硬體描述語言 簡介
Verilog HDL Design

中興大學 電機系

范志鵬 教授

cpfan@dragon.nchu.edu.tw

References

[1] S. Palnitkar, “Verilog HDL: A Guide to Digital Design and Synthesis”

[2] David R. Smith and Paul D. Franzon, “ Verilog Styles for Synthesis of Digital Systems”, Prentice Hall 2000.

[3] Bob Zeidman, “Verilog Designer’s Library”, Prentice Hall 1999.

[4] Deepak Kumar Tala, Verilog Tutorial, 25-Oct-2003

[5] Ken Coffman, “ Real World FPGA Design with Verilog”, Prentice Hall 2000.

[6] SystemVerilog Symposium : Track 1 SystemVerilog Basic Training, Sunburst Design.

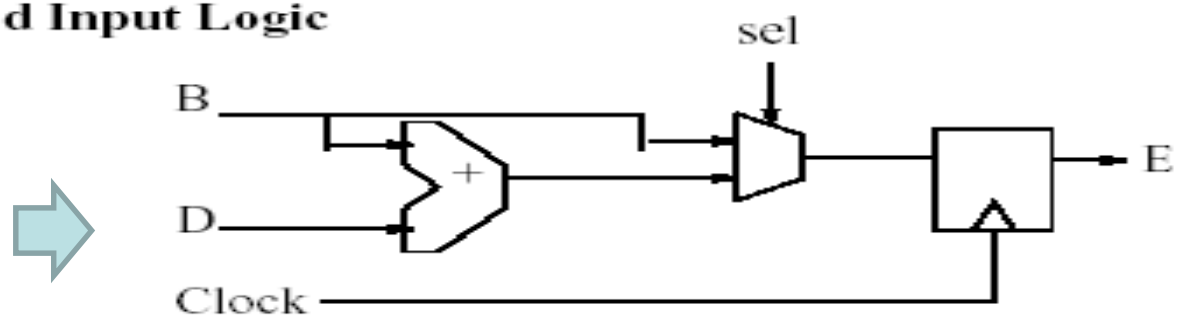
Overview

Digital circuit design has evolved rapidly over the last 25 years. The earliest digital circuits were designed with vacuum tubes and transistors. Integrated circuits were then invented where logic gates were placed on a single chip. The first integrated circuit (IC) chips were SSI (*Small Scale Integration*) chips where the gate count was very small. As technologies became sophisticated, designers were able to place circuits with hundreds of gates on a chip. These chips were called MSI (*Medium Scale Integration*) chips. With the advent of LSI (*Large Scale Integration*), designers could put thousands of gates on a single chip. At this point, design processes started getting very complicated, and designers felt the need to automate these processes. *Computer Aided Design* (CAD)[1] techniques began to evolve. Chip designers began to use circuit and logic simulation techniques to verify the functionality of building blocks of the order of about 100 transistors. The circuits were still tested on the breadboard, and the layout was done on paper or by hand on a graphic computer terminal.

Correspondence between Verilog and Hardware Being Modeled

Flip Flops and Associated Input Logic

```
always@(posedge clock)
  case (sel)
    0 : E <= D + B;
    1 : E <= B;
  endcase
```

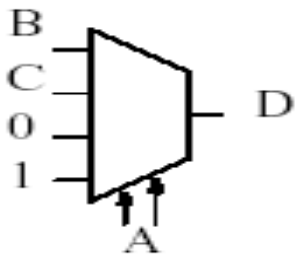


Simple Combinational Logic

```
assign H = C | F;
```

Complex Combinational Logic

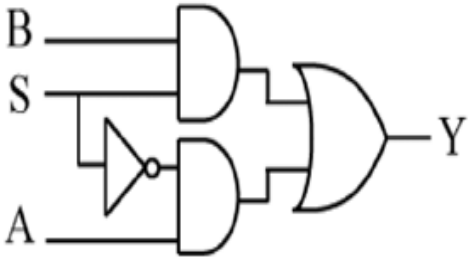
```
always@(A or B or C)
  case (A)
    2'b00 : D = B;
    2'b01 : D = C;
    2'b10 : D = 1'b0;
    2'b11 : D = 1'b1;
  endcase
```



Schematic with gate symbols

Boolean Equation

Verilog



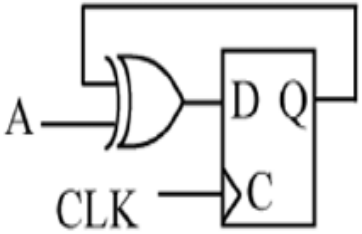
$Y = (B \wedge S) \vee (A \wedge \overline{S})$

```
assign y = (b & s) | (a & ~s);
```

or

```
always @(a or b or s)
  if (s) y = b; else y = a;
```

(a) Combinational logic



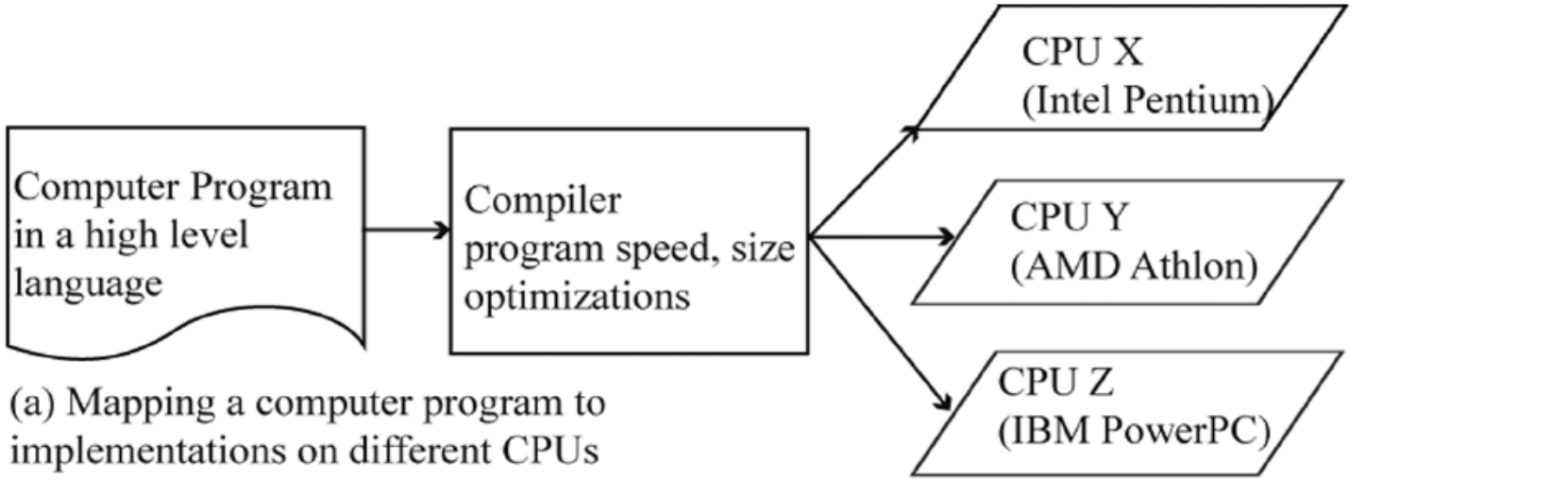
$Q_+ = Q \oplus A$

```
always @(posedge clk)
  q <= q ^ a;
```

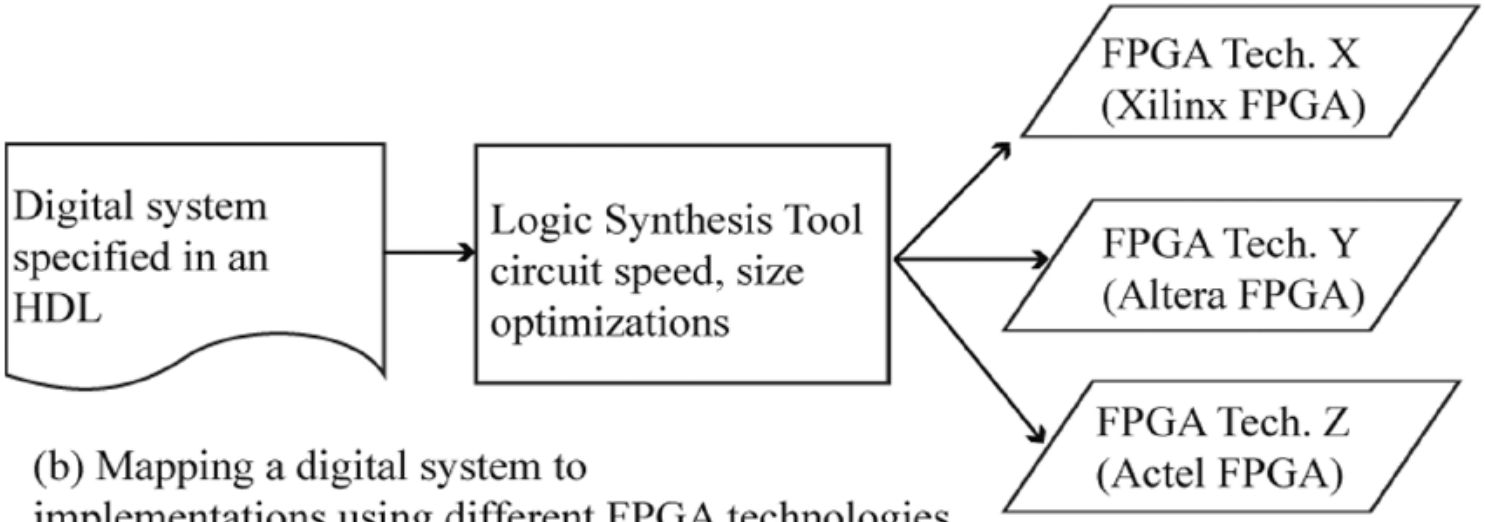
Must be annotated with truth table that describes Q+, Q dependence on D, CLK

(b) Sequential logic

Combinational and sequential logic representations



(a) Mapping a computer program to implementations on different CPUs



(b) Mapping a digital system to implementations using different FPGA technologies

HDLs vs. Computer programming

Emergence of HDLs

For a long time, programming languages such as FORTRAN, Pascal, and C were being used to describe computer programs that were sequential in nature. Similarly, in the digital design field, designers felt the need for a standard language to describe digital circuits. Thus, *Hardware Description Languages* (HDLs) came into existence. HDLs allowed the designers to model the concurrency of processes found in hardware elements. Hardware description languages such as *Verilog HDL* and VHDL became popular. Verilog HDL originated in 1983 at Gateway Design Automation. Later, VHDL was developed under contract from DARPA. Both Verilog® and VHDL simulators to simulate large digital circuits quickly gained acceptance from designers.

The advent of logic synthesis in the late 1980s changed the design methodology radically. Digital circuits could be described at a *register transfer level (RTL)* by use of an HDL. Thus, the designer had to specify how the data flows between registers and how the design processes the data. The details of gates and their interconnections to implement the circuit were automatically extracted by logic synthesis tools from the RTL description.

History of the Verilog HDL

Reference Material

- 1984: Gateway Design Automation introduced Verilog
- 1989: Gateway merged into Cadence Design Systems
- 1990: Cadence put Verilog HDL into the public domain
- 1993: OVI enhanced the Verilog language - *not well accepted*
- 1995: IEEE standardized the Verilog HDL (IEEE 1364-1995)
- 2001: IEEE standardized the Verilog IEEE Std1364-2001
- 2002: IEEE standardized the Verilog IEEE Std1364.1-2002
- 2002: Accellera standardized SystemVerilog 3.0
  - Accellera is the merged replacement of OVI & VHDL International (VI)
- 2003: Accellera standardized SystemVerilog 3.1
- 200?: IEEE Verilog with SystemVerilog enhancements

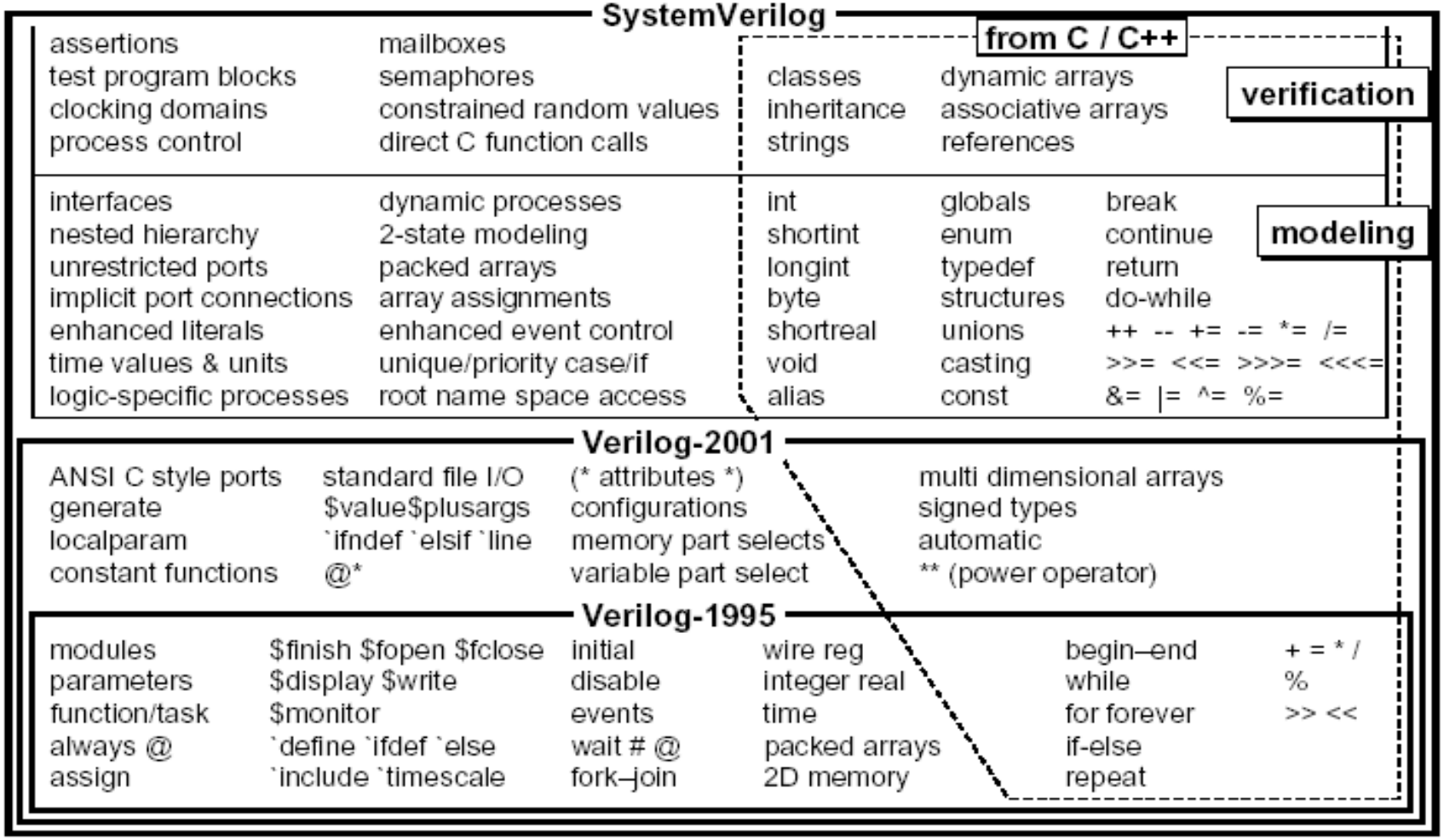
RTL synthesis subset

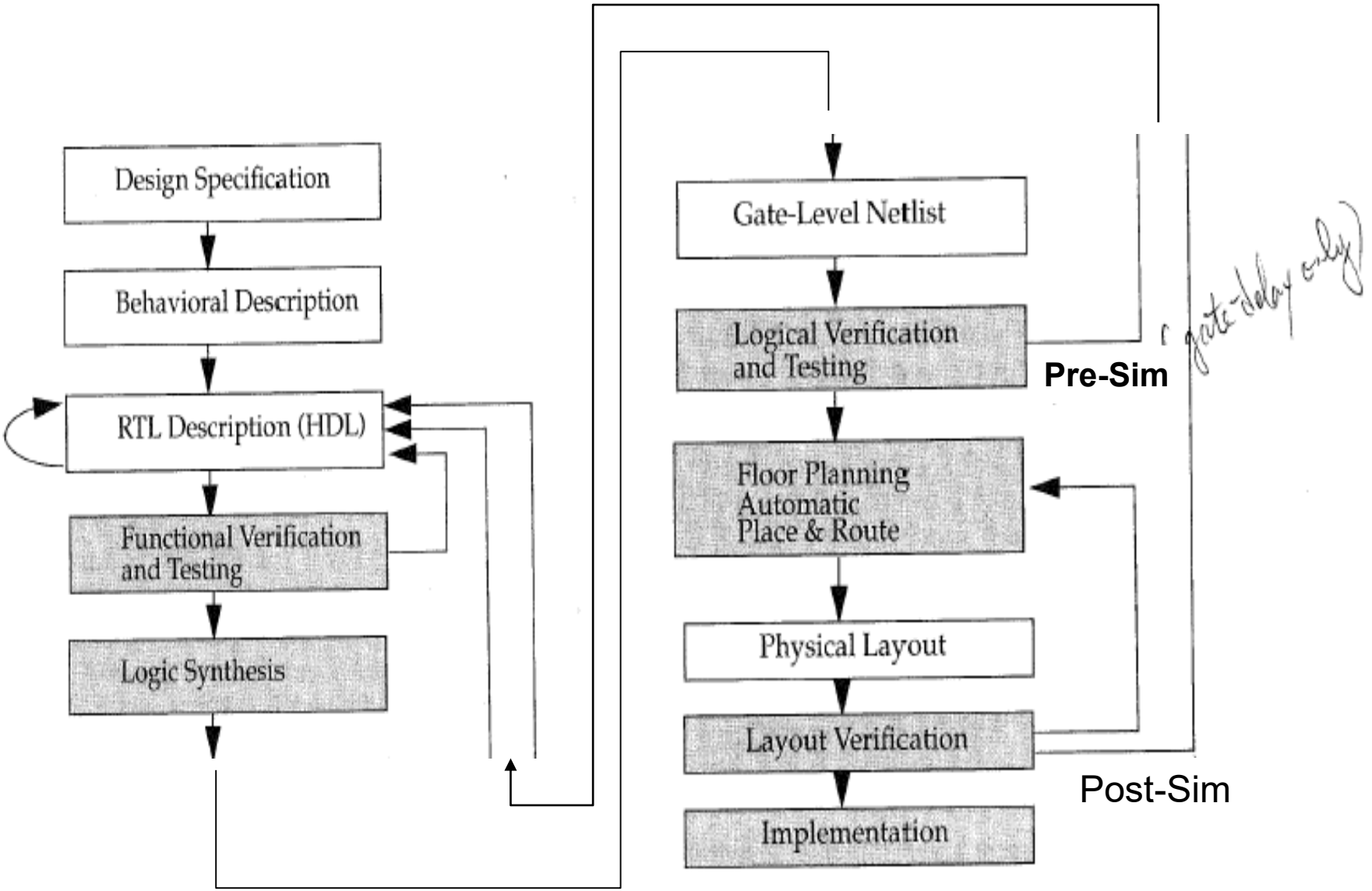
Why Call It SystemVerilog 3.x?

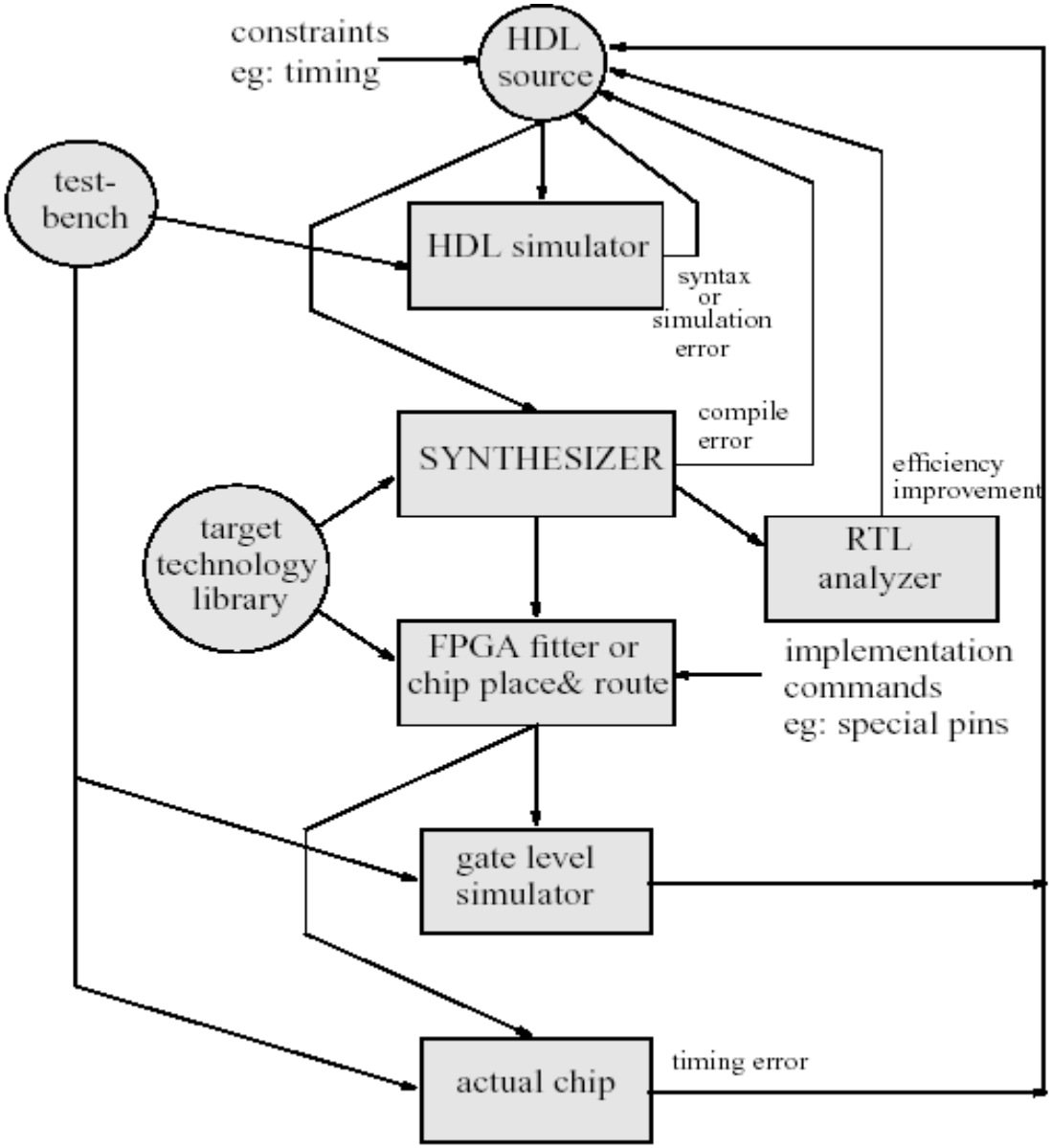
- SystemVerilog is *revolutionary evolution* of Verilog
- Verilog 1.0 - IEEE 1364-1995 "Verilog-1995" standard
  - The first IEEE Verilog standard
- Verilog 2.0 - IEEE 1364-2001 "Verilog-2001" standard
  - The second generation IEEE Verilog standard
  - Significant enhancements over Verilog-1995
- SystemVerilog 3.x - Accellera extensions to Verilog-2001
  - A third generation Verilog standard
  - DAC-2002 - SystemVerilog 3.0
  - DAC-2003 - SystemVerilog 3.1

The intention is to donate SystemVerilog to the IEEE Verilog committee for IEEE standardization

SystemVerilog Superset of Verilog-2001




Design Specification
Behavioral Description
RTL Description (HDL)
Functional Verification and Testing
Logic Synthesis
Gate-Level Netlist
Logical Verification and Testing
Pre-Sim
Floor Planning Automatic Place & Route
Physical Layout
Layout Verification
Post-Sim
Implementation

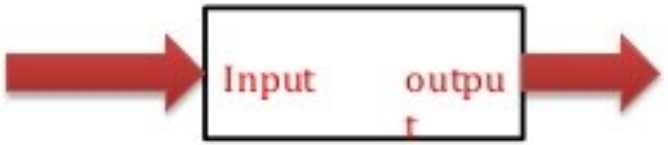


Hardware

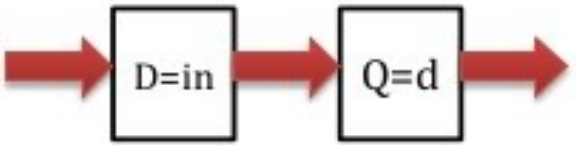
Synthesis

Based on Abstraction

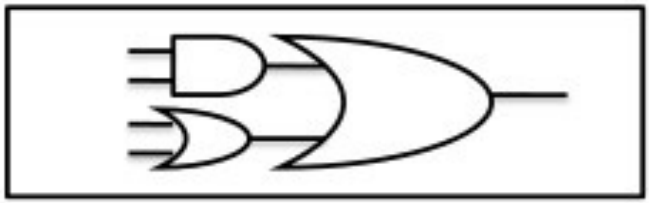
- Behavioral Level



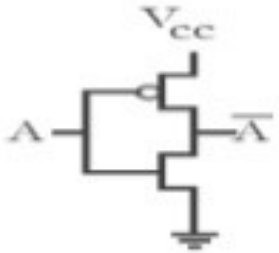
- Data flow Level



- Gate level

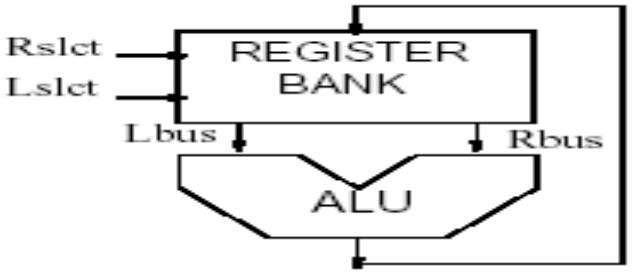


- Switch level

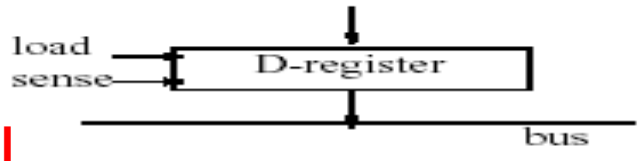


Hierarchy of Modeling Levels

(Behavioral-level) Architectural level

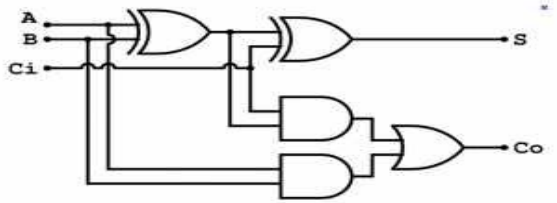


(Data flow-level) register transfer level

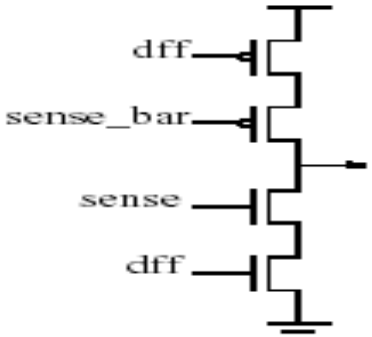


RTL-level

gate level



(Transistor-level) switch level



Synthesizable ?

- **Behavioral or algorithmic level**

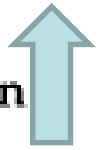
  This is the highest level of abstraction provided by Verilog HDL. A module can be implemented in terms of the desired design algorithm without concern for the hardware implementation details. Designing at this level is very similar to C programming.

- **Dataflow level**

  At this level the module is designed by specifying the data flow. The designer is aware of how data flows between hardware registers and how the data is processed in the design.

- **Gate level**

  The module is implemented in terms of logic gates and interconnections between these gates. Design at this level is similar to describing a design in terms of a gate-level logic diagram.



- **Switch level**

  This is the lowest level of abstraction provided by Verilog. A module can be implemented in terms of switches, storage nodes, and the interconnections between them. Design at this level requires knowledge of switch-level implementation details.