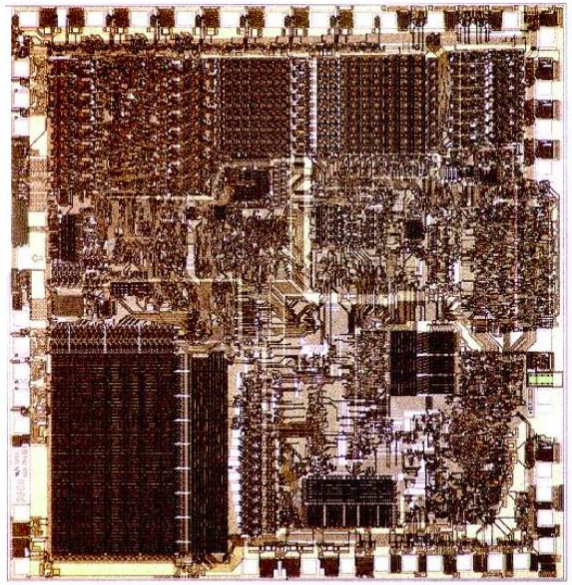
EE-429
Fundamentals of VLSI Design

The RTL (Semi-Custom) Design Flow

Andreas Burg

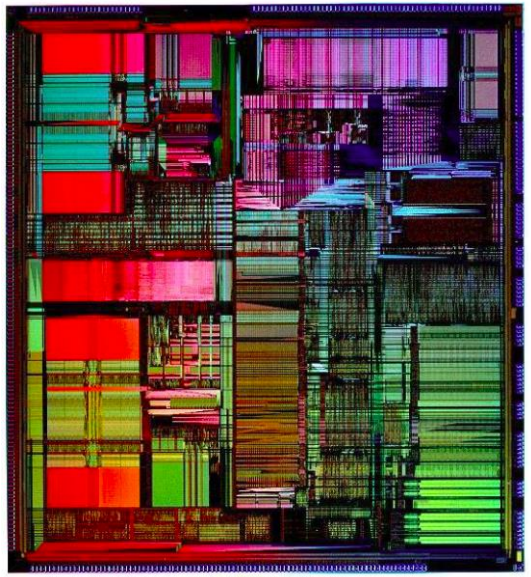
Reminder: Limits of Full Custom Design

- **Increasing integration density no longer allows for design on transistor level, neither on schematic, nor on layout level**



http://download.intel.com/museum/exhibits/hist_micro/hof/large_jpegs/8088B1.jpg

Intel 8088, 1979
Full-custom design



http://download.intel.com/museum/exhibits/hist_micro/hof/large_jpegs/pent8.jpg

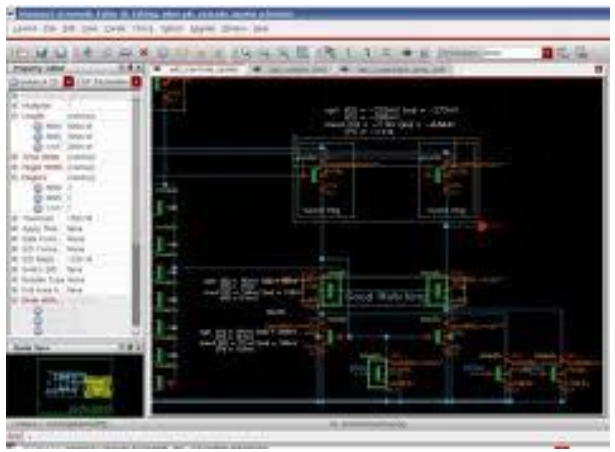
Intel Pentium, 1993
Few macros, but mostly built using automatic tools

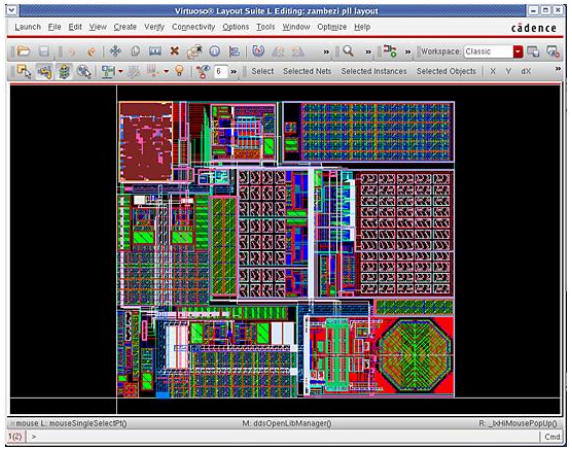
- **Need for a more automated that leaves the details to EDA tools**

Full-Custom vs. Semi-Custom Design

- **Different design styles offer tradeoffs between control over details and the complexity that can be handled efficiently**

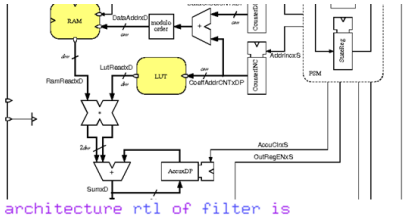
Full Custom Design





- Design and verification on transistor level
- Schematic and layout done manually by the designer with few automatic tools

Semi-Custom Design

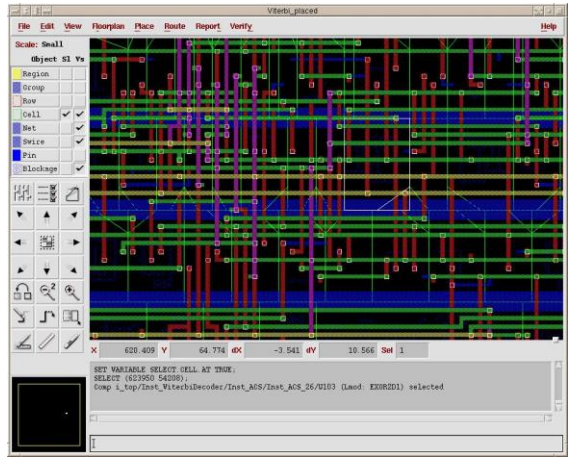


```
architecture rtl of filter is

begin  -- rtl

  ----------------------------------------------------------
  -- Arithmetic unit (multiply - add)
  ----------------------------------------------------------
  p_ALU : process (LutReadxD, RamReadxD, AccuxDP)
    variable Product : signed(27 downto 0);
  begin  -- process ALU
    Product := signed(RamReadxD) * signed(LutReadxD);
    SumxD   <= Product + AccuxDP;
  end process p_ALU;

end rtl;
```

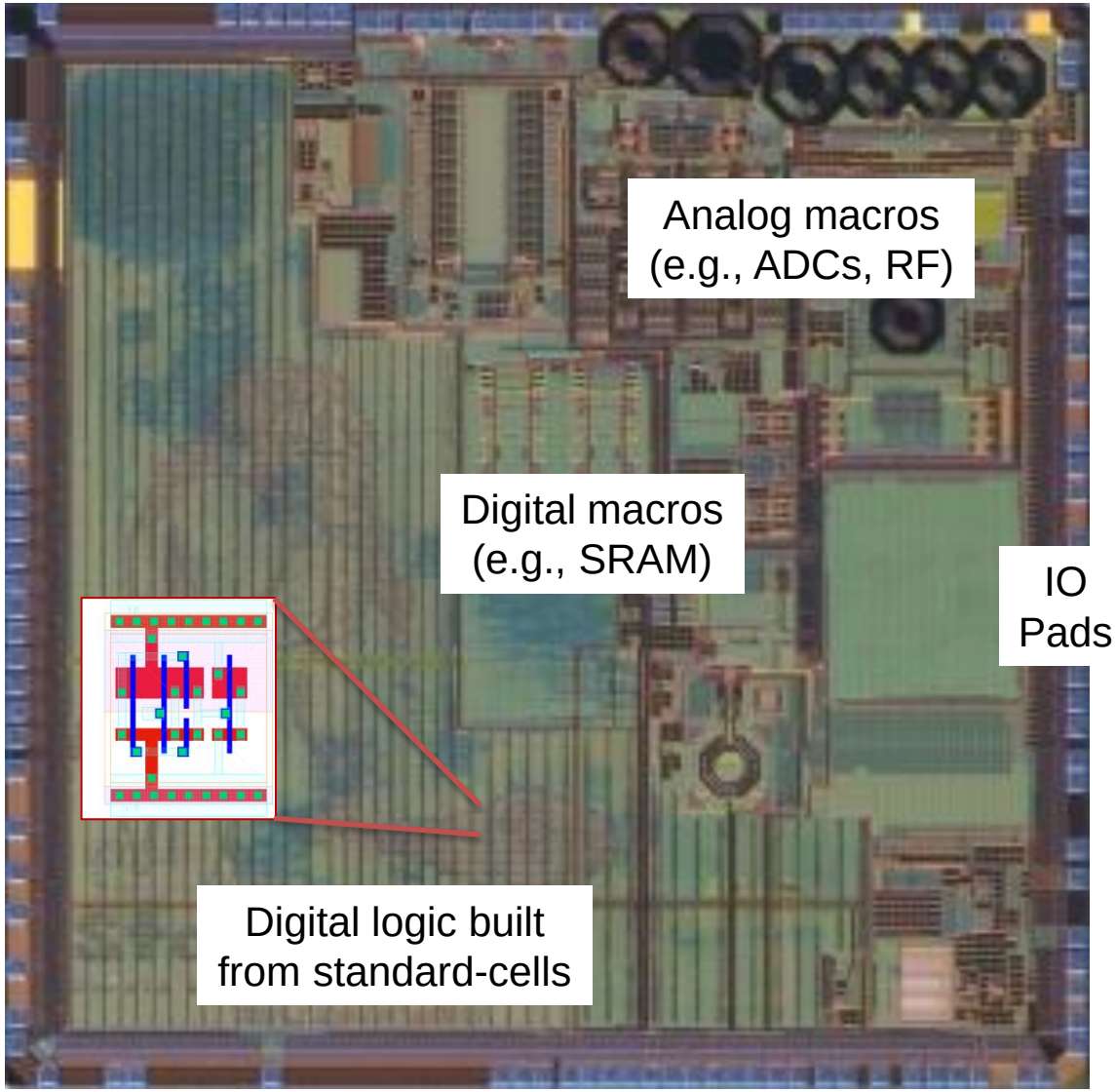


- Design on Register Transfer Level (RTL)
- Heavy use of automatic tools and basic libraries to derive schematic and layout

Anatomy of a Complex Chip

- **Complex SoCs are often a mix of design styles**

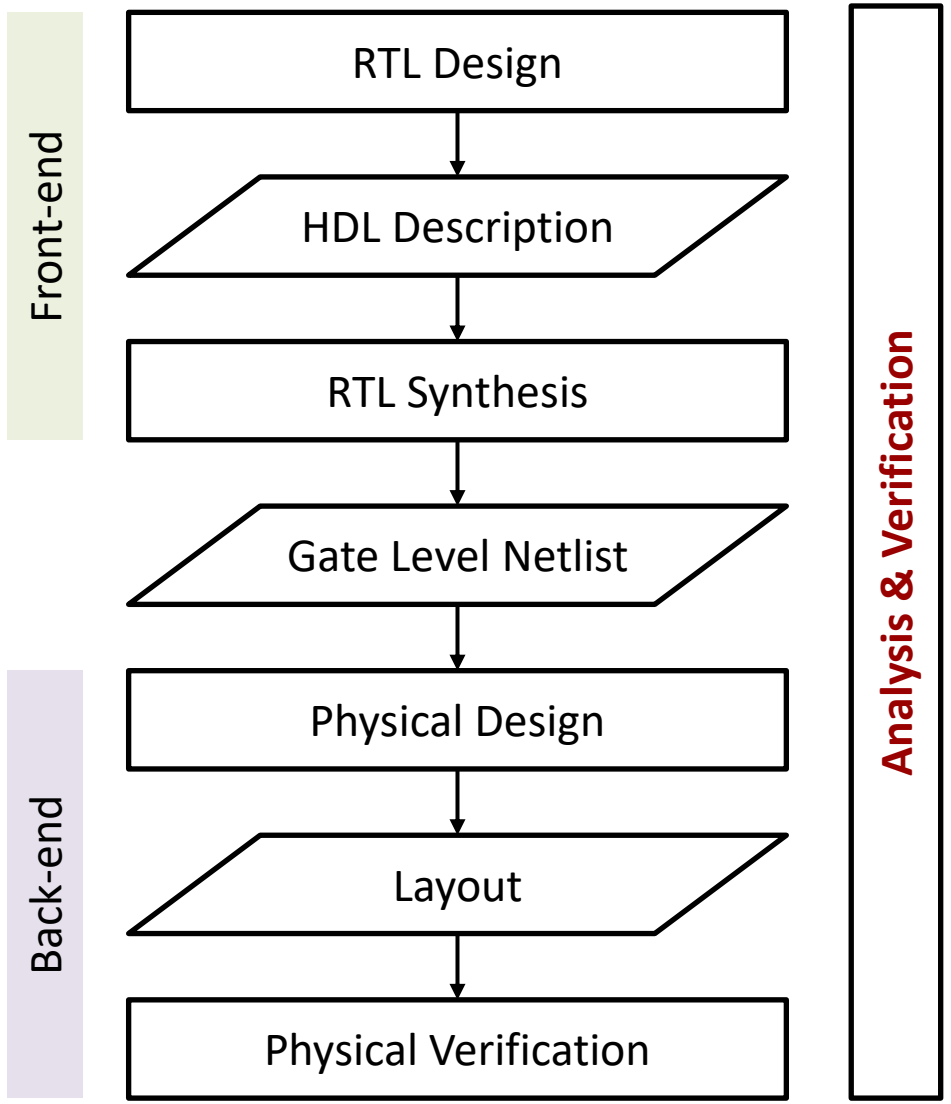
- **Composed from**
  - A pad ring and few special top-level components such as PLLs and I/O cells (pads)
  - **Analogue macros created with a full-custom design flow**
  - Complex **digital sub-systems built from random logic and few large IP macros** (e.g., SRAMs) **using a semi-custom RTL design flow**



2G Cellular SoC, IIS, ETHZ

Semi-Custom (Digital) ASIC Design Flow

- **Semi-custom design flow:**
  - Starts from a Register Transfer Level description in a hardware description language (HDL)
  - **Front-end flow**: handles the transition from RTL to the gate level
  - Back-end flow: handles the transition from a netlist to physical design data
- **Each step is** always **accompanied** by **analysis & verification**
  - Check functionality, timing, and physical constraints



Reminder: Principles of Efficient Design

- **Compiling a large design without caring about details requires**

  - **Hierarchy**: **partition** complex systems **into** smaller **sub-systems** and **repeat** this process **until** either **complexity** of the sub-systems **is manageable OR** a **pre-built block (IP) is available**
    - Comparable to using functions in programming, but related to components in hardware design
    - Closely **related to** the need for **regularity**: breaking down blocks favours **re-using common blocks**

  - **Abstraction**: **simplify the *description***/characterization ***of components*** *as a model (black box)* **to facilitate using them** on the next level of hierarchy
    - Abstraction happens in different design representations/views: beavioral, structural, or physical

  - **Design automation**: **use algorithms and tools to translate abstract design descriptions into detailed inplementations**, often building on pre-built basic building blocks (basic IPs)

Reminder: Design Abstraction Levels

System/Algorithm Level

- Rapid realization
- Technology/implementation independent
- Rapid simulation of functionality, often floating point
- No information on timing/delay/latency/throughput/complexity

Register Transfer Level

- Requires full architecture design and fixed-point optimization
- Simulations are significantly slower
- Provides cycle accurate delay simulation
- No accurate timing (in ns)

Gate Level
Pre- and Post-Layout

- Requires synthesis / potentially P&R for a given technology
- Simulations are slow
- Reasonably accurate area estimation
- Enable timing and limited power analysis

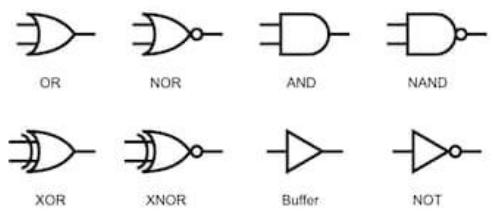
Transistor Level (Electrical)

Mask Level (Physical)

Fabrication Mask Level

Basic Building Blocks: Standard Cells and IP Macros

- **Semicustom designs combine three main ingredients** to efficiently implement complex designs on the basis of hierarchy, abstraction, and automation

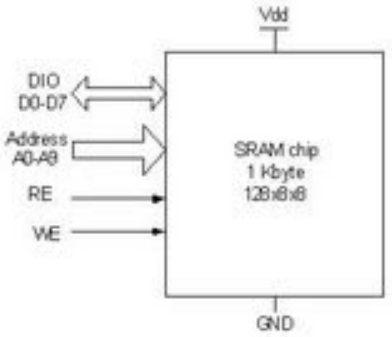


Instantiated automatically by synthesis tools

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity signed_adder is
6   port
7   (
8     aclr : in   std_logic;
9     clk  : in   std_logic;
10    a    : in   std_logic_vector;
11    b    : in   std_logic_vector;
12    q    : out  std_logic_vector
13  );
14 end signed_adder;
15
16 architecture signed_adder_arch of signed_adder is
17   signal q_s : signed(a'high+1 downto 0); -- extra bit wide
18
19 begin -- architecture
20   assert(a'length >= b'length)
21     report "Port A must be the longer vector if different sizes!"
22     severity FAILURE;
23   q <= std_logic_vector(q_s);
24
25   adding_proc:
26   process (aclr, clk)
27     begin
28       if (aclr = '1') then
29         q_s <= (others => '0');
30       elsif rising_edge(clk) then
31         q_s <= ('0'&signed(a)) + ('0'&signed(b));
32       end if; -- clk'd
33     end process;
34
35 end signed_adder_arch;
```

RTL Code in a HDL

instantiated by the designer in the RTL code



Standard Cells
used by synthesis tools to implement the RTL code with logic gates

IP Macros
ready made components for complex or critical functions

Standard Cells

- **Standard-cells are the basic components for logic synthesis and P&R**
  - Always in the form of hard-IP
  - **Very basic functionality that can be understood by automatic synthesis tools**
  - Precise and detailed characterization across many design corners to enable timing and power analysis of large netlists

- **As opposed to IP components, standard cells are rarely not explicitly instantiated, but inferred automatically during synthesis**

- **Standard cells are usually collected in libraries containing**
  - Standard-cell libraries often contain hundreds of cells
    - Logic gates for Boolean logic
    - Sequential elements
    - Special purpose cells

Guidelines for a Good Core Library

- **A variety of drive strengths for all cells**
  - Larger varieties of drive strengths for inverters and buffers
  - Cells with balanced rise and fall delays (for clock tree buffers/gated clocks)
  - (Same logical function and its inversion as separate outputs, within same cell)
- **Complex cells (e.g. AOI, OAI)**
  - High fan-in cells: Using high fan-in reduce the overall cell area, but may cause routing congestion inadvertently causing timing degradation. Therefore they should be used with caution
- **Variety of flip-flops, both positive and negative edge triggered, preferably with multiple drive strengths**
  - Single or Multiple outputs available for each flip-flop (e.g. Q only, or Qbar only or both), preferably with multiple drive strengths
  - Flops to contain different inputs for Set and Reset (e.g. Set only, Reset only, both)
  - To enable scan testing of the designs, each flip-flop should have an equivalent scan flop
- **Variety of latches, both positive and negative level sensitive**
- **Several delay cells. Useful for fixing hold time violations**

Special Standard Cell Libraries

- **Basic cell libraries are often complemented by libraries with special cells**
  - Typically not instantiated automatically during synthesis: manual instantiation or instantiated by special tools for specific purposes

- **Special gates and cells**
  - Clock gating cells
  - Balanced clock buffers

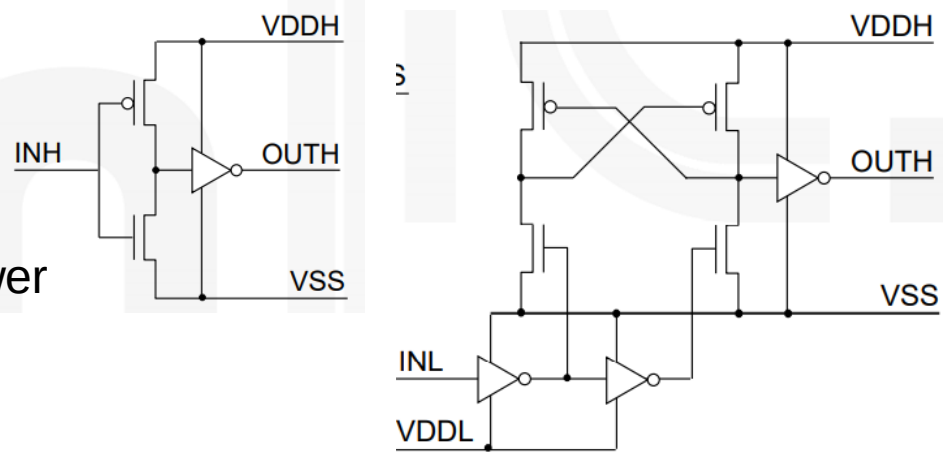
  } Clock network design: used during backend

  - Level shifters
  - Isolation cells
  - Retention registers
  - Power switches

  } Design with multiple voltage domains, mostly for low power

  - ECO cells for mask fixes to repair errors with limited cost

  } Special cells for debug and post-fabrication repair used in industrial designs



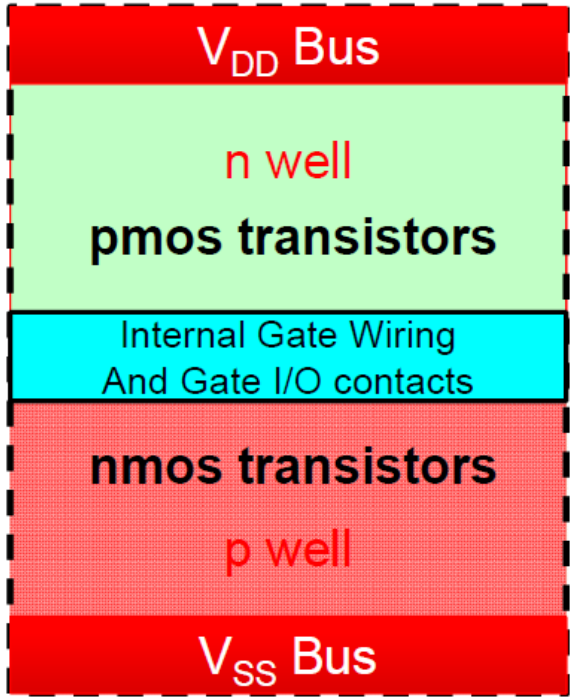
Standard Cell Design Requirements

- **Standard cells are the smallest building block of complex logic**
  - Each cell is instantiated thousands or millions of times
  - Cells used in very different contexts (different drivers, different loads, different layout locations)
- **Standard cells must be simple and robust** by design
  - Standard cells follow a very conservative design paradigm
  - Stability across corners and electrical conditions is a plus
- Cells must be able to handle different electrical environments:
  - Different loads require different drive strength
  - Cells are fixed and can not be re-sized individually
  - **Libraries contain the same standard cell function with many drive strengths** (sizes)
    - Drive strength variants are often called X1, X2, X4, …
  - **Different** libraries often can be used together offering different **VT options**

Standard Cell Physical Design

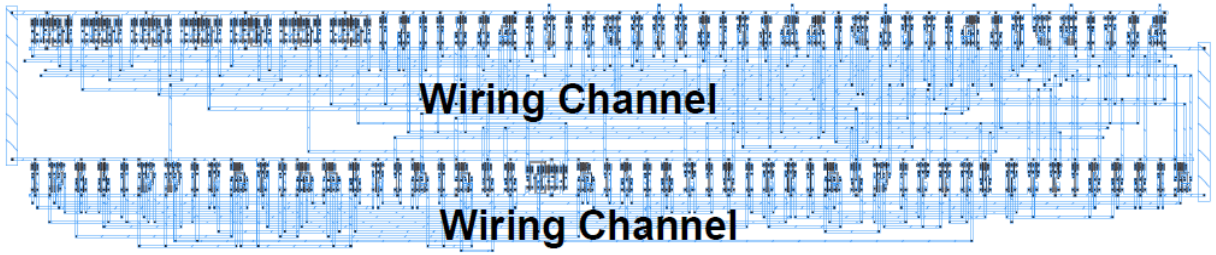
- **Semicustom layout is a collection of thousands of standard cells**
  - EDA tools need to be able to place cells quickly in a dense fashion

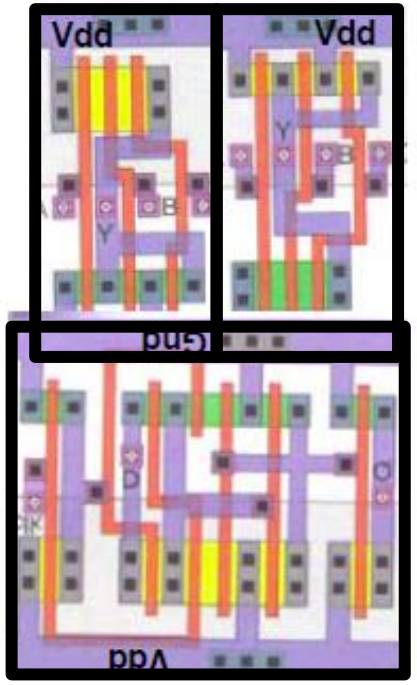
- **Standard cells follow a very regular layout with defined guidelines**
  - All cells have the same hight
  - Only width varies depending on the complexity of the cell
  - Power and gorund connections run horizontally with same hight
  - Cells can abut without DRC violations
  - PMOS on one side, NMOS on other side
  - Ideally use only few layers to leave other layers free for cell-to-cell routing



Standard Cell Physical Design

- **Layout with standard cells**
  - Cells are placed in rows
  - Cells in a row abut horizontally
- **Horizontal stacking (up to 350nm):**
  - Routing channels between cell rows

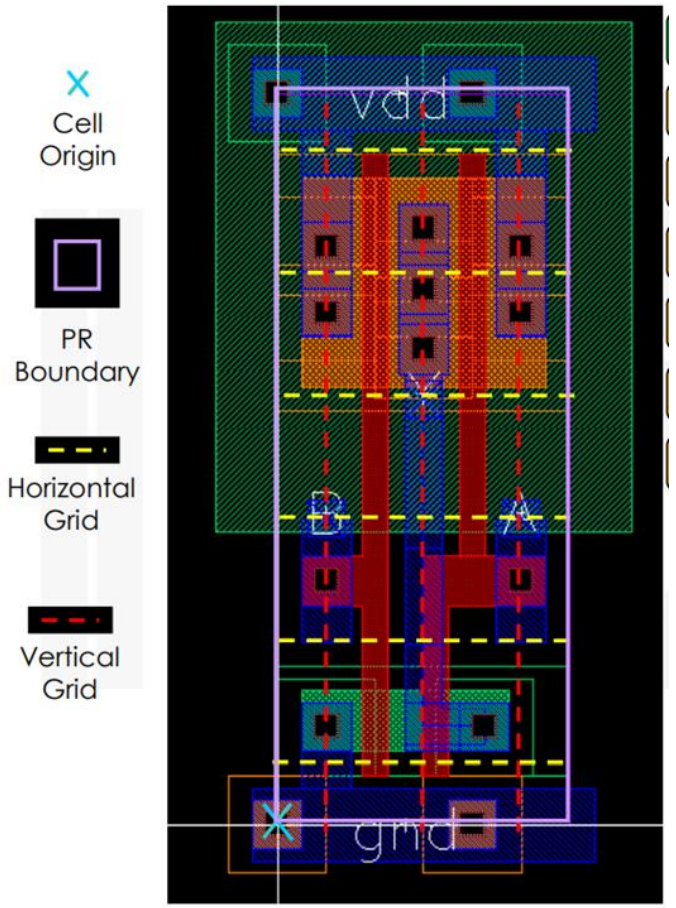




  - 250nm and below: horizontal stacking of cells by flipping every second row upside down
    - Avoids the need for vertical spacing between rows
    - Power and ground connections can be shared

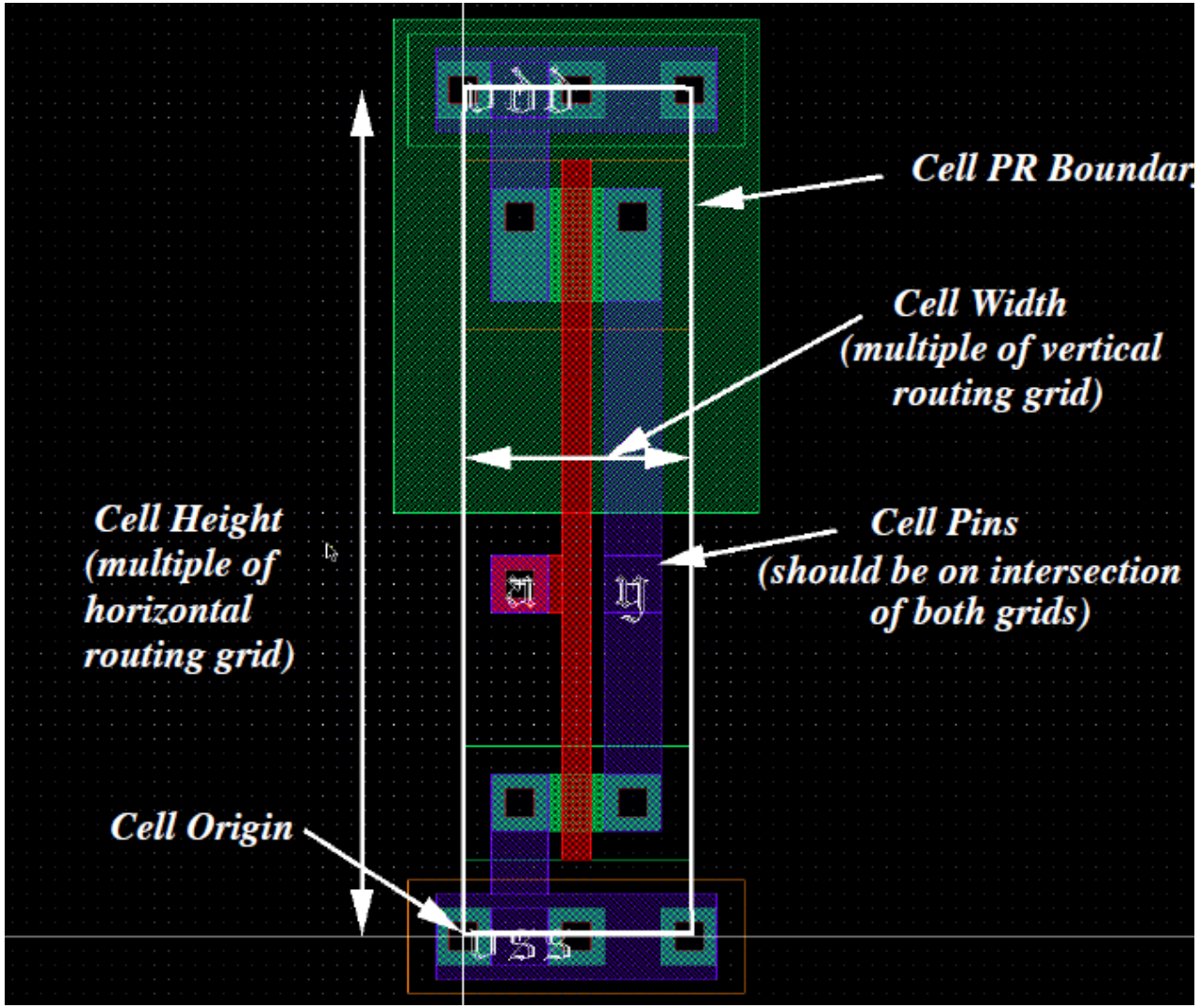
Standard Cell Layout – Pins / Grids

- **Standard cell layouts are routed on routing grids**
  - Both vertical and horizontal routing grids need to be defined
  - HVH or VHV routing is defined for alternating metals layers
  - All standard cell pins should ideally be placed on intersection of horizontal and vertical routing grids
  - Exceptions are abutment type pins (VDD and GND)
  - Grids are defined w.r.t. the cell origin
  - Grids can be offset from the origin, however by exactly half the grid spacing
  - The cell height must be a multiple of the horizontal grid spacing
  - All cells must have the same height, but some complex cells can be designed with double height
  - The cell width must be a multiple of the vertical grid spacing
  - However, limited routing tracks are the bottleneck even with wider cells



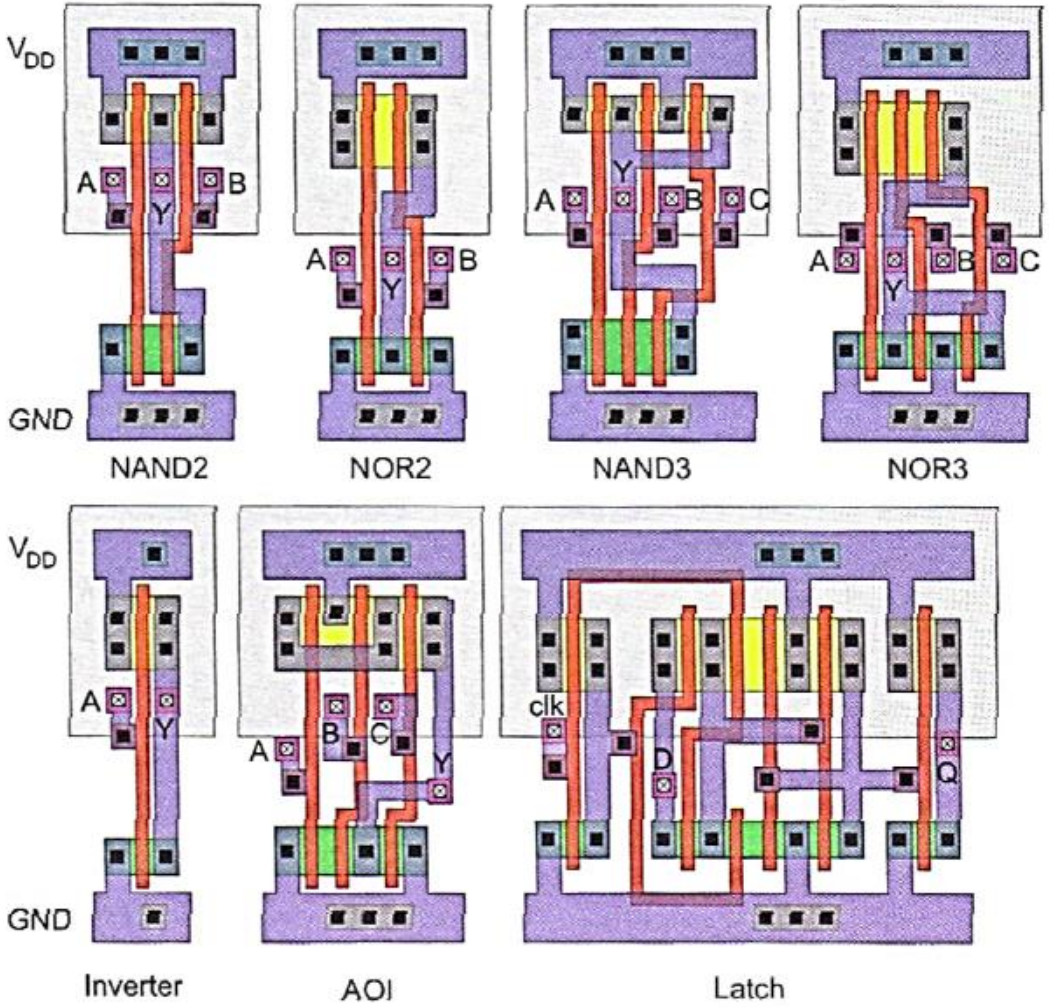
- **Cell libraries are often named based on the number of routing grids per cell height (e.g., 8 track or 12 track)**

Standard Cell Layout - Example



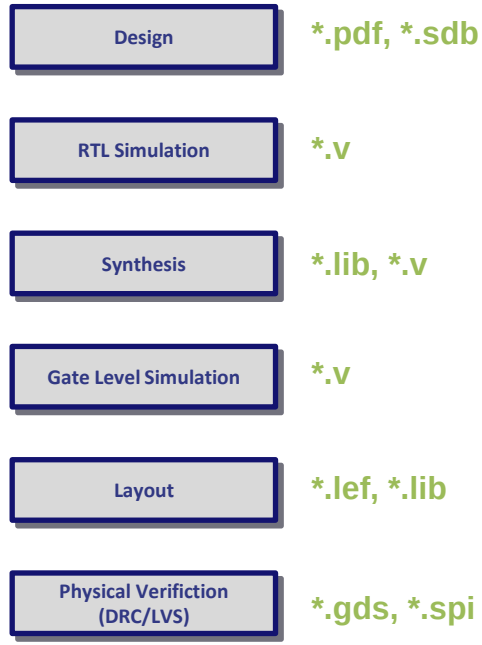
Standard Cell Layout - Example

- **Some more standard cell layouts**



Design Views for Standard Cells (and IP Macros)

- **Efficiently using Standard Cells and IPs requires abstraction**
  - Different properties and different information required for different steps in the design flow
- **IP and standard-cells come with different (design) views (abstract models)**
  - **Datasheet** (*.pdf)
  - Graphical **icon** for use in schematics (*.sdb)
  - **Behavioral model** for simulation (*.v or *.vhd)
  - **Timing and functional view** for synthesis and P&R (*.lib, *.db)
  - Port definition/entity declaration (*.v)
  - Model for timing simulation (*.v or *.vhd)
  - An **abstract layout view** for semi-custom layout (*.lef)
  - Transistor level (spice) **netlist** (or schematic (*.spi, *.cds, *.oa))
  - A **detailed layout** for Virtuoso or other tools (*.gds, *.oa)
  - and many others …



Complex IP Macros

- **Complex sub-circuits are often packaged as IP macros**
  - They may already be available and tested
  - They may contain full-custom or carefully optimized circuits that should not be touched

- **IP macros can be full-custom circuits or blocks designed on RTL**

- **Two types of IP components:**
  - Soft IP: provided as RTL code
    - Used to include other RTL designs
    - Advantages: flexible to be adapted to fit the design in which it is used
    - Drawback: limited to RTL and often not fully optimized until the last step
    - Examples: processors, interfaces, DSP circuits, arithmetic units, …
  - **Hard IP (macros):** ready-made layouts
    - Often used to include full-custom designs
    - Advantages: highly optimized and often silicon proven
    - Drawback: can not be altered or co-optimized with the design
    - Examples: SRAMs, ROMs, PLLs, Analog or RF components, …

Standard Cells vs. (Hard) IP Macros

- **Note: soft-IP macros are treated in almost same way as your own HDL code**
  - A soft macro is a hierarchical block that is itself eventually built from standard cells and possibly from Hard IP Macros

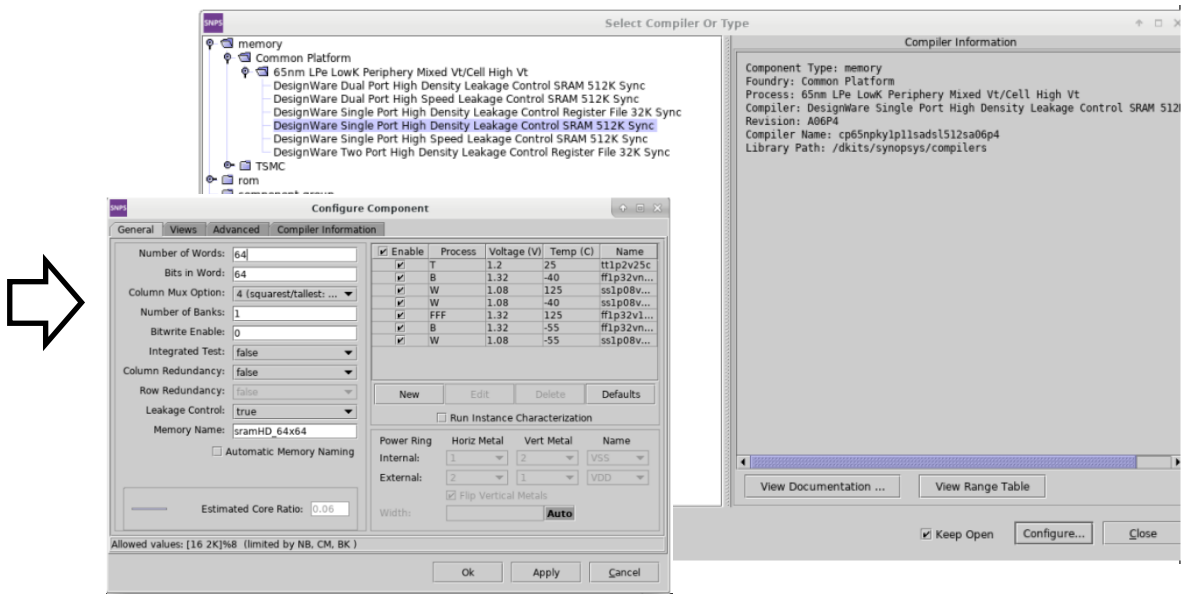
- **There is almost no difference between a standard cell and Hard IP Macro**
  - Both have similar or even identical set of “views”
  - With very few exceptions, both are treated in the same way in the design flow
- **Difference between Hard-IP and Standard Cells lies in how they are used:**

Standard Cells	**Hard IP Macros**
• Basic logic/sequential elements: sufficiently basic to be understood by synthesis tools	• Complex functionality: can not be understood by synthesis tools
• Instantiated by logic synthesis to implement the RTL description with logic gates	
• (can in rare cases be instantiated manually)	• Always instantiated manually by the designer

IP Macro Example: Memories

- **Embedded memories are the most frequently used IP in digital designs**
  - Often many memories with different configurations (based on design requirements)
- **Memories are typically generated by a "Memory Compiler"**
  - Software tool that takes specifications and generates corresponding macro
  - Usually based on basic building blocks & algorithms to generate all required views

	• Memory type (e.g., Single Port, Dual Port, Two Port, ...) Memory flavour (e.g., High Density, Low Power, High Speed, ...) • #words & bits/word • Physical structure: MUX factor/folding, banks, ... • Test options • Control options • ...



- **Datasheet** (*.pdf)
- Graphical **icon** for use in schematics (*.sdb)
- **Timing and functional view** for synthesis and P&R (*.lib, *.db)
- **Behavioral model** for simulation (*.v or *.vhd)
- An **abstract layout view** for semi-custom layout (*.lef)
- Model for timing simulation (*.v or *.vhd)
- Transistor level (spice) **netlist** (or schematic (*.spi, *.cds, *.oa))
- A **detailed layout** for Virtuoso or other tools (*.gds, *.oa)
- Port definition/entity declaration (*.v)

Integration of IP Macros on RTL

- **IP Macros (hard or soft) are instantiated in the RTL code**
  - Exactly like instantiating your own hierarchical components
- **Component definition can be**
  - part of your RTL code, based on the datasheet and must match the ports (names, types, direction) of the IP
  - in a package that is part of the IP deliverable that you can simply include in your RTL code

```
architecture rtl of seqmemwrap is

  ------------------------------------------------
  -- component declarations
  ------------------------------------------------
  component SYAA90_128X42X1CM2
    port (
      DO  : OUT std_logic_vector (41 downto 0);
      A   : IN  std_logic_vector (6 downto 0);
      DI  : IN  std_logic_vector (41 downto 0);
      WEB : IN  std_logic;
      CK  : IN  std_logic;
      CSB : IN  std_logic);
  end component;
```

Component declaration
(sometimes available in a HDL package)

IP Macro name

```
  -- memory instantiation
  ramwrap : SYAA90_128X42X1CM2
    port map (
      DO  => DOxD,
      A   => std_logic_vector(AddrxDI),
      DI  => DIxD,
      WEB => WriteEnxSBI,
      CK  => ClkxCI,
      CSB => MemEnxSBI);

end rtl;
```

Component instantiation

Semicustom Design Flow Tools

Design Entry

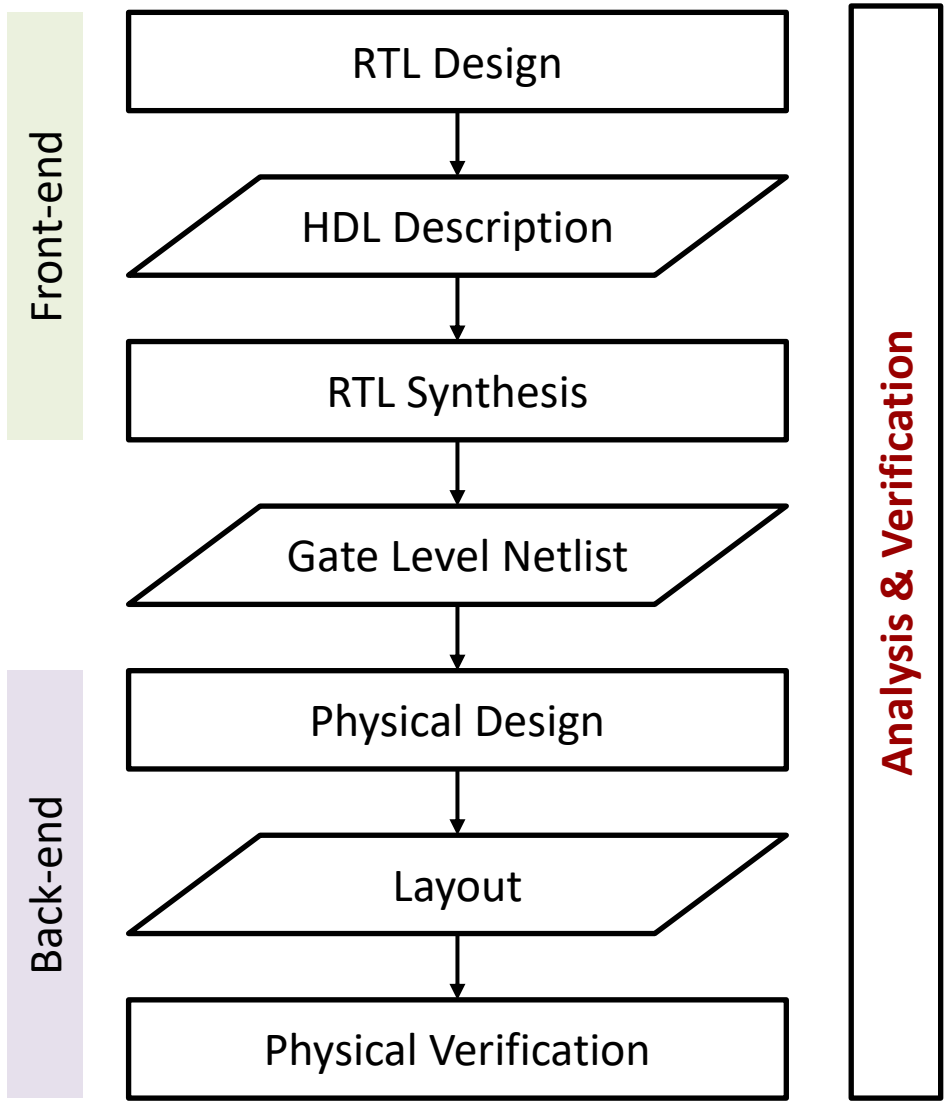
- VS Code
- EMACS

Synthesis

- Synopsys Design Compiler
- Cadence Genus
- Siemens EDA LeonardoSpectrum

Place & Route

- CADENCE Innovus
- Synopsys IC Compiler
- Siemens EDA Olympus SoC



Verification

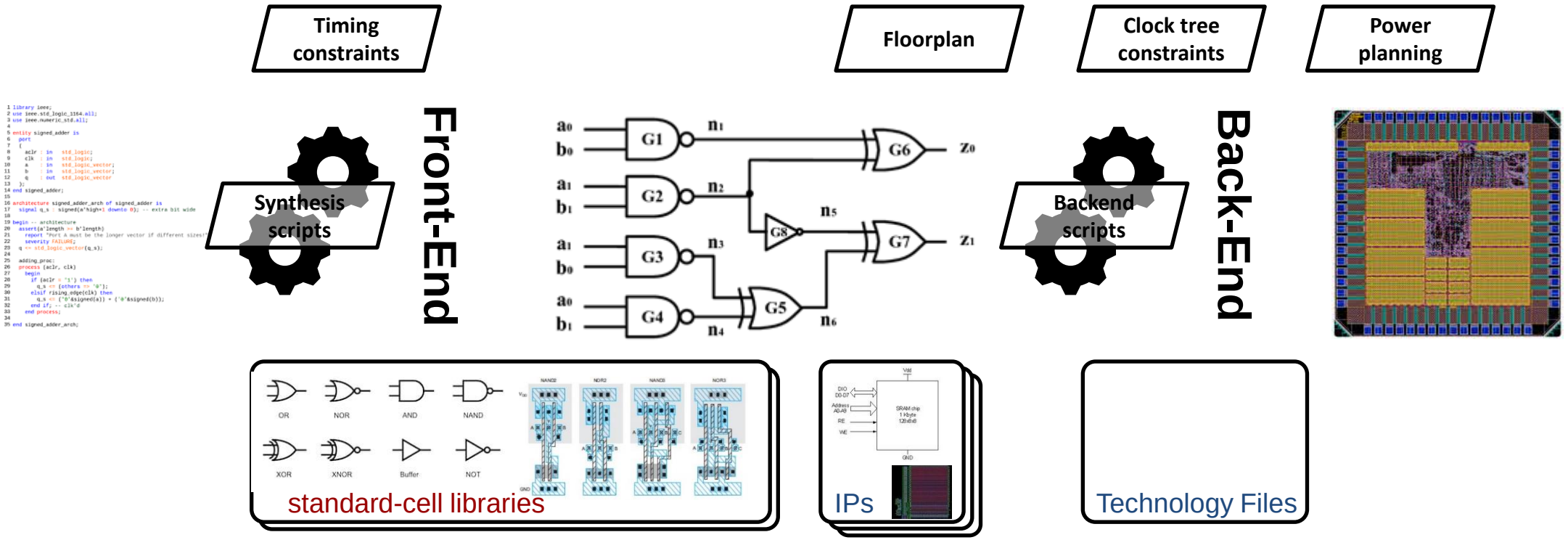
- Siemens EDA Questa Sim
- Synopsys VCS
- CADENCE Incisive
- Synopsys Formality
- CADENCE Conformal
- Siemens EDA QuestaSLEC

Physical Verification

- Siemens EDA Calibre

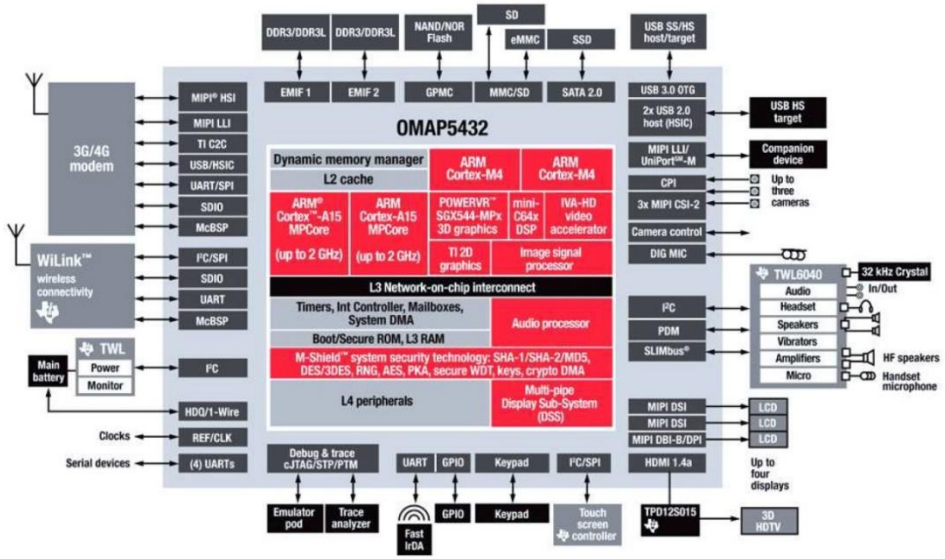
Design Flow: Required Inputs

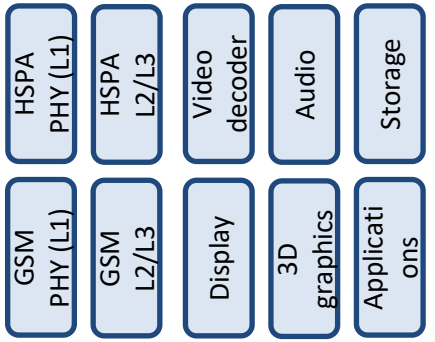
- **Each step requires different inputs and produces outputs**
  - Design data base from the previous step
  - Additional information and constraints
  - Commands that tell the tool what to do
  - Technology files with technology information



Design Definition

- **Chip design projects start from a specification**
  - Description of the functionality
  - Algorithms
  - Performance requirements
  - Golden models in a high-level language
- **Chip specifications & requirements**
  - Description of the application and environment (system in which the design is used)
  - IO and interface requirements
- **Process technology and fab**
- **Chip architecture definition**
  - Defining the main functional components and tasks
  - Partitioning of the functionality
  - Connectivity between main components

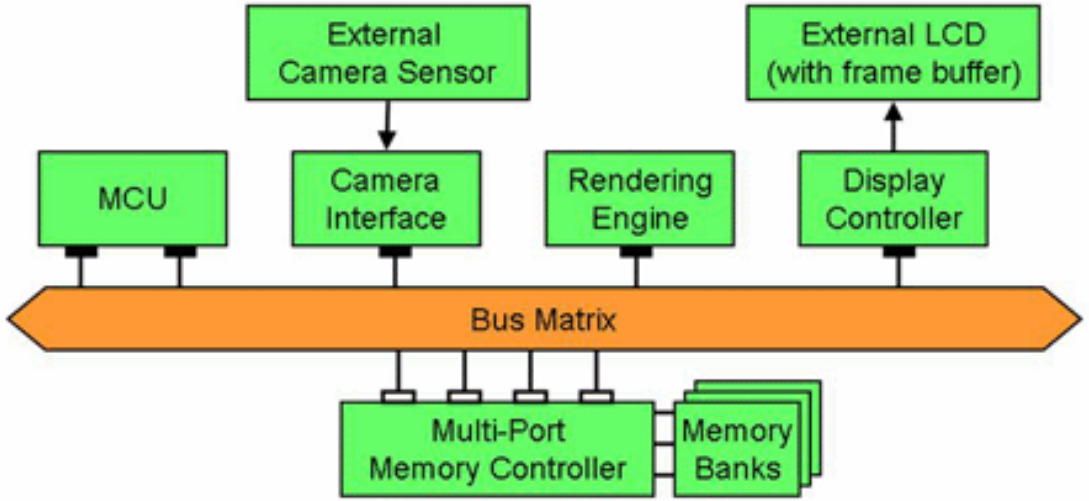




Functional view

Architecture Design

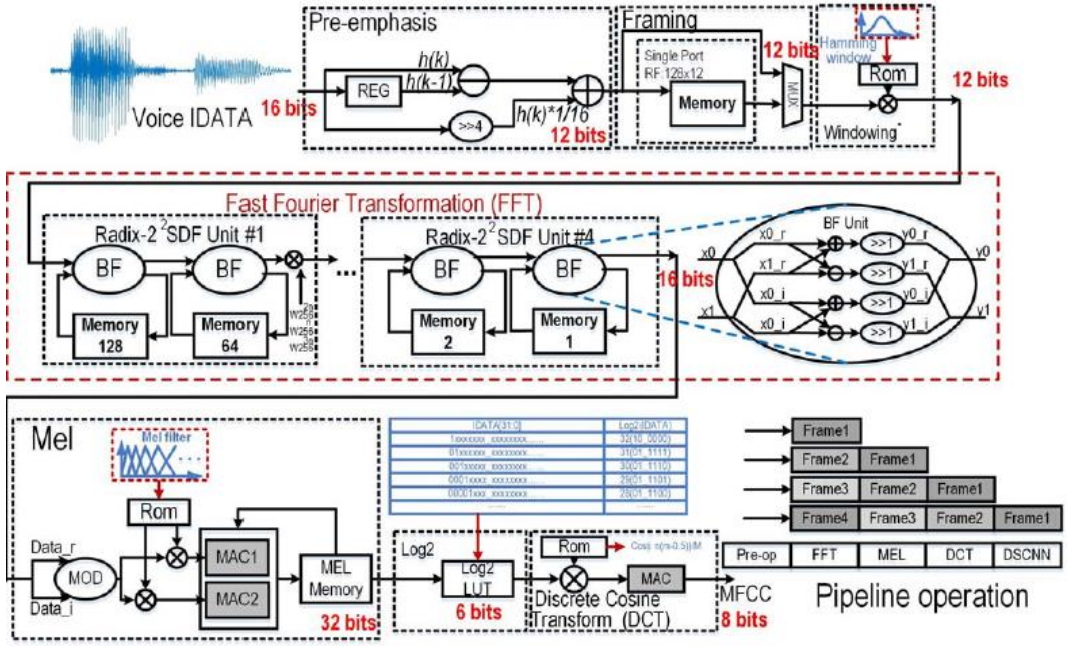
- **Top level architecture definition**
  - **Top level block diagram** with detailed partitioning
  - Definition of the individual sub-blocks
  - Definition of the block-level interfaces
- **Top-level clocking strategy**
  - Definition of system clocks



- **Block level specification and modelling**
  - Define clearly the functionality for each block
  - Ideally provide a golden model for each critical block
- **Identification of key IP components**
  - Identify standard-cell libraries and technology details
  - Identify main required IP components (e.g., memories) for each sub-block

RTL Design

- **Detailed RTL Diagrams**
  - RTL block diagrams
  - State machines
  - Waveforms for interfaces
- **Refinement of clocking strategies**
  - Special clocks on block level
- **RTL design using synchronous design**
  - Implementation of RTL diagrams in a HDL
  - Typically in Verilog or VHDL



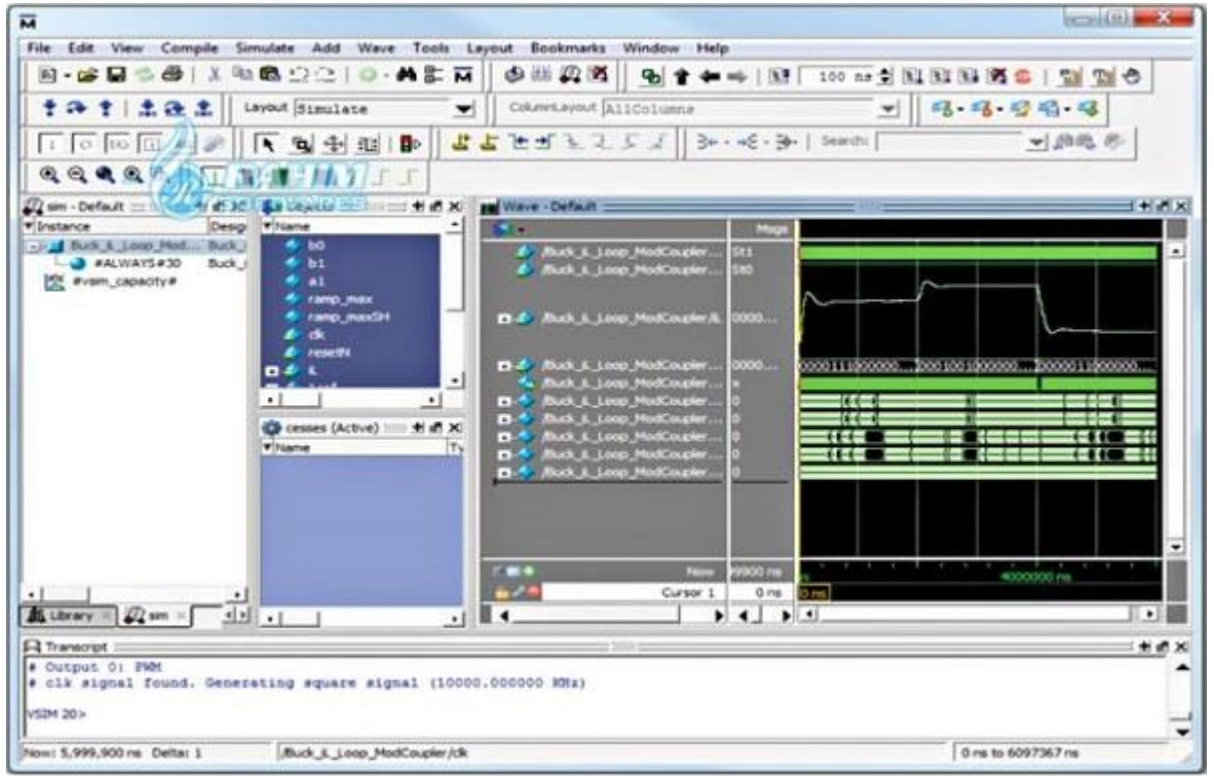
Functional RTL Verification

- **Objective: verify that each block functionally adheres to its specification**

- **Functional verification is mostly performed based on simulations**
  - Definition of test cases
  - Definition of test benches
  - Simulation of RTL code

- **Debugging based on simulation waveforms**

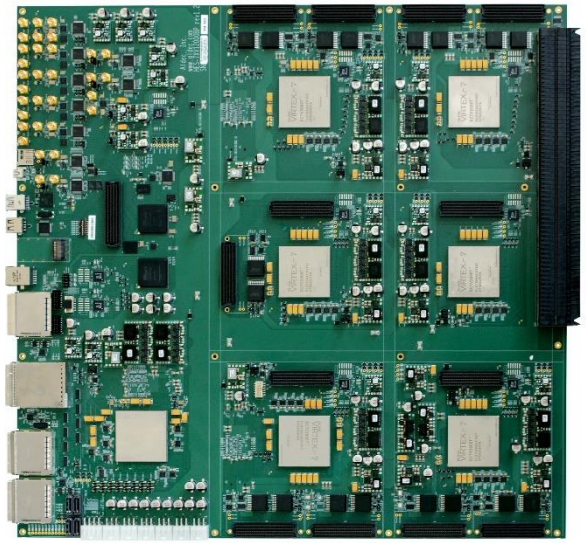
- **Verification starts on RTL level, but is repeated later on the implemented design (gate level)**



Verification Acceleration

- **Complex simulations can be supported by hardware**
  - FPGAs: map the entire design onto FPGAs, sometimes enabling even real-time in-system simulation
  - Hardware emulators (CADENCE Palladium): hardware platforms for simulation speedup

- **Drawback of hardware support:**
  - Visibility for debug can be quite limited
  - Compiling code for accelerated simulation can take quite some time



Logic Synthesis

- **Objective: mapping of RTL code to a gate-level netlist**
- **Inputs:**
  - RTL design files
  - Standard cell libraries
  - Models (views) for hard IPs
  - Constraint files
  - Synthesis commands
- **Outputs:**
  - Gate-level netlist
  - Side information for physical design
  - Reports on area, timing, power, ...

Synthesis steps

- **Analysis**
  - Read and check HDL description
- **Elaboration**
  - Translate code into an abstract schematic
- **Optimization**
  - Optimize design for an area/performance target
  - Various algorithms
- **Technology mapping**
  - Map abstract design to the target libraries
- **Post synthesis analysis**
  - Formal verification
  - Area analysis
  - Static timing analysis
  - Power analysis

Physical Design

- **Objective: implement a physical layout of the chip while adapting the design to the layout and complementing it with missing parts**
- **Inputs:**
  - Verilog netlist
  - Standard cell libraries
  - Models (views) for hard IPs
  - Technology files (PDK)
  - Constraint files (e.g., timing, clock) Floorplan instructions
  - Power supply information
- **Outputs:**
  - Physical design layout (.gds file)
  - Optimized gate-level netlist
  - Reports on area, timing, power, ...

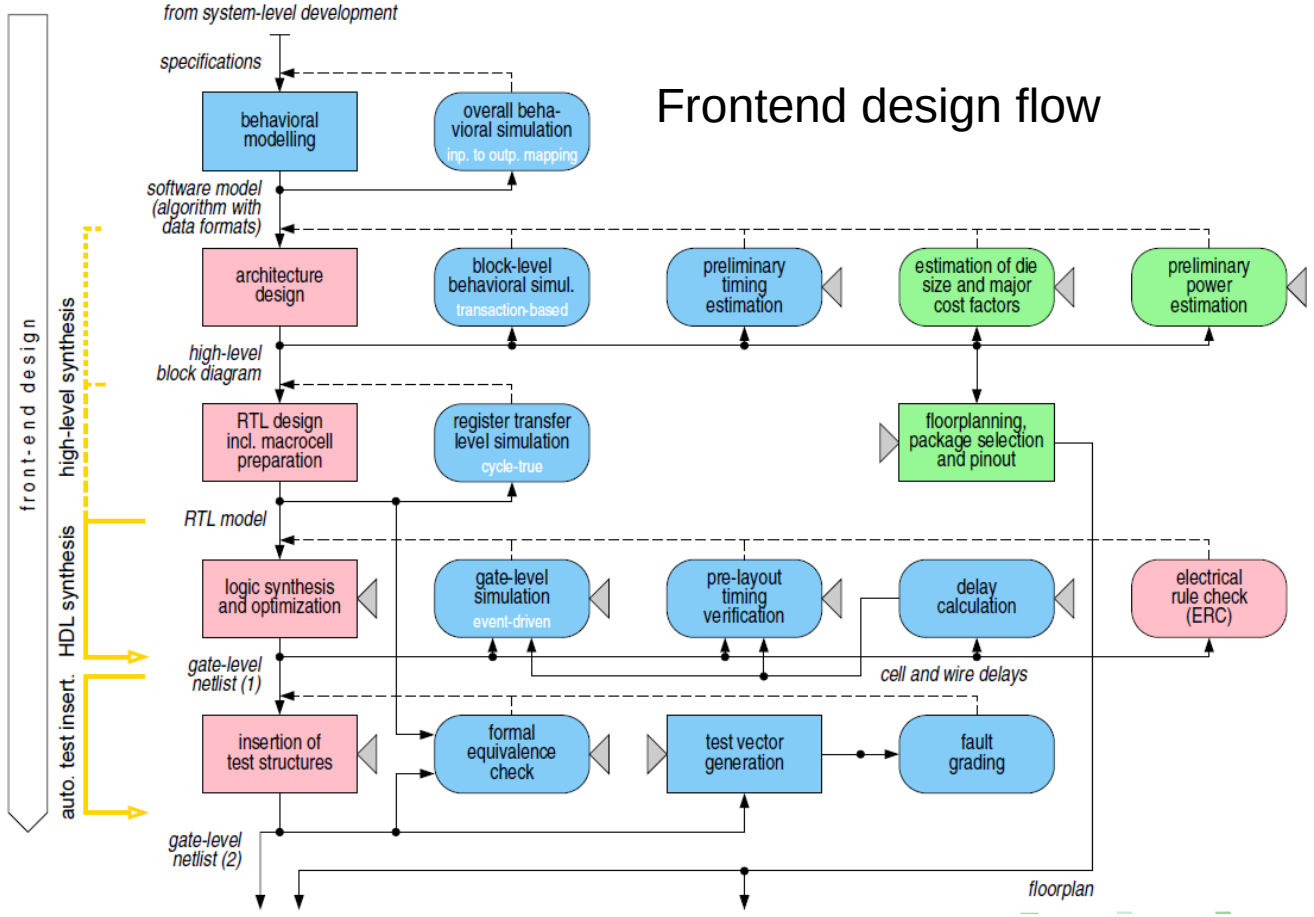
- **Setup**
  - Read netlist and technology files
- **Floorplan**
  - Abstract plan of the layout
  - Placement guidance and placement of macros
  - IO placement
- **Automatic placement**
  - Place standard cells
- **Design optimization**
  - Optimize design based on parasetics
- **Clock tree design**
  - Define and synthesize a clock tree
- **Routing**
  - Route power
  - Route signals
- **Post-route analysis**
  - Check physical rules

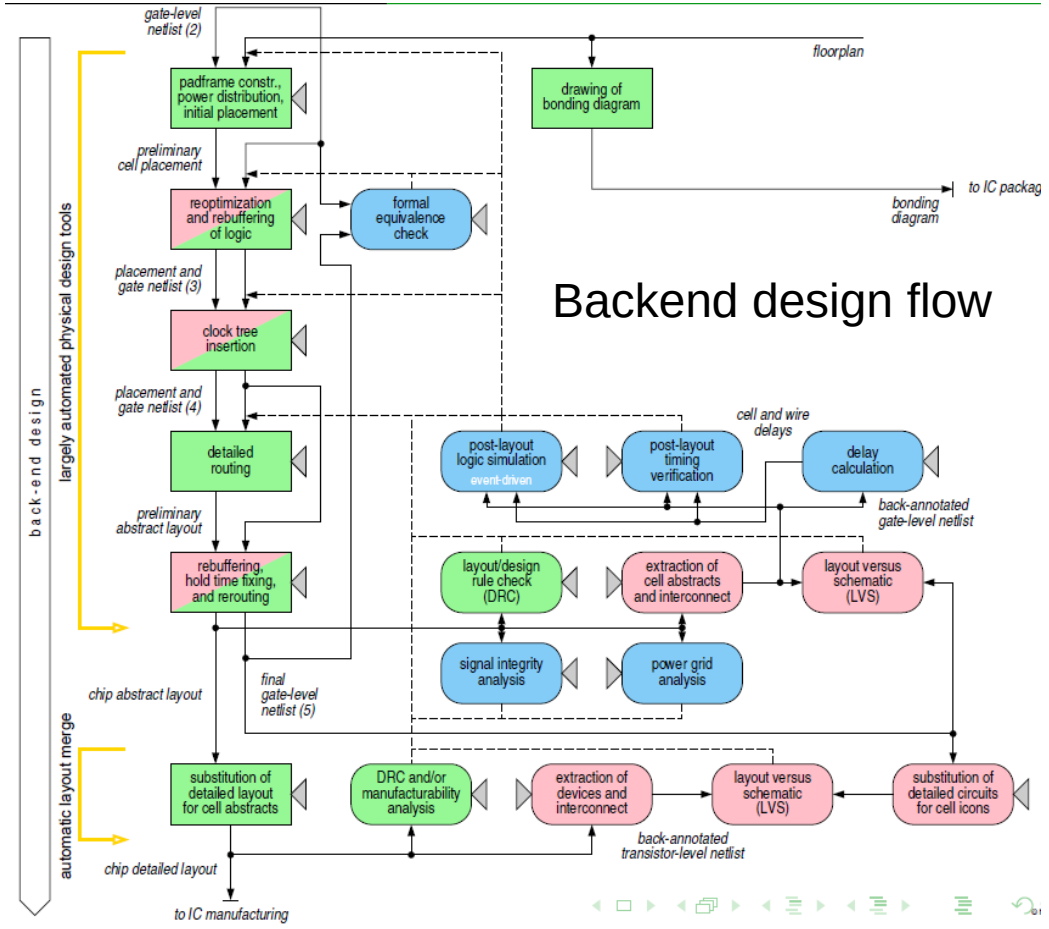
Chip-Level Integration

- **Eventually, analog and digital blocks need to be combined**
- **Two strategies:**
  - **Analogue-on-top**: package digital components to hide complexity and include in analog top
  - **Digital-on-top**: package analog components to hide details and include in digital top level

Abstract Detailed Design Flow

- **Generic Abstract Design Flow defines all the steps of the design process**
  - Characterized by “loops” and design iterations between design, verification, and analysis

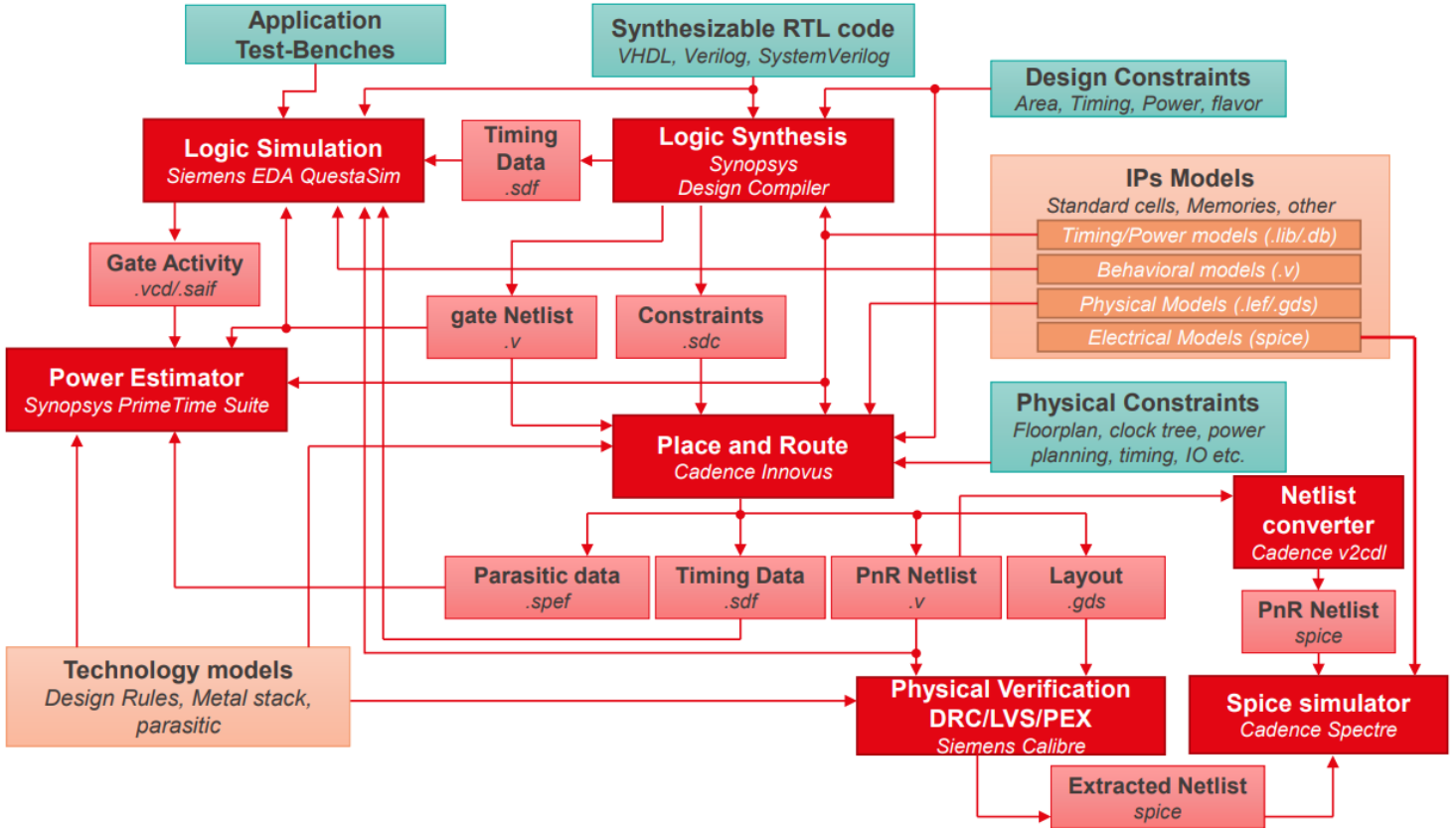




Specific Design Flow Example

- **For every project, we define a specific more concrete design flow**
  - Defines specific tools and the exchange of information based on design files
  - Each tool requires its own set of inputs and technology and IP information (views)
- **Details of the design flow are defined by the EDA/CAD support groups and by the project and are the same for every designer**

Design flow example from the lab you see in 1-2 weeks based on UMC 65nm



Setting Up a Design Environment

- **Need for a clean and structured work environment**
- **The semicustom design flow is based on a collection of tools**
  - Each tool has its own environment, requires inputs, delivers outputs and generates numerous intermediate files and reports
- **Every design starts with a file/directory structure**
  - Avoid chaos and allow the tools to run independently and in their preferred manner
- **Defined by the project (same for all designers) and often common or similar for all projects in a company**
  - Typically given by the EDA / CAD support team

Work environment example from EDA-Labs UMC 65nm

```
edalabs_phase2_2023/
├── DesignCompiler - - - Synopsys design compiler workspace for logic synthesis
├── edadk.conf     - - - configuration file used in EPFL for EDA tools
├── EMBEDIT        - - - Memory compiler folder - used to generate memory IPs
├── HDL            - - - VHDL/Verilog source files
├── IPS            - - - Hardware IPs : standard cells, memories
├── LibraryCompiler- - - Synopsys Library compiler folder - used to compile lib into db files
├── PrimeTime      - - - Synopsys Prime Time working folder
└── QUESTASIM      - - - Siemens QuestaSim logic simulator folder
```