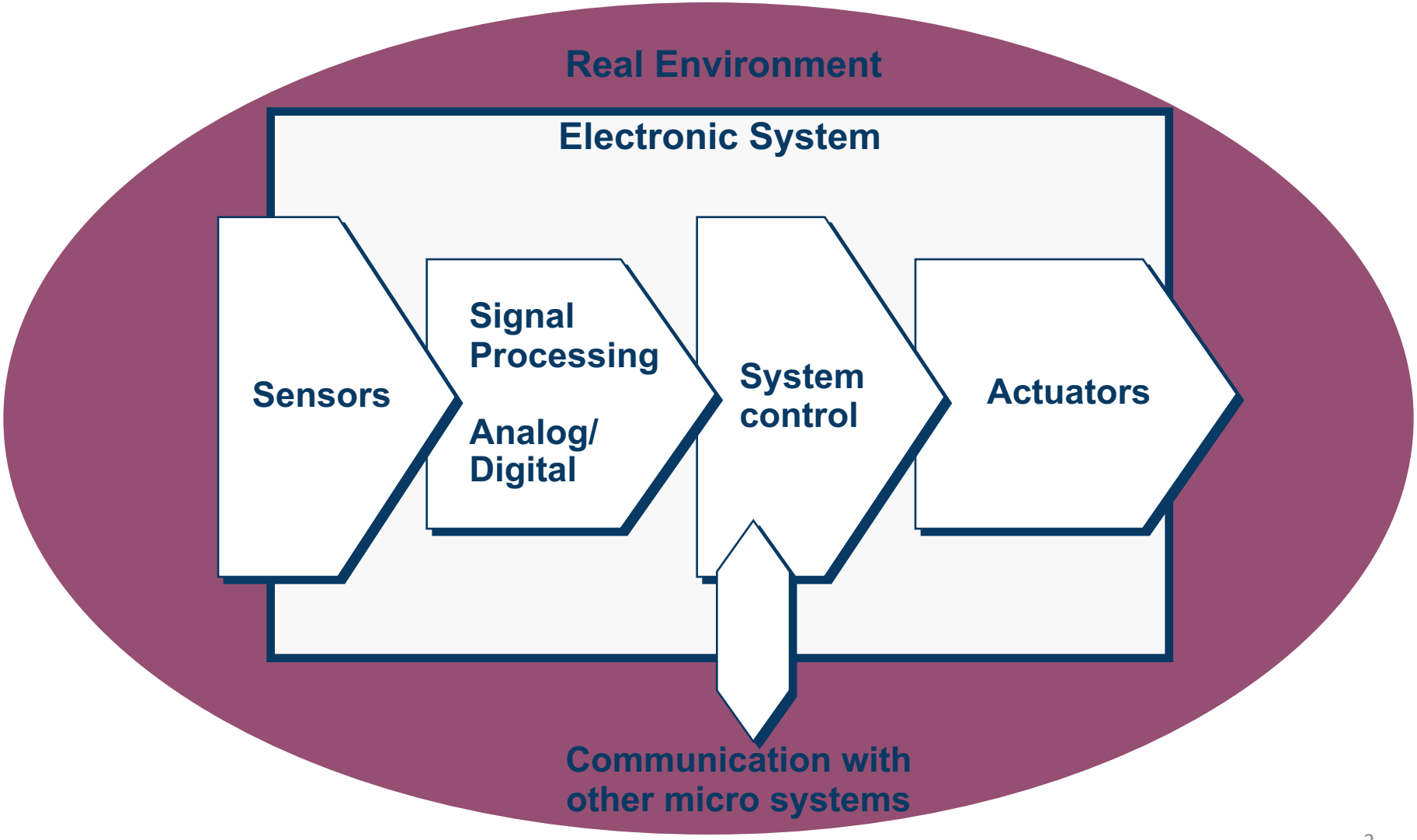
1. Design Process for Integrated Circuits (ICs) and Printed Circuit Boards (PCBs)

   1.1 Design and Realization Alternatives

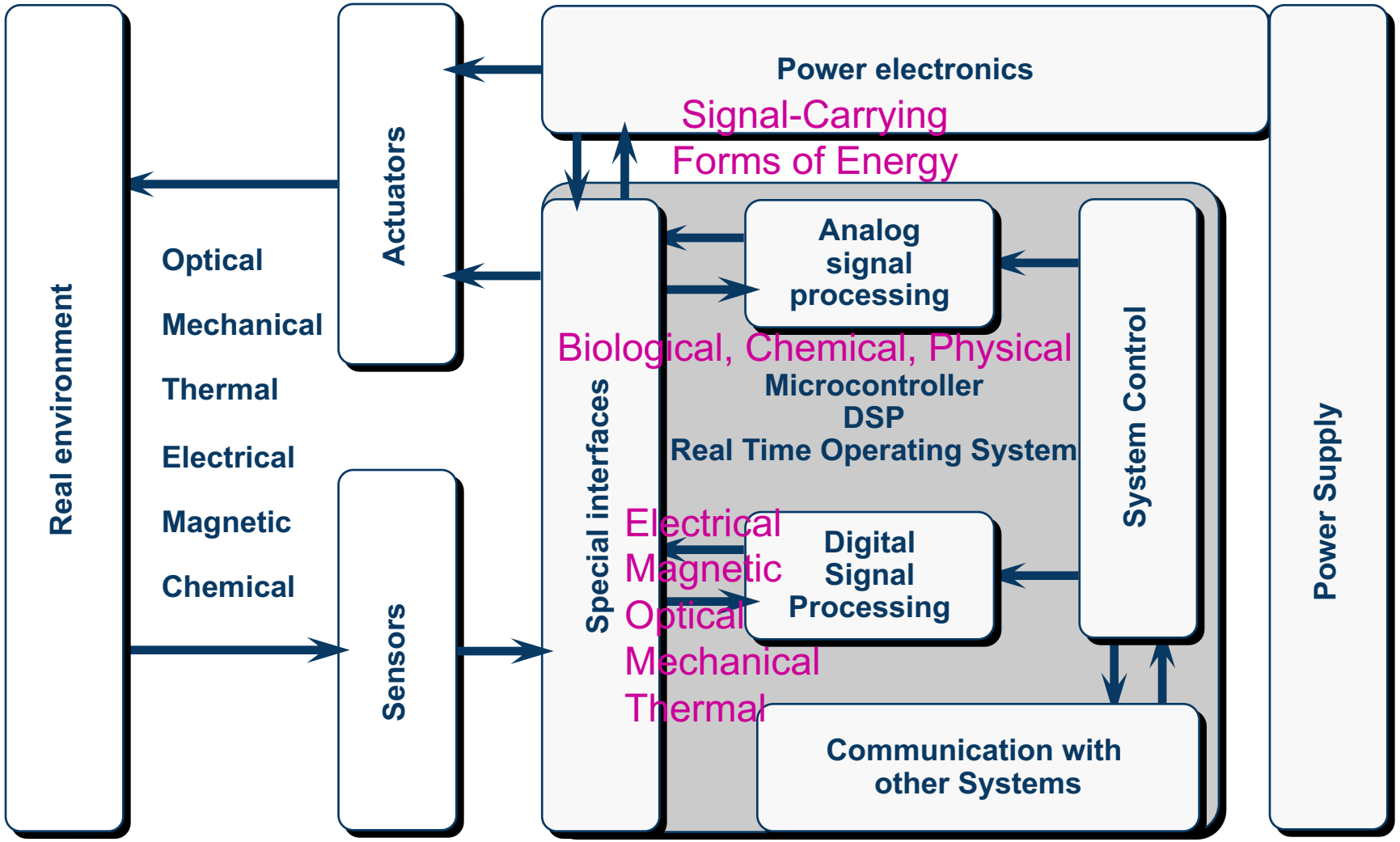
   1.2 Design Methods

   1.3 Use of Hardware Description Languages (HDLs)

Basic Functions of Electronic Systems



Smart Systems - Principal Structure



There are more embedded electronic systems around us than we think

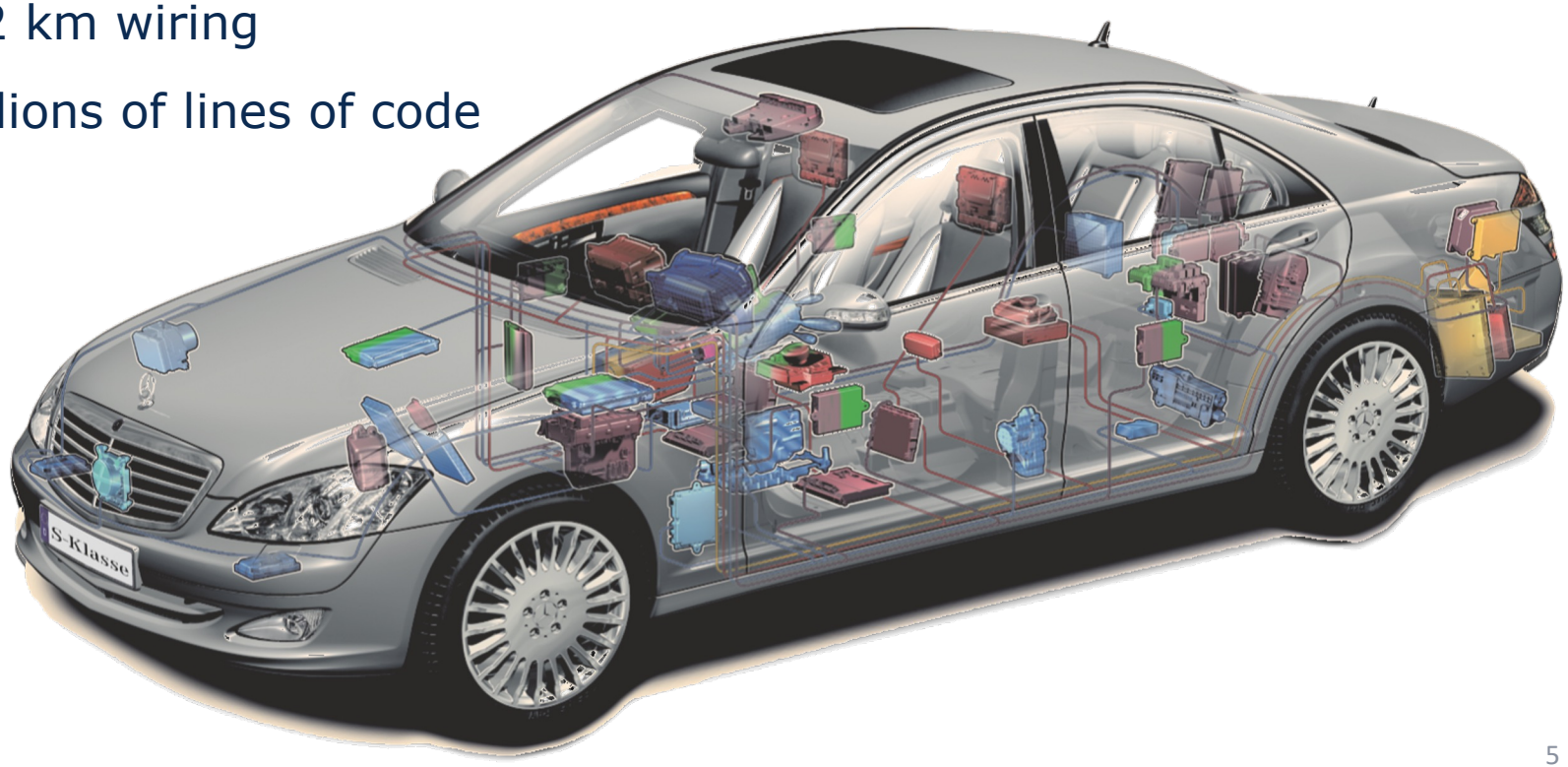
- **Audio**
- **Video**
- **Television**
- **Medical Equipment**
- **House keeping**
- **Office**

In an automobile

Characteristics: distributed system

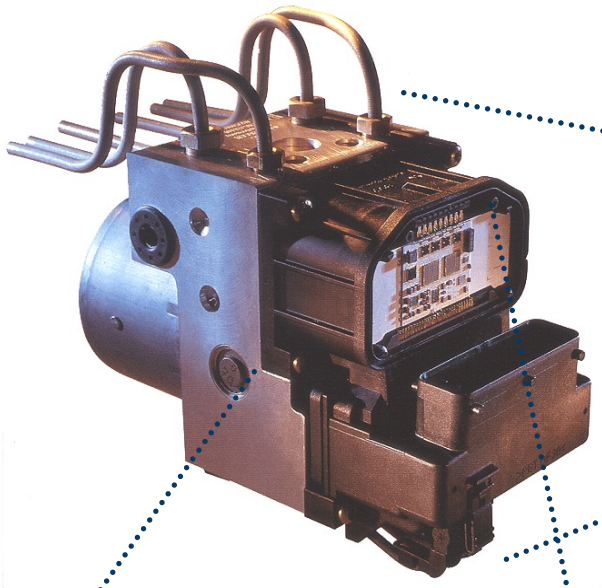
In a premium class car up to:

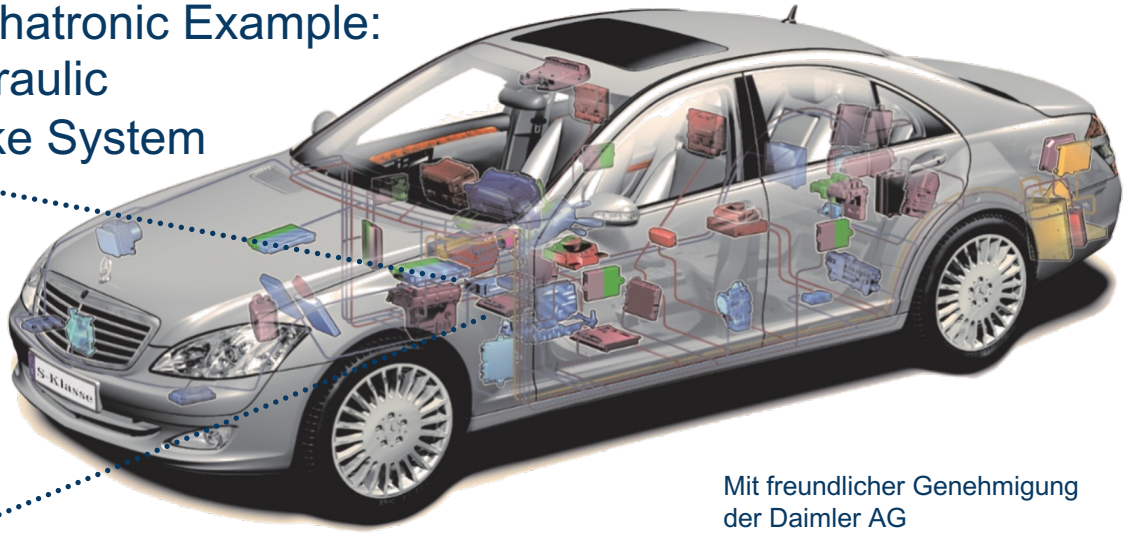
- 80 computers (Electronic Control Units ECU)
- > 100 Electrical Motors
- > 2 km wiring
- millions of lines of code



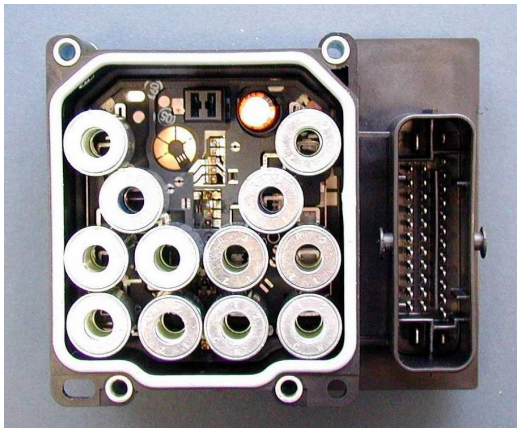
Automotive Electronic Control Units
Characteristics: distributed, mechatronic

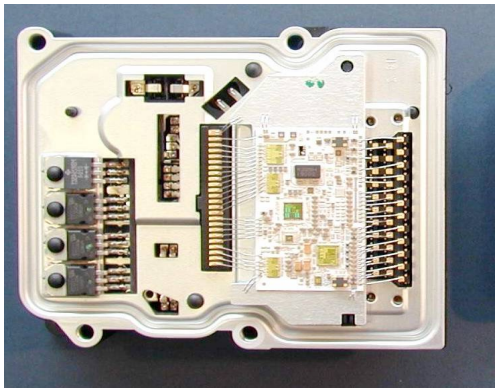
Mechatronic Example:
Hydraulic
Brake System





Mit freundlicher Genehmigung
der Daimler AG

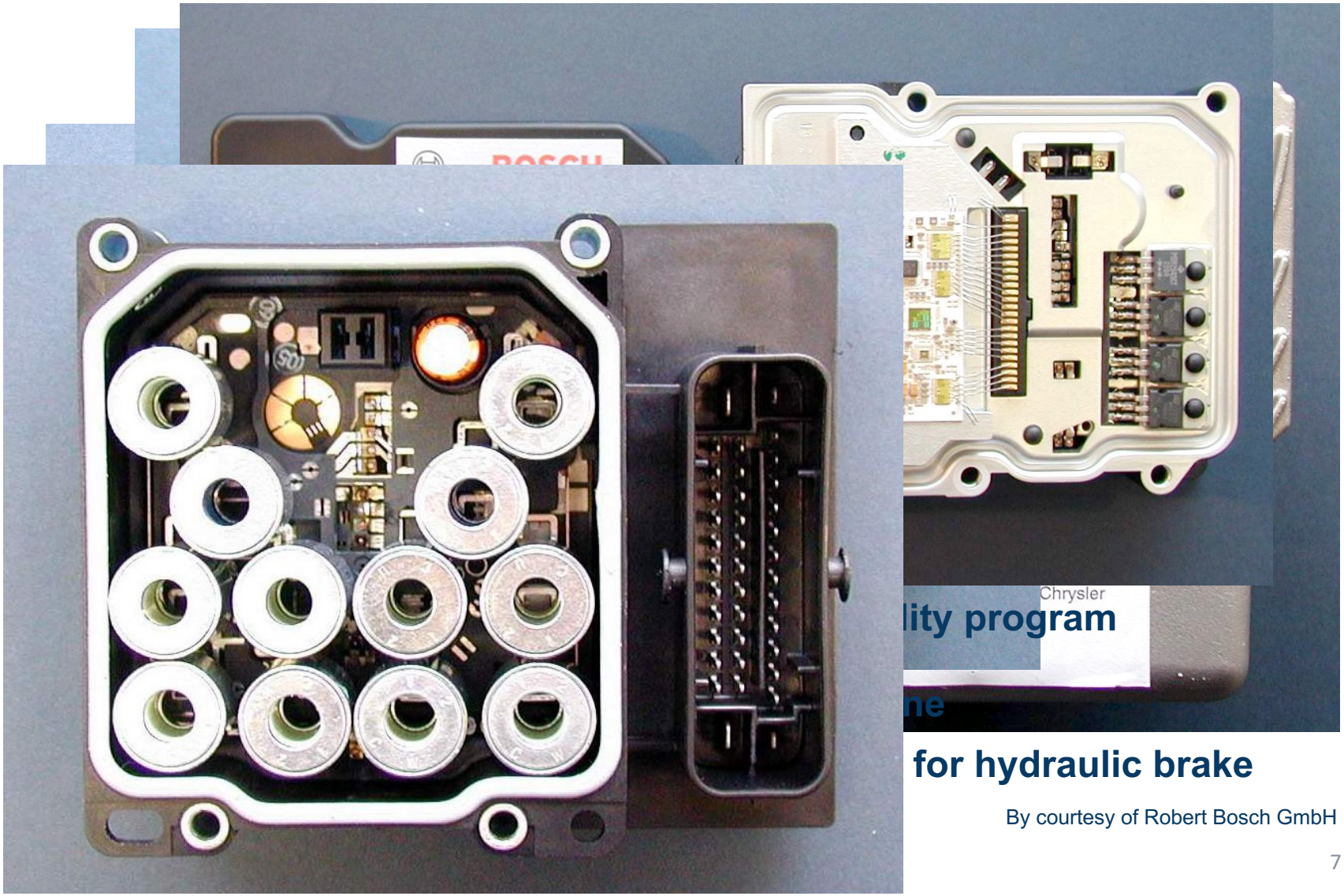




ECU's are part of mechatronic systems for measurement and control

Software is part of ECU

Electronic Control Units – ECU in a car



for hydraulic brake

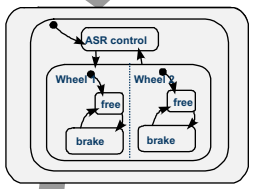
By courtesy of Robert Bosch GmbH

Typical Design Flow

Idea

System-Analysis
Executable Specs

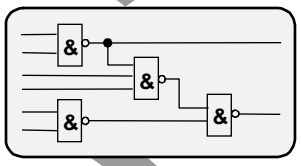
System Design
Customer Requirements
Technical Requirements
System Architecture
Interfaces
Modeling and Simulation
Real-Time Requirements



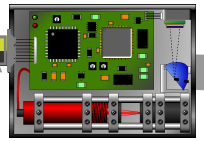
HW/SW-Requirements Analysis
Preliminary HW/SW-Design
HW-Architecture, SW-Architecture
Interface Description

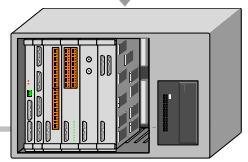
```
PROCESS (slippage, state)
BEGIN
  CASE state IS
    WHEN free =>
      IF slippage > 0 THEN
        next_state <= brake;
      ELSE
        next_state <= chk;
```

Rapid Prototyping
Hardware Platform
Code Generation
Real Time Operating System
Configurable Interfaces



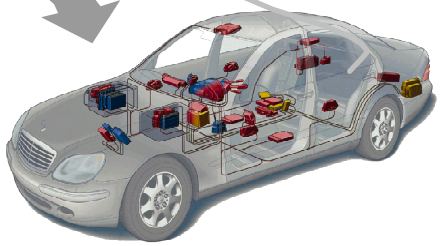
Detailed HW/SW-Design
SW-Design, Data Dictionary
HW-Drawings
HW-Analysis Report



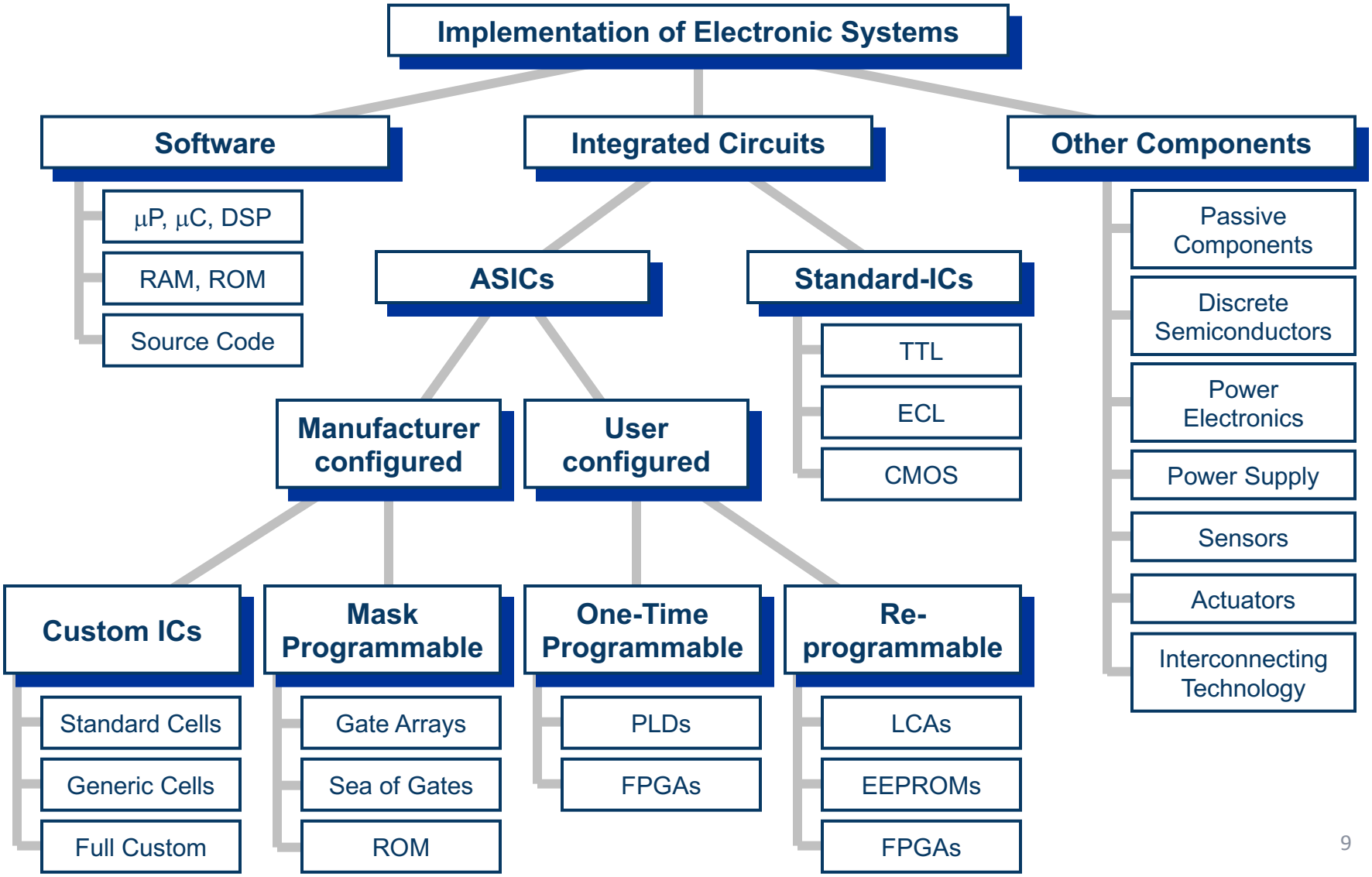


HW/SW-Implementation
Integration
SW-Modules, Data Dictionary,
SW-Component, HW-Component,
HW-Module, HW-Realization, Documents

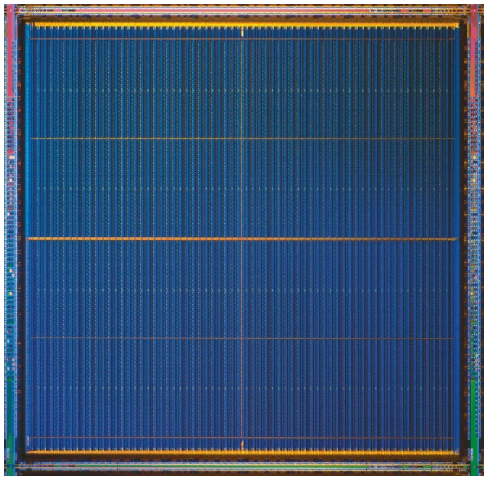
System Integration
Calibration, Application
Transition to Utilization



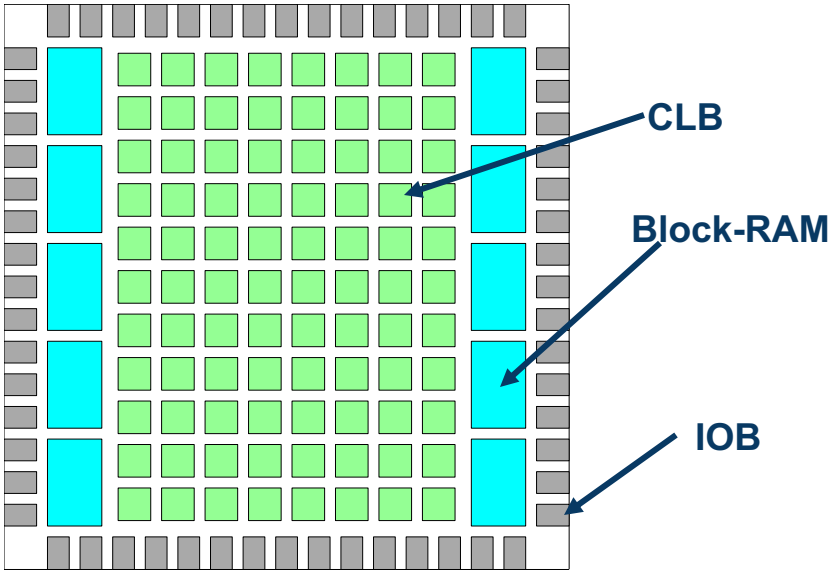
Implementation Alternatives



RAM-based FPGA



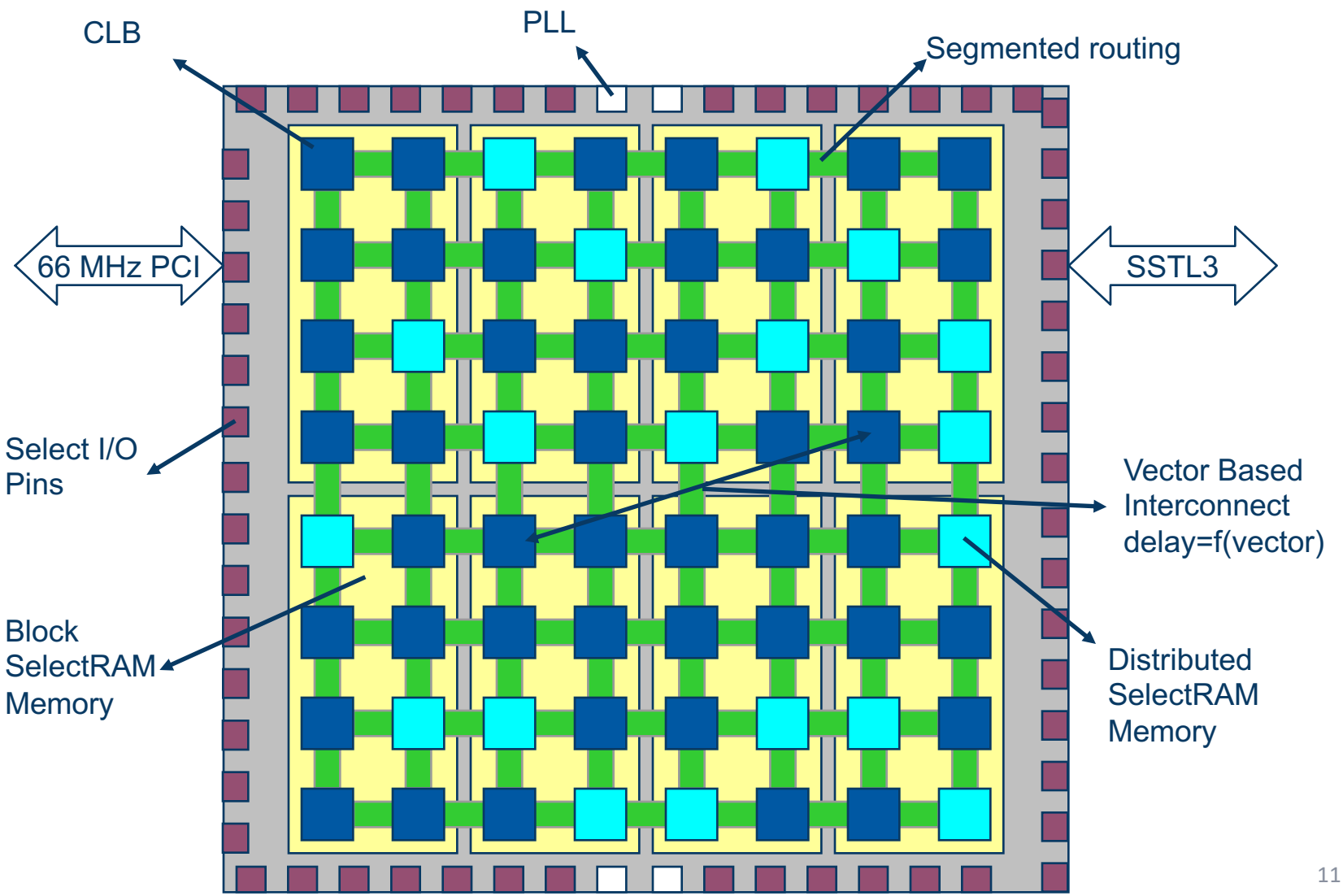
Xilinx VIRTEX
XCV2000 E



Properties:

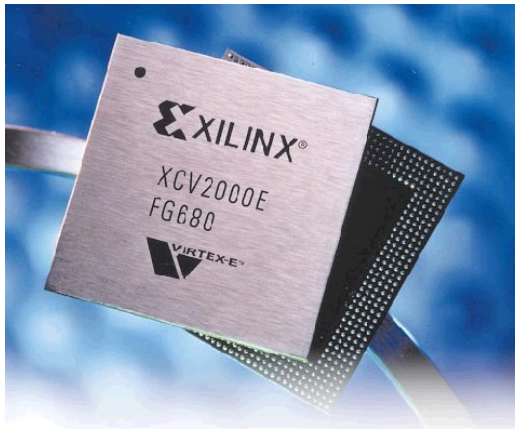
- ca. 2.500.000 gates
- 80*120 CLB-Array
- 43.200 “Logic Cells”
- Board system clock: 33MHz
- 0.18 μm 6- Layer CMOS-Process
- Can be reprogrammed infinitely times
  -> thanks to SRAM memory
- 1 MB internal SRAM memory
- 130 MHz internal CLB clock frequency
- I/O Performance: 622 Mb/s
- 404 In-/outputs

Xilinx Virtex: Architecture

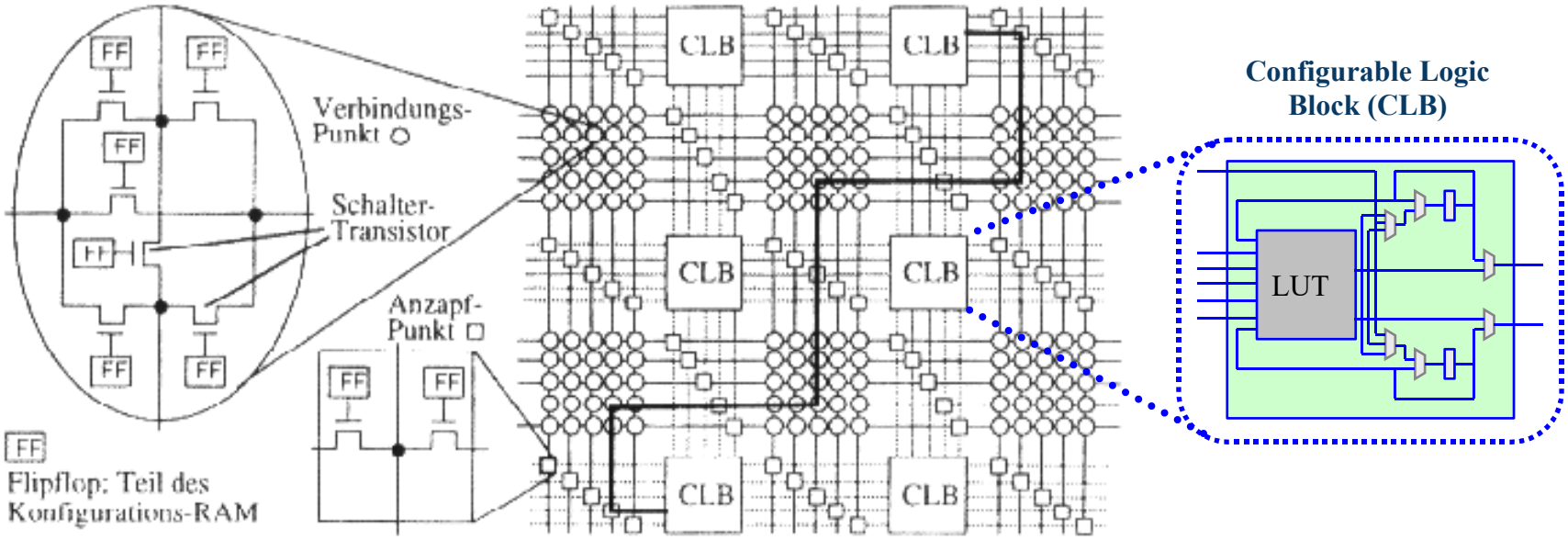


FPGA-Architectures:

- SRAM-based Look-up Tables (LUTs)
- Problems:
  - Interconnection: reduces performance
  - Relation: active / passive elements

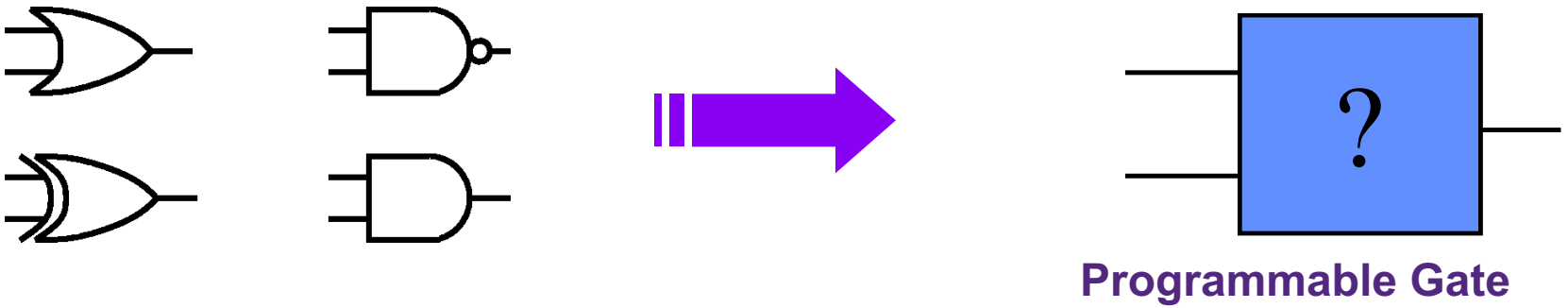


Rekonfigurable Interconnections (*Switching Boxes*)



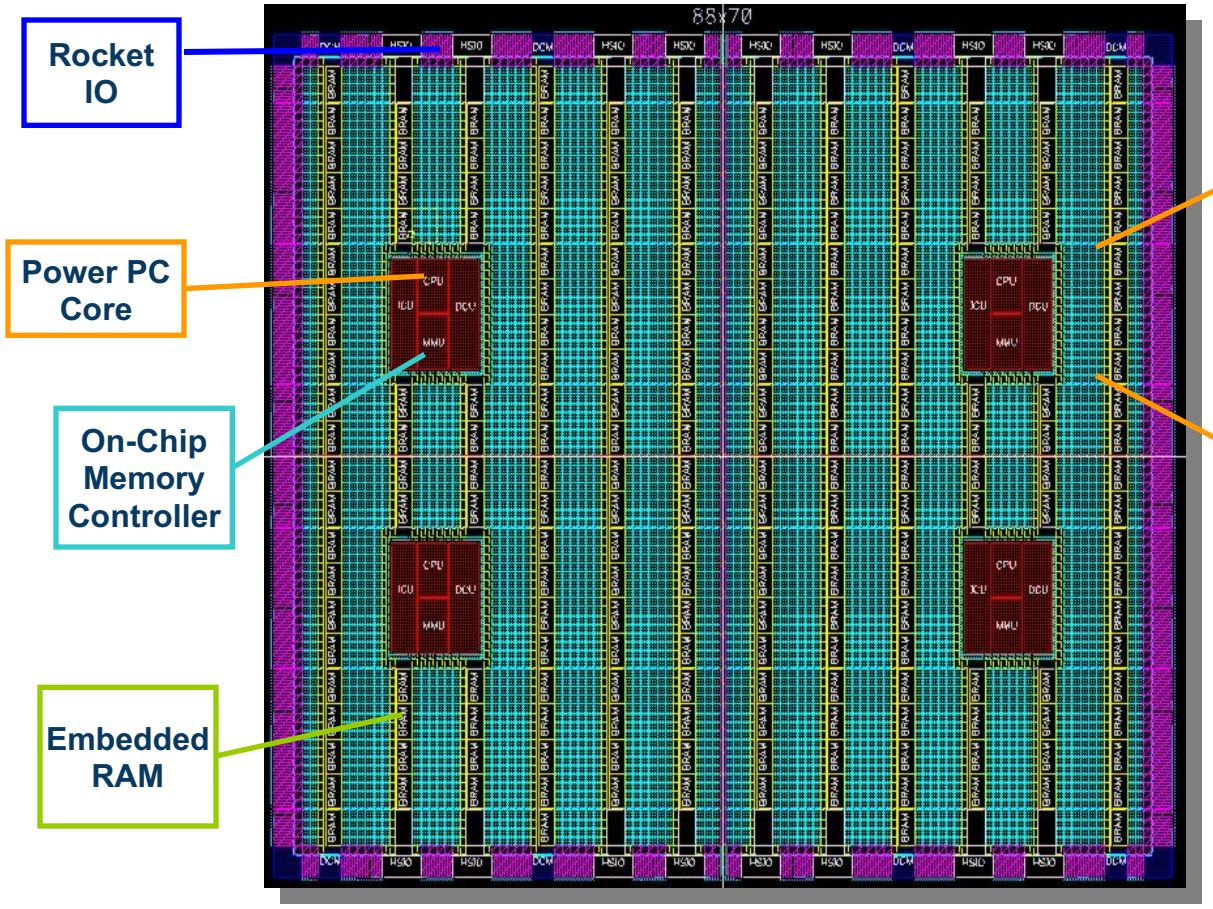
Source: R. Hartenstein

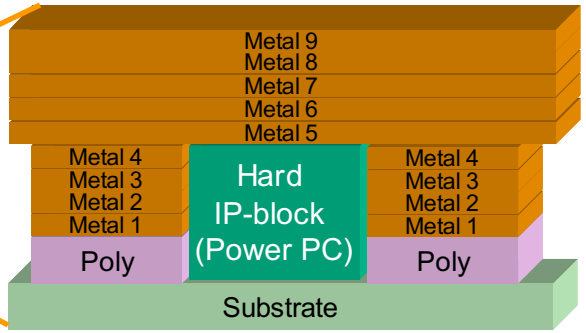
Set of programmable "Logic Gates", is integrated into a flexible interconnection network -> a "user-programmable" alternative to dedicated ciruits



Solution:

Virtex-II FPGA: integrated PowerPC ® 405 RISC CPUs (PPC405)

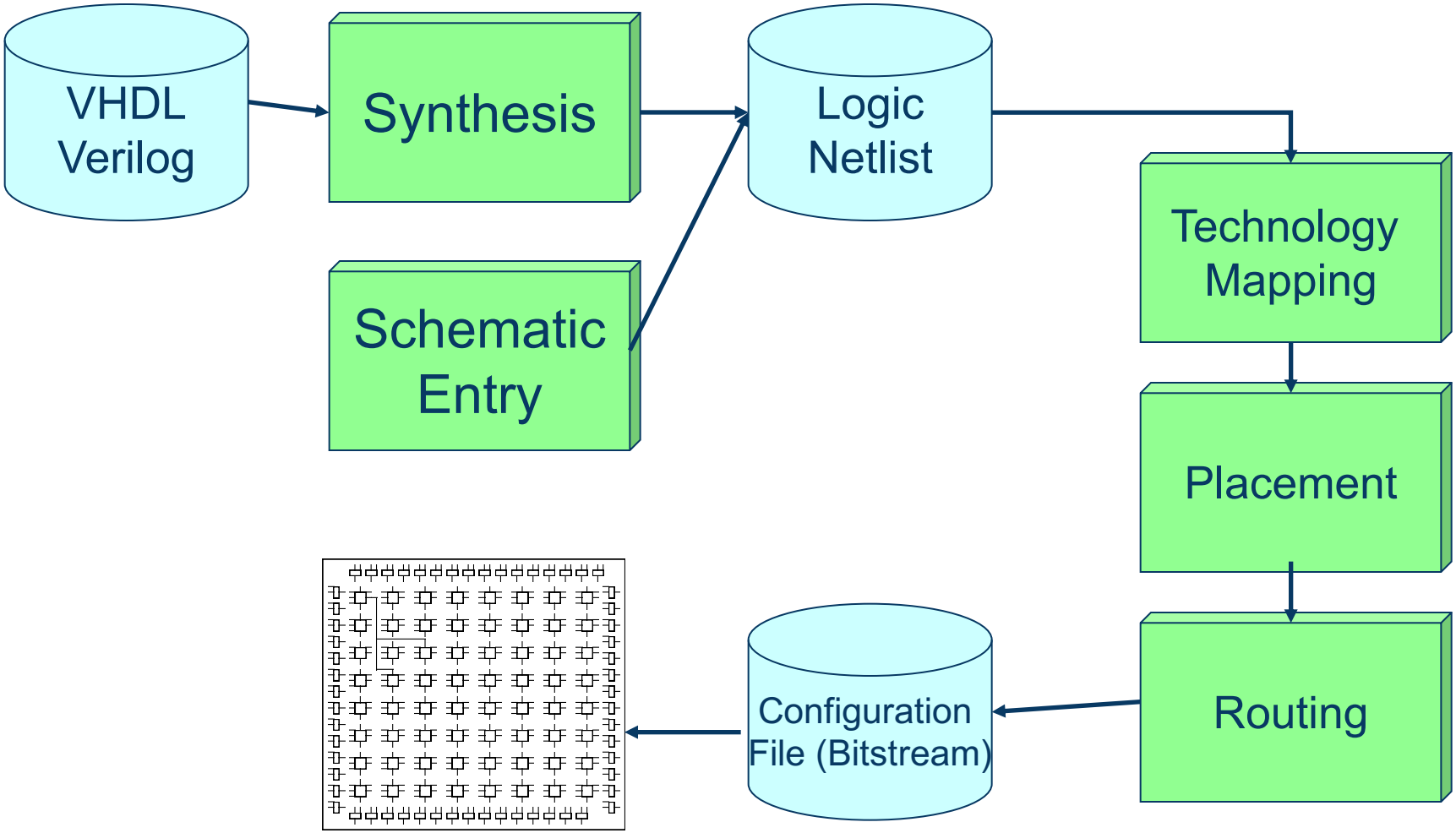




	Fastest DSP	Virtex-II
8x8 MAC	4.8 Billion MAC/s	0.5 Tera MAC/s
256-FIR	9.3 MSPS	180 MSPS
1024-FFT	10 µs	1 µs

Quelle: Ivo Bolsens, Xilinx, USA

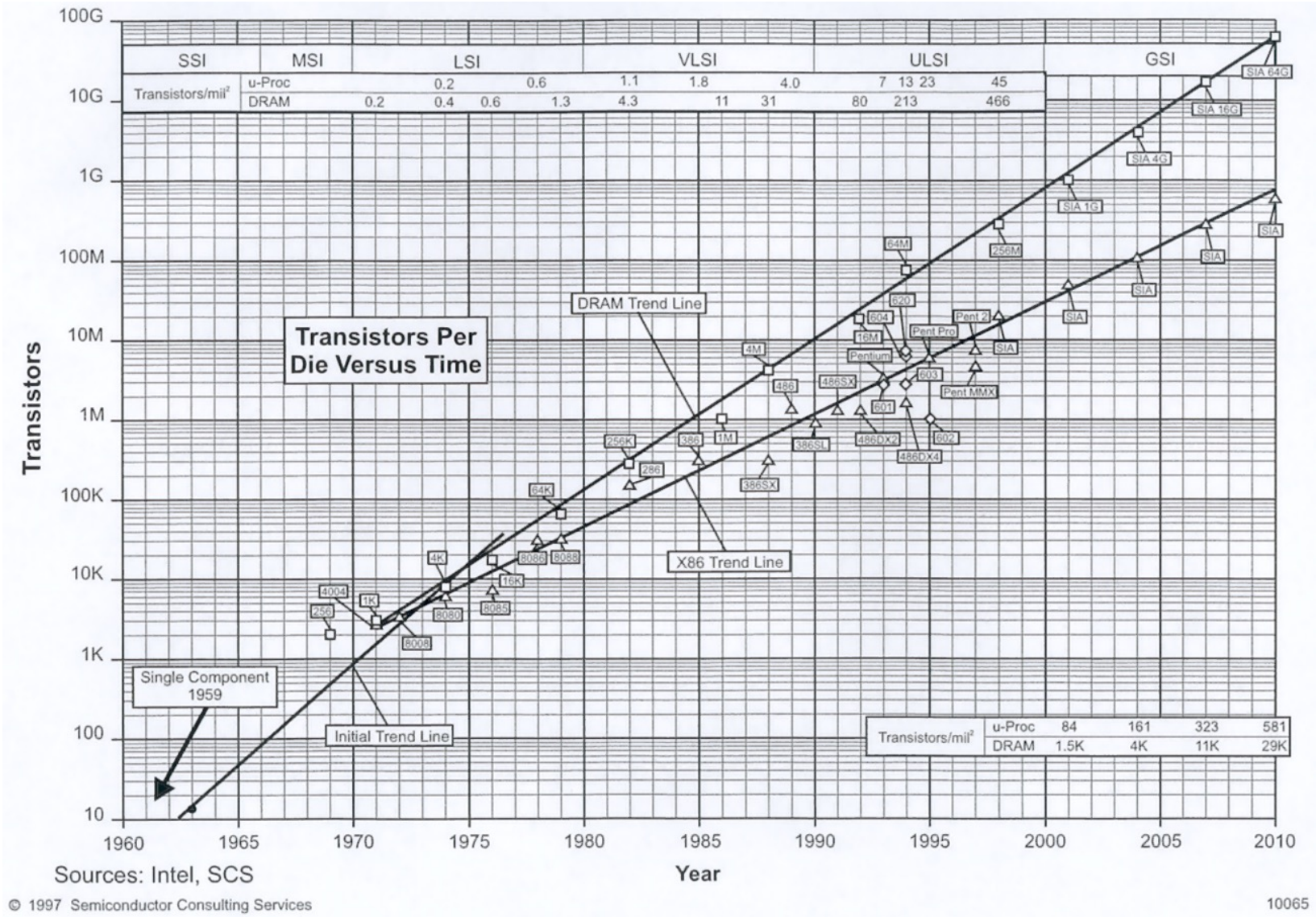
FPGA – Design and Implementation



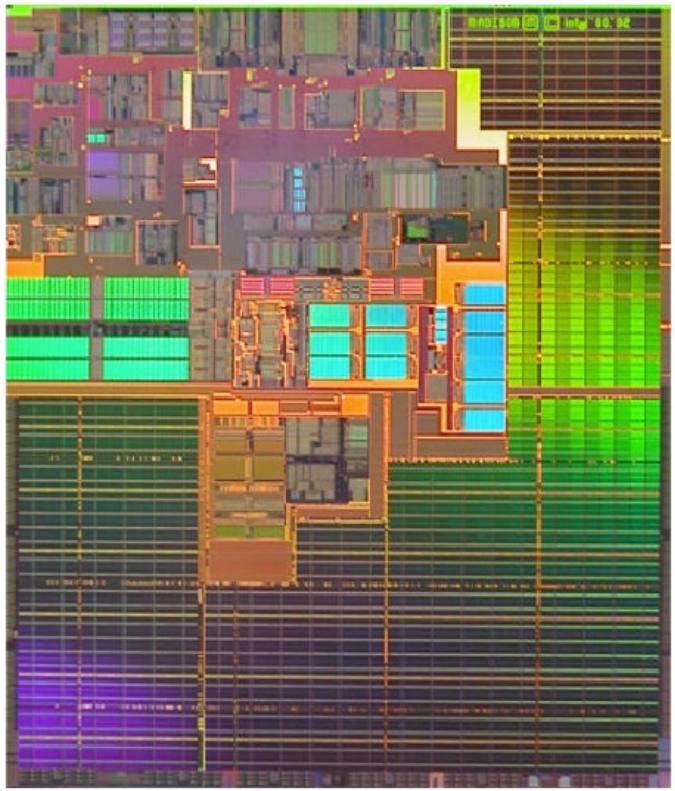
Design Constraints

- Increasing Complexity
  - Structure size decreases
  - Chip dimensions increase
  - → Number of transistors per chip quadruples every three years
- Higher Performance (clock, speed, power dissipation)
- Pressure of Competition
  - Almost no monopolies: extreme price pressure
  - Continually decreasing time for development
  - ➠ "Time to market" decides about success or failure of a product
- "Design Reuse" for Productivity and Quality Improvement
  - ➠ Requires complete and understandable documentation (IP - Intellectual Property)

Trend: Transistors Per Die



130nm Itanium® 2 Processor Highlights



- 410M transistors
- 374mm² die size
- 6MB on-die L3 cache
- 1.5GHz at 1.3V
- 6.4GB/s 400MT/s 4-way bus interface
- Plug-in compatible with existing platforms
- Extensive RAS, DFT and DFM features

Quelle: Jason Stinson (Intel), DAC 2003

Dunnington:

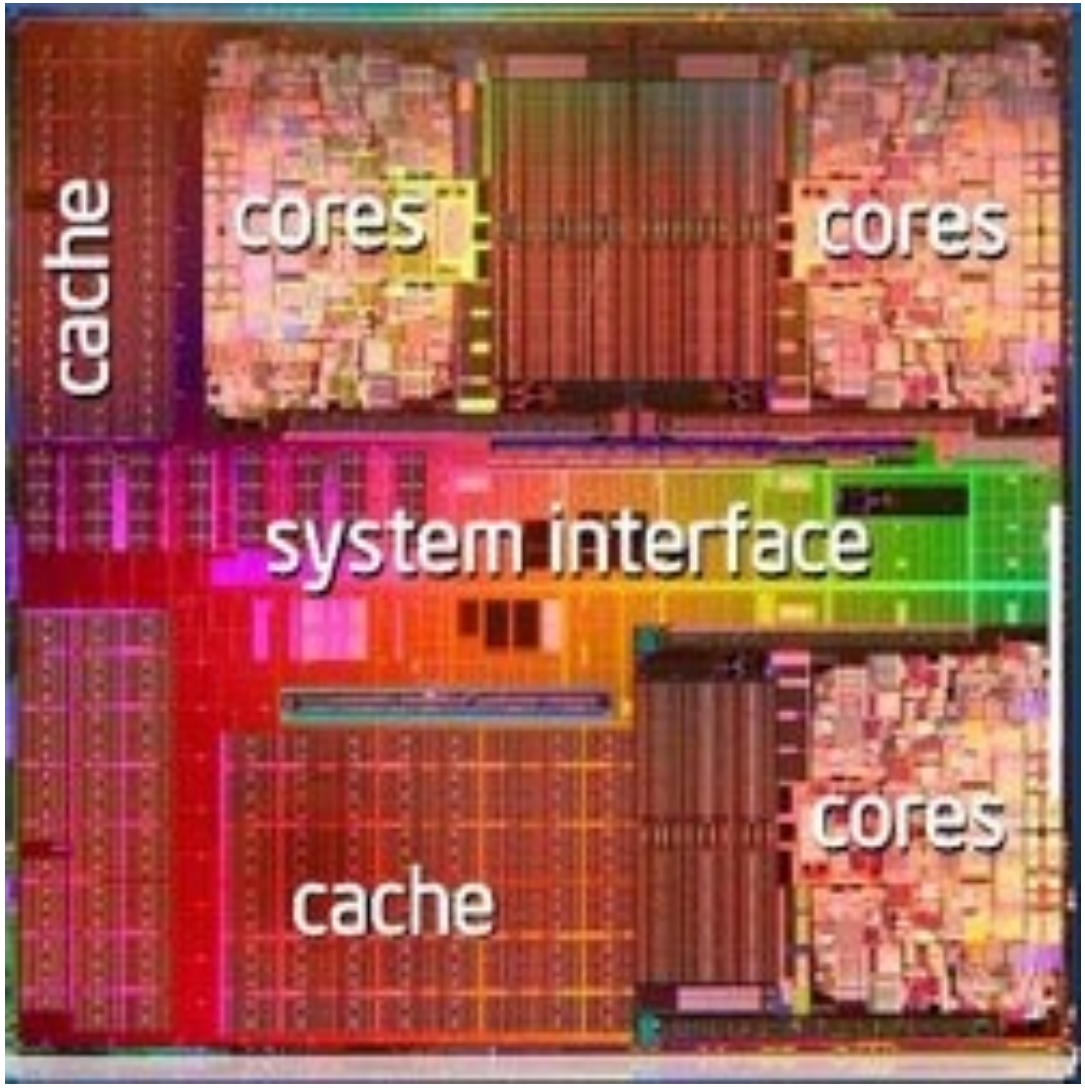
6 Penryn cores on one chip

3 MByte L2-Cache per dual-core

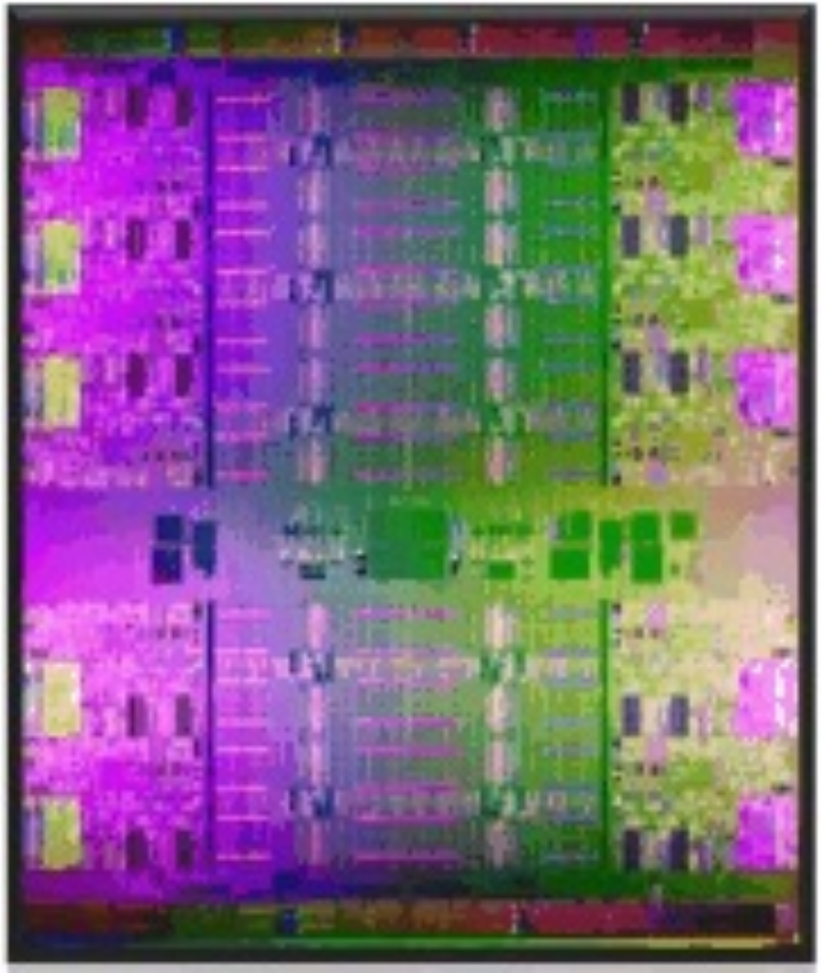
16 MByte L3-Cache

1,8 billion transistors

Publication:
Intel Developer Forum
Spring 2008



Intel Processors



Westmere-EX:

10 cores, 20 logical cores

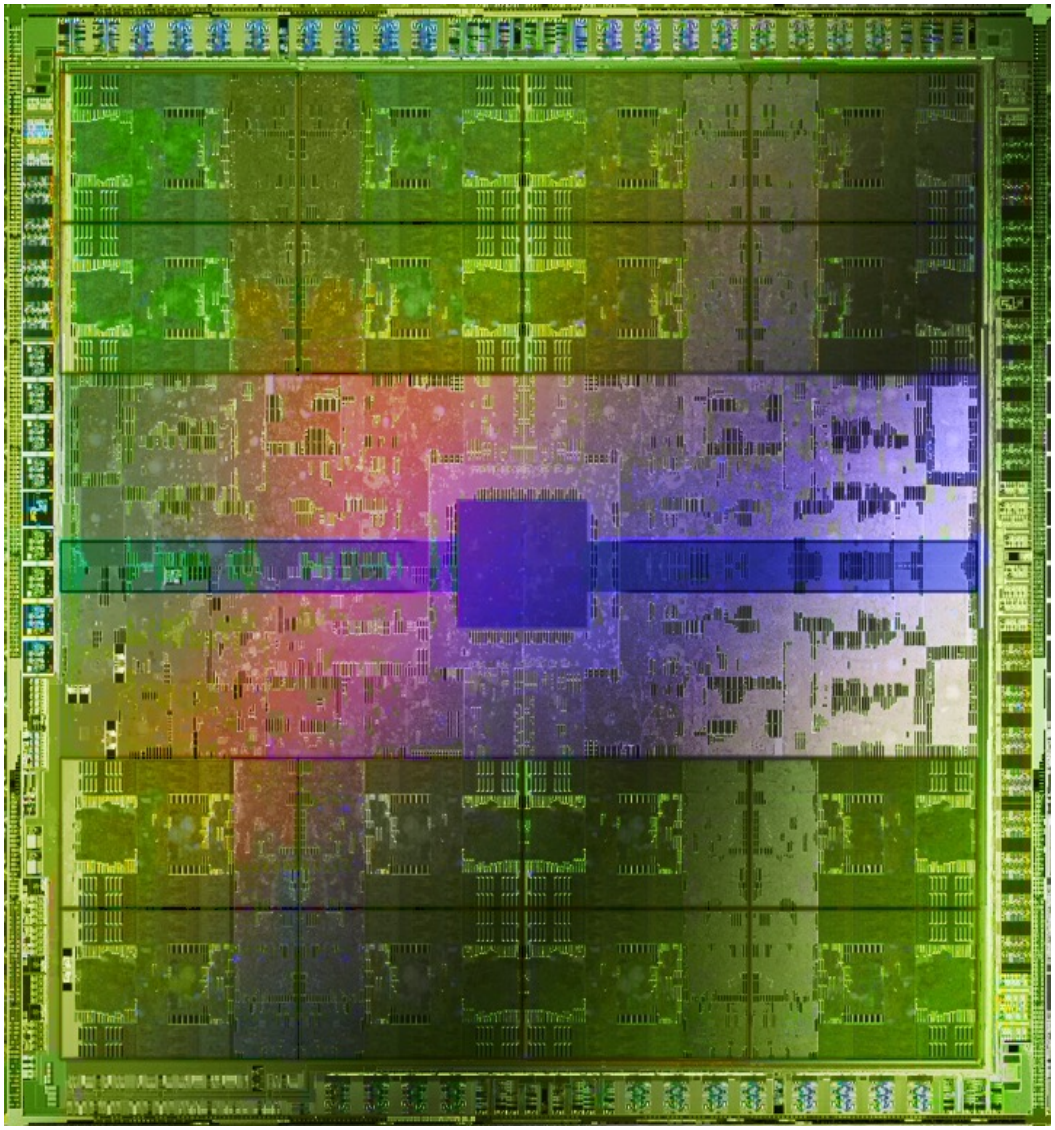
30 MByte L3-Cache

2,9 billion transistors

32 nm structures

Publication:
Spring 2011

NVIDIAs Fermi GPU



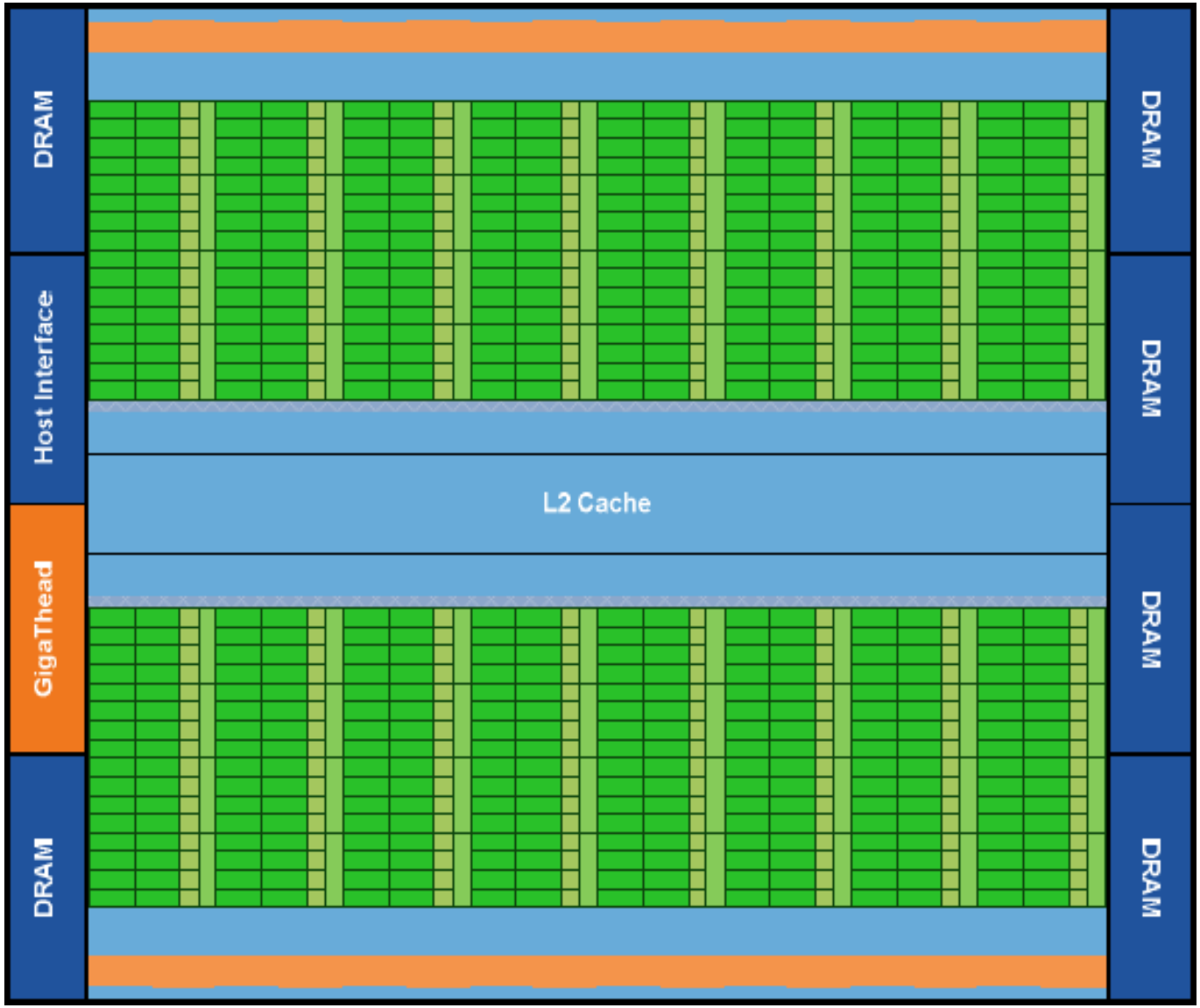
General Purpose Graphics Processing Unit (GPGPU)

Spring 2011
largest chip ever built

3 billion transistors

512 cores

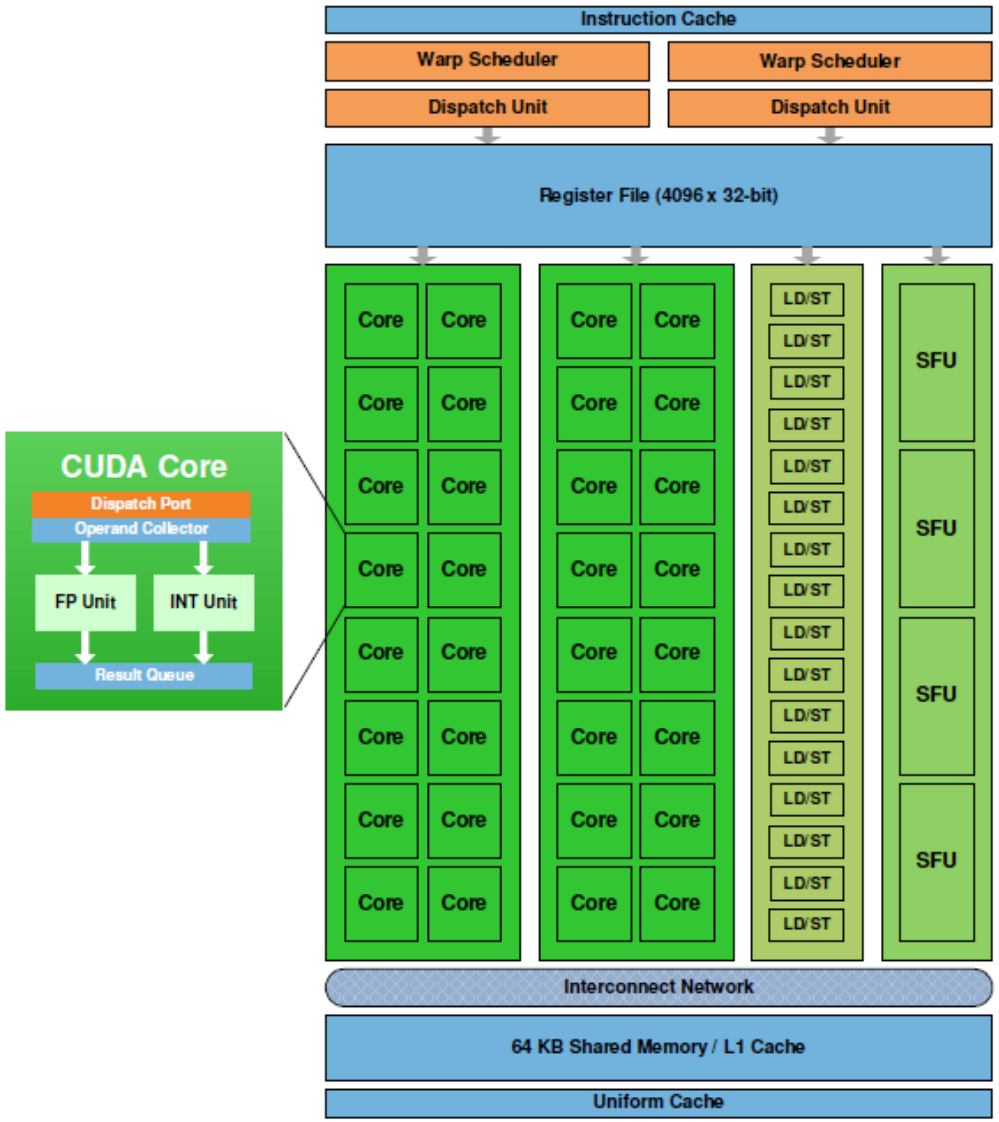
NVIDIAs Fermi GPU



- 32 cores form a SM Streaming Multiprocessor
- 16 SIMD units
- ECC support Error Detection and Correction
- Unified Level 2 Cache
- Six 64-bit memory partitions (384 bit memory interface)

NVIDIAs Fermi GPU

- 512 CUDA Cores (Compute Unified Device Architecture)
- Parallel computing architecture developed by NVIDIA
- Executes one floating point and one integer instruction per clock
- 16 Load / Store Units per SM
- SFU special function unit handles sine, cosine, square root, interpolation etc.
- 64KB Level 1 Cache



NVIDIAs GPU Development

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point Capability	None	30 FMA ops / clock	256 FMA ops /clock
Single Precision Floating Point Capability	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock
Warp schedulers / SM	1	1	2
Special Function Units (SFUs) / SM	2	2	4
Shared Memory / SM	16 KB	16 KB	Configurable 48 KB or 16 KB
L1 Cache	None	None	Configurable 16 KB or 48 KB
L2 Cache	None	None	768 KB
ECC Memory Support	No	No	Yes
Concurrent Kernels	No	No	Up to 16

Principles for mastering complexity:

1. **Modularization**
2. **Hierarchy**
3. **Abstraction**

(Divide and Conquer)

Modularization

Modularization = subdivide a system into several smaller, logically related parts (subsystems, modules), based on several criteria*:

- Black Box Concept
- Completeness
- Fully defined interfaces
- Manageability
- Testability
- Limited access
  (information hiding, access only through well defined interfaces)
- Minimum (no) mutual interference

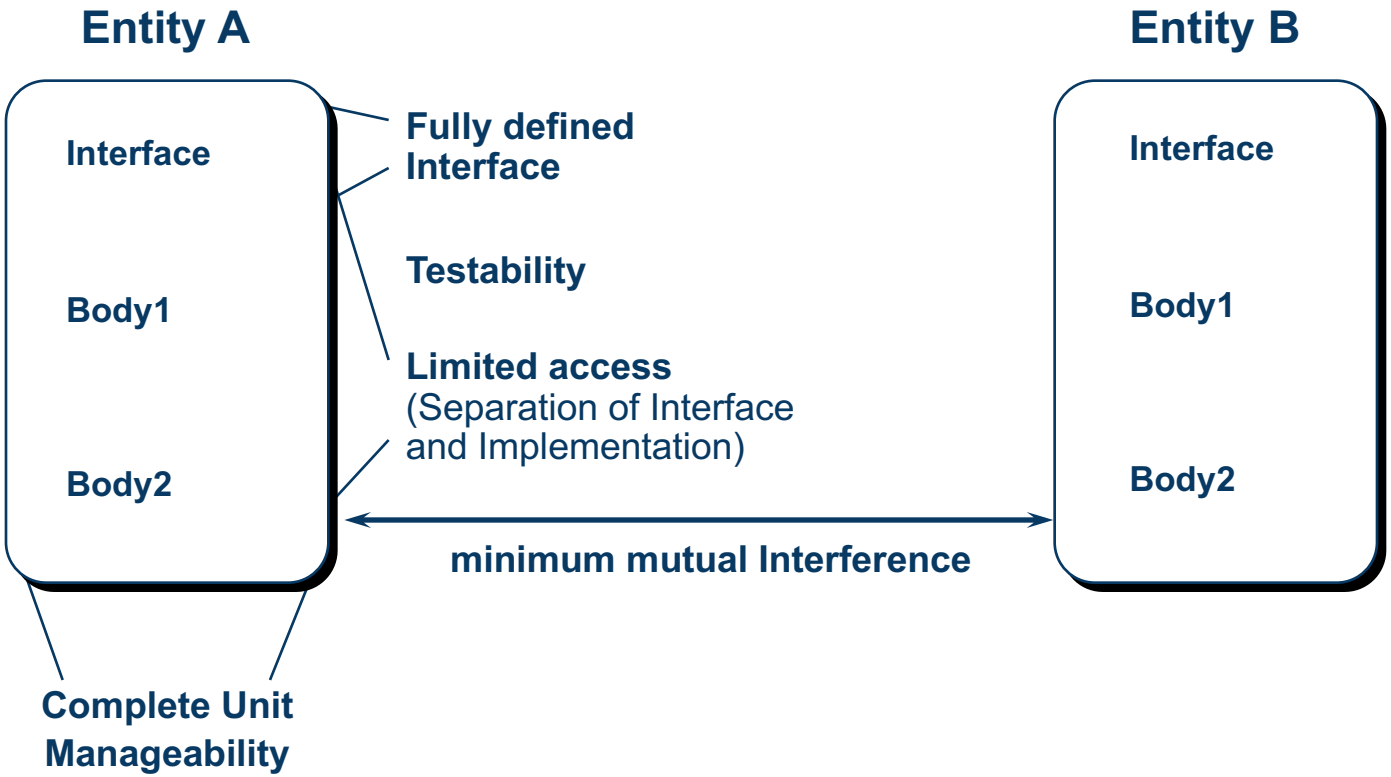
* according to E. Denert: "Software Modularisierung", Informatik Spektrum, Volume 2, Magazine 4, 1979

Modularization supported in VHDL

Entity concept in VHDL

VHDL = VHSIC Hardware Description Language

VHSIC = Very High Speed Integrated Circuits

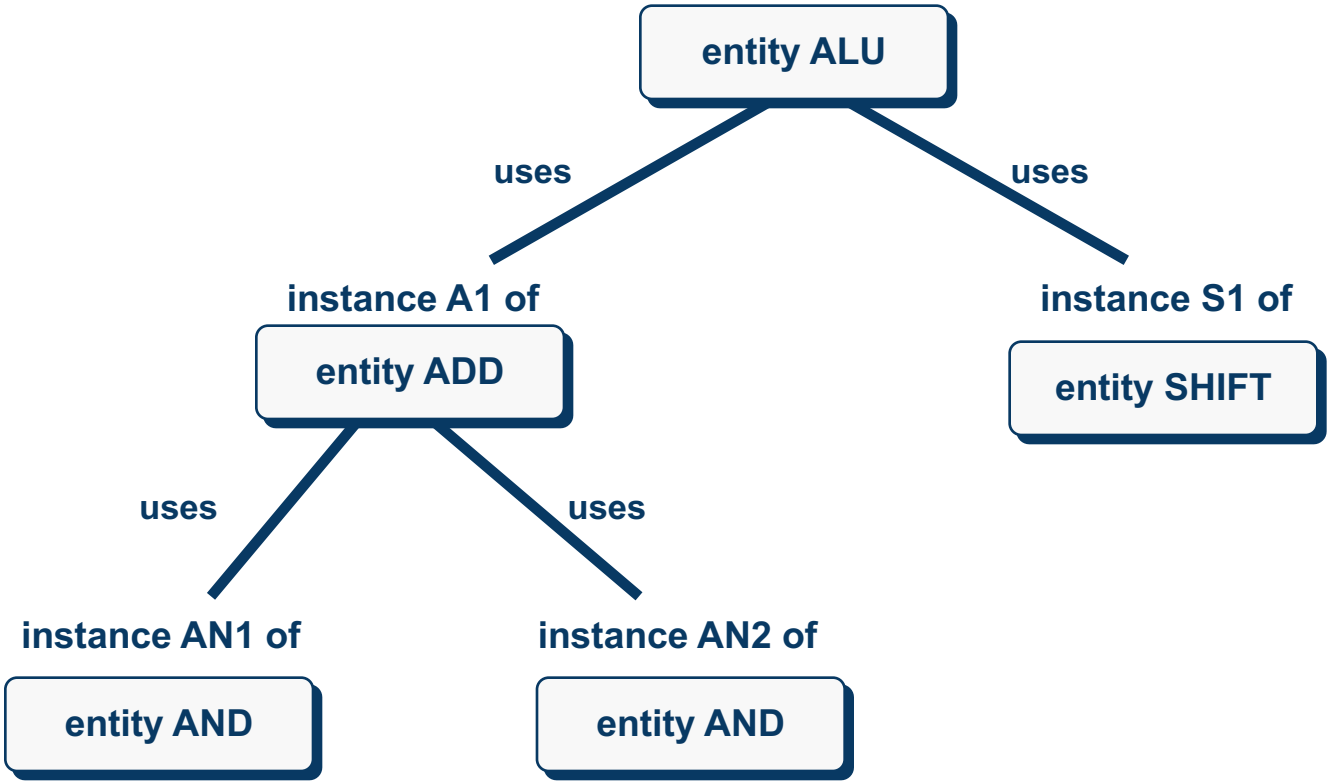


Hierarchy

- Hierarchy = Principle of order (formally: partial order) for the components of a system. Different relations between the components are considered for the creation of hierarchies.

- Example: “Uses” hierarchy
  Module A uses module B, if the correct operation of module A depends on the correct operation of module B, but not the other way round.

"Uses" hierarchy in VHDL via instances



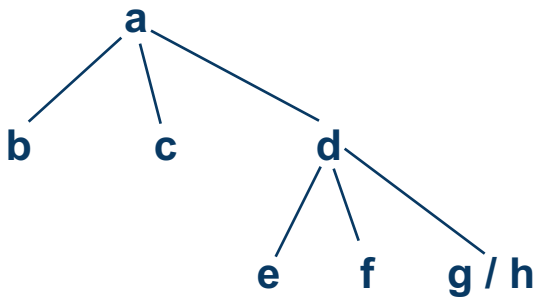
Partial Order

- A partial order ≤ on a set X is a binary relation that is reflexive, antisymmetric and transitive, i.e., it holds for all a, b and c in X that:
  - $a \leq a$ (reflexivity)
  - $a \leq b \wedge b \leq c \rightarrow a \leq c$ (transitivity)
  - $a \leq b \wedge b \leq a \rightarrow a = b$ (antisymmetry)

- Example:

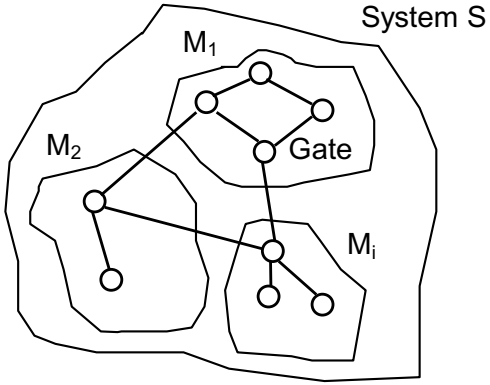
  Set X := {a, b, c, d, e, f, g, h}

  Relation: ≤ (Hierarchy)



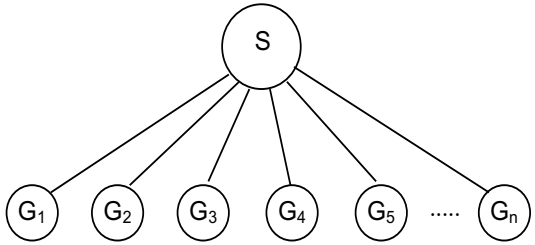
Hierarchy ≤ on X is a partial order:

a) $\forall x \in X: x \leq x$ (a ≤ a, b ≤ b, ...)
b) From d ≤ a and e ≤ d results e ≤ a
c) From h ≤ g and g ≤ h results g = h

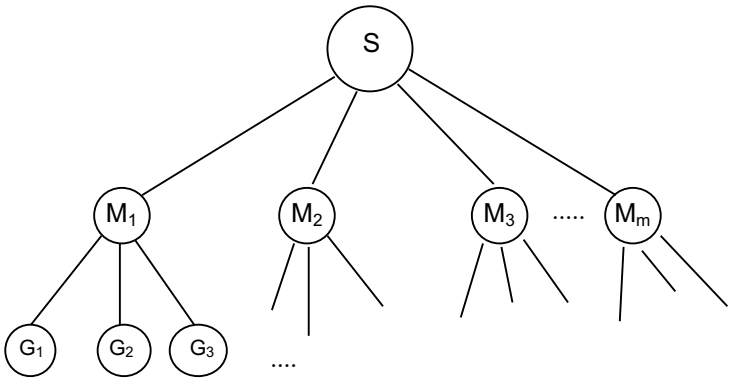


n: Number of Gates
m: Number of Modules

Without Modularization



With Modularization and Hierarchy



Abstraction

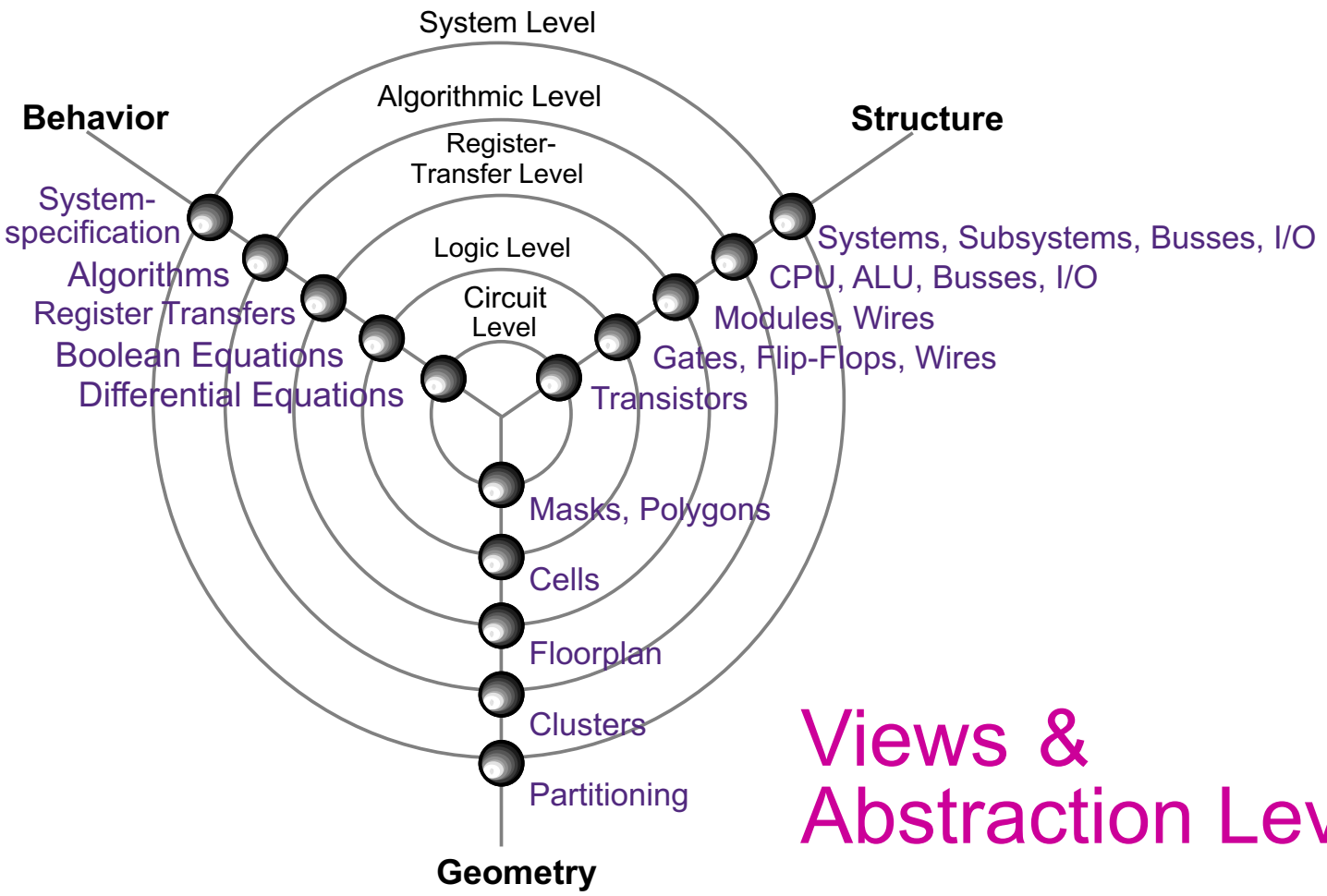
- Abstraction is hiding of details:
  Important and unimportant objects are determined.

- Creation of abstraction levels:
  Only those features of an object that are important on a certain level are considered on that level; unimportant details are not considered.

- Important basis for the creation of abstraction levels is the evenness of abstraction:
  All data selected for the description on a certain abstraction level should have the same degree of abstraction.

- Abstraction supported in VHDL:
  Algorithmic Level, Register Transfer Level, Logic Level

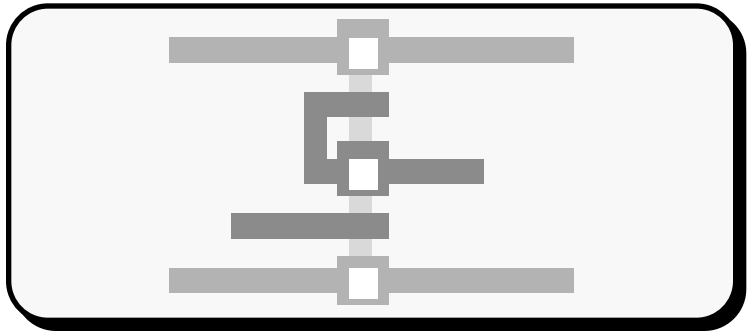
Y-Diagram according to Gajski and Walker



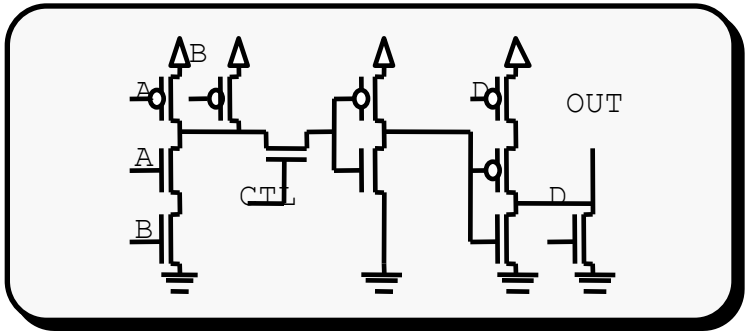
Views &
Abstraction Levels

Design Views

- Geometrical view
  - Polygons
  - Cells
  - Floor plan



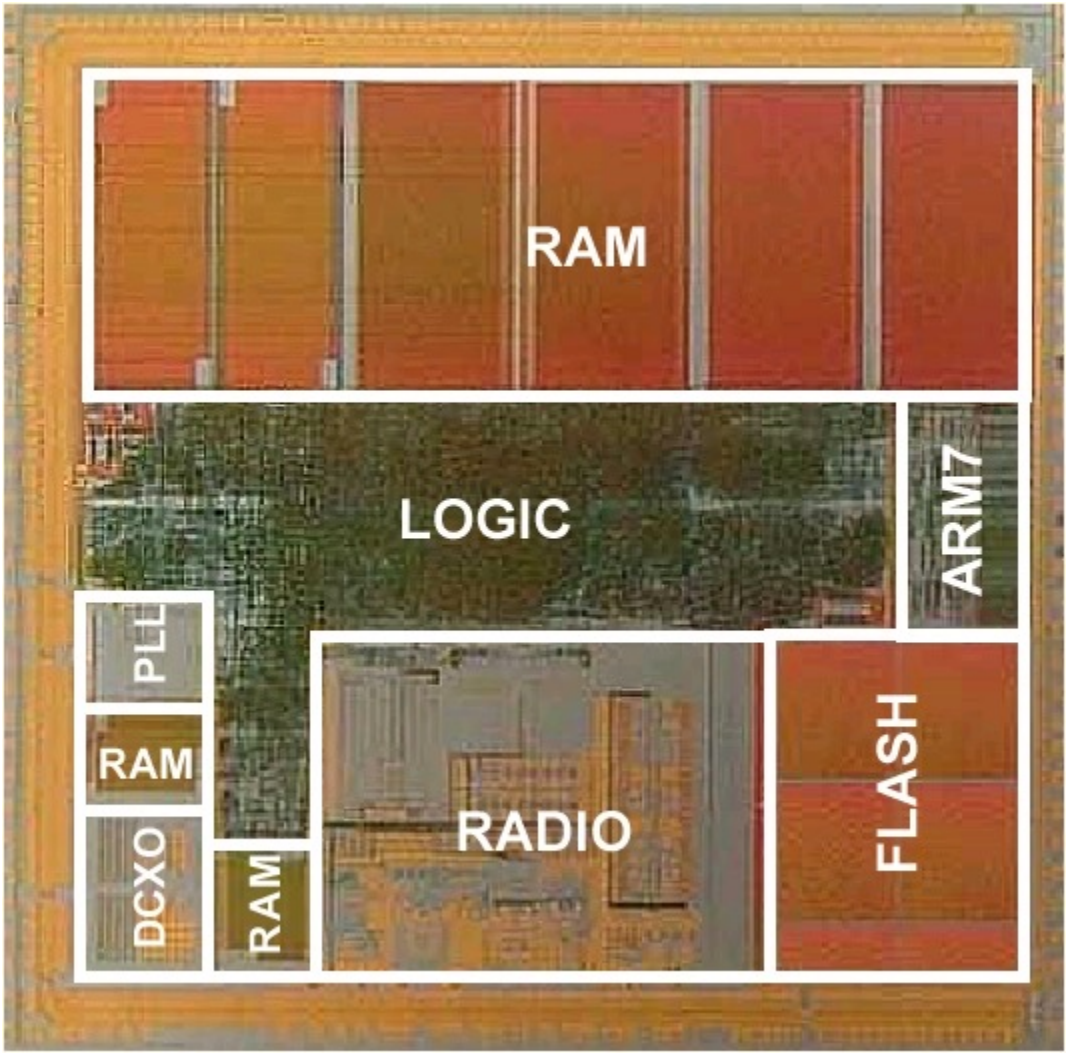
- Structural view
  - Transistors, Wires
  - Gates, Flip-Flops
  - Modules, Subsystems



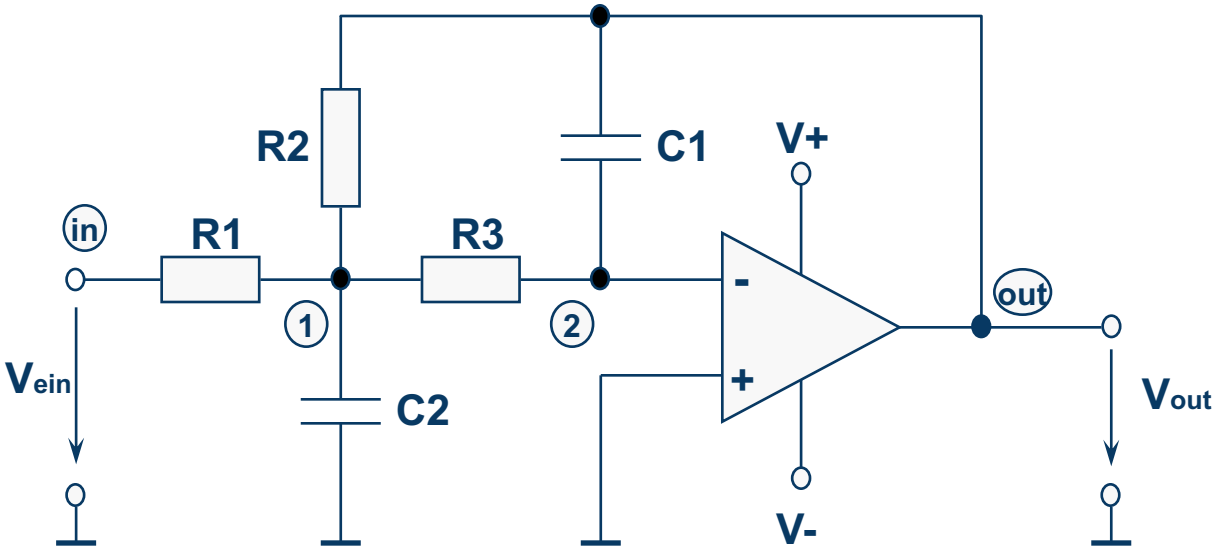
- Behavioral view
  - Differential Equations
  - Boolean Equations
  - Register Transfers, Algorithms

```
A := 3*B+C
IF (D=TRUE) THEN
  A := A + 1
ELSE A := A - 1
ENDIF
```

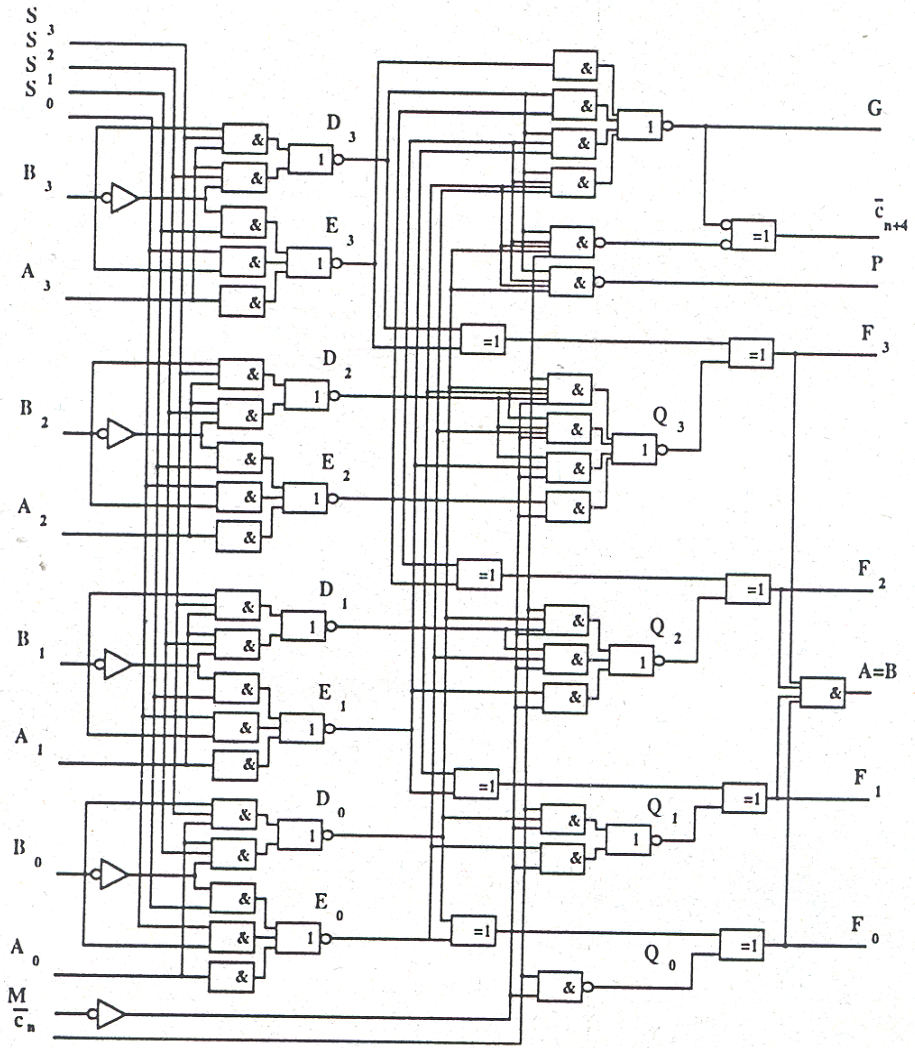
Bluetooth Chip - Geometrical View (Floorplan)



Analog Circuit - Structural View



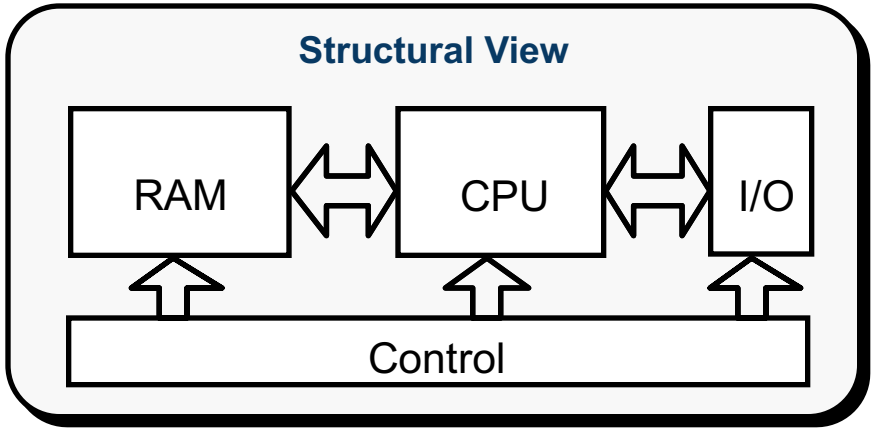
Digital Circuit - Structural View

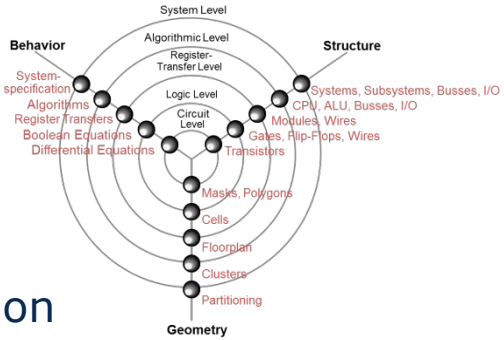


```
-- Mealy Machine
ENTITY mealy IS
   PORT (x, clock : IN  bit;
         z        : OUT bit);
END mealy;

ARCHITECTURE behavioral OF mealy IS
   TYPE state_type IS (s0, s1, s2, s3);
   SIGNAL current_state, next_state:
          state_type;
BEGIN
   -- process to hold synchronous elements
   sync: PROCESS
   BEGIN
      WAIT UNTIL clock'EVENT AND clock='1';
      current_state <= next_state;
   END PROCESS;

   -- process to hold combinational logic
   comb: PROCESS (current_state, x)
   BEGIN
      CASE current_state IS
      WHEN s0 =>
         IF x='0' THEN
            z <= '0';
            next_state <= s0;
         ELSE
            z <= '1';
            next_state <= s2;
         END IF;
      WHEN s1 =>
         IF x='0' THEN
            z <= '0';
            next_state <= s0;
         ELSE
            z <= '0';
            next_state <= s2;
         END IF;
      WHEN s2 =>
         IF x='0' THEN
            z <= '1';
            next_state <= s2;
         ELSE
            z <= '0';
            next_state <= s3;
         END IF;
      WHEN s3 =>
         IF x='0' THEN
            z <= '0';
            next_state <= s3;
         ELSE
            z <= '1';
            next_state <= s1;
         END IF;
      END CASE;
   END PROCESS;
END behavioral;
```

Abstraction Levels

- System Level
  - Basic Characteristics compute performance and communication bandwidth
  - Simplified Functionality
  - No Timing





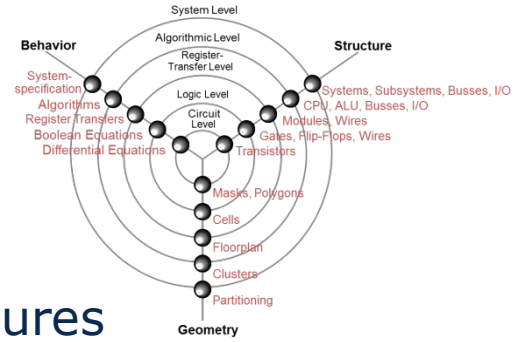
- Description Methods:
  - Behavior: System/Block Specification
  - Structure: CPUs, Memory, Interface Units
  - Geometry: Partitioning of Chip/Board Area

Abstraction Levels

- Algorithmic Level
  - Small kernel algorithms
  - No consideration of target architectures (floating point, word length
  - No Timing Details
  - No System Clock

Behavioral View

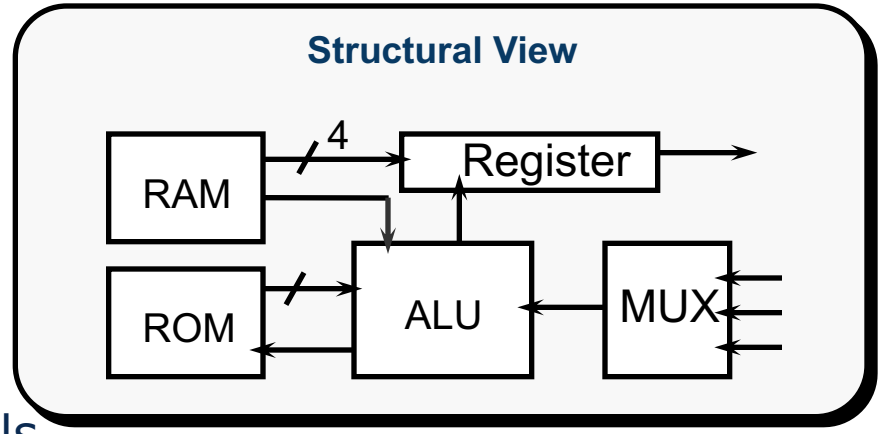
```
A := 3*B+C
IF (D=TRUE) THEN
   A := A + 1
ELSE A := A - 1
ENDIF
```

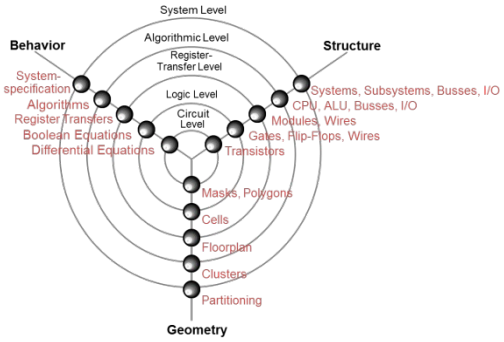


- Description Methods:
  - Behavior: Functions and Procedures
  - Structure: Subsystems, Busses
  - Geometry: Cluster

Abstraction Levels

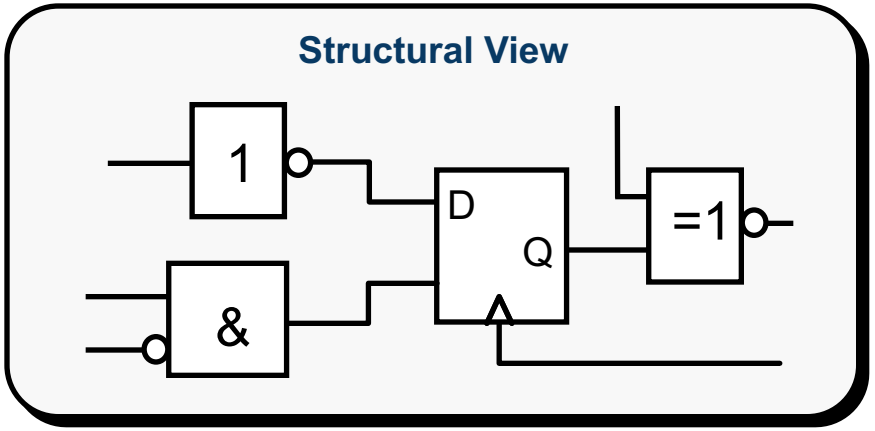
- Register-Transfer Level
  - Basic Operations (e.g. Addition, bit_vectors)
  - Transfers between Registers
  - Timing via Clock/Reset Signals
  - discrete values, discrete time
    Two- (multi-) valued logic signals

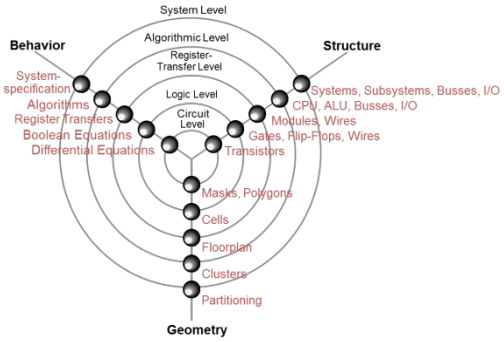




  - Description Methods:
    - Behavior: Arithmetic Operations, Finite State Machines
    - Structure: Modules (Registers, Encoder, ...)
    - Geometry: Floor Plan

- Logic Level
  - Multi-valued logic signals still discrete time
  - Single wires and busses
  - Timing (Delays)

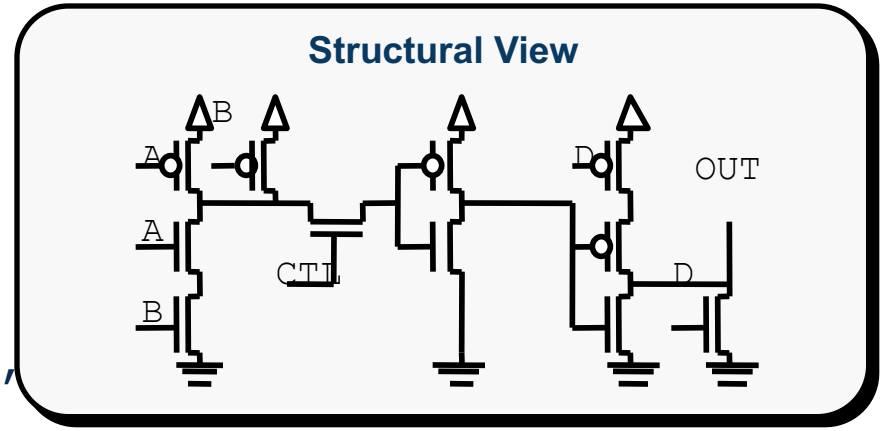


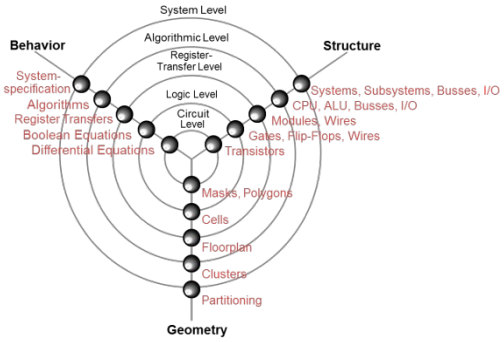


  - Description Methods:
    - Behavior: Boolean Equations
    - Structure: Gates, Flip-Flops (Net List)
    - Geometry: Cells

Abstraction Levels

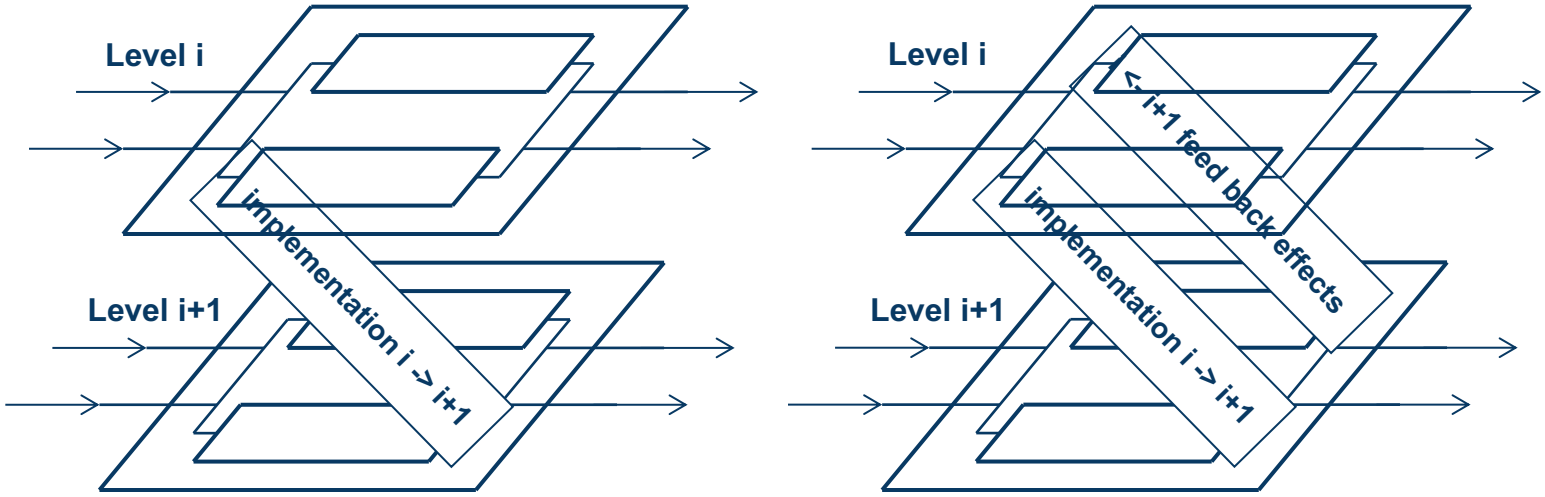
- Circuit Level
  - Value- and Time-Continuous Signals (Voltages and Currents)
  - Actual Components (transistors, diodes, resistors, capacitors, inductors)
  - Detailed Component Models



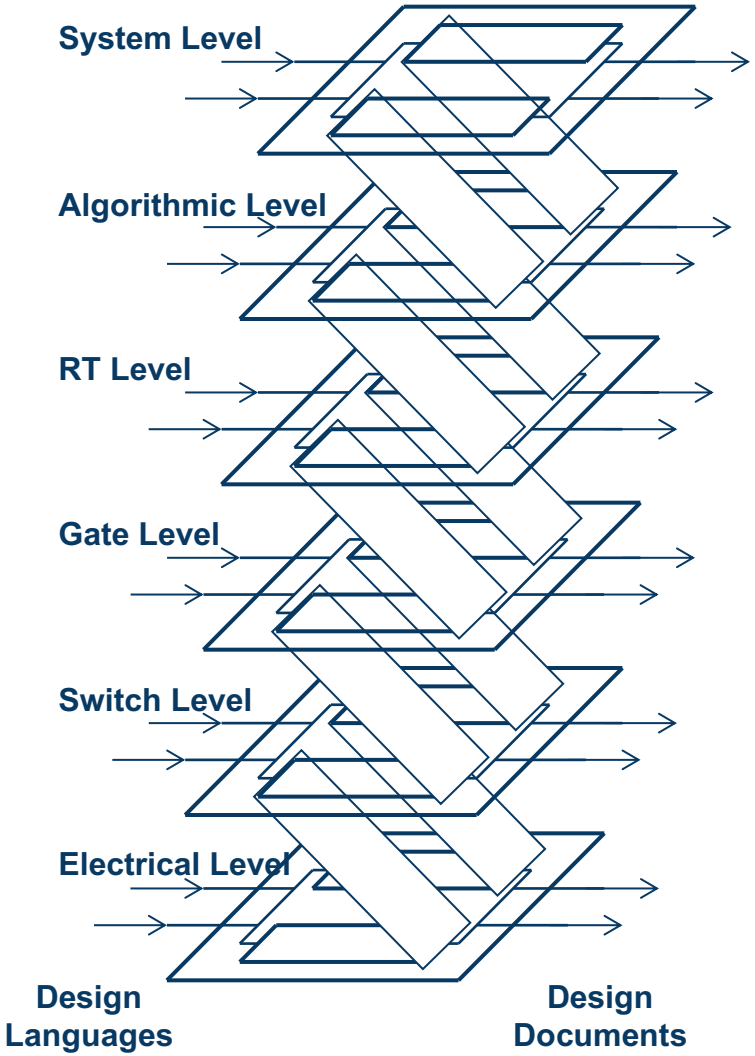


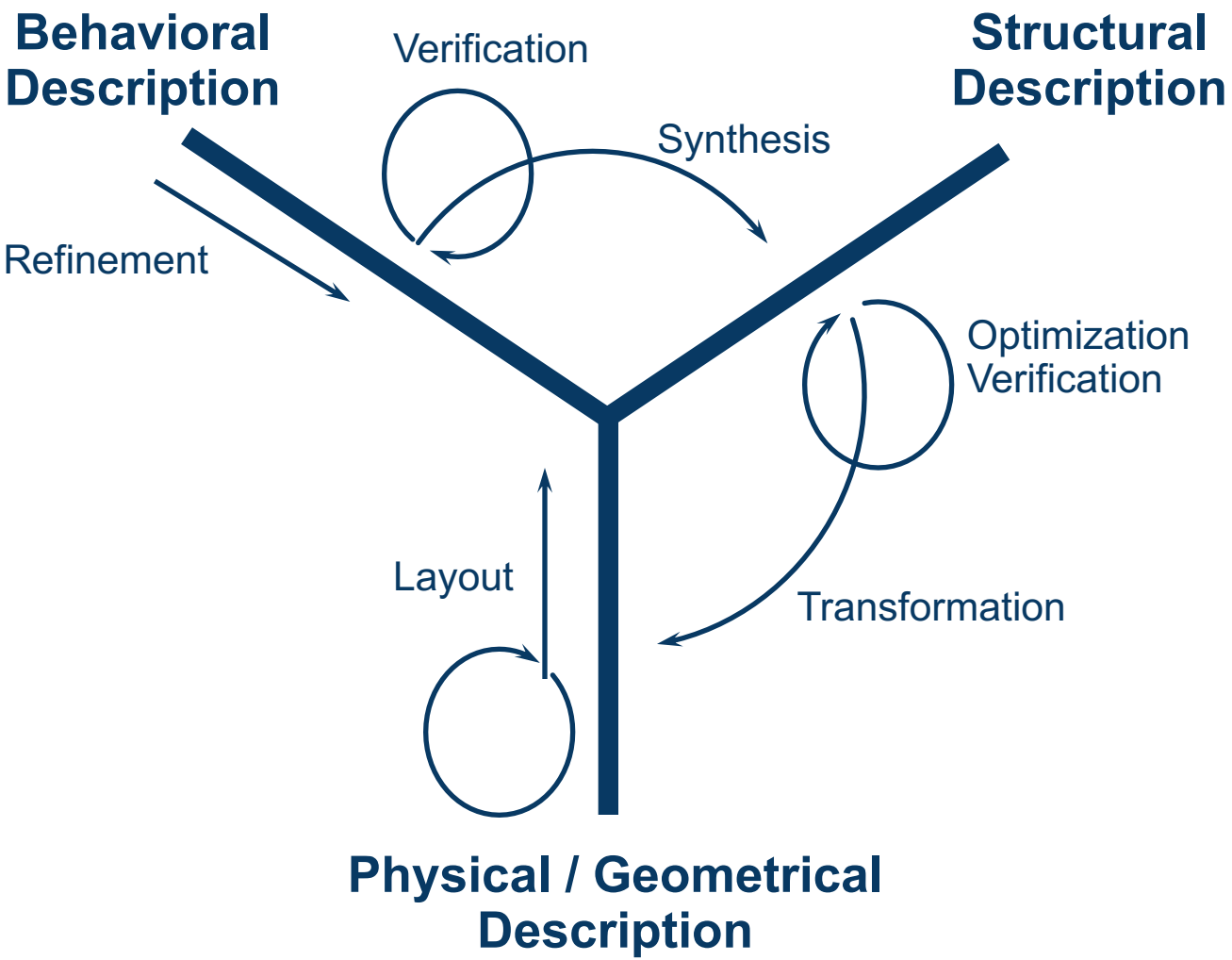
  - Description Methods:
    - Behavior: Ordinary Nonlinear Differential Equations
    - Structure: Transistors, Resistors, Capacitors, Wires, Current/Voltage Sources
    - Geometry: Polygons

Model of the Design Process

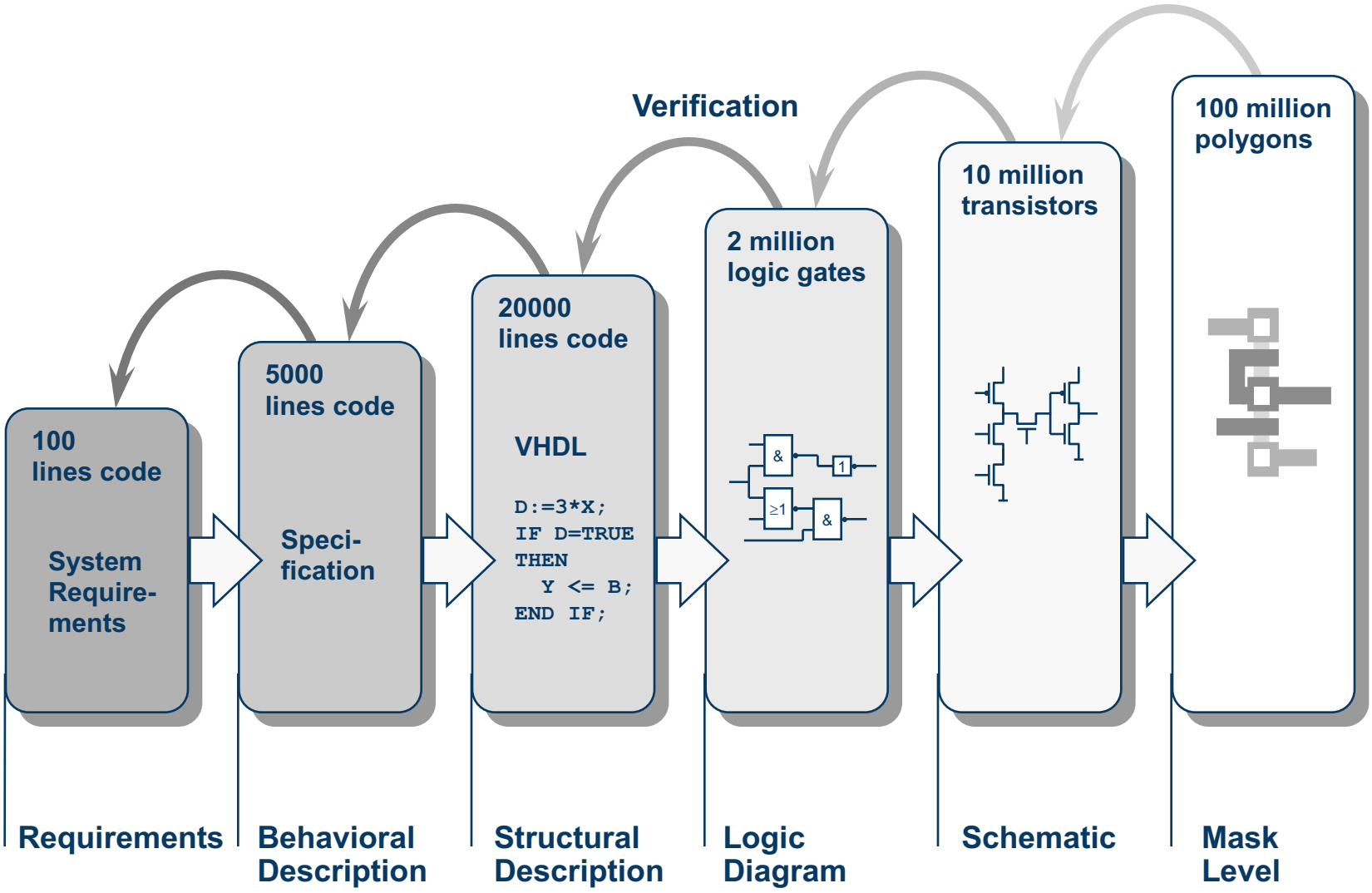


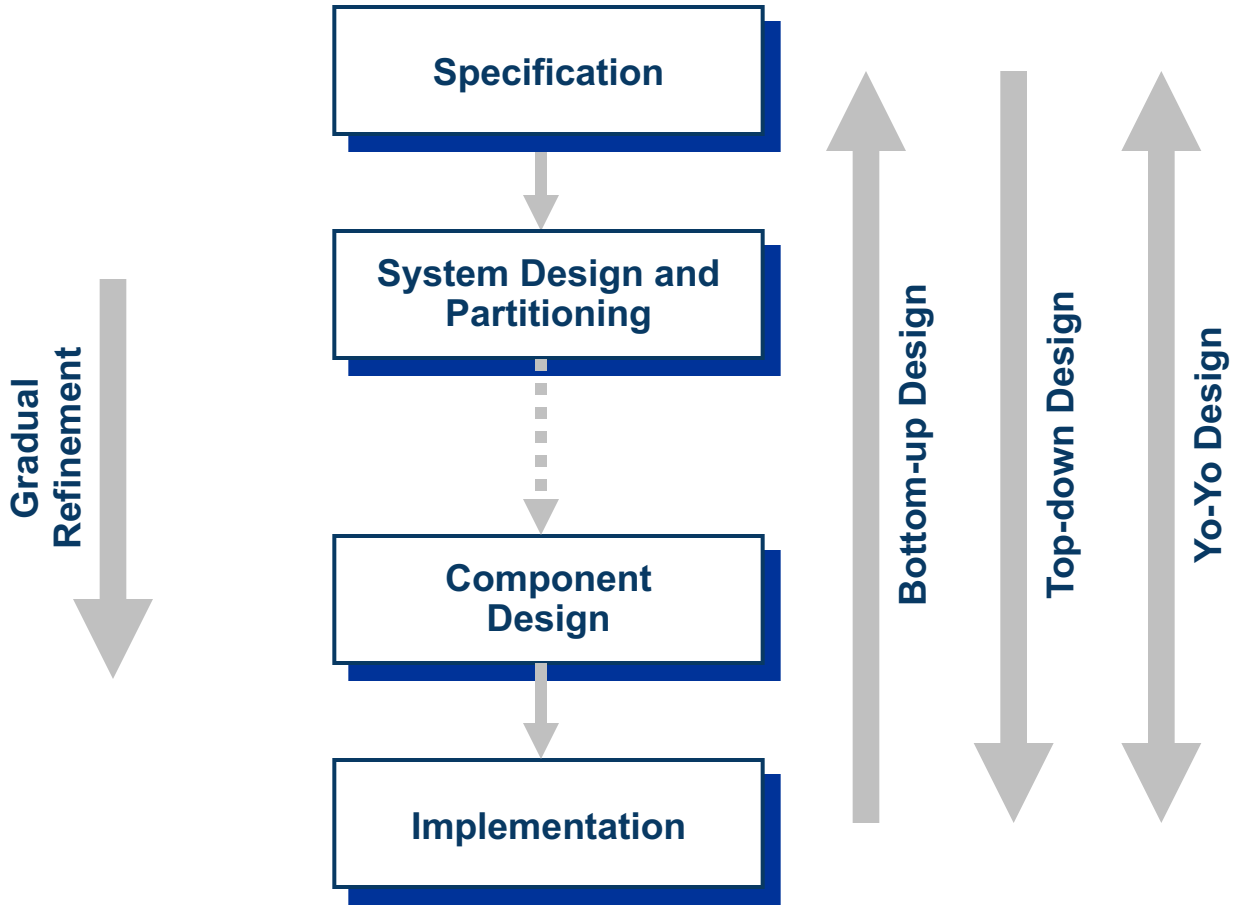
Model of the Design Process




Behavioral Description
Verification
Structural Description
Synthesis
Refinement
Optimization
Verification
Layout
Transformation
Physical / Geometrical Description

Design Complexity

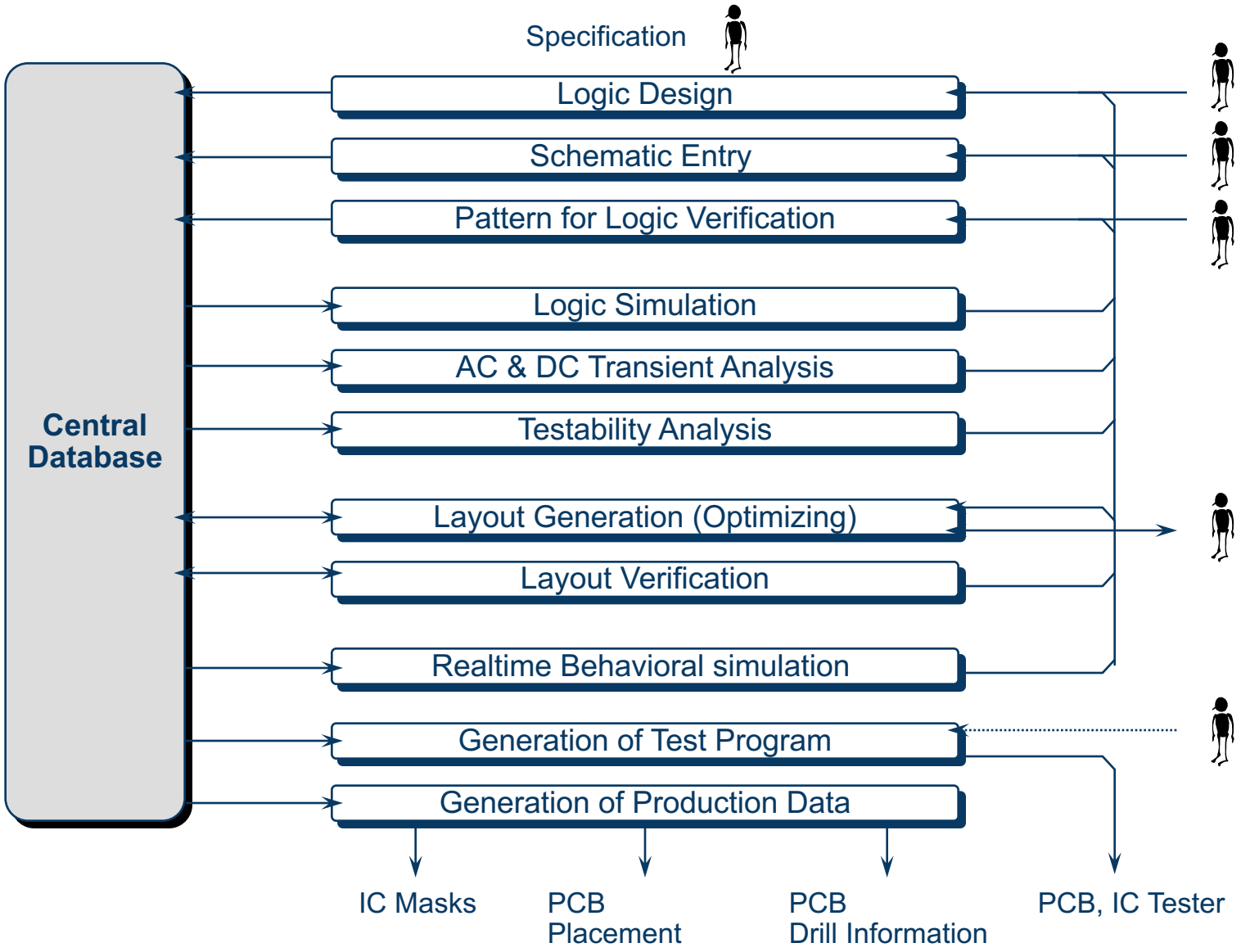



Specification
System Design and Partitioning
Component Design
Implementation
Gradual Refinement
Bottom-up Design
Top-down Design
Yo-Yo Design

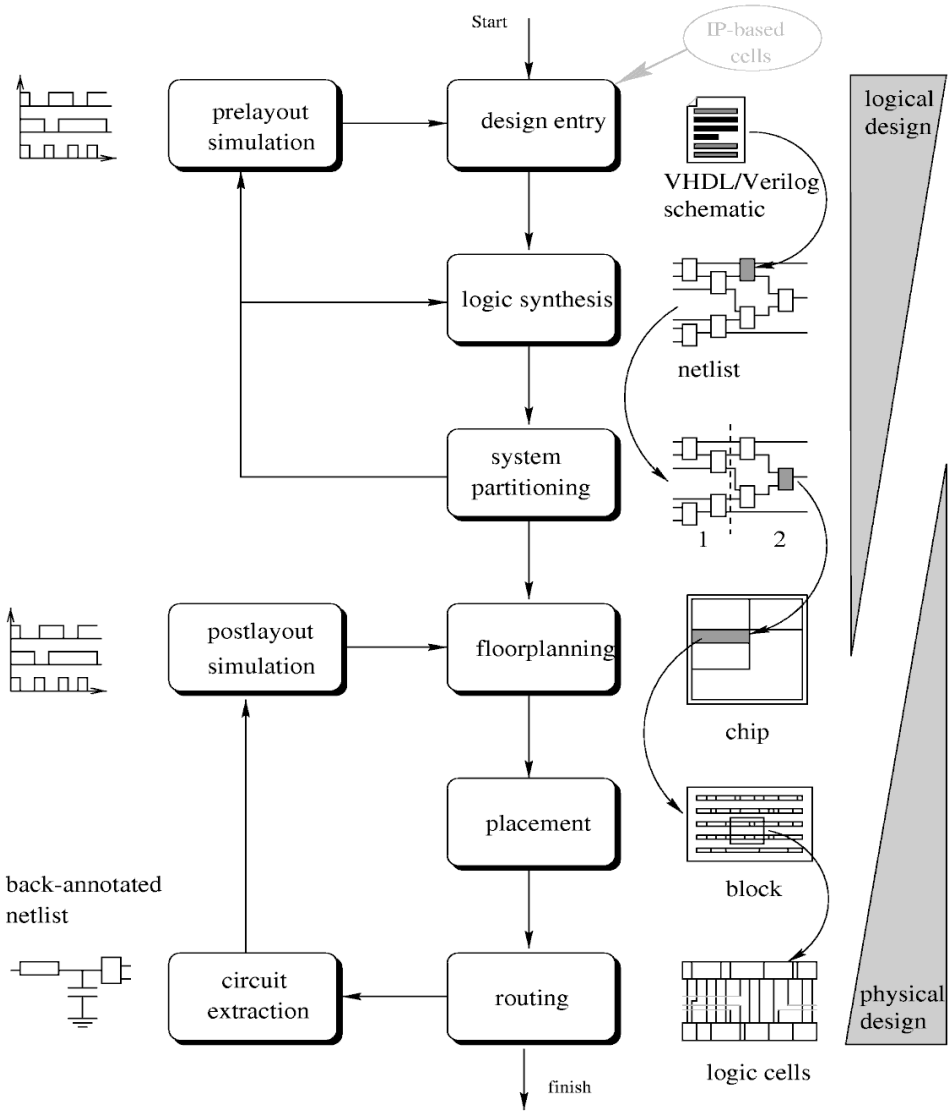
Phase Model for System Design

Phase	Subjects	Result
Study	Brainstorming, Market analysis, Realization analysis, Check for principle realization	Definition of the requirements General description Rough cost / time estimate
Concept	Behavior oriented description, Defining the function as a goal, Defining test strategy	Task manual Behavioral specification
Design	Architecture development, Hardware /Software tradeoff, Technical description of the development, Detail specification, test methods	Target specification Architecture specification Test specification
Implementation	Refinement Encoding, Manual design and synthesis, Module Test	Verified components Implementation documentation Integration specification
Integration and Test	Integration of modules Verification of subsystems Verification of system	Verified prototype Integration documentation
Production Commissioning	Transfer to product management Production	Cleared product
Maintenance and Service	Changes, Improvements, Upgrades	Maintenance / error protocols
Field Service	Recycling , Disposal	Lifecycle documentation

Computer Support for IC and PCB Design



Motivation: Typical ASIC Design Flow



Independent of Technology

Dependent on Technology

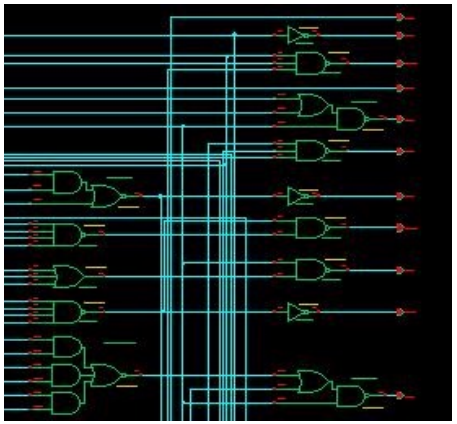
Design Methods for Current Technologies

Hardware Description

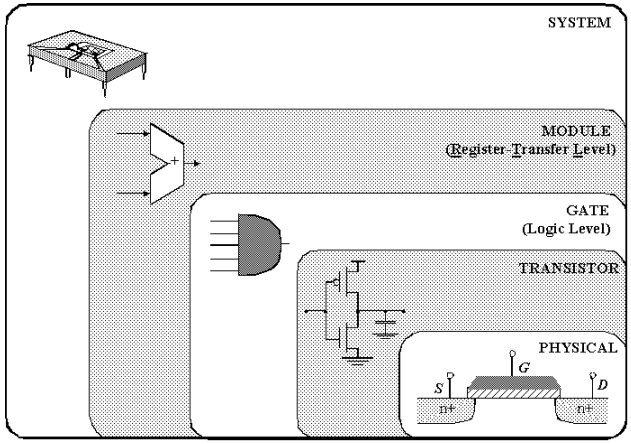
```
architecture structural of first_tap is

signal x_q,red  : std_logic_vector(bitwidth-1
downto 0);
signal mult     : std_logic_vector(2*bitwidth-1
downto 0);

begin

 delay_register:
  process(reset,clk)
  begin
   if reset='1' then
     x_q <= (others => '0');
   elsif (clk'event and clk='1') then
     x_q <= x_in;
   end if;
  end process;

 mult <= signed(coef)*signed(x_q);
```

Synthesis
(Synopsys)

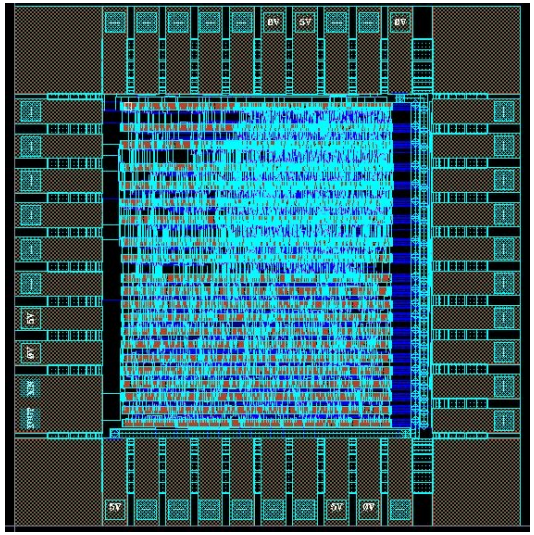
Net List



Levels of Abstraction

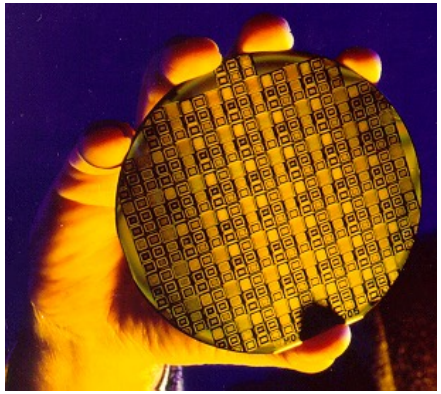


Place & Route
(Cadence/ Mentor)



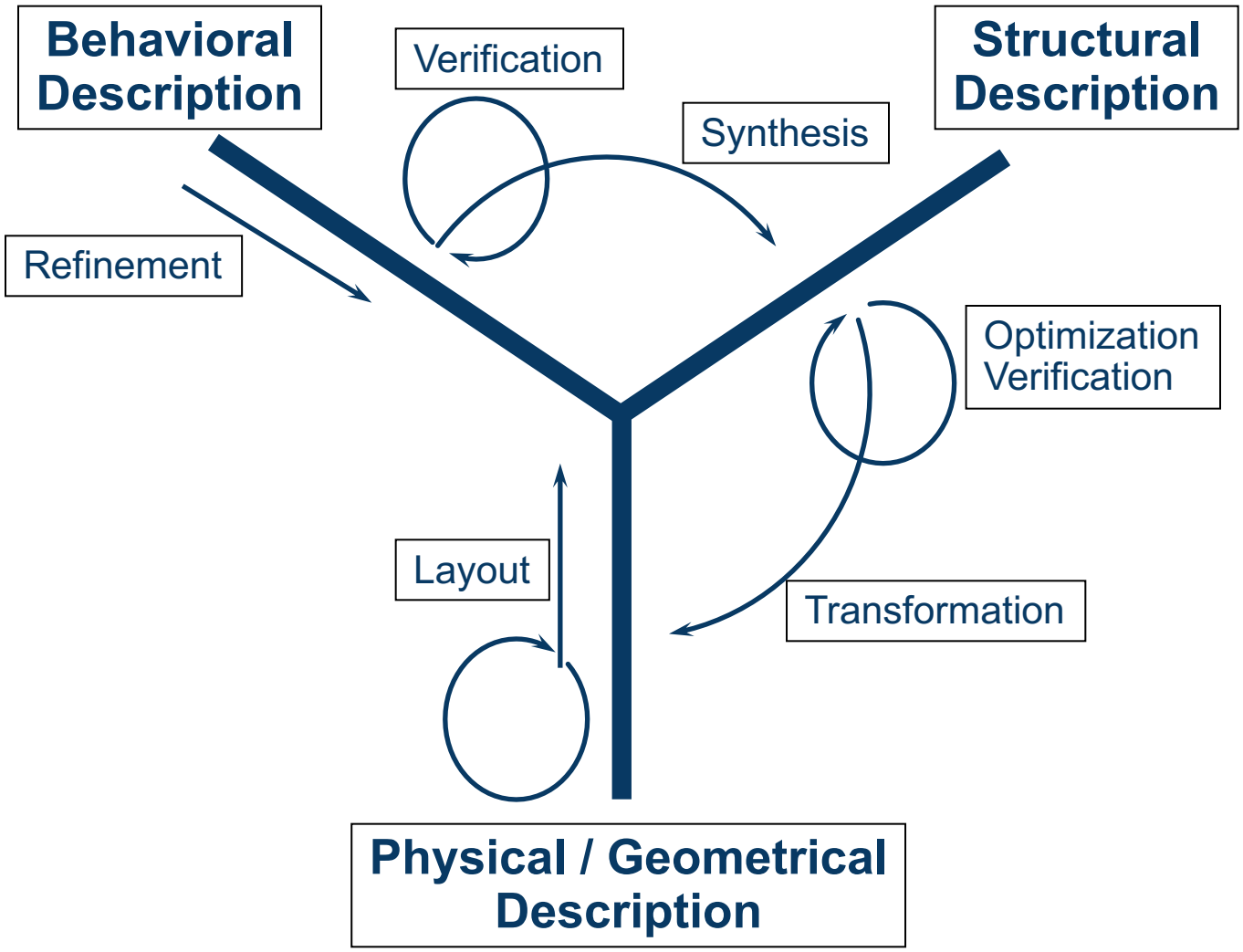
Mask Layout

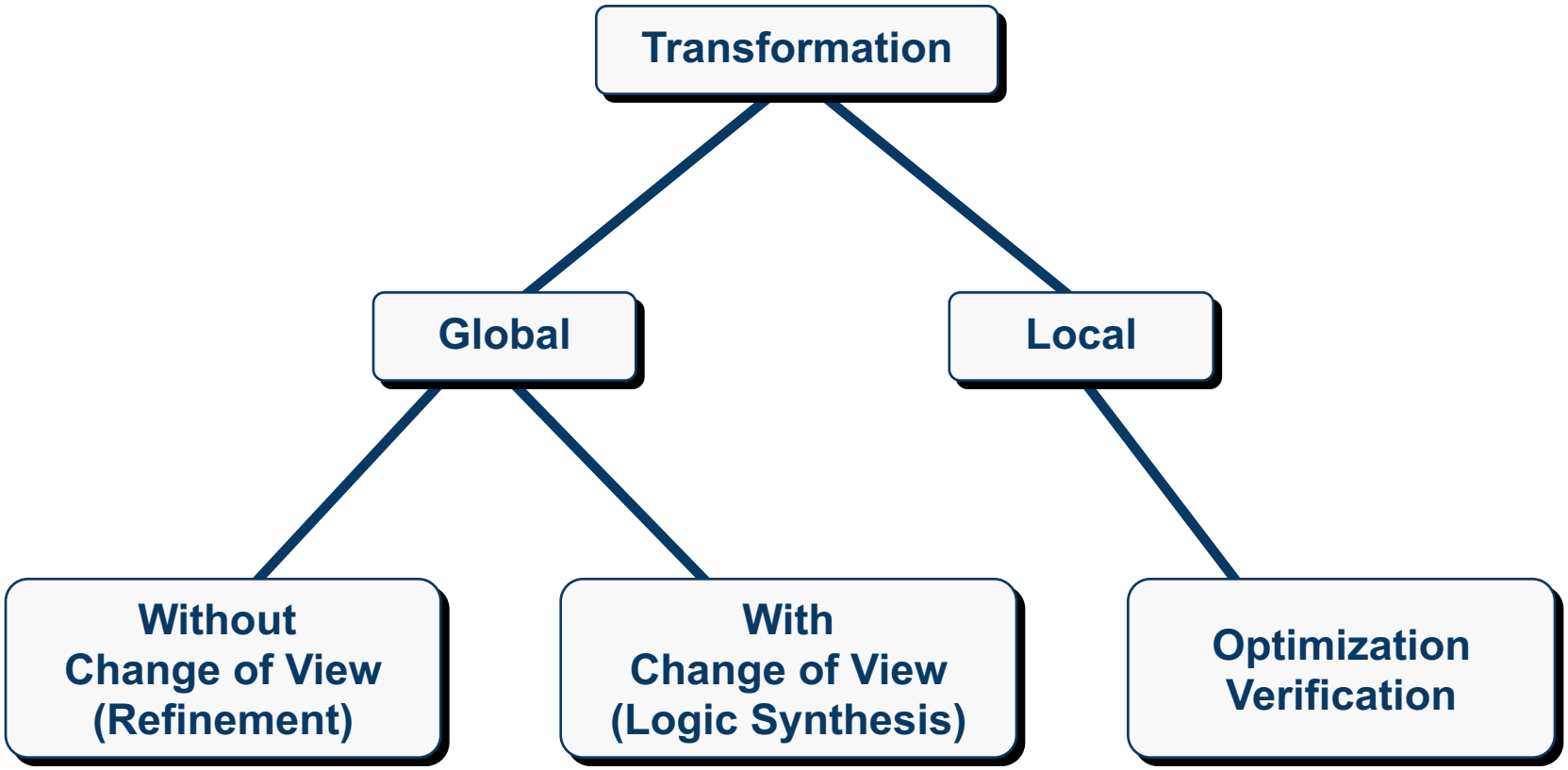
Manufacture

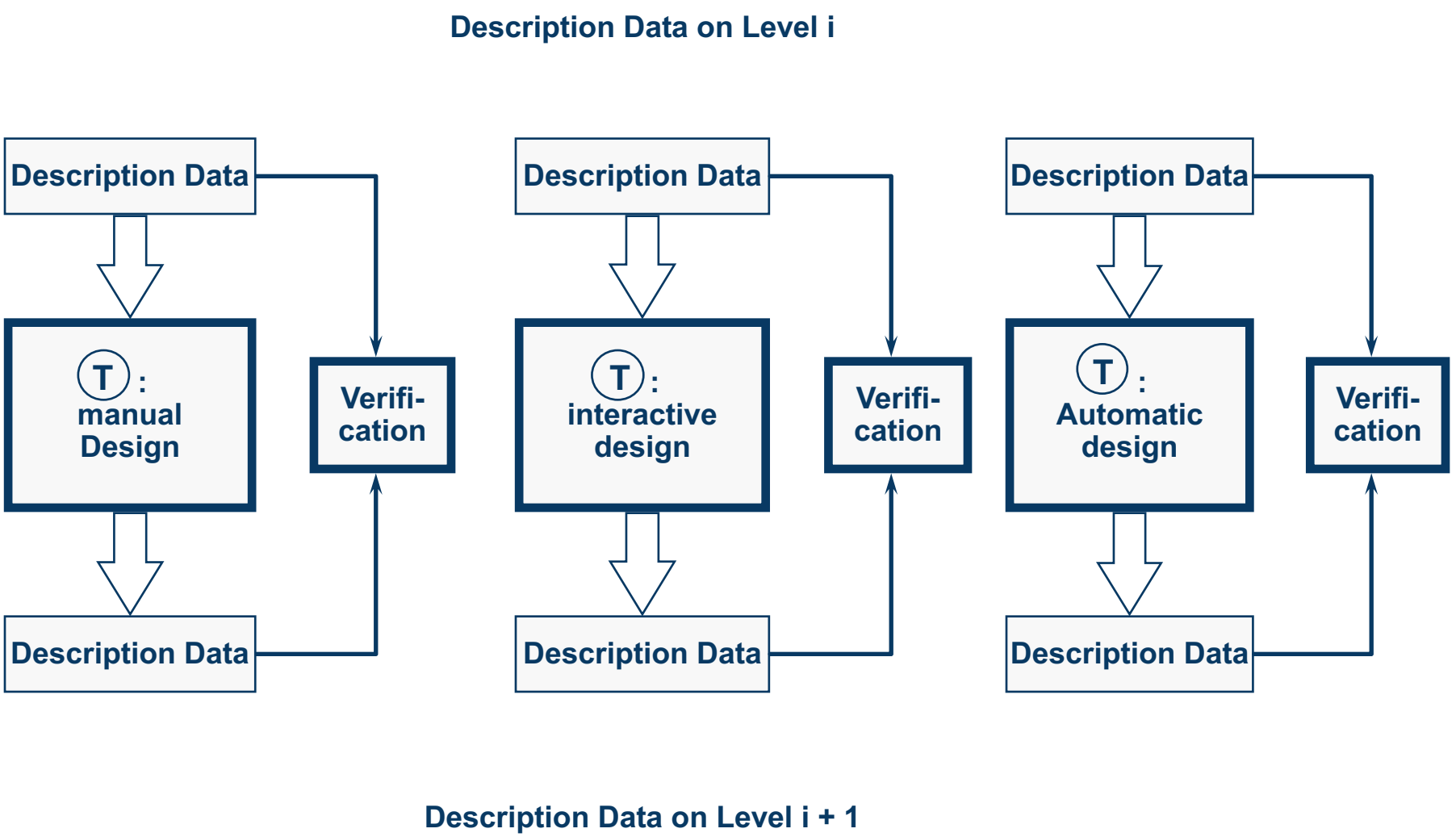


Chip Wafer

Design Process




Transformation
Global
Local
Without Change of View (Refinement)
With Change of View (Logic Synthesis)
Optimization Verification


Description Data on Level i
Description Data
T : manual Design
Verifi- cation
Description Data
Description Data
T : interactive design
Verifi- cation
Description Data
Description Data
T : Automatic design
Verifi- cation
Description Data
Description Data on Level i + 1

Development of Electronic Systems

=

Sequence of Design Steps

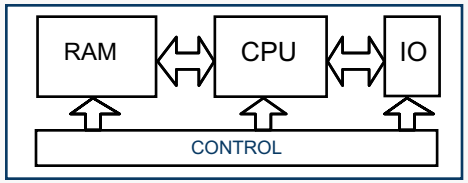
(Analysis and Synthesis, creative, manual and automatic)

and

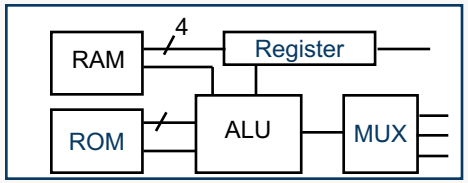
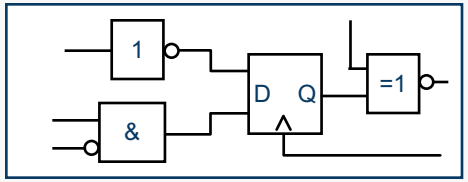
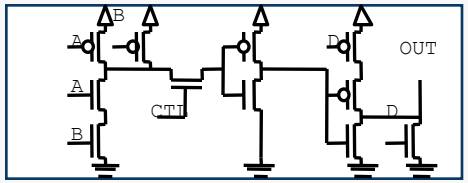
Verification Cycles

Abstraction, Hierarchy, Modularization

Abstraction

1	Architecture		Modulari-zation

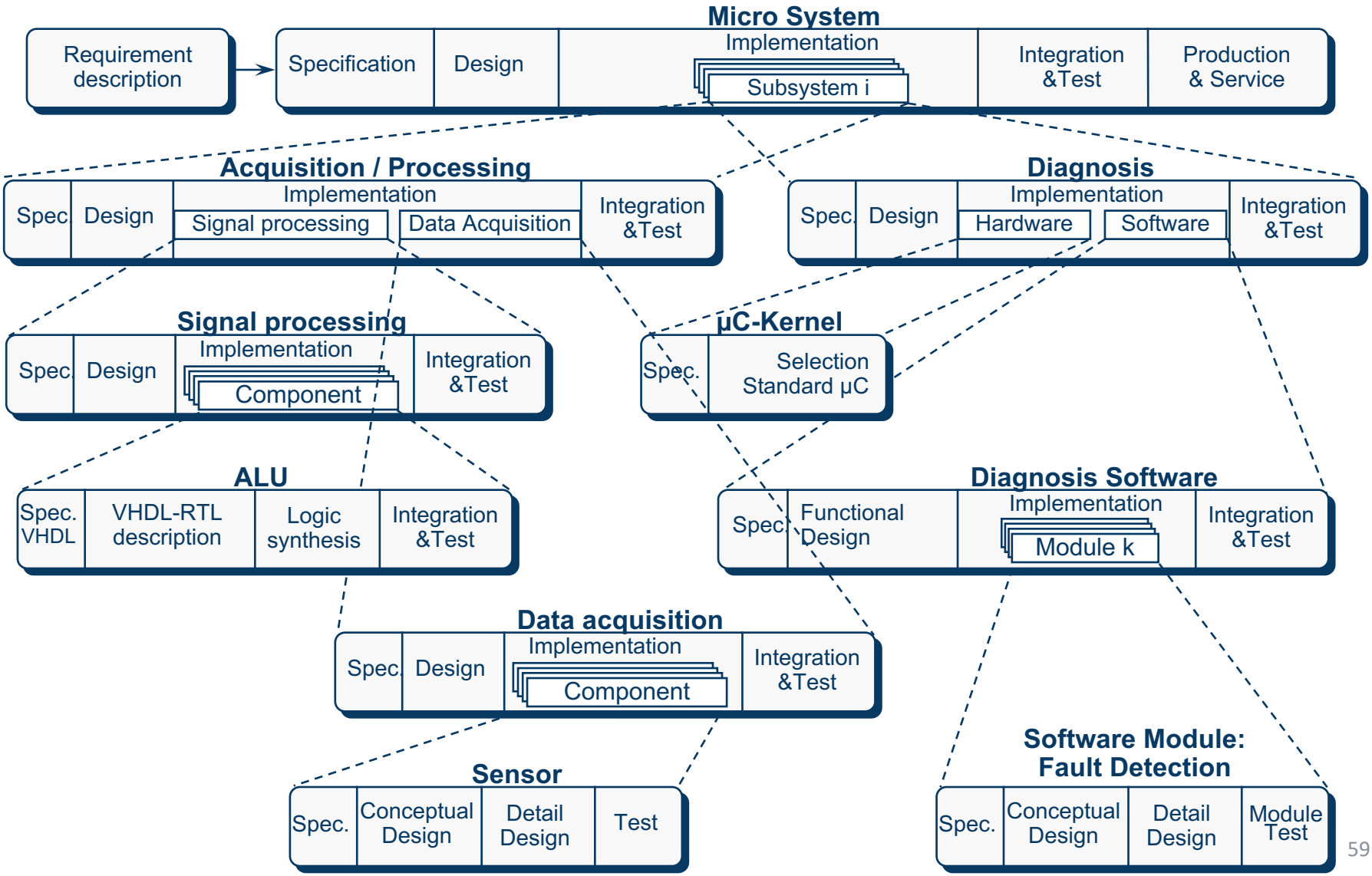
2	Algorithm	`A := 3*B+C`
		`IF (D=TRUE) THEN`
		`  A := A + 1`
		`ELSE A := A - 1`
		`ENDIF`
			Modulari-zation

3	Register Transfer		Modulari-zation
4	Logic		Modulari-zation
5	Electrical Circuit		Modulari-zation

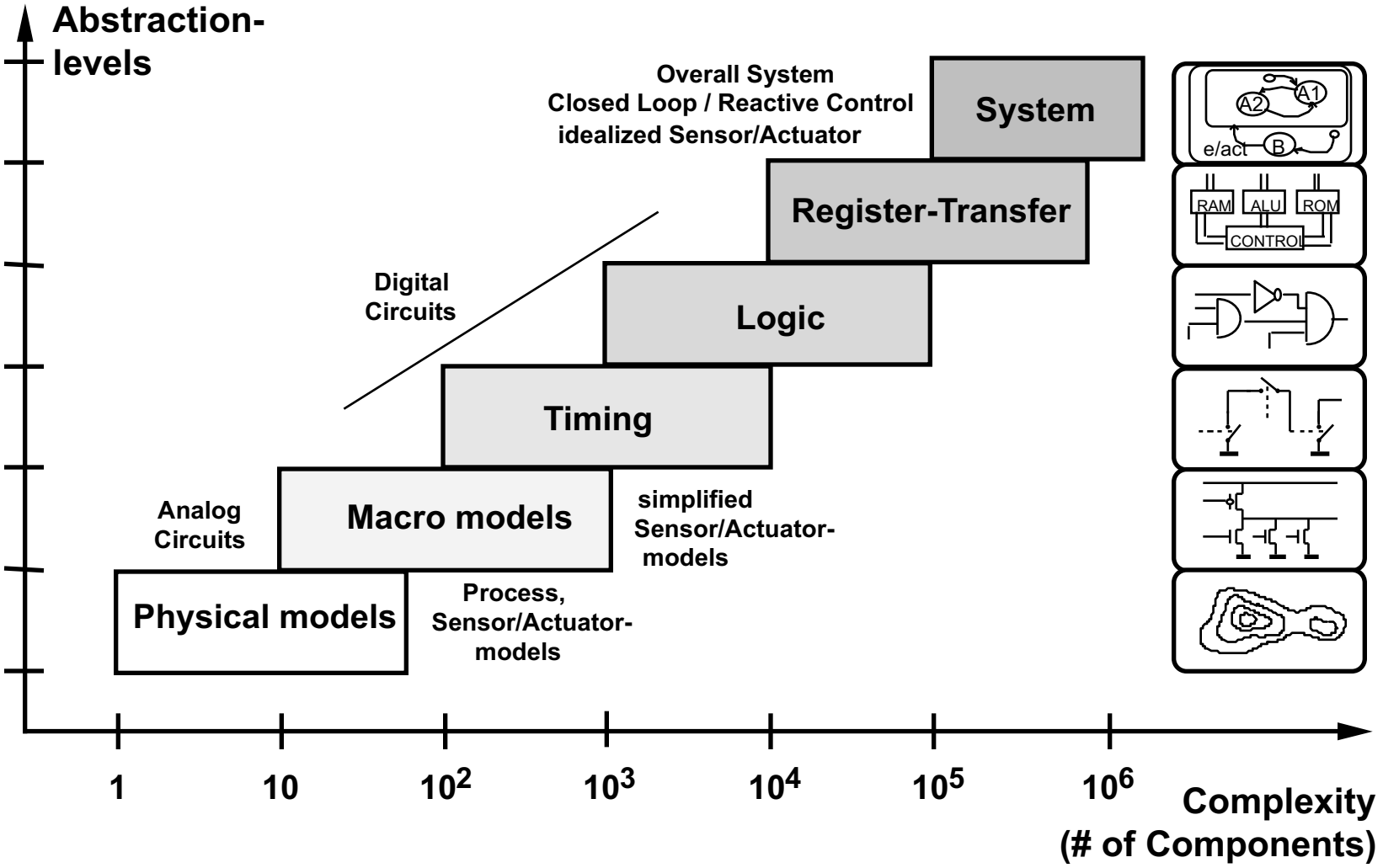
HIERARCHY

- Abstraction is applied both Top-Down and Bottom-Up
  - Example:
    - Abstraction top-down:
      No separate data path and control path after switching from register-transfer level to logic level.
    - Abstraction bottom-up:
      Value- and time-continuous description on the electric circuit level becomes value- and time-discrete description on the logic level.
- Modularization is possible on every abstraction level
- Hierarchy (concept orthogonal to abstraction)
  - A hierarchical description may comprise different abstraction levels.
  - A hierarchical description can be applied to one distinct abstraction level.

Hierarchy of the System Design

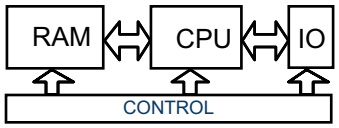


Abstraction Levels for Modeling and Simulation



Synthesis across Abstraction Levels

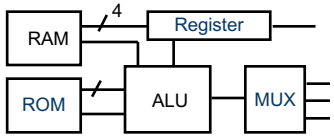
Architecture specification



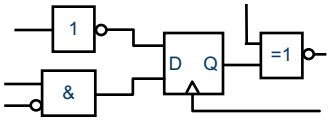
Algorithmic specification

```
A := 3*B+C
IF (D=TRUE) THEN
  A := A + 1
ELSE A := A - 1
ENDIF
```

Block Diagram (independent of manu-facturer and technology)



Schematic Net list (independent of manu-facturer and technology)



Implementation

Architecture Synthesis
- Partitioning
- Parallel processes
- Analysis

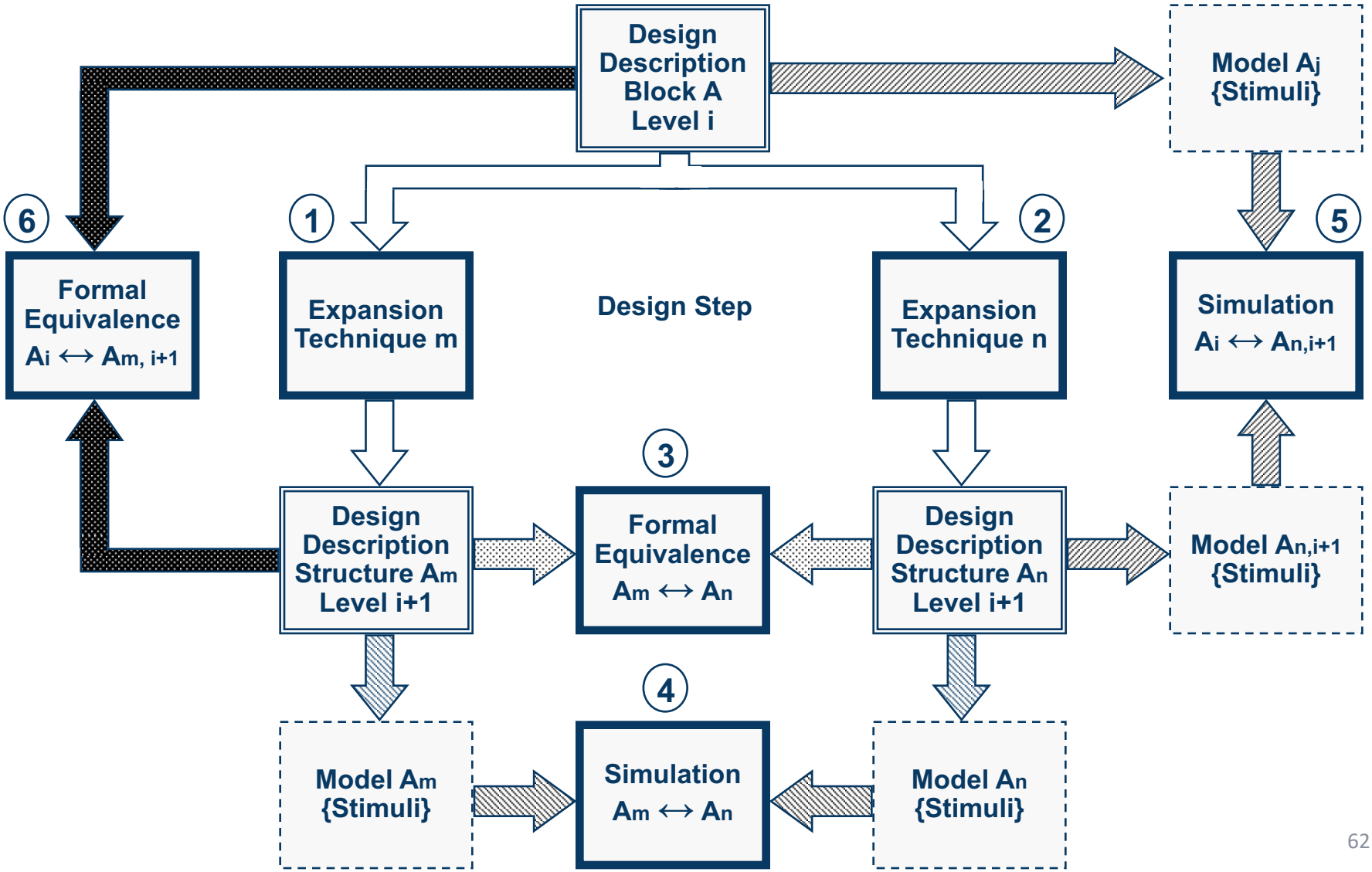
Algorithmic Synthesis
- Synthesis of digital circuits
- Optimization
- Design alternatives (serial / parallel)

Net list optimization
- Inclusion of Libraries
- Refining the structural description

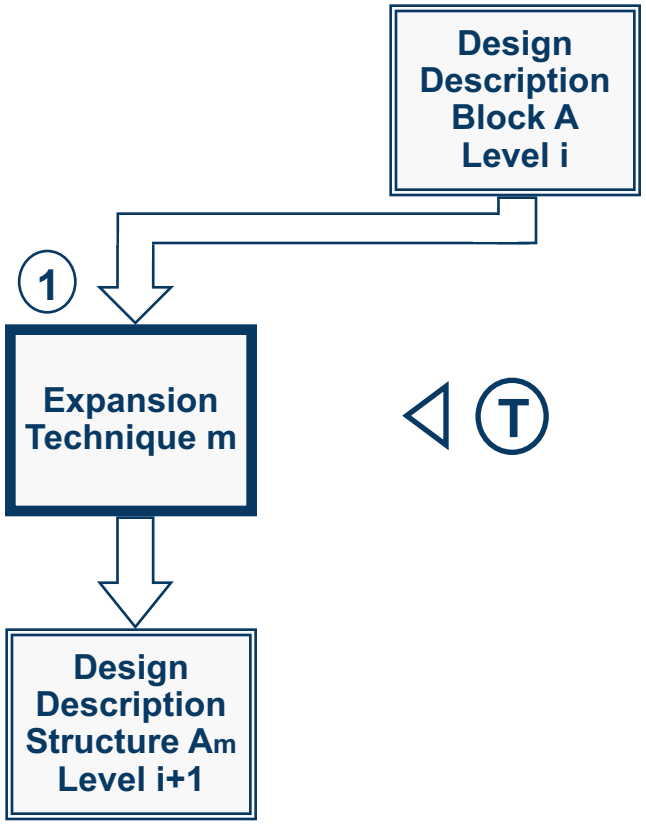
Conventional CAD-System

Standard ICs PCBs	Standard cells Gate Arrays

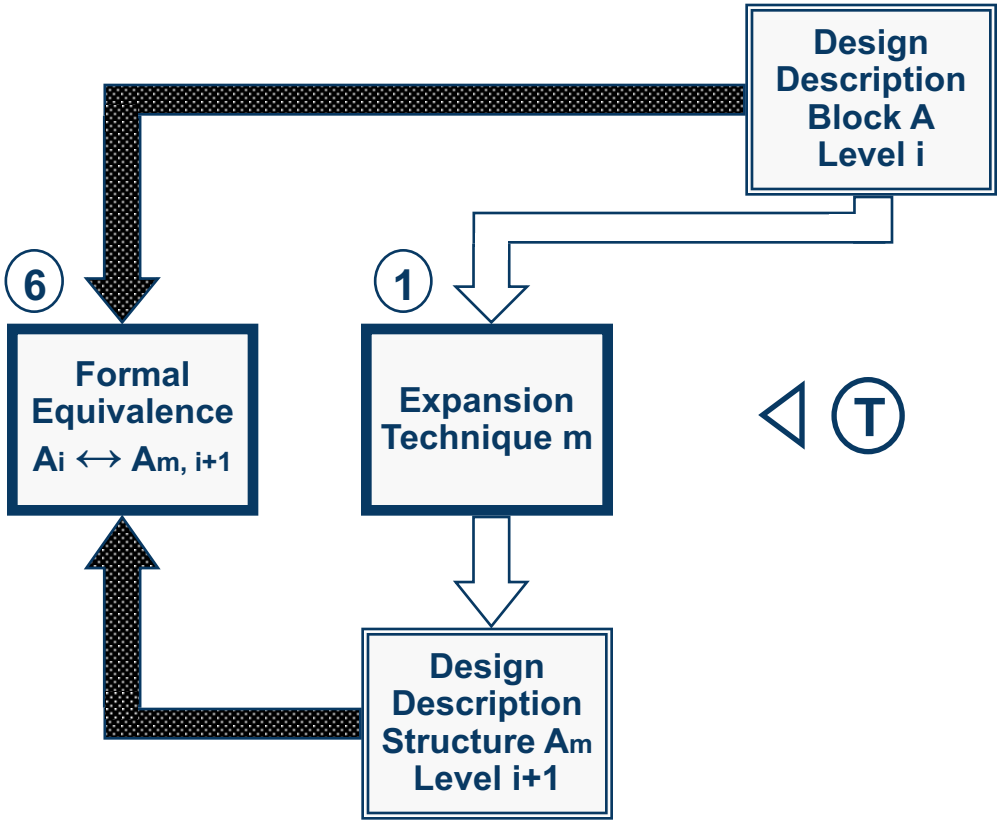
Verification Methods

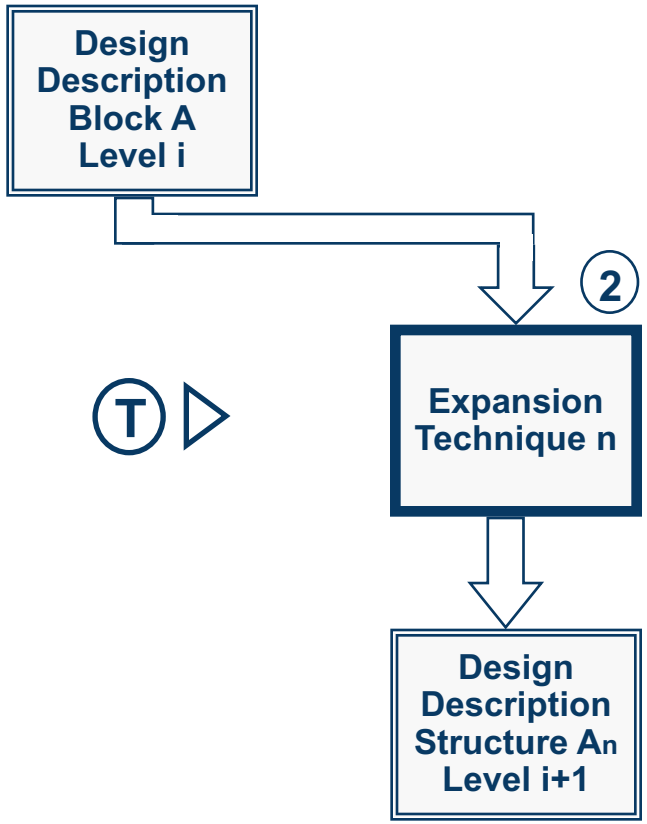


Verification Methods

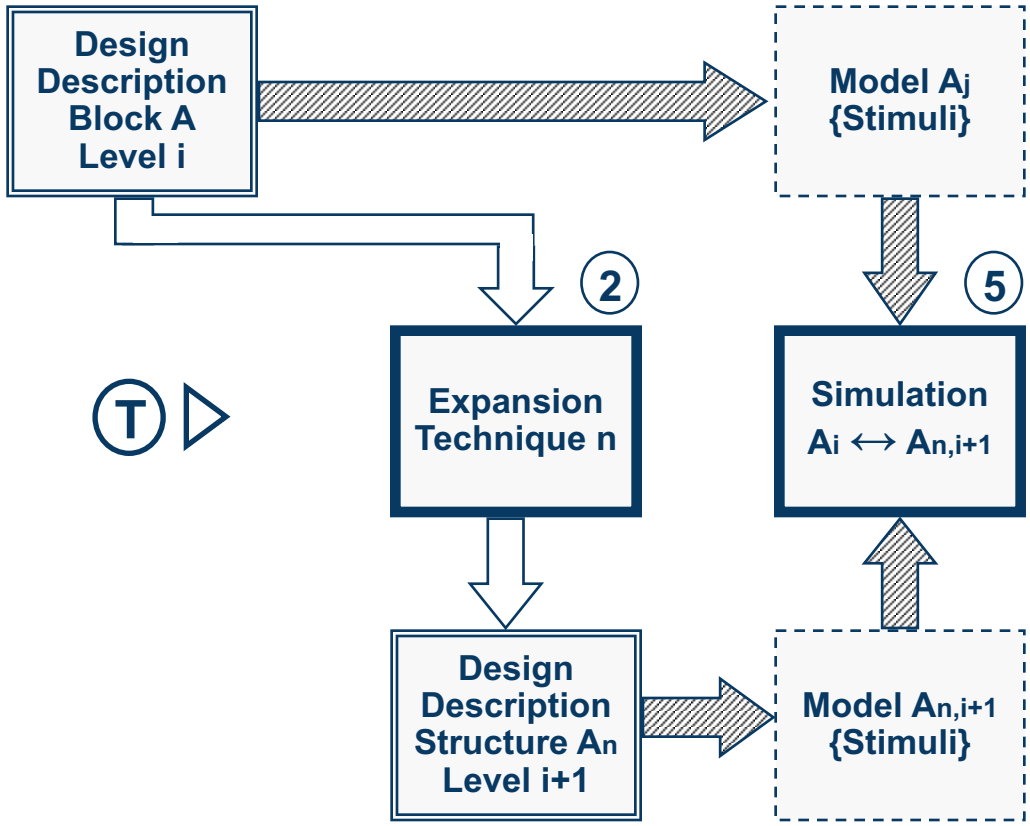


Verification Methods

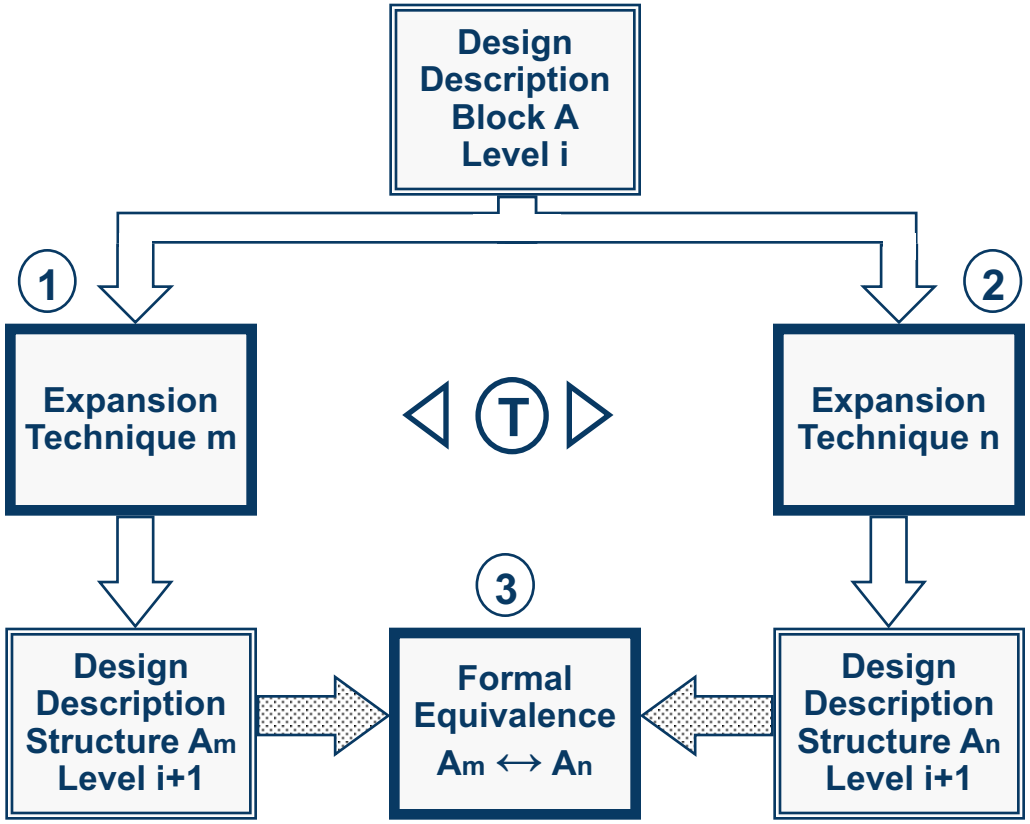


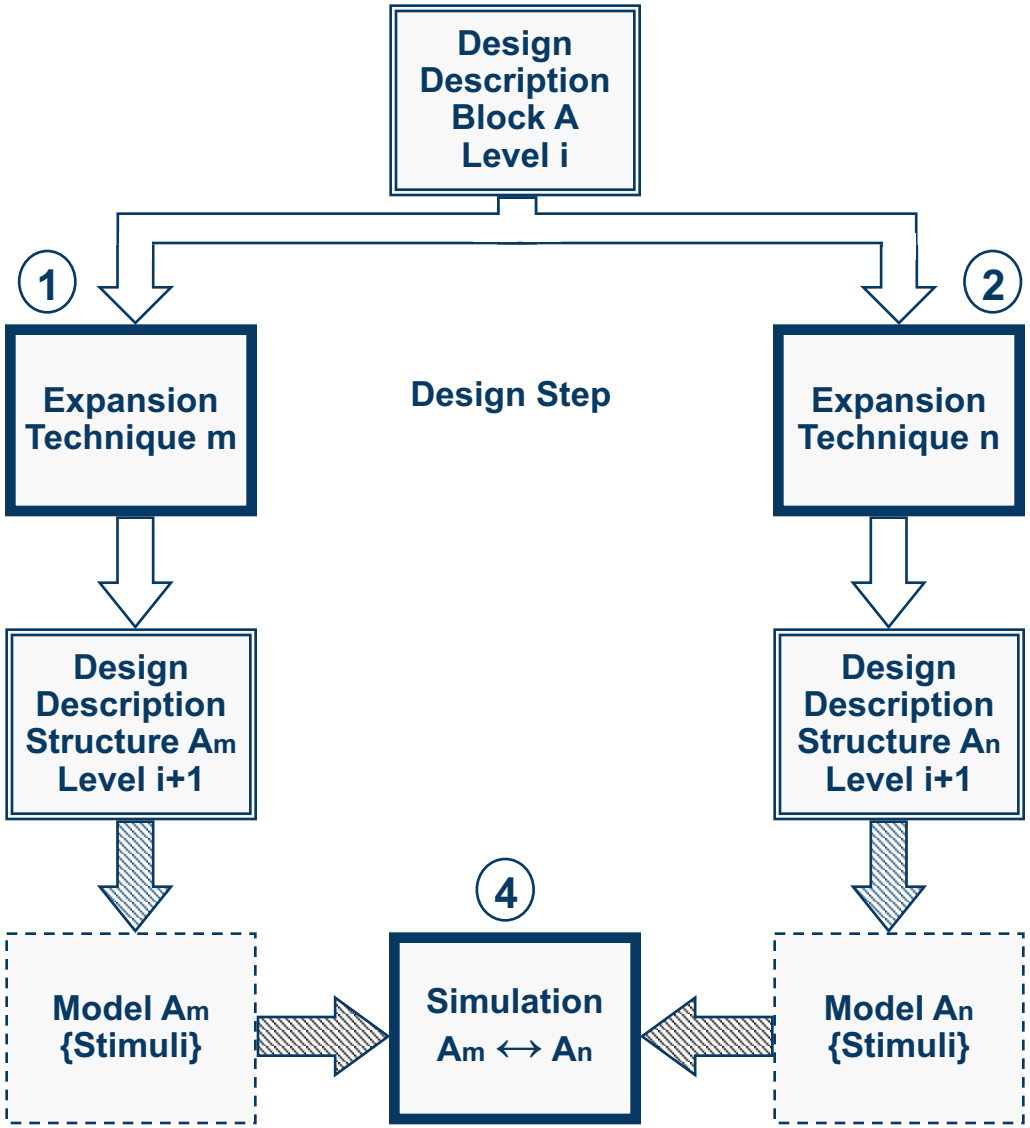

Design Description Block A Level i
2
T
Expansion Technique n
Design Description Structure An Level i+1

Verification Methods

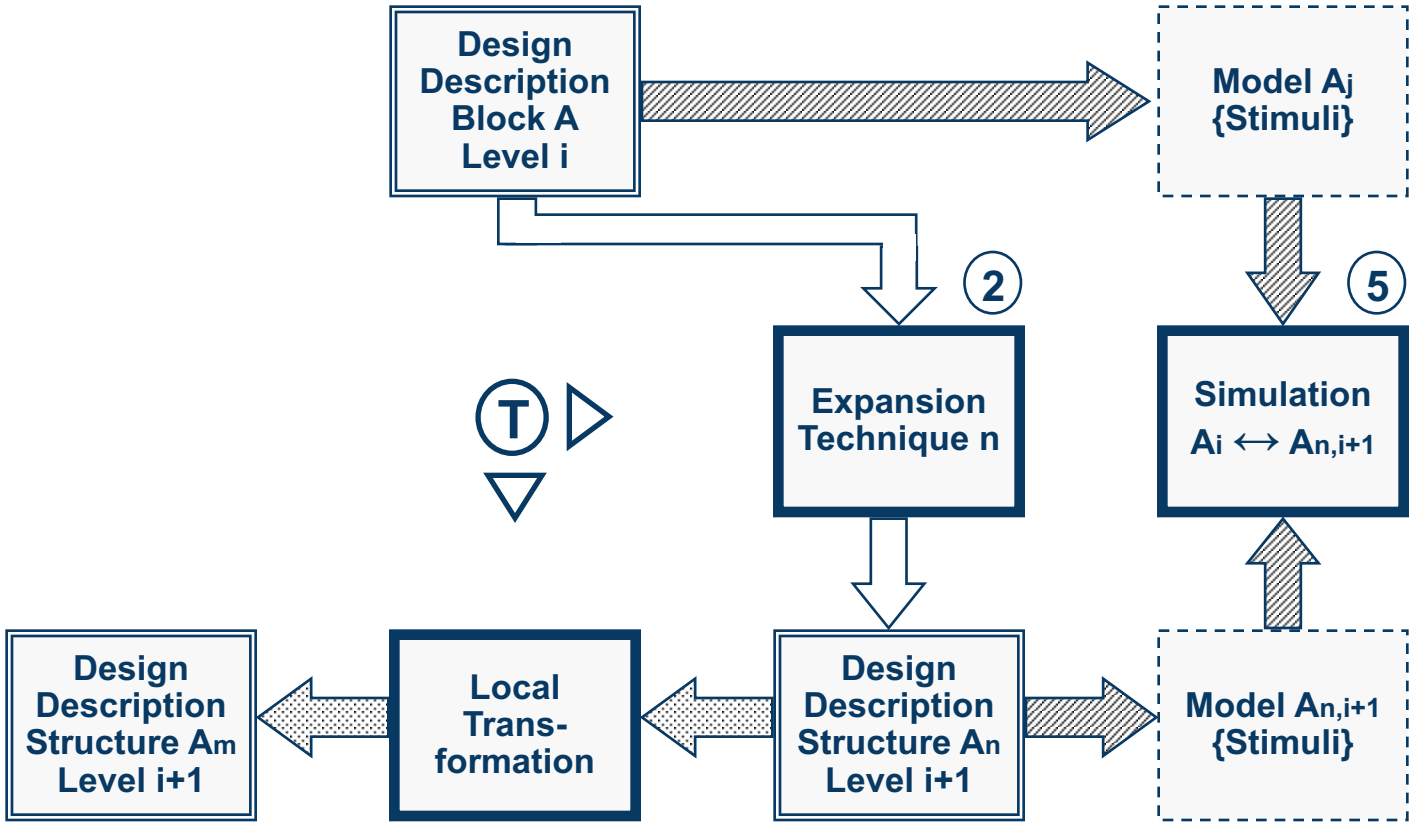


Verification Methods

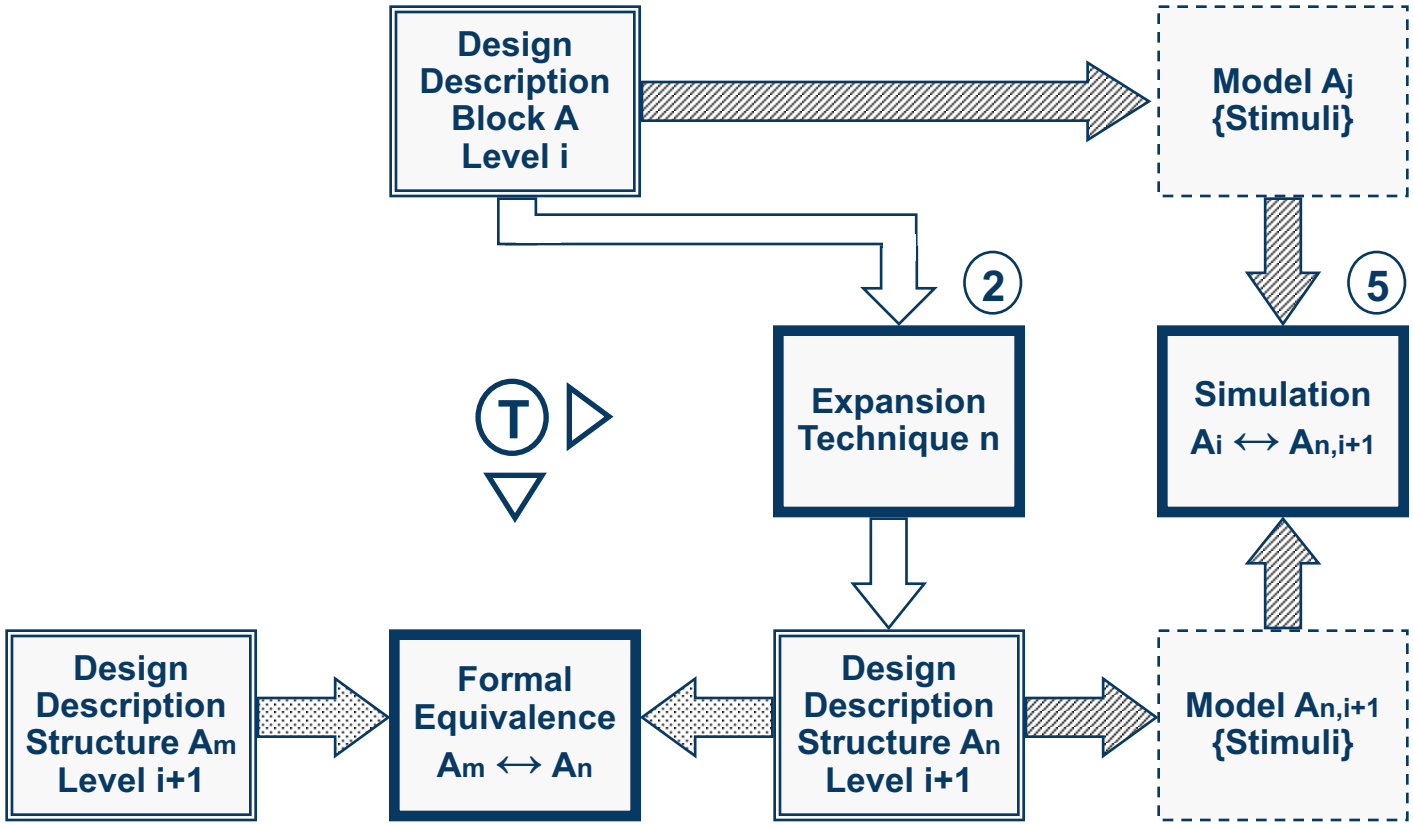



Design Description Block A Level i
1
2
Expansion Technique m
Design Step
Expansion Technique n
Design Description Structure Am Level i+1
Design Description Structure An Level i+1
4
Model Am {Stimuli}
Simulation Am ↔ An
Model An {Stimuli}

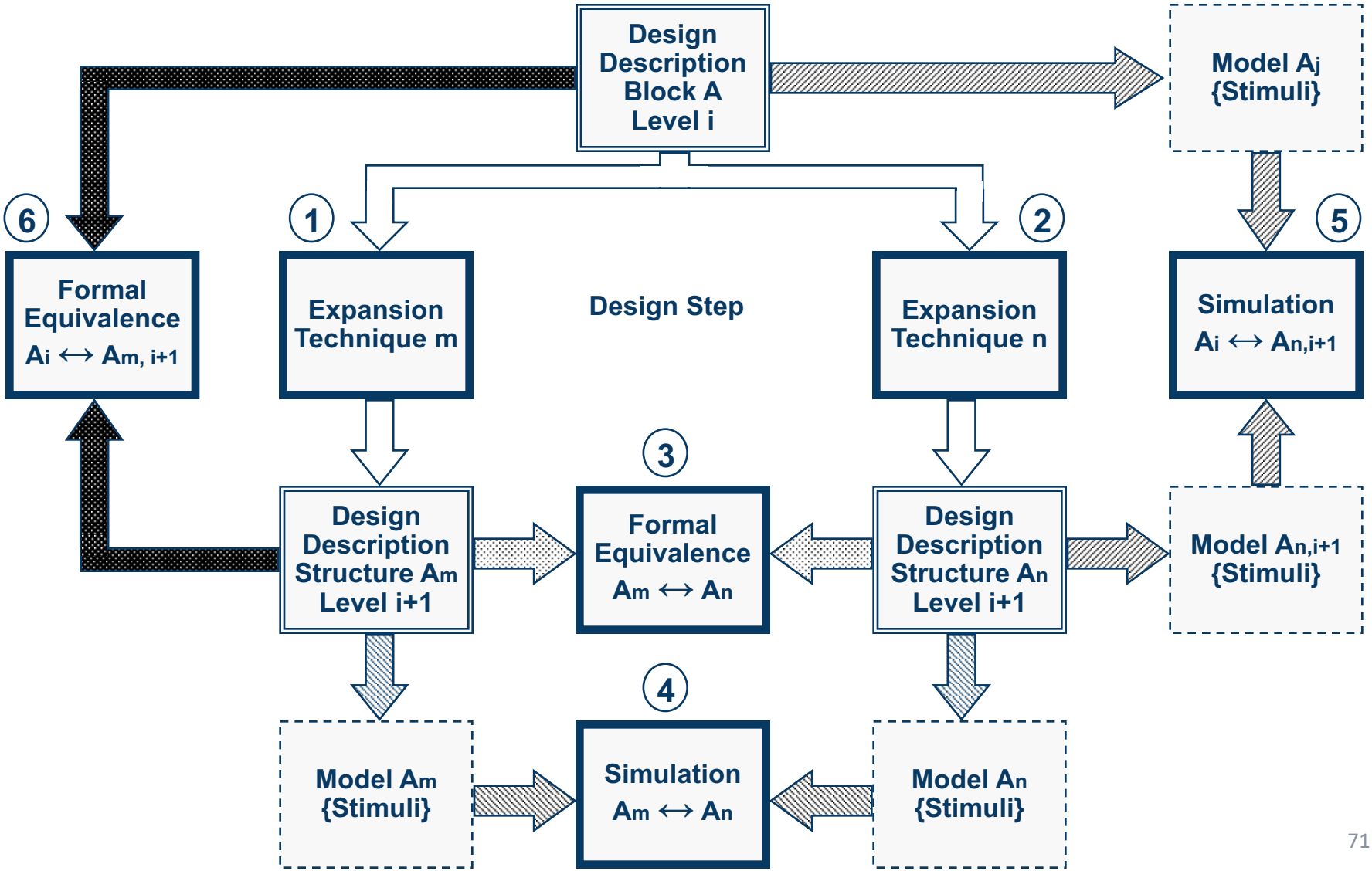
Verification Methods



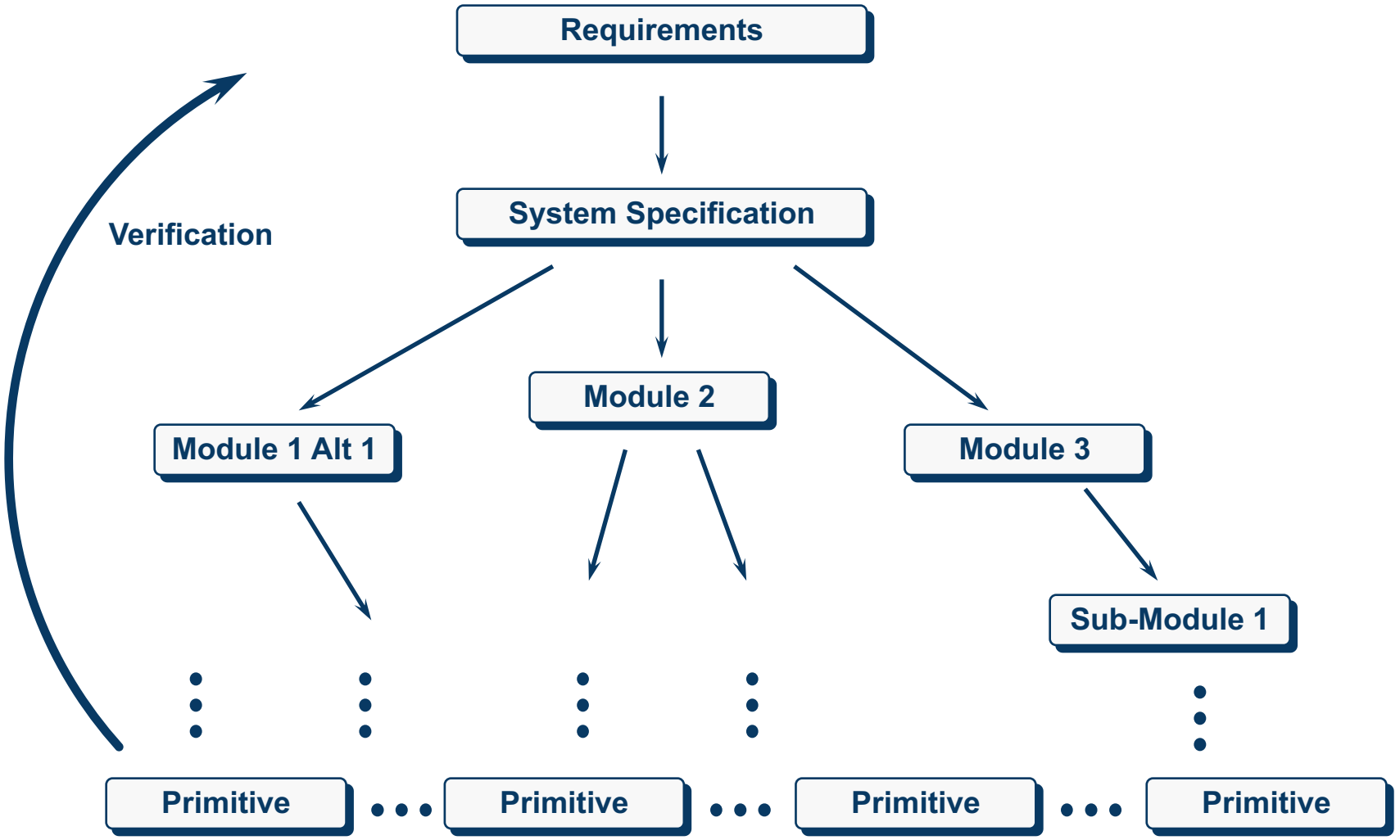
Verification Methods



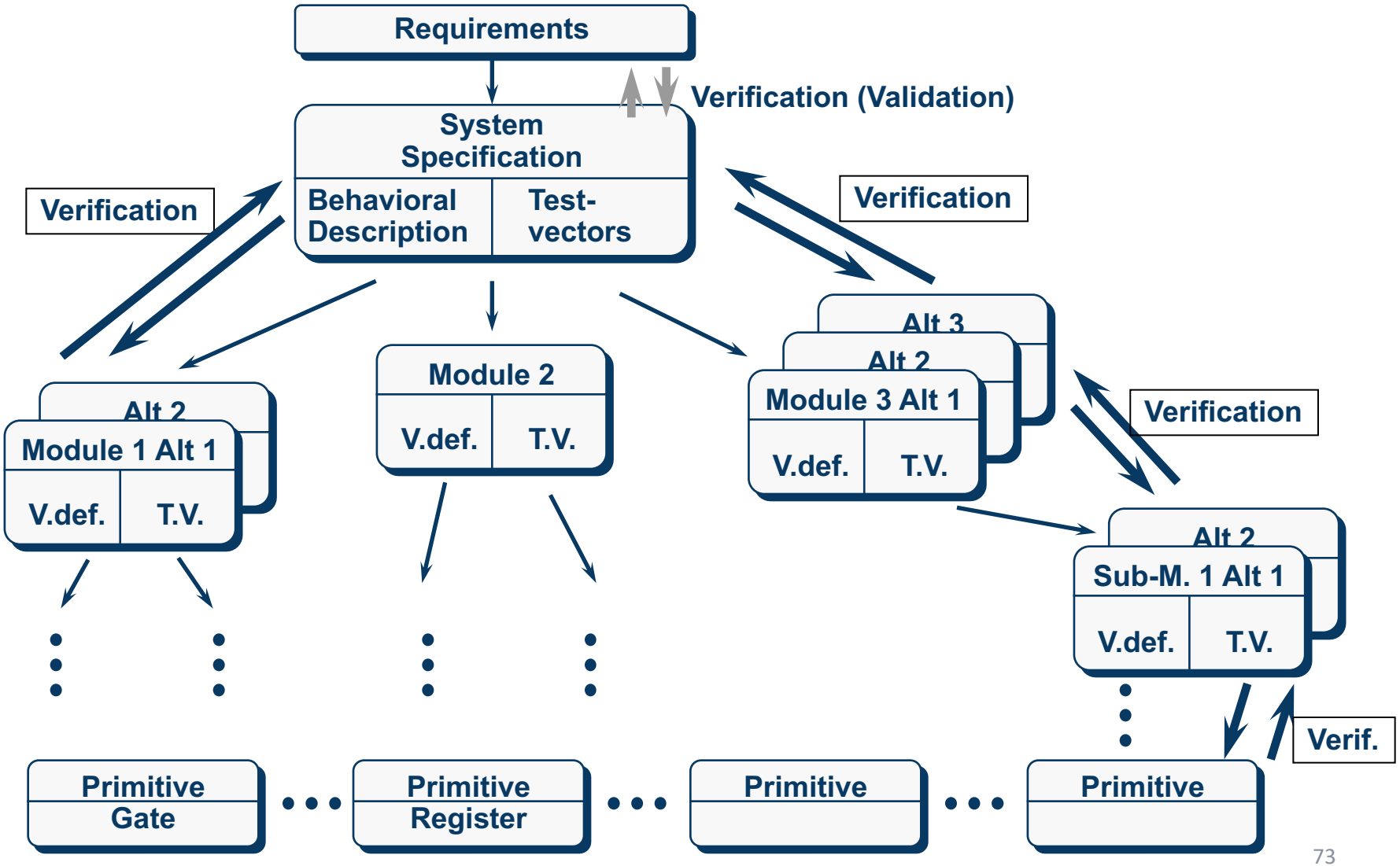
Verification Methods



Bottom-Up Design Verification



Top-Down Design Verification



Motivation for HDL

- Complexity
  - About every three years: complexity * 4
  - Today: typically >> 100 000 gates/chip
  - Computer aided design necessary as manual design is no longer possible
- Computer aided design tools in various areas:
  - Specification
  - Simulation
    - Prototype production is too expensive and time-consuming
    - Early verification is only possible through simulation
  - Synthesis
    - Given current design complexity, manual design is no longer possible
  - Machine-readable description for specification, simulation and synthesis is necessary

- Data exchange
  - Between customer and developer (requirements specification, target specification)
  - During development between
    - Team members
    - Different tools
    - Different computer systems
  - Between development and production (production data)
  - ➠ Uniform format for data exchange is necessary
- Documentation
  - Maintenance / Service / Further development
  - Re-use of designs
  - ➠ Human readable format necessary for documentation

- System design without using an HDL:
  - System developer defines architecture and requirement specification for modules
  - Modules are developed bottom-up by different development teams, each team designing and verifying its own module
  - All modules must have been designed and connected before simulation of the system is possible
  - Comparison of specified data and characteristic data occurs very late in the development cycle
  - Specification data is generally subject to change during the design process

- System design with an HDL:
  - System developer defines architecture and functional requirement specification for modules using an HDL
  - HDL as functional specification can be simulated, therefore the system's performance can be determined at an early stage: e.g. hardware/software-tradeoff, hardware and micro-code can be developed in parallel
  - Tools for logic synthesis can be used earlier. A specification written in an HDL is no longer a reference document for the design, but actually represents the design itself. (Synthesis simplifies verification: "correct by construction")
  - Regular comparison of specified data and characteristic data is possible