AMD Vivado ChipScope Debug Tools (ILA and VIO)

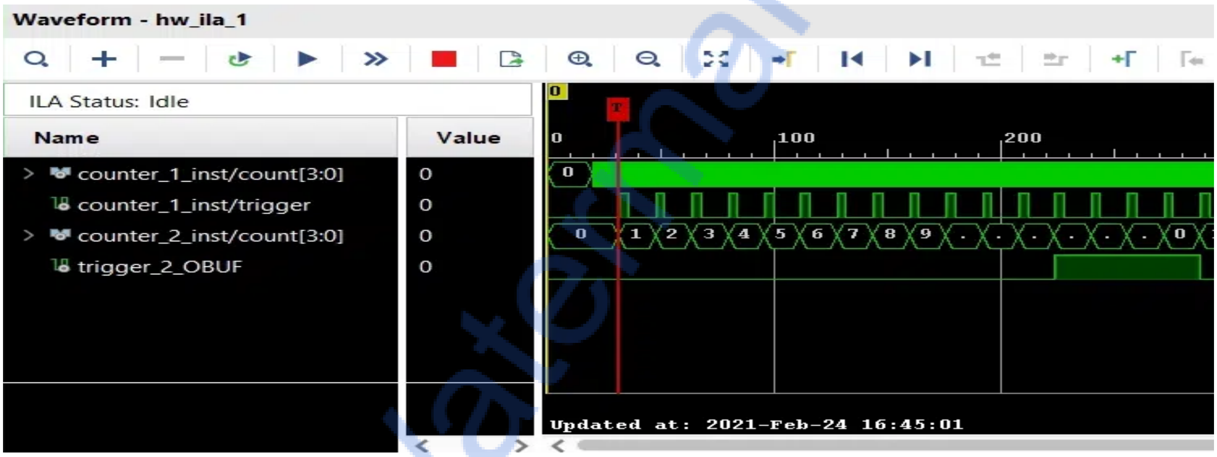
Overview

Vivado’s ChipScope Debug Suite provides advanced hardware debugging tools for AMD FPGAs. These tools allow real-time visibility and control of internal signals on the programmed FPGA device. ChipScope is AMD's equivalent to Intel's SignalTap (logic analysis) and ISSP (in-system signal probing and stimulation).

The two primary components of the ChipScope suite are:

1. Integrated Logic Analyzer (ILA)

The ILA core functions as a built-in logic analyzer inside the FPGA. It captures and displays internal signals during real operation. This is essential for debugging complex timing issues, sequencing errors, or interface problems that are not detectable in simulation.



Waveform - hw_ila_1

ILA Status: Idle

Name	Value
counter_1_inst/count[3:0]	0
counter_1_inst/trigger	0
counter_2_inst/count[3:0]	0
trigger_2_OBUF	0

Updated at: 2021-Feb-24 16:45:01

Key Features:

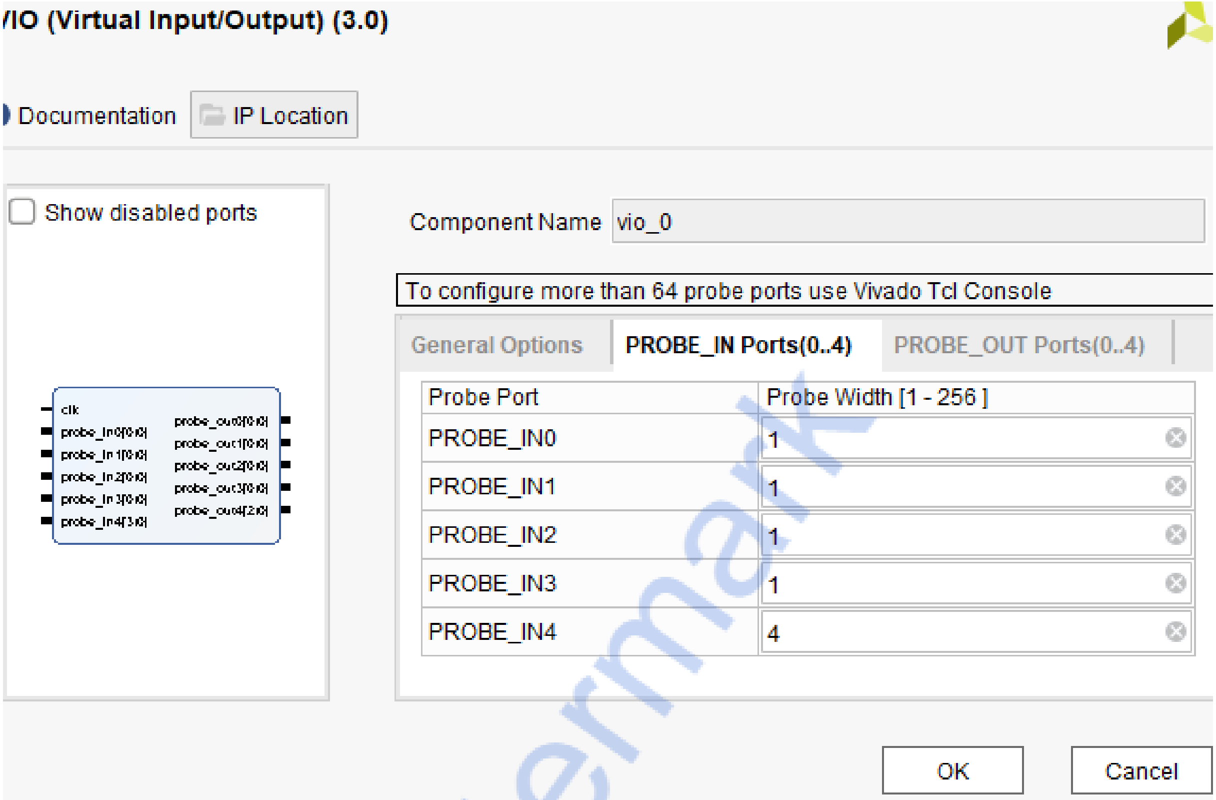
- Supports trigger conditions (such as edge detection, level detection, or combinations).
- Allows multiple simultaneous probes and configurable capture depth.
- Displays real-time waveforms through Vivado Hardware Manager.
- Allows customization of data widths and buffer depths per probe group.

Typical Use Cases:

- Verifying state machines, controlling logic, and internal registers.
- Debugging timing and synchronization issues.
- Monitoring and validating interfaces like UART, SPI, AXI, etc

2. Virtual Input/Output (VIO)

The VIO core enables interaction with the design's internal signals at runtime, without needing to modify the VHDL code or recompile the project. It is especially useful for live validation and testing scenarios.



□ Show disabled ports

Component Name vio_0

To configure more than 64 probe ports use Vivado Tcl Console

Probe Port	Probe Width [1 - 256]
PROBE_IN0	1
PROBE_IN1	1
PROBE_IN2	1
PROBE_IN3	1
PROBE_IN4	4

Key Features:

- Virtual inputs: allow forcing values on internal signals.
- Virtual outputs: monitor the status of internal logic and registers.
- Entirely configured through the Vivado GUI; no HDL modifications needed.
- Works seamlessly with ILA for simultaneous observation and control.

Typical Use Cases:

- Triggering control signals or changing design modes dynamically.
- Reading counters, flags, or internal registers in real-time.
- Injecting scenarios to test exception handling or corner cases.

Comparison with Intel Tools

Feature	Vivado (AMD)	Quartus (Intel)
Logic Analyzer	ILA	SignalTap
Signal Injection	VIO	ISSP
Trigger Configuration	GUI-based (flexible)	GUI-based (similar)
Probe Insertion	Netlist level (pre-bitstream)	Post-synthesis insertion

ChipScope Workflow in Vivado

1. **Add ILA or VIO IP Cores**:
   Add the ILA or VIO cores to your design via IP Integrator (Block Design) or instantiate them directly in VHDL.
2. **Connect Probes and Configure Triggers**:
   Select the signals of interest and configure the capture or control behavior using the Vivado GUI.
3. **Synthesize and Implement the Design**:
   With ILA or VIO included, the tools will automatically route and preserve the connected signals during synthesis.
4. **Program and Debug**:
   Use Vivado Hardware Manager to load the design onto the FPGA and interact with signals live via ILA and VIO.

Integration with IP Integrator (IPI/Block Design)

ILA and VIO can be easily integrated into Vivado's IP Integrator (IPI) Block Design environment. This provides the ability to connect to complex systems and buses graphically.

Benefits:

- ILA becomes a permanent part of the system design, making debug capability always available without modifying the project.
- Probes can be connected to any AXI interface, peripheral signals, or processor components (such as MicroBlaze or Zynq).
- Simplifies debugging in complex designs with multiple subsystems and data paths.

This setup is especially useful when working with embedded processors, where **co-debugging** is required, debugging both the hardware (HDL) and the software (e.g., C code in Vitis) simultaneously.

RTL Traceability in VHDL Using mark_debug

To ensure traceability between your VHDL source code and the signals visible in the ILA probe configuration, you can use the mark_debug attribute in your VHDL files.

Example:

```vhdl
signal my_internal_signal : std_logic;
attribute mark_debug : string;
attribute mark_debug of my_internal_signal : signal is "true";
```

Benefits of mark_debug:

- Ensures that the signal is preserved through synthesis and implementation.
- Allows the signal to appear in the ILA GUI even if Vivado renames or optimizes it away.
- Provides clear traceability between your RTL code and captured hardware signals.
- Speeds up debug workflows by avoiding confusion with signal naming and hierarchy changes.

Advantages of Vivado ChipScope Tools

- Integrated debugging directly inside the FPGA without external equipment.
- Full VHDL compatibility and no need for extra I/O pins or test logic.
- Permanent inclusion of ILA via Block Design ensures debug readiness without redesign.
- Native support for AXI interfaces and SoC-level systems.
- Enables hardware-software co-debugging when using processors like MicroBlaze.
- mark_debug ensures reliable and consistent access to key RTL signals.

Further learning more and details can be found at the link below.

https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado/chipscope-hardware-debug.html

This demonstration provides a step-by-step practical example of using ILA and VIO in Vivado. Click on the link below!

https://www.youtube.com/watch?v=DMX9zTngN5c