Politecnico di Torino

From HDL to CMOS Adding Watermarking

An overview on how to apply watermarking to your design

Alessandra Dolmeta - Luigi Giuffrida

What is watermaking?

Watermarking is a technique used to hide a “signature” or identifying mark inside a hardware design (an IP core).

Effect: It cannot stop piracy or overproduction; however, it can help prove that the IP was used.

Constraints:

- It must not alter the functionality of the IP core.
- It must be so lightweight that it is essentially unnoticeable in terms of performance or area.
- It should be hard for an attacker to detect or remove, and yet.
- The owner must still be able to verify it (prove that the unique signature belongs to them).

Watermaking techniques

Static watermarking

- Typically inserted directly into the structural netlist or RTL code or netlist.
- Detectable by reverse engineering (examining the final netlist or GDS).
- Does not require a special test pattern in the field.

Constraint-Based Watermarking

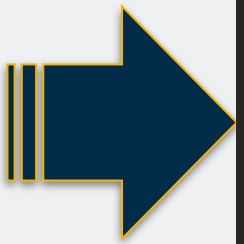
Adds watermark constraints at various steps in the design flow (system, register allocation, synthesis, or physical layout)

- ❑ ***FPGA Example***: Manipulate unused configurable logic blocks to form a recognizable pattern.
- ❑ ***ASIC Example***: Enforce specific cell placements in a subset of the core.

Generally, less intrusive than static watermarking and can be integrated into standard design constraints without altering functionality.

HDL

```
module counter_gen #(
  parameter NBIT = 4
) (
  input                 clk,
  input                 rstn,
  output reg [NBIT-1:0] out
);

  always @ (posedge clk, negedge rstn) begin
    if (! rstn)
        out <= 0;
    else
        out <= out + 1;
  end

endmodule
```



```
module counter_gen
  #(
    parameter NBIT = 4
  )
  (
    input                 clk,
    input                 rstn,
    output reg [NBIT-1:0] out
  );

  always @ (posedge clk, negedge rstn) begin
    if (!rstn)
      out <= 0;
    else
      out <= out + 1;
  end

  (* keep = "true" *) reg [2:0] watermark_shift = 3'b101;

  always @ (posedge clk or negedge rstn) begin
    if (!rstn) begin
      watermark_shift <= 3'b101;
    end else begin
      watermark_shift <= {watermark_shift[1:0], ~watermark_shift[2]};
    end
  end

  (* keep = "true" *) wire watermark_detect = (watermark_shift == 3'b111);

endmodule
```

Behavioural description of the circuit

HDL

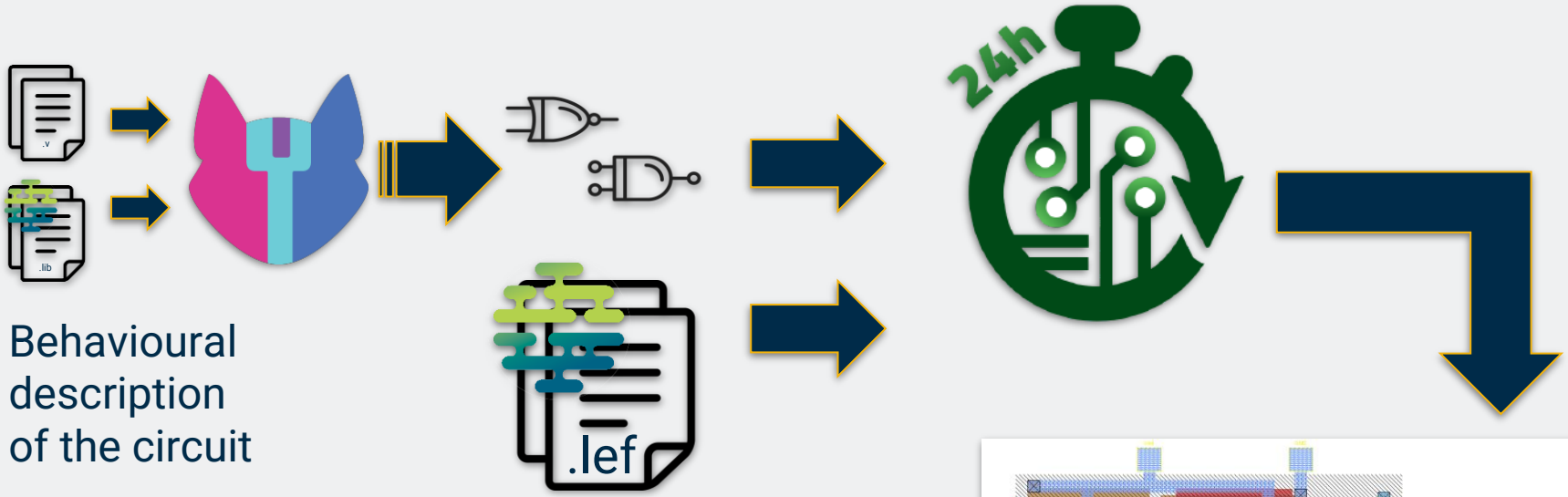
The extra **shift register** serves no essential role in the counter's behavior.

It is purely a "*dummy logic*" block, generating a small pattern that can be spotted later in the synthesized or placed-and-routed design.
That means it behaves like a hidden watermark or signature.

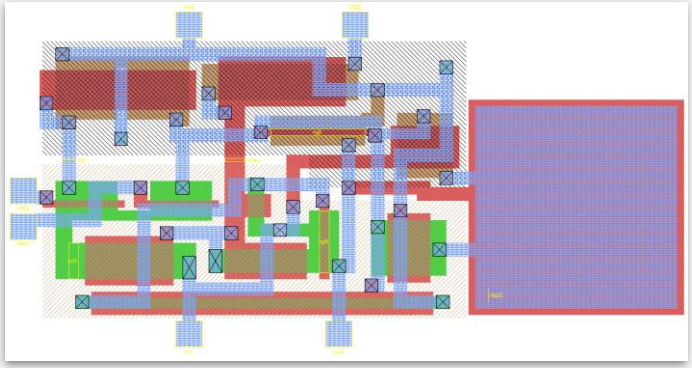
The **keep** attributes are used to prevent synthesis from discarding it.

```
module counter_gen
  #(
    parameter NBIT = 4
  )
  (
    input               clk,
    input               rstn,
    output reg [NBIT-1:0] out
  );

  always @ (posedge clk, negedge rstn) begin
    if (!rstn)
      out <= 0;
    else
      out <= out + 1;
  end

  (* keep = "true" *) reg [2:0] watermark_shift = 3'b101;

  always @ (posedge clk or negedge rstn) begin
    if (!rstn) begin
      watermark_shift <= 3'b101;
    end else begin
      watermark_shift <= {watermark_shift[1:0], ~watermark_shift[2]};
    end
  end

  (* keep = "true" *) wire watermark_detect = (watermark_shift == 3'b111);

endmodule
```

Flow overview - recap



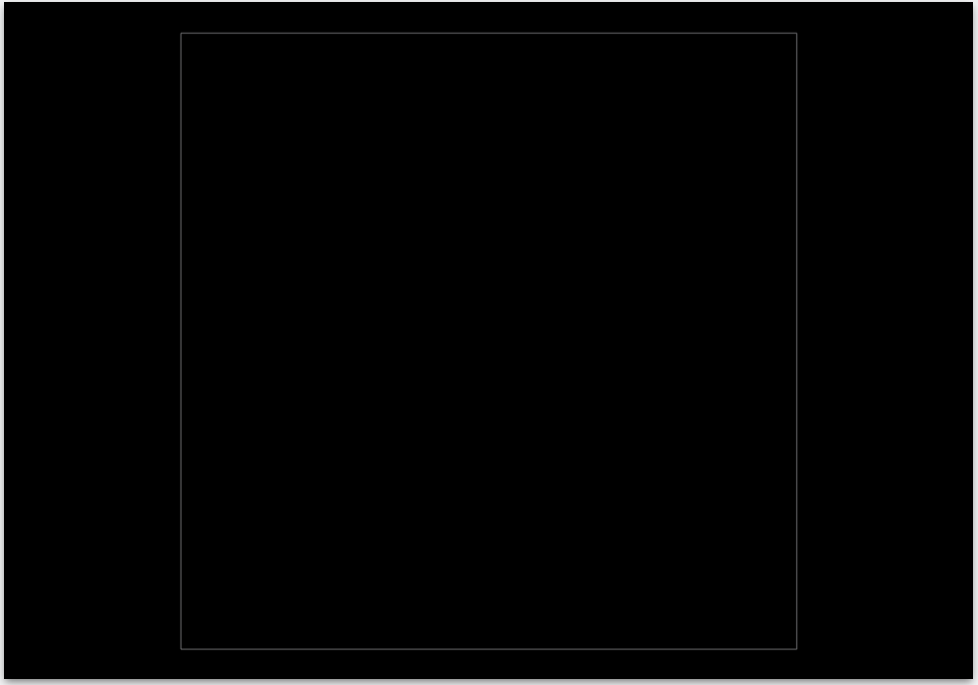
Behavioural description of the circuit



Physical implementation and electrical optimization of the circuit

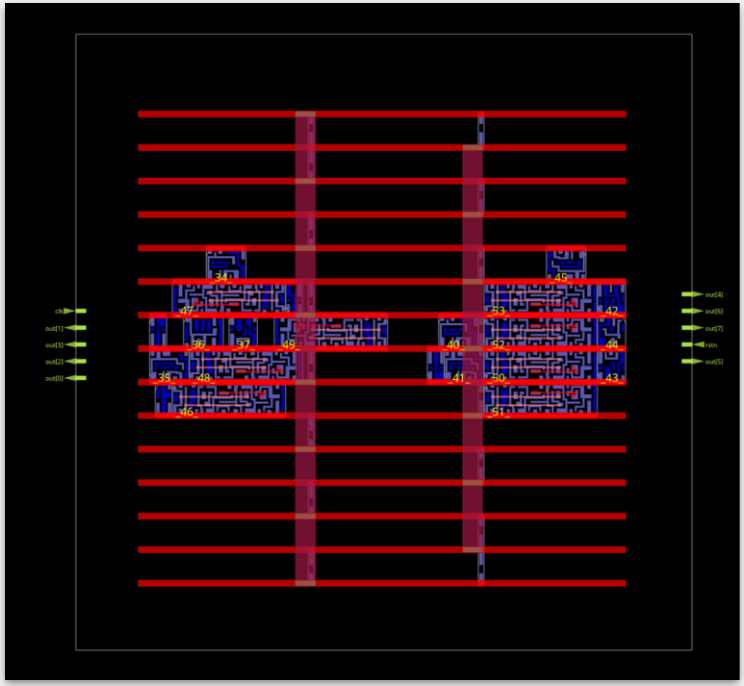
Floorplanning

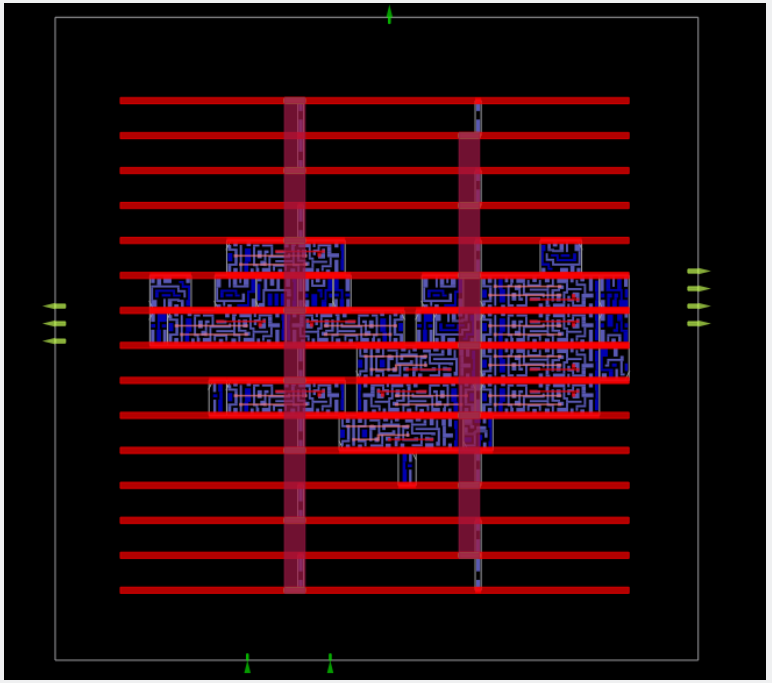
Floorplanning defines the area of IC and where the submodules of the design should be placed.



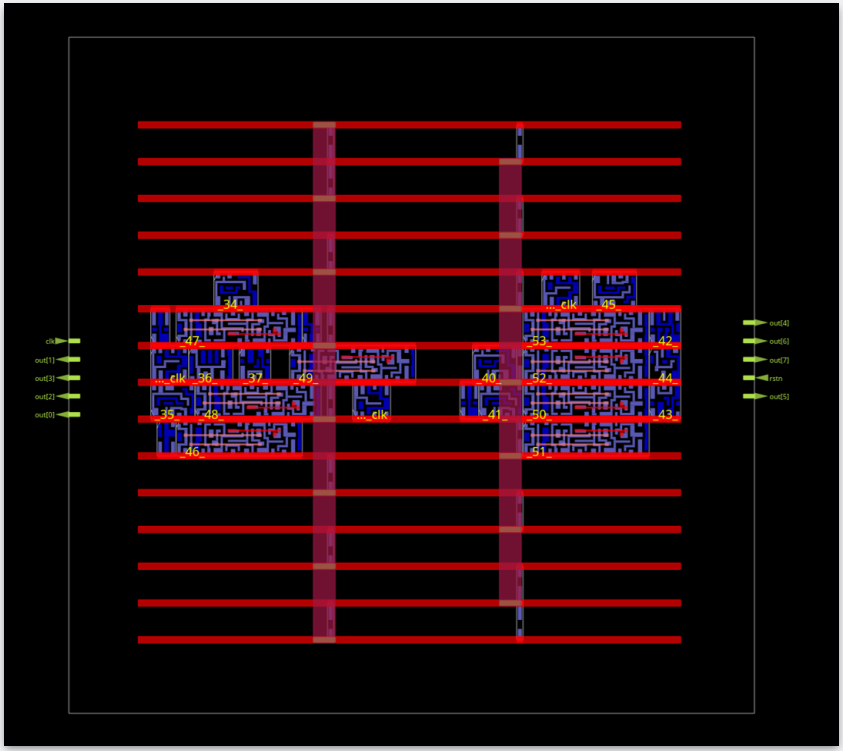
No explicit floorplanning in the example!

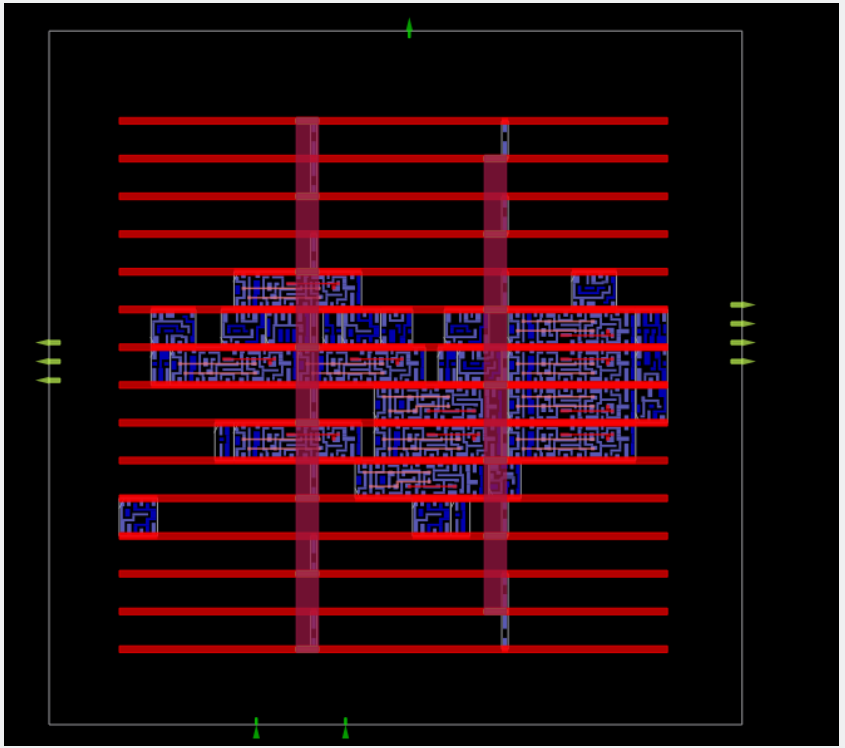
Placement



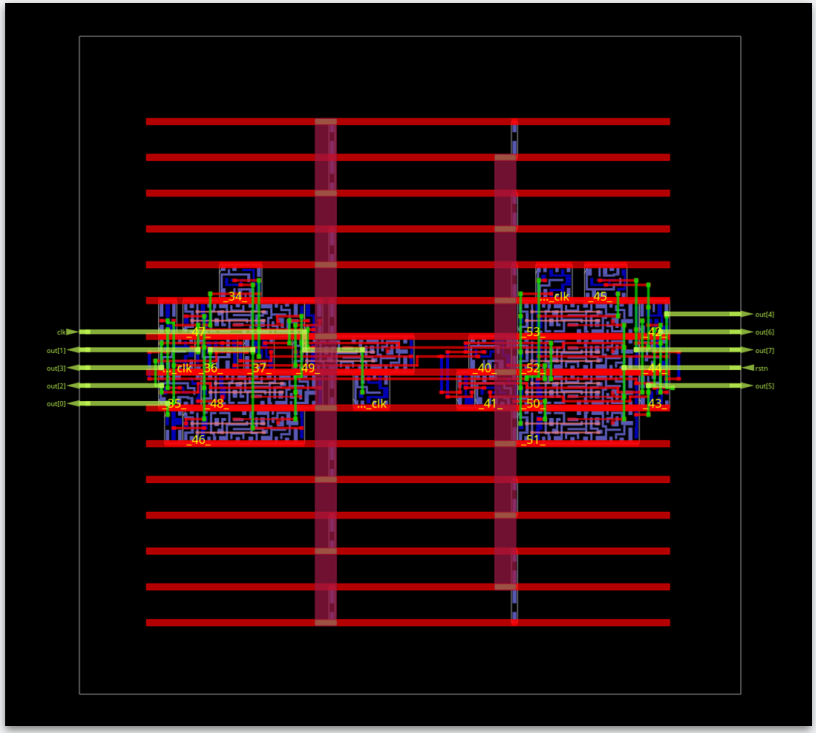


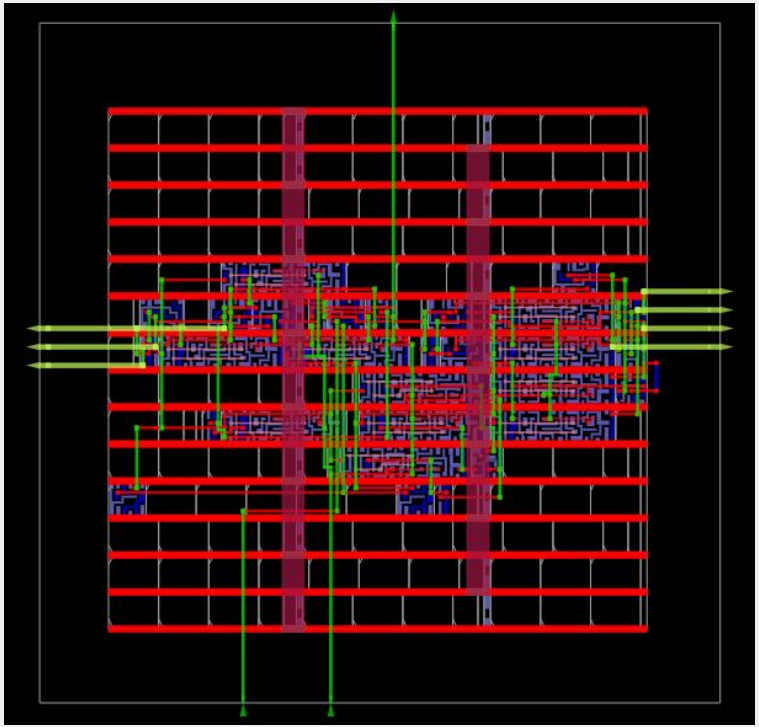
Clock tree synthesis





Routing





Check the differences!

Open the **lab2_description.pdf** you find in the Material, and follow the steps.

You can reconstruct the project, starting from the same folders structure you had last week.

Remember: substitute in the src folder, **counter_gen.v** file with the new one shown in the lab2_description.

- **Cell Placement**: How does the watermark logic affect the distribution of cells in the floorplan—are extra cells clustered or dispersed?
- **Wirelength & Displacement**: What changes in total/average displacement can arise, and how might the placer shift existing cells to fit watermark logic?
- **Timing & Constraints**: Does inserting a few extra cells alter timing paths or constraints significantly, and how might this affect placement/routing decisions?

- Can you visually spot watermark logic in the layout, and do you see any grouping differences compared to the original design?

THANK YOU FOR YOUR ATTENTION!

ANY QUESTION? FEEL FREE TO ASK!

TIME TO EXPERIMENT!

OPEN THE LAB PDF AND FOLLOW THE GUIDE TO TRY IT ON YOUR OWN!