MULTILEVEL SIMULATION TECHNIQUES

Franz J. Rammig

Universität-GH Paderborn

0. Abstract

Caused by the increasing complexity of design objects simulation on higher levels of abstraction becomes more and more important. It is discussed in this paper how simulation on various levels of abstraction can be integrated.

At first the levels of abstraction and the modelling concepts used are investigated. Then the three main simultation techniques (straightline code, equitemporal iteration, and event scheduling) are discussed. Finally two approaches to multilevel simulation are compared. They are characterized as Multi Simulation Approach and Broadband Approach. The first one tries to offer multilevel simulation by combining dedicated simulators while the second on tries to provide a unified concept for all levels envolved.

1. Levels of Abstraction

Highly complex digital systems have to be described at various levels of abstraction. Higher levels allow the modelling of entire systems while lower levels are necessary for detailled studies of single components. It should be noted that abstraction and hierarchy are orthogonal concepts. A level of abstraction is constituted by certain modelling concepts while a hierarchy organizes an object in a treestructured manner. On each level of abstraction a hierarchical description may be provided. On the other hand within a hierarchical description different components may be described at different levels of abstraction.

There is no "standard" on levels of abstraction, but the following system seems to be widely accepted. This system consist of six levels, level 0 (electrical level) being the lowest one and level 5 (system level) being the highest one. As we are concentrating on simulation only behavioural aspects of systems are of interest.

Level 5: System Level

The modelling concept on this level is given by a system of cooperating semi-autonomous modules like processors, channels, etc.. All these components are viewed at as "processors" in a wider sense, i.e. a device that reacts in a well defined way on instructions. So abstract data types (ADT) are a well suited point of view for this level. Usually the timing model is reduced to a causality structure. At observation points arbitrary values within a freely definable finite domain may occur. SIMULA [3] may serve as the typical dedicated language for this level, but also OCCAM [14] can be used for this purpose.

Level 4: Algorithmic Level

The components of the system level being processors in a wider sense, per component the interpretation algorithm of the instruction set has to be defined. This is done at the algorithmic level. Usually this interpretation algorithm is a highly concurrent one (e.g. if pipelining is envolved). Therefore modelling concepts like CCS [17] or Interpreted Petri Nets [20] seem to be best suited for this purpose. For a compositional view we have to distinct between components of the overall structure of the algorithm like loops or fork/join constructs and components of the operational part of the description. Here usually rather hardware-oriented components like registers or busses are used. In most cases at this level of abstraction still a causality structure is used as timing model. In some cases it makes sense to introduce a first clocking scheme. Values that can be observed are bitstings with interpretaton (e.g. as integers). ISPS [2] is the best known dedicated language at this level.

Level 3: Register Transfer Level

The RT level may be charcterized as inverted Algorithmic level. The global view of the algorithm (imperative view) is replaced by a component-oriented reactive view. The system now is described by a collection of components sharing some common connection lines. Each component is passive until a certain condition becomes true. Whenever this happens the components perform a well defined action that may influence the conditions of other components. The typical observable value is the bitstring. CDL [5] may serve as the classical example for a dedicated language at this level. More recent examples are DDL [7], CASSANDRE [15], RTS [18], and KARL [10].

Level 2: Gate Level

Gate level descriptions are obtained by simply expanding the components of the RT level to gate level implementations, i.e. to interconnection structures of gates. It should be noted that by this expansion we are loosing the information about the distinction between control and data signals. The modelling concept is now a system of Boolean equations. The components are gates and simple flipflops. The timing model is given in real numbers. Observable values are "bits" within a 2-16 valued logic. Typical dedicated languages for this level are the description languages for gate level simulators, e.g. the DISIM language [12]. The structural aspects of this level are also covered by schematic entry systems.

Level 1: Switch Level

Switch level descriptions are obtained either by expanding gates to switch networks or by describing switch circuits that have no equivalent at the gate level. The modelling concept is given by a finite automaton. Its state is given by the charge distribution on the capacitances envolved. The components are simplified transistors (switches) and "nodes" (capacitances). Observable values are interpreted chargings, in most cases within a discrete domain. The timing model is given in real numbers. The MOSSIM [4] description language may serve as a typical dedicated language for this level.

The structural aspects of this level are also covered by stick diagrams. They add some topological information to behavioral switch level descriptions.

Level 0: Electrical Level

At this level the digital interpretation of a circuit is reduced to its analog behaviour. The modelling concept is given by a system of differential equations over a real domain and with a continuous timing model in mind. The components used are resistors, capacitances, etc.. The input language of SPICE [21] may serve as an example of a dedicated language for this level.

The structural aspects of this level are also covered by the layout of a circuit. The layout together with the parameters of the process it is made for defines the electrical behaviour.

Unfortunately a lower level of abstraction does not automatically mean that more information is present. In most cases the increase of local information is paid for by loosing information about the global interaction and the design intentions. E.g. from the algorithmic level to the RT level the information about the algorithm to be performed is lost, at the gate level we lost the information about the distinction between control and data signals while at the lowest levels even the information about the logical function to be performed is no longer present.

2. Modelling Concepts

A multi-level simulator has to deal with all the modelling concepts mentioned above. To study this problem in more detail the concepts at the six different levels of abstractions have to be investigated further.

2.1 External Modelling Concepts

Two main concepts can be identified immediately:

- Continuous evaluation
- Triggered evaluation.

Continuous evaluation is used at the gate, switch, and electrical level. In all cases we have a system of equations having a stable state as global solution (if such a solution exists). This equilibrium may be distorbed by an external event (stimulus) or endogenous if there is no stable state at all. In both cases the system tries to restablize in equlibrium. This global concept is common to all three levels, only the type of the individual functions is different. In the case of the gate level we have Boolean or quasi Boolean equations, in the case of the switch level more general but still discrete functions are envolved while at the electrical level differential equations are used.

Triggered evaluation is used at the system, algorithmic, and register transfer level. Here we have to distinct between the active and passive view. I.e. either the system is described from the point of view of the instance that genęrated the triggering signals or from the point of view of the comparents that receive the triggering signals.

The algorithmic level make use of the active view only. This is the imperative nature of an algorithm. On the other hand the RT level is completely passive. The system level with its object oriented approach is both active and passive. A component reacts on certain requests (passive view) but also explicitely causes other requests (active view).

So the modelling concepts can be summarized in the following diagram:

Modelling concept

- continuous evaluation
  - (quasi) Boolean equations (gate level)
  - discrete equations (switch level)
  - differential equations (electrical level)
- triggered evaluation
  - active view (system level & algorithmic level)
  - passive view (system level & register transfer level)

2.2 Internal Modelling Concepts

Multilevel simulation systems may either try to provide mechanisms for each of the above mentioned concepts or may be based on few but powerful internal modelling concepts where all external ones can be mapped on.

Interpreted Petri Nets may serve as an example of such a powerful internal concept.

Def. 2.2.1

$PG=(P,T,E)$ is called Petri Net Graph : <=>

P finite set (of "places")
T finite set (of "transitions")
$E \subseteq PxT \cup TxP$
$P \cap T = \emptyset$
$\forall x \in P \cup T \quad \exists y \in P \cup T : (x,y) \in E \vee (y,x) \in E$

Def. 2.2.2

$PN=(PG,m_o,R)$ is called Petri Net : <=>

$PG=(P,T,E)$ Petri Net Graph
$m_o \in M=\{m \mid m:P \rightarrow \mathbb{N}_o\}$ (initial marking)
$R \in \{r \mid r:T \rightarrow f_T\}$ with $f_T=\{f_t \mid t \in T\}$ and
$\forall t \in T : (f_t:M \rightarrow M)$ (firing rule of t)

In Petri Nets "places" are used to model conditons. If a place contains a token, the associated condition is assumed to be true. So a "marking" models a global state of the overall condition space. Actions are modelled by transitions. A transition is firable if a certain condition on its input places is true (e.g. all input places are marked). By firing it manipulates the marking of its input places and output places (e.g. demarks all input places and marks every output place). By this a transistion modifies locally the global state of the condition space.

Classical Petri Nets make use of exactly one firing rule (the rule mentioned as example) while our definition also allows heterogenous rules.

Def. 2.2.3

$IPN=(PN,ID)$ is called Interpreted Petri Net : <=>

$PN=((P,T,E),m_o,R)$ Petri Net
$I \in \{ili:T \rightarrow \sigma \cup \{\lambda\}\}$ with
$\sigma=\{o \mid o:$ dom $(o) \subset X(D) \rightarrow$ codom $(o) \subset X(D)\}$
where D is a manysorted set (of "data objects") and
$X(D)$ denotes the Cartesion product over all elements of D.

Interpreted Petri Nets are obtained by attaching datamanipulations $o(t)$ to transitions t. Whenever t fires its attached operation is performed. This is called an Interpreted Firing.

Def. 2.2.4

$TIPN=(IPN,\Delta)$ is called Timed Interpreted Petri Net : <=>

$IPN=(((P,T,E),m_o,R),I,D)$ Interpreted Petri Net
$\Delta \in \{\delta \mid \delta:T \rightarrow \tau\}$ with
$\tau=\{o' \mid o':$ dom $(o') \subset X(D) \rightarrow \mathbb{R}\}$

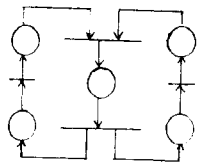
A timed interpreted firing of a transition is defined as follows:

Assume that transition t becomes firable at point of time t_o. At this point of time the attached operation (if existing) $i(t)=o$ is initiated. That means that the values of dom (o) at this point of time are evaluated. At the same point of time the delay function $\delta(t)=o'$ is evaluated based on the values of dom (o') at point of time t_o. Assume that the result of o' is k. Then at point of time t_{o+k} the values calculated by o are stored in codom (o) and the firing (i.e. the token game) takes place.

The usual graphical representation of Petri Nets is as follows:

For each place a "-O-" is drawn, for each transition a "+". These sets are connected by directed edges according to E. The marking of a place is indicated by little dots, within the symbol for this place. Interpretation and timing of a transition are just written close to the symbol of the transition.

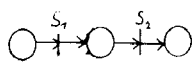
Example:



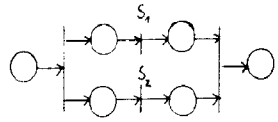
This net describes two circular processes, synchronized by a common condition.

Obviously a concurrent algorithm can be represented by a timed interpreted Petri Net. Instead of a formal proof we just give some net templates for typical algorithmic constructs:

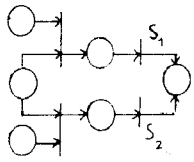
a) seqbegin $S_1;S_2$ end



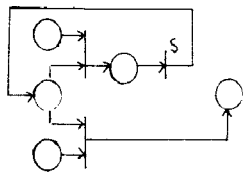
b) conbegin $S_1;S_2$ end



c) if c then S_1 else S_2

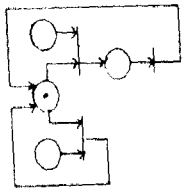


d) while c do S



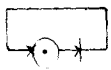
So timed interpreted Petri nets seem to be well suited for the algorithmic level. But what about the other levels. First of all passive triggered evaluation should be investigated. For this purpose the following template is appropriate:

at C do S



For each guarded command in such a description (and such a description is completely given by an unordered set of guarded commands) a net template of this type has to exist. From the Petri net point of view these nets are autonomous. They are only linked via the conditions, i.e. by the attached interpretation.

A continuous evaluation can be viewed at as a triggered one where the triggering signal is always present. Therefore the following net template is appropriate:



Again there is one autonomous net per equation S and the link between the nets is only via intersections between dom and codom of different equations.

3. Simulation Techniques

The simulation algorithm has to map the different modelling concepts to the architecture of the host computer. The most efficient simulation algorithm is present if the architecture of the host computer is identical to the modelling concept or at least rather similar. This is the basic idea of one class of simulation engines, e.g. [6, 9]. (Another class of such engines uses pipelining to speed up sequential algorithms, e.g. [1]). In our context we will only discuss sequential v.Neumann computers as host architecture.

3.1 Streamline Code Simulation (SCS)

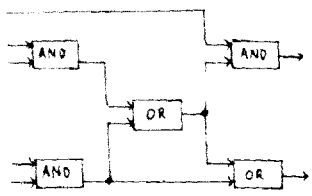
This class of simulators is also known as "compiled mode" simulators. The idea is to produce executable code for the host machine from the circuit description. SCS is applicable only under certain restrictions:

- Modelling concept is continuous evaluation.
- The object to be simulated is either combinational or a strictly synchronous sequential circuit.
- Timing information is of no interest.

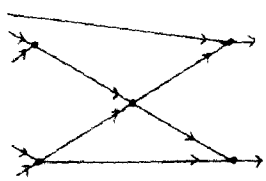
The classical application is gate level simulation for combinational circuits. So this example may be explained first:

A combinational circuit can be represented as a directed acyclic graph (dag). The gates are mapped on the nodes of the dag while the interconnecting nets result in edges of the dag.

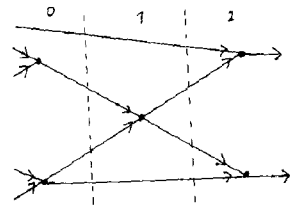
Example



is represented as



The nodes of a dag can be semi-ordered due to their longest distance to the primary inputs. This is called levelization. The level of each node is given by the maximal number of nodes between it and a primary input. In our example we get



Levelization now indicates the order of the code to be generated. Code for the nodes (in the case of gate level simulation one instruction per gate in most cases) have to be sequenced in accordance to increasing level number. Nodes within one level can be arranged in arbitrary sequence as they can not effect each other.

So within the restrictions mentioned above a highly efficient simulation algorithm is obtained.

Strictly synchronous circuits can be handled as well. In this case the registers are updated after every simulation cycle (which in this case is equivalent to one clock cycle) from the values at the respective primary outputs.

SCS is used mainly at low levels [21] and for fault simulation at the gate level [11, 13].

3.2 Equitemporal Iteration (EI)

This is the simplest tabel driven simulation technique. Like in SCS a sweep over the entire model of the system takes place iteratively. After each sweep the global time is increased by a step which may vary from iteration to iteration but is always the same for all components visited. Each component i of the system to be simulated is modelled by a triplet (c_i,a_i,d_i). Here c stands for the executability condition attached to the component. By this also triggered evaluation can be performed and assignable delay is possible. Assume that c is true in a certain iteration. Then action a is performed. As a consequence some variables get new values. However this assignment is not carried out directly to the target variables d but to buffers. So the components visited next during the actual sweep are not affected by this assignment. After the entire sweep all buffers are copied into the target variables stored in a common memory. A skeleton of such an algorithm is given by:

```
while t≤t_f do
  begin
    for all d_i do
      if c_i then d'_i:=a_i(D)
    t:=t+h;
    for all d_i do
      d_i:=d'_i
  end
```

with

t actual time, t_f final time, D global data space, $D=(d_1,...,d_n)$

This simulation technique is very simple and easy to implement. Therefore it has been very popular for gate level simulation and is still popular at the RT-level. Unfortunately it is rather inefficient in most cases. The reason is that typically at a certain point of time more than 95% of a circuit is stable. This means that the probability that an operation other than the identity has to be performed for a certain component during a given iteration is less than 0.05.

3.3 Critical Event Scheduling (CES)

This method is an attempt to overcome the efficiency problems of EI by concentrating on nontrivial computations.

CES is applicable to modelling concepts that match the following restrictions:

- The time of the next occurence of an event is predictable.
- If the time of the next occurence of an event is not predictable this event does not take place until it becomes predictable by the occurence of other events.
- Between two suceeding events there happens nothing influencing the system.

These restrictions are fulfilled in all modelling concepts discussed above. So CES is a universal algorithm in our context. From the three statements the CES algorithm can be deduced directly. In contrary to global approaches like SCS and EI, CES is a local method.

The following skeleton illustrates the algorithm:

```
begin
   time := 0;
   while time ≤ final_time and queue ≠ empty do
      begin
         extract event E_t with t minimal from queue;
         execute event E_t;
         for all predictable events E_1,...,E_n influenced by E_t
            do begin
                  calculate hatching time of E_i;
                  insert E_i properly into queue
               end
      end
end
```

Obviously this algorithm reduces the number of components to be visited and the operations to be executed drastically. The price to be payed is the overhead of keeping the queue sorted. But this is neglectable as the queue typically holds less than 5% of all components of a model. CES is the most widly used algorithm today. It seems to be the most efficient one if the restrictions of SCS can not be tolerated.

4. Multilevel Simulation

The traditional approach to offer a dedicated simulator for each level of abstraction runs into problems if complex systems have to be handled. The first problem is that accompaning the design process different simulators have to be used. So the user has to be familiar with a couple of rather complex software systems and the design data has to be transformed several times.

The most severe drawback however is that always the entire design object has to be modelled and simulated at one level of abstraction. The typical design style of piecewise refinement is not supported at all. Therefore the request for multilevel/mixed level simulation arises. Such simulators are coupled into a more or less integrated system or a broadband simulator is offered.

4.1 Multi Simulator Approach

The simplest approach to multilevel simulation seems to be the coupling of existing dedicated simulators. Two main problems have to be solved:

- The exchange of data.
- Synchronization.

Concerning data exchange the easiest situation is given in the case of identical domains and identical representations of data. This situation is rarely found in our context, as we are interested in multilevel solutions. If only the representations differ then a simple conversion is neccessary whenever data has to be exchanged.

Typically in our context however we have different domains. Typically at a lower level of abstraction domains of a high cardinality are used. Certain subsets of such a domain are interpreted as a certain value at a higher level, such forming the domain at the higher level. This indicates that it is relatively simple to hand over data from a low level simulator to such one at a higher level. In most cases a simple threshhold function has to be performed. The reverse direction is much more complicated. E.g. if a gate level simulator hands over a logical value to an electrical simulator this data carries much too less information for the electrical simulator. So certain assumptions about the driver part of the sending gate have to be made. Such assumptions may either be part of the coupling software or may be provided by the user.

Concerning synchronization either a centralized supervisor has to be installed or control may be distributed with the possibility of mutual interuption. In both cases the simulators envolved have to be modified.

In the case of a centralized supervisor each simulator of the system has to hand over control to this superviser before advancing time. During this in addition it has to indicate the next point of time it wants to handle. The supervisor now identifies the simulator with the closest future time to be handled and hands over control to this simulator. Passing data to another simulator can be handled like a stimulus for the receiving system. This causes no extra problem if the target simulator is able to handle dynamic stimuli. This synchronization is rather simple and only very slight modifications are neccessary at the simulators envolved. However at each point of time only one simulator can be active even when running on different processors.

The other approach to synchronization is to have relatively free running simulators. Whenever simulator S_1 wants to hand over data d to simulator S_2 at point of time t it sends an interrupt packet (d,t) to S_2.S_2 compares t with its current point of time t'. If $t>t'$ it simply schedules d as stimulus at point of time t'. Otherwise it has to restore its state at point of time t-1 and recalculate with the stimulus d at point of time t in mind. This restoring operation is a crucial one. Either the simulator is able to simulate backwards or there have to be checkpoints at a certain frequency where the state of the simulator is saved. In this case S_2 has to look for its last checkpoint t'' with $t''<t$ and resimulate from this point of time, concerning stimulus d at t.

This method is more efficient than the supervisor approach if data exchange happens not too frequently and if a powerful restoring mechanism is included in the simulators used. However the modifications at the simulators are more complicated than in the case of a supervisor.

A couple of multisimulator projects produced reasonable results. A remarkable one seems to bee [16].

4.2 Broadband Simulators

An alternative solution is to produce one single simulator that covers a wide range of modelling concepts. For this purpose there has to exist a simple internal modelling concept that is powerful enough. In section 2.2 timed interpreted Petri nets have been introduced as an example for such a concept. Based on this idea the author has designed the broadband hardware description language CAP/DSDL (Concurrent Algorithmic Programming Language/Digital Systems Description Language) [19]. Broadband simulators for this language are the SMILE system from Siemens [8] and the DACAPO system by DOSIS [22].

CAP/DSDL covers the levels from system level via algorithmic level, RT level, gate level down to switch level. It offers a powerful modularization concept including abstract data types and generic instantiable module types. There is a wide variety of data types and an extremely precise timing concept including optional intervals of uncertainty. A rather similar but much more restricted language is VHDL [23] but CAP/DSDL in its present version is in practical use since 1980, earlier versions since 1977. CAP/DSDL descriptions are compiled into an internal representation. This code is exactly a timed interpreted Petri net. The data structure representing this net is separated into two main parts: One part to describe the control structure (the pure Petri net) and one for interpretation and timing. This part is rather conventional code for a virtual three address machine. The main feature that has been added is an event mechanism that allows to trigger operations just by a write access to a data object.

The internal representation as generated by the compiler is interpreted by a virtual CAP-engine. It is implemented as an event scheduling algorithm. This algorithm fits very naturally to the concept of Petri nets. The firing of a transition is treated as an event. As we have timed Petri nets (i.e. timed firings) in fact we have two subevents: The initiation of the firing and its termination. CAP nets are defined in such a way that the firing of a transition is initiated immediately when the firing condition of this transition becomes true. At the same point of time the attached data operation is initiated and based on values of this point of time the proper delay is calculated (CAP/DSDL allows dynamic delay expressions that may dependent on the actual state of a system). The assignments take place after the calculated delay time elapsed. This terminates the data operation. At the same point of time the firing terminates, i.e. the token game is played according to the firing rule of the transition.

Following this concept efficient multilevel simulators have been implemented. A reacent version of the DACAPO system is even more efficient by replacing the interpretation algorithm by generation of directly executable code. This combines the flexibility and generality of event scheduling with the high performance of compiled mode simulation.

5. Summary

Multilevel/mixed level simulation is an important tool for the design of highly complex systems. From the designer's point of view it is essential to look at a design object from very different modelling concepts. In a stepwise refinement process several concepts have to be present at the same time. This causes problems to the simulation system. These problems may either be solved by coupling dedicated simulators or by building one broadband system. The first solution is preferable when few interaction between the levels takes place and if the use of special simulators is essential, e.g. because of the

offered libraries. If the levels vary rapidely and high flexibility is requested then the broadband approach seems to be more appropriate.

6. References

[1] Abranovici, M. etal.: "A Logic Simulation Machine", IEEE Transactions on CAD of Integrated Circuits and Systems Vol. CAD-2, No. 2

[2] Barbacci, M.R.: "Instruction Set Processor Specification (ISPS): The Notation and its Application, Dept. of CS, Carnegie Mellon University, 1979

[3] Belsness, O.: "The Use of SIMULA for Real-Time System Implementation", Norwegian Computing Center, Oslo, 1978

[4] Bryant, R.E.: "MOSSIM: A Switch-Level Simulator for MOS-LSI", in Proceedings of 18th Design Automation Conference, 1981

[5] Chu, Y.: "Introducing CDL", IEEE Computer, Dec. 1979

[6] Dennau, M.M.: "The Yorktown Simulation Enginge: Architecture and Hardware Description", in Proceedings of 19th Design Automation Conference, 1982

[7] Duley, J.R., Dietmeyer, D.L.: "A Digital system Design Langugae (DDL)", IEEE ToC, Vol. 24, No. 2, 1975

[8] Gonauser, M. Egger, F., Frantz, D.: "SMILE - A Multilevel Simulation System" in Proceedings of ICCD'84, 1984

[9] Hahn, W., Fischer, K.: "High Performance Computing for Digital Design Simulation", in Proceedings IFIP VLSI'85, 1985

[10] Hartenstein, R.: "Fundamentals of Structured Hardware Design", North Holland, 1977

[11] Ishiura, N., et.al.: "High-Speed Logic Simulation Using a Vector Processor", in Proceedings of IFIP VLSI'85, 1985

[12] Jäger, U.: "Logik- und Fehlersimulation mit dem Programmsystem DISIM", in Seminarunterlagen Praxis der Großintegration, Abt. Elektrotechnik, Univ. Dortmund, 1983

[13] Köpper, S., Starke, C.: "Logiksimulation komplexer Schaltungen für sehr große Testlängen", in NTG-Fachtberichte, Band 87, 1985

[14] May, M.D.: "OCCAM", ACM SIGPLAN Notices, Vol. 18-4, Apr. 1983

[15] Mermet, J.: "Etude méthologique de la Conception Assistée par Ordinateur des systémes logiques: CASSANDRE" Thèse d'état, Université de Grenoble, 1973

[16] Mermet, J.: "The CASCADE Hierarchical Multilevel Mixed Mode (HM3) Simulator" in Proceedings EUROMICRO'85, 1985

[17] Milner, R.: "A Calculus of Communication Systems", in Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, 1980

[18] Piloty, R.: "RTS (Register Transfer Sprache)" Technischer Bericht, Institut für Datentechnik, TH Darmstadt, 1969

[19] Rammig, F.J.: "Preliminary CAP/DSDL Language Reference Manual" Forschungsbericht der Abt. Informatik, Univ. Dortmund, Nr. 129, 1980

[20] Reisig, W.: "Petri Nets: An Introduction", Springer-Verlag, 1985

[21] Vladimirescu, A., Liu, S.: "The Simulation of MOS Integrated Circuits Using SPICE 2", Memo VCB/ERLM 80/7, Univ. of California, Berkeley, 1980

[22] --: "DACAPO-II System User Manual", DOSIS GmbH, Dortmund, 1986

[23] --, "VHDL Language Reference Manual, Version 7.2", IEEE, June 1986