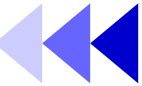
Chapter 4
IP Core Design, Modeling and Verification

C. W. Jen 任建葳

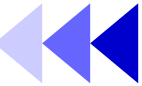
cwjen@twins.ee.nctu.edu.tw

Outline



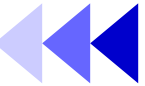
- IP Core Designs
- IP Core Verification
- IP Core Modeling and Deliverables
- System-Level Verification

Outline



- **IP Core Designs**
- IP Core Verification
- IP Core Modeling and Deliverables
- System-Level Verification

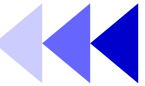
Problem in SoC Era



- Productivity gap
- Time-to-market pressure
- Increasing design complexity
  - HW/SW co-development
  - System-level verification
  - Integration on various levels and areas of expertise
  - Timing closure due to deep submicron

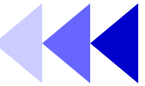
Solution: Platform-based design with reusable IPs

Design for Reuse IPs



- Design to maximize the flexibility
  - configurable, parameterizable
- Design for use in multiple technologies
  - synthesis script with a variety of libraries
  - portable for new technologies
- Design with complete verification process
  - robust and verified
- Design verified to a high level of confidence
  - physical prototype, demo system
- Design with complete document set

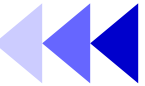
Parameterized IP Design



- Why to parameterize IP?
  - Provide flexibility in interface and functionality
  - Facilitate verification
- Parameterizable types
  - Logic/Constant functionality
  - Structural functionality
    - Bit-width、depth of FIFO、regulation and selection of sub-module
  - Design process functionality (mainly in test bench)
    - Test events
    - Events report (what, when and where)
    - Automatic check event
  - Others* (Hardware component Modeling, 1996)

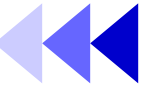
*Authors: Vicktor Preis and Sabine Marz-Rossel, Modeling Highly Flexible and Self-generating Parameterizable Components In VHDL
Collected in book "Hardware component Modeling", 1996, by Jean-Michel Berge, Oz Levia and Jacques Rouillard

IP Generator/Compiler



- User specifies
  - Power dissipation, code size, application performance, die size
  - Types, numbers and sizes of functional unit, including processor
  - User-defined instructions.
- Tool generates
  - RTL code, diagnostics and test reference bench
  - Synthesis, P&R scripts
  - Instruction set simulator, C/C++ compiler, assembler, linker, debugger, profiler, initialization and self-test code

Logic/Constant Functionality



- Logic Functionality
  - Synthesizable code

```
always @(posedge clock) begin
    if (reset==`ResetLevel) begin
      …
    end
    else begin
      …
    end
end
```

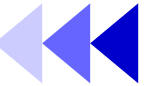
- Constant Functionality
  - Synthesizable code

```
assign tRC_limit=
  (`RC_CYC > (`RCD_CYC + burst_len)) ?
  `RC_CYC - (`RCD_CYC + burst_len) : 0;
```

  - For test bench

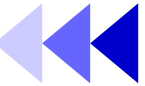
```
always #(`T_CLK/2) clock = ~clock;
…
initial begin
#(`T_CLK) event_1;
#(`T_CLK) event_2;
…
end
```

Reusable Design - Test Suite

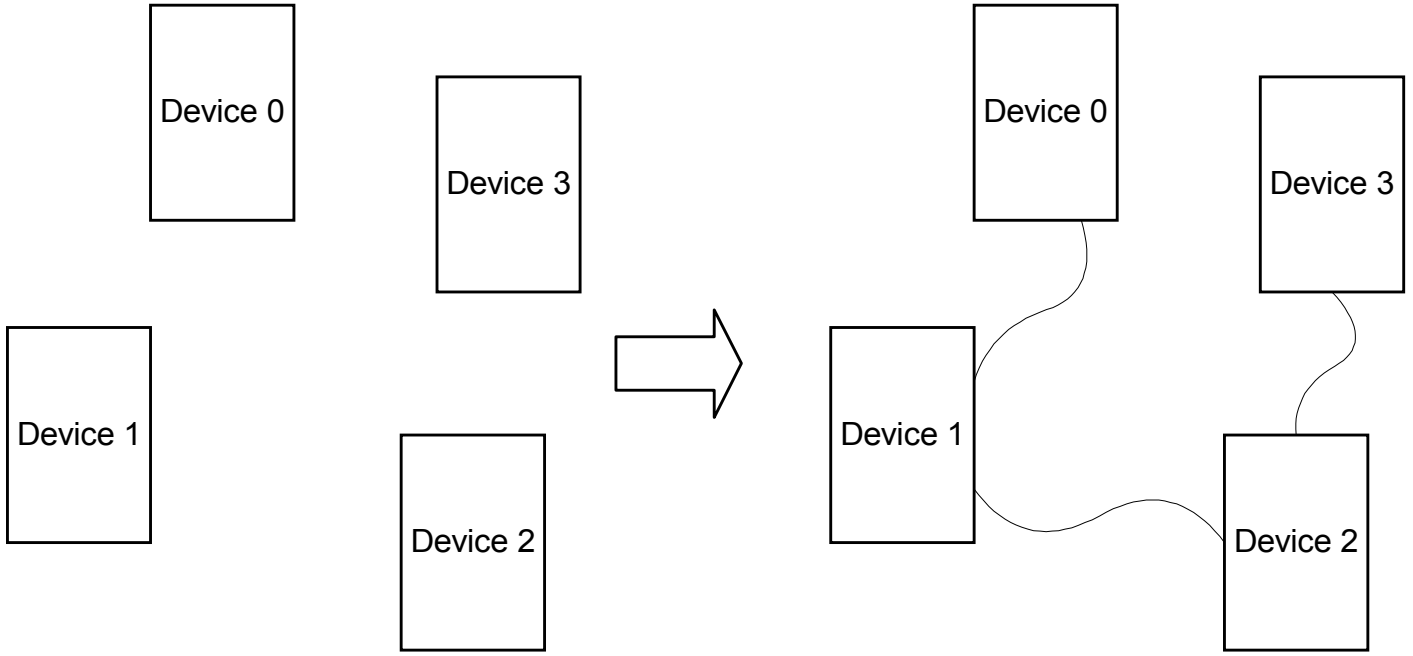


- Test events
  - Automatically adjusted when IP design is changed
  - Partition test events to reduce redundant cases when test for all allowable parameter sets at a time
- Debug mode
  - Test for the specific parameter set at a time
  - Test for all allowable parameter sets at a time
  - Test for the specific functionality
  - Step control after the specific time point
- Display mode of automatic checking
  - display[0]: event current under test
  - display[1]: the time error occurs
  - display[2]: expected value and actual value
  - ...

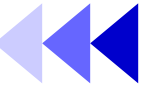
Reusable Design - Test Bench

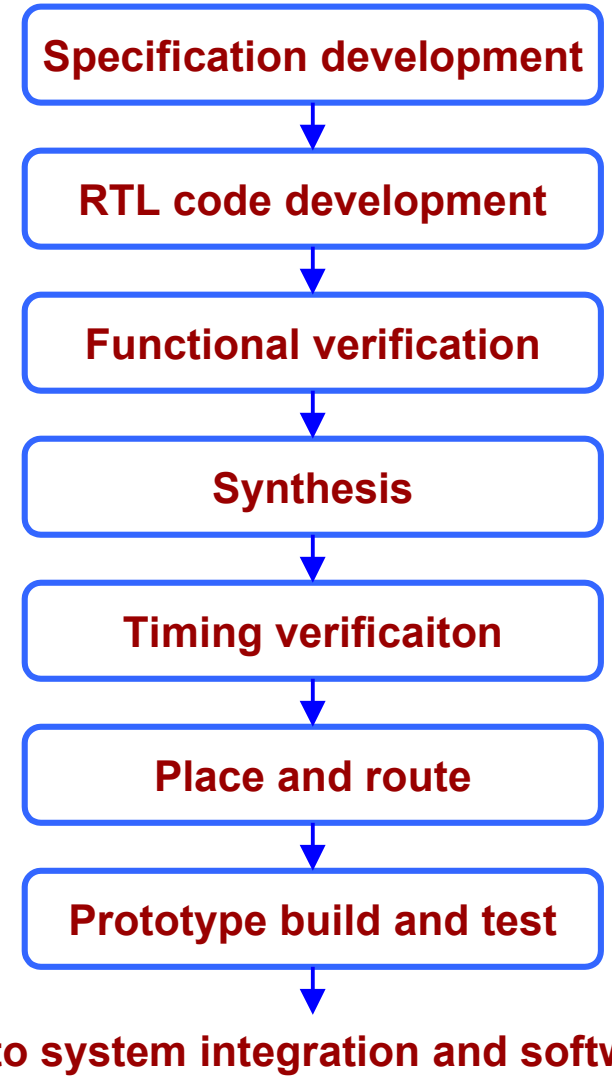


- Use Global Connector to configure desired test bench
  - E.g.: bus topology of IEEE 1394



Traditional ASIC Design Flow

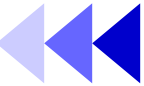




Hardware and software development are mostly serialized

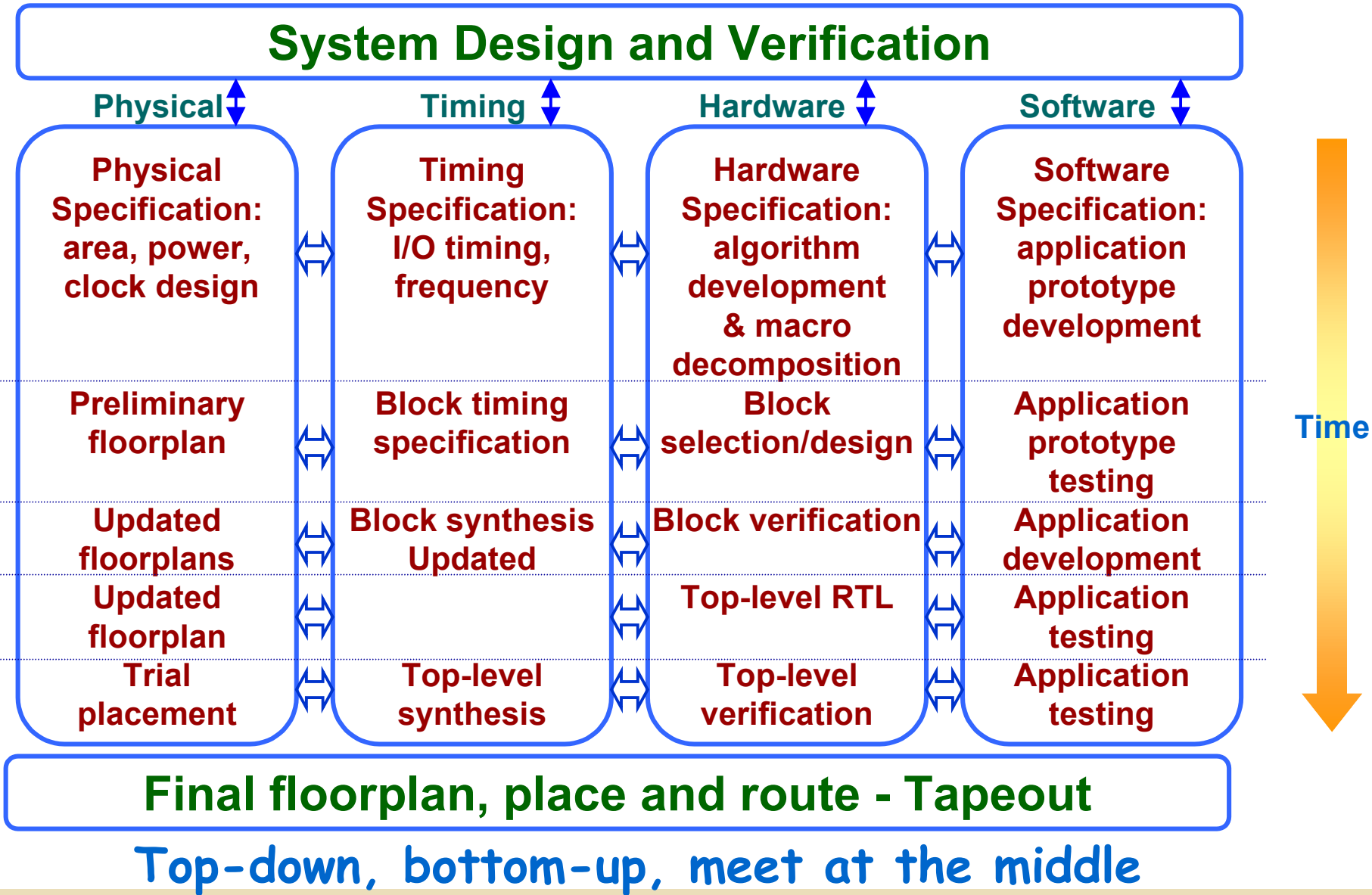
Source: Michael Keating and Pierrr Bricaud, Reuse Methodology Manual, 2nd ed. 1999.

SoC Design Flow Characteristics



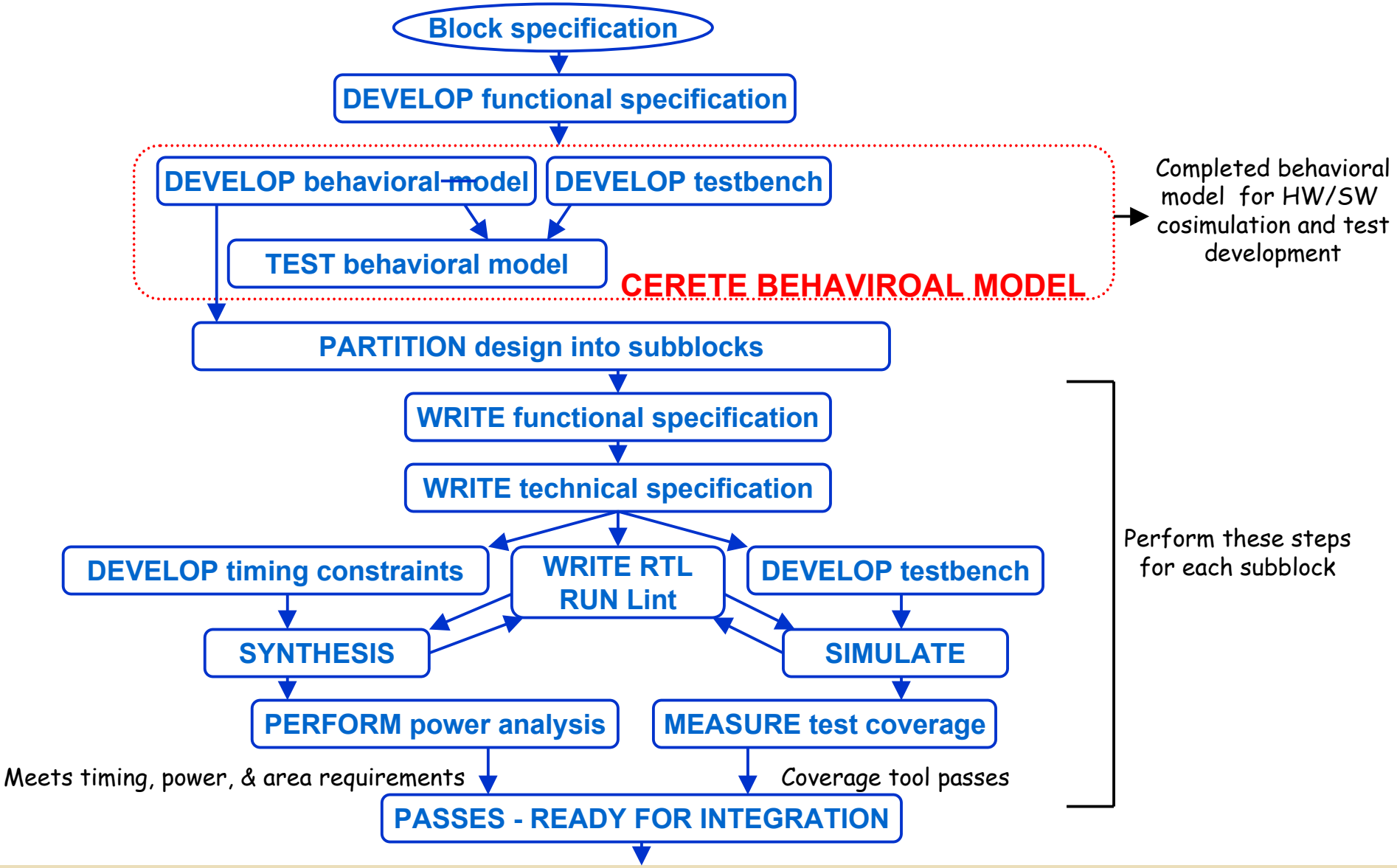
- Parallel, concurrent development of hardware and software
- Parallel verification and synthesis of modules
- Floorplanning and place-and-route included in the synthesis process

Spiral SoC Design Flow



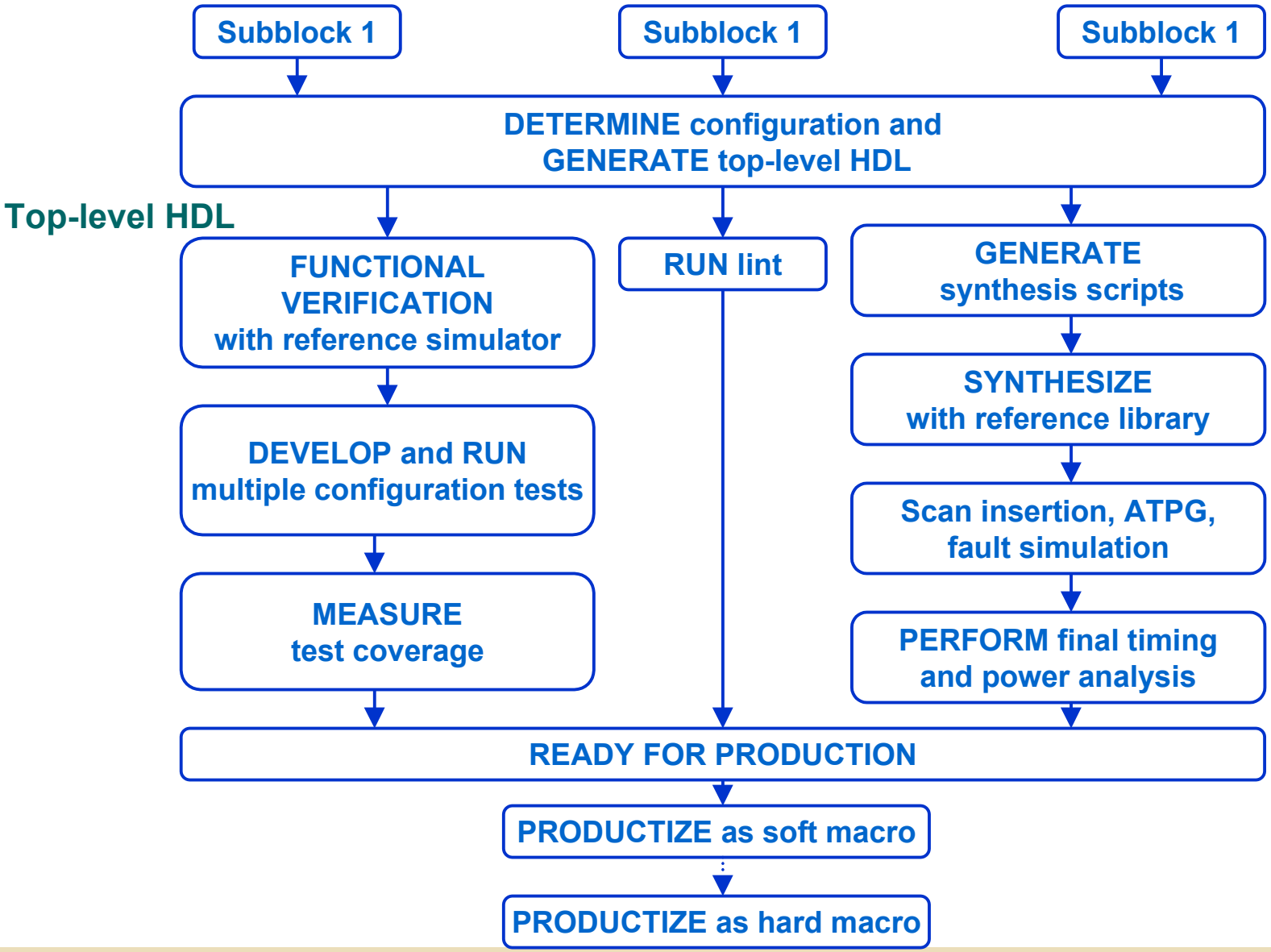
Source: Michael Keating and Pierrr Bricaud, Reuse Methodology Manual, 2nd ed. 1999.

IP Core Macro Design Process



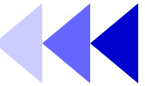
Source: Michael Keating and Pierrr Bricaud, Reuse Methodology Manual, 2nd ed. 1999.

Macro Integration Process



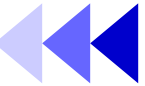
Source: Michael Keating and Pierrr Bricaud, Reuse Methodology Manual, 2nd ed. 1999.

Four Major Phases



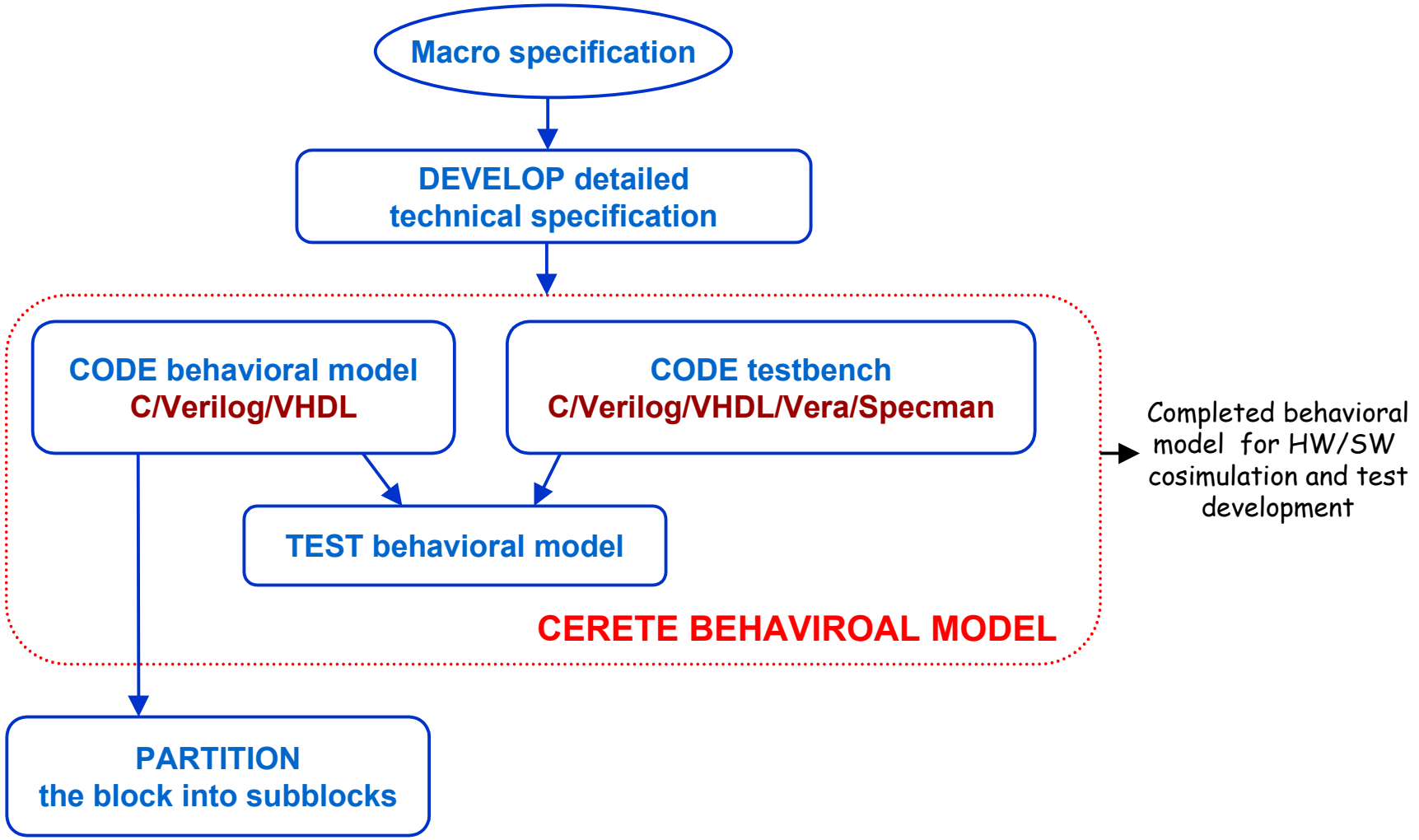
- Design top-level macro
  - macro specification; behavior model
  - macro partition
- Design each subblock
  - specification and design
  - testbench; timing, power check
- Integration subblocks
- Macro procuctization

Specification at Every Level



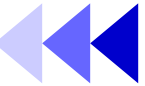
- Overview
- Functional requirements
- Physical requirements
- Design requirements
- Block diagram
- Interface to external system
- Manufacturing test methodology
- Software model
- software requirement
- Deliverables
- Verification

Top-Level Macro Design Flow



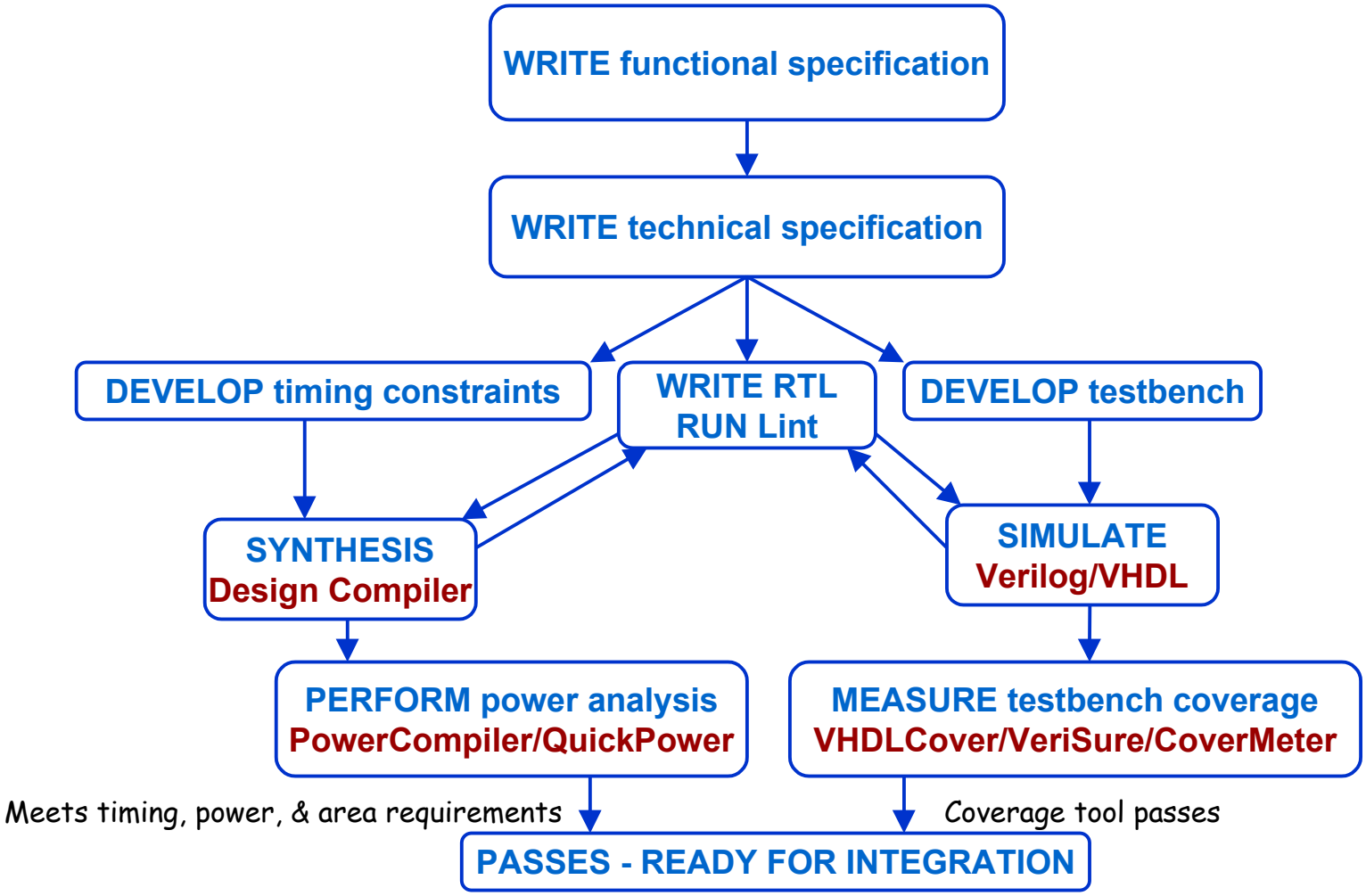
Source: Michael Keating and Pierrr Bricaud, Reuse Methodology Manual, 2nd ed. 1999.

Top-Level Macro Design

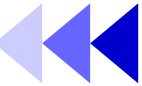


- Updated macro hardware specification
  - document
- Executable specification
  - language description
  - external signals, timing
  - internal functions, timing
- Behavioral model
  - SystemC, HDL
- Testbench
  - test vector generation, model for under test unit, monitoring and report
- Block partition

Subblock Design Flow

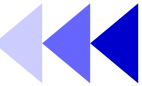


Subblock Design



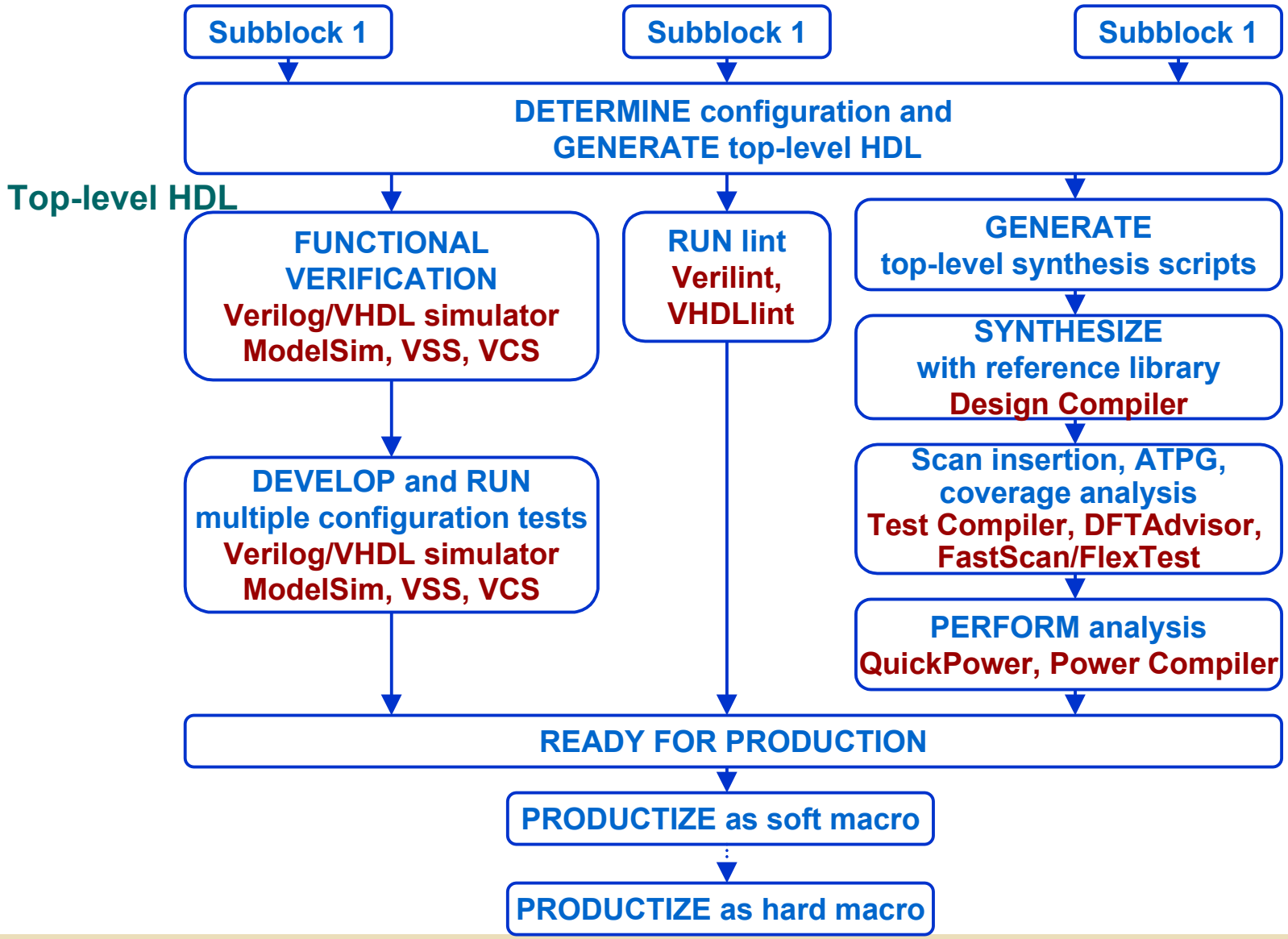
- Design elements
  - Specification
  - Synthesis script
  - Testbench
  - Verification suite
  - RTL that pass lint and synthesis

Linter

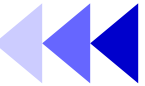


- Fast static RTL code checker
  - preprocessor of the synthesizer
  - RTL purification
    - syntax, semantics, simulation
  - timing check
  - testability checks
  - reusability checks
- Shorten design cycle by avoiding lengthy iterations

Subblock Integration Flow

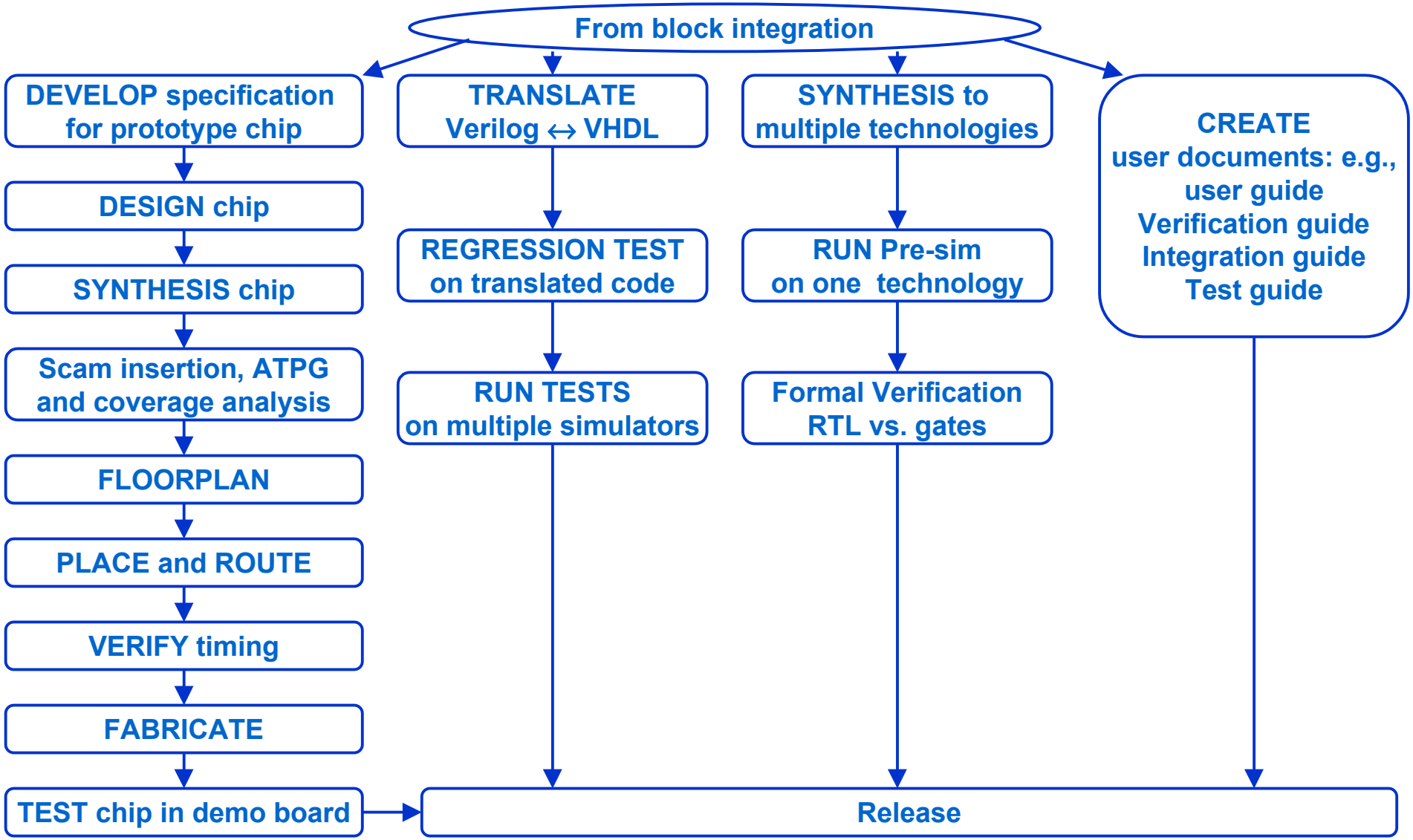


Subblock Integration

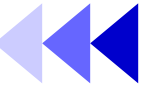


- Integration process is complete when
  - top-level RTL, synthesis script, testbench complete
  - macro RTL passes all tests
  - macro synthesizes with reference library and meets all timing, power and area criteria
  - macro RTL passes lint and manufacturing test coverage

Macro Productization

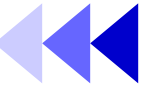


Soft Macro Production



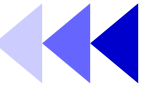
- Produce the following components
  - Verilog version of the code, testbenches, and tests
  - Supporting scripts for the design
    - installation script
    - synthesis script
  - Documentation

Principles of RTL Coding Styles



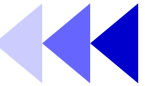
- Readability
- Simplicity
- Locality
- Portability
- Reusability
- Reconfigurability

Naming Conventions



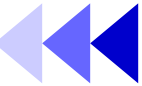
- Lowercase letters for signal names
- Uppercase letters for constants
- Case-insensitive naming
- Use *clk* for clocks, *rst* for resets
- Suffixes
  - *_n* for active-low, *_a* for async, *_z* for tri-state, …
- Identical names for connected signals and ports
- Do not use HDL reserved words
- Consistency within group, division and corporation

File Header



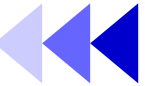
- Should be included for all source files
- Contents
  - author information
  - revision history
  - purpose description
  - available parameters
  - reset scheme and clock domain
  - critical timing and asynchronous interface
  - test structures
- A corporation-wide standard template

Ports



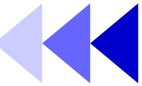
- Ordering
  - one port per line with appropriate comments
  - inputs first then output
  - clocks, reset, enables, orator controls, address bus, then data bus
- Mapping
  - use named mapping instead of positional mapping

Coding Practices



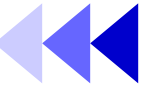
- Little-endian for multi-bit bus
- Operand sizes should match
- Expression in condition must be a 1-bit value
- Use parentheses in complex statements
- Do not assign signals don’t-case values
- Reset all storage elements

Portability



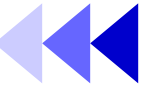
- Do not use hard-coded numbers
- Avoid embedded synthesis scripts
- Use technology-independent libraries
- Avoid instantiating gates

Clocks and Resets

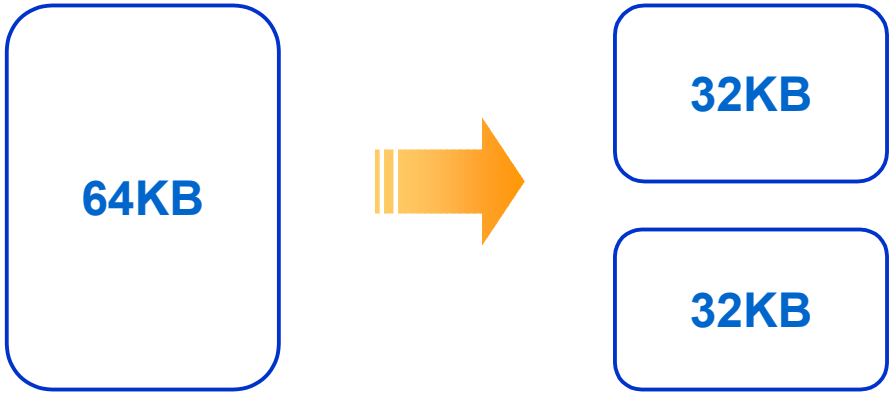


- Simple clocking is easier to understand, analyze, and maintain
- Avoid using both edges of the clock
  - duty-cycle sensitive
  - difficult DFT process
- Do not buffer clock and reset networks
- Avoid gated clock
  - Avoid internally generated clocks and resets
    - limited testability

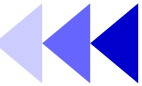
Low Power (1/2)



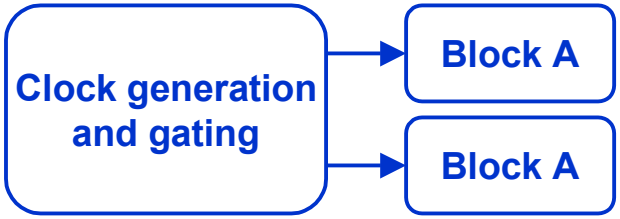
- Memory
  - low-power memory circuit design
  - parathion a large memory into several small blocks
  - gray-coded interface



Low Power (2/2)

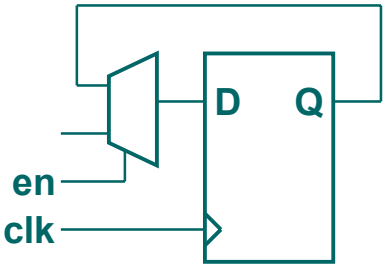


- Clock gating
  - 50% - 70% power consumed in clock network reported
  - gating the clock to an entire block
  - gating the clock to a register



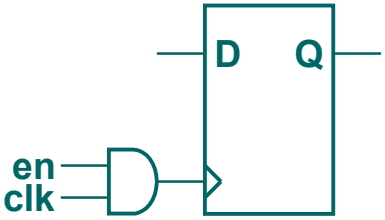
✓

```
always @(posedge clk)
  if (en)
    q <= q_nxt;
```

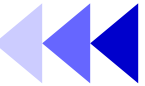


✘

```
Assign clk1 = clk & en;
always @(posedge clk1)
  if (en)
    q <= q_nxt;
```

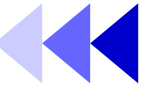


Synchronicity



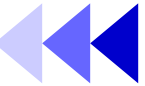
- Infer technology-independent registers
  - (positive) edge-triggered registers
- Avoid latches intentionally
  - except for small memory and FIFO
- Avoid latches unintentionally
  - avoid incomplete assignment in case statement
  - use default assignments
  - avoid incomplete if-then-else chain
- Avoid combinational feedback loops
  - STA and ATPG problem

Combinational and Sequential Blocks



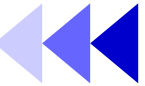
- Combinational block
  - use blocking assignments (= in Verilog)
  - minimize signals required in sensitivity list
  - assignment should be applied in topological order
- Sequential block
  - use non-blocking assignments (<= in Verilog)
  - avoid race problems in simulation
- Comb./Seq. Logic should be separated

Coding for Synthesis (1/2)



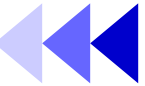
- Specify complete but no redundant sensitivity lists
  - simulation coherence
  - simulation speed
- If-then-else often infers a cascaded encoder
  - inputs signals with different arrival time
- Case infers a single-level MUX
  - case is better if priority encoding is not required
  - case is generally simulated faster than if-then-else
- Conditional assignments
  - infer a MUX, with slower simulation performance

Coding for Synthesis (2/2)



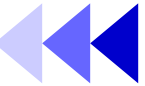
- FSM
  - partition FSM and non-FSM logic
  - partition combinational part and sequential part
  - use parameter to define names of the state vector
  - assign a default (reset) state
- No # delay statements
- Use *full_case* and *parallel_case* judiciously
- Explicitly declare wires
- Avoid glue logic at the top-level
- Avoid expressions in port connections

Partitioning (1/2)

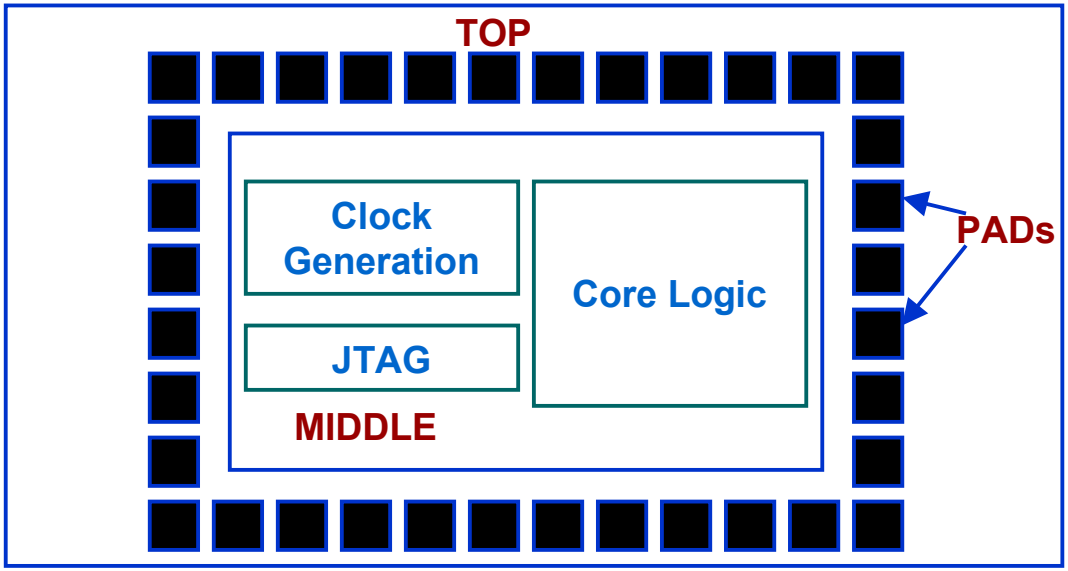


- Register all outputs
  - make output drive strengths and input delay predictable
  - ease time budgeting and constraints
- Keep related logic together
  - improve synthesis quality
- Partition logic with different design goals
- Avoid asynchronous logic
  - technology dependent
  - more difficult to ensure correct functionality and timing
  - as small as possible and isolation
- Keep sharable resources in the same block

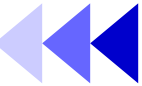
Partitioning (2/2)



- Avoid timing exception
  - point-to-point,false path, multi-cycle path
- Chip-level partitioning
  - level 1: I/O pad ring only
  - level 2: clock generation, analog, memory, JTAG
  - level 3: digital core

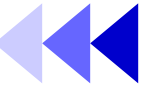


Coding for DFT



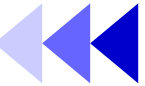
- Avoid tri-state buses
  - bus contention, bus floating
- Avoid internally generated clocks and resets
- Scan support logic for gated clocks
- Clock and set/reset should be fully externally controllable under the test mode

Outline



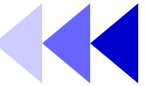
- IP Core Designs
- **IP Core Verification**
- IP Core Modeling and Deliverables
- System-Level Verification

IP Core Verification



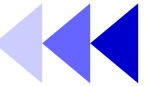
- To ensure the IP macro is 100 percent correct in its functionality and timing.
- Testbench and test suites must be reusable by other teams and compatible with verification tools

Verification Plan (1/2)



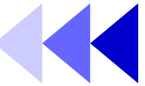
- Develop the verification environment
  - the set of testbench components such as bus functional models, bus monitors, memory models and the structural interconnect of such components with the DUT
- The verification plan include
  - a description of the test strategy, both at the block and the top level
  - a description of the simulation environment, including a block diagram
  - a list of testbench components
  - a list of required verification tools, including simulators and testbench creation tools

Verification Plan (2/2)



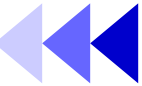
- a list of specific tests, along with the objective and estimated size of each
- an analysis of the key specification of the IP and identification of which tests verify each specification
- a specification of what functionality of the IP will be verified at the block level, and what will be verified at the IP level
- a specification of the target code coverage for each block and for the top-level IP
- a description of the regression test environment and regression procedure

Verification Strategy



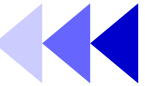
- Macro verification: 3 phases
  - verification of individual subblocks
  - macro verification
  - prototyping
- Basic types of verification tests include
  - compliance test
    - PCI interface, IEEE 1394
    - complies with the specification
  - corner case test
  - random test
  - real code test
  - regression test

Verification Tools



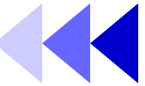
- Simulation
- Testbench automation tools
- Code coverage tools
- Hardware modeling
- Emulation
- Prototyping

Verification Support



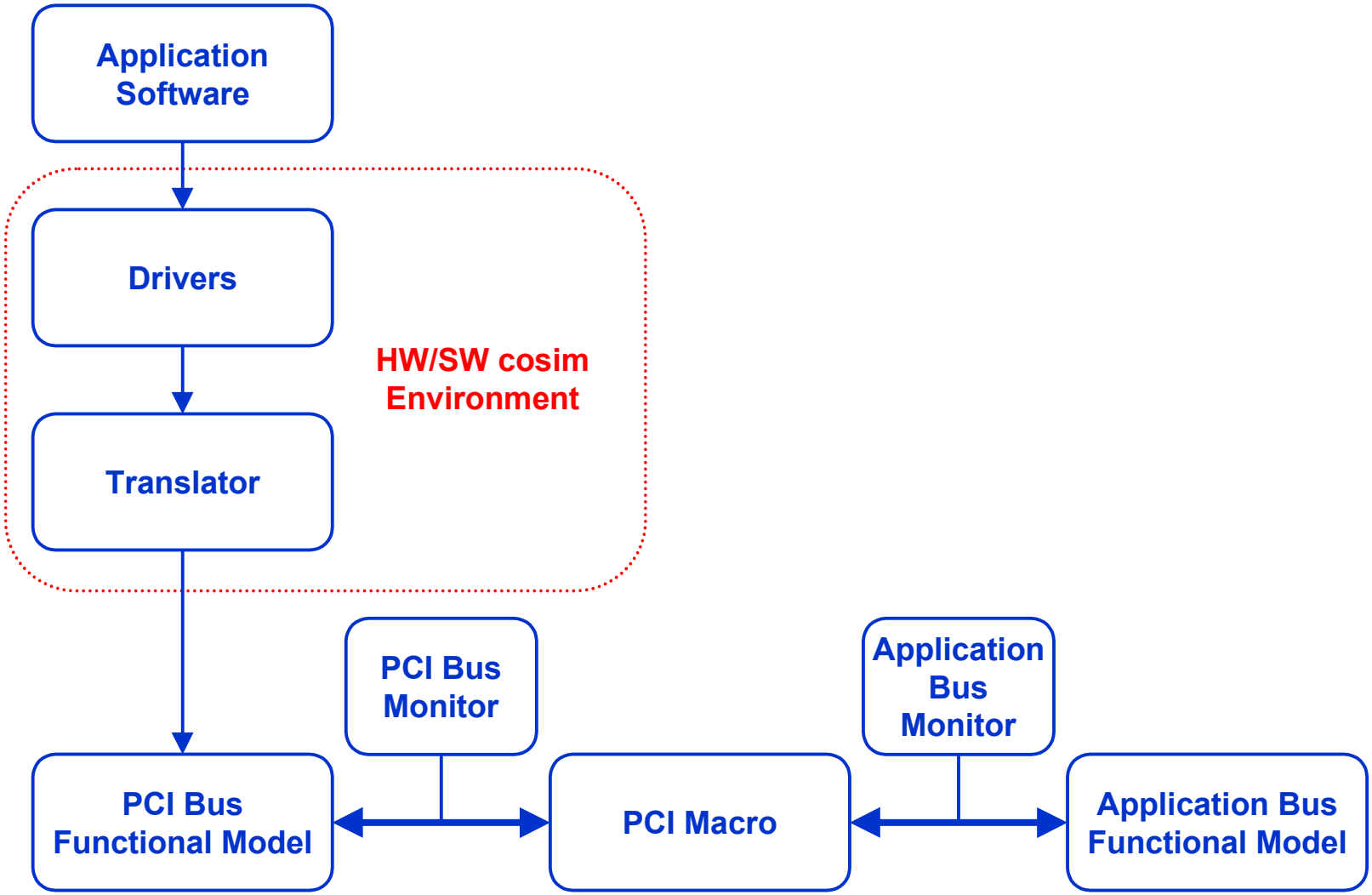
- Protocol Checker
  - Monitor the transactions on an interface and check for any invalid operation
    - Embedded in the test bench
    - Embedded in the design
  - Error and/or warning messing of bus protocol
- Expected results checker
  - Embedded in the test bench
  - Checks the results of a simulation against a previously specified, expected response file.
- Performance monitor
  - Number of transfers, idle cycles...

Testbench Design

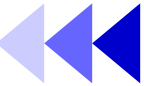


- The testbench design differs depending on the function of the macro
  - microprocessor macro, test program,
  - bus-interface macro, use bus functional models and bus monitors
- Subblock testbench

Macro Testbench

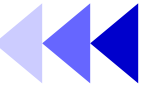


Bus Functional Models (BFM)

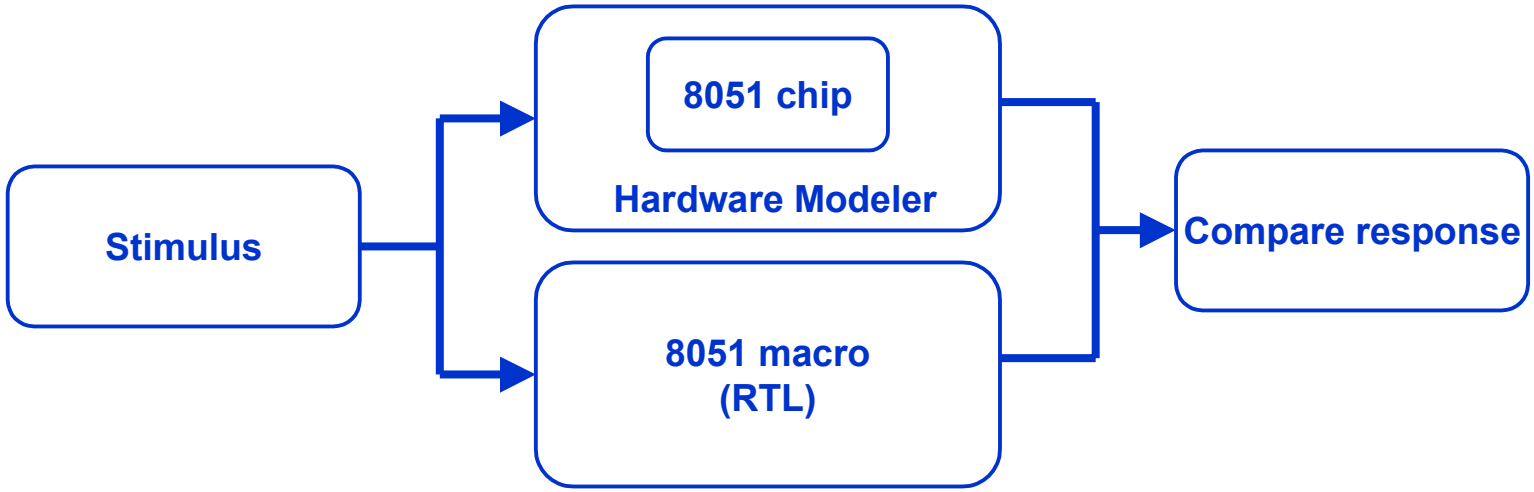


- To model the bus transactions on the bus, each read and write transaction is specified by the test developer.
- BFM is written is RTL, C/C++, or testbench automation tools and uses some form of command language to create sequences of transaction on the bus.
- BFM and monitor must be designed and coded with the same care as the macro RTL, all are deliverables

Automated Response Checking

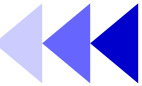


- Compare the output response with a reference design



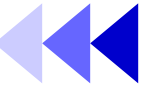
- Bus monitors and checkers
- On-the-Fly checker

Verification Suite Design



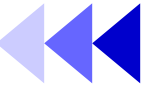
- Once built the testbench , we can develop a set of tests to verify the correct behavior of the macro
- Test sets
  - functional testing
  - corner case testing
  - code coverage
  - random testing

Outline



- IP Core Designs
- IP Core Verification
- **IP Core Modeling and Deliverables**
- System-Level Verification

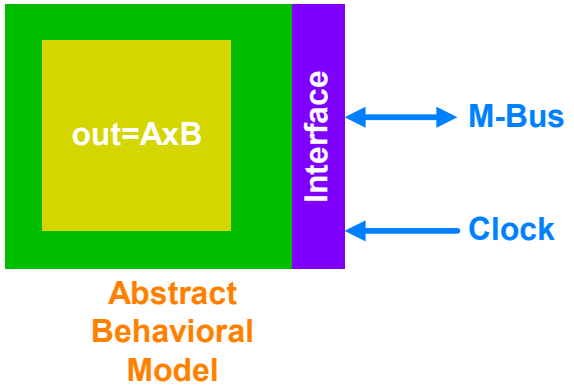
The Intent of Different Level of IP Model

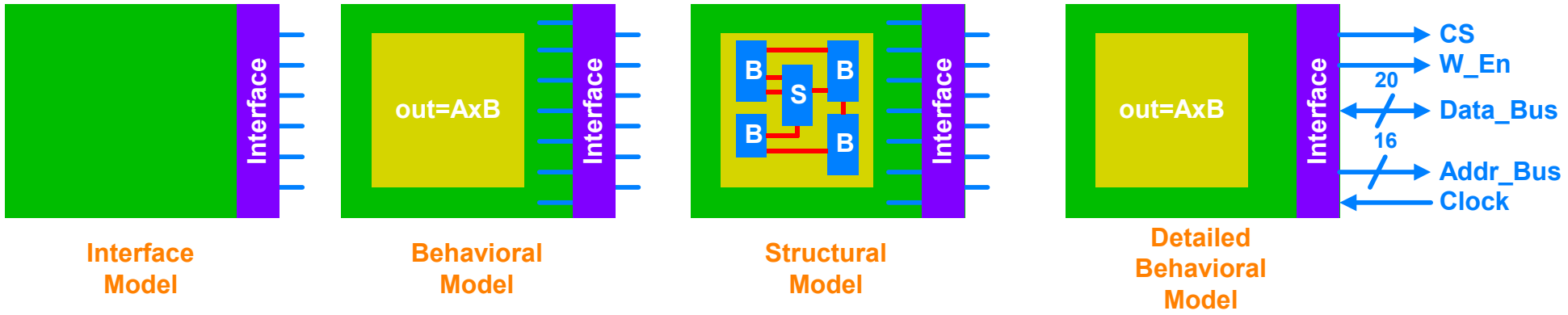


- Design exploration at higher level
  - Import of top-level constraint and block architecture
  - Hierarchical, complete system refinement
  - Less time for validating system requirement
  - More design space of algorithm and system architecture
- Simple and efficient verification and simulation
  - Functional verification
  - Timing simulation/verification
  - Separate internal and external (interface) verification
  - Analysis: power and timing
- Verification support: e.g., monitor, checker...

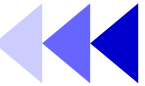
General Modeling Concepts

- Interface model
  - Synonym: bus functional, interface behavioral
- Behavioral model
  - Behavior = function with timing
  - Abstract behavioral model
  - Detailed behavioral model
- Structural model



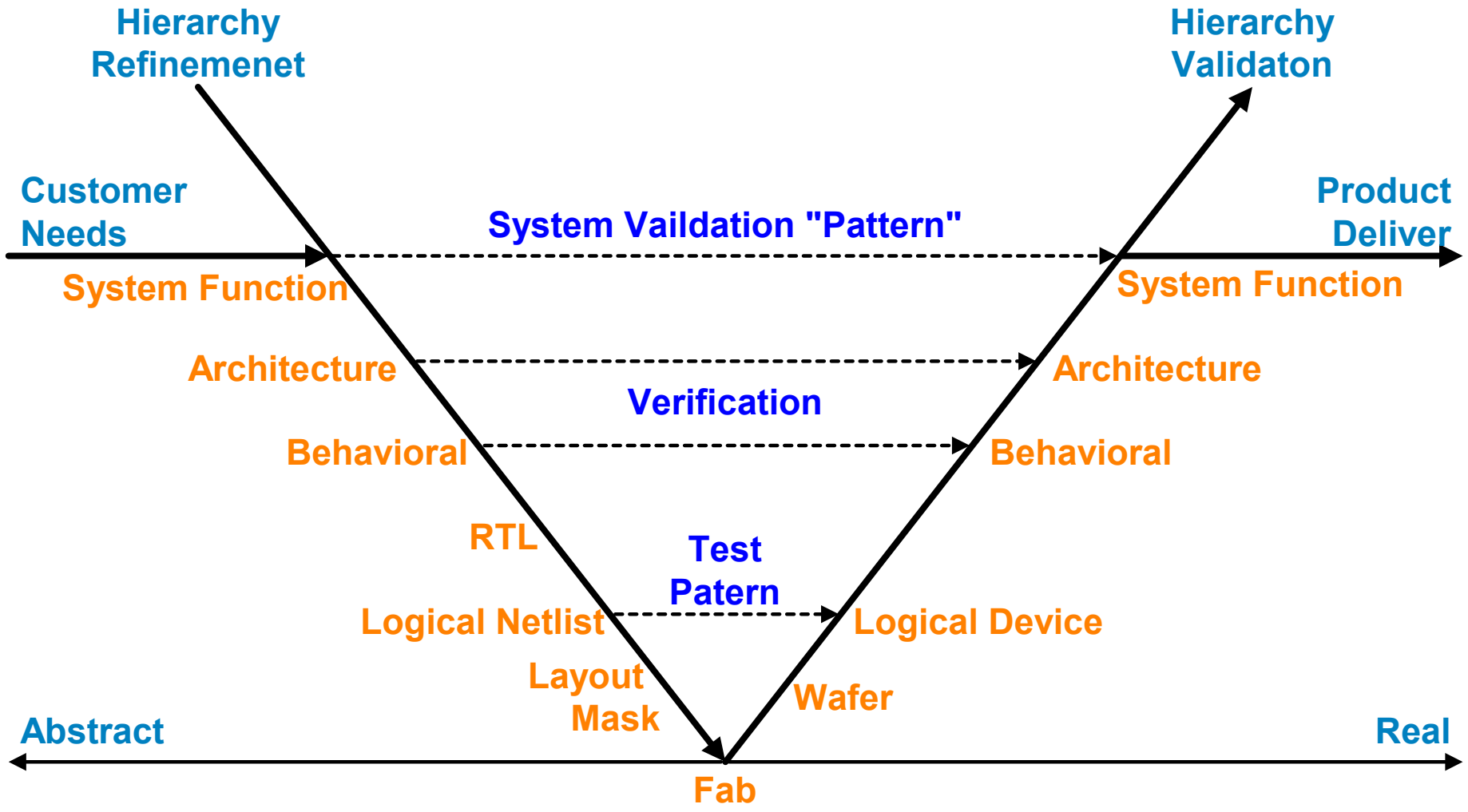


Issues of IP Modeling



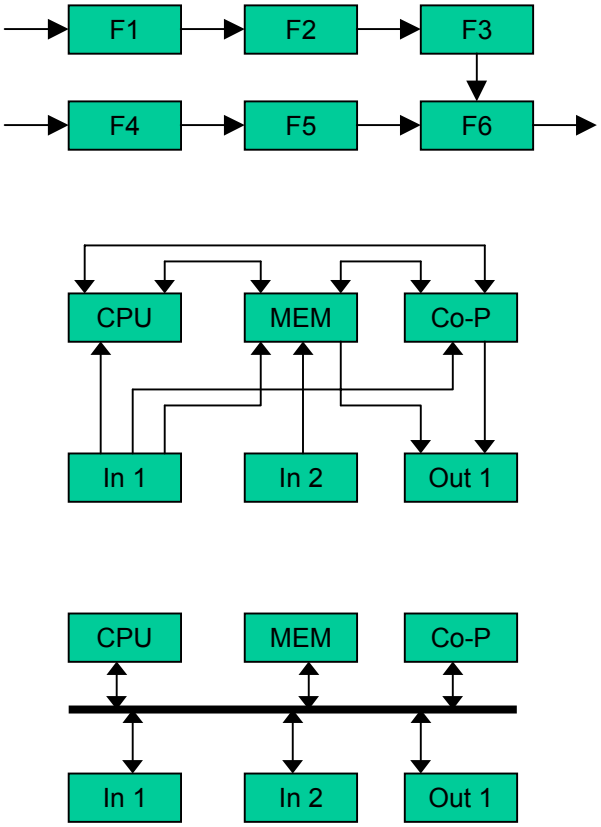
- Attributes
  - What is the sufficient set of model attributes?
  - How are these model attributes validated?
  - How is the proper application of an abstract model specified?
- Two important dimensions of time
  - Model development time is labor intensive: model reusability
  - Simulation time depends upon strategy chosen for mixed domain simulations

From Requirement to Delivery

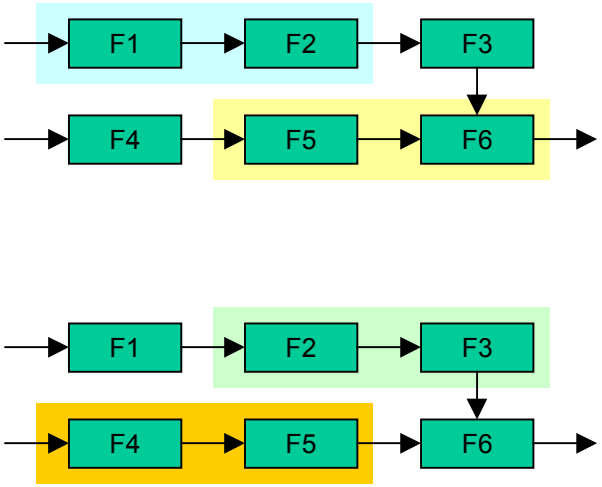


Example: Hierarchical Design Refinement

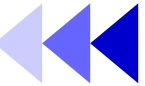
Vertical refinement



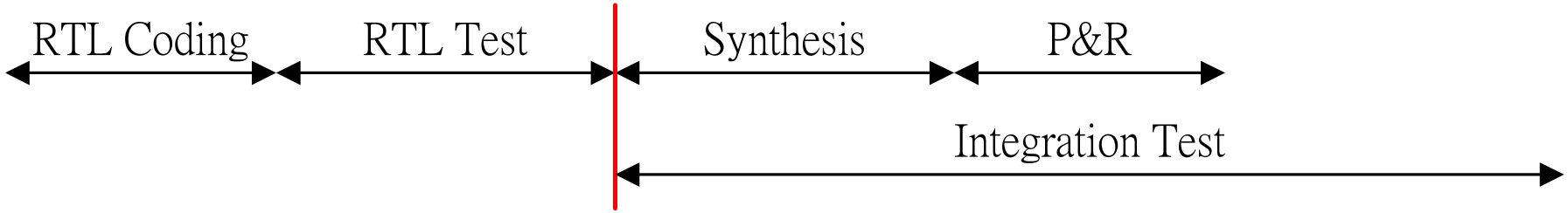
Horizontal refinement: Partition



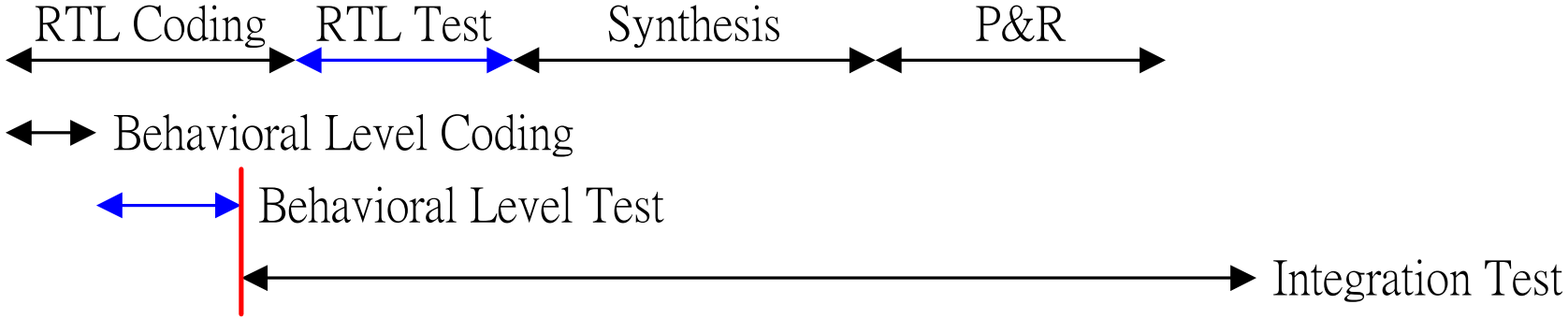
Example: Manage Size and Run-Time



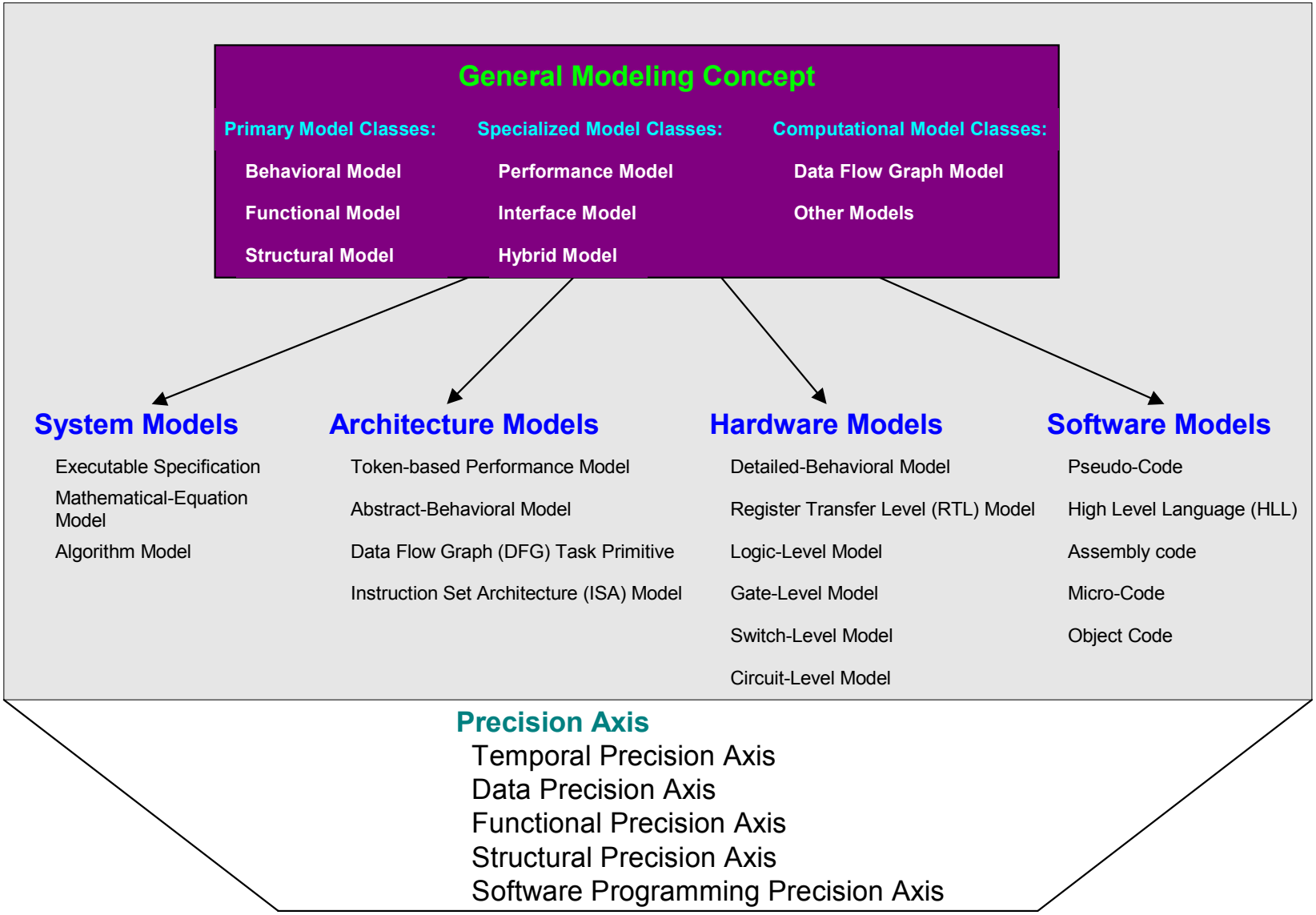
Start at RTL



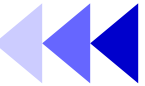
Start at behavioral level



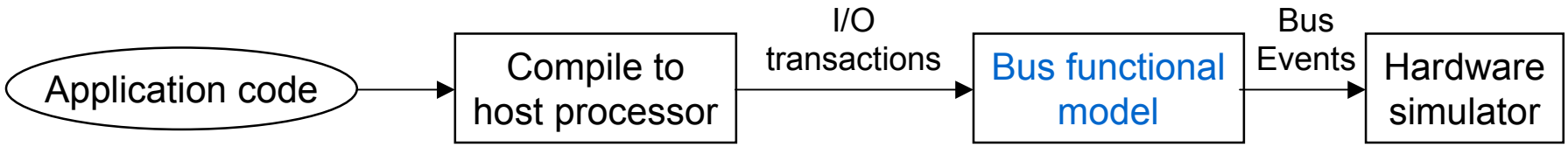
IP Modeling


General Modeling Concept
Primary Model Classes:
Behavioral Model
Functional Model
Structural Model
Specialized Model Classes:
Performance Model
Interface Model
Hybrid Model
Computational Model Classes:
Data Flow Graph Model
Other Models
System Models
Executable Specification
Mathematical-Equation Model
Algorithm Model
Architecture Models
Token-based Performance Model
Abstract-Behavioral Model
Data Flow Graph (DFG) Task Primitive
Instruction Set Architecture (ISA) Model
Hardware Models
Detailed-Behavioral Model
Register Transfer Level (RTL) Model
Logic-Level Model
Gate-Level Model
Switch-Level Model
Circuit-Level Model
Software Models
Pseudo-Code
High Level Language (HLL)
Assembly code
Micro-Code
Object Code
Precision Axis
Temporal Precision Axis
Data Precision Axis
Functional Precision Axis
Structural Precision Axis
Software Programming Precision Axis

CPU Model

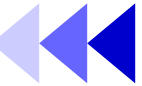


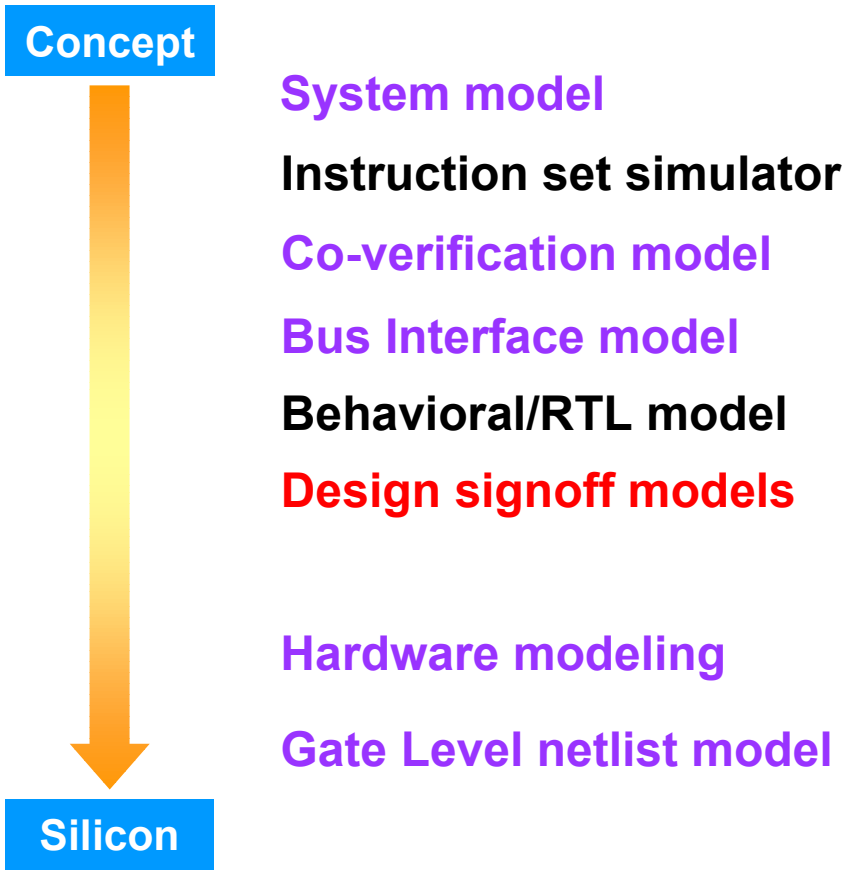
- CPU model enable
  - Estimate software performance
  - Analyze system trade offs
- CPU model
  - Bus functional model



  - Instruction set simulator (SMM)
    - Instruction accurate
    - Cycle accurate
  - Virtual processor model (Cadence VCC technology)

ARM Modeling (1/4)





System model

Instruction set simulators (ISS)

Co-verification model

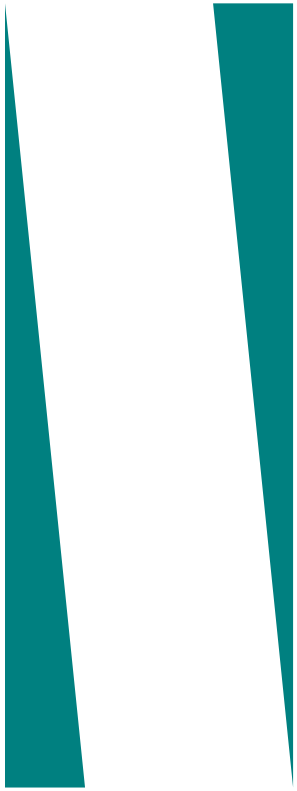
Bus Interface model

Behavioral/RTL model

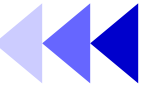
Design signoff models

Hardware modeling

Gate Level netlist model

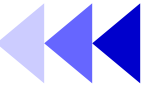


ARM Modeling (2/4)



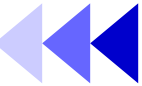
- System Model
  - Provision of customized Software Debugger/ARMulator packages, suitable for dataflow simulation environments.
  - Cadence Signal Processing Worksystem (SPW) and Synopsys COSSAP Stream Driven Simulator
- Co-verification model
  - Each ARM processor core contains a co-verification simulator component and a bus interface model component
  - Co-verification simulator: combines the properties of an advanced ISS with the bus cycle accurate pin information capability required to drive a hardware simulator
  - CoWare N2C Design System, Synopsys Eaglei, to name a few.

ARM Modeling (3/4)



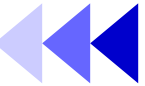
- Bus interface models (BIM)
  - Run a list of bus transactions to stimulate simulated hardware under test
  - Allowing the designer to concentrate on the hardware design without waiting for the ARM control software to be developed.
  - Generated using ModelGen
- Design signoff models
  - Full architectural functionality and full timing accurate simulation
  - Accept process specific timing and back annotated timing
  - Used to ‘sign off’’ design before committing silicon
  - Be compiled 'C' code which enables protection of the inherent IP and superior simulation execution speed over pure HDL models
  - Generated using ModelGen

ARM Modeling (4/4)



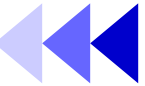
- Hardware Modeling
  - Real chip-based products, based on real silicon
  - For logic and fault simulation
  - Synopsys ModelSource hardware modeling systems
- Fault grading netlist
  - Full custom marcocells yields models suitable for hardware accelerated fault grading, system simulation and emulation
  - Emulator: IKOS, Mentor Graphics and Quickturn; Simulation: IKOS

Intent of ModelGen

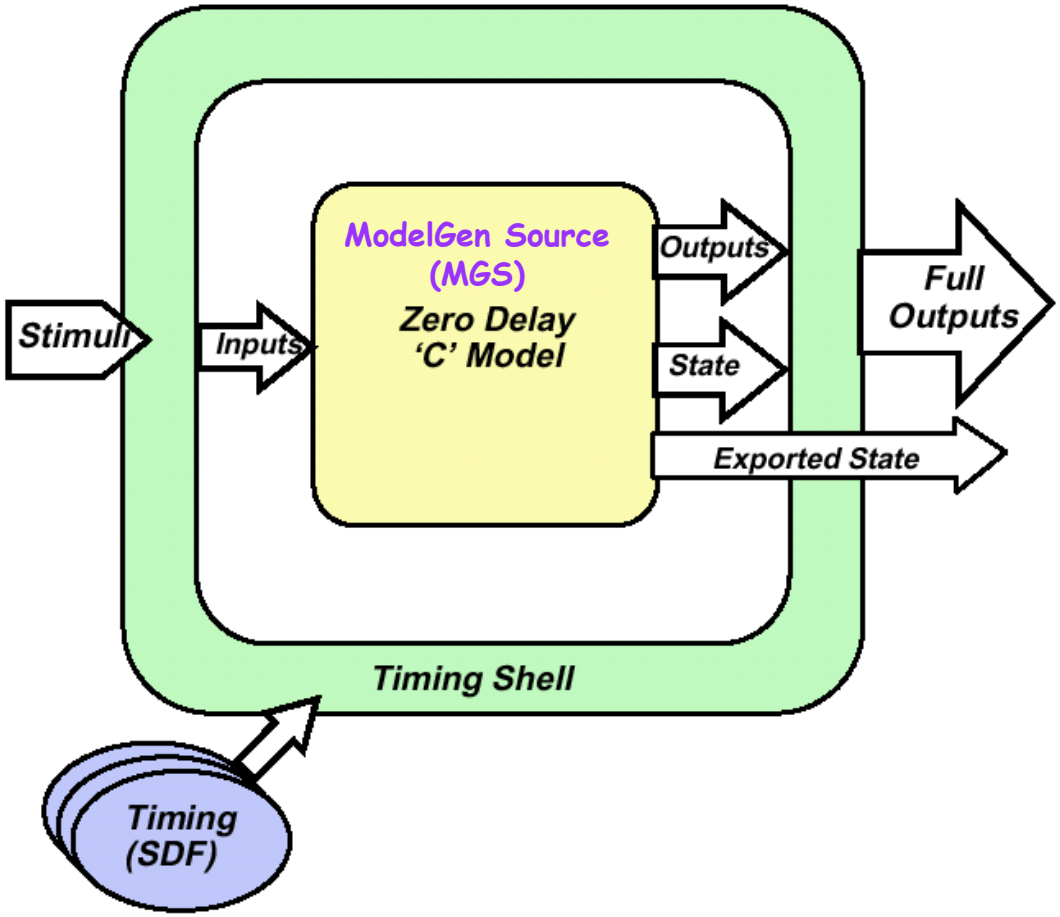


- Key requirements for ARM's modeling environment:
  - Deliver highly secure models
  - Minimize time spent creating, porting and re-verifying models
  - Support mixed-source languages—HDL, C and full custom modeling
  - Support multiple design and verification environments
  - Enable efficient simulation
  - Provide a timing annotation solution that does not compromise IP security

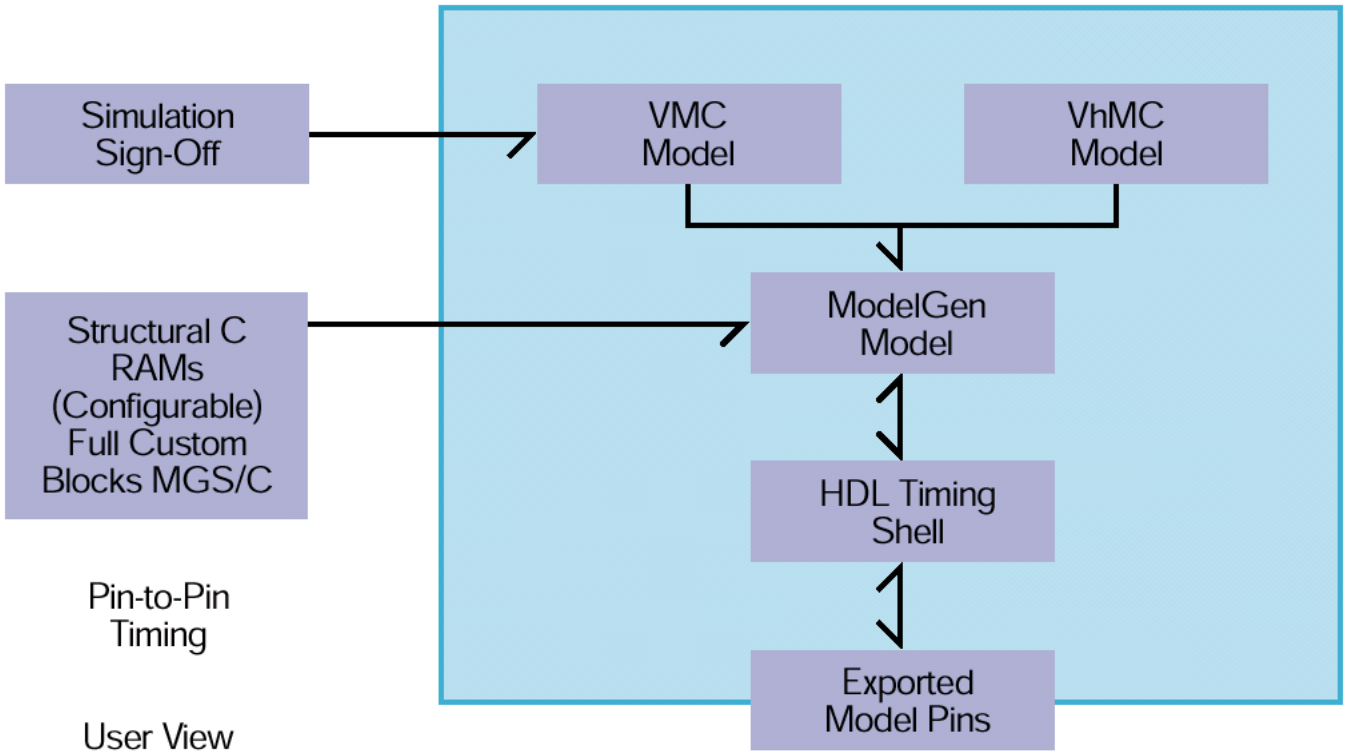
“ModelGen” Timing Shell



- Overview:
  - Black-box model
    - Obscured IP
  - User supplied timing (SDF)
  - Single model
    - Easily verifiable
  - Exported State
  - Programmer model
    - Nine-value Logic/Full
  - Supports checkpointing

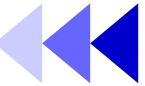


Example of Model Generation Flow

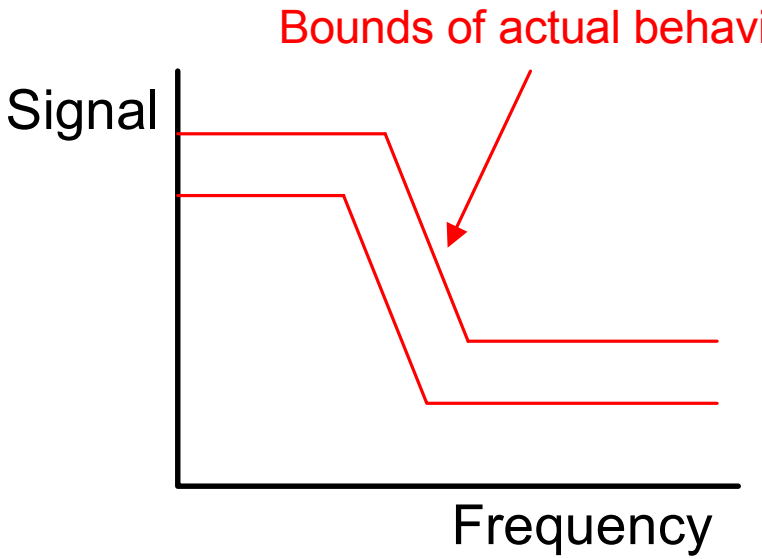


Synopsys VMC/VhMC based model generation flow

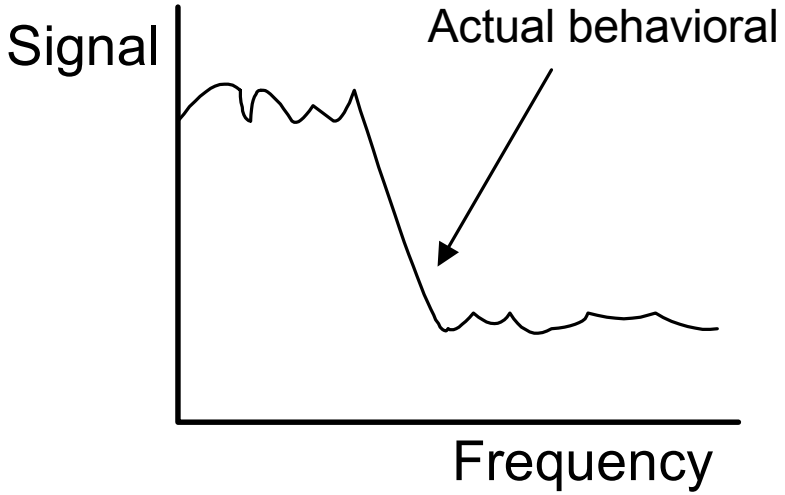
Behavioral Model for A/MS



- Describes the functionality and performance of a VC block without providing actual detailed implementation.
- Needed for system designers to determine the possibility of implementing the system architecture
- It is a kind of *abstract* behavioral model



Behavioral Model

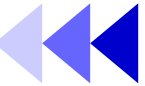


Block Detail Model

Functional/Timing Digital Simulation Model

- Used to tie in functional verification and timing simulation with other parts of the system
- Describes the functionality and timing behavior of the entire A/MS VC between its input and output pins.
- Pin accurate not meant to be synthesizable
- It is a kind of *detailed*-behavioral model
- Example of PLL: represent the timing relationship of reference clock input vs. generate output clock.
  - Model it by actually representing the structure of the PLL, or
  - Model it as just a delay value based on a simple calculation from some parameters.

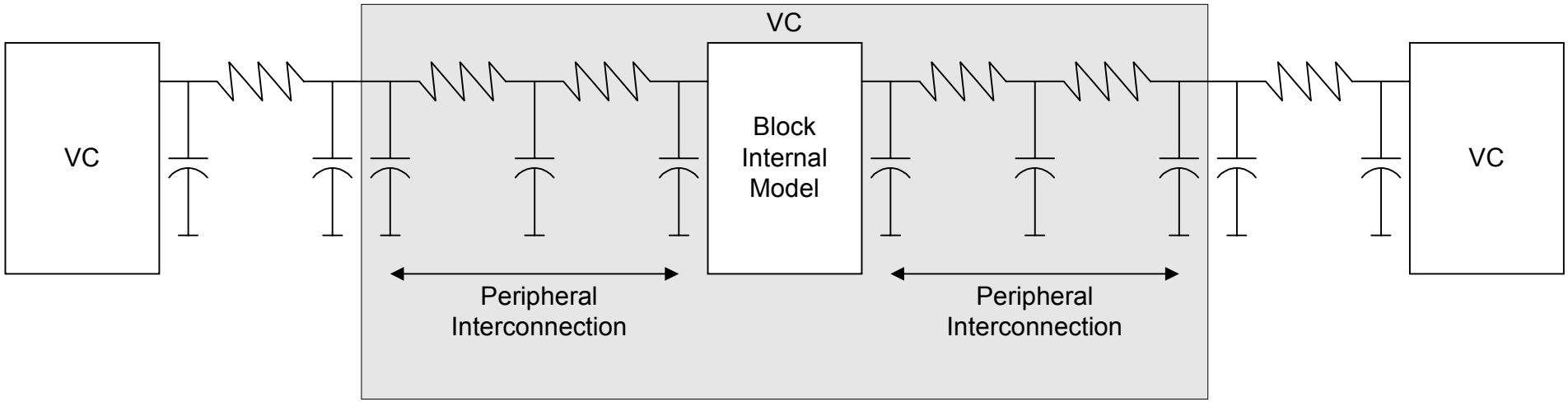
Interface Model



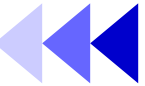
- Describes the operation of a component with respect to its surrounding environment.
- The external connective points (e.g ports or parameters), functional and timing details of the interface are provided to show how the component exchanges information with its environment.
- Also named as *bus functional model* and *interface behavioral model*
- For A/MS VC
  - Only the digital interface is described
  - Analog inputs and outputs are not considered

Peripheral Interconnect Model

- Specifies the interconnection RCs for the peripheral interconnect between the physical I/O ports and the internal gates of the VC
- Used to accurately calculate the interconnect delays and output cell delays associated with the VC
- Used only for the digital interface of the A/MS VC

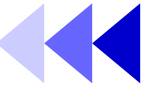


Power Model



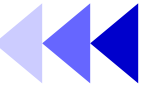
- Defines the power specification of the VC
- Should be capable of representing both dynamic power and static power
  - Dynamic power may be due to capacitive loading or short-circuit currents
  - Static power may be due to state-dependent static currents
- Required for all types of power analysis: average, peak, RMS, etc.
- Abstract level
  - Black/gray box, RTL source code and cell level

Basic Power Analysis Requirements



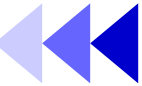
- Any power analysis should include effects caused by the following conditions and events:
  - Switching activity on input ports, output ports, and internal nodes
  - State conditions on I/O ports and optionally internal nodes
  - Modes of operations
  - Environmental conditions such as supply voltage and external capacitive or resistive loading.

Physical Modeling



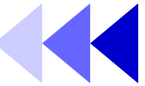
- Physical block implementation of hard, soft and firm VCs.
- Two models for hard VCs
  - Detailed model
    - Description of the physical implementation of the VC at the polygon level
    - The preferred data format is GDSII 6.0.0
  - Abstract model
    - Contains enough information to enable floorplanning, placement, and routing of the system level chip
      - Footprint
      - Interface pin/port list, shape(s), and usage
      - Routing obstructions within the VC
      - Power and ground connections
      - Signature
    - The preferred data format is the MACRO section of VC LEF 5.1

Deliverables



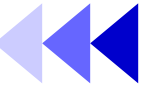
- Deliverables in different processes
  - VC transfer process
    - To find, evaluate and deliver VC
  - VC Integration process
    - Different abstract-level models
    - Comprehensive documentation
      - Application notes, known bugs, system-level verification and testing strategy, installation guides and scripts
- VC delivery specifications
  - Nomenclature/Taxonomy, formats, attributes, and structure for of VC design data and documentation
  - Encryption, archive format, directory structure, etc.

RMM Soft Macro Deliverables



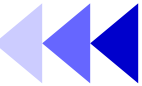
- Product files
  - Synthesizable source code
  - Application notes with HDL design example
  - Synthesis scripts & timing constraints
  - Scripts for scan insertion and ATPG
  - Reference library
  - Installation scripts
- Verification files
  - Bus functional model/monitors used in testbench
  - Testbench files including representative verification tests
- Documentation
  - User guide/Functional specification
  - Datasheet
- System integration files/tools
  - Cycle-based/emulation models as appropriate for macro and/or its testbenches and BFMs
  - Compilers, debuggers, real-time operating systems and software drivers for programmable processor IP

RMM Hard Macro Deliverables



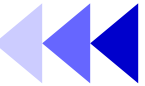
- Product files
  - Installation scripts
- Documentation
  - User guide/functional specification
  - Datasheet
  - Documentation contains version of library used and tools used
- System integration files /tools
  - ISA and/or behavioral model
  - Bus functional model
  - Cycle-based/emulation models as appropriate for macro and/or its testbenches and BFMs
  - Compilers, debuggers, real-time operating systems and software drivers for programmable processor IP

OpenMORE



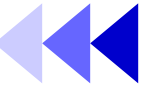
- Open Measure of Reuse Excellence (Open MORE)
  - A collaboration between Mentor Graphics and Synopsys
  - Based on Reuse Methodology Manual (RMM)
  - IP providers use OpenMORE for self-evaluation
  - Designers can ensure that each portion of a design is workable and reusable
  - Help the IP industry move to a higher quality level
  - Supported by industry groups - VSIA,VCX, RAPID and Design and Reuse

Motorola's SRS



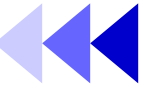
- Semiconductor Reuse Standards (SRS)
  - An efficient design methodology that will enable rapid and effective plug-and-play integration of reusable silicon IP into system-on-a-chip solutions.
- Current four standards in SRS V2.0
  - IP/VC Block Deliverables
    - Data format of deliverables of soft, firm and hard core in different phases
      - 1st phase: deliverables required for instantiation and verification
      - 2nd phase: deliverables required for backend-related views
      - 3rd phase: deliverables required for test aspects
    - Directory structure and naming convention of PC/VC deliverables

Motorola's SRS



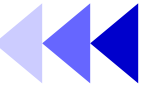
- Current four standards in SRS V2.0
  - IP Interface (IPI)
    - Colored line standard and signal definition
    - Bus Interface operation
  - Verilog HDL Coding
    - Coding style and module partition for test, synthesis and reuse
    - Based on IEEE 1364.1 synthesizable Verilog subset, Synopsys/Mentor RMM and Motorola's experience in direct design
  - Documentation
    - Defines the content and format of documents required for IP/VC such as Processor Cores, Analog/Mixed-Signal etc.
    - Document types include Creation, Use, Integration, and Manufacturing Test.
    - Long-term goal: become as independent of software and platform

FPGA Reuse Manual



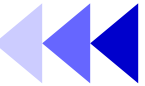
- Xilinx
  - To facilitate design reuse in SoRC
  - Xilinx design reuse methodology for ASIC and FPGA designers manual
    - FPGA Supplement to Reuse Methodology Manual (RMM)
    - Provides an overview of FPGA system level features
    - Contains general RTL synthesis coding guidelines
  - Xilinx FPGA reuse field guide
    - Reuse concept from perspective of project
      - Project specifications, project management organization, and project verification and qualification
- Actel
  - Actel HDL coding style guide
    - Provides the preferred coding styles in both VHDL and Verilog for the Actel architecture

Outline



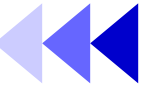
- IP Core Designs
- IP Core Verification
- IP Core Modeling and Deliverables
- **System-Level Verification**

System Verification



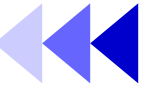
- It begins during system specification. The specification describes the basic test plan including the criteria.
- As the system-level behavior model is developed, a testbench and test suite should be developed to verify the model.
- The system software should be developed and tested using the behavior model.
- A rich set of test suites should be available for the RTL and entire chip verification.

Verification Strategy



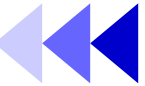
- Verify the individual IPs are functionally correct as stand-alone units
- Verify the interfaces between IPs are functional correct, first in terms of the transaction type, and then in terms of data content.
- Prototype the full chip and tun a set of complex applications on the full chip.

IP-level Verification

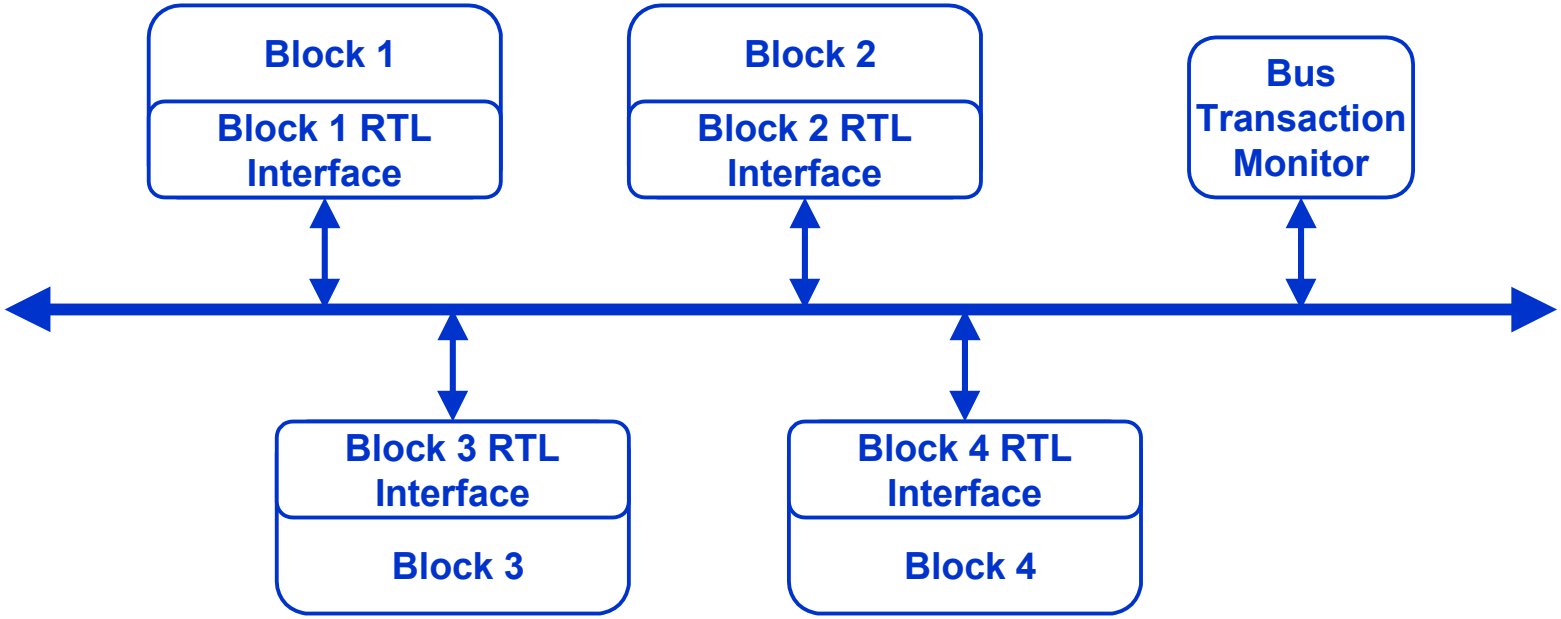


- Use code coverage tools and a rigorous methodology to verify the RTL version of the IP.
- A physical prototype is built to prove silicon verification of functional correctness.

Interface Verification

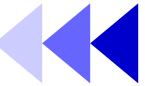


- Interface: address/data bus. Protocols
  - permitted sequence of control and data signals
  - use a bus transaction monitor to check the transaction



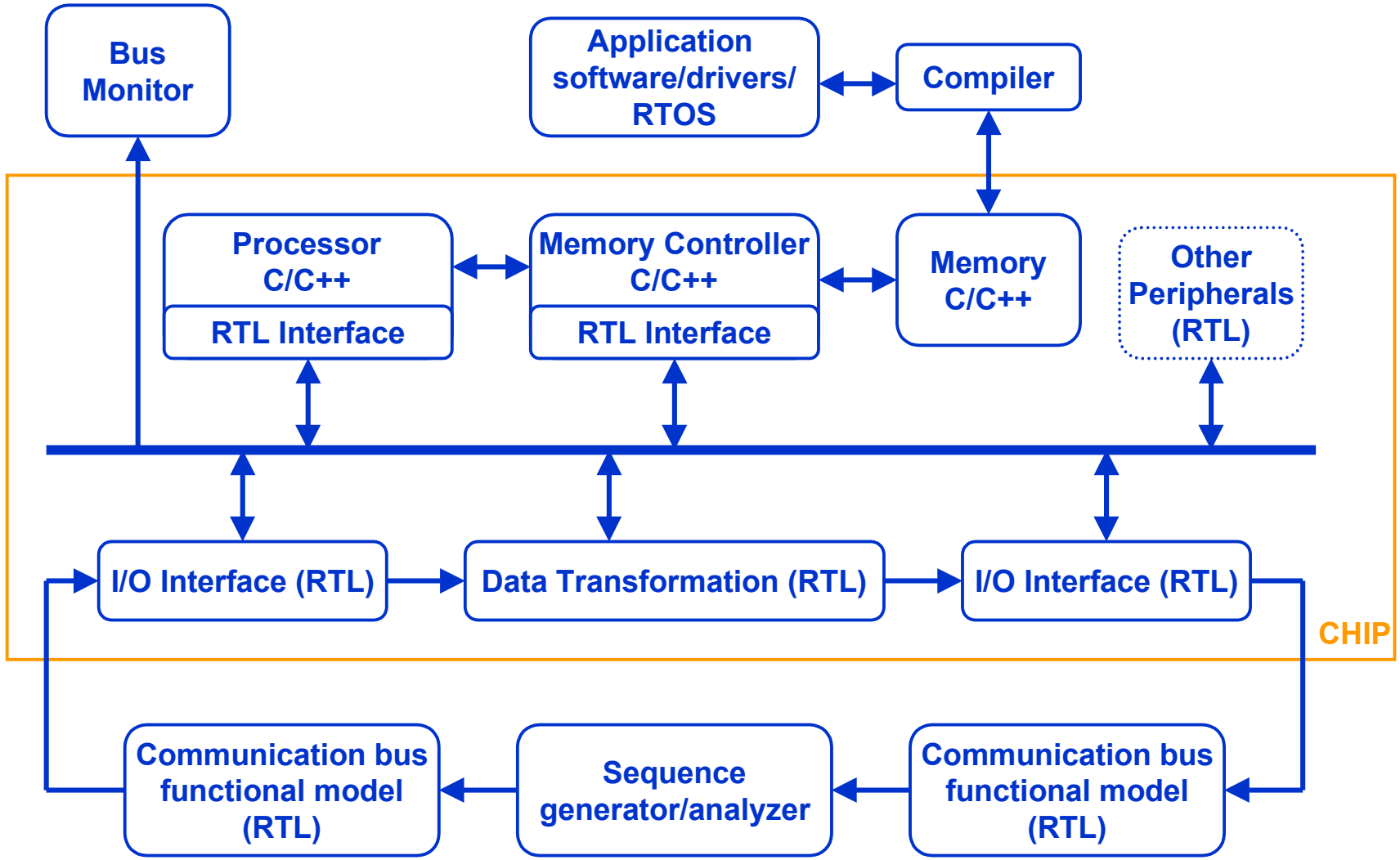
- Use BFM to check the data read and write

Functional Verification (1/2)

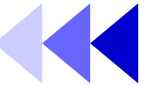


- Two basic approaches
  - increase level of abstraction so that software simulators running on workstations faster
  - use specialized hardware for performing verification, such as emulator or rapid prototyping
- Canonical SoC abstraction
  - Full RTL model for IP cores
  - behavior or ISA model for memory and processor
  - bus functional model and monitor to generate and check the transactions between IPs
  - generate real application code for the processor and run it on the simulation model

Functional Verification (2/2)



Rapid Prototyping



- FPGA prototyping
  - Aptix (FPGAs + programmable routing chips
- Emulation-based testing
  - FPGA-based or processor-based
  - QuickTurn and Mentor Graphics
- Real silicon prototyping
  - faster and easier to build an actual chip and debug it
  - design features in the real silicon chip
    - good debug structure
    - ability to selectively reset the individual IP blocks
    - ability to selectively disable various IP blocks to prevent bugs from affecting operations of the system