Digital System Design with HDL (I)

Lecture 1

Dr. Ming Xu and Dr. Kain Lu Low

Dept of Electrical & Electronic Engineering

XJTLU

In This Session

- An Overview of HDLs
- Introduction of Verilog languages

Textbook:

S. Brown and Z. Vranesic, Fundamentals of Digital Logic with Verilog Design, McGraw-Hill, 2003.

Assessment:
Final exam (70%)

2 Assignments (30%)

An Overview of HDLs

- There are different methods to describe logic circuits:
  - Boolean expression
  - Schematic diagram
  - Truth table
  - Timing diagram
- Certain situations are easier to describe using one method over another.
- Recent trends are favouring text-based language description.

An Overview of HDLs

- A **H**ardware **D**escription **L**anguage (HDL) is a computer language that is used to describe hardware.
- Unlike schematic capture, HDL based designs are highly portable and independent of technology.

An Overview of HDLs

Two main HDLs are used by industry:

Verilog (**Veri**fy **Log**ic)

- C-based.
- Originally developed for simulation and verification of digital circuits.

VHDL (**V**ery High Speed Integrated Circuit **HDL**)

- Ada based.
- Originally developed by the order of DoD for documentation of digital systems.

Facts about Verilog

- We use Verilog, not VHDL.
  - used more often in electronic and computer industry
  - programming style is very similar to C programming language
- Design examples using Verilog
  - Intel Pentium, AMD K5, K6, Atheon, ARM7, etc
  - Thousands of ASIC designs using Verilog HDL
  - You will learn how to design a MIPS processor with Verilog.

Facts about Verilog

- History of Verilog
  - In 1980s, originally developed by Gateway Design Automation.
  - In 1990, was put in public domain.
  - In 1995, adopted as an IEEE standard 1364-1995
  - In 2001, an enhanced version, Verilog 2001

Use of Verilog

- Designs in Verilog
  - Design entry, like schematic
  - Synthesis
  - Simulation and verification of your design
- Verilog is a complex, sophisticated language
  - We focus on a subset of its features.
  - For complete Verilog, refer to reference books.
  - Appendix A provides a concise summary of Verilog.

Representation of Digital Circuits in Verilog

Modules can be specified in different ways

- **Structural**: use Verilog constructs that represent simple circuit elements. Write code that connects such elements together.
- **Behavioral**: use logic expressions and C-like programming constructs that define the behavior of the circuit.

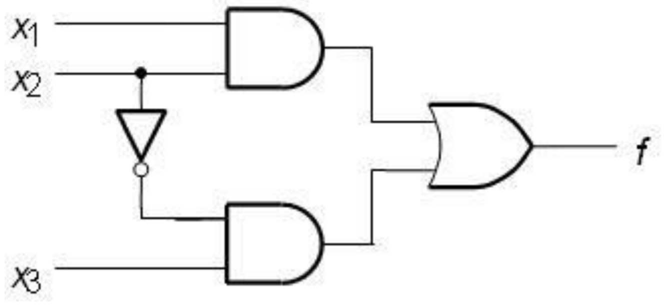
A single module can use more than one method!

Structural Representation

- Gate instantiations
  - Use gate level primitives
  - Correspond to commonly used logic gates
  - All available gate primitives are in Table A.2
- Examples (y for output and x for inputs)
  - AND gate → **and** (y, x1, x2);
  - OR gate → **or** (y, x1, x2, x3, x4);
  - NOT gate → **not** (y, x);
- Keywords: **and, or, not** are reserved

Structural Representation

- *Module*
  - A logic circuit → *module*
  - Its ports: inputs and outputs
  - Begins with **module,** ends with **endmodule**
- Example
- These gate instantiation statements are concurrent.



```
module example1 (x1, x2, x3, f);
        input x1, x2, x3;
        output f;

        and (g, x1, x2);
        not (k, x2);
        and (h, k, x3);
        or (f, g, h);

endmodule
```

Behavioral Representation

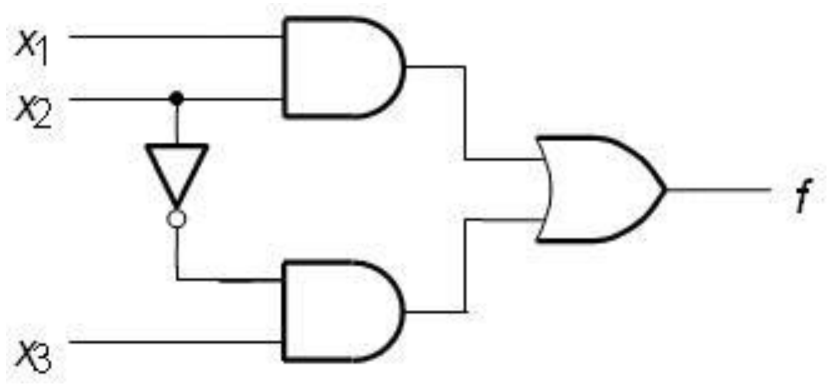
- Why behavioral representation?
  - Using primitives can be tedious when circuits are large
  - It is desired to describe the circuit in more abstract level – behavior
- Two fundamental types of behavior model
  - Logic expression
  - Procedural statements
- CAD synthesis tools use this representation to construct the actual circuit

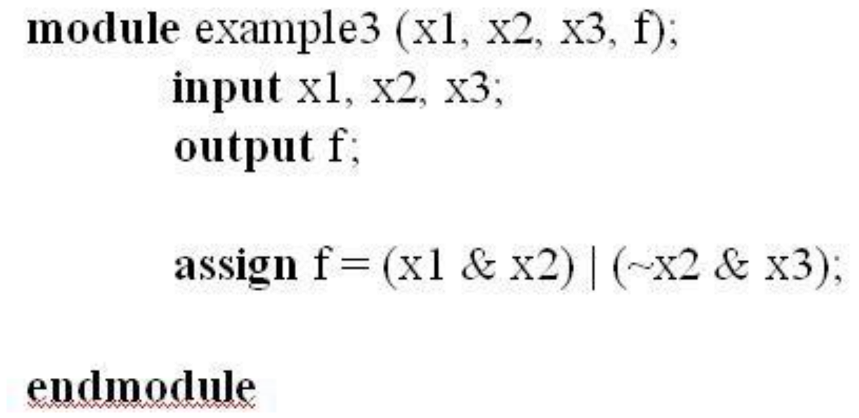
Behavioral Representation

Logic expression

continuous assignment

- Whenever any signal on the right-hand side changes its state, the value will be re-evaluated.
- Equivalent to using gate-level primitives.





```
module example3 (x1, x2, x3, f);
        input x1, x2, x3;
        output f;

        assign f = (x1 & x2) | (~x2 & x3);

endmodule
```

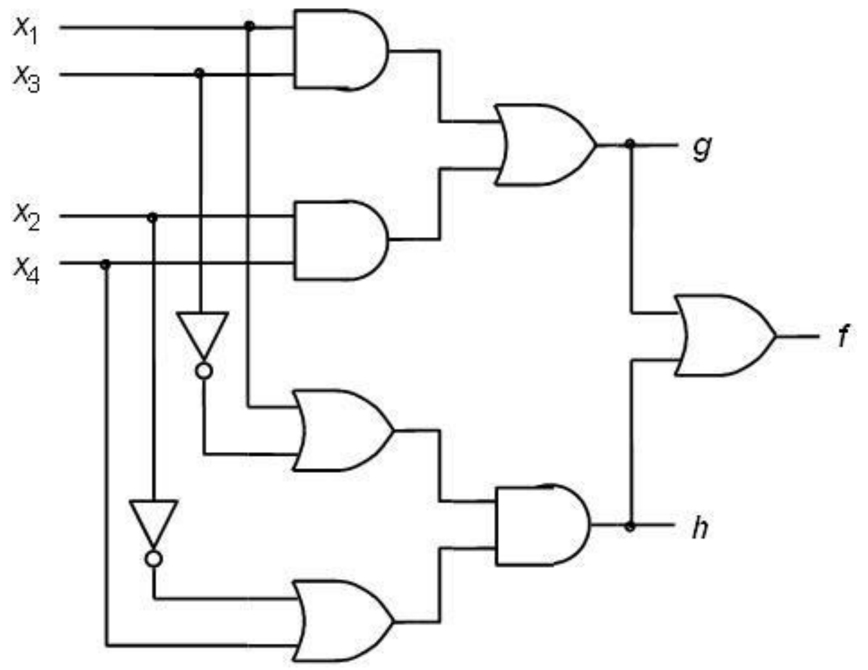
Behavioral Representation

- One more example

```
module example4 (x1, x2, x3, x4, f, g, h);
    input x1, x2, x3, x4;
    output f, g, h;

    assign g = (x1 & x3) | (x2 & x4);
    assign h = (x1 | ~x3) & (~x2 | x4);
    assign f = g | h;

endmodule
```

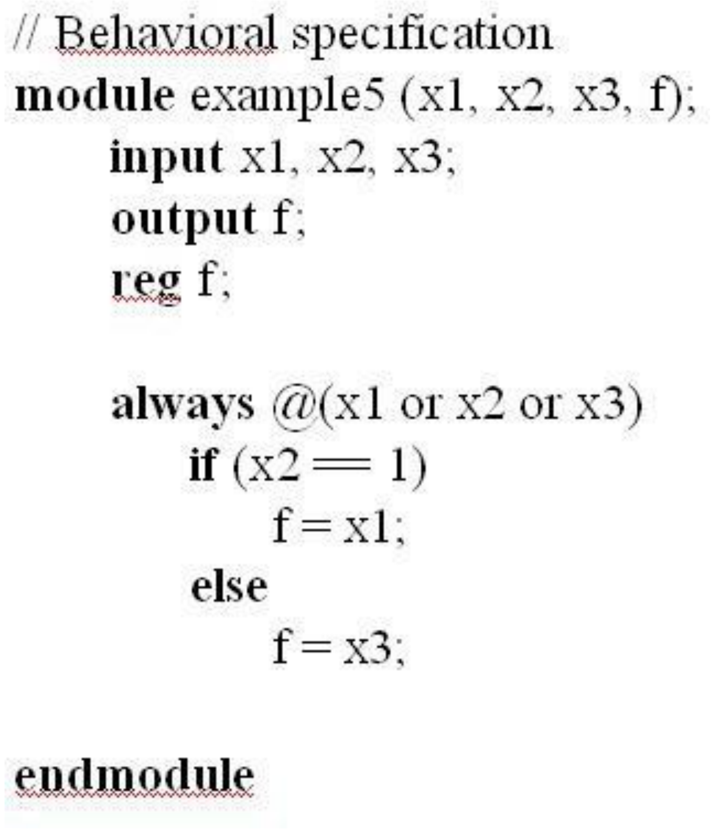


- The logic expression statements are also concurrent.

Behavioral Representation

Procedural statements are often contained in **always** blocks

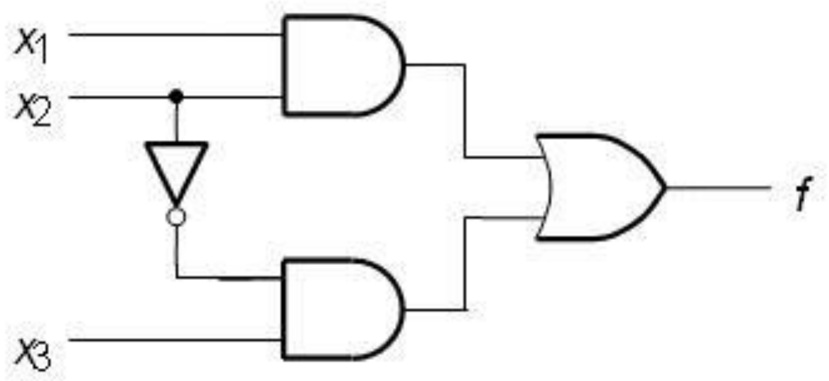
- **always** block
  - Sensitivity list: statements evaluated only when one or more signals in list changes value.
  - Statements evaluated in order



```
// Behavioral specification
module example5 (x1, x2, x3, f);
    input x1, x2, x3;
    output f;
    reg f;

    always @(x1 or x2 or x3)
        if (x2 == 1)
            f = x1;
        else
            f = x3;

endmodule
```



Concurrent vs. Procedural Statements

- Concurrent statements are considered in parallel and the ordering in the code does not matter. Examples include *gate instantiations* and *continuous assignment*.
- Procedural statements are evaluated in the order in which they appear in the code. Verilog requires that such statements be contained inside an **always** block.