http://PollEv.com/bagi915

SoC Debugging Tutorial

Digital System Integration and Programming

Barbara Gigerl, Rishub Nagpal

October 20th, 2021

IAIK – Graz University of Technology

- Simulation of hardware designs
- Using AXI VIP
- Using ILA Cores
- Debugging SW in Vitis

Simulation of hardware designs

Prerequisites

- When should I use this method?
  - I have a small hardware design and want to get a rough idea of the functionality
  - No software involved
  - I want to find functional bugs in my hardware design
- Install GTKWave `https://github.com/gtkwave/gtkwave`
- Option 1: Install Icarus Verilog `http://iverilog.icarus.com/`
- Option 2: Install Verilator `https://www.veripool.org/verilator/`
- Example design: Fibonacci numbers

  `https://extgit.iaik.tugraz.at/sip/tutorials/-/tree/master/fibonacci`

- Simple but slow
- Testbench in Verilog
  - Instantiate test module
  - Create clock and reset control signals
  - Optional: `$display`, `$dumpfile`, `$dumpvars`, `$monitor`

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

- Fast but 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>
```

- Viewer for VCD traces
- File - Open New Tab - Select vcd file
- Hint: Use Save Files to restore previous view configuration

Using the AXI VIP

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

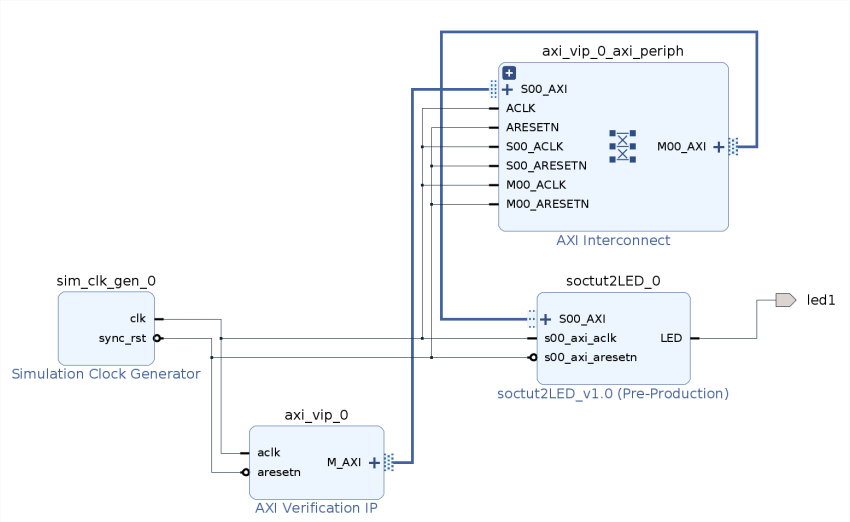
- AXI VIP = AXI Verification IP
- Simulate your IP core as an AXI master or minion
- Simulation-only (cannot be synthesized)
- Modes:
  - AXI master VIP: creates read/write transactions for AXI minion DUT
  - AXI minion VIP: reads payload, writes responses, ... for AXI master DUT
  - AXI pass-through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

Preparing the test setup

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

Hint: make sure to exclude all other (BD-)sources!

Example test setup



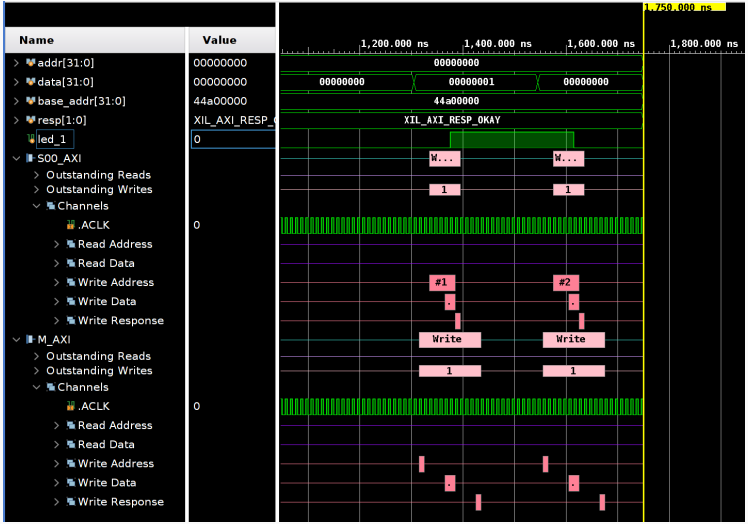
1. Add a new simulation source (`tb.sv`)
2. Import: `import axi_vip_pkg::*;` and `import <axi_vip_name>_pkg::*;` Hint: use `get_ips *vip*` to find out name
3. Instantiate the HDL wrapper
4. Add a new AXI master agent: `<axi_vip_name>_mst_t master_agent;`
   - Master agent can be used to generate AXI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

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

Example simulation



Debugging SW in Vitis

- 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 = 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

1. Build your project
2. Connect your board via USB

```
xsct% targets
  5  whole scan chain (DR shift output all ones)
  1  APU
     2  ARM Cortex-A9 MPCore #0 (Running)
     3  ARM Cortex-A9 MPCore #1 (Running)
  4* xc7z010
```

3. Bare-metal applications: Debug As - 1 Launch Hardware
4. Connect Vitis Serial Terminal

Using ILA Cores

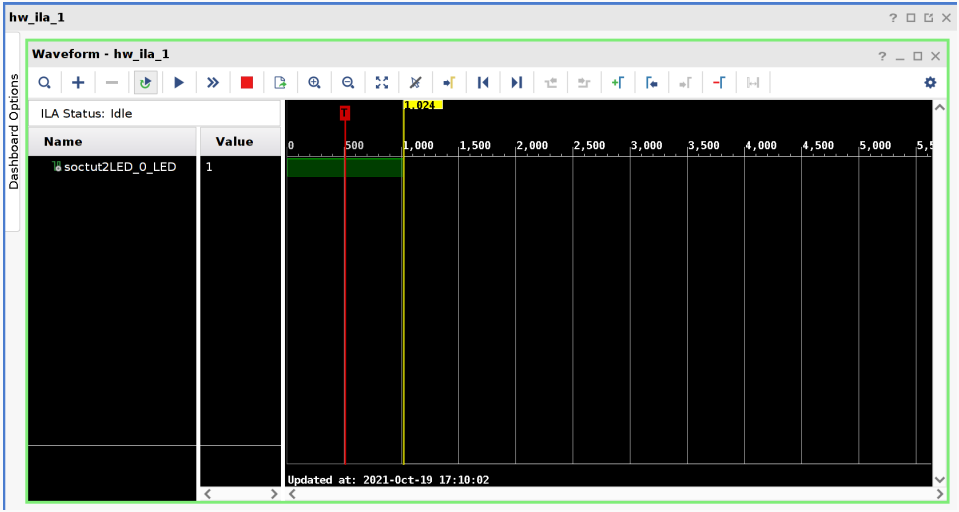
- Industry standard for debugging designs after manufacture
- Motivation: testing a board with many IO paths is difficult
- Boundary Scan Testing
  - For each IO pin: insert a small logic cell between internal logic and physical pin
  - Connect all these logic cells to the TAP (test access port)
  -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

- 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 Vitis, 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.

Example ILA


hw_ila_1
Waveform - hw_ila_1
Dashboard Options
ILA Status: Idle
Name
Value
soctut2LED_0_LED
1
1,024
0
500
1,000
1,500
2,000
2,500
3,000
3,500
4,000
4,500
5,000
Updated at: 2021-Oct-19 17:10:02