# Lecture 5
# Verilog Language 1

# ASIC Design Flow

- Application-specific Integrated Circuit (ASIC) Design Flow for Large-scale Digital Circuits

**Design Language and Tools:**

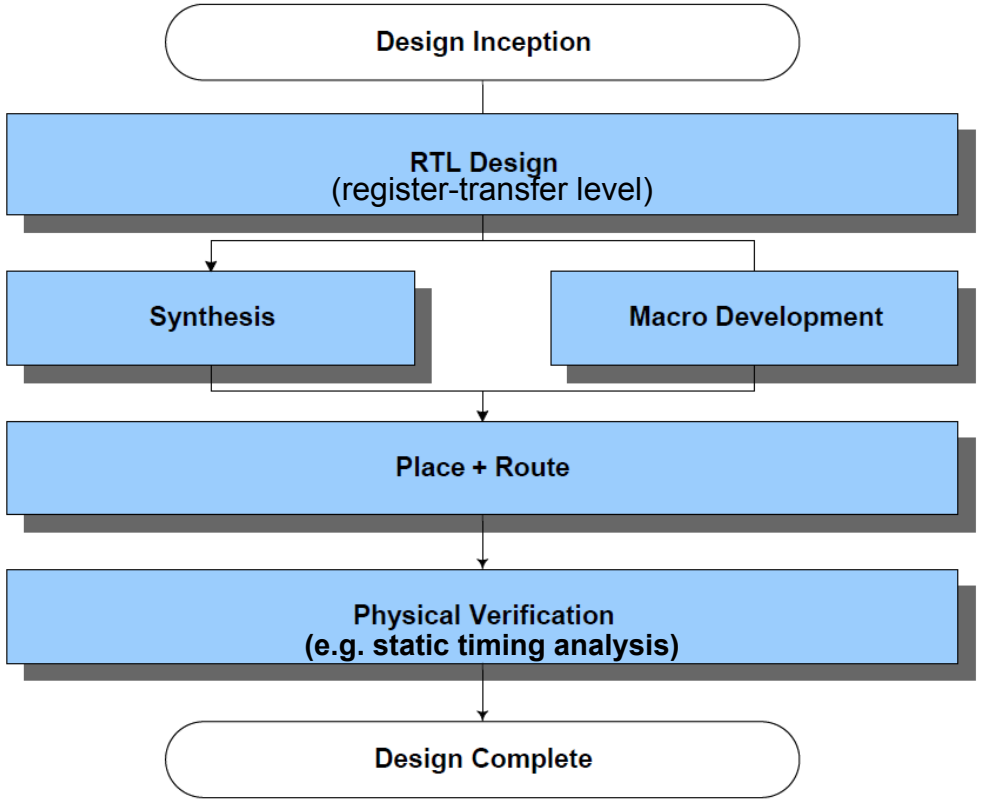
**Matlab / C**

**Verilog/VHDL (sim by Modelsim, NC-sim)**

**Design Compiler / RTL Compiler**

**Encounter**

**Encounter/ Primetime / ETS**



# Verilog and VHDL

- Hardware description language (HDL)
  - Text description
- Used to define digital logic and circuits
- Purpose:
  - Model and simulate digital system
  - Synthesize digital circuits
    - Convert from high level logic description to low level gate netlists


# Verilog and VHDL

- Two popular languages
  - Verilog
    - IEEE Standard 1364-1995/2001/2005
    - Based on the C language
  - VHDL: **V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage
    - Developed by Department of Defense (DOD) from 1983
    - IEEE Standard 1076-1987/1993/200x
    - Based on the ADA language


# Verilog and VHDL

- Verilog/VHDL has become industry standard language for hardware description
- Widely used for design:
  - General purpose CPU/GPU
  - Application Specific Integrated Circuits (ASIC)
  - FPGA
  - System level design model and verification
    - High level model
    - Mixed-signal Circuits: analog and digital co-design


# Verilog and VHDL

- Verilog vs VHDL
  - Both of them are supported by commercial design tools
  - Both of them are sufficient to complete the design and verification task
  - Verilog: simpler, easier to use, C-like
    - Slightly more popular in industry
  - VHDL: more sophisticated, precise, more data types, less error prone

# Types of Verilog Codes

- Three types of Verilog codes

Three types of codes, for different purposes

  - Behavioral:
    - Describe behavior of system
    - No indication of hardware realization
    - Not always synthesizable
    - Used to build model and testbench
  - Register-Transfer-Level (RTL):
    - Describe circuit and data operation between registers
    - Synthesizable (Can be directly converted into hardware)
    - Used to build real digital circuits
  - Gate Level:
    - Describe physical gate level implementation
    - Usually generated from RTL code

# Basic Syntax-General

- Case sensitive
- // used as comment line
- Keywords are case sensitive: e.g. module, input, output, etc.
- Start with “module”, end with “endmodule”
- Separate sentence by “;”
- Separate keywords by “,”

# Modules

- Logic is contained within modules
- Ports: declared set of inputs, outputs, and “inout”s
- Internal contents:
  - Wires (wire)
  - Registers (reg)
  - submodules

Ports, wire, reg without explicit range defaults to single bit

# Verilog HDL Models

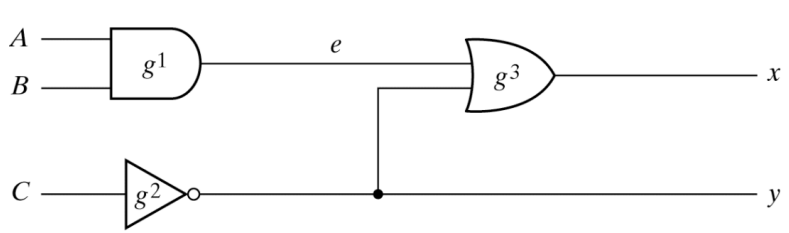
- HDL model specifies the relationship between input signals and output signals
- HDL uses special constructs to describe hardware concurrency, parallel activity flow, time delays and waveforms

Verilog code for a AND gate

```
module and_gate(y, x1, x2);
input x1, x2;
output y;
and(y, x1, x2);
endmodule
```

# Modules

```
module my_ckt(A,B,C,x,y);
    input A,B,C;          // ports
    output x,y;           // ports
    wire e;               // internal signals
    and g1(e,A,B);        // submodules
    not g2(y,C);          // submodules
    or g3(x,e,y);         // submodules
endmodule
```




Output goes first then input

# Ports

- Declare ports as “input”, “output”
- Also declare ports with number of bits (bus)
- Examples:

```
input clk;
input [15:0] data_in;
output [7:0] count;
```

# Numbers in Verilog

- Integer number can be specified in
  - Decimal;
  - Hexadecimal;
  - Octal;
  - Binary;
- If size is unspecified, stored as 32-bit binary number

# Numbers in Verilog

| Integer | Stored as |
|---|---|
| 1 | 00000000000000000000000000000001 |
| 8'hAA | 10101010 |
| 6'b100011 | 100011 |
| 'hF | 00000000000000000000000000001111 |
| 16' bZ | ZZZZZZZZZZZZZZZZ |
| 8' bx | xxxxxxxx |

Open circuit, high impedance

X - don't care

# Numbers in Verilog

- Logic values representing electrical phenomena, other than high voltage and ground voltage:
  - X, Z are not synthesizable, only used for simulation.

| Logic Value | Description |
|---|---|
| 0 | zero, low, false |
| 1 | one, high, true |
| z or Z | high impedance, floating |
| x or X | unknown, uninitialized, contention |

Contention: example - two

- Real numbers: e.g. 1.2, 0.6
  - Not synthesizable (only used for behavior simulation)

# Numbers in Verilog

- Unsigned and signed integers
  - By default, all numbers are "unsigned"
  - Use "-" sign for negative number
    - e.g. -16'h1234 (EDCC in HEX or 1110110111001100 in binary)
  - Negative number is stored in 2's complement format

# Numbers in Verilog

- Example of codes:

```
module test_number;
    reg [31:0] a;
    reg signed [31:0] b;
    wire signed [31:0] c;
    assign c=a*b;
    initial begin
        a=8'h5A;  //or 8'b01011010
        b=-1234;  // decimal format
    end
endmodule
```

Note result c will be unsigned;
(unless both a and b are signed)

# Signals

- Two types of signals:
  - wire: connection between components
    - Need to have ONLY one driver
    - Does not retain value, i.e. no memory
  - reg: registers
    - Treated as flip-flops
    - Retain values until it is updated
- Which one to use?
  - Think about circuit realization, e.g. reg is used in sequential circuits to store values
  - reg is also used in behavior simulation as internal stimulus

Immediate update

Examples: reg a;

wire signed [31:0] b; //this is a bus

# Verilog Operators

- Arithmetic Operators:

  +, -, *, /, % (modulus)

- Relational Operators:

| Operator | Description |
|---|---|
| a | a less than b |
| a>b | a greater than b |
| a<=b | a less than or equal to b |
| a>=b | a greater than or equal to b |

# Verilog Operators

- Equality Operators

| Operator | Description |
|---|---|
| a === b | a equal to b, including x and z (Case equality) |
| a !== b | a not equal to b, including x and z (Case inequality) |
| a == b | a equal to b, resulting may be unknown (logical equality) |
| a != b | a not equal to b, result may be unknown (logical equality) |

(if either operand is x or z, the result is x for == or !=)

# Verilog Operators

- Logical Operators

| Operator | Description |
|---|---|
| ! | logic negation |
| && | logical and |
| \|\| | logical or |

- Bit-wise Operators

| Operator | Description |
|---|---|
| ~ | negation |
| & | and |
| \| | inclusive or |
| ^ | exclusive or |
| ^~ or ~^ | exclusive nor (equivalence) |

(For operands with unequal length, the shorter operand is 0-filled at MSBs)

# Verilog Operators

- Shift Operators

| Operator | Description |
|---|---|
| << | left shift |
| >> | right shift |

  - Example:

```
4'b1001 >> 1;  // result is 4'b0100
4'b1001 << 1;  // result is 4'b0010
```

# Verilog Operators

- Concatenation Operators
  - Group multiple bits into a bus;
  - { }
  - Examples:

{a, b[3:0], c, 4’b1001}  // result has 24 bits if a, c are 8-bits.

# Verilog Operators

- Conditional Operators

  cond_expr ? true_expr : false_expr

Examples:

```
assign out = (sel==0) ? in0: in1;
```

(This code is synthesized into a multiplexer)

# Verilog Operators

- Operator Precedence

Order of computation if we have a nested expression

| Operator | Symbols |
|---|---|
| Unary, Multiply, Divide, Modulus | !, ~, *, /, % |
| Add, Subtract, Shift | +, – , <> |
| Relation, Equality | ,<=,>=,==,!=,===,!=== |
| Reduction | &, !&,^,^~,\|,~\| |
| Logic | &&, \|\| |
| Conditional | ? |