Verilog HDL:
Structural Design

Pravin Zode

Outline

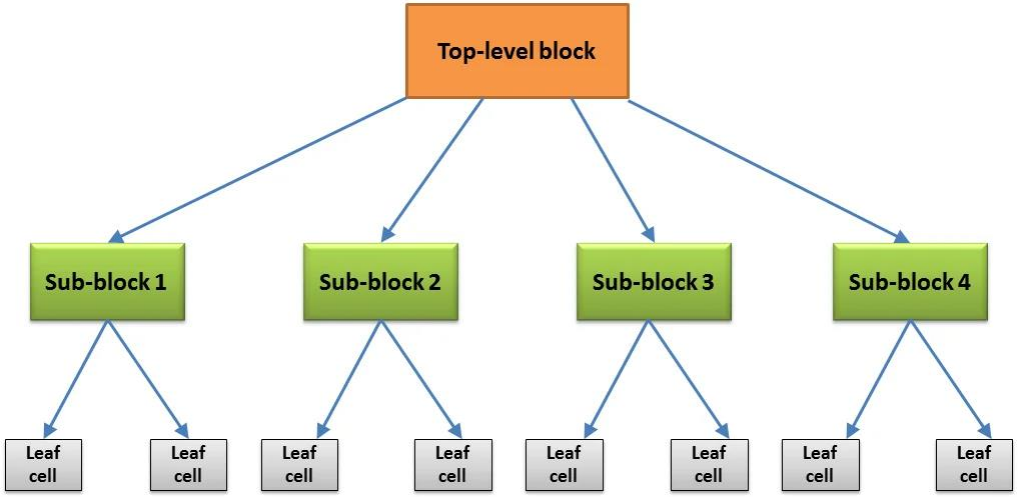
- Structural Design
- Top-Down Design Methodology
- Example : Adder Circuit

Structural Design

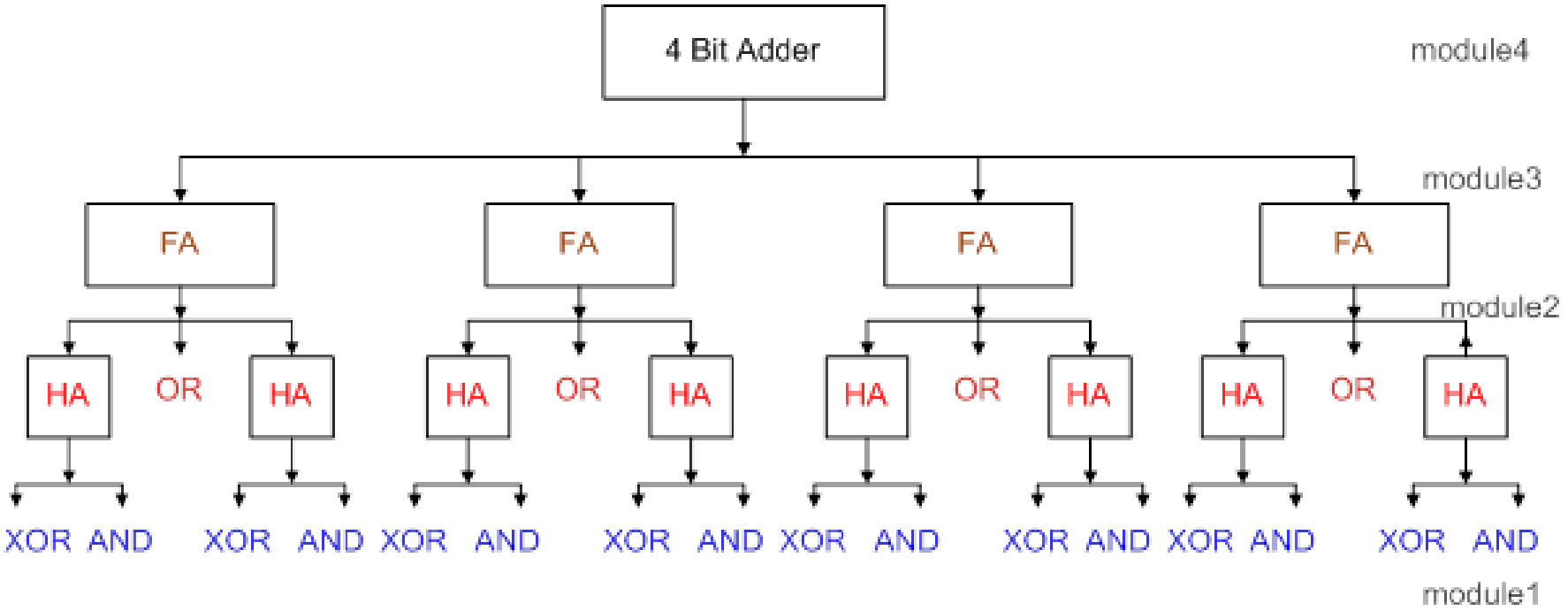
- Structural design partitioning for better organization and reuse
- Modular approach improves design clarity and maintainability.
- Simplifies debugging and verification by dividing the design into smaller components

Top-Down Design Methodology

- **Top-Down Design Methodology**
  - Starts with the top-level block and identifies required sub-blocks
  - Sub-blocks are further divided until leaf cells are reached
  - Leaf cells are the smallest units that cannot be further divided
  - Helps in systematic partitioning and abstraction.

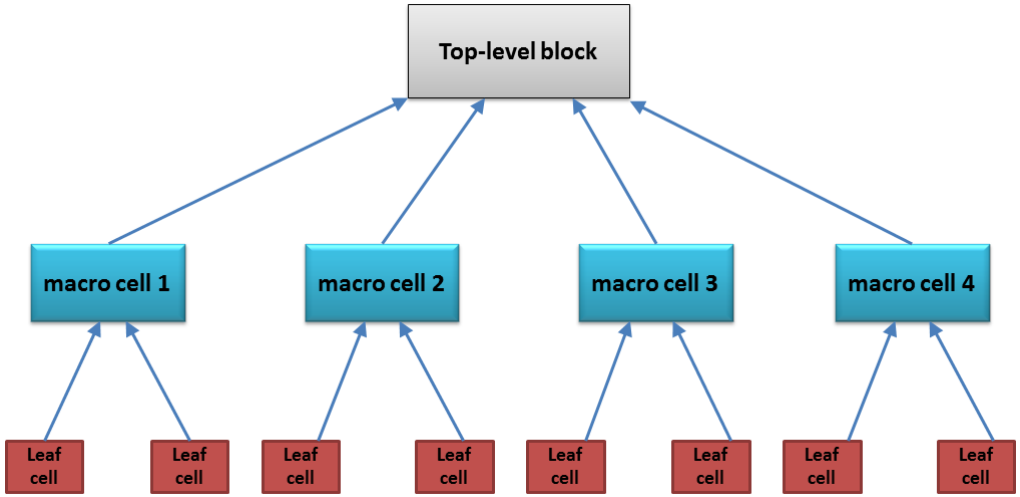


Top-Down Design Methodology



Bottom-Up Design Methodology

- Begins with available building blocks
- Smaller blocks are combined to form larger blocks
- Continues until the top-level block is constructed
- Useful when pre-designed components are available.



Mixed Design Methodology

Combination of Top-Down and Bottom-Up Approaches

- Design architects define the top-level specifications
- Logic designers break down the functionality into blocks and sub-blocks
- Circuit designers optimize leaf cells and build higher-level circuits
- The process meets at an intermediate point, integrating both approaches
- Ensures efficient design, optimization, and reusability

Verilog Gates

Name	Description	Usage
and	$f = (a \cdot b \cdot \cdots)$	**and** $(f, a, b, \ldots)$
nand	$f = \overline{(a \cdot b \cdot \cdots)}$	**nand** $(f, a, b, \ldots)$
or	$f = (a + b + \cdots)$	**or** $(f, a, b, \ldots)$
nor	$f = \overline{(a + b + \cdots)}$	**nor** $(f, a, b, \ldots)$
xor	$f = (a \oplus b \oplus \cdots)$	**xor** $(f, a, b, \ldots)$
xnor	$f = (a \odot b \odot \cdots)$	**xnor** $(f, a, b, \ldots)$
not	$f = \overline{a}$	**not** (f, a)
buf	$f = a$	**buf** (f, a)
notif0	$f = (!e \; ? \; \overline{a} : \text{'}bz)$	**notif0** (f, a, e)
notif1	$f = (e \; ? \; \overline{a} : \text{'}bz)$	**notif1** (f, a, e)
bufif0	$f = (!e \; ? \; a : \text{'}bz)$	**bufif0** (f, a, e)
bufif1	$f = (e \; ? \; a : \text{'}bz)$	**bufif1** (f, a, e)

Structural Description Adder

```
module fulladd (Cin, x, y, s, Cout);
    input Cin, x, y;
    output s, Cout;

    xor (s, x, y, Cin);
    and (z1, x, y);
    and (z2, x, Cin);
    and (z3, y, Cin);
    or (Cout, z1, z2, z3);

endmodule
```

```
module fulladd (Cin, x, y, s, Cout);
    input Cin, x, y;
    output s, Cout;

    xor (s, x, y, Cin);
    and (z1, x, y),
        (z2, x, Cin),
        (z3, y, Cin);
    or (Cout, z1, z2, z3);

endmodule
```

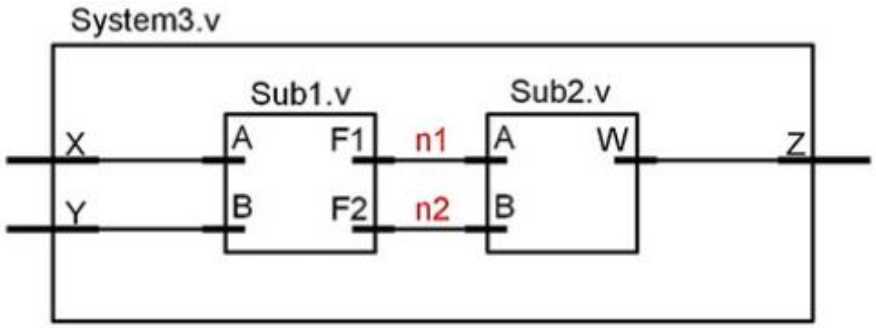
Structural Description Adder

```
module fulladd (Cin, x, y, s, Cout);
    input Cin, x, y;
    output s, Cout;

    xor (s, x, y, Cin);
    and (z1, x, y);
    and (z2, x, Cin);
    and (z3, y, Cin);
    or (Cout, z1, z2, z3);

endmodule
```

```
module fulladd (Cin, x, y, s, Cout);
    input Cin, x, y;
    output s, Cout;
    wire z1, z2, z3, z4;

    and And1 (z1, x, y);
    and And2 (z2, x, Cin);
    and And3 (z3, y, Cin);
    or Or1 (Cout, z1, z2, z3);
    xor Xor1 (z4, x, y);
    xor Xor2 (s, z4, Cin);

endmodule
```

Port Mapping

Explicit Port Mapping




```
module Sub1 (output wire F1, F2,
             input  wire A,  B);

  // behavior here...

endmodule
```

```
module Sub2 (output wire W,
             input  wire A,  B);

  // behavior here...

endmodule
```

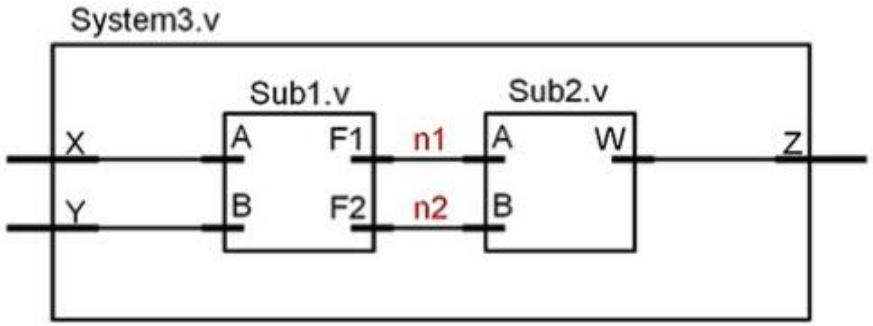
```
module System3 (output wire Z,
                input  wire X, Y);

   wire n1, n2;

   Sub1 U0 (.F1(n1), .F2(n2), .A(X),  .B(Y));
   Sub2 U1 (.W(Z),   .A(n1),  .B(n2));

endmodule
```

The lower-level port name is explicitly listed (preceded by a period).

The signal being connected to the lower-level port is listed inside of parenthesis.

Port Mapping

Positional Port Mapping



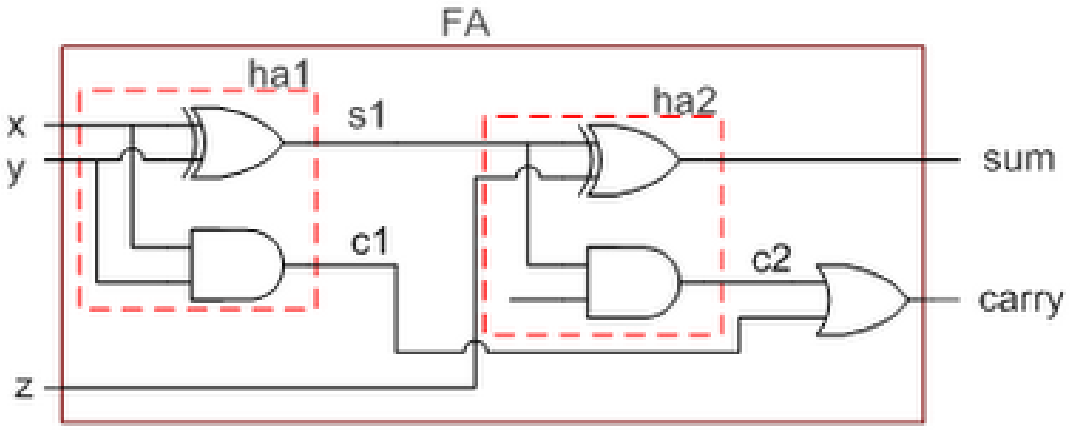

```
module Sub1 (output wire F1, F2,
             input  wire A,  B);

  // behavior here...

endmodule
```

```
module Sub2 (output wire W,
             input  wire A,  B);

  // behavior here...

endmodule
```

```
module System3 (output wire Z,
                input  wire X, Y);

   wire n1, n2;

   Sub1 U0 (n1, n2, X, Y);
   Sub2 U1 (Z, n1, n2);

endmodule
```

The signals to be connected to the lower-level module must be listed in the same order as they were defined in the lower-level system.

Full Adder



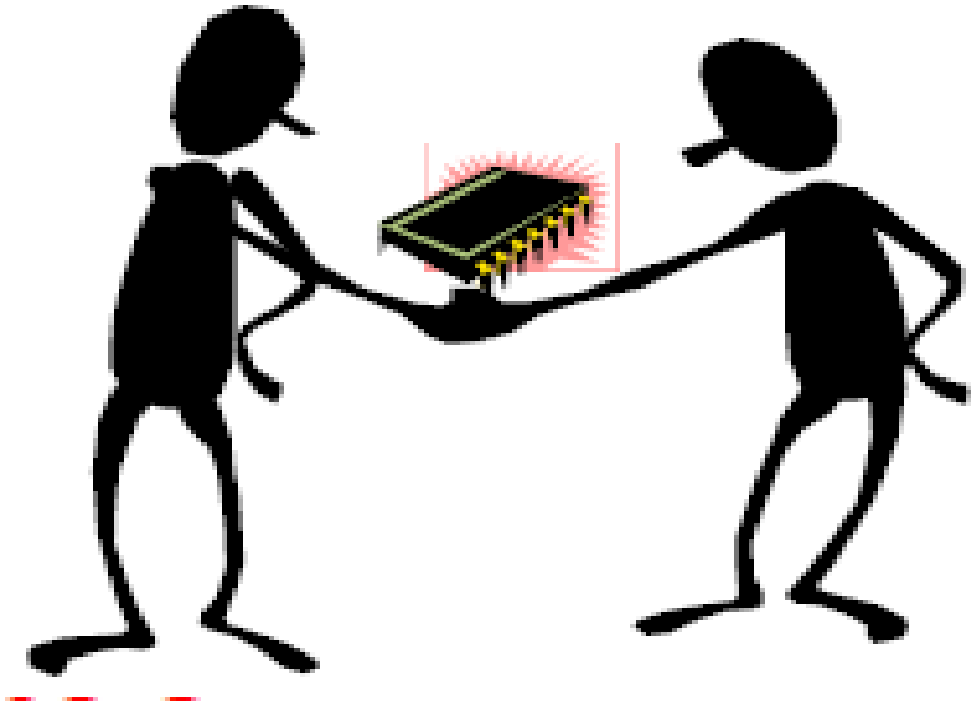

```
module full_adder(sum,carry,x,y,z)
        input x,y,z;
        output sum,carry;
                half_adder ha1(.s(s1), .c(c1), .a(x), .b(y));
                //half_adder ha1(s1,c1,a,b);//above method is good
                half_adder ha2(.s(sum), .c(c2), .a(s1), .b(z));
                or o1(.c(carry), .a(c1), .b(c2));
endmodule
```

Full Adder

```
module  adder4 (carryin, x3, x2, x1, x0, y3, y2, y1, y0, s3, s2, s1, s0, carryout);
    input  carryin, x3, x2, x1, x0, y3, y2, y1, y0;
    output  s3, s2, s1, s0, carryout;

    fulladd  stage0  (carryin, x0, y0, s0, c1);
    fulladd  stage1  (c1, x1, y1, s1, c2);
    fulladd  stage2  (c2, x2, y2, s2, c3);
    fulladd  stage3  (c3, x3, y3, s3, carryout);

endmodule

module  fulladd (Cin, x, y, s, Cout);
    input  Cin, x, y;
    output  s, Cout;

    assign  s = x ^ y ^ Cin;
    assign  Cout = (x & y) | (x & Cin) | (y & Cin);

endmodule
```

Full Adder

```
module  adder4 (carryin, X, Y, S, carryout);
    input  carryin;
    input  [3:0] X, Y;
    output  [3:0] S;
    output  carryout;
    wire  [3:1] C;

    fulladd  stage0  (carryin, X[0], Y[0], S[0], C[1]);
    fulladd  stage1  (C[1], X[1], Y[1], S[1], C[2]);
    fulladd  stage2  (C[2], X[2], Y[2], S[2], C[3]);
    fulladd  stage3  (C[3], X[3], Y[3], S[3], carryout);

endmodule
```



Thank you !

Happy Learning