Karlsruhe Institute of Technology

Design Digitaler Schaltkreise

Semicustom Design Flow introduction

Asic and Detector Lab - IPE

Prof. Ivan Peric ivan.peric@kit.edu
Richard Leys richard.leys@kit.edu

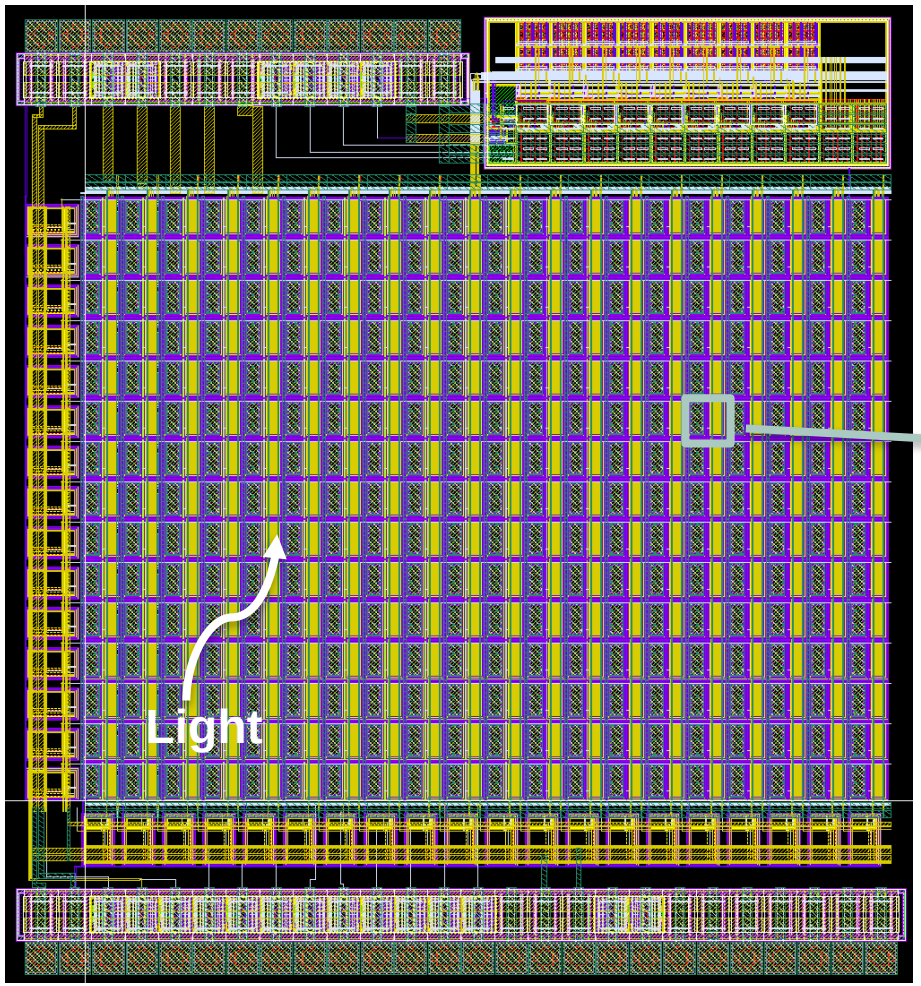
KIT ASIC and Detector Lab

Lecture Goal

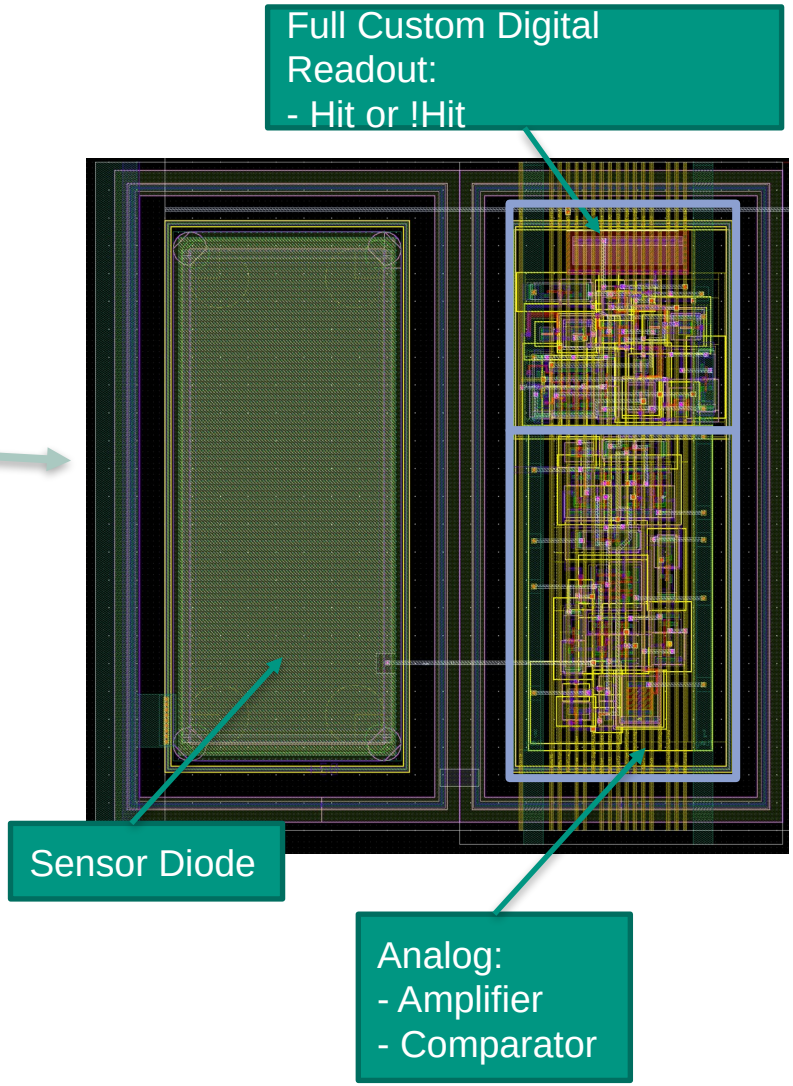
- Introduce System Level Design
  - Basic concept of Hardware Description Development
- Introduce Digital Implementation flow
  - Rapid overview
  - Some other lectures will go more in details:
    - Verilog Design
    - Synthesis
    - Place and Route 1/2
- Übung
  - Design and Plan
  - Comments are welcome

SYSTEM LEVEL BASICS

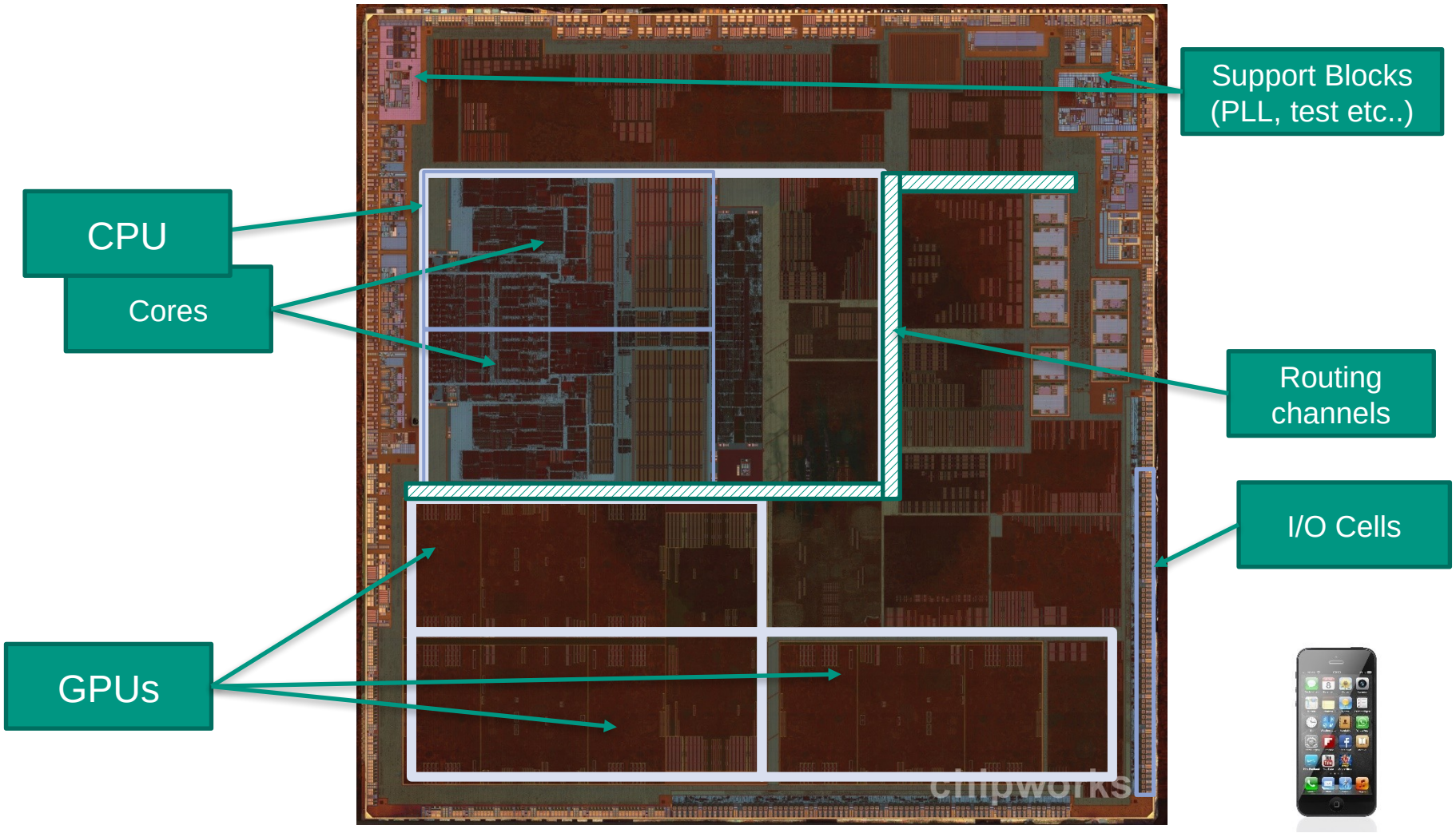
System Example: Mixed-Signal Sensor



Digital Silicon Photo Multiplier – 350nm – May 2016



System Example: Complex System on Chip

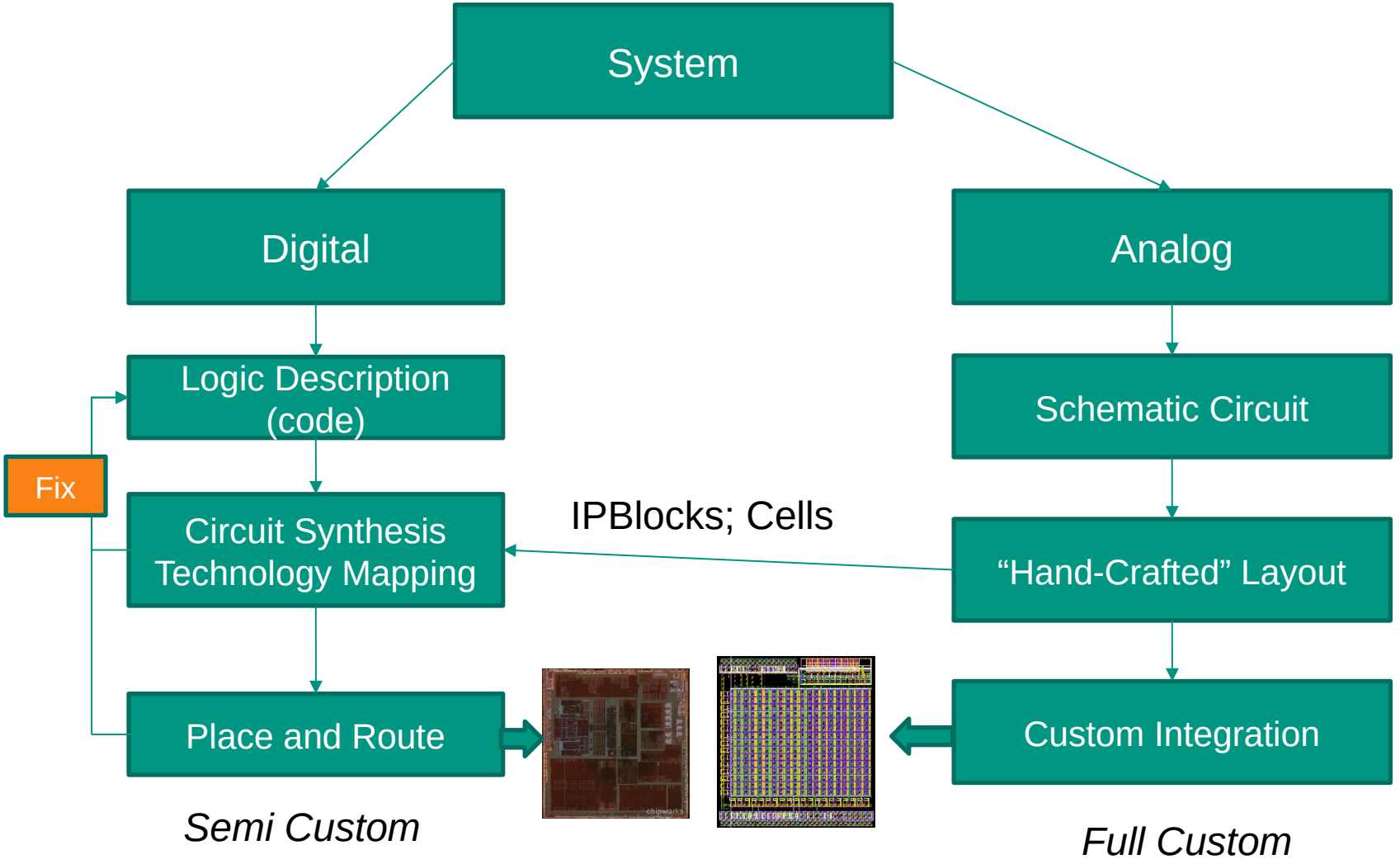


Iphone A6 APU – 28nm (picture from chipworks.com)

Stakes in Complex Digital Designs

- Features Size decreases, now 65nm is easy to access, next “standard” node for mere mortal is around 28nm, maybe 14nm
- Consequences:
  - More features per ASIC -> System on Chip
  - Each feature gets bigger -> Design and Implementation Cost higher
- System Implementation stakes:
  - Difficult: Register Transfer level design and verification
  - Difficult: System Physical Planning and constraining
  - Actual implementations: the tools do the job
- Costs Location:
  - Hardware Designer Time
  - Specification / IP Blocks
  - Computation time leads to project-parallelizing

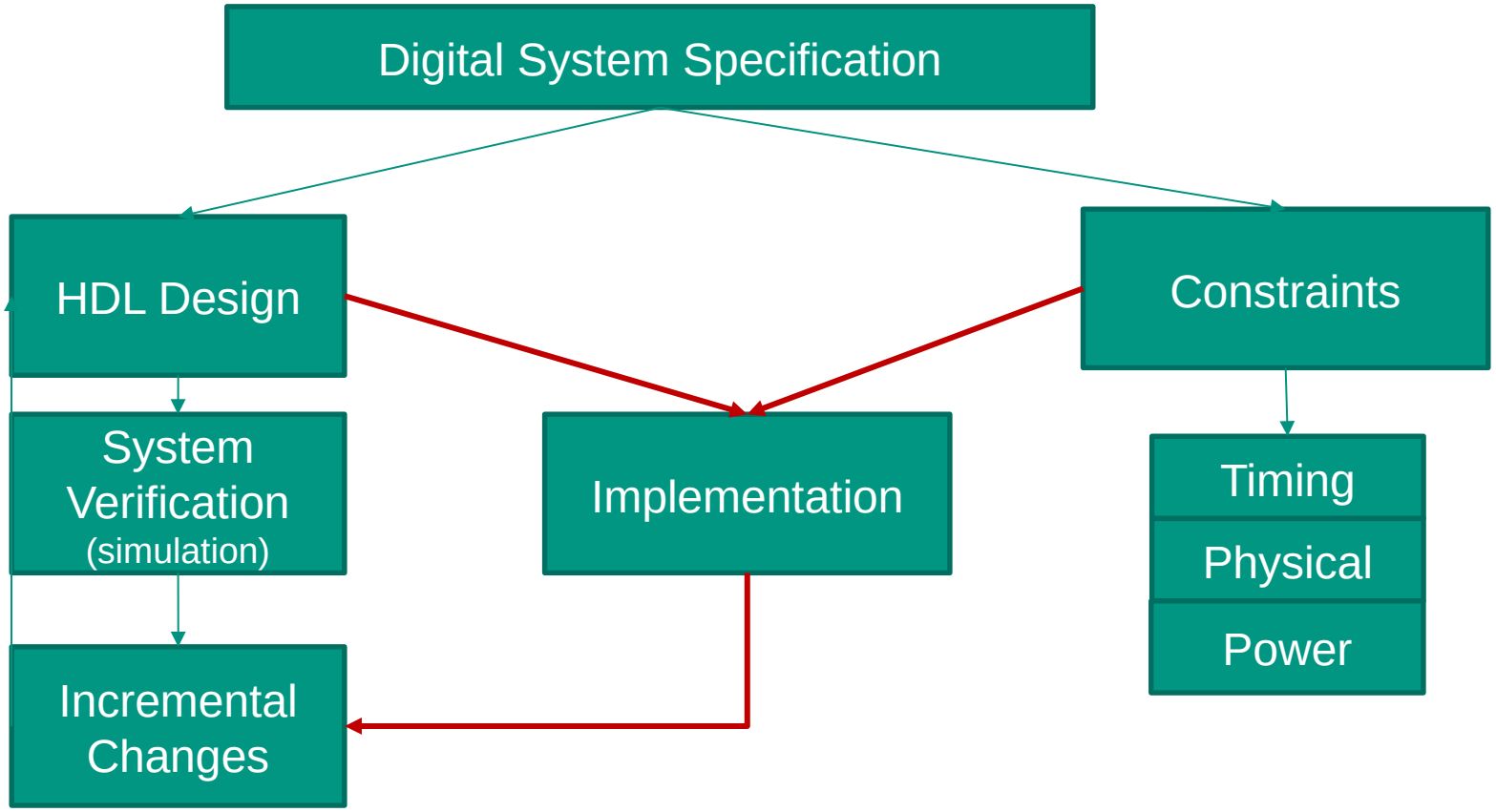
Analog and Digital implementation Flows



ASIC vs FPGA design

- We are interested in Digital ASIC target, not FPGA target
- RTL-Level (Verilog/VHDL) design is very similar
- FPGA
  - FPGA is a build system with pre-available resources
  - Synthesis maps to those functions
  - Place and route creates configuration file for routing
  - Design size and speed is limited
- ASIC:
  - Every component must be provided and placed
  - Example: SRAM memories
  - Example: PLL
  - Design can be big
  - But it’s expensive
- Key Point: Resources like RAM/PLL
  - What can I get at what Price ?
  - ASIC: Buy more IP? Develop them?
  - FPGA: Buy a bigger device ?

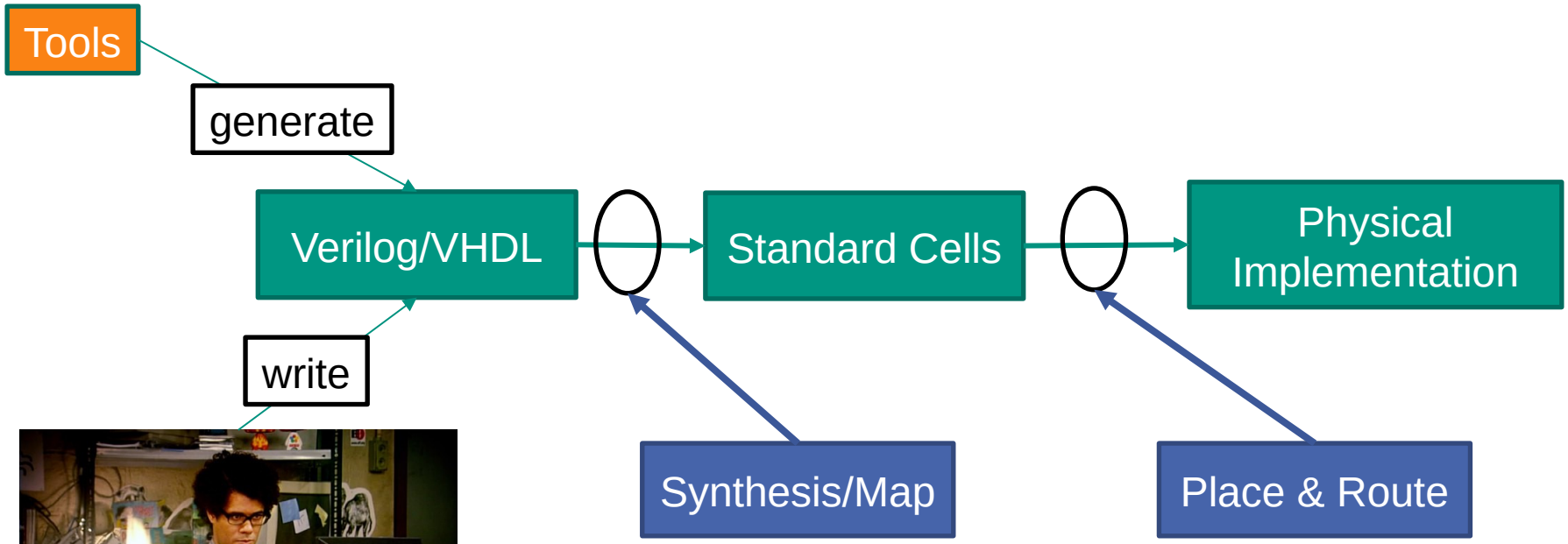
System Design Phases



DESIGN FLOW OVERVIEW

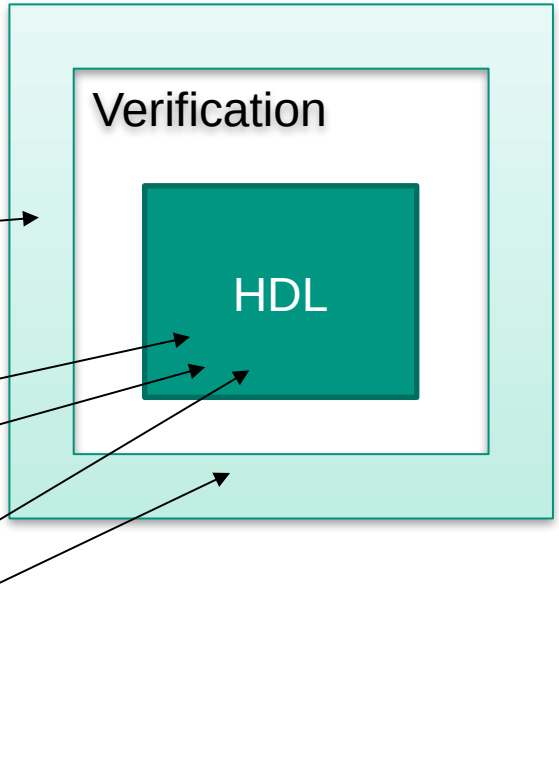
Simplified Overview

- Abstract: Hardware Description (HDL = Verilog/VHDL) + Tools
- Concrete: Technology Valid implementation



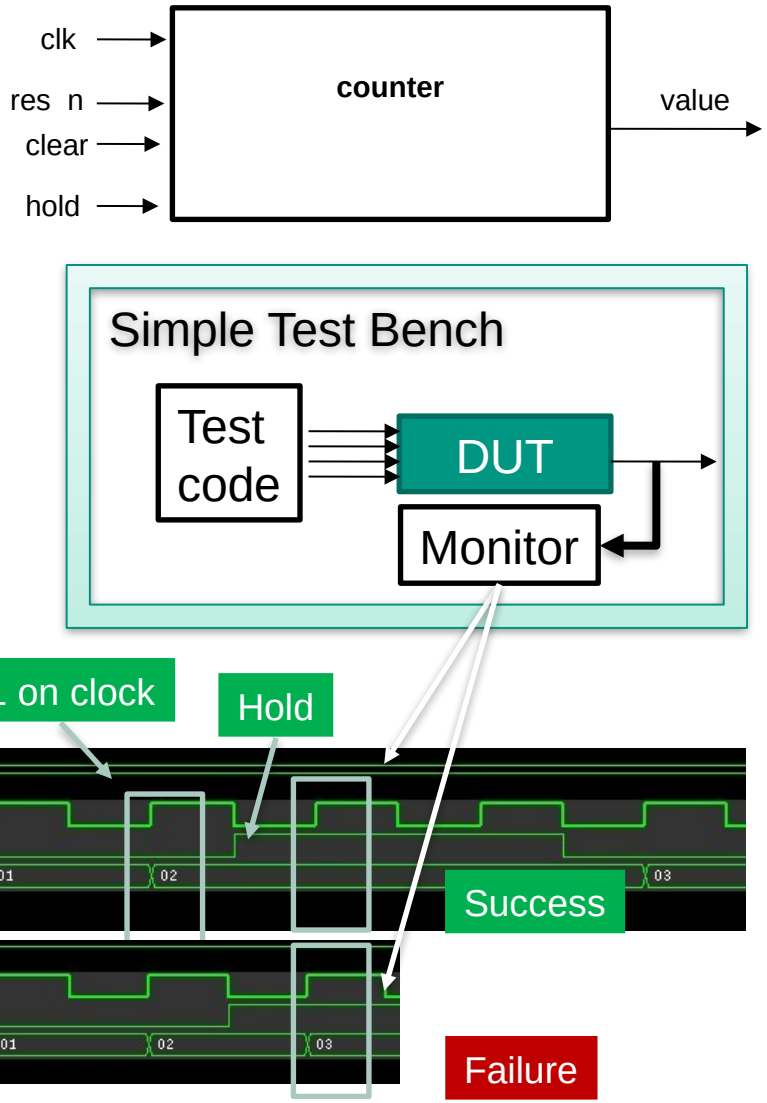
Hardware Description Methodology

- HDL - Register Transfer Level (RTL)
  - Verilog / VHDL
- System Verification and Metrics
  - Simple test benches
  - Complex Functional Verification
    - UVM with SystemVerilog/e-Language
  - Assertions
    - “Inline” signal dependencies check
  - Coverage
    - Detect logic usage
- Software Supported design
  - Abstract tools for easy design
    - FSM designers
    - Registerfile generator
    - HDL generator
    - Etc…
  - Language: Often Tool Command Kit (TCL)
  - Requires good software skills to develop tools



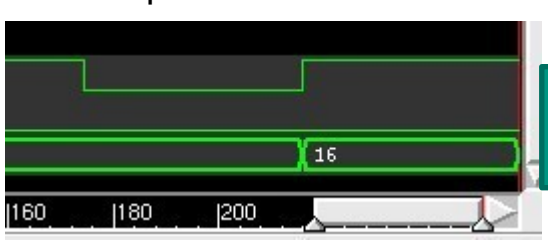
HDL - Functional Verification

- Verification of Logic by Simulation
- The design is called a Device Under Test (DUT)
- The Simulation is driven by a testbench:
  - Change the Inputs
  - Monitor the outputs (“eye” control or software comparisons)
  - Test if the outputs are matching specification
- The testbench is written in Verilog/VHDL as well…
- …but it is just like a software (can use threads etc…)
- Testbench type examples:
  - Simple Verilog File driving a small design (“eye” control)
  - Complex Object-Oriented environment with data randomisation like Universal Verification Methodology (UVM)



HDL - Coverage

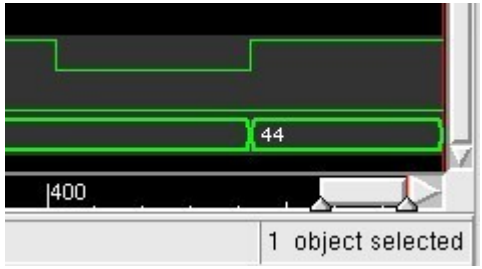
- Coverage gather statistics about the values of the signals encountered during simulation.
- They are usually carefully designed
- Typical usage:
  - A multibit value may be too large and not all the bits are required (ex: Defined 8 bits but value is never > 15 -> only 4 bits enough)
  - State machine states are represented by a value (ex 1,2,3 etc…). If a state is never matched, it is either not needed, or can indicate the source of a failure
- Example for counter value:



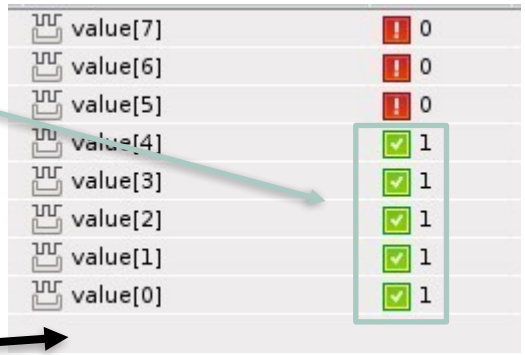
Simulation:
0 -> 16 -> Stop

4 bits hit

Simulation:
0 -> 44 -> Stop



5 bits “hit”



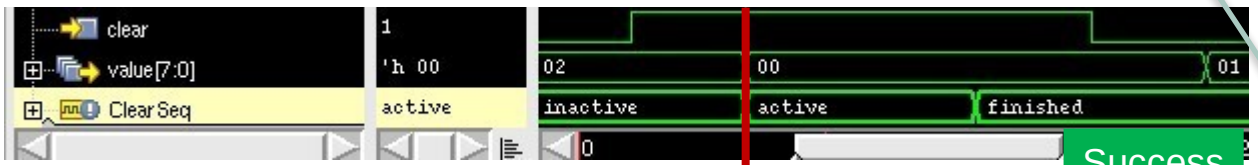
Output from Coverage Analysis Tool

HDL - Assertions

- “Inline” requirements for signal specification inside hardware module definition
- Executed during simulation
- Usually the simulation stops and fails on assertion error
- Very convenient for specification enforcement
- Example for counter: Clear behaviour

```
ClearSeq : assert property
(@(posedge clk) (clear |=> value==0));
```

Clear is 1 | Next Clock Cycle | Value is 0



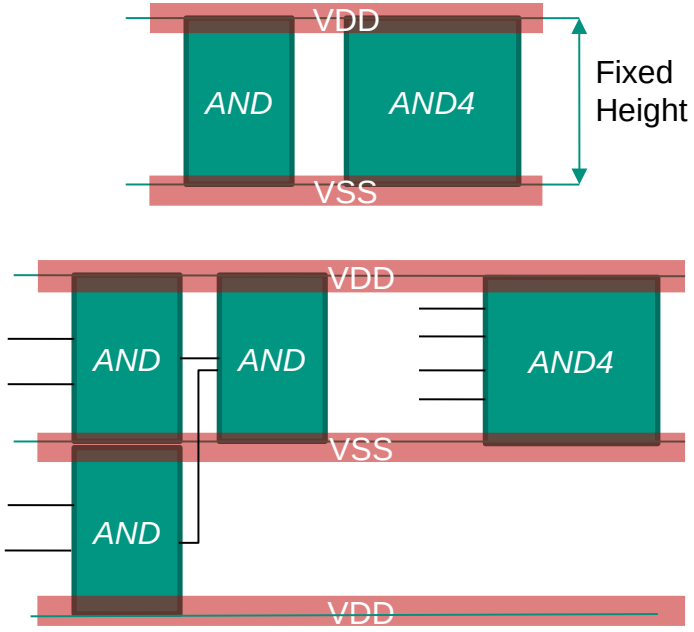

```
ncsim: *E,ASRTST (../sources/counter.v,42): (time 85 NS) Assertion tb_top.dut.ClearSeq has failed (
85 NS + 3 (Assertion output stop: tb_top.dut.ClearSeq = failed)
```

```
module counter #(parameter SIZE = 8) (

    // System
    input clk,
    input res_n,

    // Control
    input hold,
    input clear,

    // Output
    output reg [SIZE-1:0] value

    );

    always @(posedge clk or negedge res_n) begin

        // Reset
        if (!res_n) begin

            value <= {SIZE {1'b0} };

        end
        // Main
        else begin

            if (clear) begin
                //value <= {SIZE {1'b0} };
            end
            else if (!hold) begin

                value <= value +1;

            end

        end
    end

    // Assertion
    //---------------
    ClearSeq : assert property
    (@(posedge clk) (clear |=> value==0));

endmodule
```

Counter Verilog code

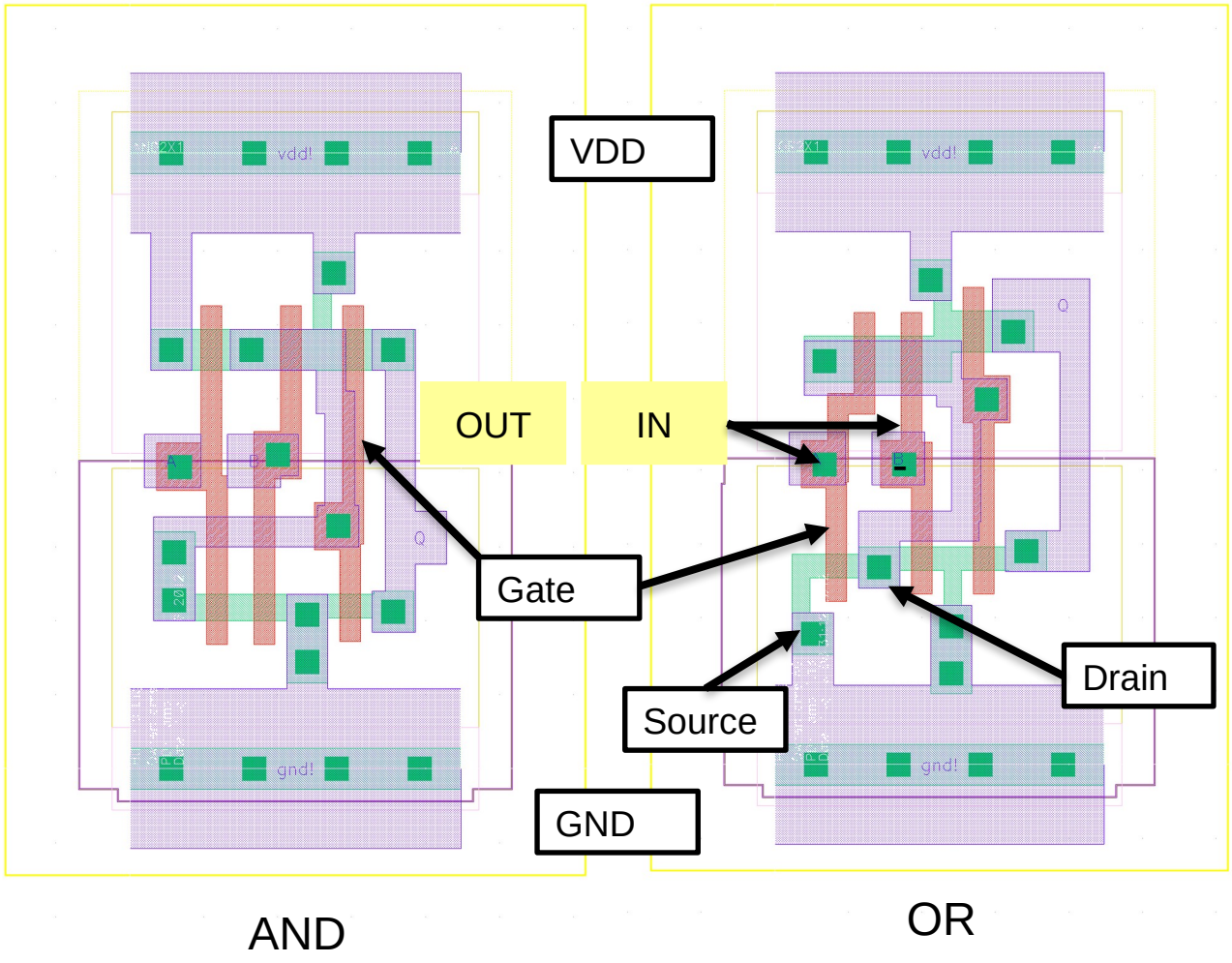
Synthesis: Standard cell mapping

- Start Point: HDL
- Synthesis creates the logic circuit from HDL
- Circuits is mapped to Standard Cells
  - Standard Cells are VLSI implementation of simple logic functions inside a standard sized block
  - Designs Kits provide standard cells
  - Each Cell is a logic function implementation:
    - Example: AND3 -> AND with 3 inputs
  - Timing is provided in Cell description
    - Example: Slow/Typical/Fast corners
- Mapping is optimised:
  - Timing
    - Example: Use Low threshold cells
  - Area
  - Power

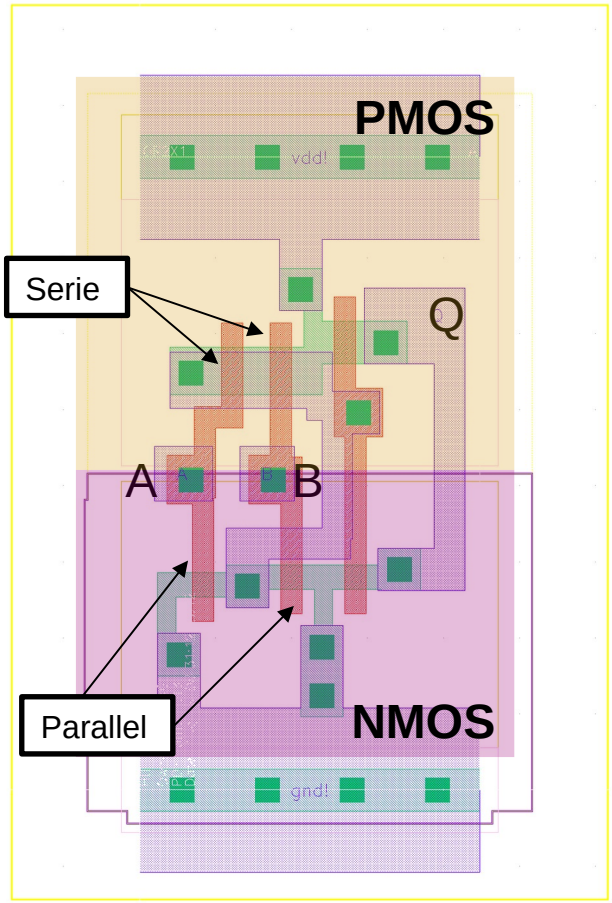


Physical Placement organised in Rows

More standard cells

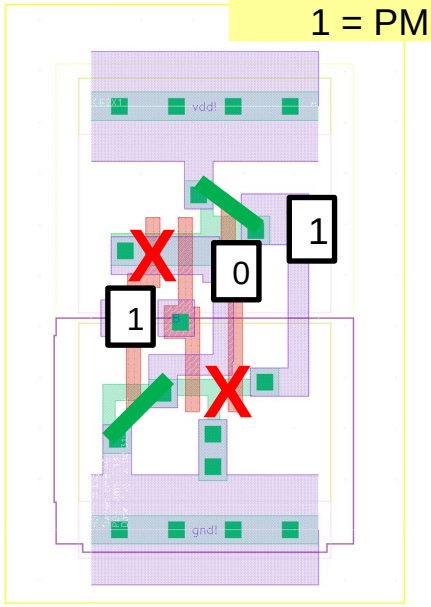


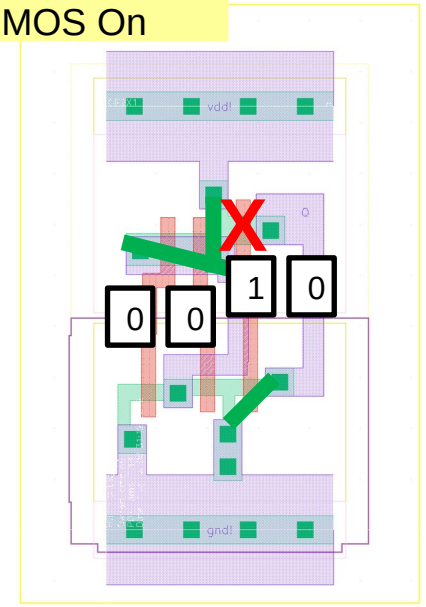
OR Function Working Principle

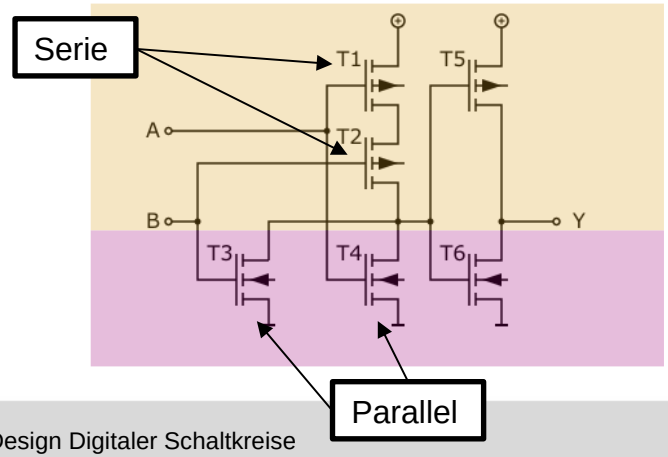


OR

0 = PMOS On ; NMOS Off
1 = PMOS Off; NMOS On





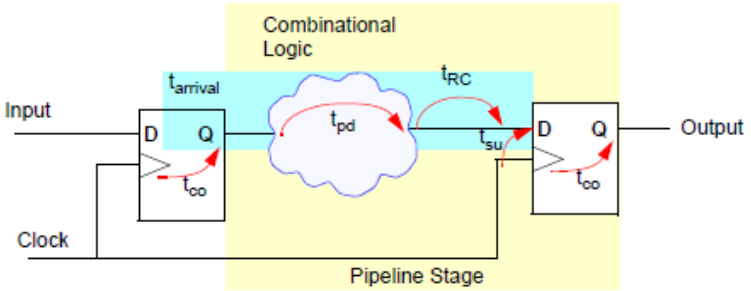


A	B	Q
0	0	0
1	0	1
0	1	1
1	1	1

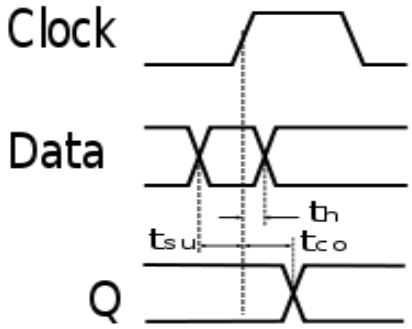
Synthesis: Static Timing Analysis (STA)

- Lowest Level in Register Transfer Level
  - Register – Logic - Register
- Timing Constraints:
  - Clock cycle definition
  - Input/Output Delays
- STA:
  - D->Q time from FlipFlop
  - + Logic (gates + interconnect) time
  - Must fit in the clock cycle
    - Check setup time: Too slow
    - Check hold time: Too Fast
- <u>Fixes happen along the whole implementation process</u>

$t_{cyc} - t_{arrival} \geq t_{su}$ must be satisfied by all pathes



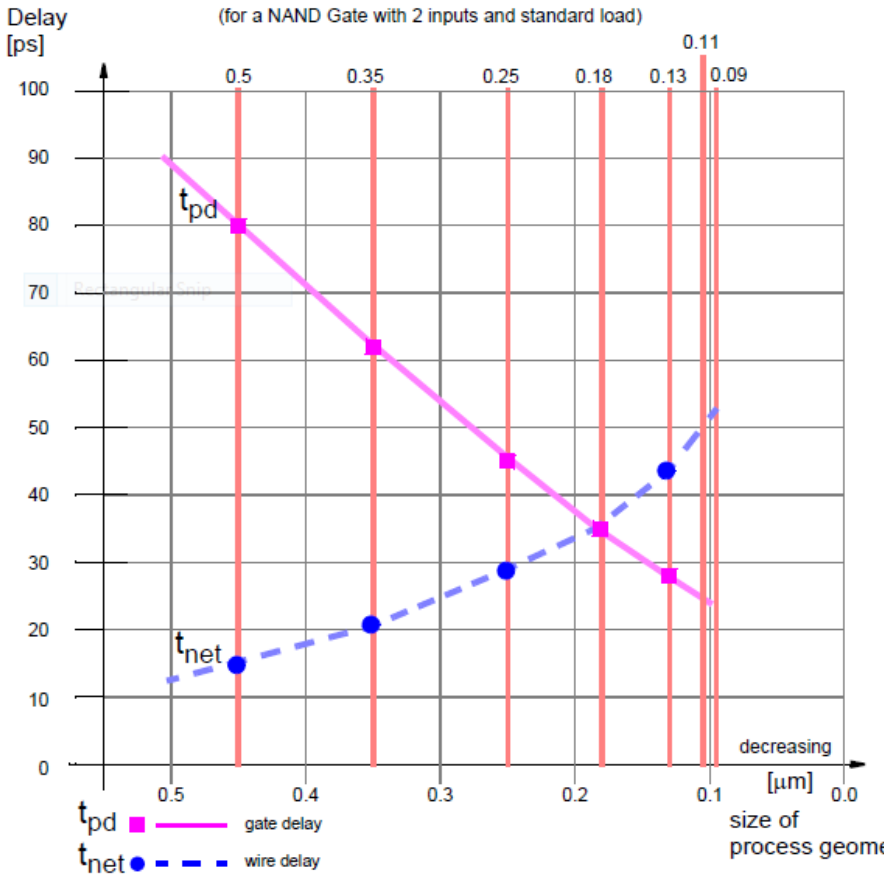
where: $t_{arrival} = t_{co} + t_{pd} + t_{RC}$



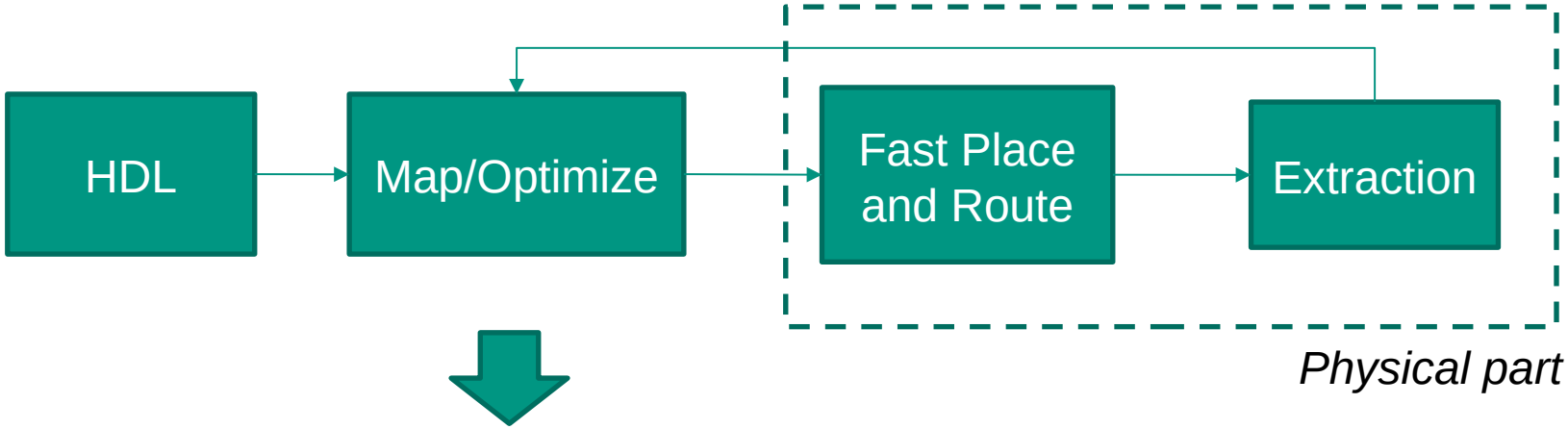
Picture: Uni. Heidelberg – LS Rechnerarchitektur

Synthesis: Static Timing Analysis (STA)

- During Circuit mapping, the STA checks the timing paths
- Two delay paths:
  - Gate Delay: Given by Standard Cell Specification
  - Wire Delay: Interconnection
- Wire Delay:
  - Small feature size makes wire delay predominant
  - Need for physical synthesis



Synthesis: Physical Synthesis

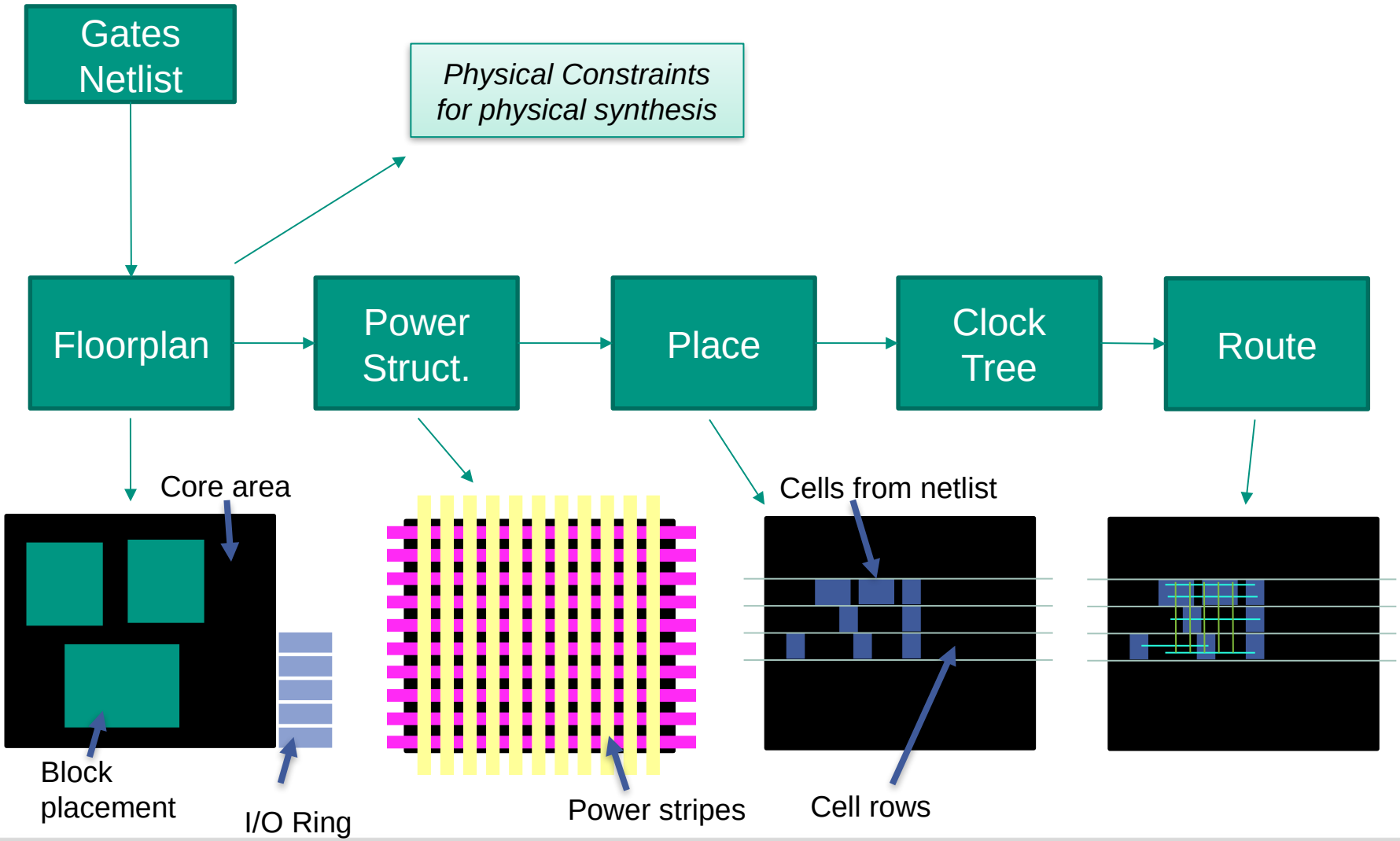


Gates Netlist

- Verilog file with module instances
- Modules are Standard Cell instances

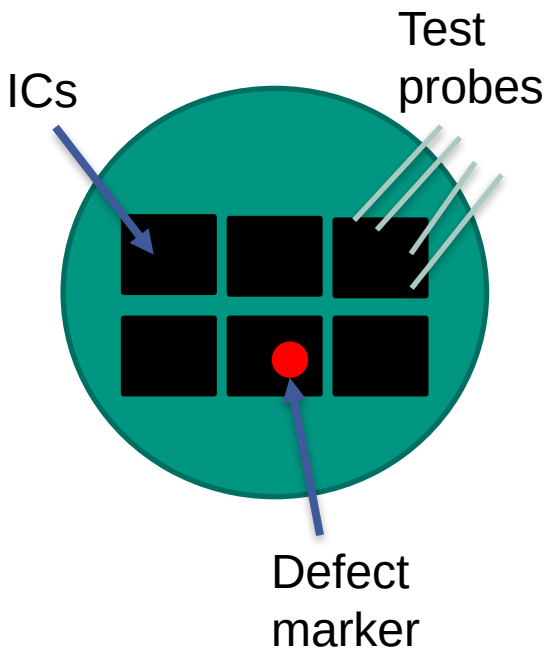
.AND(a,b,net1);
.OR(c,net1,net2);

Place and Route overview



Design For test

- After Production, sort:
  - Working Ics
  - Defect Ics
  - Defects/Working = Yield
- Tests types:
  - Optical defect detection
  - Flip-Flop test patterns: JTAG
  - Self test for some blocks
    - SRAM memories have self testing
  - System level test function (like a testbench but in real life)
- When to test
  - Wafer level during manufacturing
    - Expensive
  - In Lab/Facility using system level tests
- How to implement tests
  - Test patterns can be added during synthesis
  - System Level:
    - Embedded test functionality
    - System test: Setup a system and see what happens

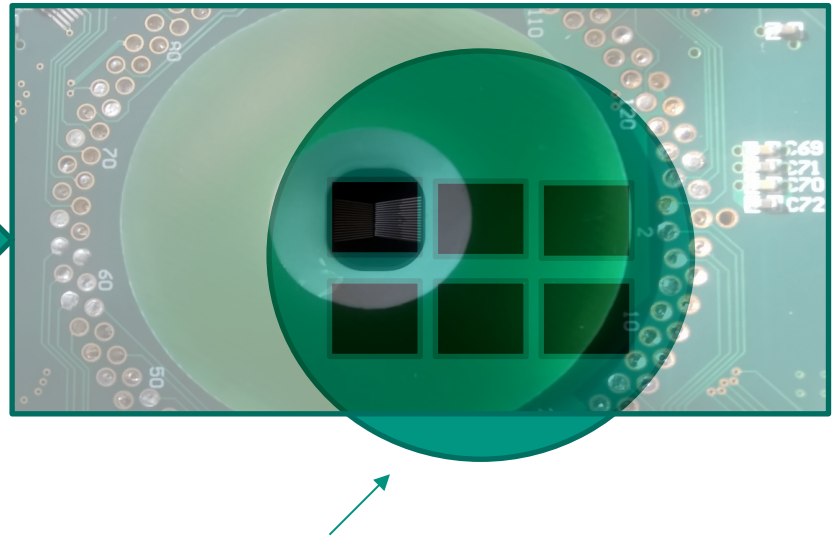


Wafer level test example

Wafer Level / Single Chip Test example

- Probe Card with an FPGA
- Software drives the Input/Output of the ASIC
- Data is readout and Measurements saved to qualify the ASIC
- Can be done in chain with the help of a camera driven XYZ table

Probe Card Tester for Belle 2
(ASIC in AMS 180um HV process)



Wafer or Single Chips

ÜBUNG

Content

- Goal: Lear to use Industry tools and get used to the design flow
- HDL Design
  - Very Simple Hardware Design to learn Verilog/VHDL
    - One or multiple counter types is enough
    - More Complex System: See PSOC Praktikum im WS16
  - Simulation using Cadence INSICIV suite
- Synthesis
  - Synthesis to a Generic 45nm technology
    - Learn about technology data, Logic Cell types etc…
  - See some synthesis options like Clock Gating , Reset Types, Physical Synthesis…
  - Simulation of Synthesised design
- Implementation
  - Place the Design in a “ready to produce” ASIC
  - Prepare Chip Constraints:
    - Target Clocks
    - I/O Delays
    - Floorplan
  - Go through physical implementation with timing checks
  - Simulation of placed design with timing information

Tools: Cadence INSICIV and EDI suites

- Simulation: “irun” tool from INSICIV Cadence Suite
- Programming: TCL Language
- Synthesis:
  - Cadence RTL Compiler
- Place and route
  - Cadence First Encounter
  - Also called during physical synthesis
- Unlike Analog Design tools:
  - Write TCL scripts
  - TCL contains function calls for various steps:
    - read_design
    - elaborate
    - read_constraints
    - synthesize –to_placed
- GUI View:
  - See the results of scripts run mostly
  - Analyse Timing

Übung organisation

- Groups of ~20
- 2 groups
- Every 2 weeks
- Begin: 12th May
- Where/When: Thursdays 14h00 at IMS (Westhochschule)
- Registration on ILIAS
- Overlap with ITIV Hardware Design Lab: OK
  - HD-Lab is ok as DDS Übung
  - Maybe we will run synthesis on HD-Lab content at the end of semester

End Slide!