Unit 5. Hardware description languages

Digital Electronic Circuits
(Circuitos Electrónicos Digitales)
E.T.S.I. Informática
Universidad de Sevilla
October, 2012

Jorge Juan <jjchico@dte.us.es> 2010, 2011, 2012
You are free to copy, distribute and communicate this work publicly and make derivative work provided you cite the source and respect the conditions of the Attribution-Share alike license from Creative Commons.
You can read the complete license at:
http://creativecommons.org/licenses/by-sa/3.0

Contents

- Hardware description languages
- Verilog example: voter
- Types of descriptions
- Verilog description structure
- Verilog tips
- Test benches and simulation
- FPGA synthesis
- Tools

Departamento de Tecnología Electrónica – Universidad de Sevilla

What is a hardware description language?

- Formal language used to describe the behavior of an (digital) electronic circuit.
- Similar to a programming language but with notable differences:
  - Most statements “execute” concurrently
  - Every statement is translated to a circuit block

```
// AND operation
x = a & b;

// OR operation
y = a | b;

// Combinational function z = xy' + x'y
z = x & ~y | ~x & y;
```


Why are HDL's useful?

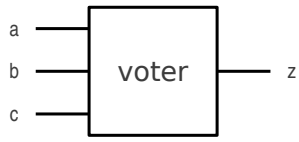
- Simulation
  - Assuring the correct operation of the circuit before implementation
- Automatic synthesis
  - Automatic circuit implementation using software tools
  - Equivalent to software's “compilation”
  - Makes digital design really simple and productive
  - Be careful! The designer should know what the tool can and cannot do.

VHDL vs Verilog

- VHDL
  - More complex syntax (ADA-like)
  - More strict syntax (reduce errors)
  - Better support for big designs
- Verilog
  - More simple syntax (C-like)
  - Default types (simpler code)
  - Multiple versions
    - 1995
    - 2001*
    - 2005

Both VHDL and Verilog are well supported by hardware vendors and can be used interchangeably. It is mostly a matter of personal taste.

Example: voter



- Logic expression
  - z = ab+ac+bc
- Verilog expression
  - z = a&b | a&c | b&c;

a b c	z
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

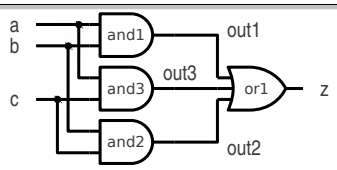
```
module voter(
    output z,
    input a,
    input b,
    input c
    );

    assign z = a&b | a&c | b&c;

endmodule
```

Types of descriptions

- Functional (continuous assignment)
  - Models combinational logic by using assignment of an expression.
- Procedural (always block)
  - Allow control structures
  - Algorithmic description similar to software
  - Easier to express complex functions
- Structural
  - Connection of circuit modules
  - Verilog includes logic gates as built-in modules

```
assign z = a&b | a&c | b&c;
```

```
always @(a, b, c)
    if (a == 1)
        if (b == 1 || c == 1)
            z = 1;
        else
            z = 0;
    else
        if (b == 1 && c == 1)
            z = 1;
        else
            z = 0;
```




```
wire out1, out2, out3;

and and1 (out1, a, b);
and and2 (out2, b, c);
and and3 (out3, a, c);
or or1 (z, out1, out2, out3);
```


Verilog description structure

- Preprocessor directives
- Module declaration
  - Module name
  - Input and output ports
- Signal declaration
  - Name and type of internal signals
- Design description
  - Processing structures
  - Expressions
  - ...
- Any number of modules can be described in a single file

```
`timescale 1ns / 1ps

// Module: voter
// Description: voting circuit
//   z = ab + bc + ac

module voter(
    input a,
    input b,
    input c,
    output z
    );

    wire z;

    assign z = a&b | b&c | a&c;

endmodule    // voter
```

Verilog tips: ports and signals

- An internal signal is automatically created for every input/output port (with the same name)
- Signal type can be declared with the port list or in the body of the module. By default, signals are of type "wire".
- Basic types of signal
  - wire: used for module connection and with "assign".
  - reg: variable type. Used in procedures

```
module voter(
    input wire a,
    input wire b,
    input wire c,
    output reg z
    );

    always @(a, b, c)
    if (a == 1)
        if (b == 1 || c == 1)
            z = 1;
        else
            z = 0;
    else
        if (b == 1 && c == 1)
            z = 1;
        else
            z = 0;

endmodule    // voter
```


Verilog Syntax

Verilog HDL Quick Reference Guide
by Stuart Sutherland

http://sutherland-hdl.com/online_verilog_ref_guide/verilog_2001_ref_guide.pdf

Verilog tips: procedural blocks

- initial
  - Executed only once
  - Only useful for test bench
- always
  - Executes constantly
  - @(...) sensitivity list: block only evaluates when any signal in the list changes

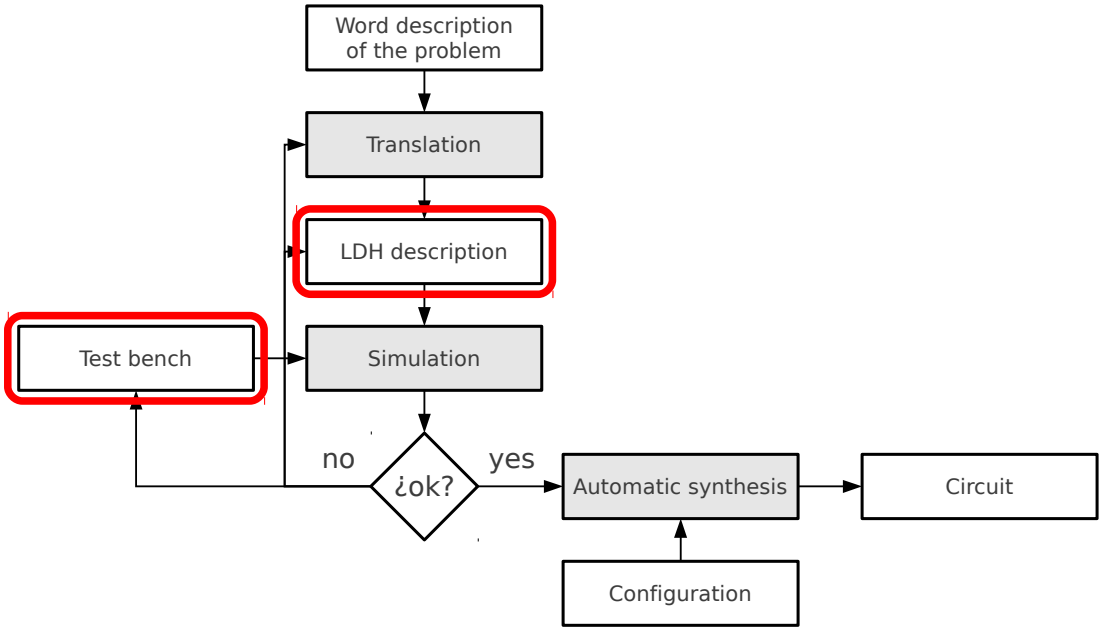
Verilog tips: concurrency

- All these executes concurrently (order does not matter)
  - Continuous assignments
  - Procedural blocks
  - Module's instances

Test bench and simulation

- A test bench is a module that contains:
  - A circuit to be simulated: Unit Under Test (UUT)
  - Verilog statements that generate the input signals to the UUT
  - Verilog simulator directives that produce simulation results.
- Test bench characteristics
  - A TB is not intended to be implemented, only to be simulated.
  - A TB module has no inputs or outputs

Design process using CAD tools



FPGA synthesis

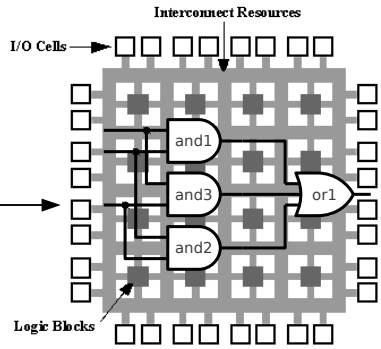
- FPGA: collection of logic devices and a programmable interconnection structure
- FPGA synthesis
  - HDL code is analyzed and HDL structures are mapped to actual logic devices (logic synthesis)
  - Logic devices on the FPGA are selected (placement)
  - The interconnections are programmed (routing)
- Restrictions
  - Only a subset of HDL constructions can be synthesized
  - Every vendor may have its own restrictions

RULE
If the designer cannot imagine what the circuit will look like, the synthesis tool cannot do it either

FPGA synthesis

```
module voter(
    input wire a, b, c,
    output reg z);
    always @(a, b, c)
    if (a == 1)
        if (b == 1 || c == 1)
            z = 1;
        else
            z = 0;
    else
        if (b == 1 && c == 1)
            z = 1;
        else
            z = 0;
endmodule
```

400K equivalent gates



Automatic synthesis → conf. file →

http://commons.wikimedia.org/wiki/File:Fpga_xilinx_spartan.jpg
http://commons.wikimedia.org/wiki/File:Fpga1a.gif

Tools

- Text editor
  - Verilog code writing
- Verilog compiler
  - Code analysis
- Simulator
  - Test bench simulation
- Synthesis tools
  - Circuit implementation on a given technology
  - Depend on the technology provider
  - FPGA
- Integrated environment
  - Includes everything above
  - Normally provided by the technology vendor

Icarus Verilog

- Icarus
  - Small and simple Verilog compiler and simulator
- Gtkwave
  - Waveform viewer: to plot simulation results
- Icarus + gtkwave: basic Verilog development environment.
  - Small
  - Easy to use
  - Free software

http://www.icarus.com/eda/verilog/

Icarus Verilog in GNU/Linux (highly recommended!)

- Icarus and gtkwave are available in most GNU/Linux distributions
- Installation in Debian/Ubuntu:
  - Install packages "iverilog" and "gtkwaves"
- Text editor
  - Any text editor should work
  - E.g. Gedit: Verilog syntax highlighting.

Icarus Verilog in MS-Windows(TM) (not recommended)

- Look for iverilog + gtkwave installer at www.bleyer.org
- Important note!
  - Install the software in a path without spaces
  - E.g. “C:\programs\verilog”.
- Text editor
  - Use a good text editor (not Notepad)
  - E.g. Notepad++ (notepad-plus-plus.org)

Xilinx's ISE

- Design and implementation for Xilinx's FPGA's
- Integrated environment including project management, code editing, simulation, synthesis and much much more.
- Complete but complex
- Heavy to download (~4GiB) and to install (~9GiB)
- Versions for MS-Windows(TM) and GNU/Linux

Summary. HDL's

- Description of the behavior of digital circuits at different levels of abstraction
- Simulation of the design before implementation
- Automatic synthesis of the design

Bibliography

- Verilog Tutorial
  - http://www.asic-world.com/verilog/veritut.html
- Verilog HDL Quick Reference Guide (Verilog-2001 standard)
  - http://sutherland-hdl.com/online_verilog_ref_guide/verilog_2001_ref_guide.pdf
- Curso Verilog (in Spanish)
  - http://www.dte.us.es/~jjchico/curso_verilog/