SoC/ASIC/SoC-FPGA/S-ASIC Design and Verification Methodology

Intelop Corporation
4800 Great America Pkwy.
Ste-201
Santa Clara, CA. 95054
Ph: 408-496-0333, Fax: 408-496-0444
www.intelop.com

Courtesy of Cadence design

Challenges in Embedded Systems Design

- Real operating environment of ES is difficult to reproduce. -> In-system verification is necessary.
- Total design flow of ES until the implementation is long and complex. -> Verification must be started early.
- Design turn-around time must be short as the life-time of ES itself is quite short. -> Short verification cycle

Critical Issues

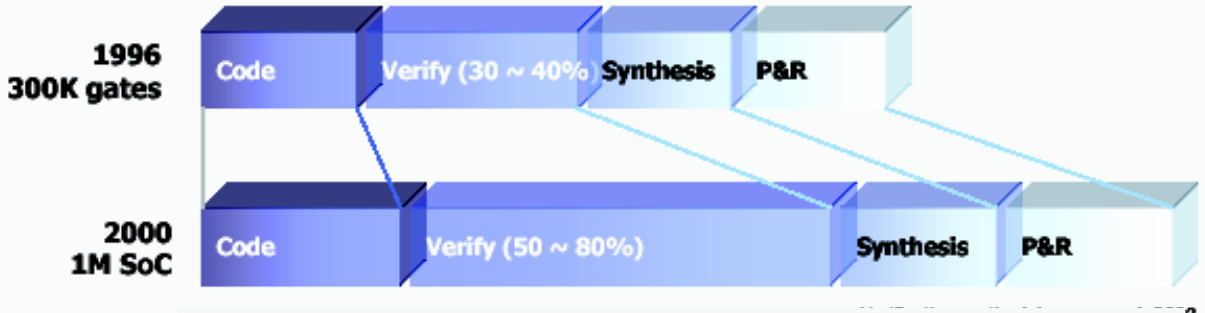
1. Verify Right :
   - Always make sure you have correct specifications to start with. (Frequent interaction with SPECIFIER, customer, marketing, etc.)
   - In-System Verification
   - Check Properties in Formal Techniques.
2. Verify Early
   - System-level, Heterogeneous Models, SW-HW
3. Verify Appropriately
   - HW-SW Co-simulation
4. Verify Fast
   - Hardware Acceleration, Emulation

Accommodate Multiple Levels of Design Representation

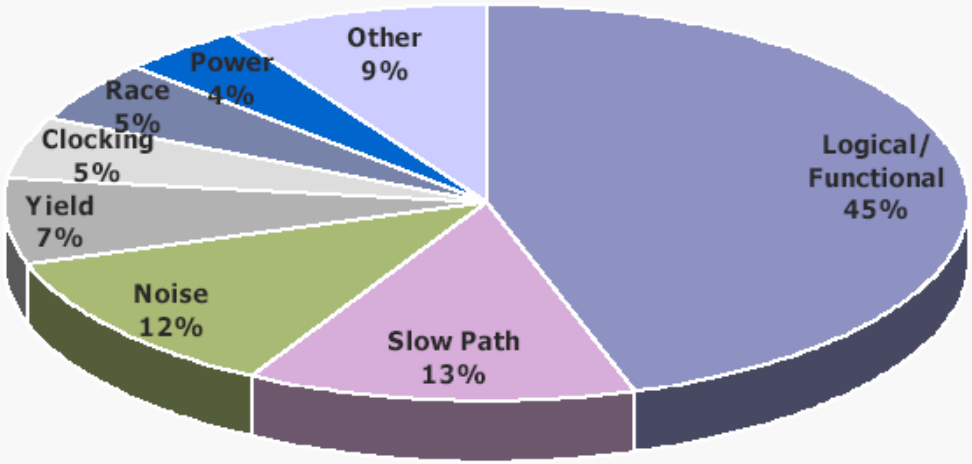
Exploit Hardware, Software and Interfacing mechanisms as Verification Tools

Verification Effort size

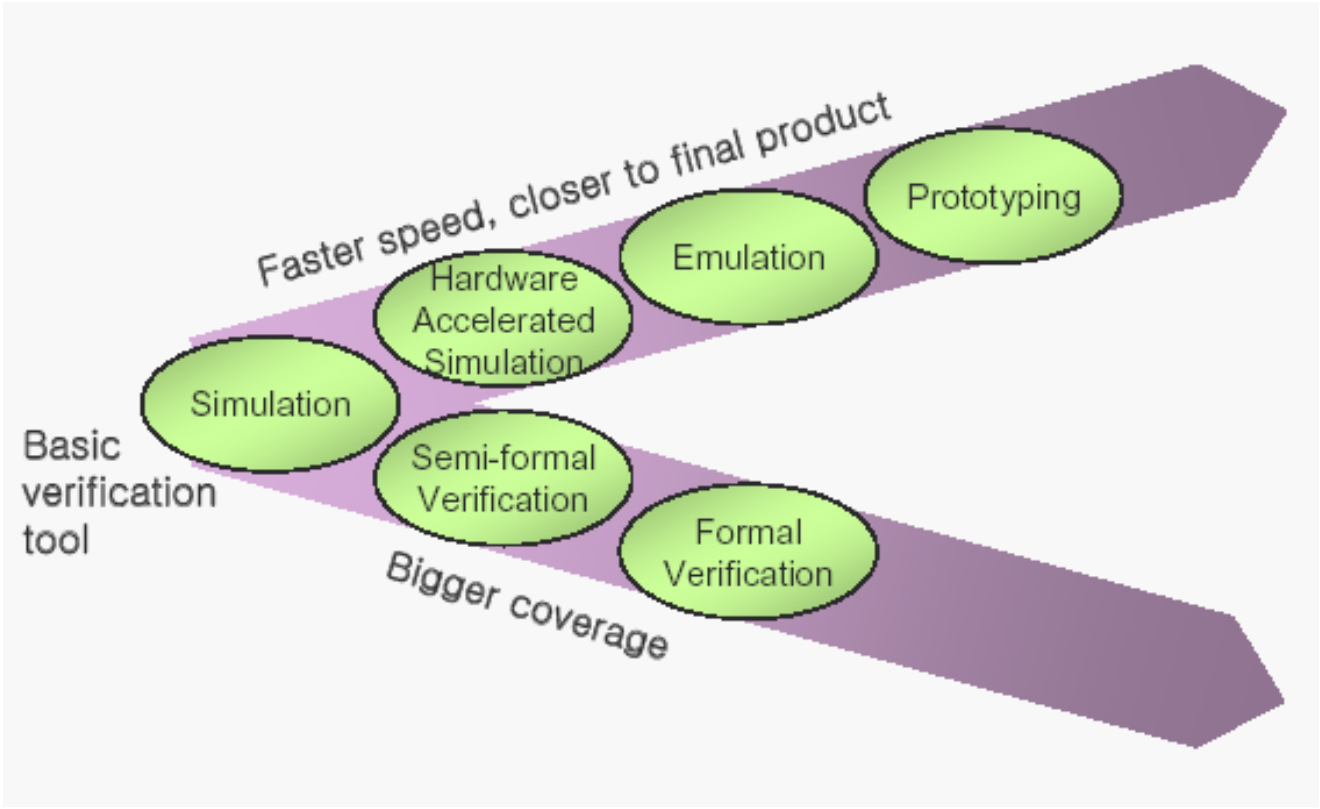
- Verification portion of design increases to anywhere from 50 to 80% of total development effort for the design.



- About 50% of flaws are functional flaws.
  - Need verification method to fix logical & functional flaws

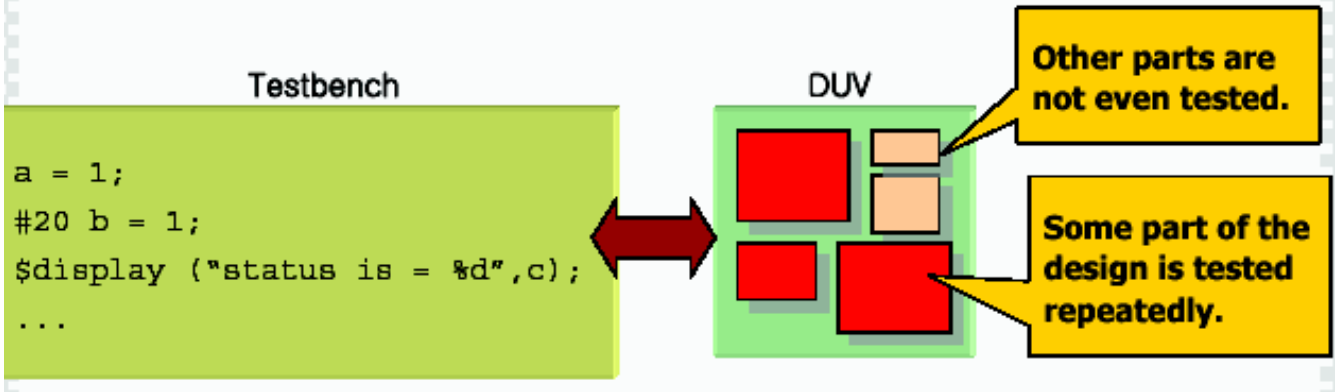


Overview of Verification Methodologies



Software Simulation

- Dynamic verification method
- Bugs are found by running the design implementation.
- Thoroughness depends on the test vector used.
- Some parts are tested repeatedly while other parts are not even tested.



Software Simulation

Pros

- The design size is limited only by the computing resource.
- Simulation can be started as soon as the RTL description is finished.
- Set-up cost is minimal.

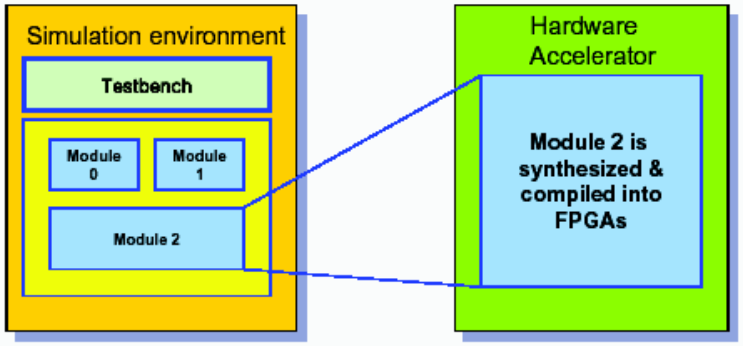
Cons

- Slow (~100 cycles/sec) ; Speed gap between the speed of software simulation and real silicon widens. (Simulation speed = size of the circuit simulated / speed of the simulation engine)
- The designer does not exactly know how much percentage of the design have been tested.

Hardware Acceleration

Simulation performance is improved by moving the time-consuming part of the design to hardware.

Usually, the software simulation communicates with FPGA-based hardware accelerator.



Pros

- Fast (100K cycles/sec)
- Cheaper than hardware emulation
- Debugging is easier as the circuit structure is unchanged.
- Not an Overhead : Deployed as a step stone in the gradual refinement

Cons (Obstacles to overcome)

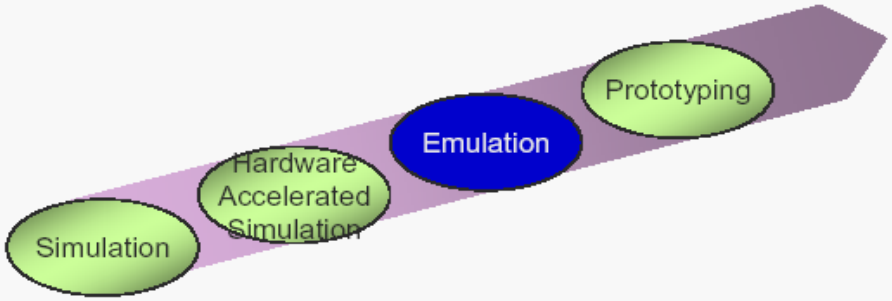
- Set-up time overhead to map RTL design into the hardware can be substantial.
- SW-HW communication speed can degrade the performance.
- Debugging of signals within the hardware can be difficult.

Emulation

Imitating the function of another system to achieve the same results as the imitated system.

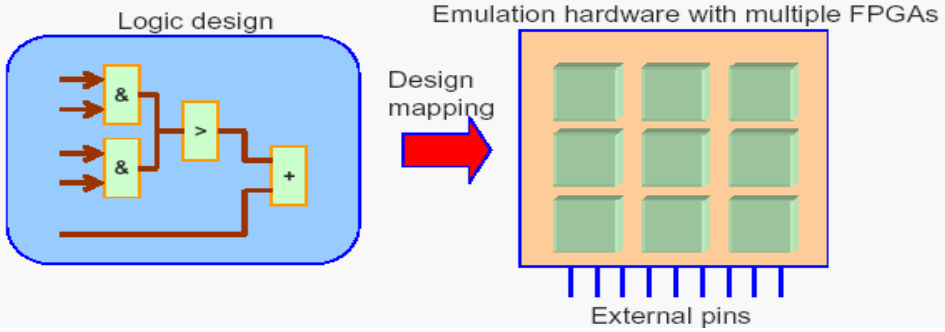
Usually, the emulation hardware comprises an array of FPGA's (or special-type processors) and interconnection scheme among them.

About 1000 times faster than simulation.



User logic design is mapped to emulation board with multiple FPGA's or special processors.

The emulation board has external interconnection hardware that emulates the pins of final chip.



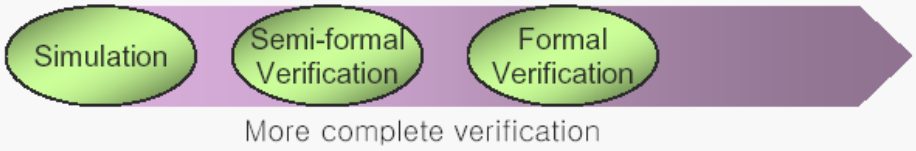
Overview of Verification Methodologies

Formal verification

- Application of logical reasoning to the development of digital system
- Both design and its specification are described by a language in which semantics are based on mathematical rigor.

Semi-formal verification

- Combination of simulation and formal verification.
- Formal verification cannot fully cover large designs, and simulation can come to aid in verifying the large design.



Formal Verification

Objective

- Check properties of model with all possible conditions

Pros

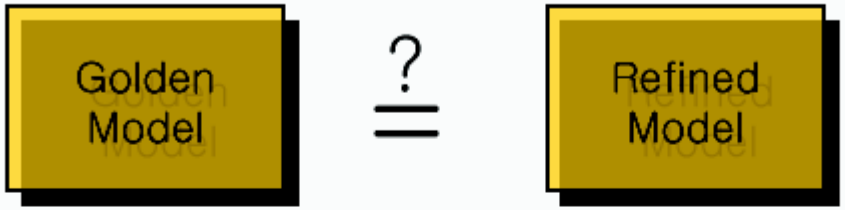
- Assures 100% coverage.
- Fast.

Cons

- Works only for small-size finite state systems.
- Uncomfortable due to culture difference (E.g., engineers are not familiar with the use of temporal logic used for "property" description in Model Checking)

Formal Verification : equivalence Check

Equivalence checker compares the golden model with the refined model.



Functional representations are extracted from the designs and compared mathematically.

Pros

- Exhaustive design coverage
- Very fast

Cons

- Memory explosion

Tools such as LEC (Verplex), Formality (Synopsys), FormalPro (Mentor) supports Equivalence checking.

Formal Verification : equivalence Check

Object

- Checks equivalence of two models
  - RTL vs. gate
  - Before optimization vs. after optimization
  - Before test insertion vs. after
  - Reference model vs. implementation

Advantage

- Guarantee functional equivalence of two models for all input values

Disadvantage

- Needs golden reference model
- Targets implementation errors rather than design bugs

Theorem Proving

Deductive verification

- Use axioms and proof rules to model the system (formal system).
- State the property to be verified as a theorem of this formal system.
- Derive this theorem with the help of a theorem-prover which generates rules derivable from axiom and premises.
- Useful for verifying algorithm

Disadvantage

- Very hard to automate.
- Requires user interaction.
- Deriving the formal system can be quite cumbersome.
- Requires an expert to use the theorem-prover.

Industrial success story

- AMD K7 floating point verification
- Intel instruction decoder verification

Formal Verification : Model Check

Model checking verifies that the design satisfies a property specified using temporal logic.

Computational Tree Logic

- Specify the temporal relationship among states in FSM with temporal operators;
  - A (always), E (exists) – path quantifier
  - G (global), F (future), X (next), U (until) – temporal modality

Formal Verification : Model Checking

Object

- Check properties of model with all possible conditions

Advantage

- Can be fully automated
- If the property does not hold, a counter-example will be generated
- Relatively easy to use

Problem

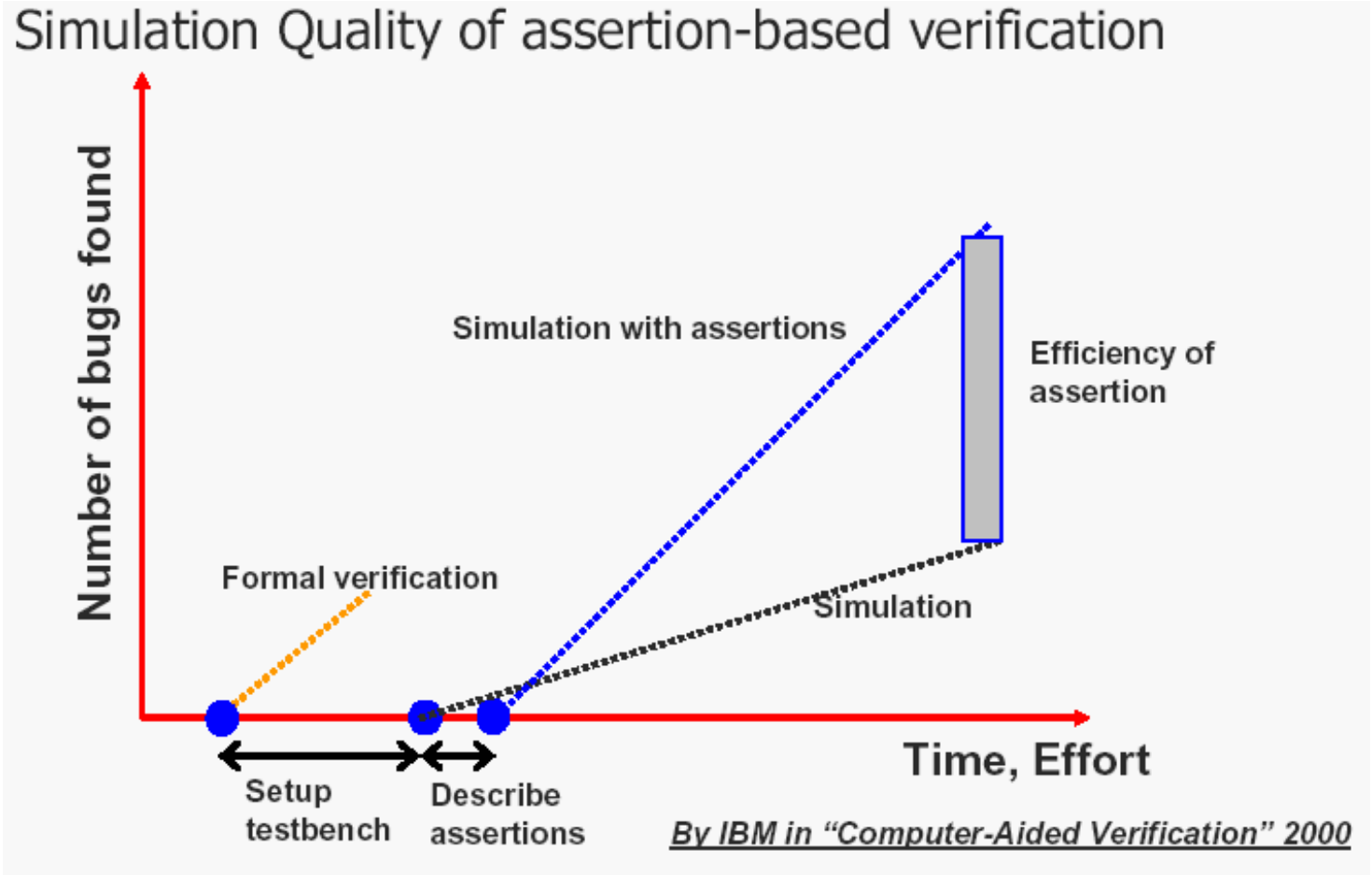
- Works (well) only for finite state systems.
- Needs abstraction or extraction which tend to cause errors

Formal Verification : Challenges

Challenges

- The most critical issue of formal verification is the "state explosion" problem.
- The application of current formal methods are limited to the design of up to 500 flip-flops.
- Researches about complexity reductions are :
  - Reachability analysis
  - Design state abstraction
  - Design decomposition
  - State projection

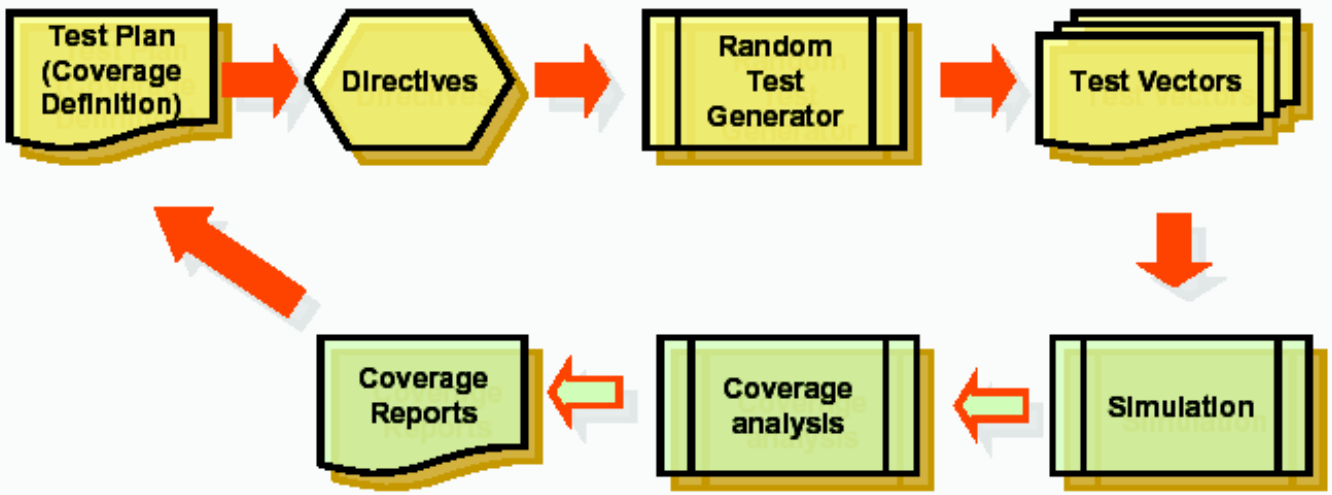
Semi-Formal Verification : Assertion



Semi-Formal Verification : Coverage

Coverage-directed verification

- Increase the probability of bug detection by checking the 'quality' of stimulus
- Used as a guide for the generation of input stimulus



Semi-Formal Verification : Coverage

Coverage metrics for coverage-directed verification

- Code-based metrics
  - Line/code block coverage
  - Branch/conditional coverage
  - Path coverage
- Circuit structure based metrics
  - Toggle coverage
  - Register activity
- State-space based metrics
  - Pair-arcs : usually covered by Line + condition coverage
- Spec.-based metrics
  - % of specification items satisfied

Semi-Formal Verification : Coverage

Coverage Checking tools

- VeriCover (Veritools)
- SureCov (Verisity)
- Coverscan (Cadence)
- HDLScore, VeriCov (Summit Design)
- HDLCover, VeriSure (TransEDA)
- Polaris (Synopsys)
- Covermeter (Synopsys)

Semi-Formal Verification

Pros

- Designer can measure the coverage of the test environment as the formal properties are checked during simulation.

Cons

- The simulation speed is degraded as the properties are checked during simulation.

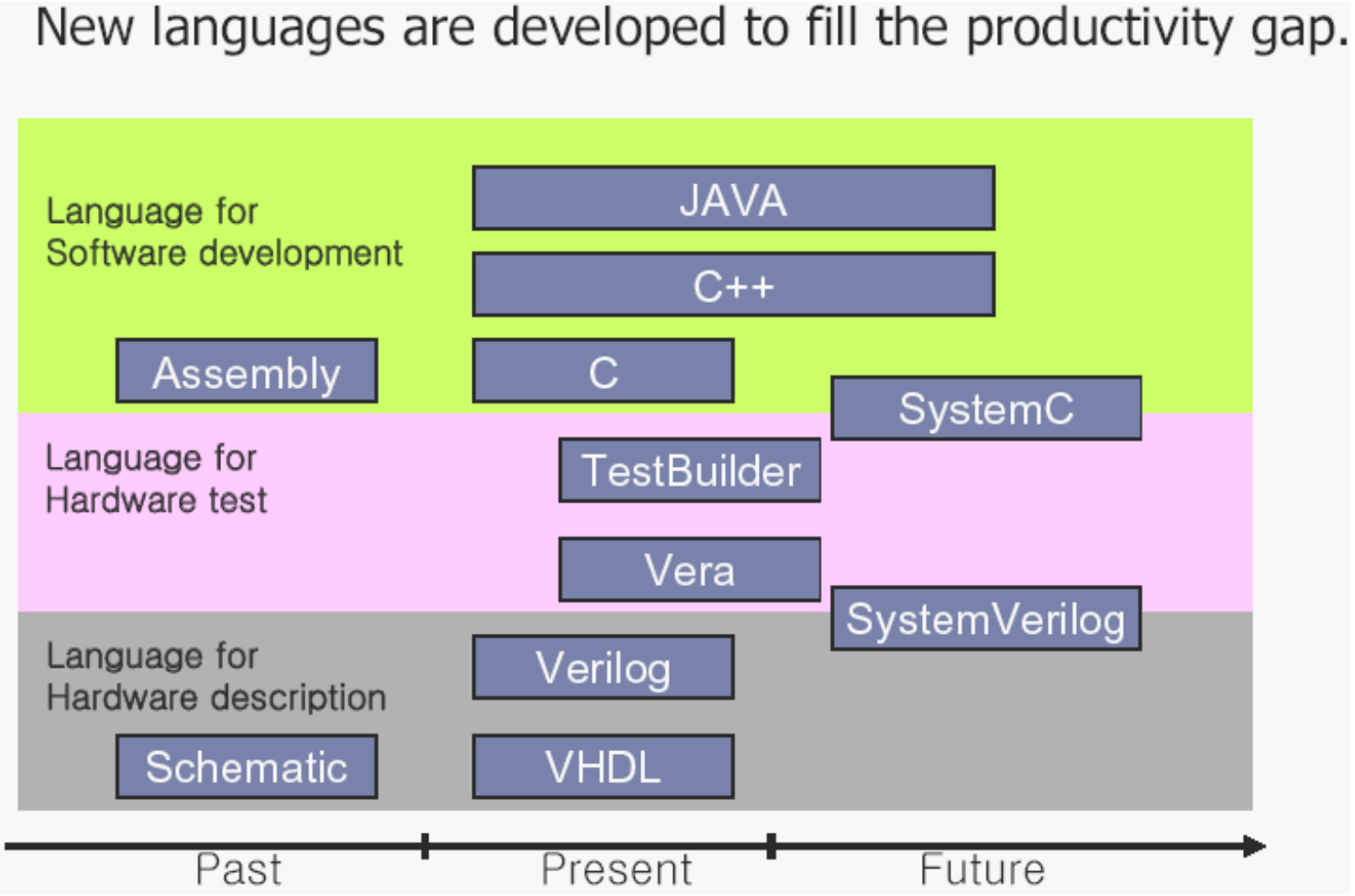
Challenges

- There is no unified testbench description method.
- It is difficult to guide the direction of test vectors to increase the coverage of the design.
- Development of more efficient coverage metric to represent the behavior of the design.

Design Complexity

	Gate counts	Comments
Simulation/Semi-formal verification	Unlimited	
Emulation/Hardware-accelerated simulation	1M~16M gates	Depends on the number of FPGA's in the architecture
Prototyping	1M~5M gates	Depends on the components on the board
Formal verification	< 10K gates	Limited to about 500 flip-flops due to state explosion

Language Heritage for SoC Design



SystemC in SoC Design

SystemC is a modeling platform consisting of

- A set of **C++ class library**,
- Including a **simulation kernel** that supports hardware modeling concepts at the system level, behavioral level and register transfer level.

SystemC enables us to effectively create

- A **cycle-accurate model** of
  - Software algorithm,
  - Hardware architecture, and
  - Interfaces of System-on-a-Chip.

Program in SystemC can be

- An **executable specification** of the system.

SystemC in SoC Design

Modules, ports, and ***signals*** ← for hierarchy

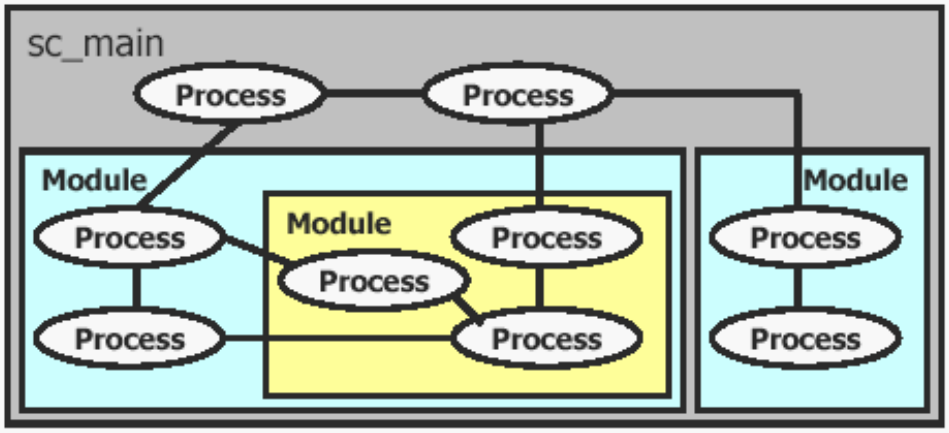
Processes ← for concurrency

Clocks ← for time

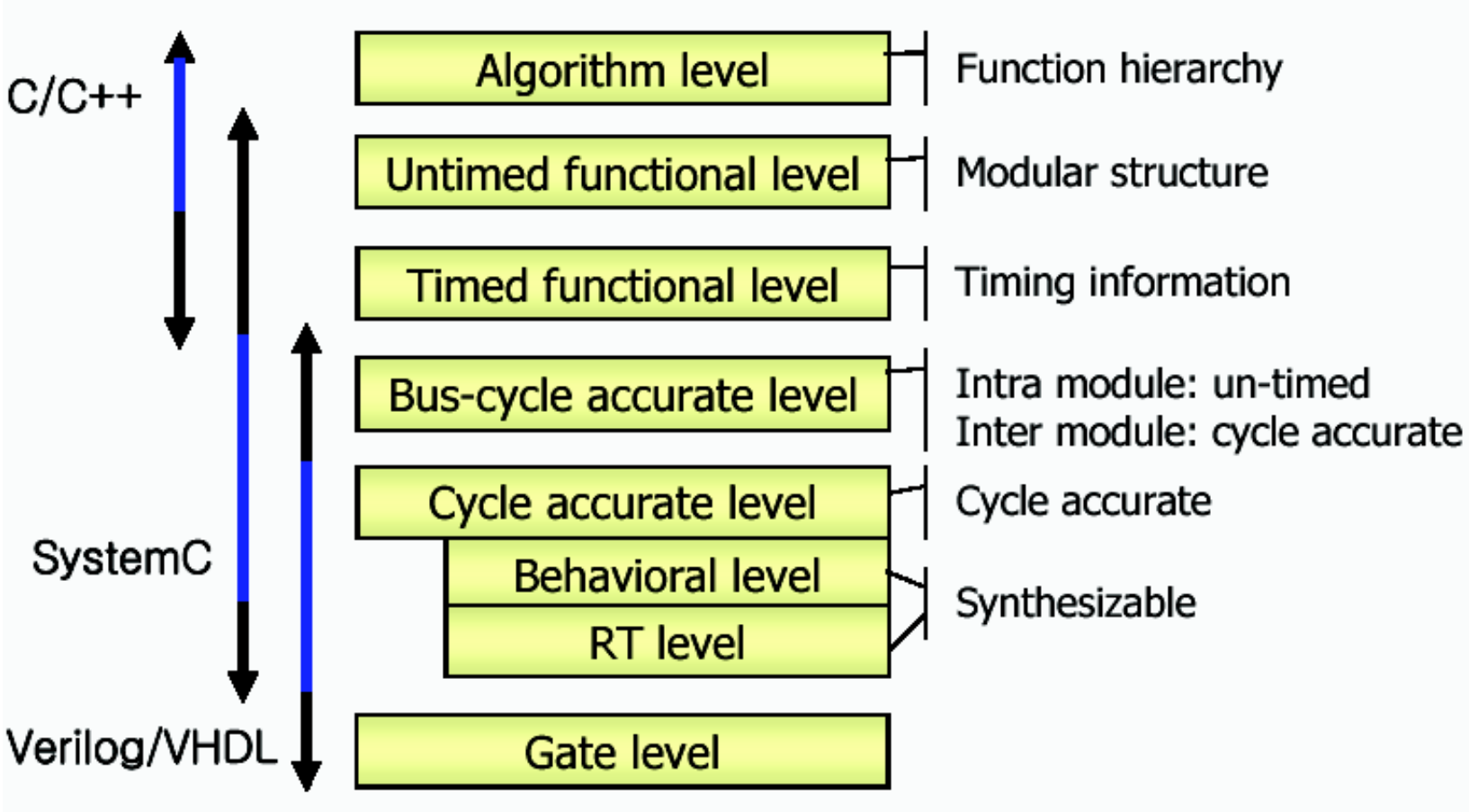
Hardware data types ← for bit vectors, 4-valued logic, fixed-point types, arbitrary precision integers

Waiting and ***watching*** ← for reactivity

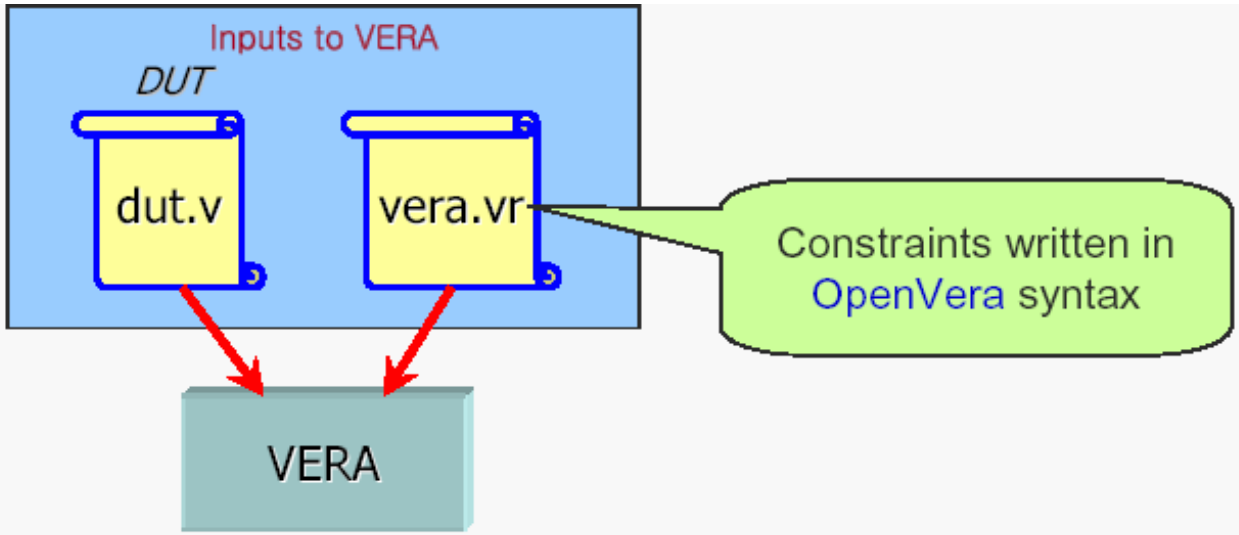
Channel, interface, and ***event*** ← for abstract communications



Abstraction Levels of SystemC



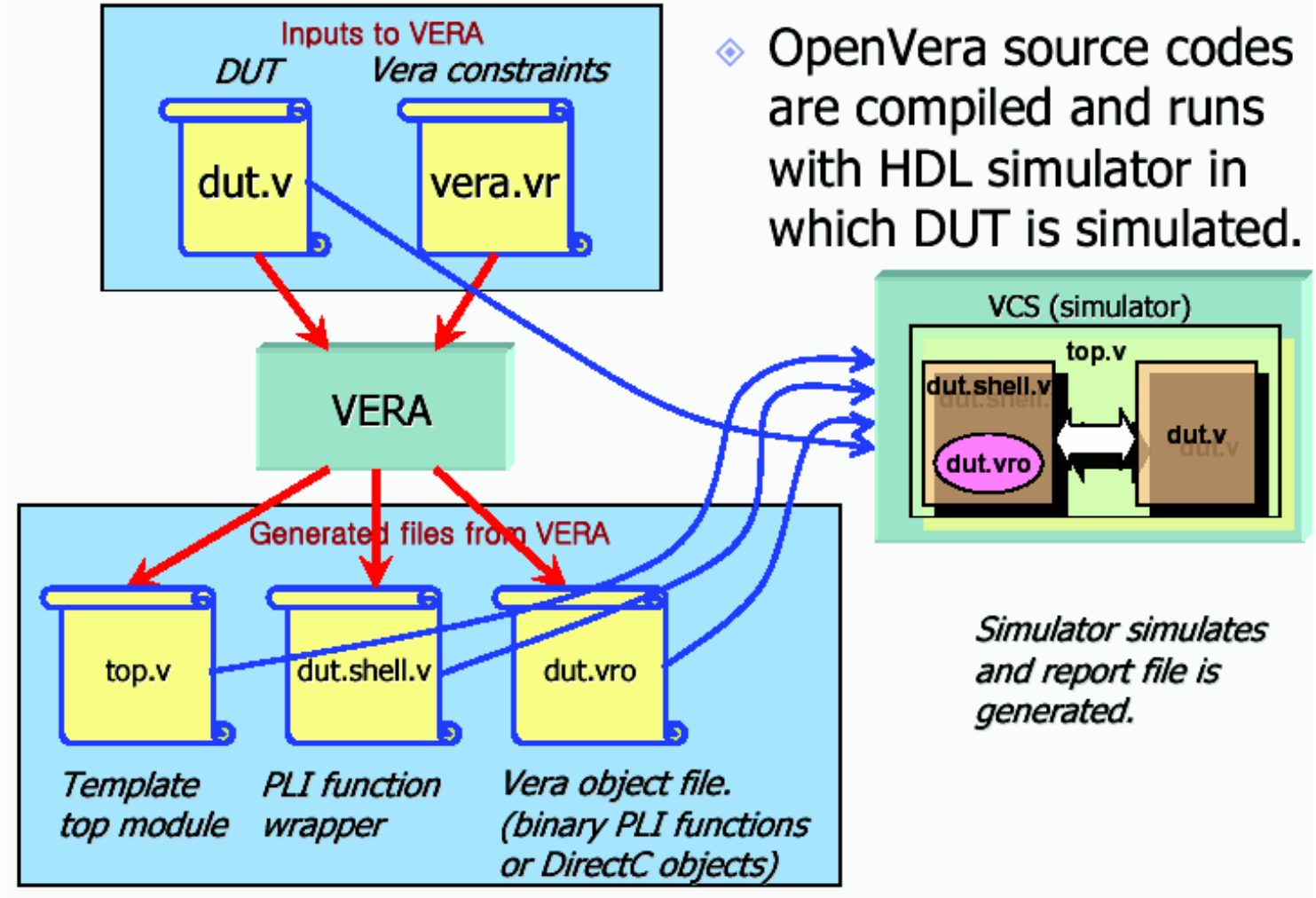
Vera (Synopsys)



Functional verification language for testbench description

- OpenVera is a language specification.
- VERA (Synopsys) is a testbench generation tool.

Vera (Synopsys)



System Verilog

SystemVerilog 3.1 provides design constructs for architectural, algorithmic and transaction-based modeling.

Adds an environment for automated testbench generation, while providing assertions to describe design functionality, including complex protocols, to drive verification using simulation or formal verification techniques.

Its C-API provides the ability to mix Verilog and C/C++ constructs without the need for PLI for direct data exchange.

System Verilog

New data types for higher data abstraction level than Verilog

- Structures, classes, lists, etc. are supported.

Assertion

- Assertions can be embedded directly in Verilog RTL.
- Sequential assertion is also supported.

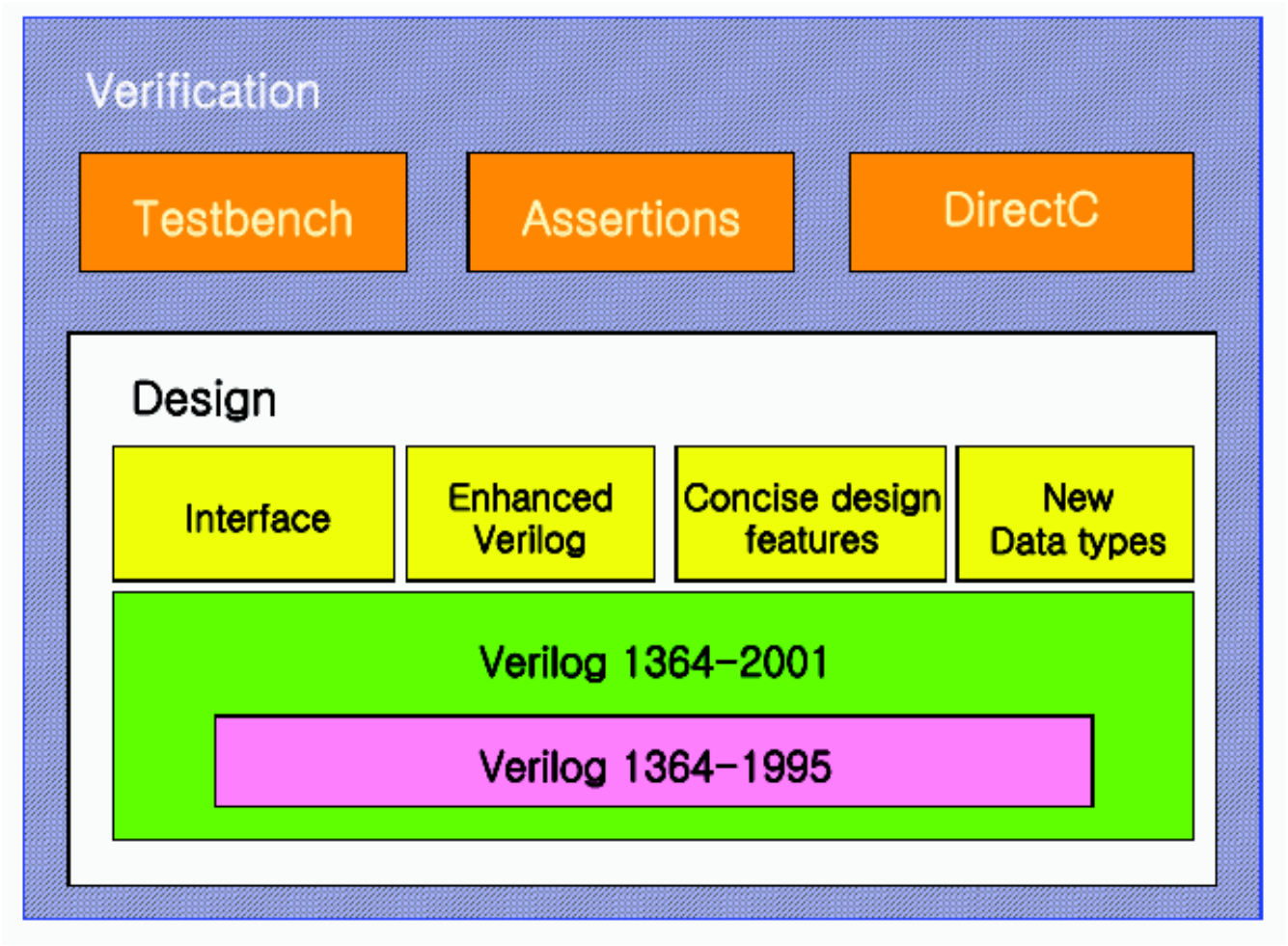
Encapsulated interfaces

- Most system bugs occur in interfaces between blocks.
- With encapsulated interfaces, the designer can concentrate on the communications rather than on the signals and wires.

DirectC as a fast C-API

- C codes can be called directly from the SystemVerilog codes.

Key Components of System Verilog



System Design Language Summary

Languge	Pros	Cons
C/C++	• Easy to write test vectors/environment	• Unable to handle some hardware environments.
HDL (Verilog, VHDL)	• Familiarity • Easy to describe H/W designs	• Focuses on the lower-level designs. • Improper for system modeling.
SystemC	• Easily connected to C/C++ codes. • Easy to model system behaviors.	• Limited tools (simulation, synthesis, etc.)
SystemVerilog	• Easy to learn for the HDL designers. • Easy to model system behaviors.	• Few tools (simulation, synthesis, etc.)

SoC Verification

Co-simulation

- Connecting ISS with HDL simulation environment
- Seamless, N2C

Co-emulation

- Emulation/rapid-prototyping equipments supporting co-emulation
- ARM Integrator, Aptix System Explorer, AXIS XoC, Dynalith iPROVE

Embedded Processor Cores in SoC

Core	Company	
ARM	ARM, Inc.	32-bit RISC microprocessor, Thumb for minimization of system cost
PPC440	IBM	Dual-issue superscalar processor
MIPS	MIPS	32/64-bit synthesizable hard cores
Xtensa	Tensilica	32-bit Xtensa architecture, Predefined reconfigurable functional units, Development of new instruction set
ZSP600	LSI Logic	6-issue superscalar DSP architecture
StarCore SC140	Motorola	16-bit fixed-point VLIW DSP core (with Agere systems)
ARCtangent	ARC, Inc.	32-bit RISC/DSP core, Reconfigurable Instruction sets for DSP/General-purpose
Pine, Oak	DSP Group	16-bit fixed-point DSP core
Jazz DSP	Improv Systems	Configurable VLIW DSP architecture

Models of Embedded Processor

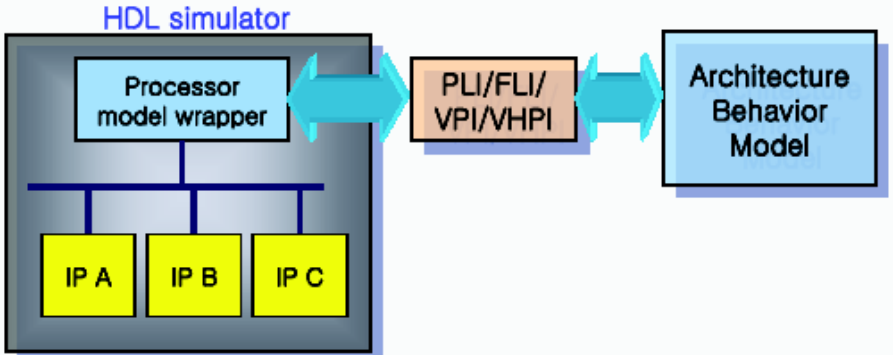
Instruction Set Simulator
Architecture Behavior Model
Processor Simulation Model
Hardware module with Real Chip

Instruction Set Simulator

- Simulates the behavior of instructions and exceptions (interrupts)
- Not cycle-accurate

Architecture Behavior Model

- Cycle-accurate behavior model usually written in C/C++ library.
- Internal registers are visible but the delays including setup and hold time are not modeled. ex) ARM's CCM
- Cosimulation of HDL simulator and architecture behavior model



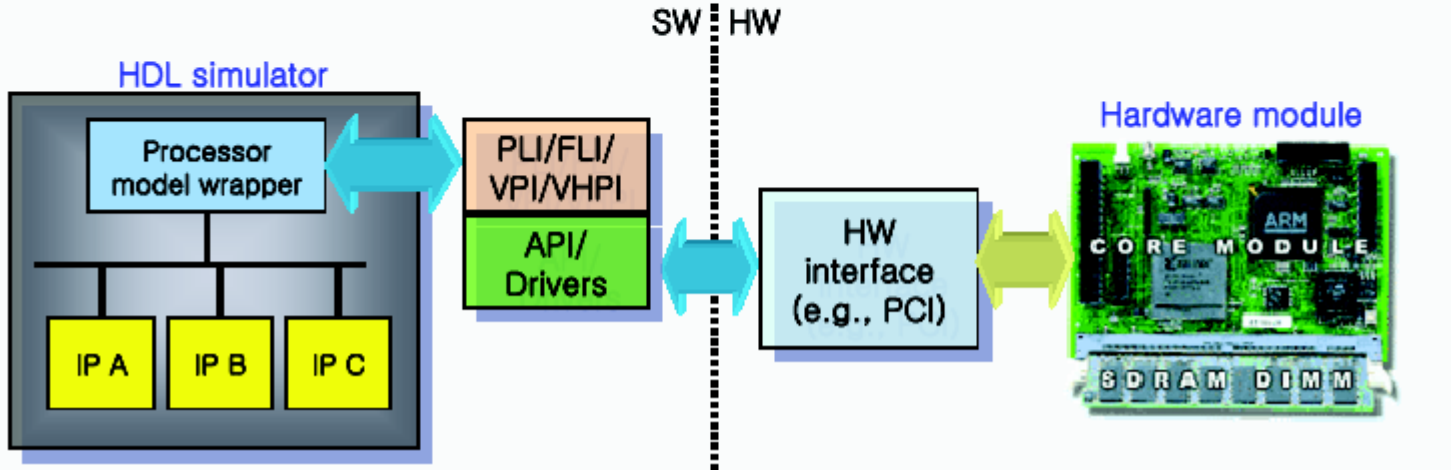
Models of Embedded Processor

Processor Simulation Model

- Behavioral model of the processor written in HDL(Verilog/VHDL)
- Fully matching the core functionality and cycle accurate
- Usually, the simulation model is composed of compiled model and timing (min/typ/max pin-to-pin delay) model.
  - Compiled model : Pre-compiled and not visible to the user
  - Timing model
    - min/typ/max pin-to-pin delay
    - setup/hold time for each pin
- Very slow : 5-500 cycles/sec.

Models of Embedded Processor

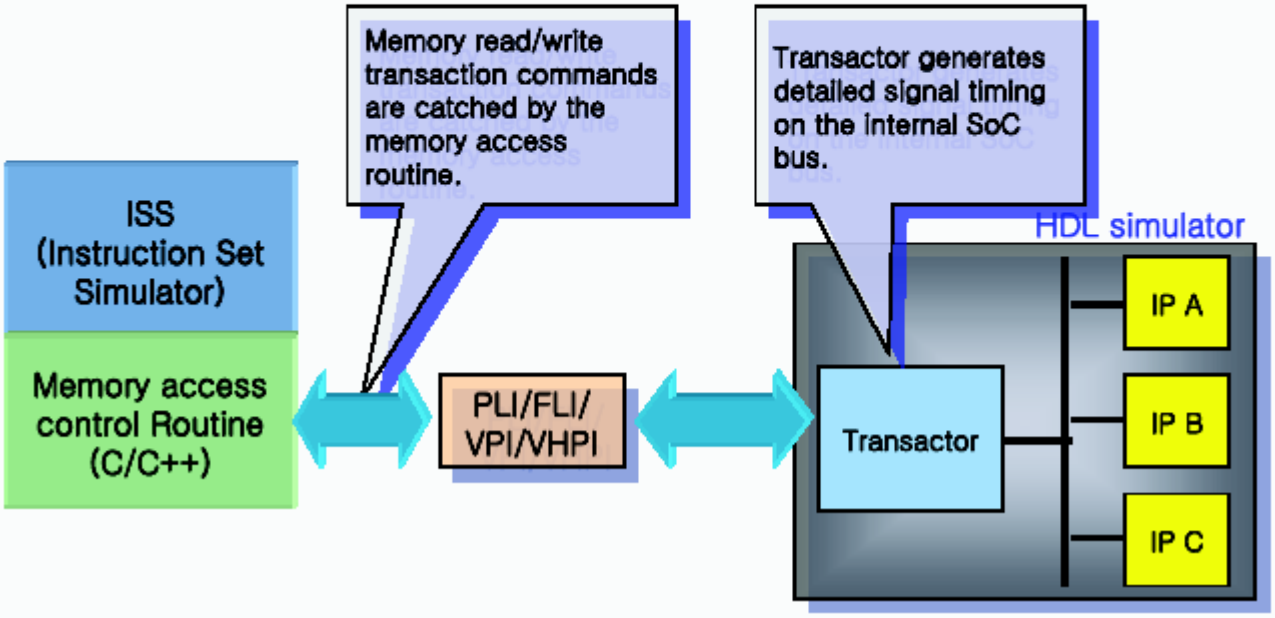
- Hardware module with Real Chip
  - Real chip includes the embedded processor core with or without peripherals
  - The hardware module is composed of the processor core chip and program/data memory.
  - Example of the coemulation with hardware module



Verification with Embedded Processor

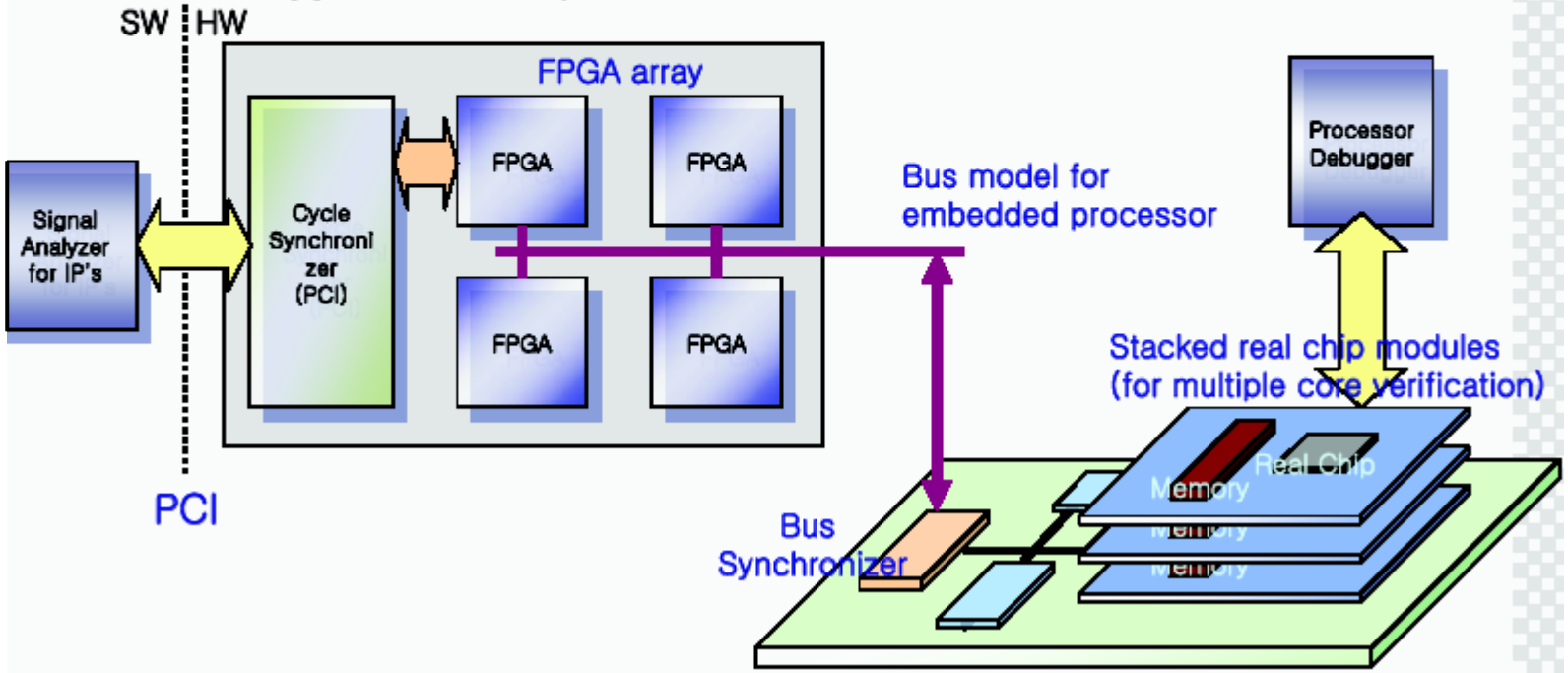
Verification with ISS

- ISS is the fastest SW model.
- The speed bottleneck is the synchronization overhead between software and HDL simulator as well as the HDL simulation.

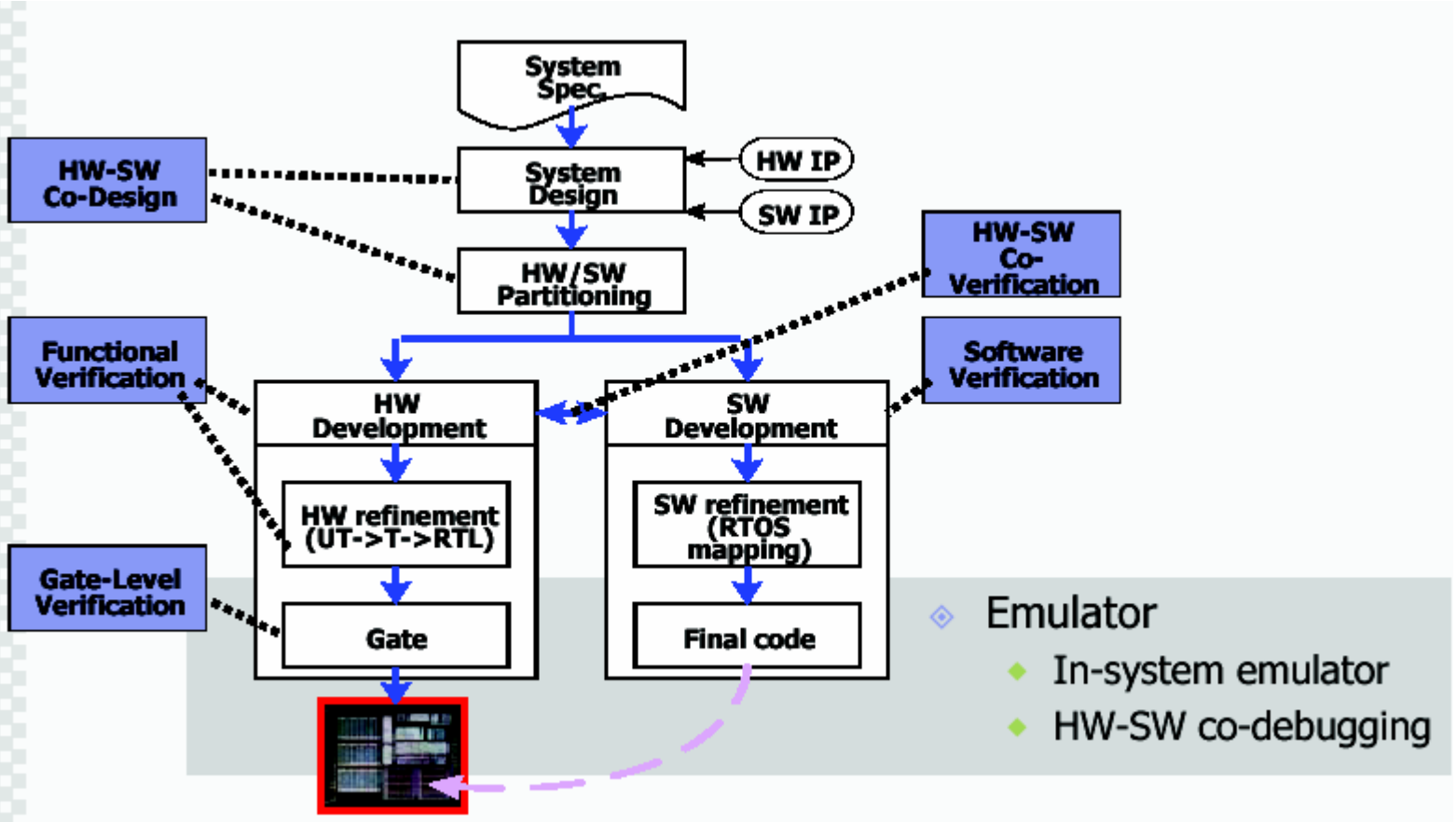


Verification with Embedded Processor

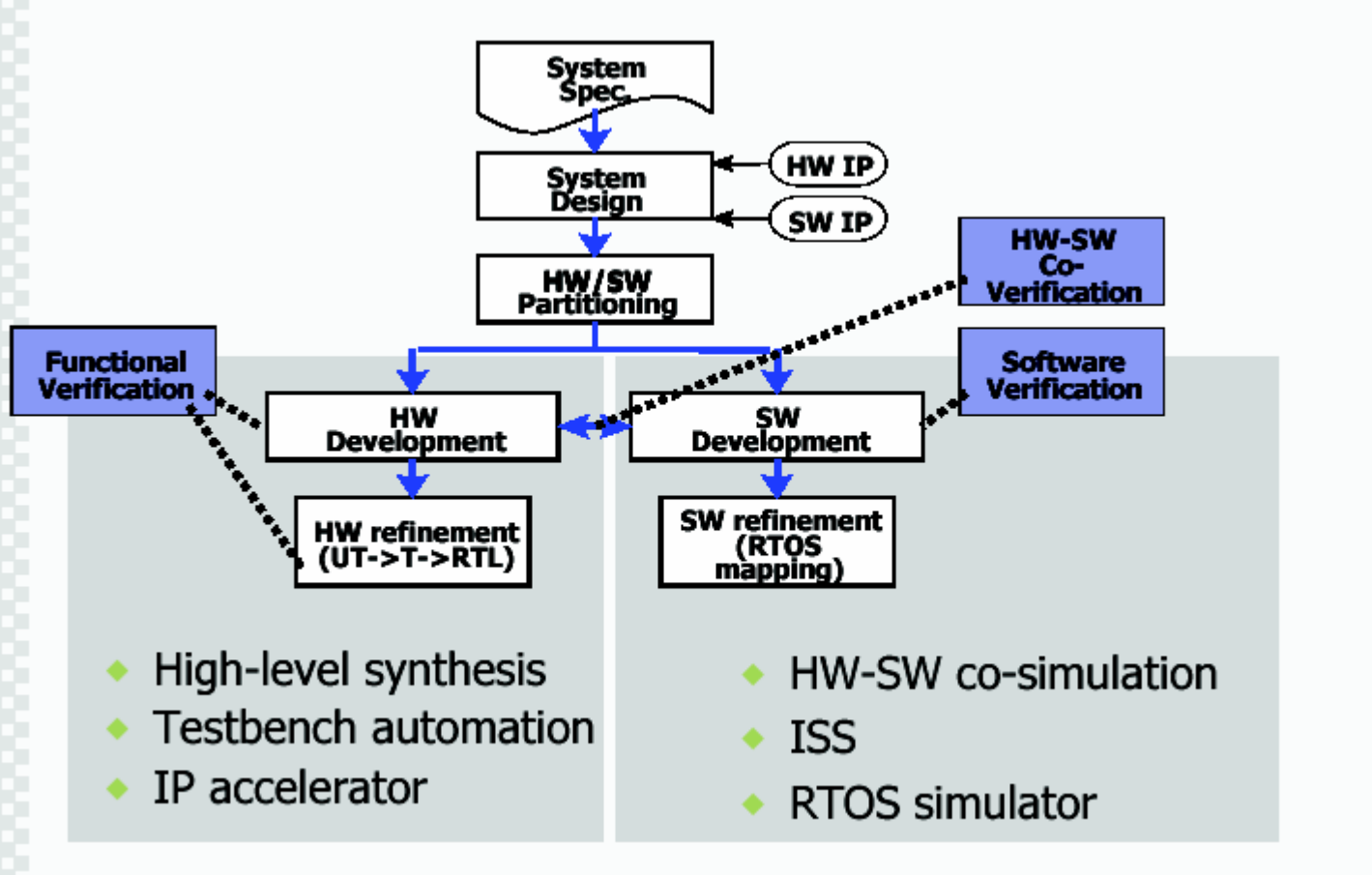
- Verification with Real Chip Modules and FPGAs
  - Stacked real chip modules are connected to FPGAs by physical bus wires.
  - Multiple FPGAs are debugged by PCI-connected signal analyzer (Waveform viewer) and processors are debugged by JTAG-based debuggers for each processor.



Simultaneous SoC design Flow

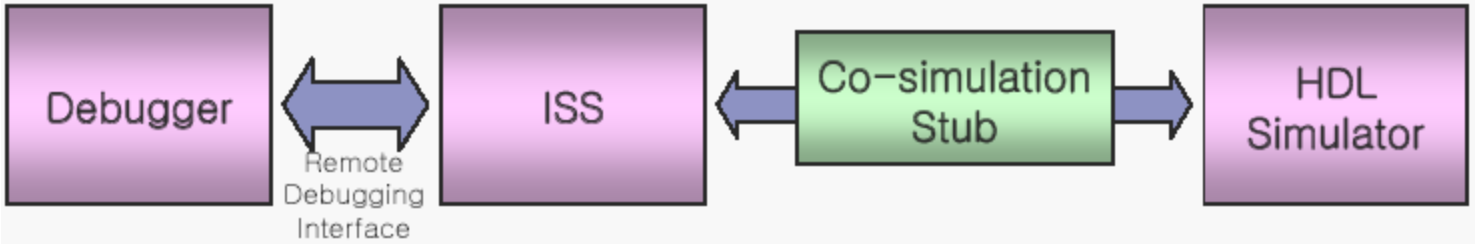


Tool utilized in HW-SW Co-Verification



Tool utilized in Co-Simulation

- Software debugging in ISS and hardware verification in HDL simulator are done in parallel way.
- Co-simulation stub manages the communication between HDL simulator and ISS.
- The most accurate solution albeit very slow
- Commercial Products
  - Eagle*i* (Synopsys), Seamless (Mentor)



Conclusion

Verification is challenging; It needs strategy!

Starategy is to apply each method when appropriate.

Verify as early as possible; Catch the bug when it is small and still isolated in a smaller region. (Don't wait until it grows and kills you.)

1st step: Apply formal methods

- Static formal verification
- Assertion-based verification

2nd step: Simulate IP with transaction level test-bench

- Test-bench automation tools

3rd step: Emulate design

- Emulate IP operation in FPGA
- In-system IP verification
- Cycle-level vs. transaction level test-bench

Conclusion

Main differences of SoC with ASIC design are

- Planned IP-reuse
- Reuse of pre-verified platform
- Focus on co-verification with software

Newly added IP's must be thoroughly verified utilizing automated testbench and formal methods, if possible.

Well-established emulation platform helps,

- Progressive refinement of newly added SoC components
- Early development and verification of software

Powerful debugging features handling both hardware part and software part are required.

Language, Tool/Data Interfaces need standardization.

DFV (Design for Verification) ; You lose in the beginning, but will win later, like Design for Reuse.