Introduction to FPGA Programming

XV Seminar on Software for Nuclear, Subnuclear and Applied Physics - Alghero (Italy)

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Mirko Mariotti

27 May - 1 June, 2018

Contents

UNIVERSITÀ DEGLI STUDI
DI PERUGIA

Computing Devices

Mirko Mariotti

Types of computing devices

- Microprocessors (CPUs).
- Microcontrollers (Arduino, PICs, Etc).
- GPU.
- SoCs (RaspberryPI, Etc).
- ASICs.

Programmability

- All are electronic components.

Programmability

- All are electronic components.
- All are digital.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Programmability

- All are electronic components.
- All are digital.
- All are programmable.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Programmability

- All are electronic components.
- All are digital.
- All are programmable.

Does the component behavior changes ?

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Programmability

- All are electronic components.
- All are digital.
- All are programmable.

Does the component behavior changes ?

There is another class of electronic components: the programmable logic ones.

Programmable Logic

A Field Programmable Gate Array (FPGA) is the most important example of programmable logic device.

There is difference among:

- Programmable CHIP.
- Programmable Logic CHIP.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Why FPGA is becaming so interesting in computing ?

Computer Architectures

Today's computer architecture are:

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
    - The power is given by the number.
    - Parallel algorithms.
- Heterogeneous, processing units of different type.
    - Cell, GPU, Parallela, TPU.

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.

Computer Architectures

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.

Computer Architectures

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.
  - Hard to schedule.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Microprocessor

Microcontroller

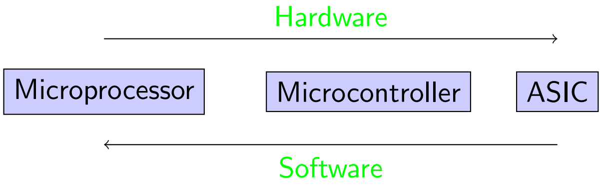
ASIC

HW/SW

Hardware →

Microprocessor | Microcontroller | ASIC

HW/SW



HW/SW

Specialization $\longrightarrow$

Microprocessor | Microcontroller | ASIC

HW/SW

Specialization →

Microprocessor | Microcontroller | ASIC

← Generalization

HW/SW

Microprocessor

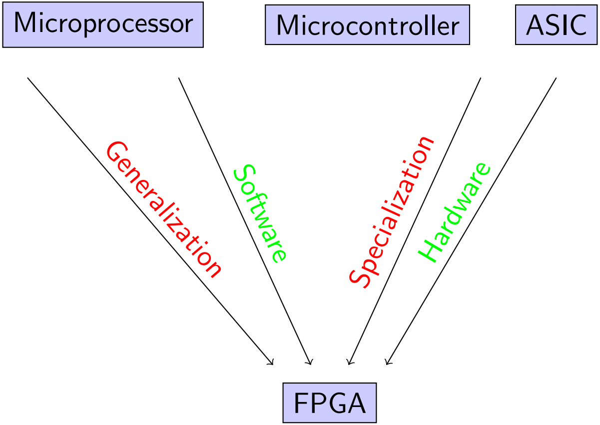
Microcontroller

ASIC

FPGA

UNIVERSITÀ DEGLI STUDI DI PERUGIA

HW/SW



UNIVERSITÀ DEGLI STUDI DI PERUGIA

Why FPGA

- We can work with a minor gap among HW/SW.

Why FPGA

- We can work with a minor gap among HW/SW.
- We can benefit from the intrinsic concurrency of FPGA.

Why FPGA

- We can work with a minor gap among HW/SW.
- We can benefit from the intrinsic concurrency of FPGA.
- We can easily build heterogeneous systems.

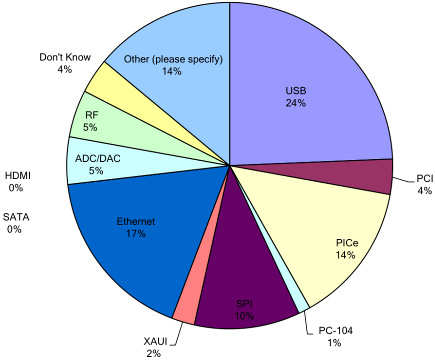
UNIVERSITÀ DEGLI STUDI
DI PERUGIA

FPGA

Mirko Mariotti

FPGA for prototyping

FPGA exists since several years, their main purpose was for prototyping.



FPGA for general purpose

Later (about 2008), we had many advances in this kind of technology:

UNIVERSITÀ DEGLI STUDI DI PERUGIA

FPGA for general purpose

Later (about 2008), we had many advances in this kind of technology:

- The number of cell increased.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

FPGA for general purpose

Later (about 2008), we had many advances in this kind of technology:

- The number of cell increased.
- The operating frequencies also increased.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

FPGA for general purpose

Later (about 2008), we had many advances in this kind of technology:

- The number of cell increased.
- The operating frequencies also increased.
- Device become cheaper.

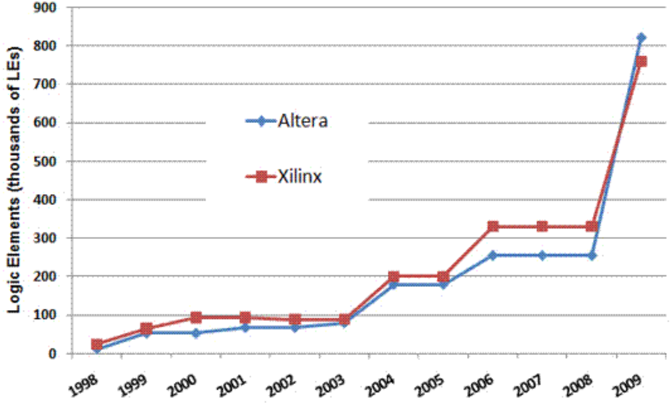
FPGA for general purpose

Later (about 2008), we had many advances in this kind of technology:

- The number of cell increased.
- The operating frequencies also increased.
- Device become cheaper.

FPGA made their appearance directly, within products.

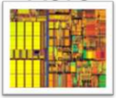
Number of Logic Elements



ASIC or FPGA

Hardware Description

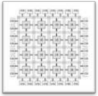
ASIC



Application Specific Integrated Circuit

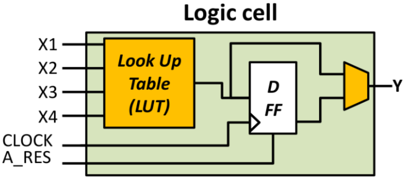
- Built with lithographic processes
- High initial costs
- Low costs per unit
- Higher time to market
- Higher frequency

FPGA



Field Programmable Gate Array

- Reconfigurable Hardware
- Null initial investment
- High cost per unit
- Immediatly operational
- Minor frequency



General reference Cell model

Cells details can change for different vendors or FPGA models.

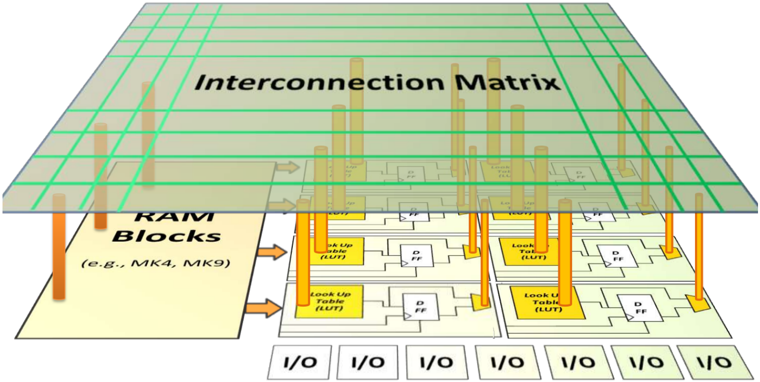
Look-Up Table

X1	X2	X3	X4	Y
0	0	0	1	?
0	0	1	0	?
...	...	...	...	?
1	1	1	1	?

Programmable element

A Cell usually contains:

- A **look-up table**: allow to map any combinatorial function of 4 inputs and 1 output.
- a **FF of type D**: to store persistent data.
- A **mux 2 -> 1**: to eventually bypass the FF for purely combinatorial cells.


Interconnection Matrix
RAM Blocks
(e.g., MK4, MK9)
Look Up Table (LUT)
D FF
I/O
I/O
I/O
I/O
I/O
I/O
I/O

FPGA Vendors and software

- Xilinx: Vivado
- Altera (Intel): Quartus Prime
- Lattice: Icecube2

Other software

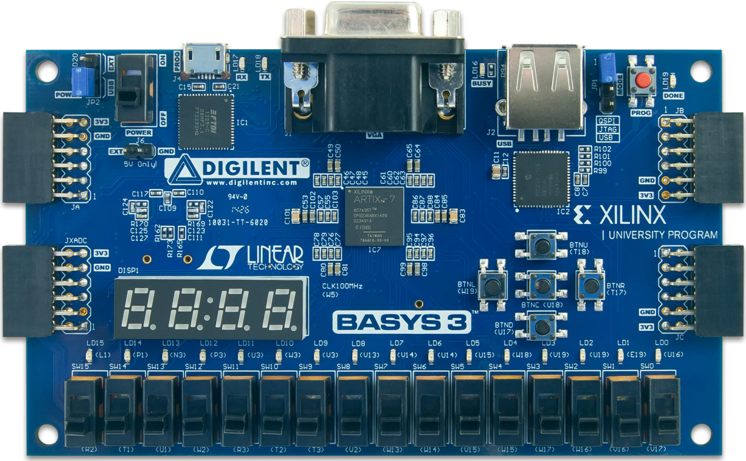
Not open source:

- Modelsim

Open source:

- Iverilog
- GHDL
- Gtkwave

Evaluation Board
Basys3



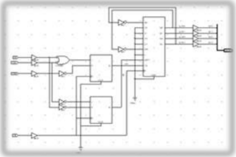
UNIVERSITÀ DEGLI STUDI
DI PERUGIA

Programming

Mirko Mariotti

Programming techniques

Blocks



PROS:
- Easy and fast development
- Legacy components

CONS:
- Non standard files
- Difficult to maintain
- Difficult for complex designs

Hardware Description Languages

VHDL, Verilog, SystemC

```
begin
  if (RESET_N = '0') then

    for col in 0 to BOARD_COLUMNS-1 loop
    ...
   elsif (rising_edge(CLOCK)) then
```

PROS:
- Standard languages
- Flexibility and easy maintenance
- Complex designs

CONS:
- Computational model

Languages

The programming of FPGA devices usually is done through HDL (Hardware Description Language) languages.

The most used are:

- Verilog
- VHDL

HDL languages offer a computational model to build hardware. Although they may appear standard programming languages like C, C++ or Java, they are not.

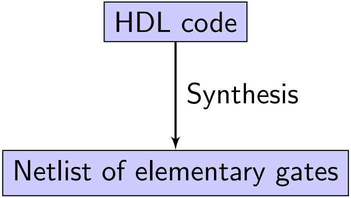
In standard programming languages, statements define instructions that are sequentially executed by the infrastructure (the CPU or a VM). On the other hand, in HDL languages statements define hardware blocks, circuits.

There is no sequential execution, no infrastructure, no run-time.

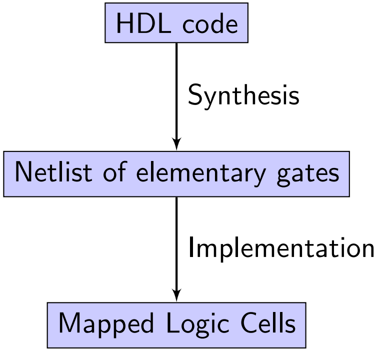
FPGA Programming workflow

HDL code

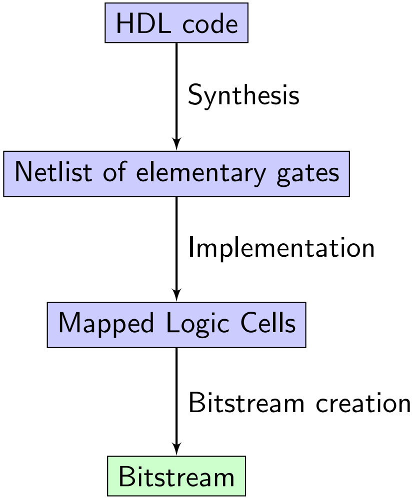
FPGA Programming workflow



FPGA Programming workflow



FPGA Programming workflow



UNIVERSITÀ DEGLI STUDI DI PERUGIA

Synthesis

The synthesis phase take an HDL description and transform it into a netlist of elementary gates

The synthesis usually acts on a subset of the HDL language

The synthesis occurs applying templates that match language structures and map them to logic components.

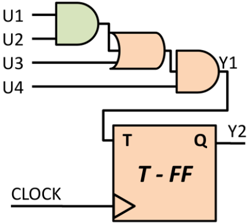
The synthesis result depends upon the mapping libraries in use and upon the used synthesizer.

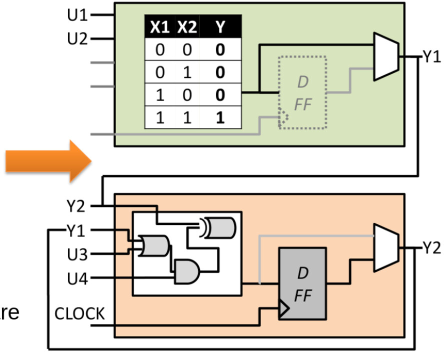
Synthesis cont.

The synthesis essentially transform the HDL code into a block diagram

This block diagram representation has not (yet) a direct correspondence with the hardware, it is only a functional purpose

Implementation





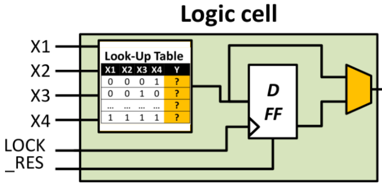
The components inferred during the synthesis phase are mapped on the hardware using the actual FPGA resources.

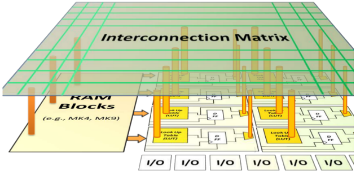
Implementation

During the implementation phase several steps are carried on:

- The memory elements are identified (registers, counters, shift-registers) and mapped to elementary Flip-Flop
- The transfer functions (boolean equations) are also identified. Paths could be among (i) registers, (ii) inputs and registers, (iii) registers and outputs, (iv) inputs and outputs.
- The equations are reduced (optimization)
- The resulting equations are mapped to the look-up table of the logic cells

The values of the look-up tables, muxes and interconnection can be packed in a file called bitstream that can be used to program a specific FPGA (filling the yellow elements)





Simulation and Verify workflow

There is another very important work-flow to consider

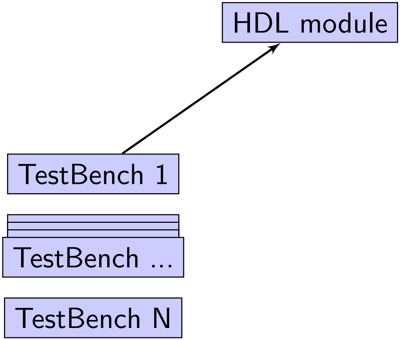
Simulation and Verify workflow

There is another very important work-flow to consider

HDL module

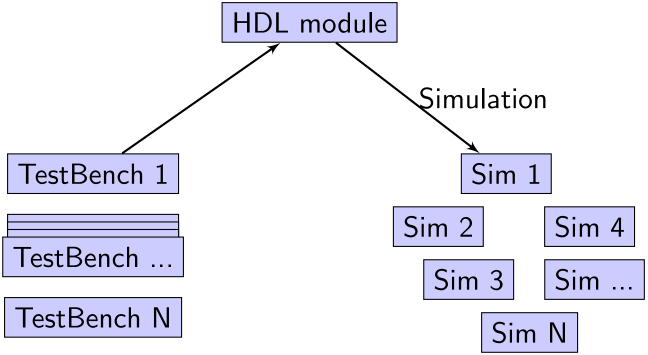
Simulation and Verify workflow

There is another very important work-flow to consider



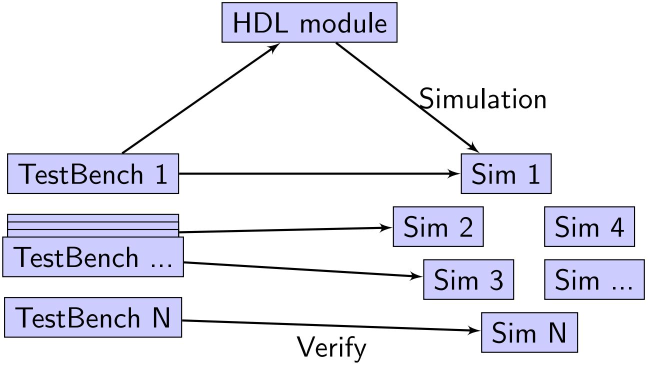
Simulation and Verify workflow

There is another very important work-flow to consider



Simulation and Verify workflow

There is another very important work-flow to consider



Simulation and Verify

An important part of FPGA programming is to simulate the components behavior

A Testbench is a verilog module created to test the functionalities of other components

UNIVERSITÀ DEGLI STUDI
DI PERUGIA

Verilog

Mirko Mariotti

Modules

- Verilog modules are blocks of code (circuit) that can be reused within other blocks (similar to functions in computer programming languages)

Modules

- Verilog modules are blocks of code (circuit) that can be reused within other blocks (similar to functions in computer programming languages)
- The keyword "module" starts a block that ends with "endmodule".

Modules

- Verilog modules are blocks of code (circuit) that can be reused within other blocks (similar to functions in computer programming languages)
- The keyword “module” starts a block that ends with “endmodule”.
- They could have input and output parameters.
-
```
module MyModule ([parameters]);
     inputs ...
     outputs ...
     internal variables ...
     ...
     Module Code ...
endmodule
```


UNIVERSITÀ DEGLI STUDI DI PERUGIA

Main Module

- A specific module is tagged as the main module. (similar to the main on a programming language)

Main Module

- A specific module is tagged as the main module. (similar to the main on a programming language)
- It is the “program” entry point.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Main Module

- A specific module is tagged as the main module. (similar to the main on a programming language)
- It is the “program” entry point.
- Usually it has inputs and outputs connected to FPGA physical IO.

Operators

<table>
<tr><th>Operator Type</th><th>Operator Symbol</th><th>Operation Performed</th></tr>
<tr><td>Arithmetic</td><td>*</td><td>Multiply</td></tr>
<tr><td></td><td>/</td><td>Division</td></tr>
<tr><td></td><td>+</td><td>Add</td></tr>
<tr><td></td><td>-</td><td>Subtract</td></tr>
<tr><td></td><td>%</td><td>Modulus</td></tr>
<tr><td></td><td>+</td><td>Unary plus</td></tr>
<tr><td></td><td>-</td><td>Unary minus</td></tr>
<tr><td>Logical</td><td>!</td><td>Logical negation</td></tr>
<tr><td></td><td>&&</td><td>Logical and</td></tr>
<tr><td></td><td>||</td><td>Logical or</td></tr>
<tr><td>Relational</td><td>></td><td>Greater than</td></tr>
<tr><td></td><td><</td><td>Less than</td></tr>
<tr><td></td><td>>=</td><td>Greater than or equal</td></tr>
<tr><td></td><td><=</td><td>Less than or equal</td></tr>
<tr><td>Equality</td><td>==</td><td>Equality</td></tr>
<tr><td></td><td>!=</td><td>inequality</td></tr>
<tr><td>Reduction</td><td>~</td><td>Bitwise negation</td></tr>
<tr><td></td><td>~&</td><td>nand</td></tr>
<tr><td></td><td>|</td><td>or</td></tr>
<tr><td></td><td>~|</td><td>nor</td></tr>
<tr><td></td><td>^</td><td>xor</td></tr>
<tr><td></td><td>^~</td><td>xnor</td></tr>
<tr><td></td><td>~^</td><td>xnor</td></tr>
<tr><td>Shift</td><td>>></td><td>Right shift</td></tr>
<tr><td></td><td><<</td><td>Left shift</td></tr>
<tr><td>Concatenation</td><td>{ }</td><td>Concatenation</td></tr>
<tr><td>Conditional</td><td>?</td><td>conditional</td></tr>
</table>

Wire:
They are used to connect different elements. They can be thought as physical wires. They can be read or assigned but they does not store information. Indeed they need to be continuously driven from an assignment or a module port.

Reg:
Store a value in Verilog. The value is kept until assigned again (similar to a variable)

Numbers

The generic number value in Verilog is:
<size>‘<base><value>

Esempi:
12 // decimal (base 10)
8 ‘h5F // hexadecimal, 8 bit
6 ‘b11_0010 // binary, 6 bit
‘o576 // octal with no dimension
32 ‘bz // binary a 32 bit Hi-Z

The char _ is ignored and can be used as separator.

Scalars and Vectors

Variable can be scalar or vectors.

Examples:

```
reg out; // scalar
reg [7:0] databus; // 8 bit bus
wire [1:0] select; // 2 bit bus
wire enabled; // scalar
```

Blocks

The verilog block are defined with begin - end.

There are two types of blocks:

Initial block
Is executed when the simulation start or as initial value.

Always block
It is always executed and is associated to a list that specify when execute the block

```
always  @ (a or b or sel)
begin
   ...
end
```

Blocking assignment
A = 3;
The expression is evaluated in the execution flux and the variable assigned immediately.

NON blocking assigment
A <= A + 1;
The expression is evaluated in the execution flux and the result stored in a temporary variable and assigned on the next step

Continuous assignment:
assign A = in1 & in2; // A is a wire type
used to model combinatorial logic and rename signals.

Flow control
If-else e case

```
if( A == B )
begin
   C = 0;
end
else
begin
   C = 1;
end
```

```
case (sel)
   2'b00: out = 0;
   2'b01: out = 1;
   default: out = x;
endcase
```


UNIVERSITÀ DEGLI STUDI DI PERUGIA

UNIVERSITÀ DEGLI STUDI
DI PERUGIA

Exercises

Mirko Mariotti

Exercises

https://www.mirkomariotti.it/courses/20180529-alghero

Specs:

- Vendor: Digilent
- Chip: Xilinx Artix 7 - XC7A35T-1CPG236C
- Cells: 33280
- Frequency: 450 MHz
- 16 user switches
- 16 user LEDs
- Part number: xc7a35ticpg236-1L
- ...

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Exercise:

- Creation of a Vivado project.
- Creation of a verilog module for the not gate.

```
module   notm (
      input wire A,
      output wire B
   );

   assign B = ~ A;

endmodule
```

- Synthesis.
- Check of the resulting block diagram.
- Check of the truth values.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Exercise:

- Try the same with the OR gate and with the AND gate.

UNIVERSITÀ DEGLI STUDI DI PERUGIA

Chack what is the result with a more complex example

```
module complex(
    input wire A,
    input wire B,
    input wire C,
    output wire D,
    output wire E

);

    assign D = A & (B | C);
    assign E = B | C;
endmodule
```

Logic ports
a more complex example 2

Chack what is the result with a more complex example

```
module complex(
    input wire A,
    input wire B,
    input wire C,
    input wire D,
    input wire E,
    input wire F,
    input wire G,
    output wire H
);

    assign H = A & ((B | C) & (~D & (F | ~G)) | E);
endmodule
```

Constraints file

- The constrains file maps the evaluation board peripherals to FPGA I/O pins
- Usually its template is made by the board vendor.
- Has to be included in the project.
- Basys3_master.xdc

Exercise:

- Set the clock as an input.
- Set a led as an output.
- Make the led blink.
-

```
module blink(
     input  clk,
     output reg [7:0] led
);
    reg counter;
    initial
        counter = 0;
    always @ (posedge clk) begin
        led[0] <= counter;
        counter <= counter + 1;
    end
endmodule
```

07 - vivado - 2hz

Make a led blink with about 2Hz freq.

```
module counter(
     input  clk,
     output reg [7:0] led
);

    reg [32:0] counter;

    initial
        counter = 0;

    always @ (posedge clk) begin
        led[0] <= counter[23];
        counter <= counter + 1;
    end

endmodule
```

08 - vivado - switchsel

Make a led blink with about 2Hz freq, but only if the first switch if on.

09 - vivado - doublefreq

Make a led blink with about 2Hz freq when the first switch is off. If the switch goes on double the frequence.

10 - vivado - counter

Create a binary counter with the 16 leds

11 - vivado - joystick

Starting from all the leds off, set the first led on when the central button of the joystick is pressed. If it is pressed again switch the led to off (and so forth).

12 - vivado - kitt

Starting from all the leds off except the first create a firmware that:

- Move the light right if the right joystick button if pressed (and the external right position is not reached)
- Move the light left if the left joystick button if pressed (and the external left position is not reached)