DIGITAL CIRCUIT DESIGN WITH HDL

Course Introduction

- **Objectives**:

- Provide knowledge of digital circuit design flow and designing digital circuits with Hardware Description Language (HDL)
- Provide necessary skills to design, test and verify digital circuits with HDL

Course Introdution

- **References:**

[1] Micheal D. Ciletti, **Advanced Digital Design with the Verilog HDL**, 2nd Edition, Prentice Hall, 2010

[2] **IEEE Standard for Verilog Hardware Description Language**. IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001).

[3] D. R. Smith, P. D. Franzon, Verilog styles for synthesis, Prentice Hall 2000

[4] Zainalabedin Navabi. Verilog Digital System Design. McGraw-Hill, 2006, 402 pages.

[5] Samir Palnitkar. Verilog HDL 2nd Edition. Prentice Hall, 03/3/2003, 496 pages.

[6] Volnei A. Pedroni. Circuit Design with VHDL. The MIT Press, 2004, 375 pages

Course Introduction

- **Lecturer:**

  MEng. Ho Ngoc Diem

  (Email: diemhn@uit.edu.vn)

- **Office Hour:**

  10:00 – 10:30 every Wednesday

  Room E6-6

Course Introduction

- **Assessment:**
  - Homework/Project (20%)
  - Lab (30%)
  - End-term exam (50%)

Course outline

- Chapter 1: Introduction
- Chapter 2: Verilog Language Concepts
- Chapter 3: Structural modeling
- Chapter 4: Behavioral modeling
- Chapter 5: State machines
- Chapter 6: Tasks and Functions
- Chapter 7: Functional Simulation/Verification (Testbench)
- Chapter 8: Synthesis of Combinational and Sequential Logic
- Chapter 9: Post-synthesis design tasks
- Chapter 10: VHDL introduction

CIRCUIT DESIGN WITH HDL

Content

- HDL – Verilog & VHDL
- CAD
- Digital Design Flow

How to represent Hardware?

- **Classical design method**

  - Schematic

  - Hand-draw or machine-draw

- **Computer-based language method**

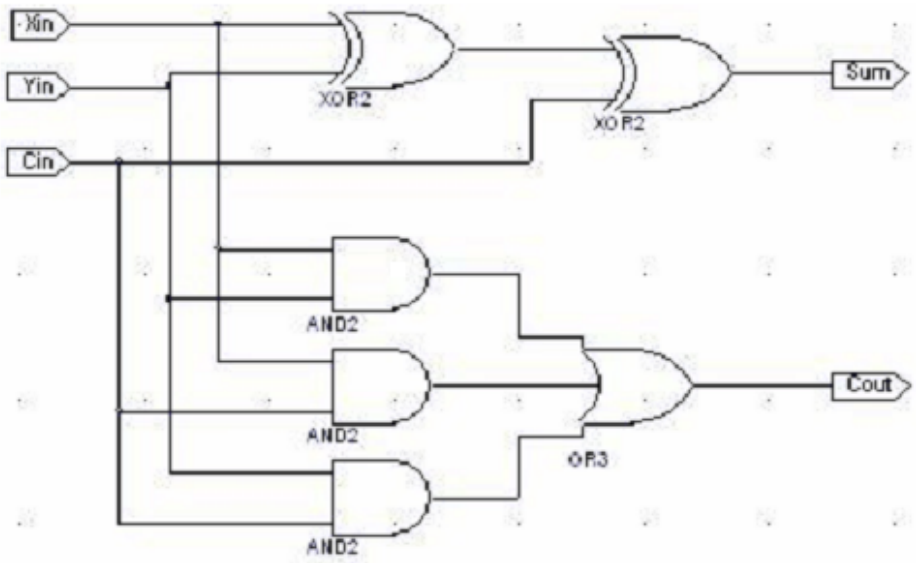
  - Hardware Description Language (HDL): **Verilog**, **VHDL**

  - Fast, popularly used to design complex circuit with large and very large scale

```
module Flop (reset, din, clk, qout);
    input reset, din, clk;
    output qout;
    reg qout;
    always @ (negedge clk) begin
        if (reset) qout <= #8 1'b0;
        else qout <= #8 din;
    end
endmodule
```

HDL design example

- Verilog HDL




```
module fulladder(a,b,cin,sum,cout);
  input a,b,cin;
  output sum,cout;

  reg sum,cout;
  always @ (a or b or cin)
  begin
    sum <= a ^ b ^ cin;
    cout <= (a & b) | (a & cin) | (b & cin);
  end
endmodule
```

Represent hardware by text!

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

HDL

- **Advantages:**

– Describe complex designs (millions to billions of gates)

– Different level of abstraction possible (switch, gate, behavioral)

– Input to synthesis tools (automatic generated circuits)

– Design exploration with simulation - less time consuming.

– Flexible, technology independent, portable across technologies

– Support for timing and concurrency

– Be supported by ASIC, FPGA synthesis tools

- **Disadvantages:**

– Much depends on Synthesis tools

– Hard to optimize design

History

- **Verilog**

  - Inspired by C programming language

  - Introduced in 1984 by Gateway (now Cadence). Firstly developed to allow simulation, afterward support for synthesis added

  - IEEE 1364-1995, IEEE 1364-2001, IEEE 1364-2005 standard

- **VHDL**

  - VHSIC HDL = Very High Speed Integrated Circuit HDL

  - Inspired by Ada programming language

  - First introduced in 1987, by DARPA

  - VHDL standardized by IEEE (‘87 and ’93)

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.

Verilog vs. VHDL

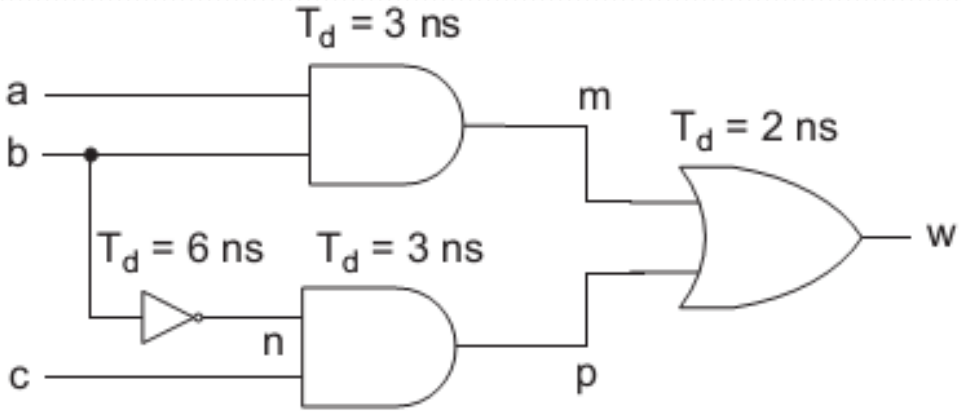
- Verilog is relatively simple and close to C
- VHDL is complex
- For commercial products, it's Verilog, Verilog has 60% of the world digital design market (larger share in US)
- For large projects such as defense and telecommunication projects from government / aerospace work, it's VHDL

Keep in heart

- HDLs are not “programming languages”
- Remember that you are specifying hardware that executes in parallel rather than software that executes sequentially.
- 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”.
- 2 main features of HDL: Timing and Concurrency

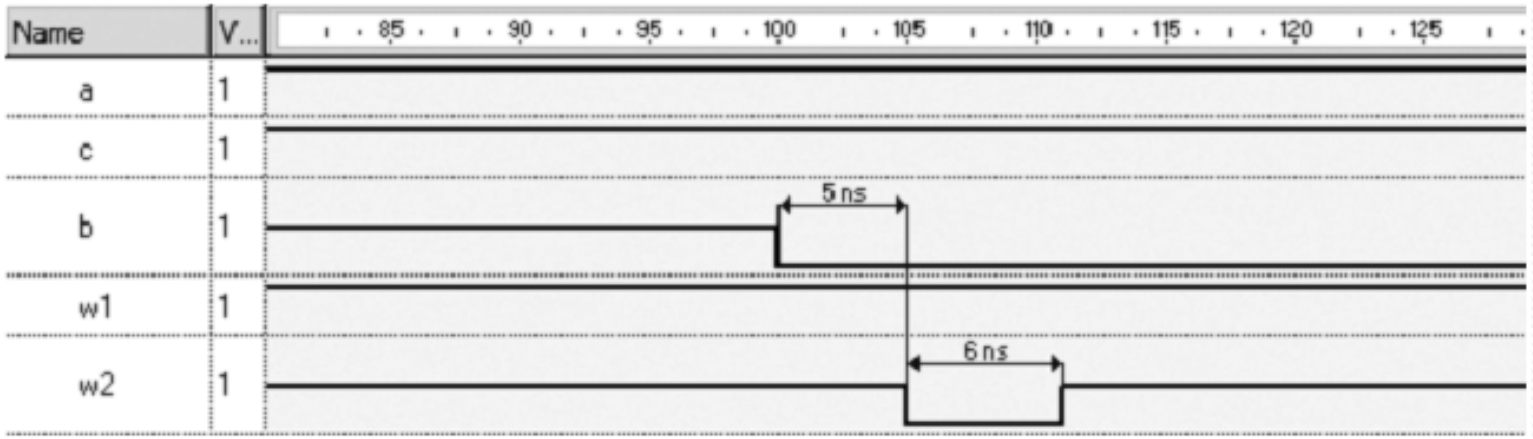
Timing & Concurrency

- Verilog code example



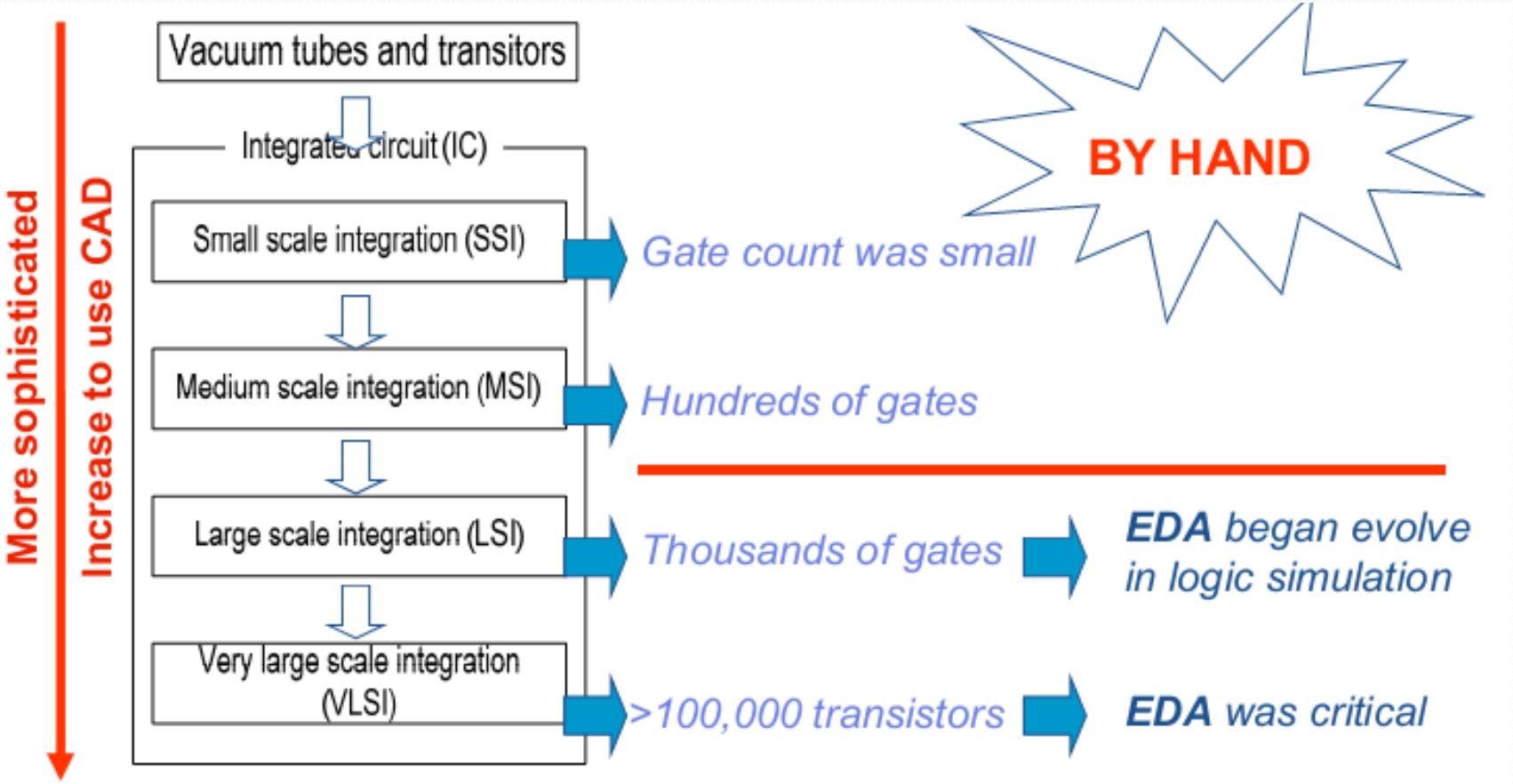

```
1) assign w1 = a & b | a & ~b;

2) assign #6 n = ~b;
     assign #3 m = a & b;
     assign #3 p = n & c;
     assign #2 w2 = m | p;
```



Computer Aided Design (CAD)

- **Evolution of Computer Aided Design** (CAD)



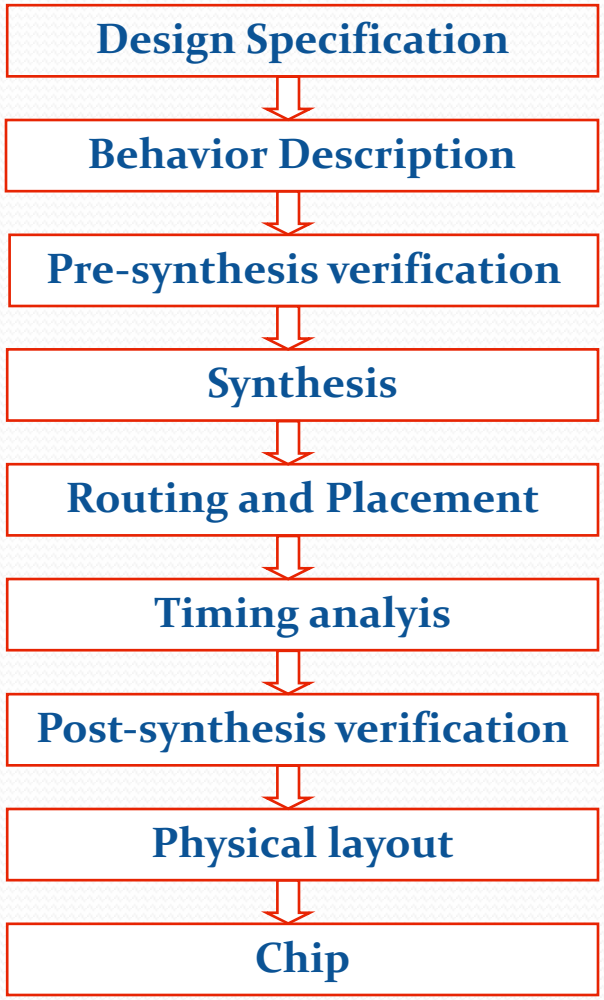
CAD

- **Computer Aided Design vs. Manual Design**

	CAD design	Schematic design
Pros	- Easy to implement function - Simulate to verify function quickly - Generate schematic and layout automatically by using CAD tools. - Helpful for system synthesis	- Customer design - Be able to optimize performance, power and area of design.
Cons	- Optimality for performance, power and area of design depends on synthesis tool.	- Must be master on IC design - Take time to simulate to verify function due to big and complicated netlist. - Hard to design for system level

Digital design flow

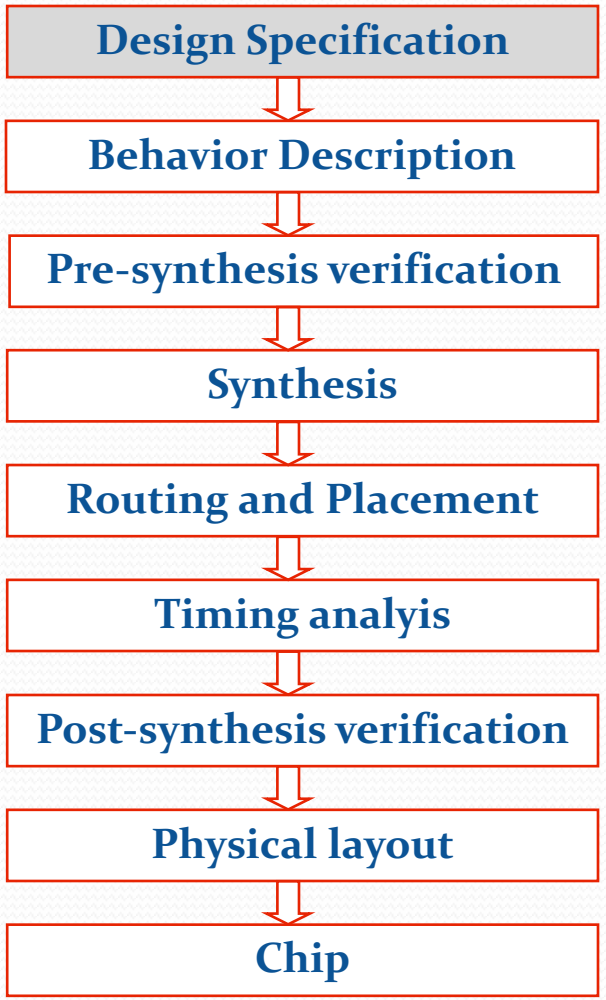
- Digital design flow



Digital design flow

+ Describe the **FUNCTIONALITY, INTERFACE**, and **OVERALL ARCHITECTURE**
+ Do not need to think about HOW to implement

- *State transition graph*
- *High-level language*

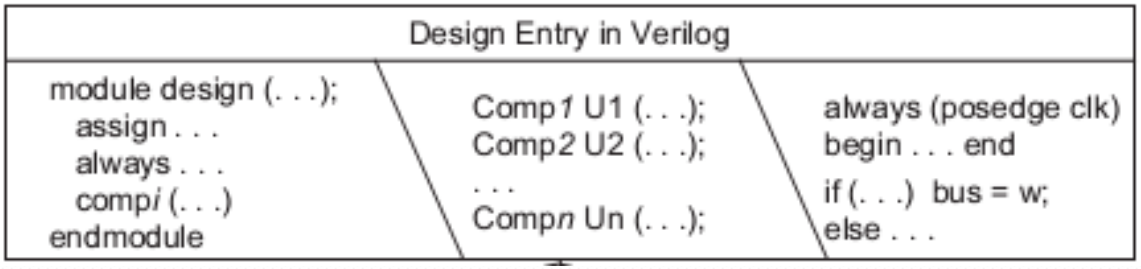


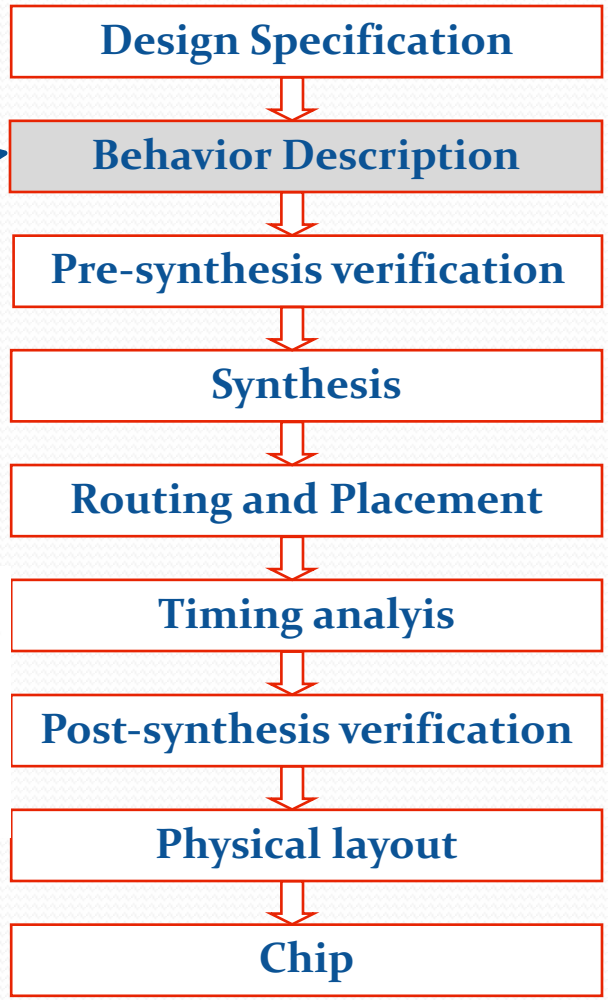
Digital design flow

+ Design is described in a top-down hierarchical fashion

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

+ Register Transfer Level (RTL) design





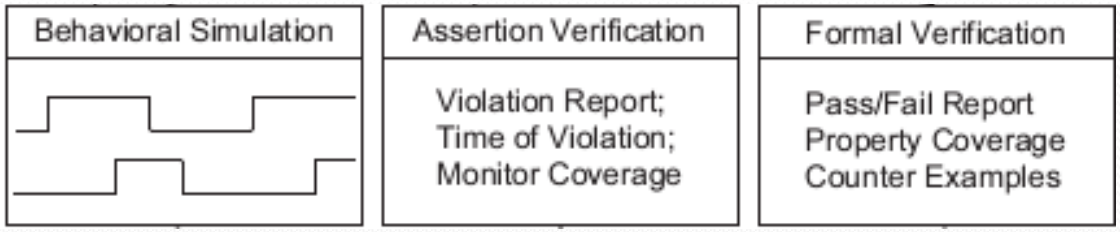
Digital design flow

Simulation & Function Verification

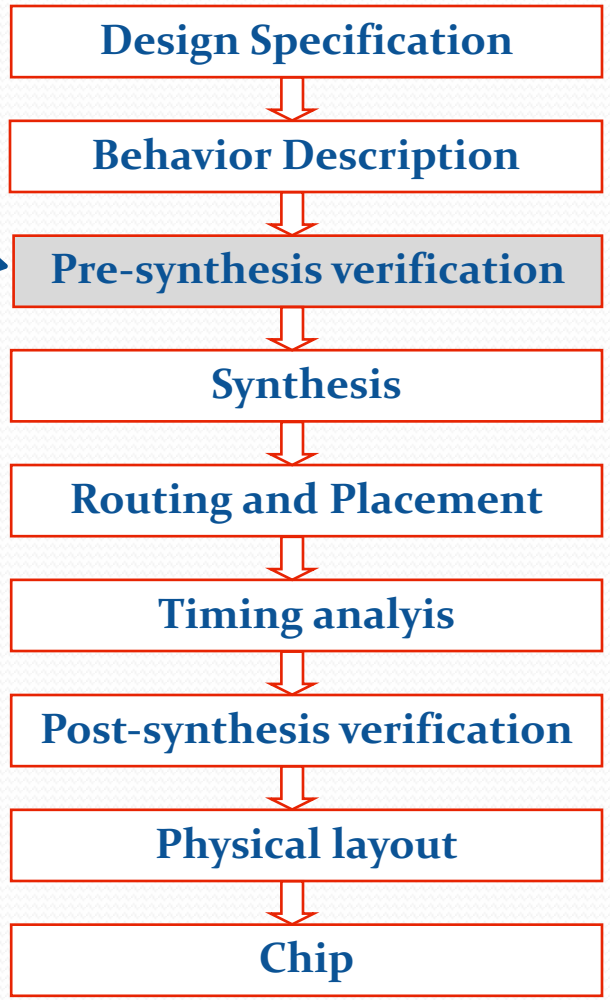
+ Design is simulated and tested for functionality before turning into hardware

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

+ Test data are generated by testbench or waveform editor

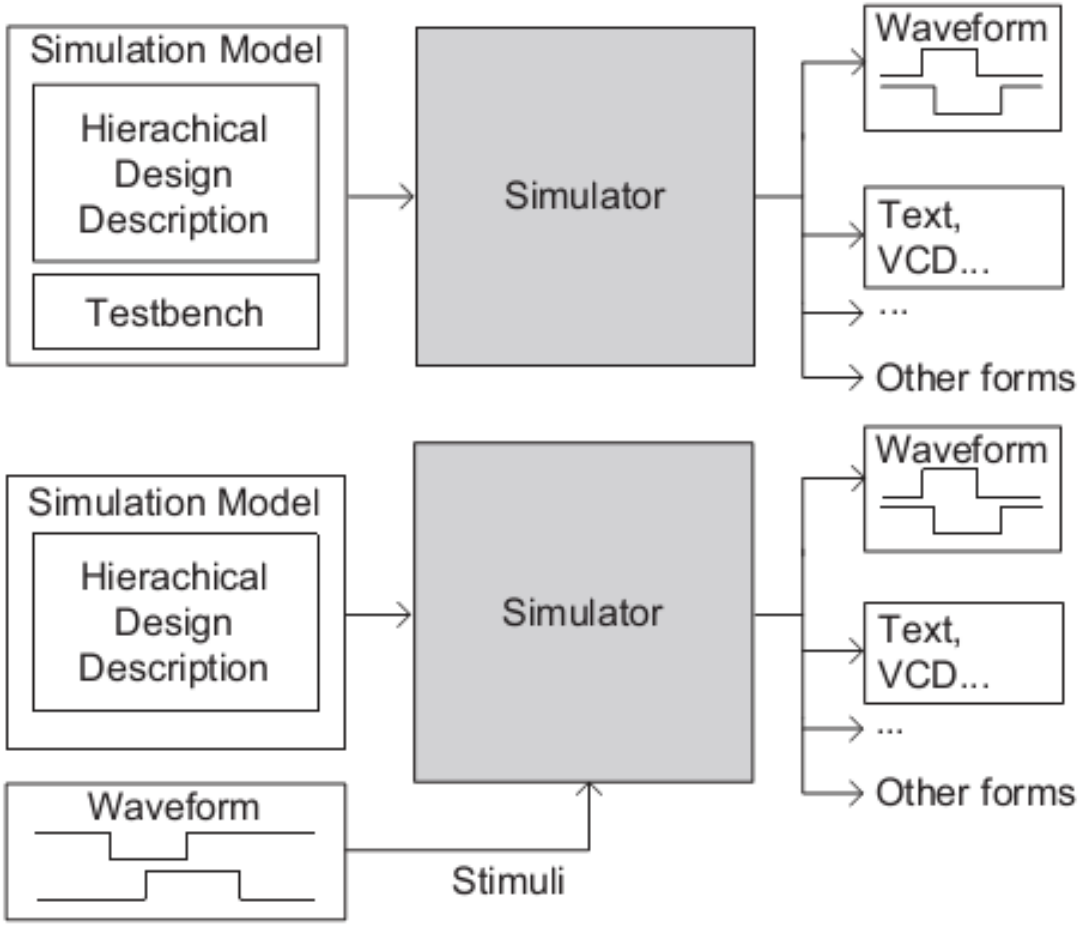


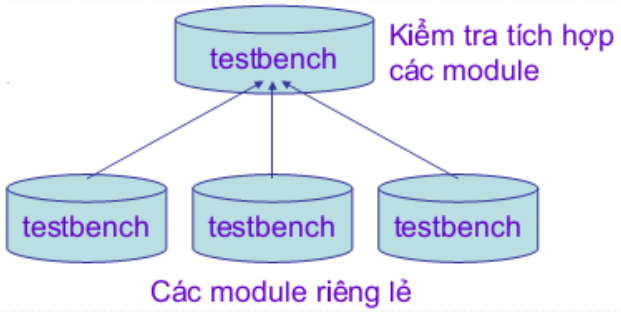
Ref. Verilog Digital System Design, Zainalabedin Navabi



Digital design flow

- Simulation by testbench or input waveform



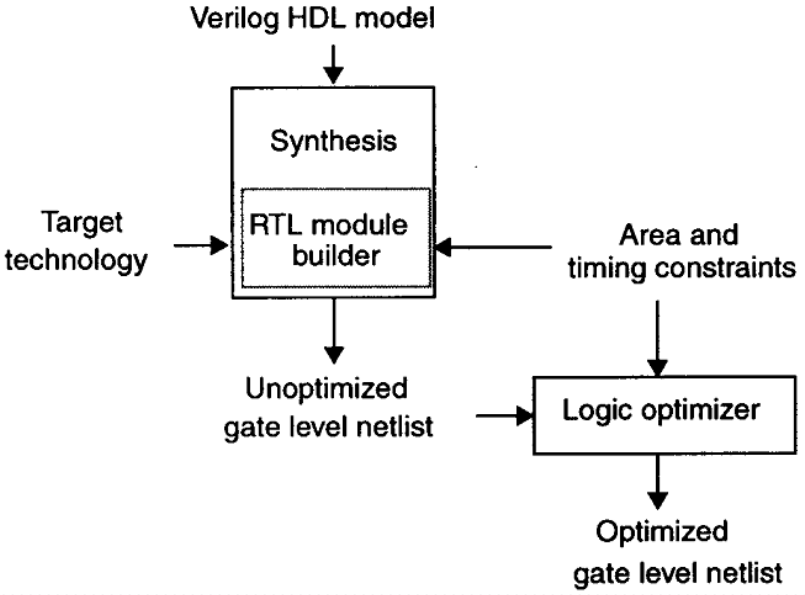


Digital design flow

Synthesis – Technology mapping

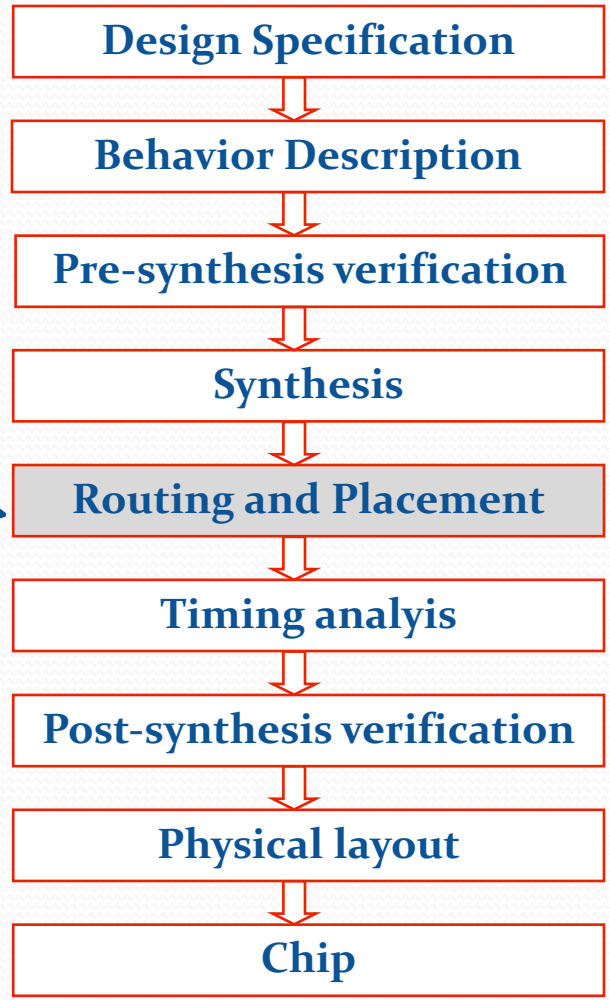
+ Logic synthesis tool converts Verilog description into gate-level netlist that is mapped into a specific technology (FPGA, ASIC, …)

+ Optimize the circuit in area and timing for the technology



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

Digital design flow



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

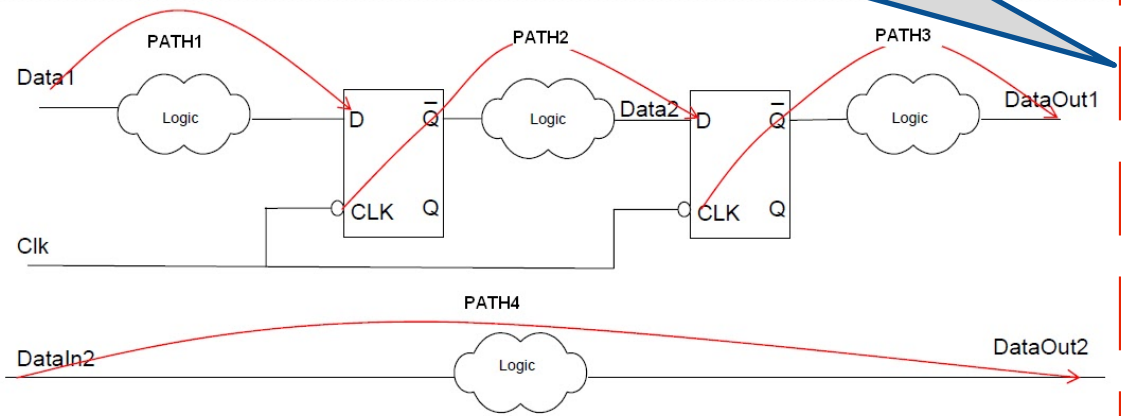
-Output is a complete netlist of target hardware including components, wiring delays of each interconnection, and load effects on gates.

Digital design flow

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

-Designers use these information to optimize design (changing routing and placement), or to decide clocking speed

- static timing analysis



Design Specification

Behavior Description

Pre-synthesis verification

Synthesis

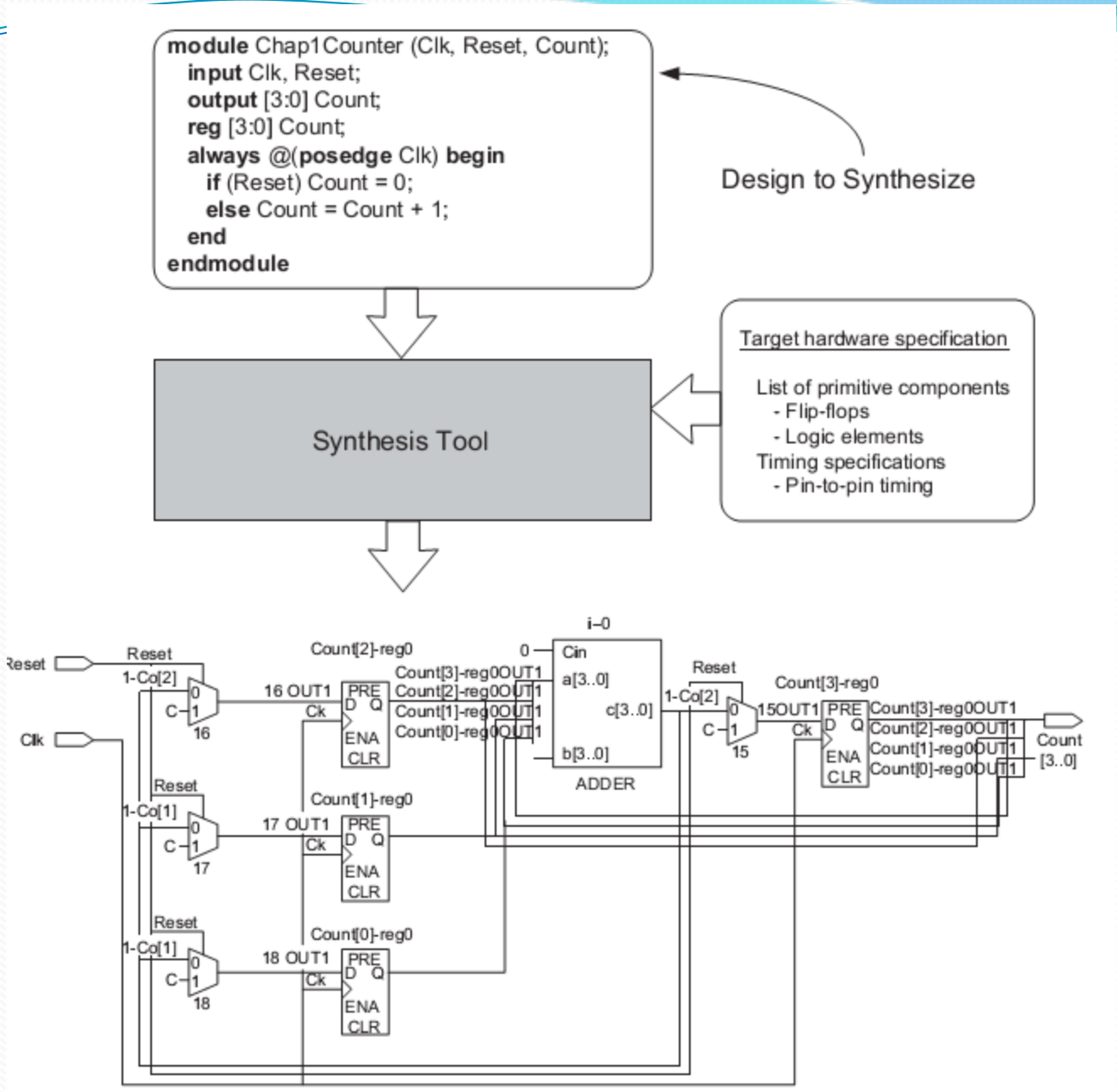
Routing and Placement

Timing analyis

Post-synthesis verification

Physical layout

Chip

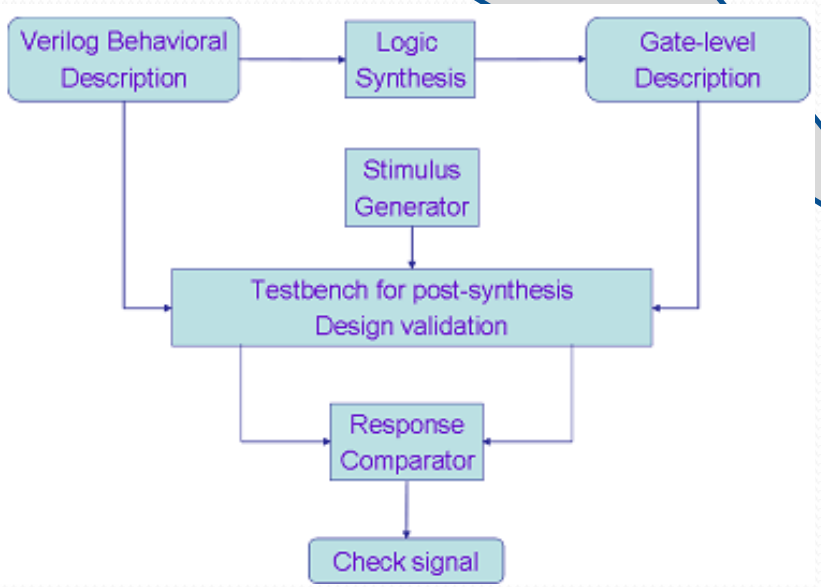

module Chap1Counter (Clk, Reset, Count);
input Clk, Reset;
output [3:0] Count;
reg [3:0] Count;
always @(posedge Clk) begin
if (Reset) Count = 0;
else Count = Count + 1;
end
endmodule
Design to Synthesize
Synthesis Tool
Target hardware specification
List of primitive components
- Flip-flops
- Logic elements
Timing specifications
- Pin-to-pin timing
Reset
Clk
1-Co[2]
1-Co[1]
16
17
18
16 OUT1
17 OUT1
18 OUT1
Count[2]-reg0
Count[1]-reg0
Count[0]-reg0
PRE
D Q
ENA
CLR
Ck
Count[3]-reg0OUT1
Count[2]-reg0OUT1
Count[1]-reg0OUT1
Count[0]-reg0OUT1
i~0
Cin
a[3..0]
b[3..0]
c[3..0]
ADDER
15
15OUT1
Count[3]-reg0
Count
[3..0]

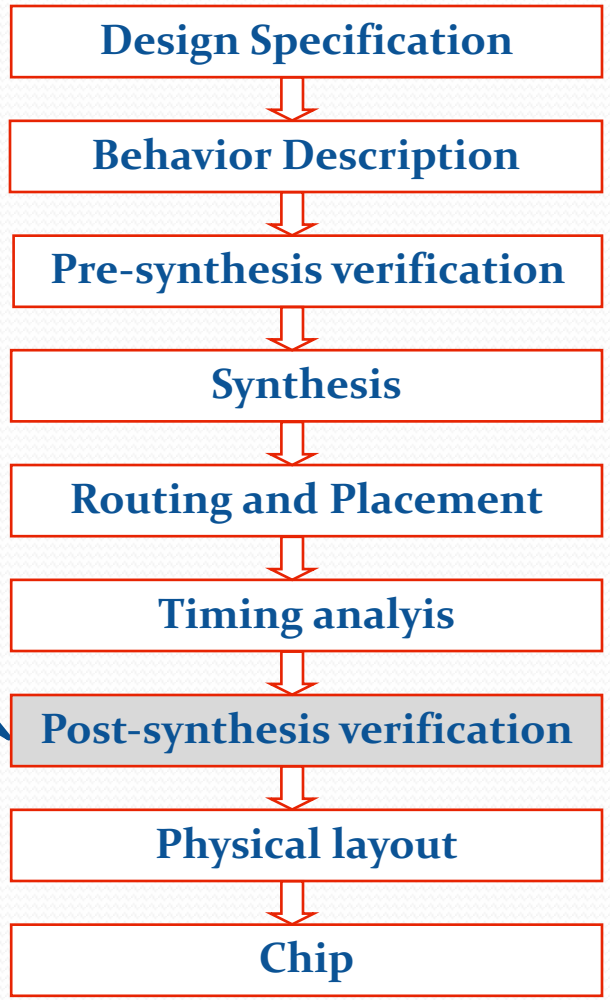
Digital design flow

- Using netlist from routing and placement phase as the input for post-synthesis simulation; the same testbench can be used.

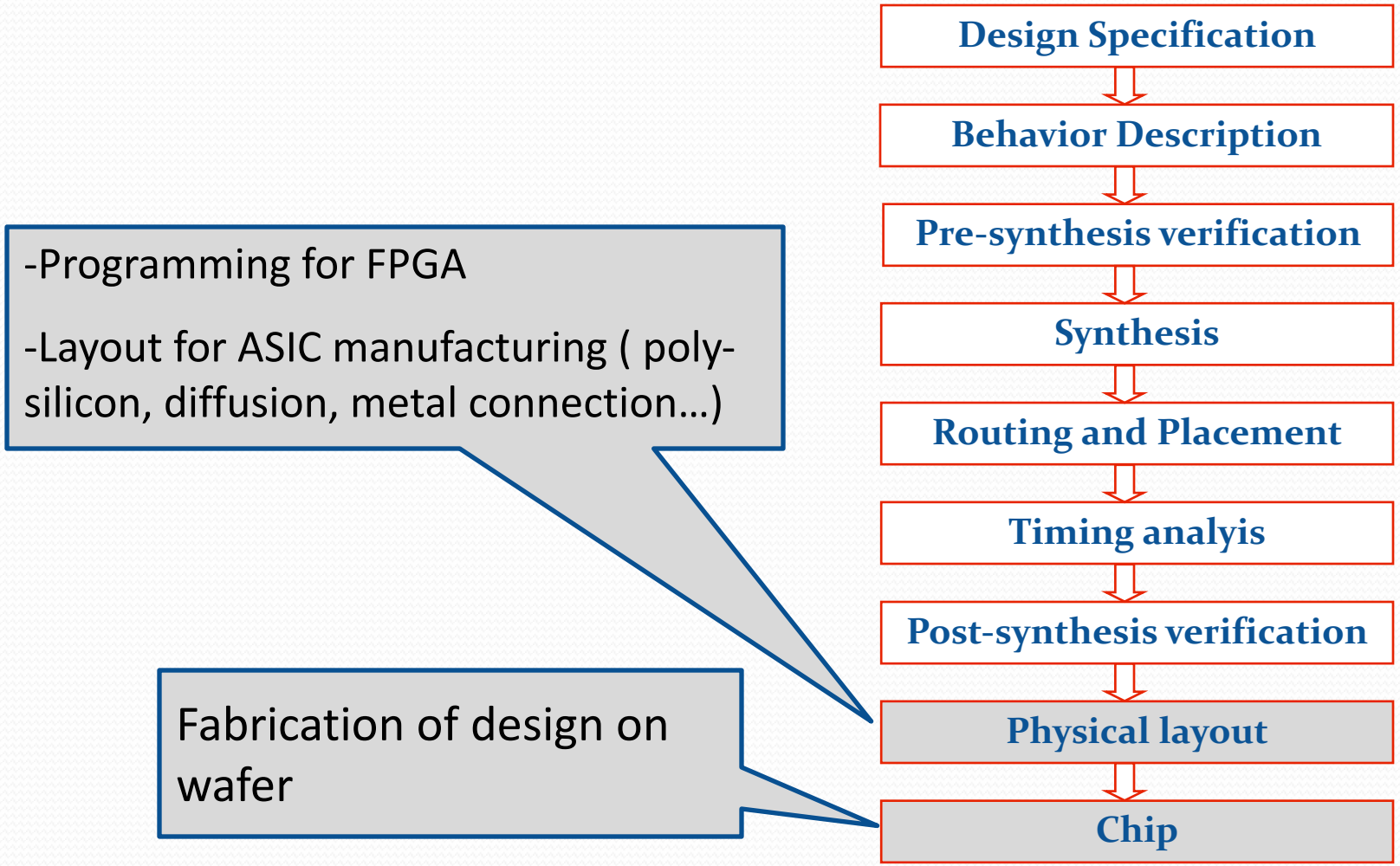
- Check functionality with timing; verify clock frequency, glitch, race condition, and delay timings

- Dynamic timing analysis





Digital design flow



Digital design flow

- Quartus CAD flow

- Reference Altera tutorial “Quartus II Introduction for Verilog user”

Summary

- HDLs are now the **dominant method** for large digital designs
- Verilog 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)

Summary

- **Verilog learning tips**

  - Pick up what you need from books and online tutorials
  - Learn from live code
  - Experiment with code, not by reading about it
  - The lowest level the course will reach is at gate level.