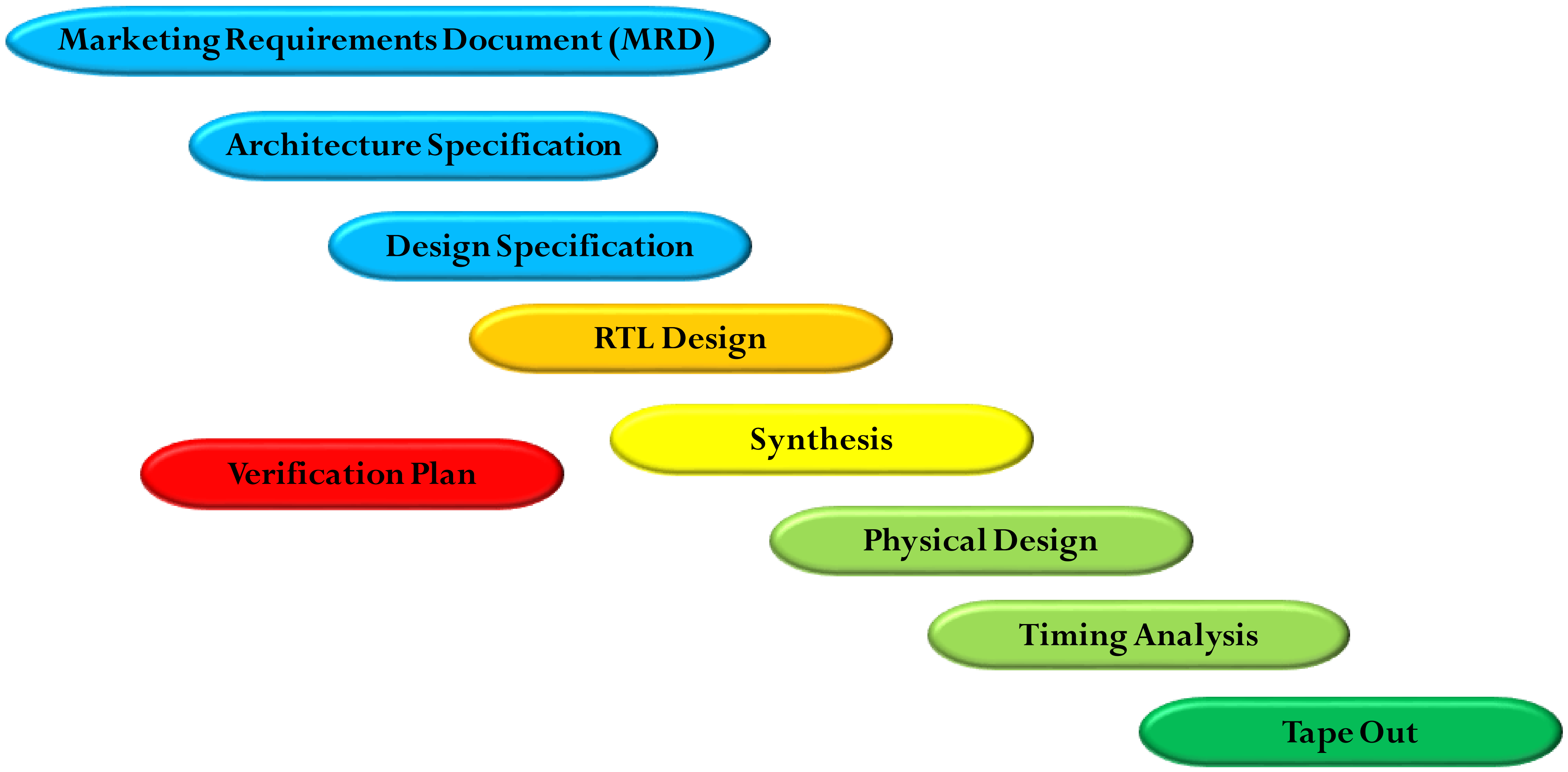
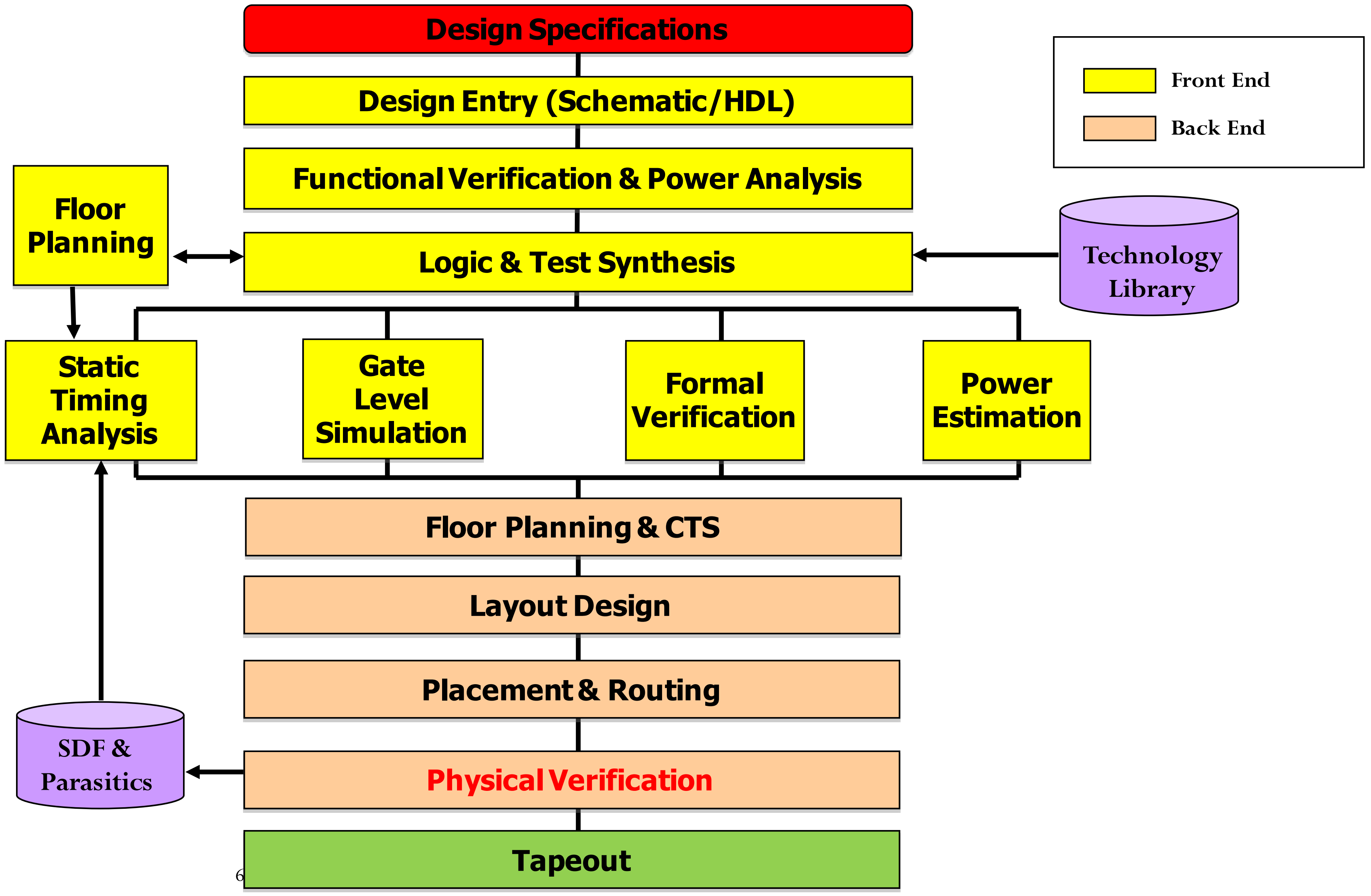
Need of Verification

- To match the design with the intent and specs
- Unexpected behavior of design
- Incorrect interaction between IPs/ sub systems
- Cost of re spin runs into millions of $$
- 70-80% of design time and resources spent on verification

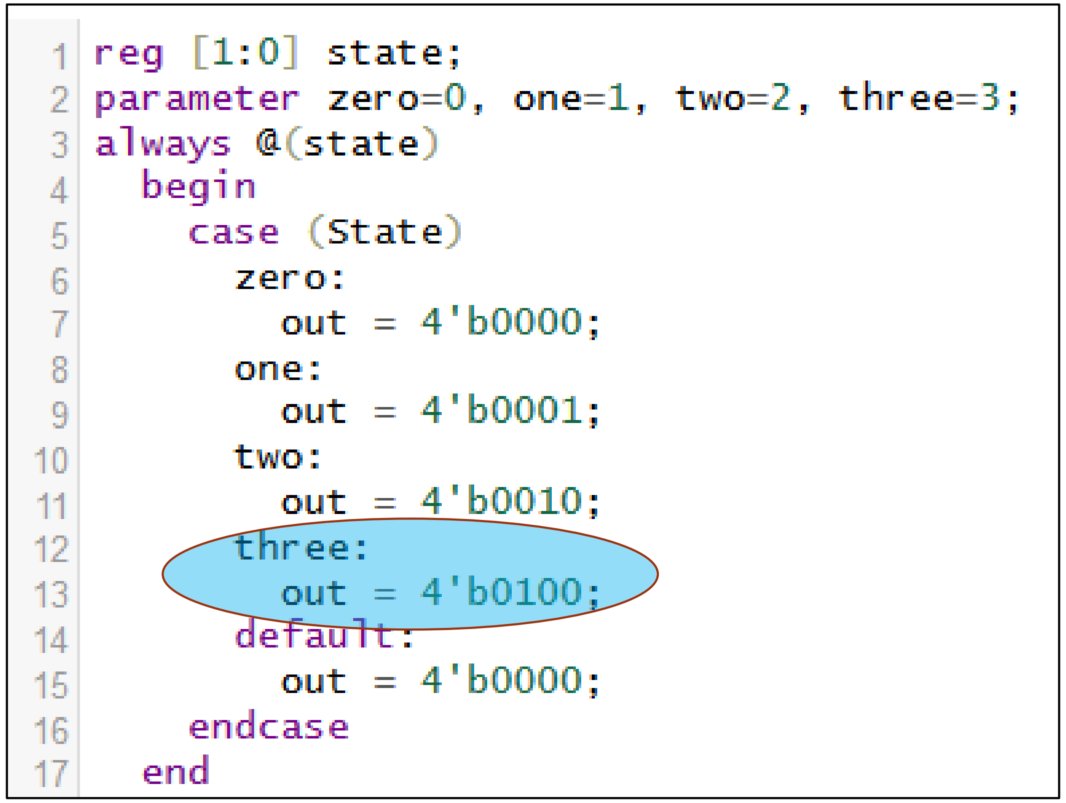
To build confidence and stay in business

- ✓ Verification became the main bottleneck in the design process.
- ✓ The functional verification bottleneck is an effect of rising the design abstraction level.
- ✓ Majority of ASICs require at least one re-spin with 71% of re-spins are due to functional bugs.


Marketing Requirements Document (MRD)
Architecture Specification
Design Specification
RTL Design
Synthesis
Verification Plan
Physical Design
Timing Analysis
Tape Out


Design Specifications
Design Entry (Schematic/HDL)
Functional Verification & Power Analysis
Floor Planning
Logic & Test Synthesis
Technology Library
Static Timing Analysis
Gate Level Simulation
Formal Verification
Power Estimation
Floor Planning & CTS
Layout Design
Placement & Routing
SDF & Parasitics
Physical Verification
Tapeout
Front End
Back End

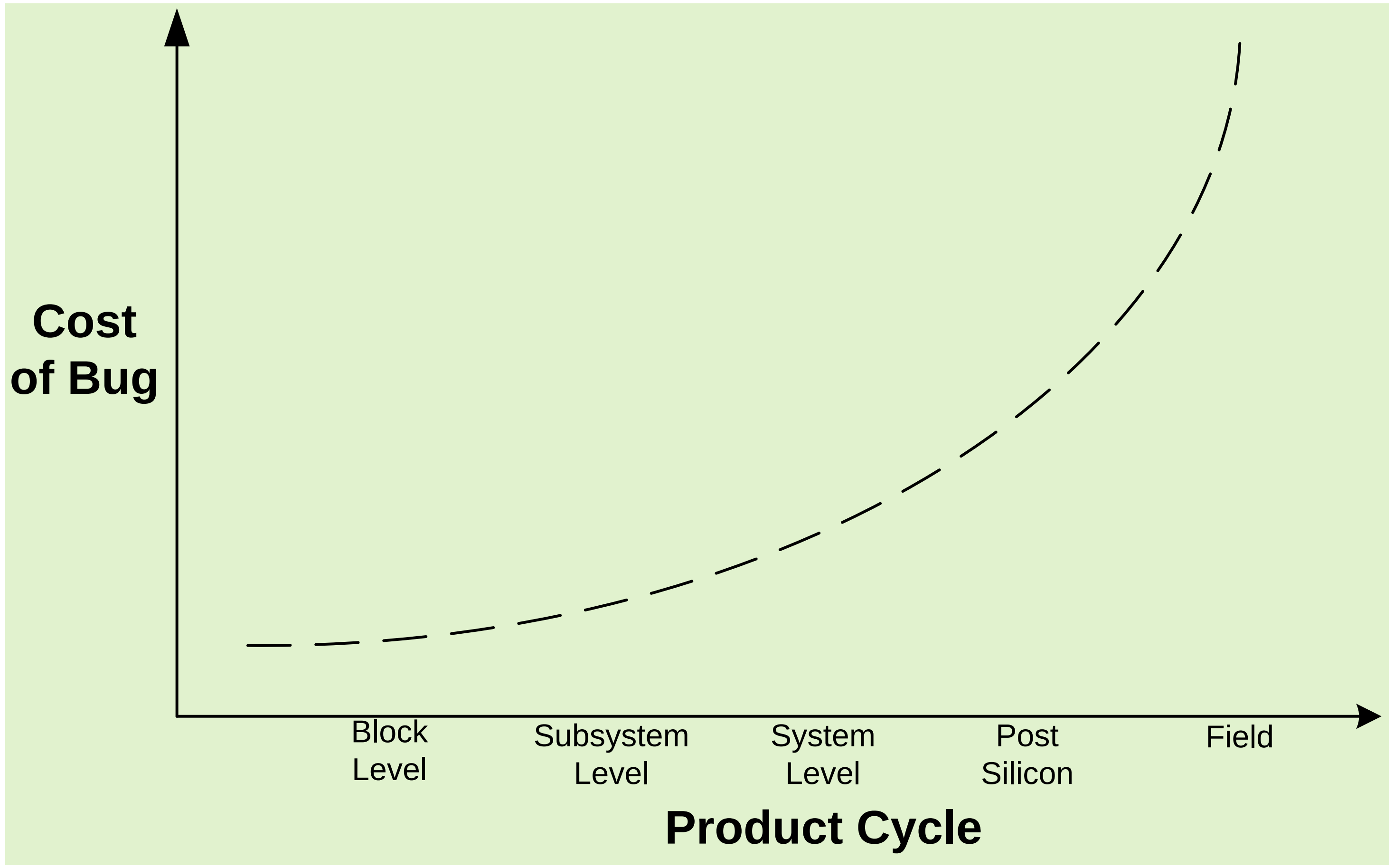
Example of coding error



```
reg [1:0] state;
parameter zero=0, one=1, two=2, three=3;
always @(state)
  begin
    case (State)
      zero:
        out = 4'b0000;
      one:
        out = 4'b0001;
      two:
        out = 4'b0010;
      three:
        out = 4'b0100;
      default:
        out = 4'b0000;
    endcase
  end
```

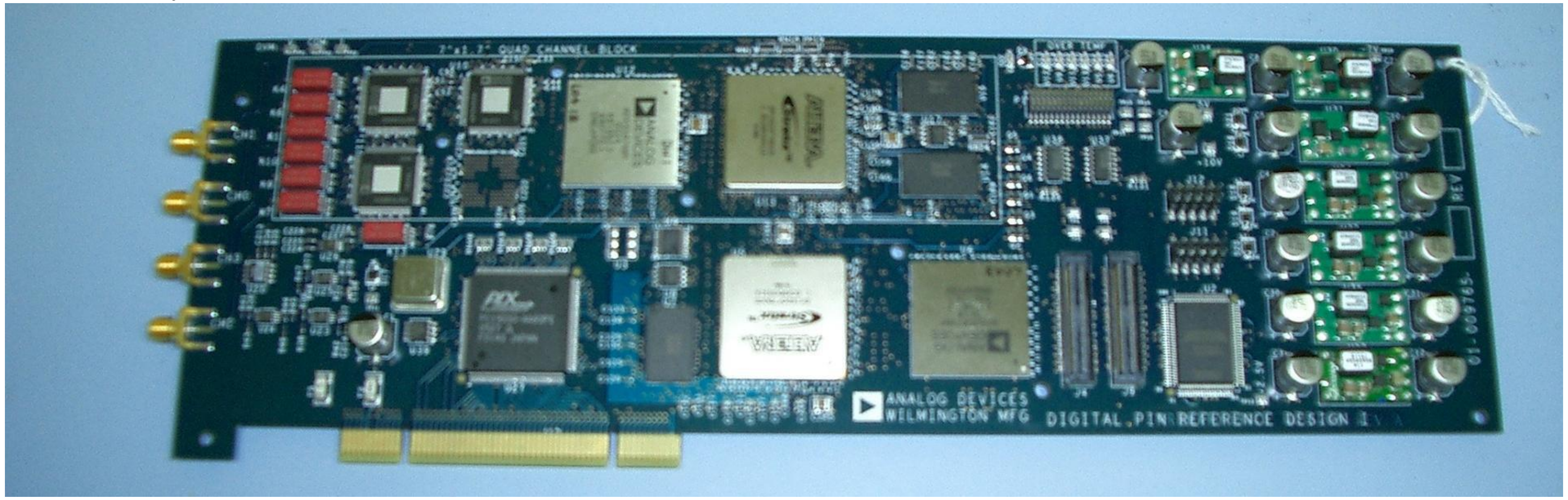
Why Verify?

The later in the product cycle a bug is found the more costly it is.



FPGAs Need Verification!

Case Study 1:
LOA Technology
2 fully custom FPGA's



45 man weeks of verification effort ← 131 bugs found

10 man-weeks of lab debug ← 9 bugs found

100% statement coverage

Source: Chris Spear: System Verilog for Verification

FPGAs Need Verification!

Case Study 2:

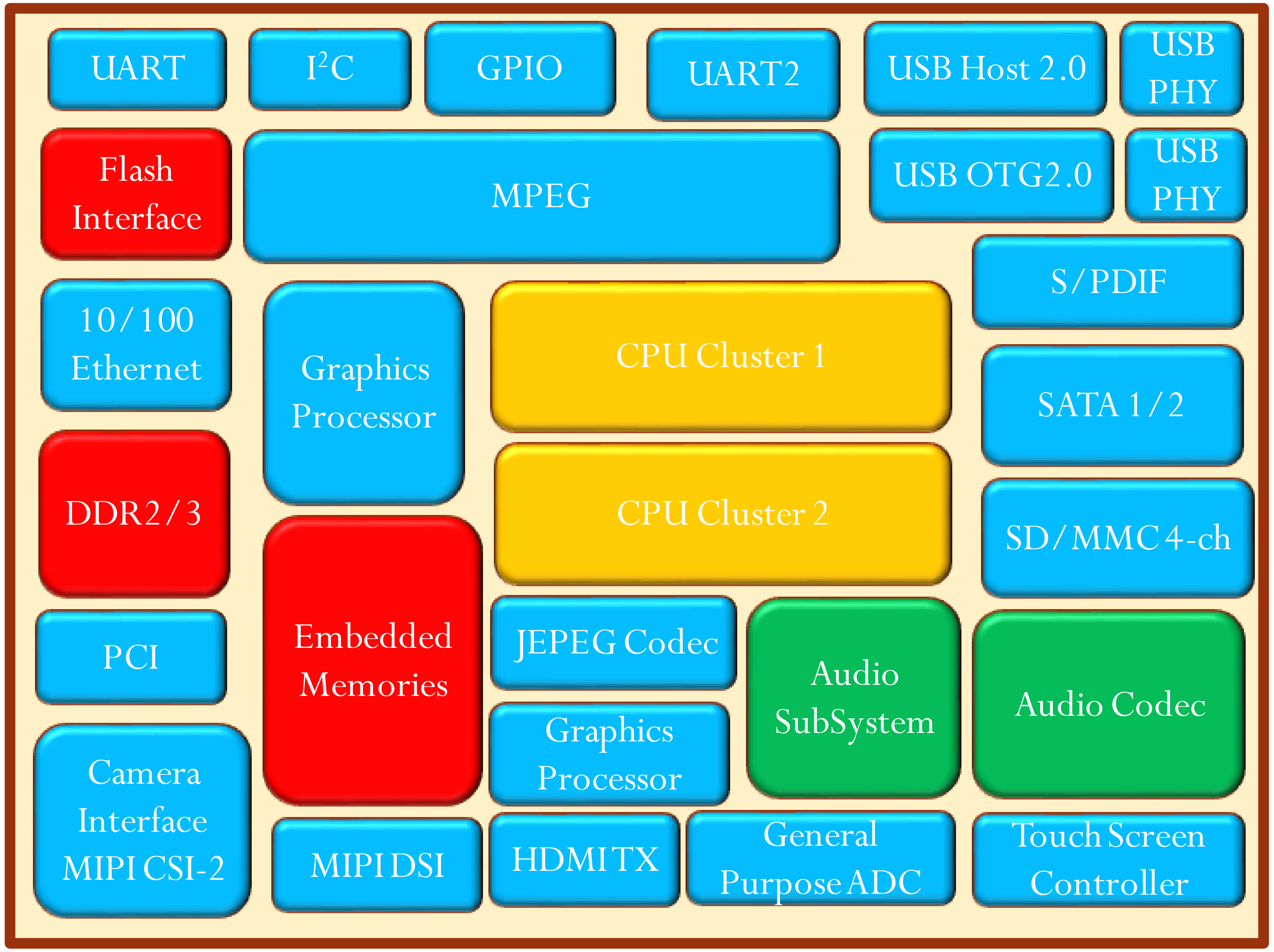
- Fore River Group
- Hardware acceleration for network security.
- Existing testbench with 100 directed tests.
- Shipping to customers.
- 7 man-months of effort.
- Using random verification found 40 bugs

Case Study 3:

- Fore River Group
- Packet Switching device
- Verification considered complete
- 6 man-months of effort.
- Using random verification found 42 bugs

Source: Chris Spear: System Verilog for Verification

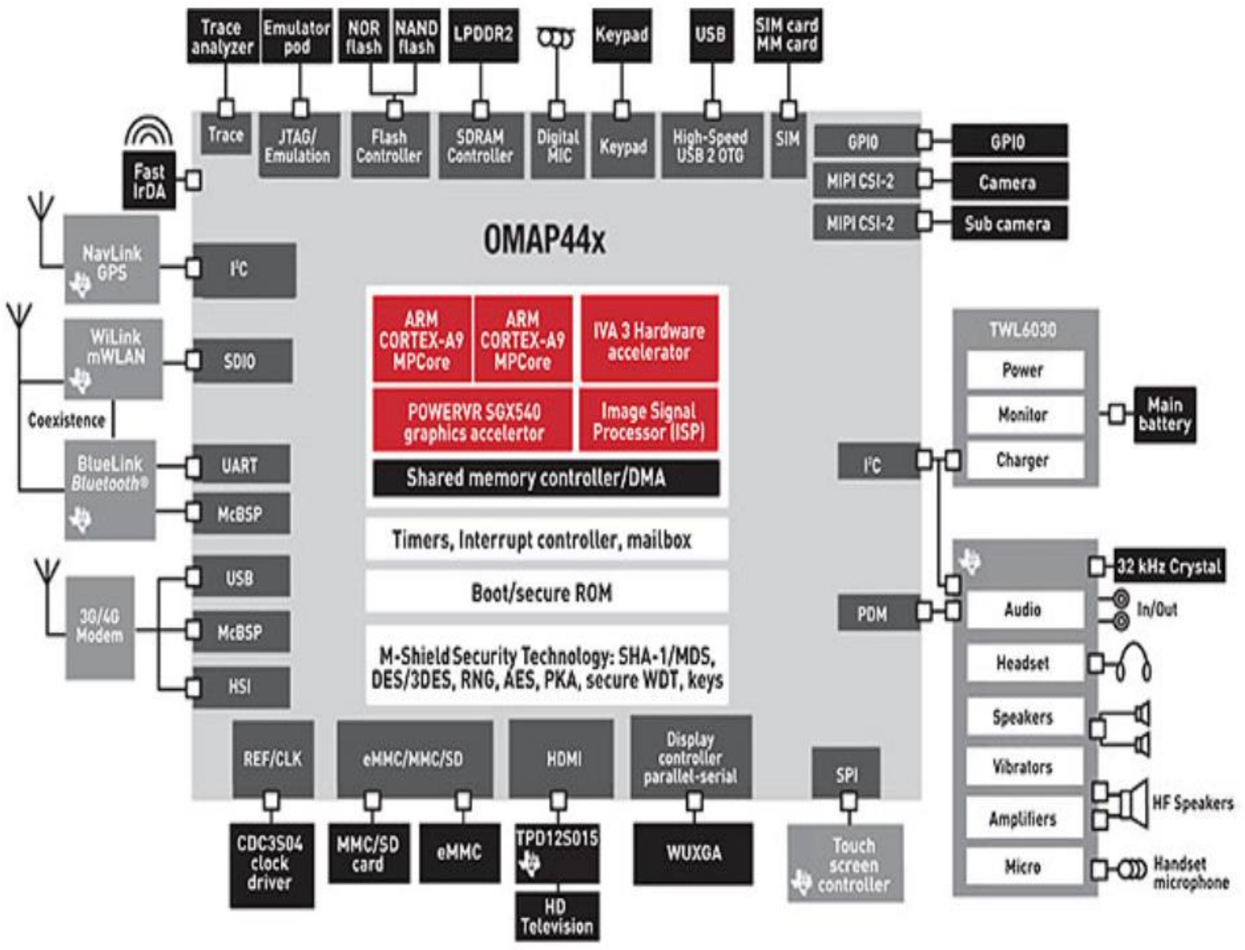
Block Diagram of Typical SoC

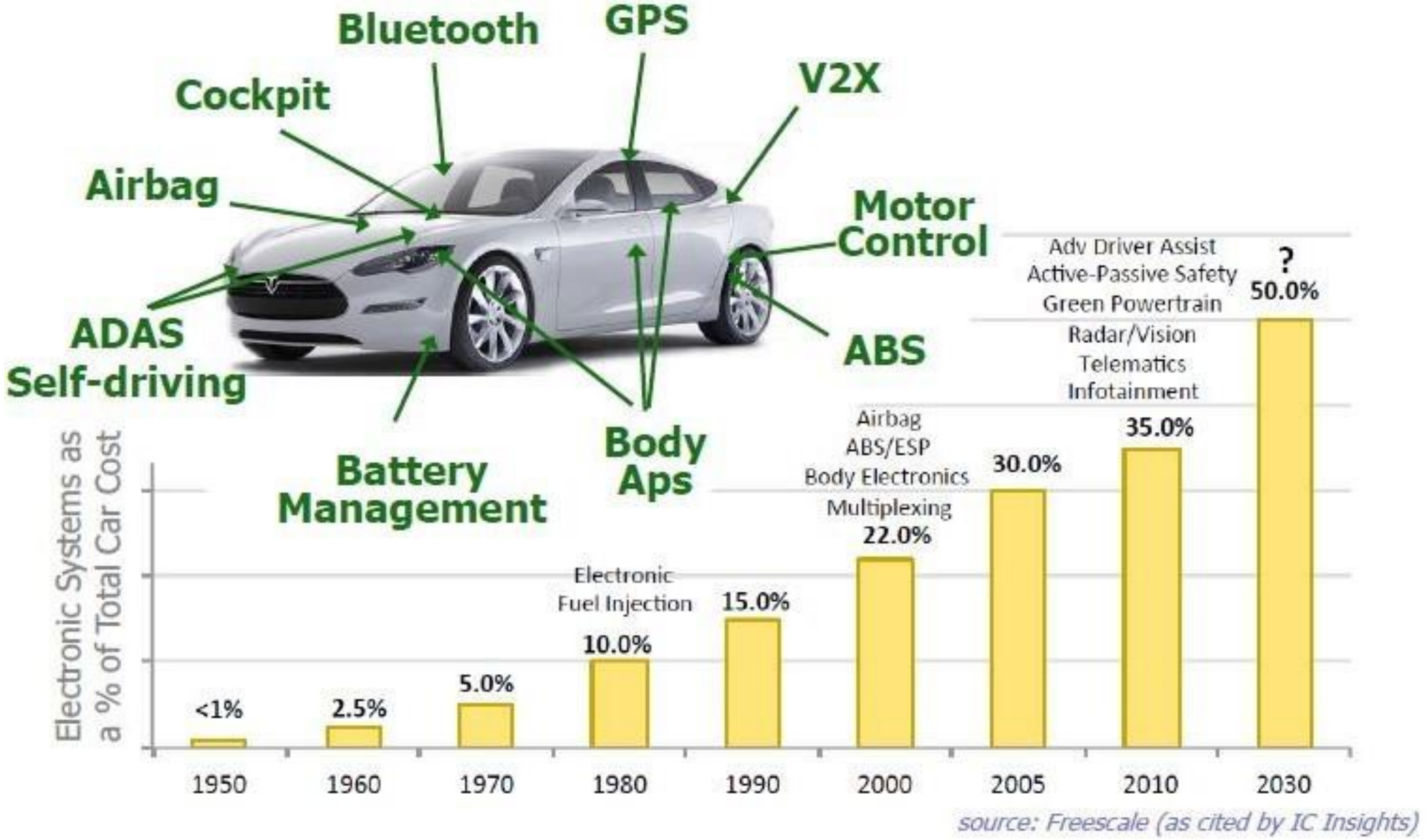


How to verify all the top-level connectivity is correct?

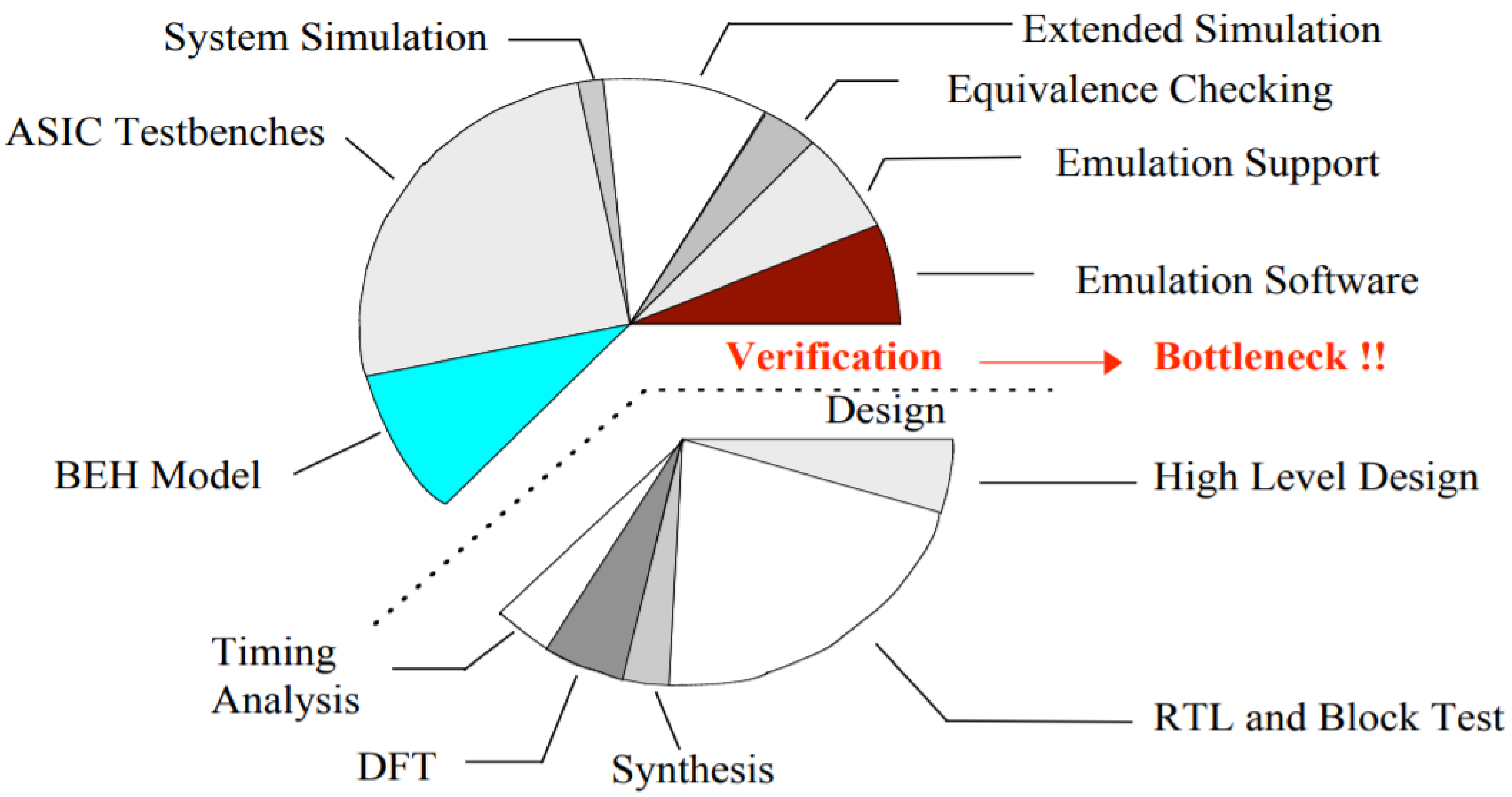
Smart Phone SoC

- System On Chip (SOC) is equivalent to Computer motherboard for phone.
- A typical SOC includes
  - CPU - multi-core
  - GPU - multi-core
  - ISP (Image Signal Processor)
  - Video Encoders/Decoders
  - Memory & caches
  - Miscellaneous components..
- A typical design cycle of around 2 years
- Involves multiple vendors supplying Design IPs & services creating inter-dependencies.




Bluetooth
GPS
Cockpit
V2X
Airbag
Motor Control
ADAS
Self-driving
ABS
Battery Management
Body Aps
Electronic Systems as a % of Total Car Cost
<1%
2.5%
5.0%
10.0%
Electronic Fuel Injection
15.0%
Airbag
ABS/ESP
Body Electronics
Multiplexing
22.0%
30.0%
35.0%
Adv Driver Assist
Active-Passive Safety
Green Powertrain
Radar/Vision
Telematics
Infotainment
?
50.0%
1950
1960
1970
1980
1990
2000
2005
2010
2030
source: Freescale (as cited by IC Insights)

Functional Verification

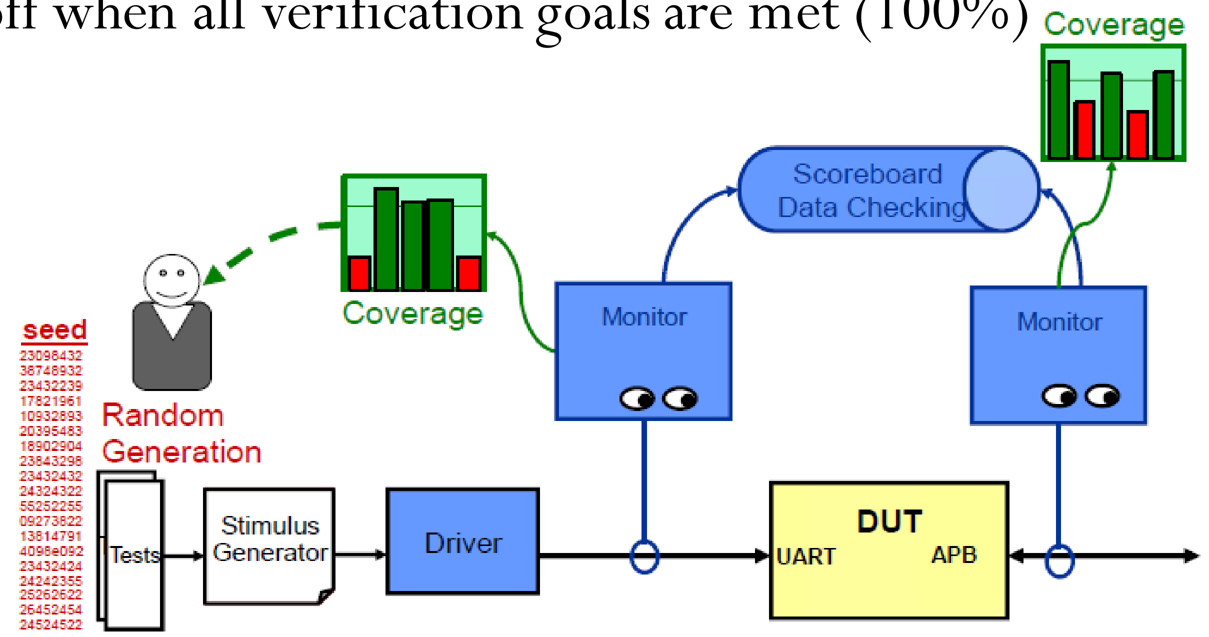


Functional Verification

- Process to demonstrate functional correctness of the design
- Accounts for 60 – 80% efforts of ASIC design cycle
- Logic Simulation using System Verilog UVM
  - Random Testing – scenarios engineer cannot anticipate
  - Functional Testing – scenarios defined by engineer
  - Corner Case Testing – Hard to hit scenarios defined
  - Regression Testing – Automated combination of all of above with repetitive runs
- Formal Verification using System Verilog Assertions
  - Immediate Assertions (similar to if statements)
  - Concurrent Assertions (Behavior spans across multiple clock cycles)
  - Cover Properties (identify hitting scenarios)
- Emulation Testing – Mimic HW to test real life scenarios

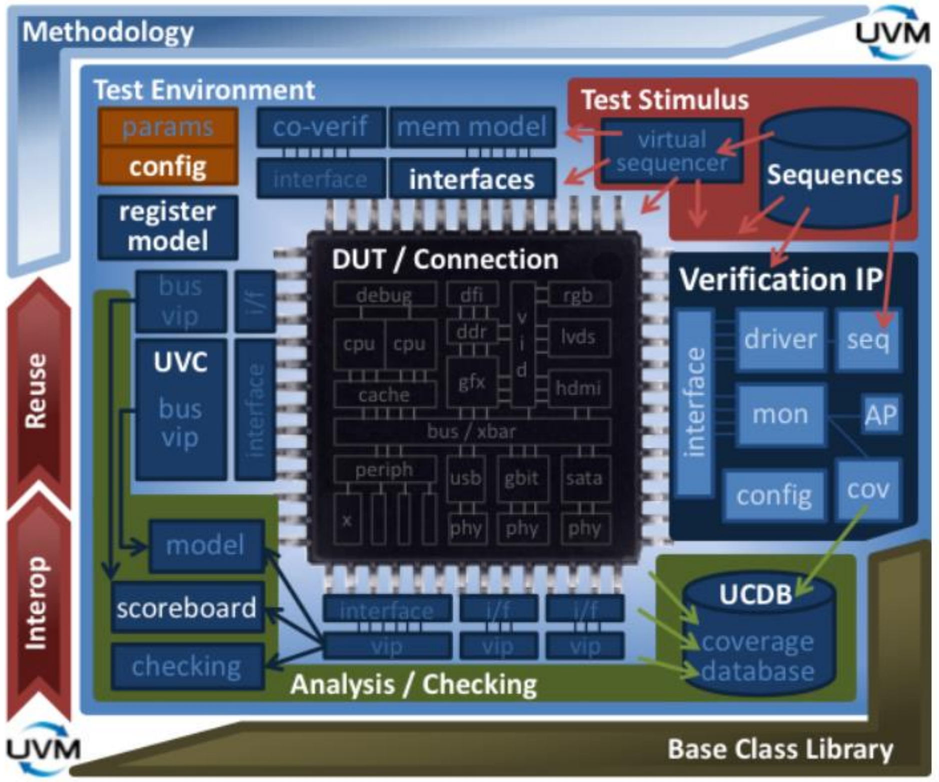
Metric Driven Verification

- A clear and un-ambiguous specification is used to create a Verification Plan
  - defines what to test & how.
- Plan defines Test cases, Coverage and Checker model
- Verification Environment consists of:
  - Monitors, Drivers, Scoreboards, Stimulus generator , coverage Model
- Stimulus generation is random, automated & user constrained to drive legal stimulus
- Verification is signed-off when all verification goals are met (100%)



Universal Verification Methodology

- UVM is a standard verification methodology from the Accellera Systems Initiative that was developed with support from multiple vendors: Aldec, Cadence, Mentor Graphics, and Synopsys.
- It is designed to enable creation of robust, reusable, interoperable verification IP and testbench components
- Includes a Reference Guide, a Reference Implementation in the form of a System Verilog base class library, and a User Guide.
- First version UVM 1.0 released in 2011
- Supports Object Oriented Programming (OOPS) concepts



Why are Methodologies Needed?

- Vertical and horizontal Reuse
- Standardized architecture across teams
- Automation
- Upgradation
- Consistent approach – naming conventions
- Coding guidelines
- Controllability
- Vendor Dependence