NATIONAL UNIVERSITY OF HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER ENGINEERING

LECTURE

***Subject:***

# VERILOG Hardware Description Language

# Chapter1: Introduction

Lecturer: Lam Duc Khai

# Agenda

1. Chapter 1: Introduction ( Week1)
2. Chapter 2: Fundamental concepts (Week1)
3. Chapter 3: Modules and hierarchical structure (Week2)
4. Chapter 4: Primitive Gates – Switches – User defined primitives (Week2)
5. Chapter 5: Structural model (Week3, Week4)
6. Chapter 6: Behavioral model – Combination circuit and Sequential circuit (Week4, 5)
7. Chapter 7: Tasks and Functions (Week6)
8. Chapter 8: State machines (Week6)
9. Chaper 9: Testbench and verification (Week7)
10. Project presentation ( Week7, Week8)

## Contents

Chapter 1: Introduction

- ➢ Requirements – Projects – Scores
- ➢ Verilog – What and Why ?
- ➢ CAD flow

## Requirement

- Must be punctual on both class and laboratory. An excess of 15 minutes is not accepted for any reason.
- Class : Start at 1:00 PM on Tuesday.
- Laboratory : Start at 8:00 AM and 1:00 PM as schedual.
- Project :
  - Submit strictly as the deadline through email: khaild@uit.edu.vn.
  - Name of project file and email subject must abide by the following format, any others format are not accepted:

    "Project name_maingroup_subgroup_date"

## Projects

- Projects :
  - Group1: A Floating Point Unit for Numerical Calculations.
  - Group2: Real-time Light-Saber Generator
  - Group3: Real-time Video Processing
  - Group4: FPGA Video Game
  - Group5: Real-time Face Detection
  - Group6: Gestural Interface for Image Browsing
  - Group7: Gesture Recognition Remote Control
  - Group8: Hardware Platform for JPEG Compression/Decompression

Reference : http://web.mit.edu/6.111/www/

## Scores

- Grading system:
  - – Final exam : 50 %
  - – Mid-exam : 20 %
  - – Laboratory : 20%
  - – Exercises: 10%
- Note: These above scores will only be valid if student attends the laboratory class fully.

# Verilog learning “tips”

- Verilog is essentially a programming language – similar to C with some Pascal-like constructs
- *The best way to learn any programming language is from live code*
- We will get you started by going through several example programs and explaining the key concepts
- *We will not try to teach you the syntax line-by-line*: pick up what you need from the books and on-line tutorials
- *Tip*: Start by copying existing programs and modifying them incrementally making sure you understand the output behavior at each step
- *Tip*: The best way to understand and remember a construct or keyword is to *experiment with it in code*, not by reading about it
- *We shall not design at the switch (transistor) level in this course* – the lowest level we shall reach is the gate level. The transistor level is more appropriate for an electronics-oriented course

## History

- HDL History
- 1970s: First HDLs
- Late 1970s: VHDL
- VHDL = VHSIC HDL = Very High Speed Integrated Circuit HDL
- VHDL inspired by programming languages of the day (Ada)
- 1980s:
- Verilog first introduced
- Verilog inspired by the C programming language
- VHDL standardized
- 1990s:
- Verilog standardized (Verilog-1995 standard)
- 2000s:
- Continued evolution (Verilog-2001 standard)
- Both VHDL and Verilog evolving, still in use today

## Why To Represent Hardware?

- If you’re going to design a computer, you need to write down the design so that:
- You can read it again later
- Someone else can read and understand it
- It can be simulated and verified
- Even software people may read it!
- It can be synthesized into specific gates
- It can be built and shipped and make money

## How To Represent Hardware?

- Draw schematics
- Hand-drawn
- Machine-drawn
- Write a netlist

    Z52BH I1234 (N123, N234, N4567);
- Write primitive Boolean equations

    AAA = abc DEF + ABC def
- Use a Hardware Description Language (HDL)

    assign overflow = c31 ^ c32;

## Custom design vs System design

- Custom design:
  - Small design . For instance : RAM, ROM, ALU, …
  - High performance
  - Designed by schematic or SPICE netlist
  - Very time consuming to design (timing, power,… verification by simulation)
- System design:
  - Large and complex design , system level ((millions to billions of gates). For instance : Chip, Micro processor, CPU, …
  - Lower performance
  - Designed by HDL.
  - Less design time ➔ more productivity.

# HDL-Advantages

- Describe complex designs (millions to billions of gates)
- Input to synthesis tools (generated circuits)
- Design exploration with simulation with less time consuming.
- Support for structure and instantiation
- Support for describing bit-level behavior
- Support for timing
- Support for concurrency

## HDL-Disadvantages

- Much depends on Synthesis tools.
- Hard to optimize design.

## Verilog vs. VHDL

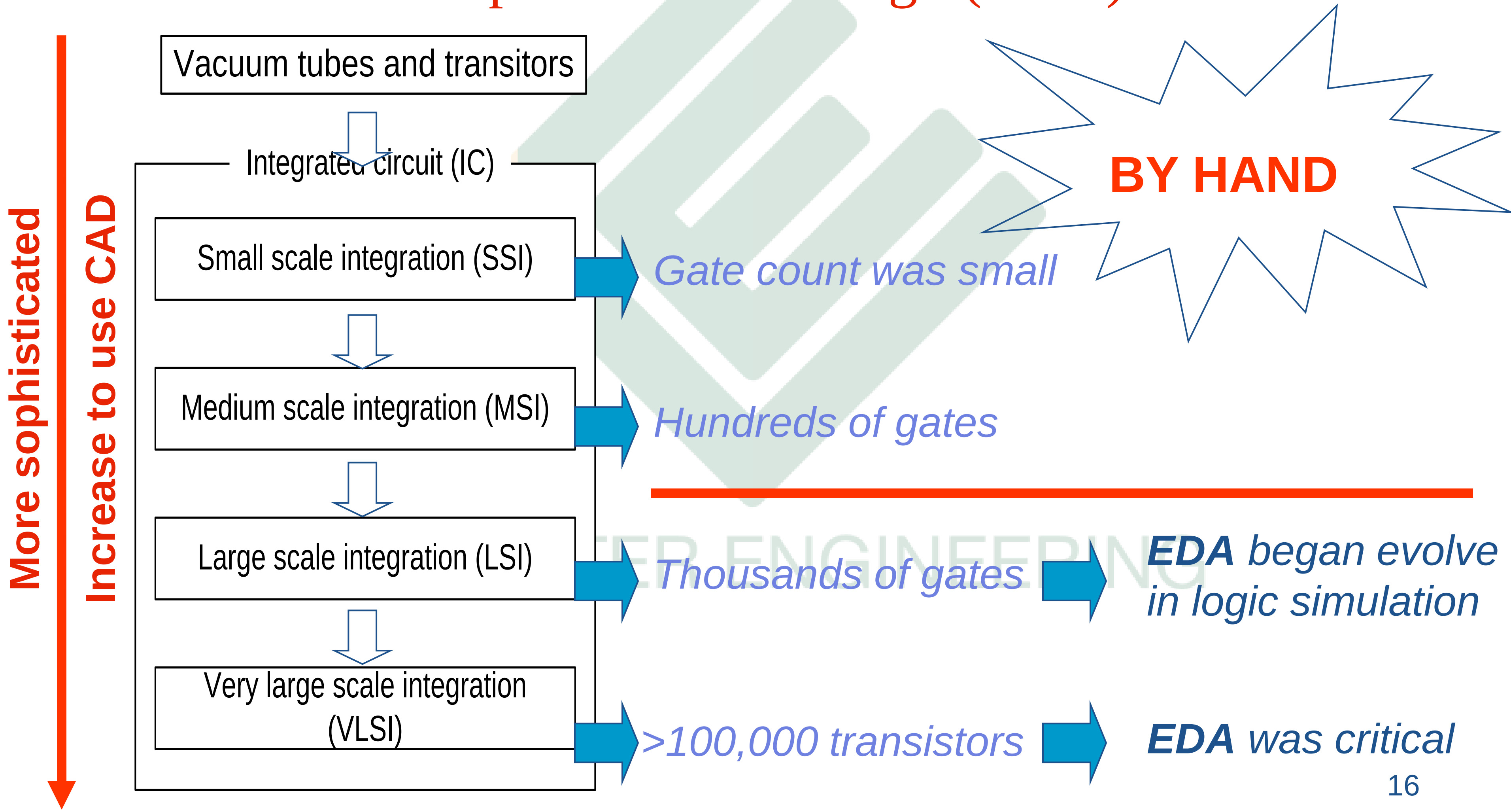
- Verilog is relatively simple and close to C
- VHDL is complex
- For commercial products, it’s Verilog, Verilog has 60% of the world digital design market (larger share in US)
- For large projects such as defense and telecommunication projects from government / aerospace work, it’s VHDL

## "Keep in heart"

- HDLs are not "programming languages"
- When coding in an HDL, it is important to remember that you are specifying hardware that executes in parallel rather than software that executes sequentially.
- In a program, we start at the beginning (e.g. "main"), and we proceed sequentially through the code as directed. The program represents an algorithm, a step-by-step sequence of actions to solve some problem
- Hardware is all active at once; there is no starting point. Everything happens concurrently.
- "Sequential design in HDL differs from sequential executive commands in programming languages".

# CAD flow

❖ Evolution of Computer Aided Design (CAD)


More sophisticated
Increase to use CAD
Vacuum tubes and transitors
Integrated circuit (IC)
Small scale integration (SSI)
Medium scale integration (MSI)
Large scale integration (LSI)
Very large scale integration (VLSI)
Gate count was small
Hundreds of gates
BY HAND
Thousands of gates
EDA began evolve in logic simulation
>100,000 transistors
EDA was critical

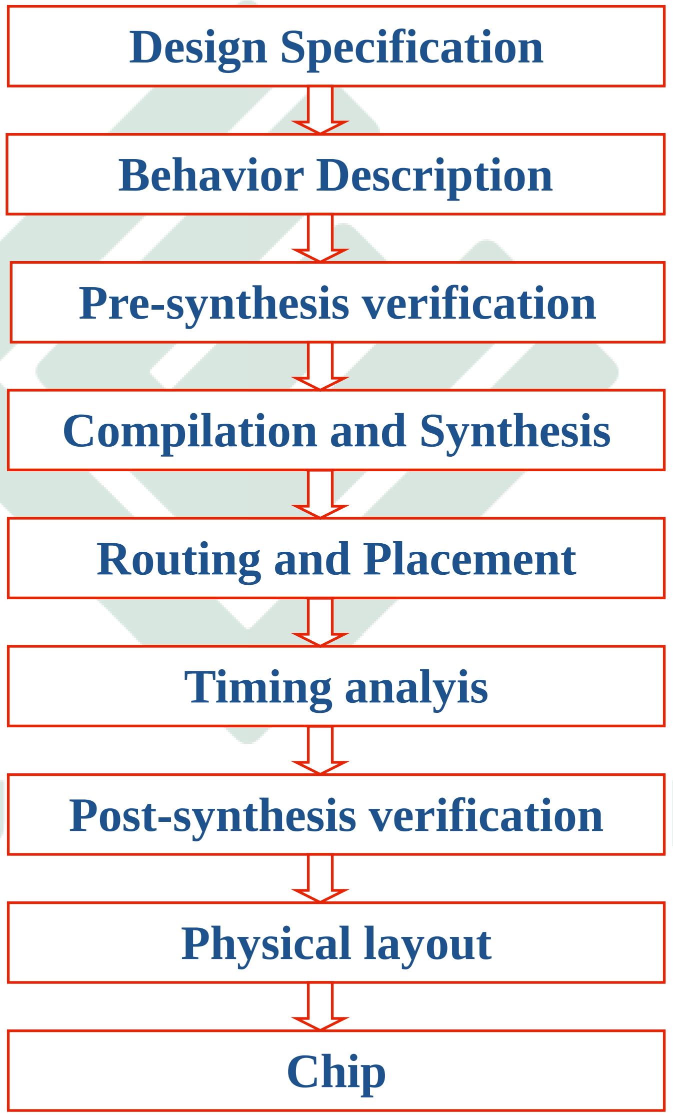
# CAD flow (cont'd)

❖ Computer Aided Design vs Manual Design

| | **Verilog design** | **Schematic design** |
|---|---|---|
| **Pros** | - Easy to implement function<br>- Simulate to verify function quickly due to simple netlist<br>- Generate schematic and layout automatically by using CAD tools.<br>- Helpful for system synthesis | - Customer design<br>- Be able to optimize performance, power and area of design. |
| **Cons** | - No optimality for performance, power and area of design. | - Must be master on IC design<br>- Take time to simulate to verify function due to big and complicated netlist.<br>- Hard to design for system level |

# CAD flow (cont’d)

- CAD Design Flow:

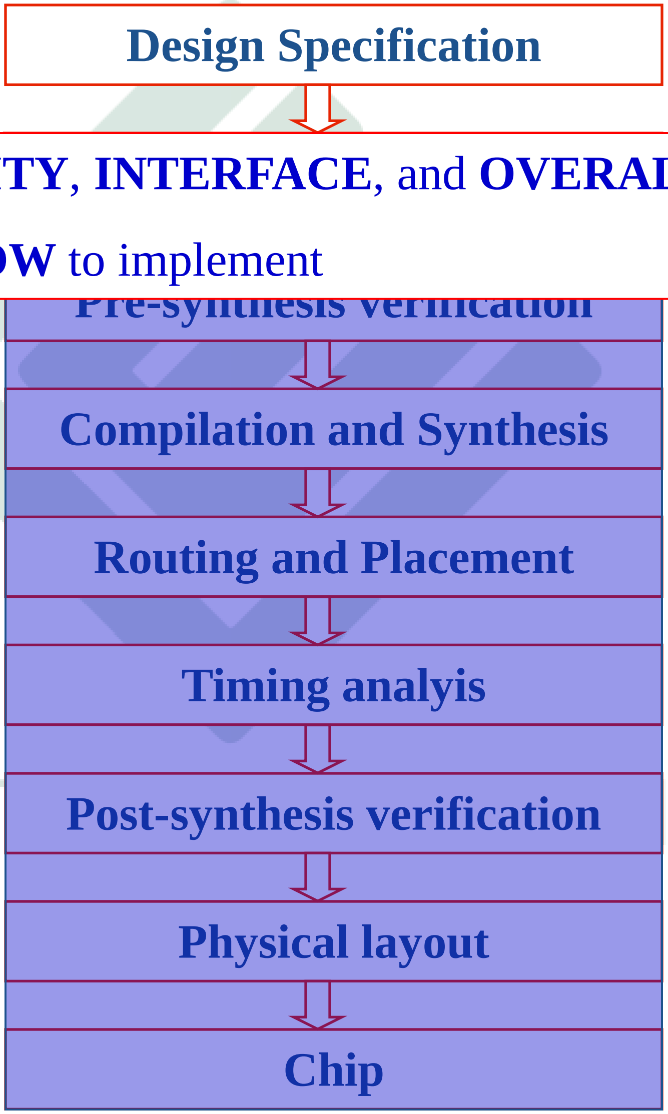


# CAD flow (cont'd)

❖ Design specification:

+ Describe the **FUNCTIONALITY**, **INTERFACE**, and **OVERALL ARCHITECTURE**

+ Do not need to think about **HOW** to implement



Design Specification
Pre-synthesis verification
Compilation and Synthesis
Routing and Placement
Timing analyis
Post-synthesis verification
Physical layout
Chip

# CAD flow (cont'd)

❖ Behavior Description:

**Design Specification**

⇓

**Behavior Description**

⇓

+ Analyze the design in terms of **FUNCTIONALITY**, **PERFORMANCE**, **COMPLIANCE** to standards, and **OTHER** high-level issues

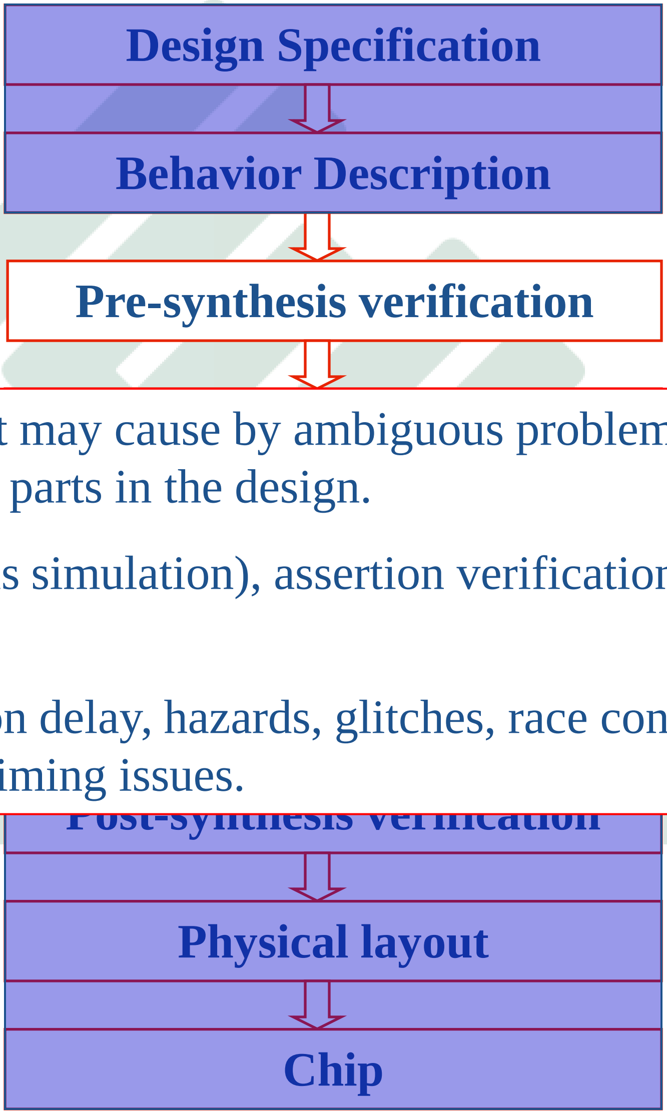
+ Often written with **HDLs** or **EDA** tools (combine HDLs and object oriented languages such as C++)

| **Behavior description for design specification in Verilog** | | |
|---|---|---|
| module design(…);<br>assign …<br>always …<br>compi (…)<br>endmodule | comp 1 U1(…);<br>comp2 U2(…);<br>…<br>compn Un(…); | always (posedge clk)<br>begin …end<br>if (…) bus = w;<br>else … |

**Chip**

## CAD flow (cont'd)

❖ Pre-synthesis verification:



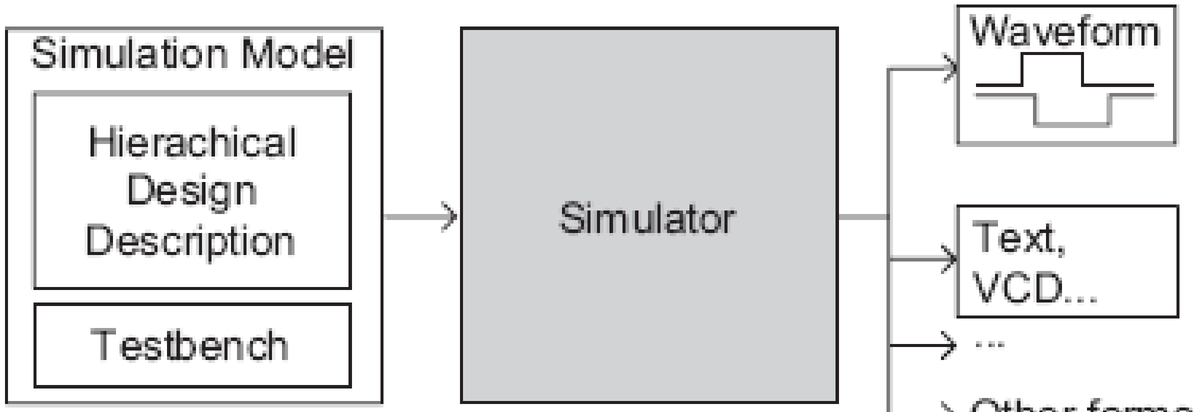
+ Check design function flaws that may cause by ambiguous problem specification, designer errors, or incorrect use of parts in the design.

+ Done by simulation (presynthesis simulation), assertion verification with **testbench** definition or **input waveform.**

+ Do not consider gate, propagation delay, hazards, glitches, race conditions, setup and hold violations, and other related timing issues.

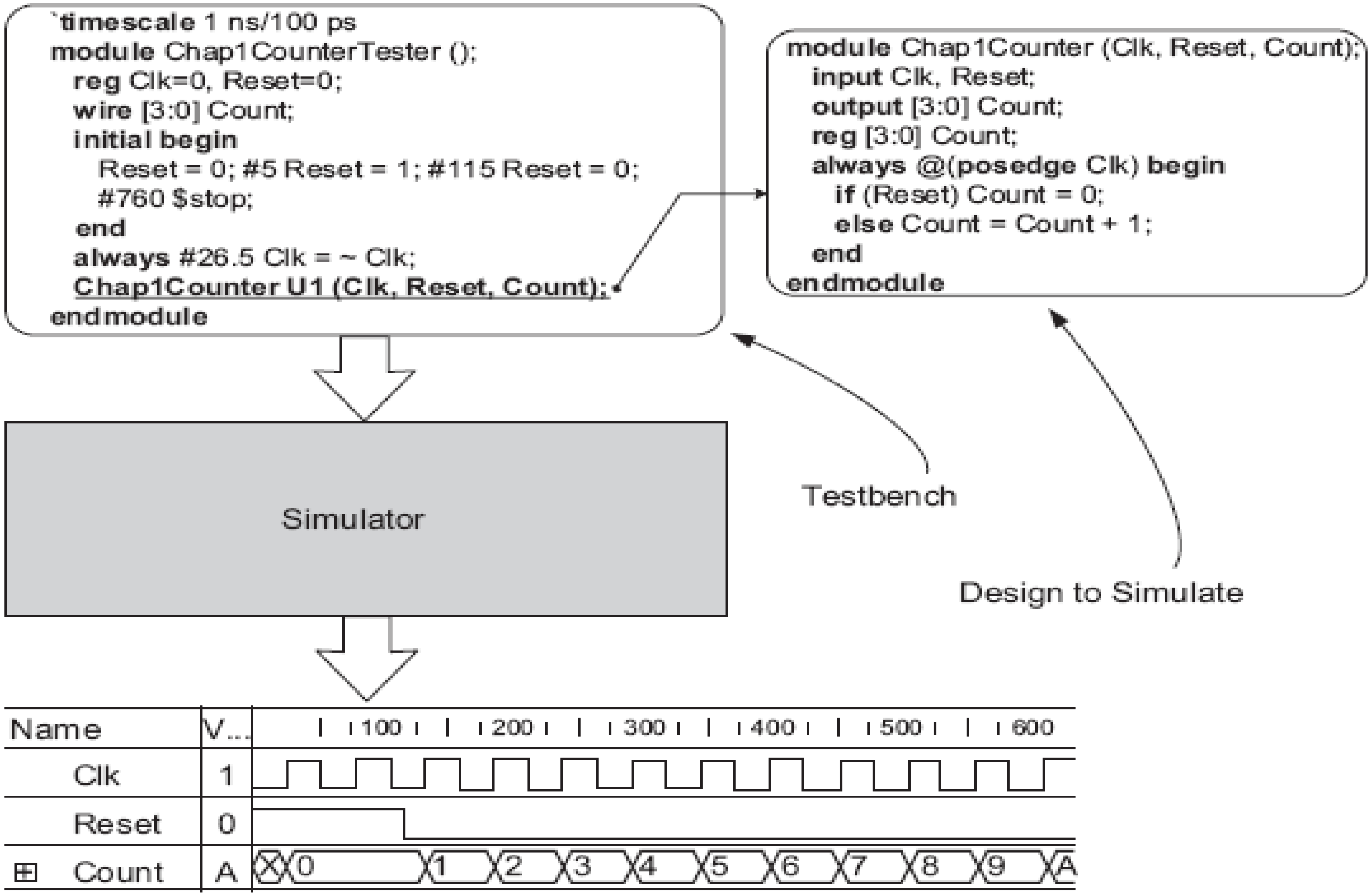
# CAD flow (cont'd)

❖ Function verification with testbench:



# CAD flow (cont’d)

❖ Function verification with testbench:

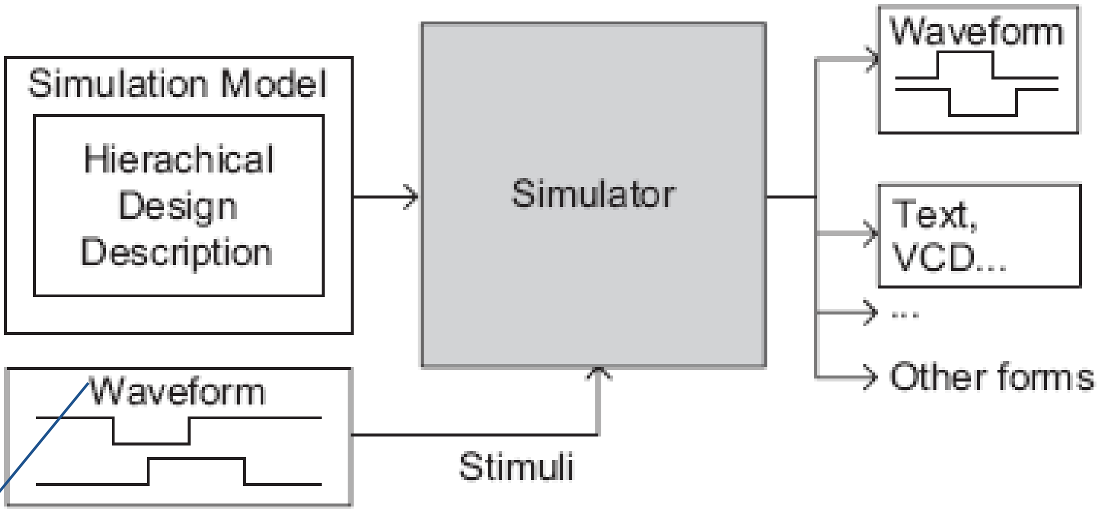


```
`timescale 1 ns/100 ps
module Chap1CounterTester ();
  reg Clk=0, Reset=0;
  wire [3:0] Count;
  initial begin
    Reset = 0; #5 Reset = 1; #115 Reset = 0;
    #760 $stop;
  end
  always #26.5 Clk = ~ Clk;
  Chap1Counter U1 (Clk, Reset, Count);
endmodule
```

```
module Chap1Counter (Clk, Reset, Count);
  input Clk, Reset;
  output [3:0] Count;
  reg [3:0] Count;
  always @(posedge Clk) begin
    if (Reset) Count = 0;
    else Count = Count + 1;
  end
endmodule
```

# CAD flow (cont'd)

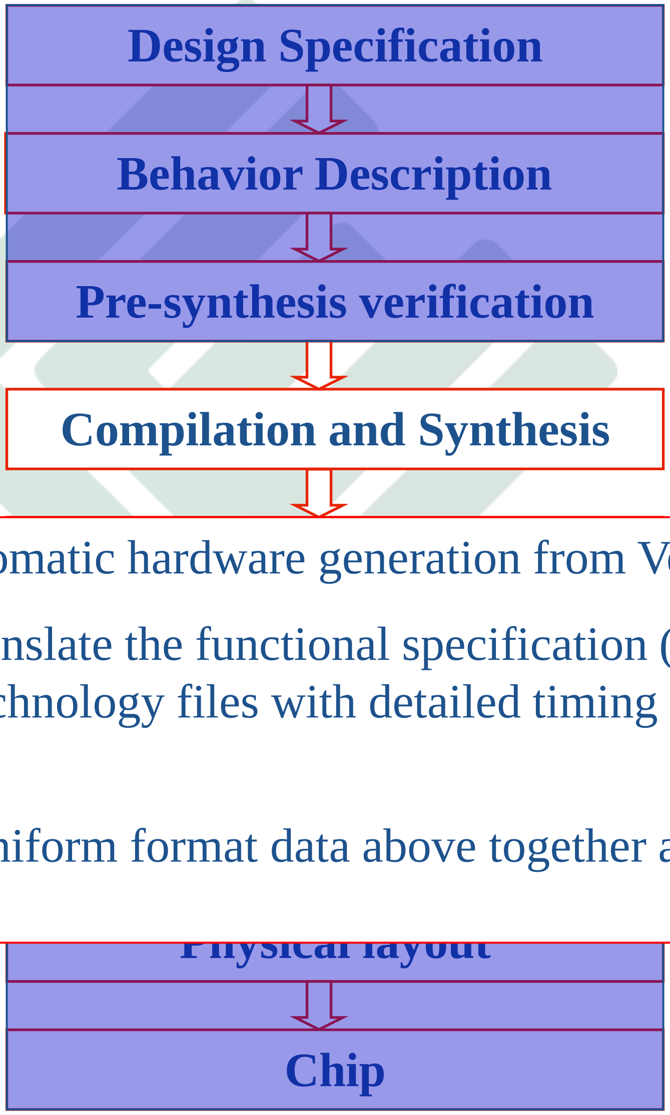
❖ Function verification with input waveform:



Define input waveform manually

# CAD flow (cont'd)

❖ Compilation and Synthesis:



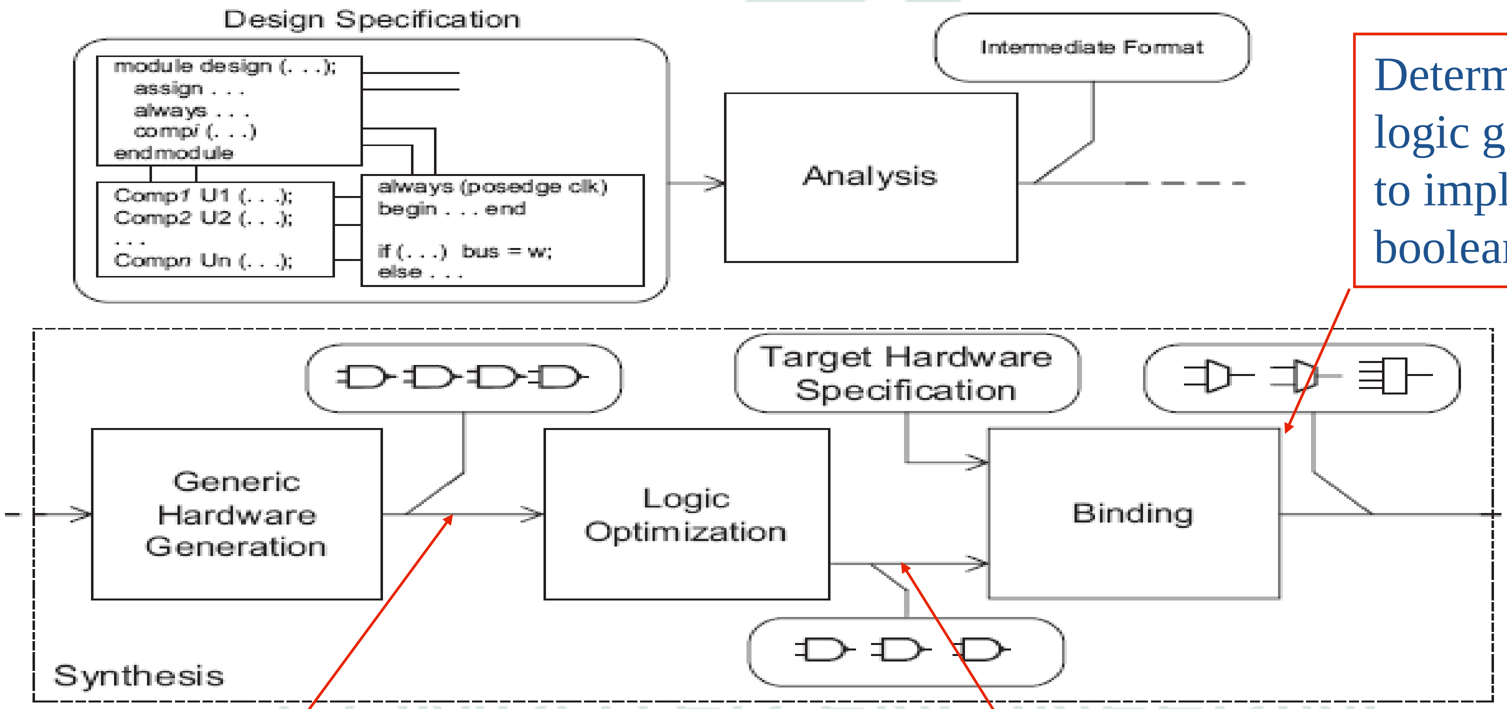
-Compilation and systhesis : automatic hardware generation from Verilog description.

-Analysis phase: Analyze and translate the functional specification ( Verilog description) the specification of hardware, technology files with detailed timing into the uniform format data.

-Synthesis phase: Combine all uniform format data above together and generate the corresponding logic.

## CAD flow (cont'd)

❖ Compilation and Synthesis



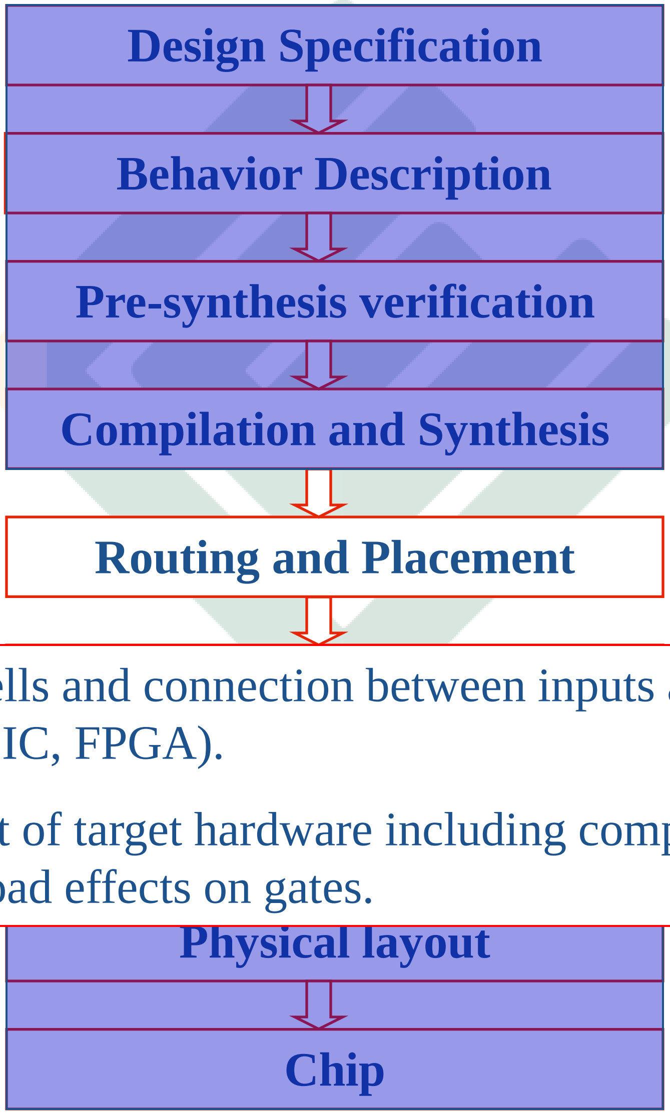
Determine which logic gates, elements to implement the boolean expressions

A set of boolean expressions or a netlist of basic gates.

A set of optimized boolean expressions, tabular logic representations or primitive gate netlists.

## CAD flow (cont'd)

❖ Routing and Placement:

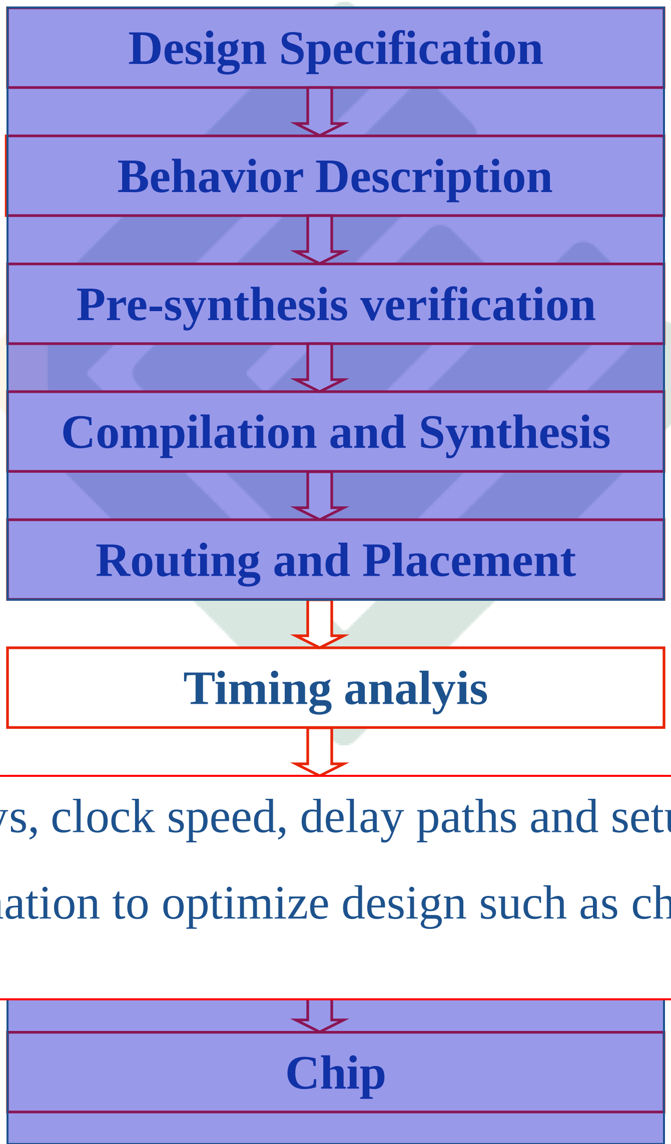


-Decide the placement of cells and connection between inputs and outputs of these cells for the target hardware ( ASIC, FPGA).

-Output is a complete netlist of target hardware including components, wiring delays of each interconnection, and load effects on gates.

# CAD flow (cont'd)

❖ Timing analysis:

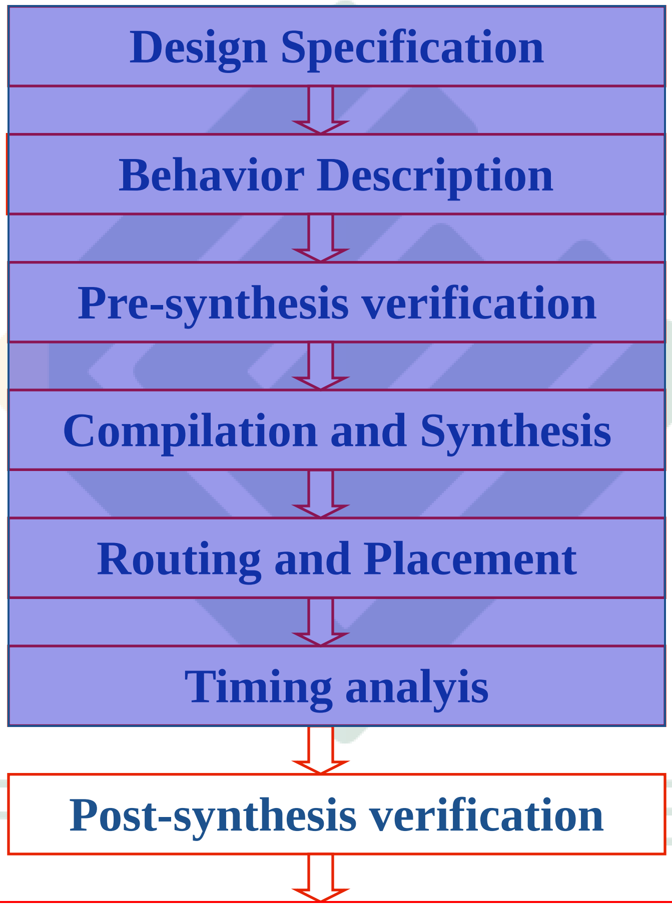


-Generates worst-case delays, clock speed, delay paths and setup times, hold times.

-Designers use these information to optimize design such as changing routing and placement.

## CAD flow (cont'd)

❖ Post-synthesis verification:

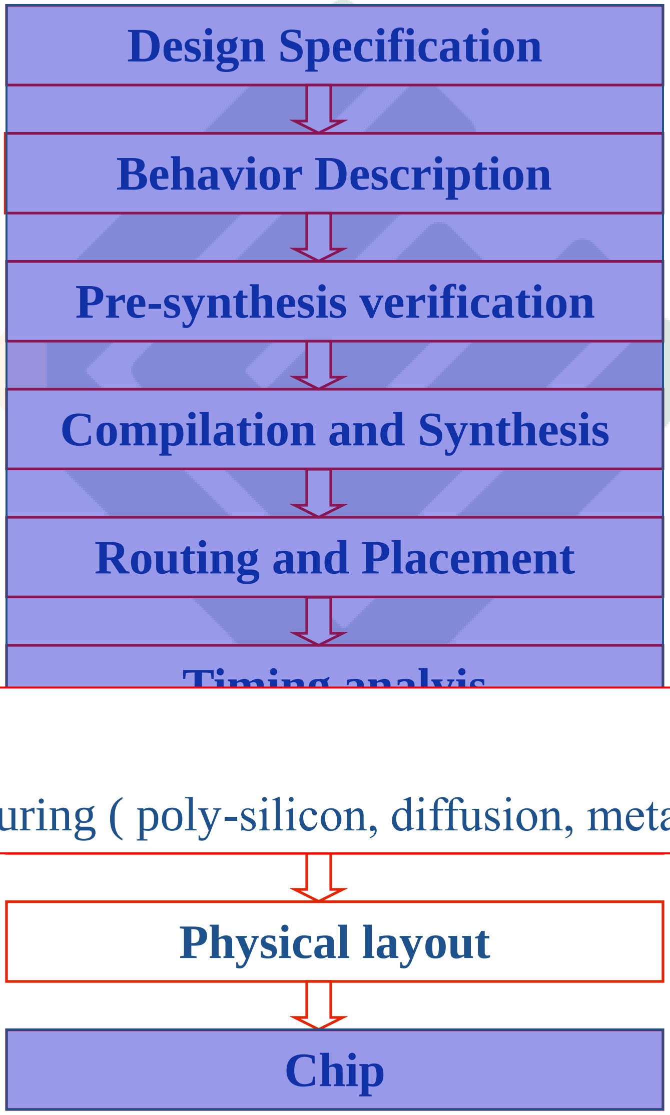

Design Specification
Behavior Description
Pre-synthesis verification
Compilation and Synthesis
Routing and Placement
Timing analyis
Post-synthesis verification

-Using netlist from routing and placement phase as the input for post-synthesis simulation.

-Verify clock frequency, race condition, hazzard condition, function and delay timings such as setup times, hold times, access times, output hold times ...

## CAD flow (cont'd)

❖ Physical layout:

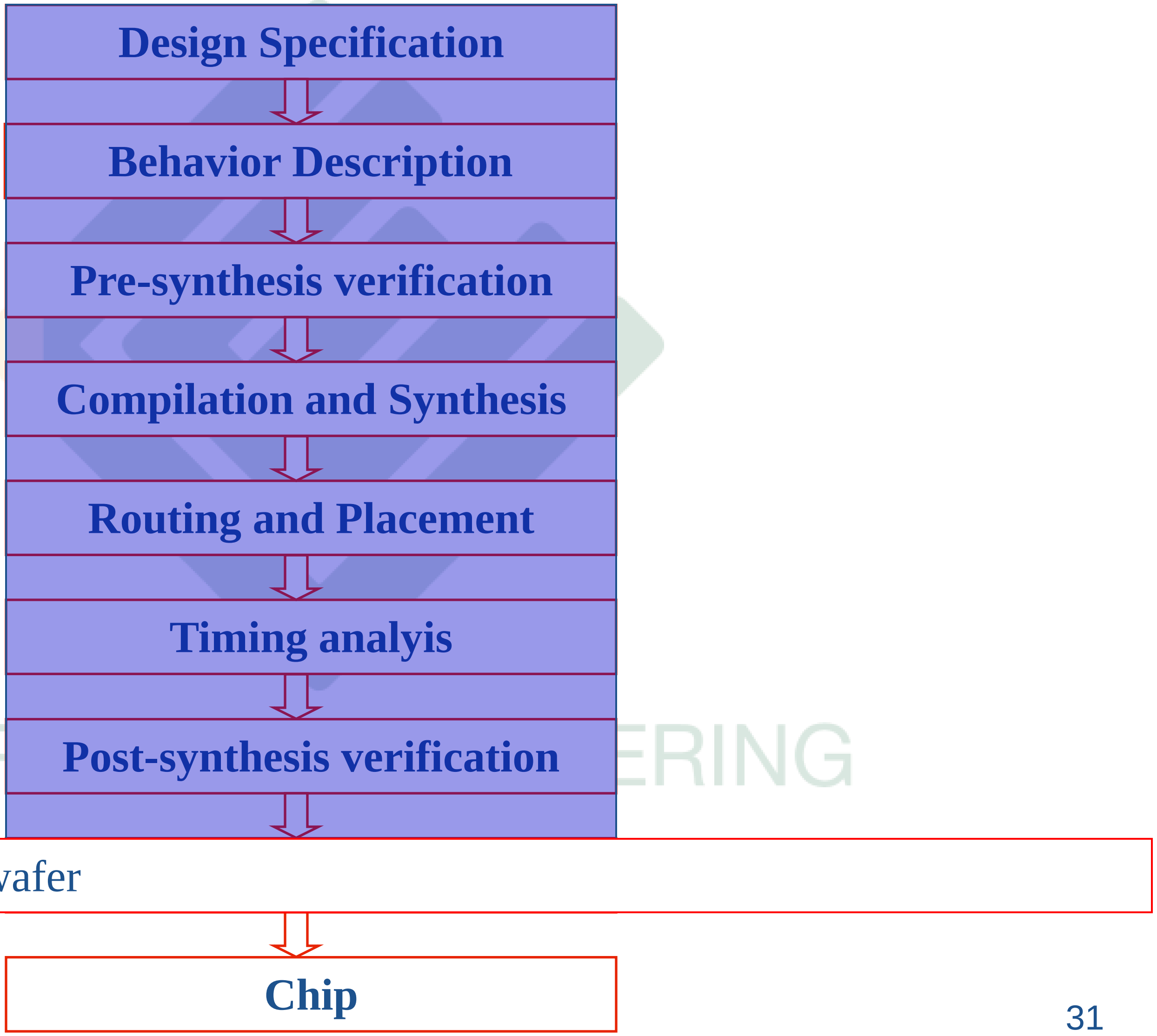


-Programming for FPGA

-Layout for ASIC manufacturing ( poly-silicon, diffusion, metal connection…)

# CAD flow (cont'd)

❖ Fabrication for Chip:


Design Specification
Behavior Description
Pre-synthesis verification
Compilation and Synthesis
Routing and Placement
Timing analyis
Post-synthesis verification
-Fabrication for design on wafer
Chip

## CAD flow (cont'd)

❖ Summary:

– HDLs are now the **dominant method** for large digital designs
– Syntax is similar to C language → easy to learn and easy to use
– Allows different levels of abstraction (switches, gates, RTL, or behavioral code) to be mixed in the same level
– Most popular logic synthesis tools support Verilog
– Allows the user to write custom C code to interact with internal data structures of Verilog by using PLI (Programming Language Interface)

# END