Unit 3.5. Hardware description languages

Digital Electronic Circuits
E.T.S.I. Informática
Universidad de Sevilla

Jorge Juan-Chico <jjchico@dte.us.es> 2010-2020

Contenidos

- Lenguajes de descripción de hardware
- Tipos de descripciones
- Estructura de una descripción Verilog
- Verilog: sintaxis y estructuras principales
- Banco de pruebas y simulación
- Síntesis desde LDH en FPGA
- Herramientas de diseño básicas

Bibliography

- Recommended:
  - LaMeres book:
    - 5.1, 5.2, 5.3: introduction to hardware description languages and to the modern digital design flow.
  - Verilog course based on examples [web]
    - Unit 1: introduction.
    - Unit 2: test benches.
    - Unit 3: combinational circuits
- Verilog reference (use when coding in Verilog)
  - Verilog HDL Quick Reference Guide (Verilog-2001 standard)
  - LaMeres 5.4 to 5.7: Verilog language, functional and structural descriptions and logic primitives.

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?

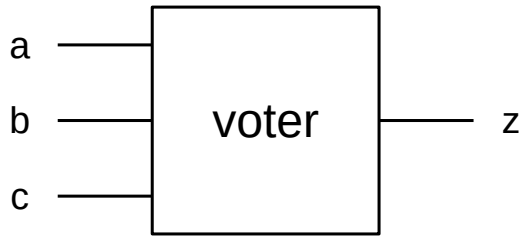
- Simulación
  - Using a circuit description, a software tool (simulator) can check how the circuit will operate before building the real circuit.
- Automatic synthesis
  - Automatic circuit implementation using software tools
  - Equivalent to software's “compilation”
  - Makes digital design really 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)
  - Easier to learn (I think...)
  - Language versions
    - Verilog
      - 1995, 2001, 2005
    - System Verilog
      - 2005, 2009, 2012, 2017, ...

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

Example: voter circuit



- 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

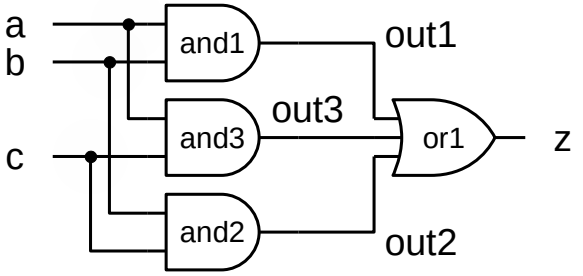
```
module voter(
    output z,
    input a,
    input b,
    input c
    );

    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
  - Equivalent to a circuit schematic
  - 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

- Pre-processor 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
- Comments
  - //, /* … */

```
`timescale 1ns / 1ps

// Module: cvoter
// Conditional voting circuit
//   z = ab + bc + ac if x=1
//   if no, z = 0;

module cvoter(
    input wire x,
    input wire a,
    input wire b,
    input wire c,
    output reg z
    );

    wire v;

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

    always @(*)
        if (x == 1)
            z = v;
        else
            z = 0;

endmodule    // cvoter
```

Verilog syntax

Your need a language reference if you are coding in Verilog. This is a good one.

Verilog HDL Quick Reference Guide

by Stuart Sutherland

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

Verilog: ports and signals

- Basic signal types
  - wire: a permanent connection. Used for modules connections and “assign”.
  - reg: a variable that can be assigned multiple times. Used in procedures like “always”.
- 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".

```
`timescale 1ns / 1ps

// Module: cvoter
// Conditional voting circuit
//   z = ab + bc + ac if x=1
//   if no, z = 0;

module cvoter(
    input wire x,
    input wire a,
    input wire b,
    input wire c,
    output reg z
    );

    wire v;

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

    always @(*)
        if (x == 1)
            z = v;
        else
            z = 0;

endmodule    // cvoter
```

Verilog: continuous assignments

- Circuit behavior description by using an expression.
- Direct way to describe a logic function if the logic expression is already known.
- Many operators can be used in expressions: logic, arithmetic, etc.
- Continuous assignments can only assign “wire” type signals.

```
...
wire v;
wire z;

assign v = a&b | b&c | a&c;
assign z = (x == 1) ? v : 0;
...
```

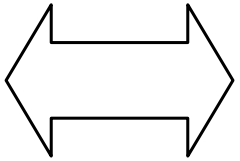
Verilog: procedural blocks

- Circuit behavior description by using control structures: *if*, *case*, *for*, etc.
- Similar to software, but the circuit doesn't “execute” a procedure at all.
- Only variables (type reg) can be assigned in a procedure.
- Main types of procedures in Verilog:
  - initial
    - Dvaluated only once during simulation.
    - Mainly useful in test benches (we'll see later).
  - always
    - Constantly evaluated during simulation.
    - A “sensitivity list” may control when the procedure's evaluation starts.

Verilog: concurrency

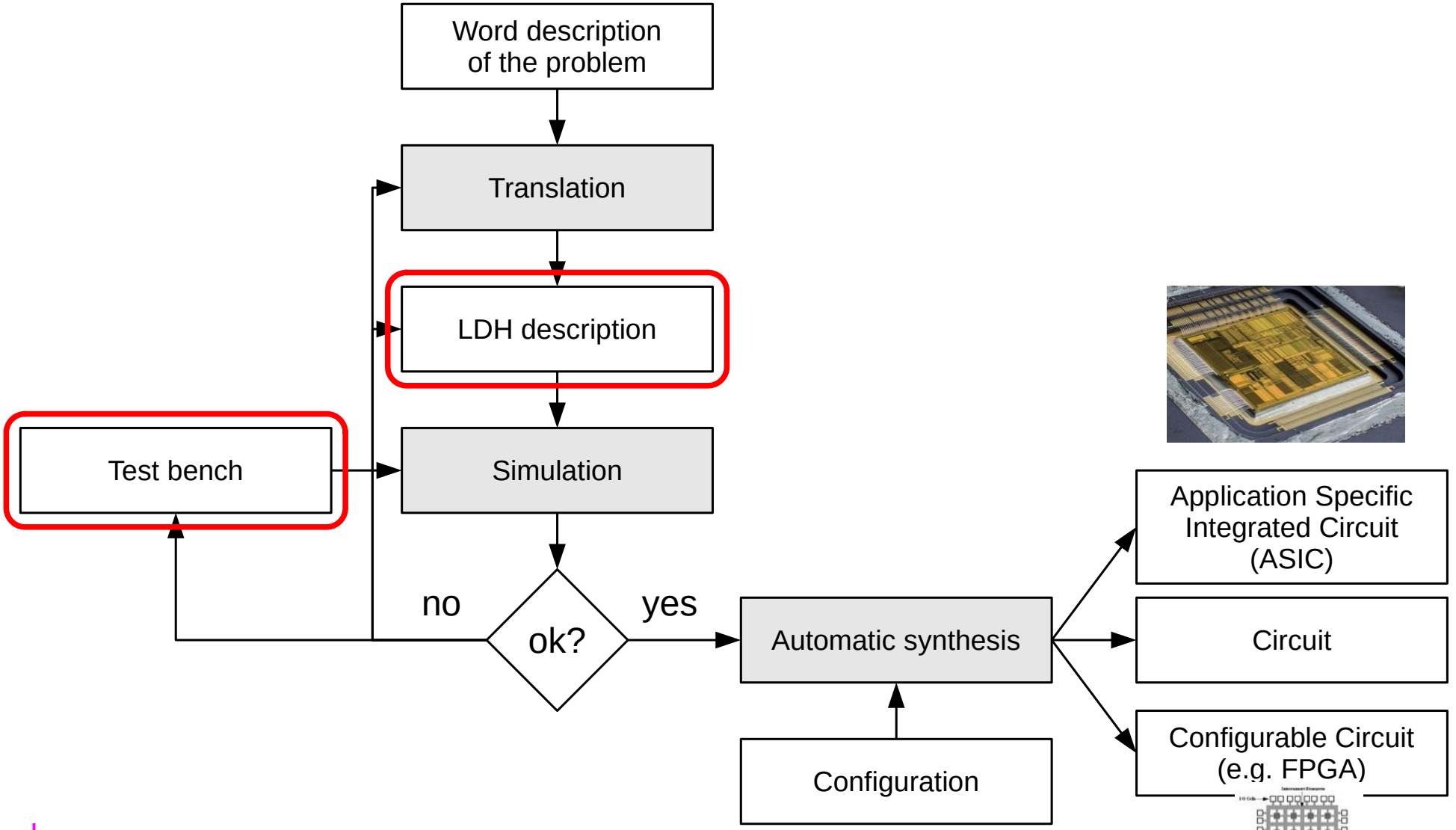
- All these structures are evaluated (during simulation) concurrently: the order in which they are placed in the Verilog source file does not matter.
  - Continuous assignments.
  - Procedural blocks.
  - Module's instances.

```
...
assign v = a&b | b&c | a&c;

always @(*)
    if (x == 1)
        z = v;
    else
        z = 0;
...
```



```
...
always @(*)
    if (x == 1)
        z = v;
    else
        z = 0;

assign v = a&b | b&c | a&c;
...
```

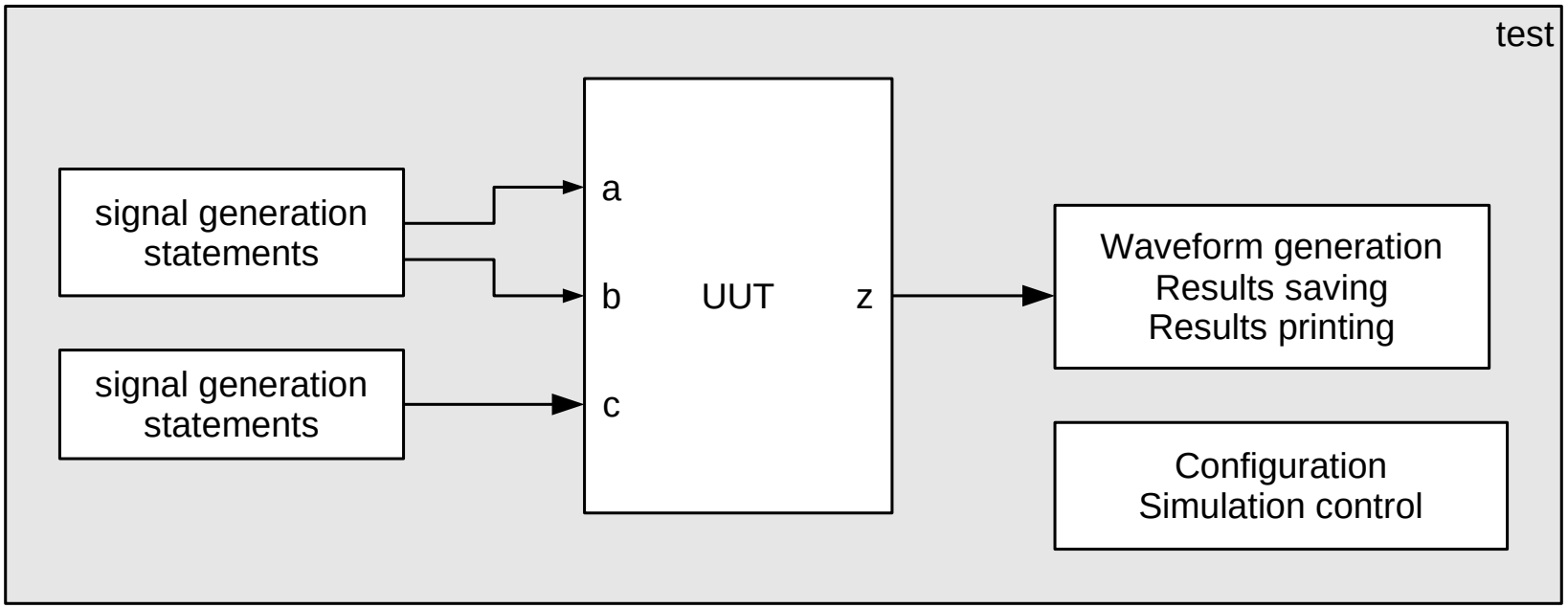
Design process using Computer-Aided Design (CAD) tools



Test bench and simulation

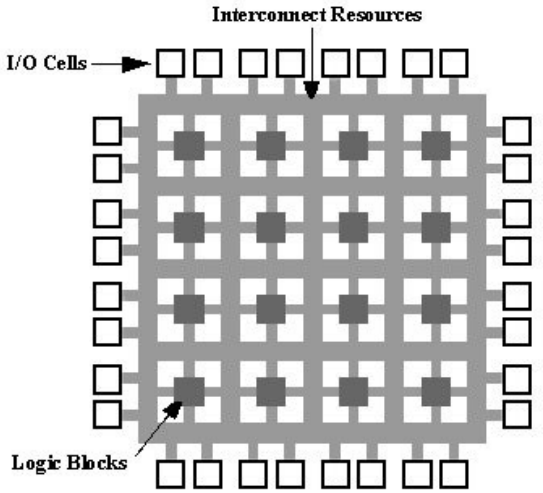
- A test bench is a module that contains:
  - A circuit to be simulated: Unit Under Test (UUT)
  - Verilog statements that generate the input signals to the UUT
  - Verilog simulator directives that control the simulation: time resolution, result printing, simulation ending, etc.
- Test bench characteristics
  - A TB is not intended to be implemented, only to be simulated.
  - A TB includes Verilog structures than are only useful for simulation (are not part of the circuit description).
  - A TB module has no inputs or outputs.

Test bench and simulation



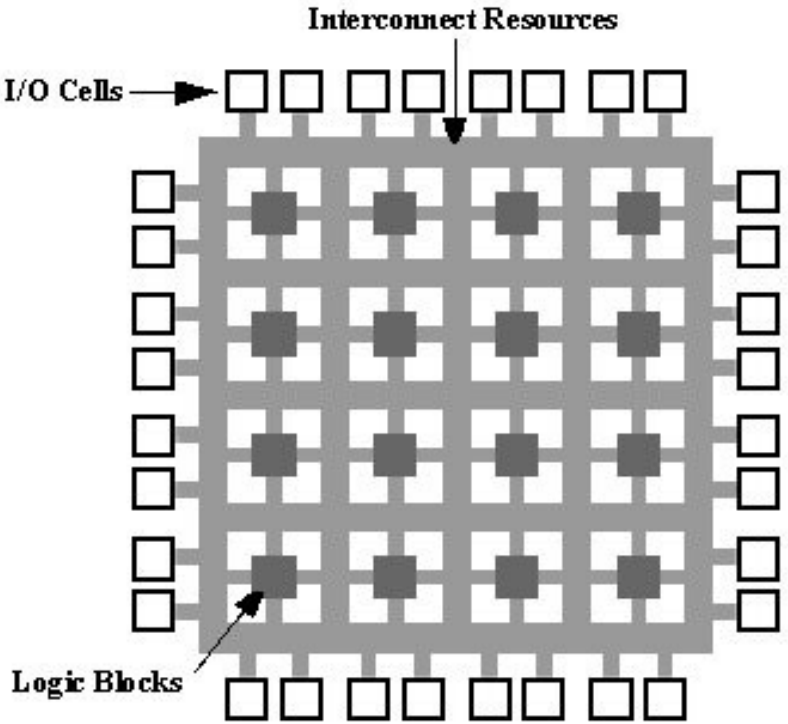
FPGA

- FPGA
  - Field Programmable Gate Array
  - Array of configurable logic devices and connections.
- Configurable Logic Block (CLB)
  - Can be configured to make any logic operation: AND, OR, XOR, etc.
- Input-Output Block (IOB)
  - Can be configured as input or output terminals and connected to internal signals.
- Interconnections
  - Can be configured to interconnect CLB's and IOB's at will.



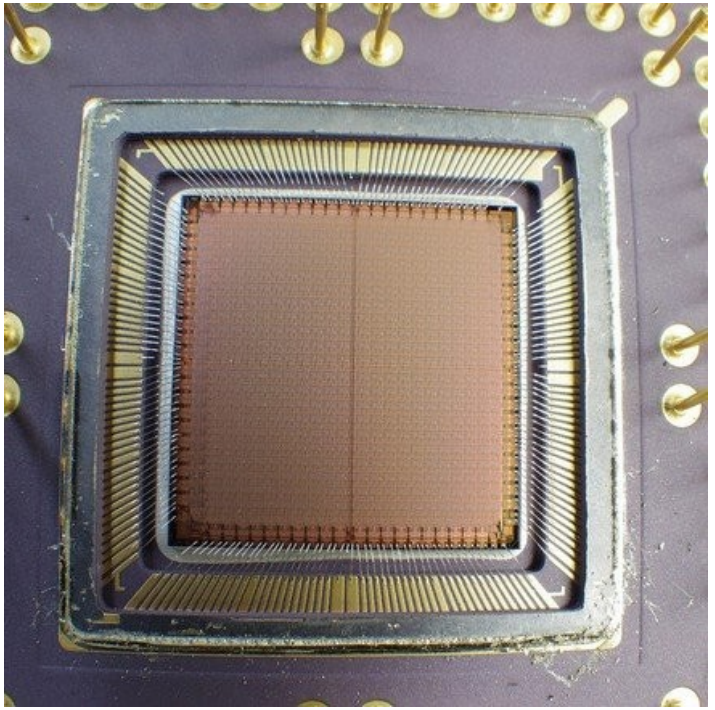
https://en.wikipedia.org/wiki/Field_Programmable_Gate_Array

FPGA



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

Xilinx XC4010-6



Xilinx XC4010-6. Dominio Público
https://www.flickr.com/photos/34923408@N07/22054207552/

FPGA Synthesis

- LDH synthesis over FPGA
  - HDL code is analyzed and HDL structures are mapped to actual logic devices (mapping).
  - CLB's are selected (placement) and configured to implement the functionality of necessary logic devices.
  - Interconnections between CLB's and IOB's are configured (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

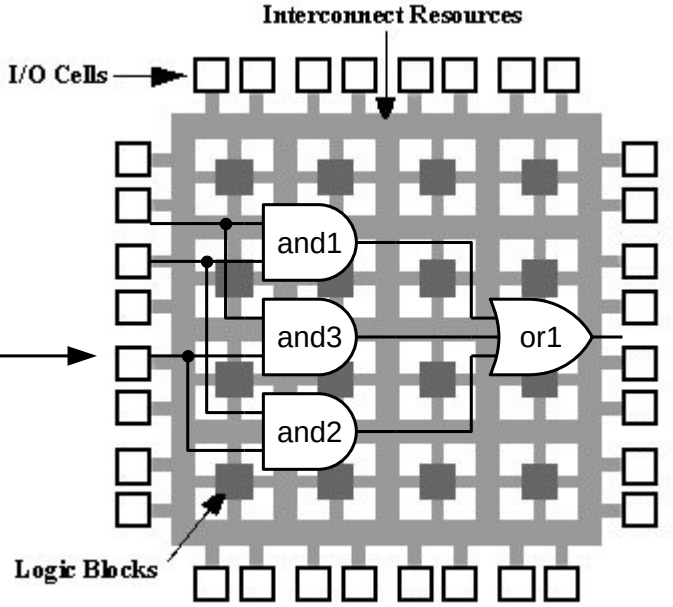
FPGA Synthesis

```
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

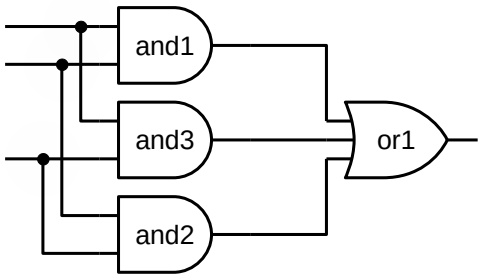
Automatic synthesis

Conf. file



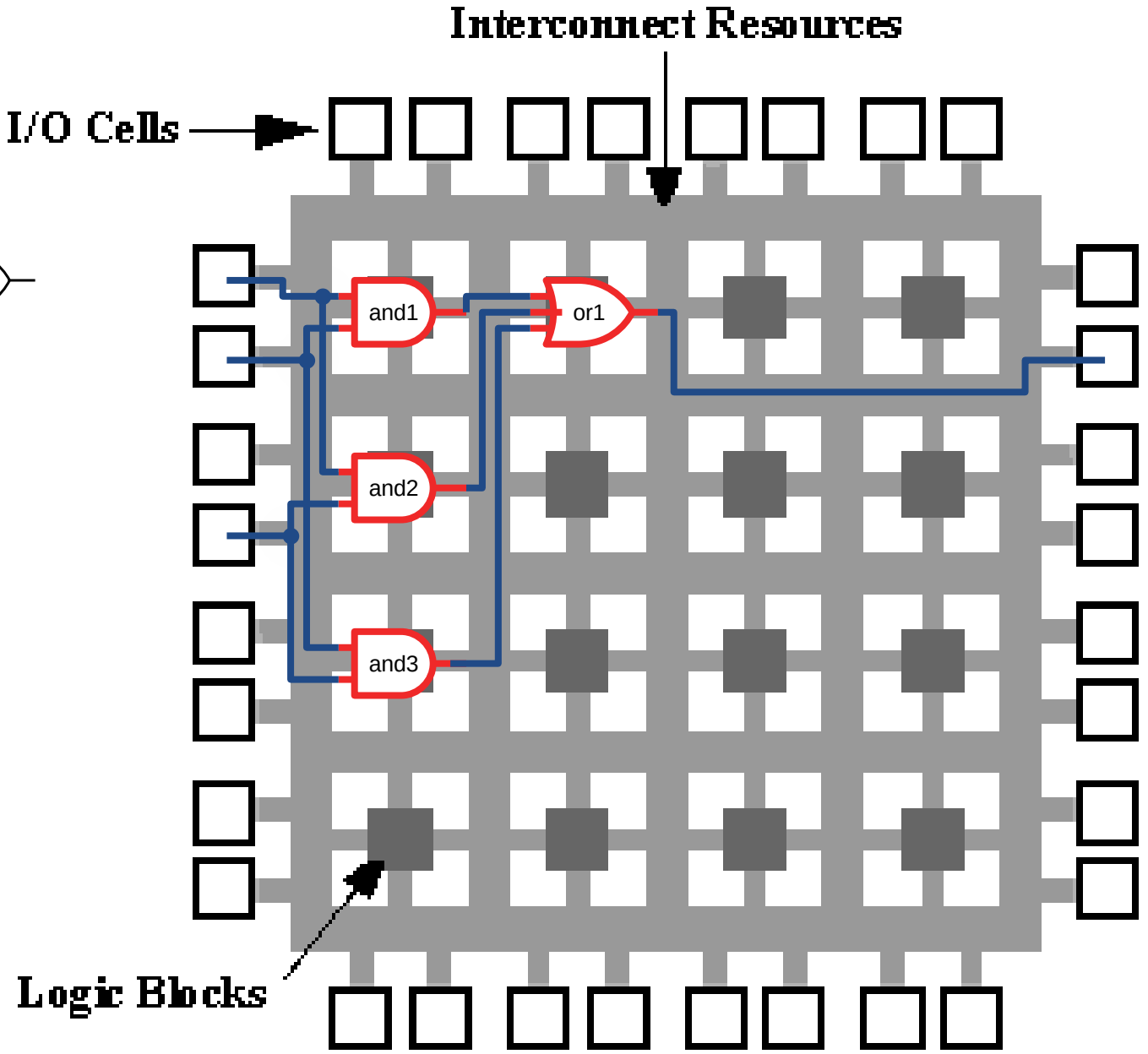
http://commons.wikimedia.org/wiki/File:Fpga_xilinx_spartan.jpg
http://commons.wikimedia.org/wiki/File:Fpga1a.gif

FPGA Synthesis



Placement

Routing



Design CAD Tools

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

Icarus Verilog + Gtkwave

- Icarus Verilog
  - Small and simple Verilog compiler and simulator.
  - Command line interface.
- Gtkwave
  - Waveform viewer: to plot simulation results.
- Text editor + Icarus + gtkwave
  - Basic Verilog development environment.
  - Very light
    - Icarus (6.5MB) + Gtkwave (4.7MB) = 11.2MB (Ubuntu 20.04)
  - Easy to use
  - Free software

Icarus Verilog in GNU/Linux (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.
    - Standard editor in Ubuntu
    - Verilog syntax highlighting.
- Ubuntu installation using a terminal:

```
$ sudo apt-get install iverilog gtkwave
...
```

Icarus Verilog + Gtkwave in MS-Windows(TM)

- Look for iverilog + gtkwave installer at www.bleyer.org
  - Install the software in a path without spaces
  - E.g. “c:\iverilog”.
- Check: open a command line (cmd, powershell, etc.)
  - Try executing “iverilog”.
  - If the command is not found, add the installation paths to Windows configuration:
    - c:\iverilog\bin
    - c:\iverilog\gtkwave\bin
- Text editor: use a good one. E.g.:
  - Notepad++
  - Atom
  - VSCodium

Iverilog Windows installation

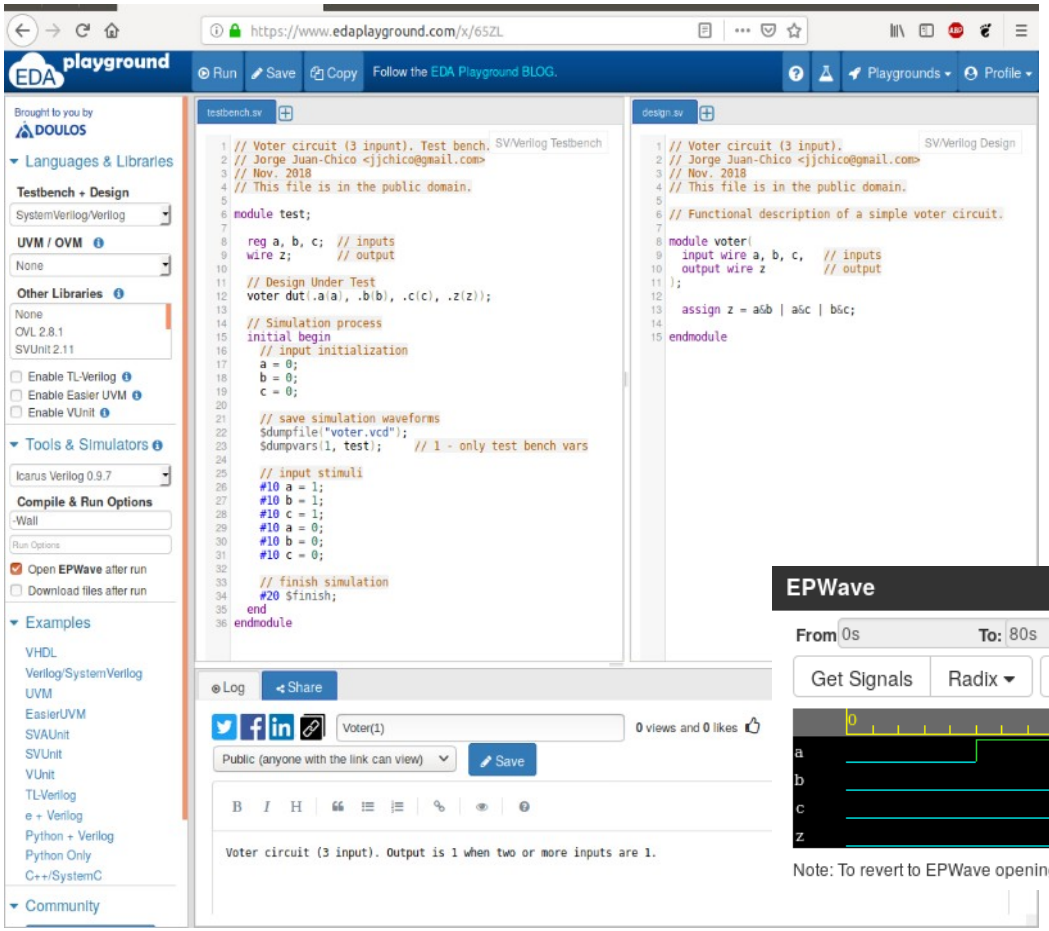
Icarus Verilog + Gtkwave typical session (Linux)

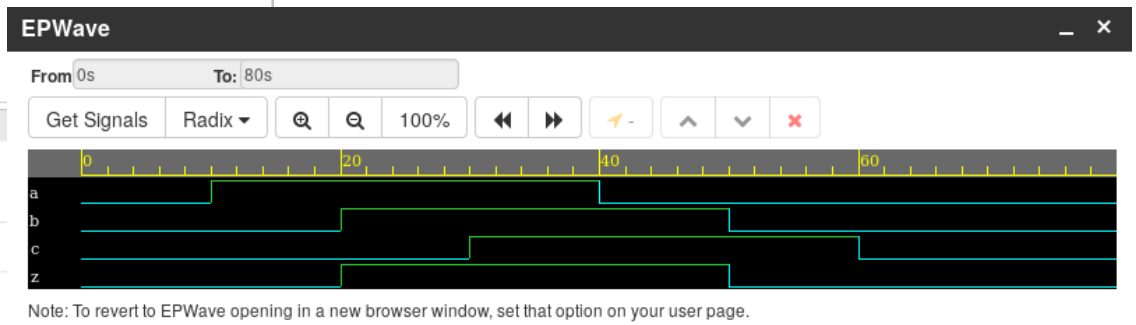
```
$ cd 02-1_voter1                        # change to workin directory

$ ls                                    # list files in the directory
voter_tb.v  voter.v

$ iverilog voter.v voter_tb.v           # compile design and test bench

$ vvp a.out                             # simulate design and test bench
VCD info: dumpfile voter_tb.vcd opened for output.

$ ls                                    # list generated files
a.out  voter_tb.v  voter_tb.vcd  voter.v

$ gtkwave voter_tb.vcd &                # open waveform viewer
...
```

Xilinx's integrated environments

- Xilinx
  - One of the main FPGA verndors
- Two design environments available
  - ISE: old FPGA devices.
  - Vivado: newer FPGA devices.
- Only for Xilinx FPGA's
- Includes everything: code editor, simulator, synthesis tools and much more.
- Complete but can be complex for beginners.
- Heavy to download (~5GiB) and install (>10GiB).
- Web registration required to obtain a free licence.
  - Free license for academic use with limitations.
- MS-Windows and GNU/Linux versions.

WWW environment: EDAPlayground





- Support Verilog and other languages.
- Icarus Verilog available.
- Includes text editor an waveform viewer.
- Requires web registration.

https://www.edaplayground.com/

Examples-based Verilog course

- Verilog examples with complete comments.
- Includes all concepts in the course.

verilog-course.v

Summary. LDH

- Behavioral description of digital circuits.
- Various description styles and levels
  - Structural, functional and procedural.
- Objectives:
  - Simulation before implementation.
  - Automatic implementation (synthesis).
- Standard modern digital circuit design methodology