Introduction to SOC Verification

Prof. Chien-Nan Liu
Institute of Electronics
National Chiao-Tung Univ.

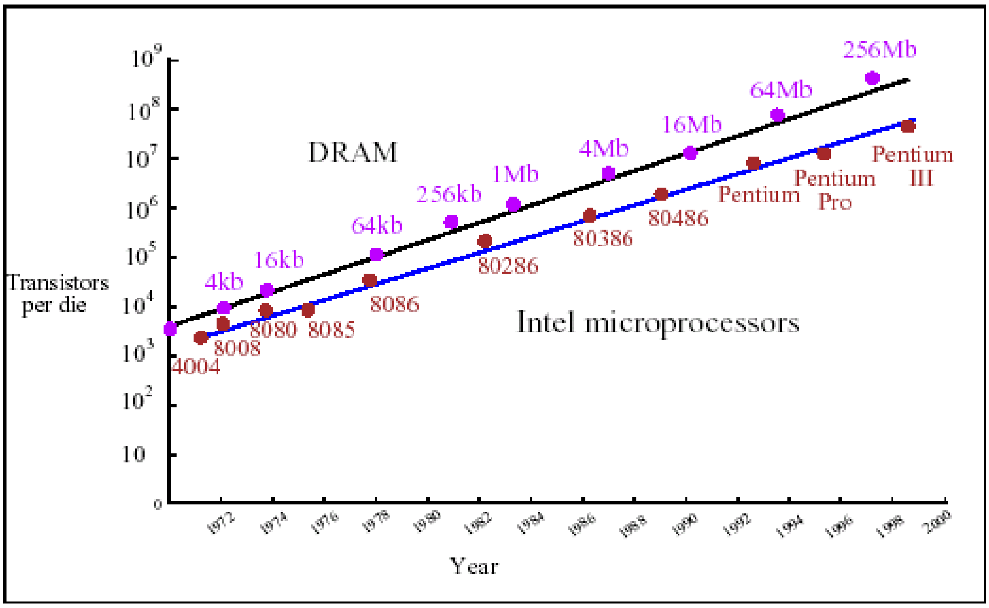
教育部
前瞻晶片系統設計人才培育先導型計畫
DAT 學程核心課程精進計畫
DAT 聯盟

Outline

- Moving to SOC
- What is Verification?
- Functional Verification Solutions
- System Verification Solutions

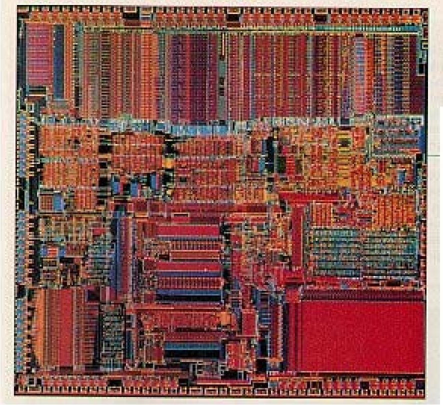
Moore's Law

- Logic capacity doubles per IC per year at regular intervals (1965)
- Logic capacity doubles per IC every 18 months (1975)

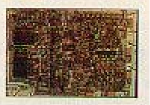


The Dies of Intel CPUs

Pentium Pro

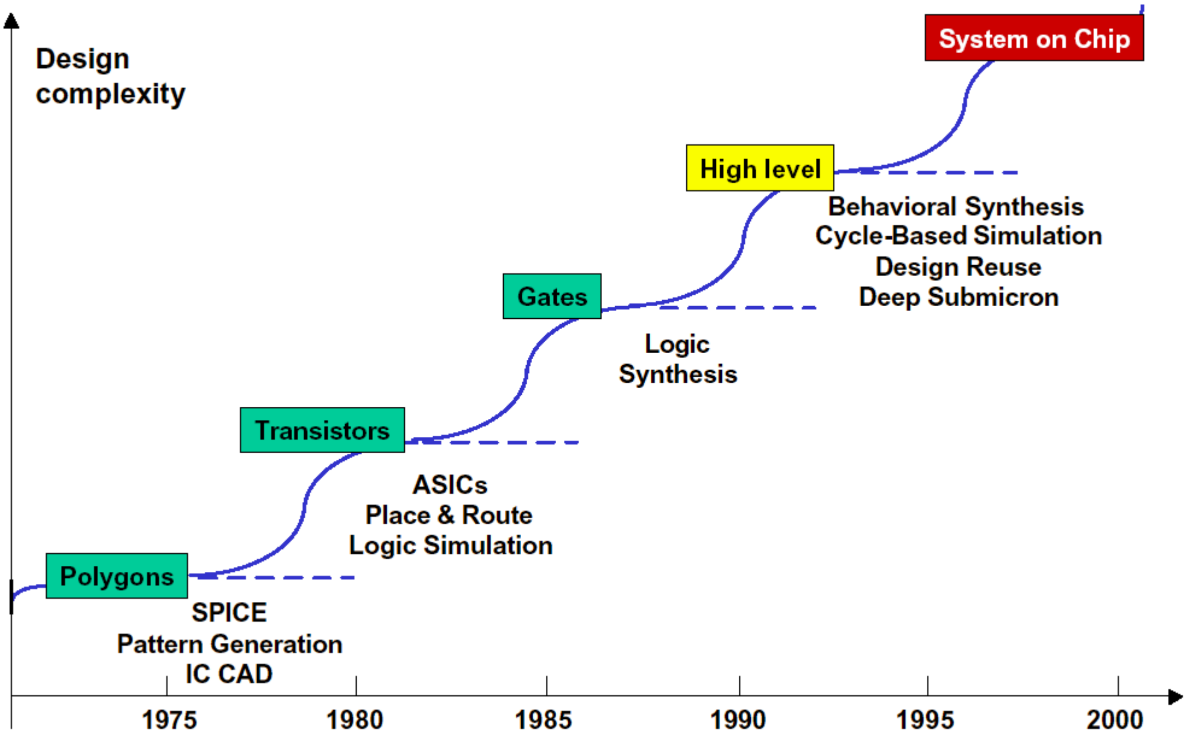


386



4004

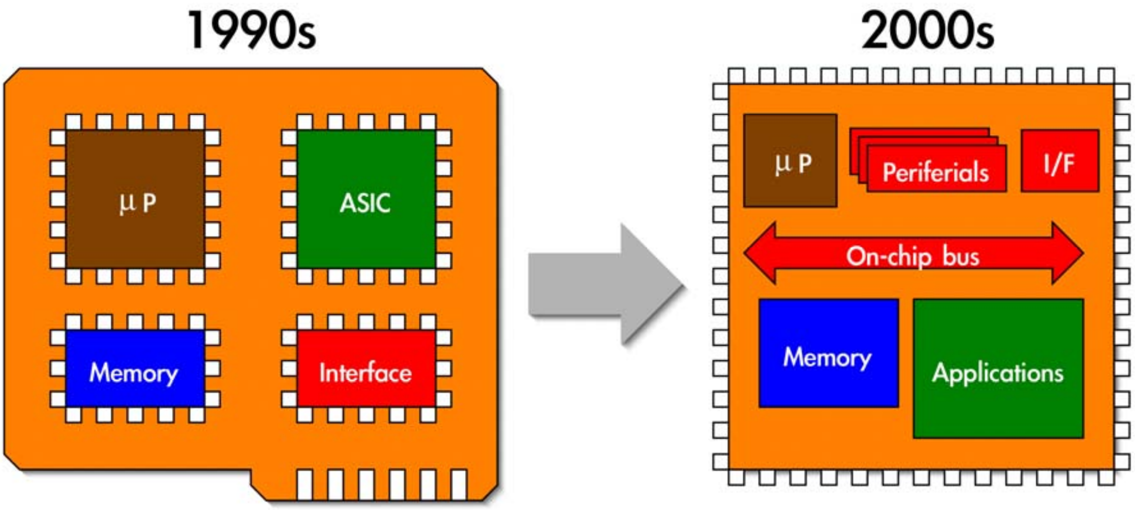
Trends of VLSI Design



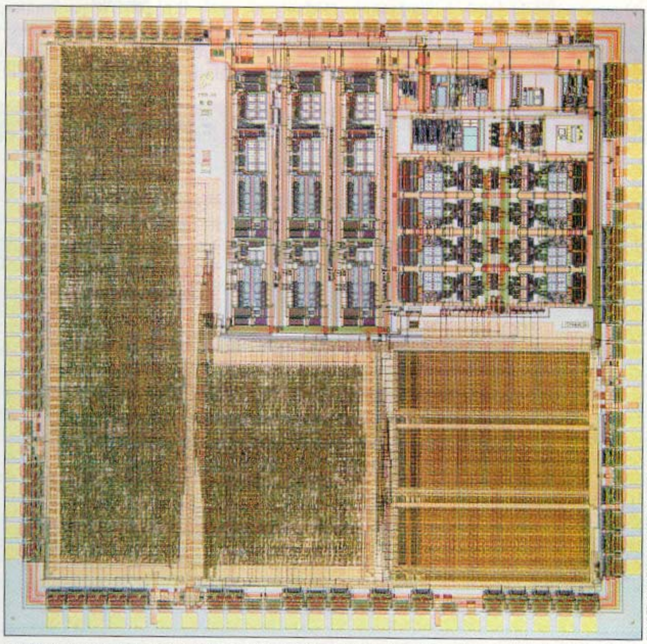
Reference：www.doulos.com

What is SoC?

- System-on-Chip
- An IC that integrates the major functional elements of a complete end-product into a single chip



Scanner on a Chip



SOURCE: NATIONAL SEMICONDUCTOR CORP.

National Semiconductor
IEEE Spectrum
Jan. 1999, p.57

[1] Merlin, the scanner-on-a-chip just introduced by National Semiconductor Corp., Santa Clara, Calif., includes an analog front end, sensor clock generation, motor control, data buffering, and parallel port interface, all in a dime-sized area. Chips like this one, which load up with components in much the same way as boards load up with chips, are being dragged into existence by the demands of consumer electronics.

An SoC …

- Usually contains
  - Reusable IP
  - Embedded processor, memory
  - Real-world interface
  - Mixed-signal blocks
  - Programmable hardware
  - RTOS and embedded software
- Has more than 500 K gates,
- Use .25 µm technology or below
- Is not an ASIC …

Why SoC ? (from System View)

- Lower power than discrete implementations
- Increased functionality/performance in reduced footprint
- Simplified PCB design
- Lower system cost
- Increased product mechanical robustness
- And more and more advantages ……

SoC Challenges

- Increasing complexity
  - Time-to-market pressure
  - Verification bottleneck
- Integration
  - Hardware v.s. software
  - Digital circuits v.s. analog circuits
  - Testing issues
- Deep submicron effects
  - Timing closure problem
  - Signal integrity problem
  - Reliability problem

Time-to-Market Pressure

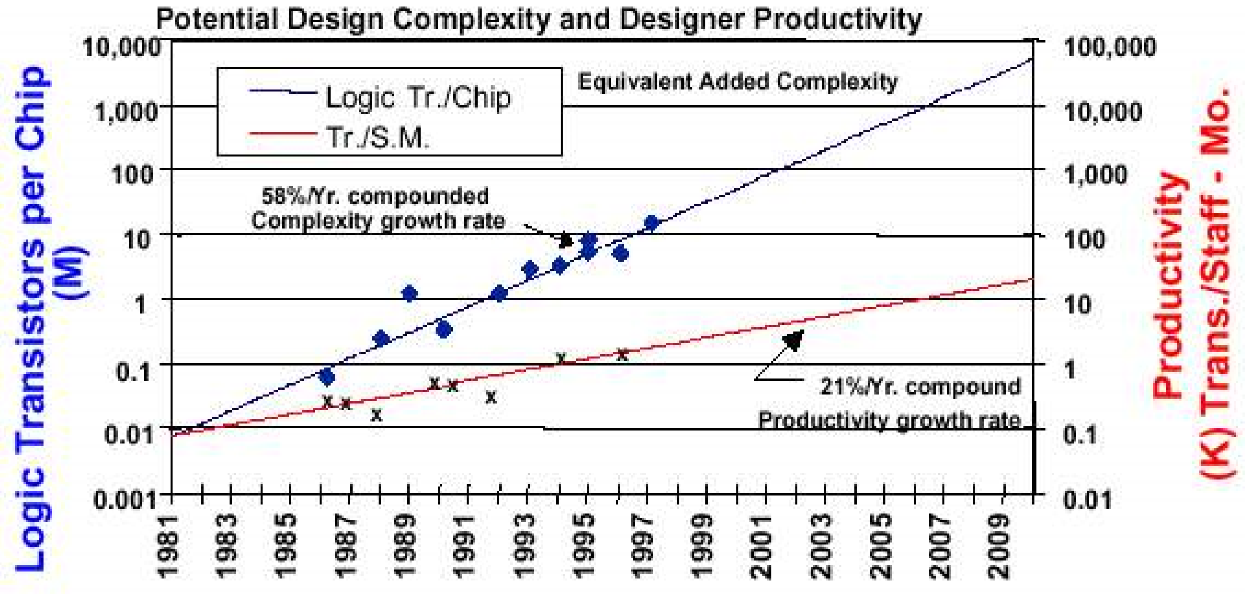
- A lot of pressure from
  - Shorter product lifespan
  - Shrinking design cycles

	1997	1998	1999	2002
Applications	Cellar, PDA, DVD	Set-top boxes, wireless PDA	Internet applications, anything portable	Ubiquitous computing, intelligent, interconnected controllers
Design cycle (month)	18 - 12	12 - 10	10 - 8	8 - 6
Derivative cycle (month)	8 - 6	6 - 4	4 - 2	3 - 2

* Adapted from "Surviving the SOC Revolution."

Productivity Gap

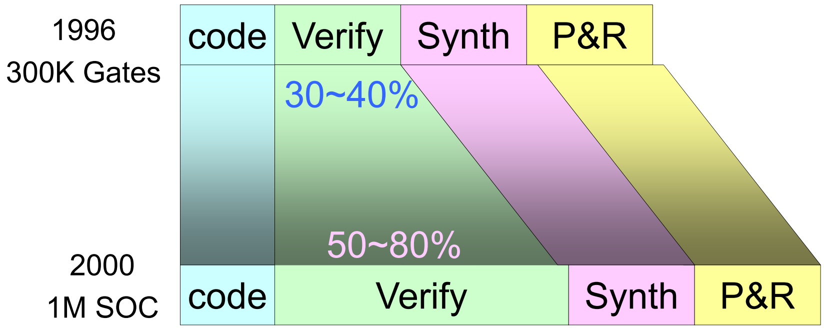
- We do need more efficient design methodology



Source : ITRS Roadmap 1999 Edition, SIA.

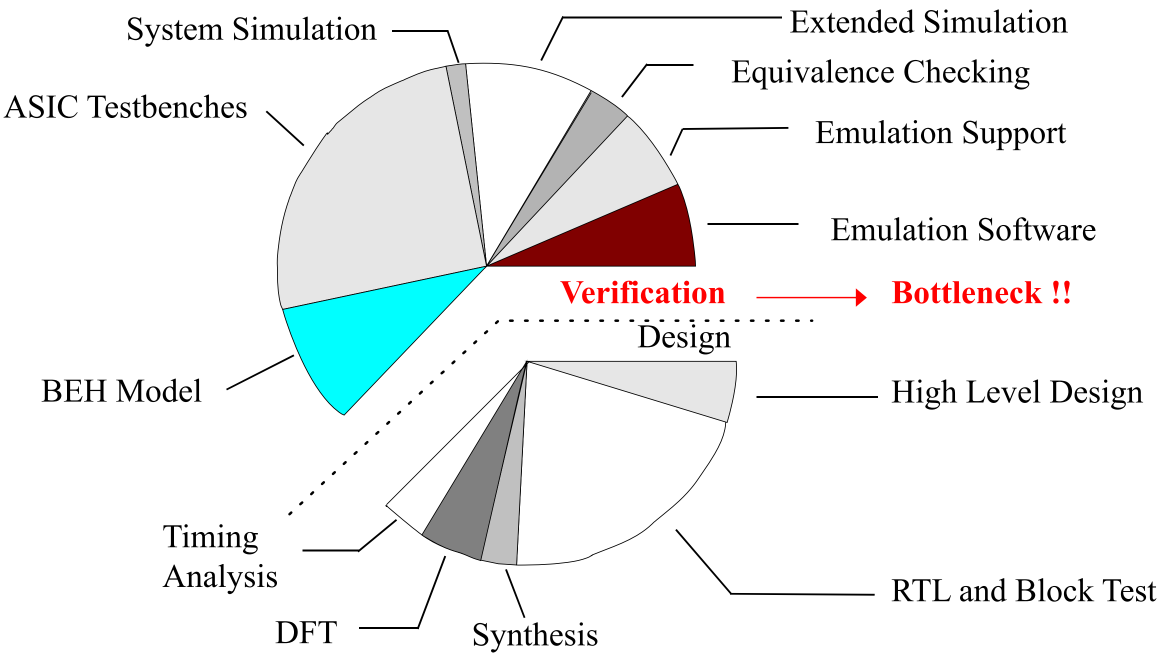
Verification Bottleneck

- Verification becomes the major bottleneck of the modern design flows
  - From 30%-40% to 50%-80%



- An effective verification methodology is also highly desirable

An Industrial Example



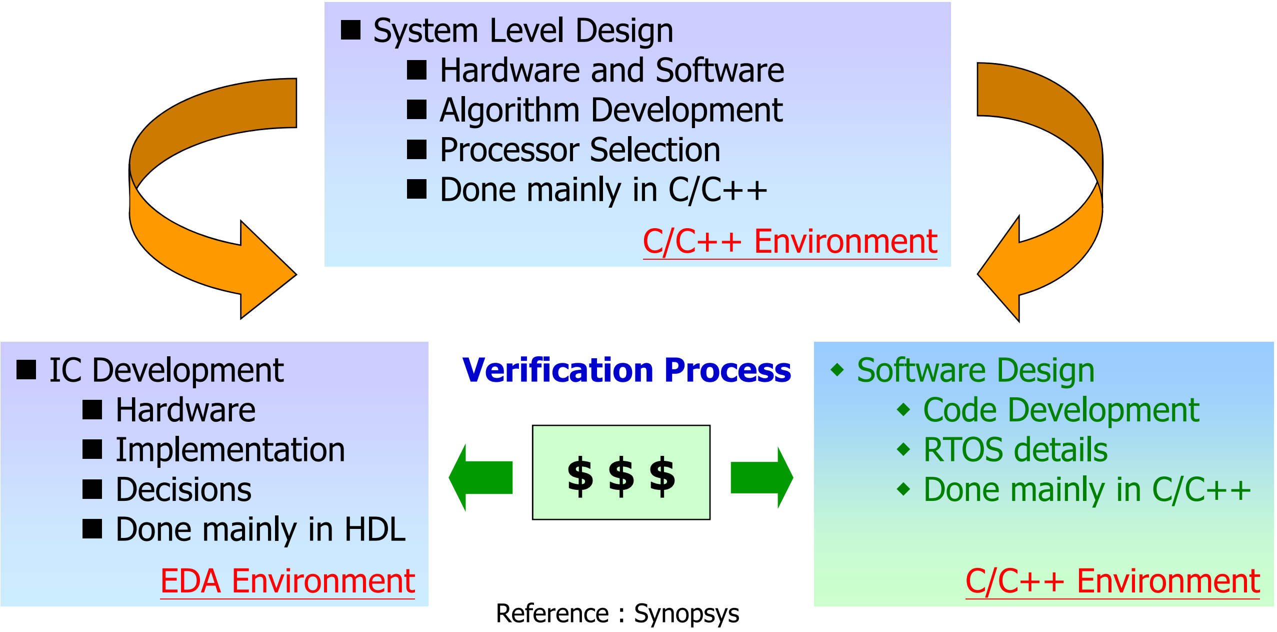
Source : "Functional Verification on Large ASICs"
by Adrian Evans, etc., 35th DAC, June 1998.

SoC Challenges

- Increasing complexity
  - Time-to-market pressure
  - Verification bottleneck
- Integration
  - Hardware v.s. software
  - Digital circuits v.s. analog circuits
  - Testing issues
- Deep submicron effects
  - Timing closure problem
  - Signal integrity problem
  - Reliability problem

HW/SW Integration

- Integrating HW/SW at the final step may require high cost to fix inconsistency problems



- System Level Design
  - Hardware and Software
  - Algorithm Development
  - Processor Selection
  - Done mainly in C/C++

C/C++ Environment

- IC Development
  - Hardware
  - Implementation
  - Decisions
  - Done mainly in HDL

EDA Environment

Verification Process

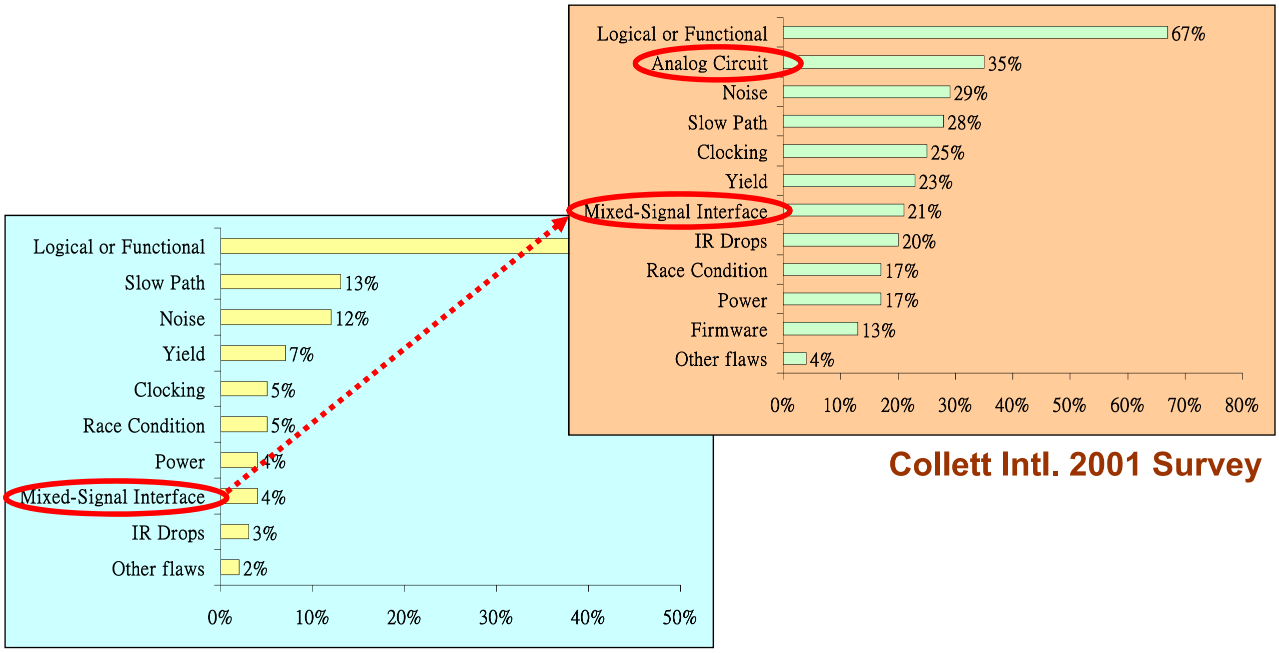
$ $ $

- Software Design
  - Code Development
  - RTOS details
  - Done mainly in C/C++

C/C++ Environment

Reference : Synopsys

Mixed Signals in SoC



Collett Intl. 2001 Survey

Collett Intl. 2000 Survey

Challenges for MS Designs

- Design challenges
  - Chip-level simulation takes too much time
  - Design budgets are not distributed in a well-defined manner
  - Too much time is spent on low-level iterations
  - Design is not completely systematic
  - There is limited or no use of HDL
- Solutions:
  - Use a systematic, top-down design approach to capture design intent
  - Develop some tools for rapid targeting of different design technologies

SoC Testing Challenges

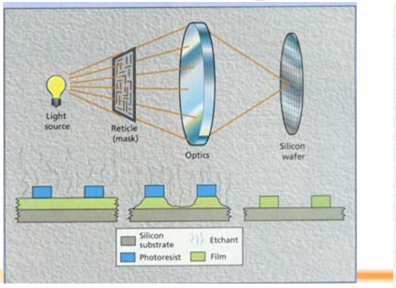
- Distributed design and test
  - Core provider does not know the target environment
  - System integrator is responsible for manufacturing testing
- Test access
  - Difficulties to access deeply embedded cores
  - Bandwidth, I/O pin count limitations
- Test optimization
  - Minimizing test cost while satisfying constraints such as power, resources, coverage, etc.

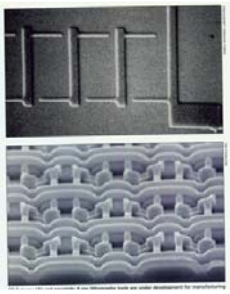
SoC Challenges

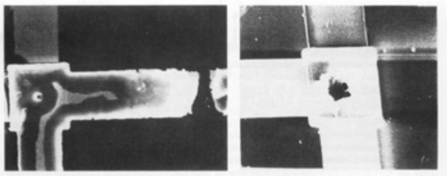
- Increasing complexity
  - Time-to-market pressure
  - Verification bottleneck
- Integration
  - Hardware v.s. software
  - Digital circuits v.s. analog circuits
  - Testing issues
- Deep submicron effects
  - Timing closure problem
  - Signal integrity problem
  - Reliability problem

Nanometer Design Challenges

- In 2005, feature size ≈ 0.1 μm, μ P frequency ≈ 3.5 GHz, die size ≈ 520 mm^2, μ P transistor count per chip ≈ 200M, wiring level ≈ 8 layers, supply voltage ≈ 1 V, power consumption ≈ 160 W.
  - **Feature size ↓ → sub-wavelength lithography (impacts of process variation)? noise? wire coupling? reliability?**
  - **Frequency ↑, dimension ↑ → interconnect delay? electromagnetic field effects? timing closure?**
  - **Chip complexity ↑ → large-scale system design methodology?**
  - **Supply voltage ↓ → signal integrity (noise, IR drop, etc)?**
  - **Wiring level ↑ → manufacturability? 3D layout?**
  - **Power consumption ↑ → power & thermal issues?**





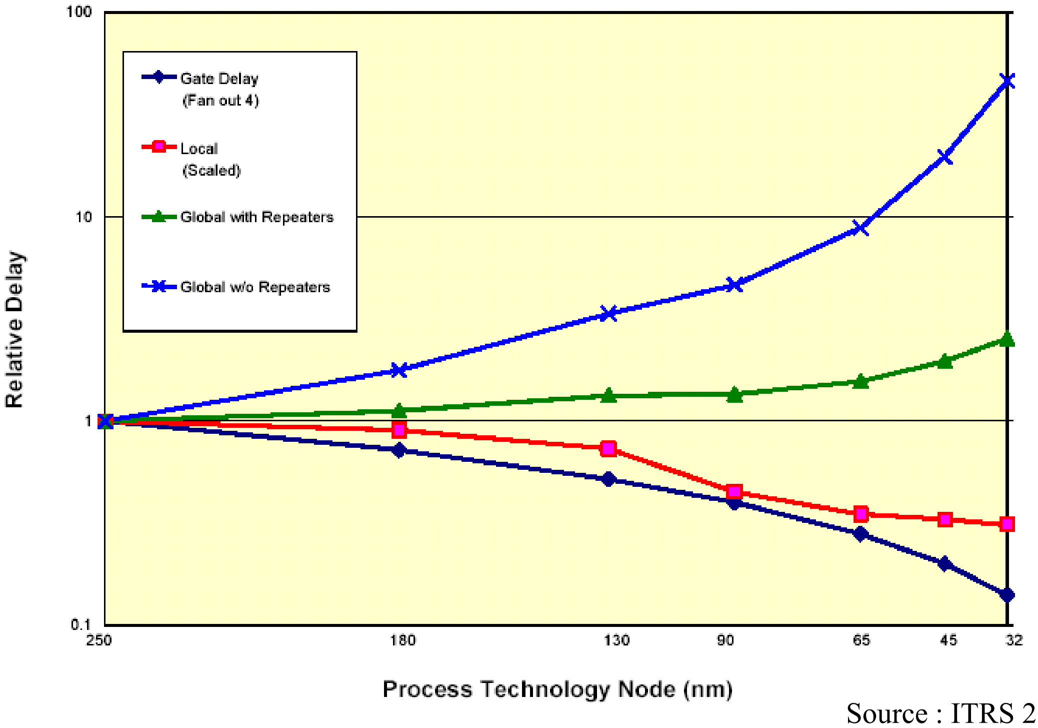


(a) Line-open failure.

(b) Open failure in contact plug.

Figure 8.30 Electromigration-related failure modes (*Courtesy of N. Cheung and A. Tao, U.C. Berkeley*).

Wiring Delay vs. Feature Size



Source : ITRS 2001

Timing Closure Problem

- Wire delay starts to dominate total delay in DSM process
  - Cannot be ignored as taught in digital design course
- Only statistical wire delay model can be used at design phase
  - Lack of physical information about wire length
- Incorrect estimations require long iterations to meeting timing
  - Design schedule will be seriously delayed !!

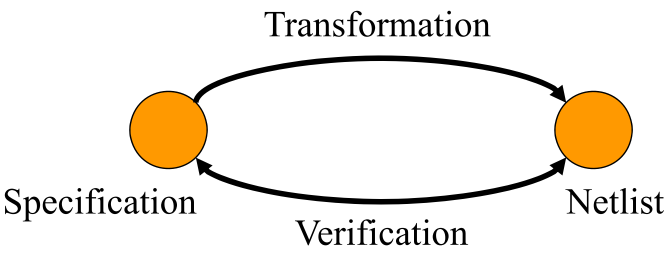
Path name	Pre-layout delay	Post-layout delay
P1	21.72	40.92 (+88.4%)
P2	6.65	7.81 (+17.4%)
P3	11.14	10.43 (-6.4%)
P4	5.03	5.44 (+8.15%)
P5	6.35	13.21 (+108.0%)
P6	6.42	13.20 (+105.6%)

Outline

- Moving to SOC
- What is Verification?
- Functional Verification Solutions
- System Verification Solutions

What is Verification ?

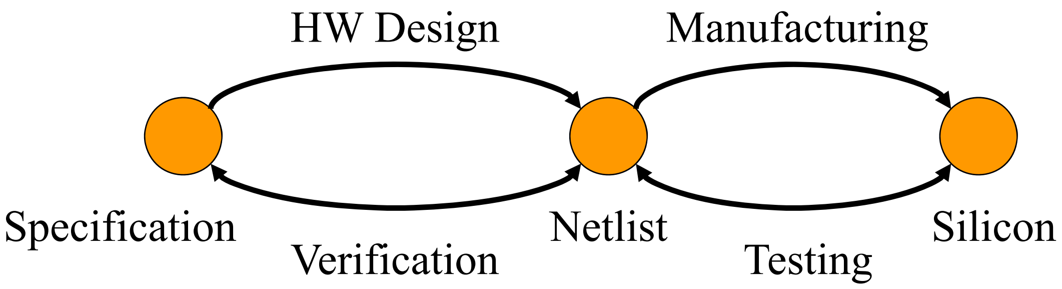
- A process used to demonstrate the functional correctness of a design
- To making sure that you are indeed implementing what you want
- To ensure that the result of some transformation is as expected



Source : “Writing Test Benches – Functional Verification of HDL Models” by Janick Bergeron, KAP, 2000.

Testing v.s. Verification

- Testing verifies manufacturing
  - Verify that the design was manufactured correctly



Source : “Writing Test Benches – Functional Verification of HDL Models” by Janick Bergeron, KAP, 2000.

Verification Complexity

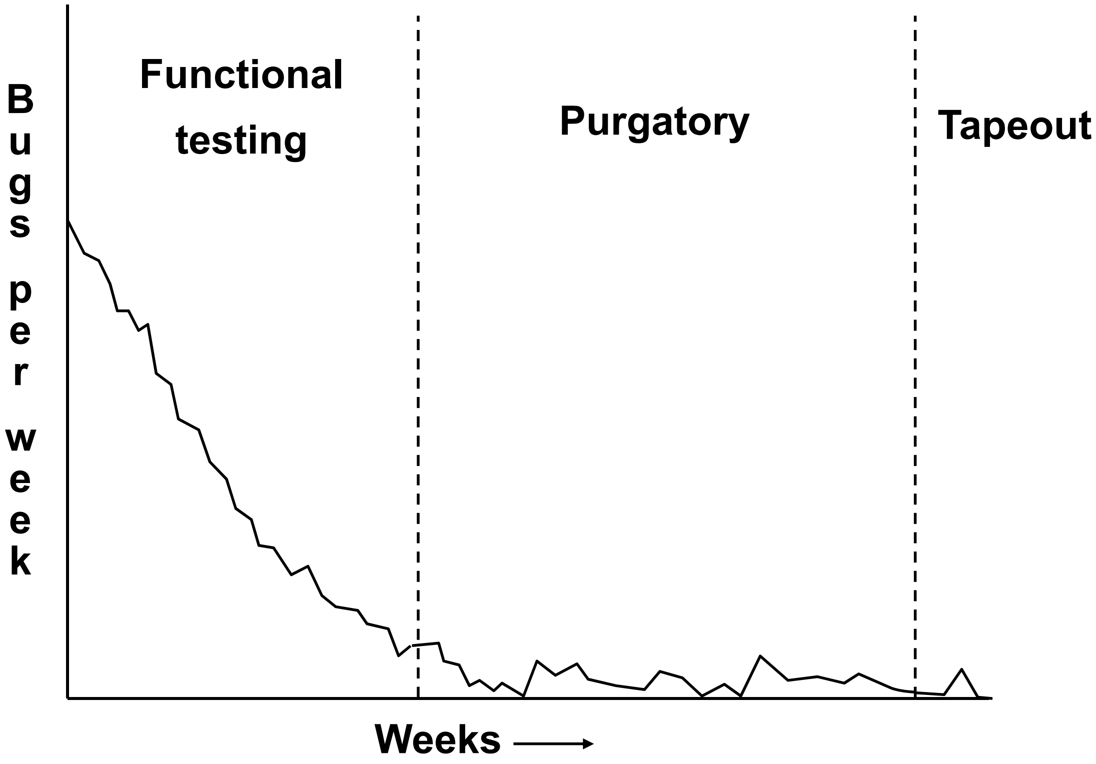
- For a single flip-flop:
  - Number of states = 2
  - Number of test patterns required = 4
- For a Z80 microprocessor (~5K gates)
  - Has 208 register bits and 13 primary inputs
  - Possible state transitions = $2^{\text{bits+inputs}} = 2^{221}$
  - At 1M IPS would take 10^{53} years to simulate all transitions
- For a chip with 20M gates
  - ??????

*IPS = Instruction Per Second

When is Verification Complete ?

- Some answers from real designers:
  - When we run out of time or money
  - When we need to ship the product
  - When we have exercised each line of the HDL code
  - When we have tested for a week and not found a new bug
  - ***We have no idea!!***
- Designs are often too complex to ensure full functional coverage
  - The number of possible vectors greatly exceeds the time available for test

Typical Verification Experience



Verification Approaches

- Top-down verification approach
  - From system to individual components
- Bottom-up verification approach
  - From individual components to system
- Platform-based verification approach
  - Verify the developed IPs in an existing platform
- System interface-based verification approach
  - Model each block at the interface level
  - Suitable for final integration verification

Bottom-Up Verification Steps

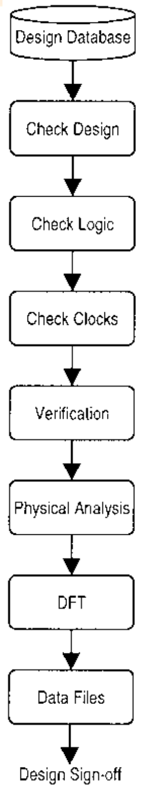
- Verify the leaf IPs
  - Many techniques are proposed and will be discussed later
- Verify the interface among IPs
- Run a set of complex applications
- Prototype the full chip and run the application software
- Decide when to release for mass production

Interesting Observations

- 90% of ASICs work at the first silicon but only 50% work in the target system
  - Do not perform system level verification (with software)
- If a SoC design consisting of 10 blocks
  - P(work)= $.9^{10}$ =.35
- If a SoC design consisting of 2 new blocks and 8 pre-verified robust blocks
  - P(work) = $.9^{2}$ * $.98^{8}$ =.69
- To achieve 90% of first-silicon success SoC
  - P(work) = $.99^{10}$ =.90

Design Sign-off

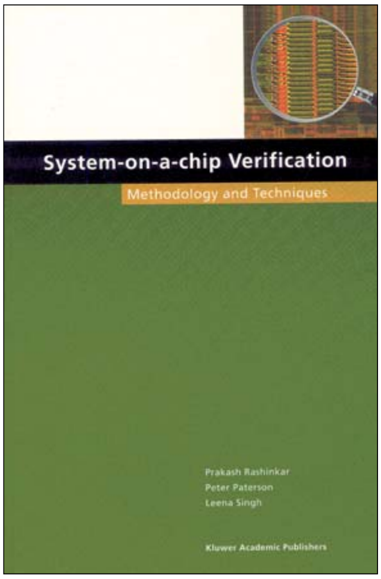
- Sign-off is the final step in the design process
- It determines whether the design is ready to be taped out for fabrication
- No corrections can be made after this step
- The design team needs to be confident that the design is 100% correct
  - Many items need to be checked



Source : "System-on-a-chip Verification – Methodology and Techniques" by P. Rashinkar, etc., KAP, 2001.

A Reference Book

- **System-on-a-Chip Verification Methodology and Techniques**
- by
  Prakash Rashinkar
  Peter Paterson
  Leena Singh
  Cadence Design Systems Inc., USA
- published by
  Kluwer Academic Publishers, 2001


System-on-a-chip Verification
Methodology and Techniques
Prakash Rashinkar
Peter Paterson
Leena Singh
Kluwer Academic Publishers

Outline

- Moving to SOC
- What is Verification?
- Functional Verification Solutions
  - Simulation-Based Techniques
  - Static Verification Techniques
- System Verification Solutions

Simulation-Based Verification

- Still the primary approach for functional verification
  - In both gate-level and register-transfer level (RTL)
- Test cases
  - User-provided (often)
  - Randomly generated
- Hard to gauge how well a design has been tested
  - Often results in a huge test bench to test large designs
- Near-term improvements
  - Faster simulators
    - Compiled code, cycle-based, emulation, …
  - Testbench tools
    - Make the generation of pseudo-random patterns better/easier
- Incremental improvements won’t be enough

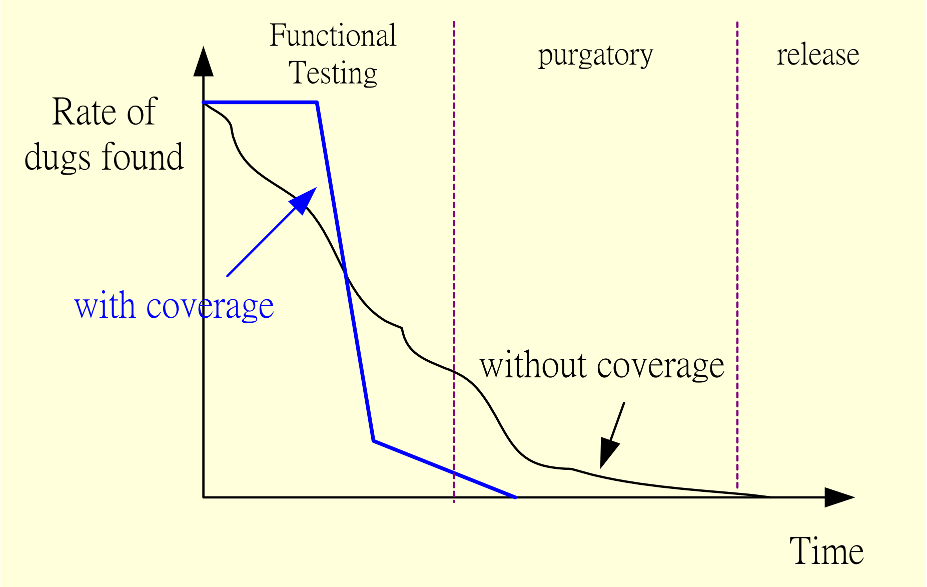
Simulator Improvements

- Event-driven:
  - Timing accurate but slower
- Cycle-based:
  - Faster simulation (5x – 100x) but only cycle accurate
  - Require other tools to check timing problems
- Transaction-based:
  - Require bus functional model (BFM) of each design
  - Only check the transactions between components
  - Have faster speed by raising the level of abstraction
- Assistant hardware:
  - To speed up logic simulation by mapping some gate-level netlist into specific hardware

Coverage-Driven Verification

- Coverage reports can indicate how much of the design has been exercised
  - Point out what areas need additional verification
- Optimize regression suite runs
  - Redundancy removal (to minimize the test suites)
  - Minimizes the use of simulation resources
- Quantitative sign-off criterion
- Easy to use, low-complexity
- A good guidance but cannot guarantee 100% error-free designs

The Rate of Bug Detection



source : "Verification Methodology Manual For Code Coverage In HDL Designs" by Dempster and Stuart

Testbench Automation

- Require both ***generator*** and ***predictor*** in an integrated environment
- Generator: constrained random patterns
  - Ex: keep A in [10 … 100]; keep A + B == 120;
  - Hardware verification languages (HVL) are often used
- Predictor: generate the estimated outputs
  - Require a behavioral model of the system
- Only pseudo-random patterns are available
  - Good to check corner cases
  - Relationship to real functionality is week
- Many improvements are made on this approach
  - Popular in industry now

Assertion-Based Verification

- Assertions are ***active*** comments that can
  - Monitor signals on interfaces when connecting blocks
  - Track expected behavior documented in the assertions
  - Watch for forbidden behavior with a design block
- Assertions turn design specification into verification objects
  - Design can be continuously checked against these spec. throughout its life cycle
- Assertion increases observability in simulation
  - Debugging time is greatly reduced
- A new approach that is worthy to be watched

Outline

- Moving to SOC
- What is Verification?
- Functional Verification Solutions
  - Simulation-Based Techniques
  - Static Verification Techniques
- System Verification Solutions

HDL Linter

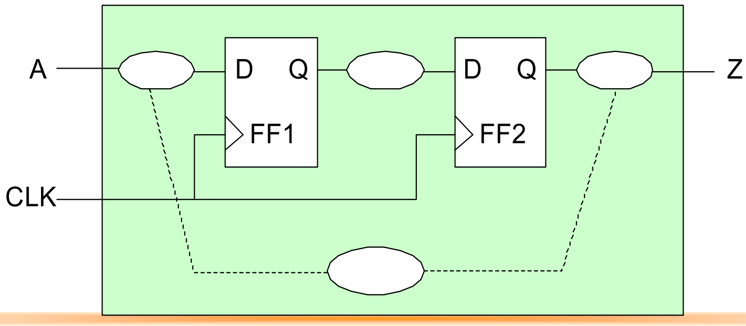
- Fast static RTL code checker
  - Preprocessor of the synthesizer
- **RTL purification (RTL DRC)**
  - Syntax, semantics, simulation
- Check for built-in or user-specified rules
  - Testability, reusability, …
- Lint-liked tools can help spot defects without simulation
  - Avoid error code that increases design iterations
- Only trivial bugs can be found
  - Depend on the completeness of the database

Formal Verification

- Ensure the consistency with specification for *all* possible inputs (100% coverage)
- Primary applications
  - Equivalence Checking
  - Model Checking
- Solve the completeness problem in simulation-based methods
- Cannot handle large designs due to its high complexity
- Valuable, but not a general solution

Static Timing Analysis (STA)

- Determine if a circuit meets timing constraints without having to simulate
  - No input patterns are required (100% coverage)
  - Functionality is not checked
- Faster but may be inaccurate (too pessimistic)
- Cannot be applied to all cases
  - Gate-level simulation may still required for some cases



Outline

- Moving to SOC
- What is Verification?
- Functional Verification Solutions
- System Verification Solutions
  - Hardware v.s. software
  - Digital circuits v.s. analog circuits

Prototyping

- Verify designs using real hardware
  - FPGA, emulator
- Better throughput in handling complex designs
- **Software-driven verification**
  - Verify HW using SW
- Interfaced with real HW components
- May have a little performance degradation
- May have capacity limitation
- Poor debugging capability
- High cost to fix design problems

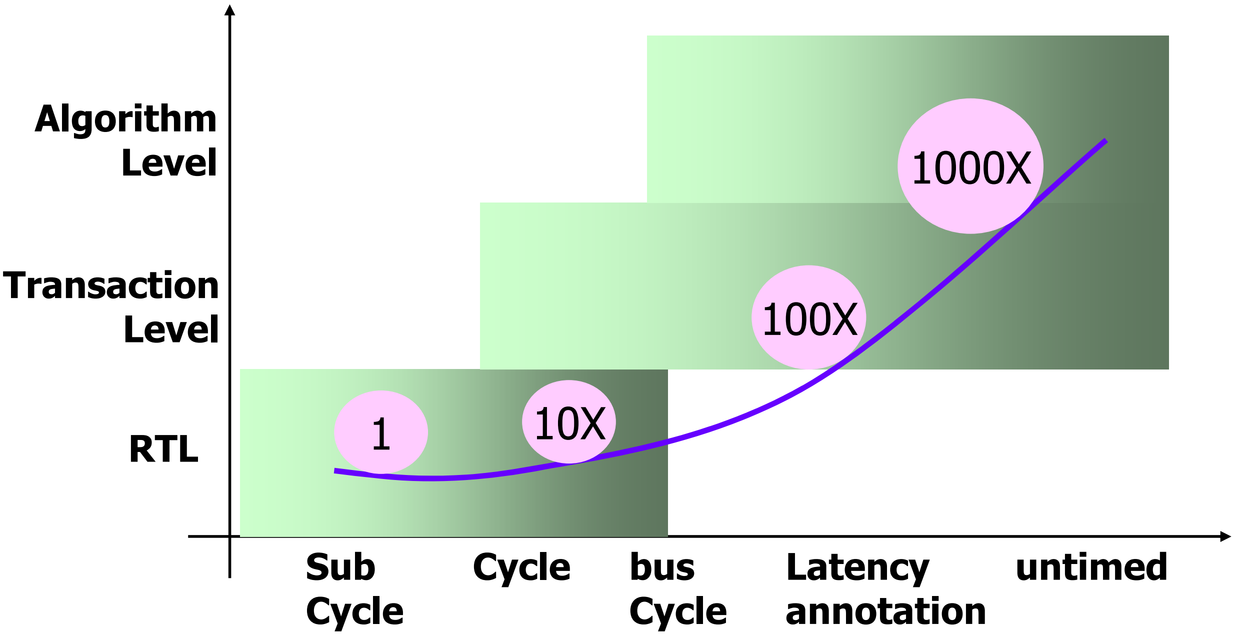
Limited Production

- Even after robust verification process and prototyping, it’s still not guaranteed to be bug-free
- A limited production for new macro is necessary
  - Often called engineering samples
  - 1 to 4 customers
  - Small volume
  - Reducing the risk of supporting problems
- Same as real cases but more ***expensive***
- Only used as the final check before mass production

HW/SW Co-Simulation

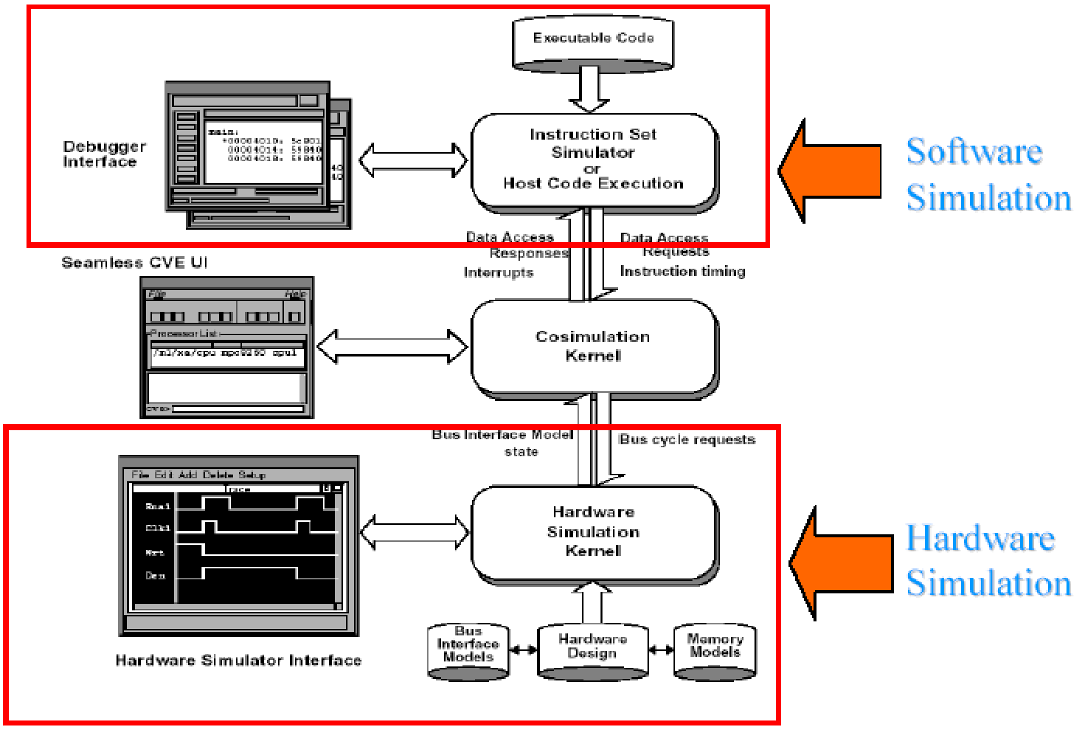
- Couple a software execution environment with a hardware simulator
- Simulate the system at higher levels
  - Software normally executed on an **Instruction Set Simulator** (ISS)
  - A **Bus Interface Model** (BIM) converts software operations into detailed pin operations
- Allows earlier system integration
- Start software development 6 months earlier
- Provide a significant performance improvement for system verification
  - Simulate 100x~1000x faster than RTL

Verification Speed



Reference：Synopsys

Many Available Tools for Co-Sim.



Reference : Mentor Graphics

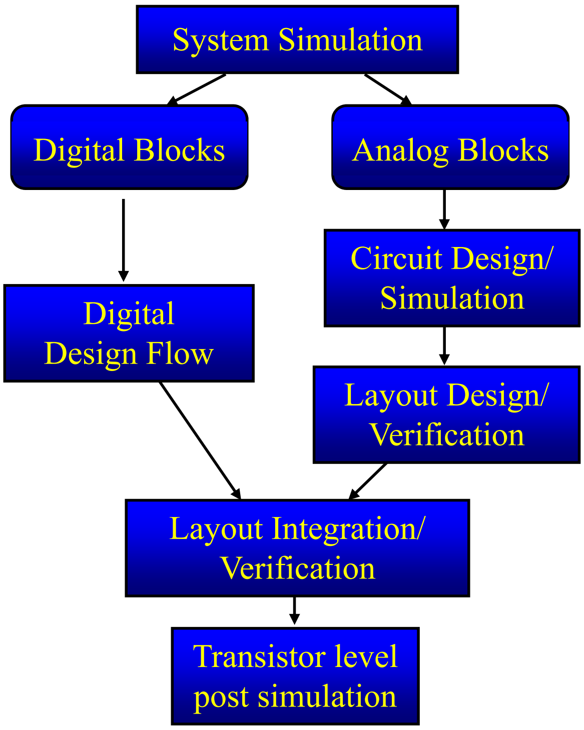
System Verification Today

- Many good verification techniques have been proposed at each design stage
  - Hard to reuse for other stages and design projects
- The fragmented functional verification processes limit the speed and efficiency
  - An unified verification methodology that supports all design domains may be the solution
- An unified ***verification platform*** that optimally supports the methodology is the key
  - Require system-level languages, such as SystemC, SystemVerilog, …

Outline

- Moving to SOC
- What is Verification?
- Functional Verification Solutions
- System Verification Solutions
  - Hardware v.s. software
  - Digital circuits v.s. analog circuits

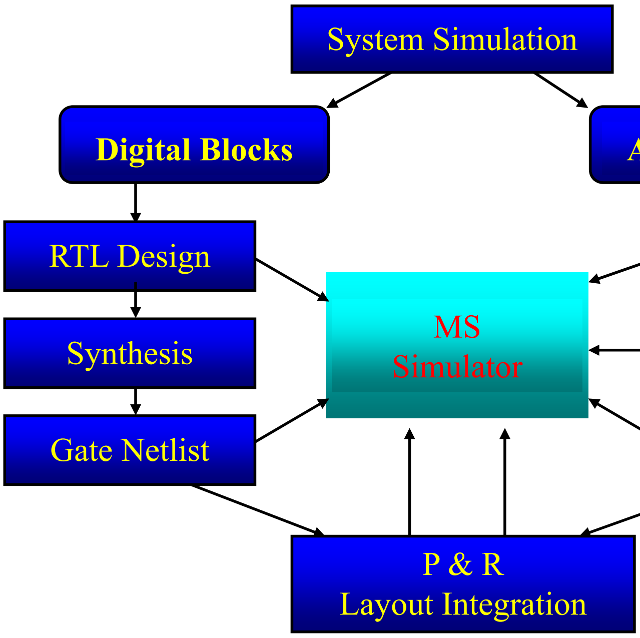
Conventional MS Design Approach


System Simulation
Digital Blocks
Analog Blocks
Circuit Design/ Simulation
Digital Design Flow
Layout Design/ Verification
Layout Integration/ Verification
Transistor level post simulation

- Design and simulate digital and analog circuits separately
  - May miss the effects between each other
- Can perform co-simulation at transistor level only
  - High complexity
  - Too slow

Top-Down MS Design Flow

- Starting from **behavioral models**
  - Can check whole system behavior in advance

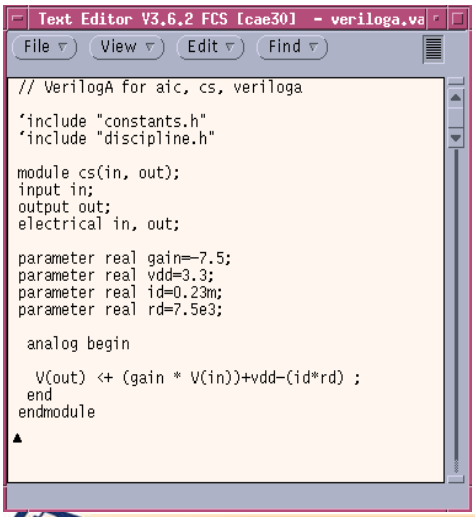


Analog Behavioral Modeling

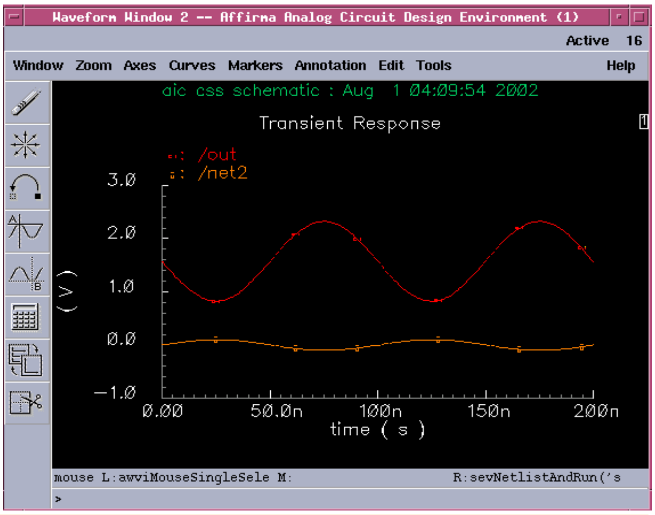
- A mathematical model written in **Hardware Description Language (HDL)**
  - Verilog-AMS
  - VHDL-AMS
  - Matlab
  - C/C++
  - ……
- Emulate circuit block functionality by sensing and responding to circuit conditions
  - Simulate at **behavioral level**
- **Faster** simulation time
  - Allow **whole chip simulation**

An Example of Verilog-A Code

- **Keys to a good behavioral model**
  - Concise mathematical equations of the behavior
    - For faster simulation time
  - Appropriate value for each parameter
    - For accurate simulation results

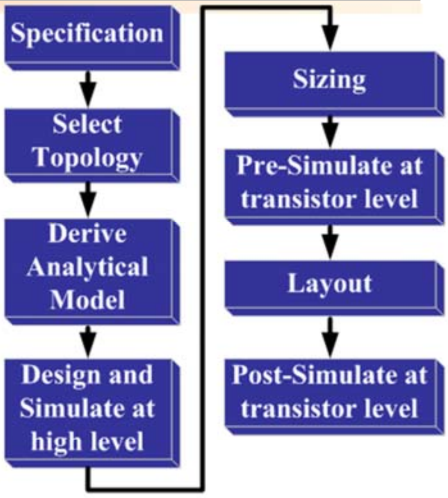


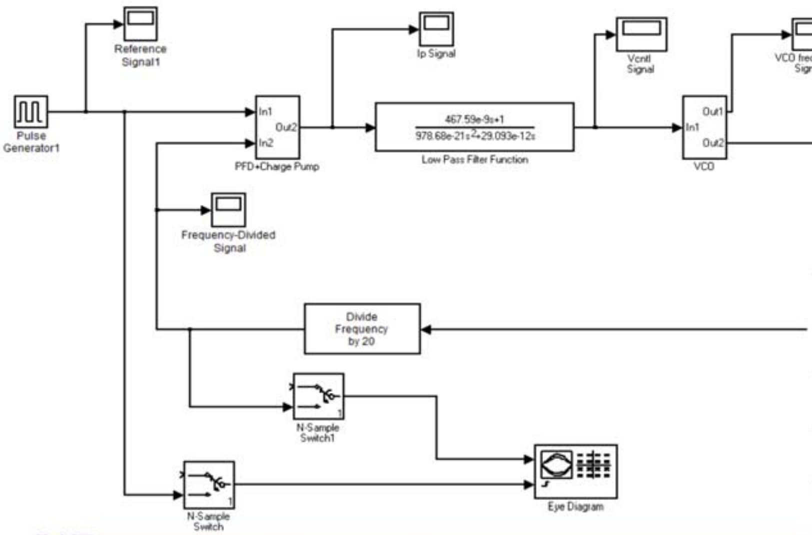
```
// VerilogA for aic, cs, veriloga

'include "constants.h"
'include "discipline.h"

module cs(in, out);
input in;
output out;
electrical in, out;

parameter real gain=-7.5;
parameter real vdd=3.3;
parameter real id=0.23m;
parameter real rd=7.5e3;

 analog begin

  V(out) <+ (gain * V(in))+vdd-(id*rd) ;
 end
endmodule
```

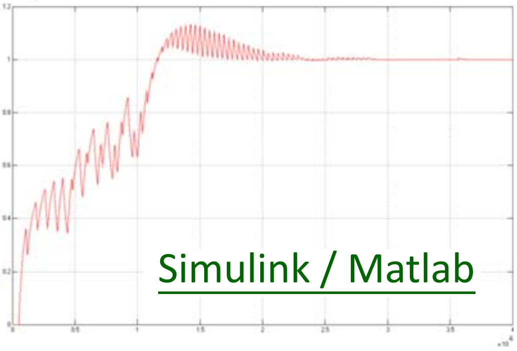


Top-Down Modeling

- Verify whole AMS system **roughly** before implemented
  - Suitable for **newly-created designs**
  - Inaccurate on non-ideal circuit properties


Specification
Select Topology
Derive Analytical Model
Design and Simulate at high level
Sizing
Pre-Simulate at transistor level
Layout
Post-Simulate at transistor level



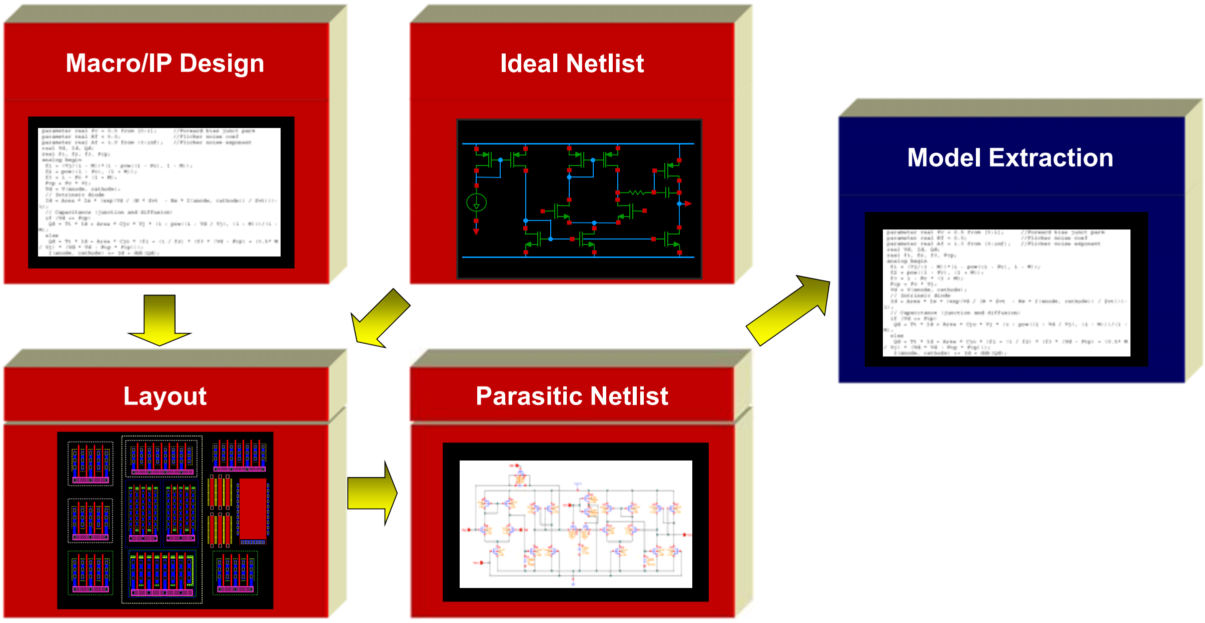


Simulink / Matlab

Bottom-Up Modeling Approach

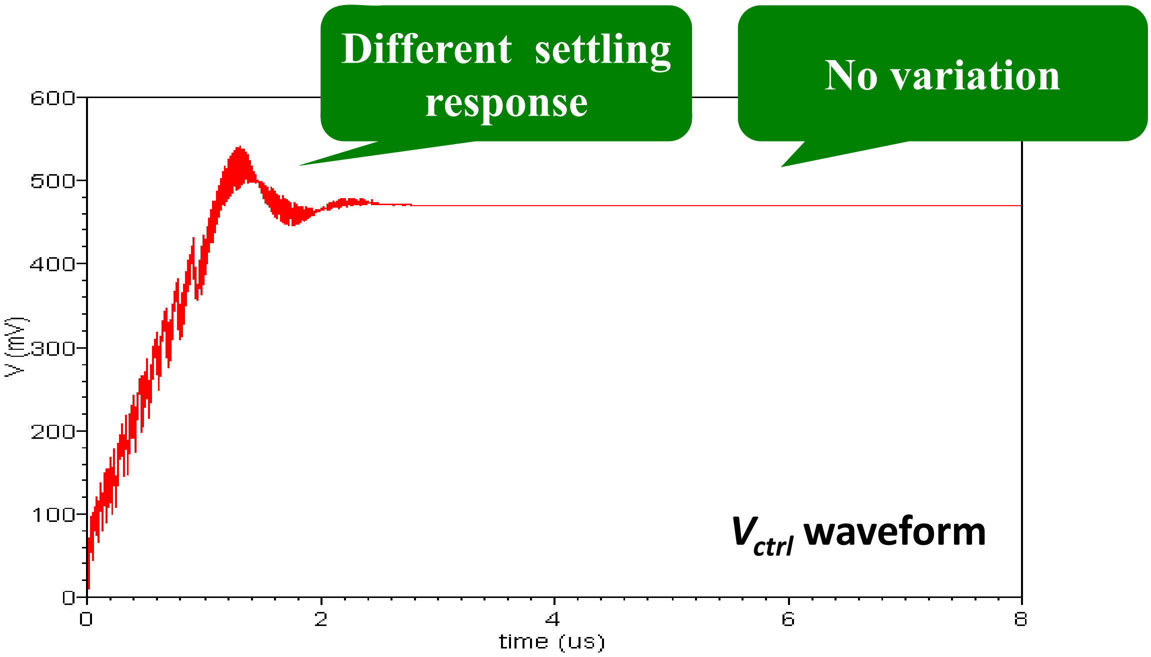
- While using ***existing*** blocks, bottom-up ***behavior extraction*** is required for system verification
  - An interesting research direction
- Bottom-up approach can be much accurate
  - Detailed effects can be considered, such as the delay time of PLL, the effects of parasitic devices (R, L, C, ...)
- Bottom-up approach can still effective when those design parameters are hard to obtain
  - Only have transistor-level design (ex: IP)
  - Circuits are already flattened

Bottom-Up Behavior Extraction



Top-Down Behavioral Model

- Use the embedded behavioral blocks from **Cadence's AHDL library** to model a PLL



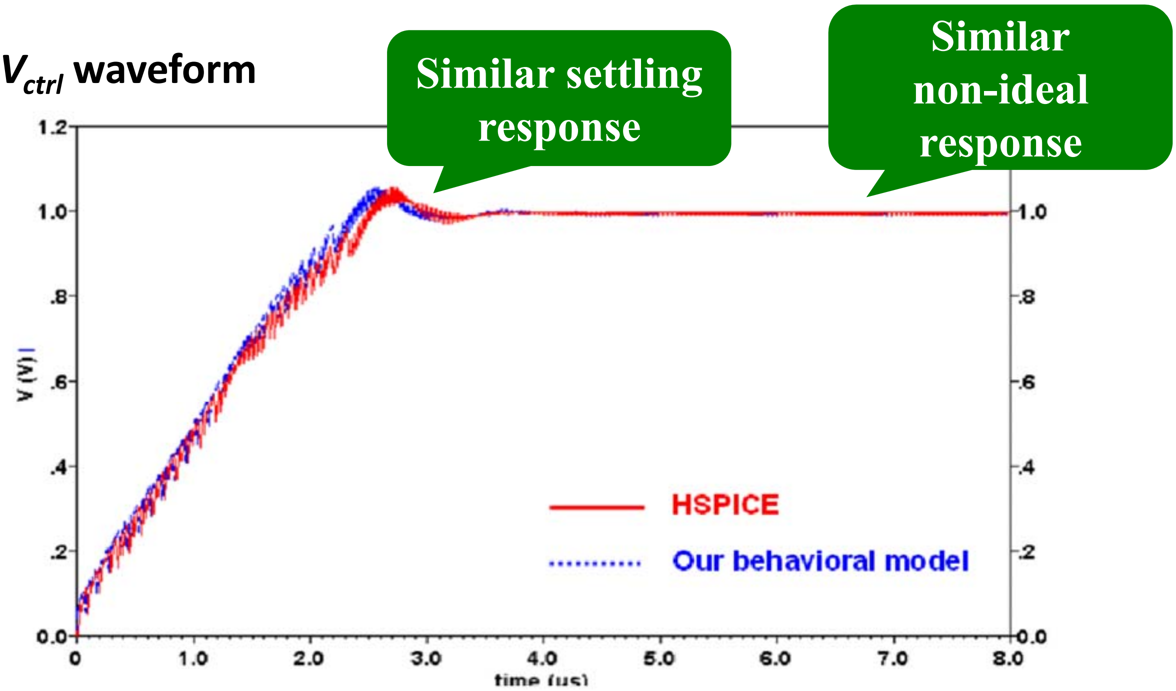
Different settling response

No variation

***V*$_{ctrl}$ waveform**

Extracted Behavioral Model

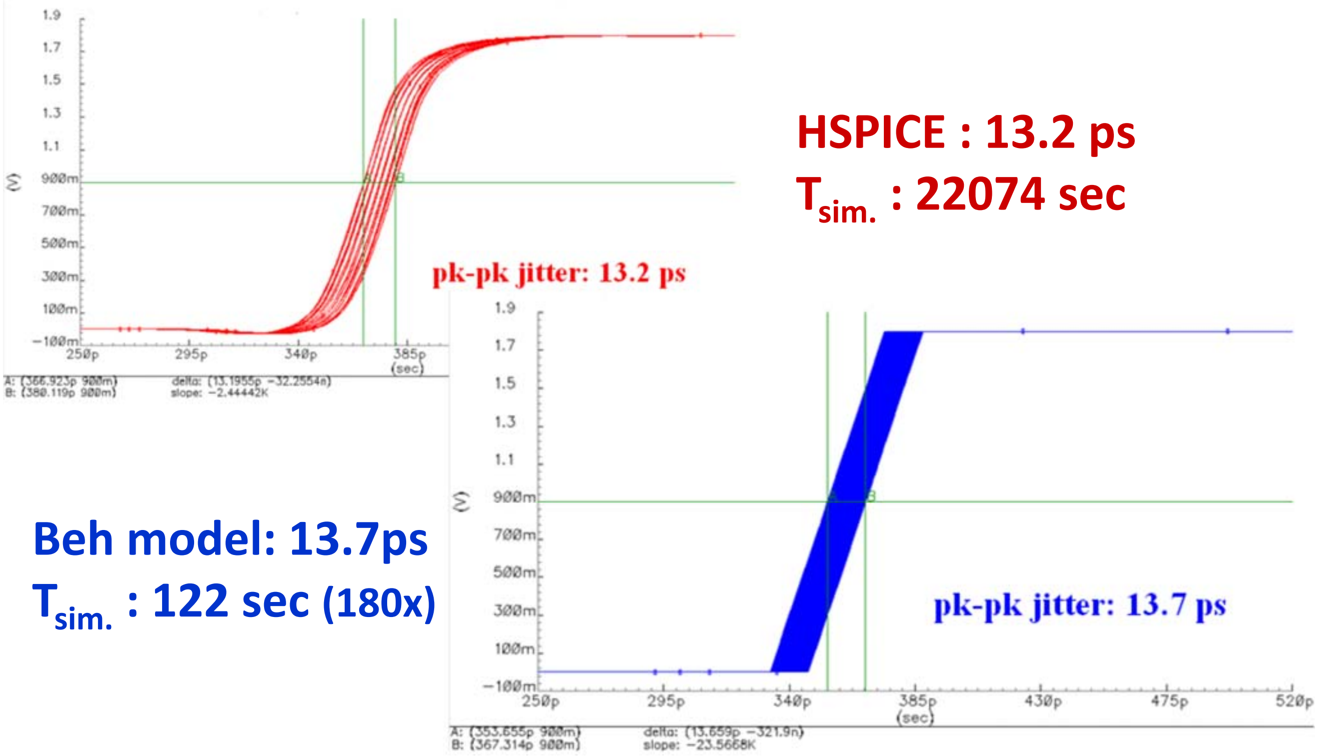
***V*$_{ctrl}$ waveform**



Similar settling response

Similar non-ideal response

Peak-to-peak Jitter



pk-pk jitter: 13.2 ps

pk-pk jitter: 13.7 ps

HSPICE : 13.2 ps

$T_{sim.}$: 22074 sec

Beh model: 13.7ps

$T_{sim.}$: 122 sec (180x)

Conclusions

工欲善其事
必先利其器‼

- SoC is really a big challenge for IC designers
- Design methodology requires a big change
  - Reusing existing IPs and platforms is the key
- Verification has become the major bottleneck
  - Simulation only is hard to cover all functionality
- More powerful tools are essential to solve those difficult design problems