Modeling levels

Peter Marwedel
TU Dortmund,
Informatik 12

Graphics: © Alexandra Nolte, Gesine Marwedel, 2003

2008/11/11

Levels of hardware modeling

Possible set of levels (others exist)

- System level
- Algorithmic level
- Instruction set level
- Register-transfer level (RTL)
- Gate-level models
- Switch-level models
- Circuit-level models
- Device-level models
- Layout models
- Process and device models

System level

- Term not clearly defined.
- Here: denotes the entire embedded system,
  system into which information processing is embedded, and possibly also the environment.
- Models may include mechanics + information processing.
  May be difficult to find appropriate simulators.
  Solutions: VHDL-AMS, SystemC or MATLAB.
  MATLAB+VHDL-AMS support partial differential equations.
- Challenge to model information processing parts of the system such that the simulation model can be used for the synthesis of the embedded system.

Algorithmic level

- Simulating the algorithms that we intend to use within the embedded system.
- No reference is made to processors or instruction sets.
- Data types may still allow a higher precision than the final implementation.
- If data types have been selected such that every bit corresponds to exactly one bit in the final implementation, the model is said to be **bit-true**.
  non-bit-true $\rightarrow$ bit-true should be done with tool support.
- Single process or sets of cooperating processes.

Algorithmic level: Example:
-MPEG-4 full motion search -

```
for (z=0; z<20; z++)
 for (x=0; x<36; x++) {x1=4*x;
  for (y=0; y<49; y++) {y1=4*y;
   for (k=0; k<9; k++) {x2=x1+k-4;
    for (l=0; l<9; ) {y2=y1+l-4;
     for (i=0; i<4; i++) {x3=x1+i; x4=x2+i;
      for (j=0; j<4;j++) {y3=y1+j; y4=y2+j;
       if (x3<0 || 35<x3||y3<0||48<y3)
        then_block_1; else else_block_1;
       if (x4<0|| 35<x4||y4<0||48<y4)
        then_block_2; else else_block_2;
}}}}}}
```

Instruction level

Algorithms already compiled for the instruction set.
Model allows counting the executed number of instructions.

Variations:

- Simulation only of the effect of instructions
- **Transaction-level modeling**: each read/write is one transaction, instead of a set of signal assignments
- **Cycle-true simulations**: exact number of cycles
- **Bit-true simulations:** simulations using exactly the correct number of bits

Instruction level: example

Assembler (MIPS)	Simulated semantics
`and $1,$2,$3`	`Reg[1]:=Reg[2] ∧ Reg[3]`
`or $1,$2,$3`	`Reg[1]:=Reg[2] ∨ Reg[3]`
`andi $1,$2,100`	`Reg[1]:=Reg[2] ∧ 100`
`sll $1,$2,10`	`Reg[1]:=Reg[2] << 10`
`srl $1,$2,10`	`Reg[1]:=Reg[2] >> 10`

Register transfer level (RTL)

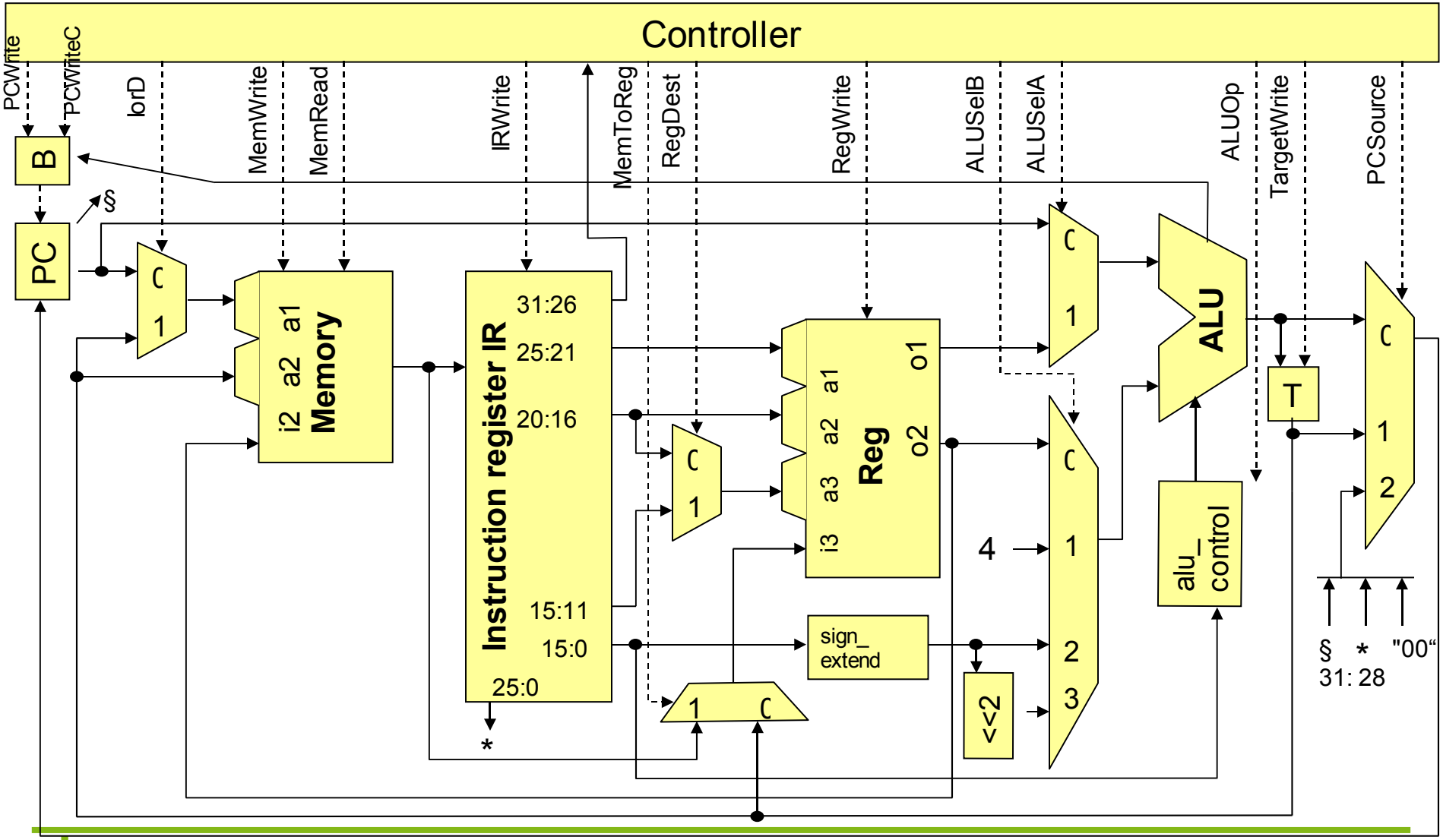
Modelling of all components at the register-transfer level, including

- arithmetic/logic units (ALUs),
- registers,
- memories,
- muxes and
- decoders.

Models at this level are always cycle-true.

Automatic synthesis from such models is **not** a major challenge.

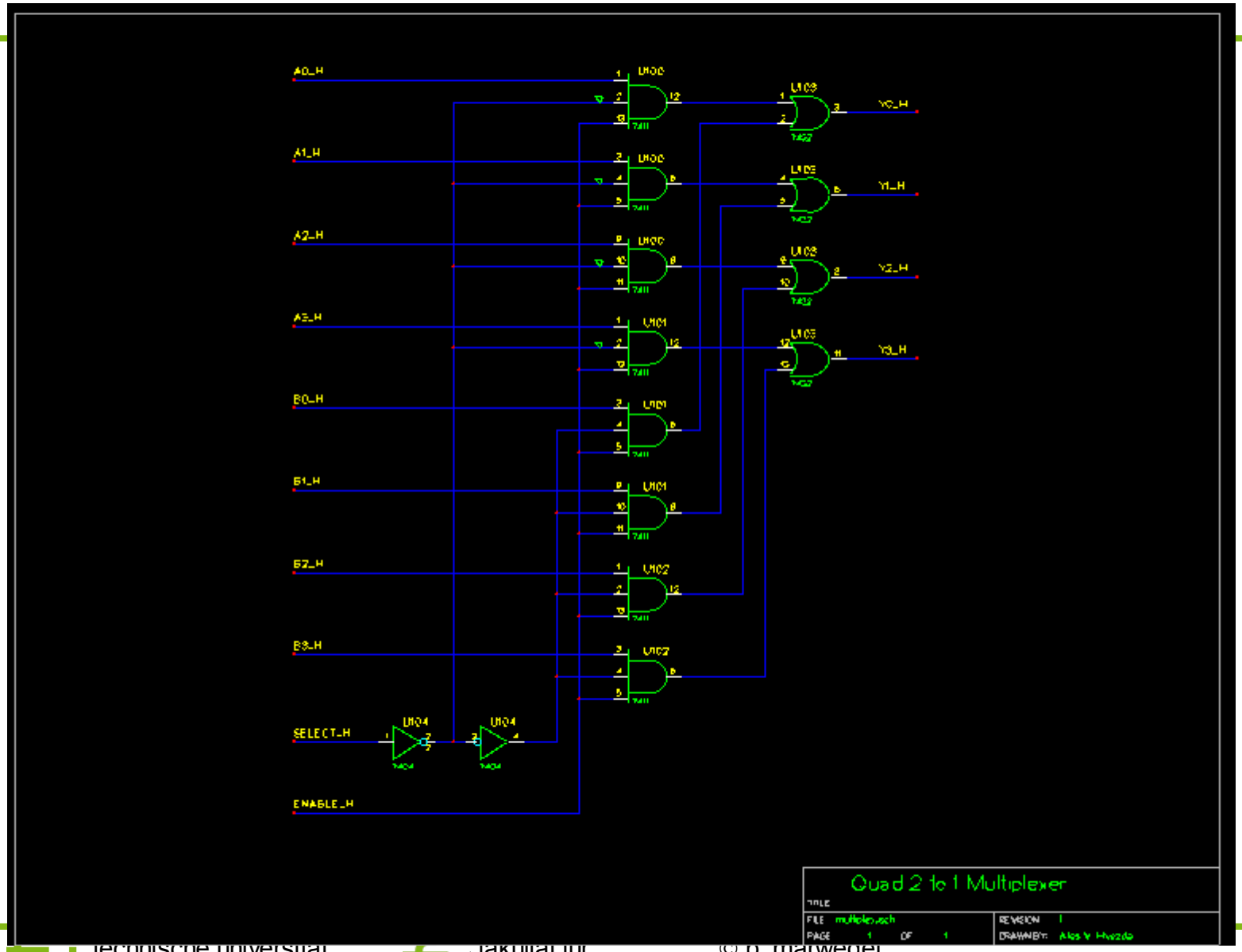
Register transfer level: example (MIPS)



Gate-level models

- Models contain gates as the basic components.
- Information about signal transition probabilities
  ☞ can be used for power estimations.
- Delay calculations can be more precise than for RTL.
  Typically no information about the length of wires
  (still estimates).
- Term sometimes also denotes Boolean functions
  (No physical gates; only considering the behavior of the gates).
  Such models should be called “Boolean function models”.

Gate-level models: Example

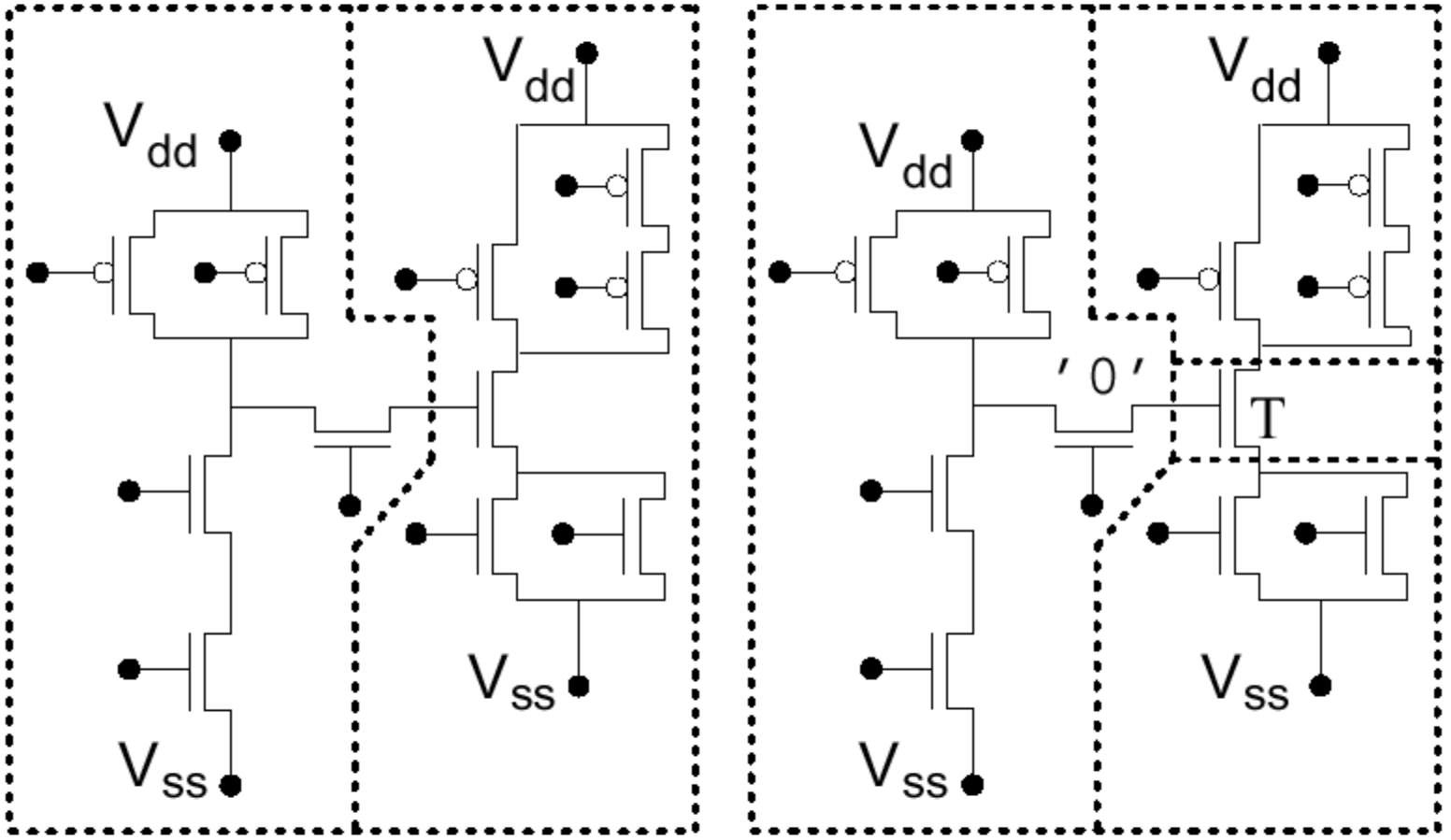


source: http://geda.seul.org/screenshots/screenshot-schem2.png

Switch-level models

- Switch level models use switches (transistors) as their basic components.
- Switch level models use digital values models.
- In contrast to gate-level models, switch level models are capable of reflecting **bidirectional** transfer of information.

Switch level model: example

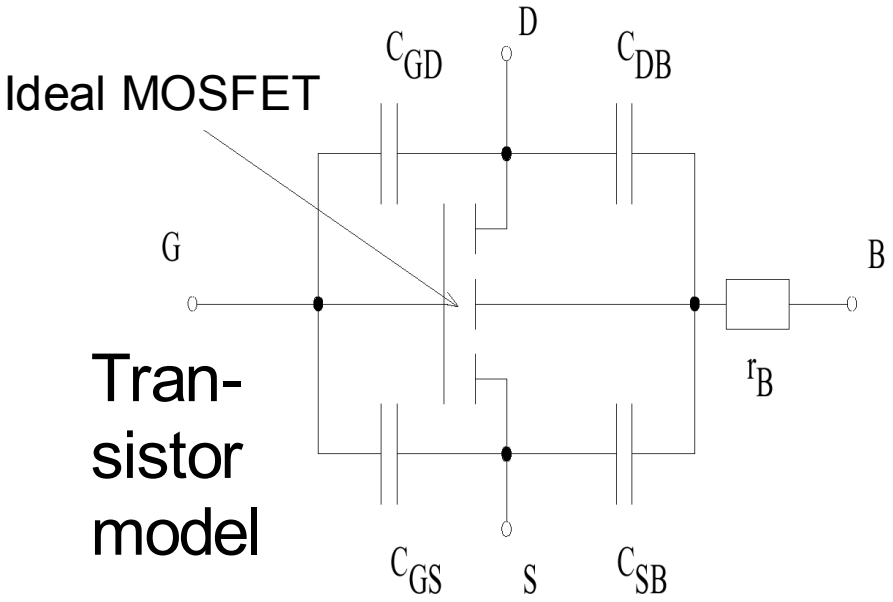


Source: http://vada1.skku.ac.kr/ClassInfo/ic/vlsicad/chap-10.pdf

Circuit level models: Example

- Models circuit theory. Its components (current and voltage sources, resistors, capacitances, inductances and possibly macro-models of semiconductors) form the basis of simulations at this level.

Simulations involve partial differential equations.
Linear if and only if the behavior of semiconductors is linearized.

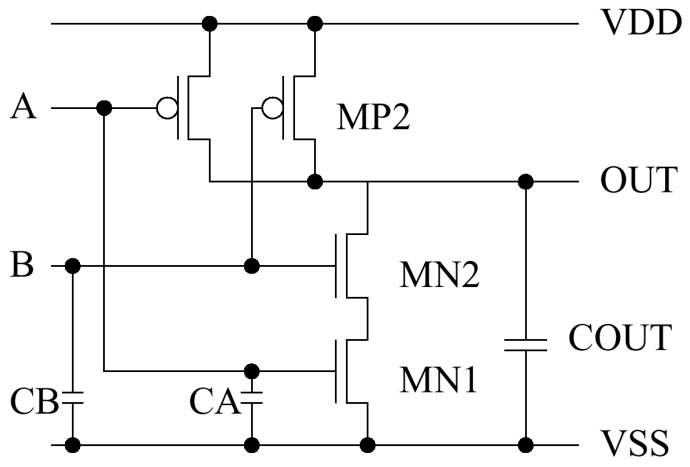


Tran-sistor model

Circuit level models: SPICE

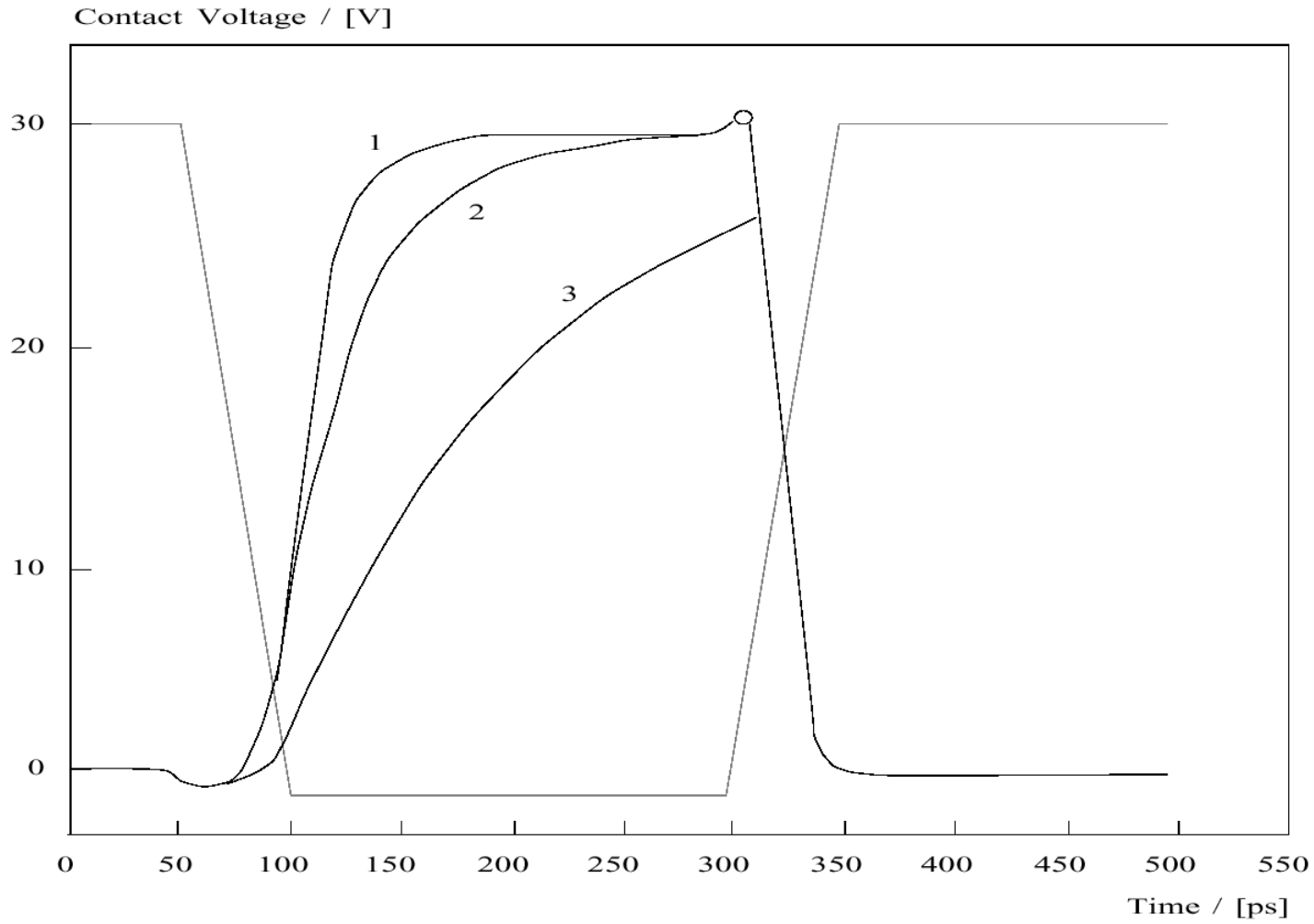
The most frequently used simulator at this level is SPICE [Vladimirescu, 1987] and its variants.

Example:



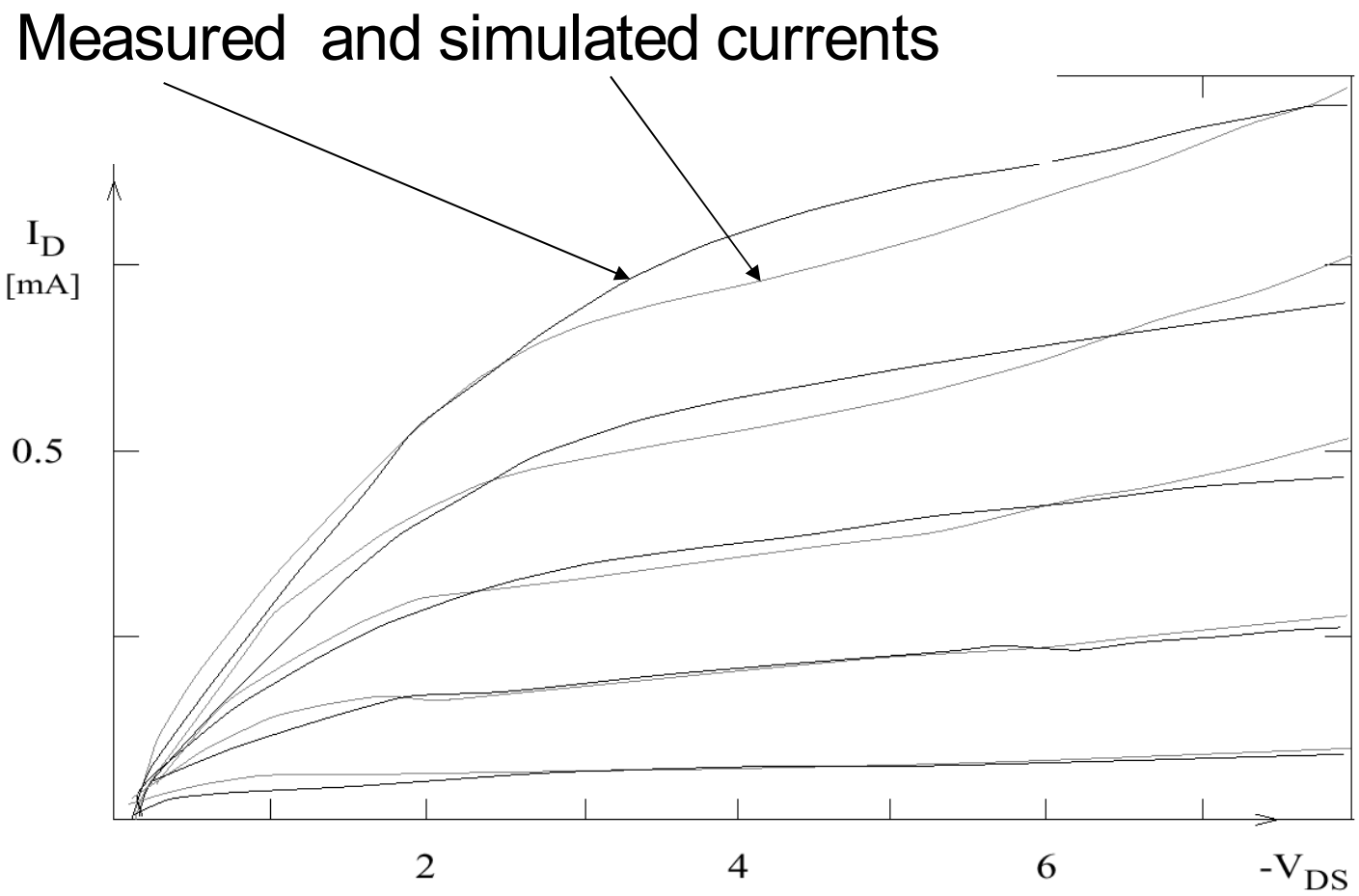

```
.SUBCKT NAND2 VDD VSS A B OUT
MN1 I1 A VSS VSS NFET W=8U L=4U AD=64P AS=64P
MN2 OUT B I1 VSS NFET W=8U L=4U AD=64P AS=64P
MP1 OUT A VDD VDD PFET W=16U L=4U AD=128P AS=128P
MP2 OUT B VDD VDD PFET W=16U L=4U AD=128P AS=128P
CA A VSS 50fF
CB B VSS 50fF
COUT OUT VSS 100fF
.ENDS
```

Circuit level models: sample simulation results



Device level

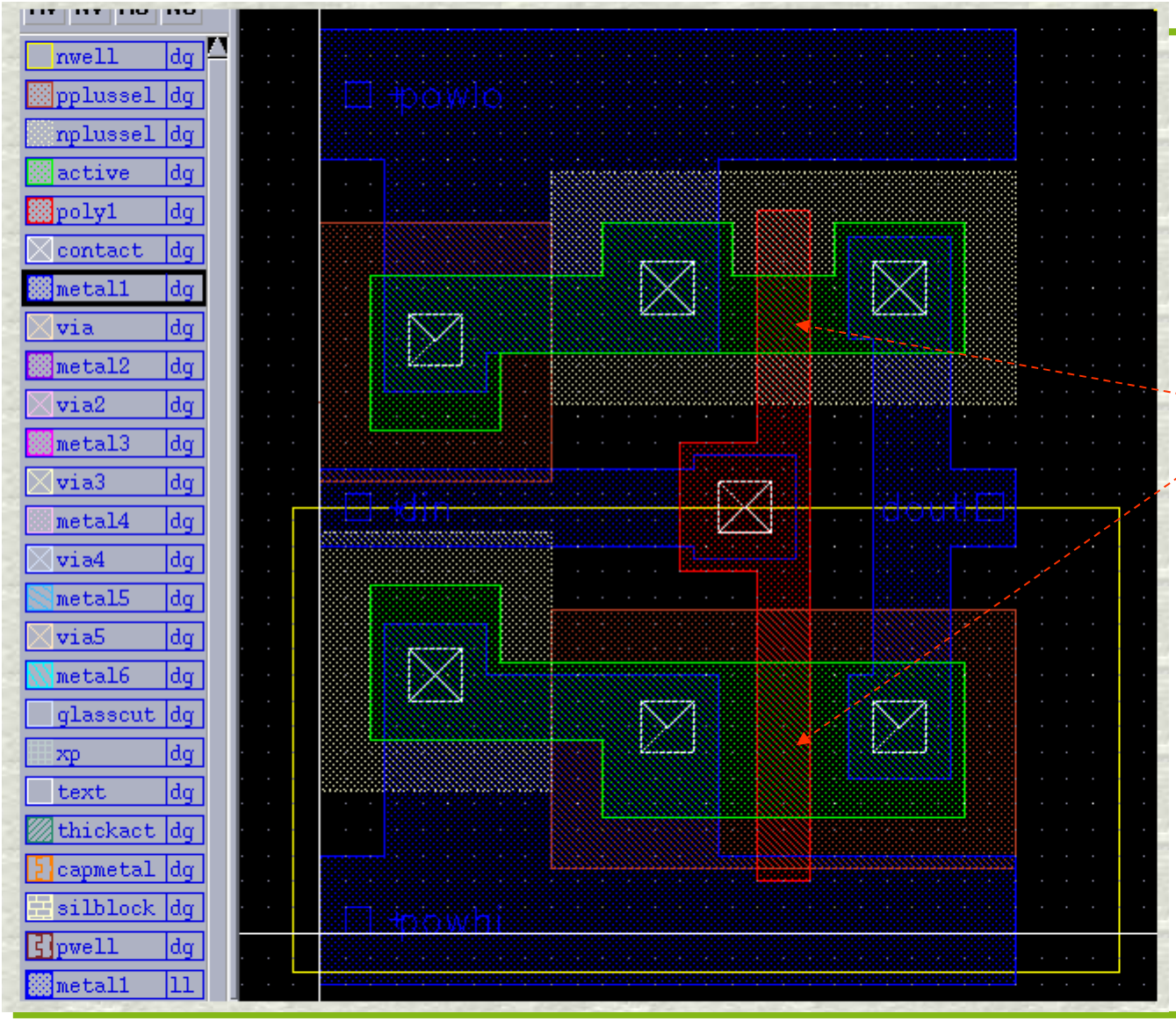
Simulation of a single device (such as a transistor). Example (SPICE-simulation [IMEC]):

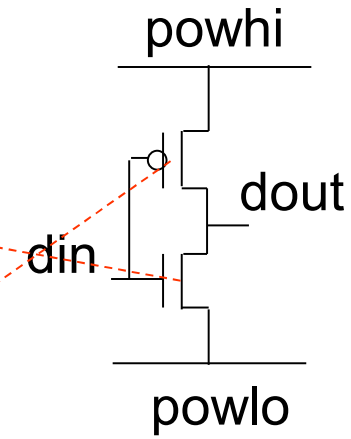


Layout models

- Reflect the actual circuit layout,
- include **geometric** information,
- cannot be simulated directly:
  behavior can be deduced by correlating the layout model with a behavioral description at a higher level or by extracting circuits from the layout.
- Length of wires and capacitances frequently extracted from the layout,
  back-annotated to descriptions at higher levels
  (more precision for delay and power estimations).

Layout models: Example

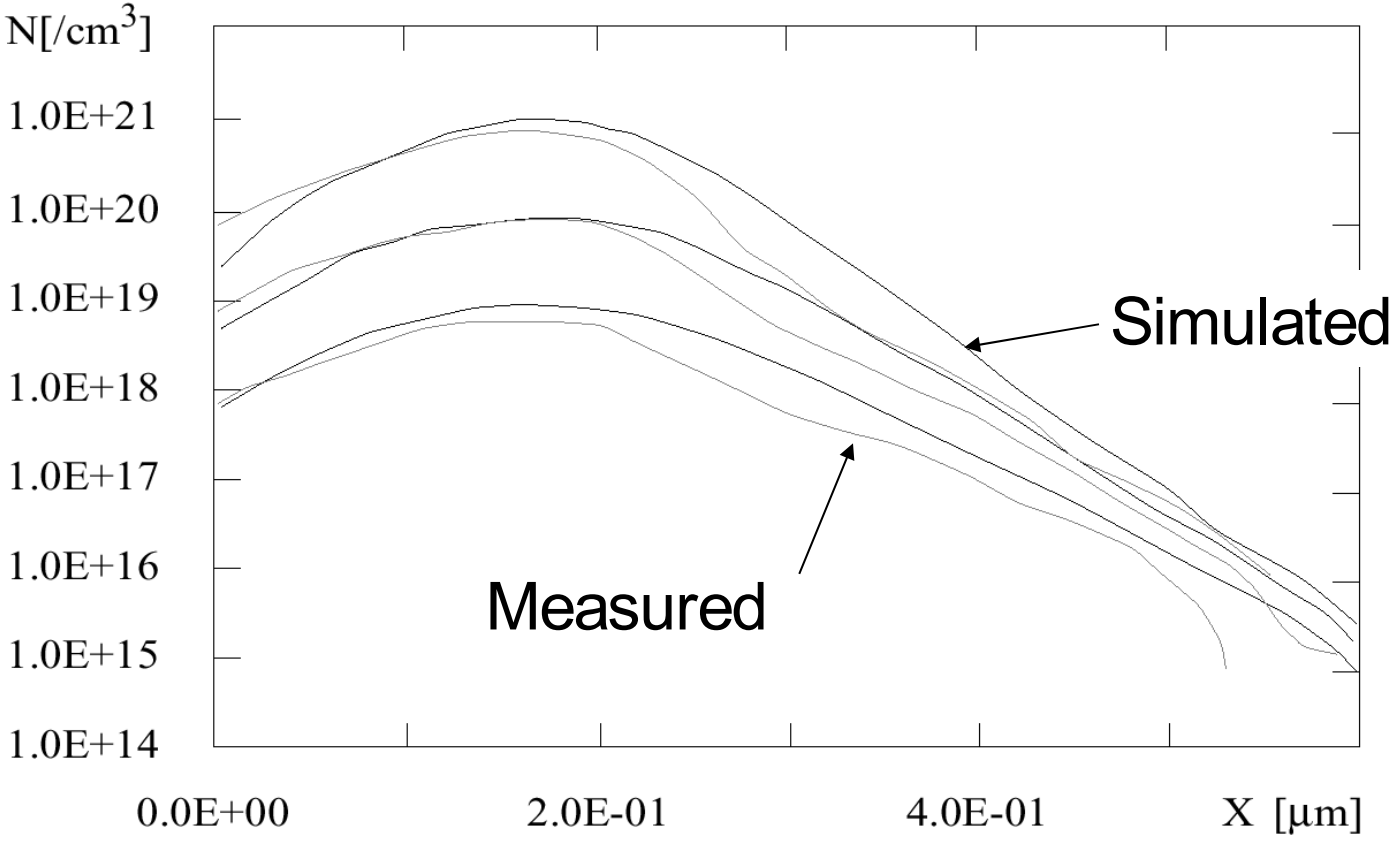




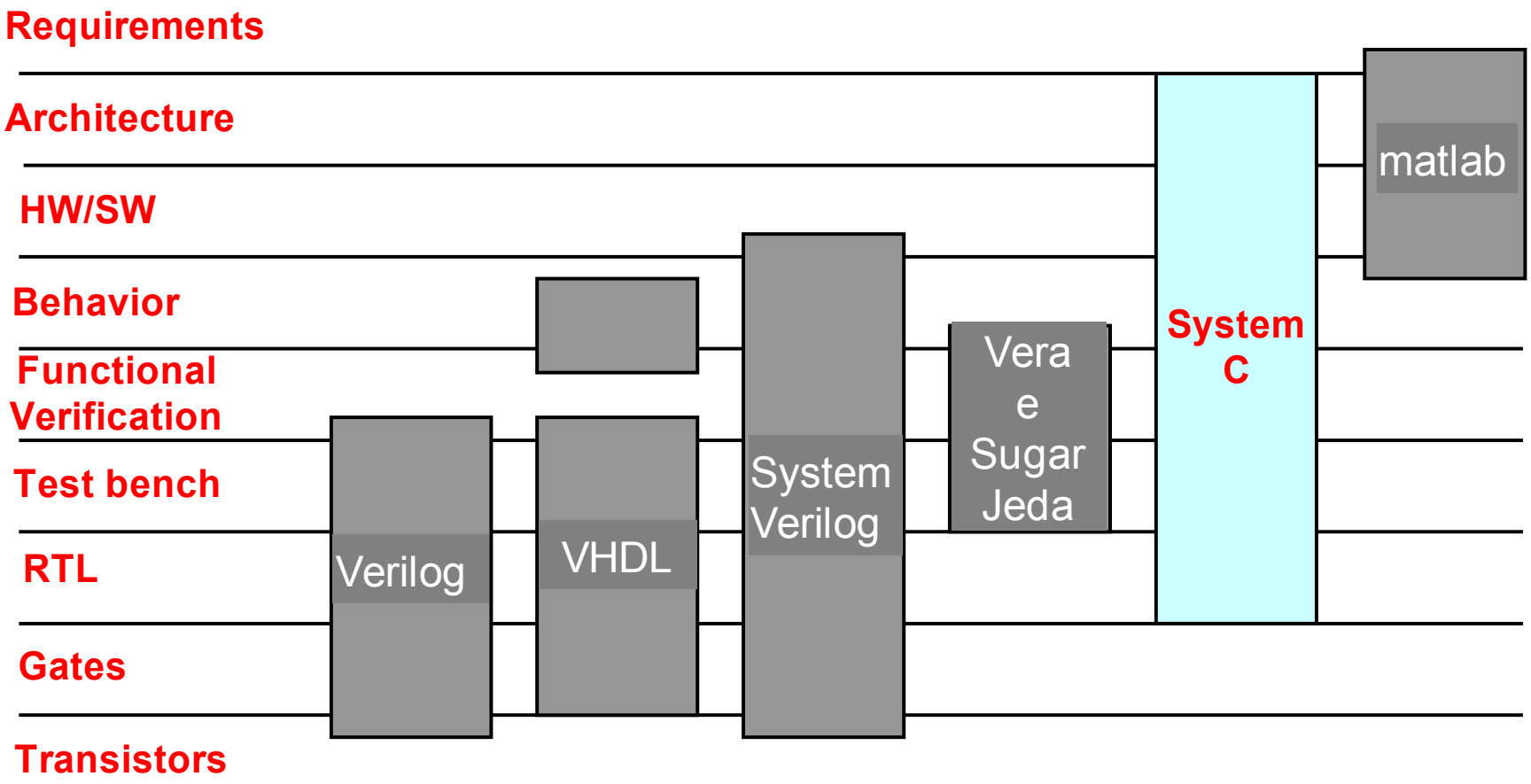
© Mosis (http://www.mosis.org/Technical/Designsupport/polyflowC.html); Tool: Cadence

Process models

Model of fabrication process; Example [IMEC]:
Doping as a function of the distance from the surface



Levels covered by the different languages



technische universität
dortmund

fakultät für informatik
informatik 12

Comparison of languages

Peter Marwedel
TU Dortmund,
Informatik 12

2008/11/1

Comparison of languages demonstrated

Communication/ local computations	Shared memory	Message passing Synchronous | Asynchronous	
Communicating finite state machines	StateCharts		SDL
Data flow model ⊂ **Computational graphs**	Not useful	Simulink Sequence dia-gram, Petri nets	Kahn process networks, SDF
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	
Discrete event (DE) model	VHDL, …	Only experimental systems, e.g. distributed DE in Ptolemy	

Models of computation in Ptolemy

1. Finite state machines
2. Communicating sequential processes
3. Discrete event model
4. Distributed discrete event model
5. Process networks, including Kahn process networks
6. Synchronous dataflow (SDF)
7. Continuous time
8. Synchronous/reactive models

Properties of processes (1)

- **Number of processes**
  static;
  dynamic (dynamically changed hardware architecture?)

- **Nesting:**
  - Nested declaration of processes
    process {
      process {
        process {
    }}}
  - or all declared at the same level
    process { … }
    process { … }
    process { … }

Properties of processes (2)

- Different techniques for **process creation**
  - **Elaboration in the source (c.f. ADA, below)**

```
declare
   process P1 …
```

  - **explicit fork and join (c.f. Unix)**

```
id = fork();
```

  - **process creation calls**

```
id = create_process(P1);
```

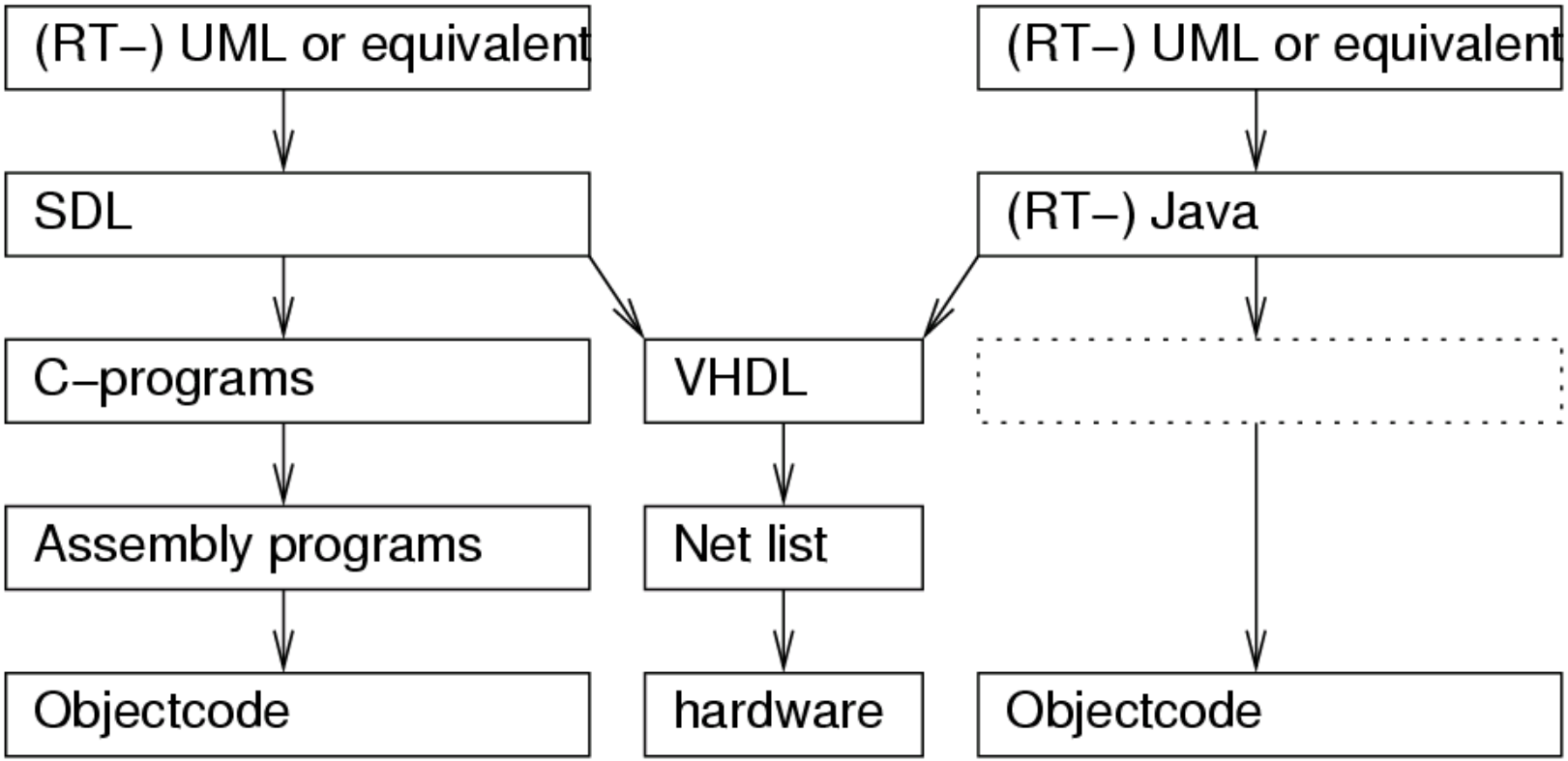
E.g.: StateCharts comprises a static number of processes, nested declaration of processes, and process creation through elaboration in the source.

Language Comparison

Language	Behavioral Hierarchy	Structural Hierarchy	Programming Language Elements	Exceptions Supported	Dynamic Process Creation
StateCharts	+	-	-	+	-
VHDL	+	+	+	-	-
SpecCharts	+	-	+	+	-
SDL	+-	+-	+-	-	+
Petri nets	-	-	-	-	+
Java	+	-	+	+	+
SpecC	+	+	+	+	+
SystemC	+	+	+	- (2.0)	- (2.0)
ADA	+	-	+	+	+

How to cope with language problems in practice?

Mixed approaches:



technische universität
dortmund

fakultät für informatik
informatik 12

Dependability requirements

Peter Marwedel
TU Dortmund,
Informatik 12

2008/11/1

Dependability requirements

Allowed failures may be in the order of 1 failure per 10^9 h.

~ 1000 times less than typical failure rates of chips.

☞ For safety-critical systems, the system as a whole must be more dependable than any of its parts.

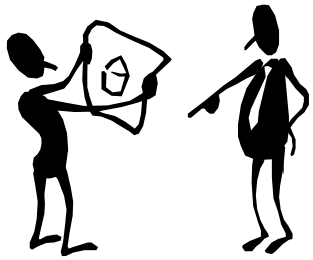
☞ fault-tolerance mechanisms must be used.

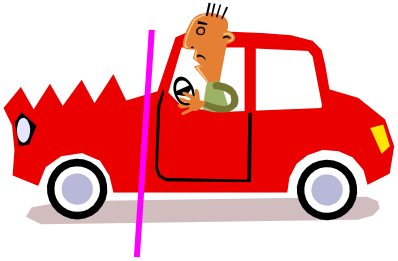
Low acceptable failure rate → systems not 100% testable.

☞ Safety must be shown by a combination of testing and reasoning. Abstraction must be used to make the system explainable using a hierarchical set of behavioral models. Design faults and human failures must be taken into account.

Kopetz's 12 design principles (1-3)

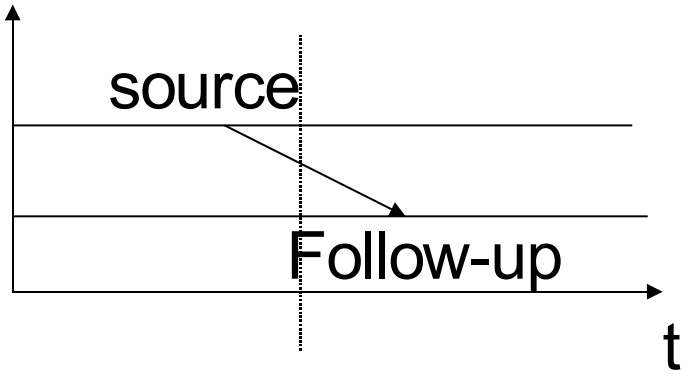
1. Safety considerations may have to be used as the important part of the specification, driving the entire design process.
2. Precise specifications of design hypotheses must be made right at the beginning. These include expected failures and their probability.
3. Fault containment regions (FCRs) must be considered. Faults in one FCR should not affect other FCRs.

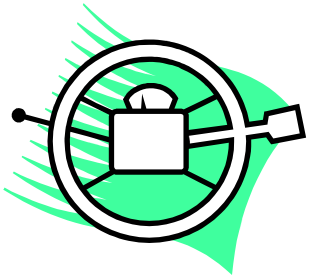




Kopetz's 12 design principles (4-6)

1. A consistent notion of time and state must be established. Otherwise, it will be impossible to differentiate between original and follow-up errors.
2. Well-defined interfaces have to hide the internals of components.
3. It must be ensured that components fail independently.





2 independent brake hose systems

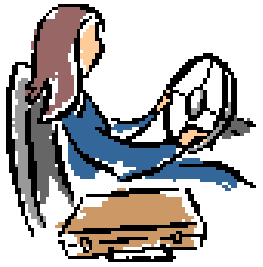
Kopetz's 12 design principles (7-9)

1. Components should consider themselves to be correct unless two or more other components pretend the contrary to be true (principle of self-confidence).
2. Fault tolerance mechanisms must be designed such that they do not create any additional difficulty in explaining the behavior of the system. Fault tolerance mechanisms should be decoupled from the regular function.
3. The system must be designed for diagnosis. For example, it has to be possible to identifying existing (but masked) errors.

one of the systems sufficient for braking

Kopetz's 12 design principles (10)

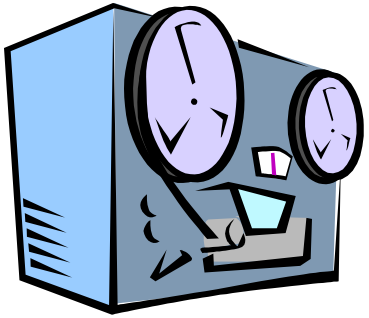
1. The man-machine interface must be intuitive and forgiving. Safety should be maintained despite mistakes made by humans



airbag

Kopetz's 12 design principles (11-12)

1. Every anomaly should be recorded. These anomalies may be unobservable at the regular interface level. Recording to involve internal effects, otherwise they may be masked by fault-tolerance mechanisms.



2. Provide a never-give up strategy. ES may have to provide uninterrupted service. Going offline is unacceptable.