BECE407E ASIC DESIGN

Dr. PRITAM BHATTACHARJEE

Assistant Professor (Senior Grade 2)

School of Electronics Engineering (SENSE)

Vellore Institute of Technology – Chennai

pritam.bhattacharjee@vit.ac.in, +91 8132863424

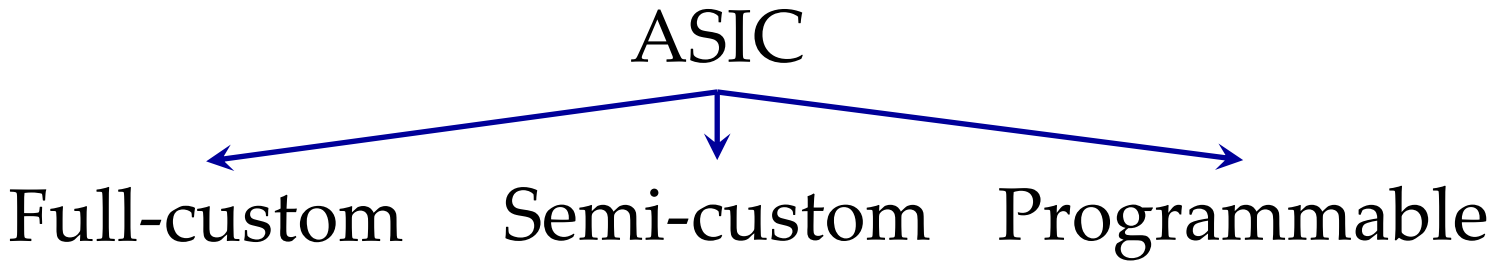
Contents

1) Introduction
2) From Custom to Semicustom and Structured-Array Design Approaches
3) Custom Circuit Design
4) Cell-Based Design Mythology
    - Standard Cell
    - Compiled Cells
    - Macrocells, Megacells, and Intellectual Property
    - Semi-custom Design Flow
5) Array-Based Implementation Approaches
    - Prediffused (or Mask-Programmable) Arrays
    - Prewired Arrays

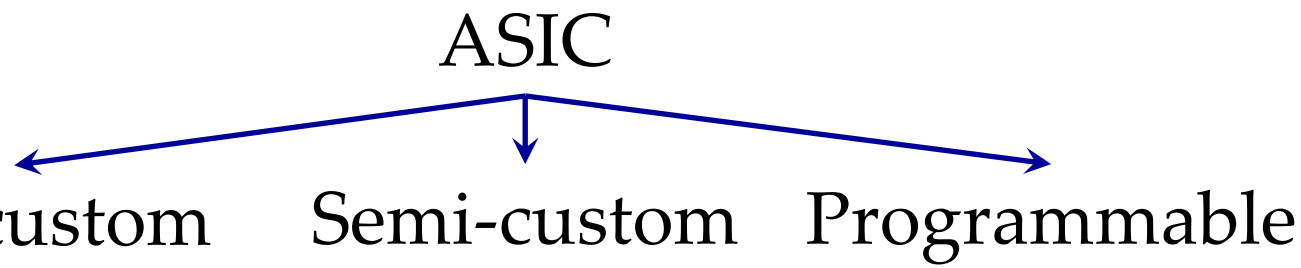
Introduction

- Application-Specific Integrated Circuit (ASIC) is a collection of logic and memory circuits on a single die.
- They are used in a wide variety of products ranging from consumer products such as video games, digital cameras, automobiles, and personal computers, to high-end technology products such as workstations and supercomputers.
- ASICs are logic chips designed by the end-customers to perform a specific function and thereby meet the specific needs of their application.
- Customers implement their designs in a single silicon die by mapping their functions to a set of predesigned, preverified logic circuits provided by the ASIC vendor which are often referred to as the ASIC vendor's library. These circuits range from the simplest functions, such as inverters, NANDs and NORs, flip-flops and latches, to more complex structures such as static memory arrays, adders, counters and phase-lock loops.
- The ASIC design process typically involves several stages of its cycle, including specification, architecture design, RTL design, functional verification, physical design, and manufacturing.

Introduction

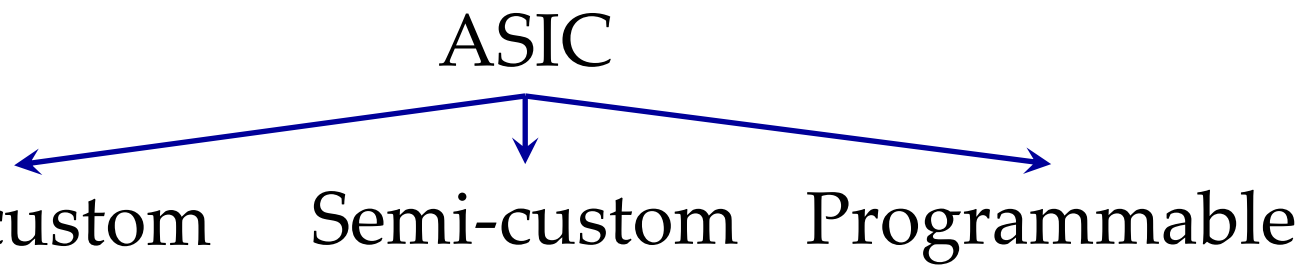


Introduction



Full Custom ASIC: Full Custom ASIC design involves designing the circuitry from scratch, including the layout of the transistors and interconnects. This approach provides the highest degree of customization and allows for the most optimized performance, but it also requires the most design time and expense.

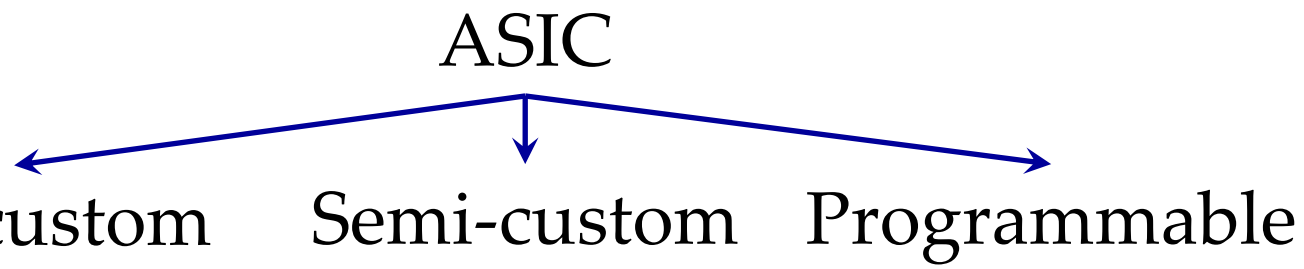
Introduction



Full Custom ASIC: Full Custom ASIC design involves designing the circuitry from scratch, including the layout of the transistors and interconnects. This approach provides the highest degree of customization and allows for the most optimized performance, but it also requires the most design time and expense.

Semi-Custom ASIC: Semi-Custom ASIC design involves using pre-designed functional blocks, also known as IP (Intellectual Property) blocks, and customizing the design to meet specific requirements. This approach allows for faster design time and lower development cost, while still providing a high level of customization and performance.

Introduction

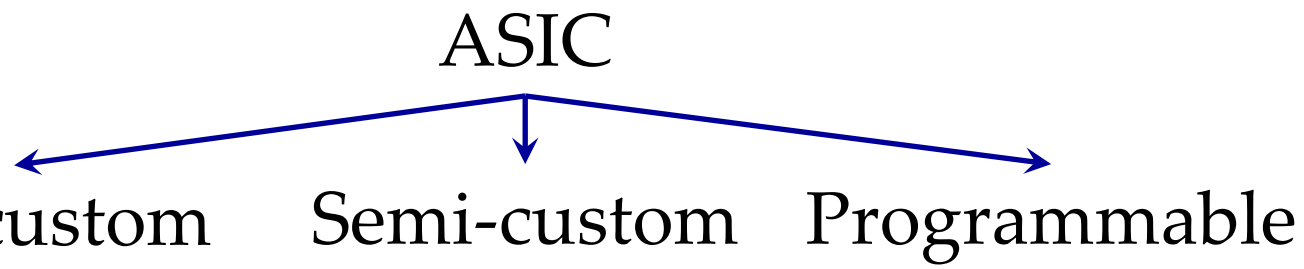


Full Custom ASIC: Full Custom ASIC design involves designing the circuitry from scratch, including the layout of the transistors and interconnects. This approach provides the highest degree of customization and allows for the most optimized performance, but it also requires the most design time and expense.

Semi-Custom ASIC: Semi-Custom ASIC design involves using pre-designed functional blocks, also known as IP (Intellectual Property) blocks, and customizing the design to meet specific requirements. This approach allows for faster design time and lower development cost, while still providing a high level of customization and performance.

Programmable ASIC: Programmable ASIC design involves using programmable logic devices, such as FPGAs (Field-Programmable Gate Arrays), to create an ASIC-like functionality. This approach provides the most flexibility and can be reprogrammed for different applications, but it may have lower performance and higher power consumption compared to custom ASIC designs.

Introduction



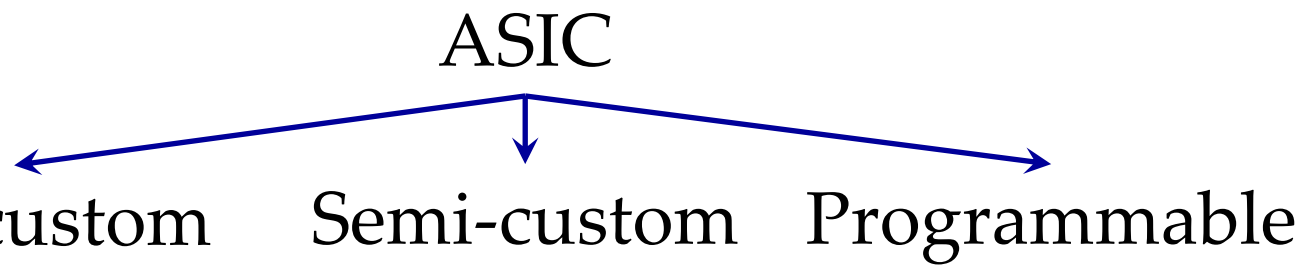
Full Custom ASIC: Full Custom ASIC design involves designing the circuitry from scratch, including the layout of the transistors and interconnects. This approach provides the highest degree of customization and allows for the most optimized performance, but it also requires the most design time and expense.

Semi-Custom ASIC: Semi-Custom ASIC design involves using pre-designed functional blocks, also known as IP (Intellectual Property) blocks, and customizing the design to meet specific requirements. This approach allows for faster design time and lower development cost, while still providing a high level of customization and performance.

Programmable ASIC: Programmable ASIC design involves using programmable logic devices, such as FPGAs (Field-Programmable Gate Arrays), to create an ASIC-like functionality. This approach provides the most flexibility and can be reprogrammed for different applications, but it may have lower performance and higher power consumption compared to custom ASIC designs.

The choice of ASIC design type depends on the specific application requirements, development time, and cost constraints. Full Custom ASIC design is typically used for applications that require the highest level of performance, while Semi-Custom ASIC design is used for applications that require customization at a lower cost and design time. Programmable ASICs are typically used for applications that require flexibility and rapid prototyping, but with lower performance and higher power consumption.

Introduction



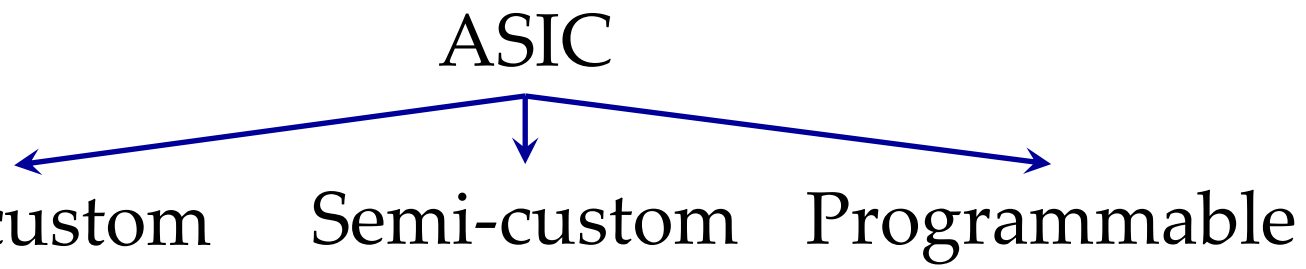
Full Custom ASIC: Full Custom ASIC design involves designing the circuitry from scratch, including the layout of the transistors and interconnects. This approach provides the highest degree of customization and allows for the most optimized performance, but it also requires the most design time and expense.

Semi-Custom ASIC: Semi-Custom ASIC design involves using pre-designed functional blocks, also known as IP (Intellectual Property) blocks, and customizing the design to meet specific requirements. This approach allows for faster design time and lower development cost, while still providing a high level of customization and performance.

Programmable ASIC: Programmable ASIC design involves using programmable logic devices, such as FPGAs (Field-Programmable Gate Arrays), to create an ASIC-like functionality. This approach provides the most flexibility and can be reprogrammed for different applications, but it may have lower performance and higher power consumption compared to custom ASIC designs.

The choice of ASIC design type depends on the specific application requirements, development time, and cost constraints. Full Custom ASIC design is typically used for applications that require the highest level of performance, while Semi-Custom ASIC design is used for applications that require customization at a lower cost and design time. Programmable ASICs are typically used for applications that require flexibility and rapid prototyping, but with lower performance and higher power consumption.

Introduction



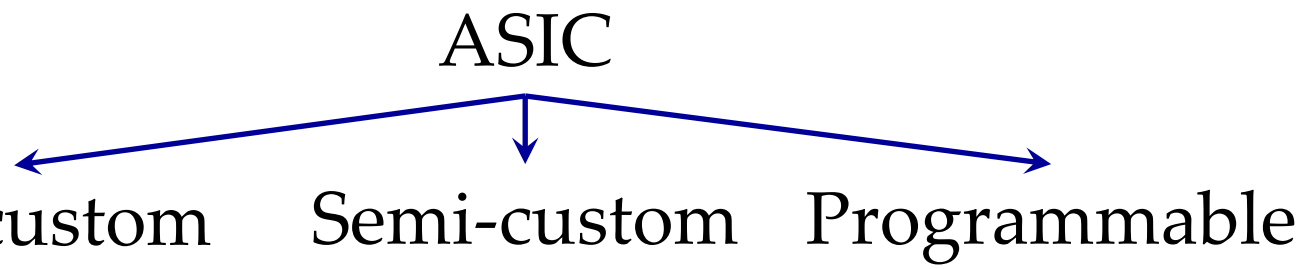
Full Custom ASIC: Full Custom ASIC design involves designing the circuitry from scratch, including the layout of the transistors and interconnects. This approach provides the highest degree of customization and allows for the most optimized performance, but it also requires the most design time and expense.

Semi-Custom ASIC: Semi-Custom ASIC design involves using pre-designed functional blocks, also known as IP (Intellectual Property) blocks, and customizing the design to meet specific requirements. This approach allows for faster design time and lower development cost, while still providing a high level of customization and performance.

Programmable ASIC: Programmable ASIC design involves using programmable logic devices, such as FPGAs (Field-Programmable Gate Arrays), to create an ASIC-like functionality. This approach provides the most flexibility and can be reprogrammed for different applications, but it may have lower performance and higher power consumption compared to custom ASIC designs.

The choice of ASIC design type depends on the specific application requirements, development time, and cost constraints. Full Custom ASIC design is typically used for applications that require the highest level of performance, while Semi-Custom ASIC design is used for applications that require customization at a lower cost and design time. Programmable ASICs are typically used for applications that require flexibility and rapid prototyping, but with lower performance and higher power consumption.

Introduction



Full Custom ASIC: Full Custom ASIC design involves designing the circuitry from scratch, including the layout of the transistors and interconnects. This approach provides the highest degree of customization and allows for the most optimized performance, but it also requires the most design time and expense.

Semi-Custom ASIC: Semi-Custom ASIC design involves using pre-designed functional blocks, also known as IP (Intellectual Property) blocks, and customizing the design to meet specific requirements. This approach allows for faster design time and lower development cost, while still providing a high level of customization and performance.

Programmable ASIC: Programmable ASIC design involves using programmable logic devices, such as FPGAs (Field-Programmable Gate Arrays), to create an ASIC-like functionality. This approach provides the most flexibility and can be reprogrammed for different applications, but it may have lower performance and higher power consumption compared to custom ASIC designs.

The choice of ASIC design type depends on the specific application requirements, development time, and cost constraints. Full Custom ASIC design is typically used for applications that require the highest level of performance, while Semi-Custom ASIC design is used for applications that require customization at a lower cost and design time. Programmable ASICs are typically used for applications that require flexibility and rapid prototyping, but with lower performance and higher power consumption.

Steps followed for ASIC Design

Steps followed for ASIC Design

Define the requirements: In this step, you need to define the requirements of the ASIC, including its functionality, performance, power consumption, size, and cost. You should also consider any constraints, such as the available technology, manufacturing process, and development budget.

Steps followed for ASIC Design

Define the requirements: In this step, you need to define the requirements of the ASIC, including its functionality, performance, power consumption, size, and cost. You should also consider any constraints, such as the available technology, manufacturing process, and development budget.

Create the specification: Once you have defined the requirements, you need to create a detailed specification that describes the functionality, input and output signals, and timing requirements of the ASIC. The specification should also include a block diagram that shows the various components and their connections.

Steps followed for ASIC Design

Define the requirements: In this step, you need to define the requirements of the ASIC, including its functionality, performance, power consumption, size, and cost. You should also consider any constraints, such as the available technology, manufacturing process, and development budget.

Create the specification: Once you have defined the requirements, you need to create a detailed specification that describes the functionality, input and output signals, and timing requirements of the ASIC. The specification should also include a block diagram that shows the various components and their connections.

Design the RTL: The next step is to design the Register Transfer Level (RTL) description of the ASIC using a Hardware Description Language (HDL) such as Verilog or VHDL. The RTL describes the functional behavior of the ASIC and the interconnection between its components.

Steps followed for ASIC Design

Define the requirements: In this step, you need to define the requirements of the ASIC, including its functionality, performance, power consumption, size, and cost. You should also consider any constraints, such as the available technology, manufacturing process, and development budget.

Create the specification: Once you have defined the requirements, you need to create a detailed specification that describes the functionality, input and output signals, and timing requirements of the ASIC. The specification should also include a block diagram that shows the various components and their connections.

Design the RTL: The next step is to design the Register Transfer Level (RTL) description of the ASIC using a Hardware Description Language (HDL) such as Verilog or VHDL. The RTL describes the functional behavior of the ASIC and the interconnection between its components.

Verify the RTL: After creating the RTL, you need to verify its functionality using simulation tools. Simulation helps to detect any logical errors or timing issues in the design. You can also use formal verification tools to ensure the correctness of the design.

Steps followed for ASIC Design

Define the requirements: In this step, you need to define the requirements of the ASIC, including its functionality, performance, power consumption, size, and cost. You should also consider any constraints, such as the available technology, manufacturing process, and development budget.

Create the specification: Once you have defined the requirements, you need to create a detailed specification that describes the functionality, input and output signals, and timing requirements of the ASIC. The specification should also include a block diagram that shows the various components and their connections.

Design the RTL: The next step is to design the Register Transfer Level (RTL) description of the ASIC using a Hardware Description Language (HDL) such as Verilog or VHDL. The RTL describes the functional behavior of the ASIC and the interconnection between its components.

Verify the RTL: After creating the RTL, you need to verify its functionality using simulation tools. Simulation helps to detect any logical errors or timing issues in the design. You can also use formal verification tools to ensure the correctness of the design.

Synthesize the netlist: Once the RTL is verified, you need to synthesize it into a netlist that describes the physical connections between the ASIC components. The netlist also includes timing constraints and other physical design requirements.

Steps followed for ASIC Design

Define the requirements: In this step, you need to define the requirements of the ASIC, including its functionality, performance, power consumption, size, and cost. You should also consider any constraints, such as the available technology, manufacturing process, and development budget.

Create the specification: Once you have defined the requirements, you need to create a detailed specification that describes the functionality, input and output signals, and timing requirements of the ASIC. The specification should also include a block diagram that shows the various components and their connections.

Design the RTL: The next step is to design the Register Transfer Level (RTL) description of the ASIC using a Hardware Description Language (HDL) such as Verilog or VHDL. The RTL describes the functional behavior of the ASIC and the interconnection between its components.

Verify the RTL: After creating the RTL, you need to verify its functionality using simulation tools. Simulation helps to detect any logical errors or timing issues in the design. You can also use formal verification tools to ensure the correctness of the design.

Synthesize the netlist: Once the RTL is verified, you need to synthesize it into a netlist that describes the physical connections between the ASIC components. The netlist also includes timing constraints and other physical design requirements.

Place and route: The next step is to place the components of the ASIC on the chip and route the connections between them. This step is critical to ensure that the design meets the timing and power requirements and to optimize the layout for manufacturability.

Steps followed for ASIC Design

Verify the physical design: After placing and routing the ASIC, you need to verify the physical design using tools that simulate the performance and power consumption of the ASIC. You should also perform design rule checks (DRC) and layout-versus-schematic (LVS) checks to ensure that the design meets the manufacturing requirements.

Steps followed for ASIC Design

Verify the physical design: After placing and routing the ASIC, you need to verify the physical design using tools that simulate the performance and power consumption of the ASIC. You should also perform design rule checks (DRC) and layout-versus-schematic (LVS) checks to ensure that the design meets the manufacturing requirements.

Generate the GDSII file: Finally, you need to generate the GDSII file, which is the industry-standard format for the physical layout of the ASIC. The GDSII file can be used to fabricate the ASIC in a semiconductor foundry.

Steps followed for ASIC Design

Verify the physical design: After placing and routing the ASIC, you need to verify the physical design using tools that simulate the performance and power consumption of the ASIC. You should also perform design rule checks (DRC) and layout-versus-schematic (LVS) checks to ensure that the design meets the manufacturing requirements.

Generate the GDSII file: Finally, you need to generate the GDSII file, which is the industry-standard format for the physical layout of the ASIC. The GDSII file can be used to fabricate the ASIC in a semiconductor foundry.

Tape-out: This is the final step of the ASIC design process, which involves submitting the GDSII file to a semiconductor foundry for fabrication. The foundry will manufacture the ASIC according to the GDSII file and the specifications provided by the design team.

Steps followed for ASIC Design

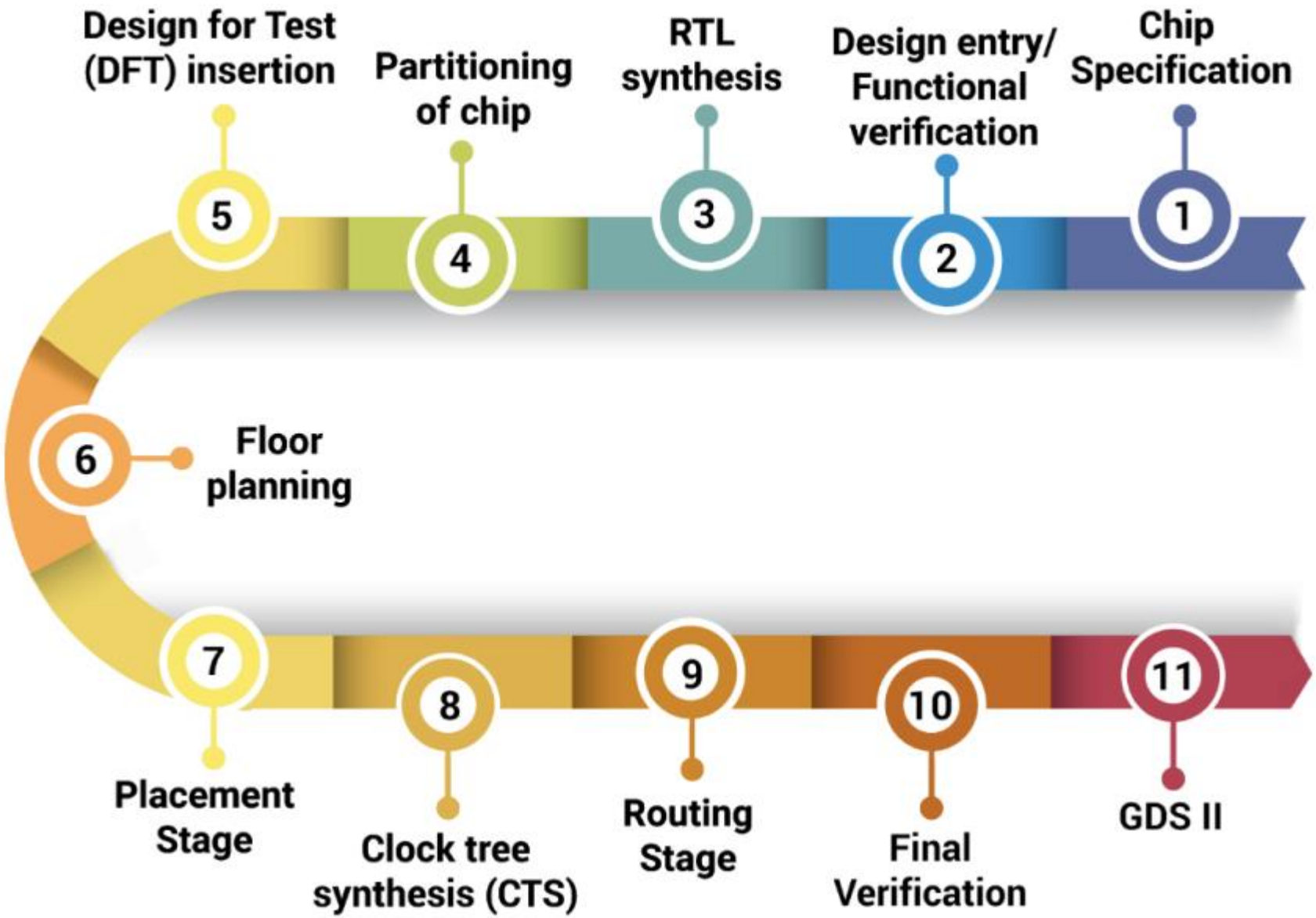
Verify the physical design: After placing and routing the ASIC, you need to verify the physical design using tools that simulate the performance and power consumption of the ASIC. You should also perform design rule checks (DRC) and layout-versus-schematic (LVS) checks to ensure that the design meets the manufacturing requirements.

Generate the GDSII file: Finally, you need to generate the GDSII file, which is the industry-standard format for the physical layout of the ASIC. The GDSII file can be used to fabricate the ASIC in a semiconductor foundry.

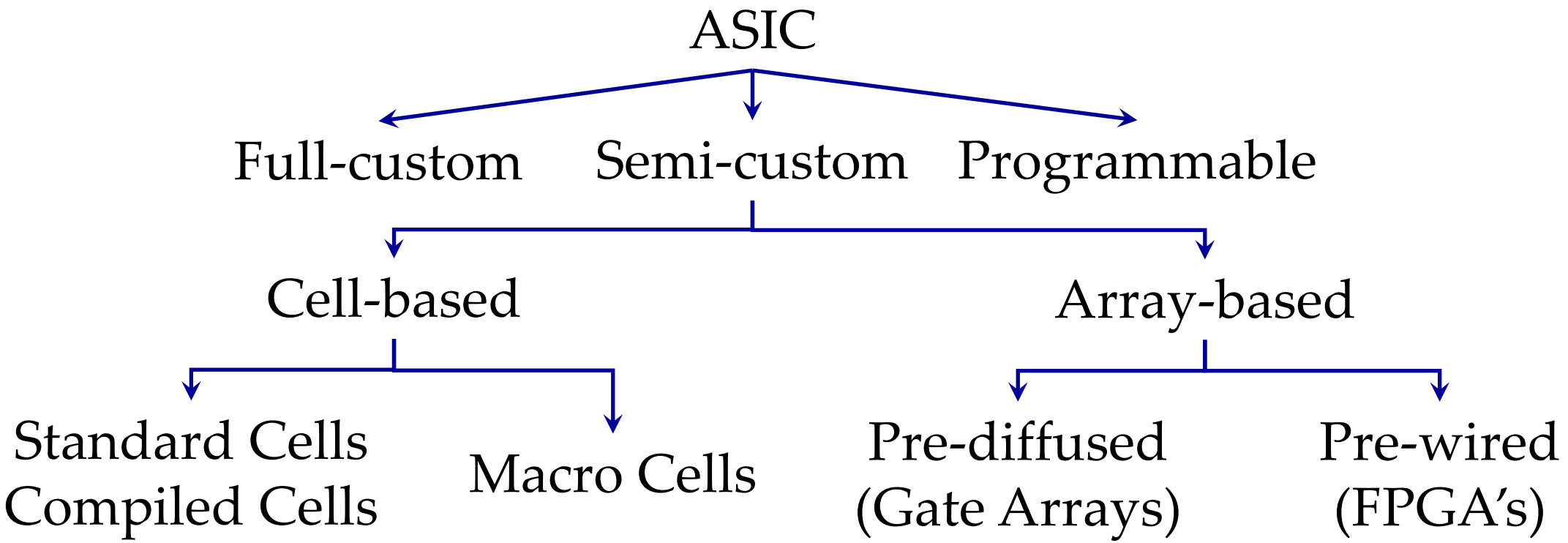
Tape-out: This is the final step of the ASIC design process, which involves submitting the GDSII file to a semiconductor foundry for fabrication. The foundry will manufacture the ASIC according to the GDSII file and the specifications provided by the design team.

In conclusion, ASIC design is a complex process that involves several stages, from concept to production.

ASIC Design Cycle



Digital Circuit Implementation Approaches



Full-Custom Circuit Design

- When performance or design density is of primary importance, handcrafting the circuit topology and physical design seems to be the only option.
- The labor-intensive nature of custom design translates into a high cost and a long time to market.
- However, it can only be justifies economically under the following conditions:
  - The custom block can be reused many times (for example, as a library cell).
  - The cost can be amortized over a large volume. Microprocessors and semiconductor memories are examples of applications in this class.
  - Cost is not the prime design criterion, as it is in supercomputers or hyper supercomputers.
- However, with continuous progress in the design-automation areas, the share of custom design reduces from year to year.
- In most advanced high-performance microprocessors, virtually all portions are designed automatically using semi-custom design approaches.
- Only the most performance critical modules such as the PLL and the clock buffers are designed manually. In fact, library cell design is the only area where custom design still thrives today.
- The amount of design automation in the custom-design process is minimal, yet some design tools have proven indispensable, like, wide range of verification, simulation, extraction and modeling tools, layout editors, design-rule and electrical-rule checkers are at the core of every custom-design environment.

Full-Custom Circuit Design

- labor intensive
- high time-to-market
- cost amortized over a large volume
- reuse as a library cell
- was popular in early designs
- layout editor, DRC, circuit extraction

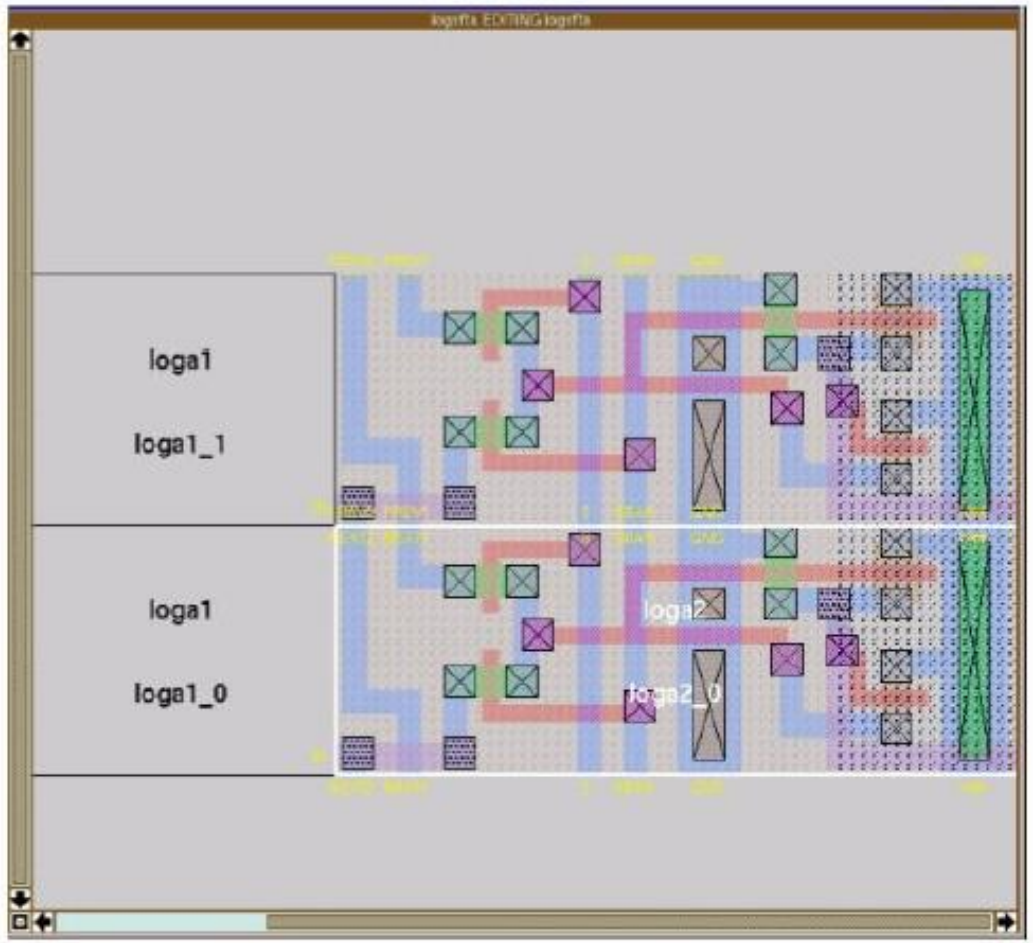
Full-Custom Circuit Design

Symbolic layout

- transistor symbols
- relative positioning
- compaction
- stick diagram description
- design rules automatically satisfied
- automatic pitch matching

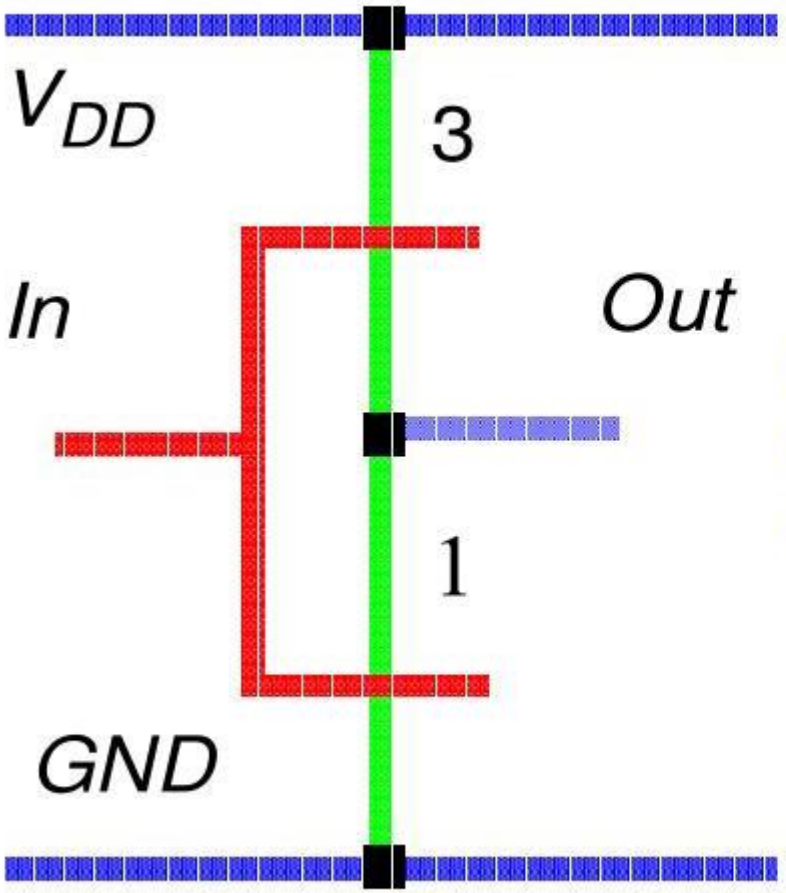
Full-Custom Circuit Design

Layout Editor



Full-Custom Circuit Design

Symbolic Layout



Stick diagram of inverter

- Dimensionless layout entities
- Only topology is important
- Final layout generated by "compaction" program

Full-Custom Circuit Design

Design rule checking

- on-line DRC
  - rules checked and errors flagged during layout
- batch DRC
  - post design verification

Full-Custom Circuit Design

Circuit extraction

Circuit schematic derived from layout

transistors are build with proper geometry

parasitic capacitances and resistances evaluated

extraction of inductance requires 3D analysis

Semi-Custom Circuit Design

Semi-Custom Circuit Design

- Since the full-custom-design approach proves to be prohibitively expensive, a wide variety of design approaches have been introduced over the -years to shorten and automate the design process.
- This automation comes at the price of reduced integration density and/or performance.
- The following rule tends to hold: **the shorter the design time**, **the larger is the penalty incurred**.

Semi-Custom Circuit Design

- Since the full-custom-design approach proves to be prohibitively expensive, a wide variety of design approaches have been introduced over the -years to shorten and automate the design process.
- This automation comes at the price of reduced integration density and/or performance.
- The following rule tends to hold: **the shorter the design time**, **the larger is the penalty incurred**.

Cell-based Design

Semi-Custom Circuit Design

- Since the full-custom-design approach proves to be prohibitively expensive, a wide variety of design approaches have been introduced over the -years to shorten and automate the design process.
- This automation comes at the price of reduced integration density and/or performance.
- The following rule tends to hold: **the shorter the design time**, **the larger is the penalty incurred**.

Cell-based Design → The idea behind cell-based design is to reduce the implementation effort by *reusing* a limited library of cells.

The advantage of this approach is that the cells only need to be designed and verified once for a given technology, and they can be reused many times, thus amortizing the design cost.

The disadvantage is that the constrained nature of the library reduces the possibility of fine-tuning the design.

Semi-Custom Circuit Design

- Since the full-custom-design approach proves to be prohibitively expensive, a wide variety of design approaches have been introduced over the -years to shorten and automate the design process.
- This automation comes at the price of reduced integration density and/or performance.
- The following rule tends to hold: **the shorter the design time, the larger is the penalty incurred**.

Cell-based Design → The idea behind cell-based design is to reduce the implementation effort by *reusing* a limited library of cells.

The advantage of this approach is that the cells only need to be designed and verified once for a given technology, and they can be reused many times, thus amortizing the design cost.

The disadvantage is that the constrained nature of the library reduces the possibility of fine-tuning the design.

Cell-based approaches can be partitioned into a number of classes depending on the granularity of the library elements.

Semi-Custom Circuit Design

- Since the full-custom-design approach proves to be prohibitively expensive, a wide variety of design approaches have been introduced over the -years to shorten and automate the design process.
- This automation comes at the price of reduced integration density and/or performance.
- The following rule tends to hold: **the shorter the design time, the larger is the penalty incurred**.

Cell-based Design → The idea behind cell-based design is to reduce the implementation effort by *reusing* a limited library of cells.

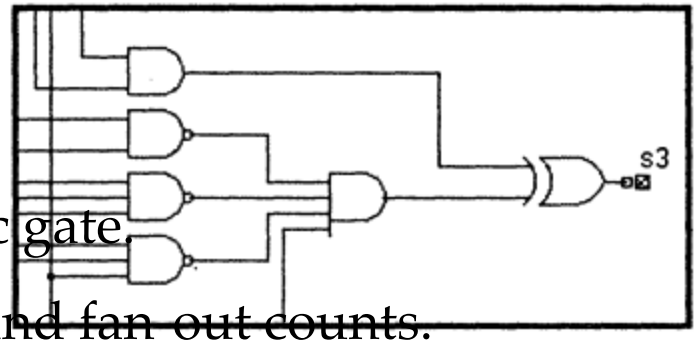
The advantage of this approach is that the cells only need to be designed and verified once for a given technology, and they can be reused many times, thus amortizing the design cost.

The disadvantage is that the constrained nature of the library reduces the possibility of fine-tuning the design.

Cell-based approaches can be partitioned into a number of classes depending on the granularity of the library elements.

- ❑ Standard Cells
- ❑ Compiled Cells
- ❑ Macro or Mega Cells
  - Hard Macro – *targeted for specific IC manufacturing technology*
  - Soft Macro – *used in SoC implementations*

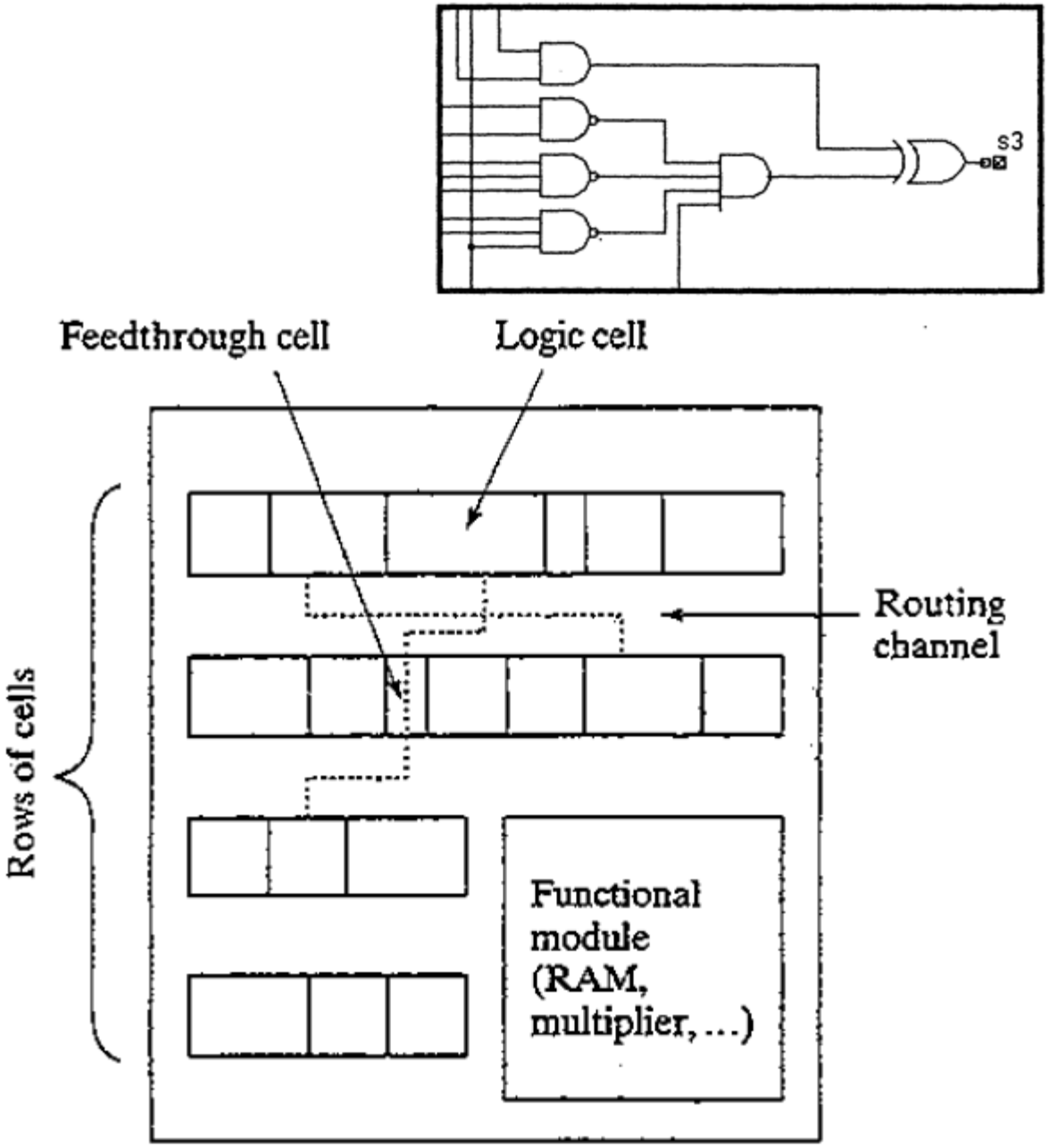
Standard Cell-based Design



- The standard-cell approach standardizes the design entry level at the logic gate.
- A library containing a wide selection of logic gates over a range of fan-in and fan-out counts.
- Besides the basic logic functions, such as inverter, AND/NAND, OR/NOR, XOR/XNOR, and flip-flops, a typical library also contains more complex functions, such as AND-OR-INVERT, MUX, full adder, comparator, counter, decoders, and encoders.
- A design is captured as a schematic containing cells available in the library, or is generated automatically from a higher level description language. The layout is then automatically generated.
- In the standard-cell philosophy, cells are placed in rows that are separated by routing channels.
- To be effective, this requires that all cells in the library have identical heights.
- The width of the cell can vary to accommodate for the variation in complexity between the cells.
- The standard-cell technique can be intermixed with other layout approaches to allow for the introduction of modules such as memories and multipliers that do not adapt easily or efficiently to the logic-cell paradigm.

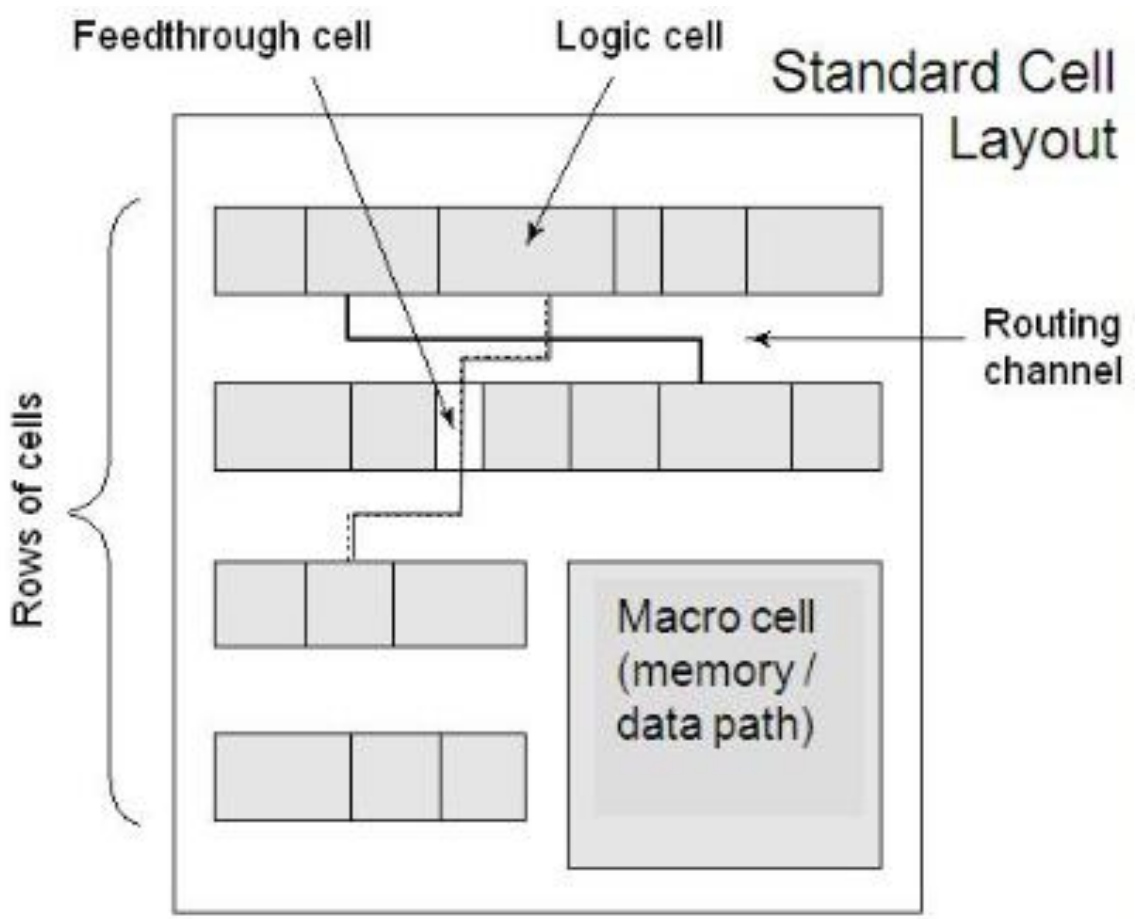
Standard Cell-based Design

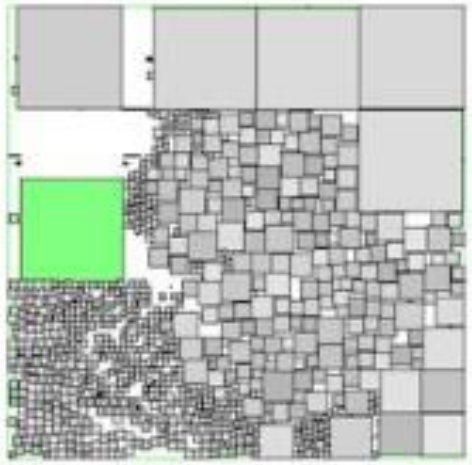
- The minimization of the interconnect overhead is the most important goal of the standard-cell placement and routing tools.
- One approach to minimize the wire length is to introduce feed-through cells that make it possible to connect between cells in different rows without having to route around a complete row.
- An important reduction in wiring overhead is obtained by adding more interconnect layers.
- Standard Cell-based design has detrimental effect on area and power consumption.



Standard Cell-based Design

- The minimization of the interconnect overhead is the most important goal of the standard-cell placement and routing tools.
- One approach to minimize the wire length is to introduce feed-through cells that make it possible to connect between cells in different rows without having to route around a complete row.
- An important reduction in wiring overhead is obtained by adding more interconnect layers.
- Standard Cell-based design has detrimental effect on area and power consumption.

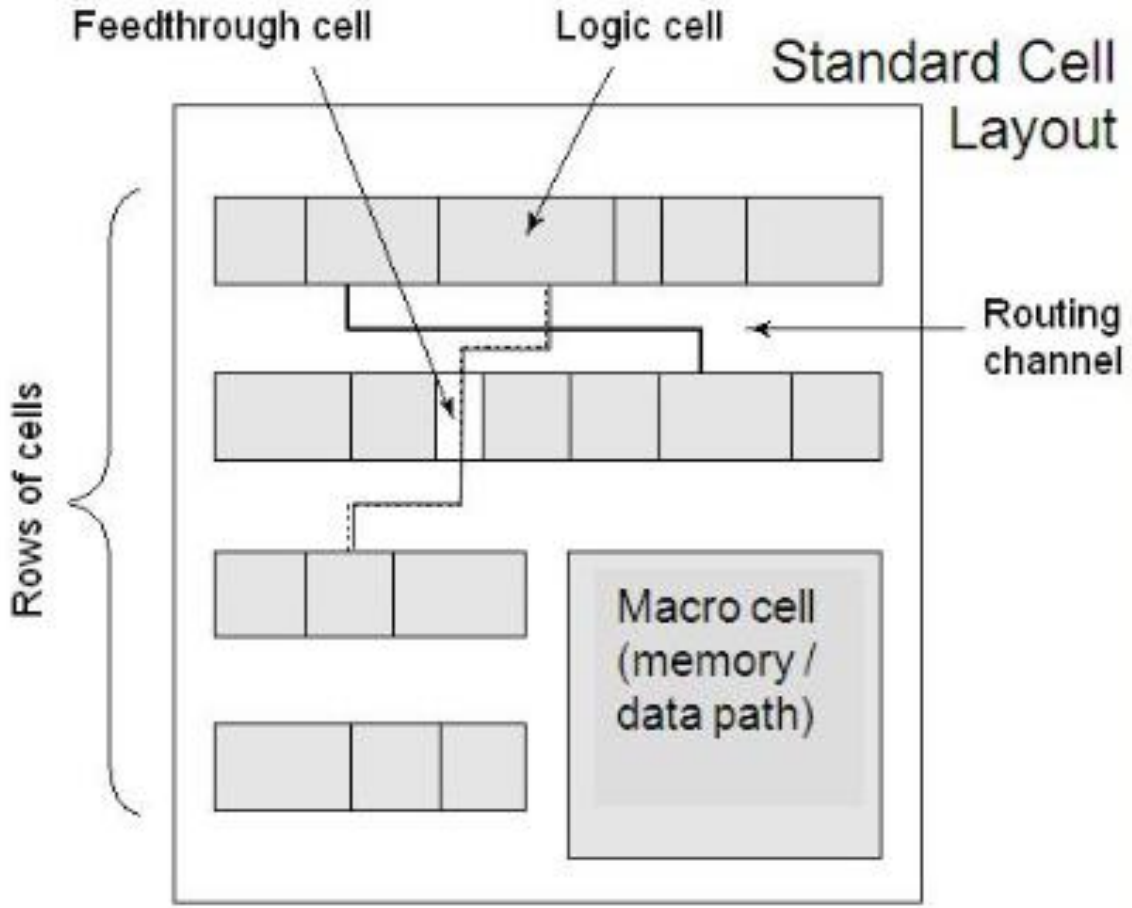


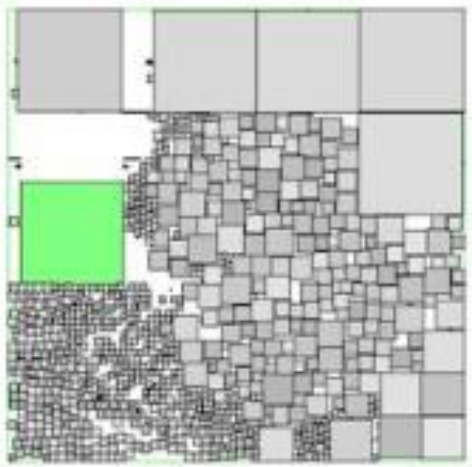


Macro Cell Layout

Standard Cell-based Design

- library contains basic logic cells
  - inverter, AND/NAND, OR/NOR, XOR/NXOR, flip-flop
  - AOI, MUX, adder, compactor, counter, decoder, encoder,
- fan-in and fan-out specified
- schematic uses cells from library
- layout automatically generated

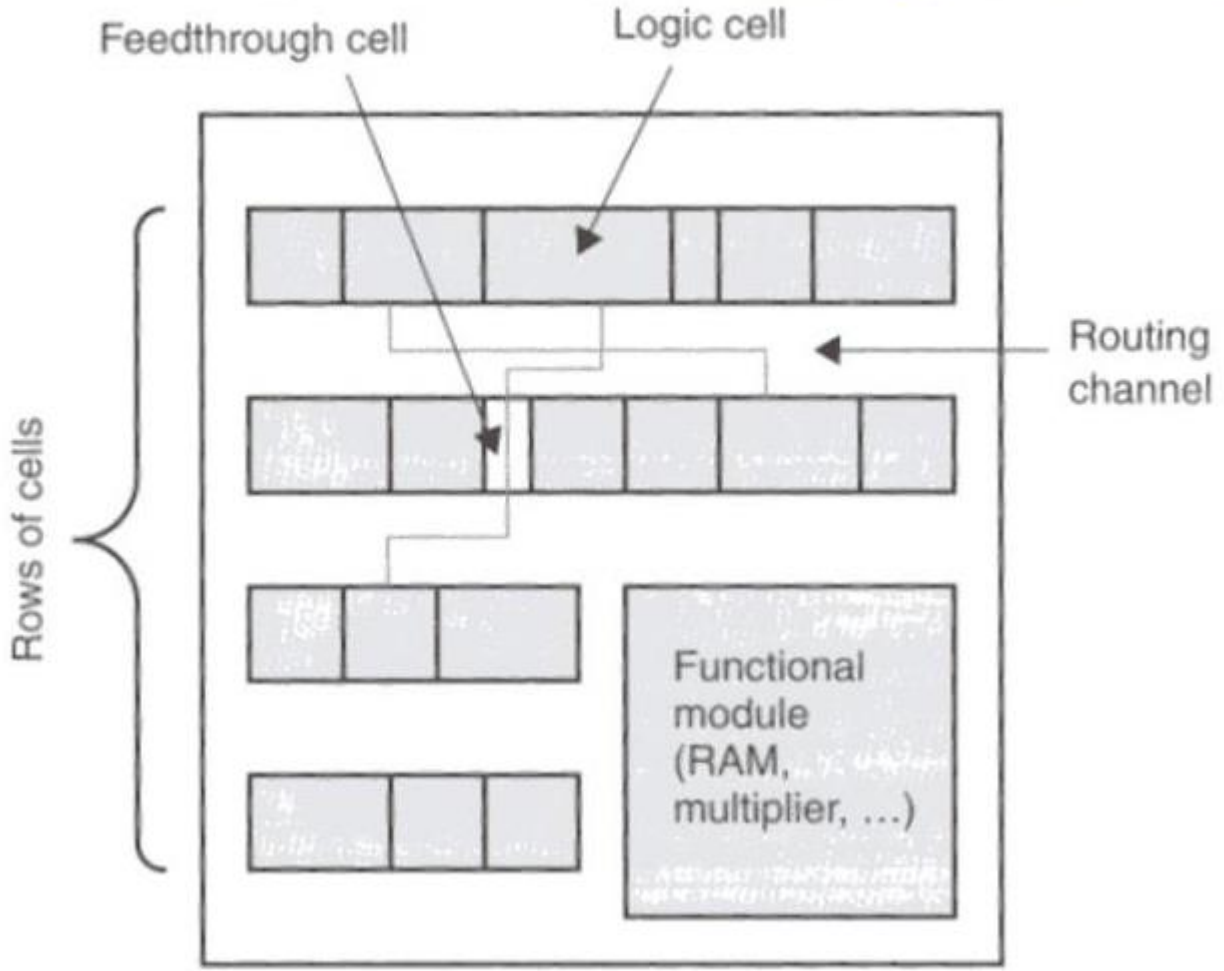




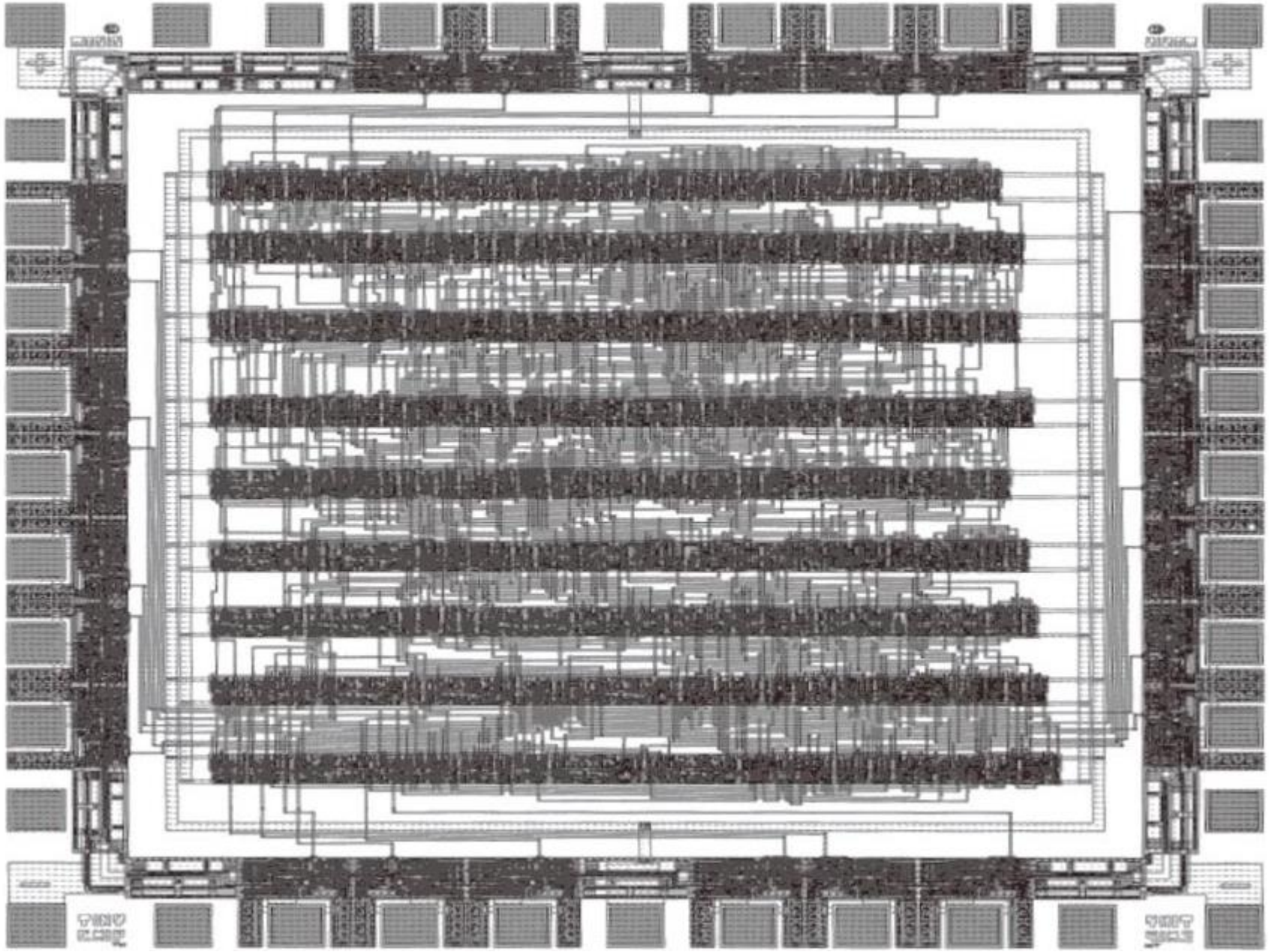
Macro Cell Layout

Standard Cell-based Design

- cells have equal heights
- cell rows separated by routing channels

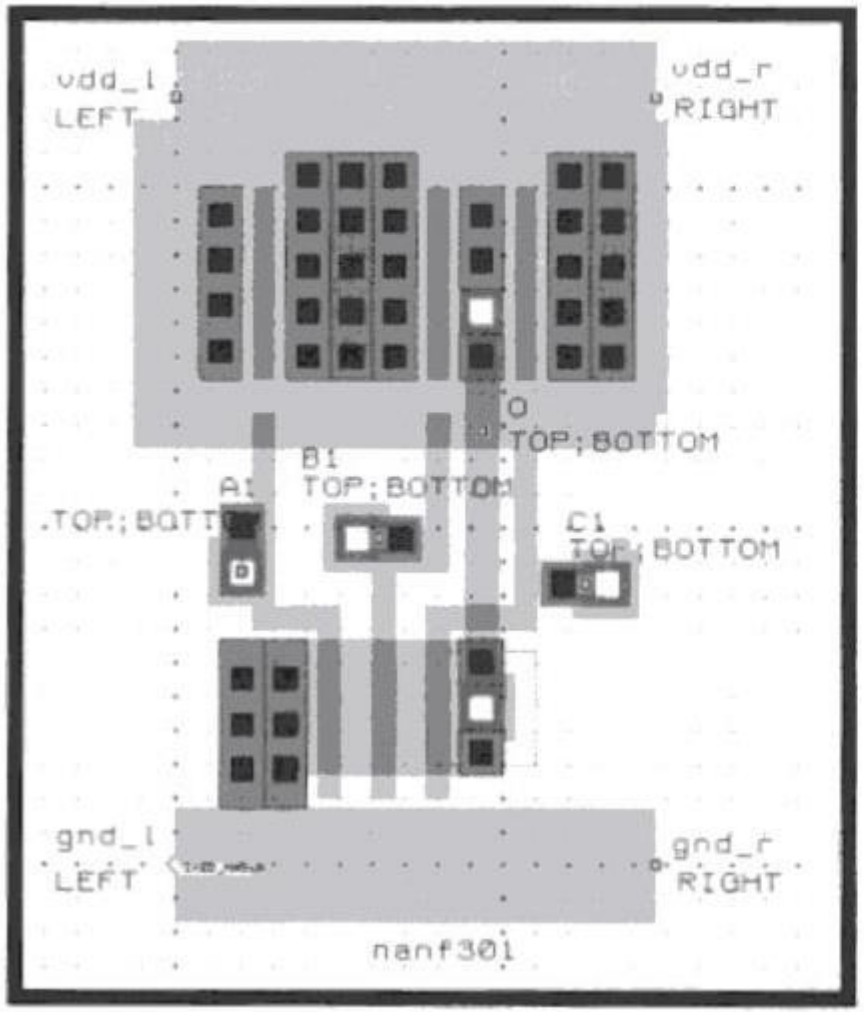


Standard Cell-based Design



Standard Cell-based Design

layout and description



Fanout 4x	0.5 µm	1.0 µm	2.0 µm
A1_tphl	0.595	0.711	0.919
A1_tplh	0.692	0.933	1.360
B1_tphl	0.591	0.739	1.006
B1_tplh	0.620	0.825	1.1.81
C1_tphl	0.574	0.740	1.029
C1_tplh	0.554	0.728	1.026

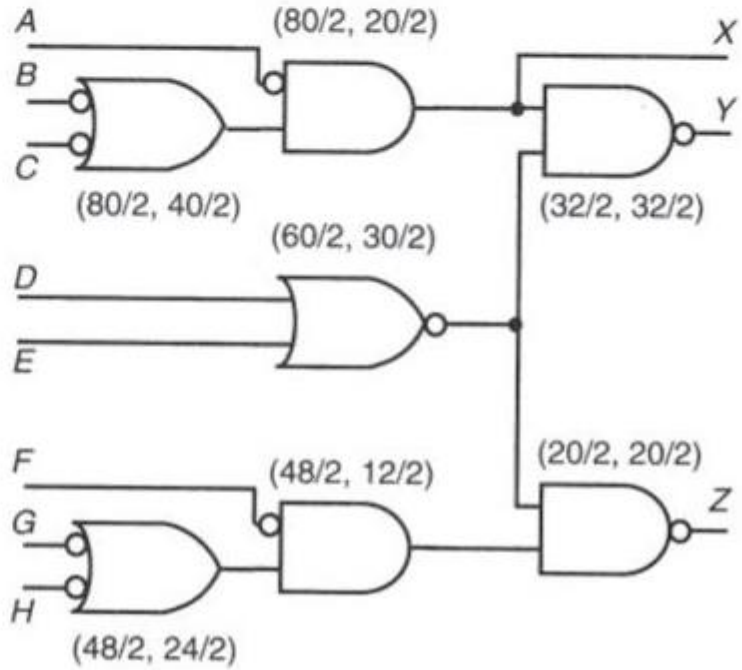
Standard Cell-based Design

- large design cost amortized over a large number of designs
- large number of different cells with different fan-ins
- large fan-out for cells to be used in different designs
- synthesis tools made standard cell design popular
- standard cell design outperform PLA in area and speed
- standard cell benefit from multi level logic synthesis

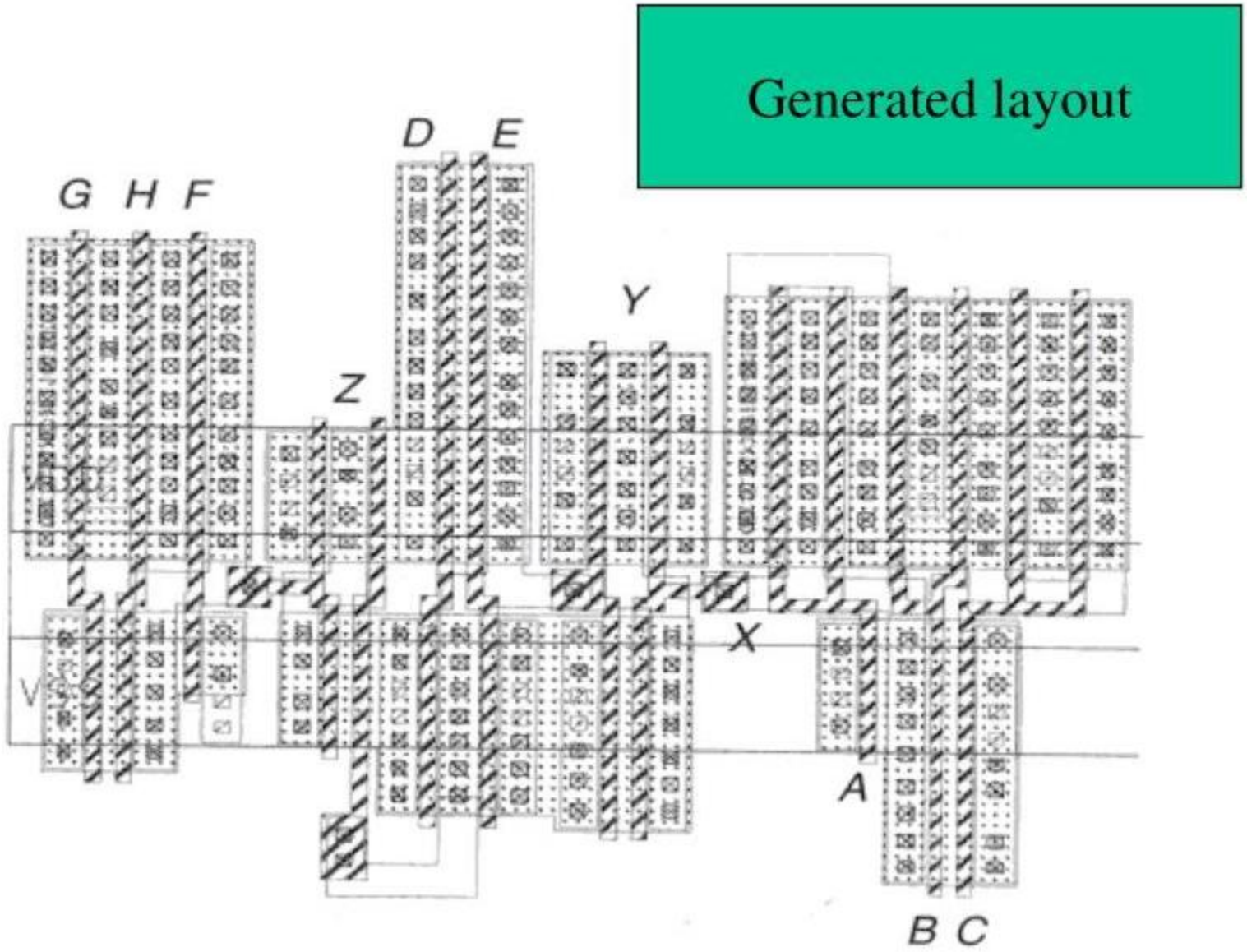
Compiled Cell

- cell layout generated on the fly
- transistor or gate level netlist used with transistor size specified
- layout densities approach that of human designers

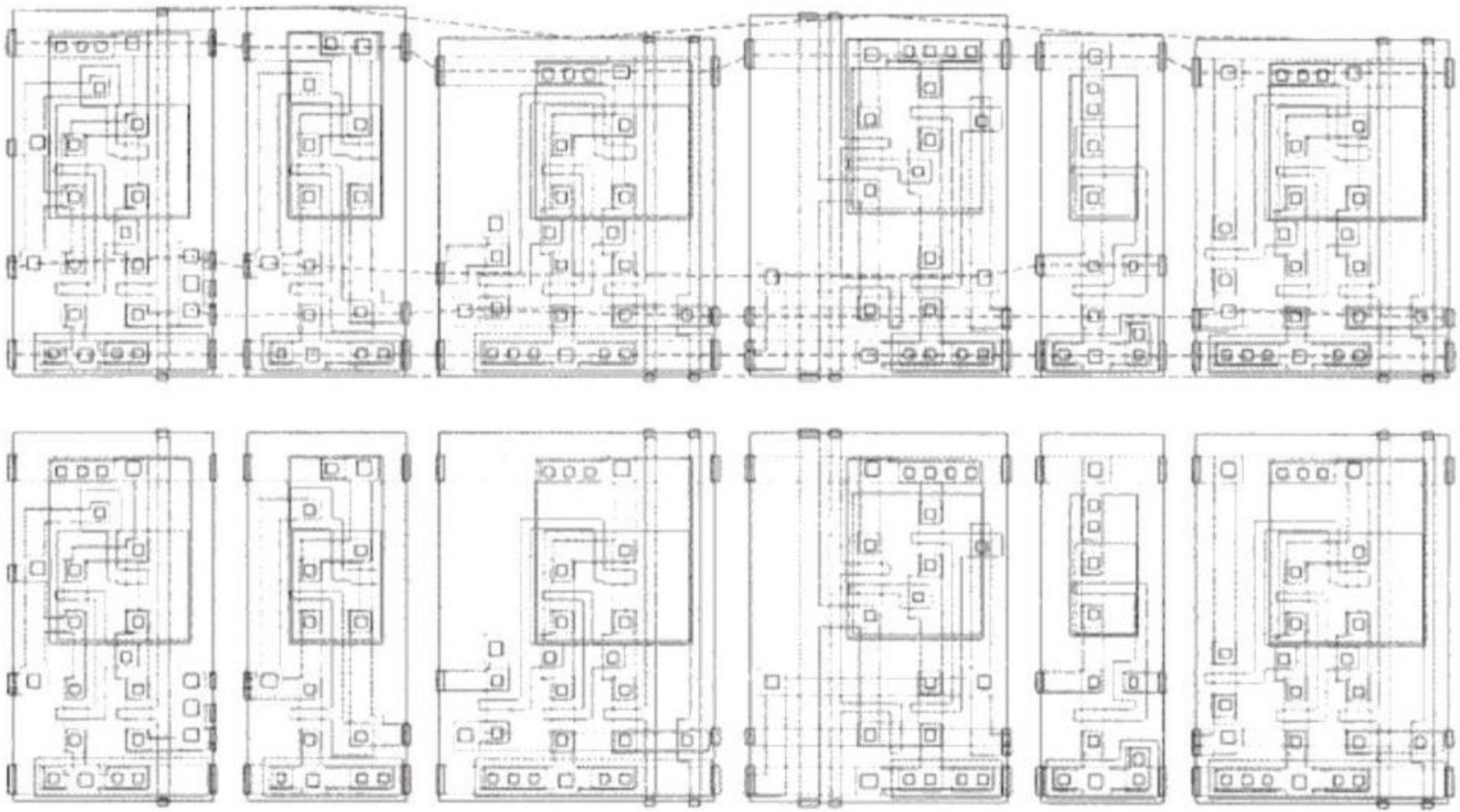
Circuit schematics with transistor sizing



Compiled Cell



Automatic pitch matching



Macro or Mega Cell

Standardizing at the logic-gate level is attractive for random logic functions, but it turns out to be inefficient for more complex structures such as multipliers, data paths, memories, and embedded microprocessors and DSPs.

By capturing the specific nature of these blocks, implementations can be obtained that outperform the results of the standard ASIC design process by a wide margin. Cells that contain a complexity that surpasses what is found in a typical standard-cell library are called *macrocells* (or, sometimes, *megacells*).

Macro or Mega Cell

Standardizing at the logic-gate level is attractive for random logic functions, but it turns out to be inefficient for more complex structures such as multipliers, data paths, memories, and embedded microprocessors and DSPs.

By capturing the specific nature of these blocks, implementations can be obtained that outperform the results of the standard ASIC design process by a wide margin. Cells that contain a complexity that surpasses what is found in a typical standard-cell library are called *macrocells* (or, sometimes, *megacells*).

The ***Hard Macro*** represents a module with a given functionality and a predetermined physical design. The relative location of the transistors and the wiring within the module is fixed. In essence, a hard macro represents a custom design of the requested function. In some cases, the macro is parameterized, which means that versions with slightly different properties are available or can be generated. Multipliers and memories are examples: A hard multiplier macro may not only generate a 32 × 16 multiplier, but also an 8 × 8 one.

Macro or Mega Cell

Standardizing at the logic-gate level is attractive for random logic functions, but it turns out to be inefficient for more complex structures such as multipliers, data paths, memories, and embedded microprocessors and DSPs.

By capturing the specific nature of these blocks, implementations can be obtained that outperform the results of the standard ASIC design process by a wide margin. Cells that contain a complexity that surpasses what is found in a typical standard-cell library are called *macrocells* (or, sometimes, *megacells*).

The ***Hard Macro*** represents a module with a given functionality and a predetermined physical design. The relative location of the transistors and the wiring within the module is fixed. In essence, a hard macro represents a custom design of the requested function. In some cases, the macro is parameterized, which means that versions with slightly different properties are available or can be generated. Multipliers and memories are examples: A hard multiplier macro may not only generate a 32 × 16 multiplier, but also an 8 × 8 one.

Advantages: dense layout, and optimized and predictable performance and power dissipation. By encapsulating the function into a macro-module, it can be reused over and over in different designs. This reuse helps to offset the initial design cost.

Macro or Mega Cell

Standardizing at the logic-gate level is attractive for random logic functions, but it turns out to be inefficient for more complex structures such as multipliers, data paths, memories, and embedded microprocessors and DSPs.

By capturing the specific nature of these blocks, implementations can be obtained that outperform the results of the standard ASIC design process by a wide margin. Cells that contain a complexity that surpasses what is found in a typical standard-cell library are called *macrocells* (or, sometimes, *megacells*).

The ***Hard Macro*** represents a module with a given functionality and a predetermined physical design. The relative location of the transistors and the wiring within the module is fixed. In essence, a hard macro represents a custom design of the requested function. In some cases, the macro is parameterized, which means that versions with slightly different properties are available or can be generated. Multipliers and memories are examples: A hard multiplier macro may not only generate a 32 × 16 multiplier, but also an 8 × 8 one.

Advantages: dense layout, and optimized and predictable performance and power dissipation. By encapsulating the function into a macro-module, it can be reused over and over in different designs. This reuse helps to offset the initial design cost.

Disadvantages: it is hard to port the design to other technologies or to other manufacturers. For every new technology, a major redesign of the block is necessary. The hard macros are used less and less, and are employed mainly when the automated generation approach is far inferior or even impossible.

Macro or Mega Cell

Standardizing at the logic-gate level is attractive for random logic functions, but it turns out to be inefficient for more complex structures such as multipliers, data paths, memories, and embedded microprocessors and DSPs.

By capturing the specific nature of these blocks, implementations can be obtained that outperform the results of the standard ASIC design process by a wide margin. Cells that contain a complexity that surpasses what is found in a typical standard-cell library are called *macrocells* (or, sometimes, *megacells*).

The ***Soft Macro*** represents a module with a given functionality, but without a specific physical implementation. The placement and the wiring of a *soft macro* may vary from instance to instance. This means that the timing data can only be determined after the final synthesis and placement and routing steps – in other words, the process is unpredictable.

Macro or Mega Cell

Standardizing at the logic-gate level is attractive for random logic functions, but it turns out to be inefficient for more complex structures such as multipliers, data paths, memories, and embedded microprocessors and DSPs.

By capturing the specific nature of these blocks, implementations can be obtained that outperform the results of the standard ASIC design process by a wide margin. Cells that contain a complexity that surpasses what is found in a typical standard-cell library are called *macrocells* (or, sometimes, *megacells*).

The ***Soft Macro*** represents a module with a given functionality, but without a specific physical implementation. The placement and the wiring of a *soft macro* may vary from instance to instance. This means that the timing data can only be determined after the final synthesis and placement and routing steps – in other words, the process is unpredictable.

Advantages: *soft macros* can be ported over a wide range of technologies and processes. This amortizes the design effort and cost over a wide set of designs. Through intrinsic knowledge of the internal structure of the module, and by imposing precise timing and placement constraints on the physical generation process, *soft macros* most often succeed in offering well-defined timing guarantees.

Macro or Mega Cell

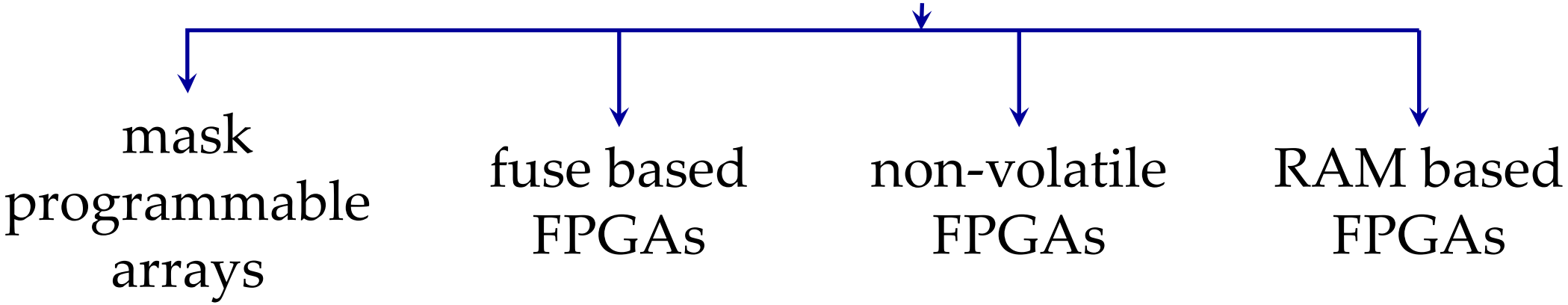
Standardizing at the logic-gate level is attractive for random logic functions, but it turns out to be inefficient for more complex structures such as multipliers, data paths, memories, and embedded microprocessors and DSPs.

By capturing the specific nature of these blocks, implementations can be obtained that outperform the results of the standard ASIC design process by a wide margin. Cells that contain a complexity that surpasses what is found in a typical standard-cell library are called *macrocells* (or, sometimes, *megacells*).

The availability of macro modules has substantially changed the semicustom design landscape in the 21st century. With the complexity of lCs going up exponentially. the idea of building every new IC from scratch becomes an uneconomic and non-plausible proposition. More and more, circuits are being built from reusable building blocks of increasing complexity and functionality. Typically, these modules are acquired from third-party vendors, who make the functions available through royalty or licensing agreements. Macro-models distributed in this style are called *intellectual property* (or IP) modules. This approach is somewhat comparable to the software world, where a large programming project typically makes intensive use of reusable software libraries.

Array-based Design Implementation

Array-based Design Implementation



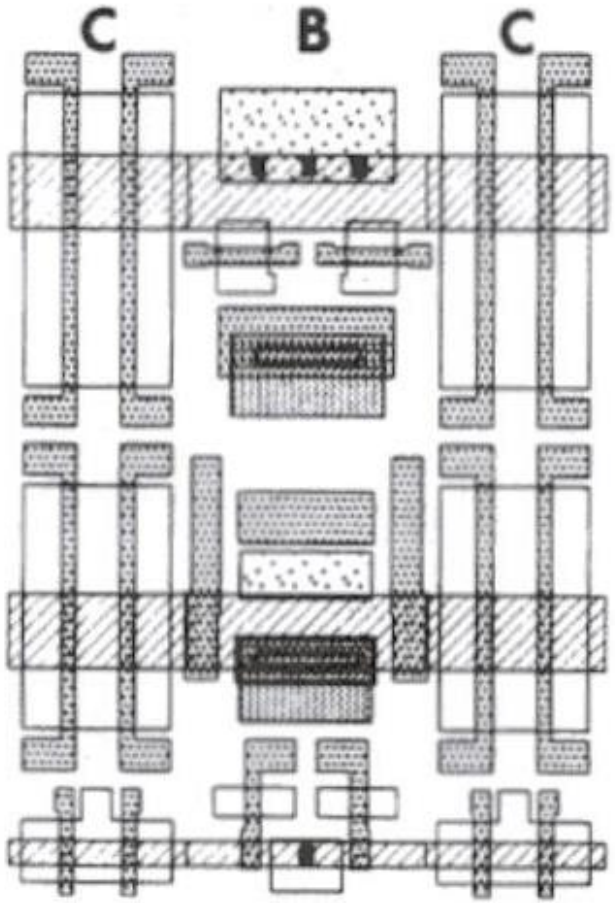
Array-based Design Implementation

Mask programmable arrays

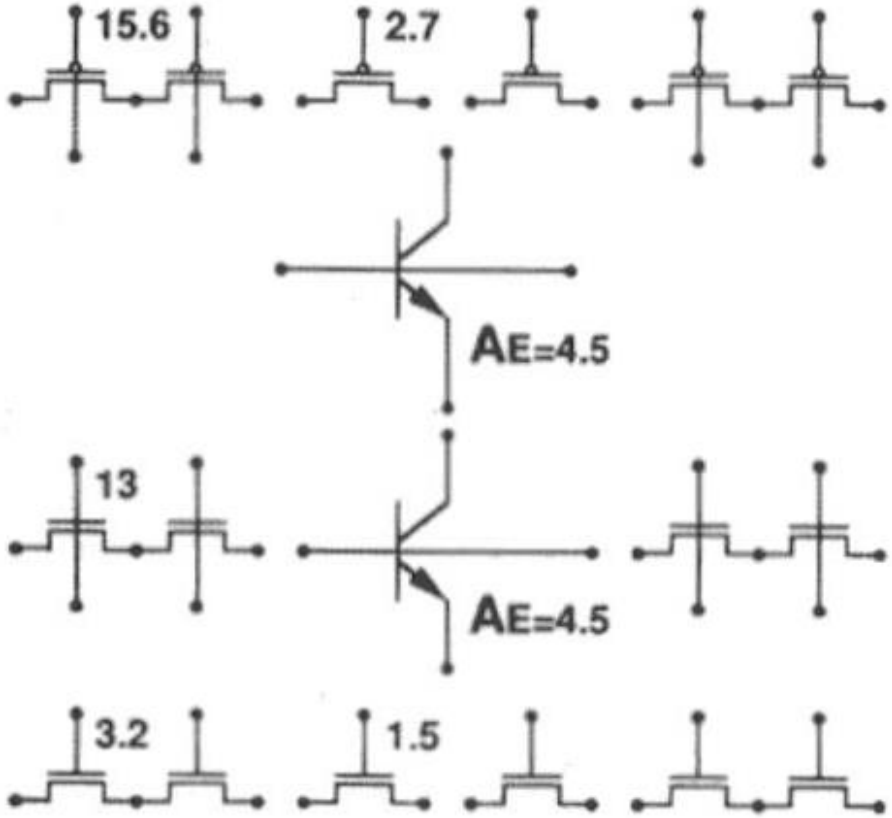
- gate-array
  - similar to standard cell
- sea-of-gate
  - routed over the cells (high density)
  - wires added to make logic gates
- challenge in design is to utilize the maximum cell capacity
- utilization < 75% for random logic design

Array-based Design Implementation

Mask programmable arrays



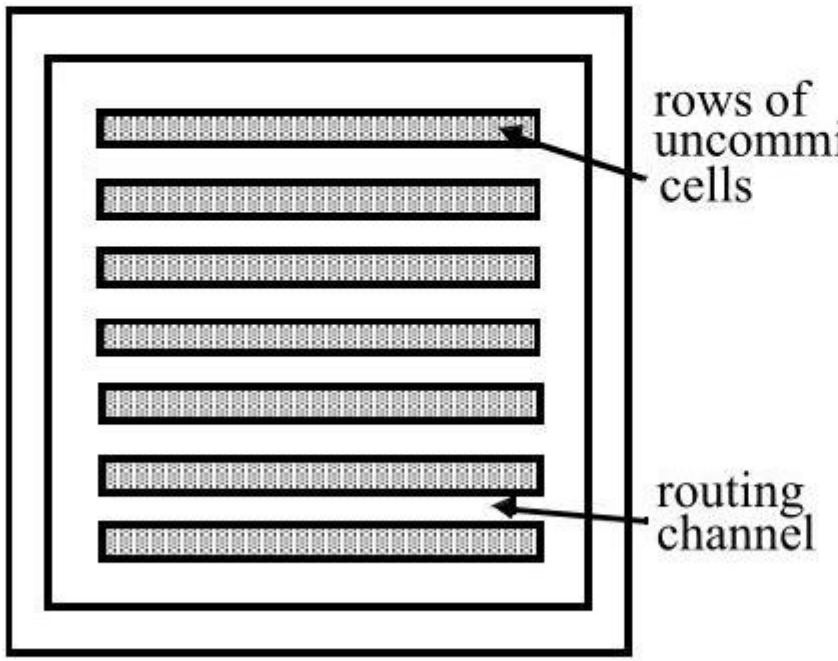
(a) Layout

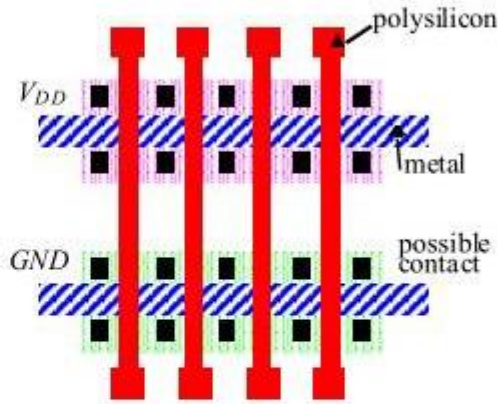


(b) Schematic

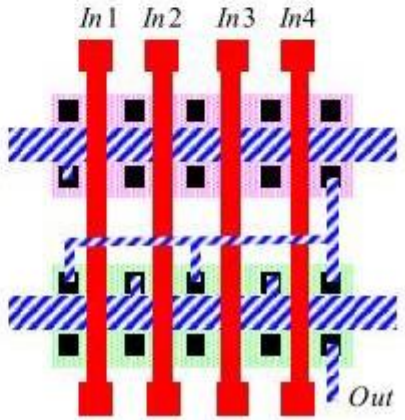
Array-based Design Implementation

Gate Array — Sea-of-gates



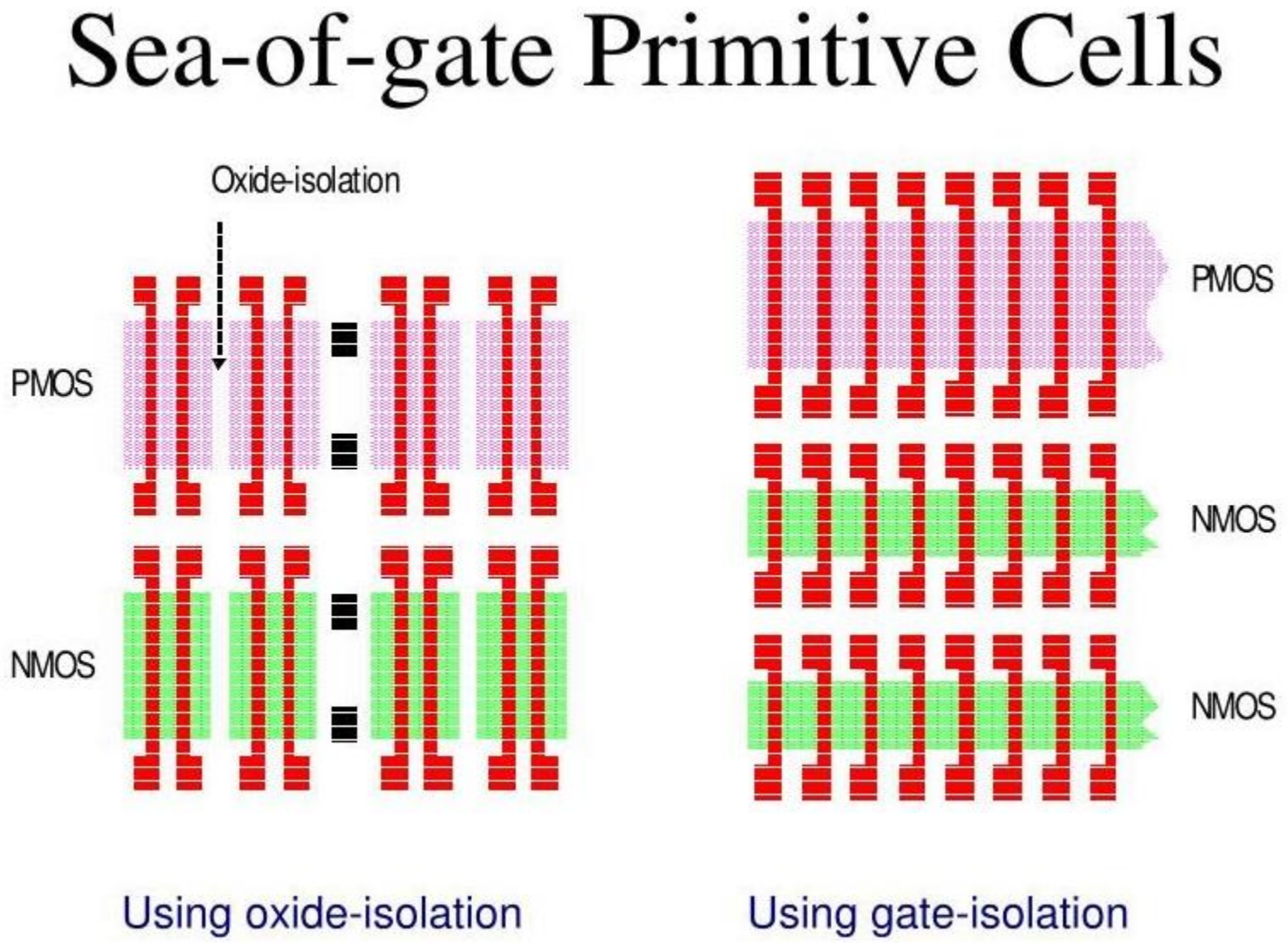


Uncommited Cell



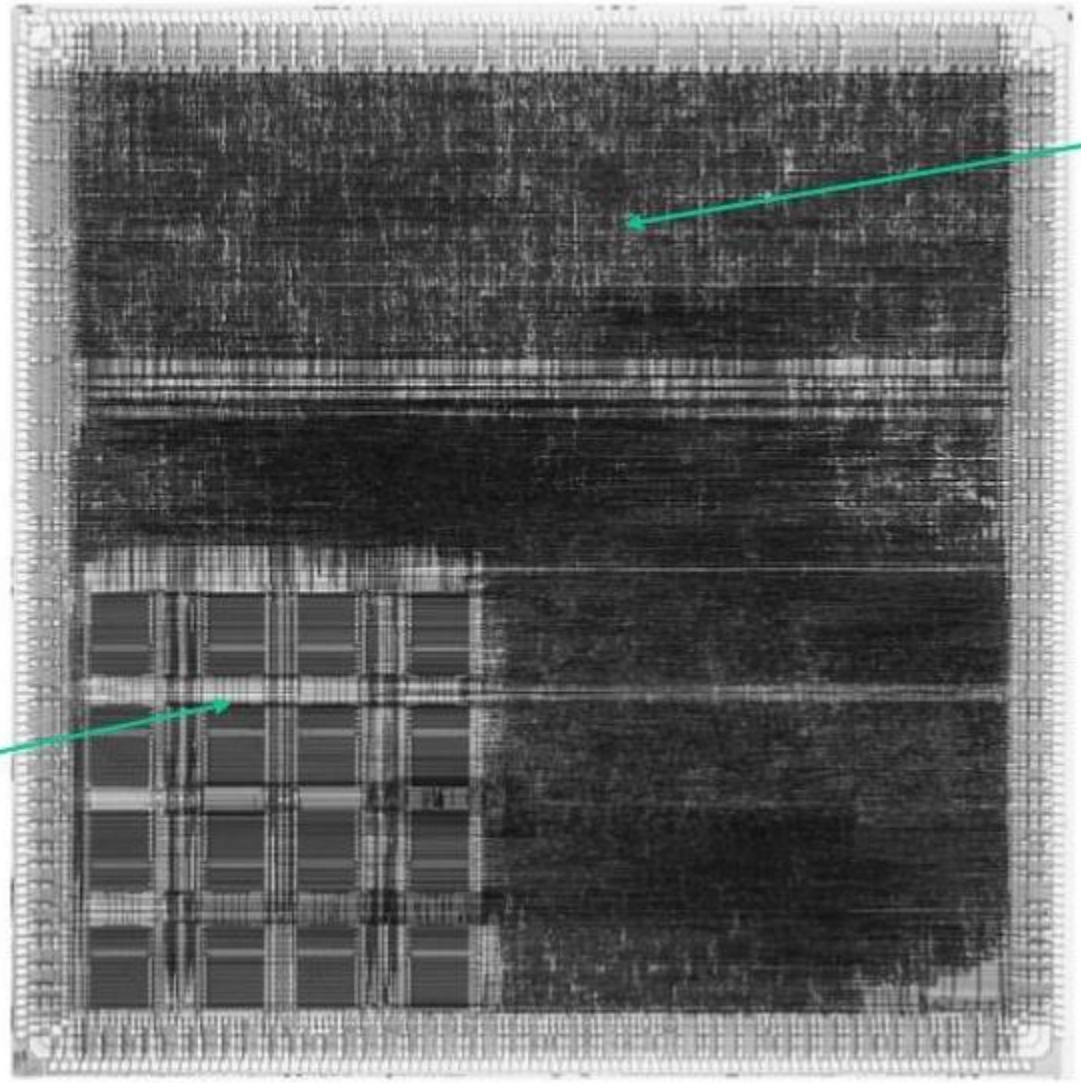
Committed Cell (4-input NOR)

Array-based Design Implementation



Array-based Design Implementation

Sea-of-gates



LSI Logic LEA300K
(0.6 μm CMOS)

Array-based Design Implementation

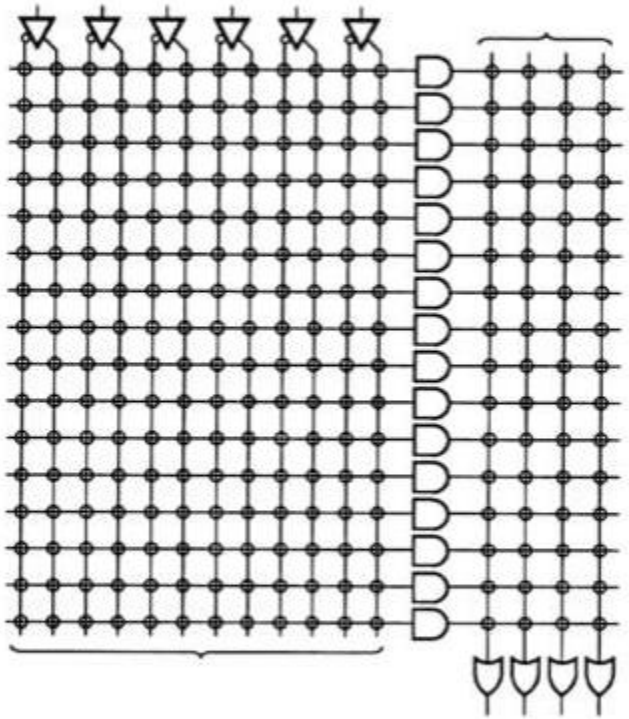
Prewired Arrays

Categories of prewired arrays (or field-programmable devices):

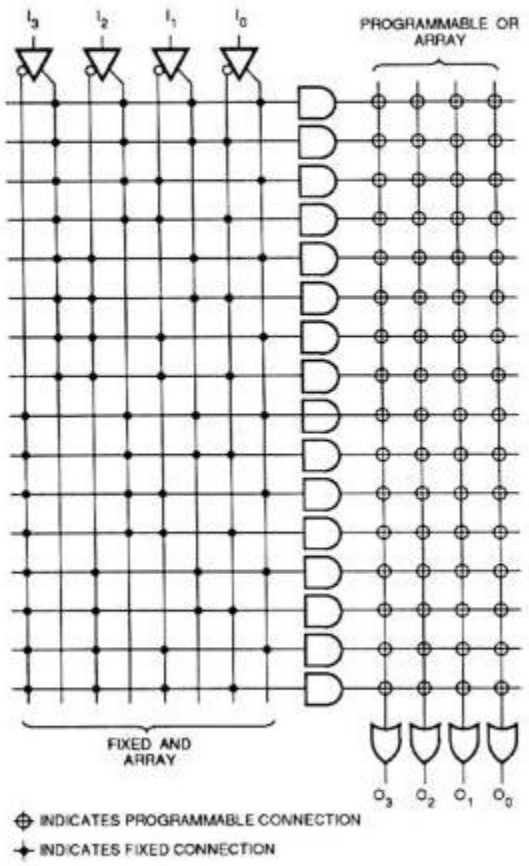
- Fuse-based (program-once)
- Non-volatile EPROM based
- RAM based

Array-based Design Implementation

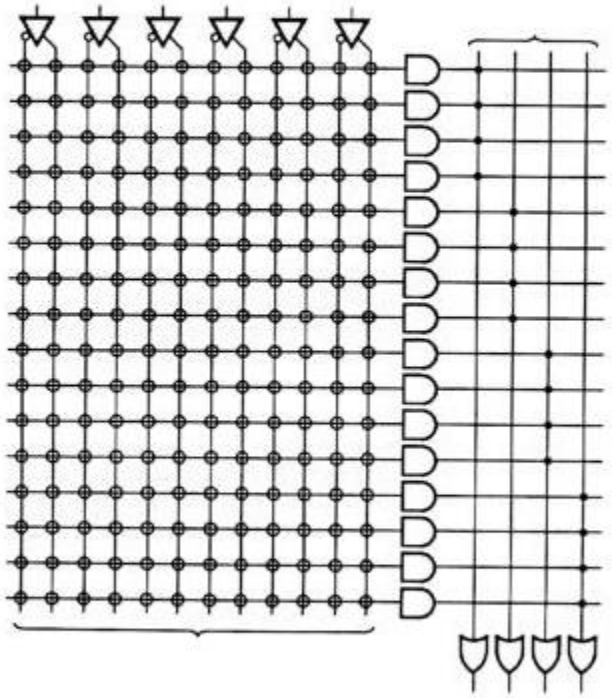
Programmable Logic Devices



PLA



PROM

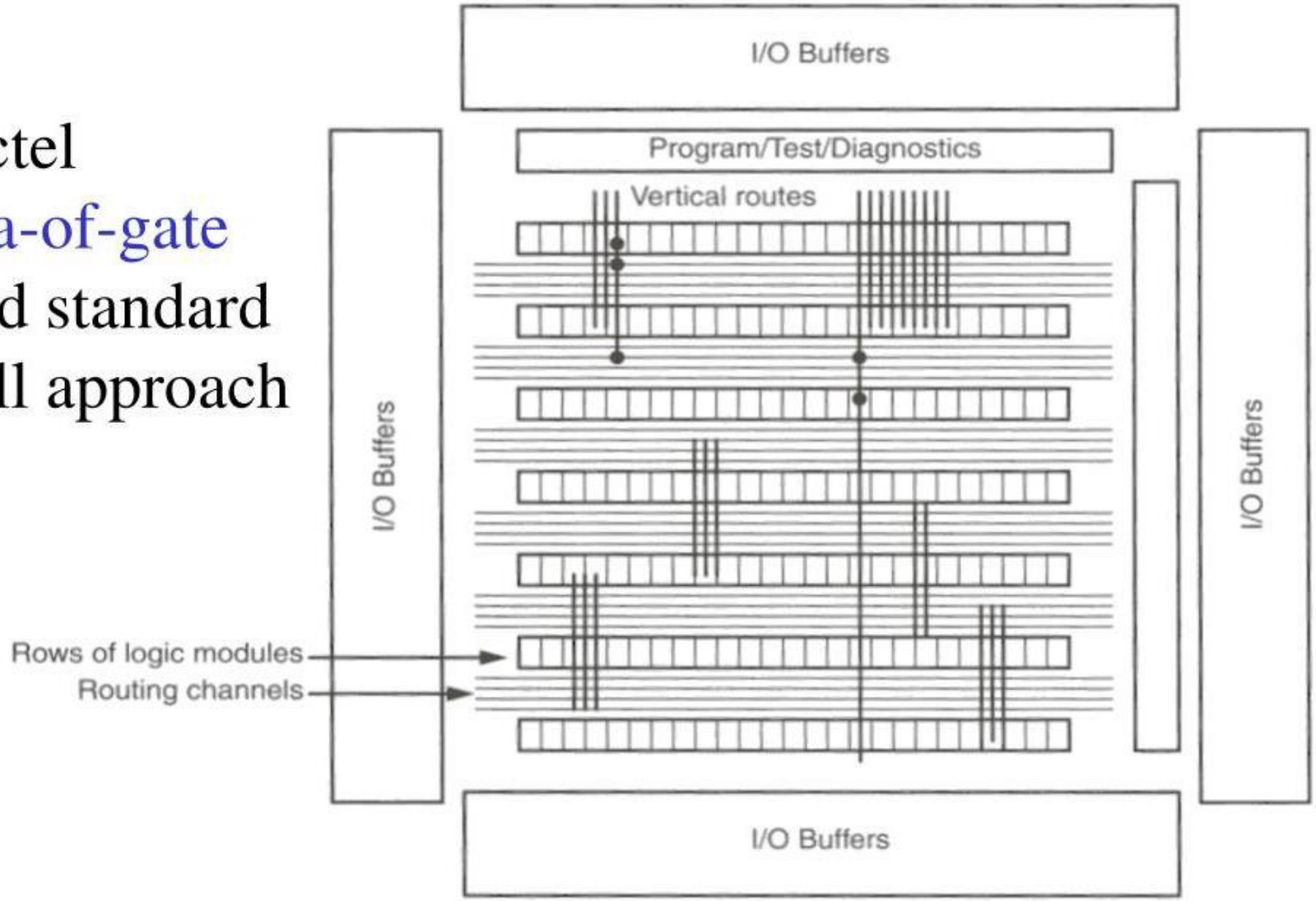


PAL

Array-based Design Implementation

Fuse-based FPGA's

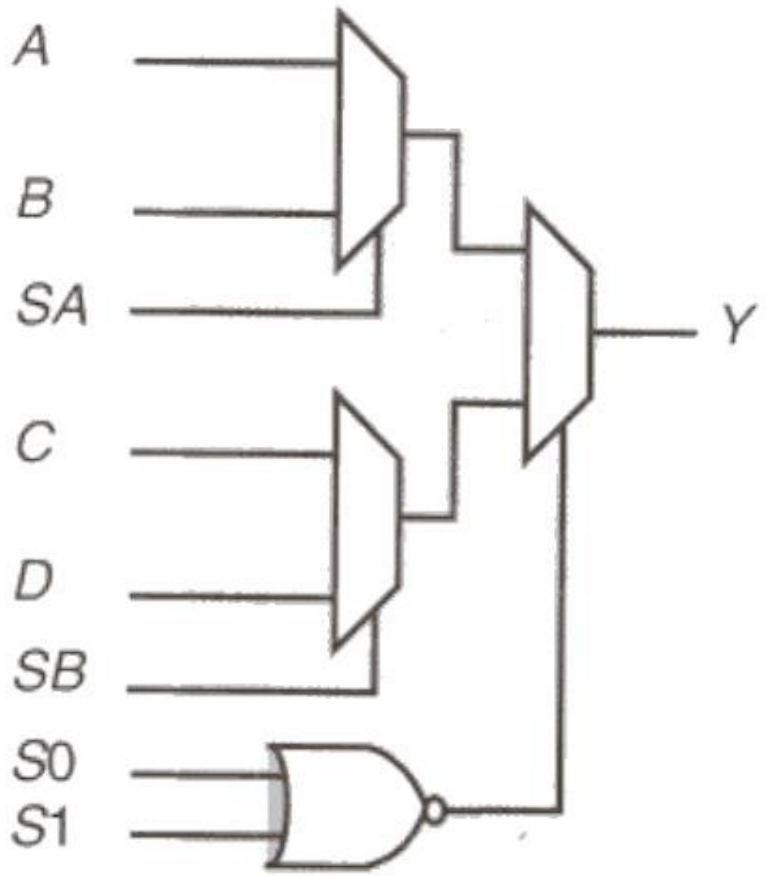
Actel
sea-of-gate
and standard
cell approach



Array-based Design Implementation

Fuse-based FPGA's

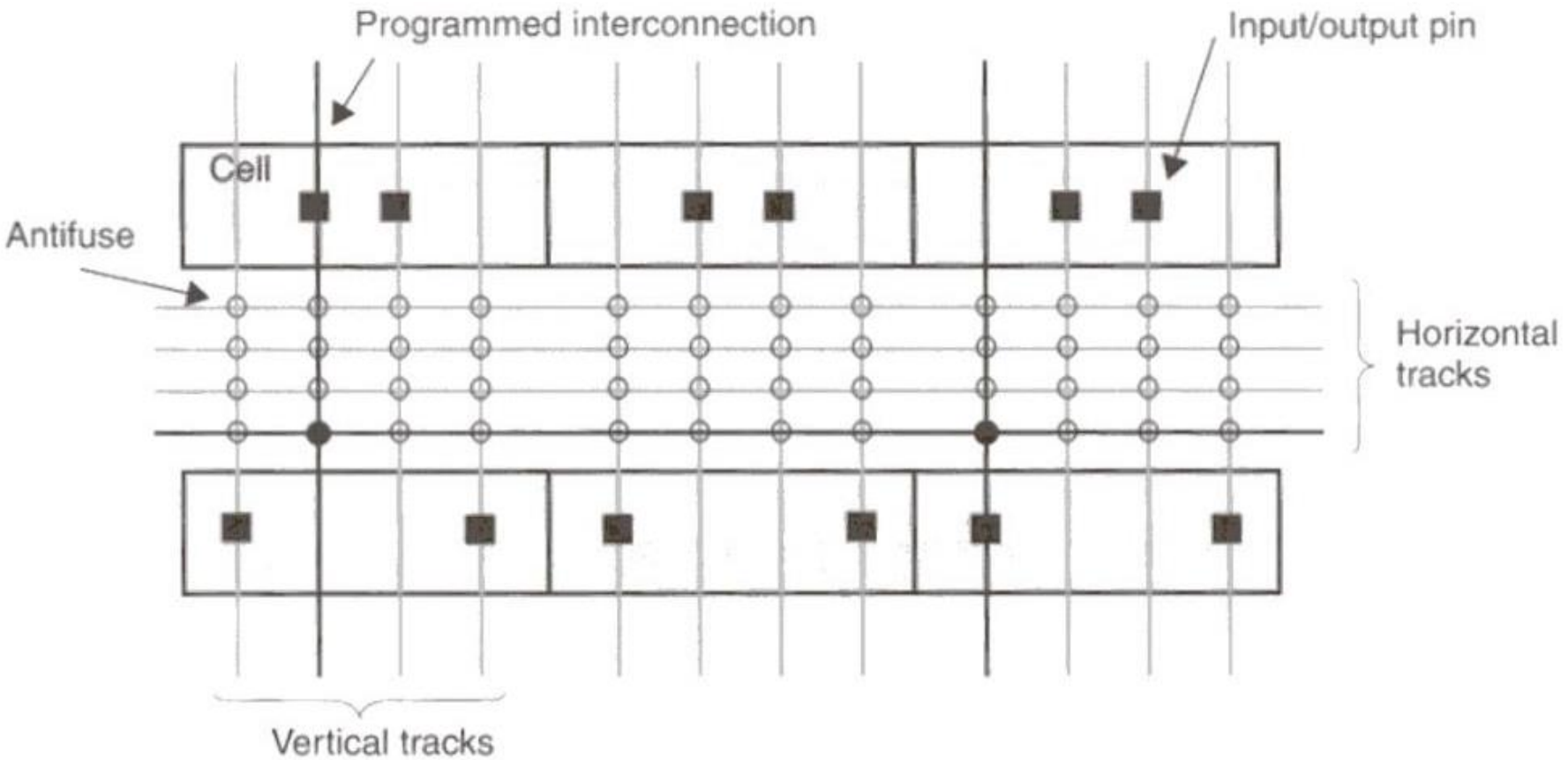
Example :
XOR gate obtained
by setting :
A=1, B=0, C=0, D=1,
SA=SB=In1,
S0=S1=In2



Array-based Design Implementation

Fuse-based FPGA's

Anti-fuse provides short (low resistance) when blown out



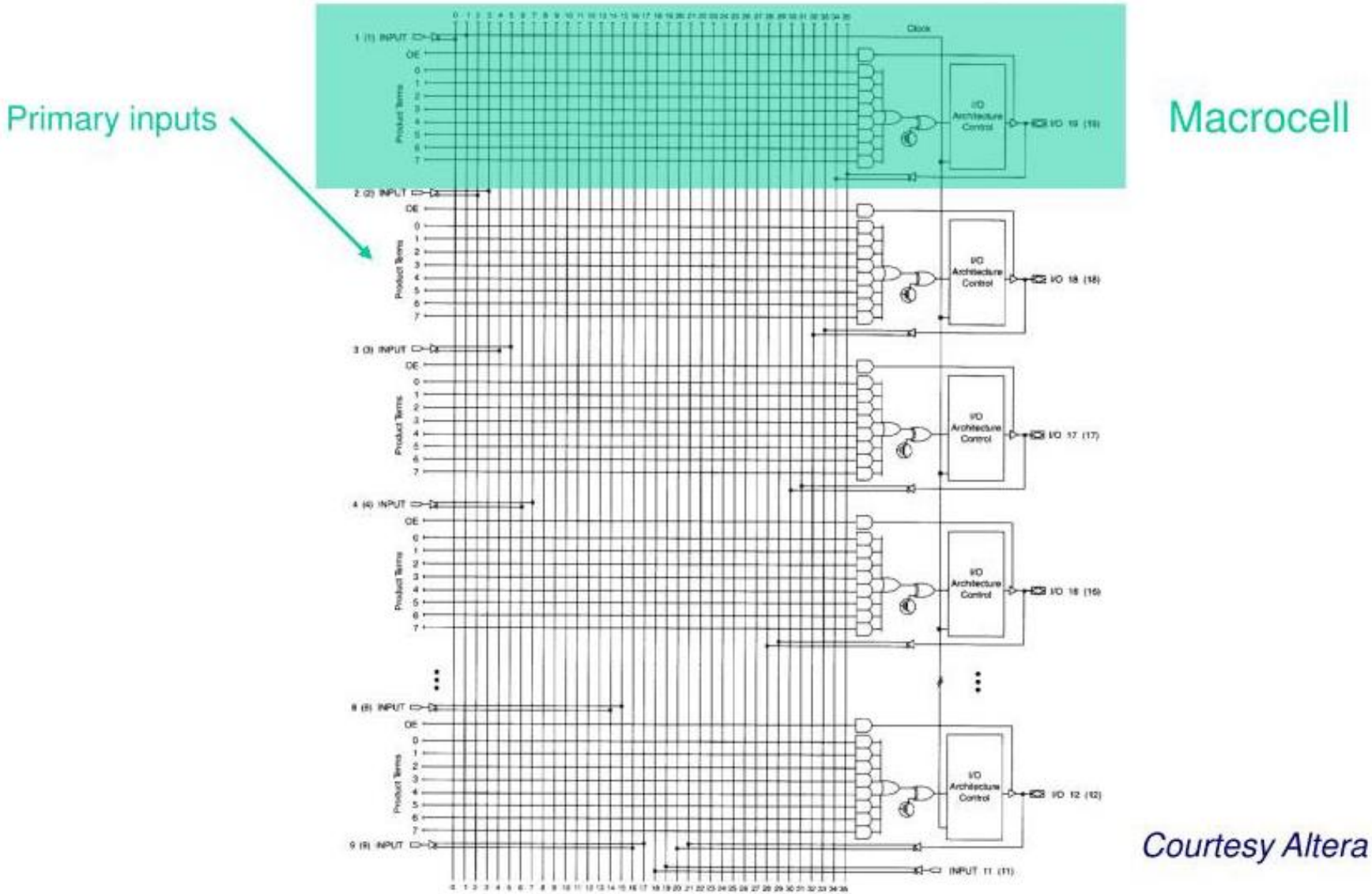
Array-based Design Implementation

Nonvolatile FPGA's

- programming similar to PROM
- erasable programmable logic devices - EPLD
- electrically erasable - EEPLD
- design partitioned into macrocells
- flip-flops used to make sequential circuits
- software used to program interconnections to optimize use of hardware
- input specified from schematics, truth tables, state graphs, VHDL code

Array-based Design Implementation

EPLD Block Diagram



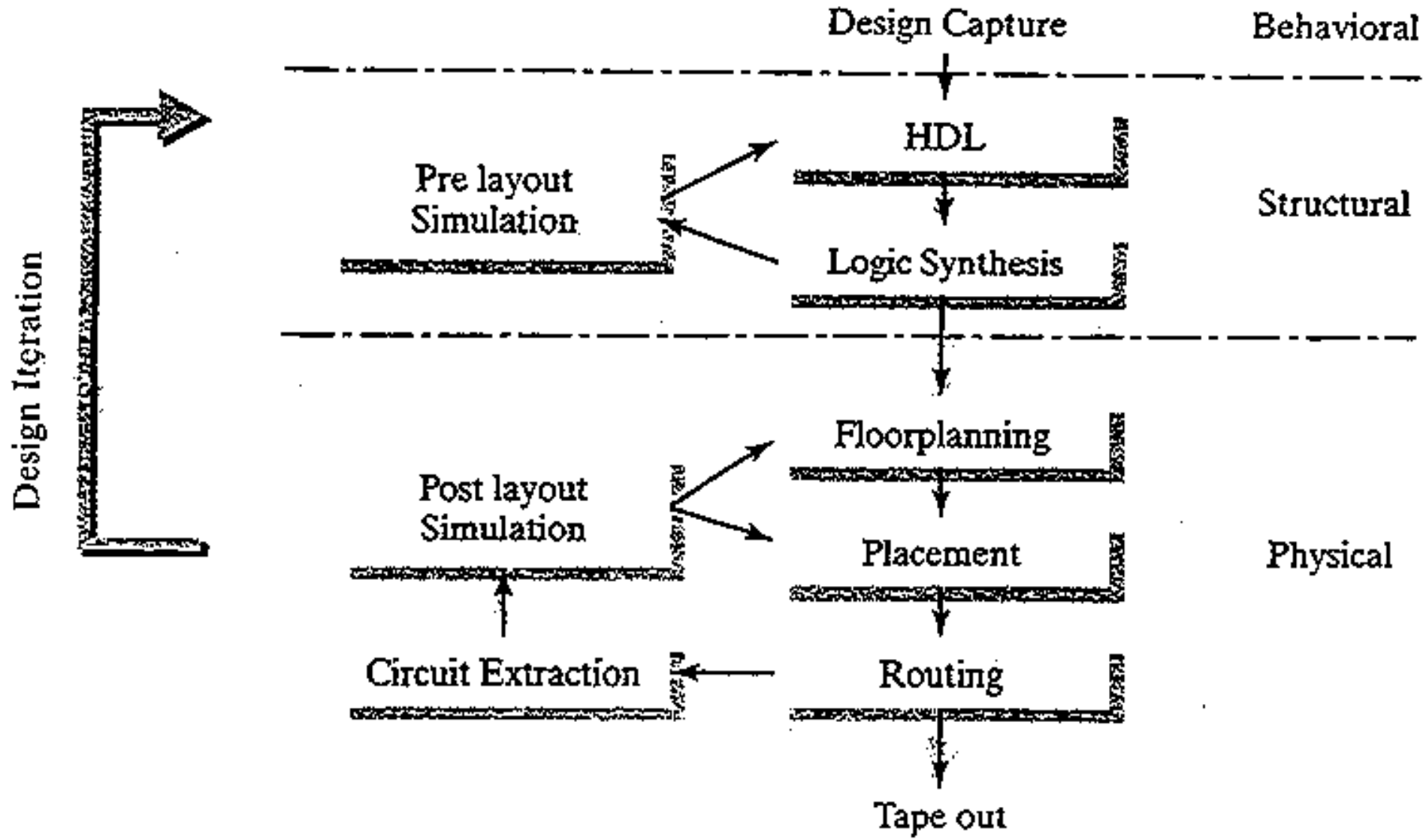
Courtesy Altera Corp.

Array-based Design Implementation

RAM based (volatile) FPGA's

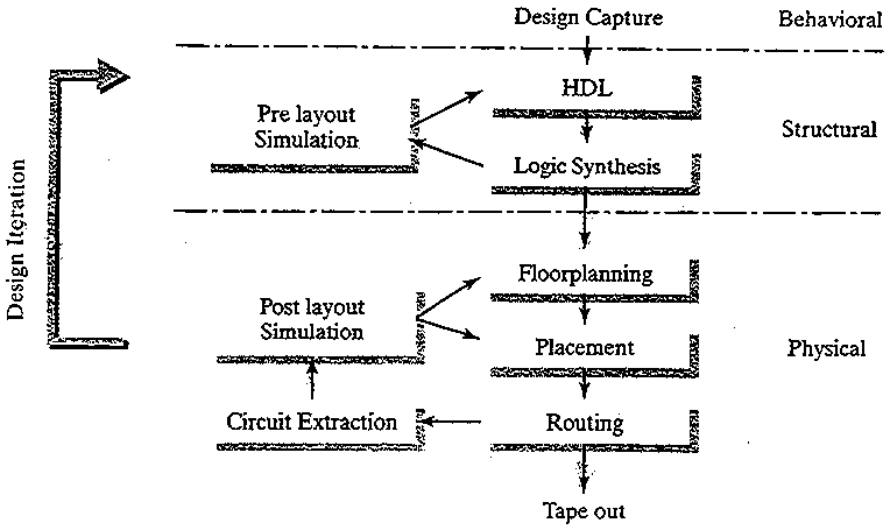
- programming is fast and can be repeated many times
- no high voltage needed
- integration density is high
- information lost when the power goes off

Semi-custom Design Flow

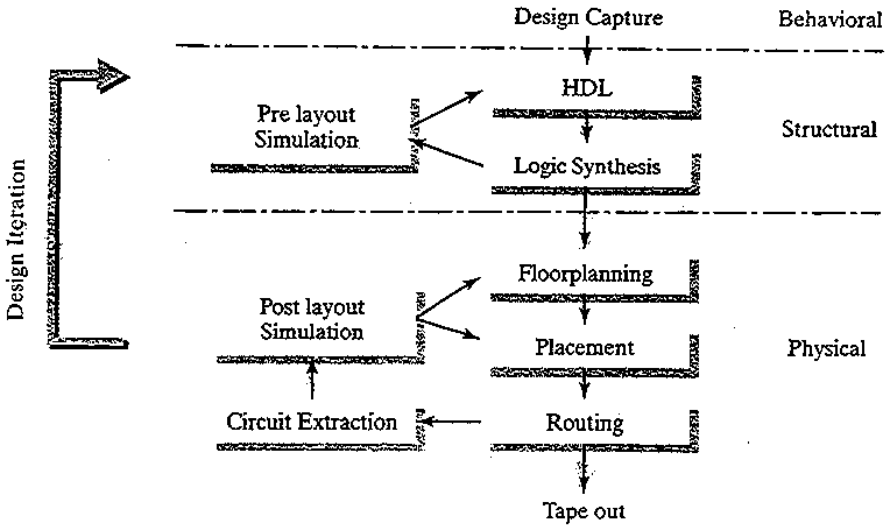


Semi-custom Design Flow

Design Capture enters the design into the ASIC design system. A variety of methods can be used to do so, including schematics and block diagrams; hardware description languages (HDLs) such as VHDL, Verilog, and, more recently, C-derivatives such as SystemC; behavioral description languages followed by high-level synthesis; and imported intellectual property modules.



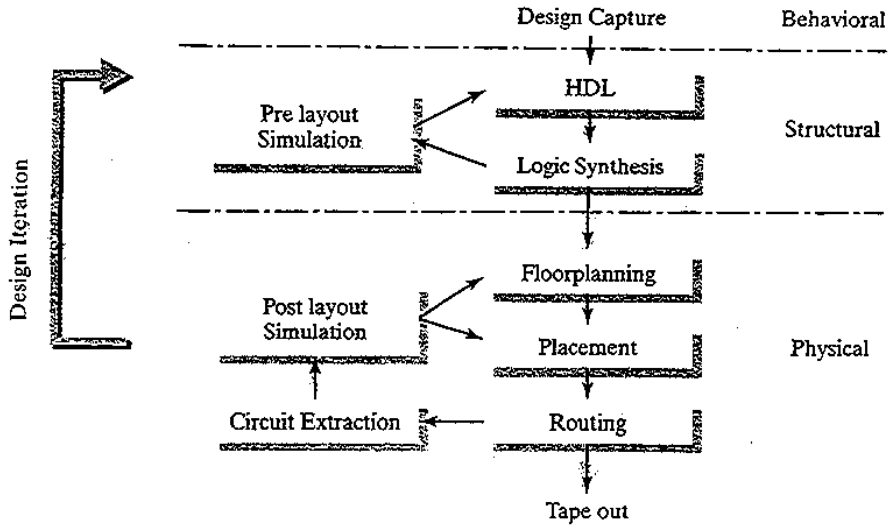
Semi-custom Design Flow



Design Capture enters the design into the ASIC design system. A variety of methods can be used to do so, including schematics and block diagrams; hardware description languages (HDLs) such as VHDL, Verilog, and, more recently, C-derivatives such as SystemC; behavioral description languages followed by high-level synthesis; and imported intellectual property modules.

Logic Synthesis tools translate modules described using an HDL language into a *netlist*. Netlist of reused or generated macros can then be inserted to form the complete netlist of the design.

Semi-custom Design Flow

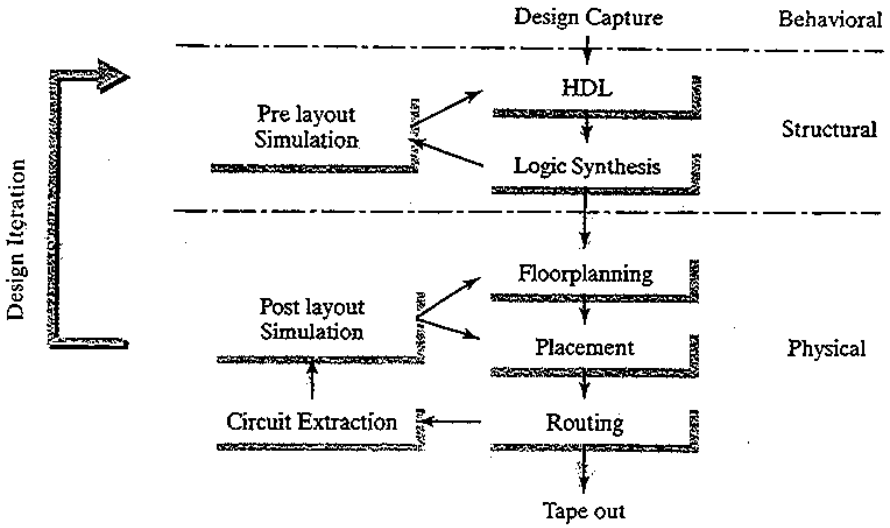


Design Capture enters the design into the ASIC design system. A variety of methods can be used to do so, including schematics and block diagrams; hardware description languages (HDLs) such as VHDL, Verilog, and, more recently, C-derivatives such as SystemC; behavioral description languages followed by high-level synthesis; and imported intellectual property modules.

Logic Synthesis tools translate modules described using an HDL language into a *netlist*. Netlist of reused or generated macros can then be inserted to form the complete netlist of the design.

Prelayout Simulation and Verification is where the design is checked for correctness. Performance analysis is performed based on estimated parasitic and layout parameters. If the design is found to be nonfunctional, extra iterations over the design capture or the logic synthesis are necessary.

Semi-custom Design Flow



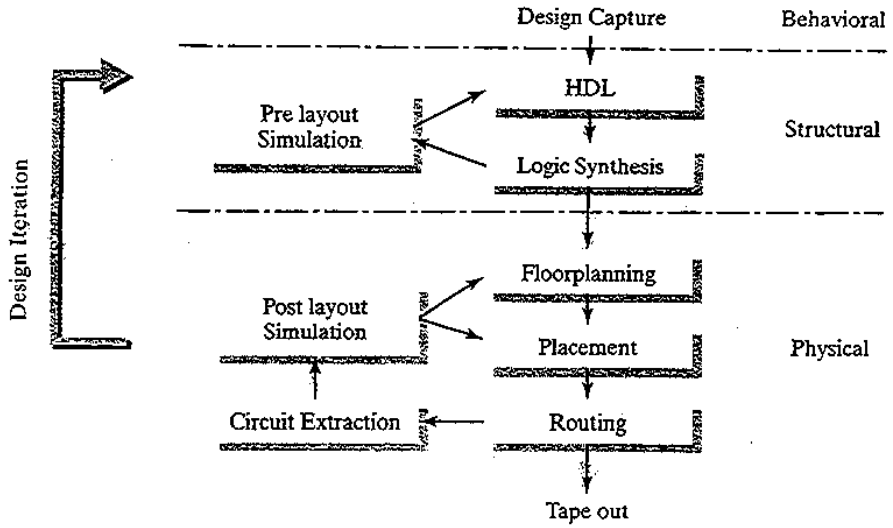
Design Capture enters the design into the ASIC design system. A variety of methods can be used to do so, including schematics and block diagrams; hardware description languages (HDLs) such as VHDL, Verilog, and, more recently, C-derivatives such as SystemC; behavioral description languages followed by high-level synthesis; and imported intellectual property modules.

Logic Synthesis tools translate modules described using an HDL language into a *netlist*. Netlist of reused or generated macros can then be inserted to form the complete netlist of the design.

Prelayout Simulation and Verification is where the design is checked for correctness. Performance analysis is performed based on estimated parasitic and layout parameters. If the design is found to be nonfunctional, extra iterations over the design capture or the logic synthesis are necessary.

Floor Planning → Based on estimated module sizes, the overall outlay of the chip is created. The global-power and clock-distribution networks also are conceived at that time.

Semi-custom Design Flow



Design Capture enters the design into the ASIC design system. A variety of methods can be used to do so, including schematics and block diagrams; hardware description languages (HDLs) such as VHDL, Verilog, and, more recently, C-derivatives such as SystemC; behavioral description languages followed by high-level synthesis; and imported intellectual property modules.

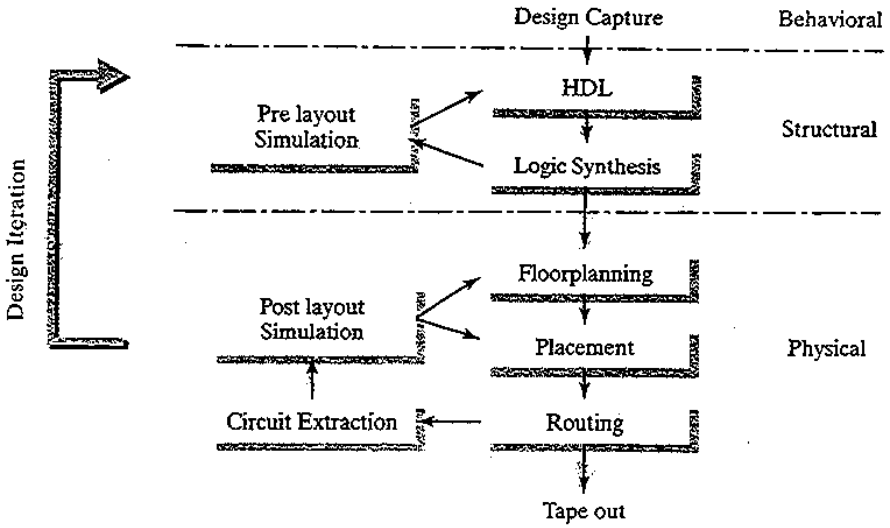
Logic Synthesis tools translate modules described using an HDL language into a *netlist*. Netlist of reused or generated macros can then be inserted to form the complete netlist of the design.

Prelayout Simulation and Verification is where the design is checked for correctness. Performance analysis is performed based on estimated parasitic and layout parameters. If the design is found to be nonfunctional, extra iterations over the design capture or the logic synthesis are necessary.

Floor Planning → Based on estimated module sizes, the overall outlay of the chip is created. The global-power and clock-distribution networks also are conceived at that time.

Placement → The precise positioning of the cells is decided.

Semi-custom Design Flow



Design Capture enters the design into the ASIC design system. A variety of methods can be used to do so, including schematics and block diagrams; hardware description languages (HDLs) such as VHDL, Verilog, and, more recently, C-derivatives such as SystemC; behavioral description languages followed by high-level synthesis; and imported intellectual property modules.

Logic Synthesis tools translate modules described using an HDL language into a *netlist*. Netlist of reused or generated macros can then be inserted to form the complete netlist of the design.

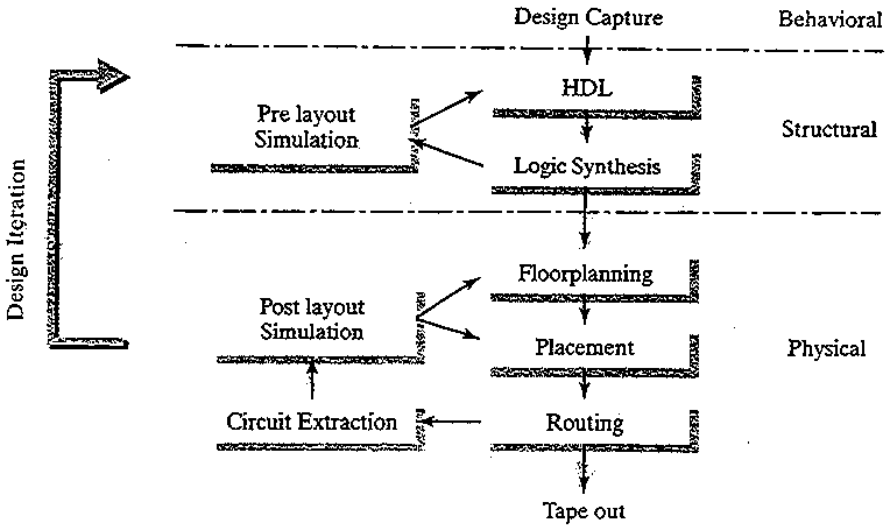
Prelayout Simulation and Verification is where the design is checked for correctness. Performance analysis is performed based on estimated parasitic and layout parameters. If the design is found to be nonfunctional, extra iterations over the design capture or the logic synthesis are necessary.

Floor Planning → Based on estimated module sizes, the overall outlay of the chip is created. The global-power and clock-distribution networks also are conceived at that time.

Placement → The precise positioning of the cells is decided.

Routing → The interconnections between the cells and blocks are wired.

Semi-custom Design Flow



Design Capture enters the design into the ASIC design system. A variety of methods can be used to do so, including schematics and block diagrams; hardware description languages (HDLs) such as VHDL, Verilog, and, more recently, C-derivatives such as SystemC; behavioral description languages followed by high-level synthesis; and imported intellectual property modules.

Logic Synthesis tools translate modules described using an HDL language into a *netlist*. Netlist of reused or generated macros can then be inserted to form the complete netlist of the design.

Prelayout Simulation and Verification is where the design is checked for correctness. Performance analysis is performed based on estimated parasitic and layout parameters. If the design is found to be nonfunctional, extra iterations over the design capture or the logic synthesis are necessary.

Floor Planning → Based on estimated module sizes, the overall outlay of the chip is created. The global-power and clock-distribution networks also are conceived at that time.

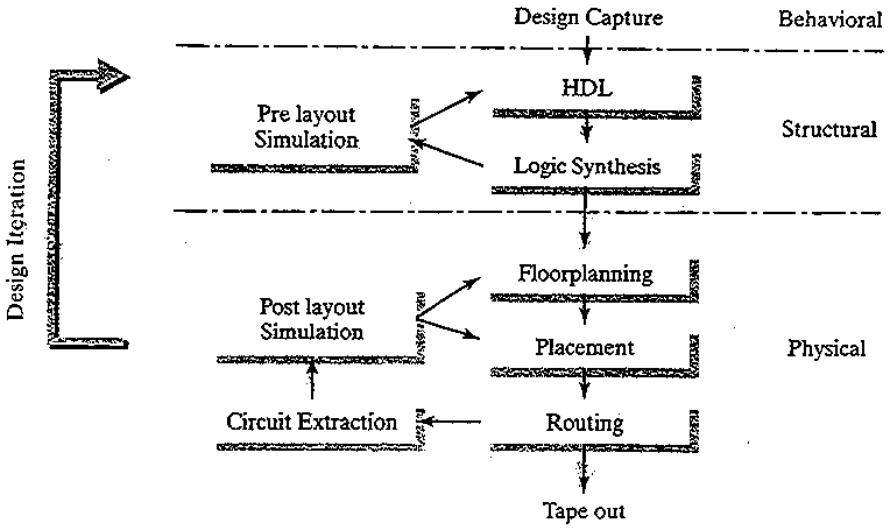
Placement → The precise positioning of the cells is decided.

Routing → The interconnections between the cells and blocks are wired.

Extraction → A model of the chip is generated from the actual physical layout, including the precise device sizes, devices parasitic, and the capacitance and resistance of the wires.

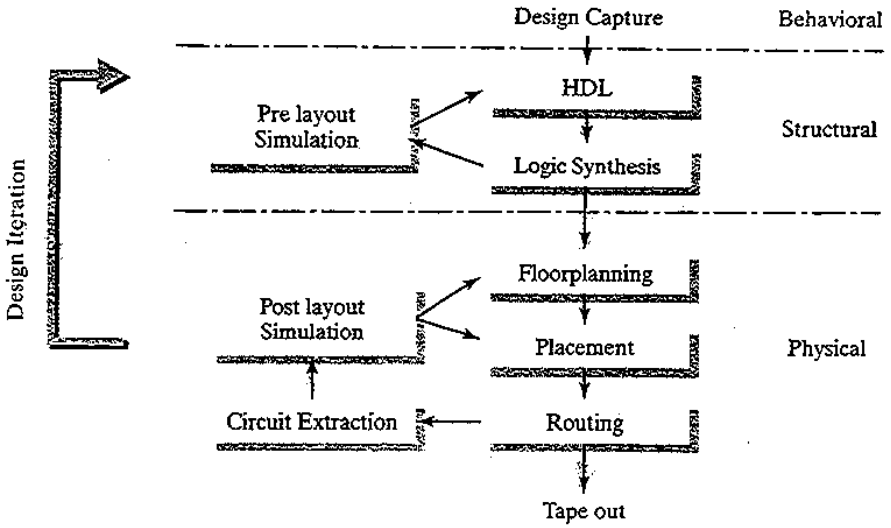
Semi-custom Design Flow

Postlayout Simulation and Verification → The functionality and performance of the chip is verified in the presence of the layout parasitic. If the design is found to be lacking, iterations on the floorplanning, placement, and routing might be necessary. Very often, this might not solve the problem, and another round of the structural design phase might be necessary.



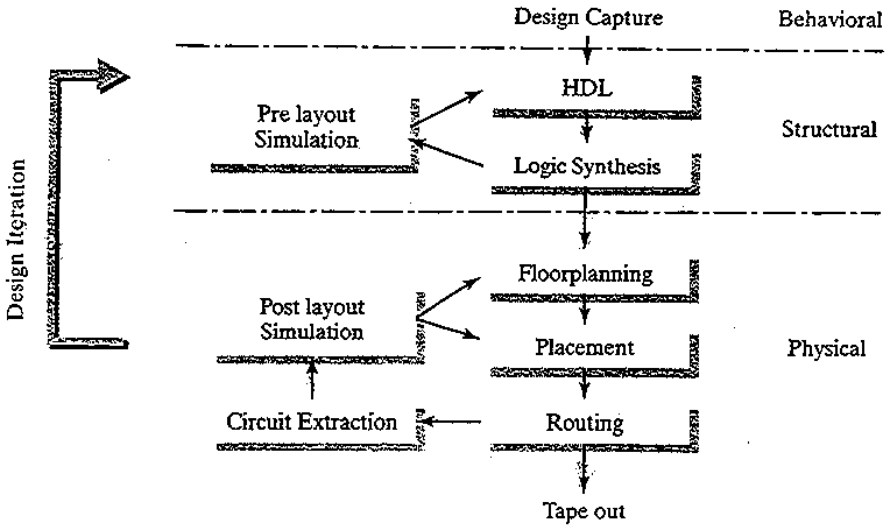
Semi-custom Design Flow

Postlayout Simulation and Verification → The functionality and performance of the chip is verified in the presence of the layout parasitic. If the design is found to be lacking, iterations on the floorplanning, placement, and routing might be necessary. Very often, this might not solve the problem, and another round of the structural design phase might be necessary.



Tapeout → Once the design is found to be meeting all design goals and functions, a binary file is generated containing all the information needed for mask generation. This file is then sent out to the ASIC vendor or foundry. This important moment in the life of a chip is called *tapeout*.

Semi-custom Design Flow



Postlayout Simulation and Verification → The functionality and performance of the chip is verified in the presence of the layout parasitic. If the design is found to be lacking, iterations on the floorplanning, placement, and routing might be necessary. Very often, this might not solve the problem, and another round of the structural design phase might be necessary.

Tapeout → Once the design is found to be meeting all design goals and functions, a binary file is generated containing all the information needed for mask generation. This file is then sent out to the ASIC vendor or foundry. This important moment in the life of a chip is called *tapeout*.