TU Graz

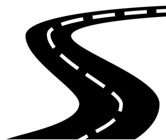
SCIENCE
PASSION
TECHNOLOGY

SoC Debugging Tutorial

Barbara Gigerl, Rishub Nagpal

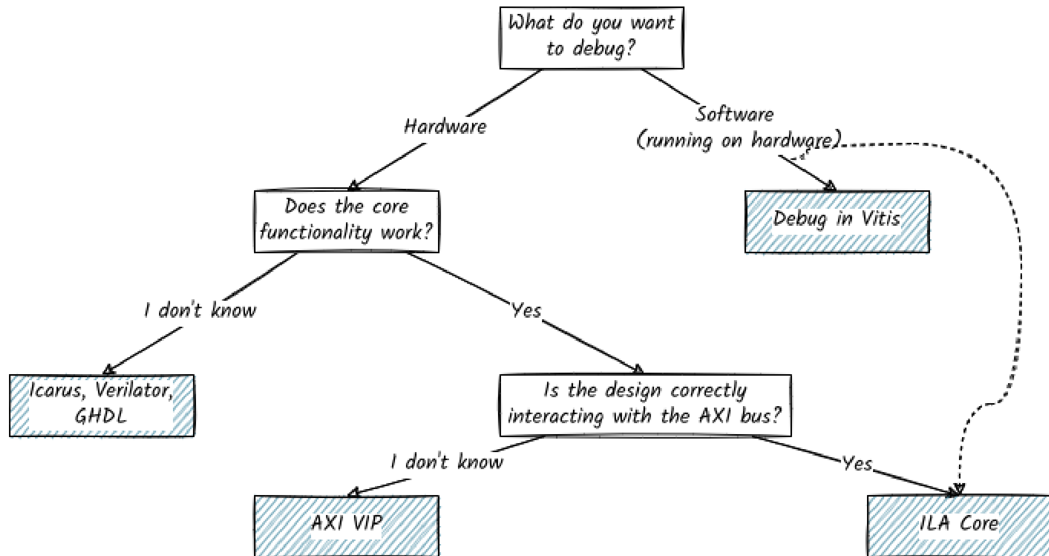
October 19th, 2022

Overview

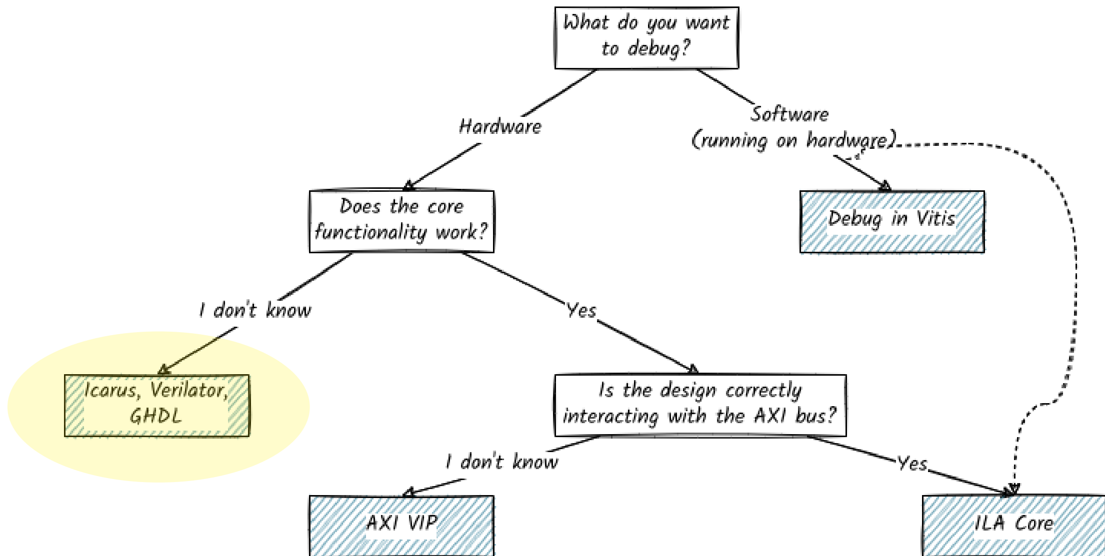


- Simulation of hardware designs (Icarus, Verilator, GHDL)
- Using AXI VIP
- Using ILA Cores
- Debugging SW in Vitis

Can’t decide?



Can’t decide?



Simulation of hardware designs

Icarus, Verilator, GHDL

Prerequisites

- When should I use this method?

Prerequisites

- When should I use this method?
  - I have a small hardware design and want to get a rough idea of the core functionality

Prerequisites

- When should I use this method?
  - I have a small hardware design and want to get a rough idea of the core functionality
  - No software involved

Prerequisites

- When should I use this method?
  - I have a small hardware design and want to get a rough idea of the core functionality
  - No software involved
  - I want to find functional bugs in my hardware design

Prerequisites

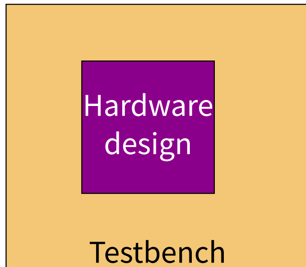
- When should I use this method?
  - I have a small hardware design and want to get a rough idea of the core functionality
  - No software involved
  - I want to find functional bugs in my hardware design

Prerequisites

- When should I use this method?
  - I have a small hardware design and want to get a rough idea of the core functionality
  - No software involved
  - I want to find functional bugs in my hardware design

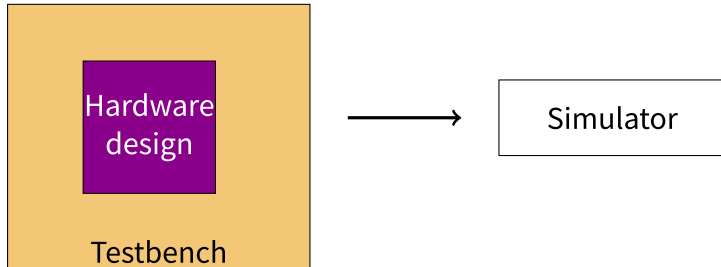
Prerequisites

- When should I use this method?
  - I have a small hardware design and want to get a rough idea of the core functionality
  - No software involved
  - I want to find functional bugs in my hardware design



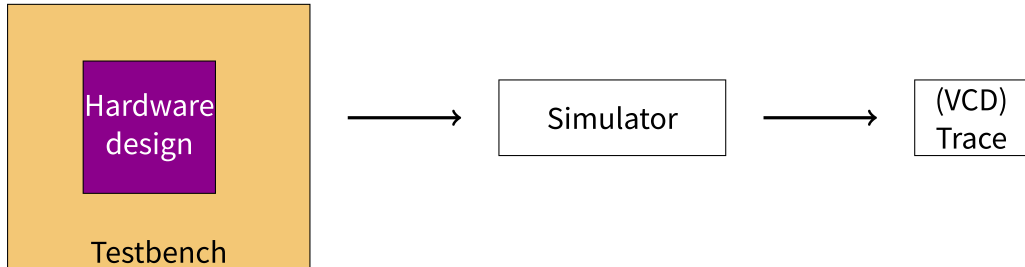
Prerequisites

- When should I use this method?
  - I have a small hardware design and want to get a rough idea of the core functionality
  - No software involved
  - I want to find functional bugs in my hardware design



Prerequisites

- When should I use this method?
  - I have a small hardware design and want to get a rough idea of the core functionality
  - No software involved
  - I want to find functional bugs in my hardware design



Hardware design and testbench

- Example: Fibonacci numbers
  https://extgit.iaik.tugraz.at/sip/tutorials/-/tree/master/fibonacci

Hardware design and testbench

- Example: Fibonacci numbers
  https://extgit.iaik.tugraz.at/sip/tutorials/-/tree/master/fibonacci
- Testbench in Verilog

Hardware design and testbench

- Example: Fibonacci numbers
  https://extgit.iaik.tugraz.at/sip/tutorials/-/tree/master/fibonacci
- Testbench in Verilog
  - Apply test data to input ports

Hardware design and testbench

- Example: Fibonacci numbers
  `https://extgit.iaik.tugraz.at/sip/tutorials/-/tree/master/fibonacci`
- Testbench in Verilog
  - Apply test data to input ports
  - Create clock/reset signals

Hardware design and testbench

- Example: Fibonacci numbers
  https://extgit.iaik.tugraz.at/sip/tutorials/-/tree/master/fibonacci
- Testbench in Verilog
  - Apply test data to input ports
  - Create clock/reset signals
  - Write to log file

Hardware design and testbench

- Example: Fibonacci numbers
  https://extgit.iaik.tugraz.at/sip/tutorials/-/tree/master/fibonacci
- Testbench in Verilog
  - Apply test data to input ports
  - Create clock/reset signals
  - Write to log file
  - ...

Icarus Verilog

- http://iverilog.icarus.com/

Icarus Verilog

- http://iverilog.icarus.com/
- Simple but slow

Icarus Verilog

- `http://iverilog.icarus.com/`
- Simple but slow
- Testbench in Verilog or SystemVerilog

Icarus Verilog

- `http://iverilog.icarus.com/`
- Simple but slow
- Testbench in Verilog or SystemVerilog
  - Instantiate test module

Icarus Verilog

- `http://iverilog.icarus.com/`
- Simple but slow
- Testbench in Verilog or SystemVerilog
  - Instantiate test module
  - Create clock and reset control signals

Icarus Verilog

- `http://iverilog.icarus.com/`
- Simple but slow
- Testbench in Verilog or SystemVerilog
  - Instantiate test module
  - Create clock and reset control signals
  - Optional: `$display`, `$dumpfile`, `$dumpvars`, `$monitor`

Icarus Verilog

- `http://iverilog.icarus.com/`
- Simple but slow
- Testbench in Verilog or SystemVerilog
  - Instantiate test module
  - Create clock and reset control signals
  - Optional: `$display`, `$dumpfile`, `$dumpvars`, `$monitor`

Icarus Verilog

- `http://iverilog.icarus.com/`
- Simple but slow
- Testbench in Verilog or SystemVerilog
  - Instantiate test module
  - Create clock and reset control signals
  - Optional: `$display`, `$dumpfile`, `$dumpvars`, `$monitor`

```
iverilog -o <bin_name> <dut>.v <tb>.v
./<bin_name>
```

Verilator

- https://www.veripool.org/verilator/

Verilator

- https://www.veripool.org/verilator/
- Fast but (slightly more) complex

Verilator

- https://www.veripool.org/verilator/
- Fast but (slightly more) complex
- Testbench in C++

Verilator

- https://www.veripool.org/verilator/
- Fast but (slightly more) complex
- Testbench in C++
  - Create clock and reset control signals

Verilator

- https://www.veripool.org/verilator/
- Fast but (slightly more) complex
- Testbench in C++
  - Create clock and reset control signals
  - Use all the C++ features you want

Verilator

- https://www.veripool.org/verilator/
- Fast but (slightly more) complex
- Testbench in C++
  - Create clock and reset control signals
  - Use all the C++ features you want
  - `verilated_vcd_c.h` for VCD dump support

Verilator

- https://www.veripool.org/verilator/
- Fast but (slightly more) complex
- Testbench in C++
  - Create clock and reset control signals
  - Use all the C++ features you want
  - `verilated_vcd_c.h` for VCD dump support

Verilator

- https://www.veripool.org/verilator/
- Fast but (slightly more) complex
- Testbench in C++
  - Create clock and reset control signals
  - Use all the C++ features you want
  - `verilated_vcd_c.h` for VCD dump support

```
verilator --trace --cc <dut>.v
cd obj_dir;make -f V<dut>.mk; cd ..
clang++ -Iobj_dir -I/usr/share/verilator/include verilator_tb.cpp
  obj_dir/V<dut>__ALL.a
  /usr/share/verilator/include/verilated.cpp
  /usr/share/verilator/include/verilated_vcd_c.cpp
  -o <bin_name>
./<bin_name>
```

GHDL

- Simulate VHDL designs
- Testbench in VHDL
- No support for VCD

GTKWave

- https://github.com/gtkwave/gtkwave

GTKWave

- https://github.com/gtkwave/gtkwave
- Viewer for VCD traces

GTKWave

- `https://github.com/gtkwave/gtkwave`
- Viewer for VCD traces
  - VCD = Value Change Dump

GTKWave

- https://github.com/gtkwave/gtkwave
- Viewer for VCD traces
  - VCD = Value Change Dump
  - 1 signal = 1 variable

GTKWave

- https://github.com/gtkwave/gtkwave
- Viewer for VCD traces
  - VCD = Value Change Dump
  - 1 signal = 1 variable
  - Whenever the variable changes, it is noted down in the file.

```
...
$timescale 1ps $end
...
$var wire 8 # data $end
$var wire 1 ' tx_en $end
...
$dumpvars
bxxxxxxxx #
x'
$end
#0
b10000001 #
1'
#2211
0'
#2296
b0 #
#2302
...
```

GTKWave

- https://github.com/gtkwave/gtkwave
- Viewer for VCD traces
  - VCD = Value Change Dump
  - 1 signal = 1 variable
  - Whenever the variable changes, it is noted down in the file.
- File - Open New Tab - Select vcd file

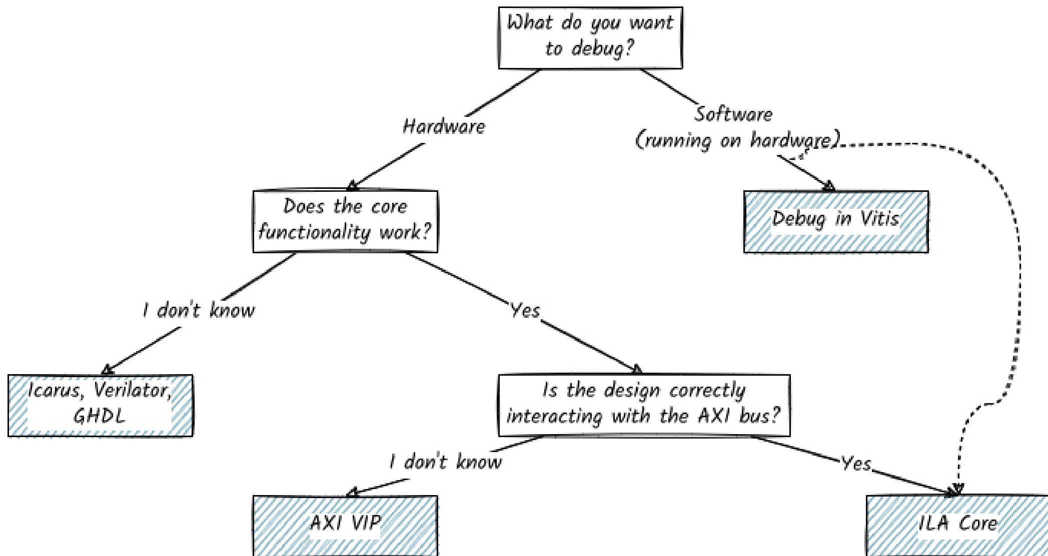
```
...
$timescale 1ps $end
...
$var wire 8 # data $end
$var wire 1 ' tx_en $end
...
$dumpvars
bxxxxxxxx #
x'
$end
#0
b10000001 #
1'
#2211
0'
#2296
b0 #
#2302
...
```

GTKWave

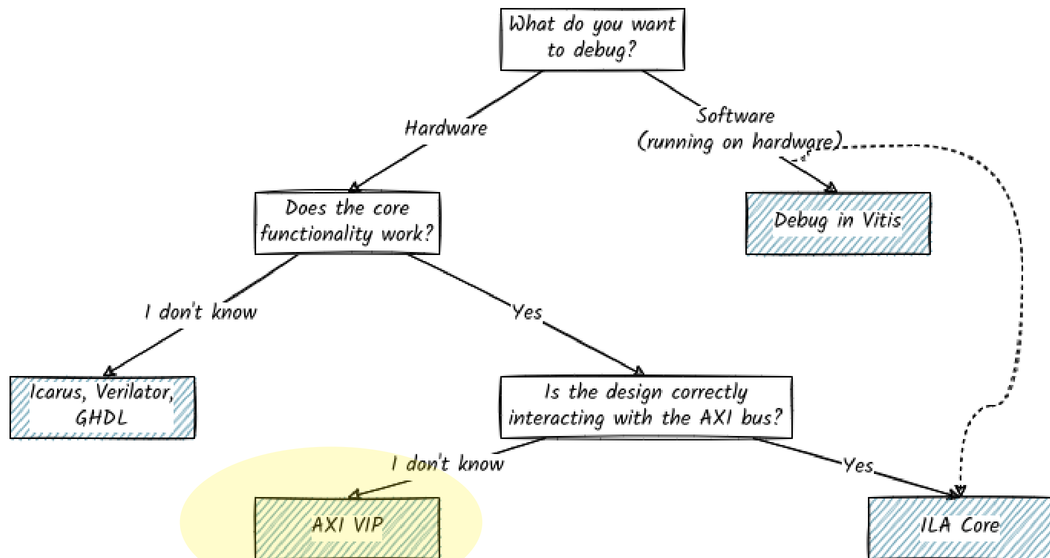
- https://github.com/gtkwave/gtkwave
- Viewer for VCD traces
  - VCD = Value Change Dump
  - 1 signal = 1 variable
  - Whenever the variable changes, it is noted down in the file.
- File - Open New Tab - Select vcd file
- Hint: Use Save Files to restore previous view configuration

```
...
$timescale 1ps $end
...
$var wire 8 # data $end
$var wire 1 ' tx_en $end
...
$dumpvars
bxxxxxxxx #
x'
$end
#0
b10000001 #
1'
#2211
0'
#2296
b0 #
#2302
...
```

Can’t decide?



Can’t decide?



Using the AXI VIP

Prerequisites

- When should I use this method?

Prerequisites

- When should I use this method?
  - I want to test my IP core

Prerequisites

- When should I use this method?
  - I want to test my IP core
    - Including AXI connectivity

Prerequisites

- When should I use this method?
  - I want to test my IP core
    - Including AXI connectivity
    - Interaction with other IP cores on the board

Prerequisites

- When should I use this method?
  - I want to test my IP core
    - Including AXI connectivity
    - Interaction with other IP cores on the board
    - In Vivado

Prerequisites

- When should I use this method?
  - I want to test my IP core
    - Including AXI connectivity
    - Interaction with other IP cores on the board
    - In Vivado
  - I want to test whether my IP core reacts correctly wrt the AXI protocol

AXI crashcourse

- AXI = Advanced eXtensible Interface
- Very popular bus protocol following a master/minion[1] structure
- Masters and minions want to communicate with each other via a shared channel.
  - Master reads data from and writes data to minion.
  - Minion does nothing without command from master.
- Based on bursts

[1]= slave

AXI crashcourse

- AXI channels:

AXI crashcourse

- AXI channels:
  - Address channels (AW, AR): address and control information

AXI crashcourse

- AXI channels:
  - Address channels (AW, AR): address and control information
  - Data channels (R, W): actual information

AXI crashcourse

- AXI channels:
  - Address channels (AW, AR): address and control information
  - Data channels (R, W): actual information
  - Write response channel: master can verify a write transaction has been completed

AXI crashcourse

- AXI channels:
  - Address channels (AW, AR): address and control information
  - Data channels (R, W): actual information
  - Write response channel: master can verify a write transaction has been completed
  - Each channel has specific signals associated with it.

AXI crashcourse

- AXI channels:
  - Address channels (AW, AR): address and control information
  - Data channels (R, W): actual information
  - Write response channel: master can verify a write transaction has been completed
  - Each channel has specific signals associated with it.
- AXI channel handshake:

AXI crashcourse

- AXI channels:
  - Address channels (AW, AR): address and control information
  - Data channels (R, W): actual information
  - Write response channel: master can verify a write transaction has been completed
  - Each channel has specific signals associated with it.
- AXI channel handshake:
  - Synchronize and control transfer

AXI crashcourse

- AXI channels:
  - Address channels (AW, AR): address and control information
  - Data channels (R, W): actual information
  - Write response channel: master can verify a write transaction has been completed
  - Each channel has specific signals associated with it.
- AXI channel handshake:
  - Synchronize and control transfer
  - VALID: used by sender to indicate that information is available

AXI crashcourse

- AXI channels:
  - Address channels (AW, AR): address and control information
  - Data channels (R, W): actual information
  - Write response channel: master can verify a write transaction has been completed
  - Each channel has specific signals associated with it.
- AXI channel handshake:
  - Synchronize and control transfer
  - VALID: used by sender to indicate that information is available
  - READY: used by the receiver to indicate that it is ready to accept information

AXI VIP

- AXI VIP = AXI Verification IP

AXI VIP

- AXI VIP = AXI Verification IP
- Simulate your IP core as an AXI master or minion

AXI VIP

- AXI VIP = AXI Verification IP
- Simulate your IP core as an AXI master or minion
- Simulation-only (cannot be synthesized)

AXI VIP

- AXI VIP = AXI Verification IP
- Simulate your IP core as an AXI master or minion
- Simulation-only (cannot be synthesized)
- Modes:

AXI VIP

- AXI VIP = AXI Verification IP
- Simulate your IP core as an AXI master or minion
- Simulation-only (cannot be synthesized)
- Modes:
  - AXI master VIP: creates read/write transactions for AXI minion DUT

AXI VIP

- AXI VIP = AXI Verification IP
- Simulate your IP core as an AXI master or minion
- Simulation-only (cannot be synthesized)
- Modes:
  - AXI master VIP: creates read/write transactions for AXI minion DUT
  - AXI minion VIP: reads payload, writes responses, ... for AXI master DUT

AXI VIP

- AXI VIP = AXI Verification IP
- Simulate your IP core as an AXI master or minion
- Simulation-only (cannot be synthesized)
- Modes:
  - AXI master VIP: creates read/write transactions for AXI minion DUT
  - AXI minion VIP: reads payload, writes responses, ... for AXI master DUT
  - AXI pass-through VIP: passive monitor

Preparing the test setup

1. Create a new block design and add:
   a. The IP core you want to test (AXI minion)
   b. `AXI verification IP`
   c. `Simulation clock generator`
2. Connect the simulation clock to the DUT-IP and VIP
3. Configure VIP: `Customize block...`
   - Interface mode: Master, minion, pass-through
4. `Run connection automation...`
5. `Validate design`
6. `Create HDL wrapper`

Writing the testbench

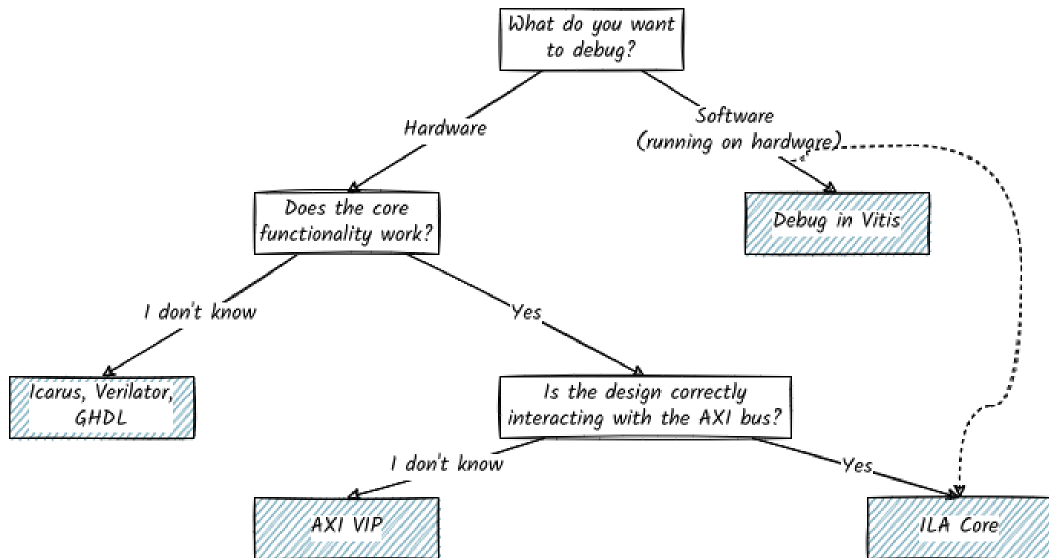
1. Add a new simulation source (`tb.sv`)
2. Import: `import axi_vip_pkg::*;` and `import <axi_vip_name>_pkg::*;` Hint: use `get_ips *vip*` to find out name
3. Instantiate the HDL wrapper
4. Add a new AXI master agent: `<axi_vip_name>_mst_t master_agent;`
   - Master agent can be used to generate AXI transactions
   - `master_agent.AXI4LITE_WRITE_BURST(...)`
   - `master_agent.AXI4LITE_READ_BURST(...)`

We provide a template testbench:
https://extgit.iaik.tugraz.at/sip/tutorials/-/tree/master/axi_tb

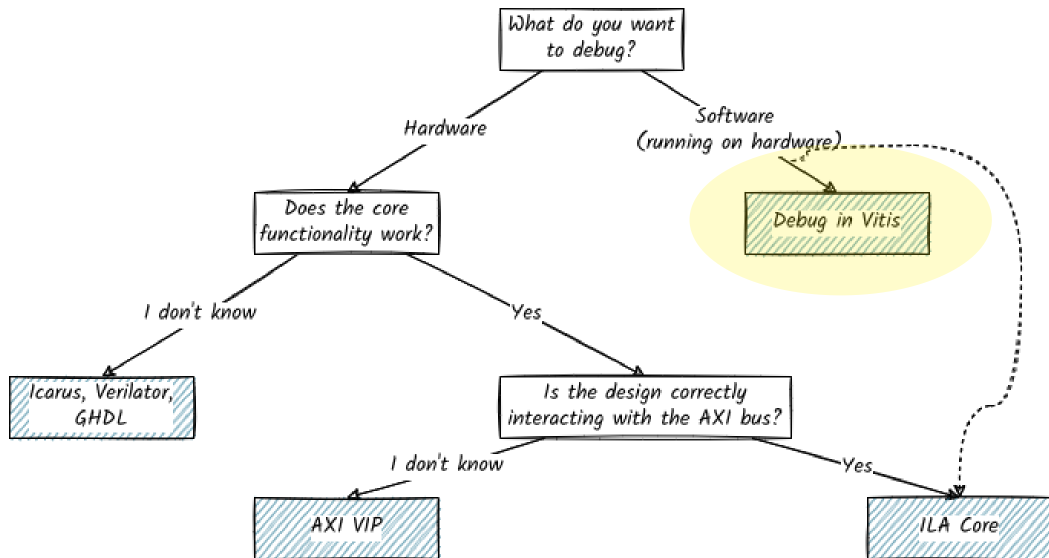
Starting the simulation

- `SIMULATION - Run Simulation - Run Behavorial Simulation`
- `Objects` : Instantiated modules
- `Protocol Instances` : can be used to view AXI protocol behavior
- Drag into simulation window

Can't decide?



Can't decide?



Debugging SW in Vitis

Prerequisites

- When should I use this method?

Prerequisites

- When should I use this method?
  - I want to find functional bugs in my software design

Prerequisites

- When should I use this method?
  - I want to find functional bugs in my software design
  - I want to debug the software I wrote by running it on real hardware

Prerequisites

- When should I use this method?
  - I want to find functional bugs in my software design
  - I want to debug the software I wrote by running it on real hardware
  - In Vitis

Prerequisites

- When should I use this method?
  - I want to find functional bugs in my software design
  - I want to debug the software I wrote by running it on real hardware
  - In Vitis
- Executable must be built in Debug mode
  (`Assistant - Select Build Configuration`)

XSDB

- XSDB = Xilinx System Debugger

XSDB

- XSDB = Xilinx System Debugger
- Uses `hw_server` as debug engine to communicate with CPU on Zybo Board

XSDB

- XSDB = Xilinx System Debugger
- Uses `hw_server` as debug engine to communicate with CPU on Zybo Board
- Launch configuration: debug settings

XSDB

- XSDB = Xilinx System Debugger
- Uses `hw_server` as debug engine to communicate with CPU on Zybo Board
- Launch configuration: debug settings
    - Open `Debug Configurations`

XSDB

- XSDB = Xilinx System Debugger
- Uses `hw_server` as debug engine to communicate with CPU on Zybo Board
- Launch configuration: debug settings
  - Open `Debug Configurations`
  - `Main` : build configuration, program arguments, ...

XSDB

- XSDB = Xilinx System Debugger
- Uses `hw_server` as debug engine to communicate with CPU on Zybo Board
- Launch configuration: debug settings
  - Open `Debug Configurations`
  - `Main` : build configuration, program arguments, ...
- Remote debugging

XSDB

- XSDB = Xilinx System Debugger
- Uses `hw_server` as debug engine to communicate with CPU on Zybo Board
- Launch configuration: debug settings
  - Open `Debug Configurations`
  - `Main` : build configuration, program arguments, ...
- Remote debugging
  - Remote machine: runs `hw_server` from XSCT console

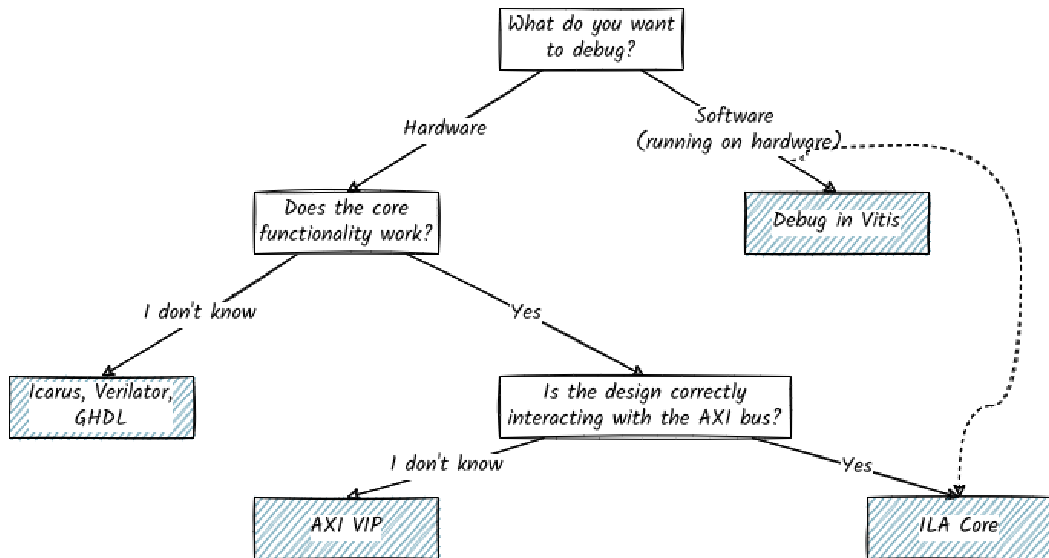
XSDB

- XSDB = Xilinx System Debugger
- Uses `hw_server` as debug engine to communicate with CPU on Zybo Board
- Launch configuration: debug settings
  - Open `Debug Configurations`
  - `Main` : build configuration, program arguments, ...
- Remote debugging
  - Remote machine: runs `hw_server` from XSCT console
  - Local machine: specify hostname/IP address and port

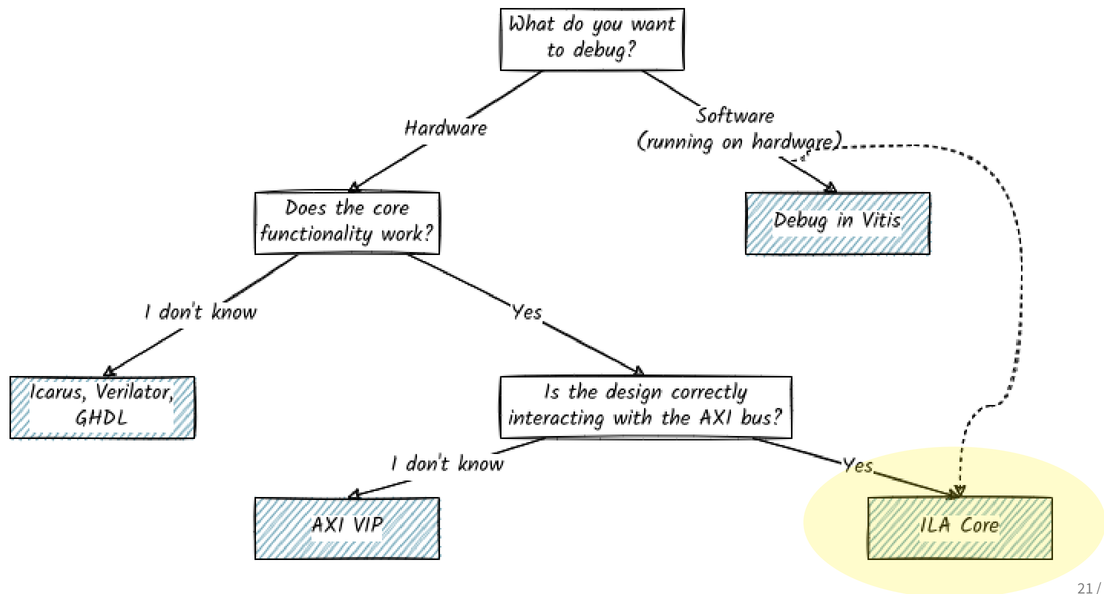
Debugging bare-metal application in Vitis

1. Build your project
2. Connect your board via USB
3. Bare-metal applications: `Debug As - 1 Launch Hardware`
4. Connect `Vitis Serial Terminal`

Can’t decide?



Can’t decide?



Using ILA Cores

JTAG

- Industry standard for debugging designs after manufacture

JTAG

- Industry standard for debugging designs after manufacture
- Motivation: testing a board with many IO pins is difficult

JTAG

- Industry standard for debugging designs after manufacture
- Motivation: testing a board with many IO pins is difficult
- Boundary Scan Testing

JTAG

- Industry standard for debugging designs after manufacture
- Motivation: testing a board with many IO pins is difficult
- Boundary Scan Testing
  - For each IO pin: insert a small logic cell between internal logic and physical pin

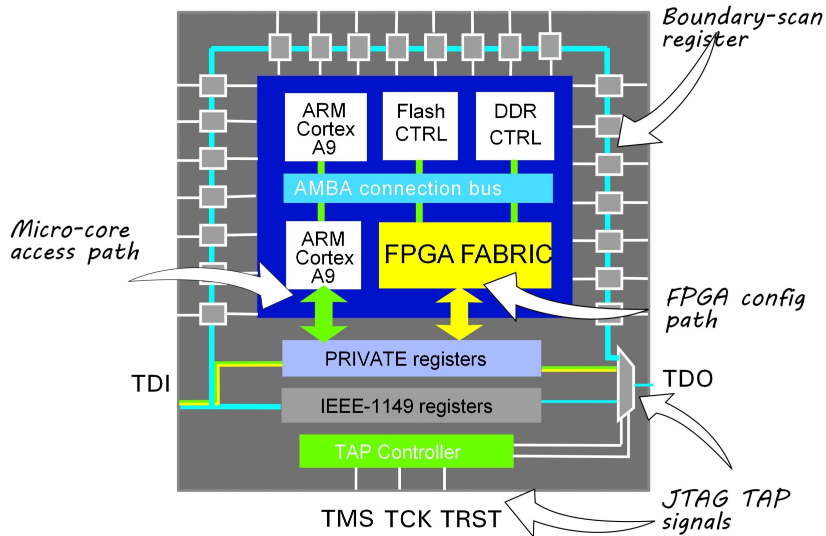
JTAG

- Industry standard for debugging designs after manufacture
- Motivation: testing a board with many IO pins is difficult
- Boundary Scan Testing
  - For each IO pin: insert a small logic cell between internal logic and physical pin
  - Connect all these logic cells to the TAP (test access port)

JTAG

- Industry standard for debugging designs after manufacture
- Motivation: testing a board with many IO pins is difficult
- Boundary Scan Testing
  - For each IO pin: insert a small logic cell between internal logic and physical pin
  - Connect all these logic cells to the TAP (test access port)
  - TAP can read and manipulate IO pin through logic cell

JTAG



ILA

- ILA = Integrated Logic Analyzer

ILA

- ILA = Integrated Logic Analyzer
- IP core to monitor internal signals of a design

ILA

- ILA = Integrated Logic Analyzer
- IP core to monitor internal signals of a design
- Only for synthesized designs (opposite of AXI VIP)

ILA

- ILA = Integrated Logic Analyzer
- IP core to monitor internal signals of a design
- Only for synthesized designs (opposite of AXI VIP)
- ILA probes: connected to internal wires, deliver wire value

ILA

- ILA = Integrated Logic Analyzer
- IP core to monitor internal signals of a design
- Only for synthesized designs (opposite of AXI VIP)
- ILA probes: connected to internal wires, deliver wire value
  - 1 probe = 1 wire

ILA

- ILA = Integrated Logic Analyzer
- IP core to monitor internal signals of a design
- Only for synthesized designs (opposite of AXI VIP)
- ILA probes: connected to internal wires, deliver wire value
  - 1 probe = 1 wire
  - Every probe is connected to trigger comparator.

ILA

- ILA = Integrated Logic Analyzer
- IP core to monitor internal signals of a design
- Only for synthesized designs (opposite of AXI VIP)
- ILA probes: connected to internal wires, deliver wire value
  - 1 probe = 1 wire
  - Every probe is connected to trigger comparator.
  - If trigger condition evaluates to true: ILA delivers trace measurement

ILA

- ILA = Integrated Logic Analyzer
- IP core to monitor internal signals of a design
- Only for synthesized designs (opposite of AXI VIP)
- ILA probes: connected to internal wires, deliver wire value
  - 1 probe = 1 wire
  - Every probe is connected to trigger comparator.
  - If trigger condition evaluates to true: ILA delivers trace measurement
- When should I use this method?

ILA

- ILA = Integrated Logic Analyzer
- IP core to monitor internal signals of a design
- Only for synthesized designs (opposite of AXI VIP)
- ILA probes: connected to internal wires, deliver wire value
  - 1 probe = 1 wire
  - Every probe is connected to trigger comparator.
  - If trigger condition evaluates to true: ILA delivers trace measurement
- When should I use this method?
  - I have already verified in simulations that my hardware and software are bug free, but something still does not work out.

ILA

- ILA = Integrated Logic Analyzer
- IP core to monitor internal signals of a design
- Only for synthesized designs (opposite of AXI VIP)
- ILA probes: connected to internal wires, deliver wire value
  - 1 probe = 1 wire
  - Every probe is connected to trigger comparator.
  - If trigger condition evaluates to true: ILA delivers trace measurement
- When should I use this method?
  - I have already verified in simulations that my hardware and software are bug free, but something still does not work out.
  - I think that synthesis/implementation introduces a bug

ILA workflow

1. In Vivado, add a new IP core to block design: `System ILA`
2. Set `Monitor Type = Native` and choose the number of probes
3. For each probe, configure the probe width and trigger.
4. Finish adding the IP and connect the ILA to the system clock.
5. For any wire to debug: select `Debug`
6. Generate bitstream and open the HW manager. Program the device (with `Bitstream file` and `Debug probes file`)
7. Run the SW in Vitis
8. In Vivado, open the HW manager and refresh target.