System Level Verification Issues

Sungho Kang
Yonsei University
Outline

- Introduction
- Test Plan
- Application-Based Verification
- Fast Prototype vs 100% Testing
- Gate Level Verification
- Choosing Simulation Tools
- Specialized Hardware
Importance of Verification

- Verifying functionality and timing at the system level is probably the most difficult and important aspect of SOC design.

- Successful system level verification depends on the following factors:
  - Quality of the test plan
  - Quality and abstraction level of the models and testbenches used
  - Quality and performance of the verification tools
  - Robustness of the individual pre-designed blocks
Test Plan

- Divide-and-conquer
  - Verify that the lead nodes - the lowest level individual blocks - of the design hierarchy are functionally correct as stand-alone units
  - Verify that the interfaces between blocks are functionally correct, first in terms of the transaction types and then in terms of data content
  - Run a set of increasingly complex applications on the full chip
  - Prototype the full chip and run a full set of application software for final verification
  - Decide when it is appropriate to release the chip to production
Block Level Verification

- Block level verification uses code coverage tools and a rigorous methodology to verify the macro at the RTL level as thoroughly as possible.
- A physical prototype is then built to provide silicon verification of functional correctness.
Interface Verification

- Once the individual blocks have been verified as stand-alone units, the interfaces between blocks need to be verified.
- The connections between blocks can be either point-to-point or on-chip buses.
- Because of the regular structure of these interfaces, it is usually possible to talk about transaction between blocks.
- The idea is that there are only a few permitted sequence of control and data signals.
Transaction Verification

- Interface testing begins by listing all of the transaction types that can occur at each interface, and systematically testing each one.
- If the system design restricts transactions to a relatively small set of types, it is fairly easy to generate all possible transaction types and sequences of transaction types and to verify the correct operation of the interfaces to these transactions.
- Once this is done, all that remains is to test the behavior of the blocks to different data values in the transactions.
Data or Behavioral Verification

- It uses simple, behavioral models, essentially bus functional models, for all blocks except the block under test
  - You should use full RTL design for the block under test and generate the desired data sequences and transaction from the behavior functional blocks
  - You can construct test cases either from your knowledge of the system or by random generation

- Automatic checking of the block’s behavior under these sequence of transaction is nontrivial and depends on how easy it is to characterize the correct behavior of the block

- Another method for dealing with the problem of unanticipated or illegal inputs is to design a checker into the block interface itself
  - This checker can suppress inputs that are not legal and prevent the block from getting into incorrect states
Application-Based Verification

- Verification based on running real application code is essential for achieving a high quality design.
- This form of verification presents some major challenges.

- Two approaches to addressing this problem:
  - Increase the level of abstraction so that software simulators running on workstation run faster.
  - Use specialized hardware for performing verification, such as emulation or rapid prototyping systems.
Canonical SOC Design

**Application-Based**

PERIPHERALS

PROCESOR

MEMORY CONTROLLER

MEMORY

SYSTEM BUS

I/O INTERFACE

DATA TRANSFORMATION

I/O INTERFACE
Testbench for Canonical Design

Application-based

YONSEI UNIVERSITY

CS&RSOC

Application software/drivers/RTOS

Compiler

Processor C/C++
RTL interface

Memory Controller C/C++
RTL interface

Memory C/C++

Other Peripherals (RTL)

I/O Interface (RTL)

Tata Transformation (RTL)

I/O Interface (RTL)

CHIP

Communication bus functional model (RTL)

Sequence generator/analyzer

Communication bus functional model (RTL)
Testbench for Canonical Design

- The full RTL model is used as the simulation model for most of the functional blocks
- Behavioral or ISA (Instruction Set Architecture) models may be used for memory and the microprocessor
- Bus functional models and monitors are used to generate and check transactions with the communication blocks
- It is possible to generate real application code for the processor and run it on the simulation model
Fast Prototype vs 100% Testing

- Running significant amounts of real application code is the only way to reach this level of confidence in an SOC design
- For most designs, this level of testing requires running at or near real time speeds
- The only available technologies for achieving this kind of performance involve some form of rapid prototyping
FPGA and LPGA Prototyping

- For small designs, it is practical to build an FPGA or LPGA (Laser Programmable Gate Array) prototype.
- FPGAs have the advantage of being programmable, allowing rapid turnaround of bug fixes.
- Both FPGAs and LPGAs lag state-of-the-art ASIC technologies in gate count and clock speed by significant amounts.
- They are much more appropriate for prototyping individual blocks or macros than for prototyping SOC design.
Emulation Based Testing

- They provide programmable interconnect, fixed board designs, relatively large gate counts, and special memory and processor support
- Recent developments in moving form FPGAs to processor-based architectures have helped to resolve partitioning and interconnect problems
- Emulation can provide excellent performance for large-chip verification if the entire design can be placed in the emulation engine itself
- If any significant part of the circuit or testbench is located on the host, there is significant degradation of performance
- Emulation can provide simulation performance of one or two orders of magnitude less than real time, and many orders of magnitude faster than simulation
Silicon Prototyping

• If an SOC design is too large for FPGA/LPGA prototyping and emulation is not practical, then building a real silicon prototype may be the best option
• Instead of extending the verification phase, it may be faster and easier to build an actual chip and debug it in the system
• Critical issues
  ▪ The bug rate from simulation testing should have peaked and be on its way down
  ▪ The time to determine that a bug exists should be much greater than the time to fix it
  ▪ The cost of fabricating and testing the chip is on the same order as the cost of finding the next \( n \) bugs, where \( n \) is the anticipated number of critical bug remaining
  ▪ Enough functionality has been verified that the likely bugs in the prototype should not be severe enough to prevent extensive testing of other features
Silicon Prototyping

- Design features
  - Good debug structures for controlling and observing the system, especially system buses
  - The ability to selectively reset individual blocks in the design
  - The ability to selectively disable various blocks to prevent bugs in these blocks from affecting operation of the rest of the system
Gate Level Verification

- Sign-Off Simulation
- Formal Verification
- Gate-Level Simulation with Unit Delay
- Gate-Level Simulation with Full Timing
Sign-Off Simulation

- In the past, gate level simulation has been the final step before signing off an ASIC design
  - ASIC vendors have required gate-level simulation and parallel test vectors as part of signoff, using the parallel vectors as part of manufacturing test
- Today, for 100K gate and larger designs, signoff simulation is typically done running verilog simulation with back-annotated delays on hardware accelerators from IKOS
  - Running full timing, gate level simulations in software simulators is simply not feasible at these gate counts
- RTL sign-off, where no gate-level simulation is performed, is becoming increasingly common
  - However, most ASIC vendors still require that all manufacture-test vectors submitted with a design be simulated on a sign-off quality simulator with fully back-annotated delay information and all hazard checking enabled
The requirement is rapidly becoming problematic

- Though, full timing simulation of a million-gate ASIC is not possible without very expensive hardware accelerators, and even then it is very slow.
- Parallel vectors typically have very low fault coverage unless a large and expensive effort is made to extend them.
  - As a result, they can be used only to verify the gross functionality of the chip.
- Parallel vectors do not exercise all the critical timing paths, for the same reason they don’t achieve high fault coverage.
  - As a result, they do not provide a sufficient verification that the chip meets timing.
Underlying problems

- Verification that synthesis has generated a correct netlist, and that subsequent operations such as scan and clock insertion have not changed circuit functionality
- Verification that the chip, when fabricated, will meet timing
- A manufacturing test that assures that a chip passes test is free of manufacturing defects
Formal Verification

• Formal verification checks
  ▪ The synthesized netlist
  ▪ The netlist after test logic is inserted
  ▪ The netlist after clock tree insertion and layout
  ▪ Hand edits

• One key benefit of formal verification is that it allows the RTL to remain the golden reference for the design, regardless of modifications made to the final netlist

• Even if the functionality of the circuit is changed by a last minute by editing the netlist, the same modification can be retrofitted into the RTL and the equivalence of the modified RTL and netlist can be verified
Gate-Level Simulation with Unit Delay

- It is much faster than full-timing simulation, but much slower than RTL simulation

- Usage
  - The chip initializes properly
  - The gate implementation functionally matches the RTL description

- Since it can be time-consuming and resource-intensive, it is usually good to begin unit-delay simulation as soon as you complete a netlist for your chip, even though the chip may not meet timing
Gate-Level Simulation with Full Timing

- Full timing simulation on large chips is very slow, and should be used only where absolutely necessary.
- This technique is particularly useful for validating asynchronous logic, embedded asynchronous RAM interfaces, and single-cycle timing exceptions.
- These tests should be run with the back-annotated timing information from the place and route tools, and run with hazards enabled.
- They should be run with worst case timing to check for long paths, and with best-case timing to check for minimum path delay problems.
Choosing Simulation Tools

- **Subblock module test stage**
  - Interpreted, event-driven simulator (VSS, Verilog-XL, VCL)

- **Block-level integration stage**
  - Compiled, event-driven simulator or cycle-based simulator

- **Chip-level integration stage**
  - Cycle-based simulation to start
  - Modules can migrate to emulation when relatively bug-free
  - Testbench migrates to emulation last

- **Software testing stage**
  - Emulation
  - Chip and testbench are in the emulator for max performance
Choosing Simulation Tools

- For the lowest level of subblock verification, an interpreted, event-driven simulator is the most useful.
- For block integration, the design team will probably want to migrate to a compiled simulator as soon as the code is relatively stable:
  - A compiled cycle-based simulator can provide even faster run times, but with some restrictions on coding style.
- When cycle-based simulation becomes too slow to find additional bugs, the design needs to be moved to an emulator.
Specialized Hardware

- Hardware acceleration
- Emulation

- These new systems hold much promise for addressing the problems of high-speed system verification.
- The success of these systems will depend on the capabilities of the software that goes with them: compilers, debuggers, and HW/SW cosimulation support.
- These systems will continue to compete against much less expensive approaches: simulation using higher levels of abstraction and rapid prototyping.
Accelerated Verification

Models
- RTL
- Memory blocks
- Soft macros

Stimulus
- Synthesizable testbenches
- Test vectors

Testbenches
- Test vectors
- Verilog-XL
- "C"

Software environment
- XRay Debugger
- Seamless CVE

Physical environment
- Hard macros
- Hardware testbenches

In-circuit verification
- Target system
Accelerated Verification

- Models
- Physical environment
- In-circuit verification
- System environment
- Testbenches
- Stimulus
RTL Acceleration

- Designers continue to use software simulators to debug their designs, but a threshold is reached where simulator performance becomes a major bottleneck for functional verification at the RTL level, especially for large SOC design.

- At some point in the design cycle, system simulation and debug may be more appropriately done on the emulator than on the simulator.
Software Driven Verification

- Verification of the hardware using real software
- Verification of the software using real hardware well before the actual chip is built

- The high performance of emulation systems allows the design team to
  - Develop and debug the low level hardware device drivers on the virtual prototype with hardware execution speeds that can approach near real time
  - Boot the operating system, initialize the printer driver, or place a phone call at the RTL phase of the design cycle
Traditional In-Circuit Verification

- As the design team reach the end of the design cycle, the last few bugs tend to be the most challenging to find.
- In-circuit testing of the design can be key tool at this stage of verification because the ultimate verification testbench is the real working system.
- One large manufacturer of routers routinely runs its next design for weeks on its actual network, allowing the router to deal with real traffic, with all of the asynchronous events that are impossible to model accurately.
Support for Intellectual Property

- Should offer very secure encryption mechanisms and advanced macro compile capabilities that allow IP developers to have complete control over what parts of the IP modules are visible to the end user.
Design Guidelines

- Use a simple clocking scheme
- Use registers (FFs), not latches
- Do not use combinational feedback loops
- Do not instantiate gates, pass transistors, delay lines, pulse generators, or any element that depends on absolute timing
- Avoid multi-cycle paths
- Avoid asynchronous memory
- Hierarchical, modular designs are generally easier to map to the emulation hardware than flat designs
- Larger register arrays should be modeled as memories, to take advantage of the special memory modeling hardware in the emulation system