Logic and Computer Design Fundamentals

Chapter 2 – Combinational Logic Circuits

Part 4 – HDLs Overview

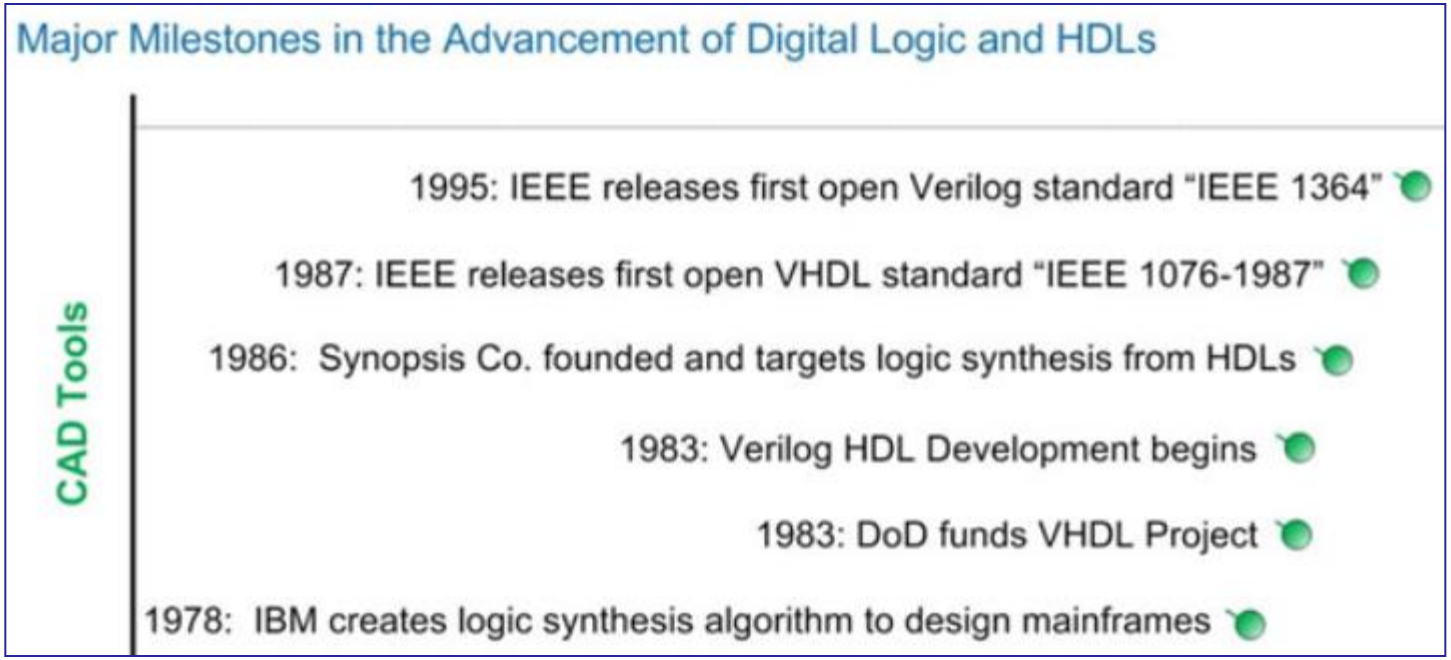
Ming Cai
cm@zju.edu.cn
College of Computer Science and Technology
Zhejiang University

Overview

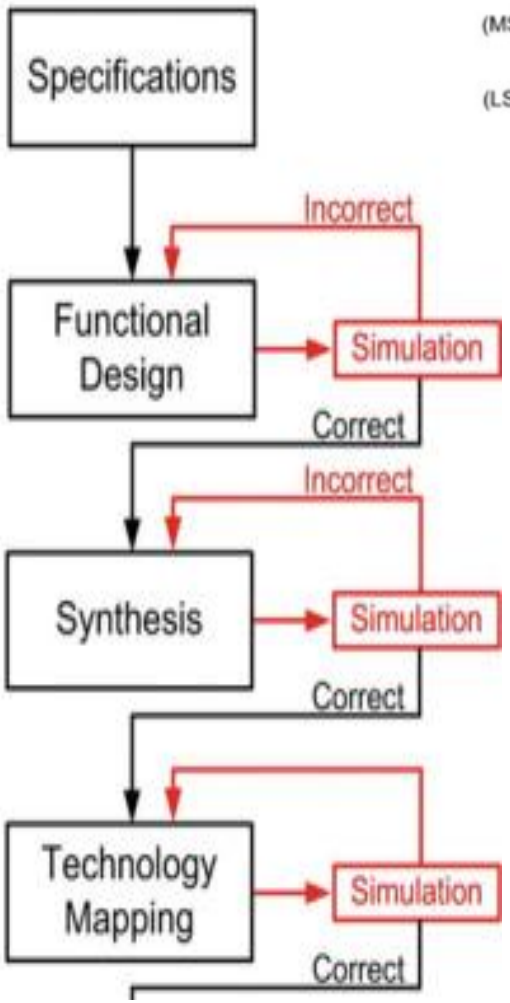
- **Part 4 – HDLs overview**
  - **HDLs Overview**
  - **Logic Synthesis**
  - **HDL Representations—Verilog**

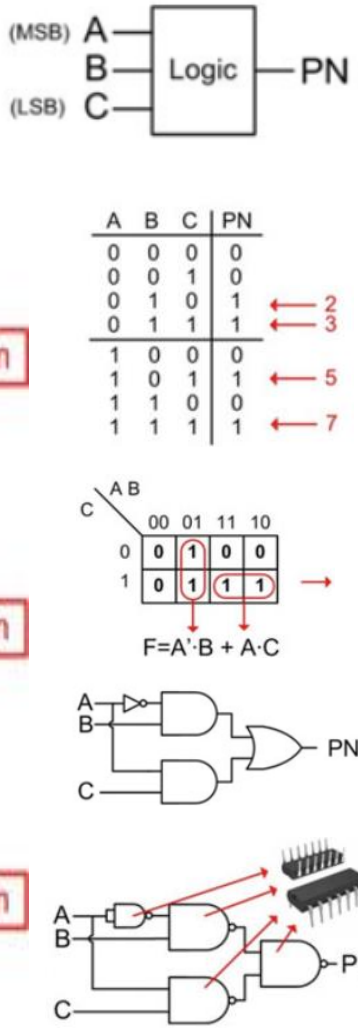
HDLs overview

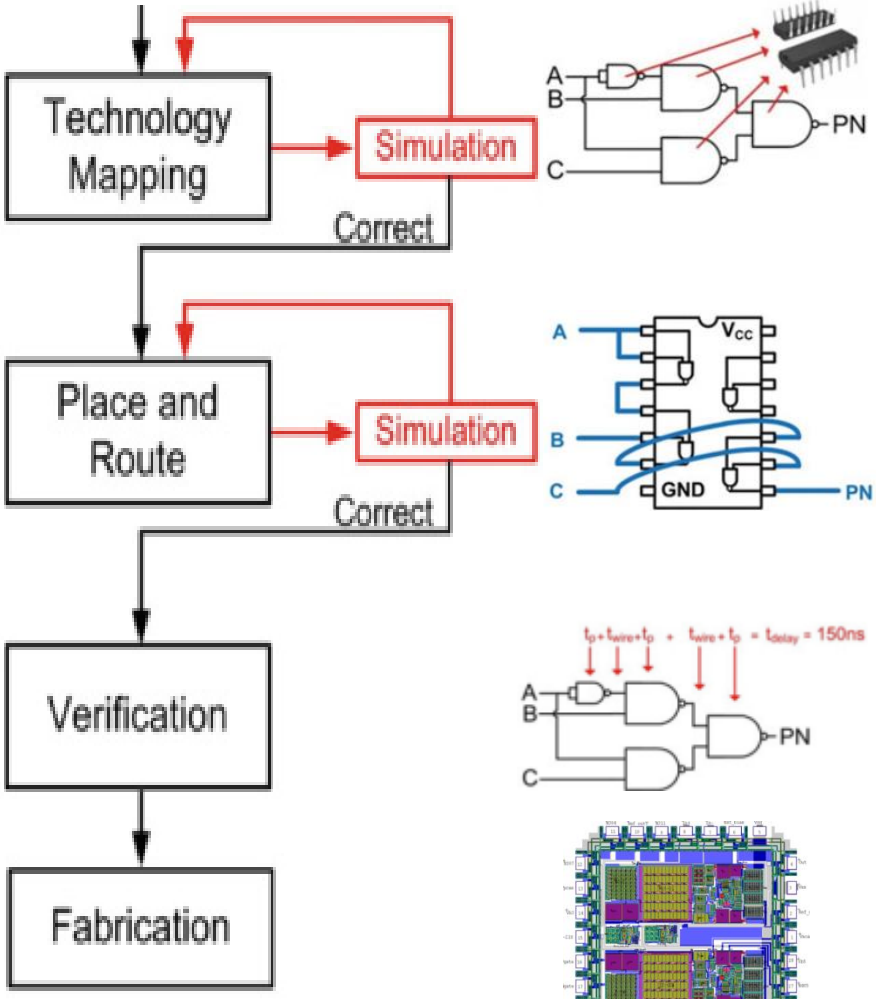
- **A hardware description language (HDL) is a specialized computer language used to describe the structure and behavior of electronic circuits.**
- **It is feasible to use CAD tools for designing complex digital systems.**



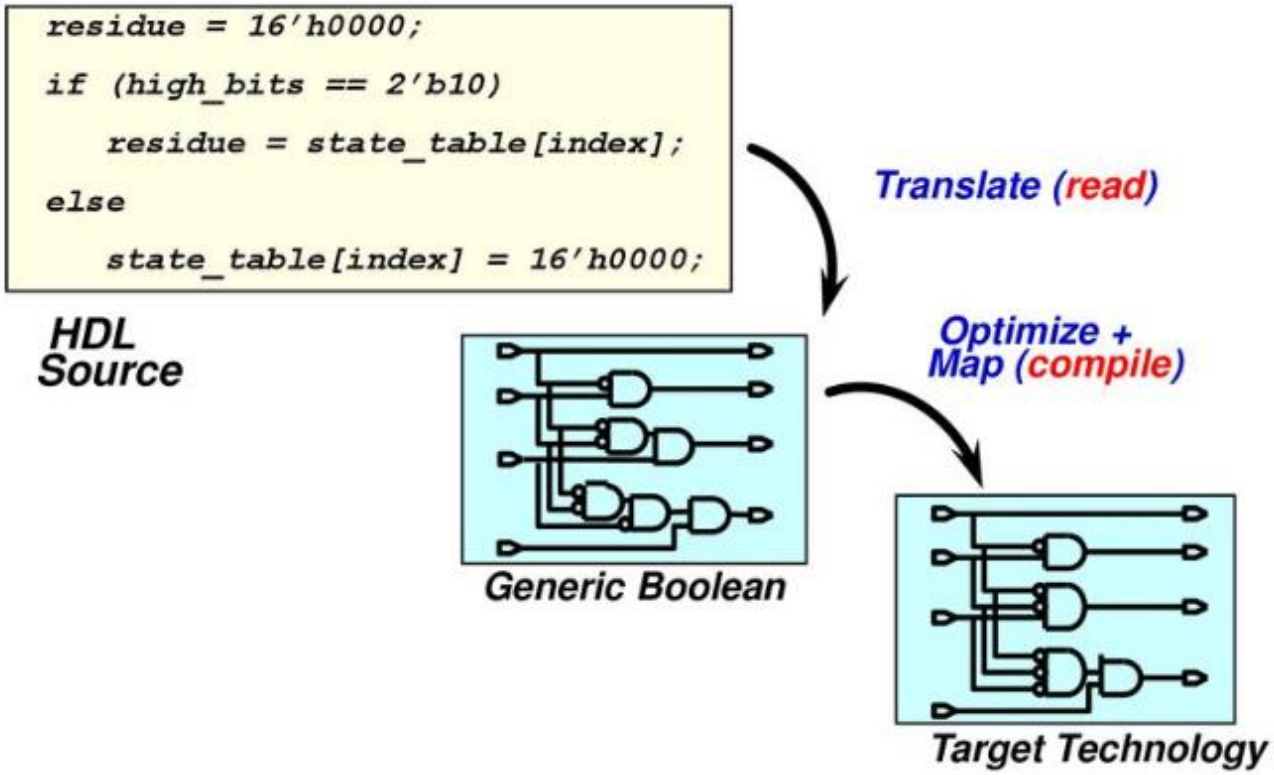
Classical Digital Design Flow





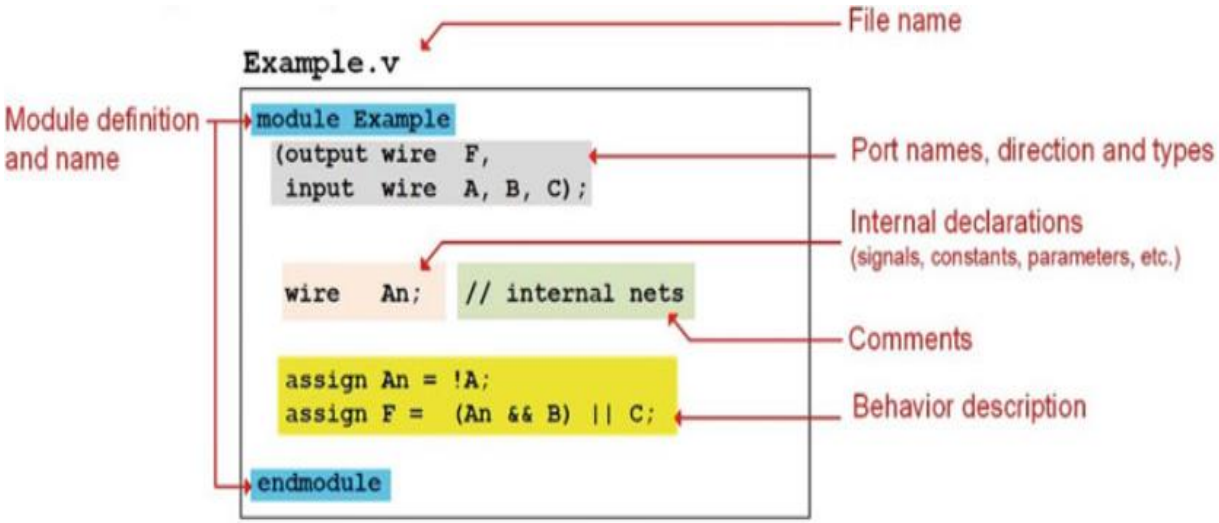


Logic Synthesis



- **Logic synthesis is a process where a high-level description of the design is converted into an optimized gate-level representation given a standard-cell library and certain design constraints (e.g., area, performance, power).**

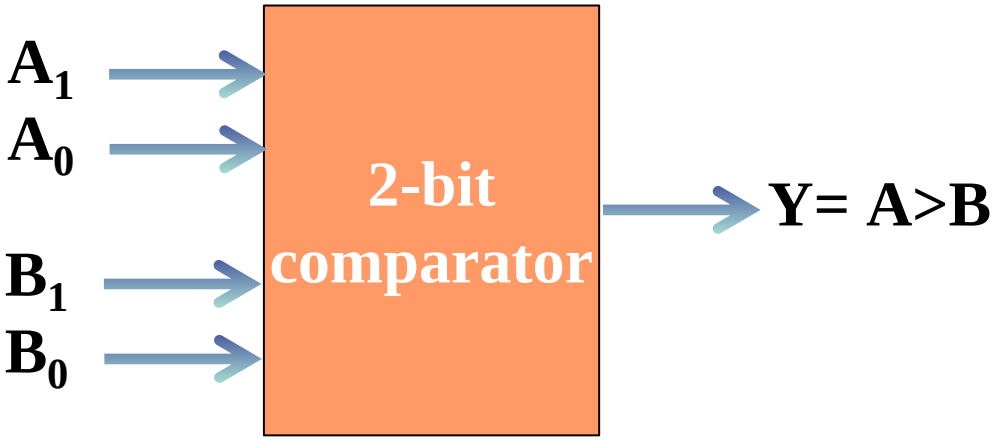
An Anatomy of a Verilog File

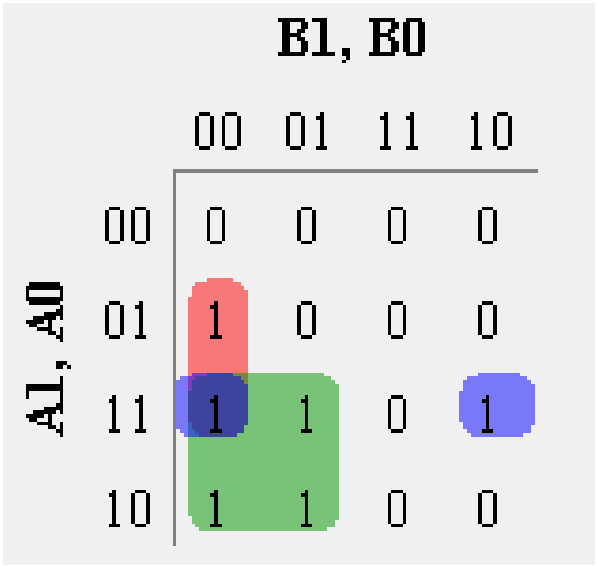


- A Verilog design consists of a **hierarchy of modules**. Modules encapsulate design hierarchy, and communicate with other modules through input, output, and bidirectional ports.
- Use **synthesizable code** to describe the function of circuit that could be built in hardware.
- **Non-synthesizable** (**delay, loop**) constructs should be used only for test benches.

Describing Hardware with Verilog

- **A two-bit greater-than comparator circuit:**
  if ($A_1A_0 > B_1B_0$) Y=1; else Y=0;



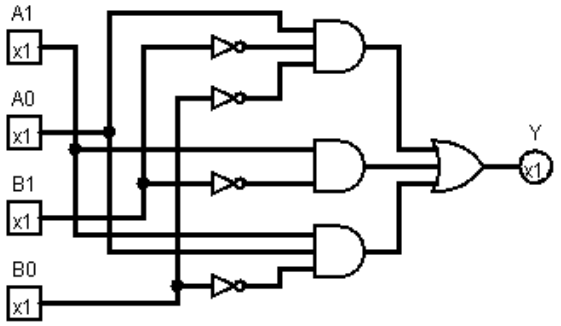


$$Y = A_1\overline{B}_1 + A_1A_0\overline{B}_0 + A_0\overline{B}_1\overline{B}_0$$

Structural Verilog Description of Two-Bit Greater-Than Circuit

$$Y = A_1\overline{B}_1 + A_1A_0\overline{B}_0 + A_0\overline{B}_1\overline{B}_0$$

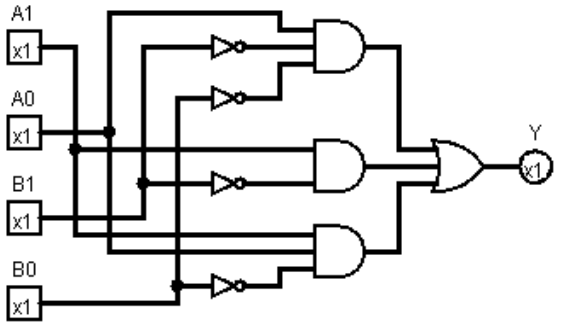
```
// Two-bit greater-than circuit: Verilog structural model                  //  1
// See Figure 2-27 for logic diagram                                       //  2
module comparator_greater_than_structural(A, B, A_greater_than_B);        //  3
 input [1:0] A, B;                                                        //  4
 output A_greater_than_B;                                                 //  5
 wire B0_n, B1_n, and0_out, and1_out, and2_out;
  not
   inv0(B0_n,  B[0]),  inv1(B1_n,  B[1]);
  and
   and0(and0_out,  A[1],  B1_n),
   and1(and1_out,  A[1],  A[0],  B0_n),
   and2(and2_out,  A[0],  B1_n,  B0_n);
  or                                                                       // 13
   or0(A_greater_than_B,  and0_out,  and1_out,  and2_out);                 // 14
endmodule                                                                  // 15
```



Dataflow Verilog Description of Two-Bit Greater-Than Comparator

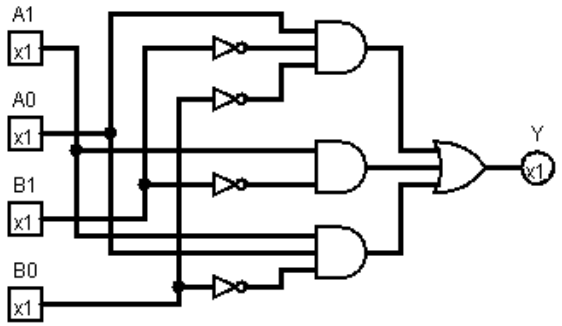
$$Y = A_1\overline{B}_1 + A_1A_0\overline{B}_0 + A_0\overline{B}_1\overline{B}_0$$

```
// Two-bit greater-than circuit: Dataflow model                    //  1
// See Figure 2-27 for logic diagram                               //  2
module comparator_greater_than_dataflow(A, B, A_greater_than_B);   //  3
 input [1:0] A, B;                                                 //  4
 output A_greater_than_B;
 wire B1_n, B0_n, and0_out, and1_out, and2_out;
 assign B1_n = ~B[1];
 assign B0_n = ~B[0];
 assign and0_out = A[1] & B1_n;
 assign and1_out = A[1] & A[0] & B0_n;
 assign and2_out = A[0] & B1_n & B0_n;                             // 11
 assign A_greater_than_B = and0_out | and1_out | and2_out;         // 12
endmodule                                                          // 13
```



Conditional Dataflow Verilog Description of Two-Bit Greater-Than Circuit

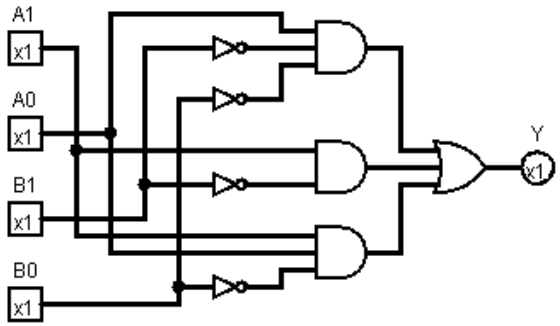
```
// Two-bit greater-than circuit: Conditional model                    // 1
// See Figure 2-27 for logic diagram                                  // 2
module comparator_greater_than_conditional2(A, B, A_greater_than_B);  // 3
 input [1:0] A, B;                                                   // 4
 output A_greater_than_B;                                            // 5
 assign A_greater_than_B = (A > B)? 1'b1 :                           // 6
      1'b0;                                                          // 7
endmodule                                                           // 8
```



$$Y = A_1\overline{B}_1 + A_1A_0\overline{B}_0 + A_0\overline{B}_1\overline{B}_0$$

Behavioral Verilog Description of Two-Bit Greater-Than Circuit

```
// Two-bit greater-than circuit: Behavioral model                    // 1
// See Figure 2-27 for logic diagram                                 // 2
module comparator_greater_than_behavioral(A, B, A_greater_than_B);   // 3
 input [1:0] A, B;                                                   // 4
 output A_greater_than_B;                                            // 5
 assign A_greater_than_B = A > B;                                    // 6
endmodule                                                            // 7
```

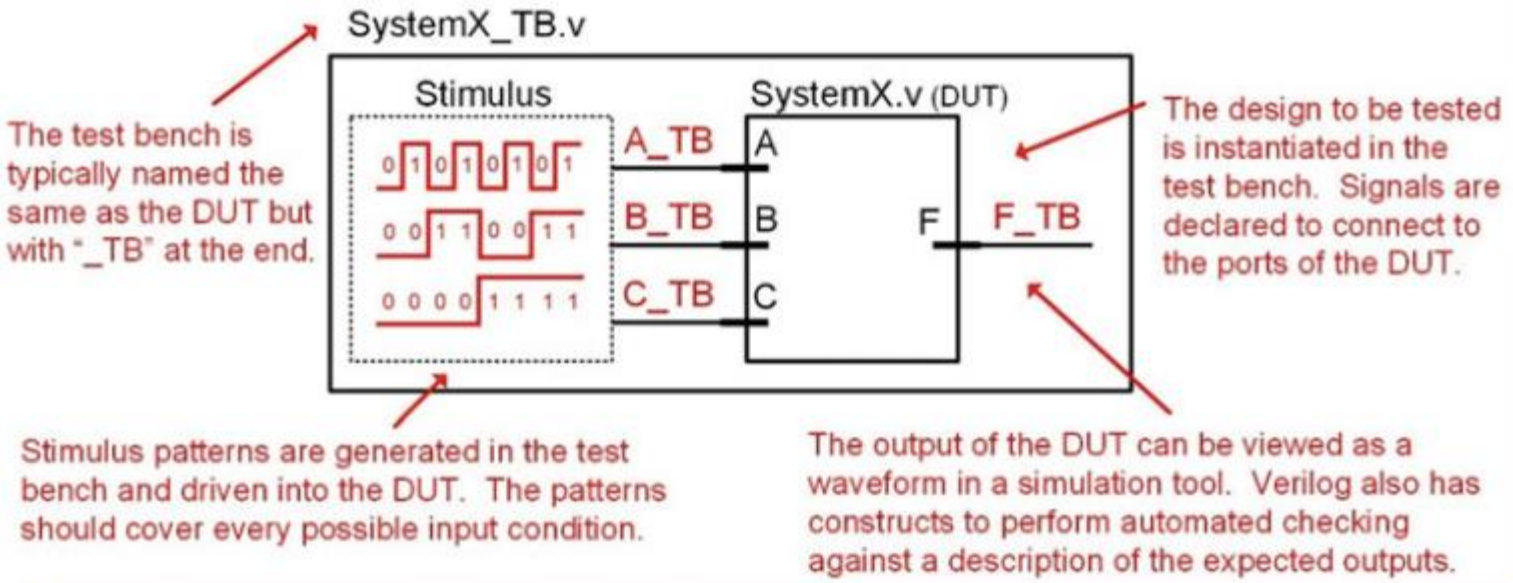


$$Y = A_1\overline{B}_1 + A_1A_0\overline{B}_0 + A_0\overline{B}_1\overline{B}_0$$

Overview of Verilog Test benches

- A test bench supplies the signals and dumps the outputs to simulate a Verilog design.
- Test bench invokes the design under test, generates the simulation input vectors, and implements the system tasks to view the results of the simulation.
- Test bench uses non-synthesizable code to check your synthesizable code.
- Test bench can use all Verilog commands.

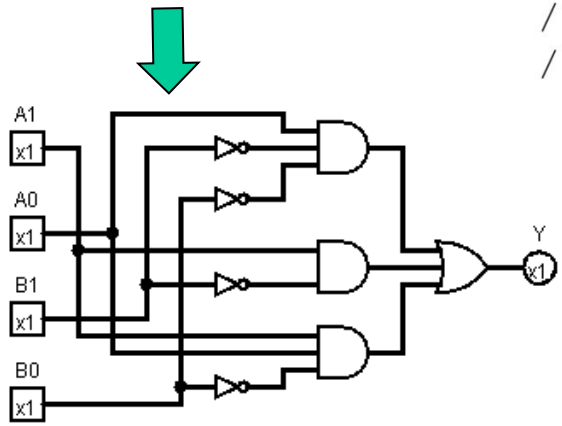
Overview of Verilog Test benches (continued)

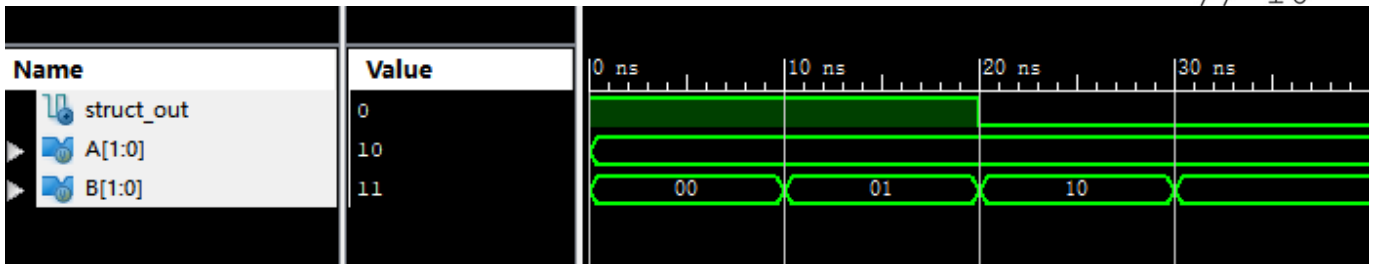



```
module SystemX_TB ();

  reg  A_TB, B_TB, C_TB;
  wire F_TB;

  SystemX DUT (F_TB, A_TB, B_TB, C_TB);

  // Stimulus Generation to Drive A_TB, B_TB and C_TB (covered in Ch. 8)

  // Automated Output Checking & Reporting for F_TB  (covered in Ch. 8)

endmodule
```

Test bench for the Structural Model of the Two-Bit Greater-Than Comparator

```
// Testbench for Verilog two-bit greater-than comparator                    //  1
module comparator_testbench_verilog();                                      //  2
 reg [1:0] A, B;                                                            //  3
 wire struct_out;                                                           //  4
 comparator_greater_than_structural U1(A, B, struct_out);                   //  5
 initial                                                                    //  6
 begin                                                                      //  7
  A = 2'b10;                                                                //  8
  B = 2'b00;                                                                //  9
  #10;                                                                      // 10
  B = 2'b01;                                                                // 11
  #10;                                                                      // 12
  B = 2'b10;                                                                // 13
  #10;                                                                      // 14
  B = 2'b11;                                                                // 15
 end                                                                        // 16
endmodule
```





Assignments

Reading:

- **2.8, 2.10**