



Hardware Debugging

Objectives

► After completing this module, you will be able to:

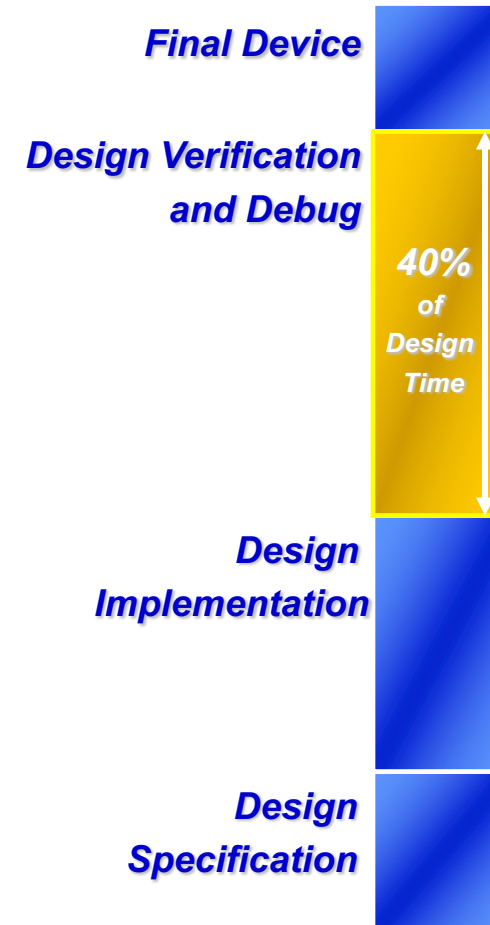
- Understand the key benefits provided by the Vivado logic debug feature
- List various debugging cores and their functionality
- Describe the process of including debug tool sampling cores in the Vivado Design Suite

Outline

- ▶ *Introduction*
- ▶ Logic Debug Cores
- ▶ Logic Debug Probing Flows
- ▶ Summary

Debug and Verification is Critical

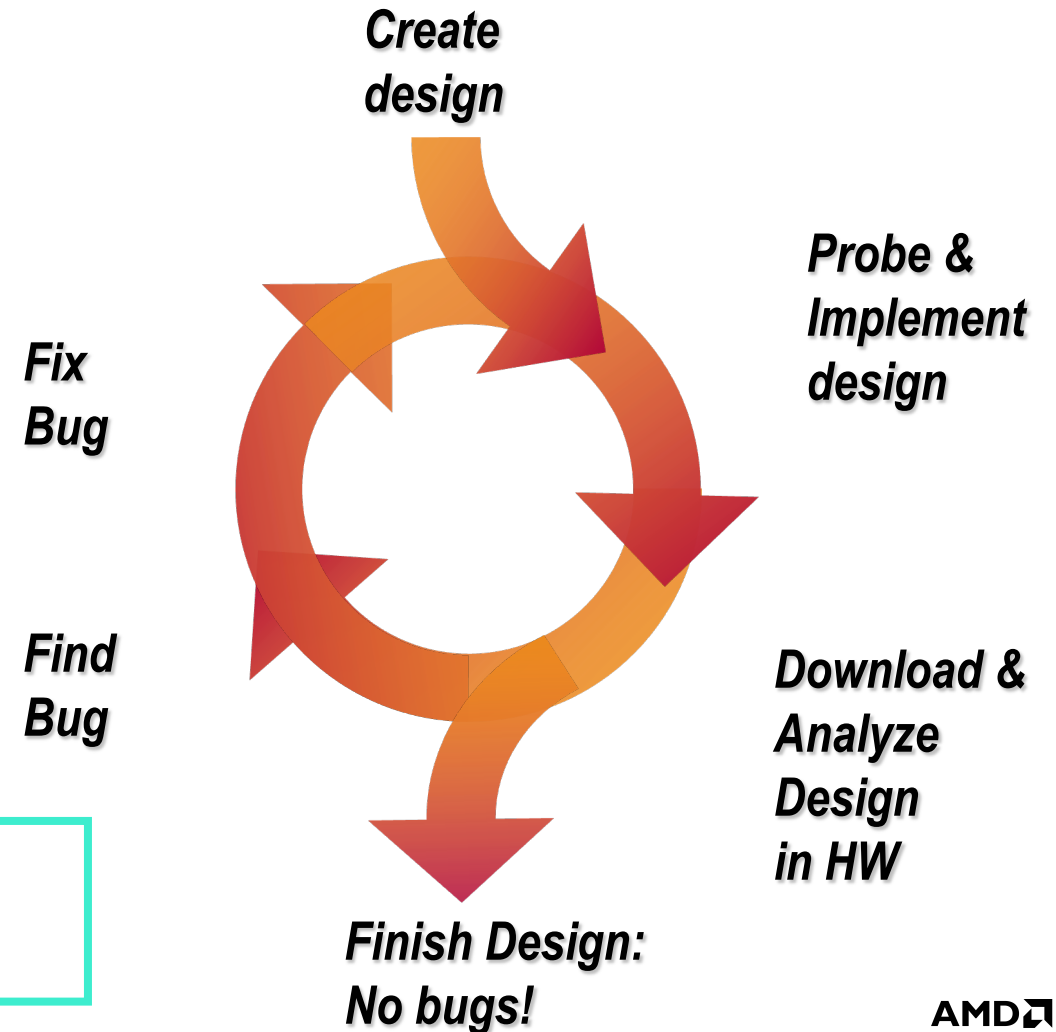
- ▶ Debug and verification can account for over 40% of creating a FPGA design
- ▶ Serial nature of debug and verification can make it difficult to optimize
- ▶ Inefficient strategy may result in product launch delay



Recommended Debug Methodology

- ▶ Engineers are trained to solve problems logically
 - *Break a problem into smaller parts*
 - *Simplify by reducing variables & variation*
 - *Make a prediction, verify the results*
 - *Plan how and where to debug early in the design cycles*
- ▶ FPGA design is an iterative process
- ▶ Debugging a FPGA design is an iterative process
 - **1) Probe**: Adding/modify debug probes
 - **2) Implement**: Compile design w/probes
 - **3) Analyze**: Look for bugs using probes
 - **4) Fix**: Fix any bugs, repeat as necessary

The reconfigurable nature of FPGAs facilitates the iterative debug process

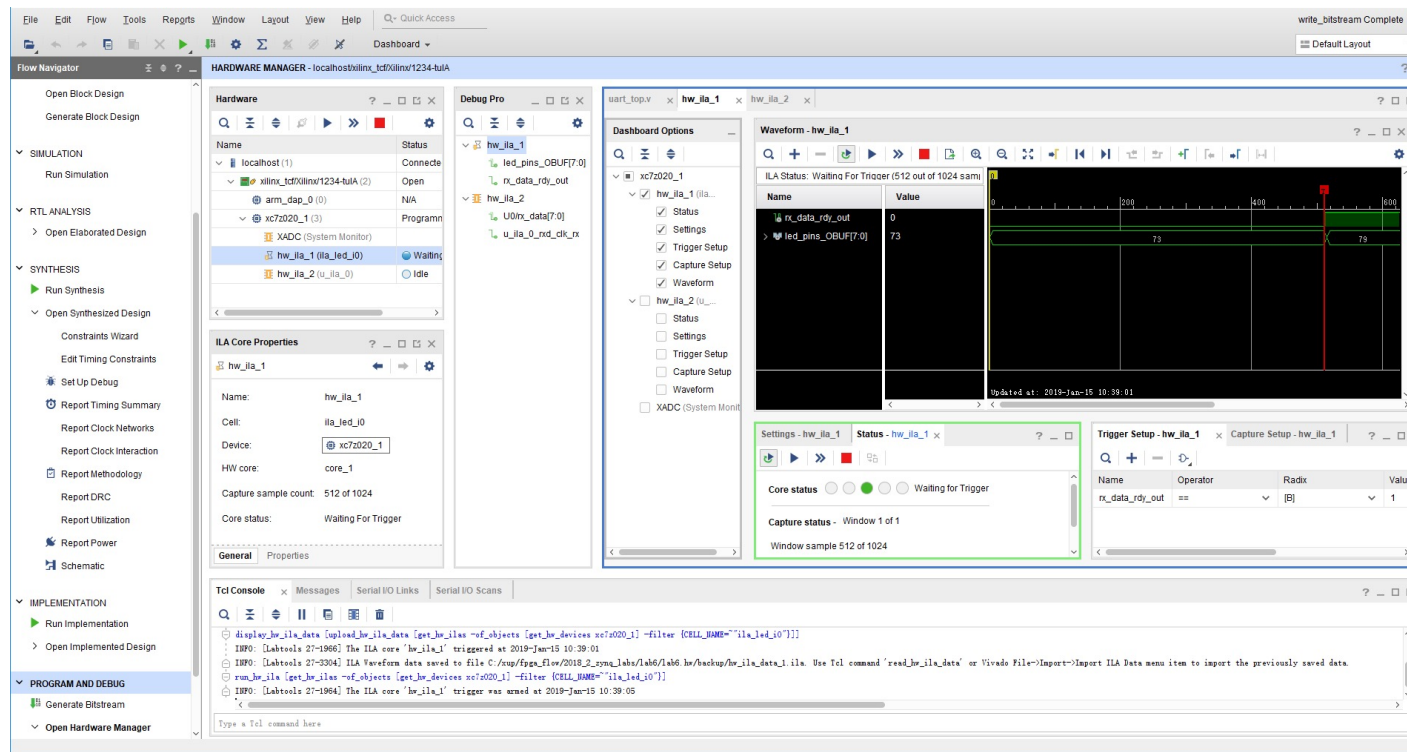


Xilinx Hardware Solutions for Debugging Designs

- ▶ Hardware debugging tool
 - Vivado logic analyzer for hardware
 - Functionally replaces need for external logic analyzer
- ▶ Now included in the Xilinx tool suite
- ▶ A single JTAG connection to the PC can be used for
 - Programming the programmable logic
 - Hardware debugging

Vivado Logic Analyzer Feature Overview

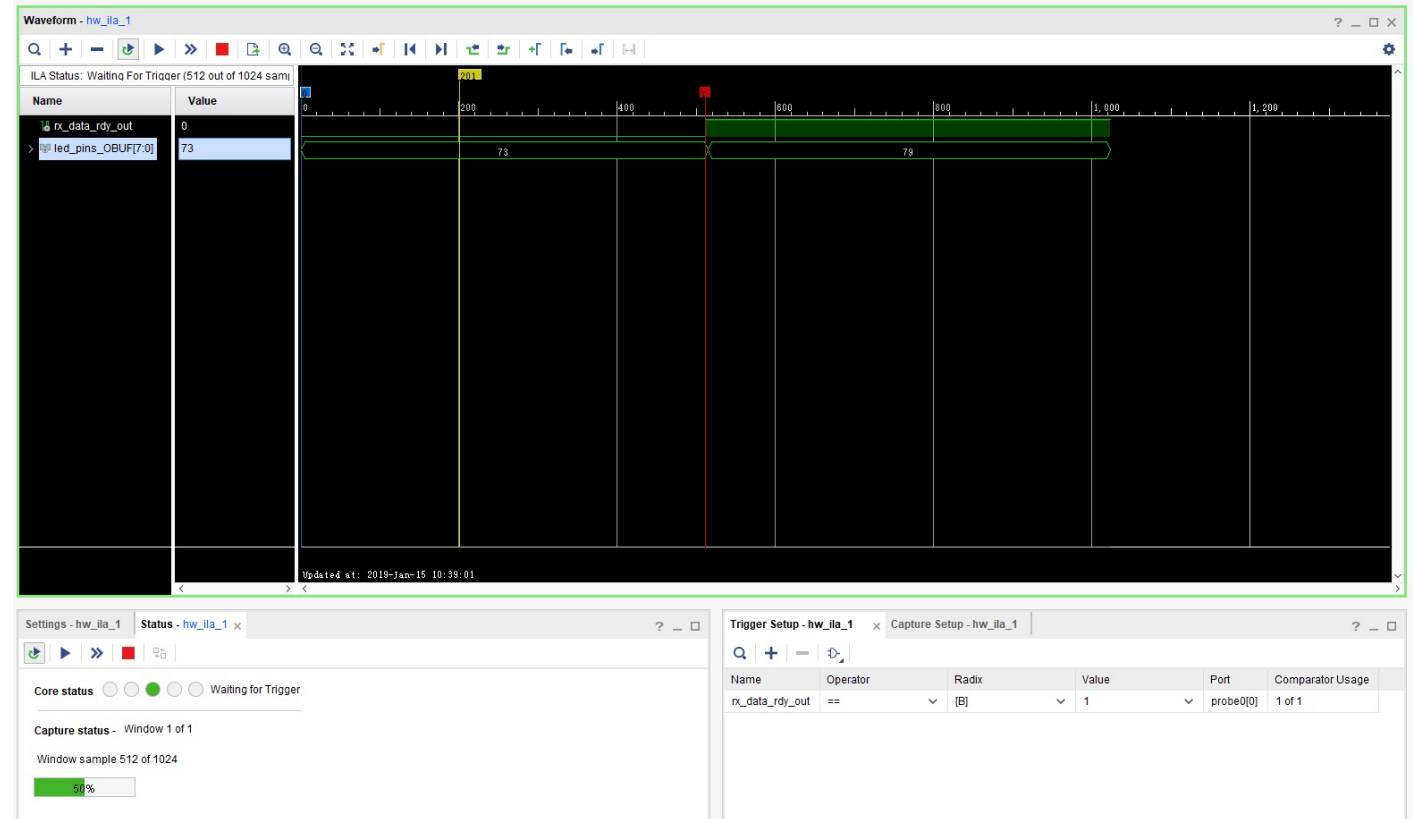
- ▶ Run-time software interface for interacting with ILA and VIO cores
- ▶ Supports simultaneous ILA waveform viewers
- ▶ Waveform formats can be saved and applied later for quick configurations



Vivado Logic Analyzer Feature

Common Waveform Viewer

- ▶ Shared across Vivado features
 - Simulator, Logic Analyzer, etc.
 - Reduces learning curve
 - Eases transition between features
- ▶ New functionality
 - Cursors and markers w/measure
 - Zoom mouse gestures
 - Custom colors
 - Find next/previous transition
 - Find signal name
 - Multiple radix selections
 - Integrated analog plot w/row resize



Vivado Logic Analyzer Feature

Tcl Scripting

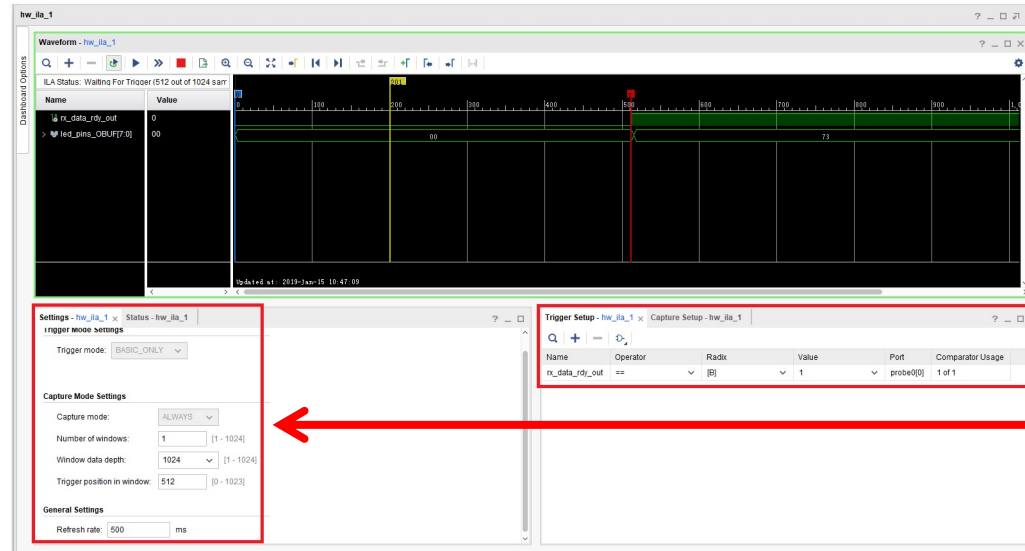
- ▶ Enables automation of logic debug
 - Uses common Vivado Tcl engine and concepts
 - Run tests in interactive or batch mode
 - Save results for future viewing
- ▶ Allows you to create custom functions and tests
 - Create repeatable tests
 - Link custom Tcl to toolbar buttons in Vivado IDE
 - More easily integrate into custom test environments

```
32 create_debug_core u_ila_0 ila
33 set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
34 set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
35 set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
36 set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
37 set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
38 set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
39 set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
40 set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
41 set_property port_width 1 [get_debug_ports u_ila_0/clk]
42 connect_debug_port u_ila_0/clk [get_nets [list clk_pin_IBUF_BUFG]]
43 set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe0]
44 set_property port_width 8 [get_debug_ports u_ila_0/probe0]
45 connect_debug_port u_ila_0/probe0 [get_nets [list {U0/rx_data[0]} {U0/rx_data[1]}]]
46 create_debug_port u_ila_0 probe
47 set_property PROBE_TYPE DATA_AND_TRIGGER [get_debug_ports u_ila_0/probe1]
48 set_property port_width 1 [get_debug_ports u_ila_0/probe1]
49 connect_debug_port u_ila_0/probe1 [get_nets [list U0/uart_rx_i0/rxd_clk_rx]]
50 set_property C_CLK_INPUT_FREQ_HZ 300000000 [get_debug_cores dbg_hub]
51 set_property C_ENABLE_CLK_DIVIDER false [get_debug_cores dbg_hub]
52 set_property C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub]
53 connect_debug_port dbg_hub/clk [get_nets clk_pin_IBUF_BUFG]
```

Benefits of Vivado Logic Debug

Simplified Debugging

- ▶ Single trigger comparator type (a.k.a. “match unit”) per PROBE
 - All comparison types, bit values
- ▶ No ICON core instantiation required
 - Handled by auto detection and connections during implementation
- ▶ Most debug parameters are set during runtime
 - Minimize unnecessary re-implementation

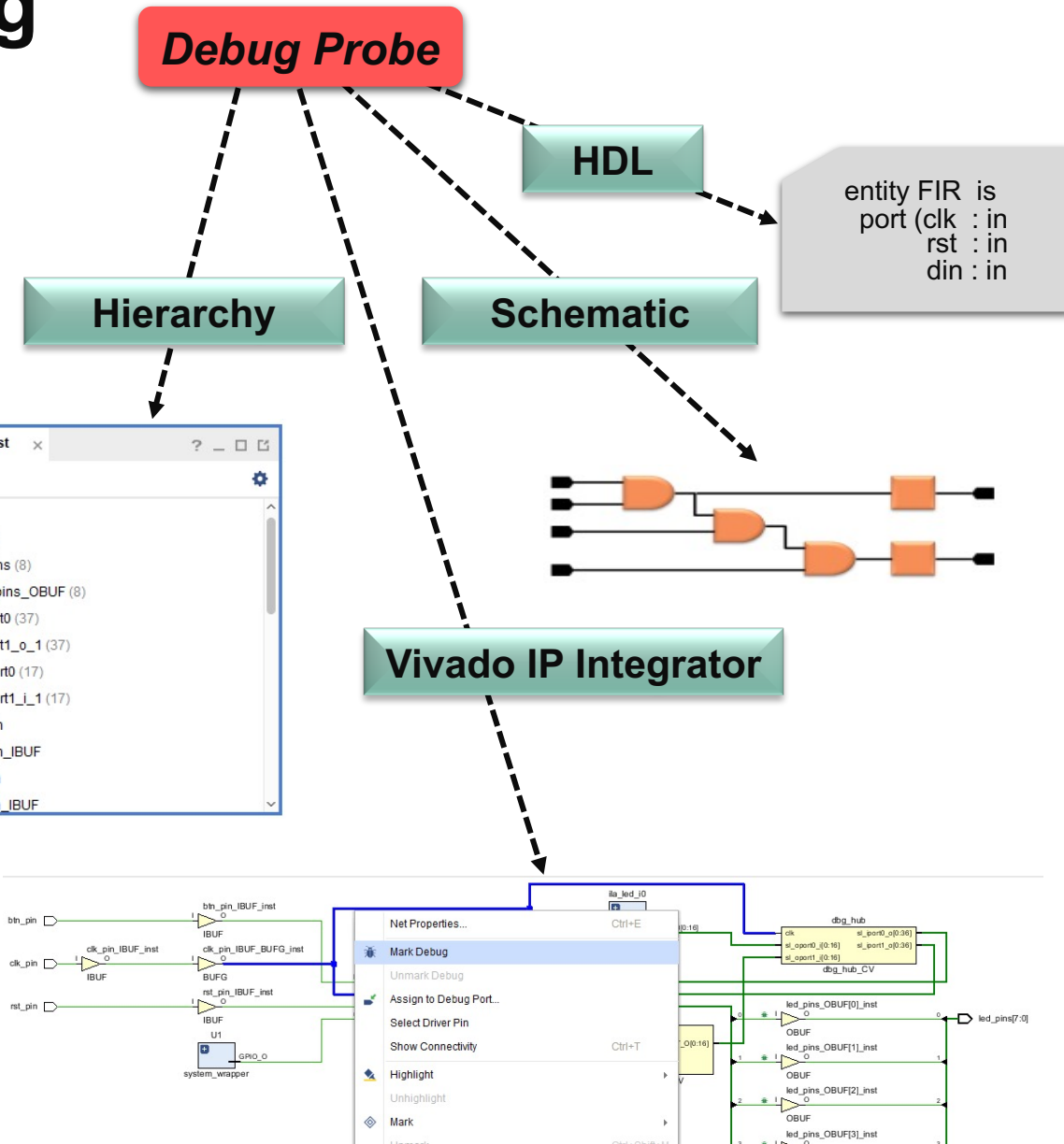


**Focus on
debugging the
design,
not on the debug
core!**

Benefits of Vivado Logic Debug

High-level Debugging

- ▶ Flexible, targeted probing of HDL design using MARK_DEBUG property
- ▶ Synthesized design probing in multiple views
- ▶ System-level probing inside of IP integrator view

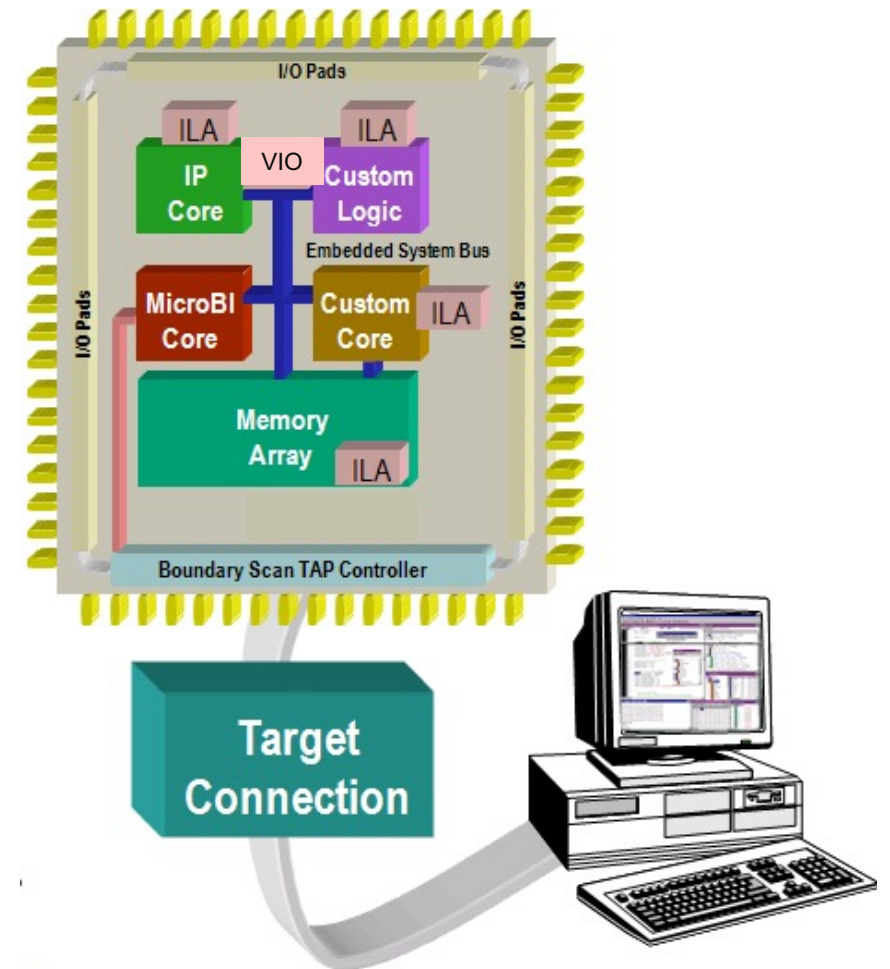


Debug the design at the appropriate level

Logic Debug Cores

Vivado Integrated Logic Analyzer System

- ▶ Vivado Design Suite debug cores provide internal visibility to all soft IP
 - Access to hard IP ports
 - Accesses all the internal signals, interfaces ports and nodes within the programmable logic (ILA v5.x)
 - Stimulus can be applied using the Virtual I/O core (VIO v3.x)
- ▶ Debugging occurs at, or near, system speeds
 - Debug on-chip using the system clock
- ▶ Minimize pins needed for debugging
 - Access via the JTAG interface (debug_core_hub)



Vivado Logic Debug IP

▶ ILA 6.2

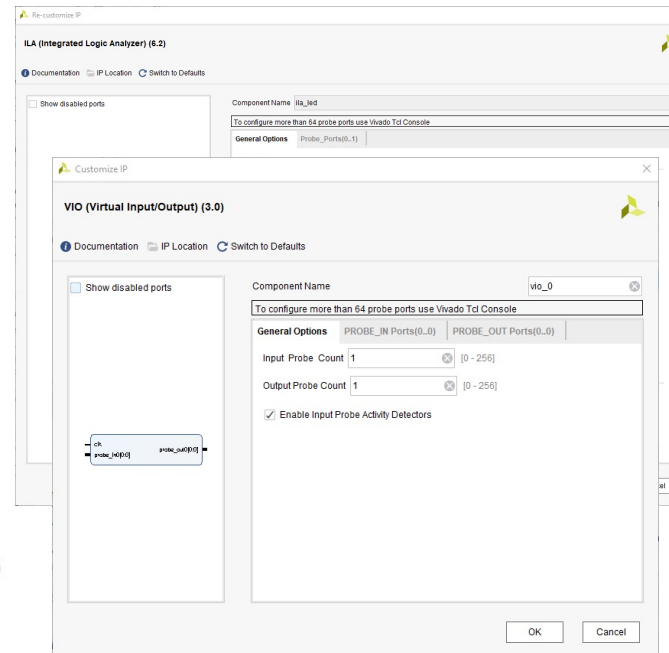
- Vivado native Integrated Logic Analyzer debug IP core
- Netlist insertion support
- HDL instantiation support

▶ VIO 3.0

- Vivado native Virtual Input / Output debug core
- HDL instantiation

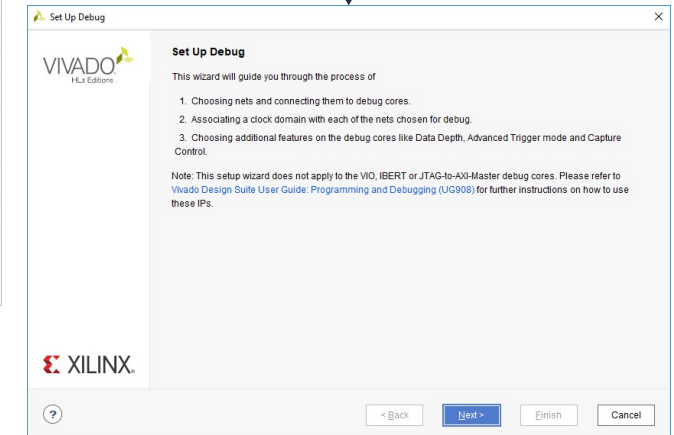
```
ila_led ila_led_i0 (  
    .clk(clk_pin), // input wire clk  
    .probe0(rx_data_rdy_out), // input wire [0:0] probe0  
    .probe1(led_pins) // input wire [7:0] probe1  
);
```

ILA 6.2 and VIO 3.0
HDL Instantiation



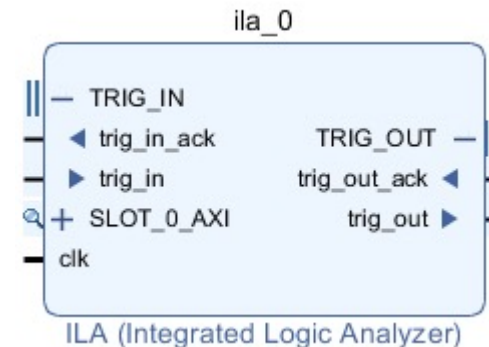
ILA 6.2
Netlist Insertion

Mark Debug Nets



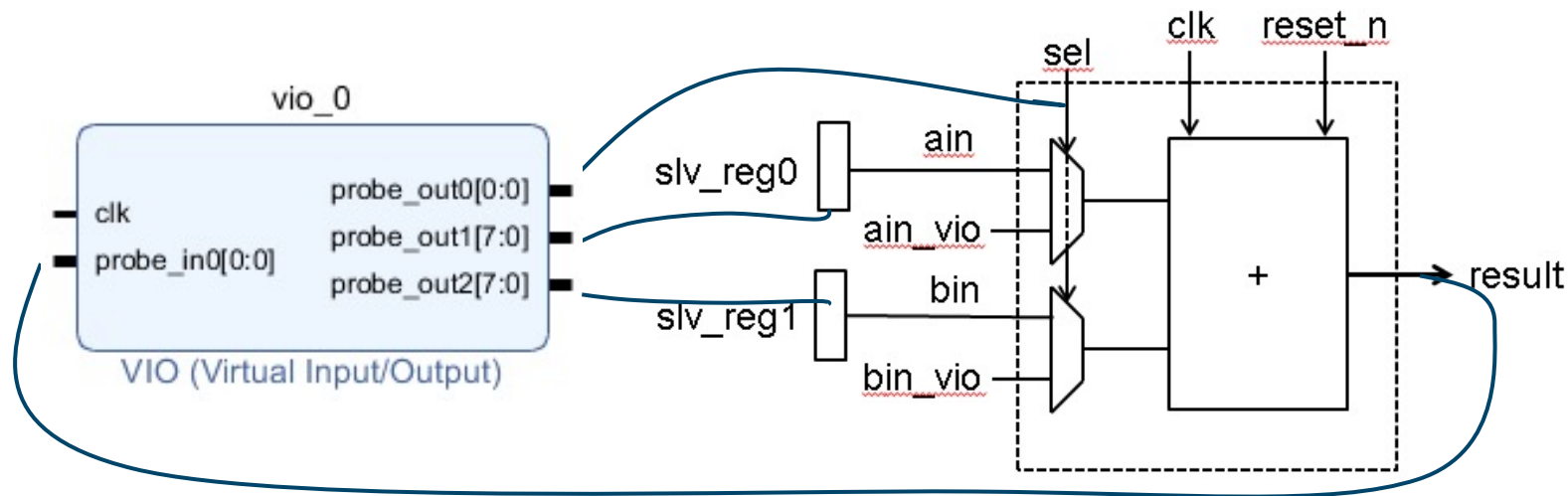
ILA Core

- ▶ Used for monitoring internal programmable logic signals and ports for post-analysis
- ▶ Multiple configurable ILA trigger units
 - Configurable trigger input widths and match types for use with different input signals types
- ▶ Separate data and trigger inputs
- ▶ Sequential triggering
- ▶ Storage qualification
- ▶ Trigger out signal for cross-probing
- ▶ Pre- and post-trigger buffering (capture data before, during, and after trigger condition is met)



VIO Core

- ▶ Support for monitoring and driving internal programmable logic signals in "real time"
- ▶ Probe input unit
- ▶ Probe output unit



Mark Debug

- ▶ Besides the IP cores insertion, Vivado logic debugging can be done by making nets in the HDL code using Mark Debug property
- ▶ 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;
```

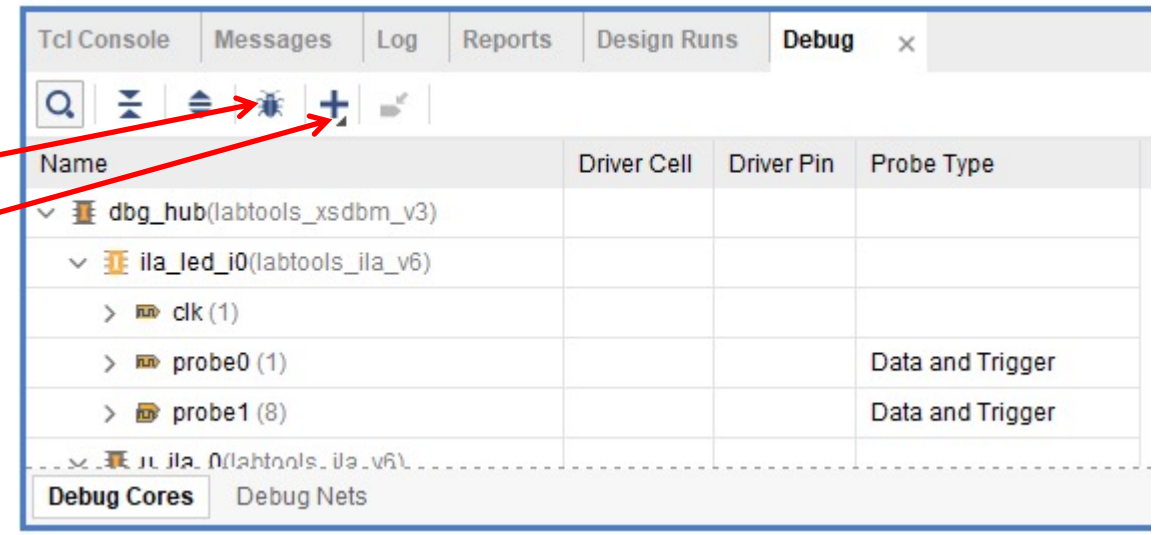
Logic Debug Probing Flows

Vivado Debug Probing Flows

- ▶ Netlist insertion flow (Highly recommended)
 - Most flexible with high predictability
 - Probing at different design levels (HDL, synthesized design, system design)
 - Compatible with various tool modes (Project, Non-project)
- ▶ HDL instantiation flow
 - Traditional flow for highest predictability, moderate flexibility
 - No longer requires ICON core instance
 - Probing at HDL design level only
 - Compatible with various tool modes (Project, Non-project)
- ▶ Netlist insertion and HDL instantiation flows can be mixed

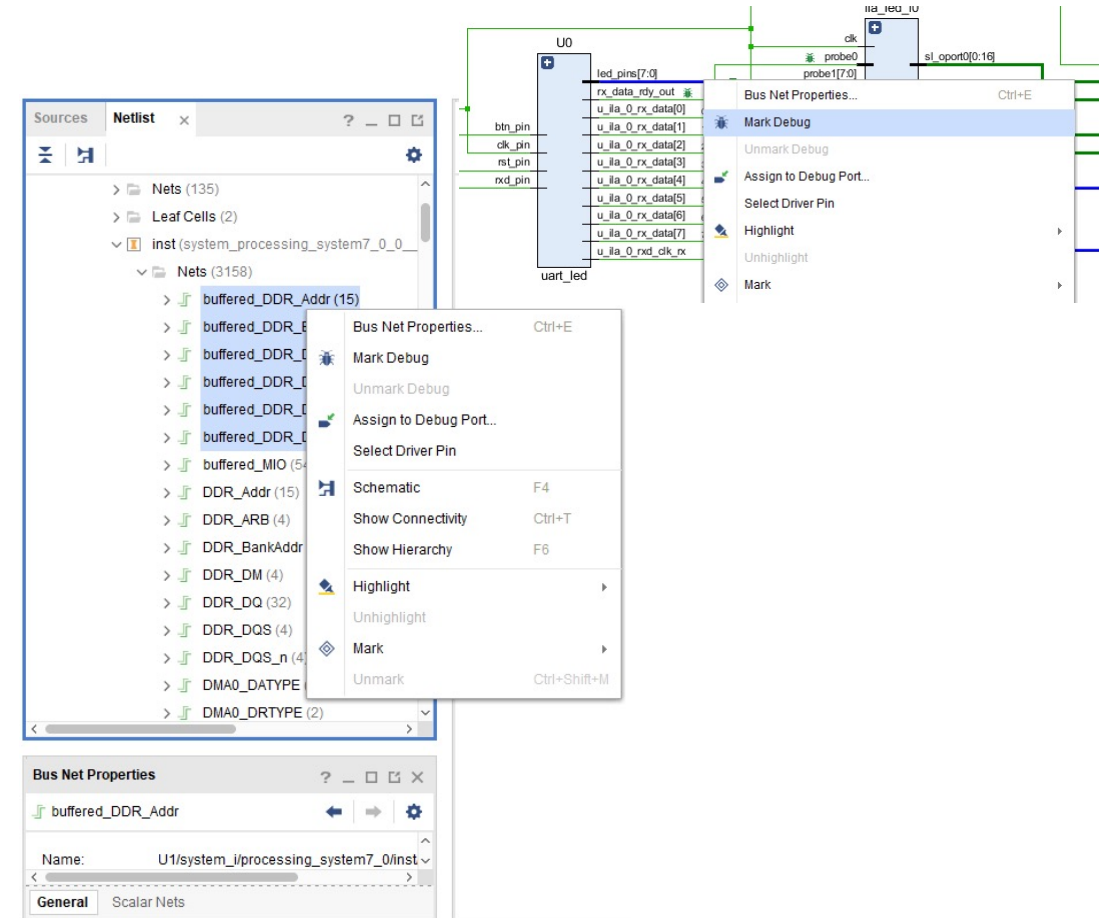
Vivado Debug Tool Access Points

- ▶ Select Tools > Set up Debug to launch the Vivado Debug Wizard
- ▶ Debug tab appears in the Synthesized Design view
 - Click on the Set Up Debug icon to re-launch the wizard
 - Set up Debug
 - Create debug core or port



Selecting Signals to Debug

- ▶ Multiple ways to select nets in Vivado
 - Netlist view (nets folders)
 - Each level of logic hierarchy
 - Schematic
 - Find results
- ▶ Right-click the net and select Mark Debug
- ▶ Nets added to Unassigned Nets folder in the Debug tab view
 - Placeholder for probable nets prior to configuring cores
- ▶ Net name search also in the Set Up Debug Wizard



Debug Tool Configuration

- ▶ The Vivado tool view displays core content and configuration
 - CLK, PROBE
 - Signal count
- ▶ Set options for cores and signals in the Properties view

Debug

Tcl Console

Messages

Log

Design Runs

Reports

🔍

⏮

⏪

⏩

⏭

🌱

Name	Driver Cell	Driver Pin	Probe Type
<div> <div>🔍</div> <div>dbg_hub(labtools_xsdbm_v3)</div> </div>			
<div> <div>🔍</div> <div>ila_led_i0(labtools_ila_v6)</div> </div>			
<div> <div>➤</div> <div>🔍</div> <div>clk (1)</div> </div>			
<div> <div>➤</div> <div>🔍</div> <div>probe0 (1)</div> </div>			Data and Trigger
<div> <div>➤</div> <div>🔍</div> <div>probe1 (8)</div> </div>			Data and Trigger
<div> <div>🔍</div> <div>u_ila_0(labtools_ila_v6)</div> </div>			
<div> <div>➤</div> <div>🔍</div> <div>clk (1)</div> </div>			
<div> <div>➤</div> <div>🔍</div> <div>probe0 (8)</div> </div>			Data and Trigger

Debug Cores

Debug Nets

The screenshot shows the 'Cell Properties' dialog box for a component named 'dbg_hub'. The dialog has a title bar with standard window controls and a help icon. Below the title bar, the component name 'dbg_hub' is displayed with a small icon to its left and navigation buttons (back, forward, and settings) to its right. The main area of the dialog lists several properties in a table-like format:

Name:	dbg_hub
Reference name:	dbg_hub_CV
Type:	Black Box
Debug core type:	labtools_xsdbm_v3
Number of cell pins:	109
Number of nets:	109

At the bottom of the dialog, there is a tabbed interface with three tabs: 'General' (which is currently selected), 'Properties', and 'Debug Core Options'. To the right of the tabs, there are additional navigation buttons including 'De <' and '> ■'.

The screenshot shows the 'Cell Properties' dialog box. The 'NAME' field is set to 'dbg_hub'. The 'CLASS' field is set to 'cell'. The 'CORE_GENERATION_IN' field is set to 'dbg_hub\labtools_xsdbsm_V'. The 'DONT_TOUCH' checkbox is checked. The 'IS_BLACKBOX', 'IS_DEBUGGABLE', 'IS_DEBUG_CORE', and 'IS_MATCHED' checkboxes are unchecked. The 'IS_ORIG_CELL', 'IS_PRIMITIVE', 'IS_REUSED', and 'IS_SEQUENTIAL' checkboxes are unchecked. The 'NAME' field is set to 'dbg_hub'.

Cell Properties

?

_

□

⌵

×

dbg_hub

◀

▶

⚙

C_USER_SCAN_CHAIN	1	▼
C_CLK_INPUT_FREQ_HZ	300000000	
C_ENABLE_CLK_DIVID...	<input type="checkbox"/>	

Select an option above to see a description of it

General

Properties

Debug Core Options

De ◀ ▶ ≡

Summary

Summary

- ▶ Vivado Logic debug features integration, common wave viewer, and Tcl command
- ▶ Debugging is an integral and important component of embedded system development
- ▶ The Vivado logic analyzer provides various cores to view the inside of a design—from individual signals to bus-level activities
 - ILA core
 - VIO core
- ▶ The Debug Configuration Wizard simplifies hardware connections and logic analyzer peripheral inclusion
- ▶ Three Vivado logic debug flows
 - HDL Instantiation
 - Netlist insertion
 - Mark Debug in netlist, schematic, and HDL



Thank You

Disclaimer and Attribution

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

© Copyright 2022 Advanced Micro Devices, Inc. All rights reserved. Xilinx, the Xilinx logo, AMD, the AMD Arrow logo, Alveo, Artix, Kintex, Kria, Spartan, Versal, Vitis, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

