# Computer Electronics

Lecture 5: IOb-SoC Purpose, Digital Circuits and Verilog

# Recap last lesson: the Iob-SoC <u>submodules</u> directory

- System submodules
  - CPU
  - CACHE
  - UART
- A git submodule is a pointer to another git repository
  - Allows for modular and distributed development :-)
  - Easy to add with the git submodule add <url> path/to/the/submodule :-)
  - Not so easy to remove:  git submodule remove does not exist :-(
- The way the system collects hardware and software components from the submodules and assembles them in the system is the topic of this class.

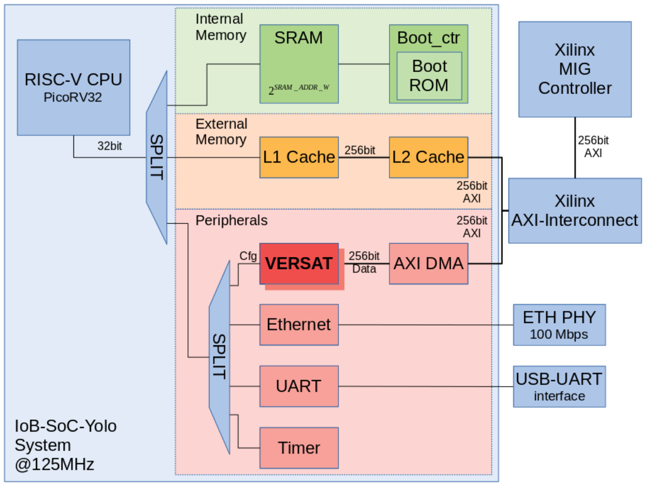
# IOb-SoC: use cases

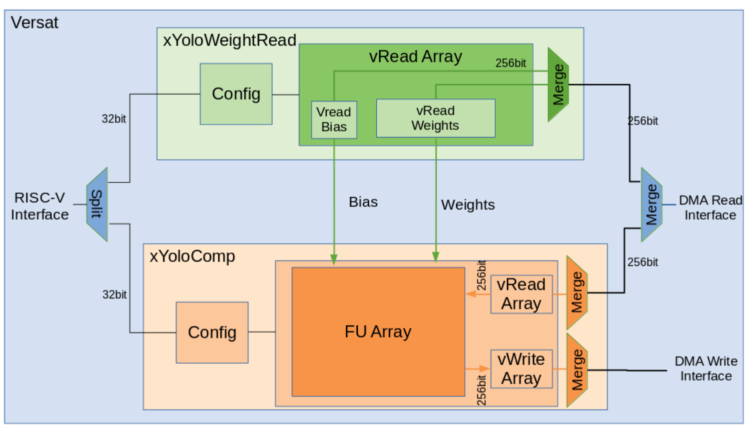
- Tool to develop Complete SoCs
- Tool to develop new processor Peripherals
- Tool to develop new Embedded Software
- Tool to develop interesting **new applications** that rely on **new custom designed hardware blocks**

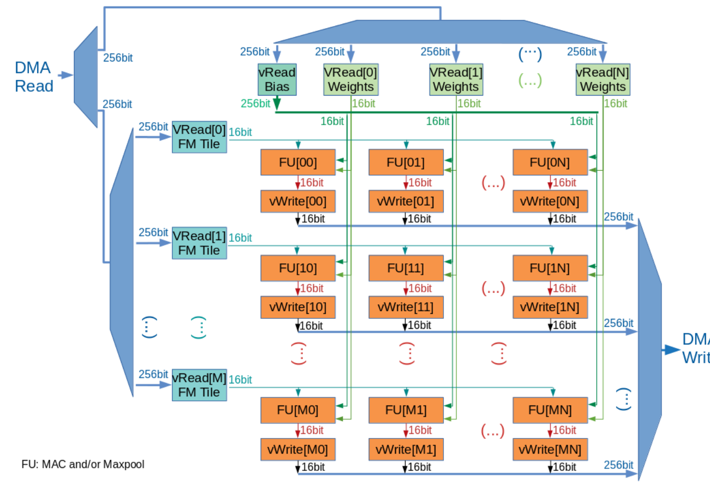
# IOb-SoC: exploiting parallelism

- Moore's (INTEL's co-founder) law is dying:
  - #transistors per $m^2$ is not doubling every 2 years!
  - Frequency of operation is not steadily increasing, it has stagnated
- Huang's (NVIDIA's chief) law is being born:
  - GPU performance is doubling every 2 years!
  - A GPU is a many simple core solution: exploits parallelism in hardware
- Any parallel design can do the same, not just GPU
- AI and Machine Learning provide plenty of opportunities
- Daniel Garigali & Pedro Miranda Yolo V3 implementation!

# IOb-SoC: Yolo V3




- CPU is less than 1% of design
- Me: could you use a more powerful CPU?
- Daniel and Pedro: what for? The bottleneck is not the CPU!
- Two brilliant master theses!

# IOb-SoC: using silicon to do the work

- CPUs have evolved to use silicon for managing purposes rather than doing the  actual work
- Memory management units
- Out of order execution management
- Cache management
- **ALU!**

# IOb-SoC: import components with git submodules

- Easy way to import hardware and respective software components into IOb-SoC
- Add peripheral as git submodule in  the submodules directory
  - *Example: /home/user/sandbox/iob-soc/submodules/TIMER*
- In <u>system.mk</u>, add peripheral name to PERIPHERALS list variable
  - PHERIPHERALS := UART TIMER

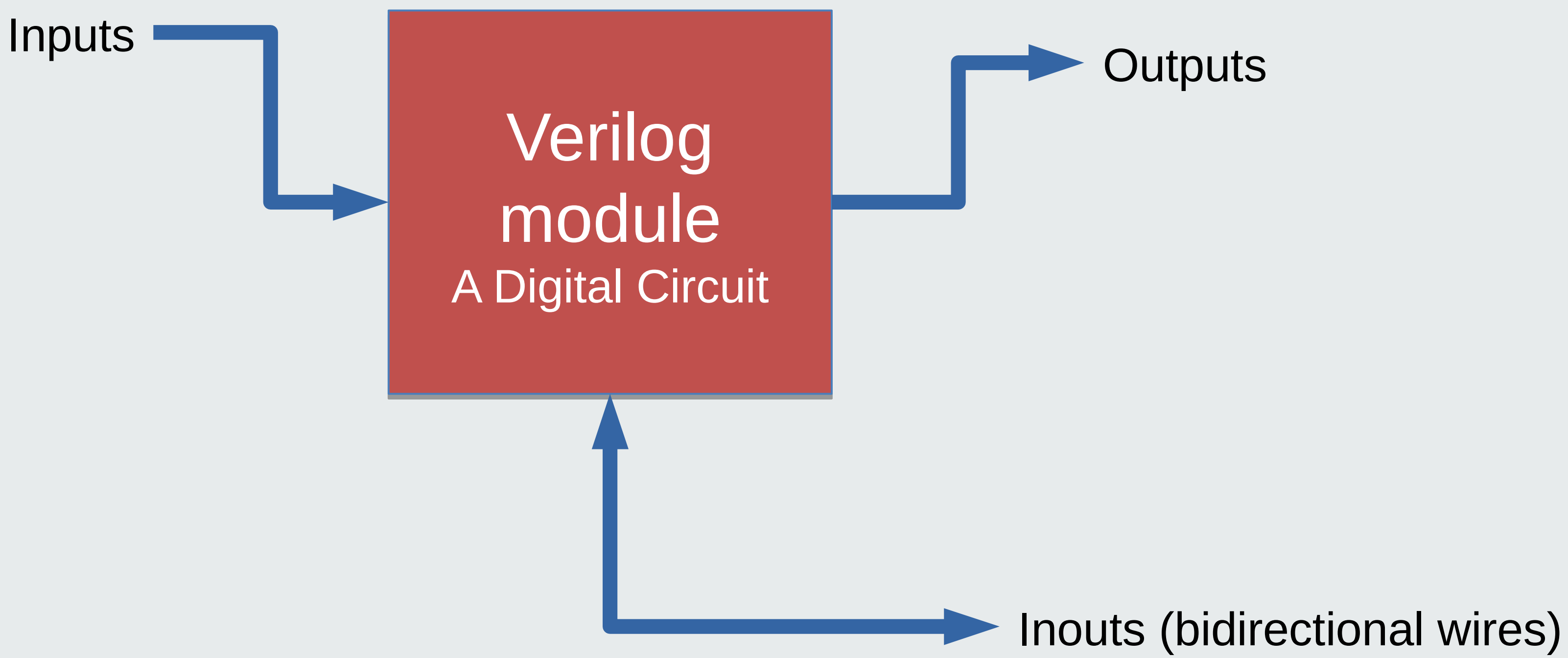
# IOb-SoC: how peripherals are imported

- The Makefile inserts peripherals automatically
- Visit Hardware.mk
- Visit software.mk:
  - Address structure:

    {Pbit, Slave ID, Not Used, Slave Internal Address}
  - Pbit: 1 bit
  - Slave ID: $clog2(N_SLAVES) bits
  - Slave internal address: $clog2(#addresses)

    (TIMER_ADDR_W, UART_ADDR_W)
- What do you do next?
  - Test in simulation: short tests with icarus, longer tests with ncsim, Verilator(!)
  - Test in FPGA: prove in silicon, run longer tests
-

# Verilog module (a digital circuit)



# Verilog module (a circuit)

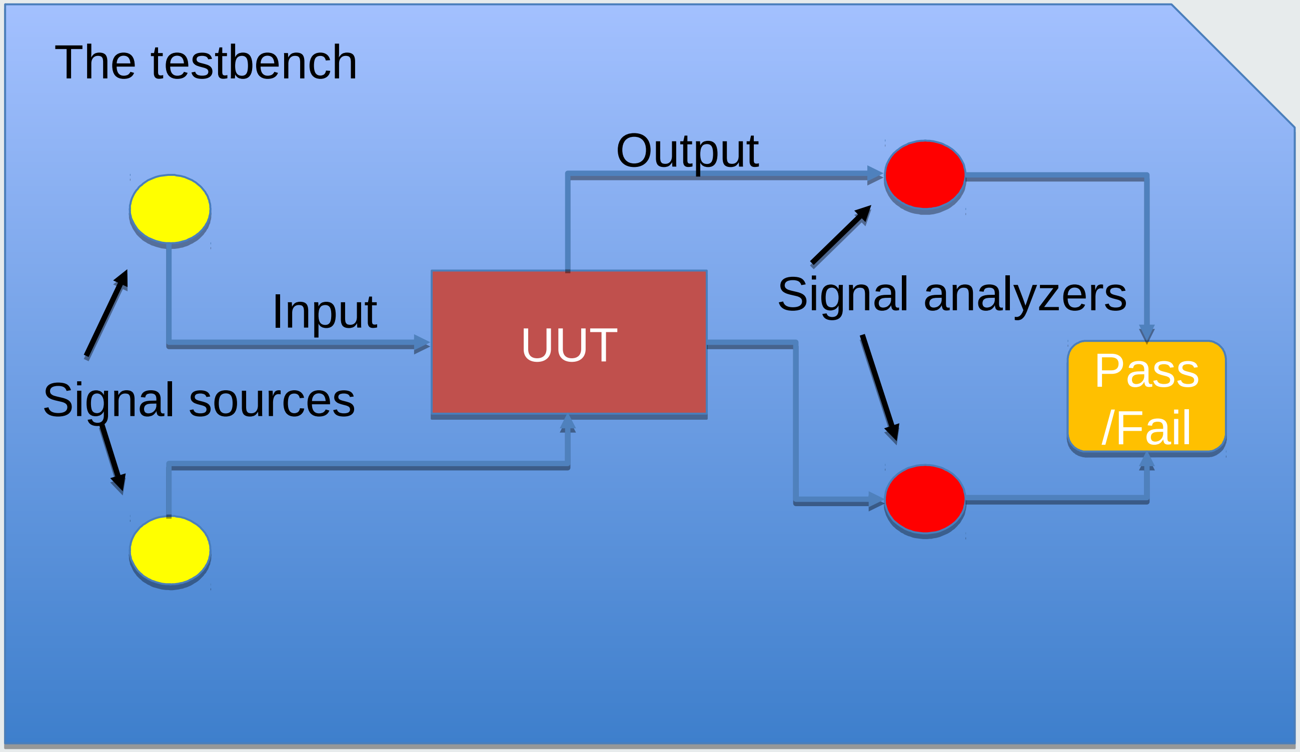
```
module my_unimaginable_peripheral
  (
   input                 clk,
   input                 rst,

   // native bus
   input                                valid, //denotes a valid read or write operation
   input   [`SRAM_ADDR_W-3:0] addr, //address to read or write
   input   [`DATA_W-1:0]             wdata, //data to be written
   input   [`DATA_W/8-1:0]          wstrb, //bytes to be written
   output  [`DATA_W-1:0]             rdata, //data to be read
   output                               ready //read or write completed and ready for next
);

//description of module goes here

endmodule
```

# The testbench directory



# Verilog testbench
# (not a real circuit!)

Used in Verilog simulation!

```
module my_megapower_testbench;

//description of test hardware

//instance of the Unit Under Test

endmodule
```

# Verilog header files

```
`include "my_humble_header_file.vh"
// this like the C language #include

module my_megapower_testbench;

//description of test hardware

//instance of the Unit Under Test

endmodule
```

# Verilog macros

**`define** X 13
// this like the C language #define

**`define** SUM(A,B) A+B
// quite comple macros can be defined

ATTENTION: macros are not functions!

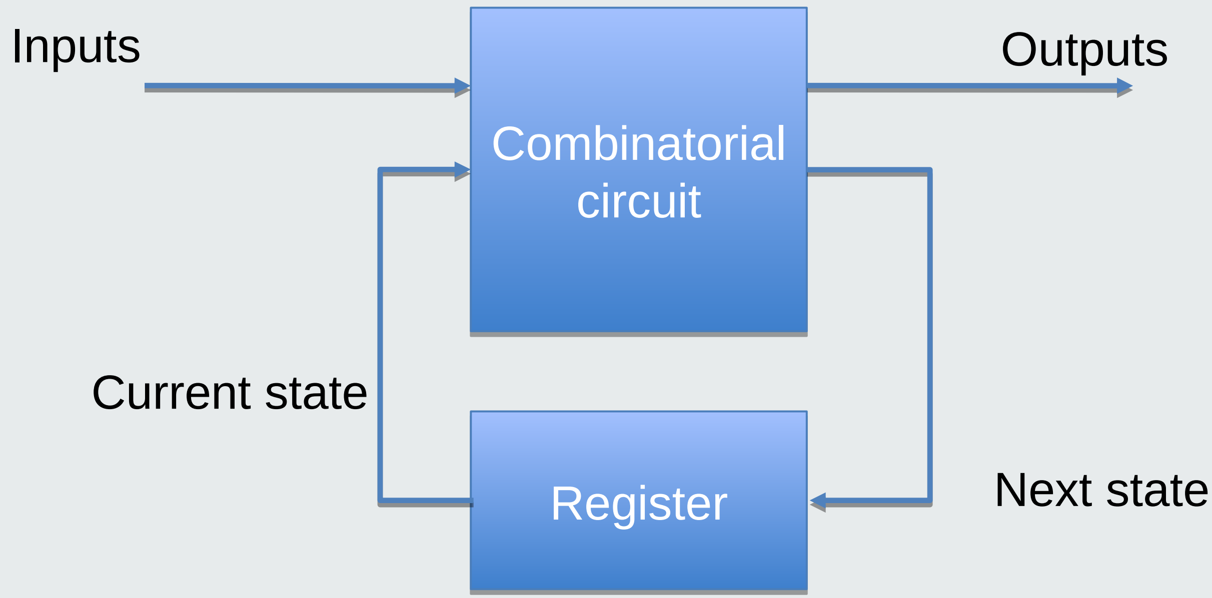
They are just a clever Cut & Paste mechanism

They are handled by the pre-processor that does the Cut & Paste

**assign** C = **`SUM(A,B)**;

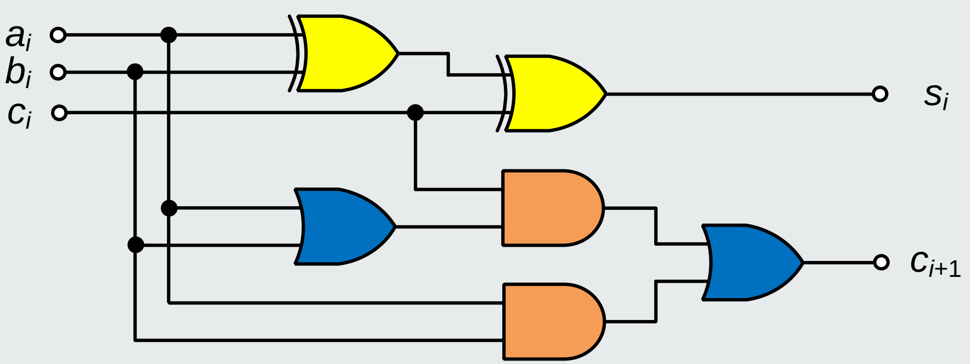
**assign** C = A + B;

# What is inside of digital circuit?



# What is a combinatorial circuit?

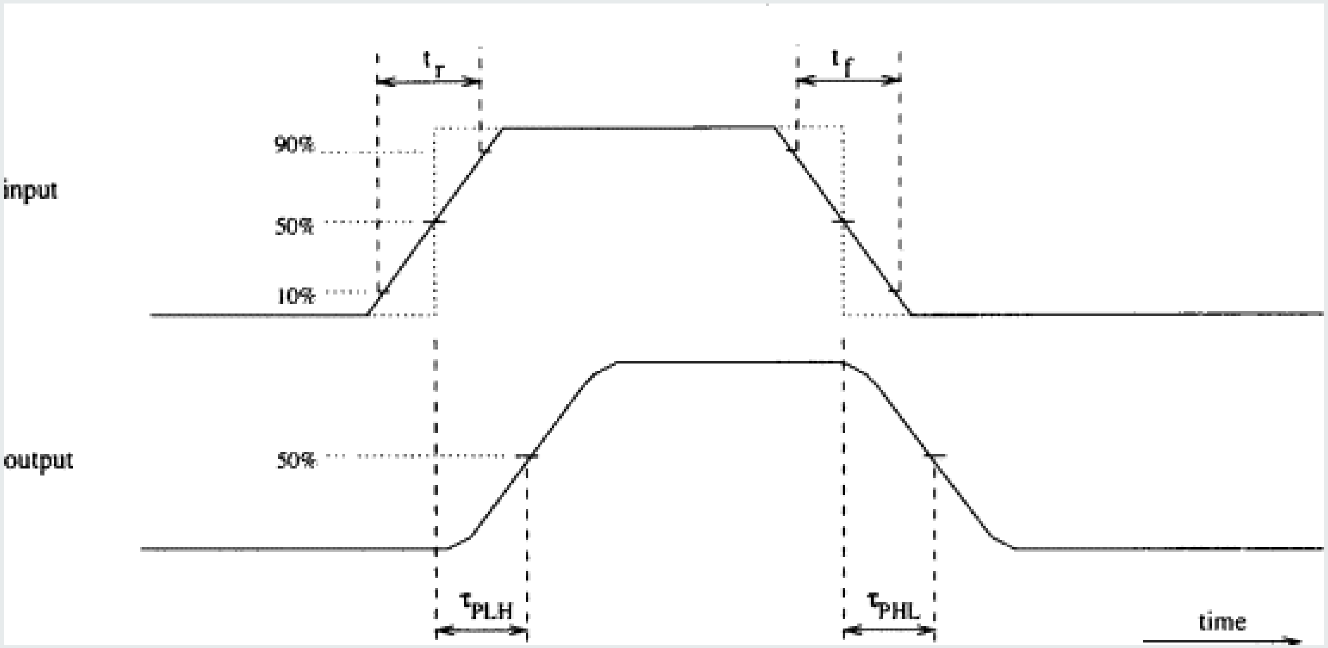
- Full adder circuit
- $S_i = S_i(a_i, b_i, c_i)$;
- $C_i+1=C_i+1(a_i, b_i, c_i)$



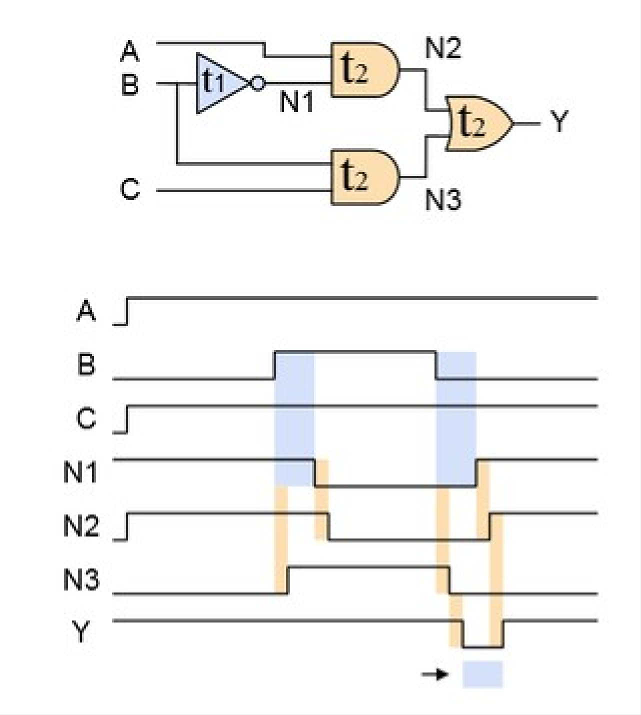

Propagation delays!

Glitches!

# Propagation delay



# Combinational glitches!



# Digital circuits

- Combinational (no feedback paths)
- Sequential (with feedback paths – remembers the past)
  - Asynchronous (no clock signal)
  - Synchronous (with clock signal)
- Asynchronous circuits
  - Very difficult to design: need to account for physical propagation delays and logic glitches
  - Only small designs: latches and flip-flops
- Synchronous circuits
  - Easy to design: propagation delays and glitches are forced out of the equation
    - During the clock period we wait for them to go way!
  - Really large designs: CPUs, GPUs, FPGAs, custom

TÉCNICO LISBOA

# What is a register

- It is a set of memory elements (flip-flops or latches)
- Latches are *transparent,* the output changes with the input if enabled
- Flip-flops are *not transparent*, the output only changes at on of the clock signal edges