

Introduction to FPGA Programming



UNIVERSITÀ DEGLI STUDI
DI PERUGIA



- 1 **Computing Devices**
 - Types of computing devices
 - Programmability
 - Programmable Logic
 - HW/SW
- 2 **FPGA**
 - FPGA History
 - FPGA Architecture
 - FPGA Vendors and Software
 - Evaluation Board
- 3 **Programming**
 - Languages
 - Synthesis
 - Implementation
 - Simulation and Verify
- 4 **Verilog**
 - Modules
 - Blocks
 - Data
- 5 **Exercises**



UNIVERSITÀ DEGLI STUDI
DI PERUGIA

Computing Devices

- **Microprocessor** - a computer processor that incorporates the functions of a central processing unit (CPU) on a single integrated circuit (IC).
- **Microcontroller** - a small computer on a single IC. May contain one or more CPUs along with memory and programmable input/output peripherals. (Arduino, PICs, Etc).
- **SoC (System on Chip)** - integrates all components of a computer or other electronic system. (RaspberryPI, Etc).
- **GPU (Graphics Processing Unit)**.
- **ASIC (Application-Specific Integrated Circuit)** - IC customized for a particular use, rather than intended for general-purpose use.

- **Microprocessor** - a computer processor that incorporates the functions of a central processing unit (CPU) on a single integrated circuit (IC).
- **Microcontroller** - a small computer on a single IC. May contain one or more CPUs along with memory and programmable input/output peripherals. (Arduino, PICs, Etc).
- **SoC (System on Chip)** - integrates all components of a computer or other electronic system. (RaspberryPI, Etc).
- **GPU (Graphics Processing Unit)**.
- **ASIC (Application-Specific Integrated Circuit)** - IC customized for a particular use, rather than intended for general-purpose use.

- **Microprocessor** - a computer processor that incorporates the functions of a central processing unit (CPU) on a single integrated circuit (IC).
- **Microcontroller** - a small computer on a single IC. May contain one or more CPUs along with memory and programmable input/output peripherals. (Arduino, PICs, Etc).
- **SoC (System on Chip)** - integrates all components of a computer or other electronic system. (RaspberryPI, Etc).
- GPU (Graphics Processing Unit).
- ASIC (Application-Specific Integrated Circuit) - IC customized for a particular use, rather than intended for general-purpose use.

- **Microprocessor** - a computer processor that incorporates the functions of a central processing unit (CPU) on a single integrated circuit (IC).
- **Microcontroller** - a small computer on a single IC. May contain one or more CPUs along with memory and programmable input/output peripherals. (Arduino, PICs, Etc).
- **SoC (System on Chip)** - integrates all components of a computer or other electronic system. (RaspberryPI, Etc).
- **GPU (Graphics Processing Unit).**
- **ASIC (Application-Specific Integrated Circuit)** - IC customized for a particular use, rather than intended for general-purpose use.

- **Microprocessor** - a computer processor that incorporates the functions of a central processing unit (CPU) on a single integrated circuit (IC).
- **Microcontroller** - a small computer on a single IC. May contain one or more CPUs along with memory and programmable input/output peripherals. (Arduino, PICs, Etc).
- **SoC (System on Chip)** - integrates all components of a computer or other electronic system. (RaspberryPI, Etc).
- **GPU (Graphics Processing Unit).**
- **ASIC (Application-Specific Integrated Circuit)** - IC customized for a particular use, rather than intended for general-purpose use.

- 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.

- 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.

- 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.

- 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.

- 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.

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

There is difference among:

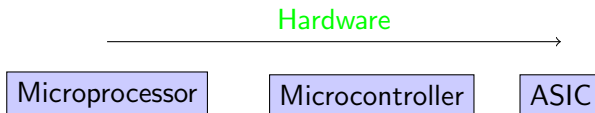
- Programmable CHIP.
- Programmable Logic CHIP.

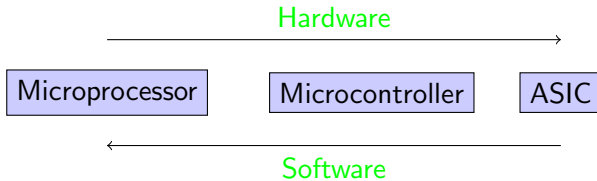


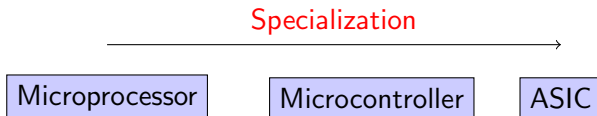
Microprocessor

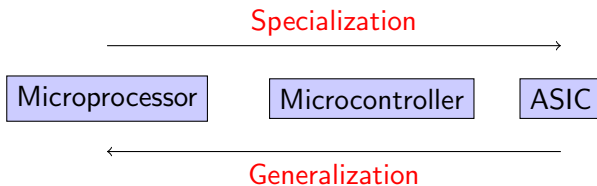
Microcontroller

ASIC









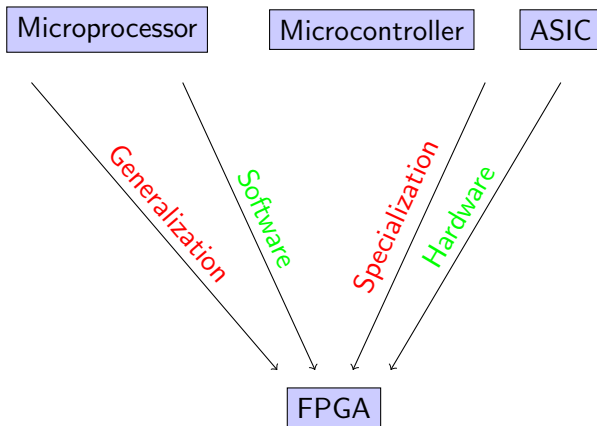


Microprocessor

Microcontroller

ASIC

FPGA





Why FPGA is becoming so interesting in computing ?

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

- 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.

- 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.

- 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.

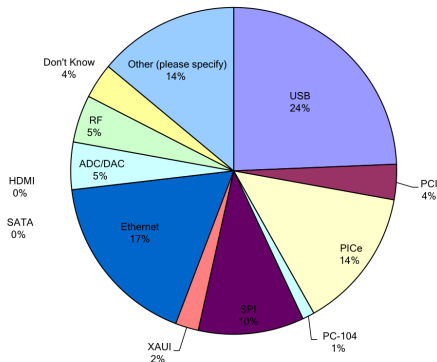
| CPU | GPU | FPGA |
|----------------------------|----------------------------|------------------------------------|
| Fixed architecture | Fixed architecture | Adaptable Architecture |
| Predefined instruction set | Predefined instruction set | No fixed instruction set |
| Fixed memory hierarchy | Fixed memory hierarchy | Customizable memory hierarchy |
| Thread-level parallelism | SIMD parallelism | Excels at all types of parallelism |



UNIVERSITÀ DEGLI STUDI
DI PERUGIA

FPGA

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



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.



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.

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.

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.

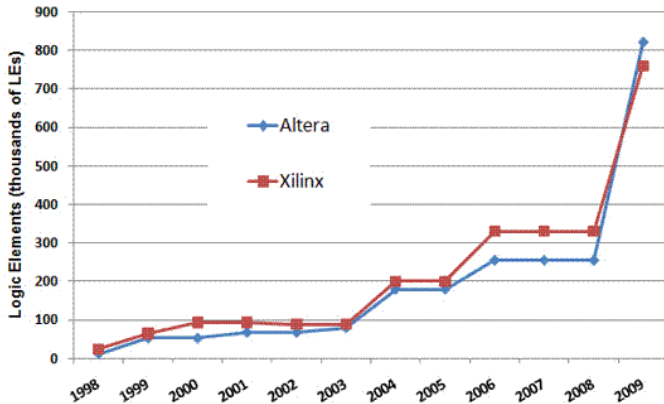
FPGA made their appearance directly, within products.

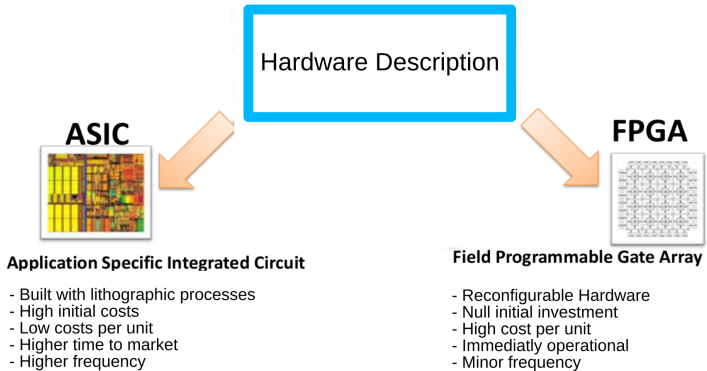
Later (about 2008), we had many advances in this kind of technology:

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

FPGA made their appearance directly, within products.

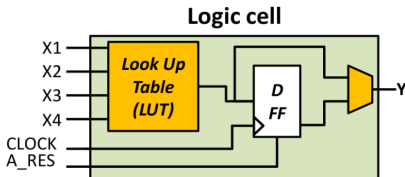
Number of Logic Elements





FPGA Architecture

Logic Cell



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 | ? |



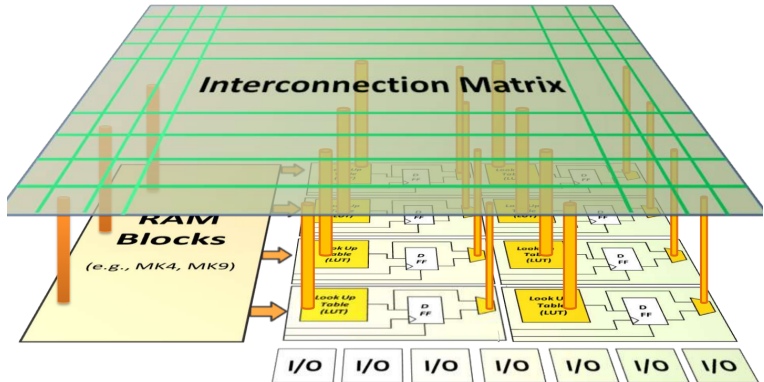
Programmable element

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.

FPGA Architecture

Interconnection





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



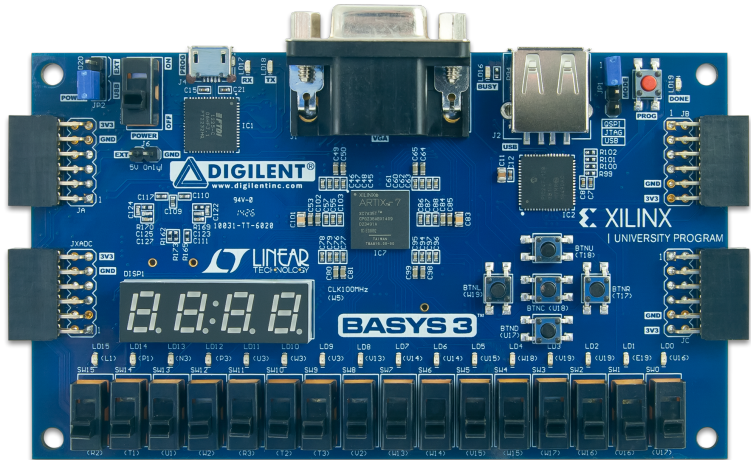
Not open source:

- Modelsim

Open source:

- Iverilog
- GHDL
- Gtkwave

Evaluation Board Basys3

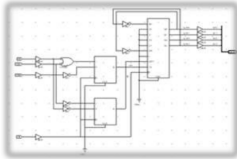




UNIVERSITÀ DEGLI STUDI
DI PERUGIA

Programming

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

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.

- 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.

- 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.

The HDL code is used to describe a circuit. It can be used as base for 2 different workflows.

- Programming workflow, the code is used to program a real FPGA device and use it
- Simulation and Verify workflow, the code is used to simulate the circuit via software

The HDL code is used to describe a circuit. It can be used as base for 2 different workflows.

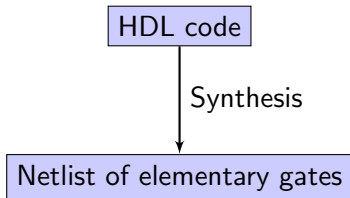
- Programming workflow, the code is used to program a real FPGA device and use it
- Simulation and Verify workflow, the code is used to simulate the circuit via software

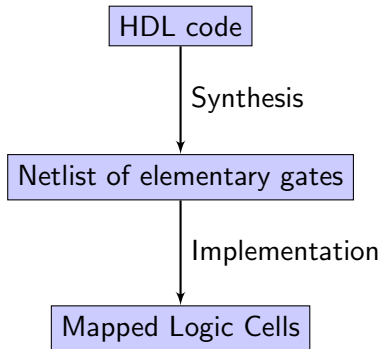
The HDL code is used to describe a circuit. It can be used as base for 2 different workflows.

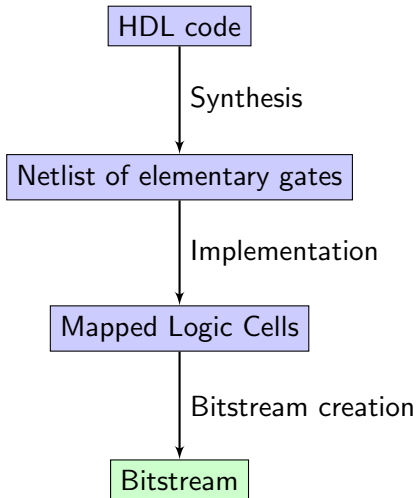
- Programming workflow, the code is used to program a real FPGA device and use it
- Simulation and Verify workflow, the code is used to simulate the circuit via software



HDL code







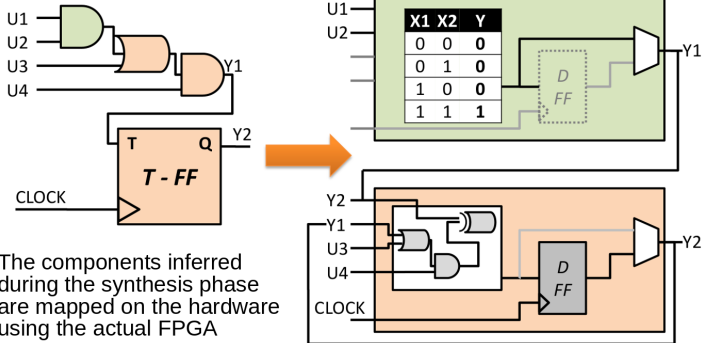
- The synthesis phase take an HDL description and transform it into a netlist of elementary gates
- The synthesis usually acts on a subset of the HDL language
- The synthesis occurs applying templates that match language structures and map them to logic components
- The synthesis result depends upon the mapping libraries in use and upon the used synthesizer

- 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

- 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

- The synthesis phase takes an HDL description and transforms it into a netlist of elementary gates
- The synthesis usually acts on a subset of the HDL language
- The synthesis occurs by 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

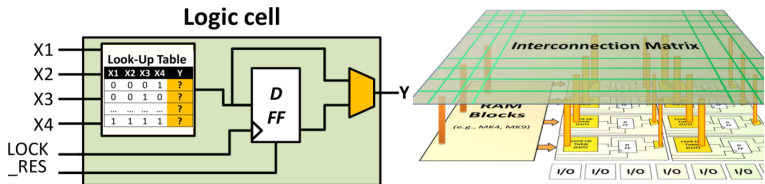
- The synthesis essentially transform the HDL code into a block diagram
- This block diagram representation has not (yet) a direct correspondence with the hardware, it is only a functional purpose



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)





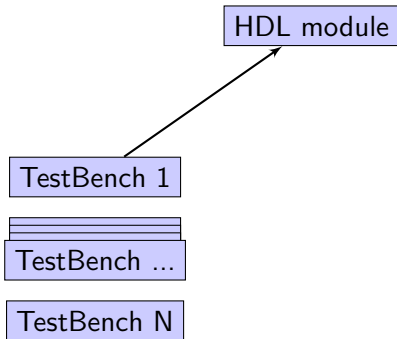
There is another very important work-flow to consider



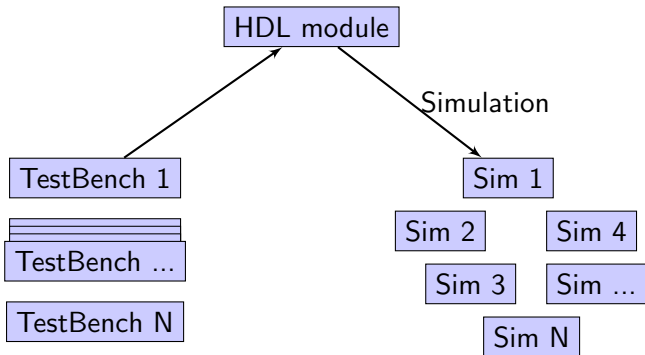
There is another very important work-flow to consider

HDL module

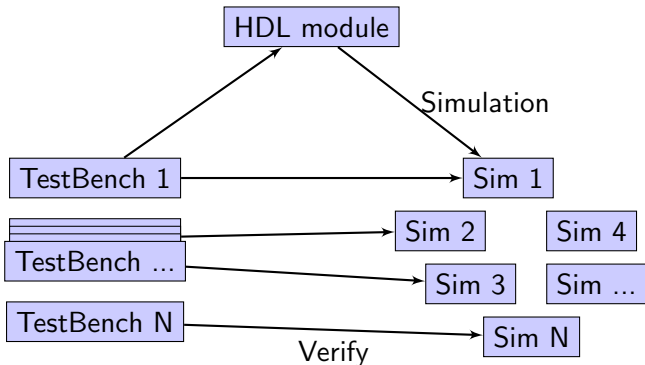
There is another very important work-flow to consider



There is another very important work-flow to consider



There is another very important work-flow to consider



- 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



- 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

- 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
```


- 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
```

- 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
```

- 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.

- 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.

- 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.

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
```

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
```

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
```


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
```

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)

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.

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
```

(reg) Blocking assignment:

`A = 3;`

The expression is evaluated in the execution flux and the variable assigned immediately.

(reg) NON blocking assignment:

`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

(wire) Continuous assignment:

`assign A = in1 & in2;`

used to model combinatorial logic and rename signals.

(reg) Blocking assignment:

`A = 3;`

The expression is evaluated in the execution flux and the variable assigned immediately.

(reg) NON blocking assignment:

`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

(wire) Continuous assignment:

`assign A = in1 & in2;`

used to model combinatorial logic and rename signals.

(reg) Blocking assignment: $A = 3;$

The expression is evaluated in the execution flux and the variable assigned immediately.

(reg) NON blocking assignment: $A \leq A + 1;$

The expression is evaluated in the execution flux and the result stored in a temporary variable and assigned on the next step

(wire) Continuous assignment: $\text{assign } A = \text{in1} \ \& \ \text{in2};$

used to model combinatorial logic and rename signals.

| Operator | Name |
|----------|---------------------------|
| [] | bit-select or part-select |
| () | parenthesis |
| ! | logical negation |
| ~ | negation |
| & | reduction AND |
| | reduction OR |
| ~& | reduction NAND |
| ~ | reduction NOR |
| ^ | reduction XOR |
| ~^ or ^^ | reduction XNOR |
| + | unary (sign) plus |
| - | unary (sign) minus |
| { } | concatenation |
| { { } } | replication |
| * | multiply |
| / | divide |
| % | modulus |

| Operator | Name |
|----------|--------------------------|
| + | binary plus |
| - | binary minus |
| << | shift left |
| >> | shift right |
| > | greater than |
| >= | greater than or equal to |
| < | less than |
| <= | less than or equal to |
| == | case equality |
| != | case inequality |
| & | bit-wise AND |
| ^ | bit-wise XOR |
| | bit-wise OR |
| && | logical AND |
| | logical OR |
| ?: | conditional |

Flow control

If-else, 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
```

Flow control

for

Similar to the C,C++ for loop. no support for ++

```
for (i = 0; i < 16; i = i +1) begin  
    $display ("Current value of i is %d", i);  
end
```



UNIVERSITÀ DEGLI STUDI
DI PERUGIA

Exercises

Specs:

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

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.



Exercise:

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



Logic ports

a more complex example 1

Check 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

Check 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
```


- The constraints 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
```

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
```



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



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



Create a binary counter with the 16 leds



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).

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

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



Starting from all the leds off except the first, create a firmware that moves it to the opposite side and then back and forth (a sort of bouncing led).



Set a binary number with the switches. The central button of the joystick store the number. Set a second number and map the other 4 buttons to the 4 operations. By pressing one of these the leds have to show the resulting operation.



Use the 7 segment display to count in hexadecimal from 0 to 10 and then from 10 to 0 and so forth.