VERILOG

Hardware Description Language

Indrajit Chakrabarti
Dept. of Electronics & E.C.E.
IIT Kharagpur
Email : indrajit@ece.iitkgp.ernet.in

Introduction

- Verilog HDL – a tool used in digital design
- Beginning in 1984 at Gateway Design Automation
- Verilog – an industry standard due to its extensive use in design of IC chips and digital systems
- Verilog – a proprietary language supported by a simulation environment that was the first to support mixed-level design representation comprising switches, gates, RTL, and higher abstractions of digital design

Verilog vs VHDL

- VHDL
  - designed to support system-level design and specification
  - provides high-level constructs not available in Verilog (user defined types, configurations, etc.).
- Verilog
  - designed primarily for digital hardware designers developing FPGAs and ASICs
  - Best for converting data types between bit vector and arithmetic notations
  - Provides comprehensive support for low-level digital design
**Why Verilog is Popular**

- Verilog is general-purpose HDL easy to learn and use, with a syntax similar to C programming language
- It allows different levels of abstraction to be mixed in same model. A h/w model can be defined in terms of switches, gates, RTL, or behavioral code
- Popular logic synthesis tools support Verilog
- Fabrication vendors provide Verilog HDL libraries for post-logic synthesis simulation. So designing a chip in Verilog HDL provides wide choice of vendors

---

**Digital IC Design Flow**

1. **Pre-layout Simulation**
   - Design Entry
   - Logic Synthesis
   - Partitioning

2. **Post-layout Simulation**
   - Floorplanning
   - Placement
   - Routing

3. **Circuit Extraction**

---

**CAD Tools**

- Based on *Hardware description language*
- HDLs provide formats for representing the outputs of various design steps
- An HDL based CAD tool transforms from its HDL input into a HDL output which contains more hardware information
  - Behavioral level to register transfer level
  - Register transfer level to gate level
  - Gate level to transistor level
Design Representation

- A design can be represented at various levels from three different angles:
  - Behavioral
  - Structural
  - Physical

- Can be represented by Y-diagram

Y Chart

BEHAVIORAL DOMAIN
- Programs
- Specifications

STRUCTURAL DOMAIN
- Gates
- Adders
- Registers

PHYSICAL DOMAIN
- Transistors / Layouts
- Cells
- Chips / Boards

Modules in Verilog

- module is basic building block
  - can be an element or a collection of lower-level design blocks
  - elements are grouped into modules to provide common functionality used at many places in design
  - It provides necessary functionality to higher-level block through its port interface (inputs and outputs), but hides the internal implementation
  - A module can be instantiated within another module
**Syntax of Module Definition**

module <module_name> (module terminal list);

input/output declarations

local net declarations

Parallel statements

dendmodule

**Abstraction levels of Module Internals**

- **Behavioral (algorithmic) level** – in this highest level, a module is implemented in terms of design algorithm with no concern for hardware implementation details
- **Dataflow level** – designer knows how data flows between hardware registers and how data is processed
- **Gate level** – a module is implemented in terms of logic gates and interconnection between gates
- **Switch level** – in this lowest level, a module can be implemented in terms of switches, storage nodes, and interconnections between them

**A Simple AND Gate**

module simpleand (f, x, y);

input x, y;

output f;

assign f = x & y; // bitwise AND

dendmodule
A Two-Level Circuit

```verilog
module two_level (a, b, c, d, f);
  input a, b, c, d;
  output f;
  wire t1, t2;
  assign t1 = a & b; // bitwise AND
  assign t2 = ~ (c | d); /* bitwise negation and OR */
  assign f = t1 ^ t2; // bitwise XOR
endmodule
```

A Hierarchical Design – 3-bit Ripple Carry Adder

```verilog
module add3 (s, cy3, cy_in, x, y);
  input [2:0] x, y;
  input cy_in;
  output [2:0] s;
  output cy3;
  wire [1:0] cy_out;
  add B0 (cy_out[0], s[0], x[0], y[0], cy_in);
  add B1 (cy_out[1], s[1], x[1], y[1], cy_out[0]);
  add B2 (cy3, s[2], x[2], y[2], cy_out[1]);
endmodule
```

Definition of a Single-Bit Full Adder

```verilog
module add (cy_out, sum, a, b, cy_in);
  input a, b, cy_in;
  output  sum, cy_out;
  sum s1 (sum, a, b, cy_in);
  carry c1 (cy_out, a, b, cy_in);
endmodule
```

```verilog
module carry (cy_out, a, b, cy_in);
  input a, b, cy_in;
  output cy_out;
  wire t1, t2, t3;
  and g1 (t1, a, b);
  and g2 (t2, a, c);
  and g3 (t3, b, c);
  or  g4 (cy_out, t1, t2, t3);
endmodule
```
Port Connection Rules

• There are two alternate ways of specifying connectivity:
  – Positional association (ordered list)
    • The connections are listed in the same order
      add A1 (c_out, sum, a, b, c_in);
  – Explicit association (by name)
    • May be listed in any order
      add A1 (.in1(a), .in2(b), .cin(c_in),
               .sum(sum), .cout(c_out));

Data Types in Verilog

• A variable belongs to one of two data types:
  – Net
    • Must be continuously driven
    • Used to model connections between
      continuous assignments & instantiations
  – Register
    • Retains the last value assigned to it
    • Often used to represent storage elements

Net Data Type

– Nets represent connections between hardware elements
– Different ‘net’ types supported for synthesis:
  • wire, wor, wand, tri, triand, trior, trireg,
    supply0, supply1
  – ‘wire’ and ‘tri’ are equivalent; when there are
    multiple drivers driving them, the outputs of the
    drivers are shorted together.
  – ‘wor’ / ‘wand’ inserts an OR / AND gate at the
    connection.
  – ‘supply0’ / ‘supply1’ model power supply
    connections.
Example Using Net Data Type

```verilog
module using_wire (A, B, C, D, f);
    input  A, B, C, D;
    output f;
    wire  f;            // net f declared as 'wire'

    assign f = A & B;
    assign f = C | D;
endmodule
```

Example Using Net Data Type (contd)

```verilog
module using_wired_and (A, B, C, D, f);
    input  A, B, C, D;
    output f;
    wand f;           // net f declared as 'wand'

    assign f = A & B;
    assign f = C | D;
endmodule
```

Example Using Net Data Type (contd)

```verilog
module using_supply_wire (A, B, C, f);
    input  A, B, C;
    output f;
    supply0 gnd;
    supply1 vdd;
    nand G1 (t1, vdd, A, B);
    xor  G2 (t2, C, gnd);
    and  G3 (f, t1, t2);
endmodule
```
Example Using Net Data Type (contd)

module using_supply_wire (A, B, C, f);
  input A, B, C;
  output f;
  supply0 gnd;
  supply1 vdd;
  nand G1 (t1, vdd, A, B);
  xor G2 (t2, C, gnd);
  and G3 (f, t1, t2);
endmodule

Register data type

- Registers represent data storage elements
- A register (which unlike a net requires no driver) retains value until another value is placed onto it
- The ‘reg’ declaration explicitly specifies the size.
  - reg x, y; // single-bit register variables
- For ‘integer’, it takes the default size, usually 32-bits

References