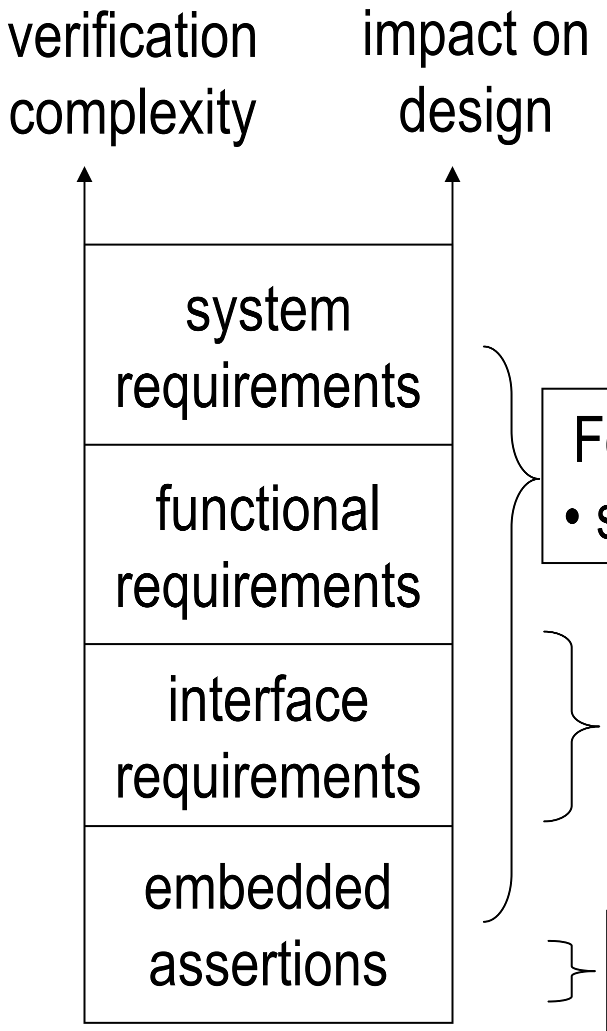
Hierarchy of Design Requirements

formal verification at the RTL interface level

C. Norris Ip
ip@tempusf.com, Tempus Fugit, Inc.

Outline



Formal Requirement Verification on RTL
- shorten the lengthy verification process

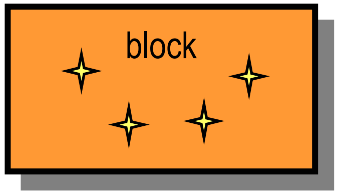
Formal Interface Verification on RTL
- easy to adopt

Formal Interface Verification Module
- second pair of eyes

Basic Formal Verification
- fully automated

Design Requirements

- Embedded Assertions
  - document designer's local decision



- Example
  - one hot state machine encoding

    state[0] + state[1] + … + state[13] == 1

  - buffer overflows

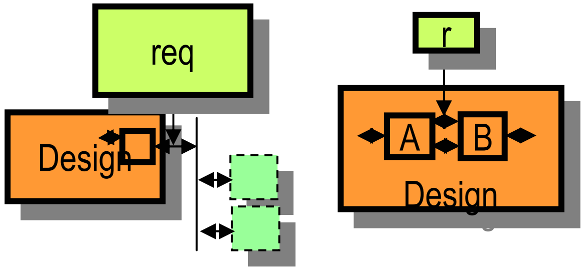
    fifo_size == 4'b1111 && next_fifo_size == 4'b0000

  - logical assumption

    ~ sig_A => sig_B | sig_C

Design Requirements

- Interface Requirements
  - avoid interoperability problem
  - avoid misunderstanding of spec
  - exchange designers' assumption

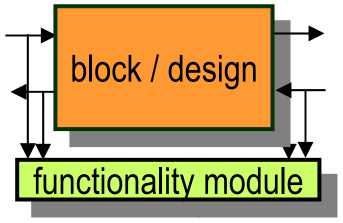


- Example:
  - PCI (frame enable control)

    last_frame_enable && ~last_frame_n => frame_enable
  - PCI (CBE interpretation)
    - the master must always assert byte enables on the upper bits of CBE when the master asserts REQ64# and the slave has not responded

Design Requirements

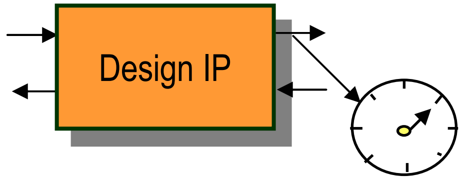
- Functional Requirements
  - corner cases in how data is transformed
  - corner cases in how data flows through the design



- Example
  - PCI Express (packet-based interface)
    - retrying a packet when the timer expires before the acknowledgement comes in (PCI-Express)
      - Sequence number is generated correctly
      - Packets come out in the right order with possible retries
      - Data integrity
      - No drop packets
      - No duplicated packets

Design Requirements

- System Requirements
  - throughput, error rates, drop rates

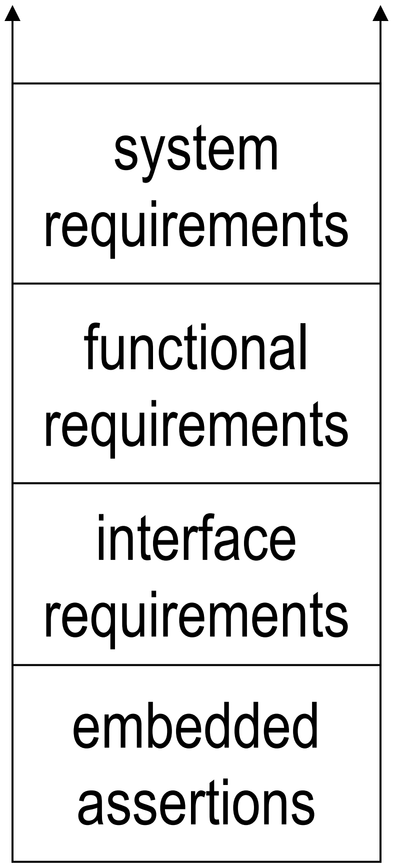


- Traditionally this is where simulation is strongest:

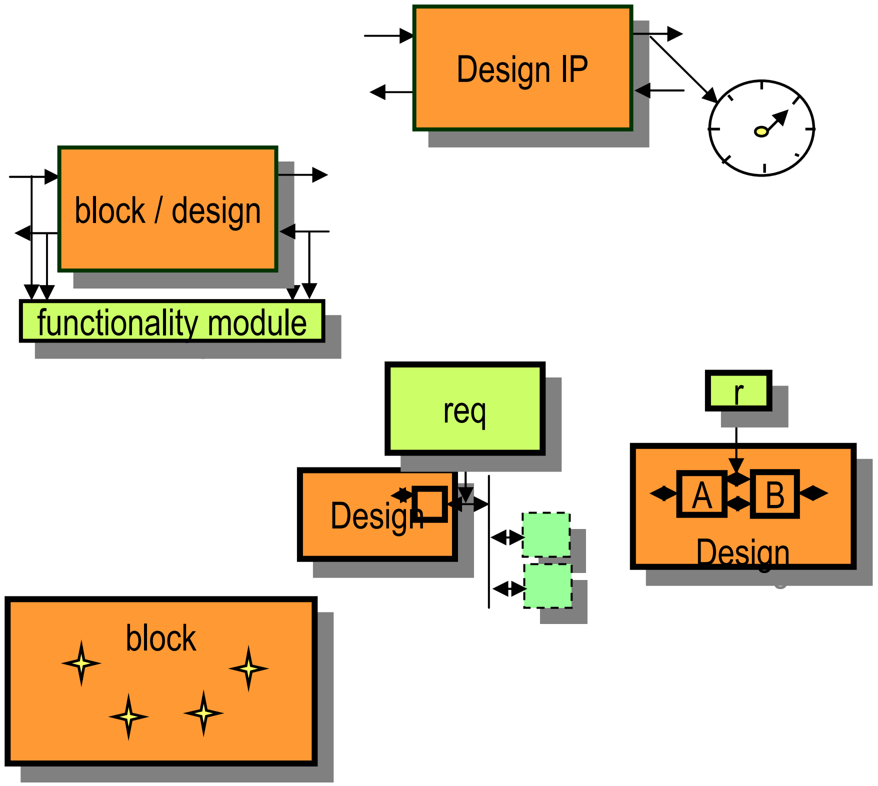
- System Requirements
  - complex distributed protocols
  - micro-architecture

- Traditionally verification of this is done on system-level description

Summary

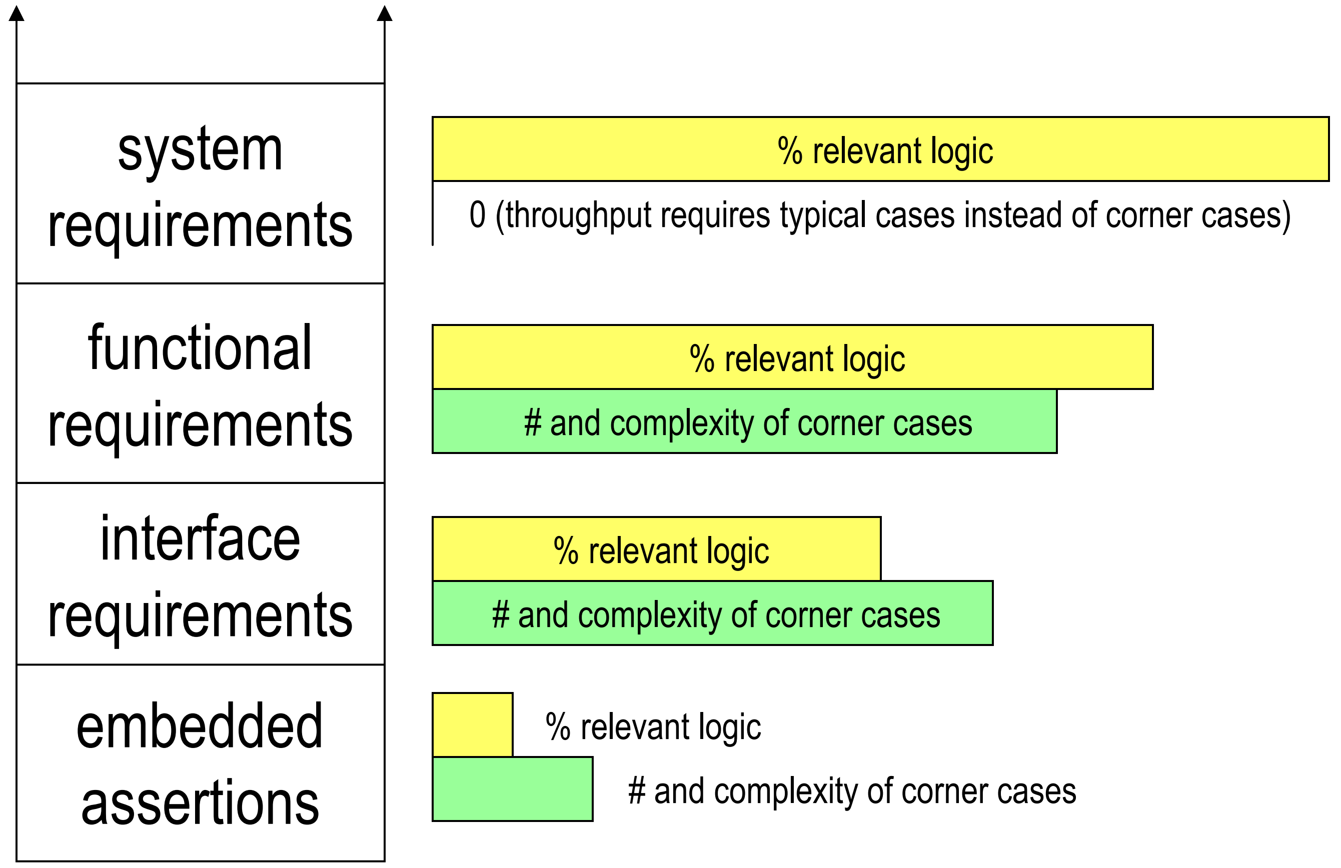


system requirements
functional requirements
interface requirements
embedded assertions

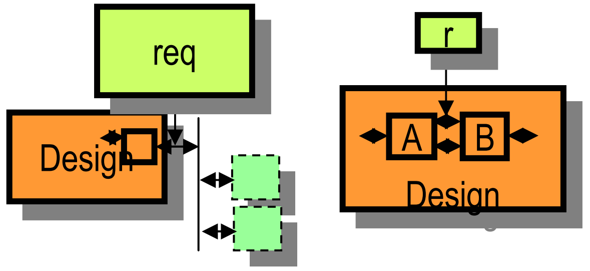


Design IP
block / design
functionality module
req
Design
r
A
B
Design
block

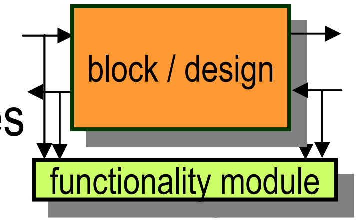
Summary of Complexity


system requirements
% relevant logic
0 (throughput requires typical cases instead of corner cases)
functional requirements
% relevant logic
# and complexity of corner cases
interface requirements
% relevant logic
# and complexity of corner cases
embedded assertions
% relevant logic
# and complexity of corner cases

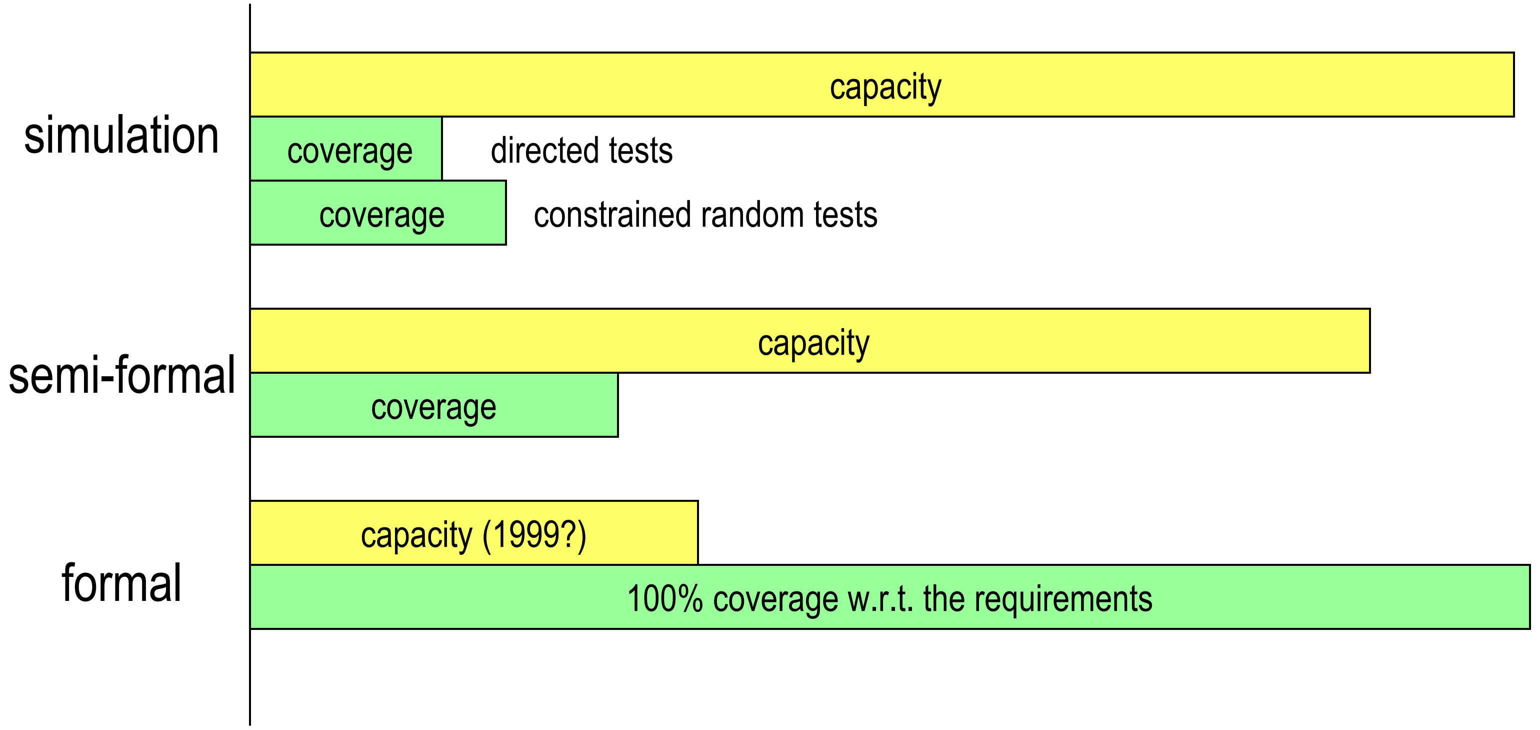
Requirements at the Interface Level



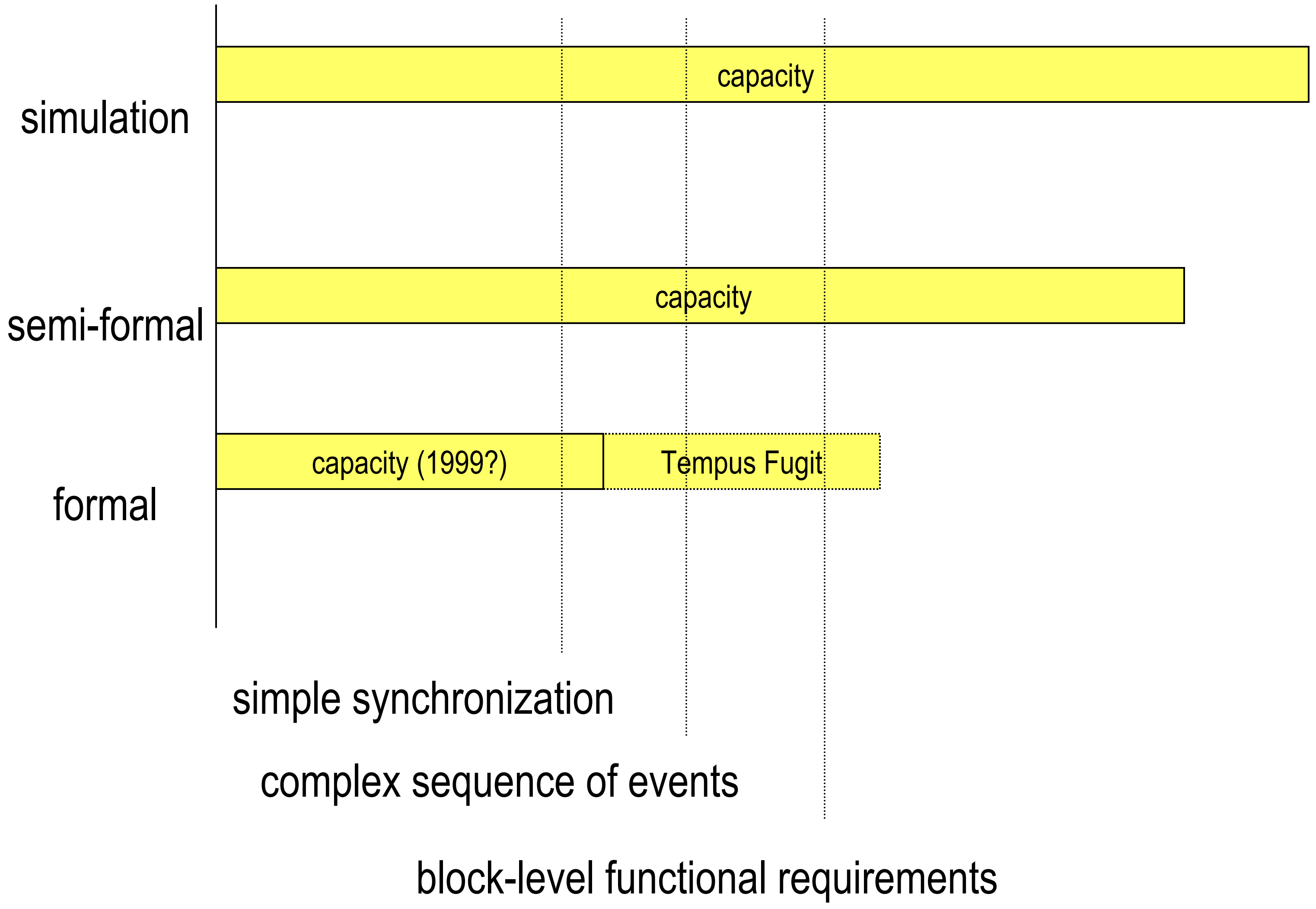
- Simple Synchronization
  - when should I assert a signal?
- Complex Sequence of Events
  - target initiates wait-state during a write-transaction/split-completion only at appropriate data phases (PCI-X)
  - FRAME is de-asserted at appropriate cycle w.r.t. the IRDY depending on the number of data phases (PCI-X)
- Functional Requirements at Interface Level !
  - data being scrambled correctly by maintaining the correct linear functional shift register values (PCI-Express)
  - sequence number is generated correctly
  - packets come out in the right order with possible retries



Pure Formal Verification !



Pure Formal RTL Verification !

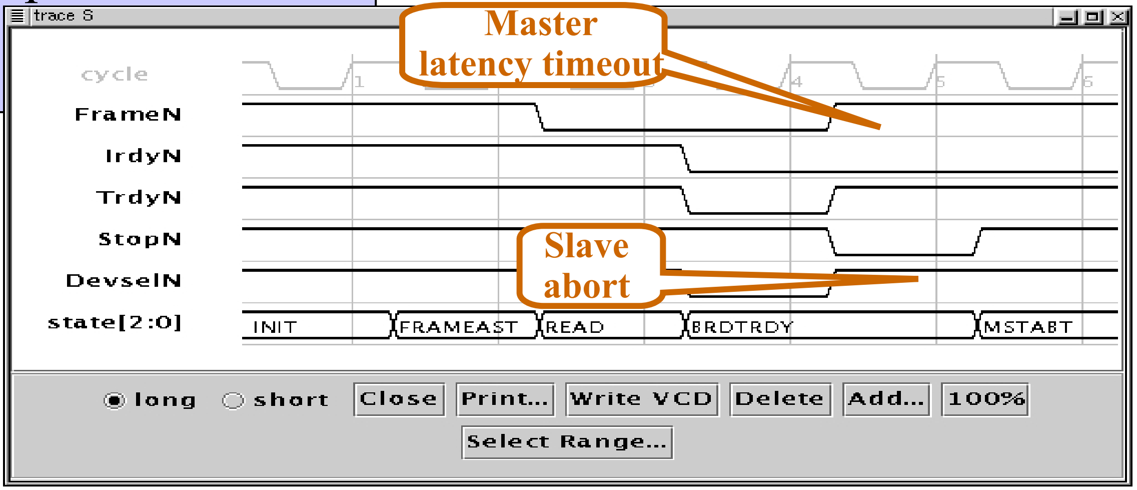


Productivity Gain

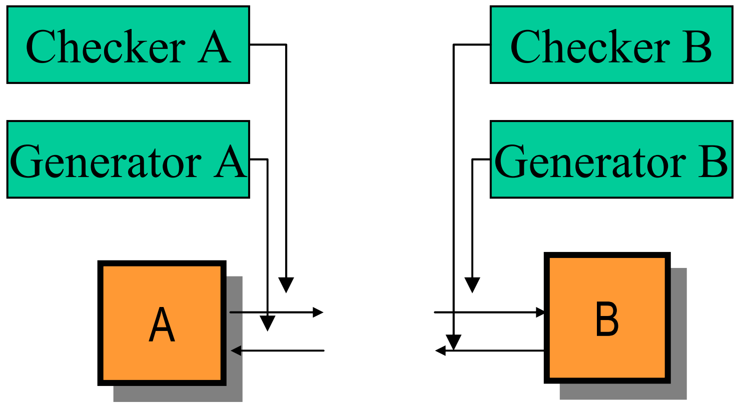
For an engineer, capturing requirements is easy;
enumerating corner cases is difficult.

- Requirement spec is >10X fewer lines than test benches
- 100% coverage vs. incomplete coverage

IUT must de-assert IRDY# the clock after the completion of last data phase
(PCI 2.2 spec 3.3.3.1)

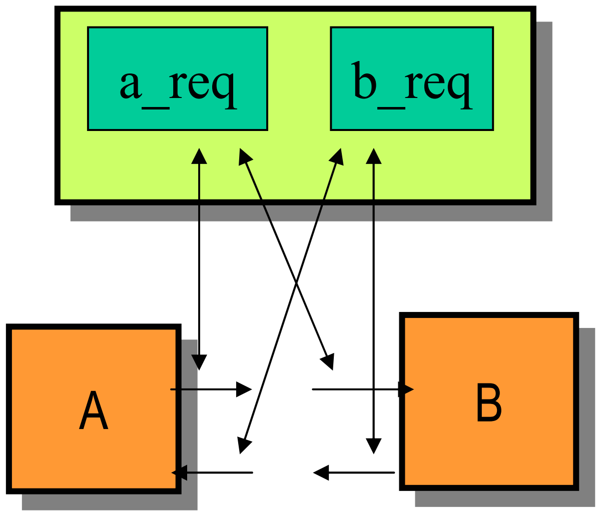


Formal Contract



Traditional unit tests

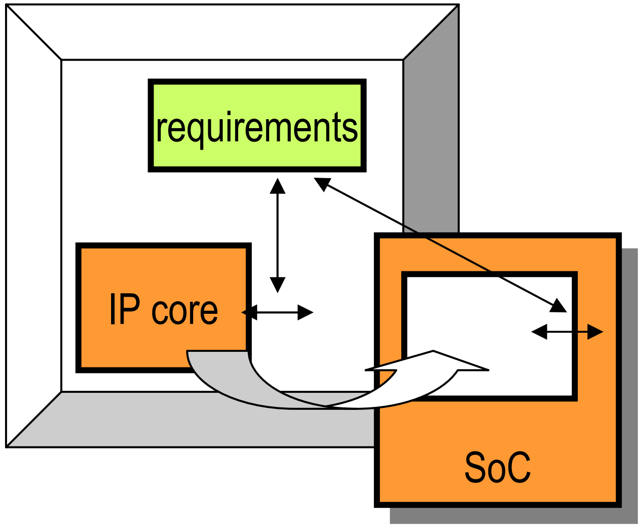
Formal Contract (100% coverage)



% assume b_req	% assume a_req
% prove a_req	% prove b_req

Formal IP Sign-off

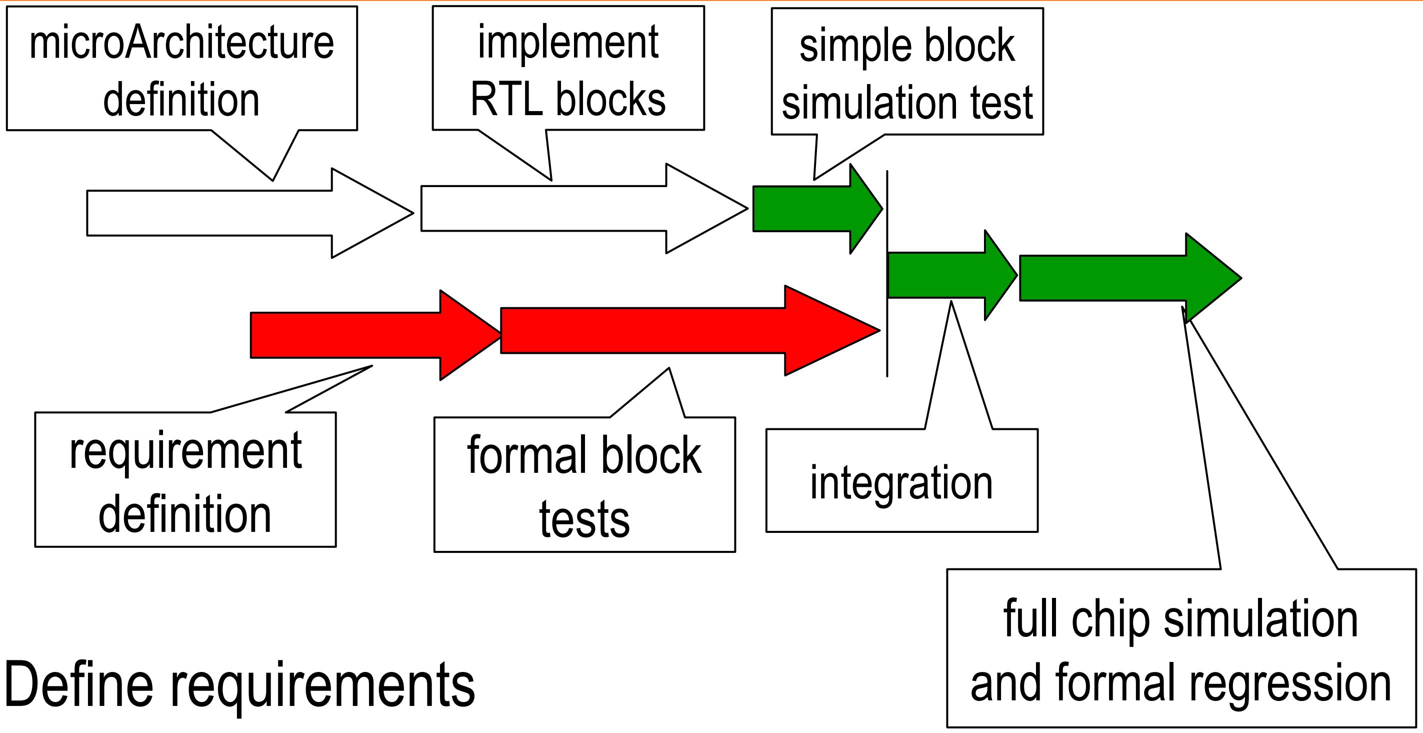
% assume env
% prove ip_req



% assume ip_req
% prove env

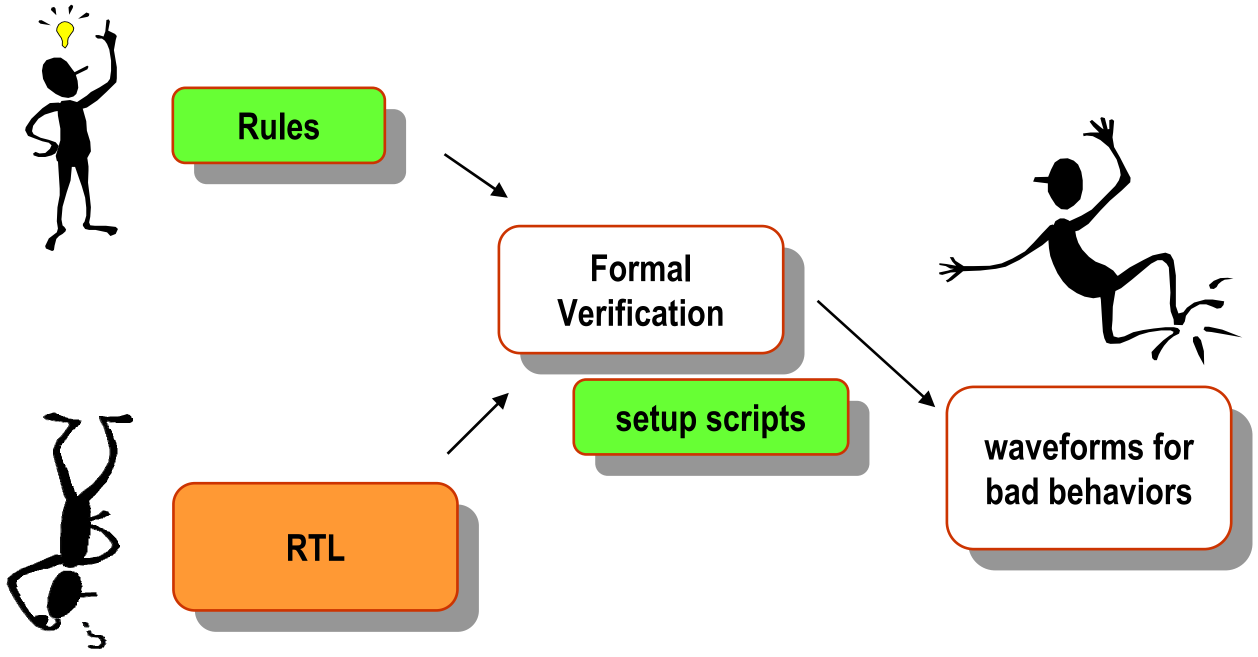
Or use "env" as simulator monitor

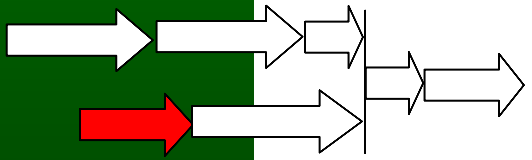
Formal Verification in a Design Flow


microArchitecture definition
implement RTL blocks
simple block simulation test
requirement definition
formal block tests
integration
full chip simulation and formal regression

- Define requirements
- Verify on partially completed RTL units
- Exchange assumptions
- Straightforward integration and regression

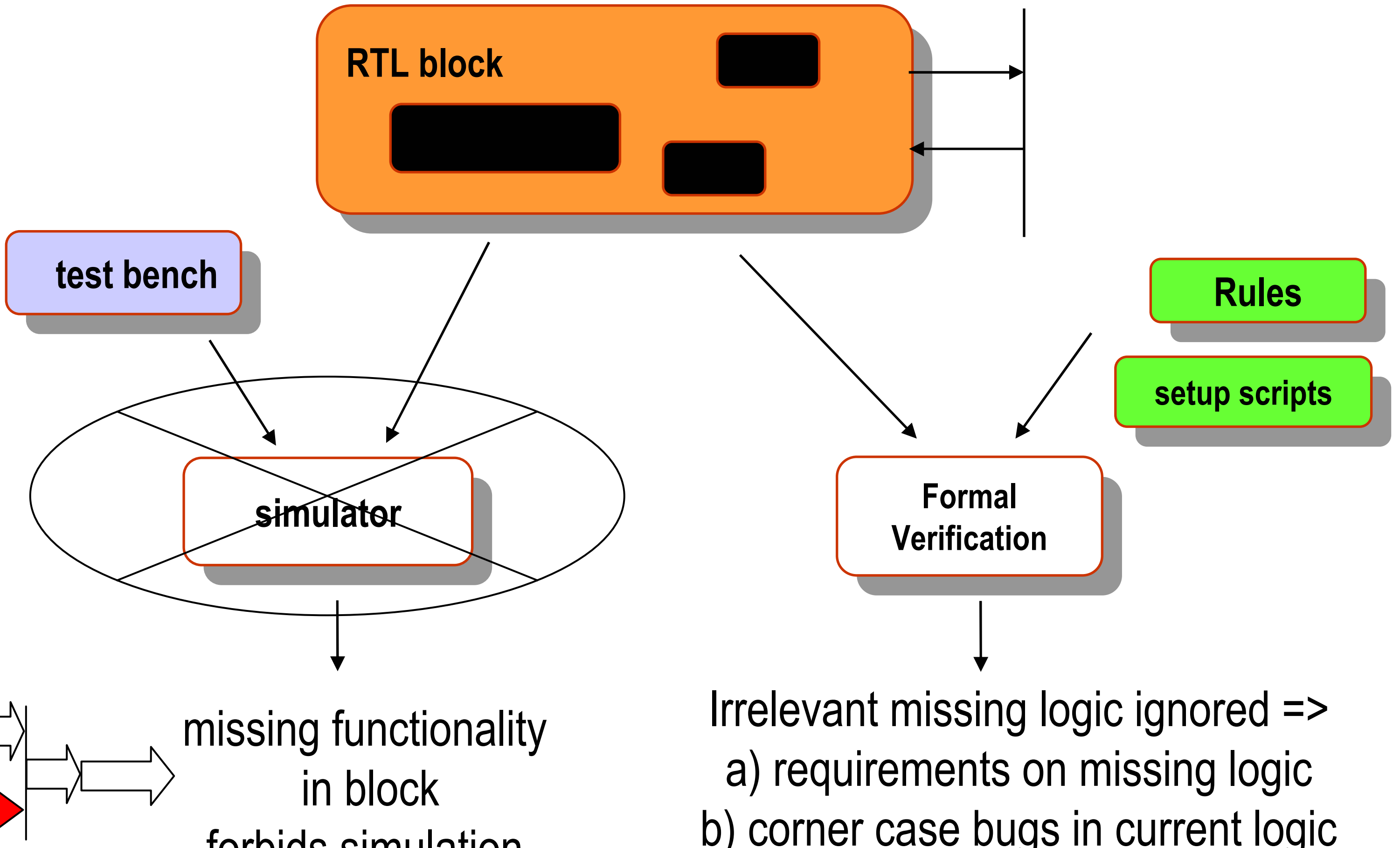
Second Pair of Eyes





Requirement development in parallel to RTL developments
- Standard Interface : purchased
- Proprietary Interface : internal CAD
- Block-level requirements : architect / verification engineer

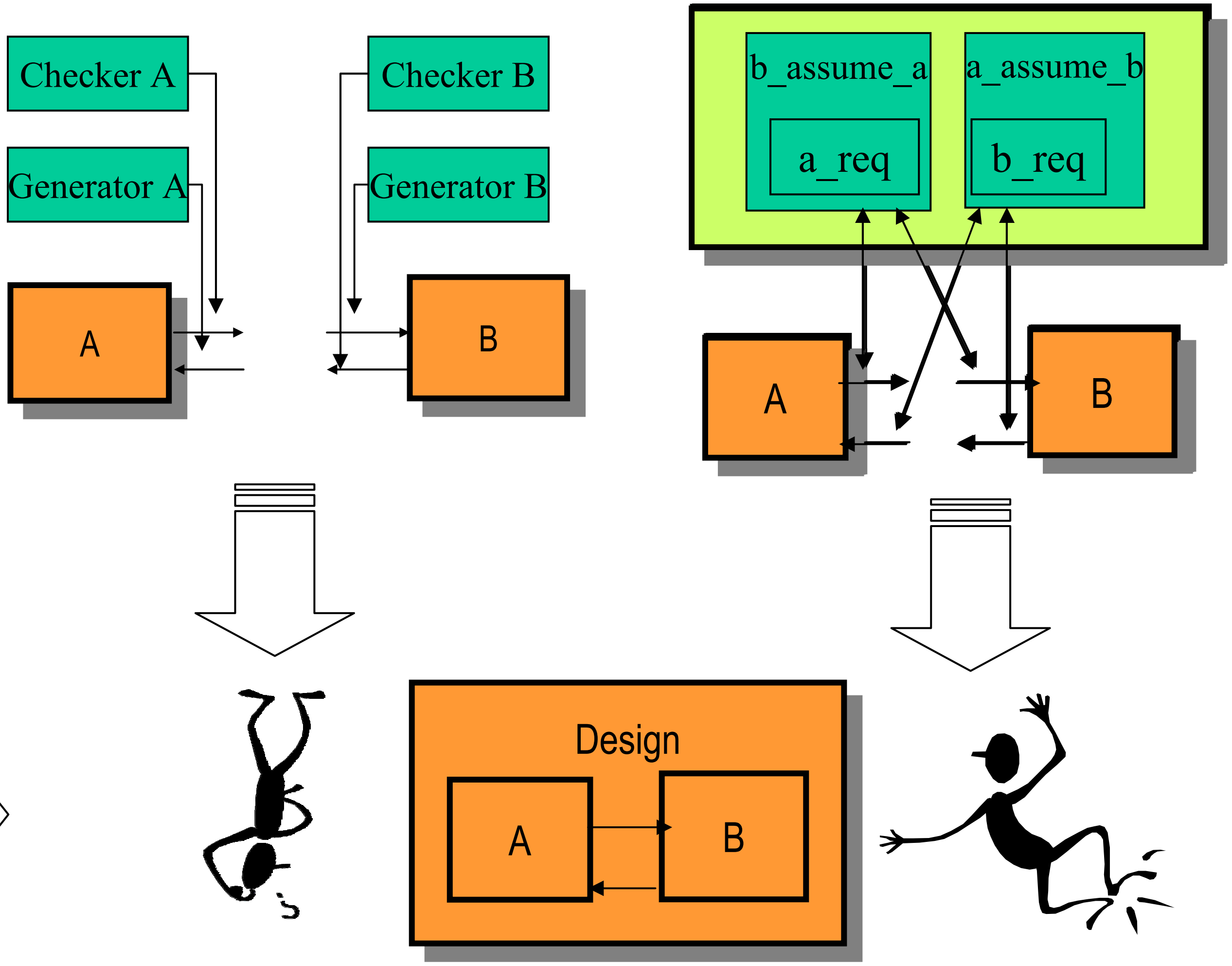
Serious Early Debugging

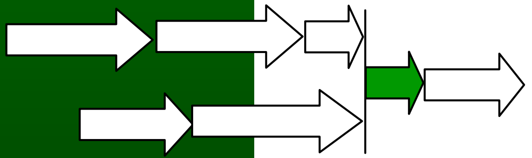


missing functionality in block forbids simulation

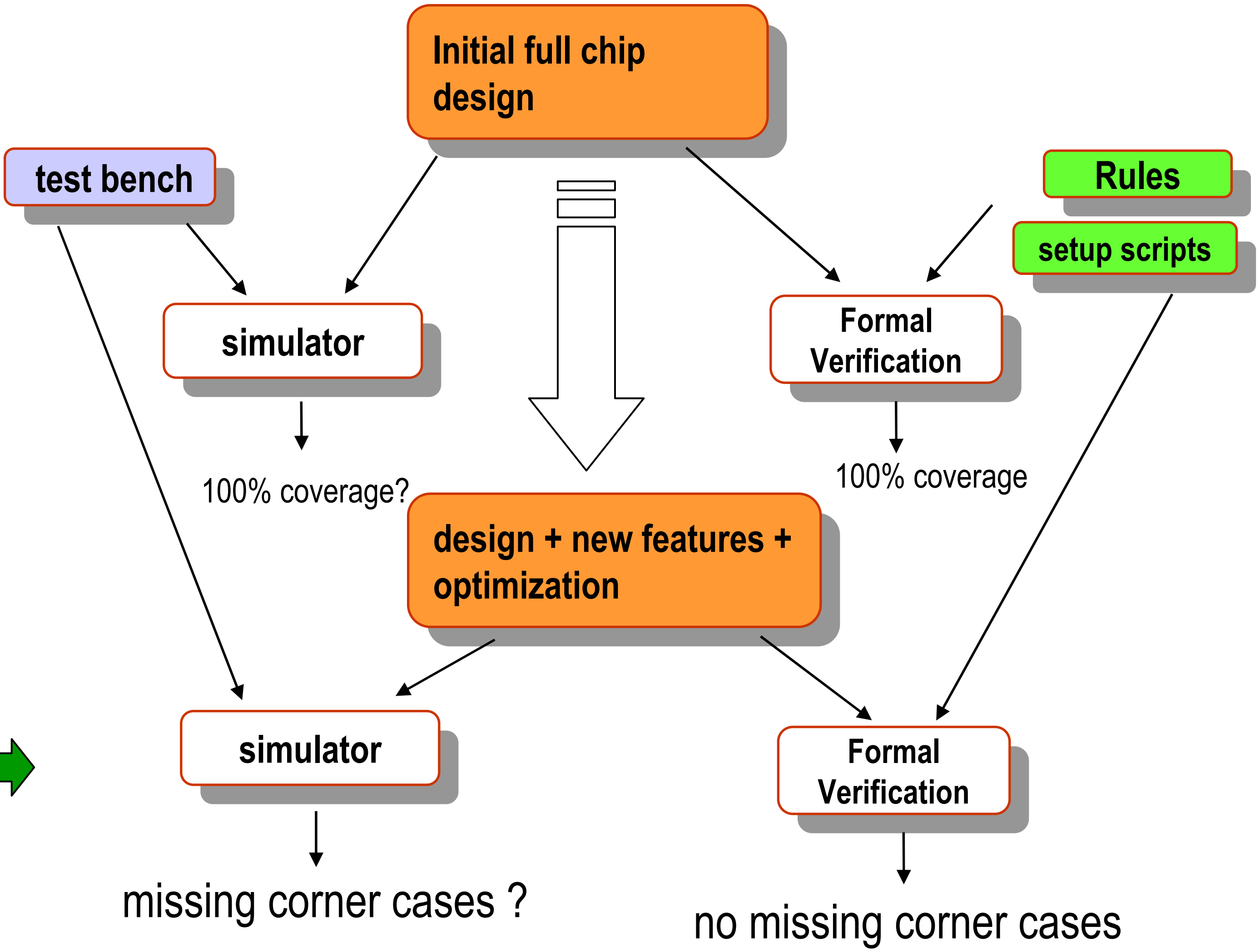
Irrelevant missing logic ignored =>
a) requirements on missing logic
b) corner case bugs in current logic
c) requirements on other blocks

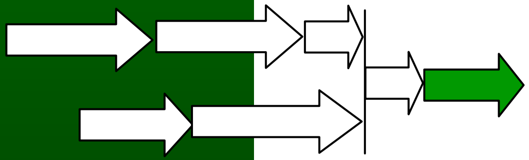
Robust Unit Tests => Easy Integration





Formal Regression => Resistance to New Bugs

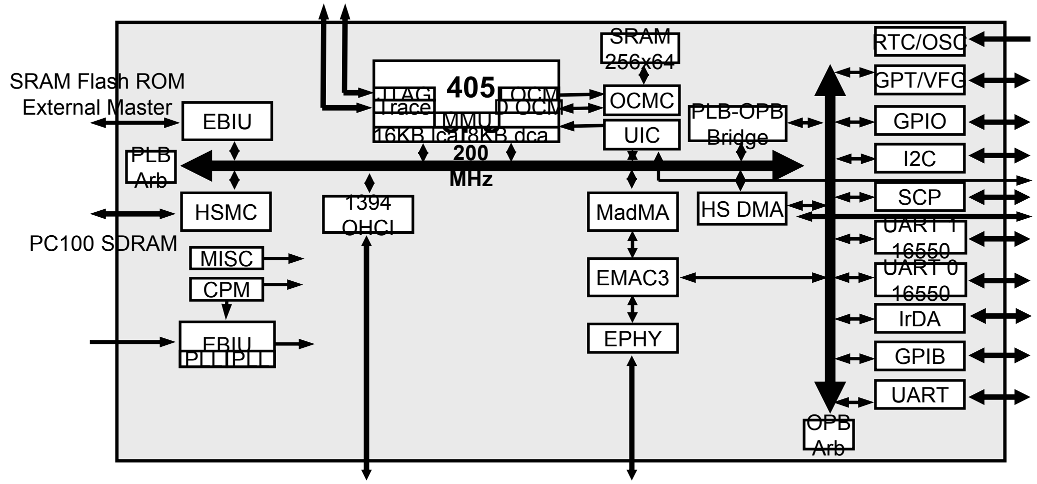




Formal Requirement Verification

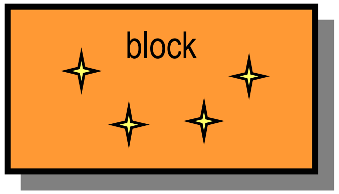
SoC consists of blocks and interfaces

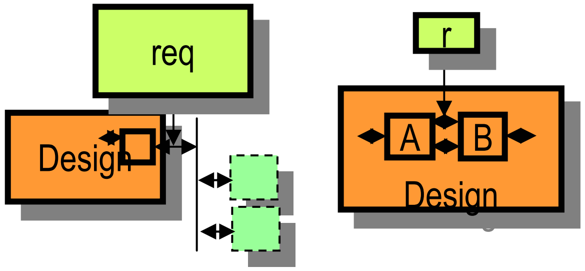
- Formal Interface Verification
  - the blocks understand one another
- Formal Functional Verification
  - the blocks generate the right data

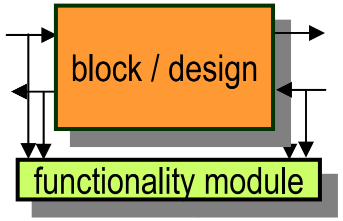


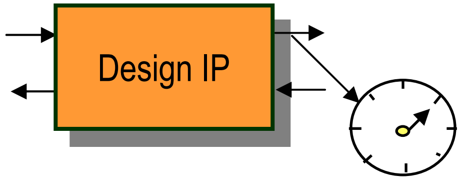
Design Requirements

- Embedded Assertions
  - document designer's decision
- Interface Requirements
  - exchange designers' assumption
  - avoid misunderstanding of spec
  - avoid interoperability problem
- Functional Requirements
  - corner cases in how data is transformed
  - corner cases in how data flows through the design
- System Requirements
  - throughput, error rates, drop rates

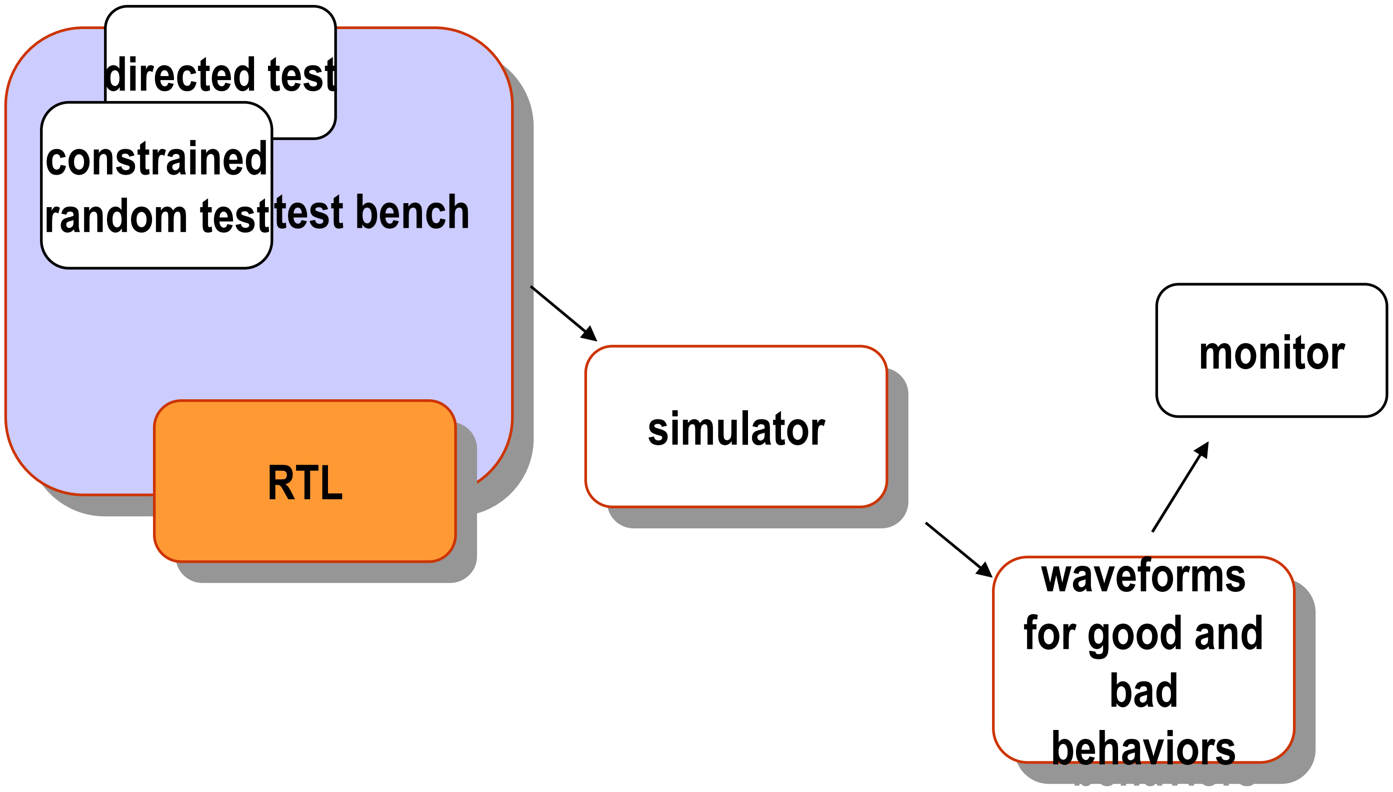




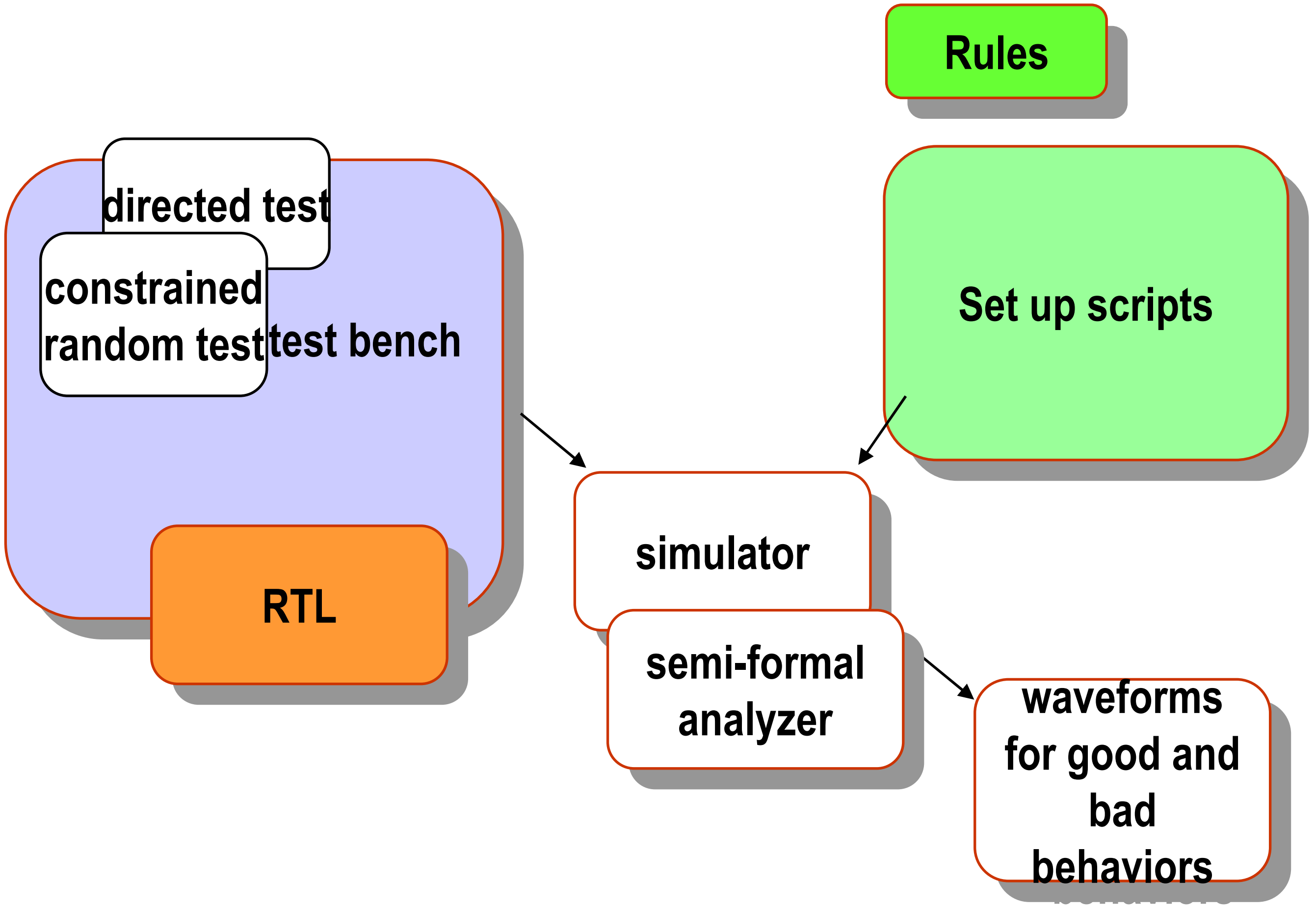




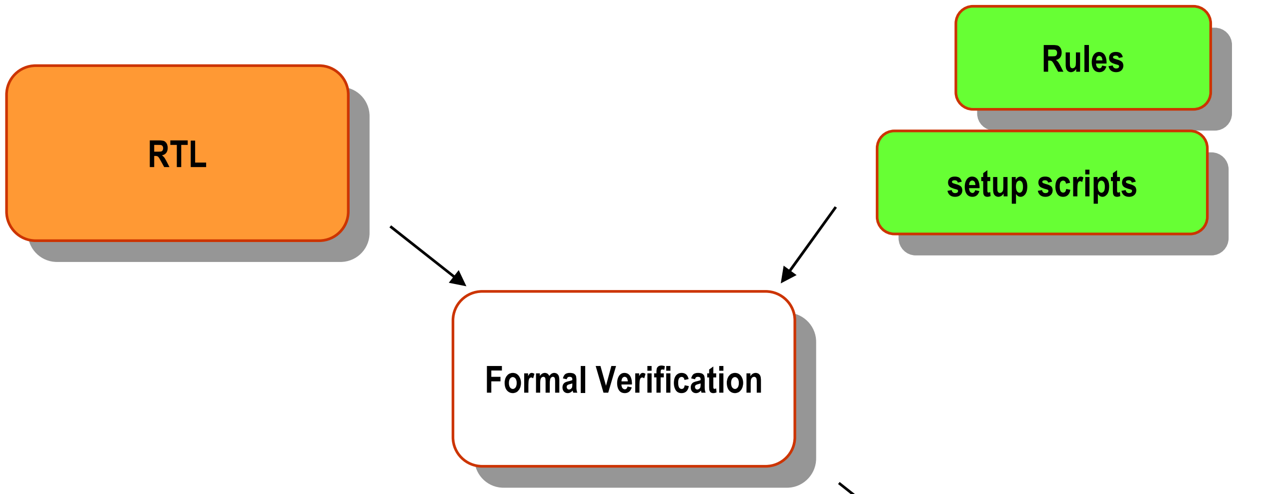
Simulation

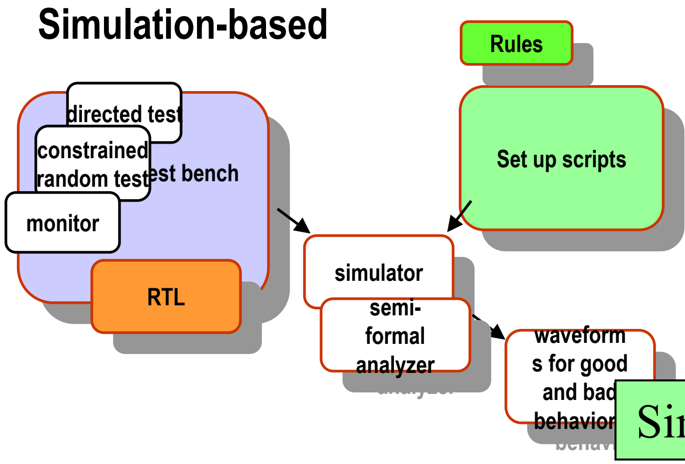

directed test
constrained random test
test bench
RTL
simulator
waveforms for good and bad behaviors
monitor

Semi-formal


Rules
directed test
constrained random test
test bench
Set up scripts
simulator
semi-formal analyzer
RTL
waveforms for good and bad behaviors

Formal Verification





Simulation never achieve 100% coverage