89		891c	-	-		256	x16	
Micron	p30	28f256p30t 89		8919	-	-		256	x16	
Micron	p30	28f512p30b 89		8961	-	-		512	x16	
Micron	p30	28f512p30e 89		8999	-	-		512	x16	
Micron	p30	28f512p30t 89		8960	-	-		512	x16	
Micron	p30	28f640p30b 89		881a	-	-		64	x16	
Micron	p30	28f640p30t 89		8817	-	-		64	x16	
Micron	p33	28f00ap33b 89		8967	-	-		1024	x16	
Micron	p33	28f00ap33e 89		899f	-	-		1024	x16	
Micron	p33	28f00ap33t 89		8966	-	-		1024	x16	
Micron	p33	28f128p33b 89		8821	-	-		128	x16	
Micron	p33	28f128p33t 89		881e	-	-		128	x16	
Micron	p33	28f256p33b 89		8922	-	-		256	x16	
Micron	p33	28f256p33t 89		891f	-	-		256	x16	
Micron	p33	28f512p33b 89		8965	-	-		512	x16	
Micron	p33	28f512p33e 89		899e	-	-		512	x16	
Micron	p33	28f512p33t 89		8964	-	-		512	x16	
Micron	p33	28f640p33b 89		8820	-	-		64	x16	
Micron	p33	28f640p33t 89		881d	-	-		64	x16	

Table: Supported Flash Memory Devices for Kintex UltraScale SPI Device Configuration

Manufacturer	Manufacturer Device ID	Device Alias	MANUFACTURER	ROM SIZE	FLASH TYPE	DENSITY	VID	Data Width Bits
Infineon	s25fl	s25fl02g	-	-	-	2048		x1, x2, x4, x8
Infineon	s25fl1	s25fl116k	1	40	15	16		x1, x2, x4
Infineon	s25fl1	s25fl132k	1	40	16	32		x1, x2, x4
Infineon	s25fl1	s25fl164k	1	40	17	64		x1, x2, x4
Infineon	s25flxxxl	s25fl064l	1	60	17	64		x1, x2, x4
Infineon	s25flxxxl	s25fl128l	1	60	18	128		x1, x2, x4
Infineon	s25flxxxl	s25fl256l	1	60	19	256		x1, x2, x4, x8
Infineon	s25flxxxp	s25fl032p	1	2	15	32		x1, x2, x4
Infineon	s25flxxxp	s25fl064p	1	2	16	64		x1, x2, x4
Infineon	s25flxxxxs	s25fl128sxxxxlx0 [s25fl127s-spi-x1_x2_x4, s25fl127s-spi-x1_x2_x4_x8]		20	18	128		x1, x2, x4, x8
Infineon	s25flxxxxs	s25fl128sxxxxlx1		20	18	128		x1, x2, x4
Infineon	s25flxxxxs	s25fl256sxxxxlx0		2	19	256		x1, x2, x4
Infineon	s25flxxxxs	s25fl256sxxxxlx1		2	19	256		x1, x2, x4, x8
Infineon	s25flxxxxs	s25fl512s	1	2	20	512		x1, x2, x4
Infineon	s25hs	s25hs02gt	-	-	-	2048		x1, x2, x4, x8
Infineon	s25hs	s25hs512t	34	2b	1a	512		x1, x2, x4
ISSI	is25l	is25lp01g	9d	60	1b	1024		x1, x2, x4, x8
ISSI	is25lp	is25lp016d	9d	60	15	16		x1, x2, x4
ISSI	is25lp	is25lp032d	9d	60	16	32		x1, x2, x4

Manufacturer	Manufacturer Family	Alias	MANUFACTURE_DATE	REVISION	MEMORY_TYPE	DATA_CAPACITY	VID	Data Width Bits
ISSI	is25lp	is25lp064a	9d	60	17	64		x1, x2, x4
ISSI	is25lp	is25lp080d	9d	60	14	8		x1, x2, x4, x8
ISSI	is25lp	is25lp128f	9d	60	18	128		x1, x2, x4
ISSI	is25lp	is25lp256d	9d	60	19	256		x1, x2, x4, x8
ISSI	is25lp	is25lp512m	9d	60	1a	512		x1, x2, x4
ISSI	is25wp	is25wp01g	9d	70	1b	1024		x1, x2, x4
ISSI	is25wp	is25wp016d	9d	70	15	16		x1, x2, x4, x8
ISSI	is25wp	is25wp032d	9d	70	16	32		x1, x2, x4
ISSI	is25wp	is25wp064a	9d	70	17	64		x1, x2, x4
ISSI	is25wp	is25wp080d	9d	70	14	8		x1, x2, x4, x8
ISSI	is25wp	is25wp128f	9d	70	18	128		x1, x2, x4
ISSI	is25wp	is25wp256d	9d	70	19	256		x1, x2, x4
ISSI	is25wp	is25wp512m	9d	70	1a	512		x1, x2, x4
Macronix mx25l	mx25l12872f c2 [mx25l12833f- mx25l12835f- mx25l12845g- spi- x1_x2_x4, mx25l12833f- mx25l12835f- mx25l12845g- spi- x1_x2_x4_x8]			20	18	128		x1, x2, x4, x8
Macronix mx25l	mx25l25673g c2 [mx25l25635f- mx25l25645g- spi- x1_x2_x4, mx25l25635f-			20	19	256		x1, x2, x4

Manufacturer	Manufacturer Family	Alias	MANUFACTURE_DATE	MEMORY_TYPE	DATA_WIDTH	Bits
		mx25l25645g- spi- x1_x2_x4_x8]				
Macronix mx25l		mx25l3273f c2 [mx25l3233f- spi- x1_x2_x4, mx25l3233f- spi- x1_x2_x4_x8]	20	16	32	x1, x2, x4
Macronix mx25l		mx25l51273g c2 [mx25l51245g- mx66l51235f- spi- x1_x2_x4, mx25l51245g- mx66l51235f- spi- x1_x2_x4_x8]	20	1a	512	x1, x2, x4
Macronix mx25l		mx25l6433f c2 [mx25l6473f- spi- x1_x2_x4, mx25l6473f- spi- x1_x2_x4_x8]	20	17	64	x1, x2, x4
Macronix mx25l		mx25v1635f c2	23	15	16	x1, x2, x4, x8
Macronix mx25l		mx25v8035f c2	23	14	8	x1, x2, x4, x8
Macronix mx25lu		mx25u8035f c2 [mx25u8033e- spi- x1_x2_x4, mx25u8033e- spi- x1_x2_x4_x8]	25	34	8	x1, x2, x4, x8

Manufacturer	Manufacturer ID	Family	Alias	MANUFACTURE_DATE	MEMORY_TYPE	DATA_WIDTH	Bits
Macronix	mx25u	mx25u12872fc2 [mx25u12832f- mx25u12835f- mx25u12843g- spi- x1_x2_x4, mx25u12832f- mx25u12835f- mx25u12843g- spi- x1_x2_x4_x8]		25	38	128	x1, x2, x4, x8
Macronix	mx25u	mx25u1635f_c2 [mx25u1632f- spi- x1_x2_x4, mx25u1632f- spi- x1_x2_x4_x8]		25	35	16	x1, x2, x4, x8
Macronix	mx25u	mx25u25673gc2 [mx25u25635f- mx25u25643g- mx25u25645g- spi- x1_x2_x4, mx25u25635f- mx25u25643g- mx25u25645g- spi- x1_x2_x4_x8]		25	39	256	x1, x2, x4, x8
Macronix	mx25u	mx25u3235f_c2 [mx25u3232f- spi- x1_x2_x4, mx25u3232f- spi- x1_x2_x4_x8]		25	36	32	x1, x2, x4, x8
Macronix	mx25u	mx25u51245gc2 [mx66u51235f- spi- x1_x2_x4,		25	3a	512	x1, x2, x4

Manufacturer	Manufacturer Family	Alias	MANUFACTURE_DATE	REVISION	MEMORY_TYPE	DATA_CAPACITY	VID	Data Width Bits
		mx66u51235f-spi-[x1_x2_x4_x8]						
Macronix mx25u		mx25u6472f c2 [mx25u6435f-mx25u6432f-spi-x1_x2_x4, mx25u6435f-mx25u6432f-spi-x1_x2_x4_x8]	25	37	64	x1, x2, x4, x8		
Macronix mx66l		mx66l1g45g c2	20	1b	1024	x1, x2, x4		
Macronix mx66l		mx66l2g45g c2	20	1c	2048	x1, x2, x4, x8		
Macronix mx66u		mx66u1g45g c2	25	3b	1024	x1, x2, x4, x8		
Macronix mx66u		mx66u2g45g c2	25	3c	2048	x1, x2, x4, x8		
Micron	mt25ql	mt25ql01g	20	ba	21	1024	x1, x2, x4	
Micron	mt25ql	mt25ql02g	20	ba	22	2048	x1, x2, x4, x8	
Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4, n25q128-3.3v-spi-x1_x2_x4_x8]	20	ba	18	128	x1, x2, x4	
Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4, n25q256-3.3v-spi-x1_x2_x4_x8]	20	ba	19	256	x1, x2, x4, x8	

Manufacturer	Manufacturer Device ID	Alias	MANUFACTURER_ID	ROW_CYCLES	COLUMN_BYTES	DATA_WIDTH	Data Width Bits
Micron	mt25ql	mt25ql512	20	ba	20	512	x1, x2, x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024	x1, x2, x4, x8
Micron	mt25qu	mt25qu02g	20	bb	22	2048	x1, x2, x4
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4, n25q128-1.8v-spi-x1_x2_x4_x8]	20	bb	18	128	x1, x2, x4
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4, n25q256-1.8v-spi-x1_x2_x4_x8]	20	bb	19	256	x1, x2, x4
Micron	mt25qu	mt25qu512	20	bb	20	512	x1, x2, x4, x8
Micron	n25q	n25q32-1.8v	20	bb	16	32	x1, x2, x4
Micron	n25q	n25q32-3.3v	20	bb	16	32	x1, x2, x4
Micron	n25q	n25q64-1.8v	20	bb	17	64	x1, x2, x4
Micron	n25q	n25q64-3.3v	20	bb	17	64	x1, x2, x4
Winbond	w25q	w25q256jw	ef	60	19	256	x1, x2, x4, x8

Kintex UltraScale+ Configuration Memory Devices

The Flash devices supported for configuration of Kintex UltraScale+ devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in the following table.

The tables in this Appendix are running lists per AMD family of non-volatile memories which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain components on this list even after they are no longer appropriate for new designs, to support long-term maintenance of end products which can contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life cycle. References to specific devices in the tables are not an assurance of their current or future availability.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Table: Supported Flash Memory Devices for Kintex UltraScale+ BPI Device Configuration

Manufacturer	Manufacturer Family	Device Alias	Manufacturer	Device ID	Kintex ID	Device Density	Mbits	Data Width Bits
Infineon	s29glxxxp	s29gl01gp	1	2228	2201	1024		x16, x8
Infineon	s29glxxxp	s29gl128p	1	2221	2201	128		x16, x8
Infineon	s29glxxxp	s29gl256p	1	2222	2201	256		x16, x8
Infineon	s29glxxxp	s29gl512p	1	2223	2201	512		x16, x8
Infineon	s29glxxxs	s29gl01gs	1	2228	2201	1024		x16
Infineon	s29glxxxs	s29gl128s	1	2221	2201	128		x16
Infineon	s29glxxxs	s29gl256s	1	2222	2201	256		x16
Infineon	s29glxxxs	s29gl512s	1	2223	2201	512		x16
Infineon	s29glxxxs	s70gl02gs	1	2248	2201	2048		x16
Infineon	s29glxxxt	s29gl01gt	1	2228	2201	1024		x16, x8
Infineon	s29glxxxt	s29gl512t	1	2223	2201	512		x16, x8
Infineon	s29glxxxt	s70gl02gt	1	2248	2201	2048		x16, x8
Macronix	mx29gl	mx29gl128f	c2	2221	2201	128		x16, x8
Macronix	mx29gl	mx29gl256f	c2	2222	2201	256		x16, x8
Micron	g18	28f128g18f	89	-	-	128		x16
Micron	g18	mt28gu01gaa 89e [28f00ag18f- bpi-x16]		-	-	1024		x16
Micron	g18	mt28gu256aa 89e [28f256g18f-		-	-	256		x16

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Density	Mbits	Data Width	Bits
		bpi-x16]								
Micron	g18	mt28gu512aa89e [28f512g18f-bpi-x16]		-	-		512		x16	
Micron	m29ew	28f00am29ew89		2228	2201		1024		x16, x8	
Micron	m29ew	28f00bm29ew89		2248	2201		2048		x16, x8	
Micron	m29ew	28f064m29ew89		2210	2200		64		x16, x8	
Micron	m29ew	28f064m29ew89		220c	2201		64		x16, x8	
Micron	m29ew	28f064m29ew89		220c	2201		64		x16, x8	
Micron	m29ew	28f064m29ew89		2210	2201		64		x16, x8	
Micron	m29ew	28f128m29ew89		2221	2201		128		x16, x8	
Micron	m29ew	28f256m29ew89		2222	2201		256		x16, x8	
Micron	m29ew	28f512m29ew89		2223	2201		512		x16, x8	
Micron	m29w	m29w128gh	20	2221	2201		128		x16, x8	
Micron	m29w	m29w128gl	20	2221	2200		128		x16, x8	
Micron	m29w	m29w256gh	20	2222	2201		256		x16, x8	
Micron	m29w	m29w256gl	20	2222	2201		256		x16, x8	
Micron	mt28ew	mt28ew01ga	89	2228	2201		1024		x16, x8	
Micron	mt28ew	mt28ew128a	89	2221	2201		128		x16, x8	
Micron	mt28ew	mt28ew256a	89	2222	2201		256		x16, x8	
Micron	mt28ew	mt28ew512a	89	2223	2201		512		x16, x8	
Micron	mt28fw	mt28fw02gb	89	2248	2201		2048		x16	
Micron	p30	28f00ap30b	89	-	-		1024		x16	
Micron	p30	28f00ap30e	89	-	-		1024		x16	
Micron	p30	28f00ap30t	89	-	-		1024		x16	
Micron	p30	28f00bp30e	89	-	-		2048		x16	
Micron	p30	28f128p30b	89	-	-		128		x16	

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Density	Mbits	Data Width Bits
Micron	p30	28f128p30t	89	-	-	-	128	x16	
Micron	p30	28f256p30b	89	-	-	-	256	x16	
Micron	p30	28f256p30t	89	-	-	-	256	x16	
Micron	p30	28f512p30b	89	-	-	-	512	x16	
Micron	p30	28f512p30e	89	-	-	-	512	x16	
Micron	p30	28f512p30t	89	-	-	-	512	x16	
Micron	p30	28f640p30b	89	-	-	-	64	x16	
Micron	p30	28f640p30t	89	-	-	-	64	x16	

Table: Supported Flash Memory Devices for Kintex UltraScale+ SPI Device Configuration

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Density	Mbits	Data Width Bits
Infineon	s25fl	s25fl02g	-	-	-	-	2048	x1, x2, x4, x8	
Infineon	s25fl1	s25fl116k	1	40	15	16		x1, x2, x4	
Infineon	s25fl1	s25fl132k	1	40	16	32		x1, x2, x4	
Infineon	s25fl1	s25fl164k	1	40	17	64		x1, x2, x4	
Infineon	s25flxxxl	s25fl064l	1	60	17	64		x1, x2, x4	
Infineon	s25flxxxl	s25fl128l	1	60	18	128		x1, x2, x4	
Infineon	s25flxxxl	s25fl256l	1	60	19	256		x1, x2, x4	
Infineon	s25flxxxp	s25fl032p	1	2	15	32		x1, x2, x4	
Infineon	s25flxxxp	s25fl064p	1	2	16	64		x1, x2, x4	
Infineon	s25flxxxs	s25fl128sxxxxx0 [s25fl127s-spi-x1_x2_x4, s25fl127s-spi-x1_x2_x4_x8]			20	18	128		x1, x2, x4
Infineon	s25flxxxs	s25fl128sxxxxx1			20	18	128		x1, x2, x4, x8

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Density	Mbits	Data Width Bits
Infineon	s25flxxxxs	s25fl256sxxxx	x0		2	19	256	x1, x2, x4, x8	
Infineon	s25flxxxxs	s25fl256sxxxx	x1		2	19	256	x1, x2, x4	
Infineon	s25flxxxxs	s25fl512s	1		2	20	512	x1, x2, x4	
Infineon	s25hs	s25hs02gt	-		-	-	2048	x1, x2, x4	
Infineon	s25hs	s25hs512t	34		2b	1a	512	x1, x2, x4, x8	
ISSI	is25l	is25lp01g	9d		60	1b	1024	x1, x2, x4	
ISSI	is25lp	is25lp512m	9d		60	1a	512	x1, x2, x4	
ISSI	is25w	is25wp01g	9d		70	1b	1024	x1, x2, x4	
ISSI	is25wp	is25wp016d	9d		70	15	16	x1, x2, x4	
ISSI	is25wp	is25wp032d	9d		70	16	32	x1, x2, x4, x8	
ISSI	is25wp	is25wp064a	9d		70	17	64	x1, x2, x4, x8	
ISSI	is25wp	is25wp080d	9d		70	14	8	x1, x2, x4, x8	
ISSI	is25wp	is25wp128f	9d		70	18	128	x1, x2, x4, x8	
ISSI	is25wp	is25wp256d	9d		70	19	256	x1, x2, x4, x8	
ISSI	is25wp	is25wp512m	9d		70	1a	512	x1, x2, x4, x8	
Macronix	mx25l	mx25l12872f	c2		20	18	128	x1, x2, x4	
		[mx25l12833f-							
		mx25l12835f-							
		mx25l12845g-							
		spi-							
		x1_x2_x4,							
		mx25l12833f-							
		mx25l12835f-							
		mx25l12845g-							

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Density	Mbits	Data Width	Bits
			spi-x1_x2_x4_x8]							
Macronix mx25l			mx25l25673g c2 [mx25l25635f- mx25l25645g- spi- x1_x2_x4, mx25l25635f- mx25l25645g- spi- x1_x2_x4_x8]	20	19		256		x1, x2, x4, x8	
Macronix mx25l			mx25l51273g c2 [mx25l51245g- mx66l51235f- spi- x1_x2_x4, mx25l51245g- mx66l51235f- spi- x1_x2_x4_x8]	20	1a		512		x1, x2, x4, x8	
Macronix mx25lu			mx25u8035f c2 [mx25u8033e- spi- x1_x2_x4, mx25u8033e- spi- x1_x2_x4_x8]	25	34		8		x1, x2, x4, x8	
Macronix mx25u			mx25u12872fc2 [mx25u12832f- mx25u12835f- mx25u12843g- spi- x1_x2_x4, mx25u12832f- mx25u12835f- mx25u12843g- spi- x1_x2_x4_x8]	25	38		128		x1, x2, x4	
Macronix mx25u			mx25u1635f c2 [mx25u1632f-	25	35		16		x1, x2, x4	

Manufacturer	Manufacturer Family	Alias	Manufacturer Device ID	Device Index	Device ID (Hex)	Bank Density	Mbits	Data Width	Bits
		spi-x1_x2_x4, mx25u1632f-spi-x1_x2_x4_x8]							
Macronix mx25u		mx25u25673gc2 [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4, mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4_x8]	25	39	256			x1, x2, x4	
Macronix mx25u		mx25u3235f_c2 [mx25u3232f-spi-x1_x2_x4, mx25u3232f-spi-x1_x2_x4_x8]	25	36	32			x1, x2, x4	
Macronix mx25u		mx25u51245gc2 [mx66u51235f-spi-x1_x2_x4, mx66u51235f-spi-x1_x2_x4_x8]	25	3a	512			x1, x2, x4, x8	
Macronix mx25u		mx25u6472f_c2 [mx25u6435f-mx25u6432f-spi-x1_x2_x4, mx25u6435f-mx25u6432f-spi-x1_x2_x4_x8]	25	37	64			x1, x2, x4	

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Density	Mbits	Data Width Bits
		Macronix mx66u	mx66u1g45g c2		25	3b	1024		x1, x2, x4, x8
		Macronix mx66u	mx66u2g45g c2		25	3c	2048		x1, x2, x4, x8
Micron	mt25qu	mt25qu01g	20	bb	21		1024		x1, x2, x4
Micron	mt25qu	mt25qu02g	20	bb	22		2048		x1, x2, x4, x8
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4, n25q128-1.8v-spi-x1_x2_x4_x8]	20	bb	18		128		x1, x2, x4
Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4, n25q256-1.8v-spi-x1_x2_x4_x8]	20	bb	19		256		x1, x2, x4, x8
Micron	mt25qu	mt25qu512	20	bb	20		512		x1, x2, x4
Micron	n25q	n25q32-1.8v	20	bb	16		32		x1, x2, x4
Micron	n25q	n25q64-1.8v	20	bb	17		64		x1, x2, x4
Winbond w25q	w25q256jw	ef	60	19			256		x1, x2, x4, x8

Virtex UltraScale Configuration Memory Devices

The Flash devices supported for configuration of AMD Virtex™ UltraScale™ devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in the following table.

The tables in this Appendix are running lists per AMD family of non-volatile memories which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain

components on this list even after they are no longer appropriate for new designs, to support long-term maintenance of end products which can contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life cycle. References to specific devices in the tables are not an assurance of their current or future availability.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Table: Supported Flash Memory Devices for Virtex UltraScale BPI Device Configuration

Manufacturer	Device Alias	MANUFACTURER	TYPE	REV	DEVICE_ID	DENSITY	MB	Data Width Bits
Infineon s29glxxxp	s29gl01gp	1	227e	2228	2201	1024		x16, x8
Infineon s29glxxxp	s29gl128p	1	227e	2221	2201	128		x16, x8
Infineon s29glxxxp	s29gl256p	1	227e	2222	2201	256		x16, x8
Infineon s29glxxxp	s29gl512p	1	227e	2223	2201	512		x16, x8
Infineon s29glxxxp	s70gl02gp	1	227e	2248	2201	2048		x16
Infineon s29glxxxs	s29gl01gs	1	227e	2228	2201	1024		x16
Infineon s29glxxxs	s29gl128s	1	227e	2221	2201	128		x16
Infineon s29glxxxs	s29gl256s	1	227e	2222	2201	256		x16
Infineon s29glxxxs	s29gl512s	1	227e	2223	2201	512		x16
Infineon s29glxxxs	s70gl02gs	1	227e	2248	2201	2048		x16
Infineon s29glxxxt	s29gl01gt	1	227e	2228	2201	1024		x16, x8
Infineon s29glxxxt	s29gl512t	1	227e	2223	2201	512		x16, x8
Infineon s29glxxxt	s70gl02gt	1	227e	2248	2201	2048		x16, x8
Macronixmx29gl	mx29gl128f c2		227e	2221	2201	128		x16, x8
Macronixmx29gl	mx29gl256f c2		227e	2222	2201	256		x16, x8
Micron g18	28f128g18f 89		8900	-	-	128		x16
Micron g18	mt28gu01ga891e [28f00ag18f-bpi-x16]		88b0	-	-	1024		x16
Micron g18	mt28gu256a891e [28f256g18f-		8901	-	-	256		x16

Manufacturer	Device Alias	MANUFACTURE_DATE	MANUFACTURE_MONTH	MANUFACTURE_YEAR	DEVICE_IDD	DEVICE_IDD	Density	Mb	Data Width Bits
	bpi-x16]								
Micron	g18	mt28gu512a891e [28f512g18f- bpi-x16]	887e	-	-	512	x16		
Micron	m29ew	28f00am29e89	227e	2228	2201	1024	x16, x8		
Micron	m29ew	28f00bm29e89	227e	2248	2201	2048	x16, x8		
Micron	m29ew	28f064m29e89	227e	2210	2200	64	x16, x8		
Micron	m29ew	28f064m29e89	227e	220c	2201	64	x16, x8		
Micron	m29ew	28f064m29e89	227e	220c	2201	64	x16, x8		
Micron	m29ew	28f064m29e89	227e	2210	2201	64	x16, x8		
Micron	m29ew	28f128m29e89	227e	2221	2201	128	x16, x8		
Micron	m29ew	28f256m29e89	227e	2222	2201	256	x16, x8		
Micron	m29ew	28f512m29e89	227e	2223	2201	512	x16, x8		
Micron	m29w	m29w128gh 20	227e	2221	2201	128	x16, x8		
Micron	m29w	m29w128gl 20	227e	2221	2200	128	x16, x8		
Micron	m29w	m29w256gh 20	227e	2222	2201	256	x16, x8		
Micron	m29w	m29w256gl 20	227e	2222	2201	256	x16, x8		
Micron	m29w	m29w640gh 20	227e	220c	2201	64	x16, x8		
Micron	m29w	m29w640gl 20	227e	220c	2200	64	x16, x8		
Micron	mt28ew	mt28ew01ga89	227e	2228	2201	1024	x16, x8		
Micron	mt28ew	mt28ew128a89	227e	2221	2201	128	x16, x8		
Micron	mt28ew	mt28ew256a89	227e	2222	2201	256	x16, x8		
Micron	mt28ew	mt28ew512a89	227e	2223	2201	512	x16, x8		
Micron	mt28fw	mt28fw02gb 89	227e	2248	2201	2048	x16		
Micron	p30	28f00ap30b 89	8963	-	-	1024	x16		
Micron	p30	28f00ap30e 89	899a	-	-	1024	x16		
Micron	p30	28f00ap30t 89	8962	-	-	1024	x16		

Manufacturer	Manufacturer Device Alias	Device Alias	MANUFACTURER	TYPE	REVISION	PROGRAMMING	DEVICE_ID	IDD	Density	MB	Data Width Bits
Micron	p30	28f00bp30e 89	899a	-	-	-	-	2048	x16		
Micron	p30	28f128p30b 89	881b	-	-	-	-	128	x16		
Micron	p30	28f128p30t 89	8818	-	-	-	-	128	x16		
Micron	p30	28f256p30b 89	891c	-	-	-	-	256	x16		
Micron	p30	28f256p30t 89	8919	-	-	-	-	256	x16		
Micron	p30	28f512p30b 89	8961	-	-	-	-	512	x16		
Micron	p30	28f512p30e 89	8999	-	-	-	-	512	x16		
Micron	p30	28f512p30t 89	8960	-	-	-	-	512	x16		
Micron	p30	28f640p30b 89	881a	-	-	-	-	64	x16		
Micron	p30	28f640p30t 89	8817	-	-	-	-	64	x16		
Micron	p33	28f00ap33b 89	8967	-	-	-	-	1024	x16		
Micron	p33	28f00ap33e 89	899f	-	-	-	-	1024	x16		
Micron	p33	28f00ap33t 89	8966	-	-	-	-	1024	x16		
Micron	p33	28f128p33b 89	8821	-	-	-	-	128	x16		
Micron	p33	28f128p33t 89	881e	-	-	-	-	128	x16		
Micron	p33	28f256p33b 89	8922	-	-	-	-	256	x16		
Micron	p33	28f256p33t 89	891f	-	-	-	-	256	x16		
Micron	p33	28f512p33b 89	8965	-	-	-	-	512	x16		
Micron	p33	28f512p33e 89	899e	-	-	-	-	512	x16		
Micron	p33	28f512p33t 89	8964	-	-	-	-	512	x16		
Micron	p33	28f640p33b 89	8820	-	-	-	-	64	x16		
Micron	p33	28f640p33t 89	881d	-	-	-	-	64	x16		

Table: Supported Flash Memory Devices for Virtex UltraScale SPI Device Configuration

Manufacturer	Manufacturer Device Alias	Device Alias	MANUFACTURER	TYPE	REVISION	PROGRAMMING	DEVICE_ID	IDD	DENSITY	MB	Data Width Bits
Infineon	s25fl	s25fl02g	NA	-	-	-	-	2048	x1, x2, x4, x8		

Manufacturer	Device ID	Device Alias	MANUFACTURER	MEMORY TYPE	capacity	Yield	Data Width Bits
Infineon	s25fl1	s25fl116k	1	40	15	16	x1, x2, x4
Infineon	s25fl1	s25fl132k	1	40	16	32	x1, x2, x4
Infineon	s25fl1	s25fl164k	1	40	17	64	x1, x2, x4
Infineon	s25flxxxxl	s25fl064l	1	60	17	64	x1, x2, x4
Infineon	s25flxxxxl	s25fl128l	1	60	18	128	x1, x2, x4, x8
Infineon	s25flxxxxl	s25fl256l	1	60	19	256	x1, x2, x4, x8
Infineon	s25flxxxxp	s25fl032p	1	2	15	32	x1, x2, x4
Infineon	s25flxxxxp	s25fl064p	1	2	16	64	x1, x2, x4
Infineon	s25flxxxxs	s25fl128xxxxxxxx01 [s25fl127s-spi-x1_x2_x4, s25fl127s-spi-x1_x2_x4_x8]		20	18	128	x1, x2, x4
Infineon	s25flxxxxs	s25fl128xxxxxxxx11		20	18	128	x1, x2, x4, x8
Infineon	s25flxxxxs	s25fl256xxxxxxxx01		2	19	256	x1, x2, x4
Infineon	s25flxxxxs	s25fl256xxxxxxxx11		2	19	256	x1, x2, x4, x8
Infineon	s25flxxxxs	s25fl512s	1	2	20	512	x1, x2, x4
Infineon	s25hs	s25hs02gt	NA	-	-	2048	x1, x2, x4, x8
Infineon	s25hs	s25hs512t	34	2b	1a	512	x1, x2, x4, x8
ISSI	is25l	is25lp01g	9d	60	1b	1024	x1, x2, x4, x8
ISSI	is25lp	is25lp016d	9d	60	15	16	x1, x2, x4, x8
ISSI	is25lp	is25lp032d	9d	60	16	32	x1, x2, x4

Manufacturer	Manufacturer Alias	Device Alias	MANUFACTURER	MEMORY TYPE	DATA WIDTH	Bits
ISSI	is25lp	is25lp064a	9d	60	17	64 x1, x2, x4, x8
ISSI	is25lp	is25lp080d	9d	60	14	8 x1, x2, x4, x8
ISSI	is25lp	is25lp128f	9d	60	18	128 x1, x2, x4, x8
ISSI	is25lp	is25lp256d	9d	60	19	256 x1, x2, x4, x8
ISSI	is25lp	is25lp512m	9d	60	1a	512 x1, x2, x4, x8
ISSI	is25w	is25wp01g	9d	70	1b	1024 x1, x2, x4, x8
ISSI	is25wp	is25wp016d	9d	70	15	16 x1, x2, x4
ISSI	is25wp	is25wp032d	9d	70	16	32 x1, x2, x4
ISSI	is25wp	is25wp064a	9d	70	17	64 x1, x2, x4
ISSI	is25wp	is25wp080d	9d	70	14	8 x1, x2, x4
ISSI	is25wp	is25wp128f	9d	70	18	128 x1, x2, x4
ISSI	is25wp	is25wp256d	9d	70	19	256 x1, x2, x4, x8
ISSI	is25wp	is25wp512m	9d	70	1a	512 x1, x2, x4
Macronix mx25l	mx25l12872f [mx25l12833f- mx25l12835f- mx25l12845g- spi-x1_x2_x4, mx25l12833f- mx25l12835f- mx25l12845g- spi- x1_x2_x4_x8]		c2	20	18	128 x1, x2, x4, x8
Macronix mx25l	mx25l25673g [mx25l25635f- mx25l25645g- spi-x1_x2_x4,		c2	20	19	256 x1, x2, x4

Manufacturer	Manufacturer Alias	Device Alias	MANUFACTURER	MEMORY TYPE	DATA WIDTH	Bits	
		mx25l25635f- mx25l25645g- spi- x1_x2_x4_x8]					
Macronix mx25l		mx25l3273f [mx25l3233f- spi-x1_x2_x4, mx25l3233f- spi- x1_x2_x4_x8]	c2	20	16	32	x1, x2, x4
Macronix mx25l		mx25l51273g [mx25l51245g- mx66l51235f- spi-x1_x2_x4, mx25l51245g- mx66l51235f- spi- x1_x2_x4_x8]	c2	20	1a	512	x1, x2, x4, x8
Macronix mx25l		mx25l6433f [mx25l6473f- spi-x1_x2_x4, mx25l6473f- spi- x1_x2_x4_x8]	c2	20	17	64	x1, x2, x4
Macronix mx25l		mx25v1635f	c2	23	15	16	x1, x2, x4
Macronix mx25l		mx25v8035f	c2	23	14	8	x1, x2, x4, x8
Macronix mx25lu		mx25u8035f [mx25u8033e- spi-x1_x2_x4, mx25u8033e- spi- x1_x2_x4_x8]	c2	25	34	8	x1, x2, x4, x8
Macronix mx25u		mx25u12872f [mx25u12832f- mx25u12835f- mx25u12843g- spi-x1_x2_x4, mx25u12832f-	c2	25	38	128	x1, x2, x4

Manufacturer	Manufacturer Alias	Device ID	MANUFACTURER	TYPE	MEMORY	DATA WIDTH	Bits
	mx25u12835f-[mx25u12843g-spi-x1_x2_x4_x8]						
Macronix mx25u	mx25u1635f [mx25u1632f-spi-x1_x2_x4, mx25u1632f-spi-x1_x2_x4_x8]	c2	25	35	16	x1, x2, x4, x8	
Macronix mx25u	mx25u25673g [mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4, mx25u25635f-mx25u25643g-mx25u25645g-spi-x1_x2_x4_x8]	c2	25	39	256	x1, x2, x4, x8	
Macronix mx25u	mx25u3235f [mx25u3232f-spi-x1_x2_x4, mx25u3232f-spi-x1_x2_x4_x8]	c2	25	36	32	x1, x2, x4, x8	
Macronix mx25u	mx25u51245g [mx66u51235f-spi-x1_x2_x4, mx66u51235f-spi-x1_x2_x4_x8]	c2	25	3a	512	x1, x2, x4, x8	
Macronix mx25u	mx25u6472f [mx25u6435f-mx25u6432f-spi-x1_x2_x4, mx25u6435f-mx25u6432f-spi-x1_x2_x4_x8]	c2	25	37	64	x1, x2, x4, x8	

Manufacturer	Manufacturer Alias	Device Alias	MANUFACTURER	MEMORY TYPE	DATA WIDTH	DATA WIDTH Bits	
Macronix	mx66l	mx66l1g45g	c2	20	1b	1024	x1, x2, x4
Macronix	mx66l	mx66l2g45g	c2	20	1c	2048	x1, x2, x4, x8
Macronix	mx66u	mx66u1g45g	c2	25	3b	1024	x1, x2, x4, x8
Macronix	mx66u	mx66u2g45g	c2	25	3c	2048	x1, x2, x4, x8
Micron	mt25ql	mt25ql01g	20	ba	21	1024	x1, x2, x4
Micron	mt25ql	mt25ql02g	20	ba	22	2048	x1, x2, x4, x8
Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4, n25q128-3.3v-spi-x1_x2_x4_x8]	20	ba	18	128	x1, x2, x4
Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4, n25q256-3.3v-spi-x1_x2_x4_x8]	20	ba	19	256	x1, x2, x4
Micron	mt25ql	mt25ql512	20	ba	20	512	x1, x2, x4
Micron	mt25qu	mt25qu01g	20	bb	21	1024	x1, x2, x4
Micron	mt25qu	mt25qu02g	20	bb	22	2048	x1, x2, x4, x8
Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4, n25q128-1.8v-spi-x1_x2_x4_x8]	20	bb	18	128	x1, x2, x4, x8

Manufacturer	Manufacturer Device ID	Device Alias	MANUFACTURER	TYPE	MEMORY	SIZE	DENSITY	VID	Data Width Bits
Micron	mt25qu	mt25qu256 [n25q256- 1.8v-spi- x1_x2_x4, n25q256- 1.8v-spi- x1_x2_x4_x8]	20	bb	19	256			x1, x2, x4, x8
Micron	mt25qu	mt25qu512	20	bb	20	512			x1, x2, x4, x8
Micron	n25q	n25q32-1.8v	20	bb	16	32			x1, x2, x4
Micron	n25q	n25q32-3.3v	20	bb	16	32			x1, x2, x4
Micron	n25q	n25q64-1.8v	20	bb	17	64			x1, x2, x4
Micron	n25q	n25q64-3.3v	20	bb	17	64			x1, x2, x4
Winbond	w25q	w25q256jw	ef	60	19	256			x1, x2, x4, x8

Virtex UltraScale+ Configuration Memory Devices

The Flash devices supported for configuration of AMD Virtex™ UltraScale+™ devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in the following table.

The tables in this Appendix are running lists per AMD family of non-volatile memories which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain components on this list even after they are no longer appropriate for new designs, to support long-term maintenance of end products which can contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life cycle. References to specific devices in the tables are not an assurance of their current or future availability.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Table: Supported Flash Memory Devices for Virtex UltraScale+ BPI Device Configuration

Manufacturer	Manufacturer Device ID	Device Alias	Manufacturer	Device ID	Device VID	Density	Mbits	Data Width Bits
Infineon	s29glxxp	s29gl01gp	1	2228	2201	1024		x16, x8
Infineon	s29glxxp	s29gl128p	1	2221	2201	128		x16, x8

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Bank Density	Mbits	Data Width	Bits
Infineon	s29glxxxp	s29gl256p	1	2222	2201		256		x16, x8	
Infineon	s29glxxxp	s29gl512p	1	2223	2201		512		x16, x8	
Infineon	s29glxxxs	s29gl01gs	1	2228	2201		1024		x16	
Infineon	s29glxxxs	s29gl128s	1	2221	2201		128		x16	
Infineon	s29glxxxs	s29gl256s	1	2222	2201		256		x16	
Infineon	s29glxxxs	s29gl512s	1	2223	2201		512		x16	
Infineon	s29glxxxs	s70gl02gs	1	2248	2201		2048		x16	
Infineon	s29glxxxt	s29gl01gt	1	2228	2201		1024		x16, x8	
Infineon	s29glxxxt	s29gl512t	1	2223	2201		512		x16, x8	
Infineon	s29glxxxt	s70gl02gt	1	2248	2201		2048		x16, x8	
Macronix	mx29gl	mx29gl128f	c2	2221	2201		128		x16, x8	
Macronix	mx29gl	mx29gl256f	c2	2222	2201		256		x16, x8	
Micron	g18	28f128g18f	89	-	-		128		x16	
Micron	g18	mt28gu01gaa 89 e [28f00ag18f- bpi-x16]		-	-		1024		x16	
Micron	g18	mt28gu256aa 89 e [28f256g18f- bpi-x16]		-	-		256		x16	
Micron	g18	mt28gu512aa 89 e [28f512g18f- bpi-x16]		-	-		512		x16	
Micron	m29ew	28f00am29ew89		2228	2201		1024		x16, x8	
Micron	m29ew	28f00bm29ew89		2248	2201		2048		x16, x8	
Micron	m29ew	28f064m29ew89		2210	2200		64		x16, x8	
Micron	m29ew	28f064m29ew89		220c	2201		64		x16, x8	
Micron	m29ew	28f064m29ew89		220c	2201		64		x16, x8	
Micron	m29ew	28f064m29ew89		2210	2201		64		x16, x8	
Micron	m29ew	28f128m29ew89		2221	2201		128		x16, x8	

Manufacturer	Manufacturer ID	Device Alias	Manufacturer ID	Device ID	Device Index	Density	Mbits	Data Width	Bits
Micron	m29ew	28f256m29ew89		2222	2201	256		x16, x8	
Micron	m29ew	28f512m29ew89		2223	2201	512		x16, x8	
Micron	m29w	m29w128gh	20	2221	2201	128		x16, x8	
Micron	m29w	m29w128gl	20	2221	2200	128		x16, x8	
Micron	m29w	m29w256gh	20	2222	2201	256		x16, x8	
Micron	m29w	m29w256gl	20	2222	2201	256		x16, x8	
Micron	mt28ew	mt28ew01ga	89	2228	2201	1024		x16, x8	
Micron	mt28ew	mt28ew128a	89	2221	2201	128		x16, x8	
Micron	mt28ew	mt28ew256a	89	2222	2201	256		x16, x8	
Micron	mt28ew	mt28ew512a	89	2223	2201	512		x16, x8	
Micron	mt28fw	mt28fw02gb	89	2248	2201	2048		x16	
Micron	p30	28f00ap30b	89	-	-	1024		x16	
Micron	p30	28f00ap30e	89	-	-	1024		x16	
Micron	p30	28f00ap30t	89	-	-	1024		x16	
Micron	p30	28f00bp30e	89	-	-	2048		x16	
Micron	p30	28f128p30b	89	-	-	128		x16	
Micron	p30	28f128p30t	89	-	-	128		x16	
Micron	p30	28f256p30b	89	-	-	256		x16	
Micron	p30	28f256p30t	89	-	-	256		x16	
Micron	p30	28f512p30b	89	-	-	512		x16	
Micron	p30	28f512p30e	89	-	-	512		x16	
Micron	p30	28f512p30t	89	-	-	512		x16	
Micron	p30	28f640p30b	89	-	-	64		x16	
Micron	p30	28f640p30t	89	-	-	64		x16	

Table: Supported Flash Memory Devices for Virtex UltraScale+ SPI Device Configuration

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Density	Mbits	Data Width	Bits
--------------	-----------------	--------	-------	-----------------	-----------	--------------	---------	-------	------------	------

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Density	Mbits	Data Width Bits
Infineon	s25fl	s25fl02g		-	-	-	2048		x1, x2, x4, x8
Infineon	s25fl1	s25fl116k	1		40	15	16		x1, x2, x4
Infineon	s25fl1	s25fl132k	1		40	16	32		x1, x2, x4
Infineon	s25fl1	s25fl164k	1		40	17	64		x1, x2, x4
Infineon	s25flxxxl	s25fl064l	1		60	17	64		x1, x2, x4
Infineon	s25flxxxl	s25fl128l	1		60	18	128		x1, x2, x4
Infineon	s25flxxxl	s25fl256l	1		60	19	256		x1, x2, x4
Infineon	s25flxxxp	s25fl032p	1		2	15	32		x1, x2, x4
Infineon	s25flxxxp	s25fl064p	1		2	16	64		x1, x2, x4
Infineon	s25flxxxs	s25fl128sxxxxlx0 [s25fl127s-spi-x1_x2_x4, s25fl127s-spi-x1_x2_x4_x8]			20	18	128		x1, x2, x4
Infineon	s25flxxxs	s25fl128sxxxxlx1			20	18	128		x1, x2, x4, x8
Infineon	s25flxxxs	s25fl256sxxxxlx0			2	19	256		x1, x2, x4, x8
Infineon	s25flxxxs	s25fl256sxxxxlx1			2	19	256		x1, x2, x4, x8
Infineon	s25flxxxs	s25fl512s	1		2	20	512		x1, x2, x4, x8
Infineon	s25hs	s25hs02gt		-	-	-	2048		x1, x2, x4, x8
Infineon	s25hs	s25hs512t	34		2b	1a	512		x1, x2, x4
ISSI	is25l	is25lp01g	9d		60	1b	1024		x1, x2, x4, x8
ISSI	is25lp	is25lp512m	9d		60	1a	512		x1, x2, x4, x8

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Bank Density	Mbits	Data Width Bits
ISSI	is25w	is25wp01g	9d	70	1b		1024		x1, x2, x4, x8
ISSI	is25wp	is25wp016d	9d	70	15		16		x1, x2, x4, x8
ISSI	is25wp	is25wp032d	9d	70	16		32		x1, x2, x4, x8
ISSI	is25wp	is25wp064a	9d	70	17		64		x1, x2, x4
ISSI	is25wp	is25wp080d	9d	70	14		8		x1, x2, x4
ISSI	is25wp	is25wp128f	9d	70	18		128		x1, x2, x4
ISSI	is25wp	is25wp256d	9d	70	19		256		x1, x2, x4
ISSI	is25wp	is25wp512m	9d	70	1a		512		x1, x2, x4
Macronix mx25l		mx25l12872f c2 [mx25l12833f- mx25l12835f- mx25l12845g- spi- x1_x2_x4, mx25l12833f- mx25l12835f- mx25l12845g- spi- x1_x2_x4_x8]		20	18		128		x1, x2, x4
Macronix mx25l		mx25l25673g c2 [mx25l25635f- mx25l25645g- spi- x1_x2_x4, mx25l25635f- mx25l25645g- spi- x1_x2_x4_x8]		20	19		256		x1, x2, x4
Macronix mx25l		mx25l51273g c2 [mx25l51245g- mx66l51235f- spi- x1_x2_x4,		20	1a		512		x1, x2, x4, x8

Manufacturer	Manufacturer Family	Alias	Manufacturer	Device	Device Index	Bank Density	Mbits	Data Width	Bits
		mx25l51245g- mx66l51235f- spi- x1_x2_x4_x8]							
Macronix	mx25lu	mx25u8035f_c2 [mx25u8033e- spi- x1_x2_x4, mx25u8033e- spi- x1_x2_x4_x8]		25	34	8		x1, x2, x4, x8	
Macronix	mx25u	mx25u12872fc2 [mx25u12832f- mx25u12835f- mx25u12843g- spi- x1_x2_x4, mx25u12832f- mx25u12835f- mx25u12843g- spi- x1_x2_x4_x8]		25	38	128		x1, x2, x4, x8	
Macronix	mx25u	mx25u1635f_c2 [mx25u1632f- spi- x1_x2_x4, mx25u1632f- spi- x1_x2_x4_x8]		25	35	16		x1, x2, x4, x8	
Macronix	mx25u	mx25u25673gc2 [mx25u25635f- mx25u25643g- mx25u25645g- spi- x1_x2_x4, mx25u25635f- mx25u25643g- mx25u25645g- spi- x1_x2_x4_x8]		25	39	256		x1, x2, x4	

Manufacturer	Manufacturer ID	Family	Alias	Manufacturer ID	Device ID	Device Index	Bank Density	Mbits	Data Width	Bits
Macronix	mx25u	mx25u3235f	c2 [mx25u3232f- spi- x1_x2_x4, mx25u3232f- spi- x1_x2_x4_x8]	25	36		32		x1, x2, x4	
Macronix	mx25u	mx25u51245gc2	[mx66u51235f- spi- x1_x2_x4, mx66u51235f- spi- x1_x2_x4_x8]	25	3a		512		x1, x2, x4	
Macronix	mx25u	mx25u6472f	c2 [mx25u6435f- mx25u6432f- spi- x1_x2_x4, mx25u6435f- mx25u6432f- spi- x1_x2_x4_x8]	25	37		64		x1, x2, x4, x8	
Macronix	mx66u	mx66u1g45g	c2	25	3b		1024		x1, x2, x4, x8	
Macronix	mx66u	mx66u2g45g	c2	25	3c		2048		x1, x2, x4, x8	
Micron	mt25qu	mt25qu01g	20	bb	21		1024		x1, x2, x4, x8	
Micron	mt25qu	mt25qu02g	20	bb	22		2048		x1, x2, x4	
Micron	mt25qu	mt25qu128	[n25q128- 1.8v-spi- x1_x2_x4, n25q128- 1.8v-spi- x1_x2_x4_x8]	20	bb	18	128		x1, x2, x4	

Manufacturer	Manufacturer ID	Device Alias	Manufacturer	Device ID	Device Index	Device Density	Mbits	Data Width	Bits
Micron	mt25qu	mt25qu256 [n25q256- 1.8v-spi- x1_x2_x4, n25q256- 1.8v-spi- x1_x2_x4_x8]		20	bb	19	256	x1, x2, x4	
Micron	mt25qu	mt25qu512		20	bb	20	512	x1, x2, x4	
Micron	n25q	n25q32- 1.8v		20	bb	16	32	x1, x2, x4	
Micron	n25q	n25q64- 1.8v		20	bb	17	64	x1, x2, x4	
Winbond	w25q	w25q256jw		ef	60	19	256	x1, x2, x4, x8	

Zynq 7000 Configuration Memory Devices

The flash devices supported for configuration of Zynq 7000 devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in following table.

The tables in this appendix are running lists per AMD family of non-volatile memories, which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain components on this list even after they are no longer appropriate for new designs to support long-term maintenance of end products which might contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life cycle. References to specific devices in the tables are not an assurance of their current or future availability.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Note: The U-Boot tags used to build the Configuration Memory Device Programmer are as follows:

Interface	U-Boot Tags
qspi	xilinx-v2015.2.01
nor	xilinx-14.3-build1
nand	xilinx-v2015.2.01

Table: Supported Flash Memory Devices for Zynq 7000 Device Configuration

Interface	Manufacturer	Manufacturer Part Number	Voltage	Data Width Bits
nand	Infineon	s34ml	1.8v	16, x8
nand	Infineon	s34ml	2.0v	16, x8
nand	Micron	mt29f	2.0v	8, x8
nand	Micron	mt29f	2.0v	16, x16
nor	Micron	m29ew	2.0v	256m29ewt
nor	Micron	m29ew	2.0v	64m29ewt
nor	Micron	m29ew	2.0v	128m29ewh
nor	Micron	m29ew	2.0v	256m29ewh
nor	Micron	m29ew	2.0v	512m29ewh
qspi	Infineon	s25flxxxl	1.8v	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxl	3.3v	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxxp	1.8v	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxxs	1.8v	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxxs	3.3v	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxxs	1.8v	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_parallel

Interface	Manufacturer	Manufacturer Part Number	Voltages	Data Width Bits
				dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxxs	826fl256s-x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel 3.3v	
qspi	Infineon	s25flxxxxs	520fl512s-x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel 1.8v	
qspi	Infineon	s25flxxxxs	520fl512s-x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel 3.3v	
qspi	Infineon	s25hs	804ls02gtx1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s70flxxxxp	802401gs_x10dual_stacked	
qspi	ISSI	is25l	is25lp01g x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp016x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp032x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp064x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp080x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-	

Interface	Manufacturer	Manufacturer Part Number	Family	Attributes	Data Width Bits
				dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp128	128fx1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp256	256fx1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp512	512fx1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25w	is25wp01g	1g1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp016d	16d1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp032d	32d1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp064d	64d1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp080d	80d1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp128f	128f1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-	

Interface	Manufacturer	Manufacturer Part Number	Family	Variant	Data Width Bits
					dual_stacked, x4-single, x8-dual_parallel
qspi	ISSI	is25wp	IS25WP	256	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	ISSI	is25wp	IS25WP	512	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	128	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	256	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	384	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	512	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	768	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	1024	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	1280	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	1536	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	1792	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	2048	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	2304	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	MX25L	2560	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel

Interface	Manufacturer	Manufacturer Part Number	Family	Variant	Data Width Bits
					dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	51225l51245	g	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25l	51225l51273	g	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25u	12825u12885	f	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25u	12825u12843	f	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25u	12825u12872	f	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25u	25625u25685	f	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25u	25625u25673	f	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25u	51225u51245	f	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25u	6425u6472f	f	-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-

Interface	Manufacturer	Manufacturer Part Number	Family	Capacity	Data Width Bits
					dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx25u	61x25u8035f	1g	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx66l	10266l1g45g	1g	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx66l	20466l2g45g	1g	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx66l	51166l51235f	1g	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx66u	10266u1g45g	1g	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Macronix	mx66u	20466u2g45g	1g	dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Micron	mt25ql	1025ql01g	1g	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Micron	mt25ql	2028ql02g	1g	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Micron	mt25ql	10285ql128	1g	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-

Interface	Manufacturer	Manufacturer Part Number	Family	Available Bits	Data Width Bits
					dual_stacked, x4-single, x8-dual_parallel
qspi	Micron	mt25ql	2505ql	256x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	5125ql	512x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	1025qu01	1g1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	2028qu02	2g1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	1285qu12	1281-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	2505qu25	2561-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	5125qu51	5121-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	n25q	625q64-1.8v		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Micron	n25q	625q64-3.3v		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-

Interface	Manufacturer	Manufacturer Part Number	Density Mbits	Data Width Bits
				dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25h	128M02jv	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25q	128fx128fw	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25q	128fw	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel

 **Note:** The Macronix MX66U2G45G_54 sub-family is not supported.

Zynq UltraScale+ MPSoC Configuration Memory Devices

The flash devices supported for configuration of Zynq UltraScale+ MPSoC devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in the following table.

The tables in this appendix are running lists per AMD family of non-volatile memories which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain components on this list even after they are no longer appropriate for new designs, to support long-term maintenance of end products which might contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life cycle. References to specific devices in the tables are not an assurance of their current or future availability.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Table: Supported Flash Memory Devices for Zynq UltraScale+ MPSoC Device Configuration

Interface	Manufacturer	Manufacturer Part Number	Device Alias	Density Mbits	Data Width Bits
nand	Infineon	s34ml	s34ml01g1	1024	x8-dual, x8-single
nand	Infineon	s34ml	s34ml02g1	2048	x8-dual, x8-single

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
nand	Micron	mt29f	mt29f16g08ab	16384	x8-dual, x8-single	
nand	Micron	mt29f	mt29f2g08ab	2048	x8-dual, x8-single	
nand	Micron	mt29f	mt29f32g08ae	32768	x8-dual, x8-single	
nand	Micron	mt29f	mt29f64g08ae	65536	x8-dual, x8-single	
nand	Micron	mt29f	mt29f8g08ab	8192	x8-dual, x8-single	
emmc	Micron	mtfc	mtfc8gakajcn-4m	65536	-	
emmc	-	-	jedec4.51	524288	-	
emmc	-	-	jedec4.51-16gb	131072	-	
emmc	-	-	jedec4.51-32gb	262144	-	
emmc	-	-	jedec4.51-4gb	32768	-	
emmc	-	-	jedec4.51-64gb	524288	-	
emmc	-	-	jedec4.51-8gb	65536	-	
qspi	Infineon	s25flxxxl	s25fl256l	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s25flxxxp	s25fl129p	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s25flxxxs	s25fl128s-1.8v	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s25flxxxs	s25fl128s-3.3v	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Infineon	s25flxxxS	s25fl256s-1.8v	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s25flxxxS	s25fl256s-3.3v	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s25flxxxS	s25fl512s-1.8v	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s25flxxxS	s25fl512s-3.3v	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s25hs	s25hs02gt	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s70flxxxp	s70fl01gs_00	1024	x4-dual_stacked	
qspi	GigaDevice	gd25b	gd25b512me	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25l	is25lp01g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp016d	16	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp032d	32	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	ISSI	is25lp	is25lp080d	8	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp128f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp256d	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp512m	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25w	is25wp01g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp016d	16	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp032d	32	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp064a	64	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp080d	8	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	ISSI	is25wp	is25wp128f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp256d	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp256e	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp512m	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l12833f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l12835f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l12872f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l25635f	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l25673g	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Macronix	mx25l	mx25l3273f	32	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l51245g	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l51273g	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u12835f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u12843g	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u12872f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u25635f	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u25673g	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u51245g	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Macronix	mx25u	mx25u6472f	64	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u8035f	8	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66l	mx66l1g45g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66l	mx66l2g45g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66u	mx66u1g45g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66u	mx66u2g45g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66u	mx66u51235f	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql01g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql02g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Micron	mt25ql	mt25ql128	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql256	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql512	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu01g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu02g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu128	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu256	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu512	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	n25q	n25q64-1.8v	64	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Micron	n25q	n25q64-3.3v	64		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25h	w25h02jv	2048		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25q	w25q02nw	2048		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25q	w25q128fw	128		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25q	w25q256jw	256		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25q	w25q256jwm	256		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel

 **Note:** The Macronix MX66U2G45G_54 sub-family is not supported.

Zynq UltraScale+ RFSoC Configuration Memory Devices

The flash devices supported for configuration of Zynq UltraScale+ RFSoC devices that can be erased, blank checked, programmed, and verified by AMD Vivado™ software are shown in the following table.

The tables in this appendix are running lists per AMD family of non-volatile memories which Vivado software is capable of erasing, blank checking, programming, and verifying. AMD strives to retain components on this list even after they are no longer appropriate for new designs, to support long-term maintenance of end products which might contain them.

!! Important: Given the evolving nature of the commodity non-volatile memory market, AMD recommends contacting your non-volatile memory supplier to confirm device availability and life

cycle. References to specific devices in the tables are not an assurance of their current or future availability.

!! Important: Flash devices manufactured by Spansion are now known as Infineon. There is no functionality difference as long as the part number is the same.

Table: Supported Flash Memory Devices for Zynq UltraScale+ RFSoC Device Config

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
nand	Infineon	s34ml	s34ml01g1	1024	x8-dual, x8-single	
nand	Infineon	s34ml	s34ml02g1	2048	x8-dual, x8-single	
nand	Micron	mt29f	mt29f16g08ab	16384	x8-dual, x8-single	
nand	Micron	mt29f	mt29f2g08ab	2048	x8-dual, x8-single	
nand	Micron	mt29f	mt29f32g08ae	32768	x8-dual, x8-single	
nand	Micron	mt29f	mt29f64g08ae	65536	x8-dual, x8-single	
nand	Micron	mt29f	mt29f8g08ab	8192	x8-dual, x8-single	
emmc	Micron	mtfc	mtfc8gakajcn-4m	65536	-	
emmc	-	-	jedec4.51	524288	-	
emmc	-	-	jedec4.51-16gb	131072	-	
emmc	-	-	jedec4.51-32gb	262144	-	
emmc	-	-	jedec4.51-4gb	32768	-	
emmc	-	-	jedec4.51-64gb	524288	-	
emmc	-	-	jedec4.51-8gb	65536	-	
qspi	Infineon	s25flxxxl	s25fl256l	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s25flxxxp	s25fl129p	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Infineon	s25flxxxS	s25fl128s-1.8v	128		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxS	s25fl128s-3.3v	128		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxS	s25fl256s-1.8v	256		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxS	s25fl256s-3.3v	256		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxS	s25fl512s-1.8v	512		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxS	s25fl512s-3.3v	512		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25hs	s25hs02gt	2048		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s70flxxxp	s70fl01gs_00	1024		x4-dual_stacked
qspi	GigaDevice	gd25b	gd25b512me	256		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	ISSI	is25l	is25lp01g	1024		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	ISSI	is25lp	is25lp016d	16	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp032d	32	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp080d	8	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp128f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp256d	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp512m	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25w	is25wp01g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp016d	16	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp032d	32	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	ISSI	is25wp	is25wp064a	64	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp080d	8	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp128f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp256d	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp256e	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp512m	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l12833f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l12835f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l12872f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Macronix	mx25l	mx25l25635f	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l25673g	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l3273f	32	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l51245g	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l51273g	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u12835f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u12843g	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u12872f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u25635f	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Macronix	mx25u	mx25u25673g	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u51245g	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u6472f	64	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u8035f	8	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66l	mx66l1g45g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66l	mx66l2g45g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66u	mx66u1g45g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66u	mx66u2g45g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66u	mx66u51235f	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Micron	mt25ql	mt25ql01g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql02g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql128	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql256	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql512	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu01g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu02g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu128	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu256	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Micron	mt25qu	mt25qu512	512		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Micron	n25q	n25q64-1.8v	64		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Micron	n25q	n25q64-3.3v	64		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25h	w25h02jv	2048		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25q	w25q02nw	2048		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25q	w25q128fw	128		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25q	w25q256jw	256		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Winbond	w25q	w25q256jwm	256		x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel

 **Note:** The Macronix MX66U2G45G_54 sub-family is not supported.

Versal Configuration Memory Devices

 **Note:** For flash memory devices with official industry specifications, such as eMMC and SD, the device support is based on compliance to specification versions, rather than by enumeration of

specific manufacturer, manufacturer family, and device part number. This provides designers with a wider range of possibilities in initial flash selection and long-term design maintenance.

Note: When selecting QSPI or OSPI density, select the maximum, because the maximum density also supports lower density devices. For example, selecting 2 Gb density also supports lower density devices such as 1 Gb or 512 Mb. This is unlike previous architectures.

Table: Supported Flash Memory Devices for Versal Device Configuration

Interface	Manufacturer	Device Alias	Density Mbits	Data Width Bits
emmc	-	jedec4.51	524288	-
emmc	-	jedec4.51-16gb	131072	-
emmc	-	jedec4.51-32gb	262144	-
emmc	-	jedec4.51-4gb	32768	-
emmc	-	jedec4.51-64gb	524288	-
emmc	-	jedec4.51-8gb	65536	-
qspi	Infineon	s25flxxxl	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxp	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxs	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxs	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
qspi	Infineon	s25flxxxs	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Infineon	s25flxxxS	s25fl256s-3.3v	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s25flxxxS	s25fl512s-1.8v	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s25flxxxS	s25fl512s-3.3v	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s25hs	s25hs02gt	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Infineon	s70flxxxp	s70fl01gs_00	1024	x4-dual_stacked	
qspi	GigaDevice	gd25b	gd25b512me	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25l	is25lp01g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp016d	16	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp032d	32	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp080d	8	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	ISSI	is25lp	is25lp128f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp256d	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25lp	is25lp512m	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25w	is25wp01g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp016d	16	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp032d	32	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp064a	64	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp080d	8	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp128f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	ISSI	is25wp	is25wp256d	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp256e	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	ISSI	is25wp	is25wp512m	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l12833f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l12835f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l12872f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l25635f	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l25673g	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l3273f	32	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Macronix	mx25l	mx25l51245g	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25l	mx25l51273g	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u12835f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u12843g	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u12872f	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u25635f	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u25673g	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u51245g	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx25u	mx25u6472f	64	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Macronix	mx25u	mx25u8035f	8	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66l	mx66l1g45g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66l	mx66l2g45g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66u	mx66u1g45g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66u	mx66u2g45g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Macronix	mx66u	mx66u51235f	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql01g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql02g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql128	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Micron	mt25ql	mt25ql256	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25ql	mt25ql512	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu01g	1024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu02g	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu128	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu256	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	mt25qu	mt25qu512	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	n25q	n25q64-1.8v	64	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Micron	n25q	n25q64-3.3v	64	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	

Interface	Manufacturer	Manufacturer	Device Alias	Density	Mbits	Data Width Bits
qspi	Winbond	w25h	w25h02jv	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Winbond	w25q	w25q02nw	2048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Winbond	w25q	w25q128fw	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Winbond	w25q	w25q256jw	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
qspi	Winbond	w25q	w25q256jwm	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel	
ospi	Micron	mt35	mt35xu02gcba1g12,048 osit		x8-dual_stacked, x8-single	
ospi	GigaDevice	gd25lx	gd25lx256e	256	x8-dual_stacked, x8-single	
ospi	Gigadevice	gd25	gd25lx512m	512	x8-dual_stacked, x8-single	
ospi	Gigadevice	gd55	gd55lx01g	1024	x8-dual_stacked, x8-single	
ospi	Gigadevice	gd55	gd55lx02g	2048	x8-dual_stacked, x8-single	

 Note: The Macronix MX66U2G45G_54 sub-family is not supported.

Vendor Validated Flash Table

Table: Vendor Validated Flash Table

Interface	Manufacturer	Manufacturer	Device Name	Vivado Select	Density (Mb)	Architecture Support
QSPI	GigaDevice	GD25B	GD25B16E	cfgmem-qspi-<data_width_bits>	16	Zynq UltraScale+, Versal

Interface	Manufacturer	Manufacturer	Device Name	Vivado Select	Density (Mb)	Architecture Support
QSPI	GigaDevice	GD25B	GD25B32E	cfgmem-qspi-<data_width_bits>	32	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25B	GD25B64E	cfgmem-qspi-<data_width_bits>	64	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25B	GD25B128E	cfgmem-qspi-<data_width_bits>	128	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25B	GD25B256E	cfgmem-qspi-<data_width_bits>	256	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25B	GD25B512M	cfgmem-qspi-<data_width_bits>	512	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25B	GD55B01GE	cfgmem-qspi-<data_width_bits>	1024	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25B	GD55B02GE	cfgmem-qspi-<data_width_bits>	2046	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25F	GD25F64F	cfgmem-qspi-<data_width_bits>	64	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25F	GD25F128F	cfgmem-qspi-<data_width_bits>	128	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25F	GD25F256F	cfgmem-qspi-<data_width_bits>	256	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25F	GD55F512M	cfgmem-qspi-<data_width_bits>	512	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25T	GD25T512M	cfgmem-qspi-	512	Zynq UltraScale+,

Interface	Manufacturer	Manufacturer	Device Name	Vivado Select	Density (Mb)	Architecture Support
				<data_width_bits>		Versal
QSPI	GigaDevice	GD25T	GD55T01GE	cfgmem-qspi-<data_width_bits>	1024	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25T	GD55T02GE	cfgmem-qspi-<data_width_bits>	2048	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LB	GD25LB16E	cfgmem-qspi-<data_width_bits>	16	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LB	GD25LB32E	cfgmem-qspi-<data_width_bits>	32	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LB	GD25LB64E	cfgmem-qspi-<data_width_bits>	64	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LB	GD25LB128E	cfgmem-qspi-<data_width_bits>	128	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LB	GD25LB256E	cfgmem-qspi-<data_width_bits>	256	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LB	GD25LB512ME	cfgmem-qspi-<data_width_bits>	512	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LB	GD55LB01GE	cfgmem-qspi-<data_width_bits>	1024	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LB	GD55LB02GE	cfgmem-qspi-<data_width_bits>	2048	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LF	GD25LF80E	cfgmem-qspi-<data_width_bits>	8	Zynq UltraScale+, Versal

Interface	Manufacturer	Manufacturer	Device Name	Vivado Select	Density (Mb)	Architecture Support
QSPI	GigaDevice	GD25LF	GD25LF16E	cfgmem-qspi-<data_width_bits>	16	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LF	GD25LF32E	cfgmem-qspi-<data_width_bits>	32	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LF	GD25LF64E	cfgmem-qspi-<data_width_bits>	64	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LF	GD25LF128E	cfgmem-qspi-<data_width_bits>	128	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LF	GD25LF255E	cfgmem-qspi-<data_width_bits>	256	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LF	GD55LF511M	cfgmem-qspi-<data_width_bits>	512	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LT	GD25LT256E	cfgmem-qspi-<data_width_bits>	256	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LT	GD25LT512M	cfgmem-qspi-<data_width_bits>	512	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LT	GD55LT01G	cfgmem-qspi-<data_width_bits>	1024	Zynq UltraScale+, Versal
QSPI	GigaDevice	GD25LT	GD55LT02G	cfgmem-qspi-<data_width_bits>	2048	Zynq UltraScale+, Versal

Command Line Options for hw_server

The following is a list of hw_server command line options.

!! Important: When remotely connecting to a hw_server running on a PC, ensure that the firewall policy is properly configured. Make sure that `hw_server.exe` has permission to listen for new

socket connections on port 3121.

Standard hw_server Options

Table: Standard hw_server Options

Option	Purpose	Example
-d	Run hw_server in daemon mode (output is sent to system logger).	hw_server -d
--help	Basic command line help.	
-I	Exit if there are no target connections established for the specified time. Time here is specified in seconds.	hw_server -I 20
-L	Enable logging of JTAG commands from clients to the hw_server.	hw_server -L- The previous option logs output to stdout hw_server -Lmy_file.log The previous option logs output to file my_file.log
-s	Sets up an agent listening port and protocol.	hw_server -stcp::3122 The previous option is used to connect when using the local host with port 3122 at hw_server start up.
-q	Do not display version information on start up.	
-p	Assign a range of ports to be used by AMD GDB server, or disable the feature.	hw_server -p<port> <port> is the base port number for GDB server. Default is 3000 The server opens one port per target architecture: 3000: Arm® ; 3001: Arm64; 3002: MicroBlaze™ ; 3003: MicroBlaze64 hw_server -p0 disables the port
--init	This option is used to specify an initialization script file to hw_server	hw_server --init my_init.txt The file my_init.txt contains options that are applied to the hw_server at startup
	Environment variable HW_SERVER_INIT_FILE	export HW_SERVER_INIT_FILE=~/my_init.txt hw_server

Option	Purpose	Example
	Use to specify default --init file	In this case, the hw_server is initialized with the settings coming from the file pointed to by the environment variable HW_SERVER_INIT_FILE.

Advanced Options

Table: Advanced Option

Option	Purpose
detect-ir-length	<p>Disable detecting the IR detect length during scan chain discovery. This option is used to start up the hw_server with additional devices added to its device table. By default the hw_server starts with a preset list of devices that are compiled in from the XICOM SQL jtag master table. These settings can be overridden or extended using the this set device-info-file option. The file that is specified is a .csv file that ignores lines that start with a "#".</p> <p>This is how you specify a .csv file at hw_server start-up:</p> <pre>hw_server -e "set device-info-file my_file.csv"</pre> <p>The .csv file is formatted as follows: (hw_server_device_info_file.csv):</p> <pre>##### ## # File: hw_server_device_info_file.csv # Description: # This is a sample jtag ID table. This file can be used to define # additional devices to be detected by the hw_server application. # # In this file empty lines, lines with spaces, and lines that begin # with the '#' character will be ignored. # # The format of this file is as follows: # # ROW 1: fields # The standard JTAG id fields are "idcode,mask,irlen,name" # ROW 2: field types # For each field specified in ROW 1, the type is used to # interpret the field data read per device. The types # accepted are "i" for integer and "s" for string. If</pre>

Option	Purpose
	<pre># you use the standard fields on ROW 1 then you should # use "i,i,i,s" in ROW 2 to set the fields idcode,mask # irlength to integer and name as string # # To use this table you start hw_server with the # following # command line arguments: # # hw_server -e "set device-info-file <file>" # # Where <file> is replaced with this filename. ##### ### idcode,mask,irlen,name i,i,i,s 167784595,268435455,10,chipscope_soft_tap</pre>
device-info-file	Sets a default device file .csv to be used
max-jtag-devices	<p>Increases the max number of devices that can be detected in a scan chain. Default is 32.</p> <p>This option is used to start up the hw_server with the ability to detect more than the default number of devices in a scan chain. The default value for this setting is 32. You can increase this value for longer jtag chains.</p> <hr/> <p> Note: Increasing this number slows down the device discovery process, which in turn can slow down cable access. Therefore, it is only for systems with large device counts should you increase this value.</p>
	<p>This is how to specify the option at hw_server start-up:</p> <pre>hw_server -e "set max-jtag-devices 64"</pre>
xdb-user-bscan	<p>Sets which bscan are used to scan for xsdb cores.</p> <p>This option is used to start the hw_server scanning a for xsdb master cores on alternate bscans. By default, hw_server scans user 1 and 3 bscans. With this option you can launch hw_server and have it look for bscans on different bscan user slots.</p> <p>This is how to specify the option at hw_server start-up:</p> <pre>hw_server -e "set xsdb-user-bscan 1,2,3,4"</pre>

Option	Purpose												
	<p>The arguments to this parameter specify the list of parameters. The list is a comma separated list ranging from 1-4. The minimum number specified is 1 element in the list and the maximum is 4.</p>												
mdm-detect-bscan-mask	<p>Sets which bscan are used to scan for mdm cores.</p> <p>This option is used to start the hw_server scanning a for MicroBlaze master cores on alternate bscans. By default hw_server scans user 2 bscan. With this option the you can launch hw_server and have it look for MicroBlaze on different bscan user slots.</p> <p>This is how to specify the option at hw_server start-up:</p> <pre data-bbox="551 620 1155 650">hw_server -e "set mdm-detect-bscan-mask 2"</pre> <p>The bitmask is for any FPGA or adaptive SoC discovered by hw_server. Here are some examples of common bitmask settings:</p> <table data-bbox="518 783 975 1022"> <thead> <tr> <th data-bbox="518 783 682 813">Mask Value</th> <th data-bbox="682 783 975 813">BSCAN Scanned</th> </tr> </thead> <tbody> <tr> <td data-bbox="518 825 551 855">0</td> <td data-bbox="682 825 747 855">none</td> </tr> <tr> <td data-bbox="518 868 551 897">1</td> <td data-bbox="682 868 755 897">User1</td> </tr> <tr> <td data-bbox="518 910 551 939">3</td> <td data-bbox="682 910 853 939">User1, User2</td> </tr> <tr> <td data-bbox="518 952 551 982">7</td> <td data-bbox="682 952 943 982">User1, User2, User3</td> </tr> <tr> <td data-bbox="518 994 551 1024">f</td> <td data-bbox="682 994 1024 1024">User1, User2, User3, User4</td> </tr> </tbody> </table>	Mask Value	BSCAN Scanned	0	none	1	User1	3	User1, User2	7	User1, User2, User3	f	User1, User2, User3, User4
Mask Value	BSCAN Scanned												
0	none												
1	User1												
3	User1, User2												
7	User1, User2, User3												
f	User1, User2, User3, User4												
always-open-jtag	<p>Forces hw_server to open up all targets at start up.</p> <p>When hw_server starts by default no cables are initialized. When the first connection is initiated the cables are discovered and opened. This discovery period takes a couple of seconds to several minutes depending on the system. Once fully initialized the cables are read and devices discovered.</p> <p>In some cases, it is necessary to have the cables discovered and ready to be used. For instance when setting up a linux system as a board server, it can be desirable to have the cables always initialized and ready to serve connections. For these cases you can use the always-open-jtag option to force the cables to open.</p> <p>By default, this is the setting at start up (no argument needs to be passed in):</p> <pre data-bbox="551 1649 1090 1679">hw_server -e "set always-open-jtag 0"</pre> <p>To force opening the cable, pass in this argument as follows:</p> <pre data-bbox="551 1803 1090 1833">hw_server -e "set always-open-jtag 1"</pre>												
auto-open-servers	<p>Opens a cable server with specific parameters used for XVC cable opening.</p>												

Option	Purpose
	<p>This option is a debug option used to automatically open an XVC cable connection, and to control which types of USB cables <code>hw_server</code> should detect. The argument to the <code>auto-open-servers</code> parameter is a comma separated list of fields. Each field is a colon separated list of sub-fields where the first sub-field is the type and the remaining sub-fields are type specific. For the <code>xilinx-xvc</code> type the sub-fields are host and port. The default value for <code>auto-open-servers</code> is "*" which indicates that <code>hw_server</code> should detect all types of cables that it can. Cables types that require parameters, like XVC, are not covered by "*". This is how to specify the option at <code>hw_server</code> start-up:</p> <pre data-bbox="551 635 1225 699"><code>hw_server -e "set auto-open-servers xilinx- xvc:localhost:10200"</code></pre> <p>To open up two servers you would use:</p> <pre data-bbox="551 840 1225 903"><code>hw_server -e "set auto-open-servers xilinx- xvc:localhost:10200,xilinx-xvc:localhost:10210"</code></pre> <p>To open up two XVC servers in addition to all USB based cables you would use:</p> <pre data-bbox="551 1079 1258 1142"><code>hw_server -e "set auto-open-servers *:xilinx- xvc:localhost:10200,xilinx-xvc:localhost:10210"</code></pre>
auto-open-ports	<p>Controls automatic open of ports (cables or scan chains). This option is used to control the automatic open of JTAG scan chains. The argument to the <code>auto-open-ports</code> parameter is a boolean value, 1 indicating that all known JTAG scan chains should automatically be opened and 0 indicating that clients open select JTAG scan chains. The default value is 1. Setting this to 0 is useful for example, when multiple JTAG scan chains are connected to a single host and different instances of <code>hw_server</code> are used to access each scan chain.</p> <p>This is how to specify the option at <code>hw_server</code> start-up:</p> <pre data-bbox="551 1643 1073 1670"><code>hw_server -e "set auto-open-ports 0"</code></pre>
xvc-timeout	<p>Changes the XVC timeout value to help debug XVC servers. This option is a debug option used to increase the timeout period needed for an XVC transaction to terminate. The argument to the <code>xvc-timeout</code> parameter is a time in seconds. A value of 0 disables timeout and thus wait indefinitely.</p> <p>This is how you specify the option at <code>hw_server</code> start-up:</p>

Option	Purpose
	<pre data-bbox="551 196 1041 223">hw_server -e "set xvc-timeout 100"</pre>
xvc-servers	<p>Start XVC servers for cables.</p> <p>This option is used make <code>hw_server</code> start XVC servers for specified JTAG cables. The argument to the <code>xvc-servers</code> parameter is a comma (,) separated list of XVC server descriptions. Each XVC server description is a colon (:) separated list containing the cable identification, XVC server host name or number and port number. The cable identification is the manufacturer name and a unique identifier separated by slash (/) characters. The cable identification can be a subset of the full cable identification and the host name can be empty. If it is empty, it indicates that the server should listen for incoming connections on all network interfaces.</p> <p>See <code>processor-debug-claim</code> for how to avoid interference when both XVC client and XVC server are debuggers of the same device types and XVC locking mode is used.</p> <p>This is how to specify the option at <code>hw_server</code> start-up:</p> <pre data-bbox="551 988 975 1051">hw_server -e "set xvc-servers 210203356596A:localhost:3000"</pre> <hr/> <p> Note: You still have to connect to this <code>hw_server</code> instance first to initialize the cable interface.</p> <hr/> <p>If you do not connect to the cable, you see a message like the following:</p> <pre data-bbox="551 1269 1286 1332">TCF 19:11:02.417: XVC open port failed: Cannot find JTAG cable matching 210203A0314DA</pre> <p>To have the XVC cable automatically opened and locked to XVC add the <code>always-open-jtag</code> option as follows:</p> <pre data-bbox="551 1507 1325 1613">hw_server -e 'set xvc-servers 210203A0314DA:xcoatslab- 9:3122' -e 'set always-open-jtag 1'</pre>
xvc-packet-len	<p>Change max package length of XVC servers.</p> <p>This option controls the XVC package length returned by XVC servers started using the <code>xvc-servers</code> option. Current default package length is 16000, however this can change in newer versions.</p> <p>This is how to specify the option at <code>hw_server</code> start-up:</p>

Option	Purpose								
	<pre>hw_server -e "set xvc-servers 210203356596A:localhost:3000" -e "set xvc-packet-len 1000"</pre>								
xvc-version	<p>Change XVC protocol version of XVC servers.</p> <p>This is a debug option that can be used together with xvc-servers to control which XVC protocol version is exposed to XVC clients. The current default protocol version is 1.1, however this can change if new versions of the XVC protocol are defined.</p> <p>This is how to specify the option at hw_server start-up:</p> <pre>hw_server -e "set xvc-servers 210203356596A:localhost:3000" -e "set xvc-version 1.0"</pre>								
xvc-capabilities	<p>Change XVC capabilities of XVC servers.</p> <p>This is a debug option that can be used together with xvc-servers to control which capabilities are exposed to XVC clients. Current default capabilities are locking, status, and state-aware, however additional capabilities can be added in future versions. This option has no effect if xvc-version is set to 1.0.</p> <p>This is how to specify the option at hw_server start-up:</p> <pre>hw_server -e "set xvc-servers 210203356596A:localhost:3000" -e "set xvc-capabilities status,state-aware"</pre>								
processor-debug-claim	<p>Automatically claim select device types to prevent debugger from using them.</p> <p>This option is used to prevent hw_server debugger from using selected device types for debugging. The argument given to this option is a bit mask. By default, hw_server uses all known device types for debugging. Bit number meaning</p> <table> <tr> <td>Bit</td> <td>Device Type</td> </tr> <tr> <td>0</td> <td>Arm DAP</td> </tr> <tr> <td>1</td> <td>MPSOC</td> </tr> <tr> <td>2</td> <td>FPGA or adaptive SoC</td> </tr> </table> <p>This option is useful for example when starting XVC server using xvc-servers with an XVC client that is a debugger of one or more of the previous devices, or when hw_server connects to an XVC server that is a debugger of one or more of the previous devices. In both cases this is</p>	Bit	Device Type	0	Arm DAP	1	MPSOC	2	FPGA or adaptive SoC
Bit	Device Type								
0	Arm DAP								
1	MPSOC								
2	FPGA or adaptive SoC								

Option	Purpose
	<p>only relevant when using the locking capability because that allows time sharing between the debuggers.</p> <p>This is how to specify the option at hw_server start-up:</p> <pre data-bbox="551 340 1155 371">hw_server -e "set processor-debug-claim 2"</pre>
jtag-poll-delay	<p>Delay in uS. Default 50000.</p> <p>This option is a polling option used to reduce the JTAG polling frequency. The JTAG polling frequency is the smallest period taken between JTAG poll operations. The default and smallest value is 50,000 uS. The argument to the jtag-poll-delay parameter is a number in uS.</p>
help	<p>Displays hw_server "e" options</p> <p>This option is used to display all the available "e" options for hw_server.</p> <p>This is how to specify the option at hw_server start-up:</p> <pre data-bbox="551 931 801 963">hw_server -e help</pre>
show-all	<p>Shows all options passed set in hw_serv.</p> <p>This option is used to display all the "e" option settings for hw_server.</p> <p>This is how you specify the option at hw_server start-up:</p> <pre data-bbox="551 1195 858 1227">hw_server -e show-all</pre>
jtag-default-frequency	<p>Sets default frequency for all cables.</p> <p>This option sets the default JTAG TCK frequency. The jtag-default-frequency parameter is a number in Hz.</p> <p>This is how to specify the option at hw_server start-up:</p> <pre data-bbox="551 1512 1258 1543">hw_server -e "set jtag-default-frequency 5000000"</pre>
jtag-port-filter	<p>Set JTAG port filter.</p> <p>This option is used to control JTAG port filtering. When it is set, hw_server ignores JTAG ports that do not match the filter. The argument to this parameter is a comma separated list of complete or partial port identifiers. A port identifier is a string of the form <manufacturer>/<productid>/<serial><port>. The filter matches when any of the port filter's strings can be found anywhere in the port identifier string.</p>

Option	Purpose
	<p>This parameter is useful when running multiple instances of hw_server on the same host to specify which cable should be handled by which hw_server.</p> <p>This is how to specify the option at hw_server start-up:</p> <pre data-bbox="551 382 1041 451">hw_server -e "set jtag-port-filter Xilinx/DLC10/0000128f515601"</pre> <p>Another example filtering any AMD DLC9 or DLC10 cable while having the hw_server start on port 3122:</p> <pre data-bbox="551 614 1209 684">hw_server -stcp::3122 -e "set jtag-port-filter DLC9,DLC10"</pre>
bscan-switch-user-mask	<p>Enables bscan switch.</p> <p>This option is used to control the bscan switch detection.</p> <p>This is how to specify the option at hw_server start-up:</p> <pre data-bbox="551 931 1364 963">hw_server -e "set bscan-switch-user-mask <user-bit-mask>"</pre>
jtag-port-devices	<p>Sets JTAG port device list.</p> <p>This option is used to specify a static list of devices for a JTAG scan chain. When this is specified hw_server does not read the IDCODE registers to detect devices on the scan chain. This is useful when a scan chain contains devices that do not conform to the IEEE 1149.1 specification. The value given to this parameter is a comma separated list of IDCODE values in the same order as the devices on the scan chain.</p> <p>This is how to specify the option at hw_server start-up:</p> <pre data-bbox="551 1459 1057 1529">hw_server -e "set jtag-port-devices 0xe970203f,0x03632093"</pre>
max-ir-length	<p>Enables devices in JTAG chain whose ir length is greater than 64 bits.</p> <p>This option is used to start up the hw_server with the ability to enable ir lengths greater than 64 bits. The default value for this setting is 64. You can increase this value for devices in the JTAG chains whose ir length is wide (for example, 93).</p> <hr/> <p> Note: Increasing this number slows down the device discovery process which in turn can slow down cable access.</p> <hr/> <p>Therefore, only for systems with long ir lengths device counts should you increase this value.</p>

Option	Purpose
	<p>This is how to specify the option at hw_server start-up:</p> <pre data-bbox="551 264 1057 291">hw_server -e "set max-ir-length 93"</pre>

Additional Resources and Legal Notices

Finding Additional Documentation

Documentation Portal

The AMD Adaptive Computing Documentation Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Documentation Portal, go to <https://docs.xilinx.com>.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the AMD Vivado™ IDE, select Help > Documentation and Tutorials.
- On Windows, click the Start button and select Xilinx Design Tools > DocNav.
- At the Linux command prompt, enter docnav.

 **Note:** For more information on DocNav, refer to the *Documentation Navigator User Guide (UG968)*.

Design Hubs

AMD Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the Design Hubs View tab.
- Go to the [Design Hubs](#) web page.

Support Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Support](#).

References

These documents provide supplemental material useful with this guide:

1. [Vivado Design Suite Documentation](#)
2. [Vivado Design Suite User Guide: Logic Simulation \(UG900\)](#)
3. [Vivado Design Suite User Guide: Synthesis \(UG901\)](#)
4. [Vivado Design Suite User Guide: Dynamic Function eXchange \(UG909\)](#)
5. [Vivado Design Suite Tutorial: Dynamic Function eXchange \(UG947\)](#)
6. [UltraFast Design Methodology Guide for FPGAs and SoCs \(UG949\)](#)
7. [Vivado Design Suite User Guide: Implementation \(UG904\)](#)
8. [Vivado Design Suite User Guide: Release Notes, Installation, and Licensing \(UG973\)](#)
9. [Vivado Design Suite User Guide: Design Flows Overview \(UG892\)](#)
10. [Vivado Design Suite User Guide: I/O and Clock Planning \(UG899\)](#)
11. [Vivado Design Suite Tutorial: Programming and Debugging \(UG936\)](#)
12. [Vivado Design Suite Tcl Command Reference Guide \(UG835\)](#)
13. [SmartLynq Data Cable User Guide \(UG1258\)](#)
14. [7 Series FPGAs Configuration User Guide \(UG470\)](#)
15. [7 Series FPGAs and Zynq 7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide \(UG480\)](#)
16. [UltraScale Architecture Configuration User Guide \(UG570\)](#)
17. [UltraScale Architecture System Monitor User Guide \(UG580\)](#)
18. [Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator \(UG994\)](#)
19. [Virtex UltraScale+ FPGAs GTM Transceivers User Guide \(UG581\)](#)
20. [Debug Bridge LogiCORE IP Product Guide \(PG245\)](#)
21. [Xilinx Virtual Cable Running on Zynq 7000 Using the PetaLinux Tools \(XAPP1251\)](#)
22. [Vivado Design Suite Tutorial: Embedded Processor Hardware Design \(UG940\)](#)
23. [Xilinx In-System Programming Using an Embedded Microcontroller \(ISE Tools\) \(XAPP058\)](#)
24. [Using Encryption to Secure a 7 Series FPGA Bitstream \(XAPP1239\)](#)
25. [Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream \(XAPP1267\)](#)
26. [Virtual Input/Output LogiCORE IP Product Guide \(PG159\)](#)
27. [Integrated Bit Error Ratio Tester 7 Series GTX Transceivers LogiCORE IP Product Guide \(PG132\)](#)
28. [Integrated Bit Error Ratio Tester 7 Series GTP Transceivers LogiCORE IP Product Guide \(PG133\)](#)
29. [Integrated Bit Error Ratio Tester 7 Series GTH Transceivers LogiCORE IP Product Guide \(PG152\)](#)
30. [Integrated Logic Analyzer LogiCORE IP Product Guide \(PG172\)](#)
31. [JTAG to AXI Master LogiCORE IP Product Guide \(PG174\)](#)
32. [System Integrated Logic Analyzer LogiCORE IP Product Guide \(PG261\)](#)
33. [UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP Product Guide \(PG213\)](#)
34. [Debug Bridge LogiCORE IP Product Guide \(PG245\)](#)
35. [In-System IBERT LogiCORE IP Product Guide \(PG246\)](#)
36. [UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide \(PG150\)](#)

37. AXI High Bandwidth Controller LogiCORE IP Product Guide ([PG276](#))
38. IBERT for UltraScale GTM Transceivers LogiCORE IP Product Guide ([PG342](#))
39. Control, Interface and Processing System LogiCORE IP Product Guide ([PG352](#))
40. Bootgen User Guide ([UG1283](#))

Training Resources

AMD provides a variety of training courses and QuickTake videos to help you learn more about the concepts presented in this document. Use these links to explore related training:

1. [Designing FPGAs Using the Vivado Design Suite 1](#)
2. [Designing FPGAs Using the Vivado Design Suite 2](#)
3. [Designing FPGAs Using the Vivado Design Suite 3](#)
4. [Designing FPGAs Using the Vivado Design Suite 4](#)
5. [Vivado Design Suite QuickTake Video: Targeting Zynq Devices Using Vivado IP Integrator](#)
6. [Vivado Design Suite QuickTake Video: Partial Reconfiguration in Vivado Design Suite](#)
7. [Vivado Design Suite QuickTake Video: Using Vivado Design Suite with Revision Control](#)
8. [Vivado Design Suite QuickTake Video Tutorials](#)
9. [Embedded Design Tutorial: System Design Example for High-Speed Debug Port with SmartLynq+ Module](#)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
10/19/2023 Version 2023.2	
Standard Test and Programming Language (STAPL) Programming	Added new chapter for STAPL programming.
05/16/2023 Version 2023.1	
Debugging Logic Designs in Hardware	Updated Debugging AXI Interfaces in the Hardware Manager
Serial I/O Hardware Debugging Flows	Updated Serial I/O Hardware Debugging Flows
Versal Serial I/O Hardware Debugging Flows	Updated the note.
Configuration Memory Support	Updated the tables as Flash devices manufactured by Spansion are now known as Infineon.
Vendor Validated Flash Table	Added new table.

Please Read: Important Legal Notices

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes. THIS INFORMATION IS PROVIDED "AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright

© Copyright 2012-2023 Advanced Micro Devices, Inc. AMD, the AMD Arrow logo, Artix, Kintex, Versal, Virtex, Vivado, Zynq, and combinations thereof are trademarks of Advanced Micro Devices, Inc. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the US and/or elsewhere. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.