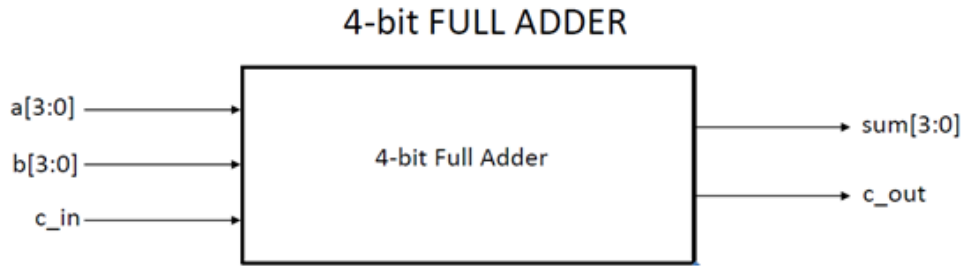
ELEC 2531

Lab 1

Introduction to digital electronics:

- Design using the (System)Verilog language
- Simulation using ModelSim.

-> Case study: the 4-bit full adder



Who's Who

- Professor: Jean-Didier Legat

- Teaching assistants:
  - Julien Verecken
  - Charles Momin
  - Pol Maistriaux

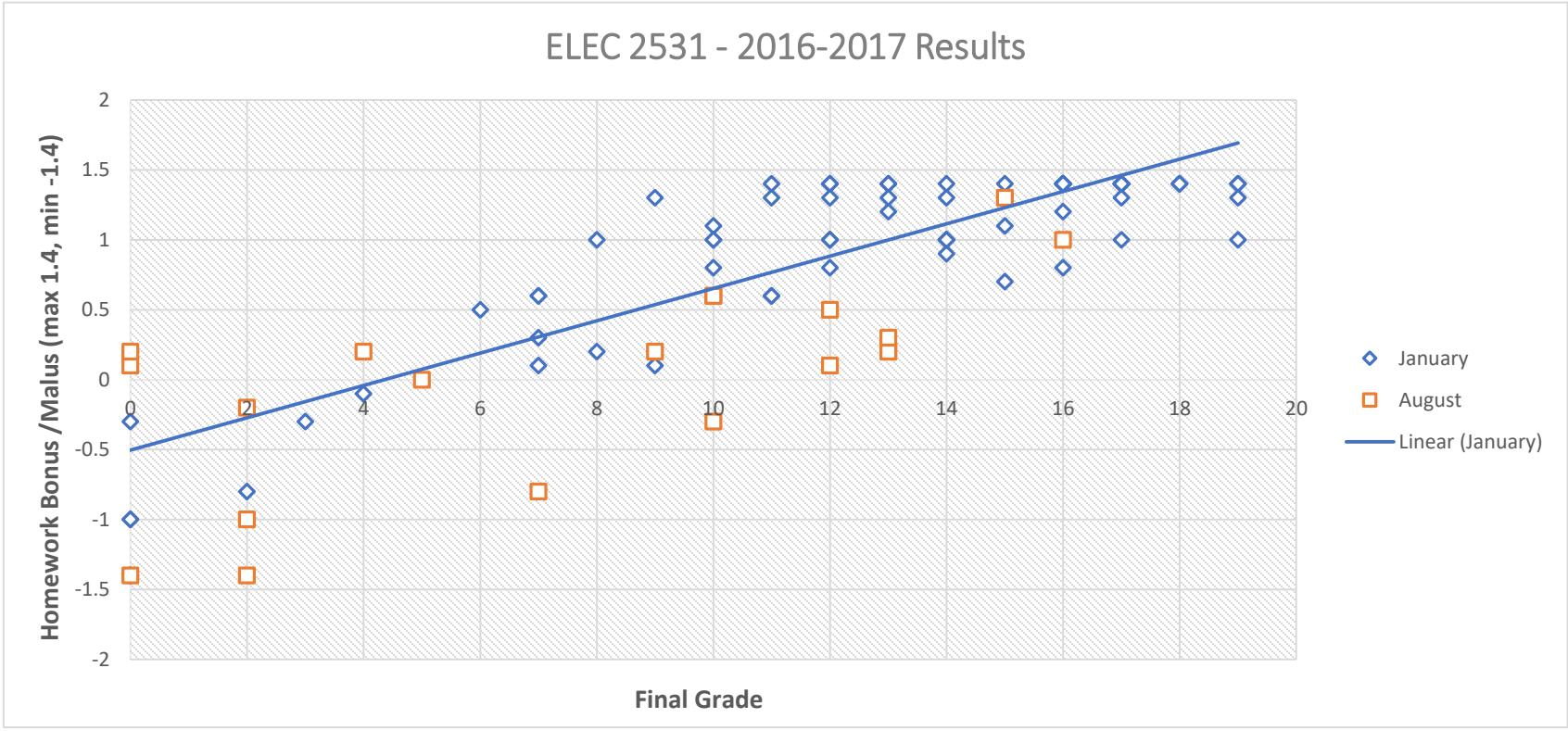
Contact: as much as possible through the student Moodle forum. Contact us by mail only for personnal matters!

Very Important remarks

- Do all the given personal tasks
  - Reading (before the lab!), exercices, quizz, Rasp-Pi stuffs, …
- Prepare and attend all the labs
  - First of all, it is mandatory
  - Secondly, there's only four of them...
- Do all the homeworks
  - Even if you don't submit them on time
- Use the Moodle forum
  - It's an important mutual aid resources, don't neglect it !

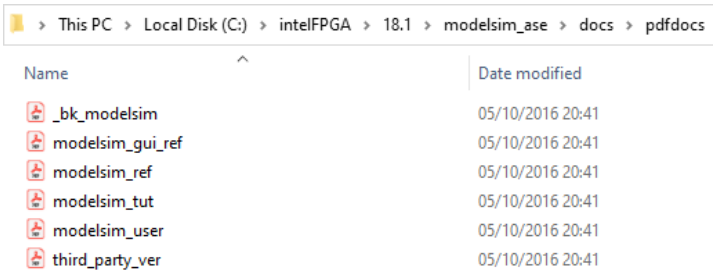
-> Do all this without delays and the evaluations will be a complete piece of cake. Do the opposite and it might turn to be highly complex…

« Do all this without delays and the evaluations will be a complete piece of cake. Do the opposite and it might turn to be highly complex… »



Documentation: where and why?

- We use Altera's Quartus suite 18.1 Lite Edition (ModelSim + Quartus)
- (System)Verilog:
  - See the web, e.g. Asic-World, Verilog 2001 Ref Guide or System Verilog Ref Guide
- ModelSim:
  1. Understand the Graphical User Interface (GUI)
  2. Command Reference Manual, if you want to use ModelSim using the command line
  3. Tutorial: the most useful document for you at this point, as it is an exhaustive introduction of ModelSim
  4. User manual: maybe less useful for you this year, but still, some chapters on Waveform Analysis, TCL and DO files can be really useful.



- Why and how to read documentation?
  - As your education goes on, we expect you to be more and more able to find information by yourself in order to understand new tools, concepts or technologies.
  - Hence, you have to know where to find the right documentation and what it contains. Obviously these documents are very long and should not be read from start to finish, but rather used as something to refer to.
  - This is especially true in electronics! At the end of your master, you should be really used to cope with this kind of large and verbose documentation, so you should start as soon as possible.

Quick Restructuring 1 : Computer-Aided Design (CAD) for FPGA

- We use a programming language - (System)Verilog - to **describe (HDL)** a circuit that we can:
  - Simulate at RT Level (ModelSim)
    - It is a behavioral simulation, meaning the circuit elements are considered ideal ; at this level, we don't care about timing or area, we want to check the functionality.
    - RT Level stands for « Register-Transfer Level » (a high-level circuit representation).
  - Synthesize (Quartus) & simulate at Gate Level (ModelSim)
    - See lab 2
  - Place & route the circuit and program your FPGA (Quartus)
    - See lab 3

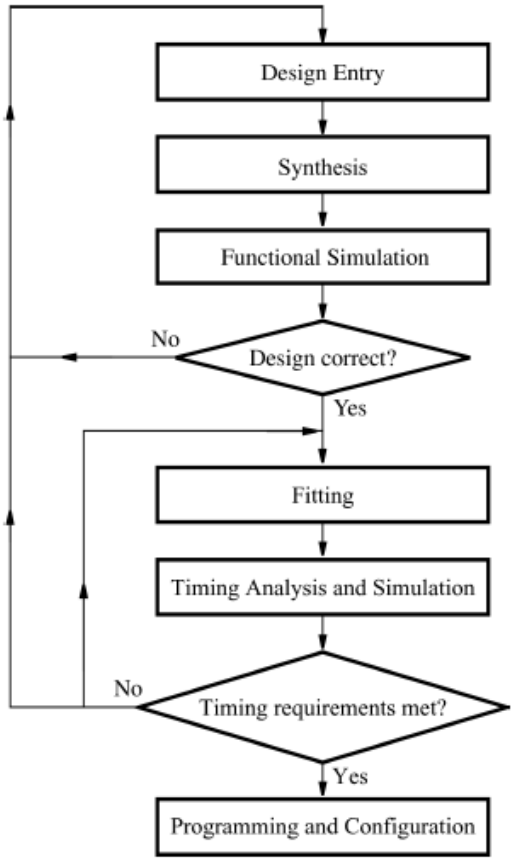
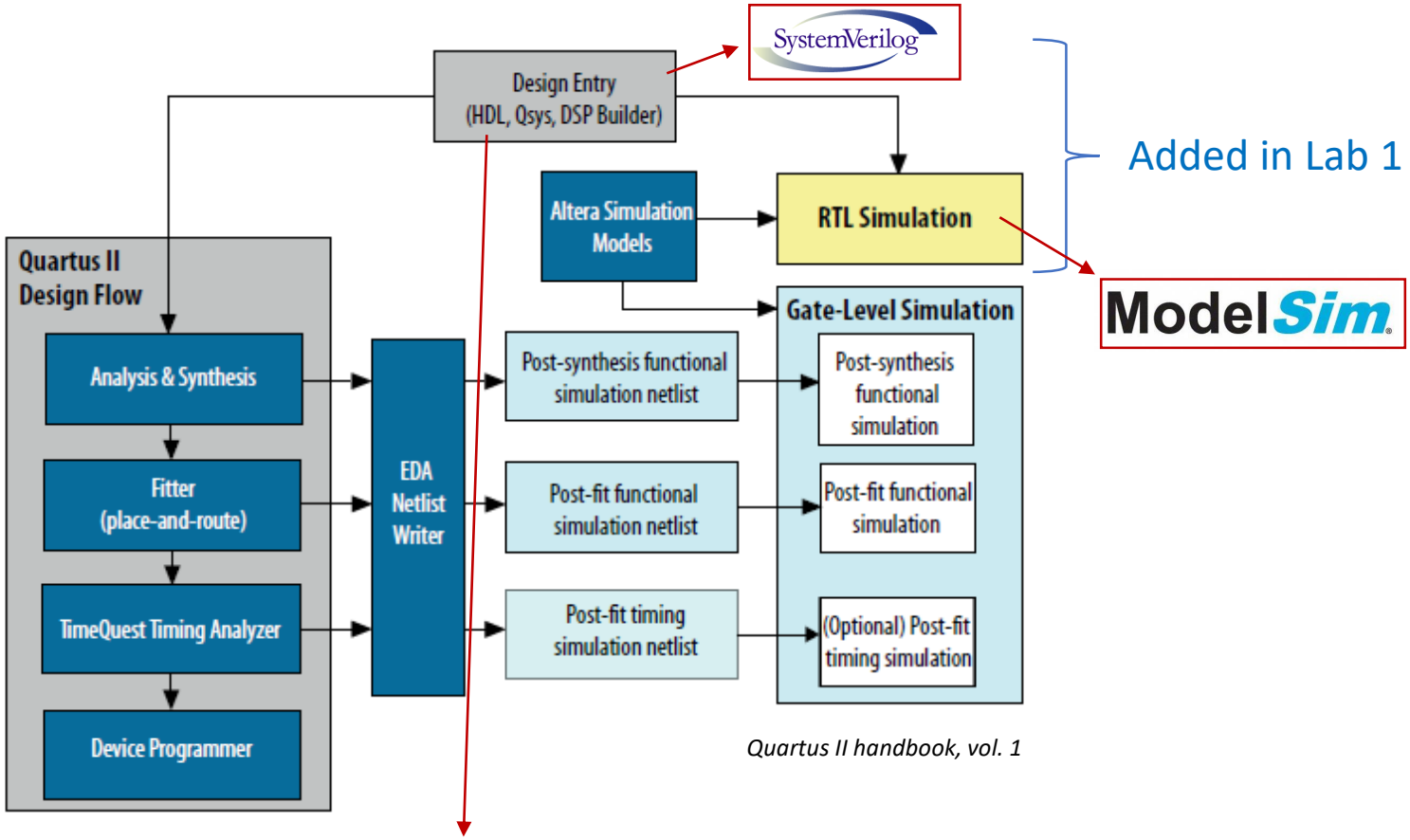


Figure 1: Typical CAD flow.

Electronic Design Automation (EDA) Flow for FPGA

- We use a programming language - (System)Verilog - to **describe (HDL)** a circuit (our design entry) that we can simulate at different levels (ModelSim) and use to program a physical device (our FPGA, using Quartus).

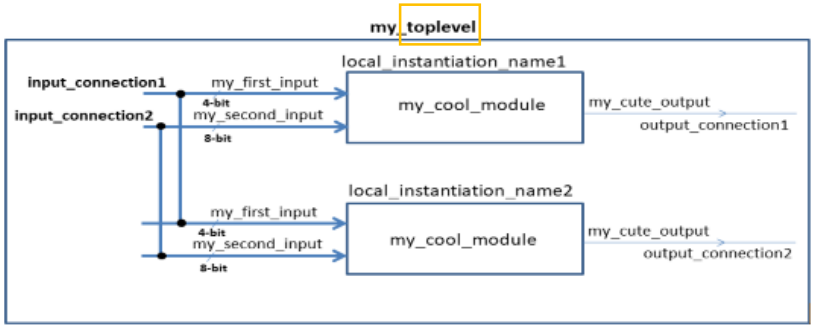


Quartus II handbook, vol. 1

- A circuit description (HDL) file is called a **netlist**.

Quick Restructuring 2 : Abstraction & 3 Y's Rule

- It is of primary importance for you to understand asap why and how we use these concepts to design digital electronics circuit:
  - Abstraction: can be used outside or inside a design level
    - Outside: gather circuit elements into higher-level abstractions to ease their usage (e.g. Verilog operators: « & »)
    - Inside: treat parts of the circuits or IPs as black boxes and care only about their I/Os.
  - 3 Y's:
    - Hierarchy, Modularity, Regularity… must be your new credo !
    - Verilog designs are hierarchized using modules (the one of highest level is called the top-level module) and use extensively the concepts of modularity and regularity to ease the design and make it robust



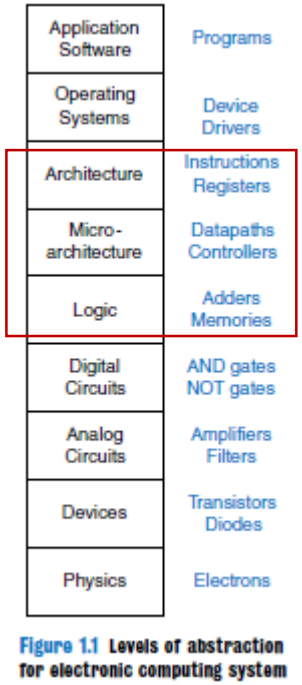
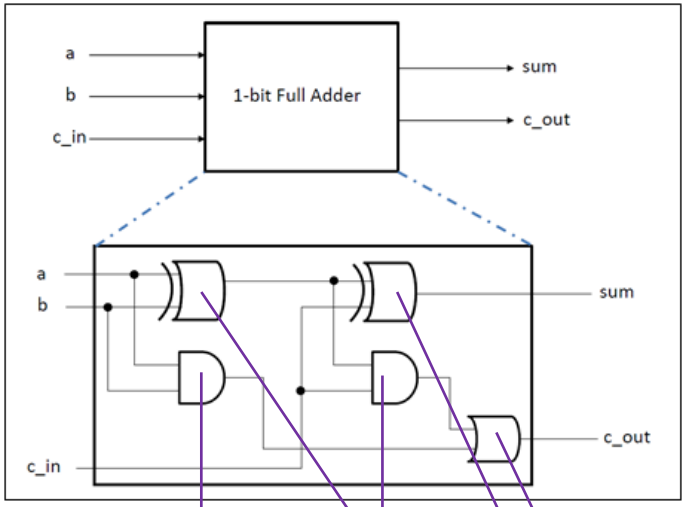


Figure 1.1 Levels of abstraction for electronic computing system

Quick restructuring 3 : Verilog's modules

Insert sublevels into a design level ; 3 Y's perfect application



2. Module instantiation:

- As much as we want
- Unique identifiers
- Achieved in a higher-level module

```
module full_adder_4_bit (
    input  logic [3:0]  a,
    input  logic [3:0]  b,
    input  logic        c_in,
    output logic [3:0]  sum,
    output logic        c_out);

wire c1, c2, c3;

full_adder_1_bit FA0 (
    .a(a[0]),
    .b(b[0]),
    .c_in(c_in),
    .sum(sum[0]),
    .c_out(c1));

full_adder_1_bit FA1 (
    .a(a[1]),
    .b(b[1]),
    .c_in(c1),
    .sum(sum[1]),
    .c_out(c2));

full_adder_1_bit FA2 (
    .a(a[2]),
    .b(b[2]),
    .c_in(c2),
    .sum(sum[2]),
    .c_out(c3));

full_adder_1_bit FA3 (
    .a(a[3]),
    .b(b[3]),
    .c_in(c3),
    .sum(sum[3]),
    .c_out(c_out));

endmodule
```

1. Module declaration:

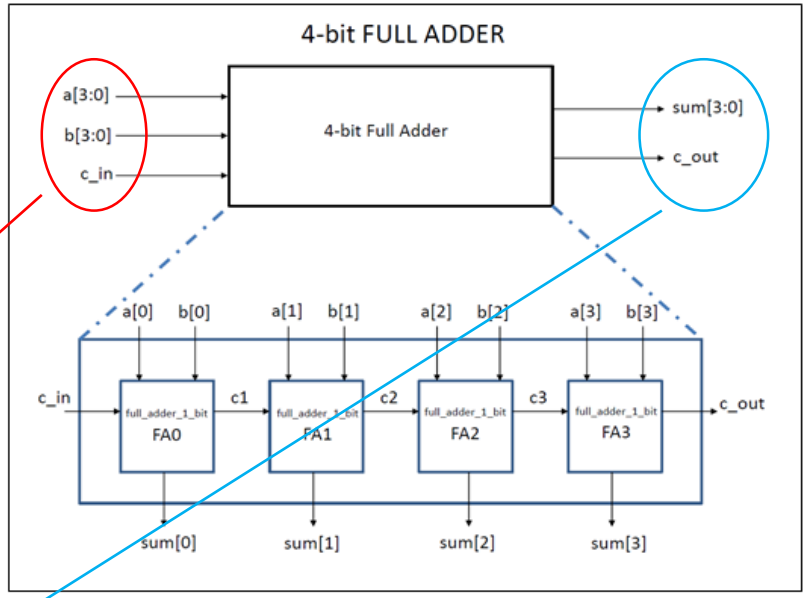
- Unique
- Describes a circuit element or part using high-level abstraction operators
- Can be done in a separate .v or .sv file as long as it is part of the design (i.e. the ModelSim or Quartus project)

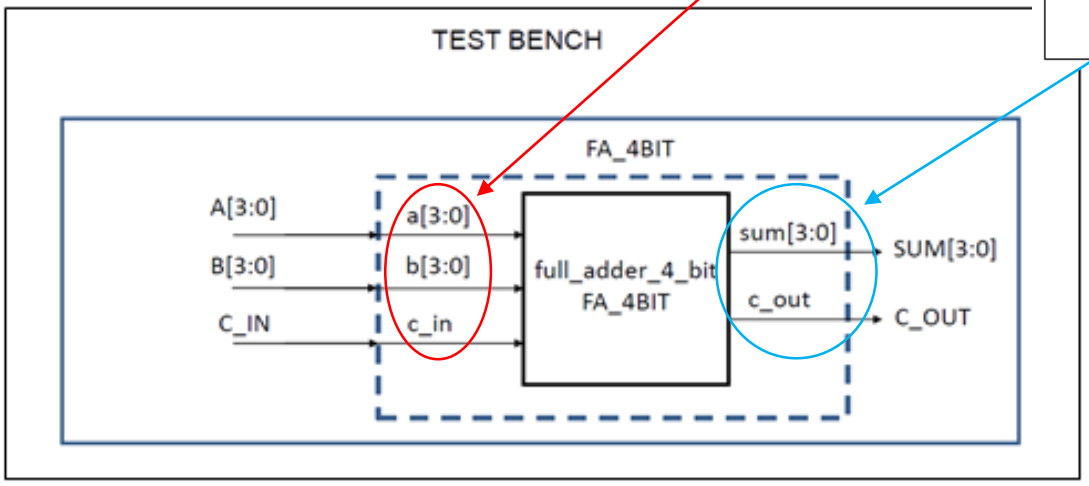
```
module full_adder_1_bit (
    input  logic a,
    input  logic b,
    input  logic c_in,
    output logic sum,
    output logic c_out);

assign sum = a ^ b ^ c_in;
assign c_out = (a & b) | ((a ^ b) & c_in);

endmodule
```

Quick restructuring 4 : Testbench

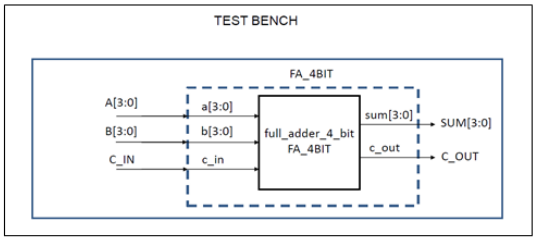
- Today's case study is hence a good illustration of the two previous slides

- To simulate it, we need to provide the external stimulis (inputs/outputs) using what is called a **testbench**





- Let's quickly review today's testbench:



- The testbench is another module, one level higher than the design's top-level module, given that it has to instantiate it
  - Don't declare any other module in the testbench file
  - Unlike the other modules, you don't have to instantiate it.
- The simulator time unit and precision are set using the « `timescale » preprocessor directive
  - ModelSim won't work if you don't define this
  - Should be defined only in the testbench file
- Some « logic » elements have to be defined, at least for the top-level I/Os
- A non-synthezisable « initial » structure is used to apply external stimuli to the top-level inputs
  - The (also non-synthezisable) « # » operator is used to provide timing delays to the simulation (the time unit being the one fixed by the timescale).

```
`timescale 1ns/1ps // time unit/compiler time precision

module test_fulladder_4bit ();

logic [3:0] A, B;
logic C_IN;
logic [3:0] SUM;
logic C_OUT;

// Instantiate the 4-bit full adder that we call FA_4BIT

full_adder_4_bit FA_4BIT (.a(A),.b(B),.c_in(C_IN),
                          .sum(SUM),.c_out(C_OUT));

// Stimulate Inputs

initial
    begin
        A = 4'd0;
        B = 4'd0;
        C_IN = 1'b0;

        # 5 // wait after 5 delay time

        A = 4'd3;
        B = 4'd4;

        # 5 // wait after 5 delay time

        (...)
    end

endmodule
```