VLSI Design Cycle

Concept of EDA

Where simulation stands

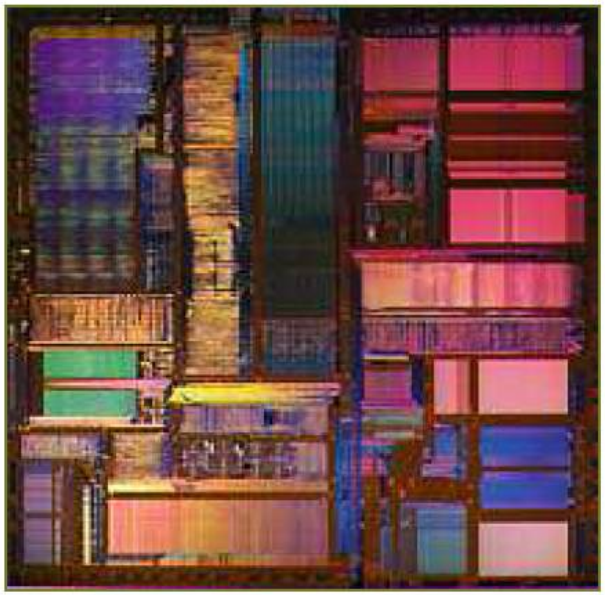
Design cycle

EDA : The current Wikipedia definition of EDA is “the category of tools for designing and producing electronic systems ranging from printed circuit boards (PCBs) to integrated circuits. This is sometimes referred to as ECAD (electronic computer-aided design) or just CAD.” The workshop attendees felt that this definition was somewhat narrow, as it focuses mainly on the use of EDA technologies.

1. EDA consists of a collection of methodologies, algorithms and tools, which assist and automate the design, verification, and testing of electronic systems.
2. It embodies a general methodology that seeks to successively refine a high-level description to low-level detailed physical implementation for designs ranging from integrated circuits (including system-on-chips), to printed circuit boards (PCBs) and electronic systems.
3. It involves modeling, synthesis, and verification at every level of abstraction.

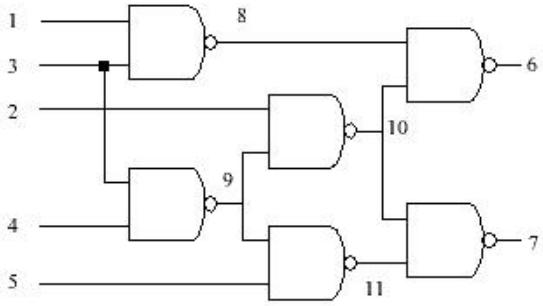
EDA: Software engineers use to design integrated circuits
Most tools focus on large digital ICs:

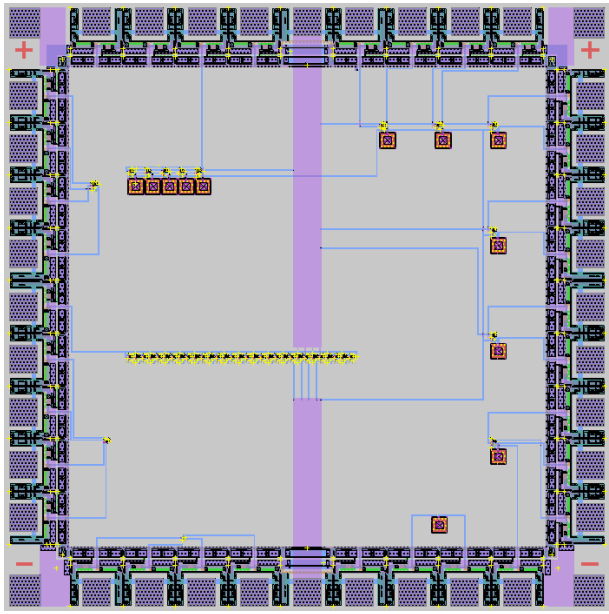
- Microprocessors (Intel, Sun, Motorola)
- Graphics chips (nVidia, S3)
- Digital signal processors (TI, Motorola)



If a city is as crowded as a chip and if you have to find out new routes with narrow lanes

- Hardware Simulation
- Hardware Compilers
- Place & Route
- Formal Verification
- Mask generation
- Semiconductor Simulation





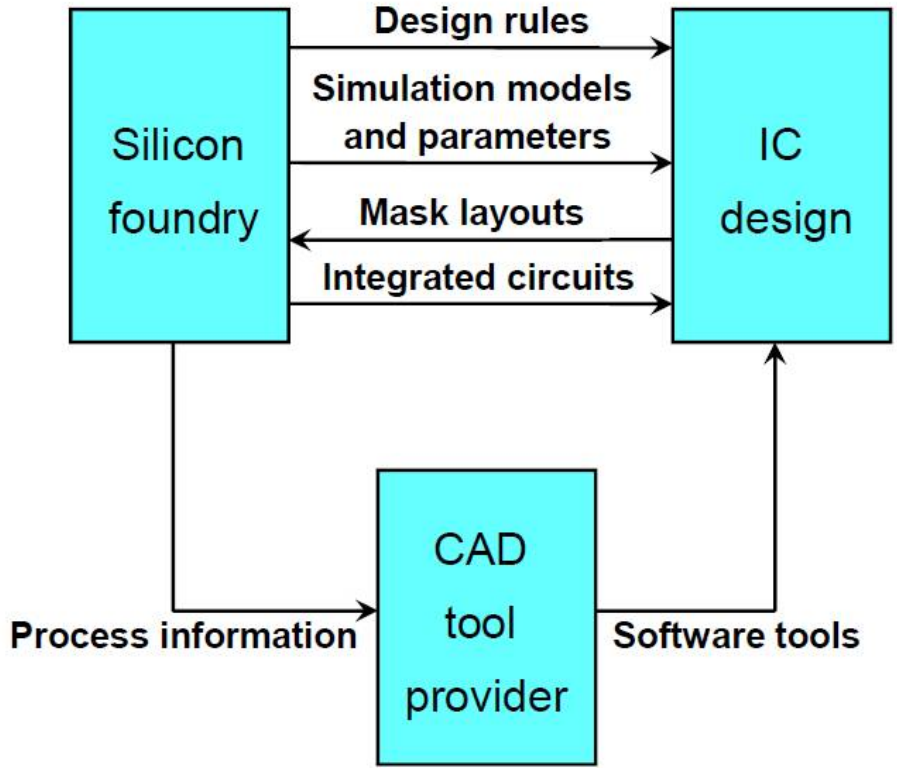
VLSI Design Methodologies

Design methodology

Process for creating a design

Methodology goals

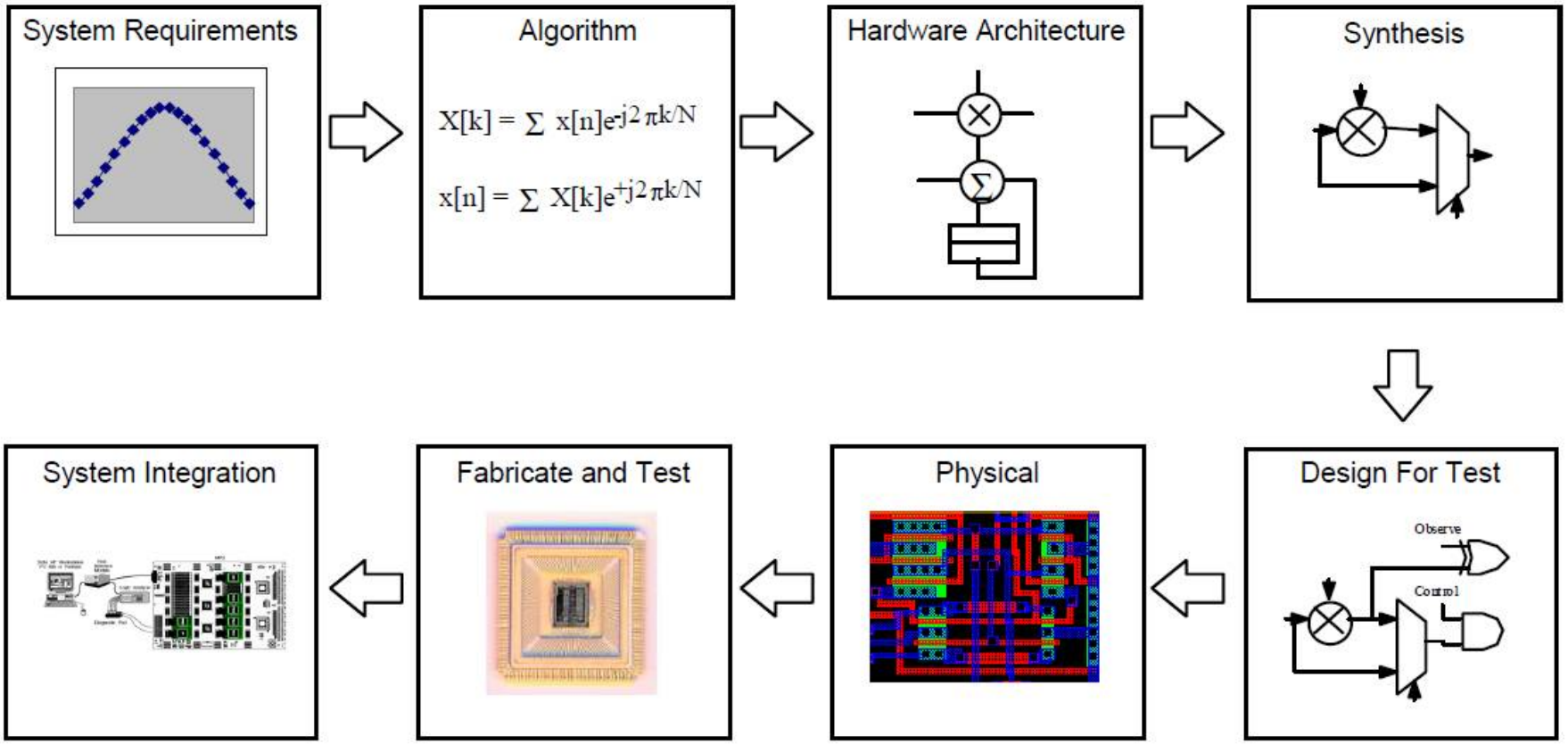
- Design cycle
- Complexity
- Performance
- Reuse
- Reliability



- SSI (Small-Scale Integration)-(1962)
  - Tens of Transistors
  - NAND, NOR
- MSI (Medium-Scale Integration)-(late 1960)
  - Hundreds of Transistors
  - Counters
- LSI (Large-Scale Integration)-(mid 1970)
  - Tens of Thousands of Transistors
  - First Microprocessor
- VLSI (Very Large-Scale Integration)-(1980)
  - started Hundreds of Thousands of Transistors-several billion transistors in 2009
  - 64 bit Microprocessor with cache memory and floating-point arithmetic units
- ULSI (Ultra Large-Scale Integration)-(late 1980)
  - More than about one million circuit elements on a single chip.
  - The Intel 486 and Pentium microprocessors, use ULSI technology

- Bipolar
  - More accuracy
- MOS
  - Gate-Aluminium
  - Low power consumption
  - Low cost
- CMOS
  - Gate-Poly-Silicon
  - Low power consumption
  - Low cost
- BiCMOS

System to Silicon Design



- Standard ICs
- Glue Logic-Microelectronic system design then becomes a matter of defining the functions that you can implement using standard ICs and then implementing the remaining logic functions (sometimes called glue logic) with one or more custom ICs.
- ASIC
- ASSP **(Application-Specific Standard Products)**

Examples of **ICs that are *not ASICs*** *include standard parts such as:*

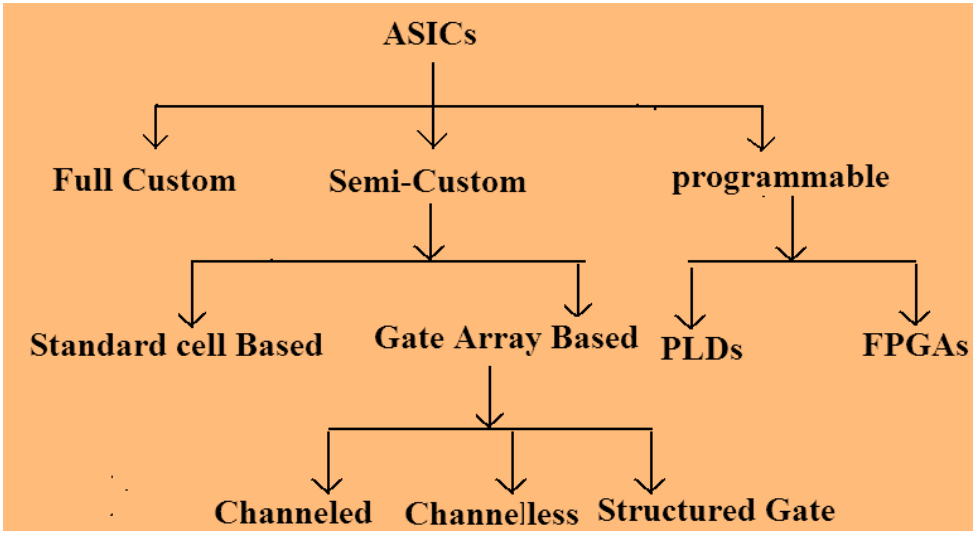
- *memory chips* sold as a commodity item—ROMs, DRAM, and SRAM; microprocessors;
- TTL or TTL-equivalent ICs at SSI, MSI, and LSI levels.

Examples of **ICs that *are ASICs* *include:***

- *a chip for a toy bear that talks;*
- *a chip for a* satellite;
- a chip designed to handle the interface between memory and a microprocessor for a workstation CPU;
- a chip containing a microprocessor as a cell together with other logic.

ASSP (two ICs that might or might not be considered ASICs)

- controller chip for a PC and a chip for a modem.
- Both of these examples are specific to an application (shades of an ASIC) but are sold to many different system vendors (shades of a standard part).



Full-Custom ASICs

- A Full custom ASIC is one which includes some (possibly all) logic cells that are customized and all mask layers that are customized.
- A microprocessor is an example of a full-custom IC . Designers spend many hours squeezing the most out of every last square micron of microprocessor chip space by hand.
- Customizing all of the IC features in this way allows designers to include analog circuits, optimized memory cells, or mechanical structures on an IC, for example. Full-custom ICs are the most expensive to manufacture and to design.

- The manufacturing lead time (the time required just to make an IC not including design time) is typically eight weeks for a full-custom IC.
- These specialized full-custom ICs are often intended for a specific application so, we might call some of them as full-custom ASICs.

Semicustom ASICs

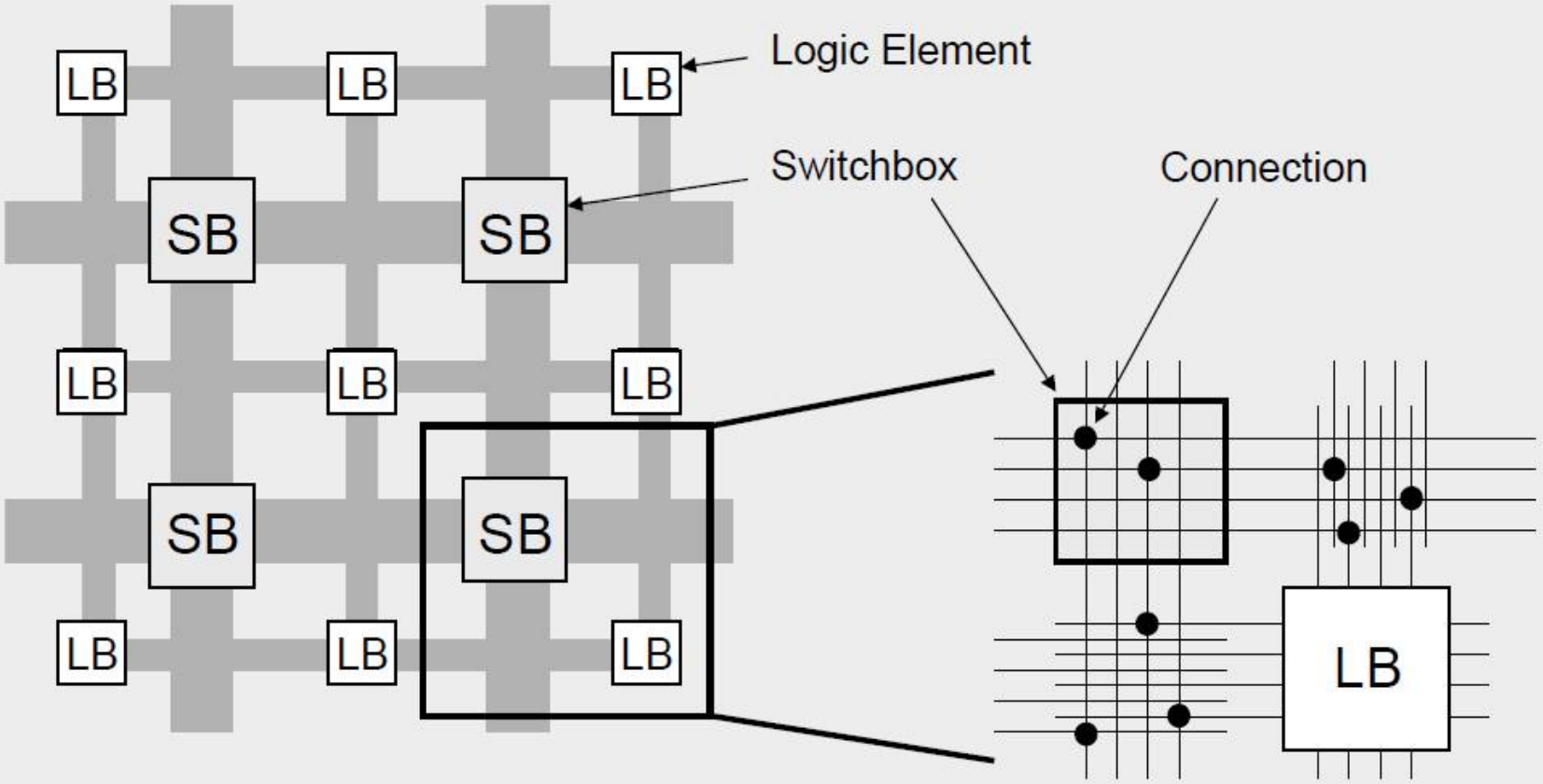
- ASICs , for which all of the logic cells are predesigned and some (possibly all) of the mask layers are customized are called semi custom ASICs.
- Using the predesigned cells from a cell library makes the design , much easier.
- There are two types of semicustom ASICs
- (i) Standard-cell–based ASICs (ii)Gate-array–based ASICs.

Standard-Cell Based ASICs

- A cell-based ASIC (cell-based IC, or CBIC pronounced sea-bick) uses predesigned logic cells (AND gates, OR gates, multiplexers, and flip-flops, for example) known as standard cells.
- One can apply the term CBIC to any IC that uses cells, but it is generally accepted that a cell-based ASIC or CBIC means a standard-cell based ASIC.

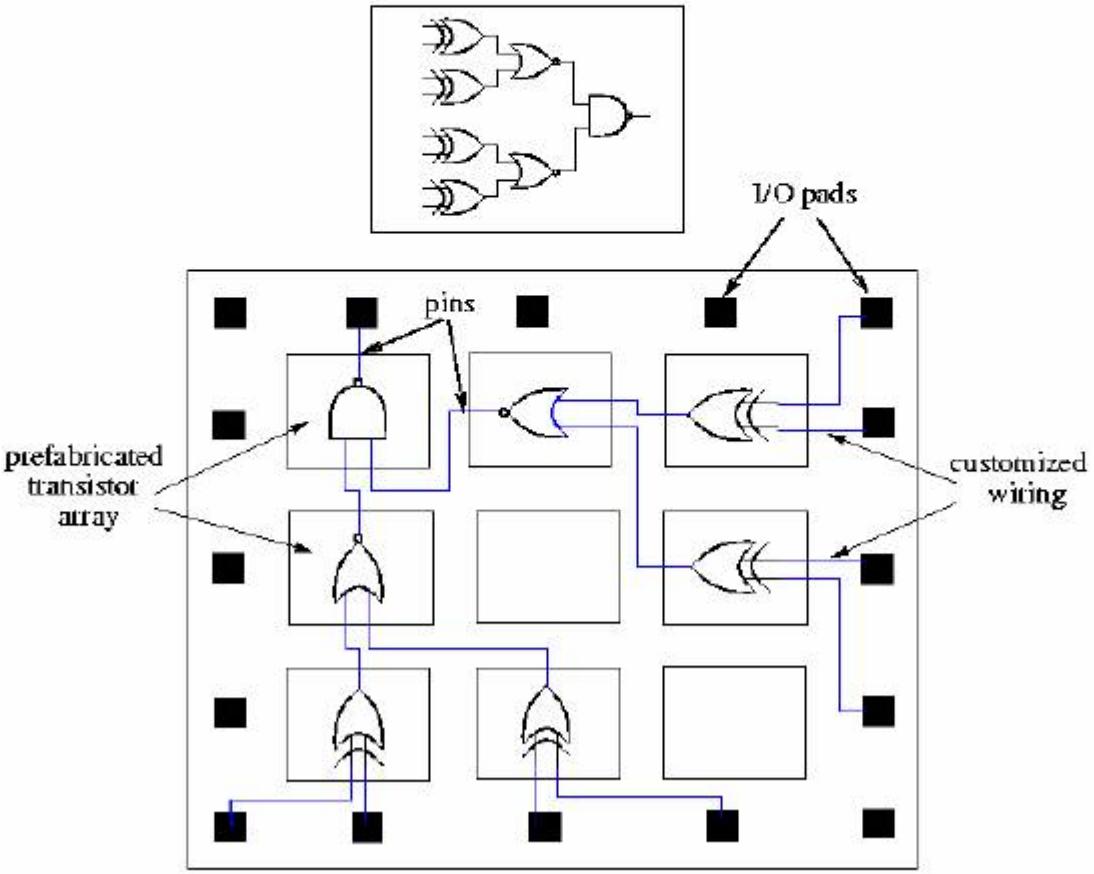
FPGA

Field-programmable gate array (FPGA)



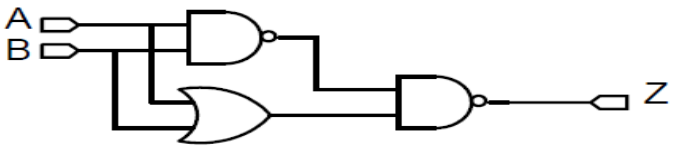
Gate Array Design Style

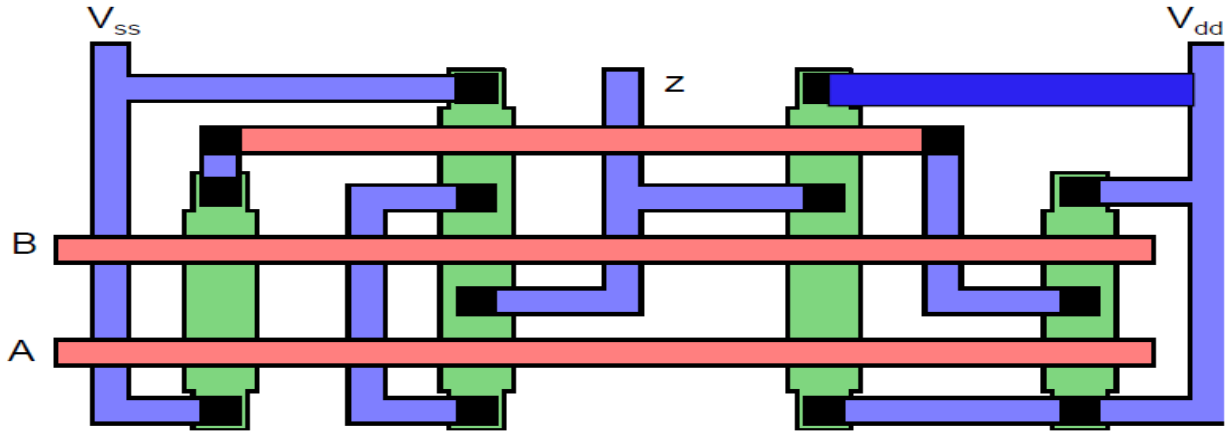
- Prefabricates a transistor array
- Needs wiring customization to implement logic



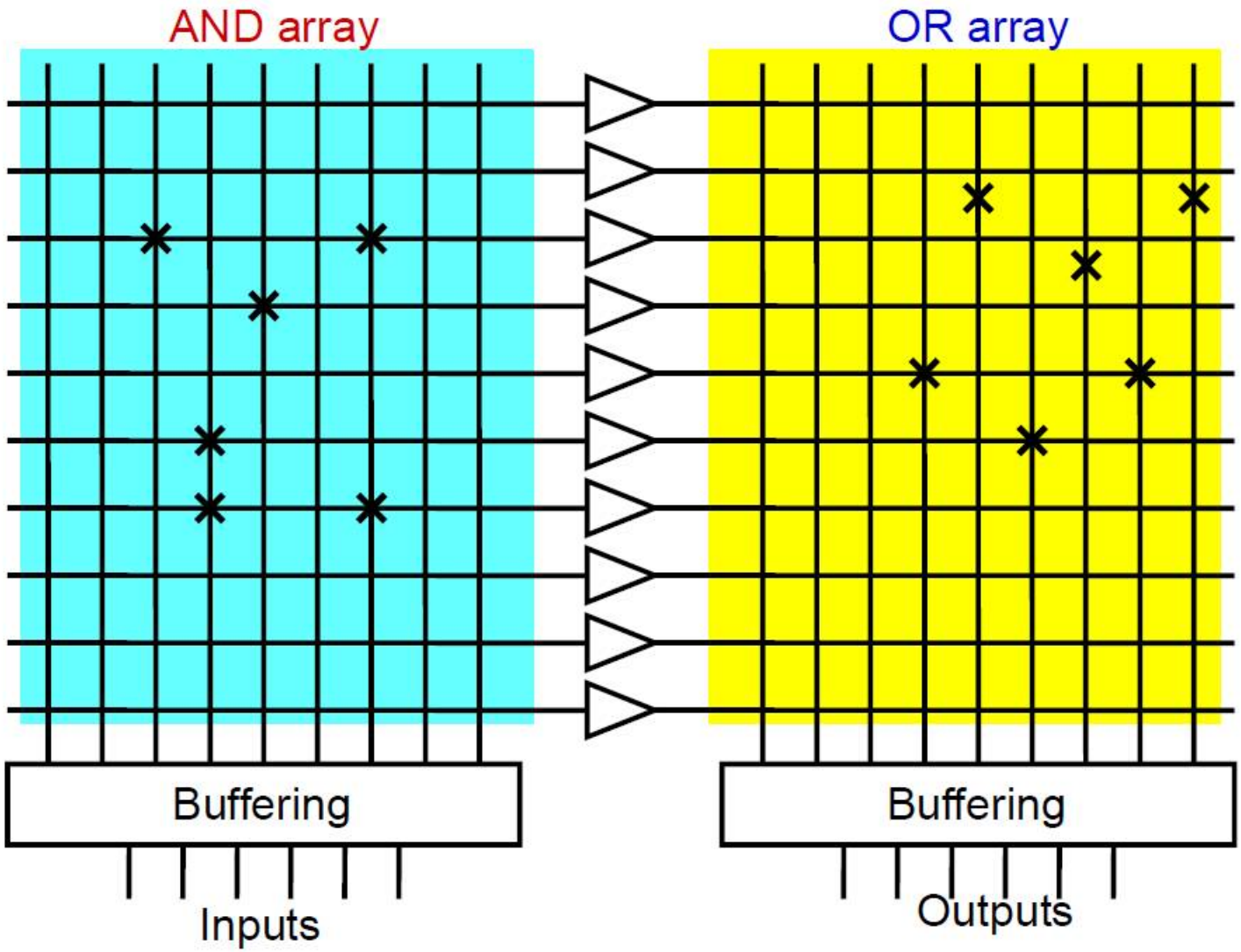
	Full custom	Standard cell	Gate array	FPGA
Cell size	variable	fixed height*	fixed	fixed
Cell type	variable	variable	fixed	programmable
Cell placement	variable	in row	fixed	fixed
Interconnections	variable	variable	variable	programmable

Full Custom

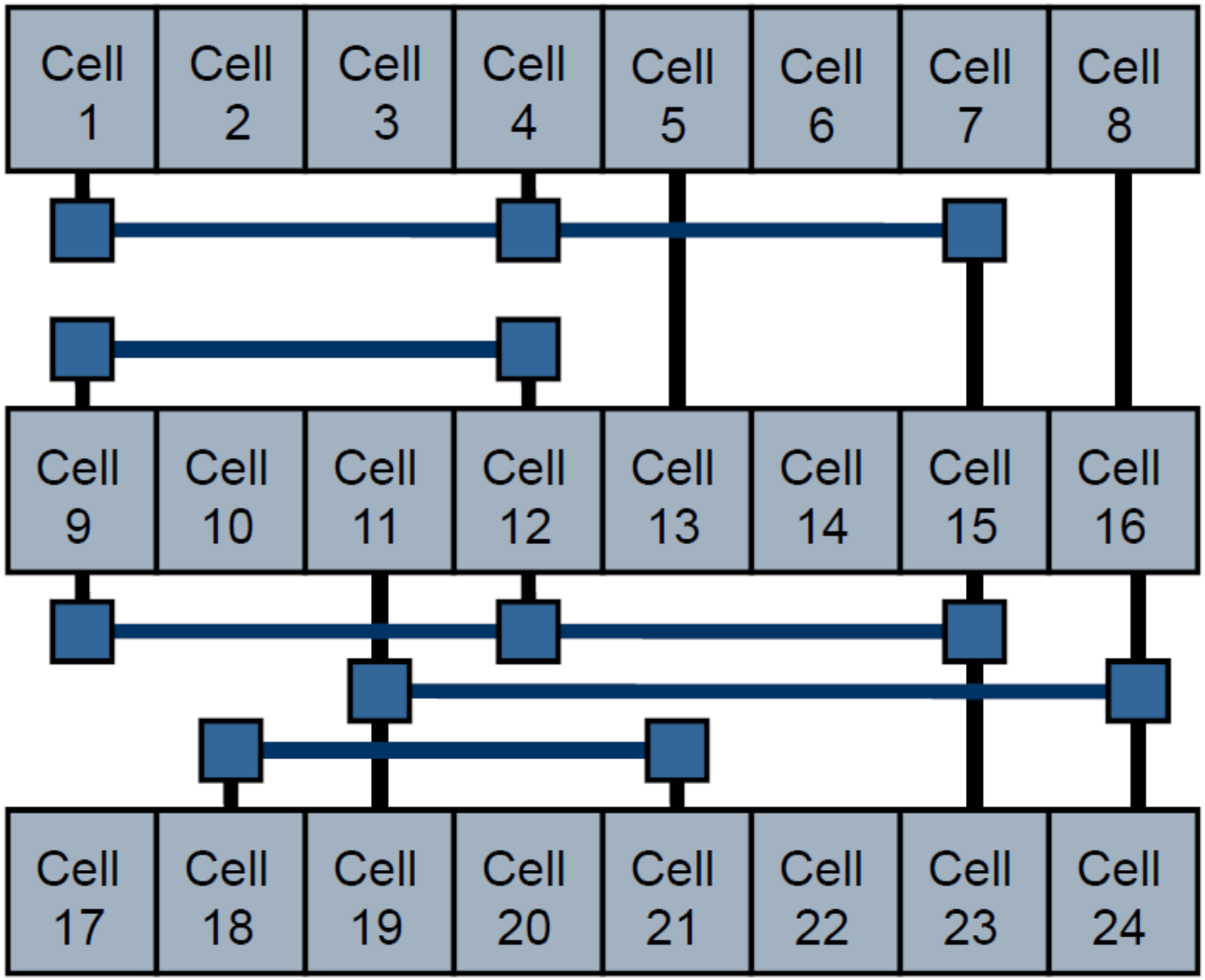




PLA



Gate Array



In a gate array (sometimes abbreviated GA) or gate-array based ASIC the transistors are predefined on the silicon wafer

Can the design process depends on frequency of operation?

PLL

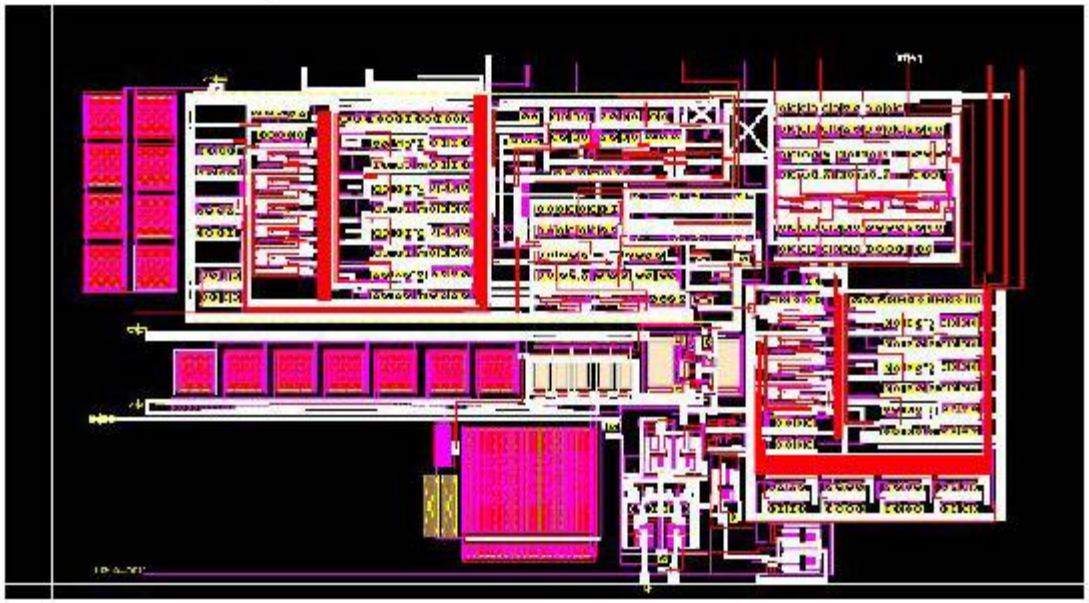
Electrical specifications:

Parameter	Descriptions	Min.	Typ.	Max.	Units
VCO	Frequency Range	500		1000	MHz
	Tuning Voltage	0.6		1.8	V
	Phase noise@100 KHz offset		–115		dBc/Hz
	Phase noise@1MHz Offset		-140		dBc/Hz
Phase Frequency Detector (PFD)	PFD frequency		10		MHz
Charge Pump	Icp Source/sink	.9		2.8	mA
Dual Mode Divider	Division Range	16		127	
IQ Generator	Phase difference		90		Degree
	Frequency Range	250		500	MHz

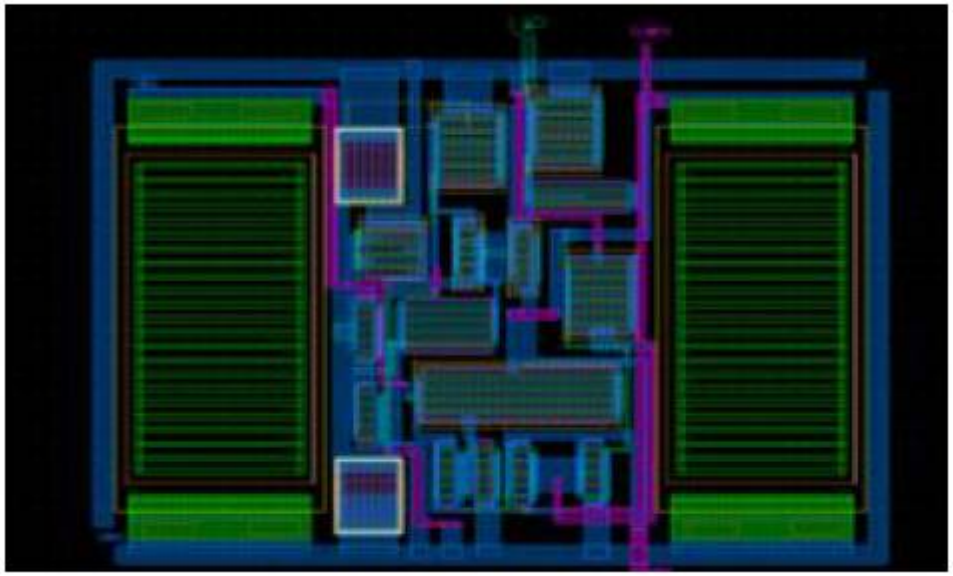
Features:

1. On-Chip VCO tunes from 500 MHz to 1000 MHz
2. Programmable Divider (16-127 division)
3. High Speed In-Phase & Quadrature-Phase output Signal (250-500 MHz)
4. Low Speed Signal (15-30 MHz)
5. Different Reference Frequency (10-40 MHz)
6. Low dynamic power consumption in locked state
7. Prescalar 4/5
8. Duty cycle (45% to 55%)
9. Temperature stability (-40 to $+125^0$ C)

PLL Layout

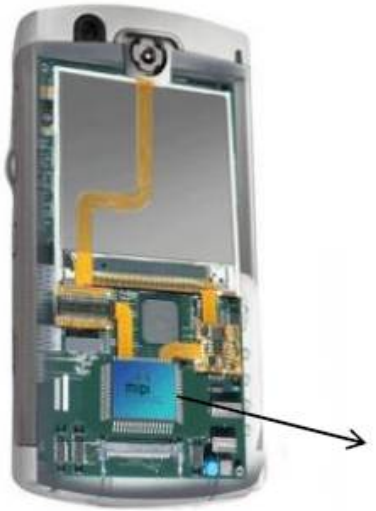


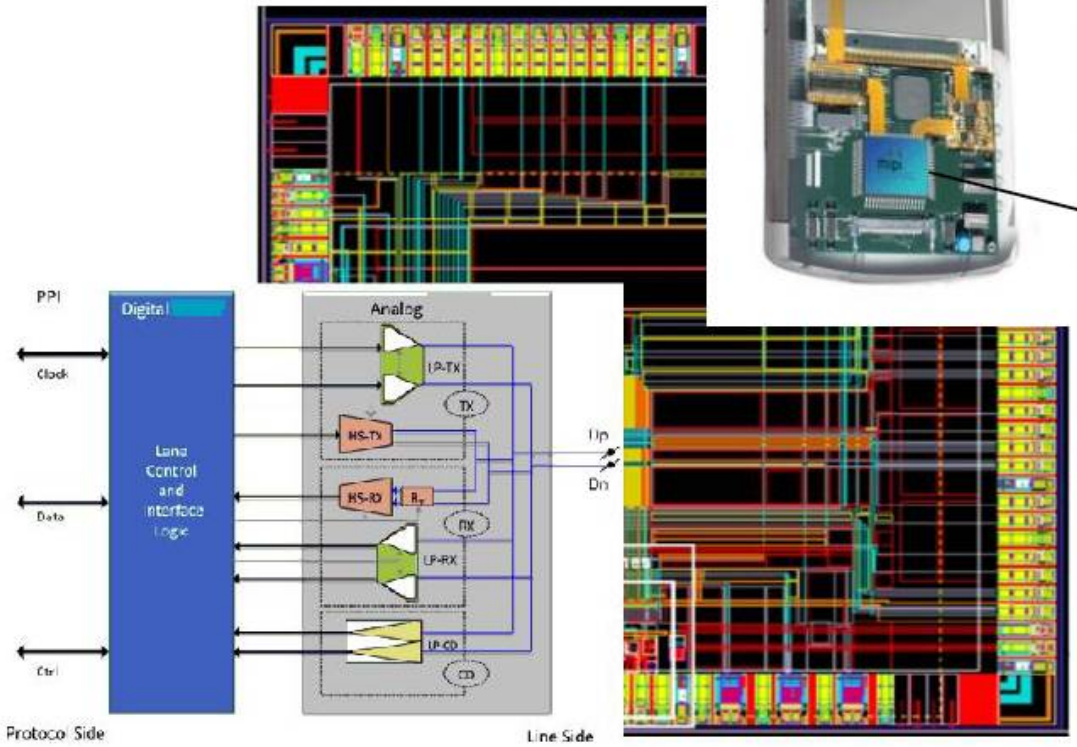
Charge pump Layout



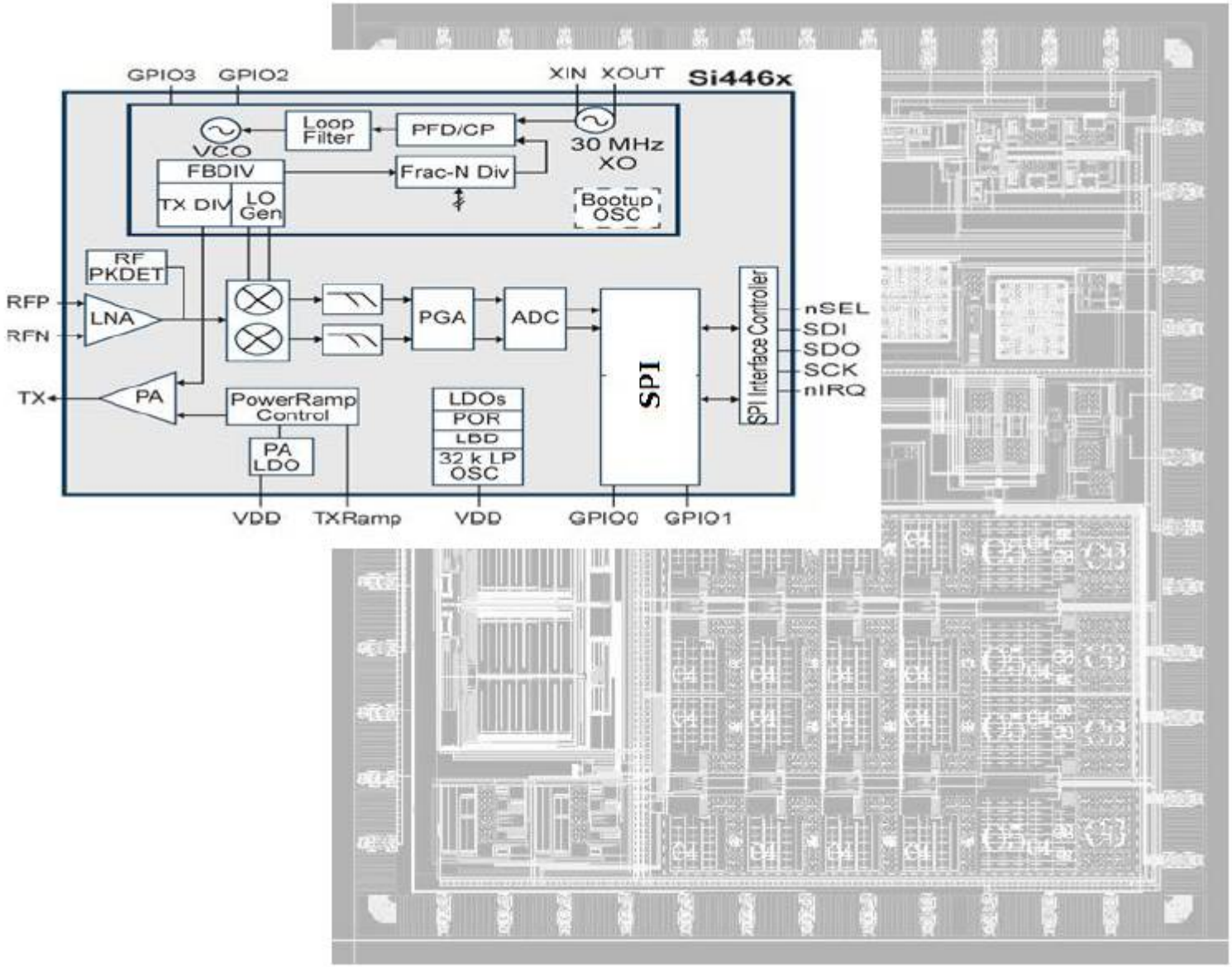
Complex CMOS transceiver design

- Latest mobile standard complaint
- Applications: - Mobile Phone Camera interface
- Mobile Phone Display interface



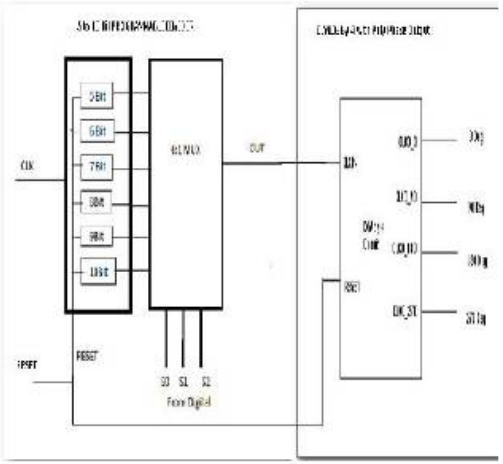


802.15.4g CMOS 915 MHz Transceiver



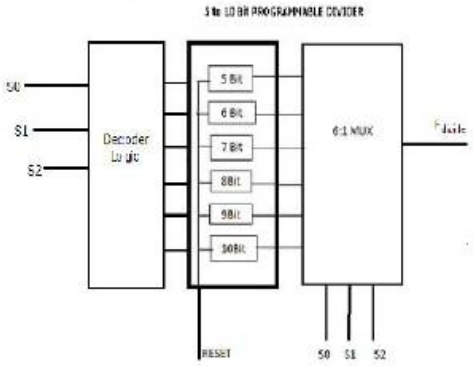
Poly Phase Programmable Frequency Divider

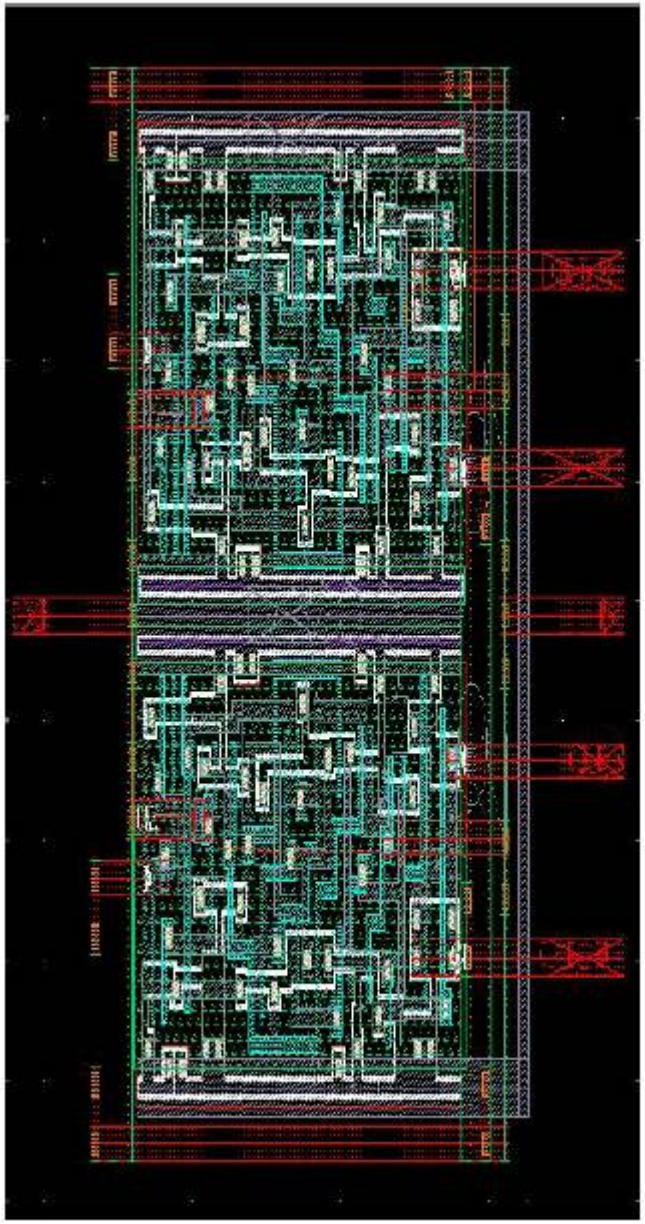
- This Poly phase CMOS Programmable Frequency Divider IP fabricated on 130 nm low power RF CMOS Technology. The module contains decoder logic, Divider logic and a multiplexer logic. This frequency divider gives the output in poly phase form with four different output phase.



Programmable Frequency Divider

- Wide range of frequency (50-2400 MHz) can be provided as input clock and programmable divider will be responsible for dividing the frequency by a particular frequency divider ration.

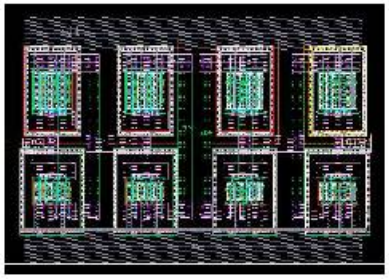




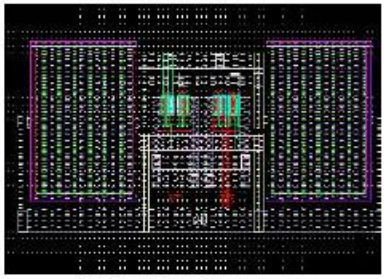
GaAs 0.5um PS 50-70 pHEMT Process

- 15 Designs for various wireless applications.
- Designs Covering 0.5GHz to 12GHz frequency range.
- Power amplifier designs for low to high power applications up to 0.5 Watt.
- High performance Low noise amplifiers covering 2GHz to 8GHz.
- Other high performance designs like DA, PA, LNA,Oscillator,RF Switch, etc along with flexible amplifier test structures on the tile.
- Fabricated with Win Semi. foundry

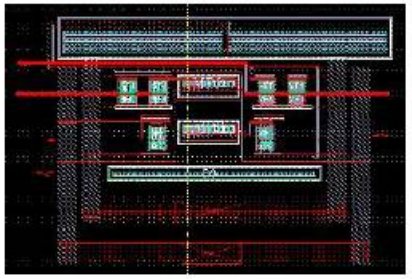
RF CMOS MMIC



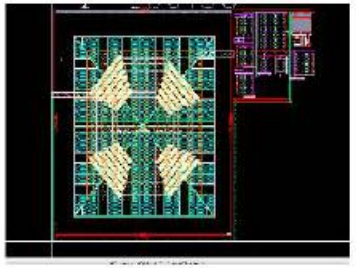
Buffer Amplifier



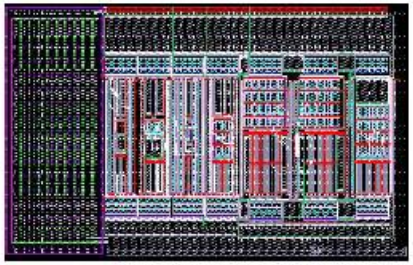
Mixer



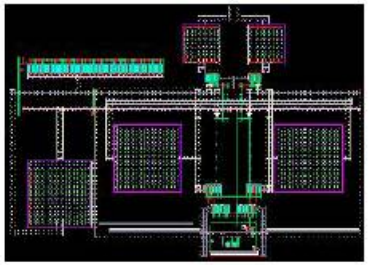
Crystal Oscillator



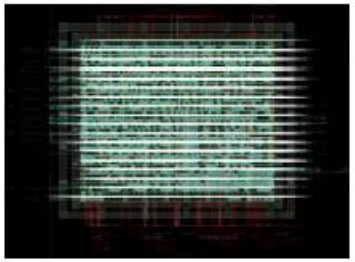
Power Amplifier



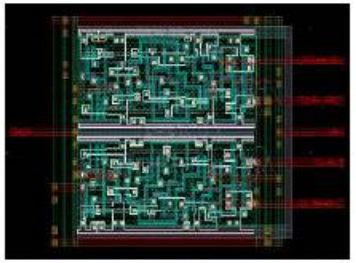
Attenuator



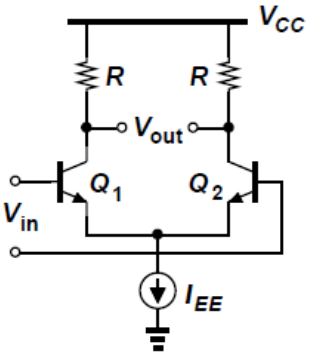
LNA

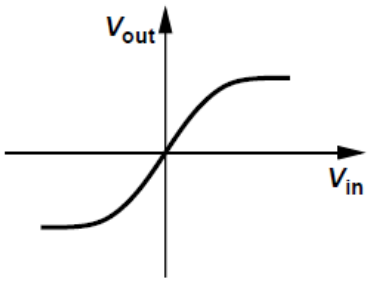


Digital Control Interface



Poly phase
Freq Divider



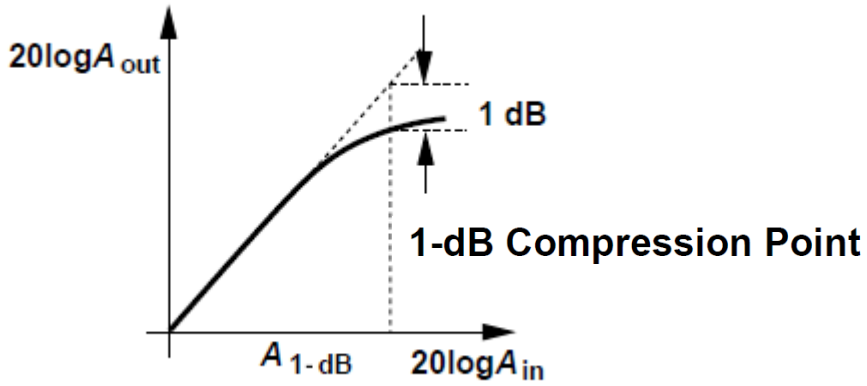


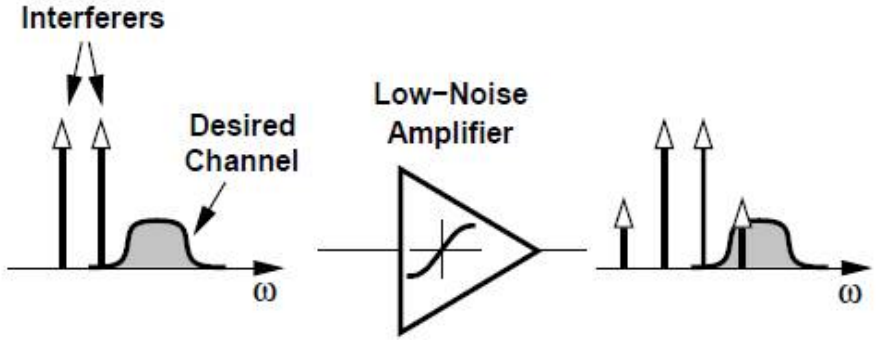
- **Harmonic distortion:**

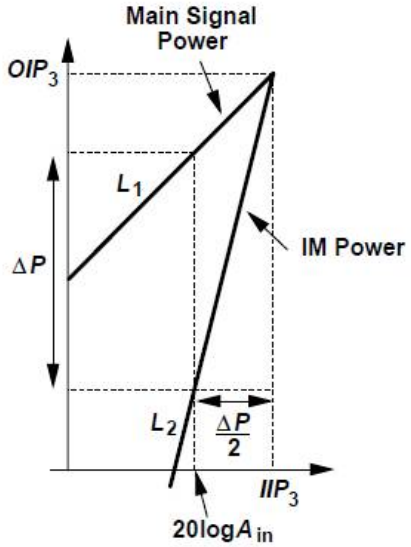
$$x(t) = A\cos\omega t,\ \ y(t) = \alpha_1 x(t) + \alpha_2 x^2(t) + \alpha_3 x^3(t) + \cdots$$

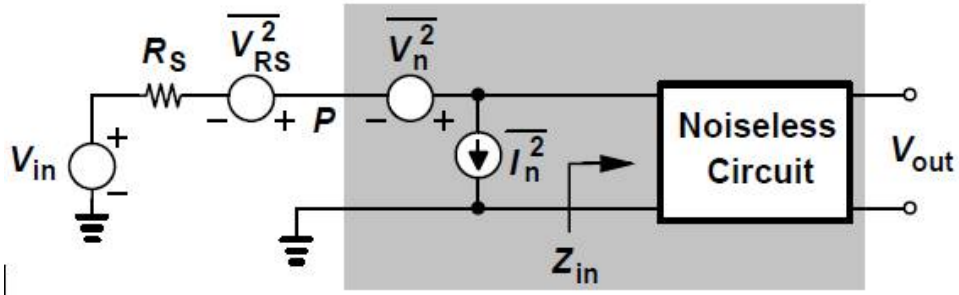
$$\Rightarrow y(t) = \alpha_1 A\cos\omega t + \alpha_2 A^2\cos^2\omega t + \alpha_3 A^3\cos^3\omega t + \cdots$$

$$= (\alpha_1 A + \frac{3}{4}\alpha_3 A^3)\cos\omega t + (\ )\cos 2\omega t + (\ )\cos 3\omega t + \cdots$$





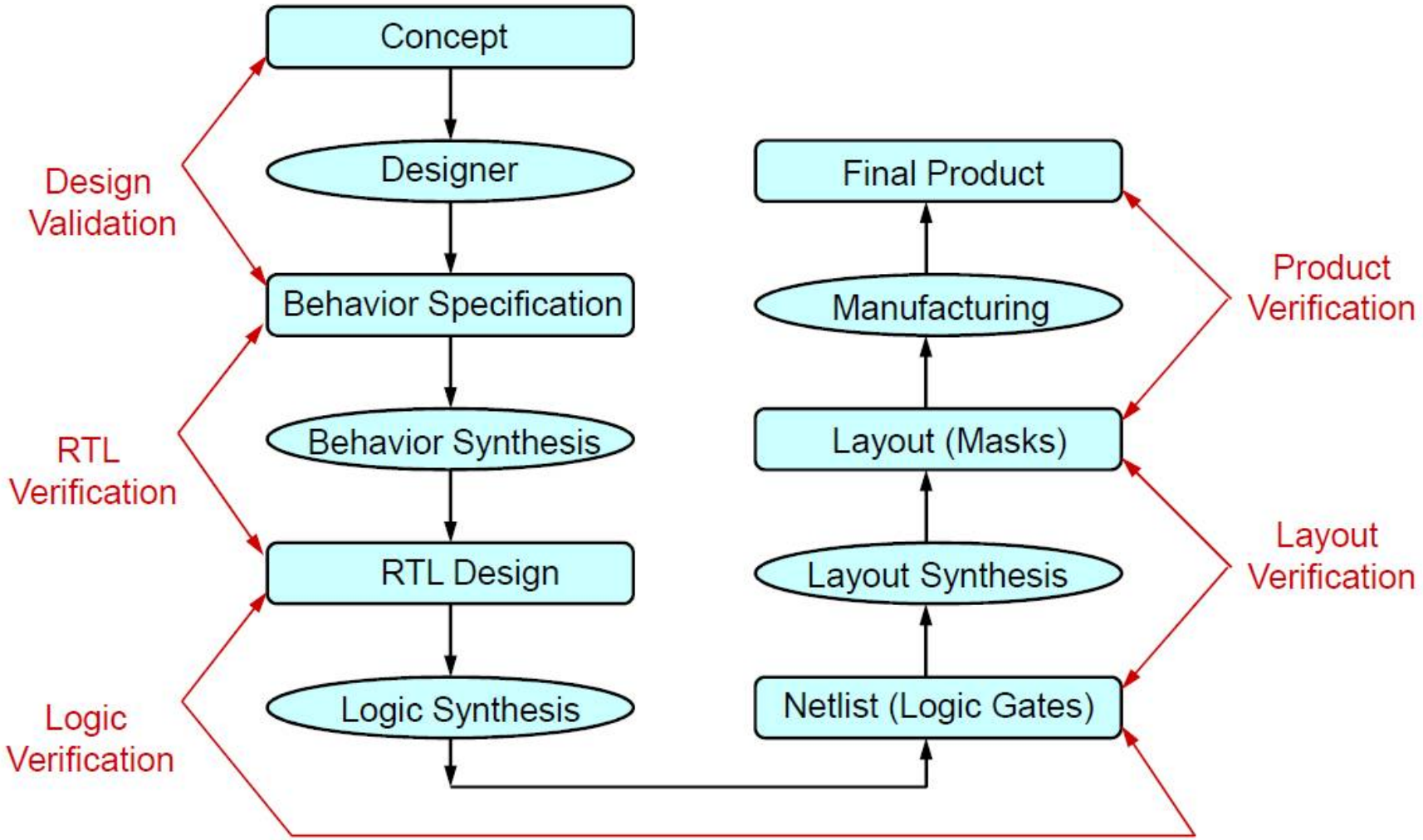


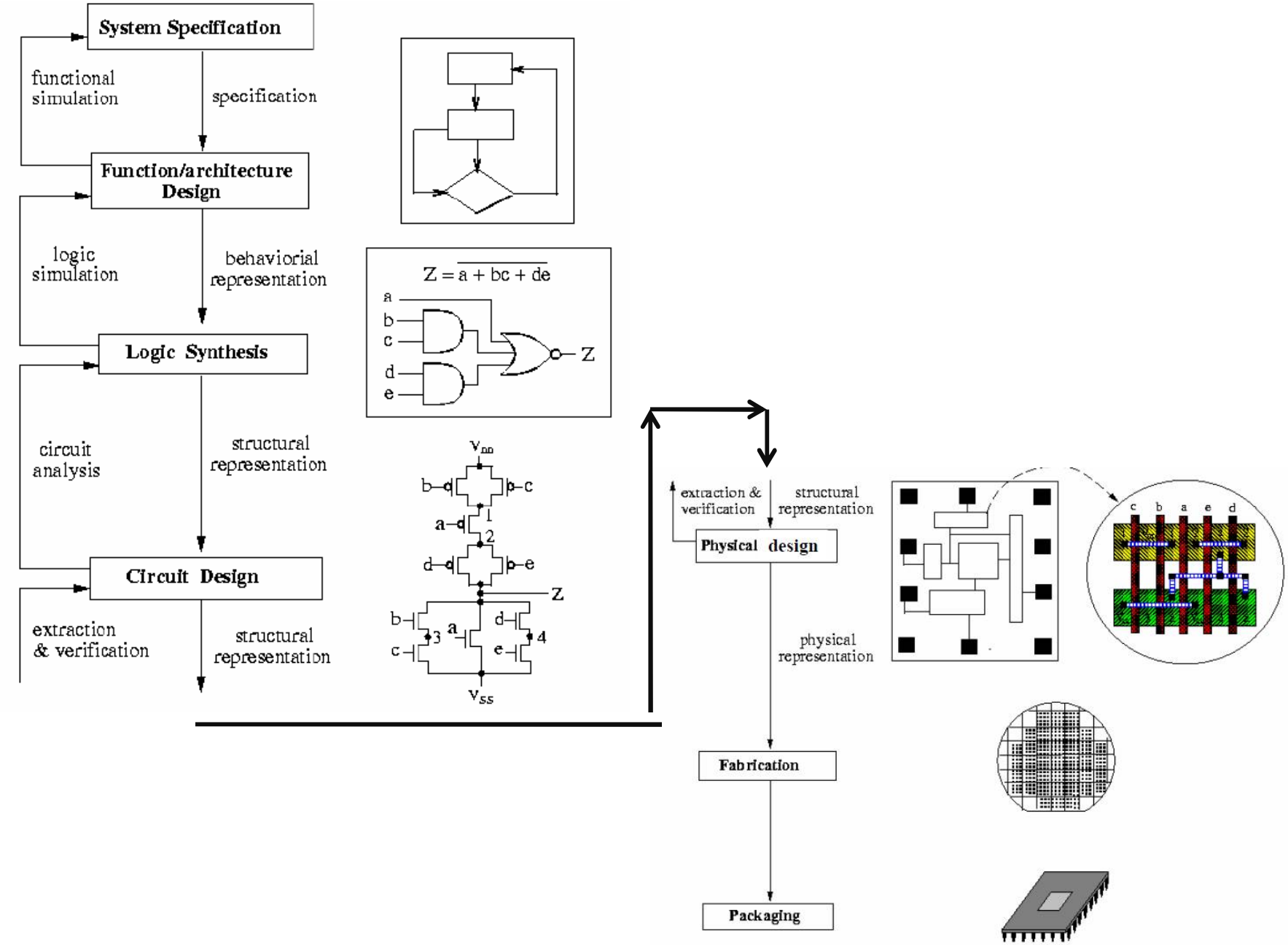


Traditional VLSI Design Cycles

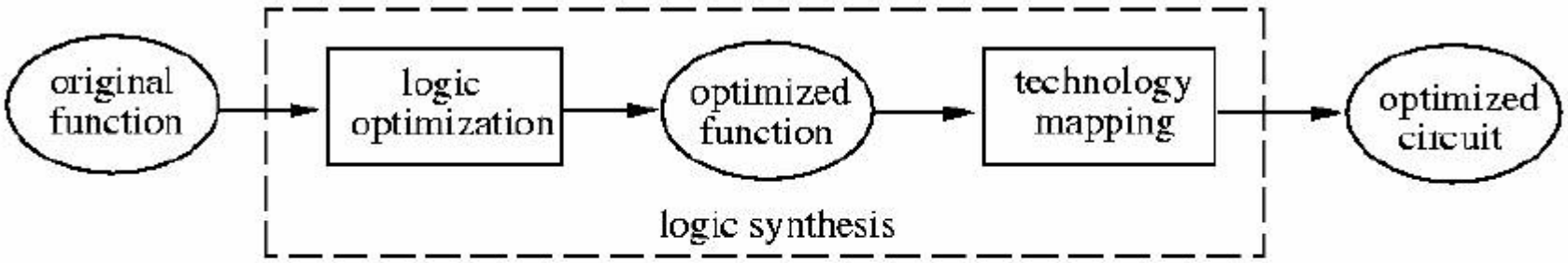
1. System specification
2. Functional design
3. Logic synthesis
4. Circuit design
5. Physical design and verification
6. Fabrication
7. Packaging

- **Synthesis:** increasing information about the design by providing more detail (e.g., logic synthesis, physical synthesis).
- **Analysis:** collecting information on the quality of the design (e.g., timing analysis).
- **Verification:** checking whether a synthesis step has left the specification intact (e.g., layout verification).
- **Optimization:** increasing the quality of the design by rearrangements in a given description (e.g., logic optimizer, timing optimizer).
- **Design Management:** storage of design data, cooperation between tools, design flow, etc. (e.g., database).


Concept
Designer
Behavior Specification
Behavior Synthesis
RTL Design
Logic Synthesis
Final Product
Manufacturing
Layout (Masks)
Layout Synthesis
Netlist (Logic Gates)
Design Validation
RTL Verification
Logic Verification
Product Verification
Layout Verification


System Specification
functional simulation
specification
Function/architecture Design
logic simulation
behaviorial representation
Logic Synthesis
circuit analysis
structural representation
Circuit Design
extraction & verification
structural representation
Z = a + bc + de
a
b
c
d
e
Z
V_{DD}
V_{SS}
1
2
3
4
extraction & verification
structural representation
Physical design
physical representation
Fabrication
Packaging

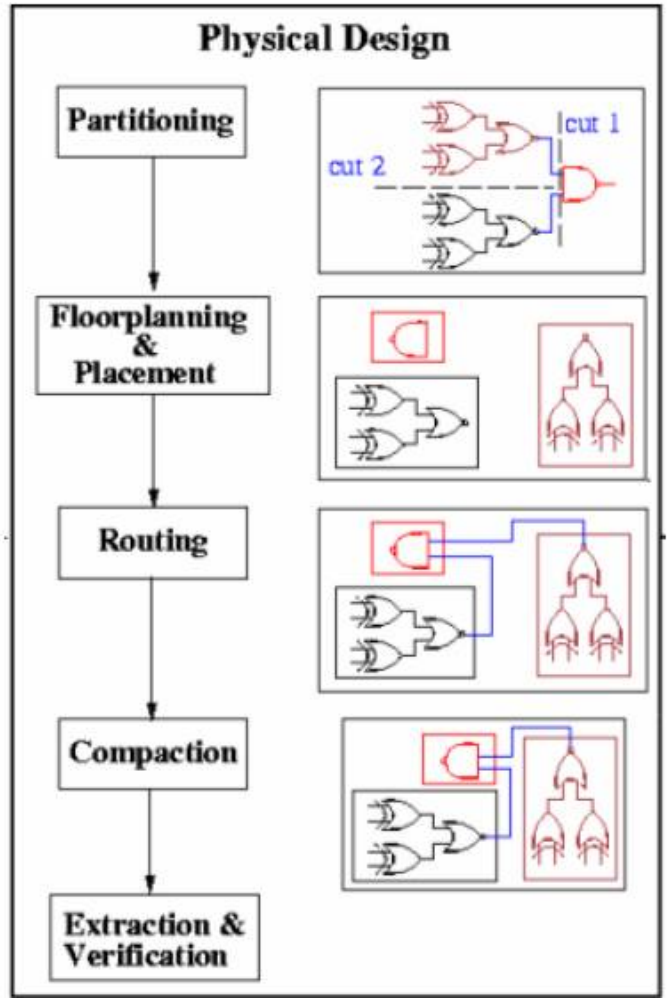
- System-level design
  - Partitioning into hardware and software, co-design, co-simulation, etc.
  - Cost estimation, design-space exploration
- Algorithmic-level design
  - Behavioral descriptions (e.g., in Verilog, VHDL)
  - High-level simulation
- From algorithms to hardware modules
  - High-level (or architectural) synthesis
- Logic design:
  - Schematic entry
  - Register-transfer level and logic synthesis
  - Gate-level simulation (functionality, power, etc.)
  - Timing analysis
  - Formal verification

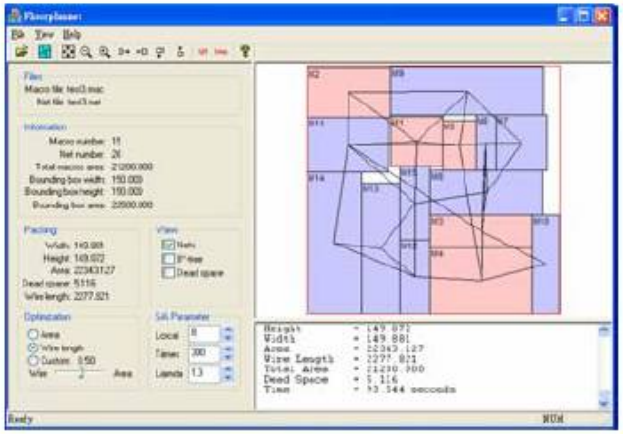


- **Logic synthesis** programs transform Boolean expressions into logic gate networks in a particular library.
- Optimization goals: minimize area, delay, power, etc
- **Technology-independent** optimization: logic optimization
  - Optimizes Boolean expression equivalent.
- **Technology-dependent** optimization: **technology mapping/library binding**
  - Maps Boolean expressions into a particular cell library.

- Transistor-level design
  - Switch-level simulation
  - Circuit simulation
- Physical (layout) design
  - Partitioning
  - Floorplanning and Placement
  - Routing
  - Layout editing and compaction
  - Design-rule checking
  - Layout extraction
- Design management
  - Data bases, frameworks, etc.
- Silicon compilation: *from algorithm to mask patterns*
  - The *idea* is approached more and more, but still far away from a single *push-buttom* operation

Physical Design Flow

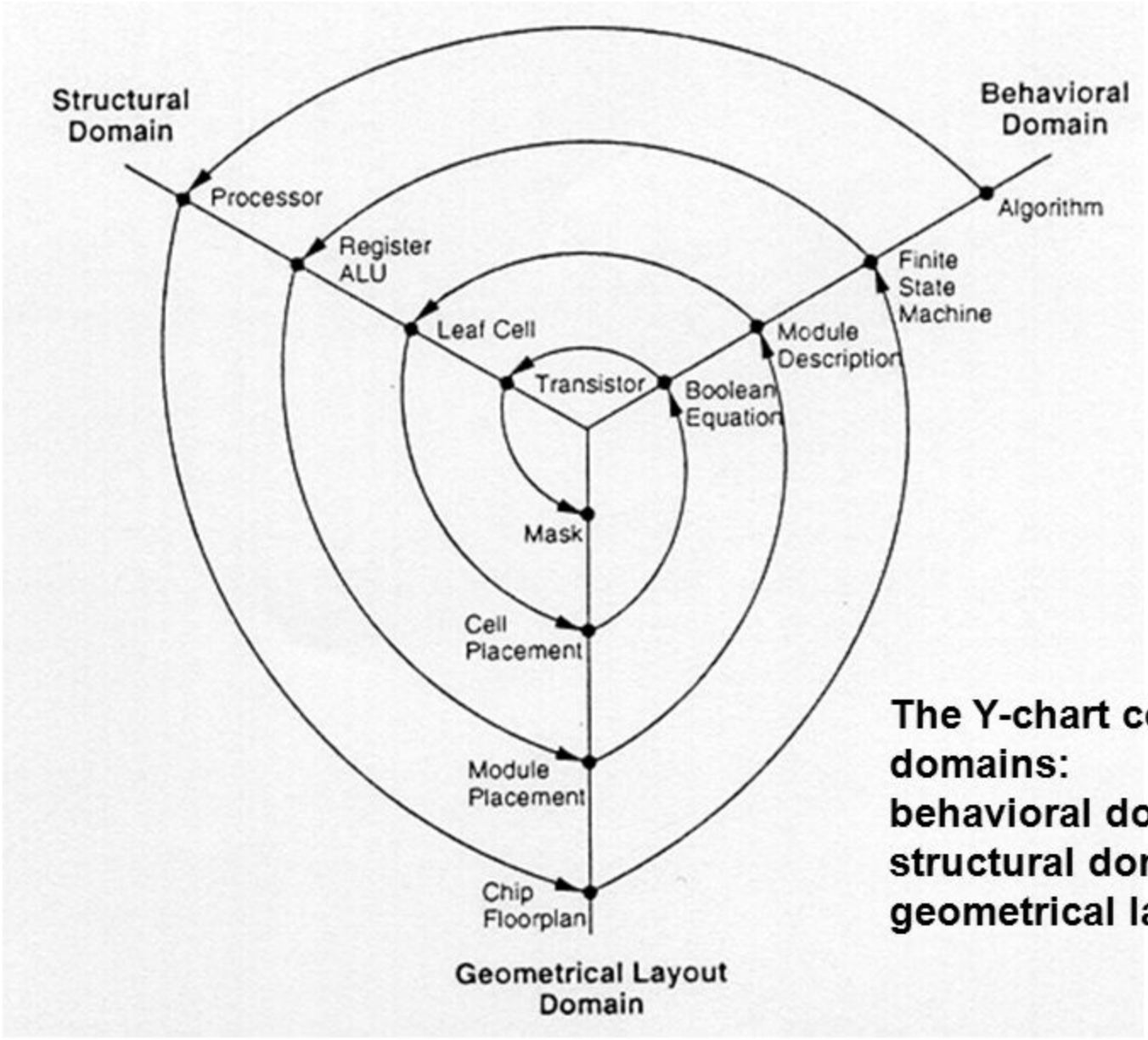




B*-tree based floorplanning system

A routing system

VLSI Design Flow

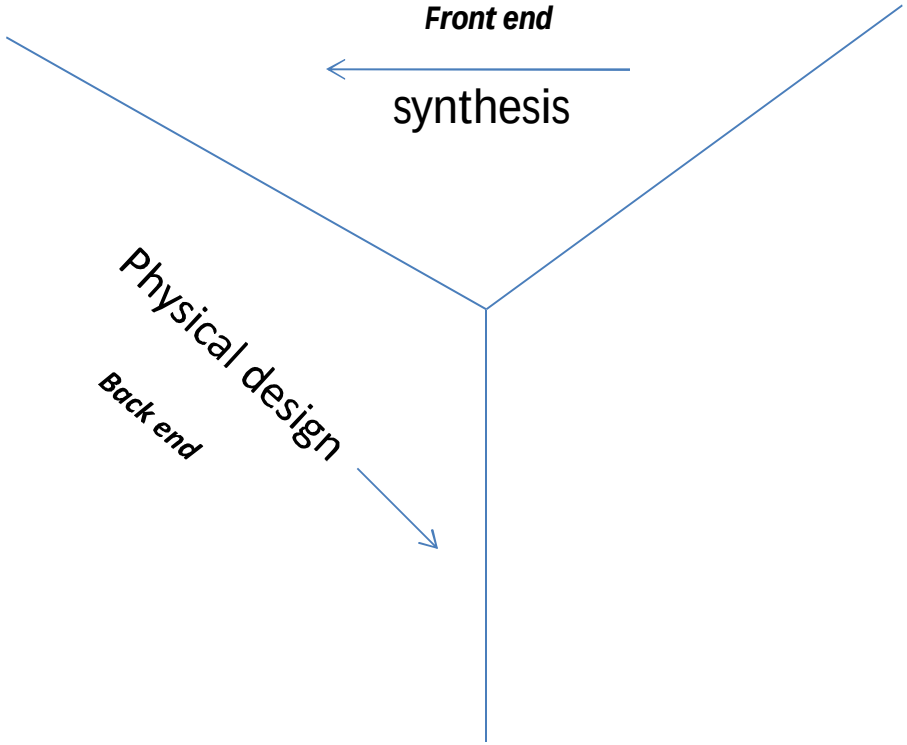


The Y-chart consists of three major domains:
behavioral domain,
structural domain,
geometrical layout domain.

The same design we can specify with respect to its behavior, with respect to its structure where at this level design would consist of a block diagram some cells and their interconnections and finally the implementation level or the physical level.

In terms of the y diagram you can assume that there are several concentric circles as we have shown out here and as we move away from the centre of this y towards outside the design becomes more and more abstract. So when you are at the outer layer of the circles we are at a very abstract level may be behavior structure or physical. But as we go towards the inner side of this y the design becomes more and more detailed.

Structural description all basic building blocks will be some components and the components may be cells from the libraries.



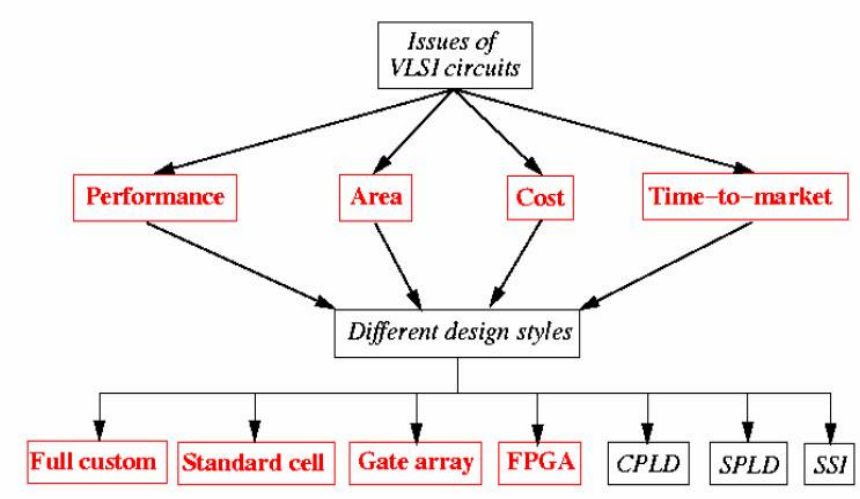
Behavioral level if you recall this specifies the behavior of the circuit or the system without explaining how this behavior has to be implemented or realized.

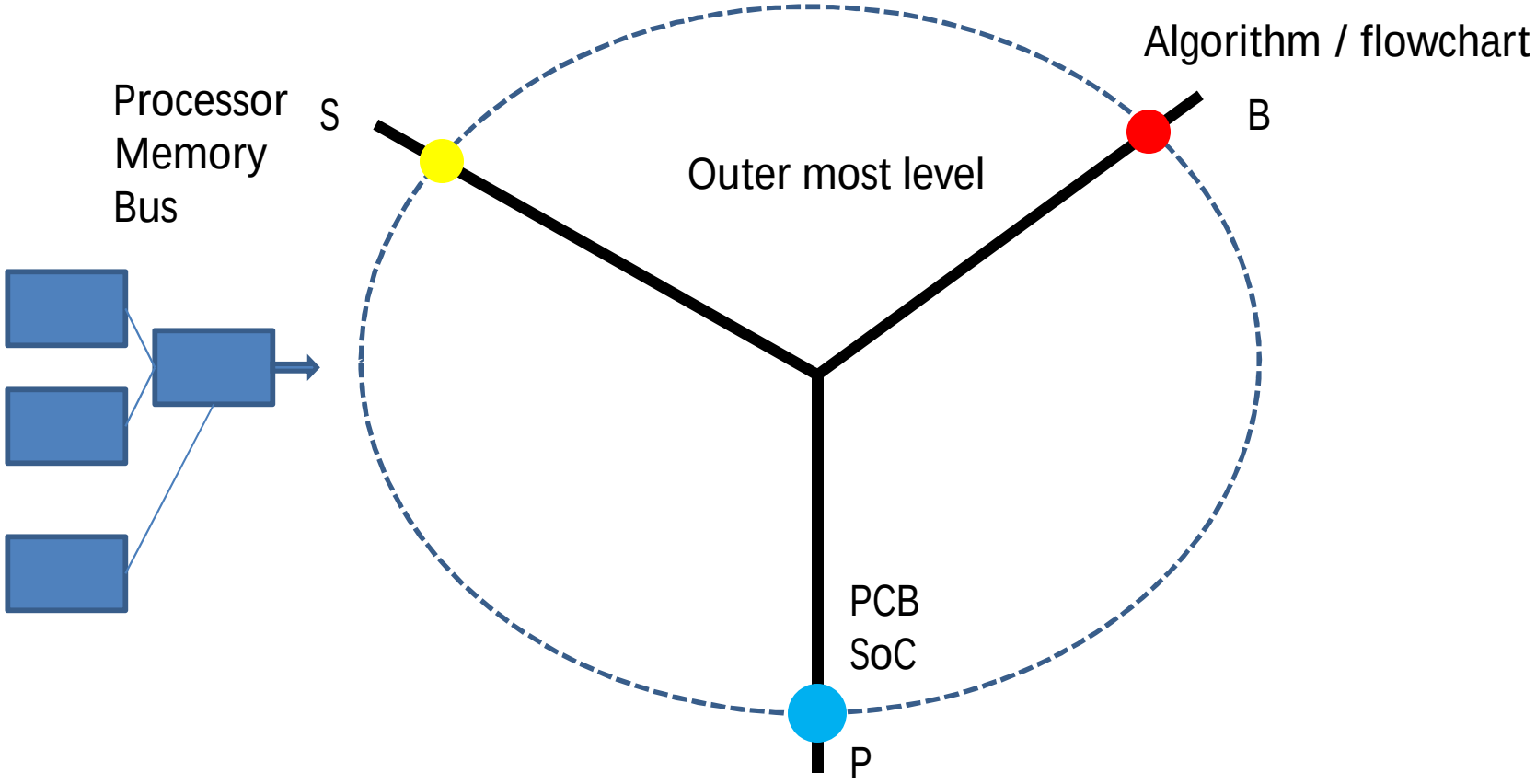
Synthesis is essentially a detailing out of the design

Suppose we have given behavioral description out here, now when we carry out synthesis we translate this behavioral description into the corresponding structural description this is the process of synthesis.

Translation from behavior to structure is also sometimes called front end design. So when we talk about front end design or front end CAD tools we are essentially talking about the translation from behavior to structure.

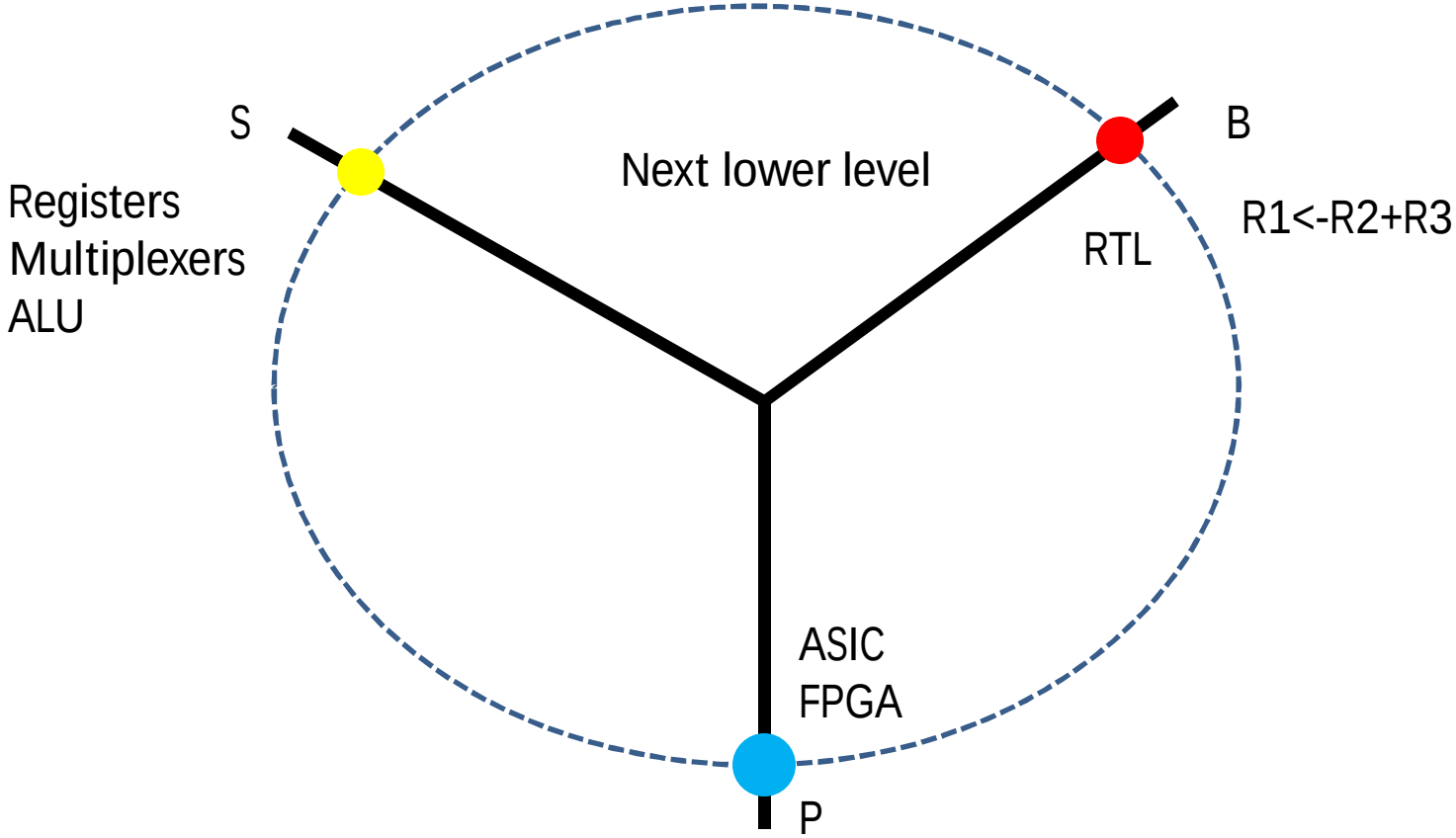
In implementation level the primary requirement is the interconnections of blocks. So the process from structural to physical design transmission is called physical design and sometimes this process of translation is also known as back end design or back end CAD tools





Behavior at the most abstract level a design can be specified by simply specifying the algorithm or if it is a system label design you can also specify the flow chart. So algorithm or flowchart is a very high level kind of a design specification. So when we synthesize a algorithm or flow chart at the corresponding structural level we will have a some kind of a block diagram - some basic building blocks and their interconnections. Now these basic building blocks at the level of algorithm or flow chart will be something like processors, memories, buses, other IO interface units. So at this abstract level we are more concerned with a block diagram kind of a design, how these high level blocks are interconnected rather than going into the details of it.

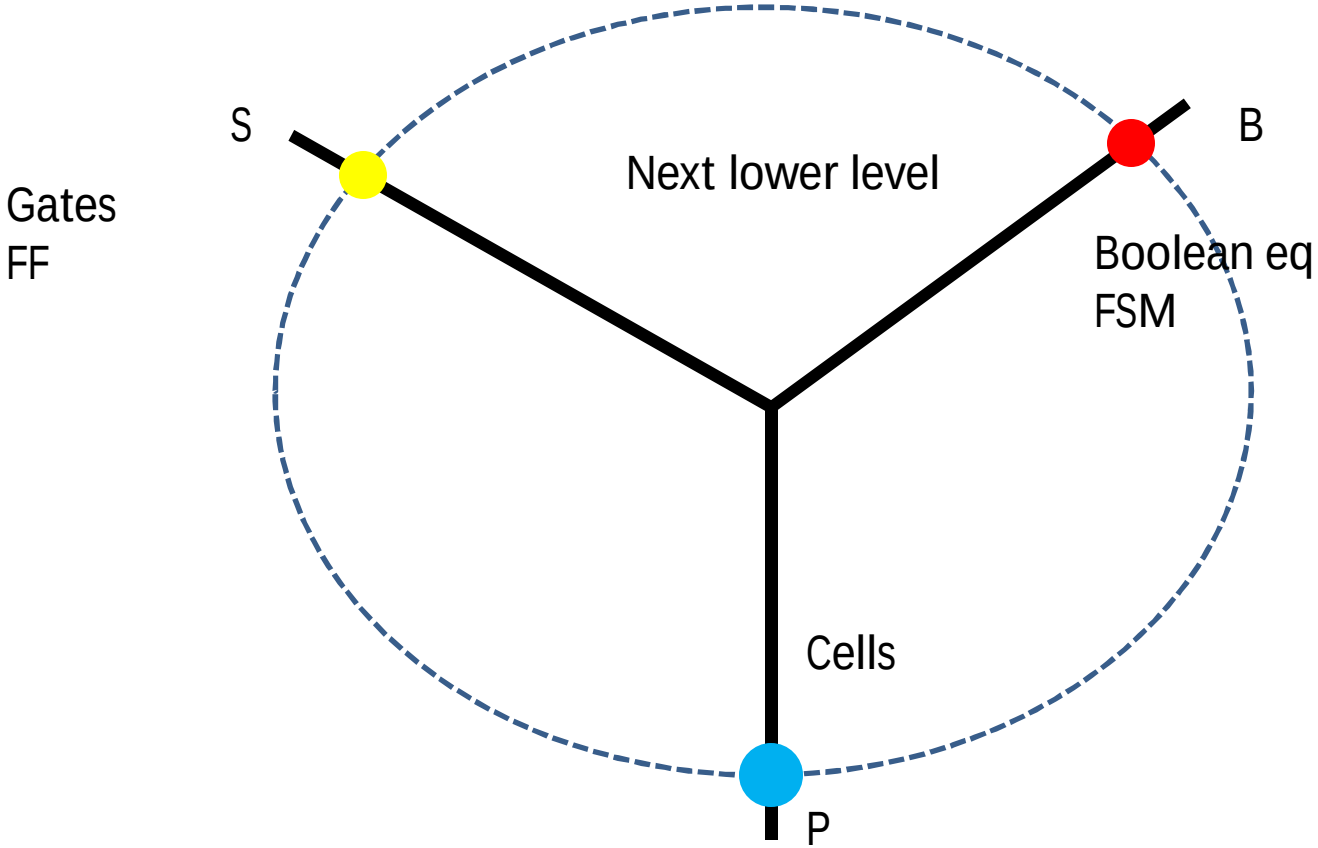
In Physical domain now we are either talking about a printed circuit board where each of these modules or cells will correspond to 1 VLSI chip or we can talk of a multi chip module were inside the same integrated circuit we have several silicon wafers which are embedded on them and they are interconnected. So this PCB and MCM are very similar kind of PCB the chips are put on a board MCM.



Now register transfer level design will look something like this. Suppose R1 assigned R2 plus R3, there can be a number of conditions. If then else kind of thing there can be a number of such register transfer operations you defined based on the inputs you give. So here we specify the behavior. But we do not say that how these register banks are implemented, how these adders, subtractors, are implemented. How many adders are there? So all those details we do not specify.

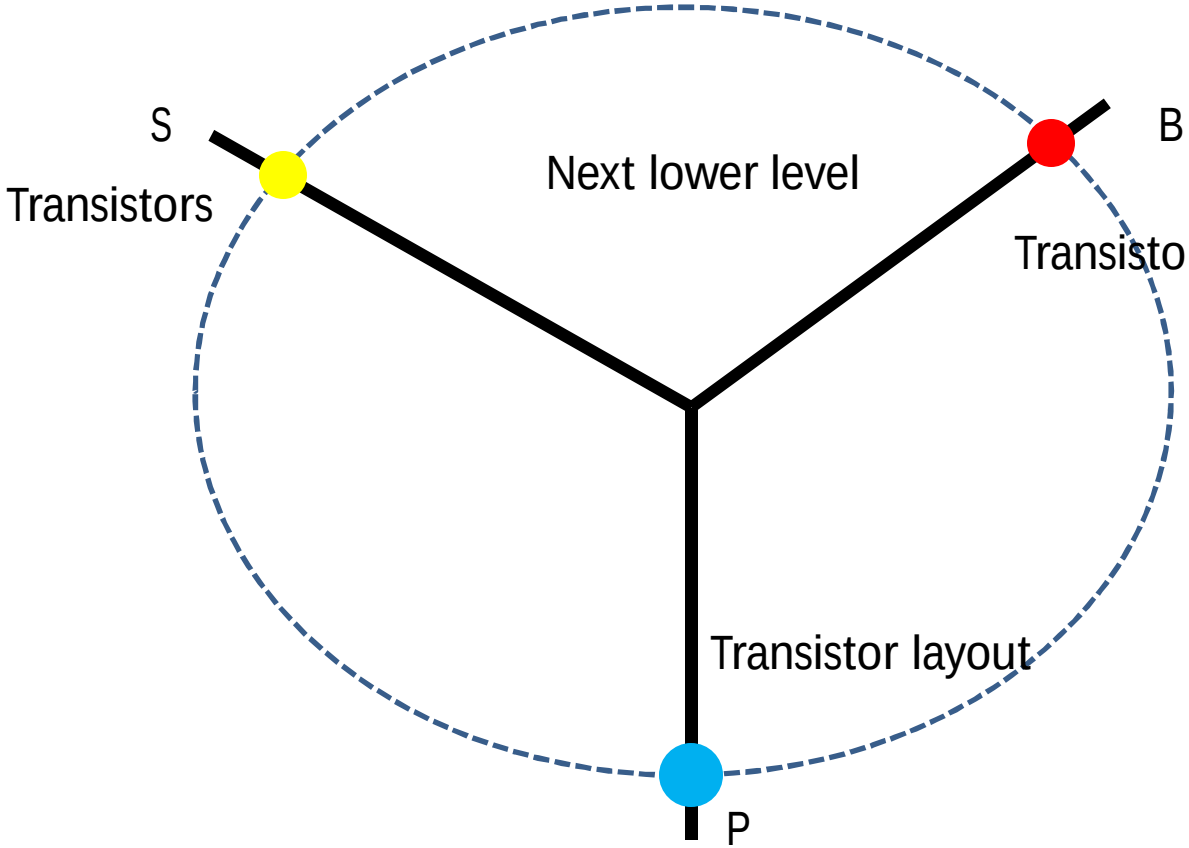
But one can understand with respect to the algorithm description, as we had said earlier this is a much more detailed description. This may correspond to the so called microinstruction level. So at a very high level you can specify a processor with the external IO pins and the instruction set. But as we go into the design of the control unit of the processor, we will have to specify the different microinstructions.

Now this register transfer level behavior when we translate into structure we get a design for the blocks are registers multiplexers ALU's and similar register transfer level components. Similarly when you map them into the physical implementation. So you can either have an application specified circuit or you can have a prototype on a field programmable gate array.



But as you recal the design further from the register transfer level the next natural jump will take in to something like a logical level behavior specification where if it is a combination logic we will specify the behavior by a set of Boolean expressions. If it is a sequential logic we will specify by finite state machines. These represent the behavior of the circuit of the system we try to design. Now as you know starting from the Boolean expression or FSM there are many methods which are available for synthesis where we get a circuit where the basic building blocks now become gates and flip flops, so we have now an interconnection of gates and flip flops starting from the behavior at this level.

In the physical implementation you are working through the back end CAD tool. There is something called a library it is sometimes also called technology library. Now in the library you have the standard components already available - standard components like gates flip flops , arithmetic logic units. So at this level once you have the gate level implementation you simply replace each gate or each flip flop by the corresponding cell available from the library. So now at this level you have actually a tentative layout on silicon where you have the cells, these cells are fairly low level cells gates and flip flops and the way they are interconnected.



And as you go even further down you get a design at the lowest level. Well with respect to the behavior; see ultimately we want to implement a circuit typically in CMOS; using CMOS technologies.

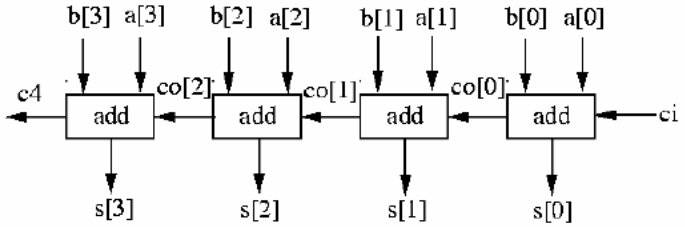
You can also specify the behavior of a CMOS net list with respect to the transistor functions typically for digital circuits a transistor or a MOS transistor is modeled in a very ideal way. We treat this as a simple switch. So each transistor is modeled as a switch this is what we mean by the transistor functions.

Well with respect to some more simplified model, when you go to the structural level. We get the actual net list of the transistors which is used to implement the behavior which we specify out here and finally at the physical level we get the actual layout of the transistors.

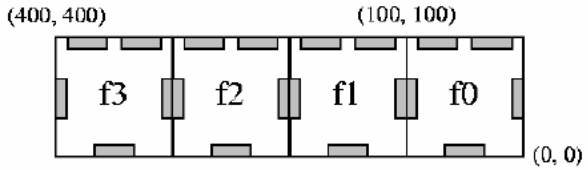
Behavior

```
module add4 (s, c4, ci, a, b);
        input [3:0] a, b;
        input ci;
        output [3:0] s;
        output c4;
        wire [2:0] co;
            add f0 (co[0], s[0], a[0], b[0], ci);
            add f1 (co[1], s[1], a[1], b[1], co[0]);
            add f2 (co[2], s[2], a[2], b[2], co[1]);
            add f3 (c4, s[3], a[3], b[3], co[2]);
endmodule
```

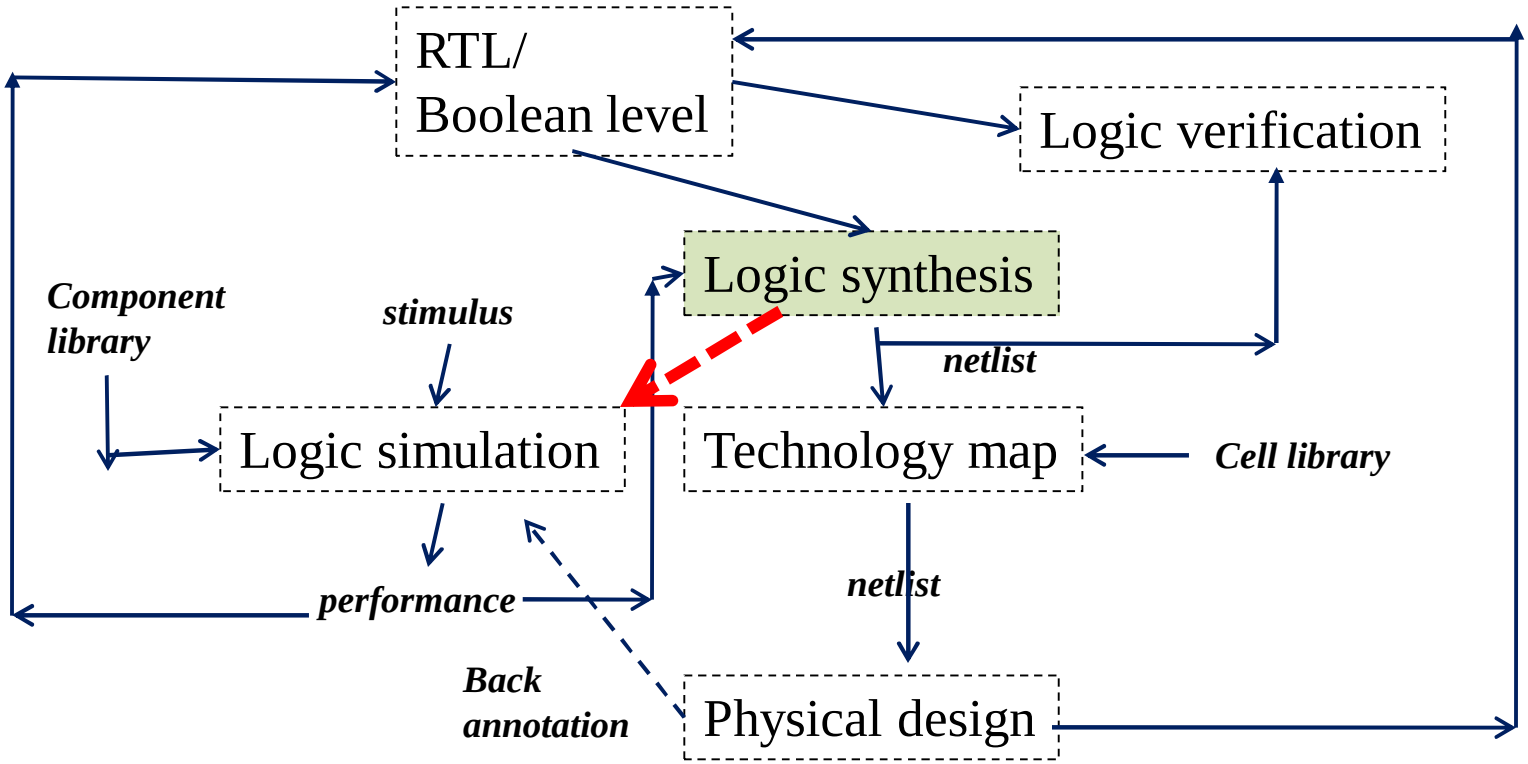
Structural



Physical



Logic design



In synthesis we are talking about translating the behavior into the structural specification.

we start with the level of logic design where the specification is given either in terms of set of Boolean expressions. It can be FSM's also or it can be slightly higher, it can be register transfer level specifications also.

We are specifying the behavior of the circuit and from there we are trying to go through the different steps of synthesis. Now this diagram will show you what are the different steps that we typically go through

logic synthesis is the block which accepts the specification as the input and generates some kind of a synthesized net list as the output net list means some basic components and their interconnection.

Now if the specification is in the terms of Boolean expressions, then this net list will consist of gates possibly flip flops and then interconnections.

But when you are trying to implement it this net list may not be sufficient. Why? See suppose you are trying to design using FPGA. So using FPGA you have some constraints you know that these are the functions I can implement using FPGA and these are the things I cannot.

Simple example

Suppose we are trying to design using CMOS. Now in CMOS again we will be having some kind of a library available with you where you will be getting a list of available gates or cells which you can use in your design. So the net list that we have obtained through the logic synthesis process, the basic cells out there, they will have to be mapped some how into the cells that belongs to that library. So that process is achieved or accomplished through a step called technology mapping

So technology mapping takes the net list which is generated by the logic synthesizer it also accepts the library this is a given cell library

And it tries to map the cells out here to the cells available in the library. And as the output, it generates another net list where the basic cells are those which are available in the library. So this net list is something which can be implemented

So this is the essential process and after completing the physical design if you find that something is wrong the circuit is not working according to the specification, then you may have to redesign and have to go back and change the original specification.

You lost : cost/ time / manpower/ load

So usually what people do before going for the actual physical design, so you take the net list which was obtained by the synthesizer and try to check whether this net list is working as per the specifications or not this you do through a process called **logic simulation.**

So logic simulation will again take this net list as the input. it will also take a stimulus which is just a set of inputs. If I have circuit and want to test the circuit for these inputs. So we apply these inputs and the simulator will tell you what will be the outputs.

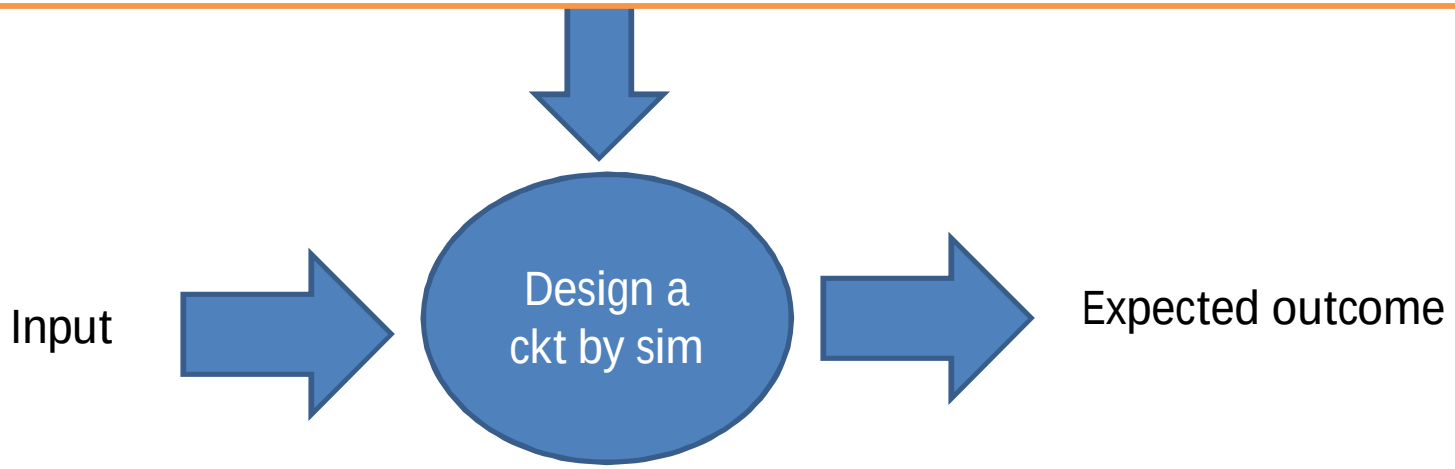
I can verify if the outputs are correct or not. So when you are doing logic simulation the behavior of all the cells of this net list must be known that is available in another library. This is called the component library. Here all the basic components that can be there in this net list are present as well as their behavior is also present which helps in carrying out the logic simulation.

So after simulation we get the performance and if we see the performance is not according to our requirement we again to do redesign go back and change the specs.

Now sometimes even during the physical design we can do something called back annotation. Back annotation means form of very low level design specification we can re extract possibly the gate level design again. So we can go back to the logic simulation phase and again simulate with the same stimulus and check the performance.

logic simulation process takes as input a logic level net list. Logic level net list means net list comparison of gates/ comparison of flip flops and it simulates the functional behavior. we would like to see whether the input output specifications are correct that if you give a particular input to the circuit what is the expected response or what is the response that we get

Simulation tool is a software program where the circuit you want to stimulate in terms of suitable data structure and through a systematic scan of the circuit of the data structure.



The simulated circuit is a close model of the netlist
Obtained from synthesis
This is an unknown system I can say which on table
Top I have to test

Component library : when you are doing simulation you need the behavior of the components behavior of the components what does that mean? Suppose if the components are simple gates like AND, OR, NOT, then what you need is the input output behavior of the gates. You need either the Boolean function or the truth table in some form. So you need the behavior of the gate in order to evaluate the output of a gate when some particular input values are applied to it okay. So behavior of the components is assumed to be present in the so called component library. Now component library can be quite exhaustive you can have gates out there. You can have sequential elements like flip flops and registers you can have slightly more combinational more complex blocks like multiplexers, adders and other functions and models like decoders

one of the requirements of logic simulation is that it should be possible to handle large circuits. So when we talk of large circuits in the present day scenario, we are actually talking about millions of gates okay. So the logic simulation process should be very efficient it should be able to handle large circuits of the order of millions.

Objectives of simulation:

1. Functional correctness -> test bench
2. Timing analysis
3. Test generations

Test Generation : you have a design and you have done some kind of synthesis on it. So what we obtained is a net list . But this is not something which you have already fabricated. This net list is still on your computer . So through simulation you try to verify whether the net list is working as per the specification . This is done on the design.

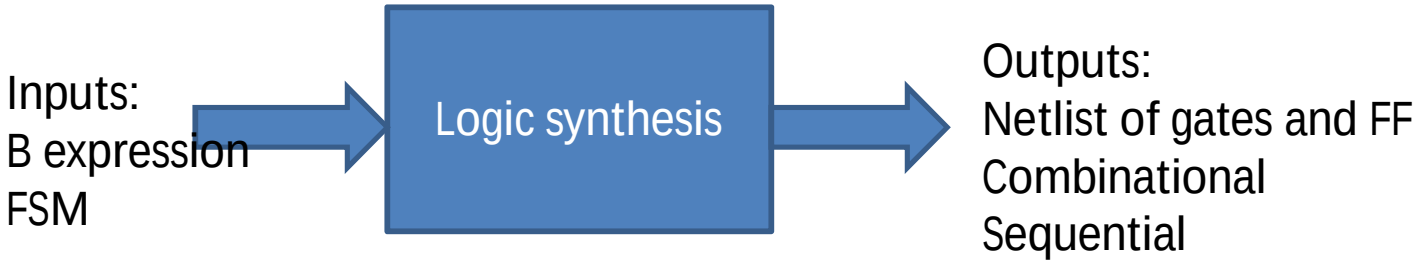
But suppose the design process is what you have completed the physical design process also and now the chips are getting fabricated. Suppose you have fabricated 1000 chips for your design. Now during fabrication there can be a large number of faults which can creep in obviously. So once you have manufactured the circuits in terms of the chips you will have to test each and every chip by applying some input stimulus input test vector and checking the output. This is the so called manufacture test

Design for manufacturability

Finding out yield

Response model

Logic synthesis



Design Goal :
1. Number of level (delay)
2. Number of gates (area)
3. Low power

Constraint
Target library

Suppose you are trying to design a circuit for a very critical application where delay is most important to you. You want to design a circuit that will work as fast as possible. So here if it is combination circuit your designed goal will be to minimize the number of gate levels. So as you reduce the number of levels

to minimize signal active means as the switching activity in the circuits get reduced the total power consumption also get reduced. So we want to ensure that the number of signal transitions that are going on signal transition means 1 to 0, 0 to 1 they are going on per unit time in the circuit is as less as possible.

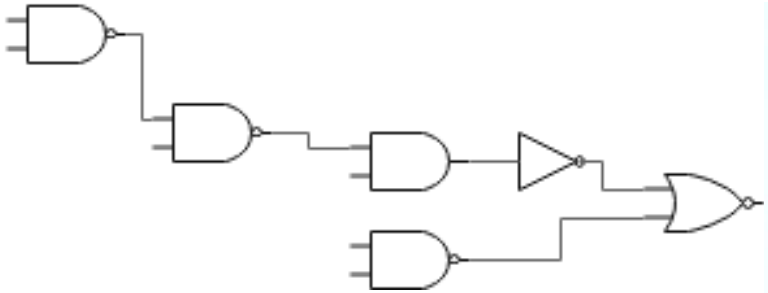
Circuit breakup

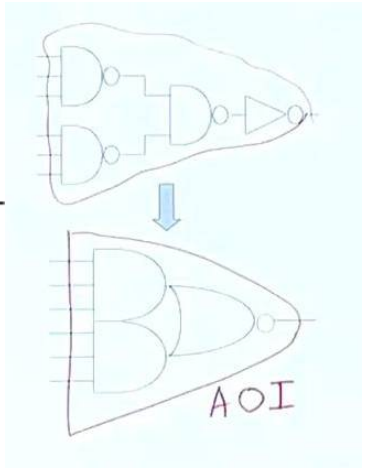
if you have a gate with large number of inputs.

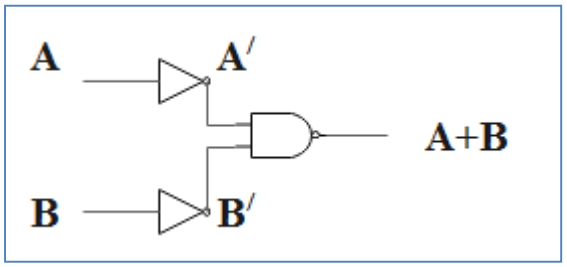
You have a AND gate with a large number of inputs. So instead of having 1 AND gate with a large number of inputs you may be possibly break it up into several smaller AND gates. But you may argue that if I do this I am increasing the number of levels. So apparently I am also increasing the delay because the number of gate delays is more. But well this is not true. ------ Assignment

Technology Mapping

Technology mapping is a process where we have the net list with us. We have a technology library with us we will have to map our net list from some modules which are available in the technology library. So the basic concept is that during synthesis we will have to map portions of the net list to so called cells which are available in the cell library. cells can be these standard gates NAND, NOR, NOT, AND, OR, invert or depending on the target where you want to map the design on if it is FPGA.



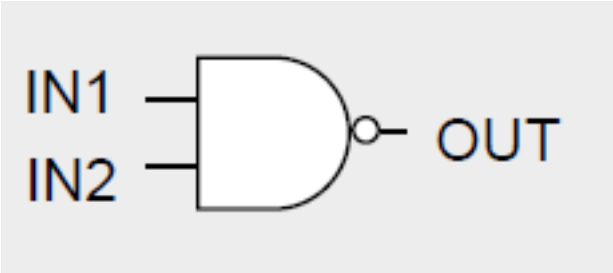


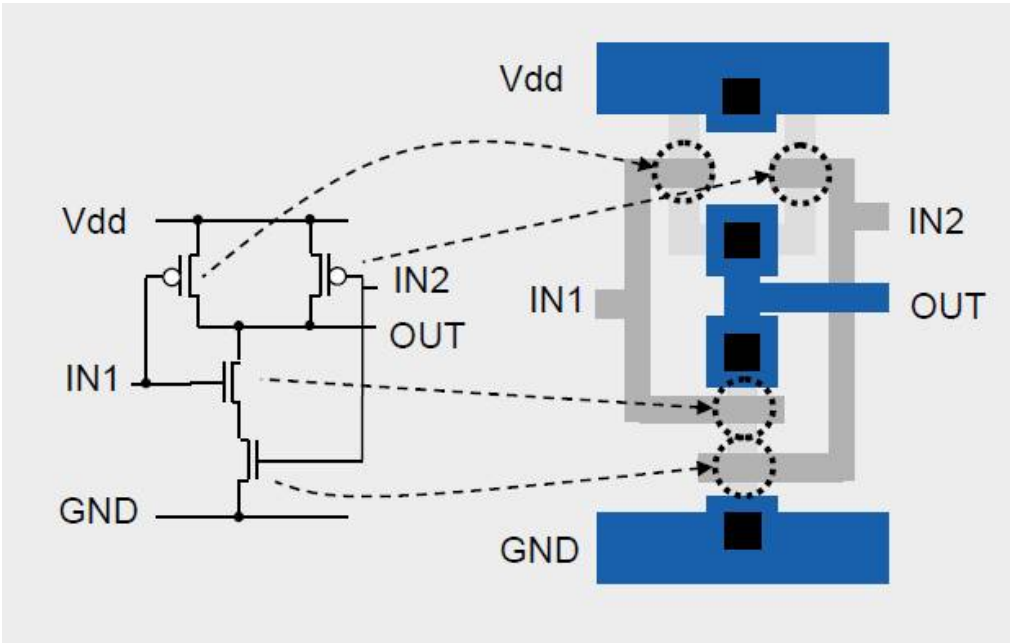


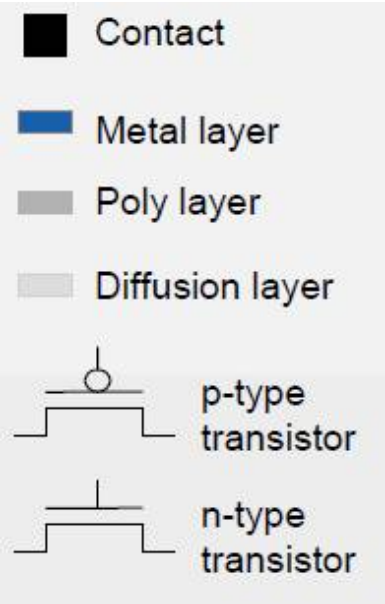
This is the standard Component library

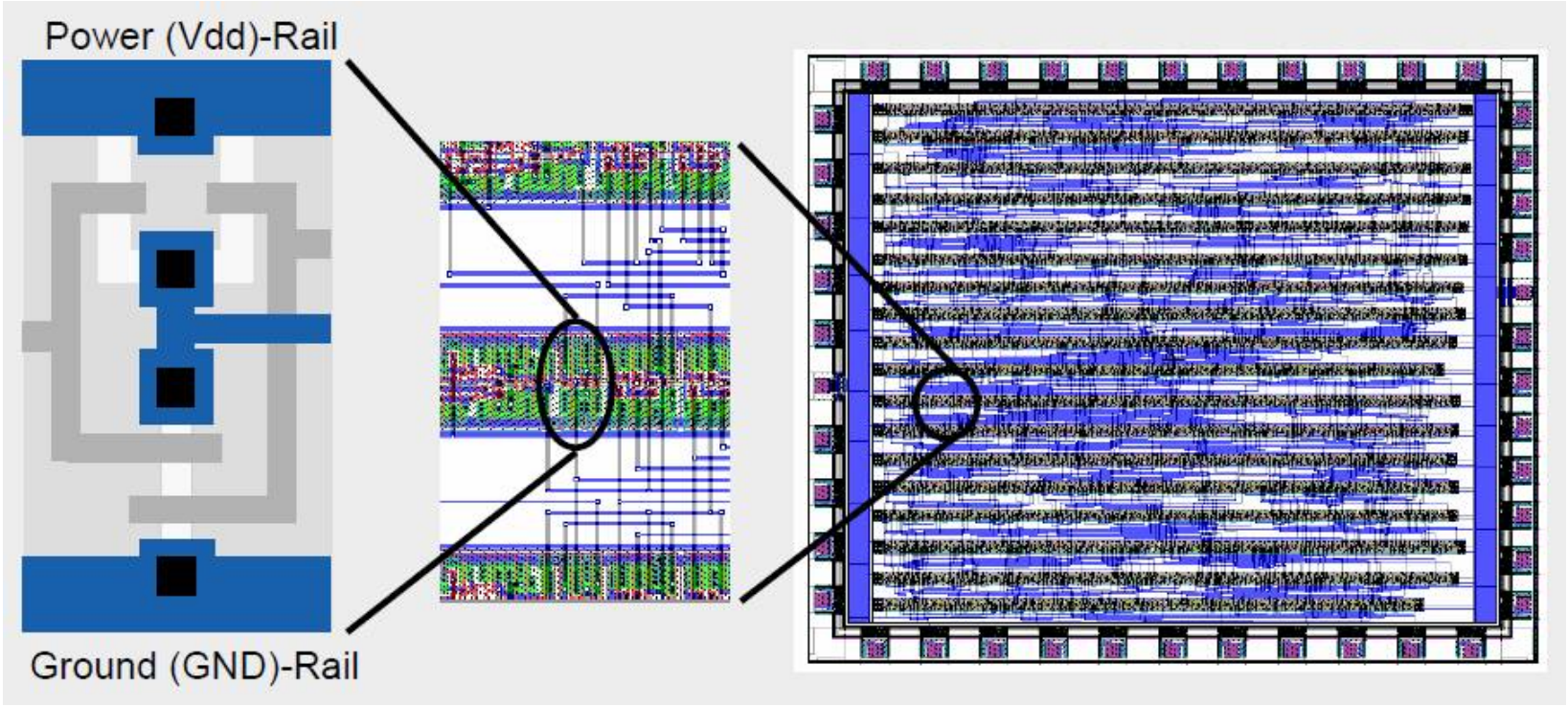
So the idea is simple you have a big net list of gates you try to find out the small sub circuits from there which matches with some net list available in your technology library. And if you find a match replace it by the corresponding component of the technology library. So if you do this what you have is that your entire net list which was there at the beginning will now get transformed into a net list of cells which have been picked up from the technology library. Now as I had said there can be overlap among the cells, the objective is to have minimum number of cells if area is important or to reduce the number of levels of the cell net list if delay is more important.

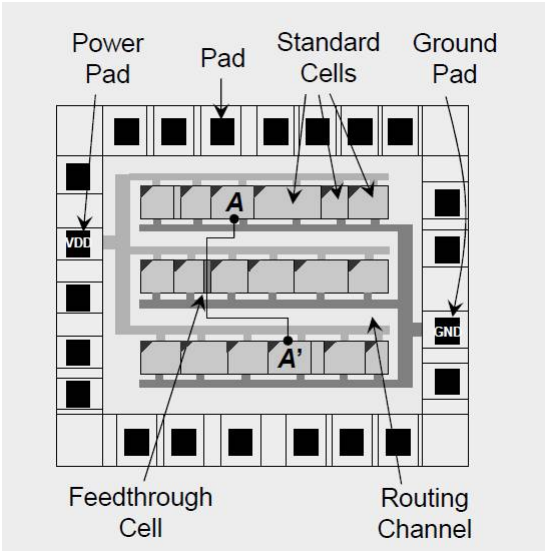
A pictorial overview of design flow








Power (Vdd)-Rail
Ground (GND)-Rail


Power Pad
Pad
Standard Cells
Ground Pad
A
A'
VDD
GND
Feedthrough Cell
Routing Channel

What is Simulation

Definition

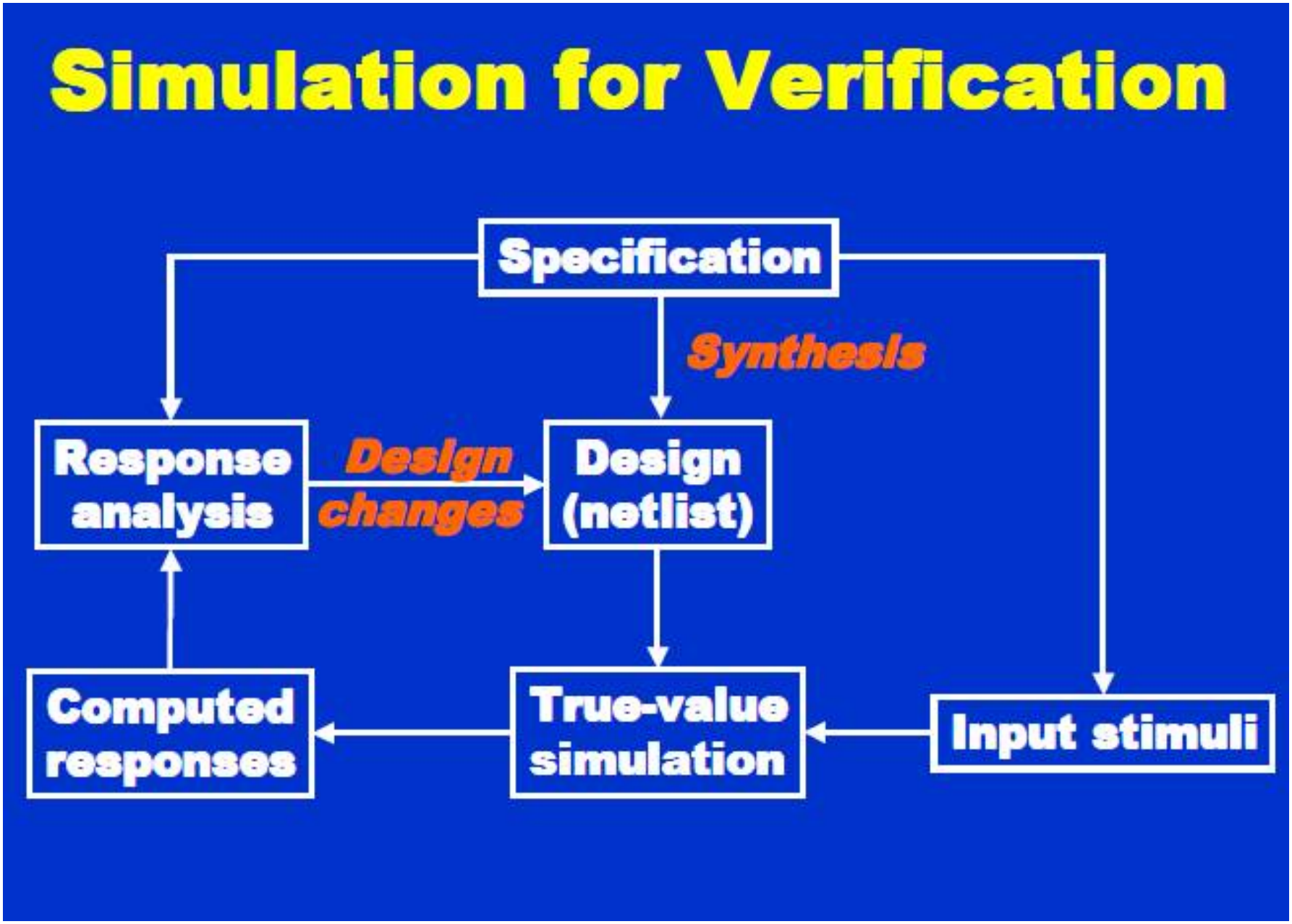
- A *simulation* is the imitation of the operation of real-world process or system over time.
  - Generation of artificial history and observation of that observation history
- A *model* construct a conceptual framework that describes a system
- The behavior of a system that evolves over time is studied by developing a simulation *model*.
- The model takes a set of expressed assumptions:
  - Mathematical, logical
  - Symbolic relationship between the *entities*

Goal of modeling and simulation

- A model can be used to investigate a wide verity of "what if" questions about real-world system.
  - Potential changes to the system can be simulated and predicate their impact on the system.
  - Find adequate parameters before implementation
- So simulation can be used as
  - Analysis tool for predicating the effect of changes
  - Design tool to predicate the performance of new system
- It is better to do simulation before Implementation.

- **Simulation is used for design verification:**
  - **Validate assumptions**
  - **Verify logic**
  - **Verify performance (timing)**
- **Types of simulation:**
  - **Logic or switch level**
  - **Timing**
  - **Circuit**
  - **Fault**

Simulation for Verification



How a model can be developed?

- Mathematical Methods
  - Probability theory, algebraic method ,…
  - Their results are accurate
  - They have a few Number of parameters
  - It is impossible for complex systems
- Numerical computer-based simulation
  - It is simple
  - It is useful for complex system

When Simulation Is the Appropriate Tool

- Simulation enable the study of internal interaction of a subsystem with complex system
- Informational, organizational and environmental changes can be simulated and find their effects
- A simulation model help us to gain knowledge about improvement of system
- Finding important input parameters with changing simulation inputs
- Simulation can be used with new design and policies before implementation
- Simulating different capabilities for a machine can help determine the requirement
- Simulation models designed for training make learning possible without the cost disruption
- A plan can be visualized with animated simulation

When Simulation Is Not Appropriate

- When the problem can be solved by common sense.
- When the problem can be solved analytically.
- If it is easier to perform direct experiments.
- If cost exceed savings.
- If resource or time are not available.
- If system behavior is too complex.
  - Like human behavior

Systems and System Environment

- A *system* is defined as a groups of objects that are joined together in some regular interaction toward the accomplishment of some purpose.
  - An automobile factory: Machines, components parts and workers operate jointly along assembly line
- A system is often affected by changes occurring outside the system: system environment.
  - Factory : Arrival orders
    - Effect of supply on demand : relationship between factory output and arrival (activity of system)
  - Banks : arrival of customers

1. ***Isolated System:*** It is a system that has no interactions beyond its boundary layer. Many controlled laboratory experiments are this type of system.

2. ***Closed System:*** It is a system that transfers energy, but not matter, across its boundary to the surrounding environment. Our planet is often viewed as a closed system.

3. ***Open System:*** It is a system that transfers both matter and energy can cross its boundary to the surrounding environment. Most ecosystems are example of open systems.

4. ***Morphological System:*** This is a system where we understand the relationships between elements and their attributes in a vague sense based only on measured features or correlations. In other words, we understand the form or morphology a system has based on the connections between its elements. We do not understand exactly how the processes work to transfer energy and/or matter through the connections between the elements.

5. ***Cascading System:*** This is a system where we are primarily interested in the flow of energy and/or matter from one element to another and understand the processes that cause this movement. In a cascading system, we do not fully understand quantitative relationships that exist between elements related to the transfer of energy and/or matter.

System State Variable

- System state variables are the collection of all information needed to define what is happening within the system to a sufficient level.
- Having defined system state variables, a contrast can be made between discrete-event models and continuous models based on the variables needed to track the system state.
- The system state variables in a discrete-event model remain constant over intervals of time and change value only at certain well defined points called event times.
- Continuous models have system state variables defined by differential or difference equations giving rise to variables that may change continuously over time.

Components of system

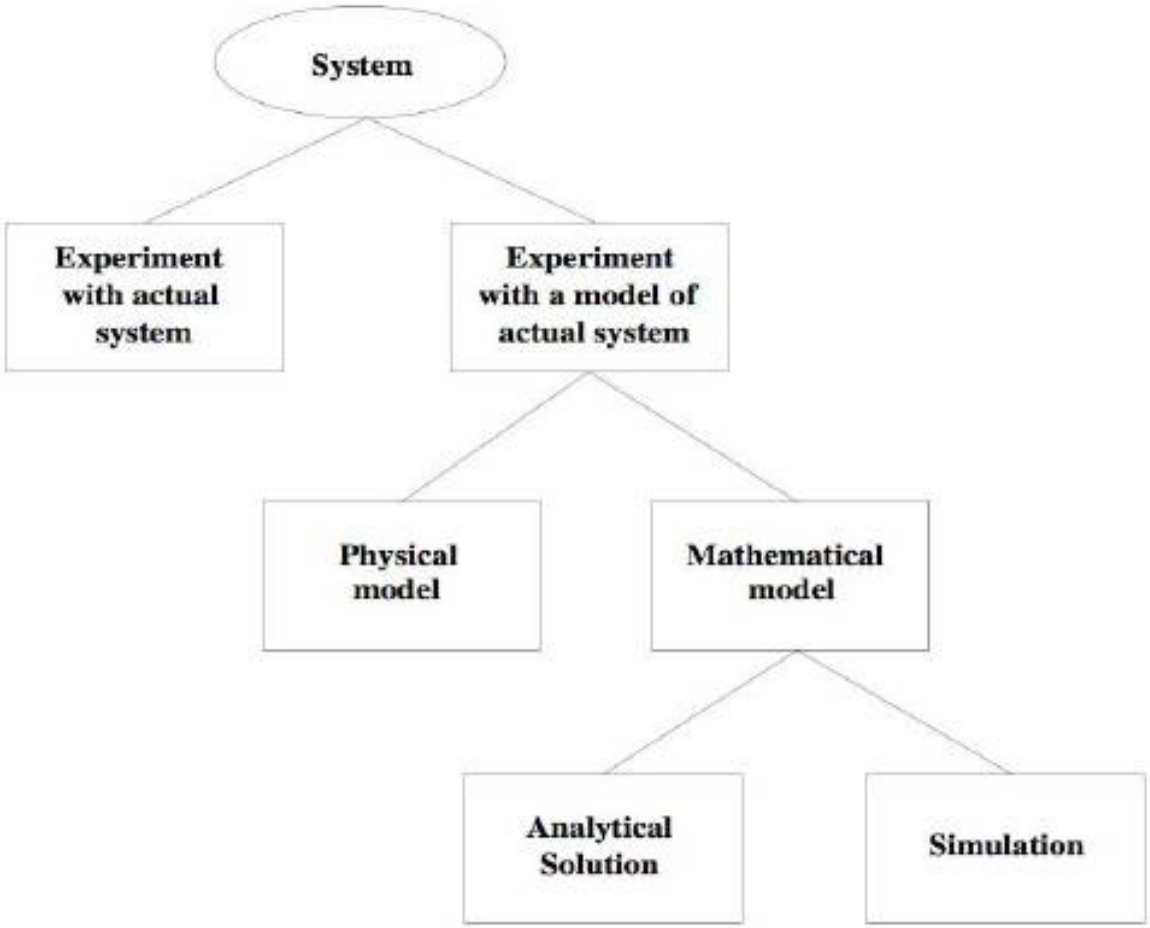
- Entity
  - An object of interest in the system : Machines in factory
- Attribute
  - The property of an entity : speed, capacity
- Activity
  - A time period of specified length :welding, stamping
- State
  - A collection of variables that describe the system in any time : status of machine (busy, idle, down,…)
- Event
  - A instantaneous occurrence that might change the state of the system: breakdown
- Endogenous
  - Activities and events occurring with the system
- Exogenous
  - Activities and events occurring with the environment

Entities and Attributes

- An entity represents an object that requires explicit definition.
- An entity can be dynamic in that it "moves" through the system, or it can be static in that it serves other entities.
- Attributes should be considered as local values.

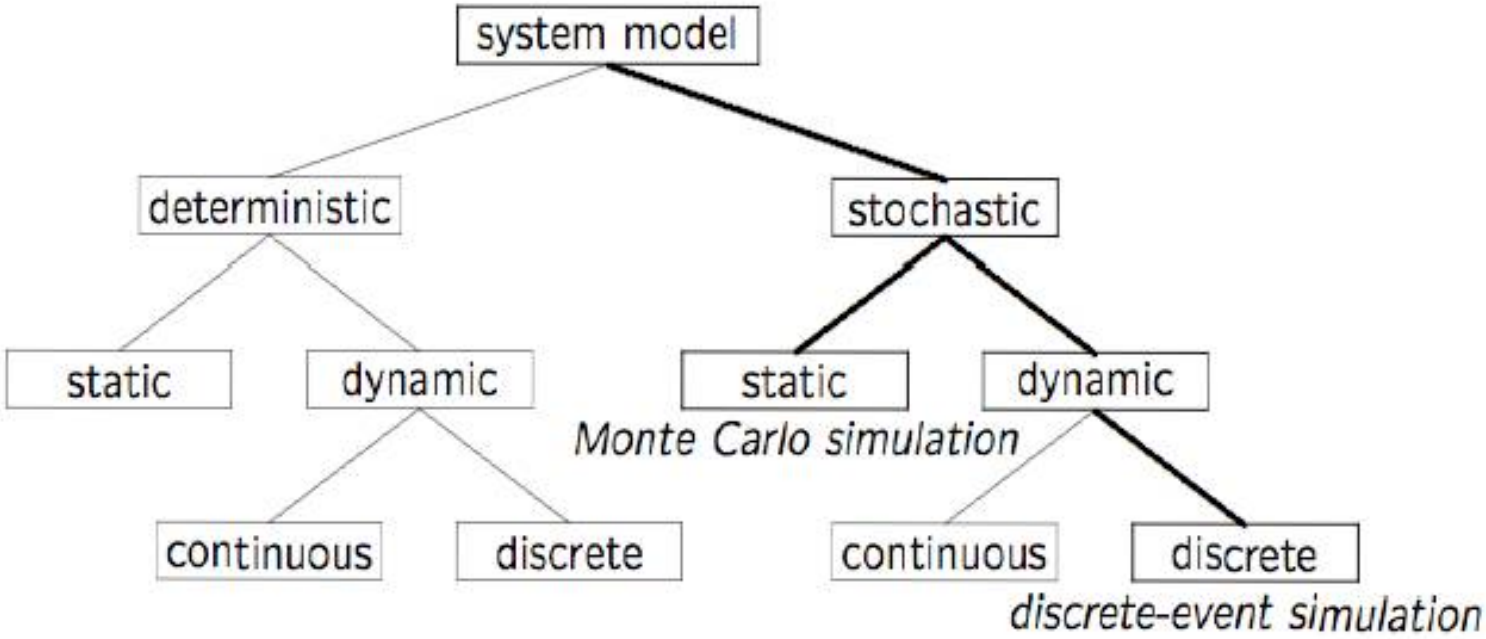
Activities and Delays

- An activity is duration of time whose duration is known prior to commencement of the activity.
- The duration can be a constant, a random value from a statistical distribution, the result of an equation, input from a file, or computed based on the event state.
- A delay is an indefinite duration that is caused by some combination of system conditions.
- When an entity joins a queue for a resource, the time that it will remain in the queue may be unknown initially since that time may depend on other events that may occur.


System
Experiment with actual system
Experiment with a model of actual system
Physical model
Mathematical model
Analytical Solution
Simulation

Characterizing a Simulation Model

- Deterministic or Stochastic
  - Does the model contain stochastic components?
  - Randomness is easy to add to a DES
- Static or Dynamic
  - Is time a significant variable?
- Continuous or Discrete
  - Does the system state evolve continuously or only at discrete points in time?
  - Continuous: classical mechanics
  - Discrete: queuing, inventory, machine shop models



Deterministic: Randomness does not affect the behaviour of the system. The output of the system is not a random variable.
Stochastic: Randomness affects the behaviour of the system. The output of the system is a random variable.

Static: A simulation of a system at one specific time, or a simulation in which time is not a relevant parameter for example, Monte Carlo & steady-state simulations.
Dynamic: A simulation representing a system evolving over time for examples, the majority of simulation problems.

Verification vs. Validation

- *Verification*
  - Computational model should be consistent with specification model
  - Did we build the model right?
- *Validation*
  - Computational model should be consistent with the system being analyzed
  - Did we build the right model?
  - Can an expert distinguish simulation output from system output?

Phases in Simulation

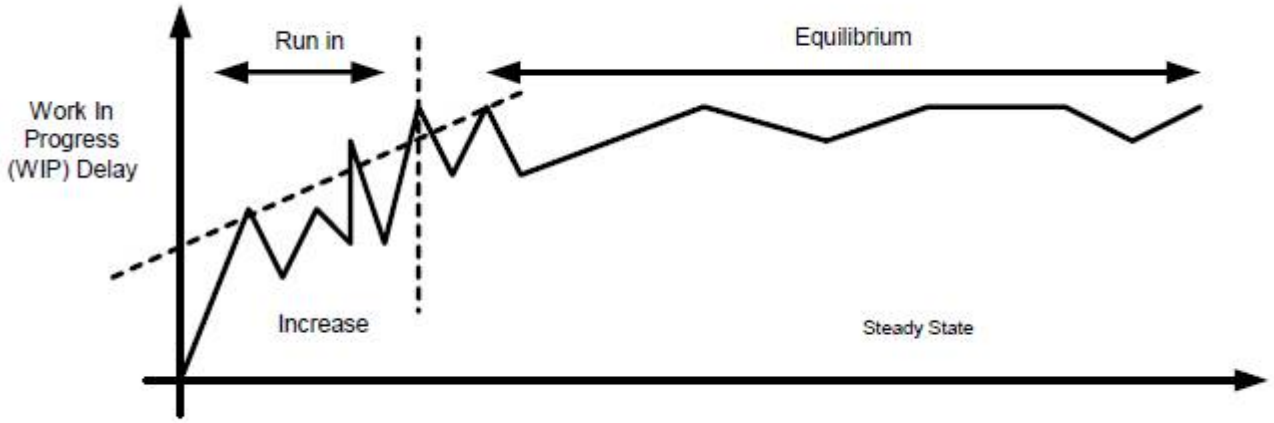


Figure 1. The two key phases of a simulation run

- A simulation run typically starts in the empty and idle state. The run is therefore characterized by a "run-in" phase followed by a "steady state" phase, see Figure 1. The run-in phase is generally ignored and is only used for investigating the effects of transient conditions such as starting up a new factory or performing radical changes within an existing facility.
- Typically the steady state phase is of greater interest. At this stage checks must be made to ensure no long term trends exist, such as continual build up of stock in the factory, that suggest the model (hence the real system) will be unstable and unworkable.

Verification and Validation

- The validation efforts can be grouped into two parts

1. validation of the abstract model itself
2. validation of its implementation

- The first part consists of examining all assumptions, which transform the real world system into the conceptual model.
- Testing the validity of an implementation is a more objective and easier task. It consists of checking the logic, the flowchart, and the computer program to ensure that the model has been correctly implemented.

Modeling Levels

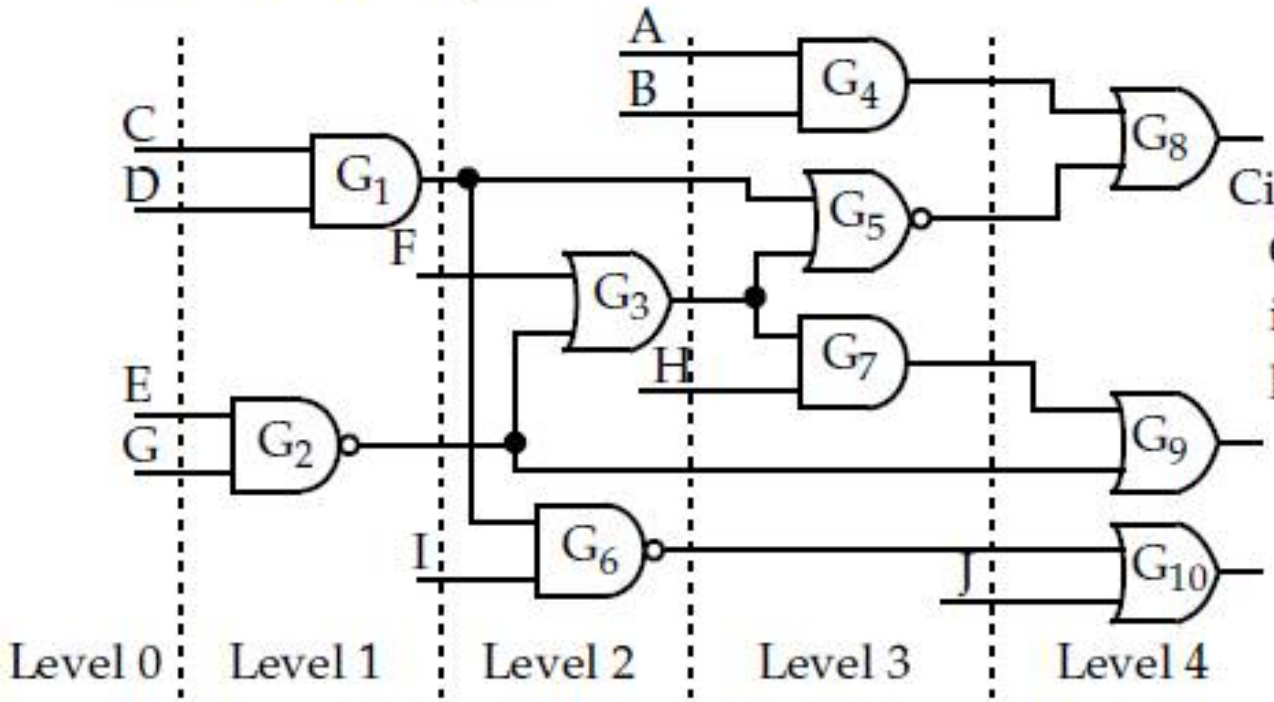
Modeling level	Circuit description	Signal values	Timing	Application
Function, behavior, RTL	Programming language-like HDL	0, 1	Clock boundary	Architectural and functional verification
Logic	Connectivity of Boolean gates, flip-flops and transistors	0, 1, X and Z	Zero-delay unit-delay, multiple-delay	Logic verification and test
Switch	Transistor size and connectivity, node capacitances	0, 1 and X	Zero-delay	Logic verification
Timing	Transistor technology data, connectivity, node capacitances	Analog voltage	Fine-grain timing	Timing verification
Circuit	Tech. Data, active/ passive component connectivity	Analog voltage, current	Continuous time	Digital timing and analog circuit verification

True-Value Simulation Algorithms

- **Compiled-code simulation**
  - Applicable to zero-delay combinational logic
  - Also used for cycle-accurate synchronous sequential circuits for logic verification
  - Efficient for highly active circuits, but inefficient for low-activity circuits
  - High-level (e.g., C language) models can be used
- **Event-driven simulation**
  - Only gates or modules with input events are evaluated (*event means a signal change*)
  - Delays can be accurately simulated for timing verification
  - Efficient for low-activity circuits
  - Can be extended for fault simulation

Compiled Simulation

Levelization example:



Circuit is levelized. Gates at a level have inputs only from lower levels

Signals treated as variables, gates translated to opcodes for AND, OR, etc.

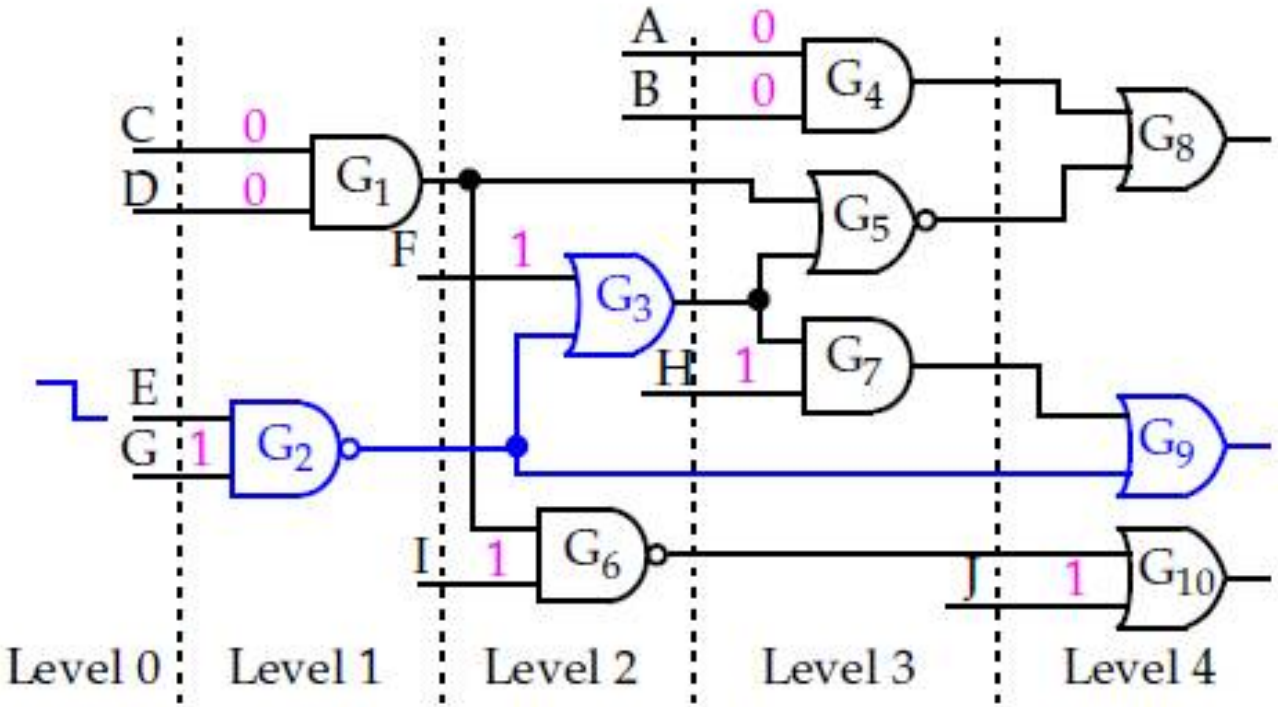
For each vector, code is repeatedly executed until steady state is achieved (handles feedback).

Event-Driven Simulation

Good match for discrete-event simulation.

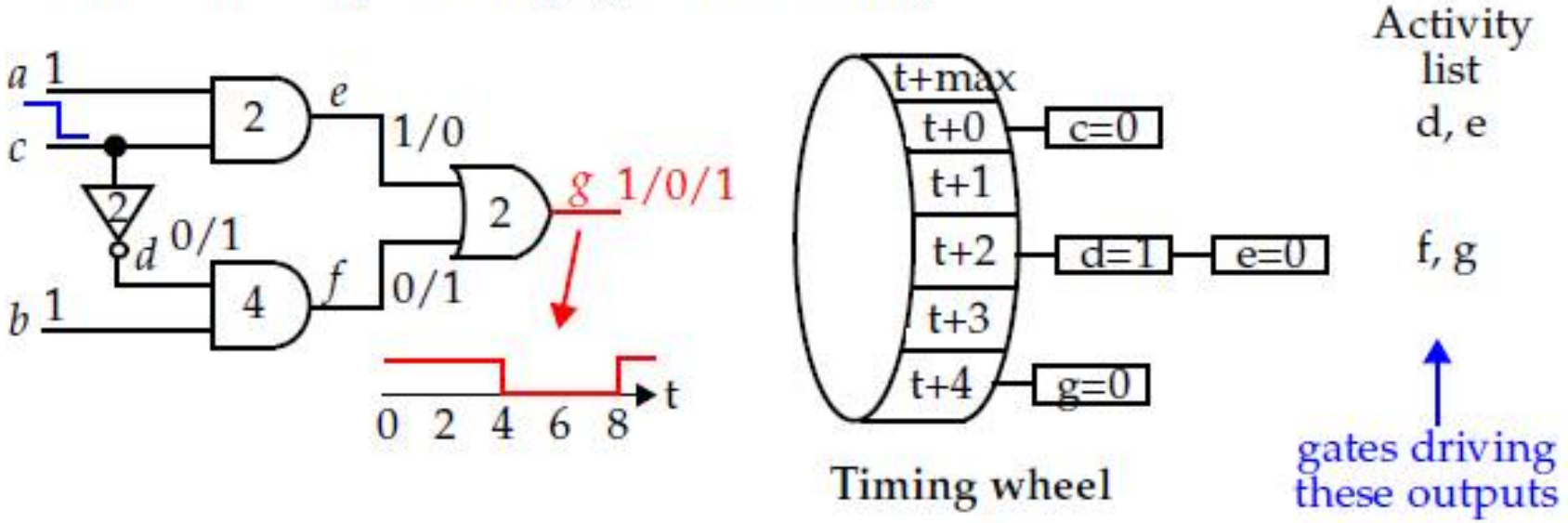
Events (changes in signal values) cause new events (future changes in other signal values).

Evaluation takes place only if an input to a gate changes.



Event-Driven Simulation

Gates whose inputs change go on the *activity list*.



Simulation involves evaluating a gate on the activity list.

If output changes, then gates at fanout added to *activity list*.

Adv:

Computationally efficient.

Ability to simulate arbitrary delays via *event scheduling*.

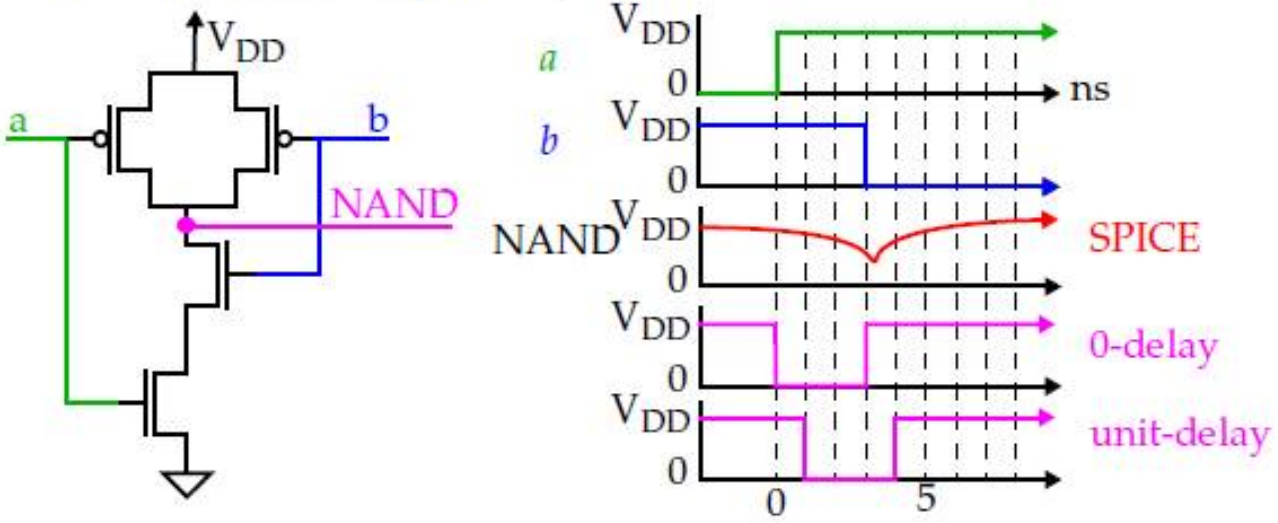
Event scheduler responsible for distributing events to the appropriate list.

Timing Analysis

Timing

Signals experience two types of delays

- **Inertial delay**: Time interval between an input change and output change of a gate.
- **Propagation delay** (transport delay): Time interval between output change and arrival at the input of a gate.



Unit delay: All gates have one unit of delay.

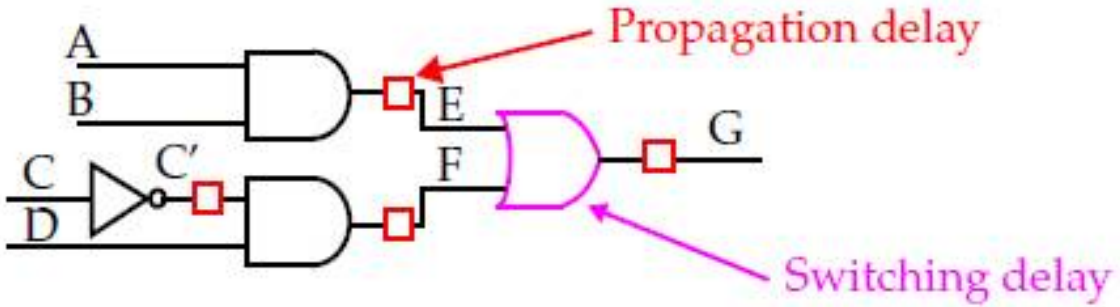
This allows circuits with feedback to be simulated since the proper sequencing of signals is maintained.

Timing

Transmission lines: Interconnect gives rise to delay, i.e., gate output does not instantaneously change "driven" gate inputs.

Propagation delay can be implemented at gate inputs to allow separate modeling of delay at each fanout branch.

Propagation delay can also be modeled by treating the entire fanout net as a circuit element, similar to the treatment of gates.



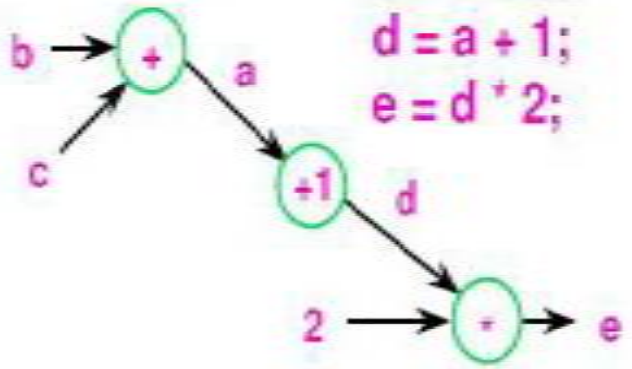
Separate rise and fall propagation delays can be modeled yielding up to 8 different delay conditions for a 2-input gate.

Micro-instructions

- Automating resource allocation

RTL implies

```
a = b + c;
d = a + 1;
e = d * 2;
```



Behavioral implies

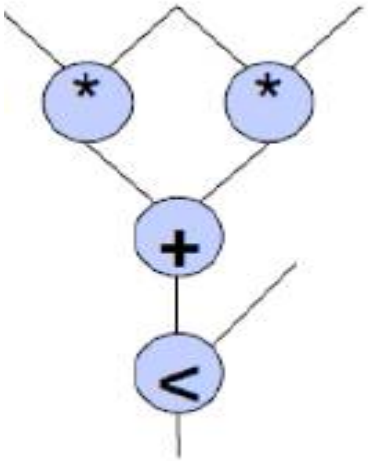
```
e = 2 * (b + c + 1)
```

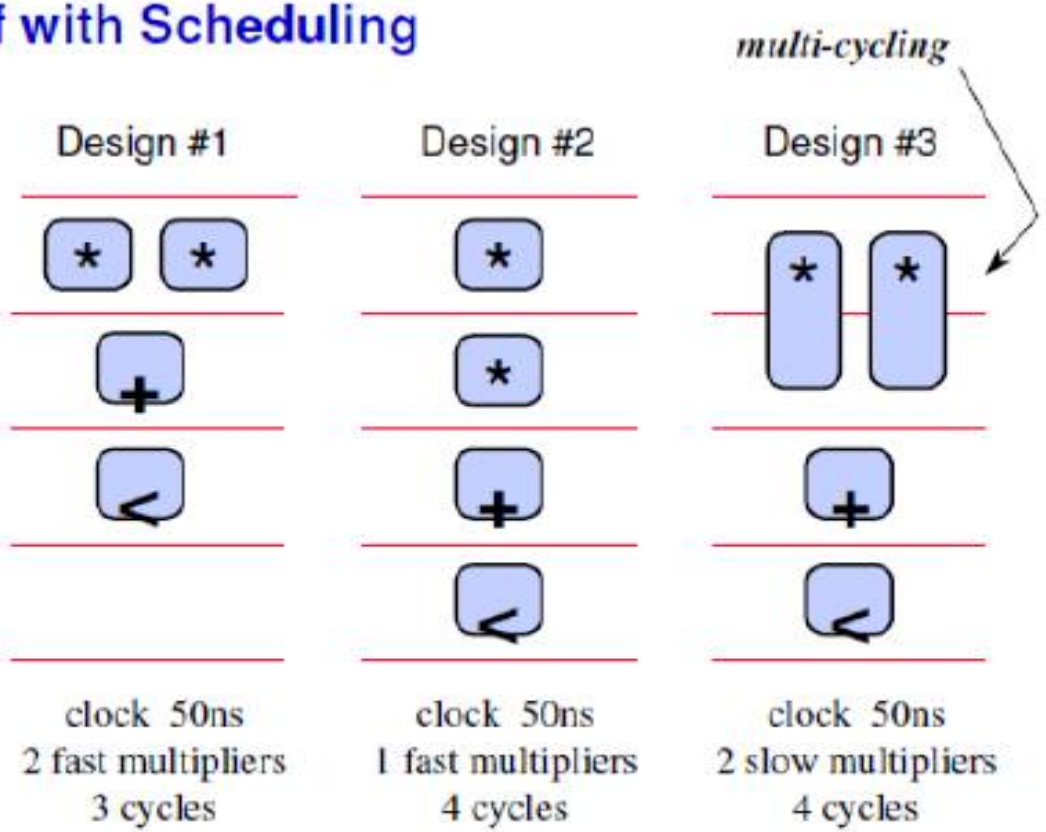
Two Steps:

1. Translation from RTL description into gates
2. Optimization of logic

Scheduling

Design Tradeoff with Scheduling

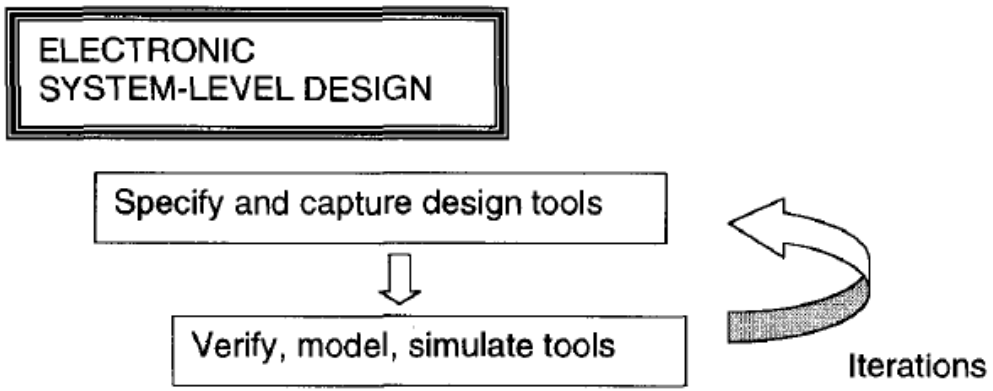


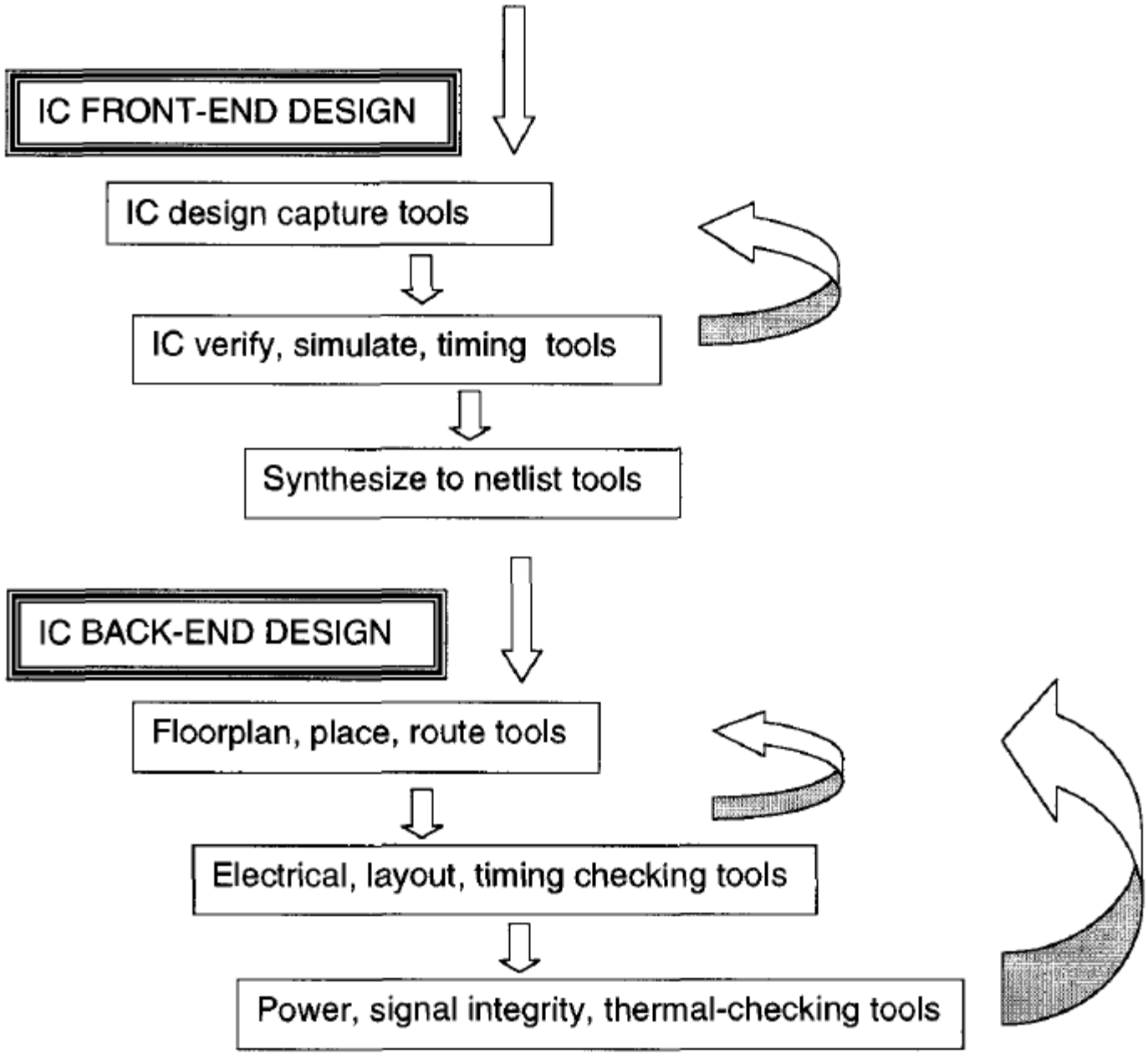


Classification of EDA

There are so many EDA tools. Is there a grouping or order to them?

Note the three main groups: *electronic system level (ESL)* design, IC *front-end (FE)*, and IC *back-end (BE)* design. Each consists of design tasks, followed by verification or checking steps. If the checking steps find errors, the design is revised. The designers repeat (iterate) this design-verify loop as needed. The iteration may even go back to the front-end or system design requirements if back-end errors are found.




IC FRONT-END DESIGN
IC design capture tools
IC verify, simulate, timing tools
Synthesize to netlist tools
IC BACK-END DESIGN
Floorplan, place, route tools
Electrical, layout, timing checking tools
Power, signal integrity, thermal-checking tools

ELECTRONIC SYSTEM-LEVEL (ESL) DESIGN

Users	Tools
Architect	**ESL Design Entry**
System Engineer	**ESL Verification**
IC System Engineer	**ESL Modeling**
	ESL Timing Analysis

ESL design uses one class of tools. Just as the house architect describes the customers' requirements, so does the electronic product engineer.

The architect builds a model (on paper or on the computer) to see how the design will look. The system engineer also makes a high-level computer *model* to see if the design will work.

The architect may try several models, exploring different materials or house layouts. The system engineer does *design exploration*, trying different design approaches (such as implementing functions in hardware or software).

FRONT-END (FE) DESIGN

REGISTER LEVEL

Users	Tools
	RTL Entry
System Engineer	Test Bench
Logic Designer	RTL Simulation
ASIC Designer	Formal Verification
Test Engineer	Design for Test
	Timing Design
	Thermal Design
	Power Design
	Signal Integrity Design
	Synthesis to Gates

Front-end IC design tools include *design capture*, *verification,* and *synthesis* tools. A house architect records design ideas on paper or on a computer. IC designers similarly capture their design ideas on a computer, using words or graphical symbols (design capture).

The next task is to *verify* or test that the design ideas look right or work correctly. For this, architects and engineers used to build detailed miniature models (*prototypes*) of their designs.

IC *design verification* includes simulation tools to verify that the design behaves and performs as intended. Additional *analysis* and *checking* tools ensure that the design meets all the *design rules*. (Design rules are like the electrical and plumbing codes which cities require for safe buildings.)

The third major front-end task is *synthesis*. An architect transforms the house design into a list of specific parts, materials, and building instructions. IC synthesis transforms the design description into a specific set of components and wire connections. A *synthesis* program selects the transistors, gates, or cells from a library.

BACK-END (BE) DESIGN

Users	Tools
	Floorplanning
	Layout—Place & Route
Layout Designer	**Electrical Rules Check**
Logic Designer	**Physical Rules Check**
ASIC Designer	**Extractors & Delay Calculators**
Test Engineer	**Timing Analysis**
	Power Analysis
	Thermal Analysis
	Other Analyses

The *back-end* design tools are collectively known as *physical design* tools. They aid with the general layout of the chip *(floorplanning).* They also help the detailed *placement* (locations) of the transistors, cells, or IP blocks on the chip.

After placement, *routing* tools *(routers)* help plan the physical routing of the interconnect paths.