Chapter 1
Digital System Design Automation with VHDL

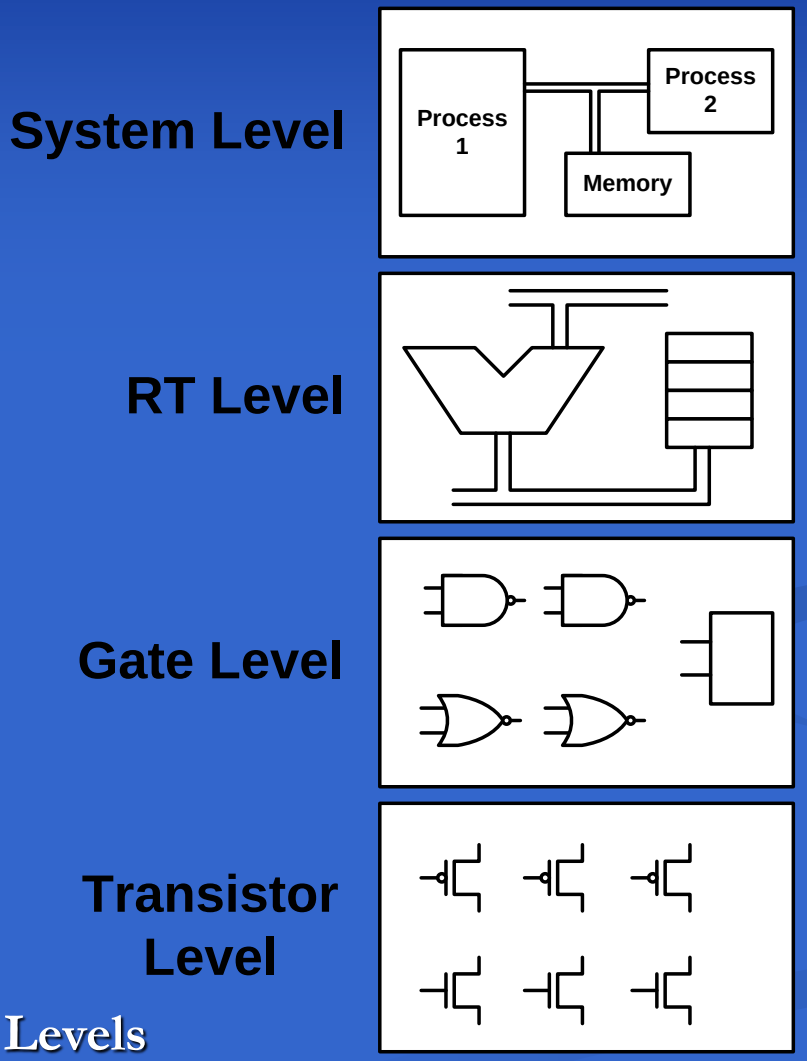
Digital System Design Automation with VHDL

Digital System Design Automation with VHDL

1.4 VHDL
- 1.4.1 VHDL Initiation
- 1.4.2 Existing Languages
- 1.4.3 VHDL Requirements
- 1.4.4 The VHDL Language

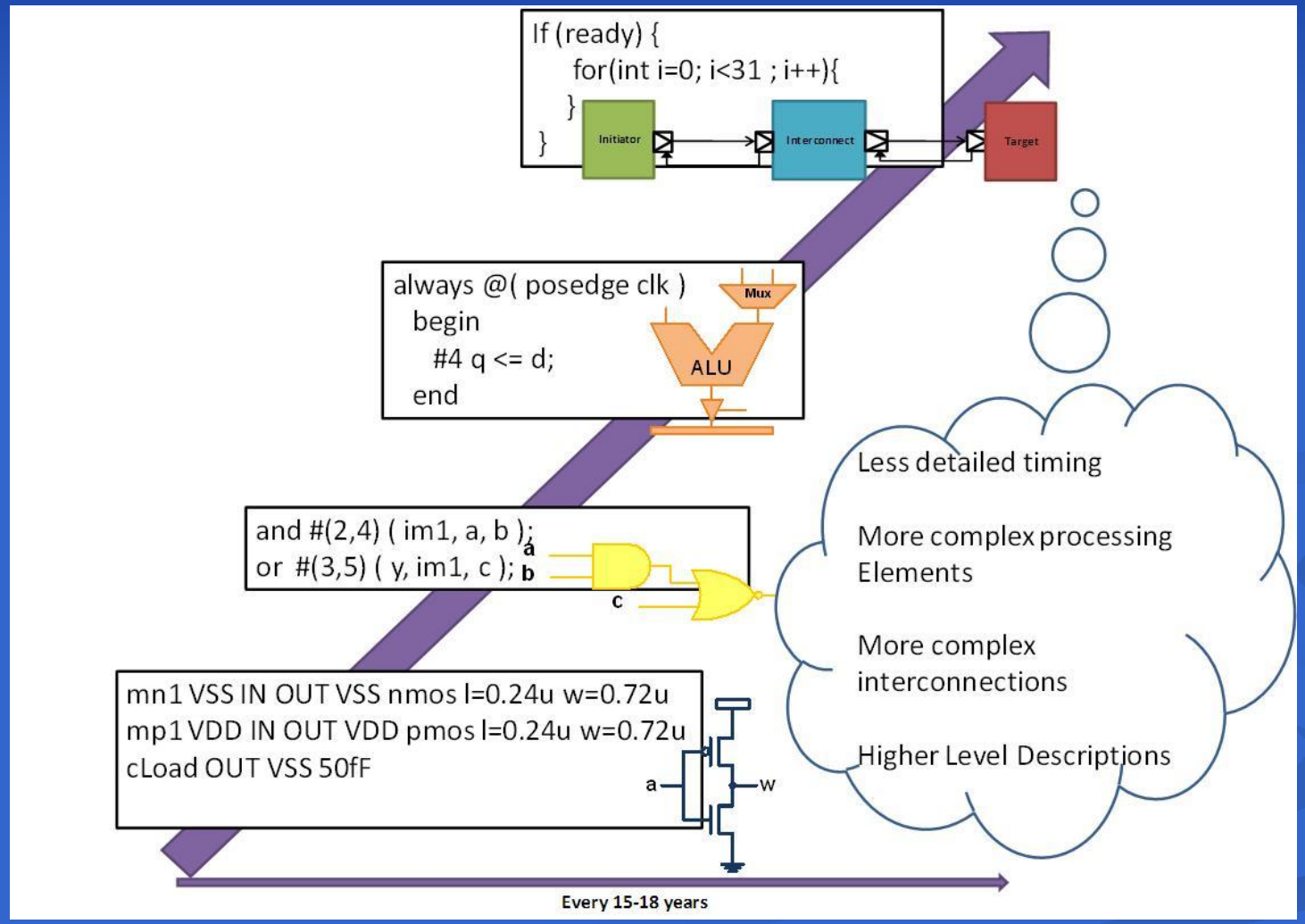
1.5 Summary

Abstraction Levels

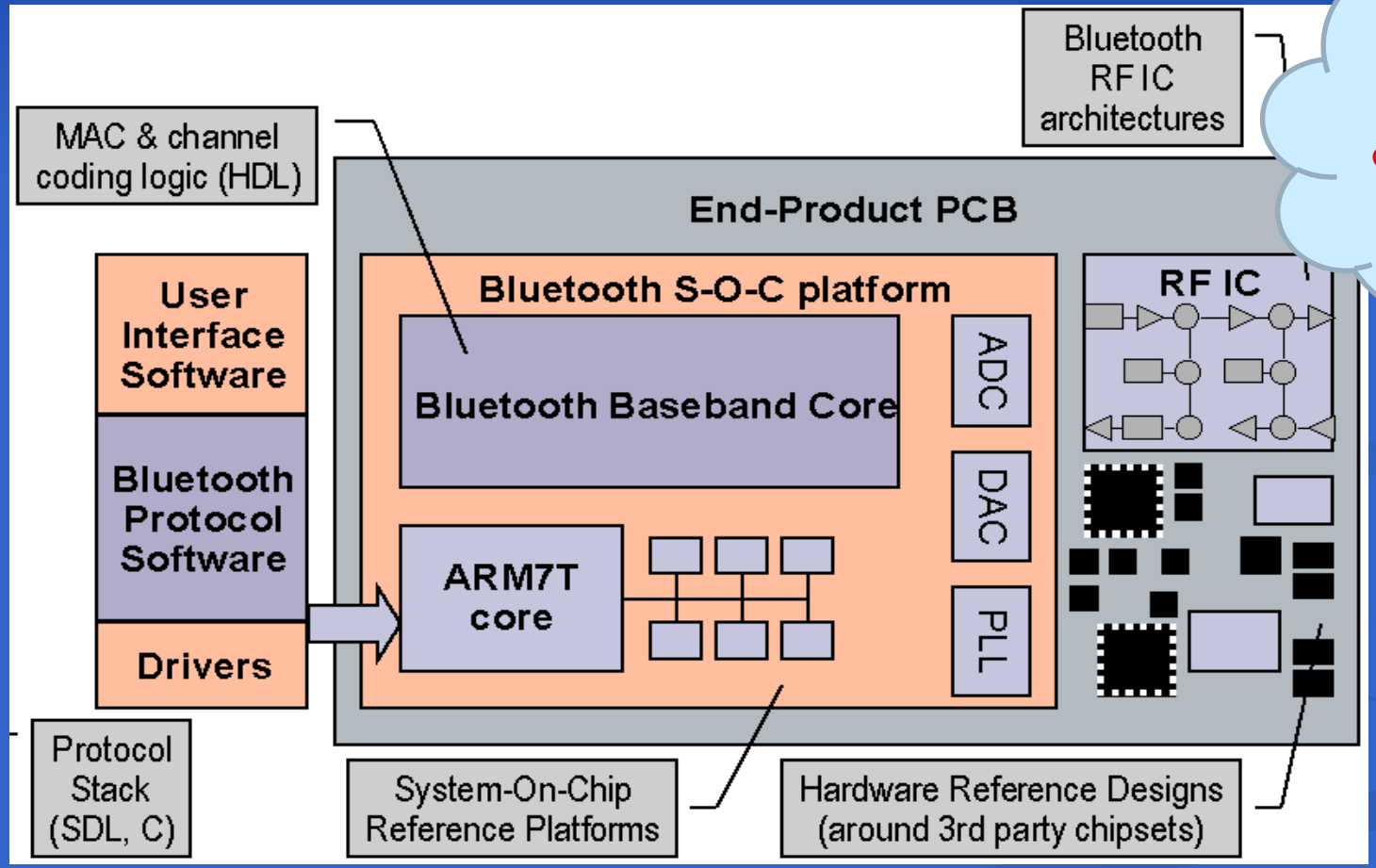


- Abstraction Levels

Abstraction Evolution



System Level Design



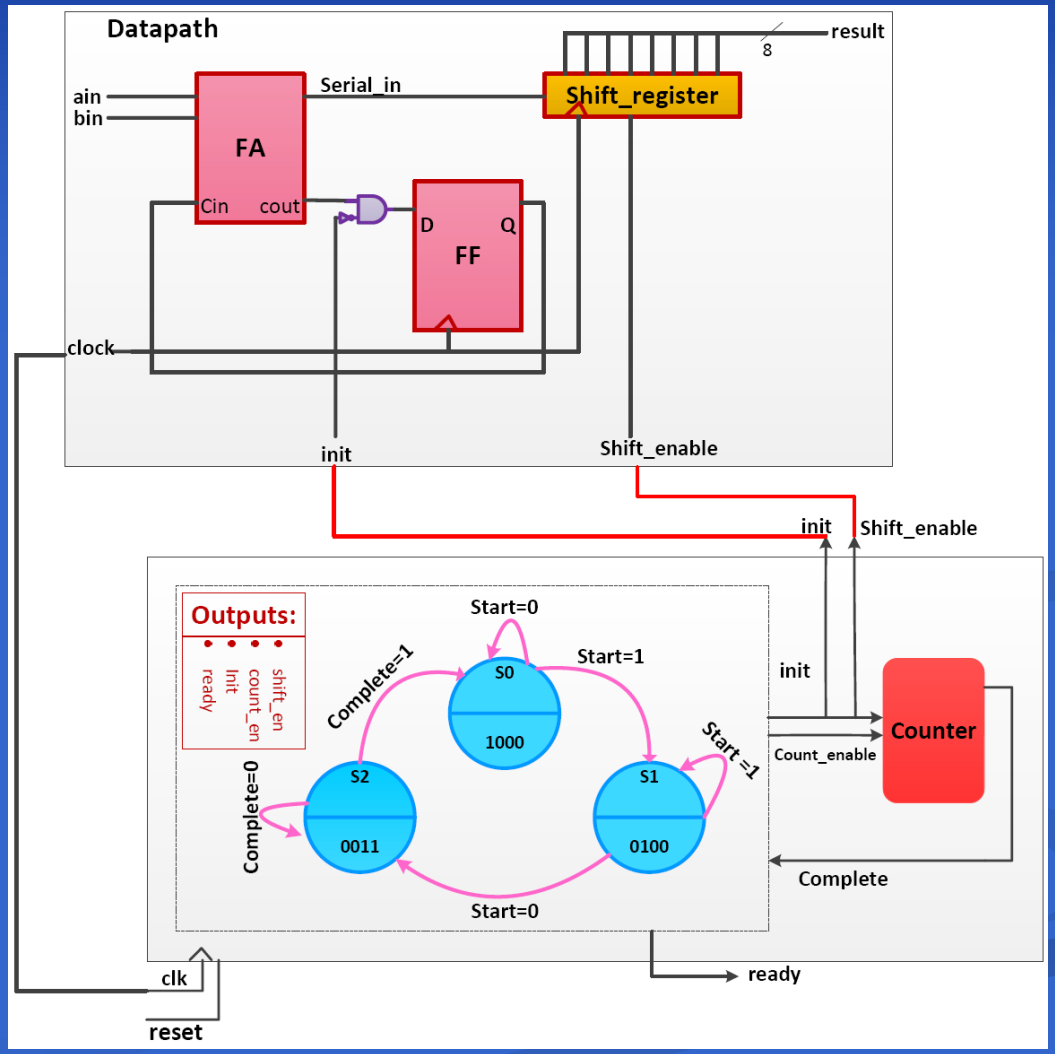
High level language description for complex and various types of core

ESL Design

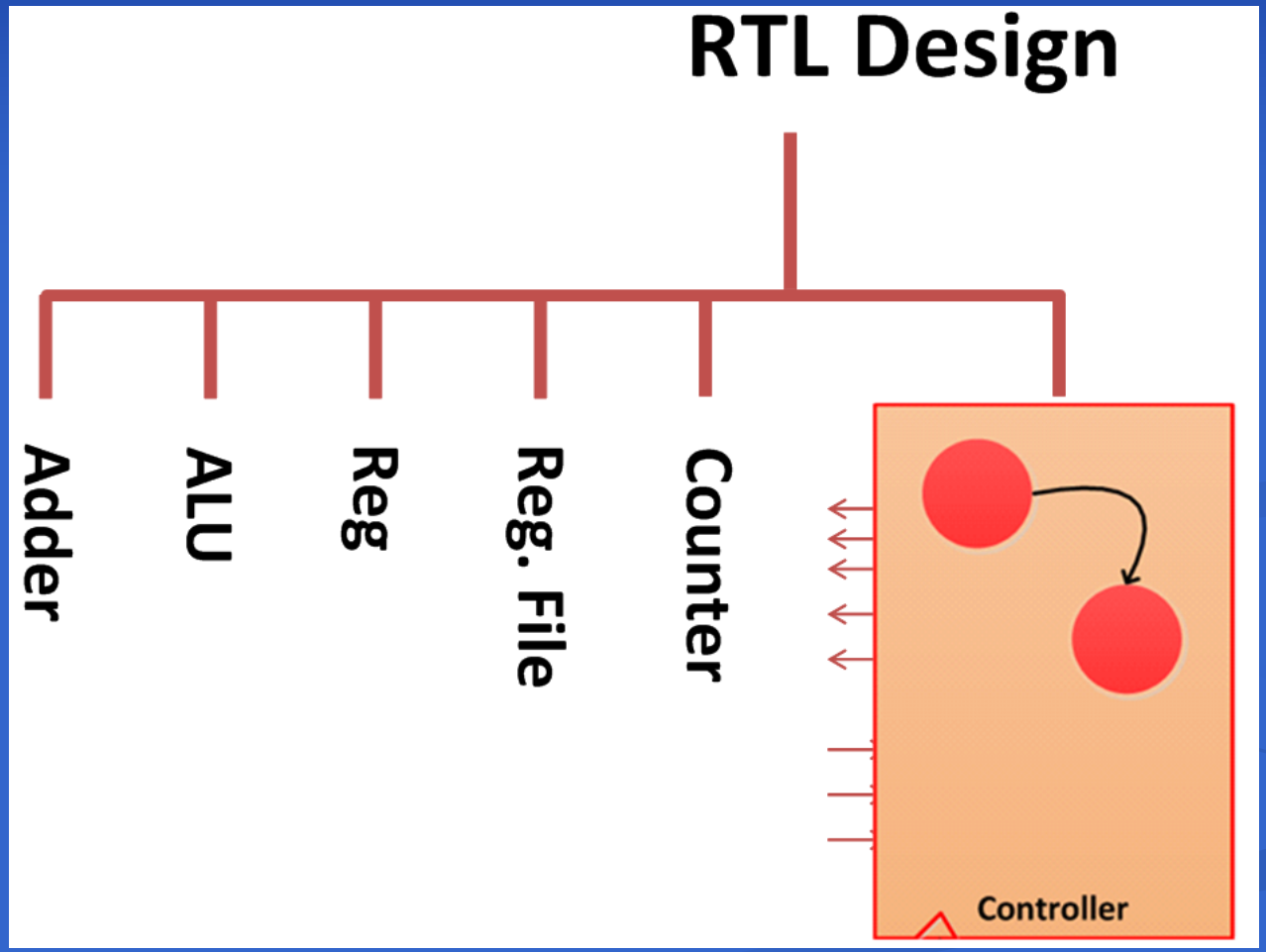
- Start with RT level HDLs early in the curriculum
- Proper use of tools
- Treat computers as component
- Master RT level by the end of UG curriculum
- Teach integration
- Emphasis on embedded design
- C++ electronic modeling
- C++ based languages such as SystemC are used
- SystemC is used for core description
- SystemC–AMS is used for non-digital core description
- TLM is used for communications
- Channels at the lower levels
- Transport functions at the upper level

RTL Design

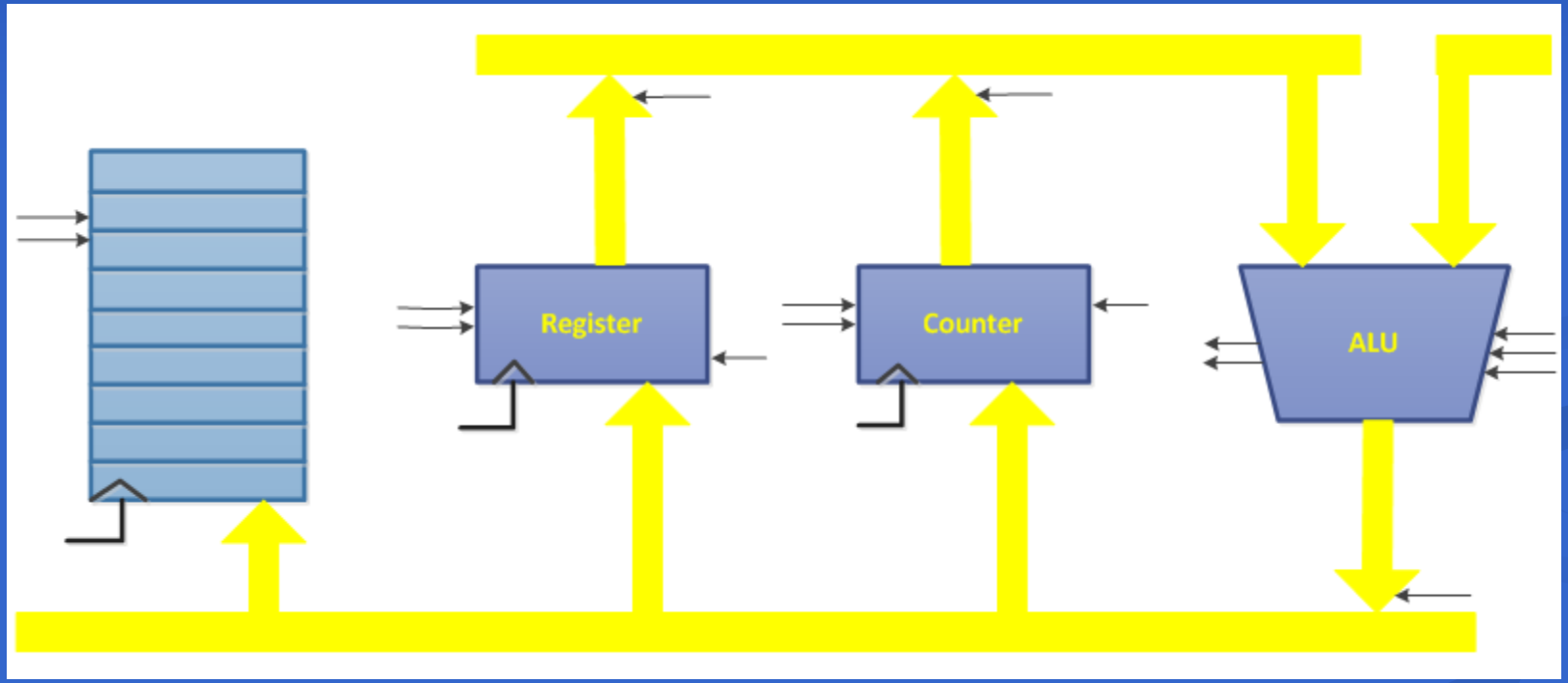
- Any component that passes, holds or processes data is a datapath component
- Controller is the thinking part of your machine
- You should decide how to wire datapath components
- When designing datapath, don't be concerned about how control signals are Issued



RTL Design

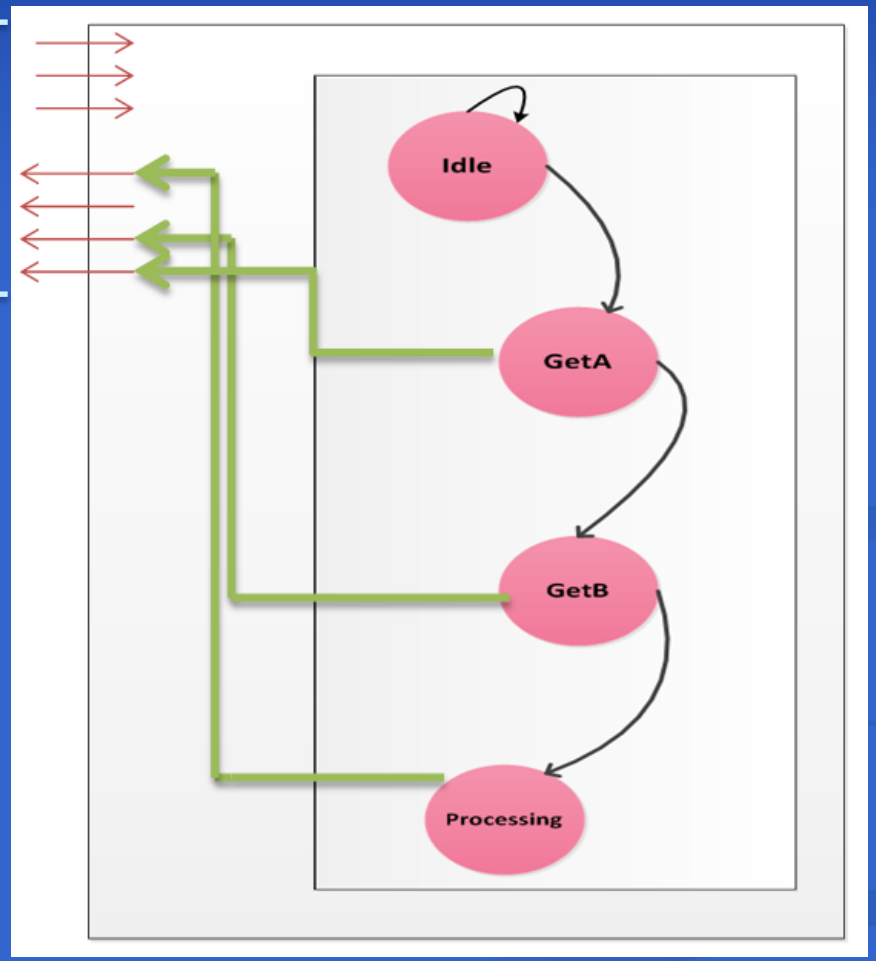


RTL Datapath Example

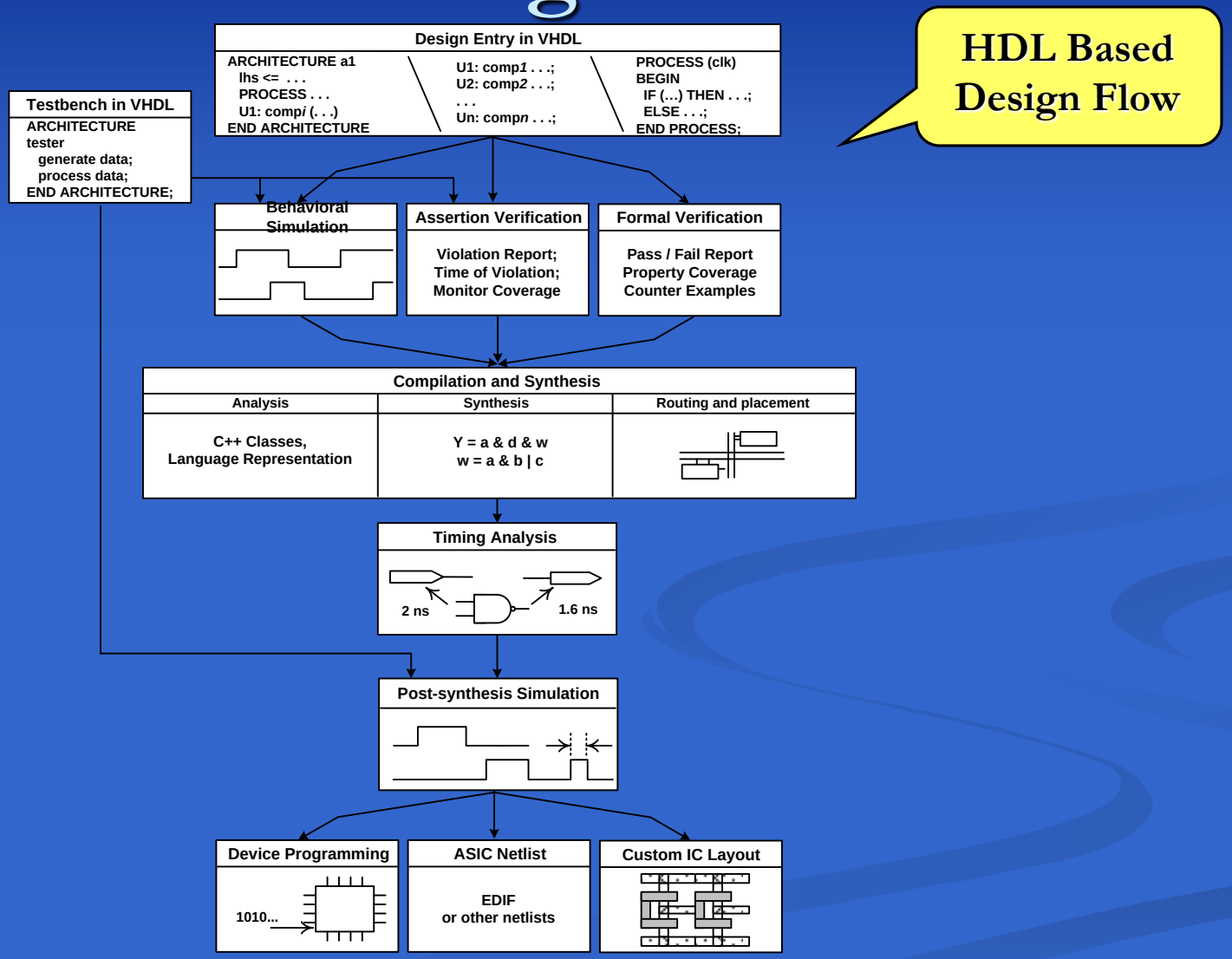


RTL Controller Example

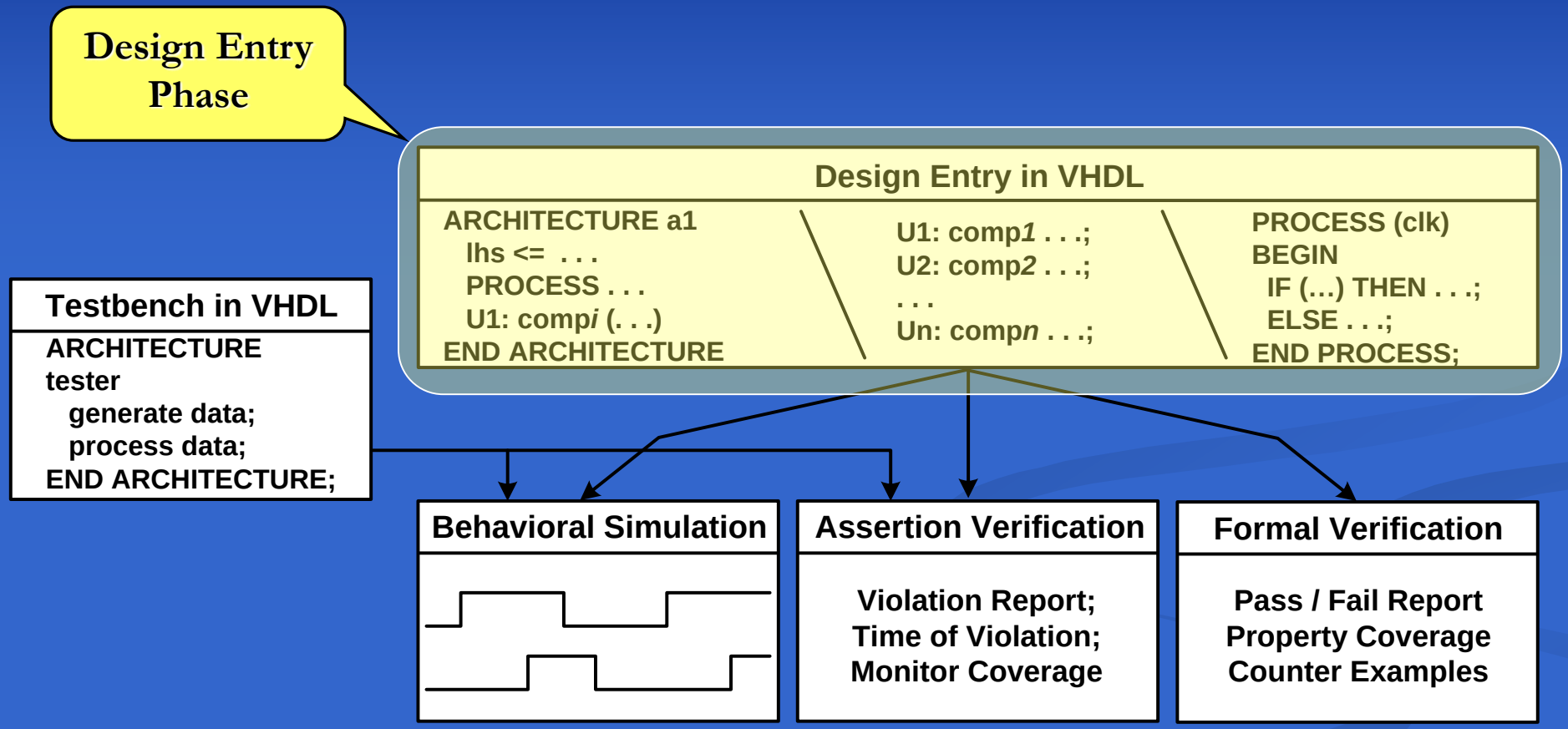
Signals that go to or come from datapath



RTL Design Flow

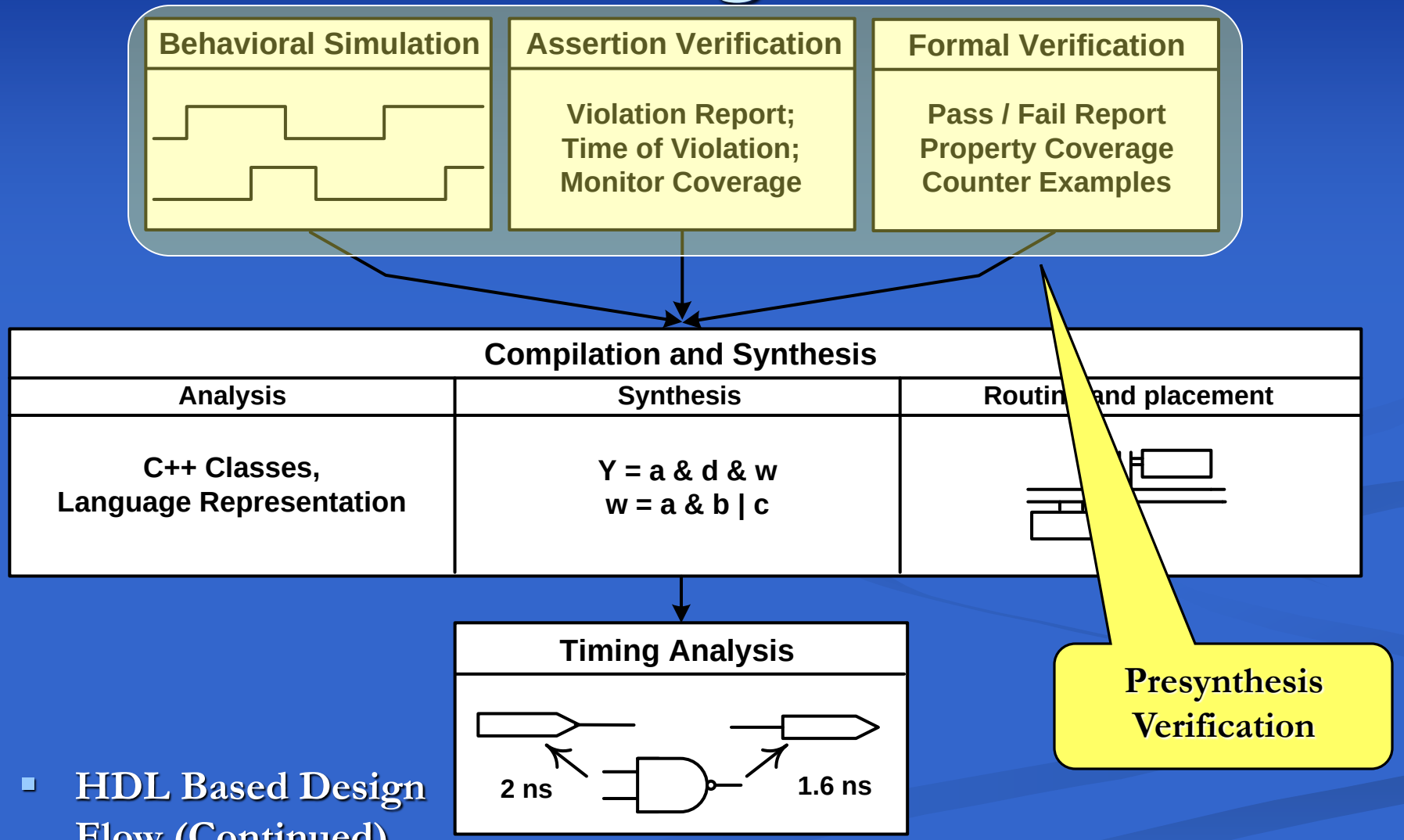


RTL Design Flow



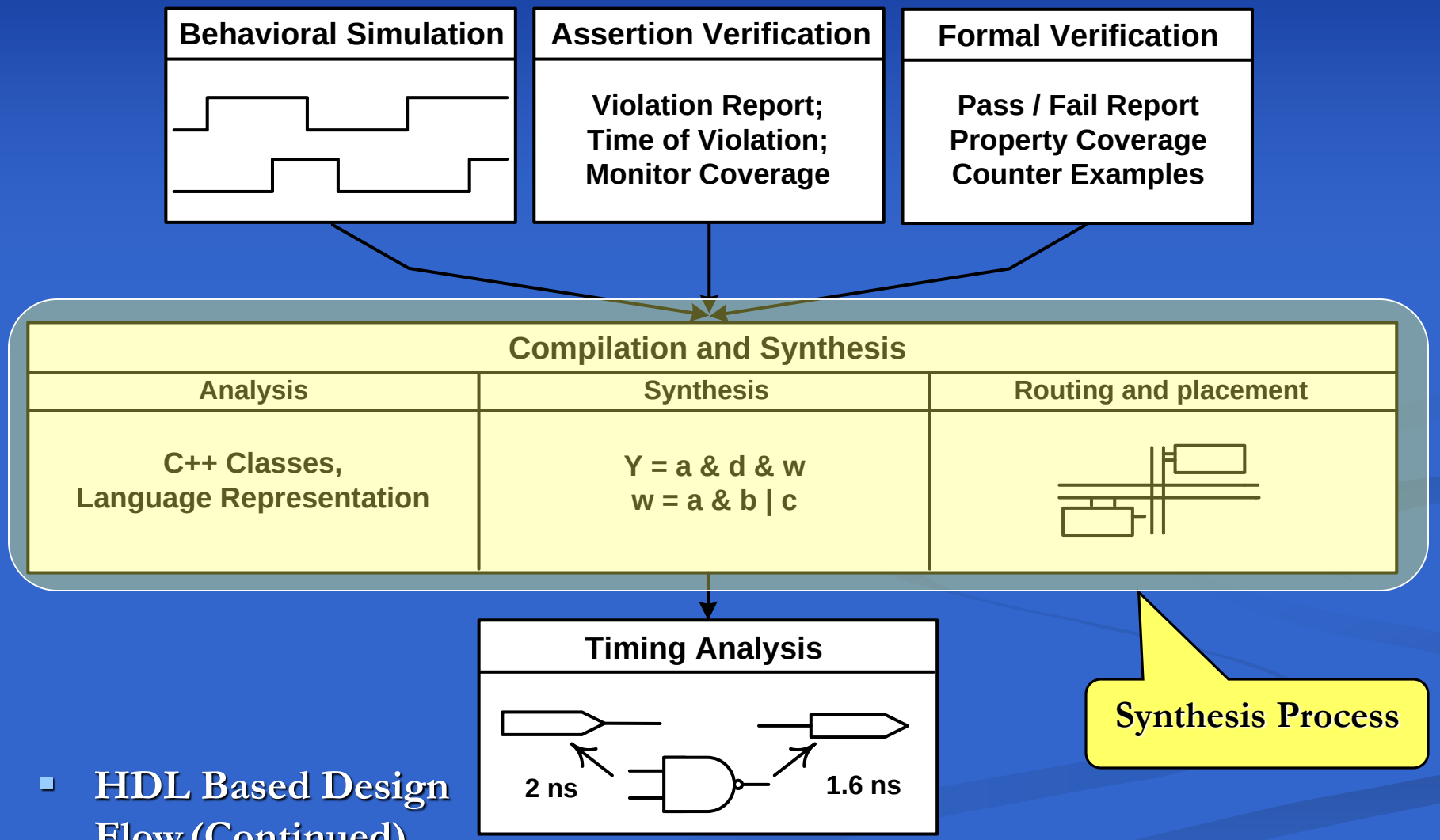
- HDL Based Design Flow

RTL Design Flow



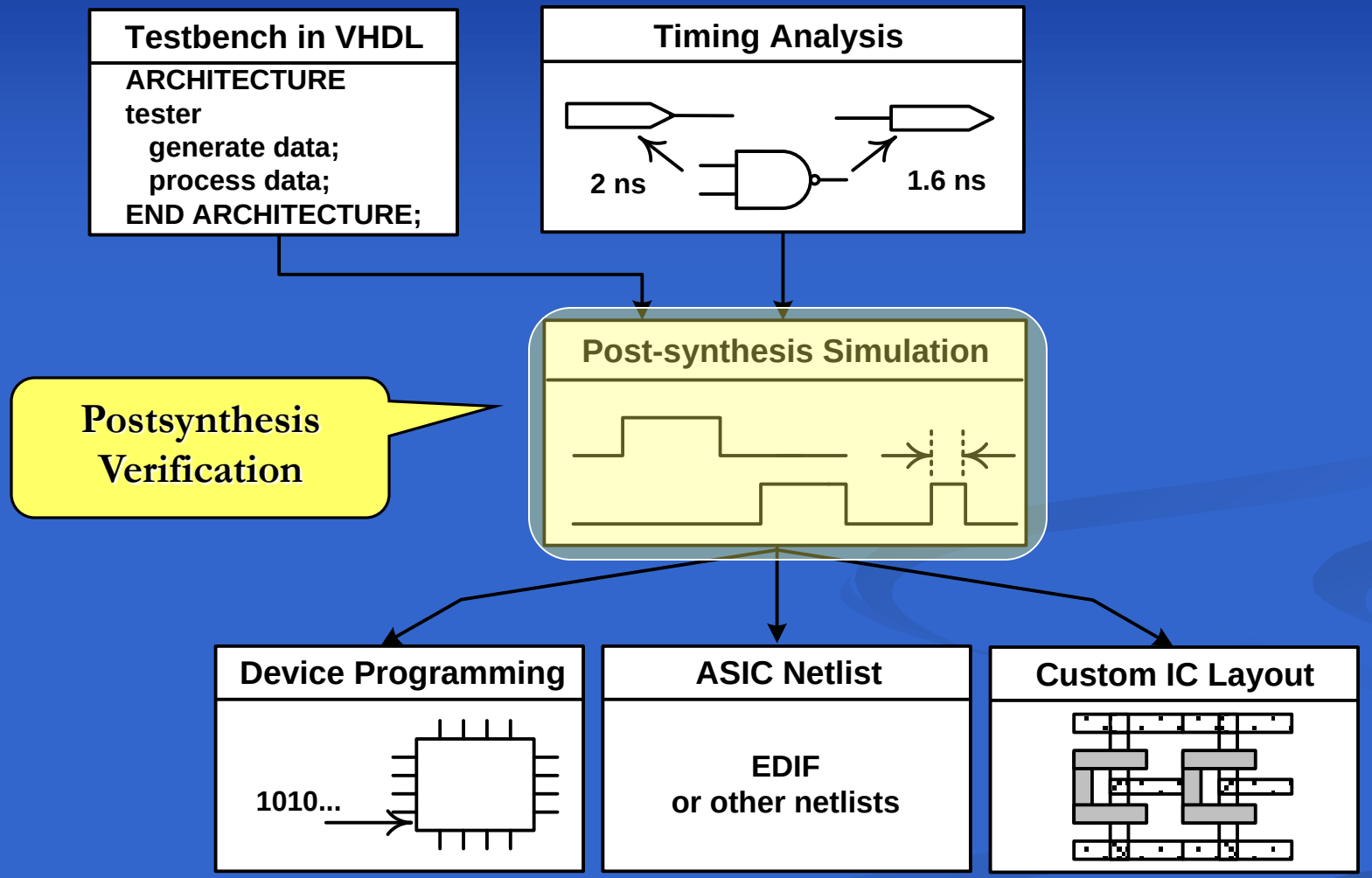
- HDL Based Design Flow (Continued)

RTL Design Flow



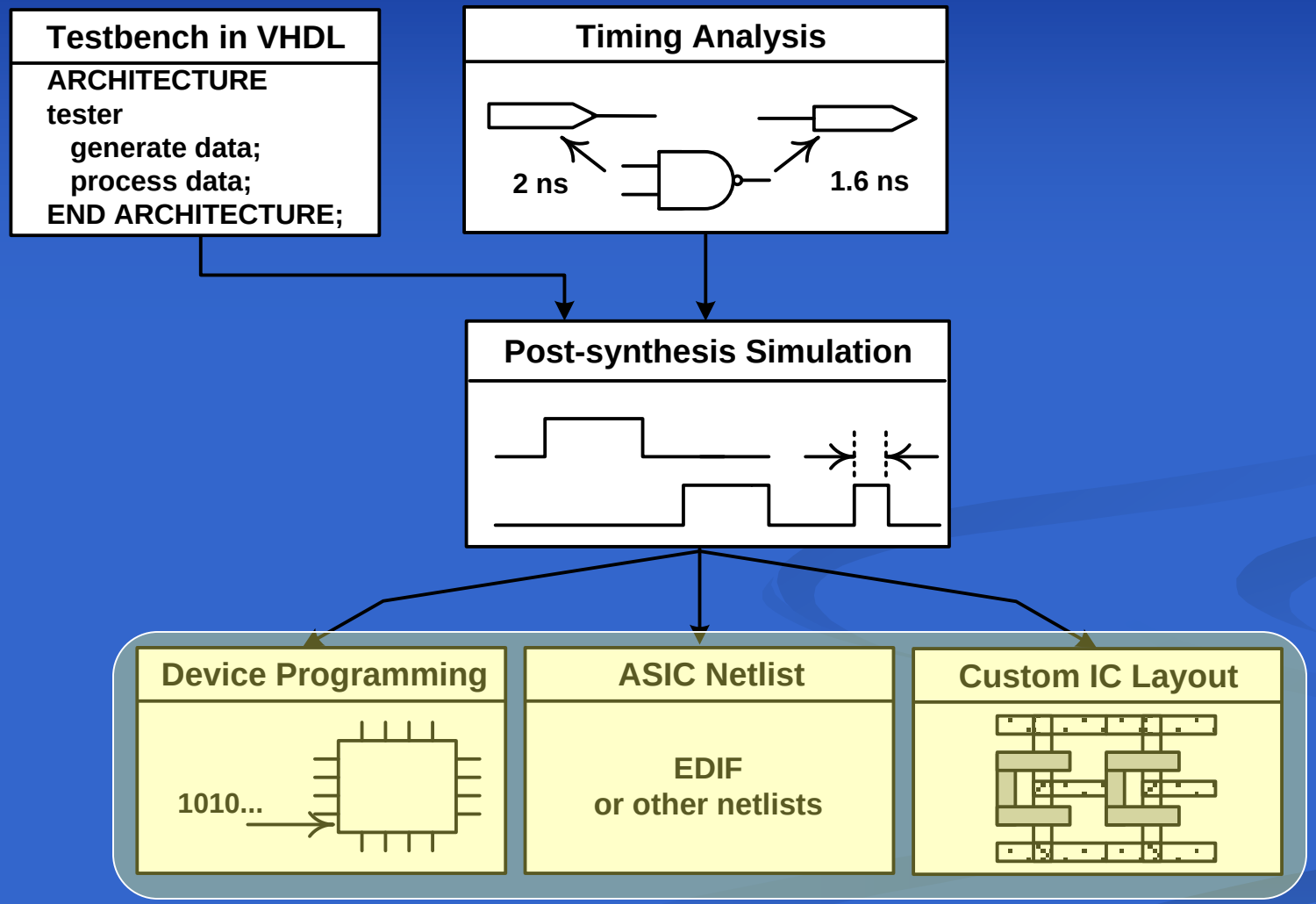
- HDL Based Design Flow (Continued)

RTL Design Flow

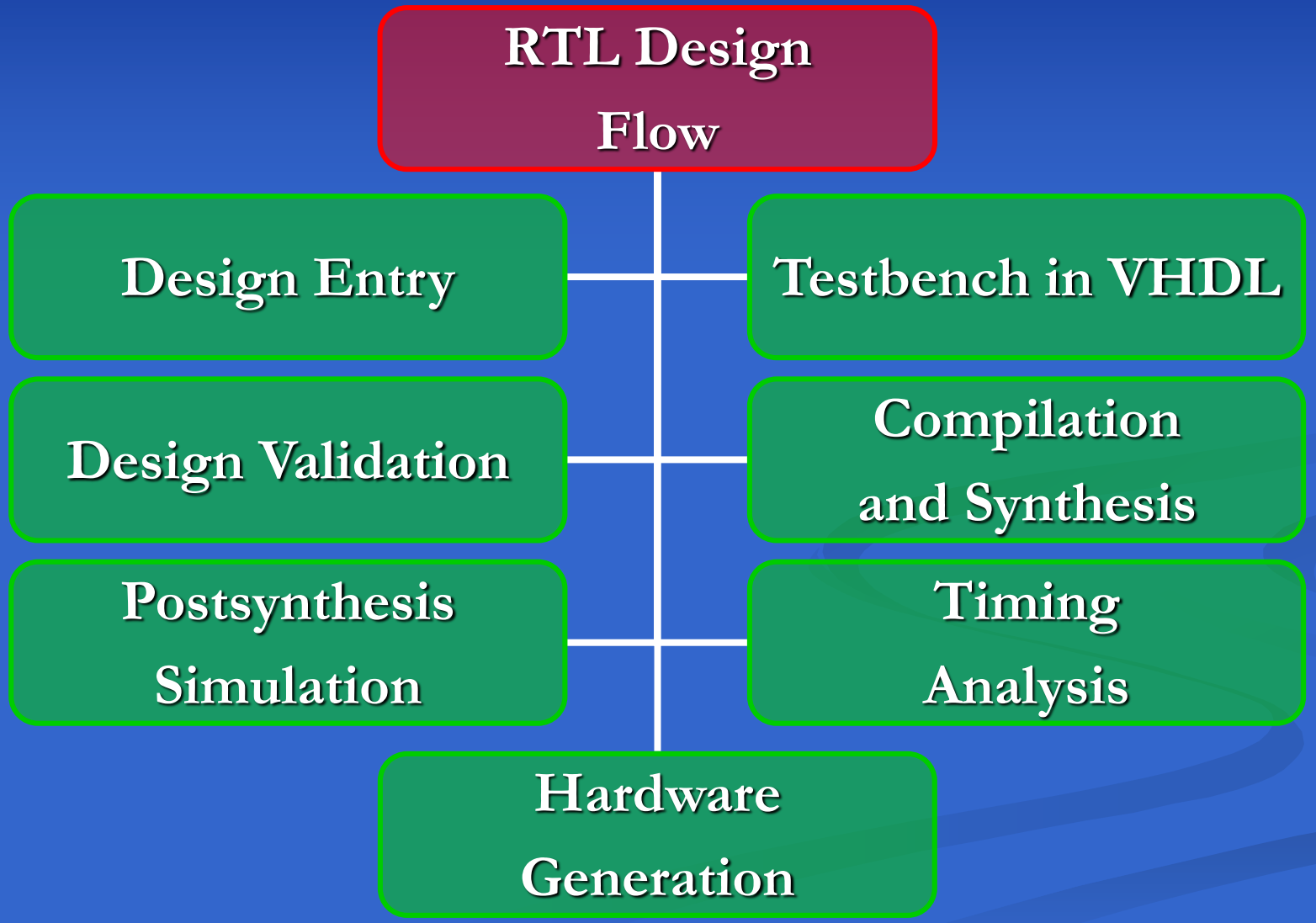


- HDL Based Design Flow (Continued)

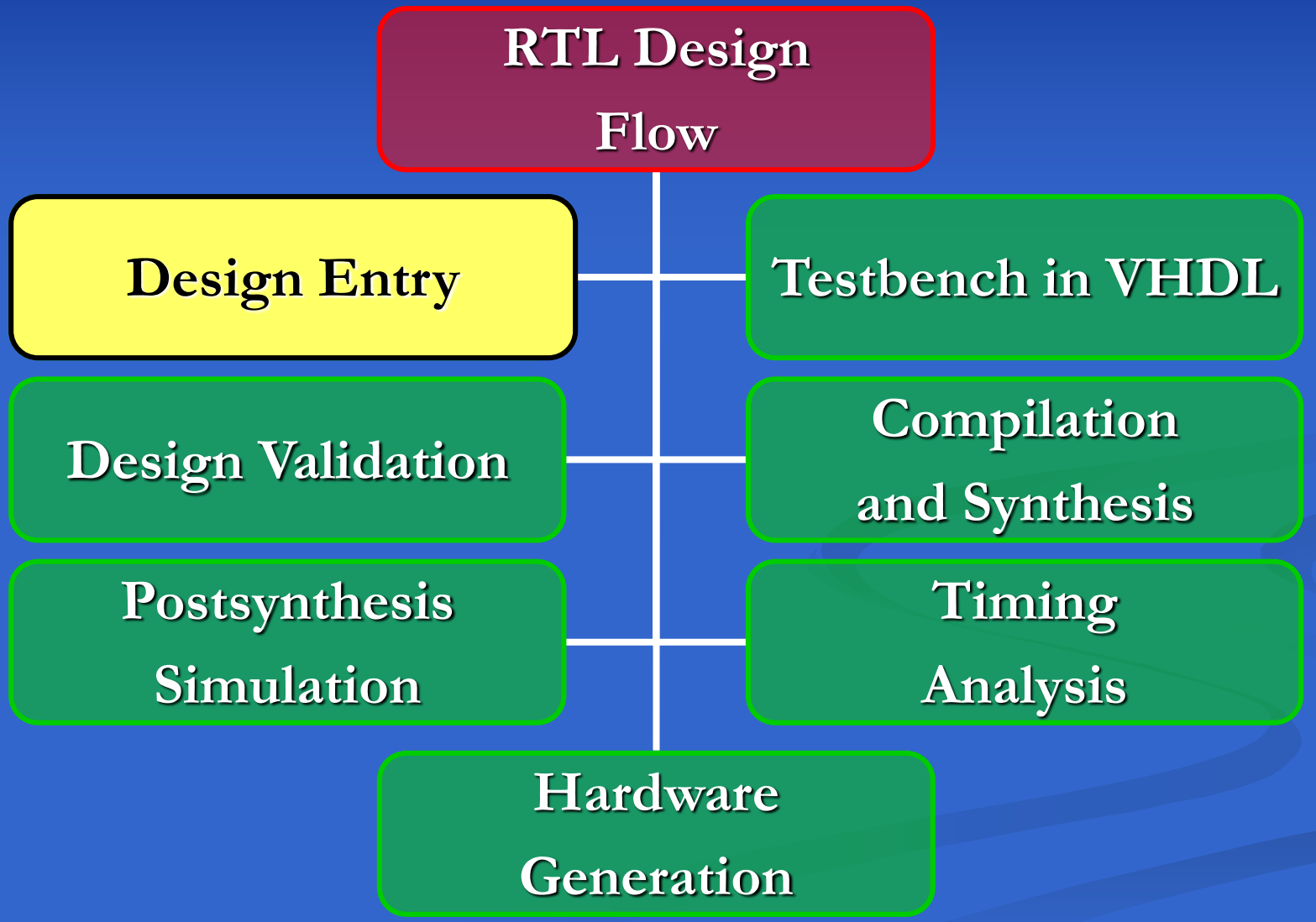
RTL Design Flow



RTL Design Flow



Design Entry

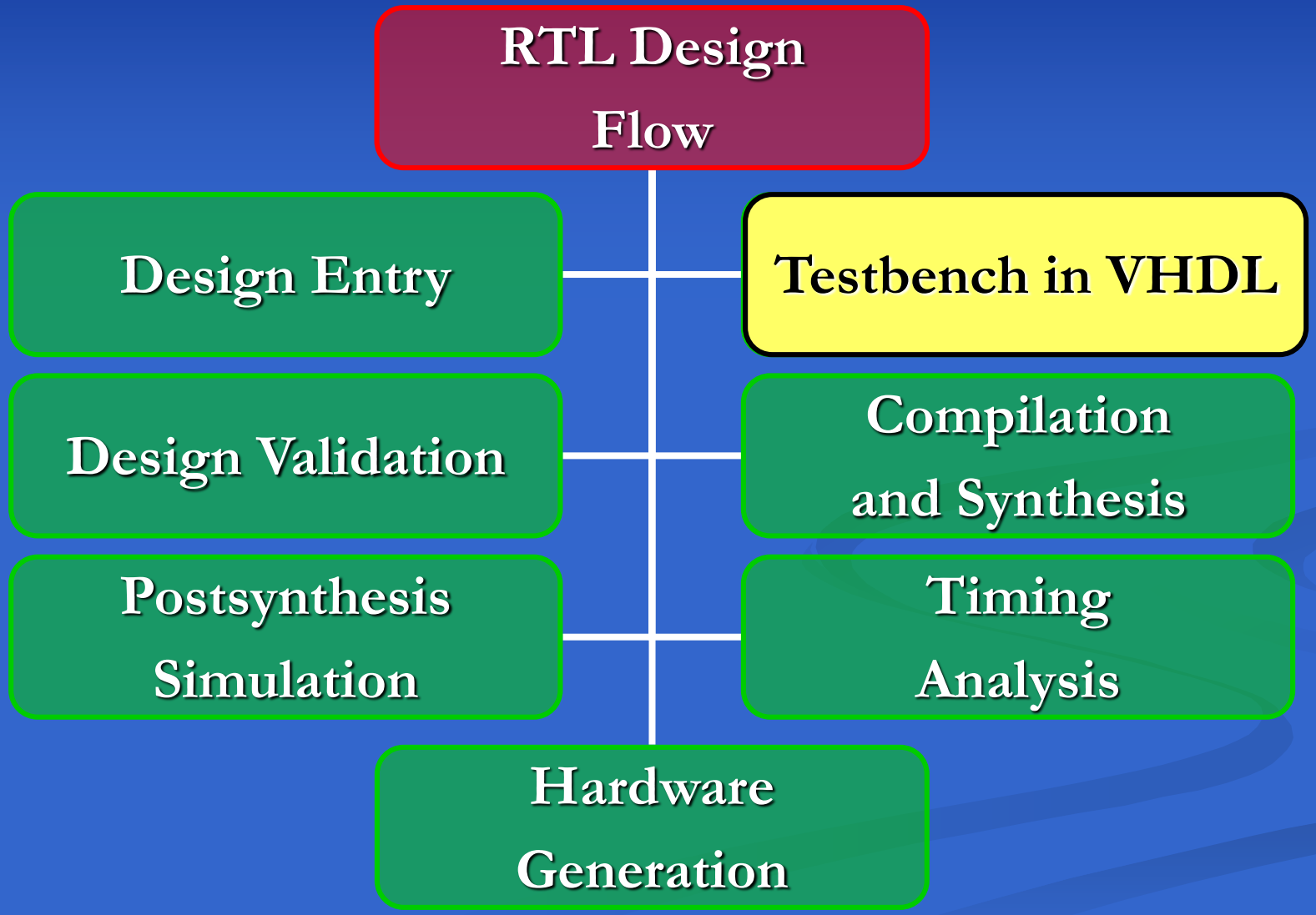


Design Entry

- The first step in the design of a digital system
- Describing the design in VHDL in a top-down hierarchical fashion
- **Register Transfer Level (RTL):** High-level VHDL designs usually described at this level

- VHDL constructs used in RT level design:
  - ***Sequential statements*** for high-level behavioral descriptions
  - ***Signal assignments*** for representing logic blocks, bus assignments, and bus and input/output interconnect specifications
  - ***Instantiation statements*** for using lower-level components in an upper-level design

Testbench in VHDL

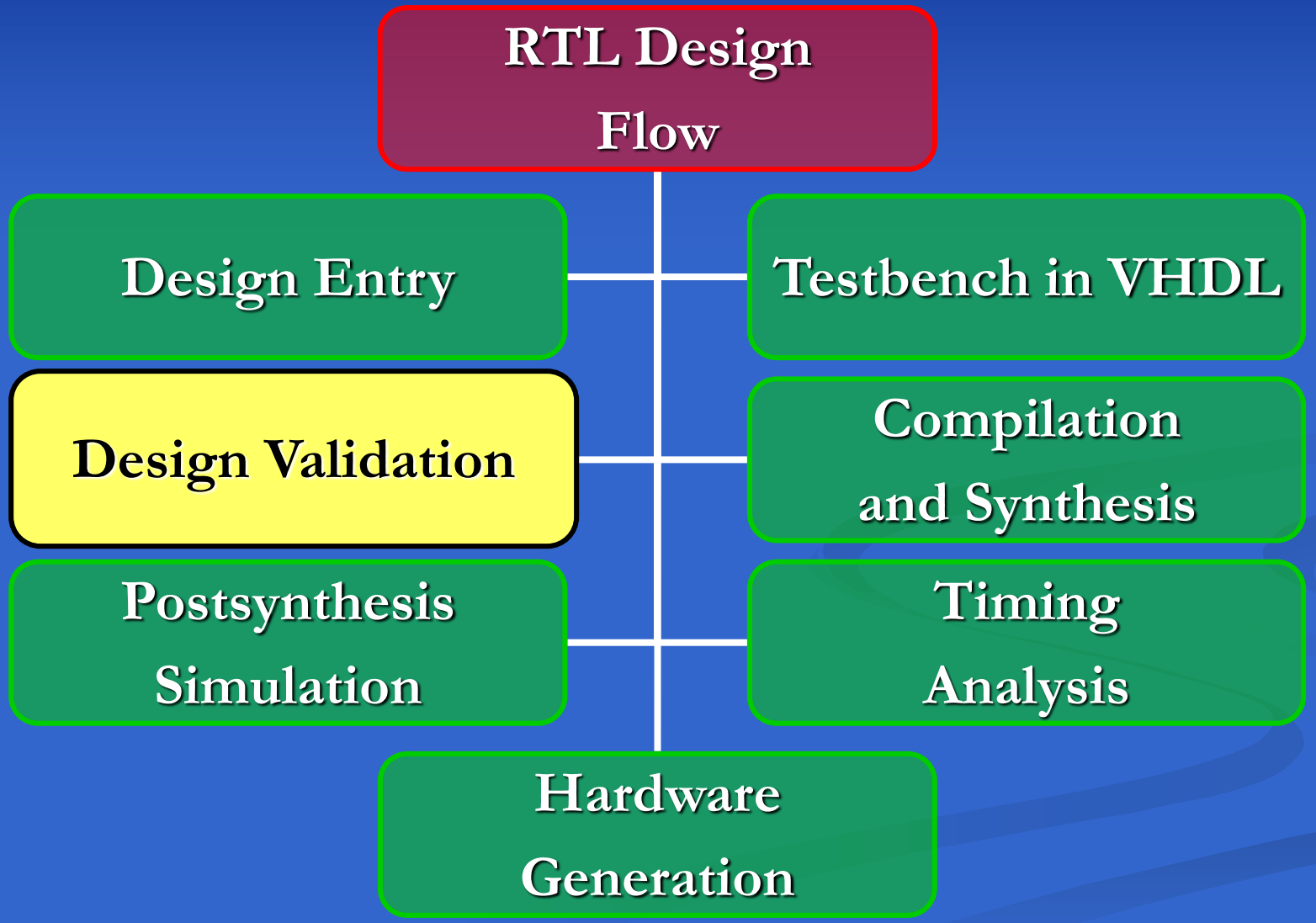


Testbench in VHDL

- Simulation and Test of a designed system functionality before Hardware generation
- Detection of design errors and incompatibility of components used
- in the design
- By generation of a test data and observation of simulation results

- **Testbench:** A VHDL module
  - Use of high-level constructs of VHDL for:
    - Data Generation
    - Response Monitoring
    - Handshaking with the design
  - Inside the Testbench: Instantiation of the design module
  - Forms a simulation model together with the design, used by a VHDL simulation engine

Design Validation



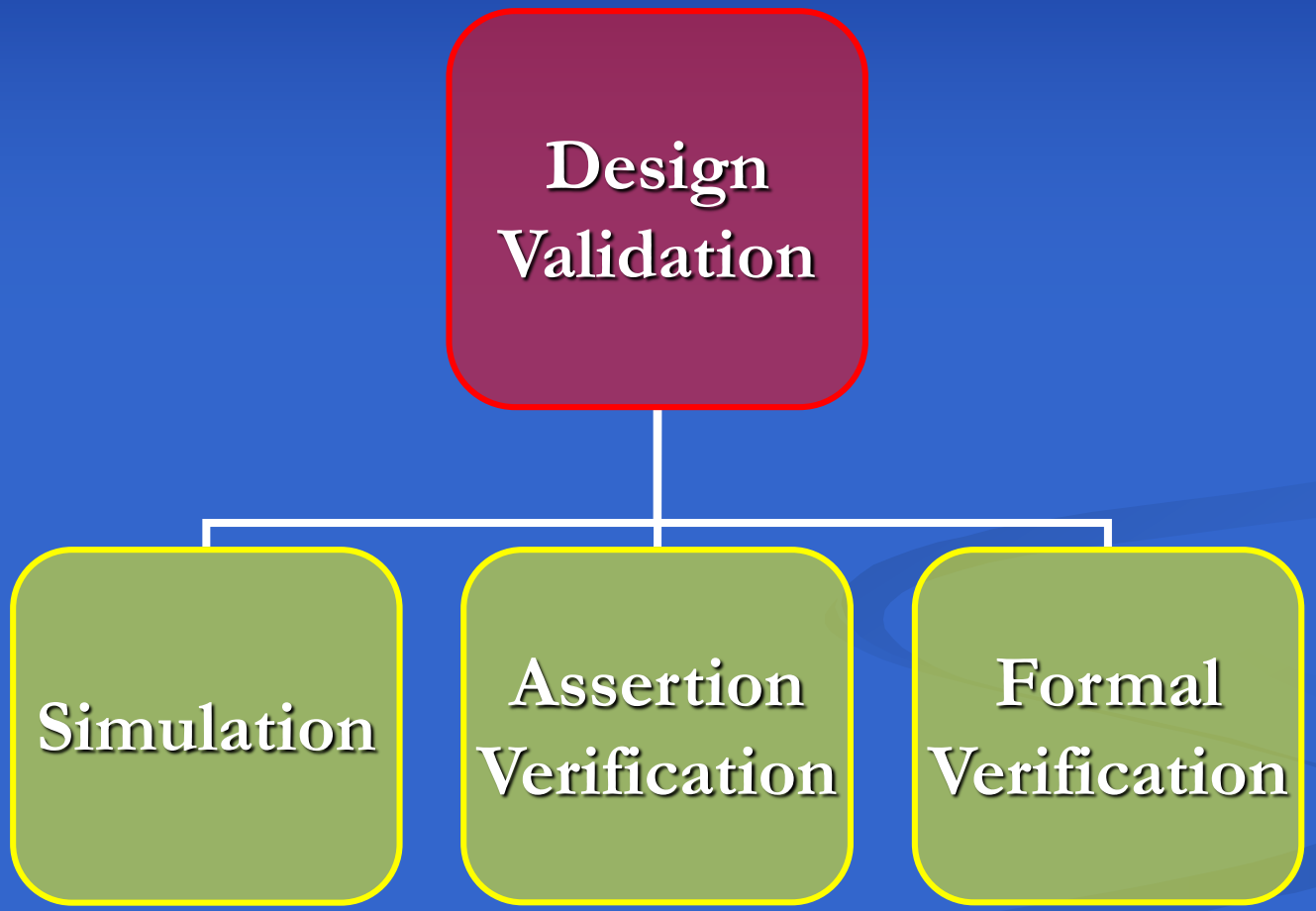
Design Validation

- An important task in any digital system design
- The process to check the design for any design flaws

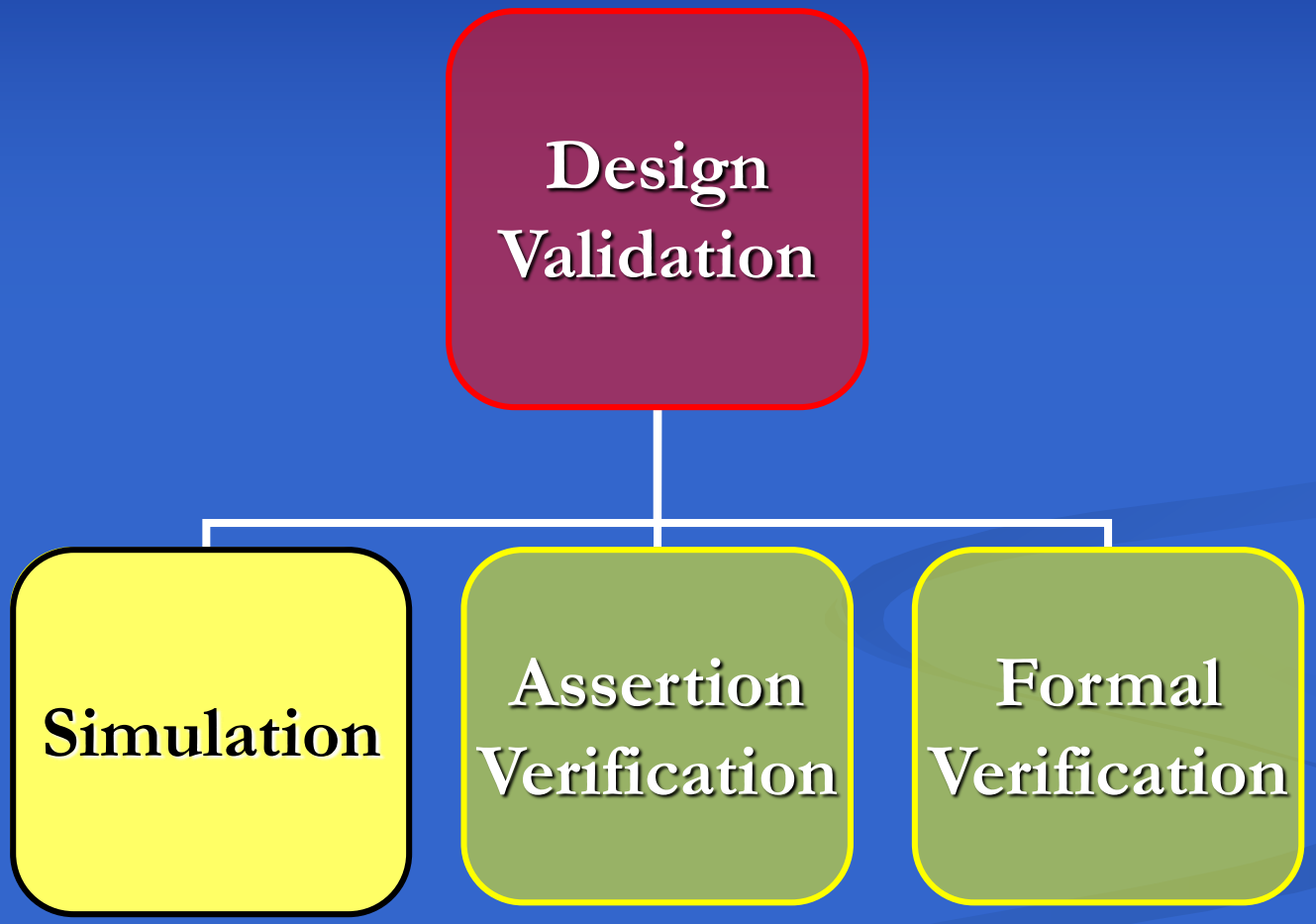
- A design flaw due to:
  - Ambiguous Problem Specifications
  - Designer Errors
  - Incorrect Use of Parts in the Design

- Can be done by:
  - Simulation
  - Assertion Verification
  - Formal Verification

Design Validation



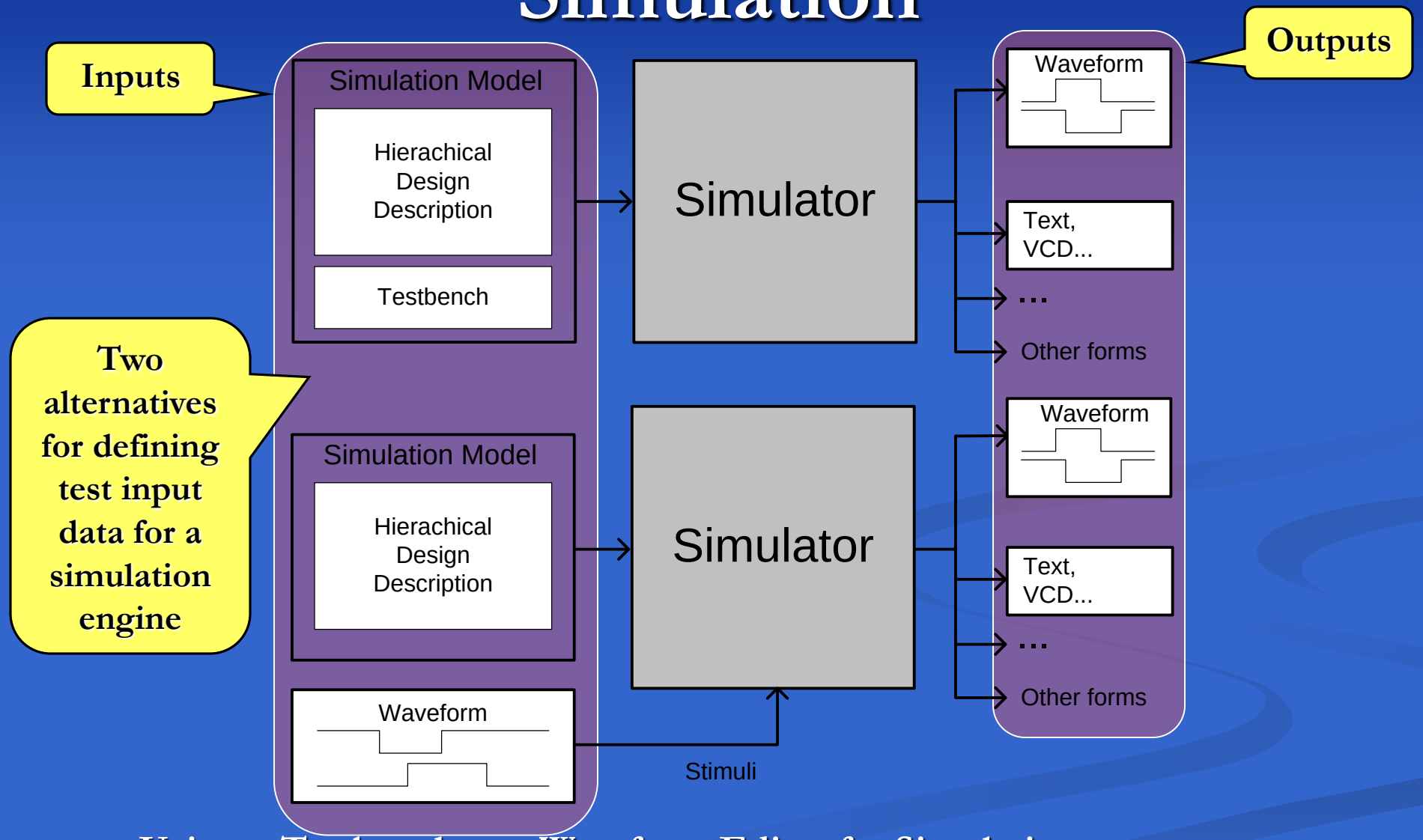
Simulation


Design Validation
Simulation
Assertion Verification
Formal Verification

Simulation

- Simulation for design validation, done before a design is synthesized
- Also Referred to as RT level, or Pre-synthesis Simulation
- Simulation at RTL level is accurate to the clock level
- The advantage: its speed compared with simulations at the gate or transistor levels
- The Required Test data: generated graphically using waveform editors, or through a testbench
- Outputs of simulators:
  - Waveforms (for visual inspection)
  - Text for large designs for machine processing

Simulation



- **Using a Testbench or a Waveform Editor for Simulation**

Simulation

Testbench for the Counter Circuit

```
ENTITY counter4_tester IS END ENTITY;
ARCHITECTURE timed OF counter4_tester IS
  SIGNAL r : std_logic;
  SIGNAL c : std_logic := '0';
  SIGNAL cnt : std_logic_vector (3 DOWNTO 0);
BEGIN
  UUT1: ENTITY WORK.counter4 (procedural) PORT MAP (r, c, cnt);
  r <= '0', '1' AFTER 09 NS, '0' AFTER 17 NS,
       '1' AFTER 59 NS, '0' AFTER 67 NS;
  c <= NOT c AFTER 3.5 NS WHEN NOW <= 75 NS ELSE '0';
END ARCHITECTURE timed;
```

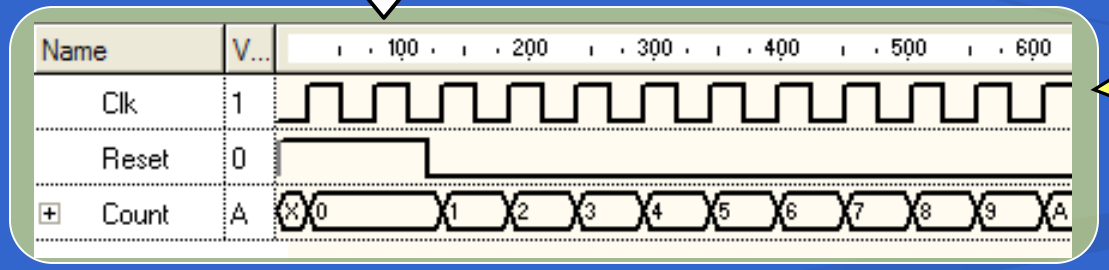
Testbench

```
ENTITY counter4 IS
  PORT (Reset, Clk : IN std_logic;
        Count : OUT std_logic_vector (3 DOWNTO
0));
END ENTITY;
ARCHITECTURE procedural OF counter4 IS
  SIGNAL cnt_reg : std_logic_vector (3 DOWNTO 0);
BEGIN
  PROCESS (Clk) BEGIN
    IF (Clk = '0' AND Clk'EVENT) THEN
      IF (Reset = '1') THEN cnt_reg <= "0000";
      ELSE cnt_reg <= cnt_reg + 1; END IF;
    END IF;
  END PROCESS;
  Count <= cnt_reg;
END ARCHITECTURE procedural;
```

Design to Simulate

VHDL Code of a Counter Circuit

Simulator



The simulation results in form of a waveform

- VHDL Simulation with a Testbench

...n

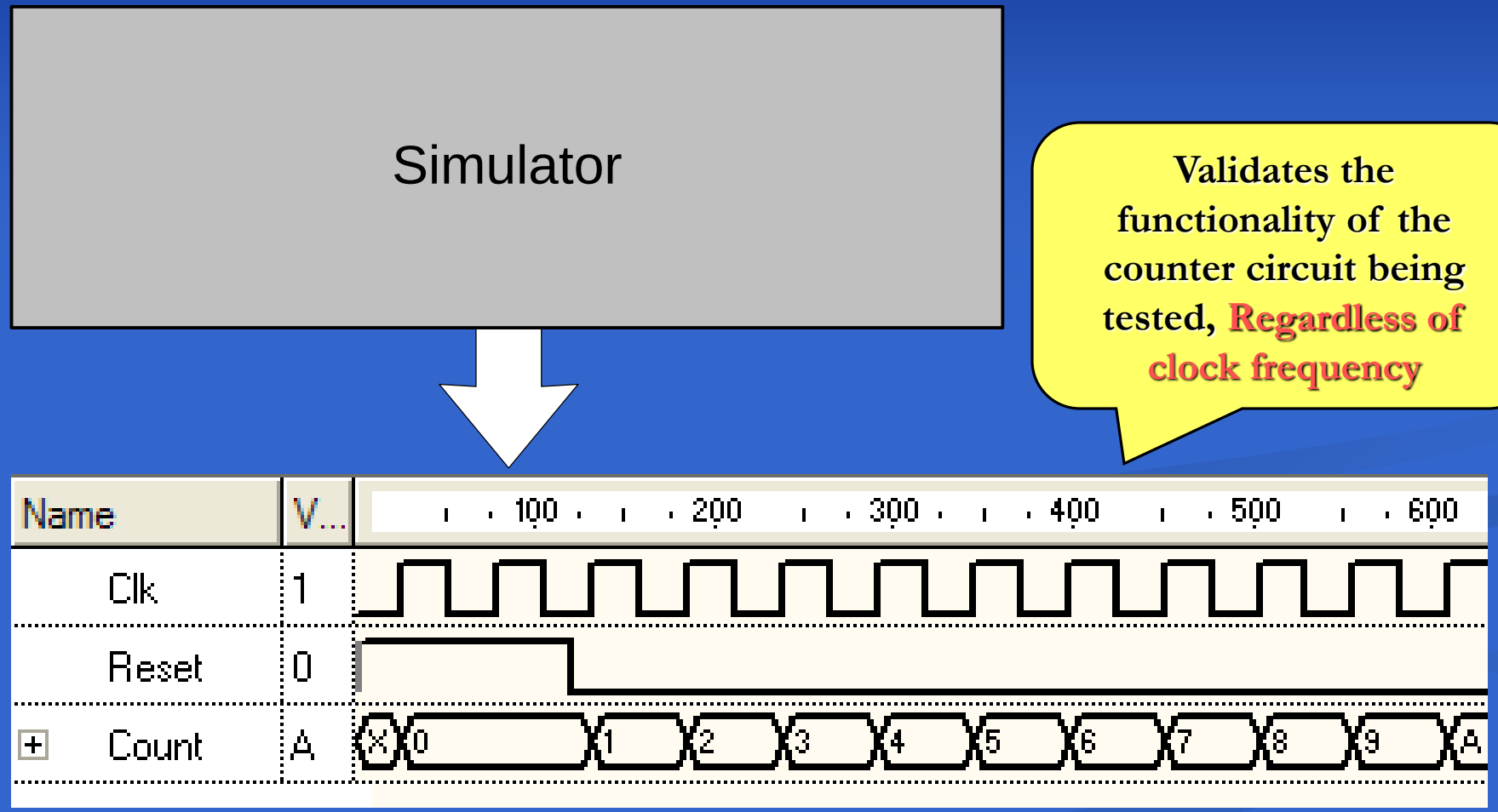
The testbench instantiates the design under test, and as part of the code of the testbench it applies test data to the instantiated circuit.

```
ENTITY counter4_tester IS END ENTITY;
ARCHITECTURE timed OF counter4_tester IS
  SIGNAL r : std_logic;
  SIGNAL c : std_logic := '0';
  SIGNAL cnt : std_logic_vector (3 DOWNTO 0);
BEGIN
  UUT1: ENTITY WORK.counter4 (procedural) PORT MAP (r, c, cnt);
  r <= '0', '1' AFTER 09 NS, '0' AFTER 17 NS,
         '1' AFTER 59 NS, '0' AFTER 67 NS;
  c <= NOT c AFTER 3.5 NS WHEN NOW <= 75 NS ELSE '0';
END ARCHITECTURE timed;
```

```
ENTITY counter4 IS
  PORT (Reset, Clk : IN std_logic;
        Count : OUT std_logic_vector (3 DOWNTO
0));
 END ENTITY;
 ARCHITECTURE procedural OF counter4 IS
  SIGNAL cnt_reg : std_logic_vector (3 DOWNTO 0);
 BEGIN
  PROCESS (Clk) BEGIN
    IF (Clk = '0' AND Clk'EVENT) THEN
      IF (Reset = '1') THEN cnt_reg <= "0000";
      ELSE cnt_reg <= cnt_reg + 1; END IF;
    END IF;
  END PROCESS;
  Count <= cnt_reg;
END ARCHITECTURE procedural;
```

- VHDL Simulation with a Testbench (Continued)

Simulation

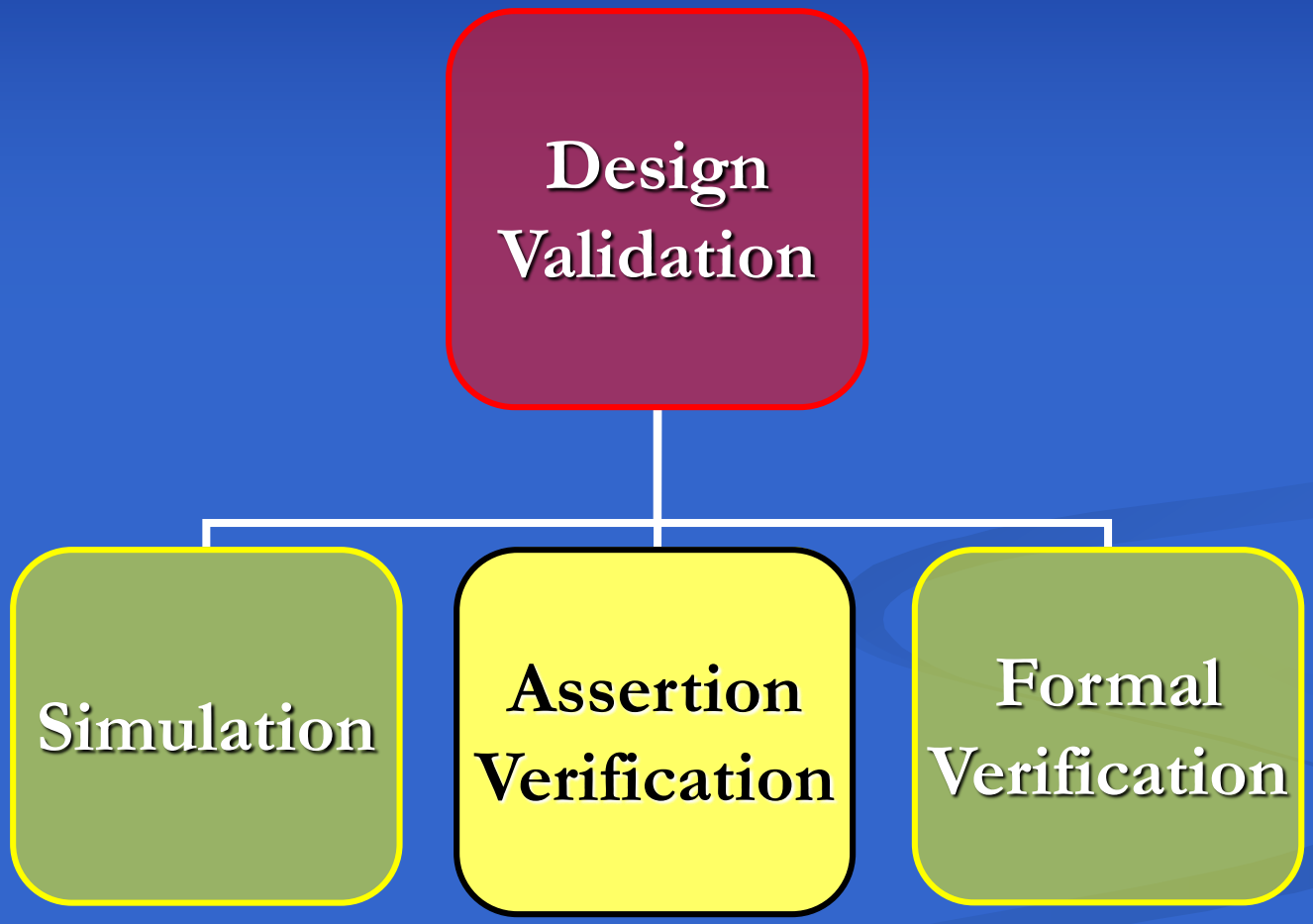


- **VHDL Simulation with a Testbench (Continued)**

Simulation

- Obviously, an actual hardware component behaves differently.
- Based on the timing and delays of the parts used, there will be a nonzero delay between the active edge of the clock and the counter output.
- Furthermore, if the clock frequency applied to an actual part is too fast for propagation of values within the gates and transistors of a design, the output of the design becomes unpredictable.
- The simulation shown here is not provided with the details of the timing of the hardware being simulated.
- Therefore, potential timing problems of the hardware that are due to gate delays cannot be detected.
- This is typical of a presynthesis or high-level behavioral simulation.

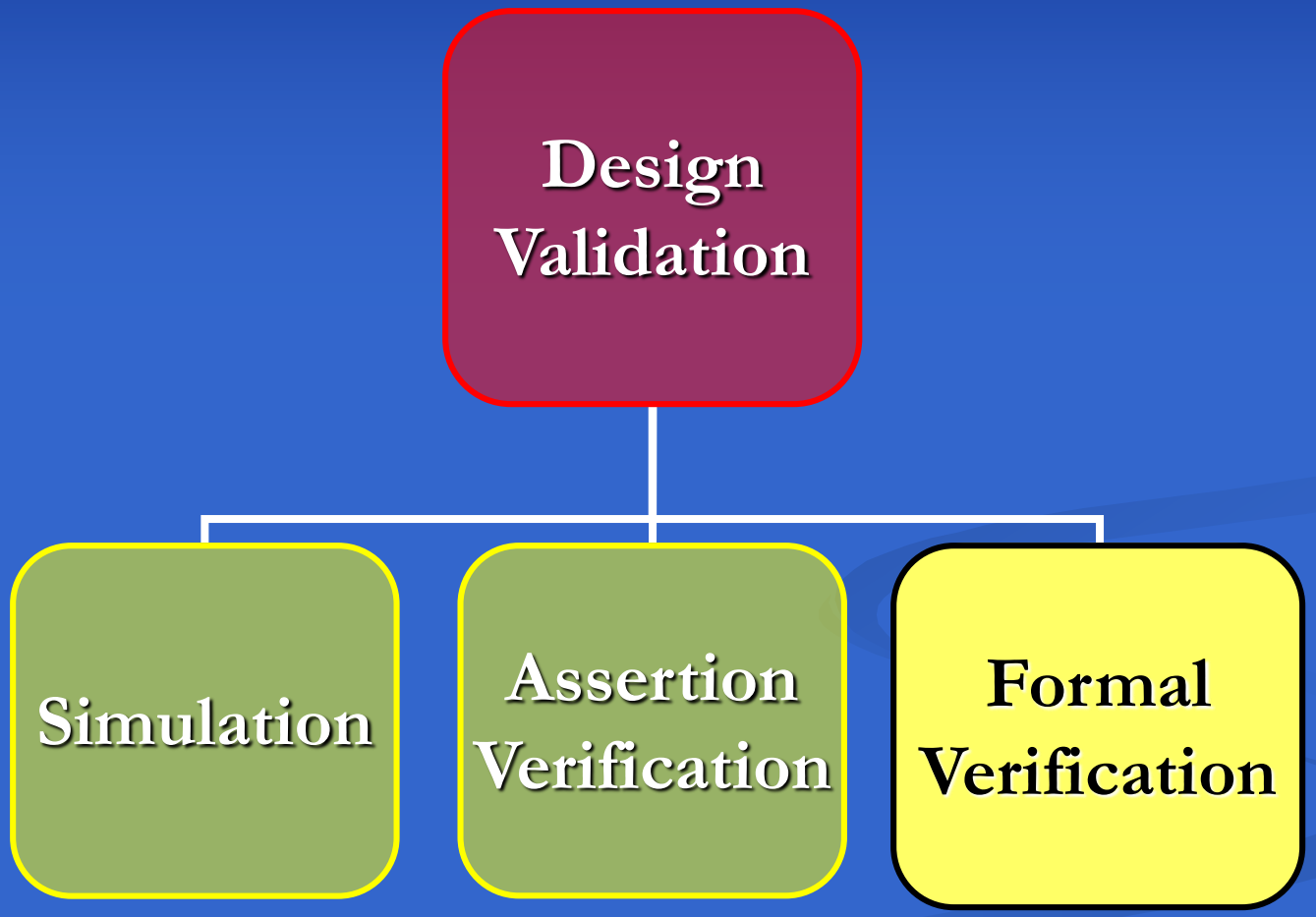
Assertion Verification



Assertion Verification

- **Aassertion Monitors:** Used to continuously check for design properties during simulation
- Instead of having to inspect simulation results manually or by developing sophisticated testbenches
- **Design Properties:** Certain conditions have to be met for the design to function correctly
- Assertion Monitors developed to assert that the Design Properties are not violated
- Firing of an assertion verification: alerts the malfunctioning of design according to the designer's expectation
- **Open verification library (OVL):** provides a set of assertion monitors for monitoring common design properties

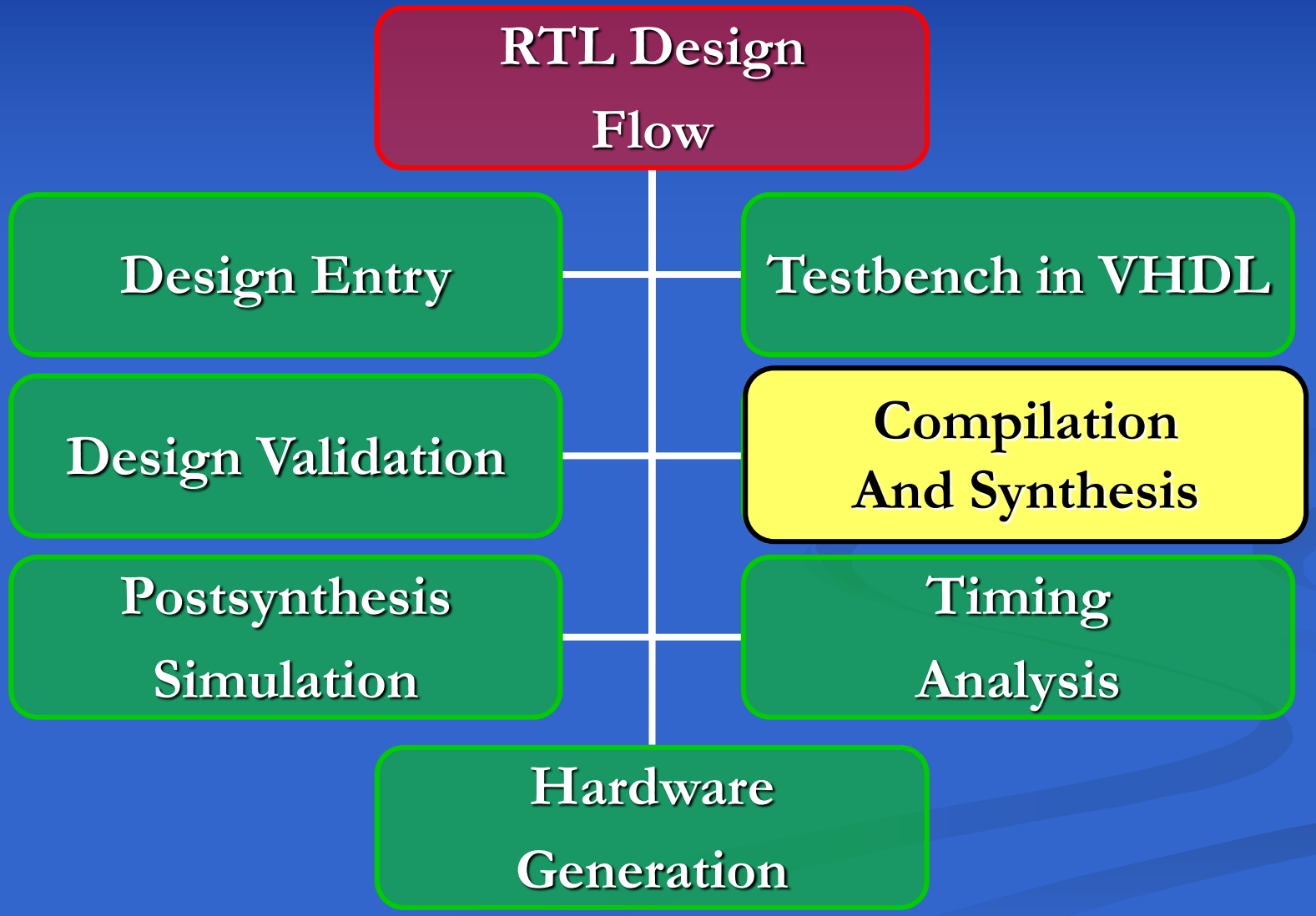
Formal Verification



Formal Verification

- **Formal verification:** The process of checking a design against certain properties
- Examining the design to make sure that the described properties by the designer to reflect correct behavior of the design hold under all conditions
- **Property's Counter Examples:** Input conditions making a property to fail
- Property coverage indicates how much of the complete design is exercised by the property

Compilation and Synthesis

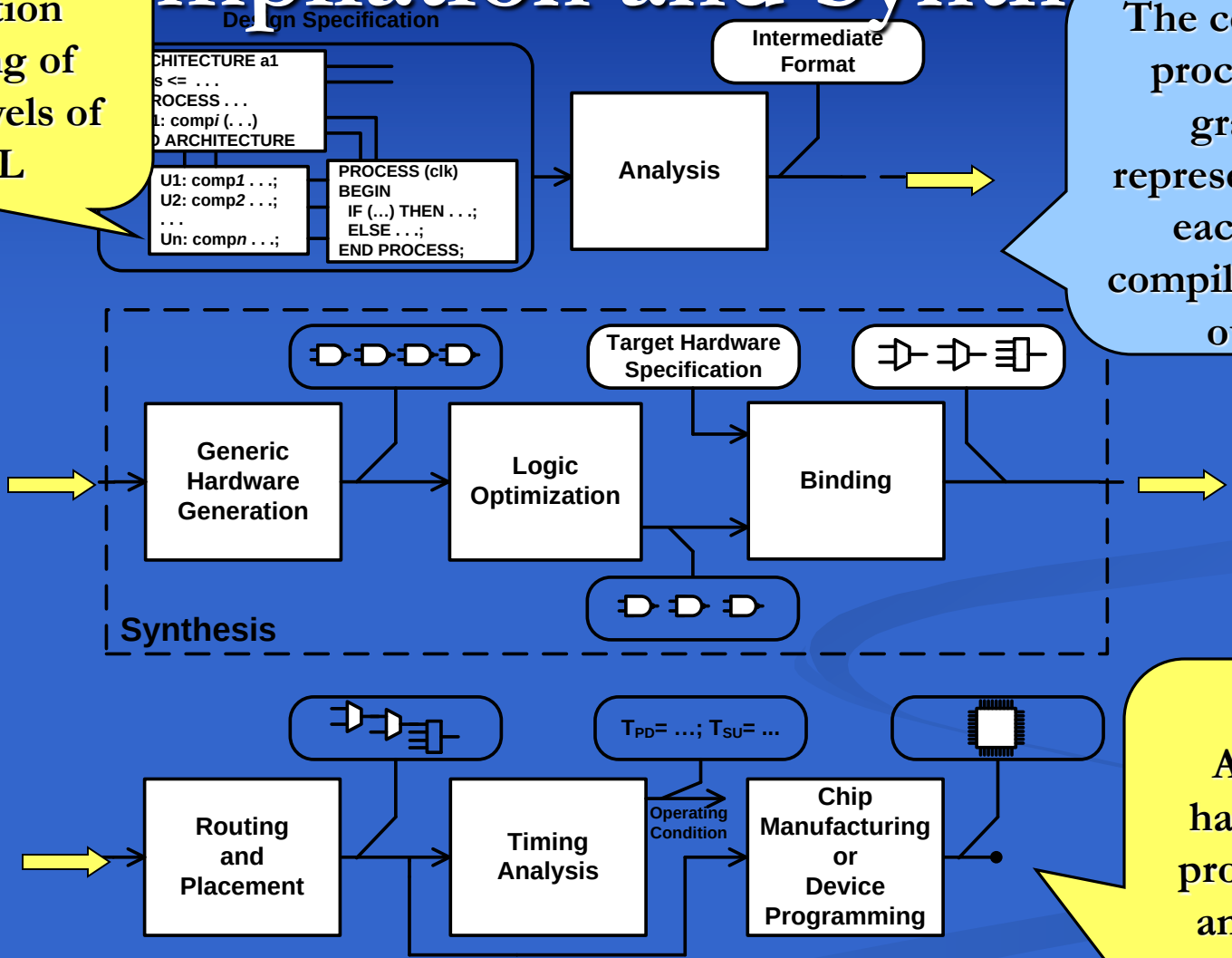


Compilation and Synthesis

- Synthesis: The process of automatic hardware generation from a design description that has an unambiguous hardware correspondence.
- A VHDL description for synthesis:
  - Cannot include signal and gate level timing specifications, file handling, and other language constructs that do not translate to sequential or combinational logic equations
  - Must follow certain styles of coding for combinational and sequential circuits
- Compilation process has three phases:
  - Analysis Phase
  - Synthesis Phase
  - Routing and Placement Phase

mpilation and Synthe

Input: Hardware description consisting of various levels of VHDL

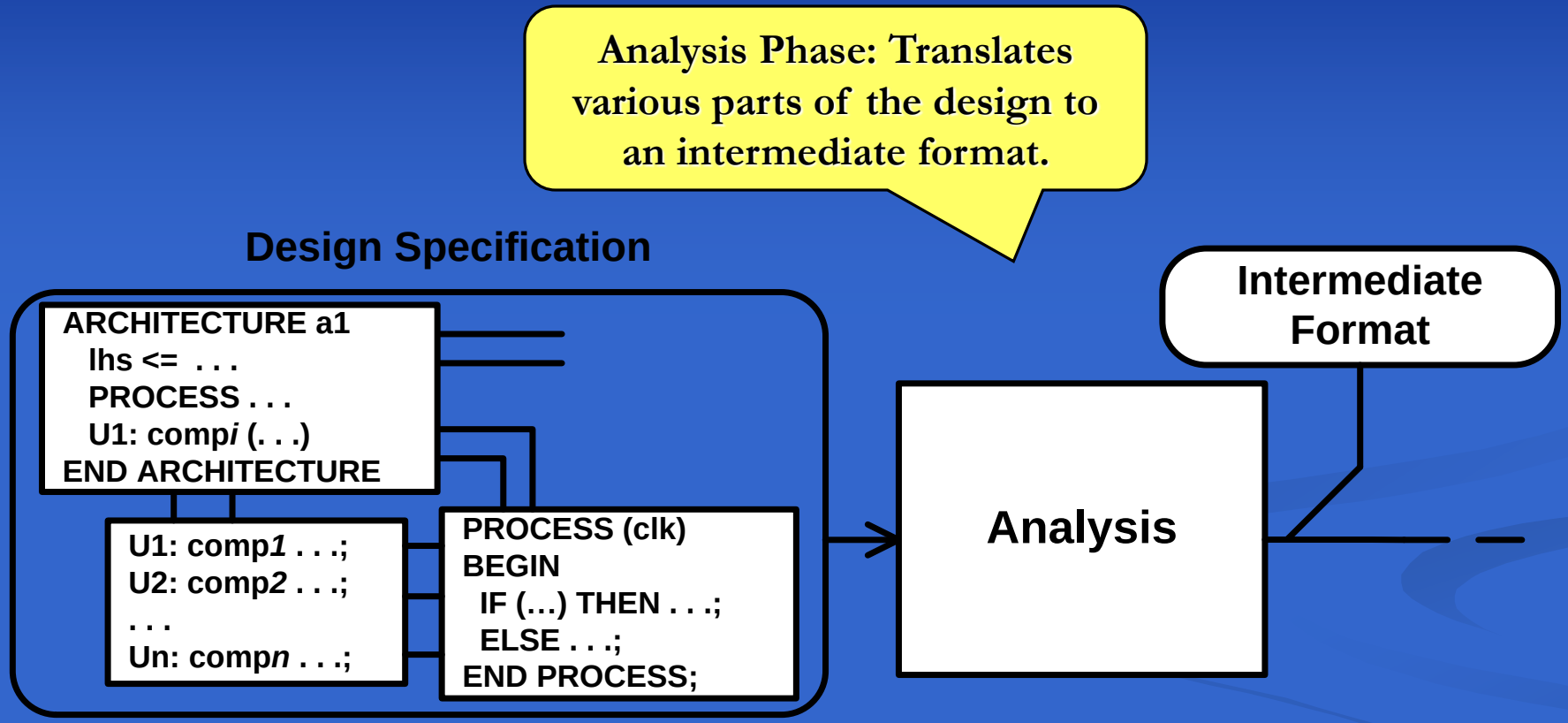


The compilation process and a graphical representation for each of the compilation phase outputs

Output: A detailed hardware for programming an FPLD or manufacturing an ASIC

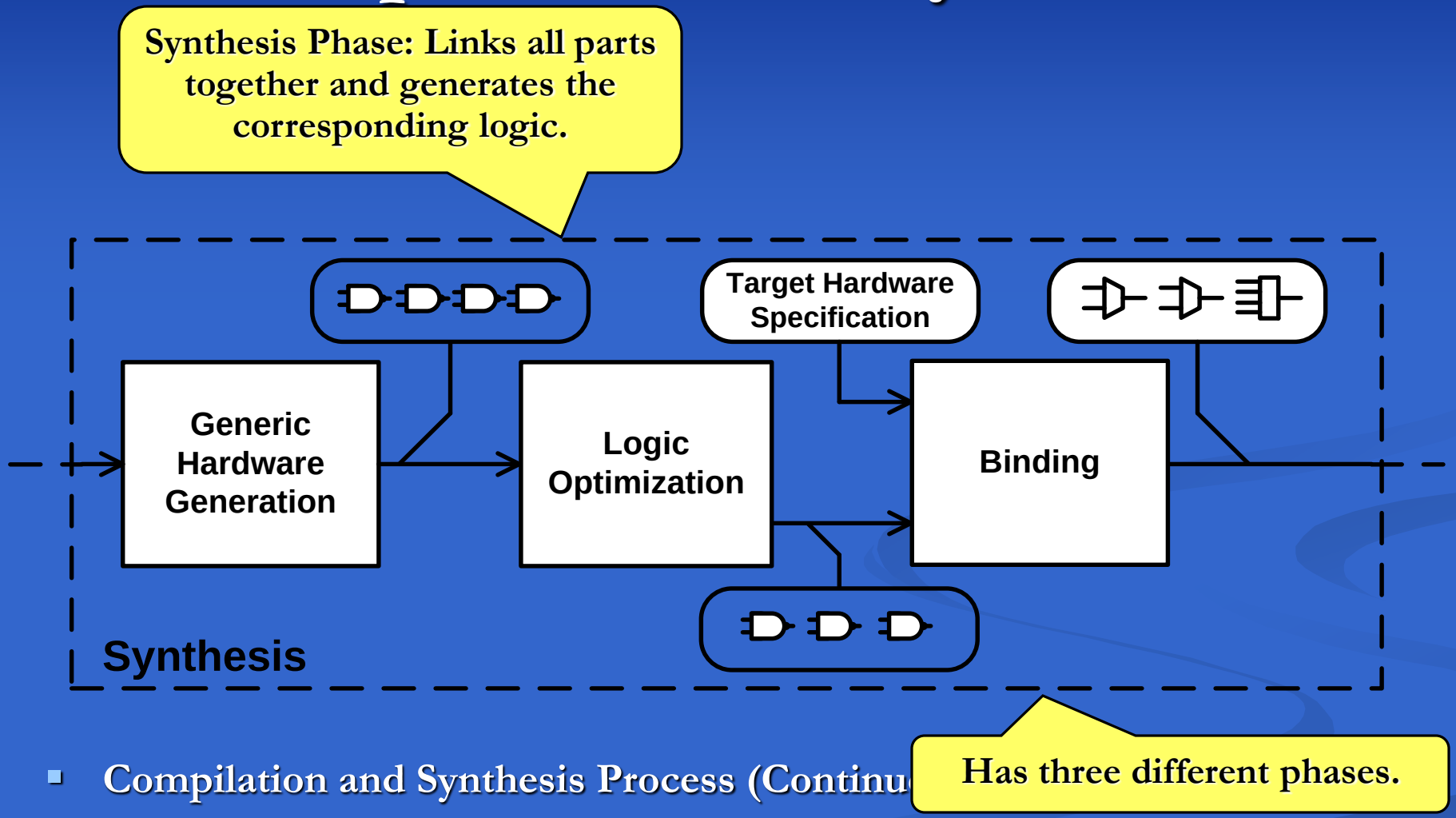
- Compilation and Synthesis Process

Compilation and Synthesis



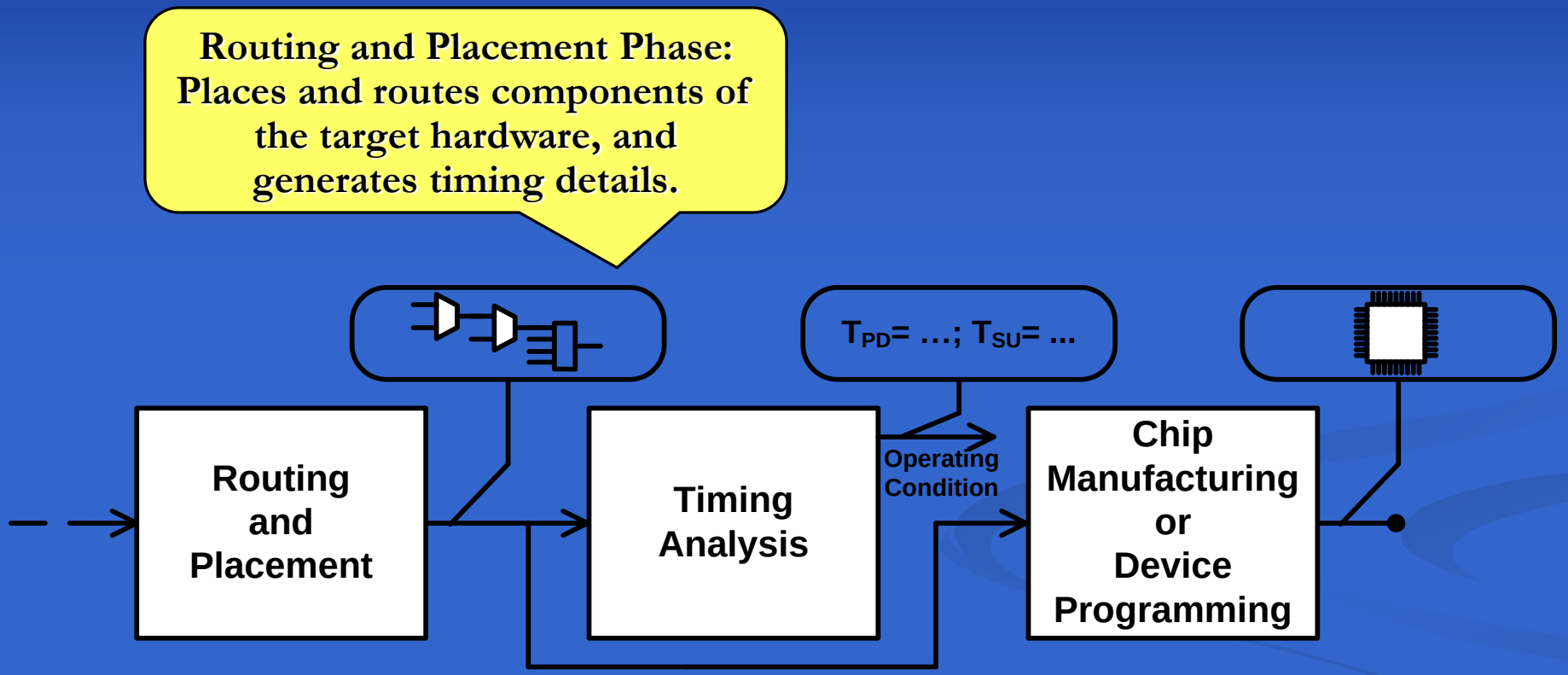
- Compilation and Synthesis Process (Continued)

Compilation and Synthesis



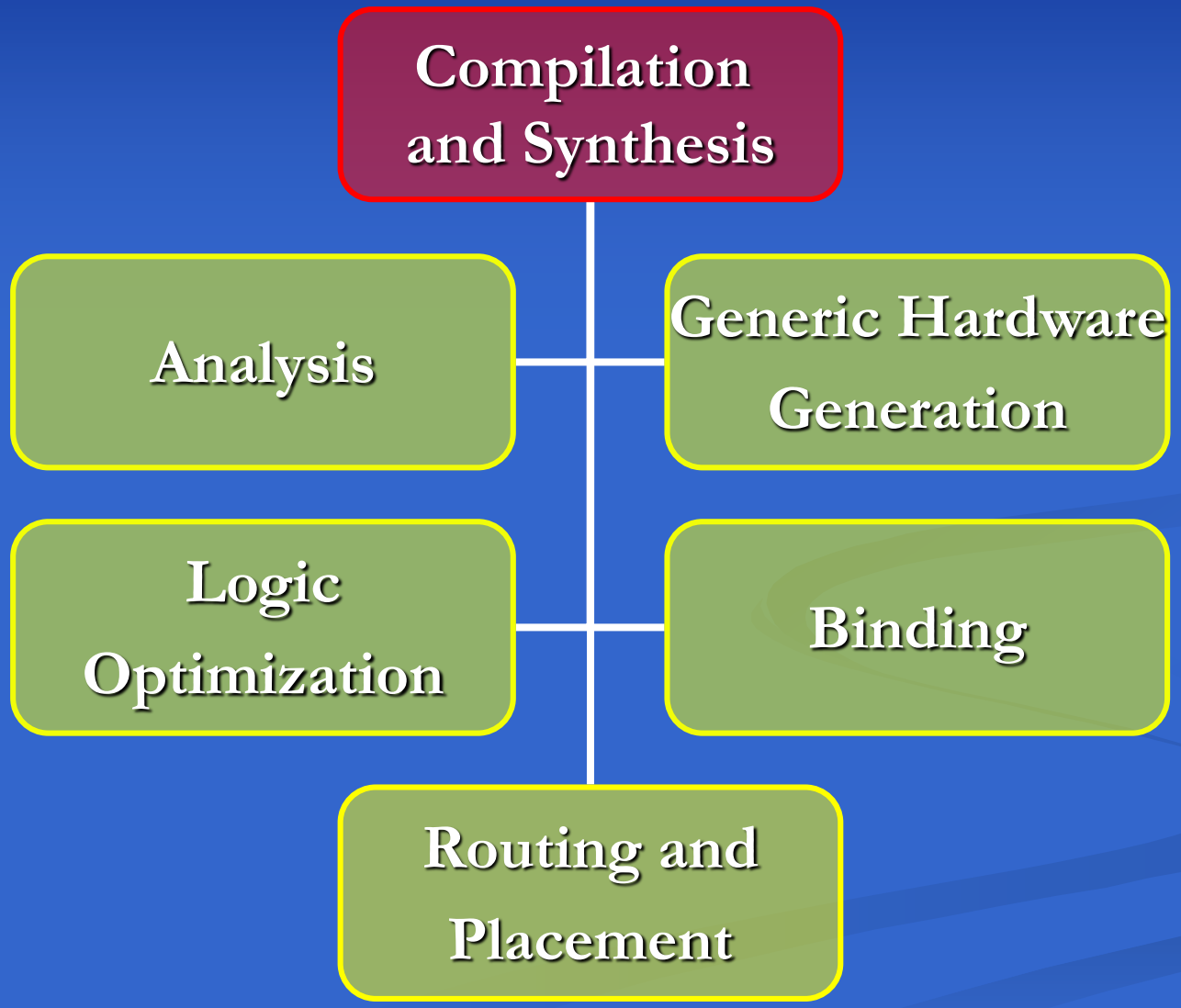
- Compilation and Synthesis Process (Continu

Compilation and Synthesis

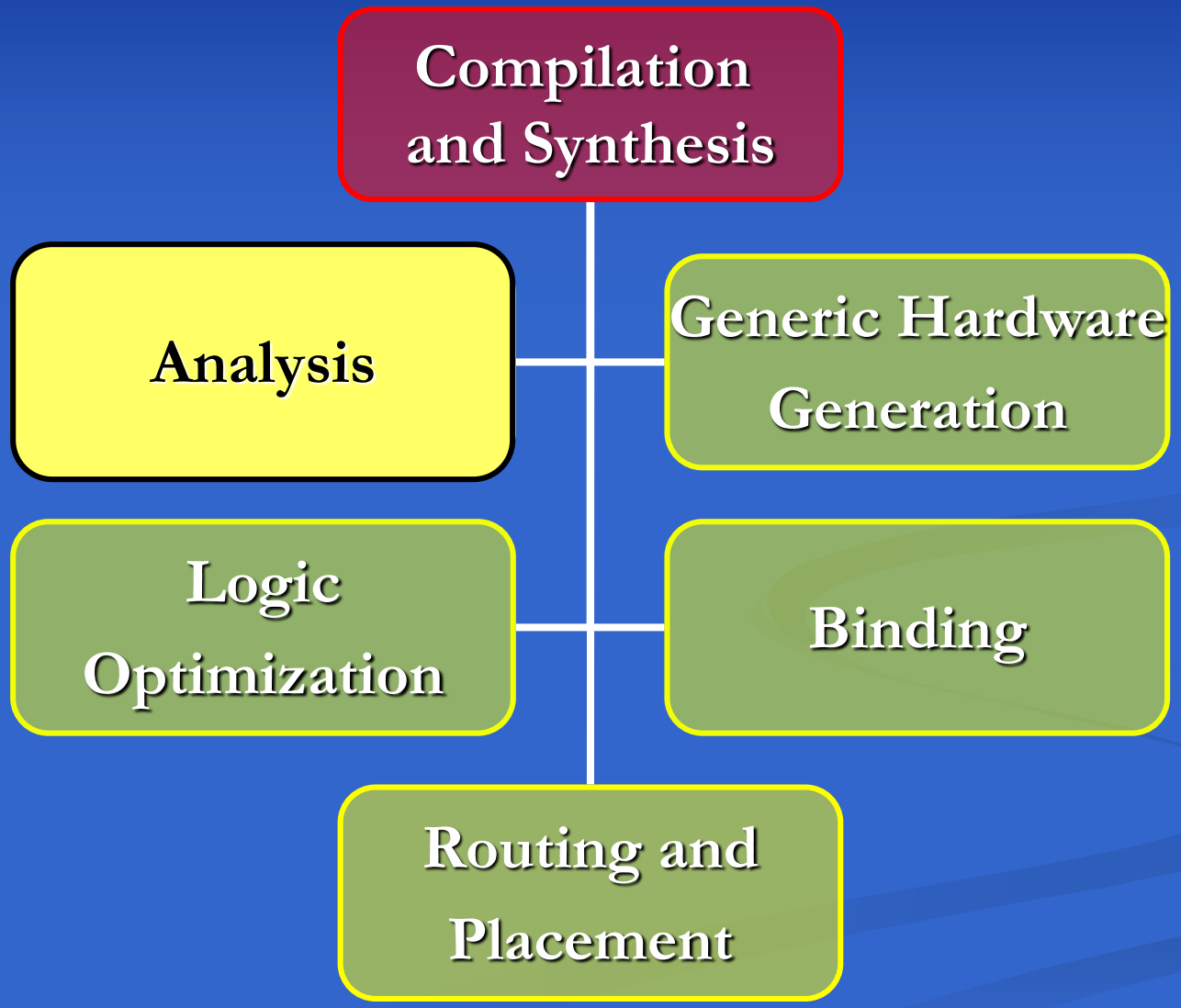


- Compilation and Synthesis Process (Continued)

Compilation and Synthesis



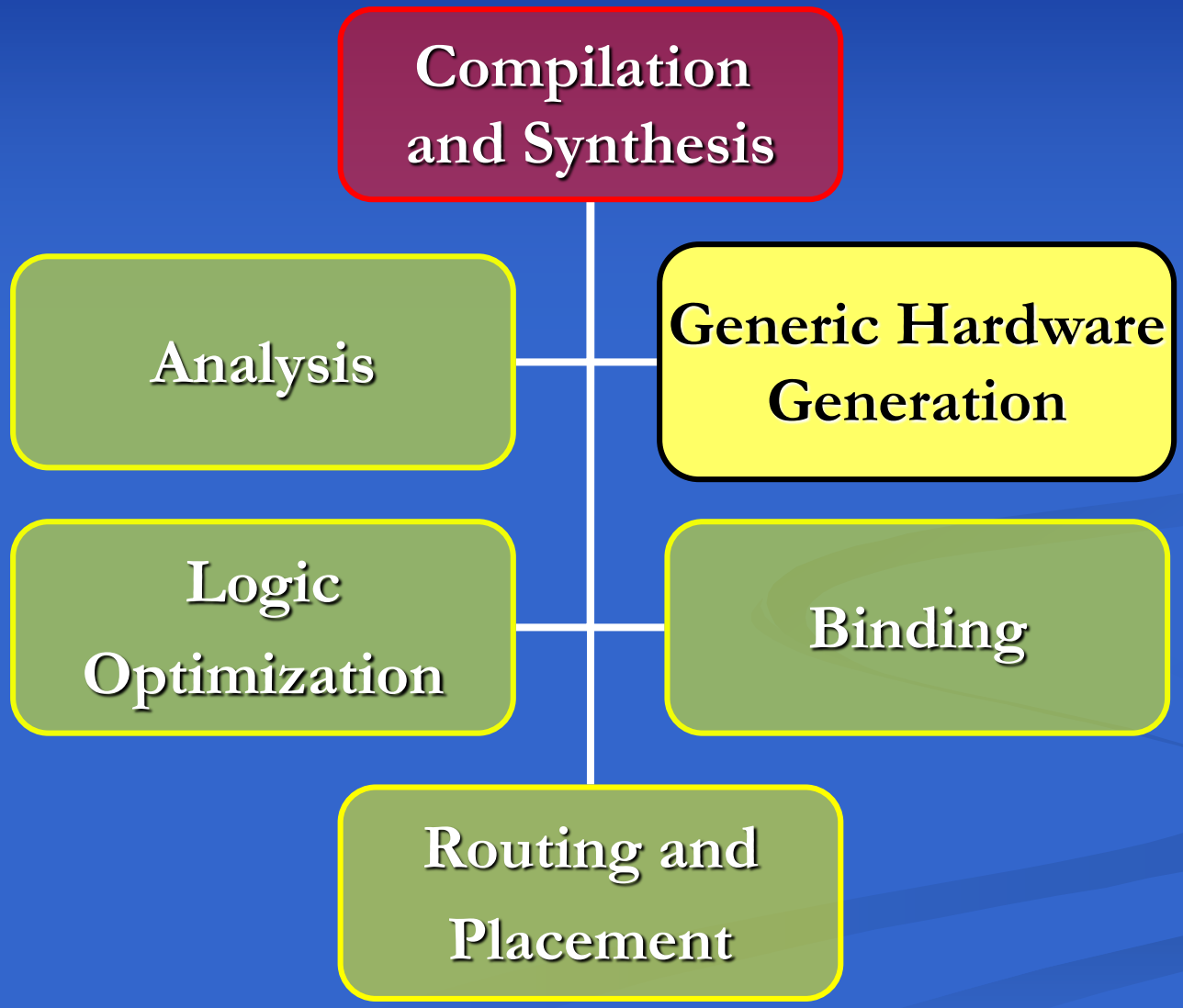
Analysis


Compilation and Synthesis
Analysis
Generic Hardware Generation
Logic Optimization
Binding
Routing and Placement

Analysis

- Before the complete design is turned into hardware
- Analyzing the design and generating a uniform format for all parts of it
- Also checks the syntax and semantics of the input VHDL code

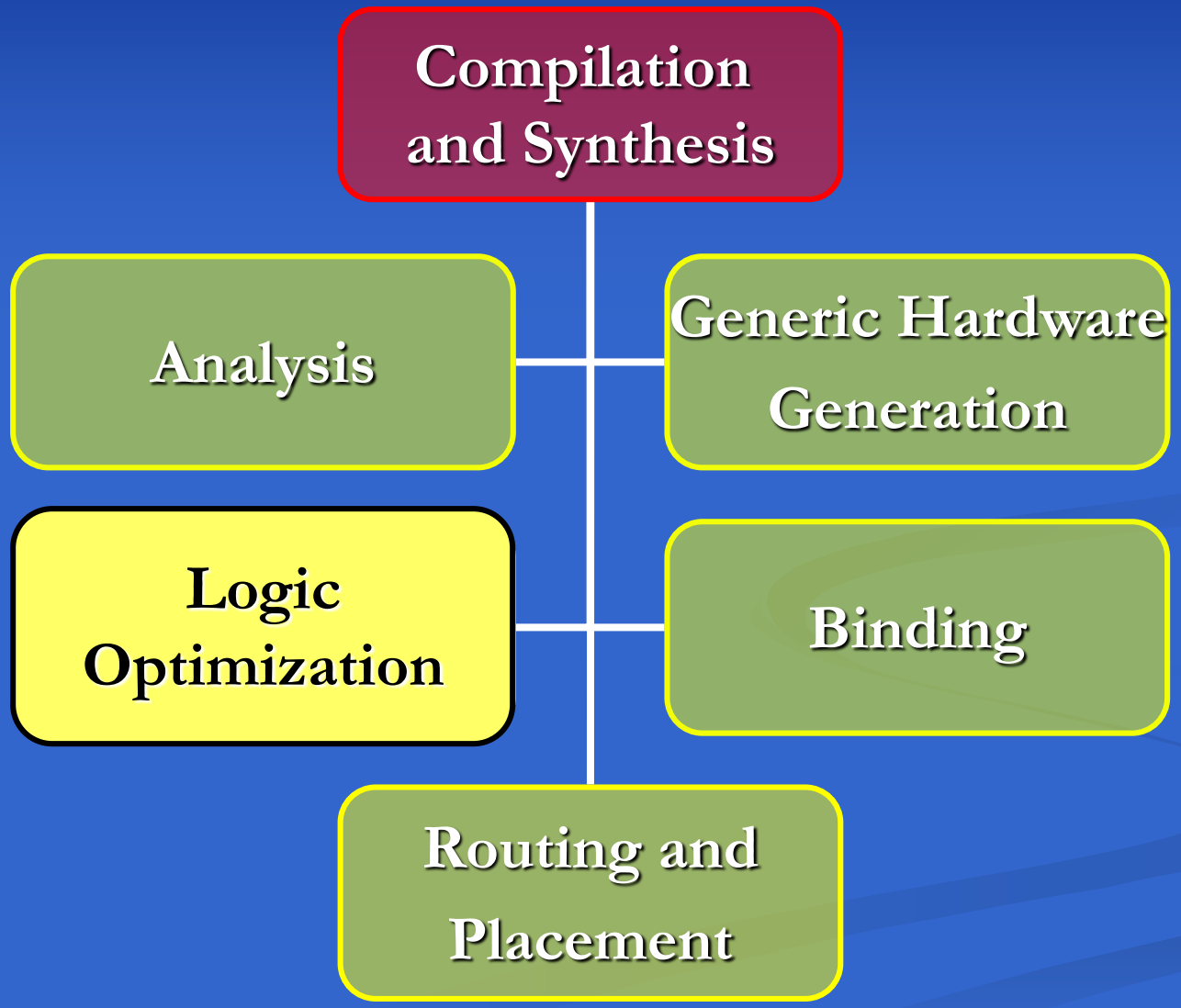
Generic Hardware Generation



Generic Hardware Generation

- **Generic Hardware Generation:** Turning the design into a generic hardware format such as a set of Boolean expressions or a netlist of basic gates

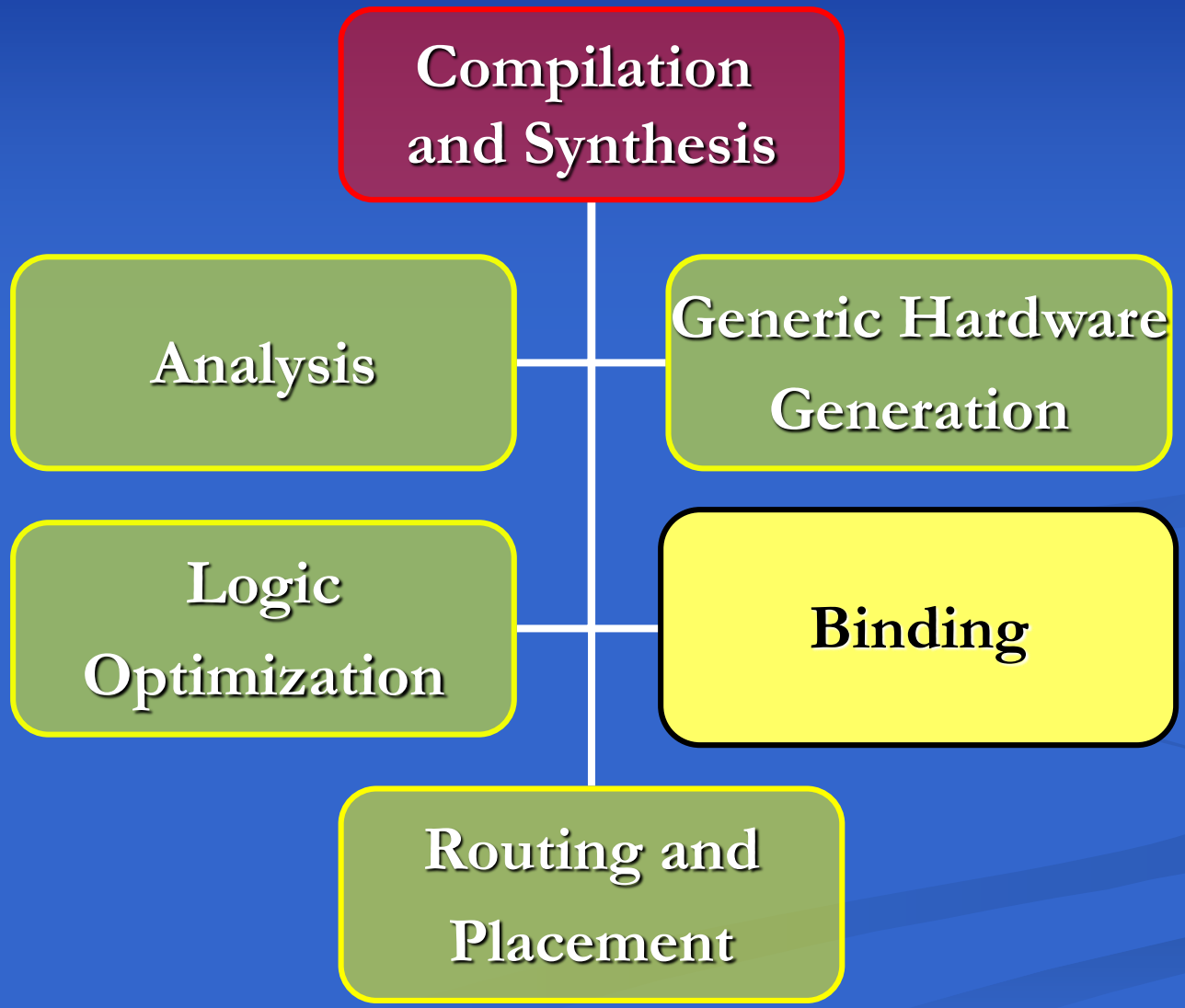
Logic Optimization



Logic Optimization

- **Logic Optimization:**
  - Reducing expressions with constant input
  - Removing redundant logic expressions
  - Two-level minimization
  - Multilevel minimization that include logic sharing
  - Output:
    - Boolean expressions
    - Tabular logic representations
    - Primitive gate netlists

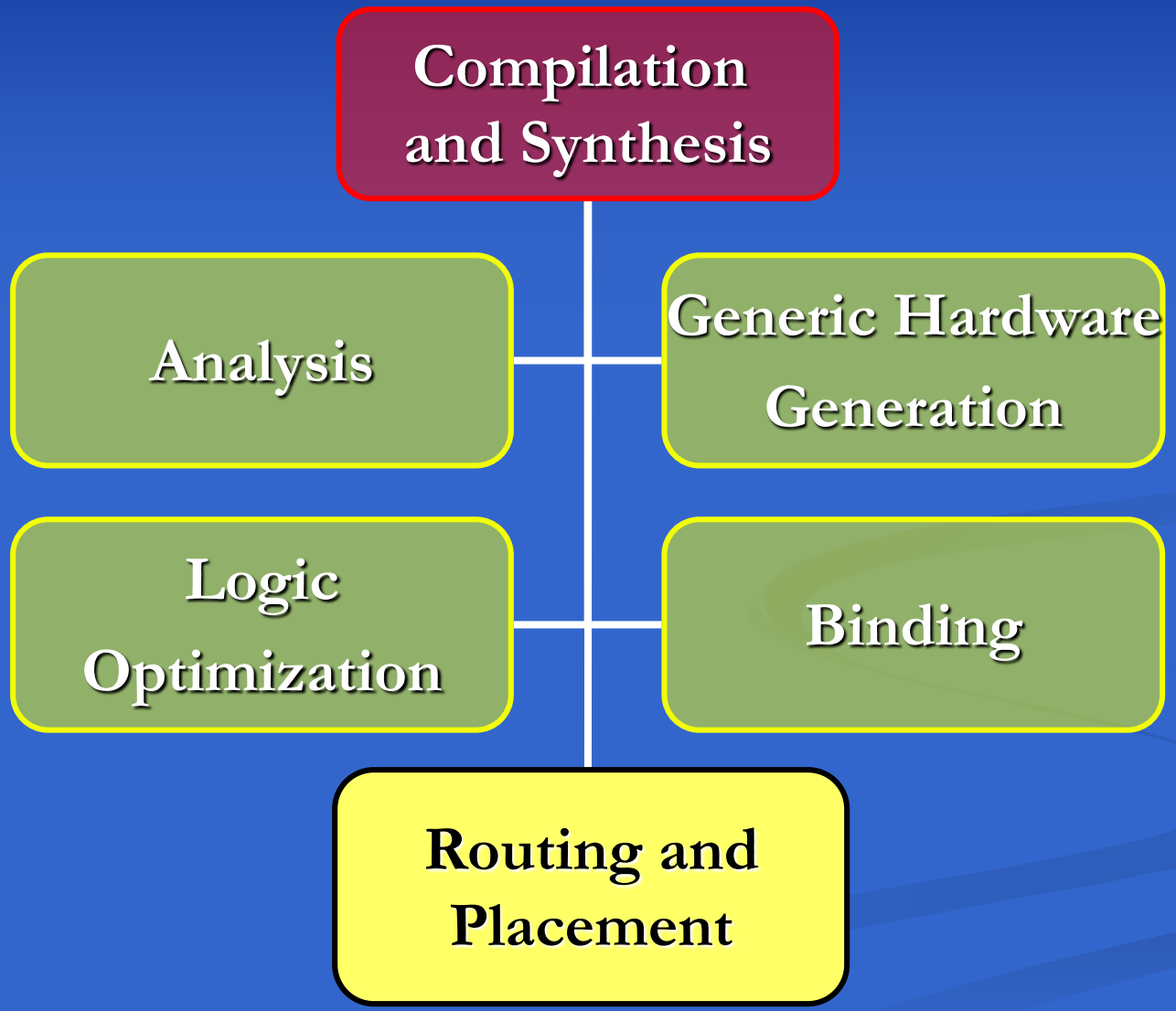
Binding


Compilation and Synthesis
Analysis
Generic Hardware Generation
Logic Optimization
Binding
Routing and Placement

Binding

- **Binding:**
  - Decide exactly what logic elements and cells are needed for the realization of the circuit using information from target hardware
  - Output is specific to the FPLD, ASIC, or custom IC being used

Routing and Placement



Routing and Placement

- Decides on the placement of cells of the target hardware
- Determines wiring of inputs and outputs of the cells through wiring channels and switching areas of the target hardware
- The output is specific to the hardware being used and can be used for programming an FPLD or manufacturing an ASIC.

Routing and Placement

```
ENTITY counter4 IS
  PORT (Reset, Clk : IN std_logic; Count : OUT std_logic_vector (3 DOWNTO
0));
END ENTITY;
ARCHITECTURE procedural OF counter4 IS
  SIGNAL cnt_reg : std_logic_vector (3 DOWNTO 0);
BEGIN
  PROCESS (Clk) BEGIN
    IF (Clk = '0' AND Clk'EVENT) THEN
      IF (Reset = '1') THEN cnt_reg <= "0000";
      ELSE cnt_reg <= cnt_reg + 1; END IF;
    END IF;
  END PROCESS;
  Count <= cnt_reg;
END ARCHITECTURE procedural;
```

An example of a synthesis run: The counter circuit is being synthesized

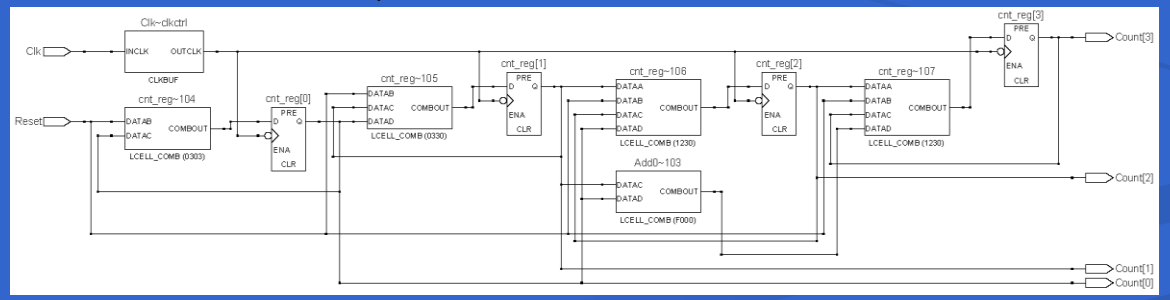
Design to Synthesize

Synthesis Tool

Target hardware specification

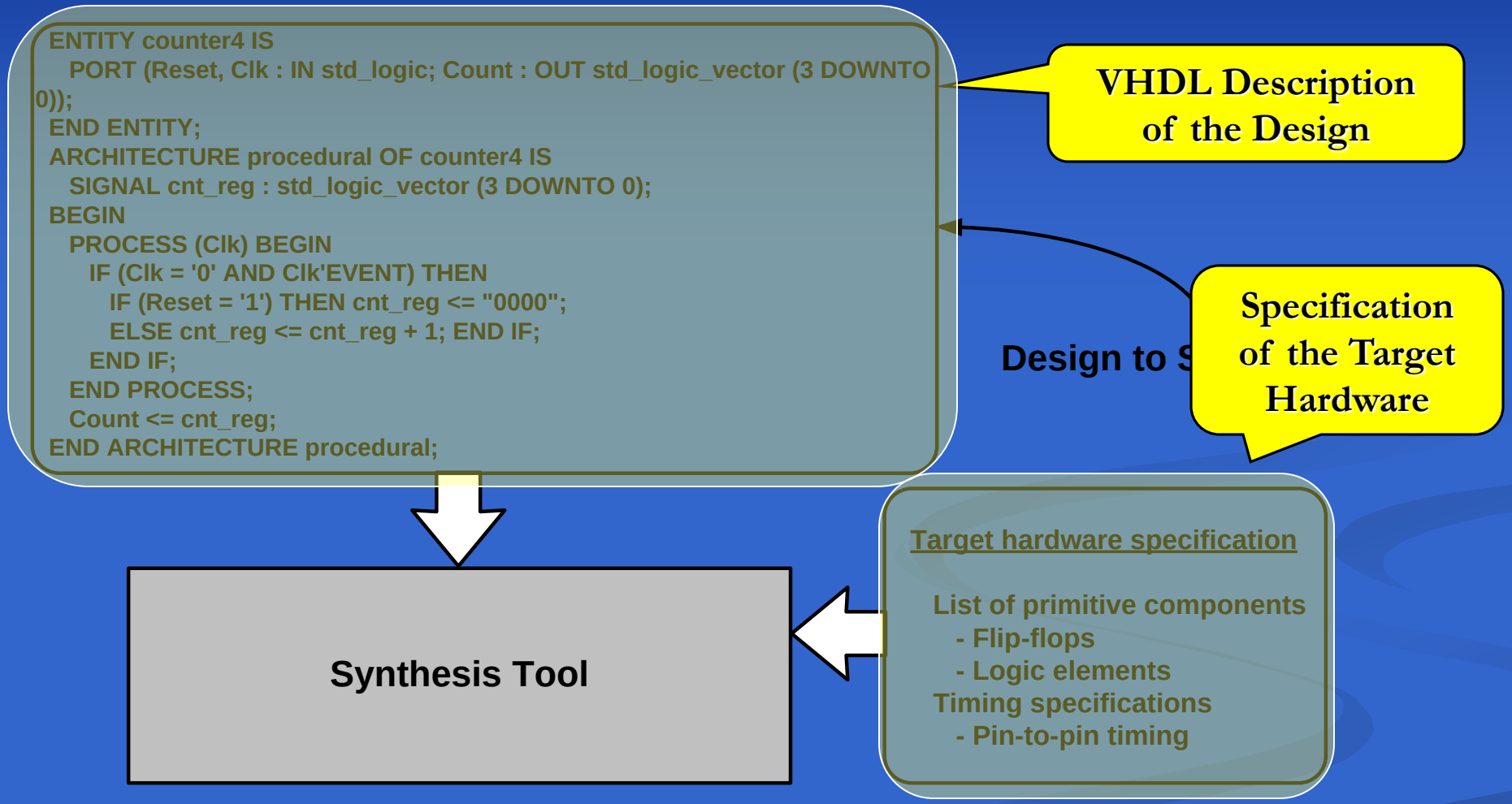
List of primitive components
- Flip-flops
- Logic elements

Timing specifications
- Pin-to-pin timing



- An Example Synthesis Run

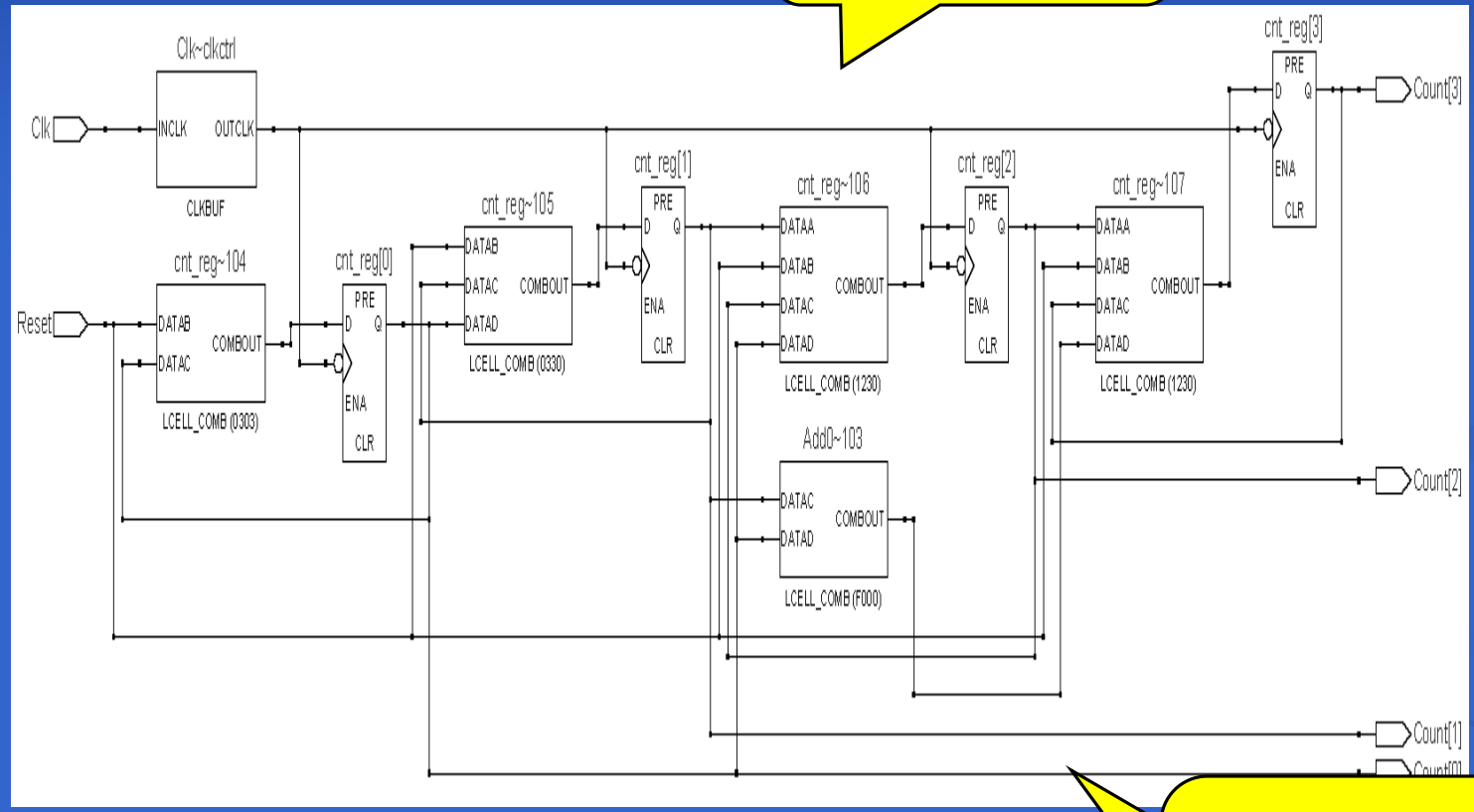
Routing and Placement



- An Example Synthesis Run (Continued)

Routing and Placement

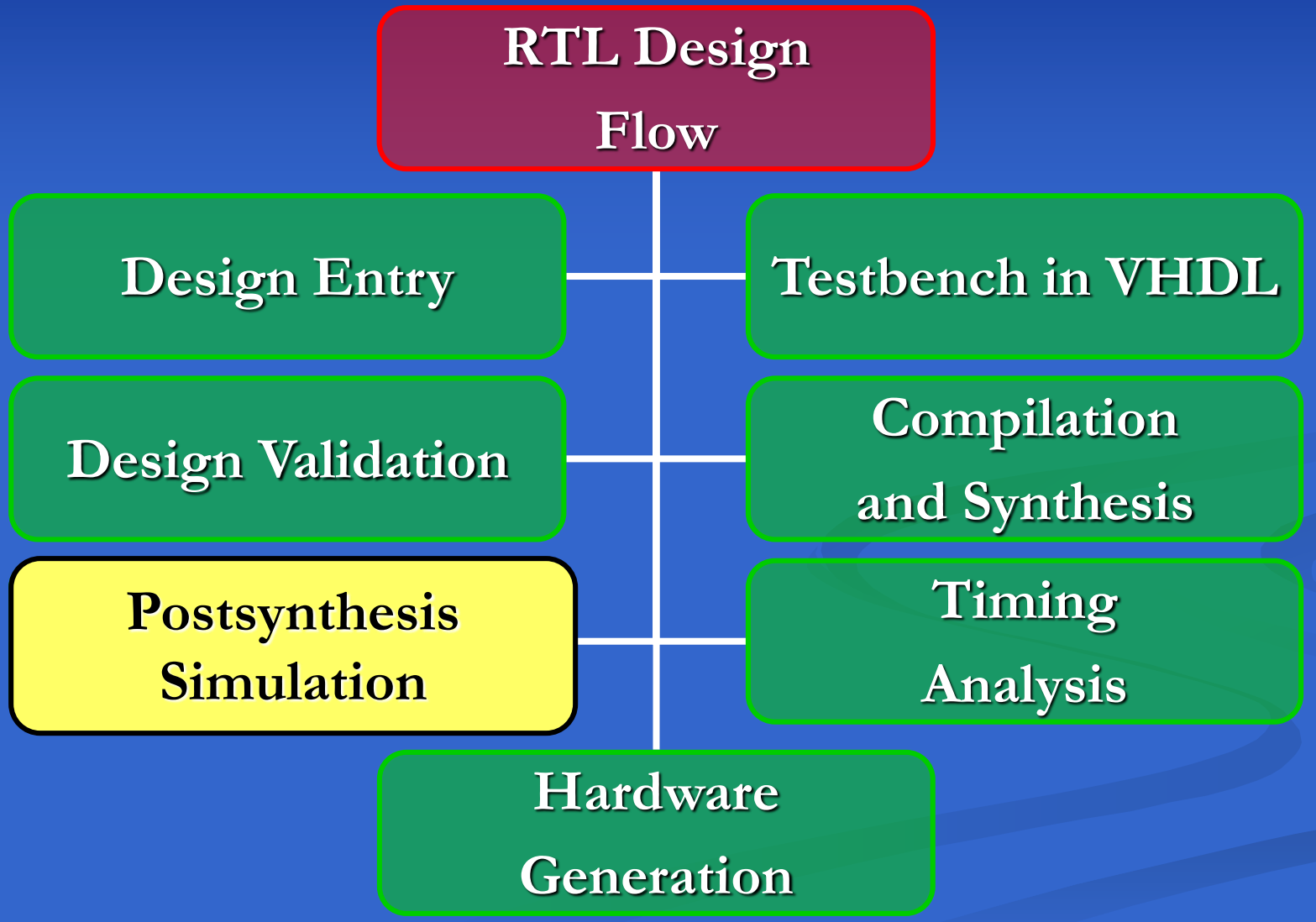
The output of synthesis tool



A list of gates and flip-flops available in the target hardware and their interconnections

- An Example Synthesis Run (Continued)

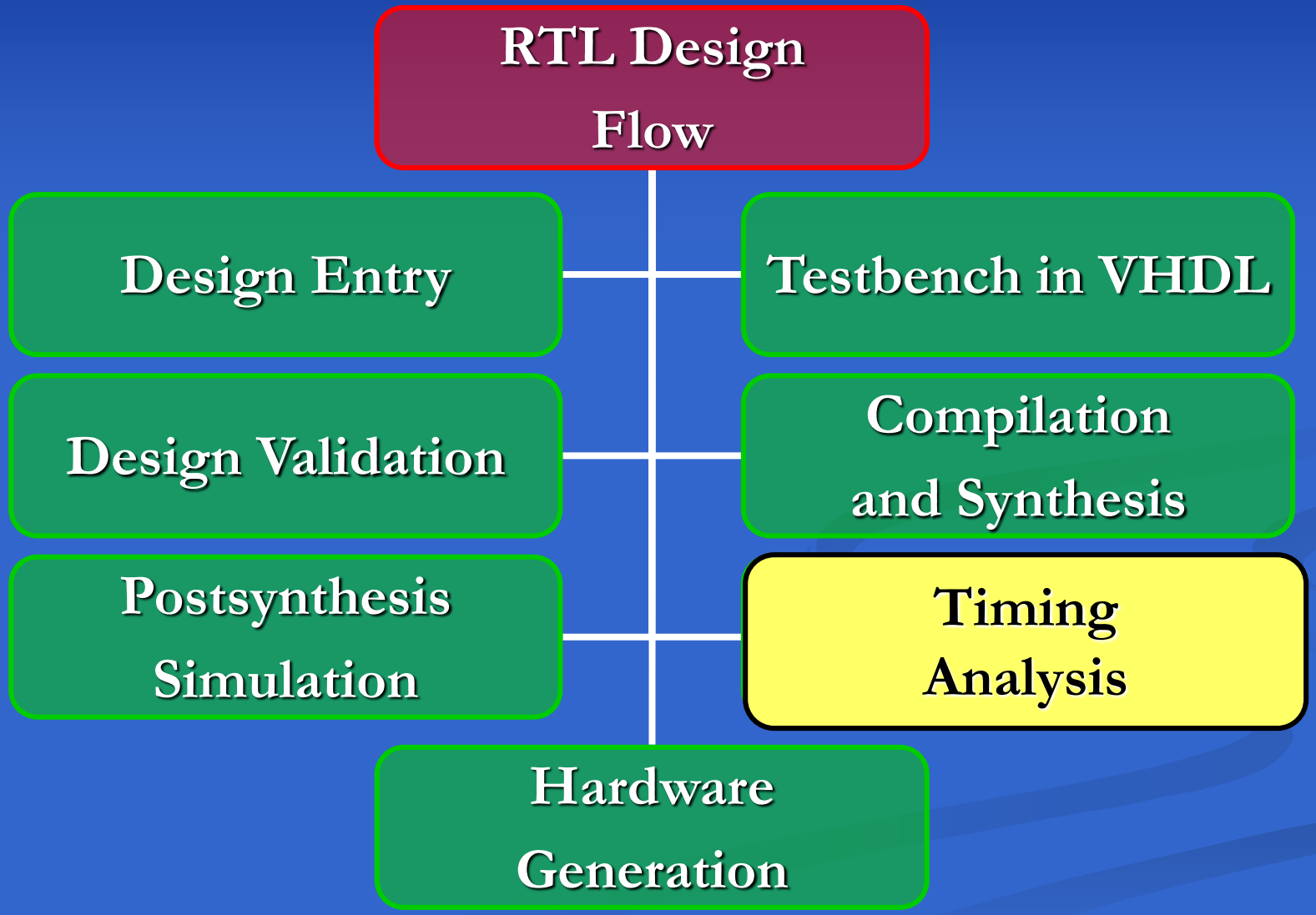
Post-synthesis Simulation



Post-synthesis Simulation

- After the Synthesis Phase a complete netlist of target hardware components and their timings is generated.
- The generated netlist includes:
  - The details of gates used for the implementation of the design
  - Wiring delays and load effects on gates used in the postsynthesis design
- The netlist output is made available in various netlist formats including VHDL
- A Postsynthesis simulation checks:
  - Timing issues
  - Determination of a proper clock frequency
  - Determination of race, and hazard considerations
- The behavior of a design as intended by the designer and its behavior after postsynthesis simulation may be different due to delays of wires and gates.

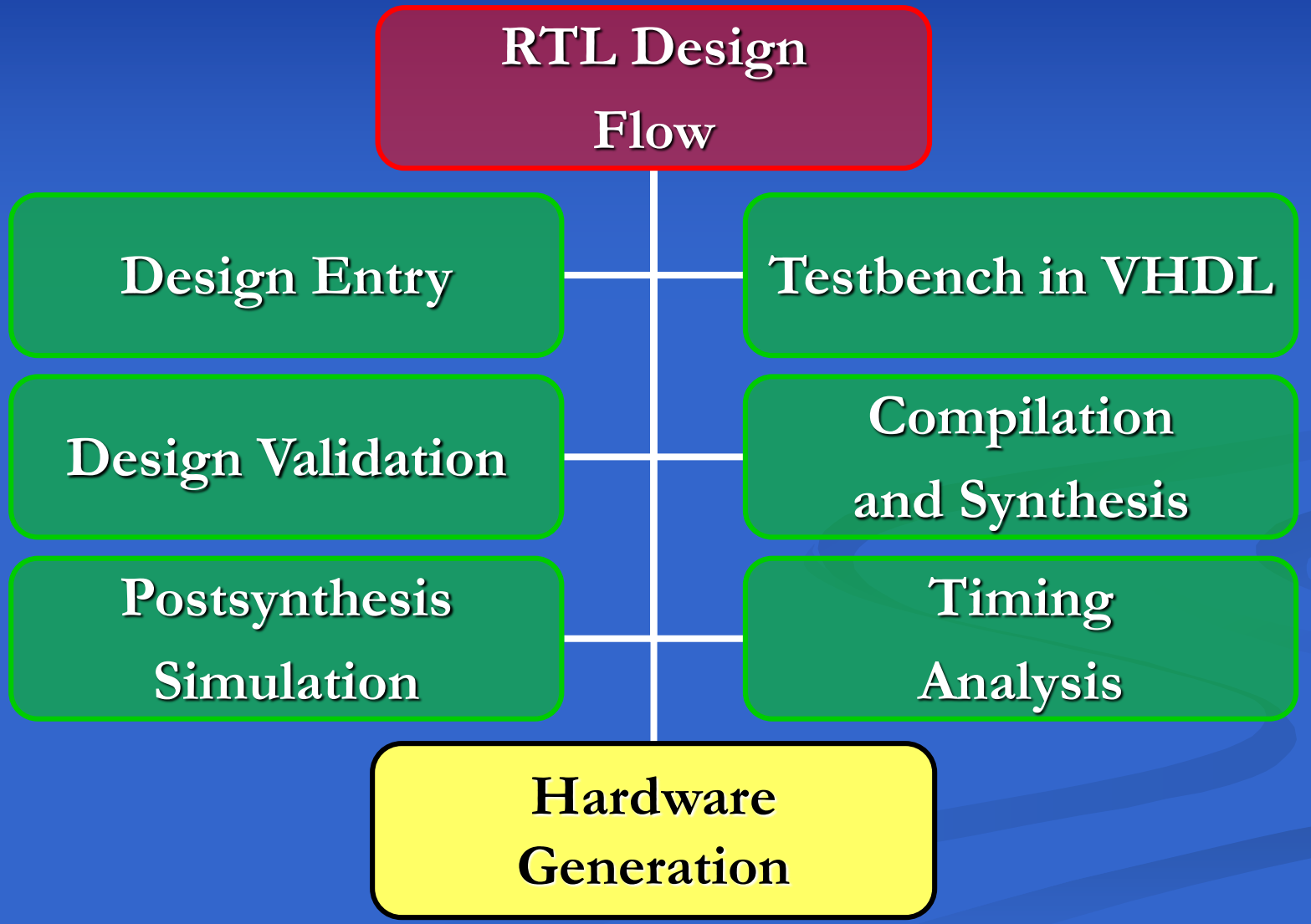
Timing Analysis



Timing Analysis

- A part of the compilation process, or in some tools after the compilation process
- Timing Analysis Phase generates:
  - Worst-case delays
  - Clocking speed
  - Delays from one gate to another
  - Required setup and hold times
- Results of timing analysis appear in Tables and/or Graphs
- The results is used by designers to decide on speed of their circuits.

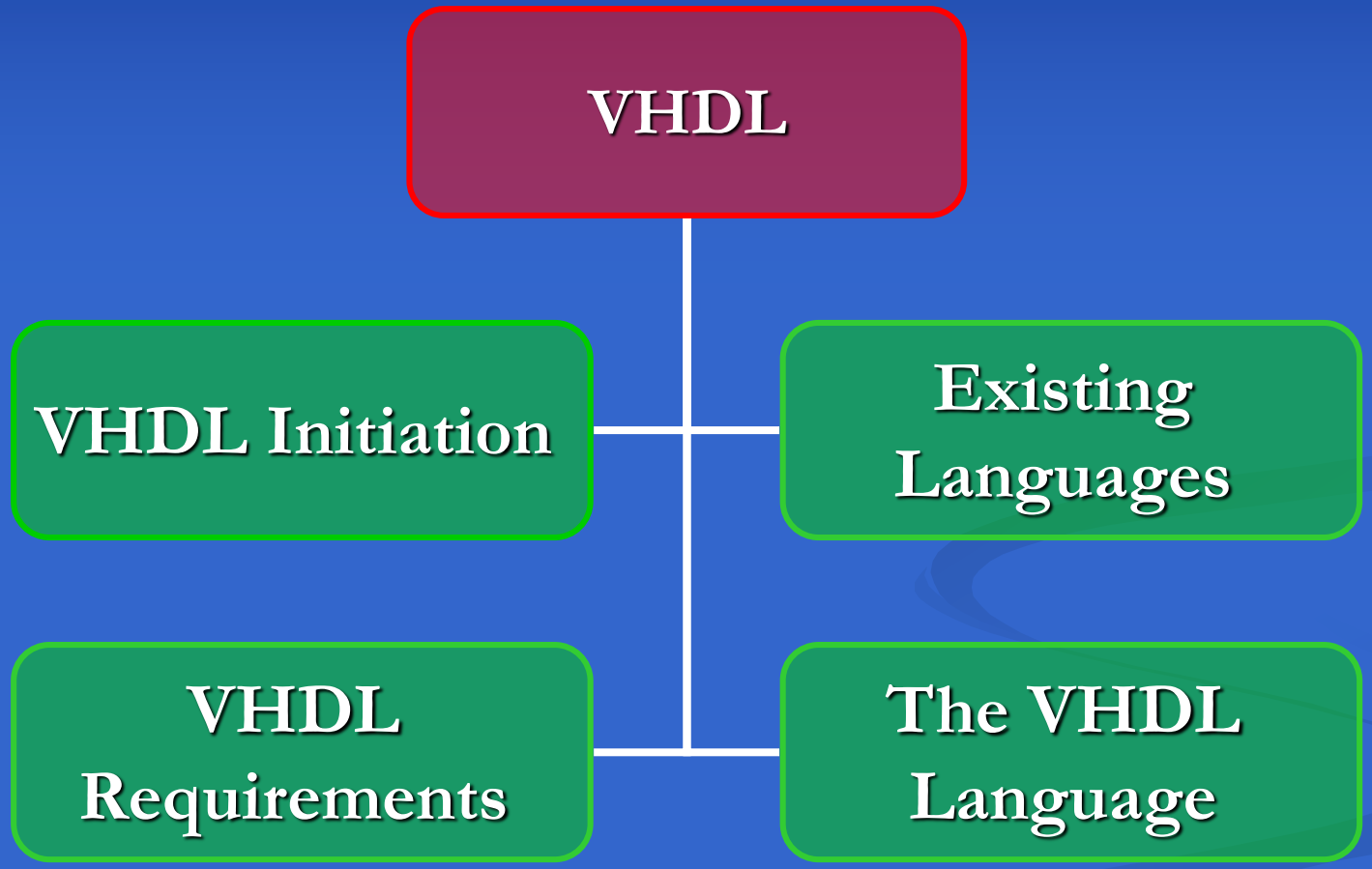
Hardware Generation



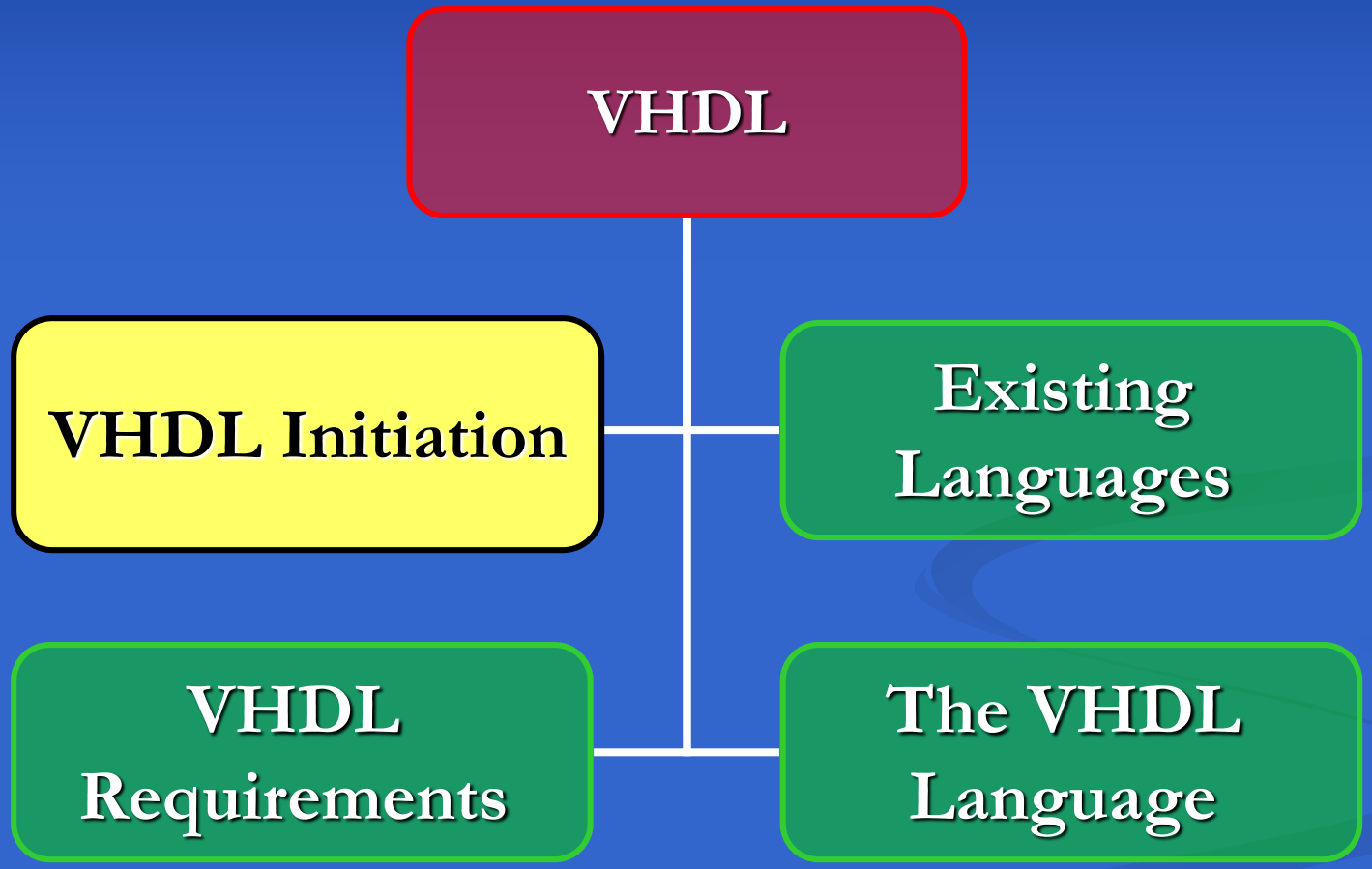
Hardware Generation

- Last stage in an automated VHDL-based design
- Generates a netlist for ASIC manufacturing, a program for programming FPLDs, or layout of custom IC cells

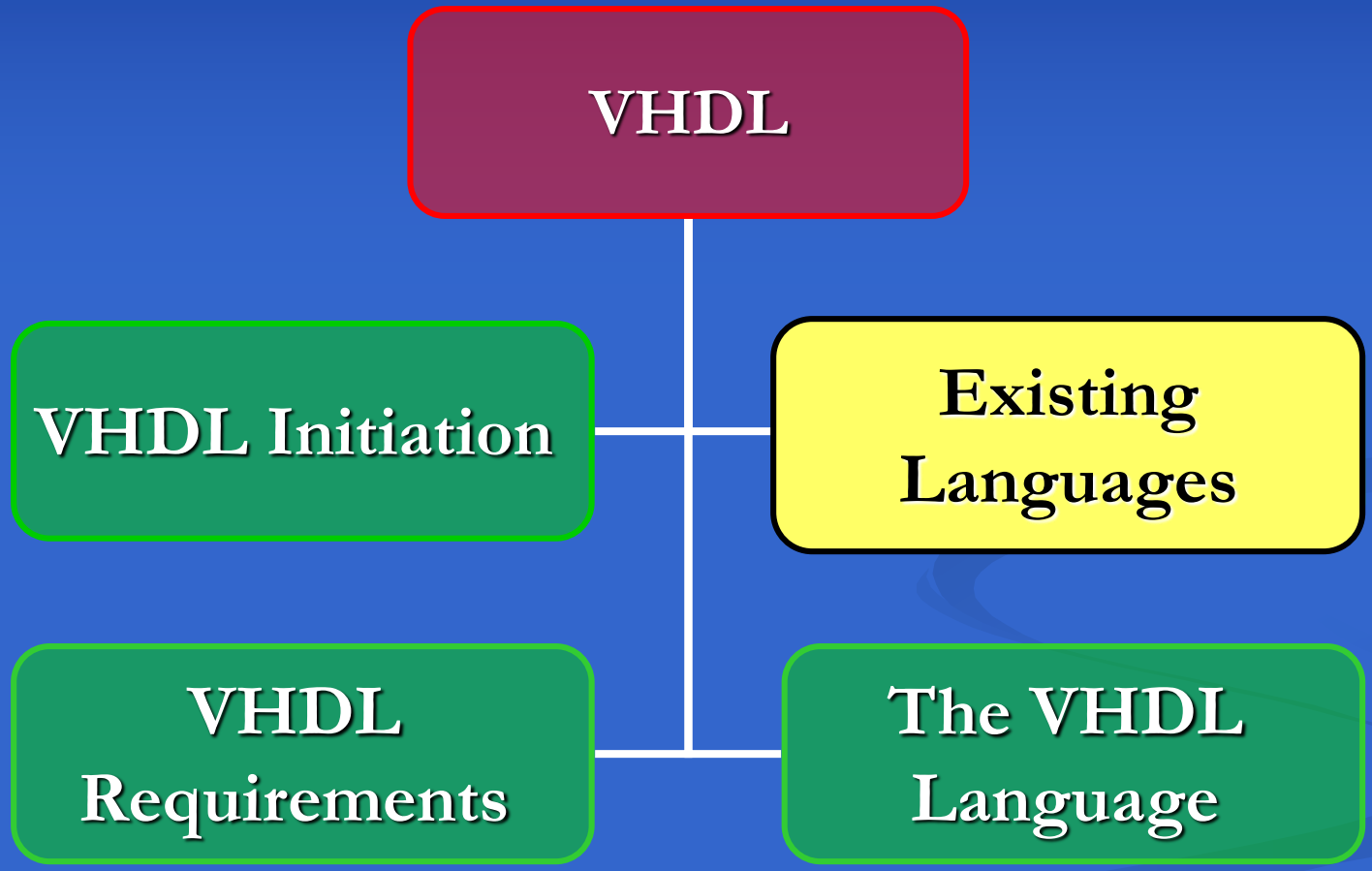
VHDL


VHDL
VHDL Initiation
Existing Languages
VHDL Requirements
The VHDL Language

VHDL Initiation



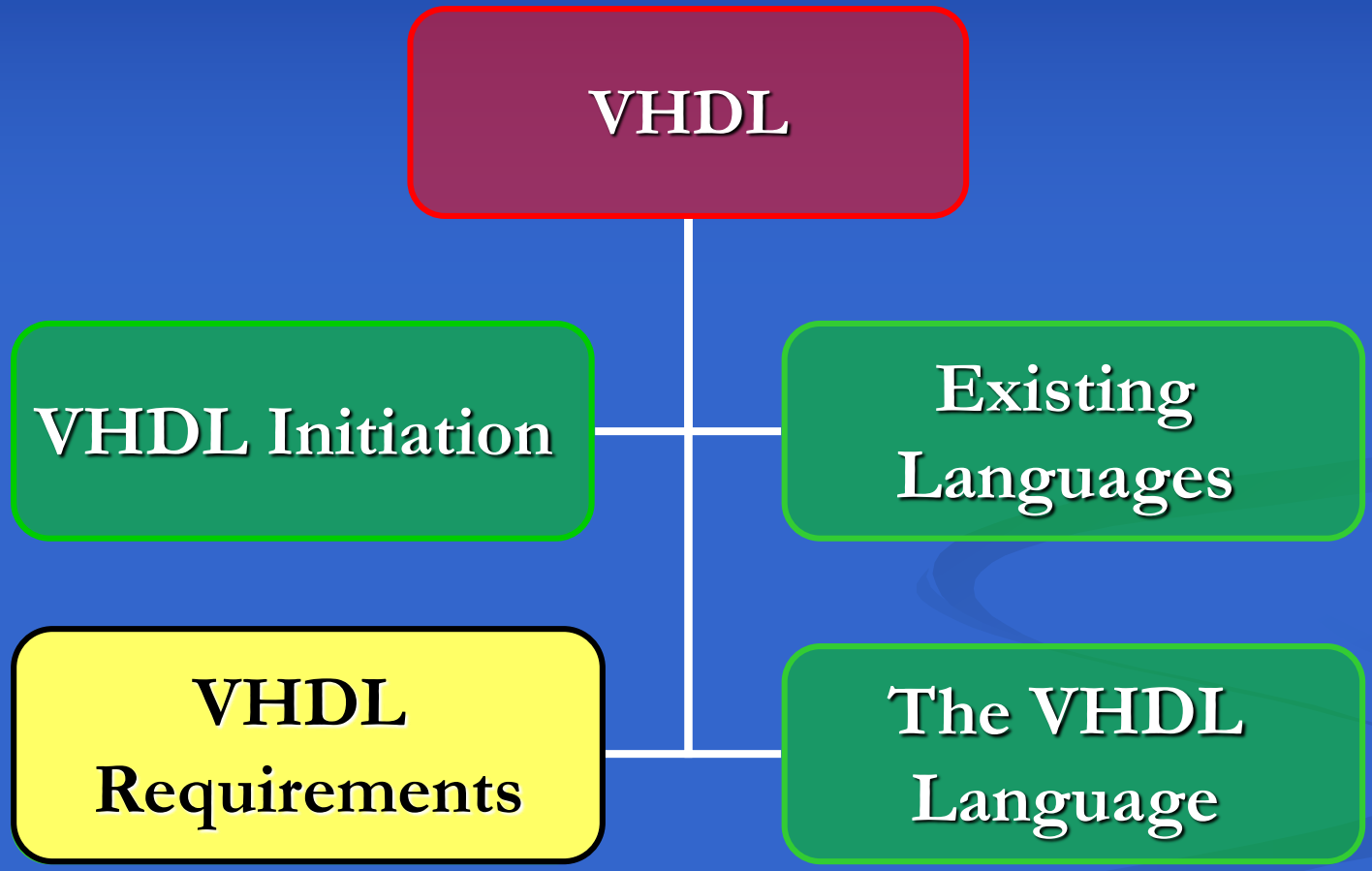
Existing Languages



Existing Languages

- AHPL
- CDL
- CONLAN
- IDL
- ISPS
- TEGAS
- TI-HDL
- ZEUS

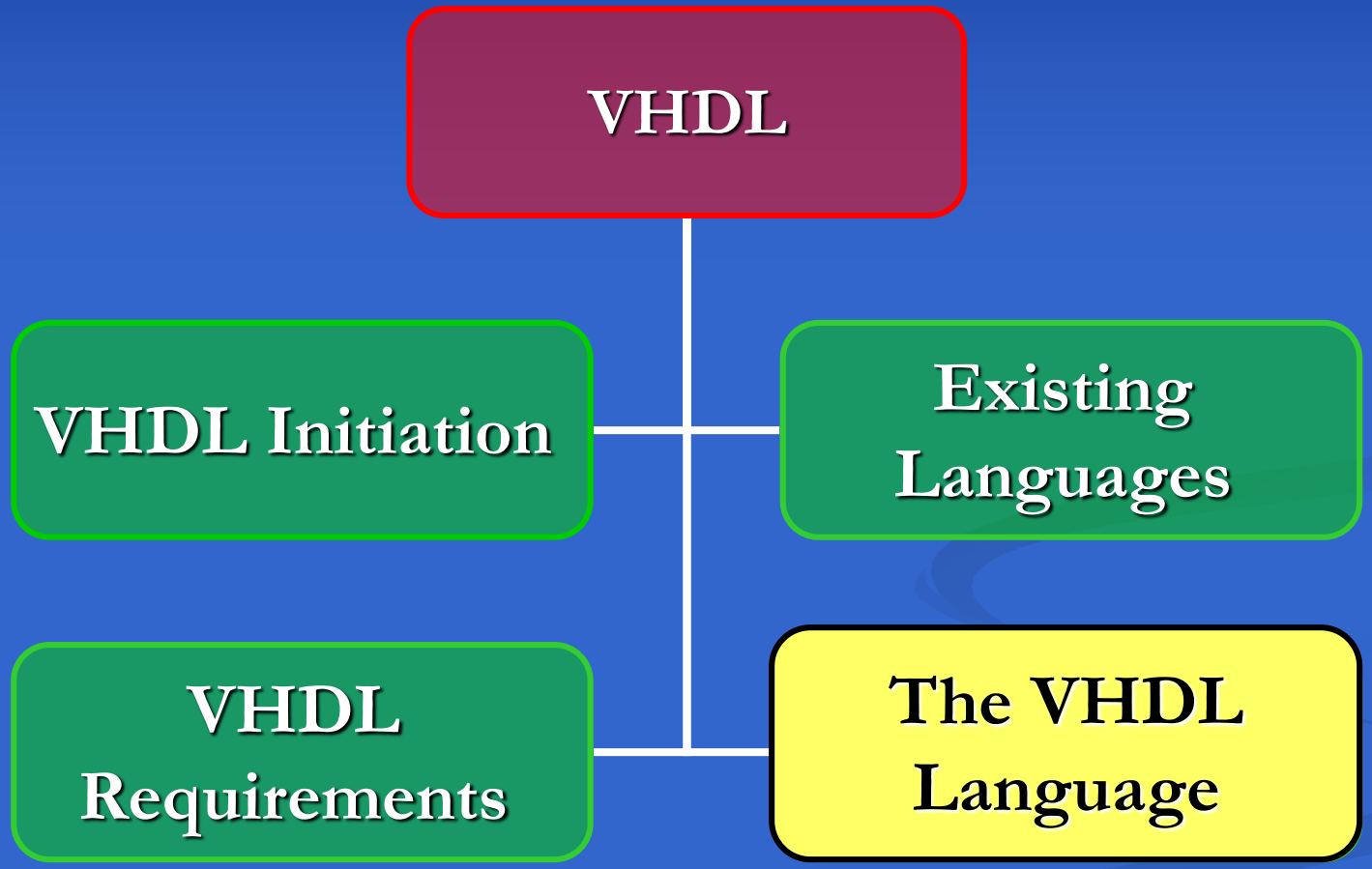
VHDL Requirements



VHDL Requirements

- Based on DoD requirements document:
  - General Features
  - Support for Design Hierarchy
  - Library Support
  - Sequential Statement
  - Generic Design
  - Type Declaration and Usage
  - Use of Subprograms
  - Timing Control
  - Structural Specification

The VHDL Language



The VHDL Language

- A hardware description language with strong emphasis on concurrency
- Supports hierarchical description of hardware from system to gate or even switch level
- Strong support at all levels for timing specification and violation detection
- Provides constructs for generic design specification and configuration

- A VHDL design entity is defined as:
  - An entity declaration
  - Its associated architecture body

- Groups subprograms or design entities by use of packages.
- Configurations for customizing generic descriptions of design entities
- Supports libraries and contains constructs for accessing packages, design entities, or configurations from various libraries.

The VHDL Language

- Vendor specific VHDL libraries used for time specification of various FPGA and ASIC libraries.
- Other libraries have specific packages for a certain design style promoted by EDA manufacturers.
- The standard IEEE library: A library of packages for type definitions, and logical operations.

- A typical VHDL design environment :
  - An analyzer program
  - Simulator
  - Hardware synthesizer
  - Test vector generator
  - Physical design tool

Summary

- This chapter presented:
  - An overview of mechanisms, tools, and processes used for taking a design from the design stage to a hardware implementation
  - The history of VHDL evolution
  - With this standard HDL, the efforts of tool developers, researchers, and software vendors have become more focused, resulting in better tools and more uniform environments.

- FPLD Design Flow