



Hochschule
München
University of
Applied Sciences

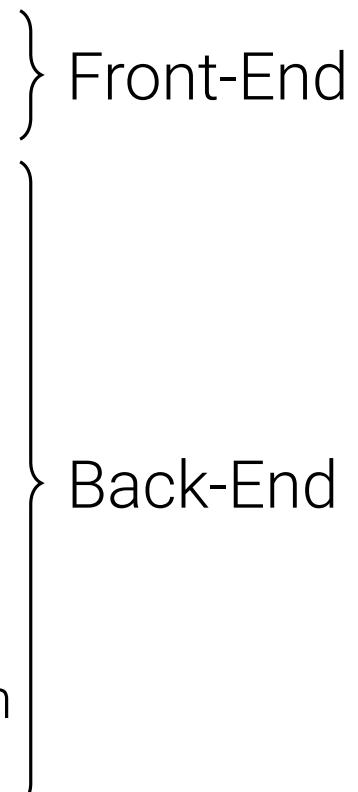
CHIP DESIGN & IMPLEMENTATION

From RTL to GDSII

Daniel Arevalos

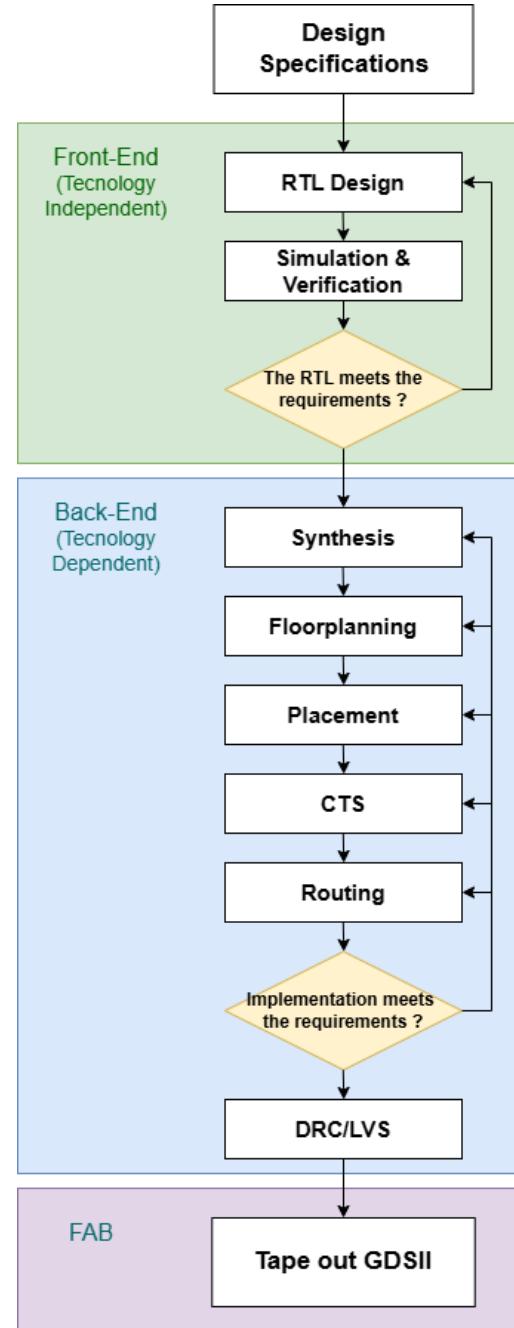
Fakultät 07 – Hochschule München

Table of Contents

- 1. Introduction
 - 2. RTL Design
 - 3. Simulation & Formal Verification
 - 4. Physical Implementation
 - 4.1 Synthesis
 - 4.2 Floorplanning
 - 4.3 Placement
 - 4.4 CTS
 - 4.5 Routing
 - 5. Signoff & Physical Verification
 - 5.1 Timing Signoff
 - 5.2 Layout (Physical) Verification
 - 5.3 Chip Finishing
 - 6. Tapeout & Fabrication
- 

Introduction

- ▶ Digital IC design follows a structured flow from high-level specifications to fabrication.
- ▶ The flow is divided into Front-End (technology-independent) and Back-End (technology-dependent) stages.
- ▶ Front-End focuses on functionality and correctness (RTL design and verification).
- ▶ Back-End transforms the RTL into a physical layout that meets timing, power, and manufacturing constraints.
- ▶ The process is iterative, with verification and checks at multiple stages before tape-out.



Front-End

RTL Design

- ▶ Describes the hardware behavior using a register-transfer level (RTL) abstraction.
- ▶ Technology-independent and focused on functionality, not physical implementation.
- ▶ Serves as the main input for simulation, verification, and synthesis.
- ▶ **Module, Body, Signal Declaration, Instantiation, Combinational Logic**

```
module top #(  
parameter int Width = 16  
)  
(  
input logic clk_i,  
input logic rst_ni,  
input logic mode_i,  
input logic [Width-1:0] data_in_i,  
output logic [Width-1:0] result_o  
);
```

```
// Declare signals to be used in the module  
logic [Width-1:0] first, second;  
logic [Width-1:0] combine_and, combine_or;  
// instantiate two blocks with different names i_reg_1 and frank  
ffs #(.Width(Width)) i_reg_1 (  
.clk_i(clk_i), .rst_ni(rst_ni), .in_i(data_in_i), .out_o(first));  
ffs #(.Width(Width)) frank (  
.clk_i(clk_i), .rst_ni(rst_ni), .in_i(first), .out_o(second));  
// combine outputs  
assign combine_and = first & second ;  
assign combine_or = first | second ;  
// assign result  
assign result_o = mode_i ? combine_and : combine_or ;  
  
endmodule // top
```

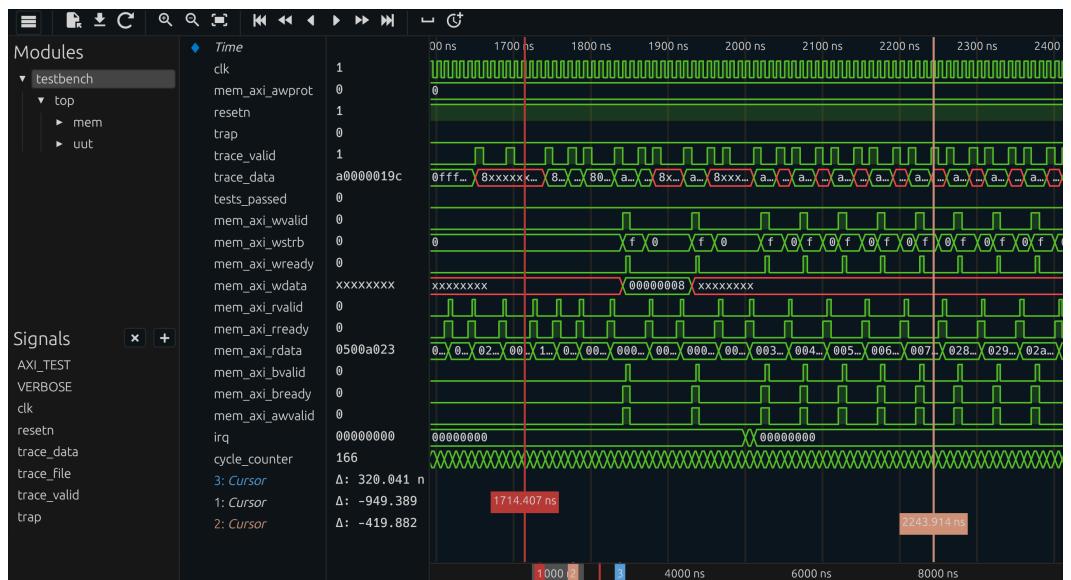
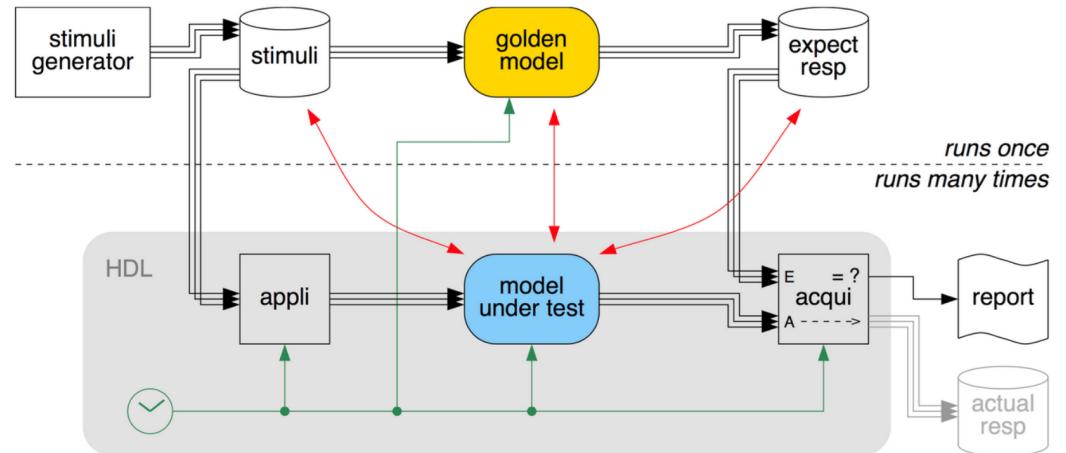
More Information on: Refresher on SystemVerilog



Simulation & Formal Verification

- ▶ Validates RTL functionality against the design specifications.
- ▶ Simulation checks behavior using testbenches and input stimuli.
 - ▶ Applies stimuli to the Design Under Test (DUT) using testbenches.
 - ▶ Compares DUT outputs against a golden reference model.
 - ▶ Detects functional mismatches and reports errors.
- ▶ Formal verification proves correctness using mathematical models, without test vectors.

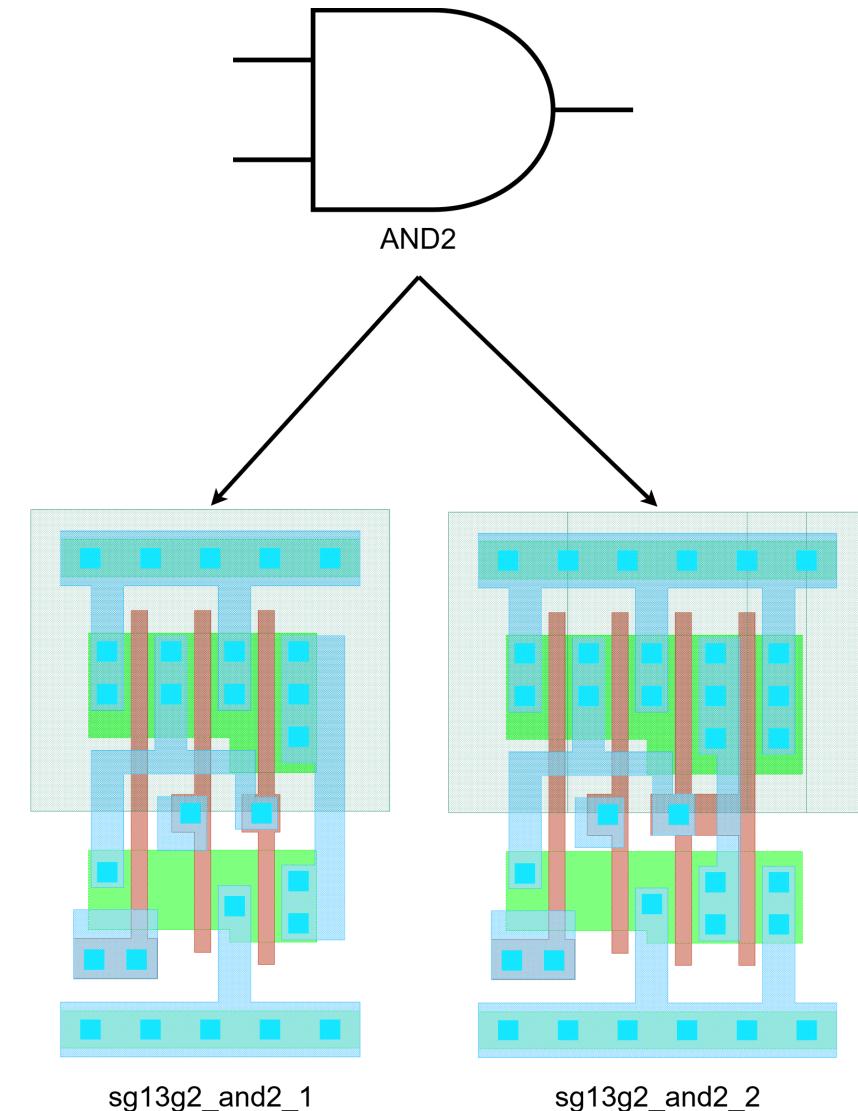
More Information on: Simulation



Back-End

Synthesis - Standard Cells

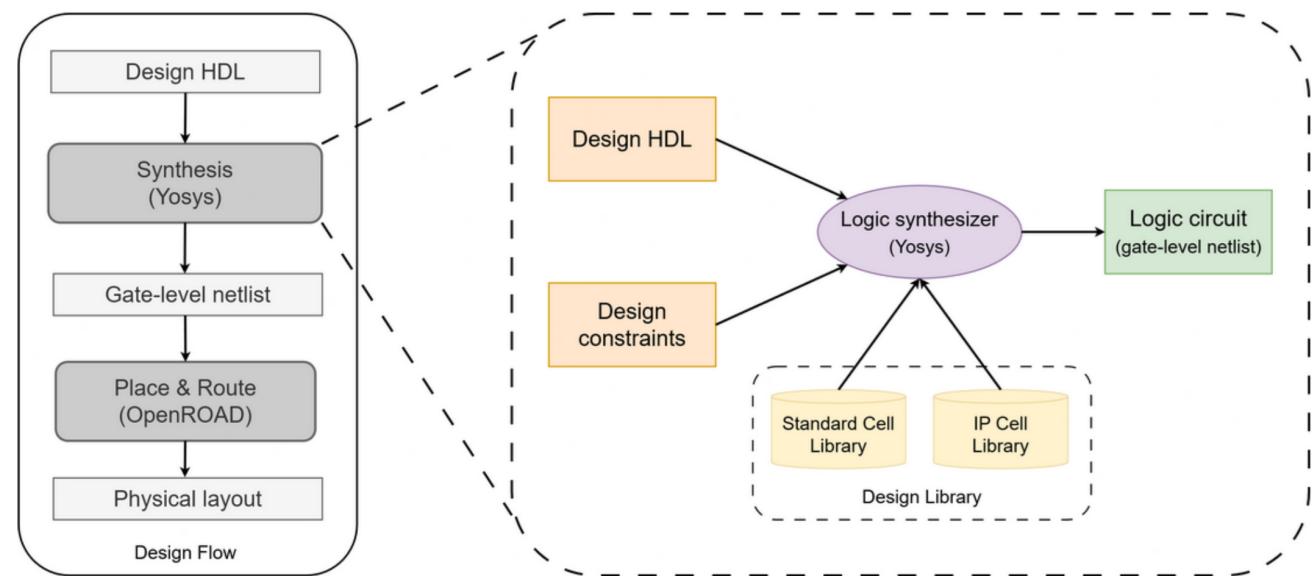
- ▶ Pre-designed and pre-characterized logic building blocks.
- ▶ Provided by the foundry or PDK as a standard cell library.
- ▶ Characterized for timing, power, and area.
- ▶ Used by synthesis tools to map RTL into hardware.



More Information on: Synthesis

Synthesis - Inputs & Outputs

- ▶ Inputs: RTL design (HDL) and design constraints.
- ▶ Uses standard cell and IP libraries for mapping.
- ▶ Translates RTL into a gate-level netlist.
- ▶ Preserves functional behavior while meeting timing and area constraints.



More Information on: Synthesis

Synthesis - How it looks in Practice?

```
module example (
    input wire clk,
    input wire rst,
    output wire y
);
    assign y = clk & ~rst;
endmodule
```

RTL code

More Information on: Synthesis

```
module fulladd(a, b, c_in, c_out, sum);
    wire _00_;
    wire _01_;
    wire _02_;
    ...
    input [3:0] a;
    wire [3:0] a;
    ...
    assign _14_ = _07_ & ~(_13_);
    assign _15_ = _06_ & ~(_14_);
    assign c_out = _15_ | _04_;
    assign sum[0] = ~(_10_ ^ c_in);
    assign sum[1] = ~(_12_ ^ _08_);
    assign sum[2] = _14_ ^ _05_;
    assign _16_ = ~(_14_ | _05_);
    assign _17_ = _16_ | ~(_02_);
    assign sum[3] = _17_ ^ _01_;
endmodule
```

Netlist after initial synthesis

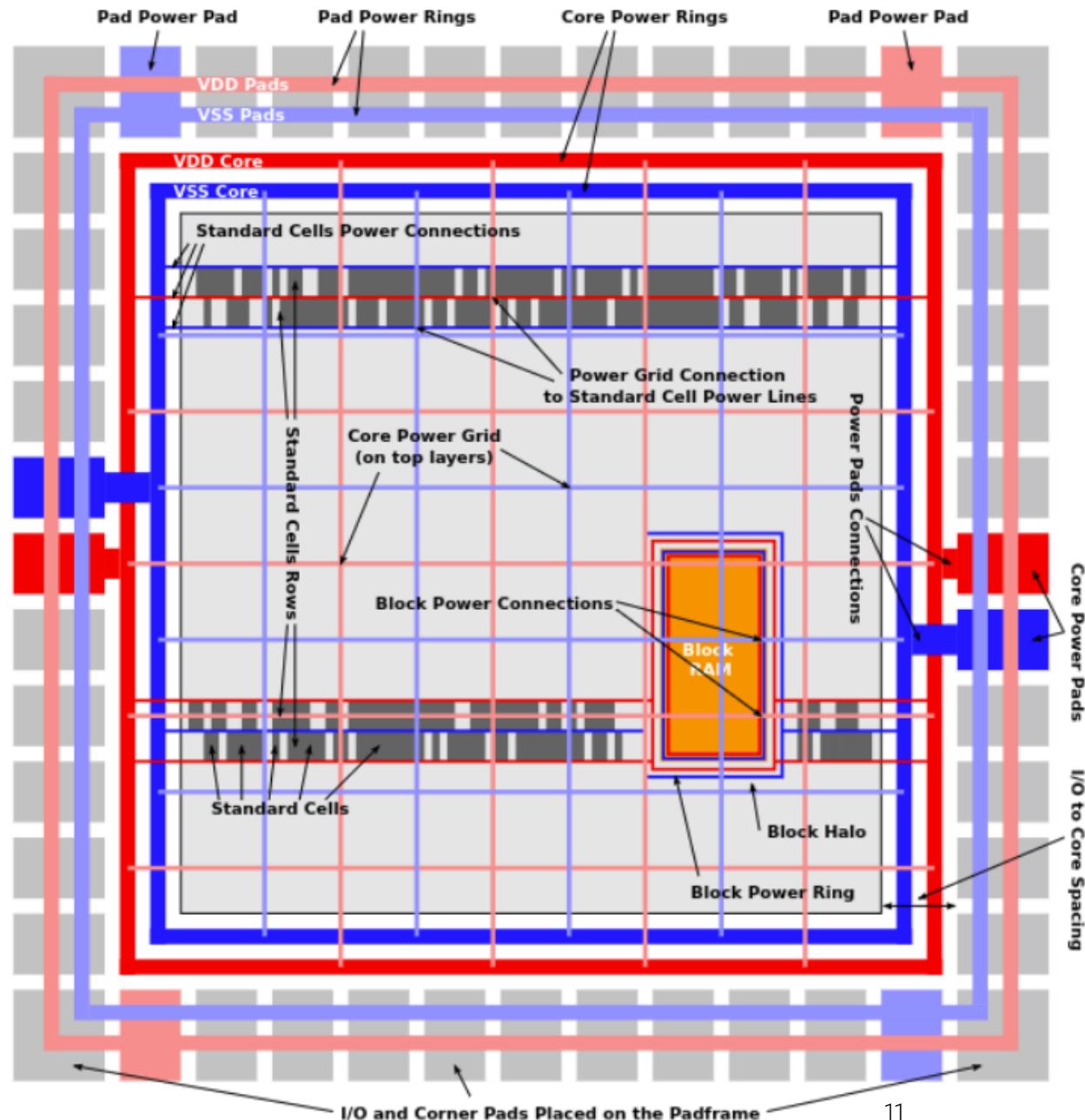
```
module fulladd(a, b, c_in, c_out, sum);
    wire _00_;
    wire _01_;
    wire _02_;
    ...
    output [3:0] sum;
    wire [3:0] sum;
    sg13g2_xor2_1 _61_ (
        .A(_25_), .B(_21_), .X(_42_)
    );
    sg13g2_o21ai_1 _62_ (
        .A1(_40_), .A2(_41_), .B1(_28_), .Y(_27_)
    );
    sg13g2_xnor2_1 _63_ (
        .A(_26_), .B(_35_), .Y(_43_)
    );
    sg13g2_xnor2_1 _64_ (
        .A(_36_), .B(_38_), .Y(_44_)
    );
    sg13g2_xnor2_1 _65_ (
        .A(_31_), .B(_39_), .Y(_45_)
    );
    ...
endmodule
```

Netlist after technology mapping

Floorplanning

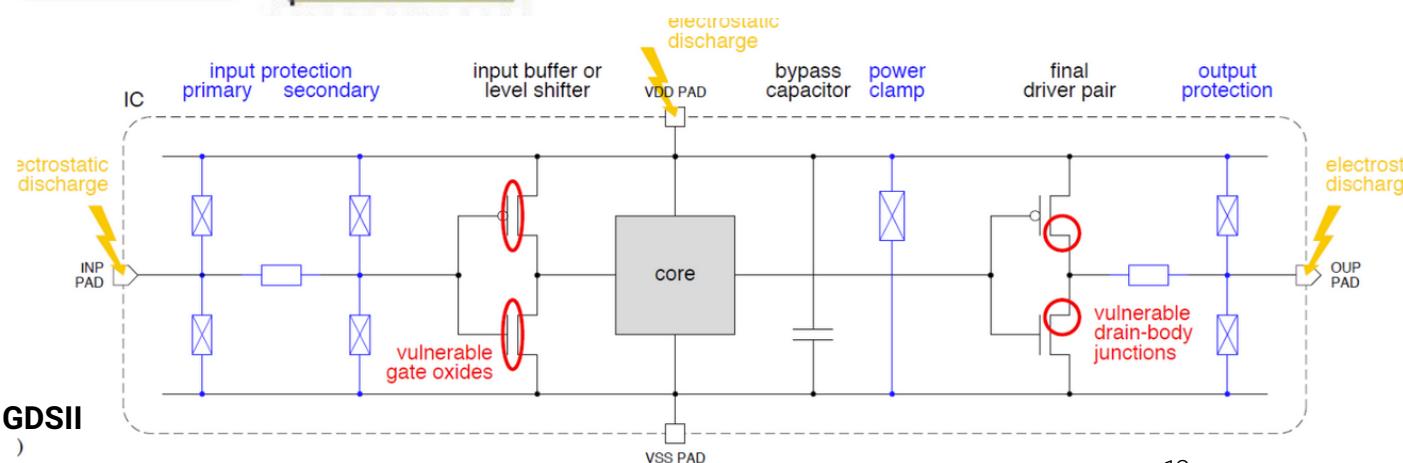
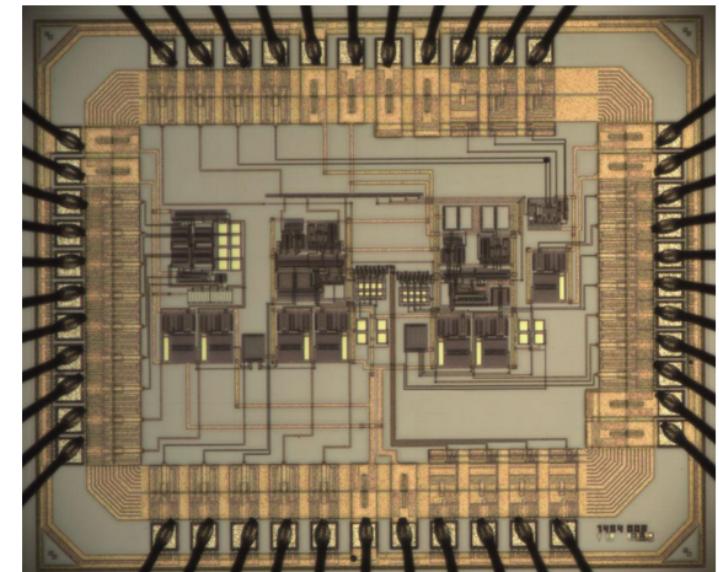
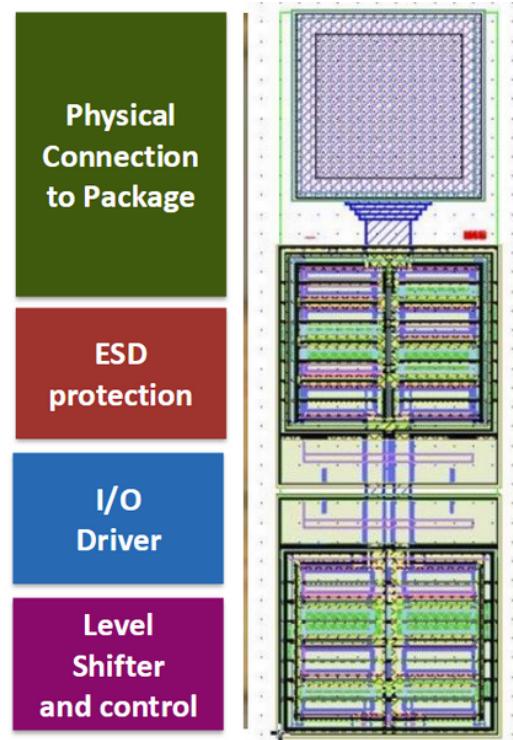
- ▶ The unit cost of IC is directly proportional to the silicon area.
- ▶ The netlist gives us the net cell area needed, but:
 - ▶ Additional area is required to bring power and timing/clock signals to all cells.
- ▶ I/O and packaging constraints must be considered.
- ▶ Macros (RAM, PLL, etc.) must be placed.

More Information on: Floorplanning



Floorplanning - Padrинг

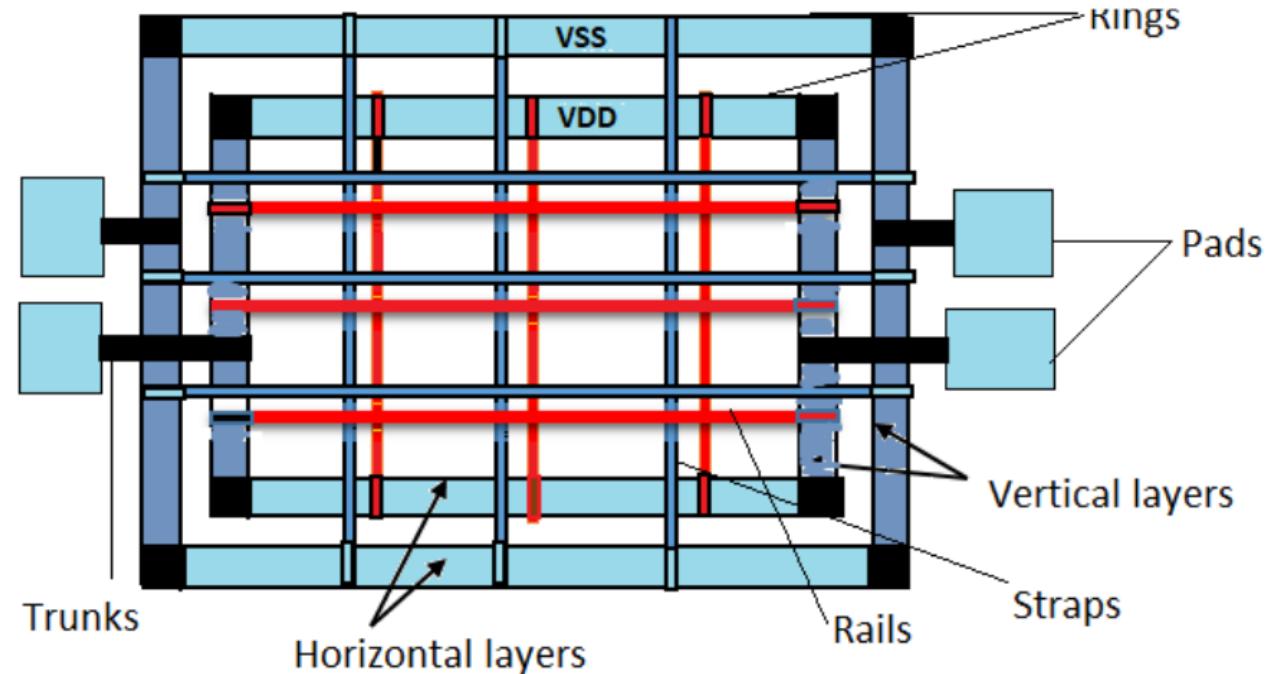
- ▶ Physical connection to package.
- ▶ Electro Static Discharge (ESD) protection.
 - ▶ Make sure that electrical discharge does not damage internal circuitry.
- ▶ I/O Drivers.
- ▶ Types of I/O Cells.
 - ▶ Digital I/O Buffers.
 - ▶ Analog I/O Cells.
 - ▶ Power/Ground Supply Pads.



More Information on: Floorplanning

Floorplanning - Power Grid

- ▶ Power Rings. Distribute power from pads to core area.
- ▶ Power Stripes. Distribute power within core area.
- ▶ Power PADs. Connect power grid to standard cells.

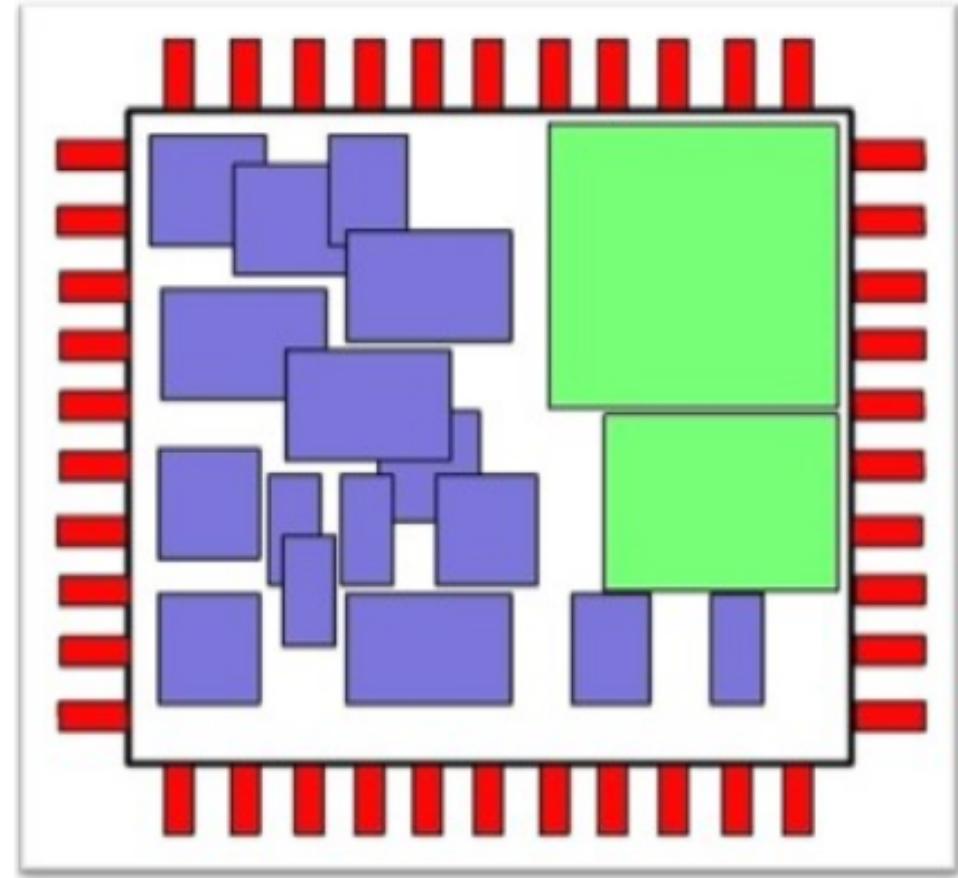


More Information on: Floorplanning

Global Placement

- ▶ Distributes standard cells across the core using approximate positions.
- ▶ Optimizes wirelength and congestion at a global level.
- ▶ Produces a continuous placement (cells may overlap or be non-legal).

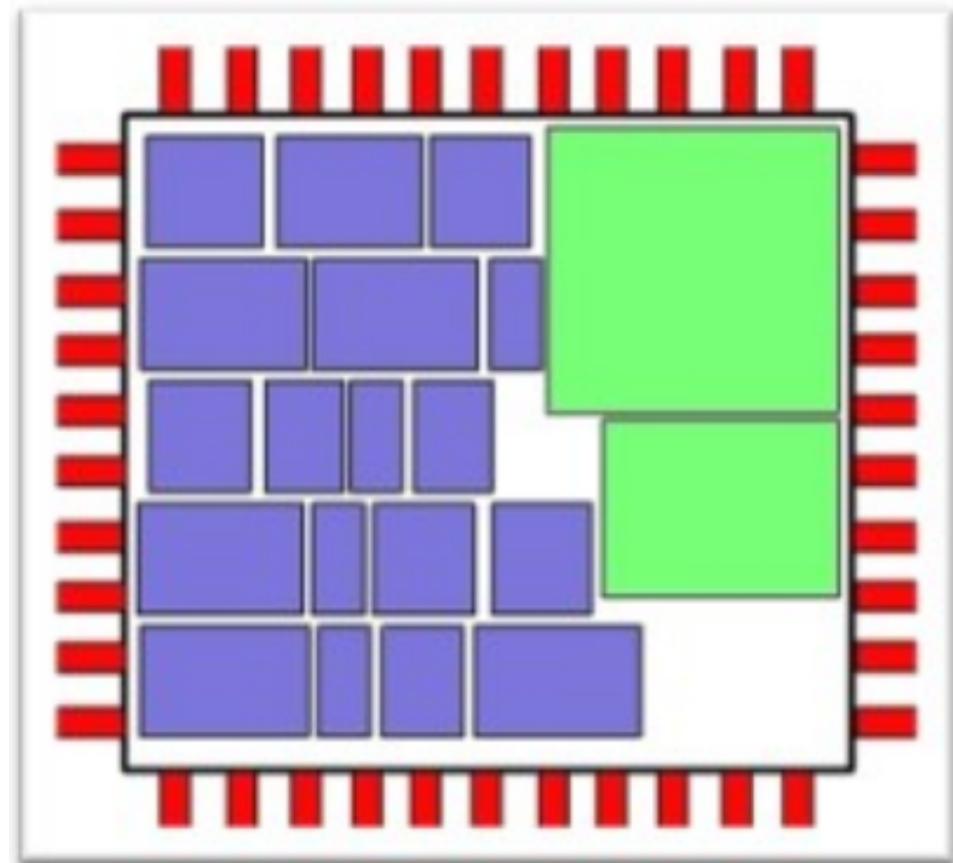
More Information on: Placement



Detailed Placement

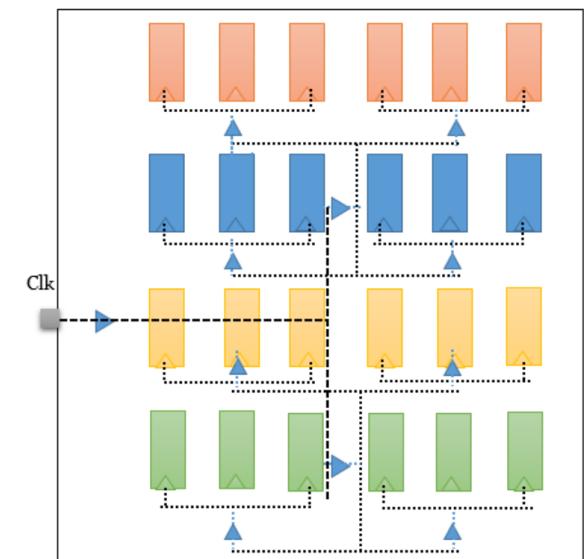
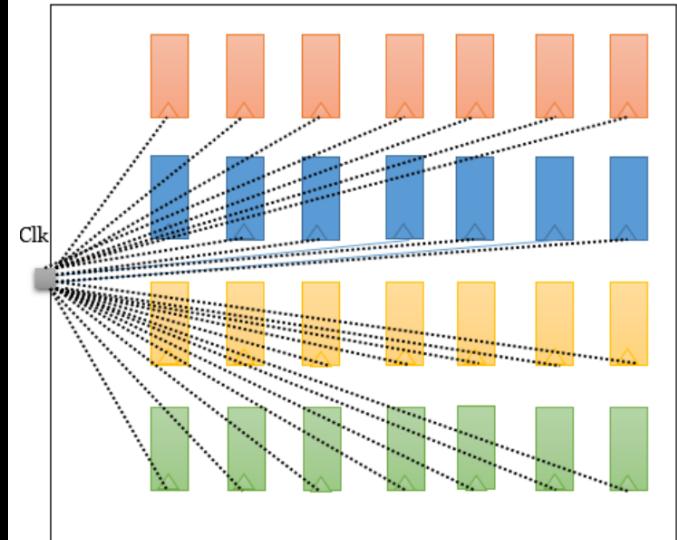
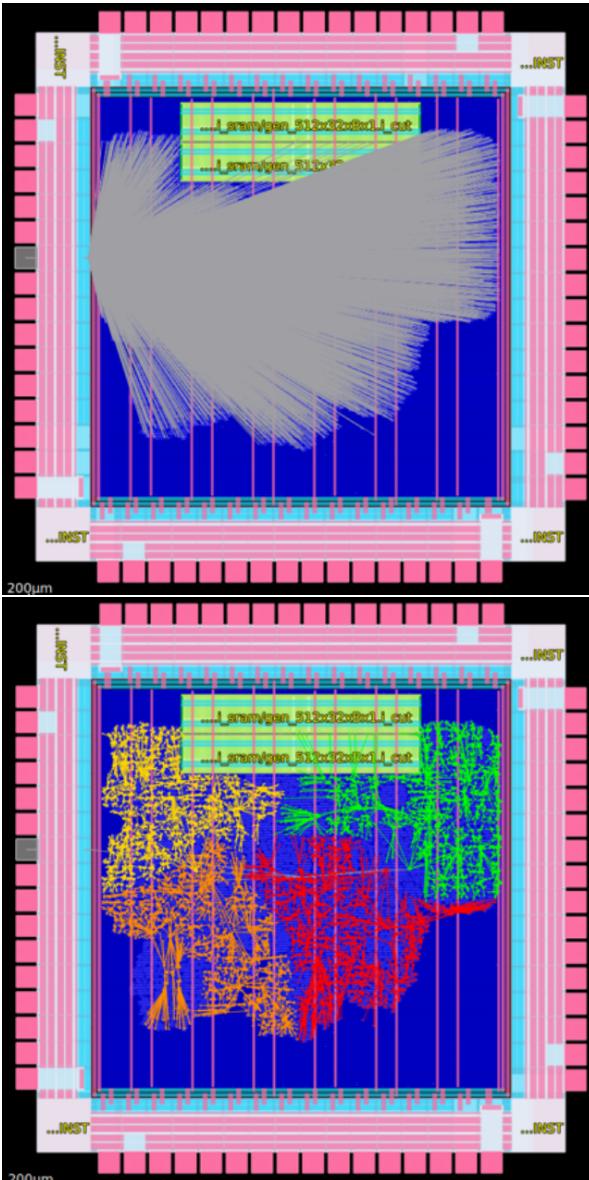
- ▶ Also known as legalization.
- ▶ Takes the global placement and fixes exact cell locations on rows.
- ▶ Performs local optimizations (shifts, swaps) to improve wirelength and timing while respecting rules.

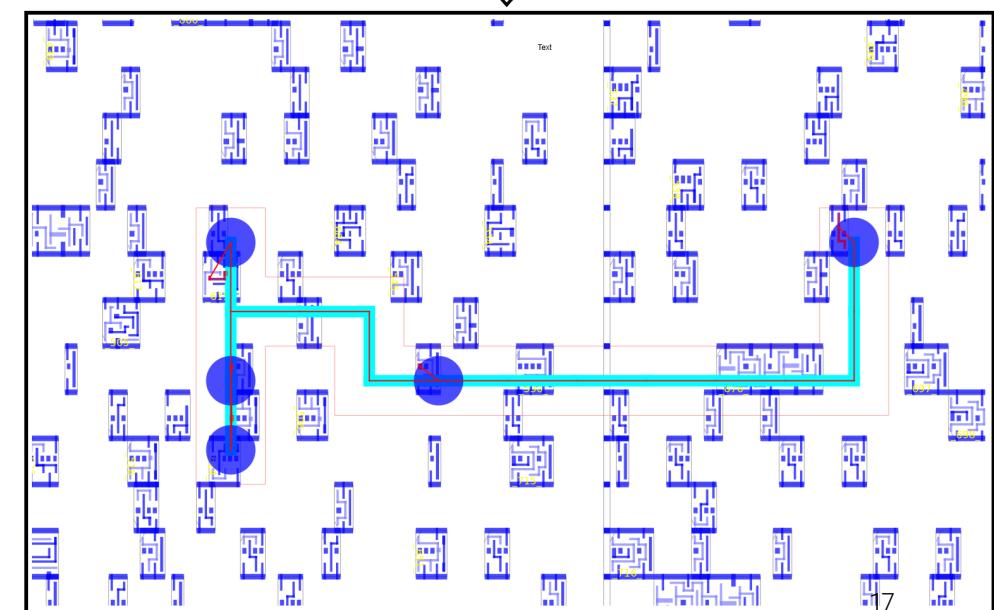
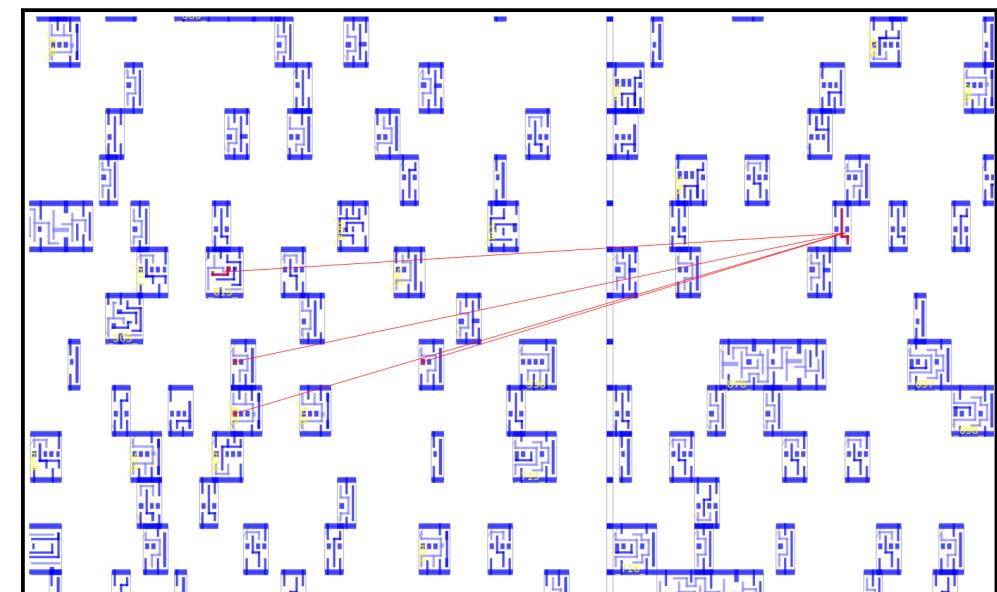
More Information on: Placement



Clock Tree Synthesis (CTS)

- ▶ Clock Tree Synthesis (CTS) builds the clock network.
- ▶ Ensures clock reaches all sequential elements.
- ▶ Minimizes clock skew and insertion delay.
- ▶ Guarantees correct timing across the design.





After Global Routing

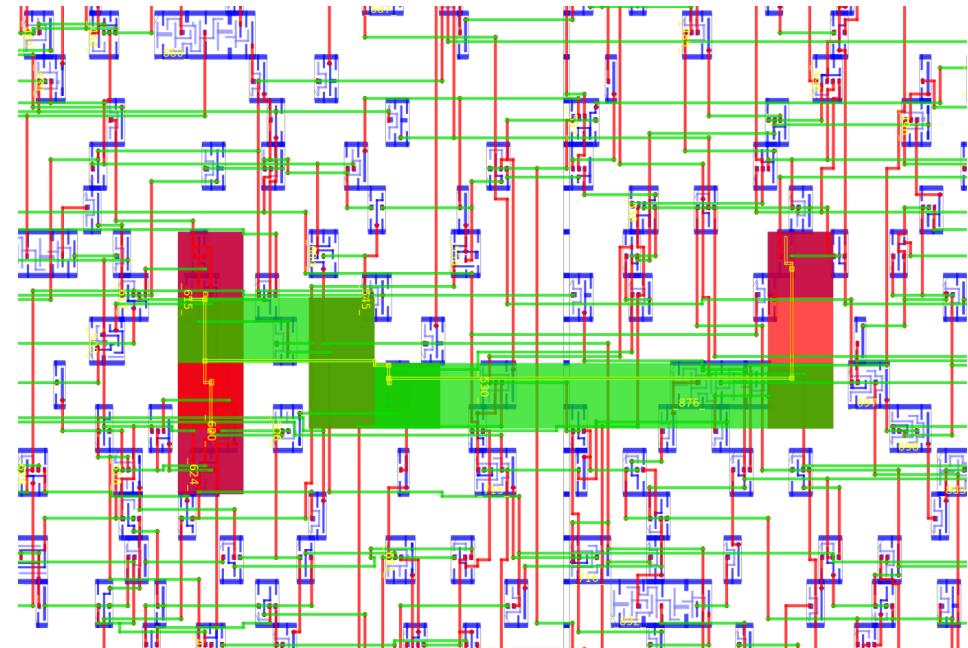
Global Routing

- ▶ Plans approximate net routes using a grid of routing cells (GCells).
- ▶ Models routing capacity and congestion across the chip.
- ▶ Produces routing guides, not final metal wires.
- ▶ Guides detailed routing toward a feasible solution.

More Information on: Routing

Detailed Routing

- ▶ Converts global routing guides into exact physical wires and vias.
- ▶ Assigns routes to specific tracks, layers, and vias.
- ▶ Strictly enforces all design rules (DRC): width, spacing, enclosures, min-area, etc.
- ▶ Produces a clean, manufacturable routing.



More Information on: Routing

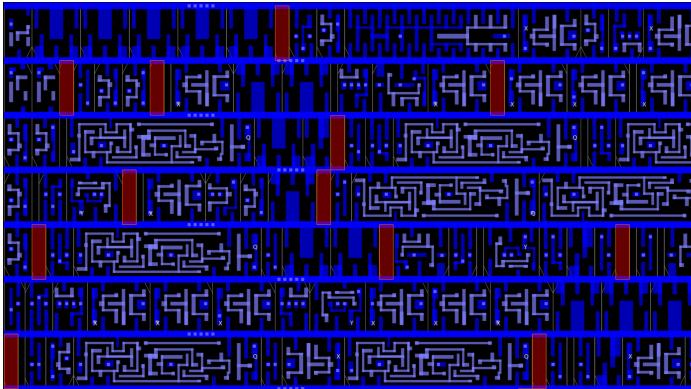
Signoff & Physical Verification

1. Timing Signoff
 - 1.1 Fill insertion
 - 1.2 RC Extraction
 - 1.3 STA post Physical Implementation
2. Physical Verification
 - 2.1 DRC
 - 2.2 LVS
3. Chip Finishing
 - 3.1 Sealring
 - 3.2 Metal Fill
 - 3.3 Others...

Timing Signoff

Fill Insertion

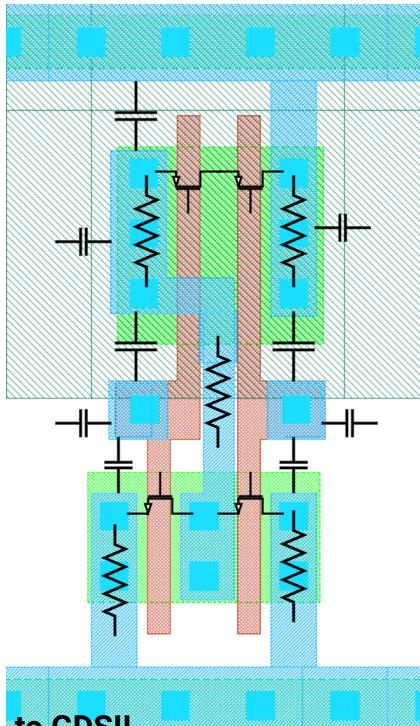
- ▶ Inserted to maintain well and power rail continuity.
- ▶ Do not implement logic; purely physical cells.



More Information on: RC Extraction

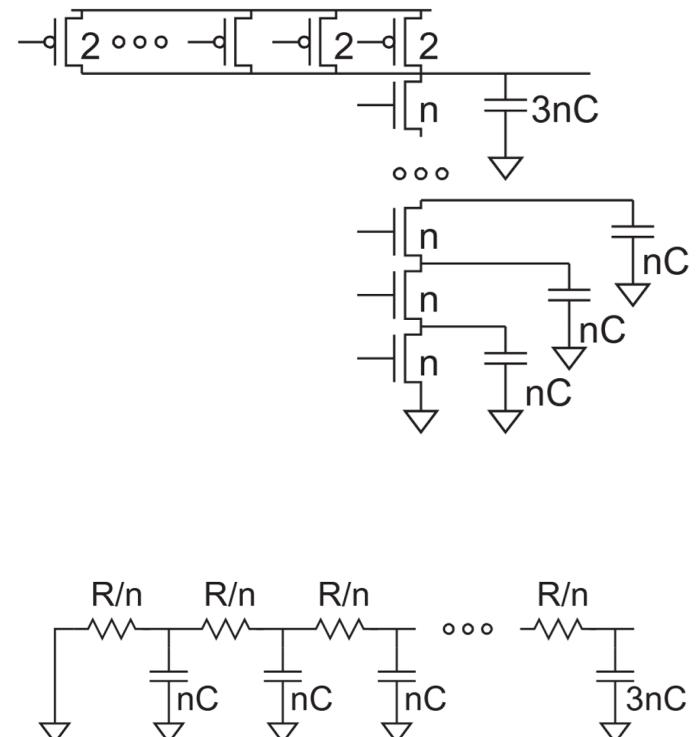
RC Extraction

- ▶ Extracts resistance and capacitance (RC) from the routed layout.
- ▶ Models parasitic effects of wires and vias.



STA post Physical Implementation

- ▶ Analyzes timing using post-layout parasitics (RC).



Physical Verification - DRC

- DRC run at the fullchip level to ensure that the layout adheres to the foundry's physical rules (geometries, spacing, etc.)

5.1 NWell

Rule	Description	Value
NW.a	Min. NWell width	0.62
NW.b	Min. NWell space or notch (same net). NWell regions separated by less than this value will be merged.	0.62
NW.b1	Min. PWell width between NWell regions (different net) (Note 3)	1.80
NW.c	Min. NWell enclosure of P+Activ not inside ThickGateOx	0.31
NW.c1	Min. NWell enclosure of P+Activ inside ThickGateOx	0.62
NW.d	Min. NWell space to external N+Activ not inside ThickGateOx	0.31
NW.d1	Min. NWell space to external N+Activ inside ThickGateOx	0.62
NW.e	Min. NWell enclosure of NWell tie surrounded entirely by NWell in N+Activ not inside ThickGateOx	0.24
NW.e1	Min. NWell enclosure of NWell tie surrounded entirely by NWell in N+Activ inside ThickGateOx	0.62
NW.f	Min. NWell space to substrate tie in P+Activ not inside ThickGateOx	0.24
NW.f1	Min. NWell space to substrate tie in P+Activ inside ThickGateOx	0.62

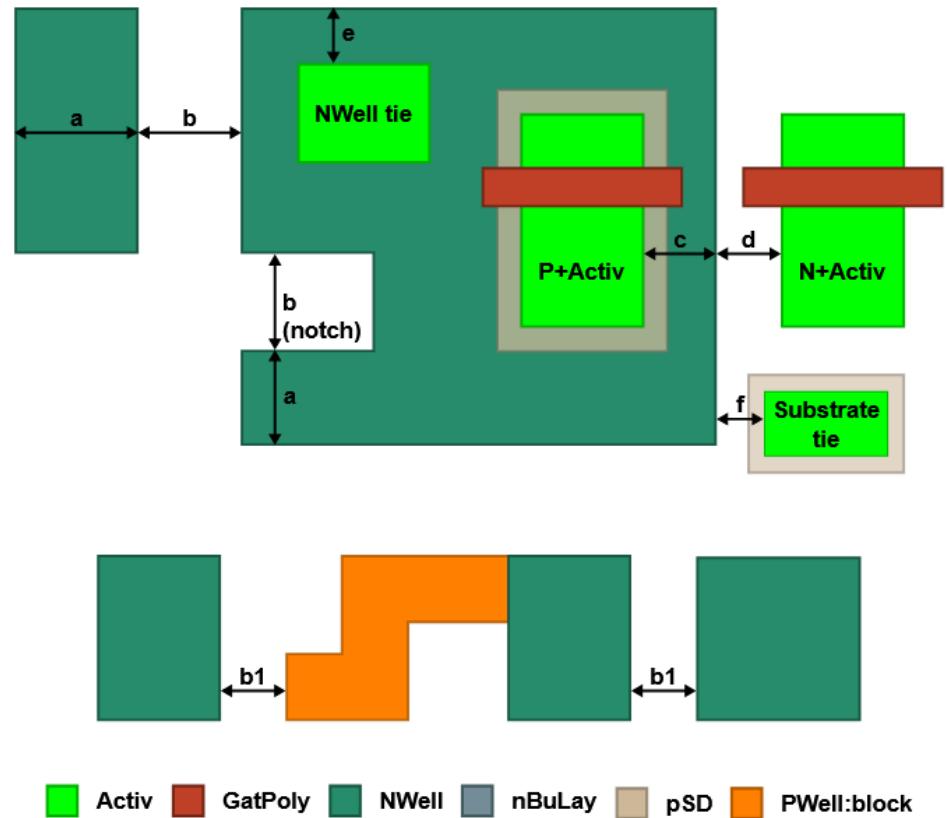
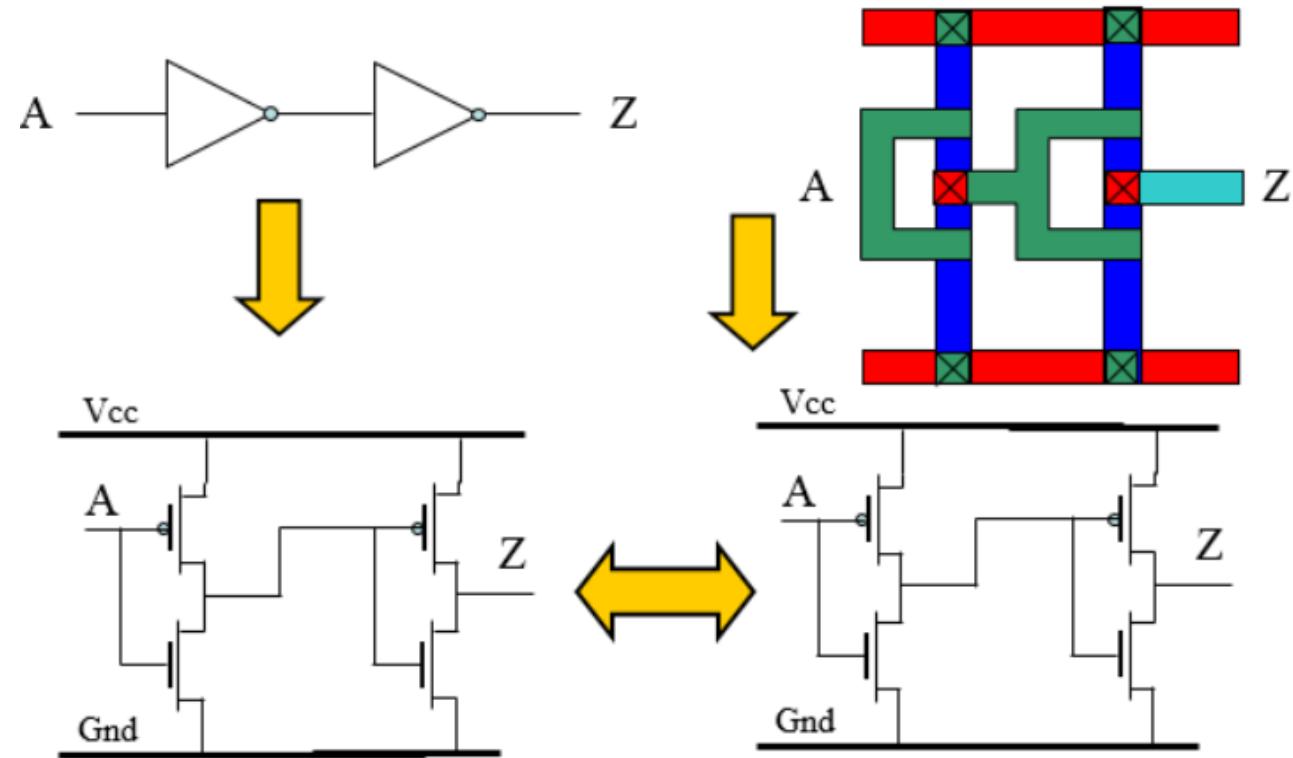


Figure 5.1: NWell dimensions (only rule variants without ThickGatOx are shown in this figure)

Physical Verification - LVS

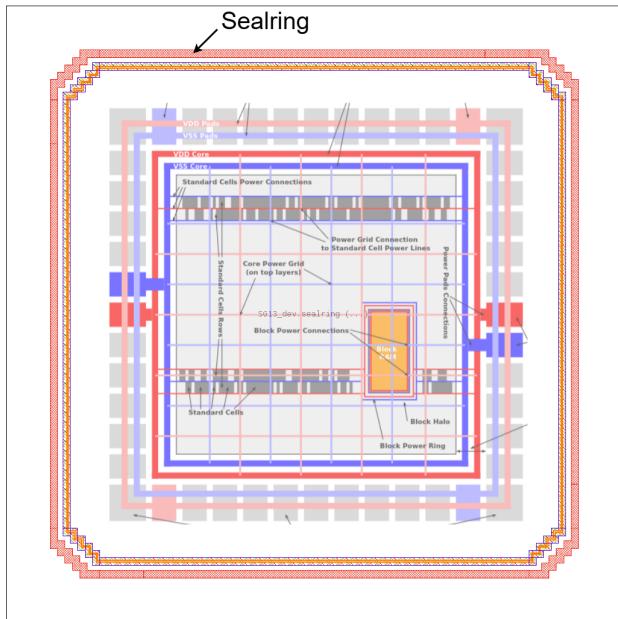
- ▶ Extract layout (GDS) and build a SPICE netlist
 - ▶ Sometimes need to black-box sensitive layouts.
- ▶ Export RTL and synthesized netlist from earlier stages.
- ▶ Compare both netlists to ensure they match.



Chip Finishing

Sealring

- ▶ Protective structure placed around the chip perimeter.
- ▶ Helps isolate the core circuitry during dicing and packaging.



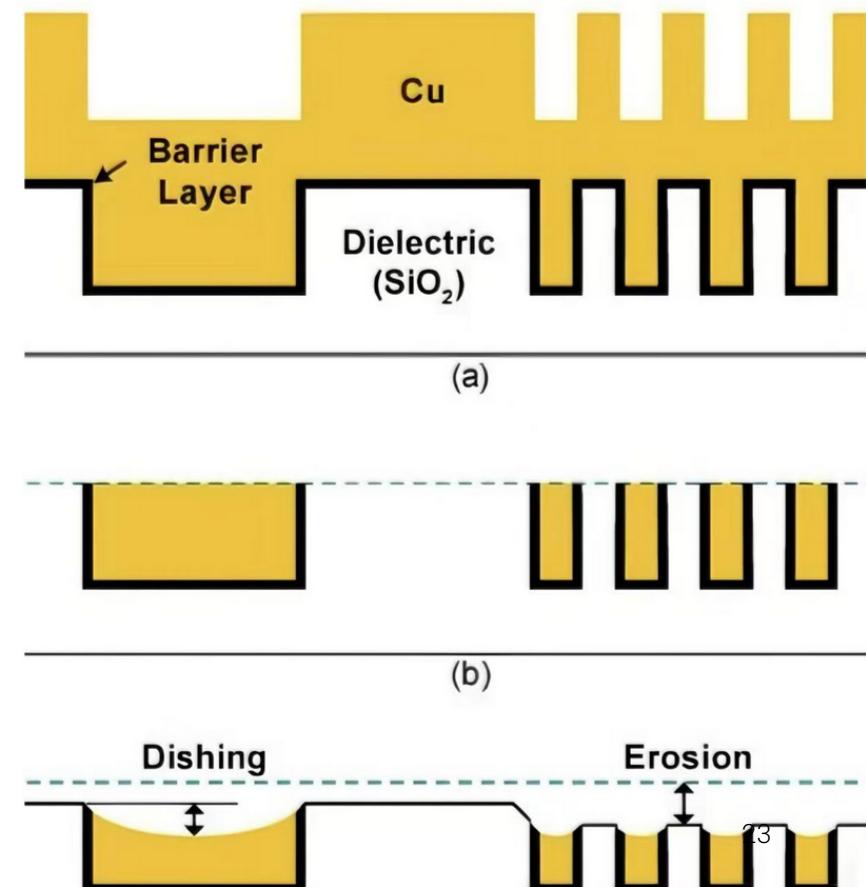
HM

Chip Design & Implementation – From RTL to GDSII

Daniel Arevalos – Fakultät 07 – Hochschule München

Metal Fill

- ▶ The wafer surface is polished to ensure flatness.
- ▶ Metal fill is added to areas of low metal density to even out the overall density across the chip.



Tapeout & Fabrication