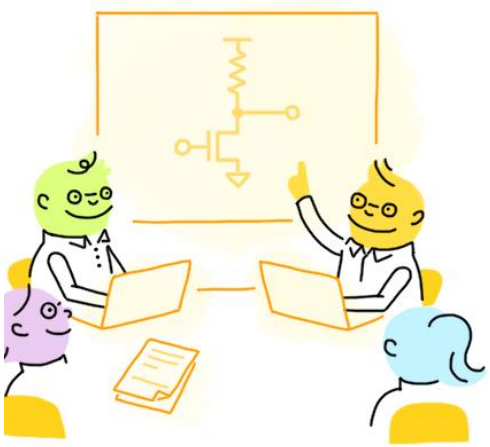
Digital System Design

Hajar Falahati

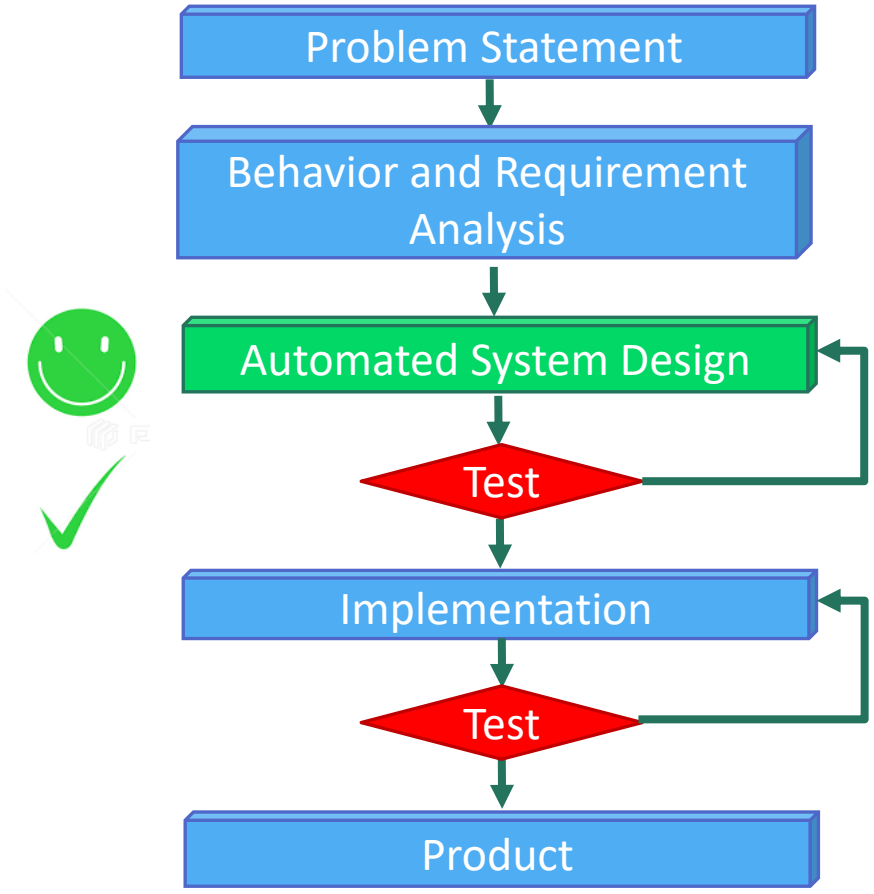
hfalahati@ipm.ir
hfalahati@ce.sharif.edu

System Design

- You are **expert** in designing
- Conventional design approach is challenging



Automated Design Flow

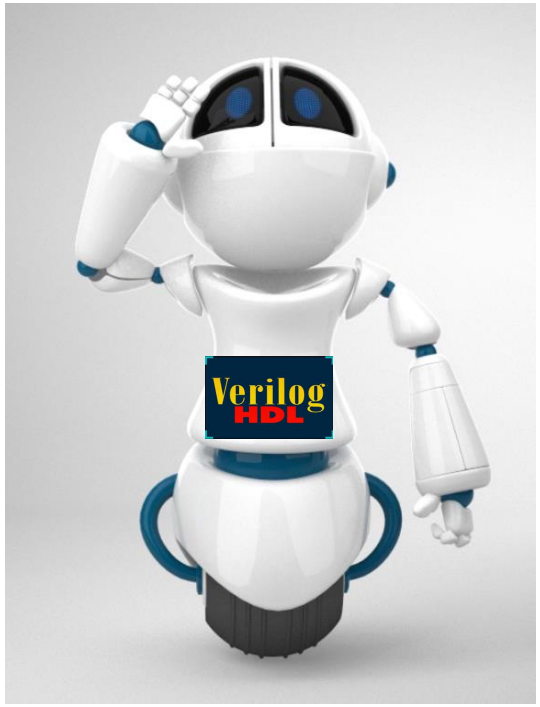


Outline

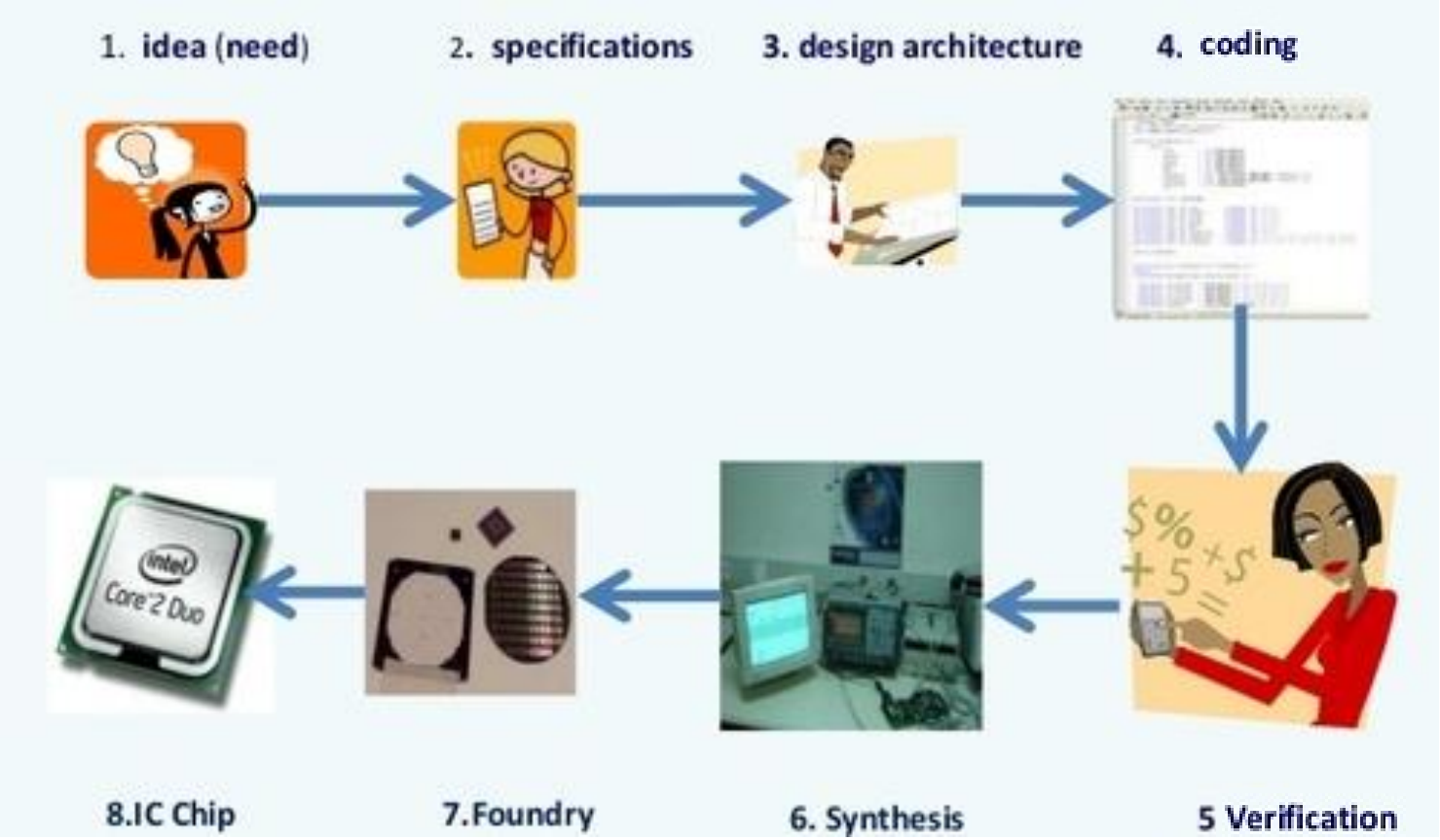
- Hello Verilog
  - Big picture
  - Levels

Hello Verilog

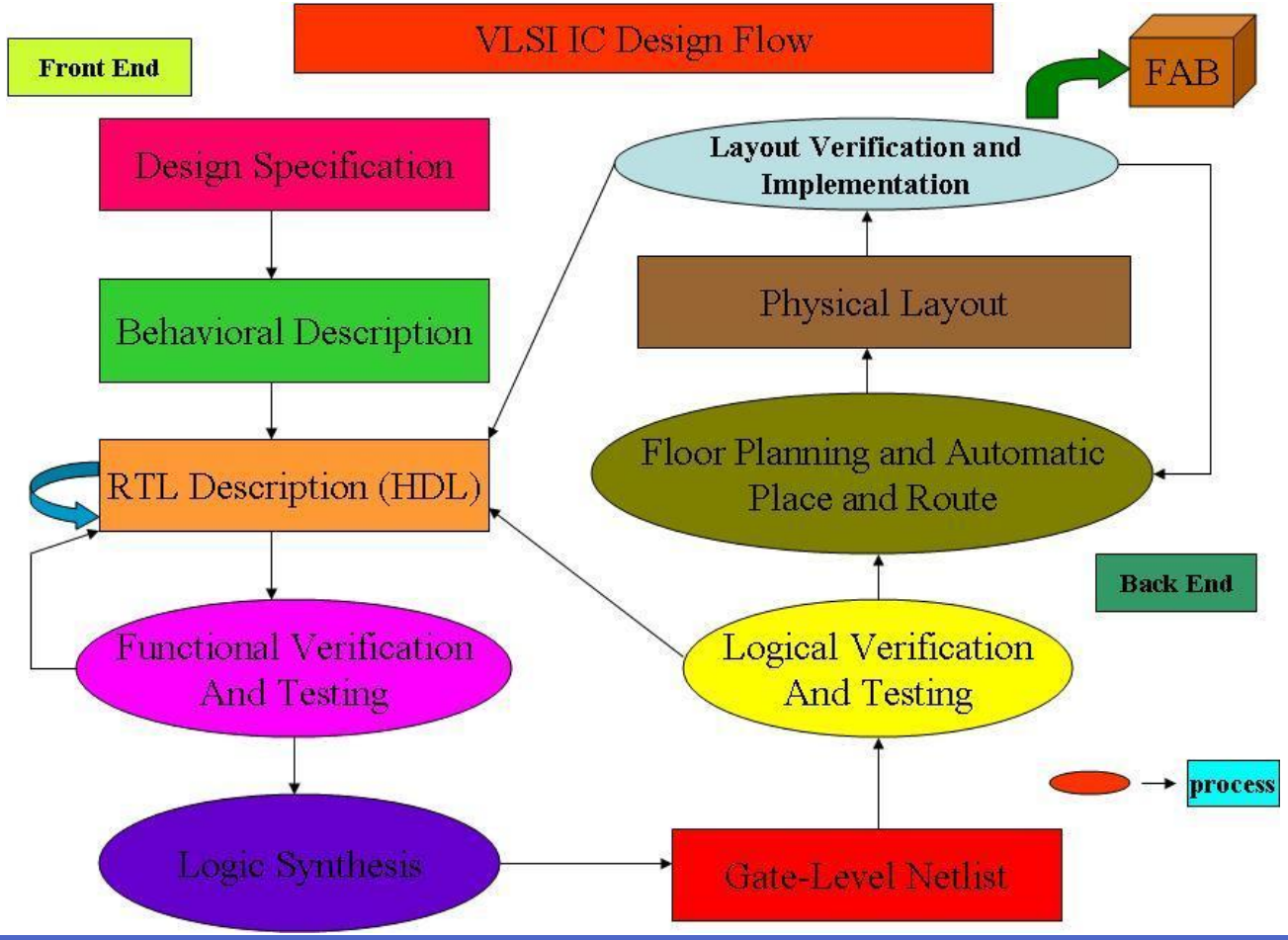
VERILOG: Hello Friends ☺



Design Flow

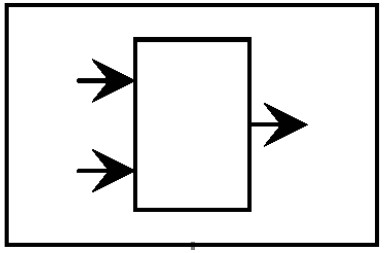

1. idea (need)
2. specifications
3. design architecture
4. coding
5 Verification
6. Synthesis
7.Foundry
8.IC Chip
intel
Core 2 Duo

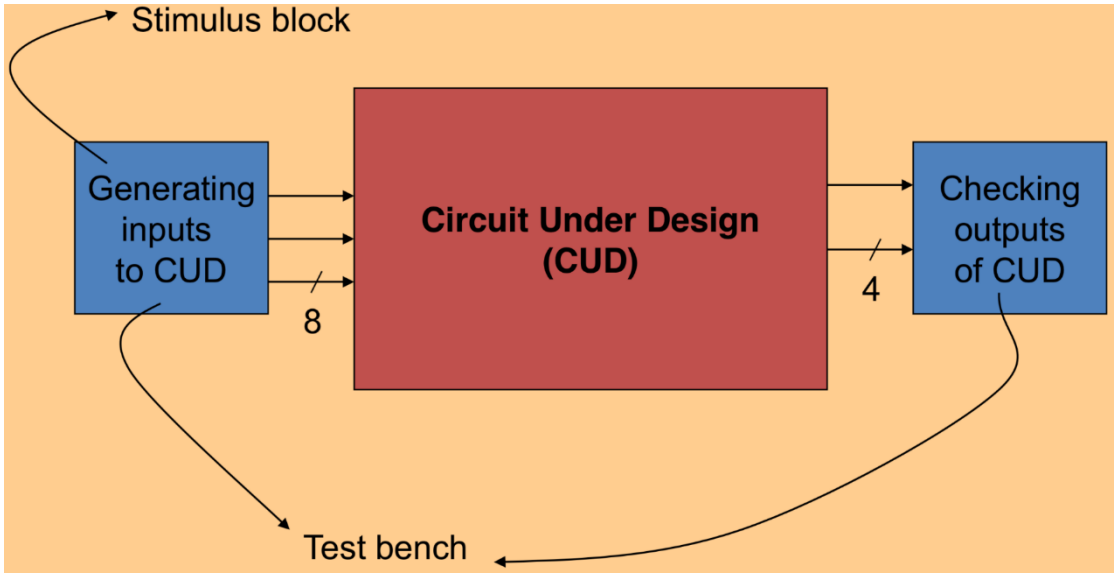
Design Flow in Detail

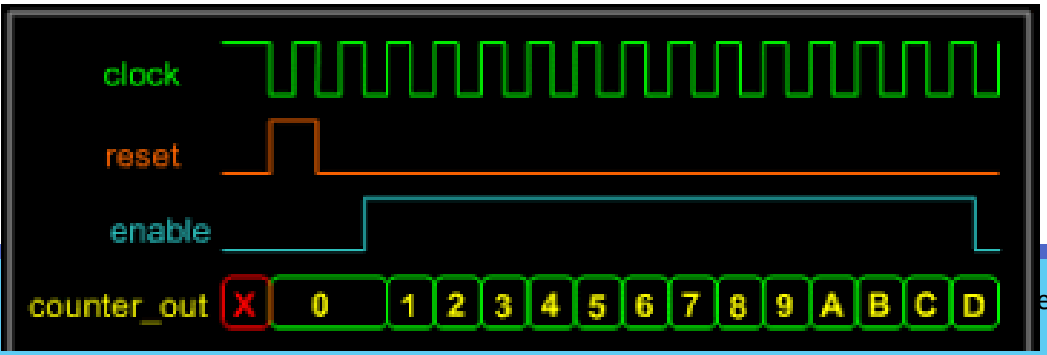


Simulation

- Design verification

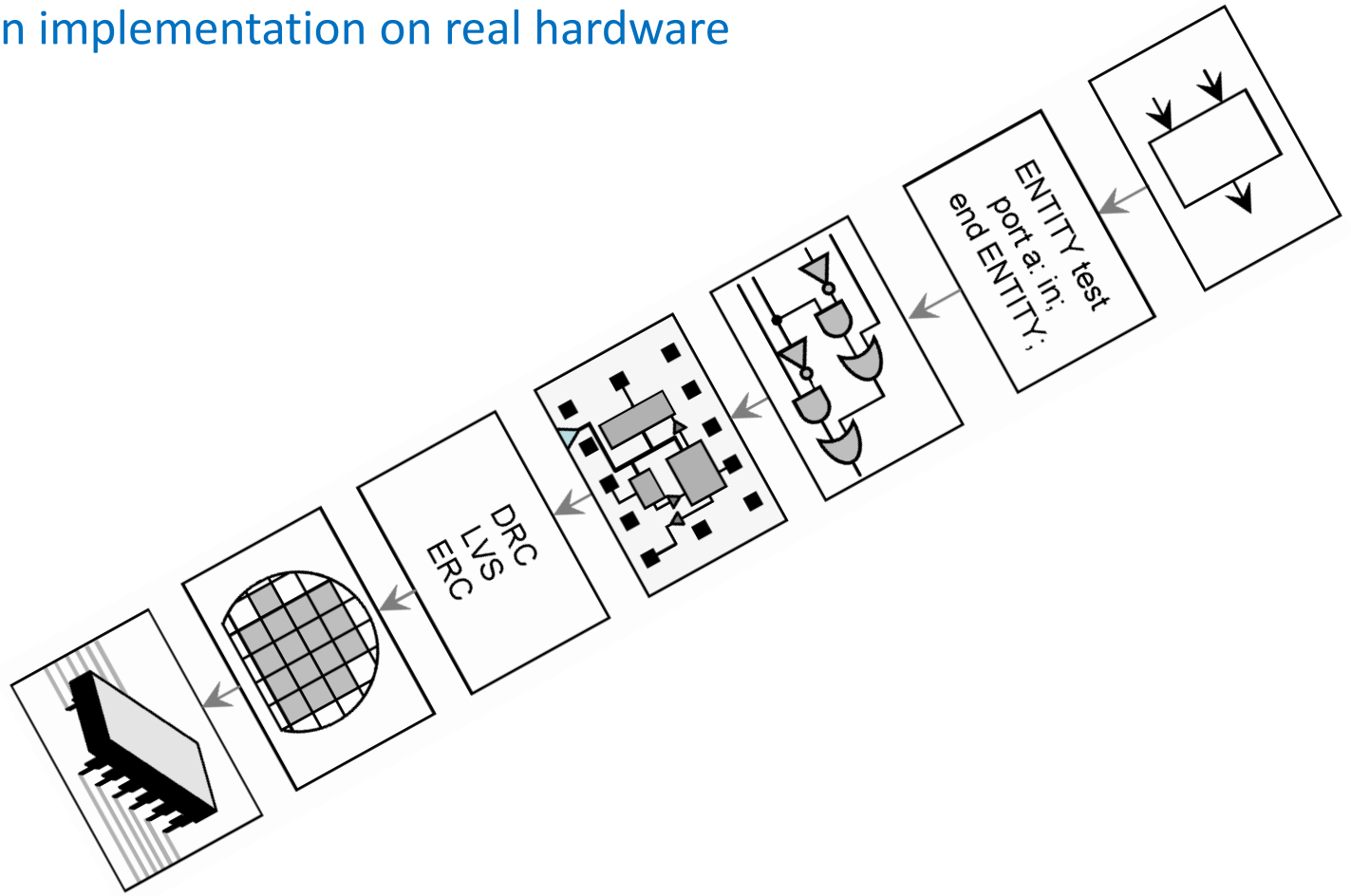


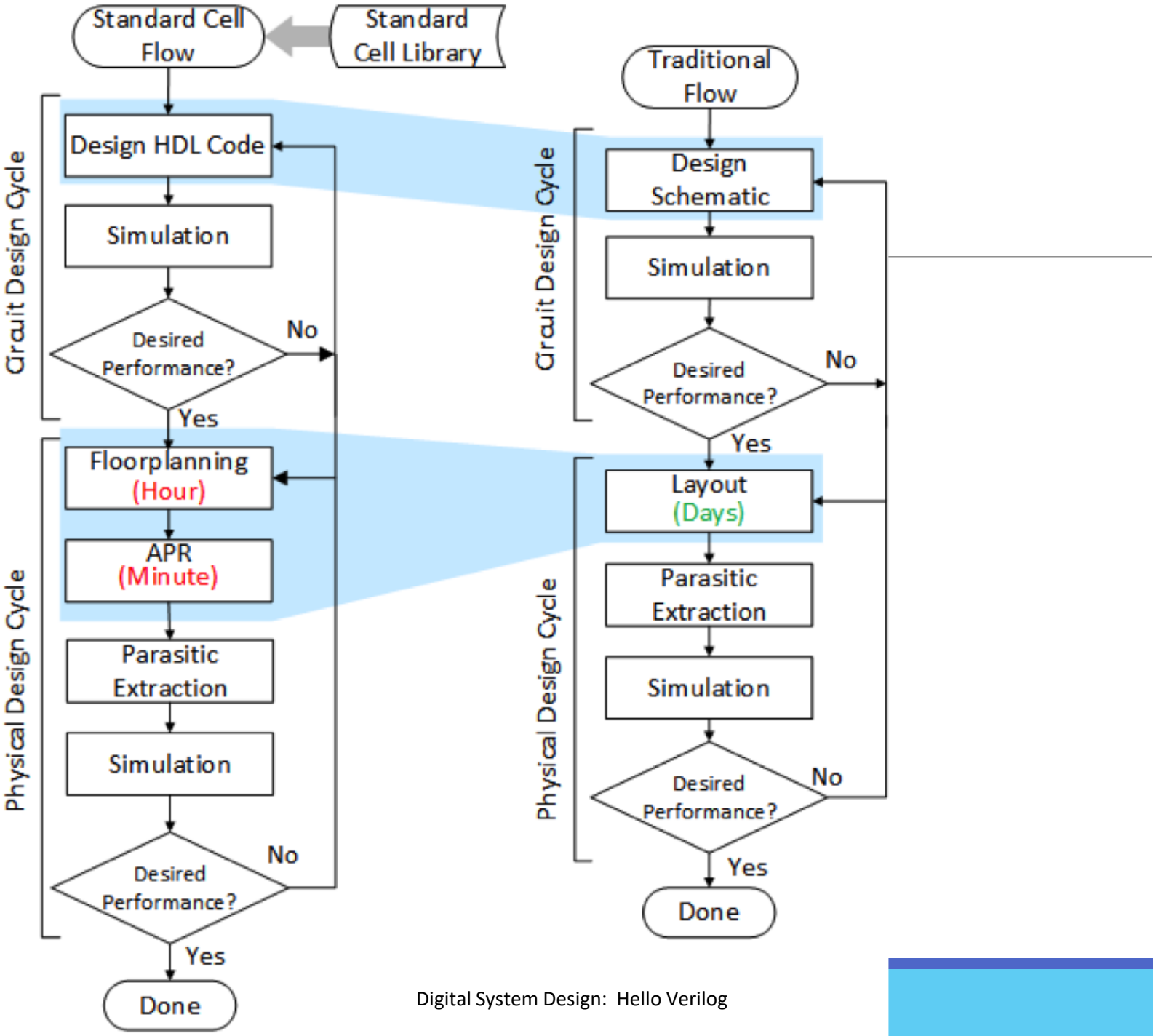




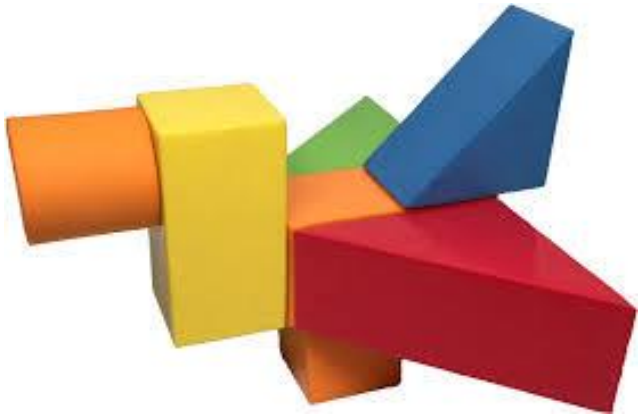
Synthesis

- Design implementation on real hardware




Standard Cell Flow
Standard Cell Library
Design HDL Code
Simulation
Desired Performance?
No
Yes
Floorplanning (Hour)
APR (Minute)
Parasitic Extraction
Simulation
Desired Performance?
No
Yes
Done
Cruit Design Cycle
Physical Design Cycle
Traditional Flow
Design Schematic
Simulation
Desired Performance?
No
Yes
Layout (Days)
Parasitic Extraction
Simulation
Desired Performance?
No
Yes
Done
Cruit Design Cycle
Physical Design Cycle

Basic Blocks



Module

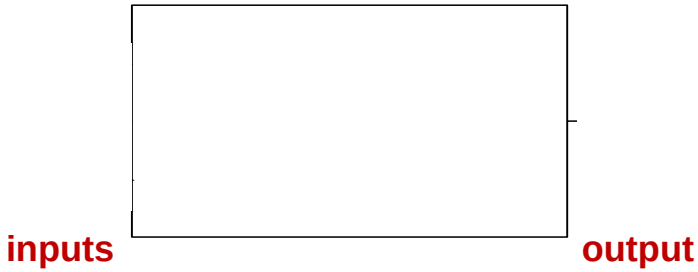
- A module is the **main building block** in Verilog
- Collection of **lower-level design block**
- Provides necessary **functionality** to **higher-level block**
- Hides **internal implementation**

Defining a Module in Verilog

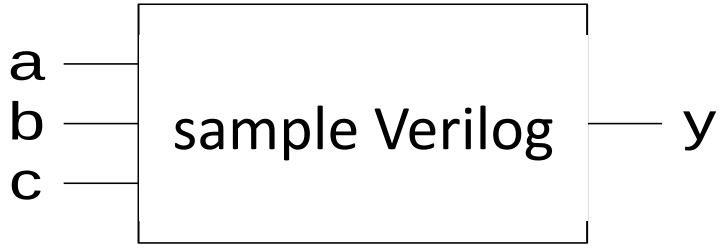
- We first need to define:
  - Name of the module
  - Directions of its ports (e.g., input, output)
  - Names of its ports

Then:

- Describe the functionality of the module



Implementing a Module in Verilog



name of module

Port list (inputs and outputs)

```
module sampleVerilog (a, b, c, y);
        input a;
        input b;
        input c;
        output y;

// here comes the circuit description
endmodule
```

ports have a declared type

a module definition

A Question of Style

The following two codes are functionally identical

```
module test ( a, b, y );
       input a;
       input b;
       output y;

endmodule
```

```
module test ( input a,
              input b,
              output y );

endmodule
```

port name and direction declaration
can be combined

What If We Have Multi-bit Input/Output?

- You can also define multi-bit Input/Output (Bus)
  - [range_end : range_start]
  - **Number of bits:** range_end – range_start + 1
- Example:
  - A represents a 32-bit value, so we prefer to define it as: `[31:0] a`
  - It is preferred over `[0:31] a` which resembles *array* definition
  - It is a good practice to be consistent with the representation of multi-bit signals, i.e., always [31:0] or always [0:31]

```
input  [31:0] a;   // a[31], a[30] .. a[0]
output [15:8] b1;  // b1[15], b1[14] .. b1[8]
output [7:0]  b2;  // b2[7], b2[6] .. b2[0]
input         c;   // single signal
```

Manipulating Bits

- Bit Slicing

```
// You can assign partial buses
   wire [15:0] longbus;
   wire [7:0] shortbus;
   assign shortbus = longbus[12:5];
```

Manipulating Bits

- Bit Slicing

```
// You can assign partial buses
   wire [15:0] longbus;
   wire [7:0] shortbus;
   assign shortbus = longbus[12:5];
```

- Concatenation

```
// Concatenating is by {}
   assign y = {a[2],a[1],a[0],a[0]};
```

Manipulating Bits

- Bit Slicing

```
// You can assign partial buses
   wire [15:0] longbus;
   wire [7:0] shortbus;
   assign shortbus = longbus[12:5];
```

- Concatenation

```
// Concatenating is by {}
   assign y = {a[2],a[1],a[0],a[0]};
```

- Duplication

```
// Possible to define multiple copies
   assign x = {a[0], a[0], a[0], a[0]}
   assign y = { 4{a[0]} }
```

Basic Syntax

- Similar to C
- **Case sensitive**
  - `SomeName` and `somename` are not the same!
- **Keywords are in lowercase**
- Names cannot **start** with **numbers**:
  - `2good` is not a valid name
- Whitespaces are **ignored**
- **Comments**

```
// Single line comments start with a //

/* Multiline comments
   are defined like this */
```

Operators

- Unary

```
a = b;
```

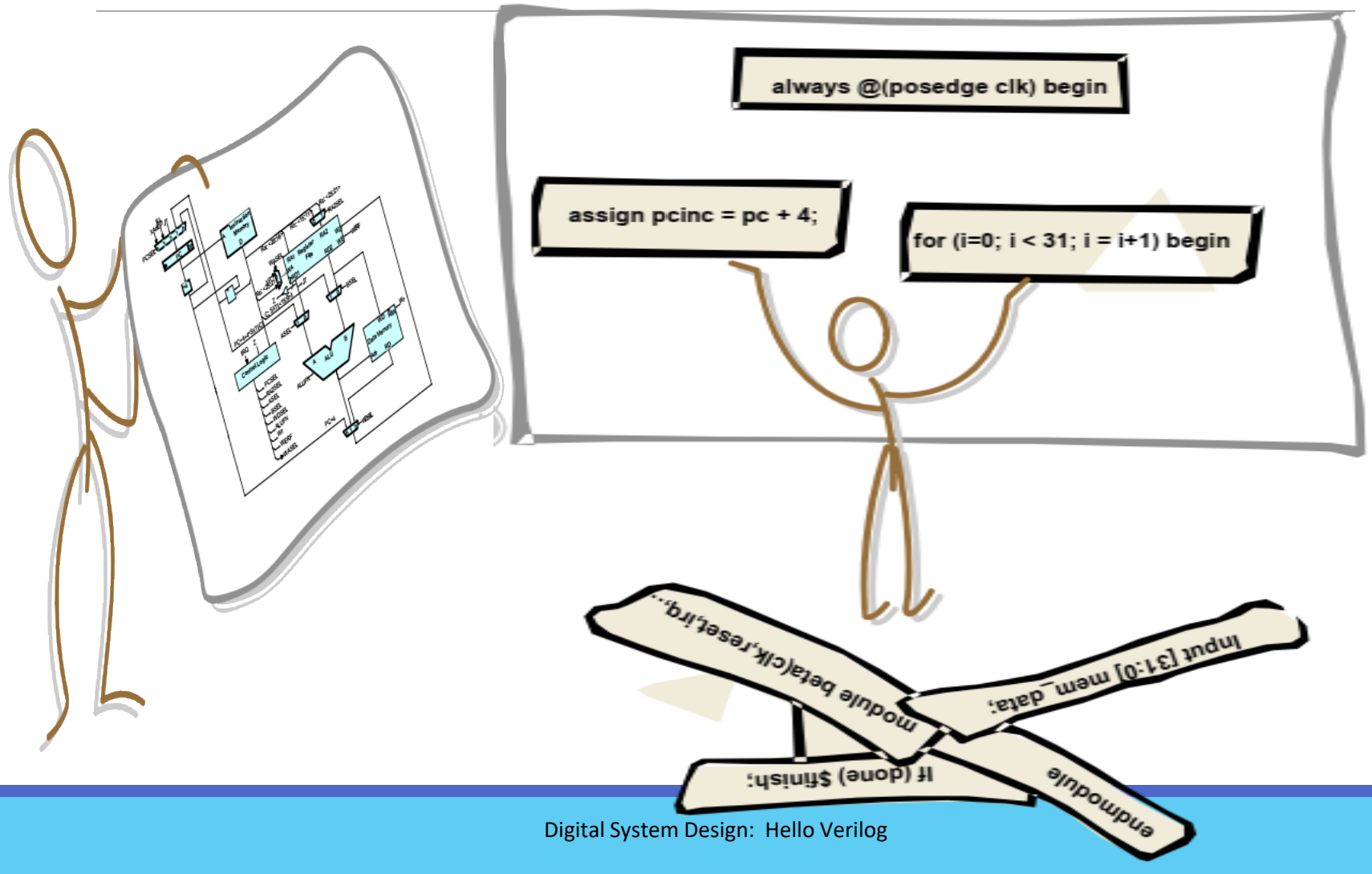
- Binary

```
a = b && c;
```

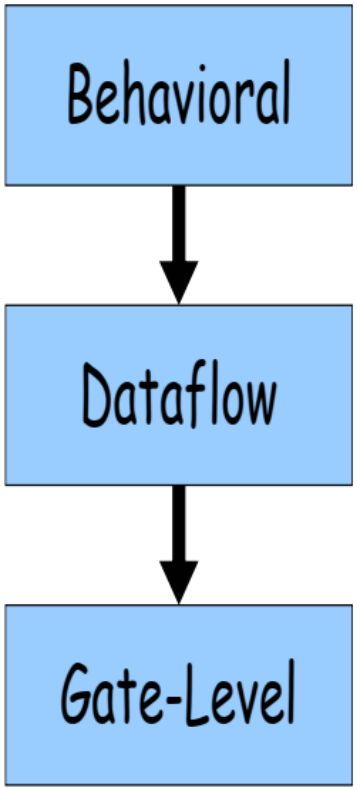
- Ternary
  - The only ternary operator

```
a = b ? c:d;
```

Design


always @(posedge clk) begin
assign pcinc = pc + 4;
for (i=0; i < 31; i = i+1) begin
module beta(clk,reset,irq,...
input [31:0] mem_data;
endmodule
if (done) $finish;

Three Common Abstraction Level

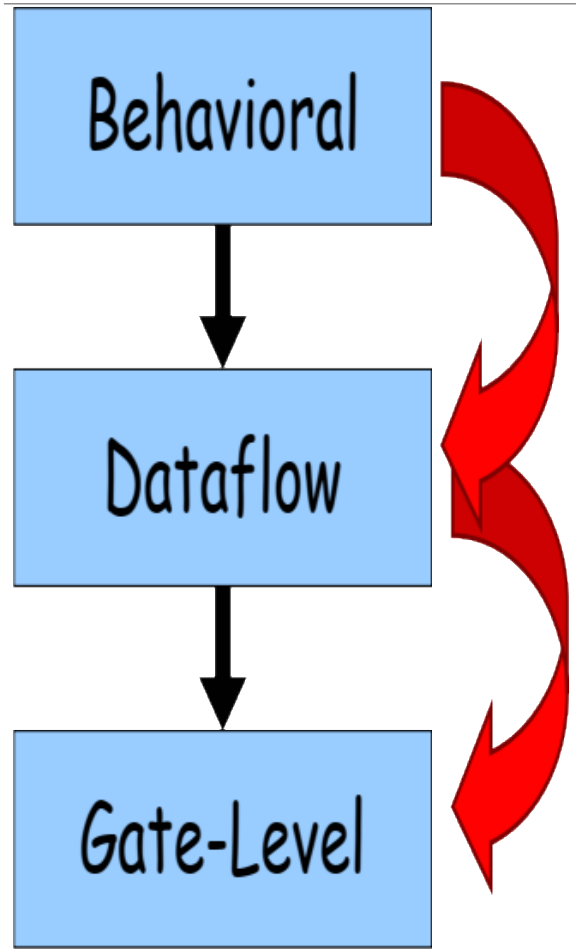


Module's high-level algorithm is implemented with little concern for the actual hardware

Module is implemented by specifying how data flows between registers

Module is implemented in terms of concrete logic gates (AND, OR, NOT) and their interconnections

Three Common Abstraction Level



Designers can create lower-level models from the higher-level models either manually or automatically

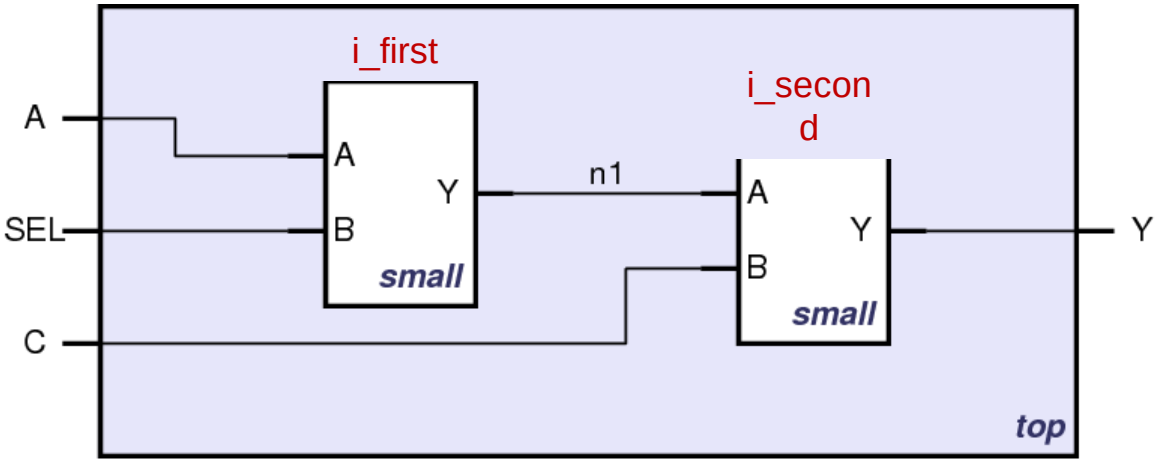
The process of automatically generating a gate-level model from either a dataflow or a behavioral model is called

Logic Synthesis

Structural Style

- **Structural (Gate-Level)**
  - The module body contains gate-level description of the circuit
  - Describe how modules are interconnected
  - Each module contains other modules (instances)
  - … and interconnections between these modules
  - Describes a hierarchy

Structural HDL: Instantiating a Module



Schematic of module "top" that is built from two instances of module "small"

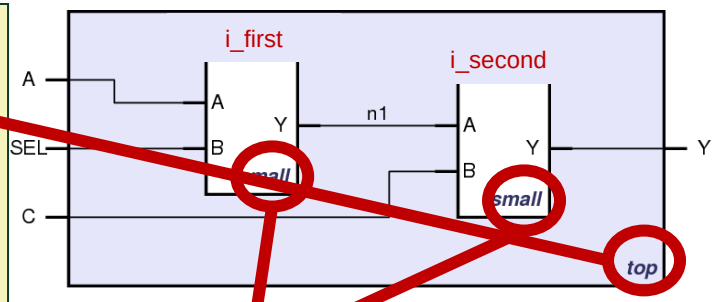
Structural HDL Example

Module Definitions in Verilog

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;




endmodule
```




```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

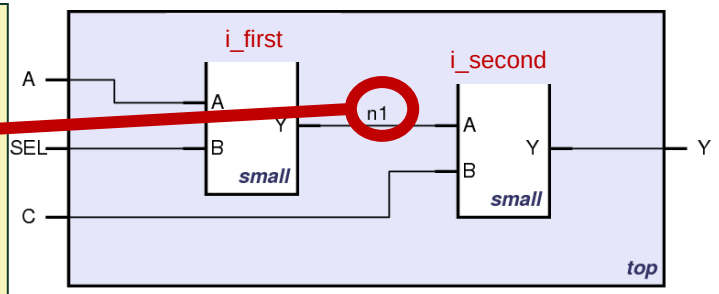
Structural HDL Example

Defining wires (module interconnections)

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;




endmodule
```




```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example

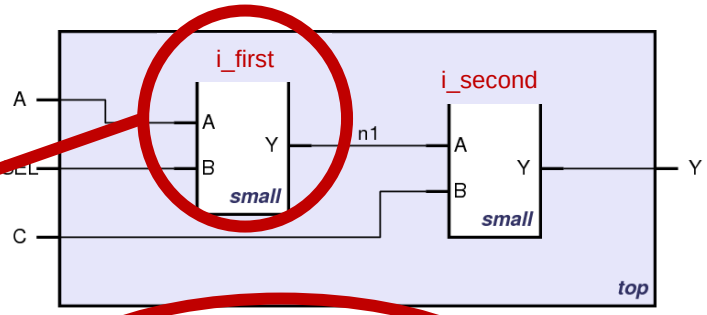
The first instantiation of the "small" module

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// instantiate small once
small i_first ( .A(A),
                .B(SEL),
                .Y(n1)   );




endmodule
```




```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example

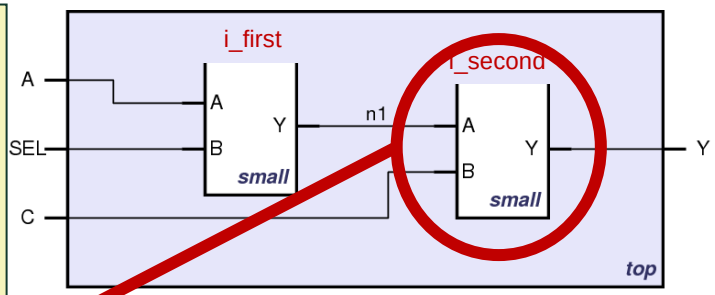
The second instantiation of the “small” module

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// instantiate small once
small i_first ( .A(A),
                .B(SEL),
                .Y(n1)   );

// instantiate small second time
small i_second ( .A(n1),
           .B(C),
           .Y(Y) );

endmodule
```




```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example

Short form of module instantiation

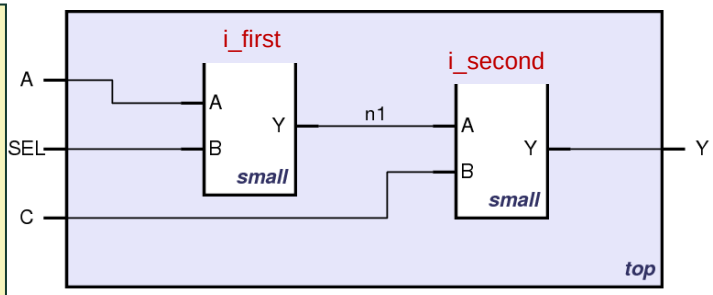
```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// alternative
small i_first ( A, SEL, n1 );

/* Shorter instantiation,
   pin order very important */

// any pin order, safer choice
small i_second ( .B(C),
           .Y(Y),
           .A(n1) );

endmodule
```




```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example 2

- Verilog supports basic logic gates as predefined *primitives*
  - These primitives are instantiated like modules except that they are predefined in Verilog and *do not need a module definition*

```
module mux2(input  [3:0] d0, d1,
            input        s,
            output [3:0] y);

   wire ns;
   and  g1 (y1, d0, ns);
   and  g2 (y2, d1, s);
   or   g3 (y, y1, y2);
   not  g4 (ns, s);
endmodule
```

Data Flow Modeling

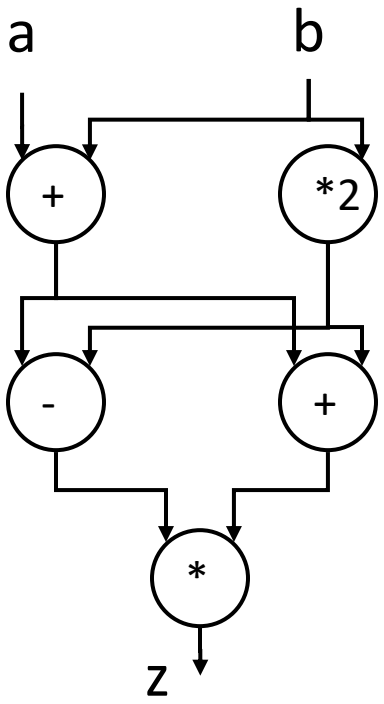
- Continuous assignment

```
v  = a + b;
w = b * 2;
x = v - w
y = v + w
z = x * y
```

Sequential

```
module sampleDataFlow (a, b, z);
        input a, b;
        output z;
        wire  v,w,x,y;
    assign v = a+b;
     assign w = b * 2;
    assign x = v-w;
    assign y = v+w;
    assign z = x*y;

endmodule
```



Dataflow

Behavioral

- The module body contains functional description of the circuit
- Contains logical and mathematical operators
- **Level of abstraction is higher than gate-level**
  - Many possible gate-level realizations of a behavioral description

Thank You