Textbook

Verilog HDL

A guide to Digital Design and Synthesis

Samir Palnitkar
SunSoft Press
1996
Hardware Description Languages

- HDL
  - Popular HDLs (IEEE Standard)
    - Verilog® = Verifying Logic
    - VHDL
  - Other HDLs
    - AHPL
    - TI-HDL
    - AHDL

- Basic Idea
  - Programming Languages
HDL Requirements

- Concurrency
- Timing
- Support for Design Hierarchy
- Structural Specification
- Pragmatics
Applications

- Communication
  - Human / Computer
  - Human / Human
- Documentation
- Computer Aided Design (CAD)
  - Simulation
  - Synthesis
Why are HDLs essential?

- SSI (Small Scale Integration)
- MSI (Medium Scale Integration)
  - Paper and Pencil
- LSI (Large Scale Integration)
  - ASM (Algorithmic State Machine)
- VLSI (Very Large Scale Integration)
  - CAD (HDLs)
Verilog VS. VHDL

- VHDL
  - More general language
  - Not all constructs are synthesizable

- Verilog
  - Not as general as VHDL
  - Most constructs are synthesizable
Design Flow

1. Design specification
2. Behavioral description
3. RTL description
4. Functional verification and testing
5. Logic synthesis
6. Gate-level netlist
7. Logical verification and testing
8. Floor planning, automatic place & route
9. Physical layout
10. Layout verification
11. Implementation
Popularity of Verilog HDL

- Easy to learn, Easy to write
- Similar in Syntax to C
- Allows different levels of abstraction and mixing them
- Supported by most popular CAD tools and vendors
- PLI to customize Verilog simulators to designers’ needs
Design Methodologies

**Figure 2-1** Top-down Design Methodology

**Figure 2-2** Bottom-up Design Methodology
4-bit Ripple Carry Counter

Figure 2-3  Ripple Carry Counter
T-flipflop and the Hierarchy

<table>
<thead>
<tr>
<th>reset</th>
<th>q_n</th>
<th>q_{n+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2-4  T-flipflop

Ripple Carry Counter

- T_FF (tf0)
- T_FF (tf1)
- T_FF (tf2)
- T_FF (tf3)

D_FF
Inverter gate

Figure 2-5  Design Hierarchy
**Modules**

```
module <module_name>(<module_terminal_list>);
  ...
  <module internals>
  ...
endmodule
```

Example:
```
module T_ff(q, clock, reset);
  ...
  <functionality of T_flipflop>
  ...
endmodule
```
module ripple_carry_counter(q, clk, reset);

output [3:0] q;
input clk, reset;

//4 instances of the module TFF are created.
TFF tff0(q[0], clk, reset);
TFF tff1(q[1], q[0], reset);
TFF tff2(q[2], q[1], reset);
TFF tff3(q[3], q[2], reset);
endmodule
module TFF(q, clk, reset);
  output q;
  input clk, reset;
  wire d;

  DFF dff0(q, d, clk, reset);

  not n1(d, q); // not is a Verilog provided primitive.
endmodule
DFF (Behavioral)

// module DFF with asynchronous reset
module DFF(q, d, clk, reset);
    output q;
    input d, clk, reset;
    reg q;

    always @(posedge reset or negedge clk)
        if (reset)
            q = 1'b0;
        else
            q = d;
endmodule
Test Bench

Stimulus block

Generating inputs to CUD

Circuit Under Design (CUD)

Checking outputs of CUD

Test bench
module stimulus;
    reg clk; reg reset; wire[3:0] q;

    // instantiate the design block
    ripple_carry_counter r1(q, clk, reset);

    // Control the clk signal that drives the design block.
    initial clk = 1'b0;
    always #5 clk = ~clk;

    // Control the reset signal that drives the design block
    initial begin
        reset = 1'b1;
        #15 reset = 1'b0;
        #180 reset = 1'b1;
        #10 reset = 1'b0;
        #20 $stop;
    end

    initial // Monitor the outputs
        $monitor($time, " Output q = %d", q);
endmodule
Verilog Syntax

• White Space
  • Space \b
  • Tab \t
  • Newline \n
• Comments
  • /* Comments */
  • // Comments

• Number Specification (‘b, ‘h, ‘o, ‘d)
  • 4’b1111
  • 12’habc
  • 16’d255

  - Default length = at least 32, Default radix = decimal
4-Value Logic

- Set of values = \{0, 1, x, z\}
  - 12’h13x, 4’b1x0z
  - Z=?

- Extension
  - Filled with x if the specified MSB is x
  - Filled with z if the specified MSB is z
  - Zero-extended otherwise
    - 6’hx, 32’bz, ‘hf0

- Two’s complement
  - -6’d3 = 6’d61 (- 000011=111101)
  - -6’b01zz00 = 6’bxxxxxx

- Readability
  - 16’b0110_1011_0100_0001
## Value Set

<table>
<thead>
<tr>
<th>Value level</th>
<th>HW Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Logic zero, false</td>
</tr>
<tr>
<td>1</td>
<td>Logic one, true</td>
</tr>
<tr>
<td>x</td>
<td>Unknown</td>
</tr>
<tr>
<td>z</td>
<td>High imp., floating</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strength level</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>supply</td>
<td>Driving</td>
</tr>
<tr>
<td>strong</td>
<td>Driving</td>
</tr>
<tr>
<td>pull</td>
<td>Driving</td>
</tr>
<tr>
<td>large</td>
<td>Storage</td>
</tr>
<tr>
<td>weak</td>
<td>Driving</td>
</tr>
<tr>
<td>medium</td>
<td>Storage</td>
</tr>
<tr>
<td>small</td>
<td>Storage</td>
</tr>
<tr>
<td>highz</td>
<td>High Impedance</td>
</tr>
</tbody>
</table>

In 1  
---
Out  
---
In 2
Signal Strength

- Driving

- Storage
wire and reg

- wire
  - Used to represent connections between HW elements
  - Default Value = z

- reg
  - Retain value until next assignment
  - NOTE: this is not a hardware register or flipflop
  - Default Value = x
Vectors

- Syntax:
  - `wire/reg [msb_index : lsb_index] data_id;`

- Example
  ```
  wire a;
  wire [7:0] bus;
  wire [31:0] busA, busB, busC;
  reg clock;
  reg [0:40] virtual_addr;
  ```

- Access to parts of a vector
  `Bus[2:0]`
Other data types

- Integer, Real, Time, …

- Integer

- 32 bits
Net Data Types (wire)

- Nets represent physical connections between structural entities.
- A net value shall be determined by the values of its drivers.
- A net may have 0, 1, 2 or more drivers.
- If no driver is connected to a net, its value shall be high-impedance (z).
Example 1: a net with two drivers

module example;
    reg i1, i2, i3, i4;
    wire o;

    and g1(o, i1, i2);
    or  g2(o, i3, i4);

    initial
    begin
        i1 = 0; i2 = 0; i3 = 0; i4 = 0;
        #4 i1 = 1; i3 = 1;
        #4 i2 = 1; i4 = 1;
        #4 i1 = 0; i3 = 1;
        #4 i2 = 0; i4 = 1;
        end

endmodule
Register Data Types (reg)

- Registers are data storage elements (like variables in programming languages).

- A register shall store a value from one assignment to the next.

- The default initialization value for a reg data type shall be the unknown value, x.
module example;
    reg i1,i2,i3,i4;
    wire o;

    and g1(o,i1,i2);
    or  g2(o,i3,i4);

    initial
    begin
        i1=0; i2=0; o=0; //Illegal reference to net: o
        i3=0; i4=0;
        #4 i1=1; i3=1;
        #4 i2=1; i4=1;
        #4 i1=0; i3=1;
        #4 i2=0; i4=1;
    end
endmodule
Assignment 1

- Design a 4-bit binary adder using top-down methodology.
  - Write a behavioral description for HA units.
    - Delay (S) = 5, Delay (C) = 2
  - Use structural description for all the other parts.
  - Write a testbench for your design.
  - What is the delay of the 4-bit binary adder (use ModelSim).
Integer

- Keyword: integer
- Integer variables are signed numbers
- Bit width: implementation-dependent (at least 32-bits)
- Designer can also specify a width:
  - integer [7:0] tmp;
- Examples:

  integer counter;
  initial
  counter = -1;
Real

- Keyword: real
- Values:
  - Default value: 0
  - Decimal notation: 12.24
  - Scientific notation: 3e6 (=3 \times 10^6)
- Cannot have range declaration
- Example:
  ```
  real delta;
  initial
  begin
    delta = 4e10;
    delta = 2.13;
  end
  integer i;
  initial
    i = delta; // i gets the value 2 (rounded value of 2.13)
  ```
Time

- Used to store values of simulation time
- Keyword: time
- Bit width: implementation-dependent (at least 64)
- $\text{time}$ system function gives current simulation time
- Example:
  
  ```
  time save_sim_time;
  initial
  save_sim_time = $\text{time}$;
  ```
Arrays

- Only one-dimensional arrays supported
- Allowed for `reg`, `integer`, `time`
  - Not allowed for `real` data type
- Syntax:
  \[
  \text{<data_type> <var_name>[start_idx : end_idx];}
  \]
- Examples:
  ```
  integer count[0:7];
  reg bool[31:0];
  time chk_point[1:100];
  reg [4:0] port_id[0:7];
  integer matrix[4:0][4:0]; // illegal
  
  count[5]
  chk_point[100]
  port_id[3]
  ```
- Note the difference between vectors and arrays