Acceleration of simulation time by hierarchical modelling

A. Hunger, A. Papathanasiou

Department for Dataprocessing, Faculty of Electrical and Electronic Engineering, University of Duisburg, Bismarckstr. 81, D-4100 Duisburg, Germany

ABSTRACT

The use of hierarchical data structures during the run time of simulation results in excellent performance values regarding model size and loading time without any loss of accuracy of the results. This paper describes the algorithms and techniques exploited by a hierarchical simulator and gives benchmark examples with regard to memory size and loading time for circuits of different size. Although this technique has been developed for the simulation of digital circuits, it is applicable to a broader spectrum of problems of discret simulation.

INTRODUCTION

Conventional simulators flatten hierarchical designs in order to get one netlist of the entire circuit with one-to-one correspondance between all physical nodes and the nodes of the simulation model. As a consequence, all information regarding the hierarchical structures get lost. In the following, we introduce a new simulation concept which retains the hierarchical structure of the model also during simulation, i.e. run time of the simulator. Based on this concept, the software package TESI (TEst & SImulation) has been implemented covering the most important functions for logic and fault simulation as well as test generation at gate level.

Exceptional feature of the concept is hierarchical modelling. The hierarchy of the circuit is used not only for model description but also during run time of the simulator and the other tools of TESI. For this purpose, a substructure of a circuit which is used in multiple areas of the model is stored only once; only a reference to its unique definition is stored at the places where the substructure is used. This fact results in very compact internal data structures. As a consequence, the required space for working memory as well as the CPU time for loading and storing the net list are dramatically reduced. Minor drawback (compared to a simulator using the traditionally flattened netlists) is the slightly increased amount of CPU time for traversing the tree of instances during

simulation, as each sub-network has to be addressed every time it is instanciated.

For the user, this exploitation of the hierarchy offers an easy way to load and simulate big circuits on a machine with limited physical memory - sometimes it may even be the only way to simulate them. In any case, the loading of the model in its proposed hierarchical form will be performed significantly faster resulting in a quicker turn-around during iterations of re-designs.

MODELING

As mentioned above the exceptional feature of TESI is its hierarchical modeling. This model structure is based on the ideas of EDIF (Electronic Design Interchange Format) [1]; in fact TESI has been constructed in parallel to the development of EDIF and the concept of the resulting hierarchical data format of EDIF was the driving force for the development of the respective simulation tools [3]. Consistency of the ideas behind EDIF and TESI as well as the resulting benefits are illustrated by the following example which describes the hierarchical modeling of a full-adder (see fig. 1 for the description in EDIF).

This EDIF example describes a full adder element and the subcells it contains. The full graphical representation of the hierarchical design is shown in figure 2. The Full-Adder design is composed of two Half-Adder cells and a OR cell. The Half-Adder cell consist of an EXOR cell and an AND cell.

```
(edif
   (level ...)
      (technology ...)
         ....
         (cell Half_Adder
            (interface ...)
            (contents
               (instance EX1 (cellref EXOR))
               (instance AN1 (cellref AND))
               (net A
                  ....
         )
         (cell Full-Adder
            (interface
               (port A (direction input))
               (port B (direction input))
               (port Cin (direction input))
                ....
            (contents
               (instance HA1 (cellref Half-Adder))
               (instance HA2 (cellref Half_Adder))
               (instance OR1 (cellref OR))
               (net S1
                  ....
         )
         ....
```

Fig. 1: Description of a full-adder element in EDIF

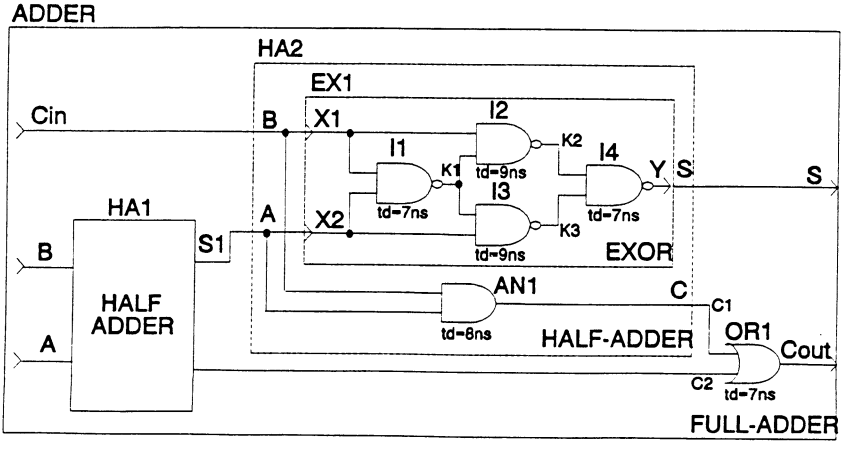

Fig. 2: Hierarchical structure of a full-adder

In the same manner like EDIF [2], TESI defines instances, cells (structures) and the behavior of an element in separated data structures. This data structures used by TESI consist of (see fig. 3):

1. an instance table containing:
   - connections between instances
   - a reference to the cell (structure) used by this instance
2. a cell (structure) table containing:
   - references to the parents of each node
   - references to the children of each node
   - references to all primitives driving each node
3. a primitive (behavior) table containing:
   - attributes of a primitive (e.g. delays, delay mode, signal strenth, etc.)

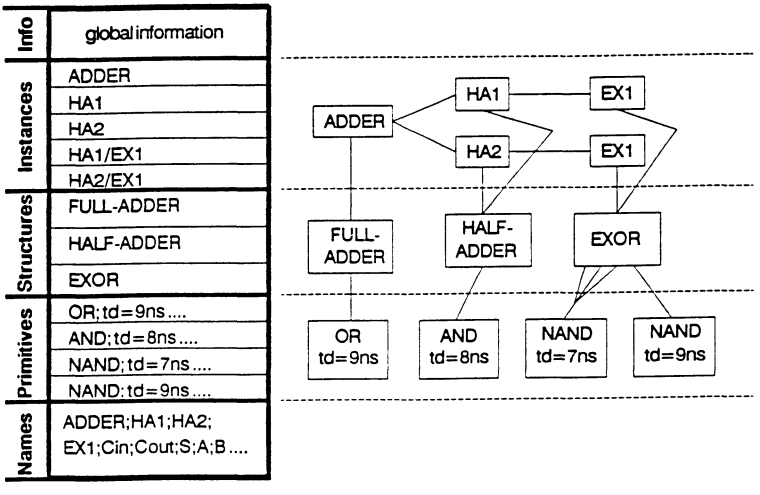

Fig. 3: Internal modeling structure

TOOLS

The set of tools implemented for the described data structure covers the common functions as logic and fault simulation, testability analyses and test pattern generation (at the time being only for combinational circuits). Experiences with these implementations show that the additional amount of calculations for referencing the nested instances and handling the

complex hierarchical addresses of instances and nodes can be balanced by adaquate programming methods.

The following example (pseudo code given in fig. 4) illustrates how a model node is retrieved from the compacted datastructure and by this made available for a simulation task. The kind of simulation is intentionally left open (represented by CalculateNode()) as this operation does not influence the performance of the tool as far as hierarchy is concerned. To distinguish between the process of retrieval by traversing the hierarchy and the commonly used mechanism of discret event simulation, the pseudo code (fig. 4) is separated into two respective parts. The recursive procedure SimulateToNodes which includes the retrieval of the nodes (ToNodes) connected to a given EventNode and the respective calculation of the given event at these ToNodes and the main program SimulateAllNodes which simulates a given circuit in total.

```
SimulateAllNodes;

   PROCEDURE SimulateToNodes ( NODE, INSTANCE);
      FOR all ToNodes of (NODE, INSTANCE) DO
         IF ToNode_i > NodesInInstance (INSTANCE) THEN
            GetNewNodeAndInstance (ToNode_i, NewNode, NewInstance);
            SimulateToNodes (NewNode, NewInstance);
         ELSE
            CalculateNode();
            PutEvent();
         END;
      END;

BEGIN (* SimulateAllNodes *)
   SetupSimulation();
   WHILE GetEvent ( EventNode, EventInstance) DO
      SimulateToNodes (EventNode, EventInstance);
   END;
END;
```

Fig. 4: Pseudo code for simulate a digital circuit

The main program given in fig. 4 is based on a time wheel which is constructed to store all discret events in proper sequence. As long as events are queued, the function GetEvent delivers the next EventNode and its EventInstance in order to have its results at the ToNodes calculated by SimulateToNodes. This procedure compares wether the retrieved $ToNode_i$ number is in the actual instance. As long as the number of $ToNode_i$ is out of this range, the next level in the hierarchy is examined until the instance corresponding to $ToNode_i$ is found. Now (ELSE part in fig. 4) the

adequate calculation is performed and all resulting events are inserted into the time wheel (by PutEvent()) for simulation at a later time.

EFFECTIVE SIMULATION TIME

The performance of simulation tools is often described by their power to process a specified amount of events per seconds. But this is not the whole truth, when using a simulation tool for validation during the design of a chip. The design of a chip must be seen as a iterative process which demands more than only to calculate a specific number of events: During each of the iterative loops the designer has to handle following steps:

- run the schematic to netlist converter,
- set up the simulation model and
- load this model into the physical memory of the workstation before the first event can be calculated.

Regarding this process, the effective simulation time is the overall ellapsed time from building and loading the simulation model to the end of simulation.

BENCHMARK EXAMPLES

The first benchmark circuit is a 16x16 bit pipeline multiplier. The platform for this benchmark is a SUN 4 (one flattening simulator was only available on an Appollo workstation) with 64 MB memory space, thus limited memory space could not affect the runtime statistics of the flattening simulators. The simulation characteristics are:

number of input events:	4.752
number of simulated events:	2.770.000

The results of this benchmark shown in tab. 1 reflects the advantage of TESI during preparation of the simulation (netlist compilation and simulation setup (e.g. loading the model and the input events)). The simulation time itself is in fact in a ratio of about 1:2 slower. But the propotion of the preparation time is much better.

	TESI . SUN4 min.	flattening simulator 1 SUN4 min.	flattening simulator 2 SUN4 min.	flattening simulator 3 SUN4 min.	flattening simulator 4 Appollo min.
netlist compilation	0:14,66	0:38,17	1:48,26	1:03,00	6:00,00
simulation setup	0:12,11	1:47,14	0:22,87	1:13,18	4:17,00
simulation	5:28,17	4:25,41	3:35,66	2:13,69	1:00,00
effective simulation time	5:54,94	6:50,72	5:46,79	4:29,82	11:17,00

Tab. 1: Runtime results of benchmark 1

The second benchmark was done to show the significant advantages (with regard to memory space and loadtime) of TESI in contrast to a flattening simulator. Only the flattening simulator 3 (see tab. 1) was available on the used platform SUN3/60 with 4 MB memory space.

The set of benchmark circuits used now consists of following multipliers:

name	function	net-count
mul8x8	multiply 8x8 bits	782
mul16x16	multiply 16x16 bits	3.493
mul32x32	multiply 32x32 bits	14.630
mulabs32	add/multiply 32x32+32x32	29.830

The structure of the circuits is given by elementary multipliers and adders which calculate the final result on the basis of provisional results on subsequent circuit levels. All circuits were simulated with 10 patterns representing 10 arithmetic operations. The results of this benchmark are presented in fig. 5 and 6.

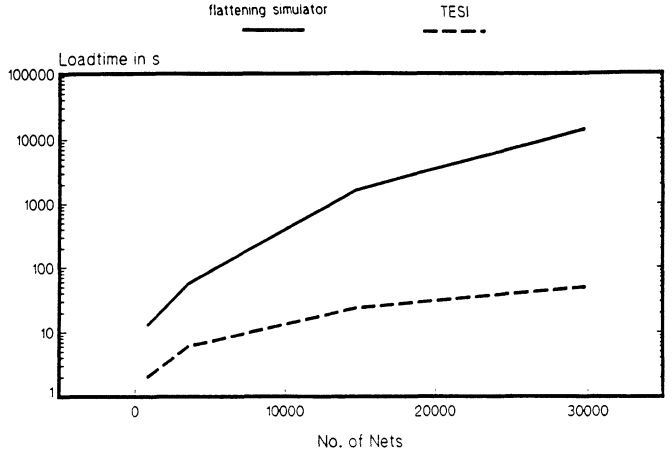

Fig. 5: Comparison of model loadtime between TESI and a flattening simulator

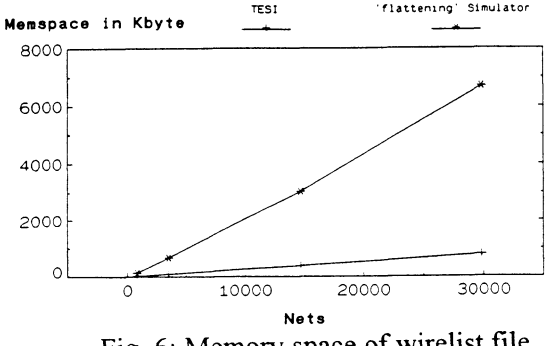

Fig. 6: Memory space of wirelist file

REFERENCES

1. EDIF Steering Commitee, 'EDIF - The Electronic Design Interchange Format V 2 0 0', Electronic Industries Association (E.I.A.), Engineering Department, Washington D.C., USA, 1987.

2. Kahn, H.J. 'Using EDIF Version 200', University of Manchester , CAD/031/MAN/001/A4-1, pp. 6.1-6.19, England , 1988.

3. Groß-Alt, W. 'Hierarchische Schaltungsmodellierung für Test und Simulation', Verlag TÜV-Rheinland, Köln, 1989.

4. Rogers W., Abraham J. 'CHIEFS: A Concurrent, Hierarchical and Extensible Fault Simulator', pp. 710-716, IEEE International Test Conference, November 1985.