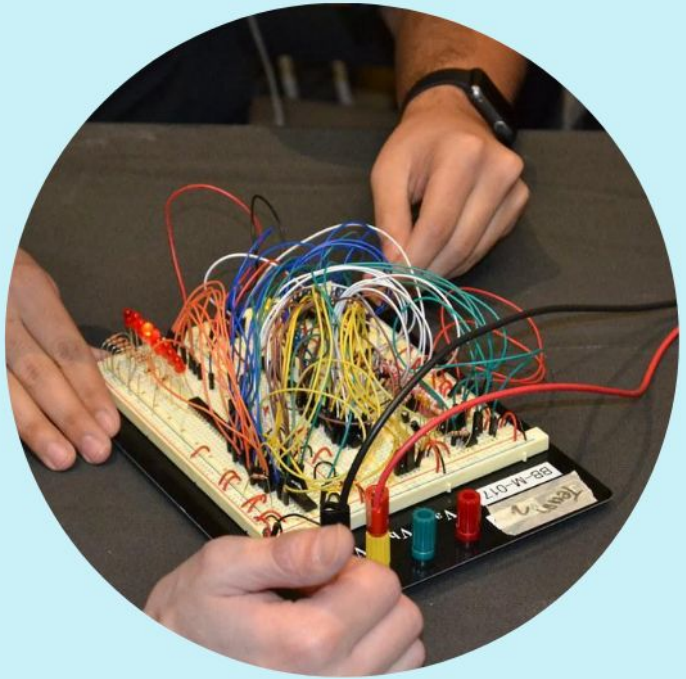
Why Verilog?

What is Electronic Design?

- Process of developing a circuit by using **known** electronic components in order to meet the given **specifications**.
- Specifications: detailed description of desired behavior of the circuit
- Known devices: devices whose behavior can be modeled by known equations or algorithms, with known values of parameters



Hardware Description Languages (HDLs)

- A computer language used to describe the structure and behavior of electronic circuits (usually digital circuits)
- HDL: easy to describe hardware circuits following a particular syntax
- Need for HDLs → Circuits were designed on PCBs, testing and designing of large circuits not easy, slow time to market and complex design flow

➔ **Verilog**

➔ VHDL (Very High Speed Integrated Circuit Hardware Description Lang.)

Why hardware? Why not more software

In some cases ,like IoT devices ,you may want to run algorithms on systems that don’t have a full-fledged computer.

By converting your algorithm into hardware using HDLs, you gain more control over the resources being used, which helps in building efficient, lightweight, and low-power solutions.

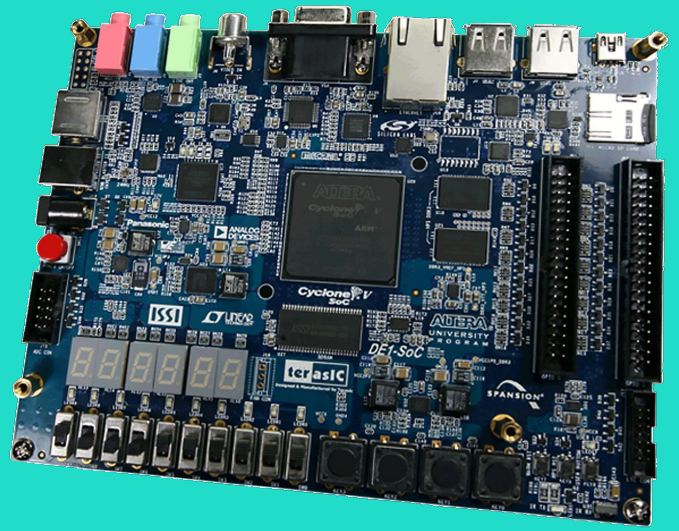
Speed-up with Verilog

Using HDLs like Verilog lets you design hardware that can run parallely i.e. side by side

So some tasks can be moved from software to hardware, which often makes them run faster and more efficiently, like in areas like signal processing, encryption, or real-time control where speed really matters.

So, this Circuit that we describe using a HDL has to either be printed by a factory or be put in a special kind of programmable hardware called FPGA

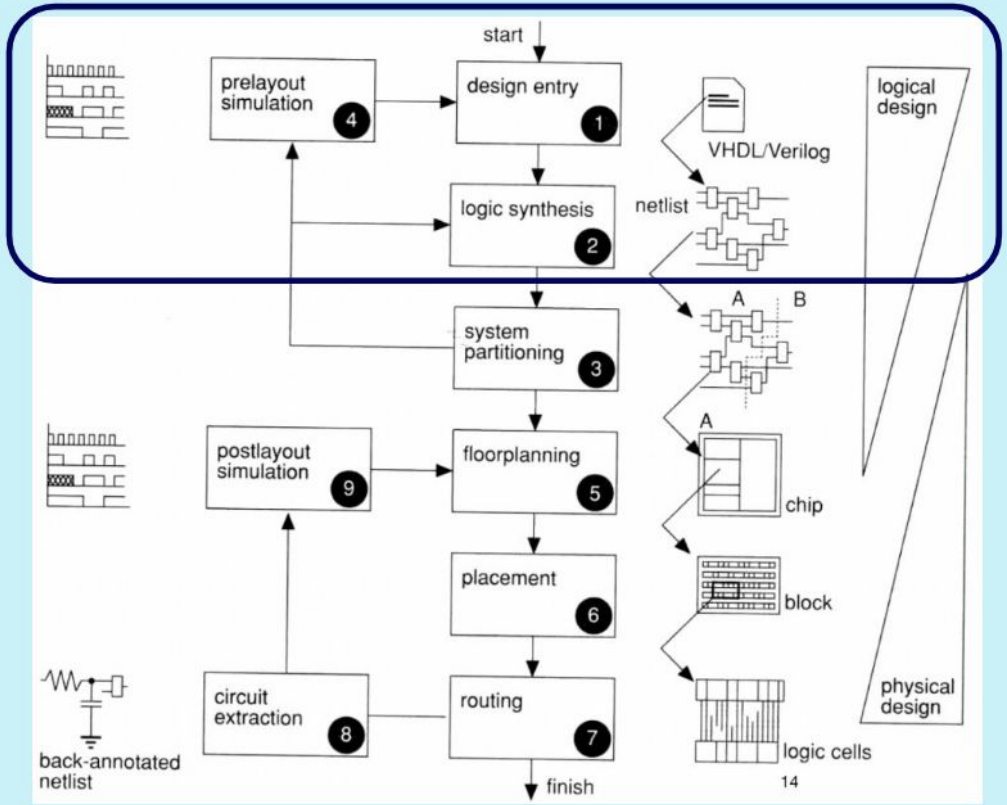
FPGA →



*look at the video linked to see an end to end synthesis

ASIC design flow

What we do as CS students →



Logic Simulation and Synthesis

- Logic Simulation
  - Runs your circuit in computer before you map it to silicon.
  - Essential for ensuring correctness, testing, etc.
- Logic Synthesis
  - Converts your logic described in high-level to a gate-level description => equivalent to a compiler.
  - Register-transfer level (RTL) to gates conversion
  - Such compilation also involves logic minimization

What does verilog code look like?

Verilog code consists of the design for the module itself and a testbench, on the testbench you assign values to the wires of the module and see its behaviour.

For example, here is what a sample module defining an and gate would look like:

```
module and_gate(
    input wire in1,
    input wire in2,
    output wire out
);

    assign out = in1 & in2;
endmodule
```

Verilog testbench

Here is what the testbench could look like:

```
`timescale 1ns/1ns
module tb;
    reg a, b;
    wire y;
    // Instantiate and_gate
    // way 1:
    and_gate uut1 (
        .in1(a),
        .in2(b),
        .out(y)
    );
    initial begin
        $monitor("At time %0t: a = %b, b = %b, y = %b", $time, a, b, y);
        $dumpfile("wave.vcd");
        $dumpvars(0, tb);
        // Test all input combinations
        a = 0; b = 0; #10;
        a = 0; b = 1; #10;
        a = 1; b = 0; #10;
        a = 1; b = 1; #10;
        $finish;
    end
endmodule
```

How to run verilog code?

Assume the testbench is stored in tb_and_gate.v and the module is stored in and_gate.v, you can compile the verilog code using iverilog, and then run it using vvp:

```
$ iverilog -o test tb_and_gate.v and_gate.v
$ vvp test
```

Here is what the output looks like:

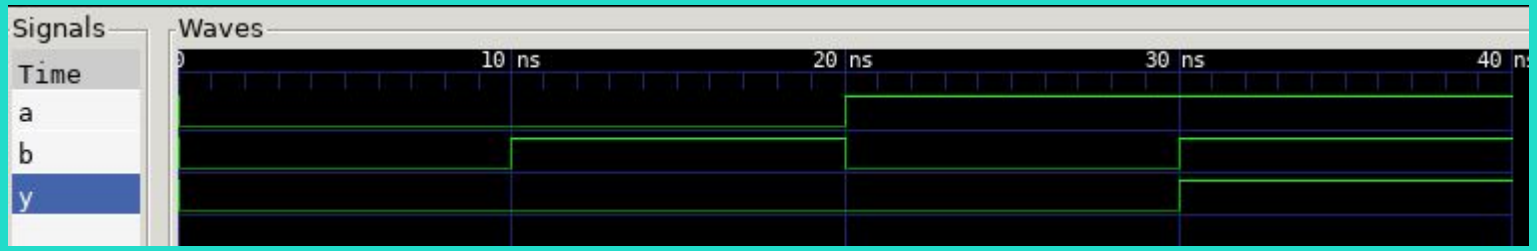
```
VCD info: dumpfile wave.vcd opened for output.
At time 0: a = 0, b = 0, y = 0
At time 10: a = 0, b = 1, y = 0
At time 20: a = 1, b = 0, y = 0
At time 30: a = 1, b = 1, y = 1
```

Visualizing waveforms

You can also visualize the waveform of the verilog code using gtkwave, run

```
$ gtkwave wave.vcd
```

Here is what the waveform looks like:



As you can see, the output y attains a high value (1 value) only when a and b are both 1. Therefore, our code is correct