Lecture: Design Simulation and Testing

Prof. Jakub Szefer
Dept. of Electrical Engineering, Yale University

EENG 428 / ENAS 968
Cloud FPGA
Design Simulation and Testing on Amazon F1

- Designs implemented on Cloud FPGAs in Amazon F1 include many modules and IP cores not just the user’s logic, and also associated software
  - The design simulation need to consider all of the components
  - The design needs to be progressively tested: from individual modules, to whole hardware, to whole software-hardware system

- Design parts simulation and testing:
  - C code interacting with the hardware
  - Python code interacting with the hardware
Simulation From Modules to Whole Design

When developing Cloud FPGA designs, they have to be simulated to validate correct execution and behavior of the hardware (and software)

1. Simulate individual Verilog modules with testbenches
2. Simulate whole custom logic hardware design with simulated AXI bus
3. Simulate whole F1 hardware and software tests
4. Run the actual design on the F1 FPGAs

Each module should be checked with a testbench for that module, catch errors in module design

All the modules connected together should be checked to work with the AXI bus (from PCIe module)

Simulate everything including software tests to ensure software-hardware communication works as expected

Full simulation may still result in some errors due to difference between real PCIe, AXI, etc. and simulated protocols
Simulating Individual Modules

First, each Verilog module should be simulated with a testbench to ensure it is working as expected.

- Testing does not require use of Amazon F1 resources, can be done locally using `iverilog`, for example.
- Fast for most basic modules, as long as size of module is small.
- Testing can be automated with frameworks such as `cocotb` which uses Python to write tests for the hardware modules, and works with `iverilog` (and others).

Simulating Custom Logic with Simulated AXI

Second, once each module is tested, the whole custom logic design needs to be check that it works with the AXI interface (used to connect to the PCIe module).

• Need to write a simulation of the AXI master (PCIe module’s interface to user’s logic)
• Testbench needs to generate correct AXI signals with the right timing
• Check that user’s logic replies with correct data or behavior
• Likewise can automate the tests with frameworks such as cocotb to improve the testing coverage
Third, the whole hardware design can be simulated with tests in SystemVerilog and/or C through Amazon’s HDK.

- This simulation covers all the parts of the design
- SystemVerilog tests can simulate `peek()` and `poke()` commands to simulate software functionality
- C tests can simulate real C code interacting with the hardware
  - The same C code can be used for simulation and for use with real hardware FPGAs
- Because the Amazon F1 designs use proprietary (Xilinx) IP modules for PCIe, DRAM, etc., the designs can not be simulated easily with open-source tools; instead commercial tools are needed to be used
RTL Simulators used in Amazon F1

• Amazon supports a number of simulators, they can all load proprietary IP for the PCIe, DRAM, etc. and simulate the whole shell and user’s custom logic

• Table of supported simulators, from fall 2019:

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Vivado 2017.4</th>
<th>Vivado 2018.2</th>
<th>Vivado 2018.3</th>
<th>Vivado 2019.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xilinx Vivado XSIM</td>
<td>Vivado v2017.4</td>
<td>Vivado v2018.2</td>
<td>Vivado v2018.3</td>
<td>Vivado v2019.1</td>
</tr>
<tr>
<td>Mentor Graphics Questa</td>
<td>10.6b</td>
<td>10.6c_1</td>
<td>10.6c_1</td>
<td>10.7c</td>
</tr>
<tr>
<td>Cadence Incisive Enterprise Simulator(IES)</td>
<td>15.20.063</td>
<td>15.20.063</td>
<td>15.20.063</td>
<td>15.20.065</td>
</tr>
</tbody>
</table>

• For course-related work, the default xsim is sufficient
Amazon F1 Simulation Framework

- The simulation framework provides simulation of the shell (PCIe, etc.) and DRAM modules, and allows for simulation of user’s software code.

SystemVerilog test programs

Actual C code for the user’s software

Simulated shell behavior

Simulated memory behavior

Actual HDL code of the Custom Logic

SH BFM

SH Tasks

CL DUT

SH_FIM

SystemVerilog Tests

Host Memory

C Tests

DPI Tasks

AXI PCIeS

Other Signals

AXI PCIeM

AXI CL to SH DDR

sh_ddr

DDR4 Model 16GB

DDR4 Model 16GB

DDR4 Model 16GB

DDR4 Model 16GB
Custom Logic Directory Structure

- Each Custom Logic (CL) design has a fixed directory structure for all the hardware code, software code, tests, etc.
  - **build** – scripts and define files related to the build process
  - **design** – actual hardware code, given code is mostly in SystemVerilog
  - **software** – software associated with the design
  - **verif** – scripts and testbenches for validating the design

SystemVerilog test for the design
SystemVerilog Tests for Custom Logic Designs

- Example test for the Hello World example:

```verilog
module test_hello_world();

import tb_type_defines_pkg::*;

'include "cl_common_defines.vh" // CL Defines with register addresses

// AXI ID
parameter [5:0] AXI_ID = 6’h8;

logic [31:0] rdata;
logic [15:0] vdip_value;
logic [15:0] vled_value;

initial begin
    tb.power_up();
    tb.set_virtual_dip_switch().dip(0);
    vdip_value = tb.get_virtual_dip_switch();

    $display("value of vdip\%hx", vdip_value);
    $display("Writing %0xDEAD_BEEF to address %0xh", "HELLO_WORLD_REG_ADDR");
    tb.poke(addr("HELLO_WORLD_REG_ADDR").data(32’hDEAD_BEEF), .id(AXI_ID), .size(DataSize:16), .intf(AxiPort::PORT_DCL)); // write register

    $display("start read & write");
    b.peek(addr("HELLO_WORLD_REG_ADDR"), .data(rdata), .id(AXI_ID), .size(DataSize:16), .intf(AxiPort::PORT_DCL)); // start read
    $display("Reading %0xh%x from address %0xh", rdata, "HELLO_WORLD_REG_ADDR");

    if (rdata == 32’hEFE8) // Check for byte swap in register read
        $display("TEST PASSED");
    else
        $display("TEST FAILED");
    
    if (rdata == 32’hDEAD_BEEF) // Check for LED register read
        $display("TEST PASSED");
    else
        $display("TEST FAILED");

    vled_value = tb.get_virtual_led();
    $display("value of vled\%hx", vled_value);
    tb.kernel_reset();
    tb.power_down();

endmodule // test_hello_world
```

Simulated `peek()` and `poke()` commands
Demo: Running Simulations on Amazon F1

Tutorial on Setting-up GUI and Running Simulations on F1 Developer AMIs will show you how to run simulations of the Amazon F1 Cloud FPGA designs.
Summary

• When developing Cloud FPGA designs, they have to be simulated to validate correct execution and behavior of the hardware and the associated software
  1. Simulate individual Verilog modules with testbenches
  2. Simulate whole custom logic hardware design with simulated AXI bus
  3. Simulate whole F1 hardware and software tests
  4. Run the actual design on the F1 FPGAs

• Simulation is much faster compared to compiling the designs for the real hardware, but:
  • Differences can exist between simulated interfaces and real interfaces
  • Physical variations are not modeled in the simulation
References

Links to HDK pages from Amazon's AWS git include the version number, some documents seem to be not updated as frequently as others, thus the listed versions are not always the same. Most recent version as of when the slides were made was v1.4.10 for the HDK. It was updated to v1.4.10 during the fall 2019 semester.

1. “RTL Simulation for Verilog/VHDL Custom Logic Design with AWS HDK, RELEASE V1.4.11” Available at: https://github.com/aws/aws-fpga/blob/master/hdk/docs/RTL_Simulating_CL_Designs.md