LOGEX - An Automatic Logic Extractor from Transistor to Gate Level for CMOS Technology

Michael Boehner
Siemens AG, Research Laboratories
Otto-Hahn-Ring 6, D-8000 Munich 83, West Germany

Abstract

A program for automatic extraction of a gate level description from a transistor level description based on the layout of a CMOS VLSI circuit is presented. The extraction algorithm combines transistors to gates of arbitrary complexity without the help of any cell library. The resulting gate level description provides the input for a digital logic simulator for further investigations.

1. Introduction

The layout verification of a VLSI circuit is usually performed by a network comparison check (NCC) of a network extracted from the geometrical layout data and a network generated by a schematic entry [1, 2]. The schematic entry is hierarchically organized and has to be verified by logic simulation. After this functional verification the whole or parts of the schematic entry according to the needs of the NCC program must be detailed to the transistor level. Then the NCC performes the structural verification.

For certain design methodologies where the layout has been designed without the assistance of a schematic entry an NCC cannot be applied. This happens for example in designing regular arrays, by laminating a small set of cells already verified, with the aid of some kind of a module generator. In this case it may be advantageous to verify the layout by a simulation based directly on a transistor netlist extracted from the layout. Thus, the structural verification by an NCC is obsolete. Transistor level logical simulation is possible by means of switch level simulators like MOSSIM [3]. Nevertheless it becomes very time consuming for circuit complexities of several thousand transistors. Furthermore, due to the bi-directional nature of MOS transistors, a lot of cell structures cannot be simulated without special assistance (e.g. node initialization).

To overcome this problem we lift the simulation level from transistor to gate level. Gate level simulators can handle much more transistors in terms of gates and are more time-efficient than switch level simulators.

2. Basic Concepts

The program LOGEX has been developed to provide a lifting from transistor level to gate level with the aim to feed directly into a gate level logic simulator. Gate recognition should not be restricted to common CMOS gates, i.e. pull-up branches of p-channel transistors complementary to pull-down branches of n-channel transistors. Instead all kind of static and dynamic logic with arbitrarily distributed n- and p-channel transistors should be considered.In order to be able to do this, LOGEX

- transforms the transistor view of the transistor netlist into a node view and rearranges the transistors in a special order,
- combines transistors to hierarchically described branches of arbitrary complexity,
- classifies these branches by building up a catalog,
- combines the branches to CMOS gates, if possible, using the branch catalog,
- eliminates several kinds of feedback branches,
- finds resistive pull-up and pull-down branches,
- defines signal flow in bidirectional branches,
- extracts pins and busses,
- analyses solitary nodes,
- generates a gate level logic description incorporating standard propagation delays as input for a logic simulator,
- writes a protocol file containing cell statistics, a list of extracted pins, an analysis of circuit structures connected to those pins not defined by the user, statistics of the conversion of bidirectional branches to unidirectional ones, a list of abnormal connected transistors (e.g. between GND and VDD) and a list of structures deleted during the extraction process.

The whole procedure runs without the use of a cell library, in contrast to other function recognition programs [4]. Although user assistance is not necessary, there are several possibilities to interact with the program on request by control statements. The user may

- define the sequence of transistor terminals in the input netlist (gate terminal is the 1st, 2nd or 3rd in the list),
- tell the program the node names for VDD and GND,
- tie nodes together by renaming them,
- assign input, output, and bidirectional pins to nodes,
- require that all nodes following special naming conventions are treated as pins,

- define that separate branches are to be built for n- and p-channel transistors,
- activate special rules to define the signal flow in bidirectional branches (in addition to universal rules which are always active),
- demand the program to replace an inverter chain by the logically equivalent gate and a corresponding sum propagation delay.

Although there are no circuit structures which could not be extracted by LOGEX, not all of them may be subjected to a subsequent logic simulation. These are analog circuit structures, e.g. differential amplifiers (read-amplifiers in RAMs), and structures where two or more cells are actively feeding the same node as in 6-transistor RAM cells. Also, cells with real bidirectional signal flow cannot be handled adequately by most logic simulators.

The program is written in PASCAL with more than 11,000 lines of code. For minimal memory consumption the internal data representation of the transistors, branches, cells and their connections is established dynamically by different types of (chained) records and associated pointers. For a fast access without search the anchors of the whole structure are fixed in an array of 64k * 4 Bytes corresponding to 64k possible nodes of the VLSI circuit. Our experience showed that this maximal node number is sufficient for circuits with more than 100,000 transistors. The address space to hold such a large structure is less than 8 MBytes. (These numbers are limitations due to the used computer hardware not due to LOGEX.)

3. From Transistor View to Node View

The input database for LOGEX is a text file comprising the transistor netlist. Each line of the file contains the type and the three node names of a single transistor. LOGEX first scans all node names, renames them, and adds pin attributes if required by control statements. Due to the automatic generation of the transistor netlist from the layout, there are two kinds of node names: some are user defined by the layout designer, the rest is generated automatically by the transistor extraction program. LOGEX is able to distinguish between these two kinds of node names and will treat all user defined node names as pins if requested by a control statement.

While scanning the node names, LOGEX assigns each node name a running number for further reference during the extraction process (index of the net structure's array of anchors mentioned in chapter 2).

Subsequently the transistor netlist is rearranged. Initially being a netlist where node names are assigned to transistors, it becomes an ordered netlist where transistors are assigned to nodes. In this new list additional transistor classes are distinguished: besides n- and p-channel types, transistors now are classified as unidirectional to GND or to VDD or as bidirectional between arbitrary nodes. Transistors connected in parallel (three identical terminal nodes) are represented by one transistor with the number of parallel transistors as an attribute. The number of transistor gate terminals connected to a node is an attribute of that node in the new list. The whole netlist rearrangement procedure is mainly performed by subsequently calling a very fast sorting and summing routine.

4. Extracting Transistor Branches

Based on the new netlist, transistors will be combined to branches. A branch is defined to be a signal path between two signal terminals. It consists of transistors arbitrarily connected by their source/drain (S/D) terminals. The corresponding gate terminals of the transistors control the signal flow through the branch according to the logic function determined by the connection of the S/D terminals.

Branches may contain any number of transistors. A single transistor is the smallest possible branch. If two branches have a signal terminal connected to the same node and nothing else is connected to that node, they can be combined to a serial branch. Thus serial branches are constructed by serially connecting transistors and parallel branches. If two branches have both their signal terminals, respectively, connected to the same nodes, then they can be combined to a parallel branch despite of other elements connected to these nodes. Thus parallel branches are constructed of transistors and serial branches. By alternate connections of parallel and serial branches arbitrary complex branches with a hierarchical structure can be defined.

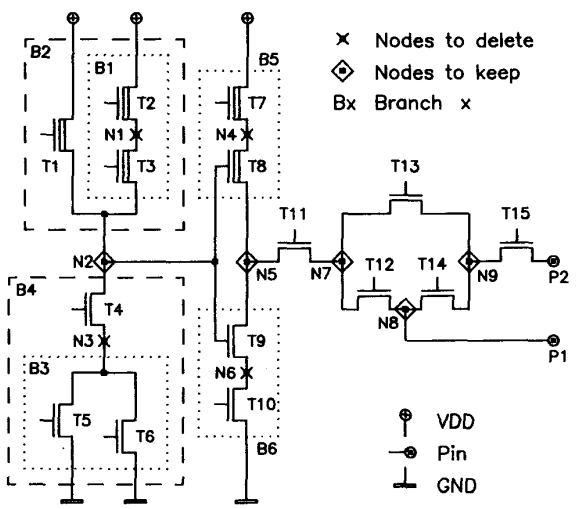

Fig. 1: Combining transistors to branches.

Branches can exist between any two nodes, but due to the combination of simple branches to more complex branches many nodes are becoming obsolete and will be skipped. Nevertheless, there are particular nodes which must be kept, such as nodes connected to transistor gate terminals, nodes defined to be pins, and of course, the GND and VDD nodes.

As can be seen in fig. 1, transistors T2 and T3 can be combined to a serial branch B1, making obsolete node N1. The same holds for branch B5 (T7, T8, N4), for branch B6 (T9, T10, N6), and for branch B4 consisting of transistor T4 and the parallel branch B3 (T5, T6) and node N3. On the other hand, T8 and T9 cannot be combined to a serial branch, because the common node N5 is also connected to a 3rd branch (T11) and therefore must be kept. The transistors between node N5 and pin P2 cannot be combined due to the existence of node N8 defined to be a pin (P1). Hence each of these transistors forms a separate branch.

Three types of branches can be distinguished:
1: unidirectional branches from GND,
2: unidirectional branches from VDD,
3: bidirectional transfer branches between nodes, which are neither GND nor VDD.
All branch types are equipped with a uniform standard propagation delay.

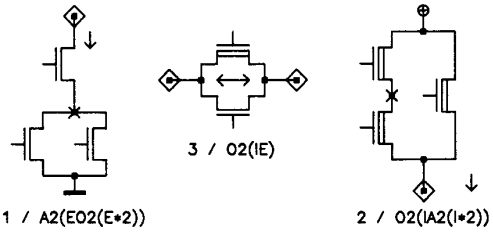

Fig. 2: Characterization of branches by shorthand notation of branch type / logic function.

In fig. 2 branch types precede the slash. The shorthand notation behind the slash represents the logic function of the corresponding signal path. Its meaning is:
Ox: logic OR with x inputs (parallel branch of x elements),
Ay: logic AND with y inputs (serial branch of y elements),
I : inverting input (p-channel transistor),
E : non-inverting input (n-channel transistor),
*z: last preceding element {Ox(..)|Ay(..)|I|E} exists z times.
Brackets are used to separate the logic hierarchy levels.

As can be seen in fig. 3, the extracted branches do not always represent what the designer expects. Instead of showing separately the three tri-state inverters connected to the same bus node, LOGEX first combines each of them to a serial GND- and a serial VDD-branch and then combines all three GND- and VDD-branches, respectively, to single parallel branches.

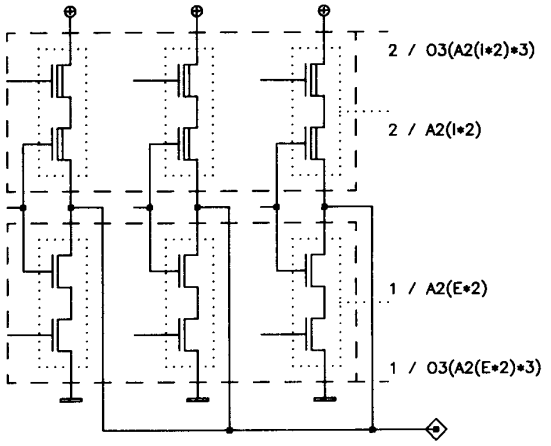

Fig. 3: Branch extraction for three tri-state inverters connected in parallel.

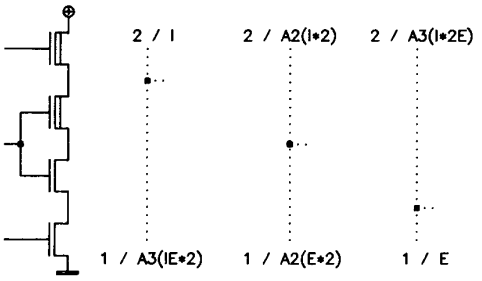

Fig. 4: No defined branch building due to unused output. Symmetrical extraction enforced by control statement.

Fig. 4 shows other strange looking logic functions. Since the output signal of the tristate inverter is unused, the real output node is not known to LOGEX. Thus there exist three possibilities to construct a GND- and a VDD-branch, respectively, but only one of them has the well known symmetrical structure. The latter can be enforced by the control statement not to combine p- and n-channel transistors to a serial branch.

In general, branches are always extracted up to the highest possible logic hierarchy without respect to cell definitions in the designer's mind. The resulting propagation delay will not be augmented by raising the logic hierarchy.

5. Combining Branches to Cells

After all branches have been extracted, a catalog of the different logic functions of the branches is established. For each new logic function a corresponding second logic function is generated, which is the complement of the original function (Ax -> Ox, Ox -> Ax, E -> I, I -> E).

Once the catalog is completed the node list is scanned to find nodes where a GND- and a VDD-

branch are connected together. If these two branches have complementary logic functions as defined by the catalog (e.g. 1/02(E*2) and 2/A2(I*2)) and their control terminals are tied to the same nodes, they can be combined to a real CMOS gate. The resulting logic function is the same as the one of the GND-branch. The VDD-branch is now obsolete and will be skipped. To distinguish the new cell from a normal GND-branch it gets the new branch type 7. The shorthand logic notation will be preceded by an "N" (e.g. 7/NO2(E*2)). If the new gate is an inverter, it gets the new branch type 6. If LOGEX finds that an inverter is fedback by another inverter (e.g. to form a latch), it assigns type 5 to both inverters.

LOGEX is able to compress a chain of inverters into a single inverter by multiplying the standard propagation delay with the sum of inverters in the entire chain. Guided by the number of standard delay times of the new inverter, LOGEX knows if the overall logic function is an inverter or if it has to be changed to a buffer. The same may be done with inverters following gates. The execution of this merging process is initiated by a control statement.

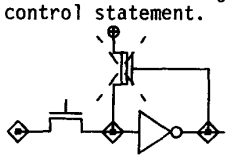

Fig. 5: Level refreshing transistors are deleted.

The next step in LOGEX is the elimination of single transistors between an input of an inverter and VDD (or GND), which are controlled by that inverter's output (fig. 5). Such transistors are used as level refreshers but they don't have any meaning to the logic function.

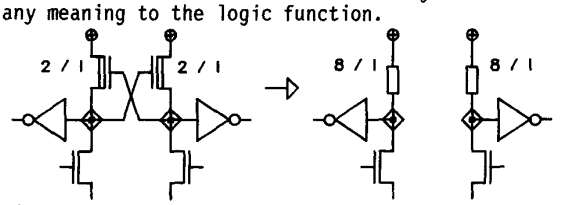

Fig. 6: Conversion of cross coupled transistors to resistive pull-up branches.

Finally, resistive pull-up and pull-down branches (types 8 and 9) will be installed. They are extracted from single transistors to VDD and GND, respectively, which are permanently conducting due to a connection to GND or VDD at their control gate. Cross coupled transistors as in fig. 6 will be converted to the same category of branches. If pull-up (pull-down) branches have no connection to other elements of the circuit, they are deleted. This situation may arise, for example, in automatically generated PLAs where not all product lines are used.

A GND-branch connected to a resistive pull-up branch is treated as a GND-branch connected to its complementary VDD-branch: it is converted to a logic gate (fig. 7). The same holds for a VDD-branch connected to a resistive pull-down branch. For this reason LOGEX could easily be adapted to NMOS technology with its characteristic pull-up branches due to depletion loads.

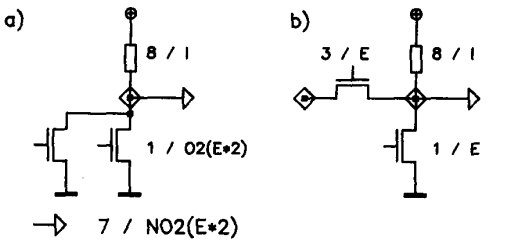

Fig. 7: a) Conversion of pull-up and GND branch, b) no conversion, if a 3rd branch is connected to the same node.

6. Defining Direction of Signal Flow in Bidirectional Branches

As experience shows, all logic simulators have difficulties in simulating bidirectional data flow. For that reason LOGEX tries to determine the direction of signal flow in bidirectional transfer branches (type 3 and type 5).

A couple of rules may be used for this purpose. Two of them are of universal nature and can be applied to all kinds of circuits and design styles, whereas other rules are more restricted in their use. LOGEX will always try the universal rules first. The other rules may be invoked by control statements. Bidirectional branches are changed to unidirectional ones by changing their branch types.

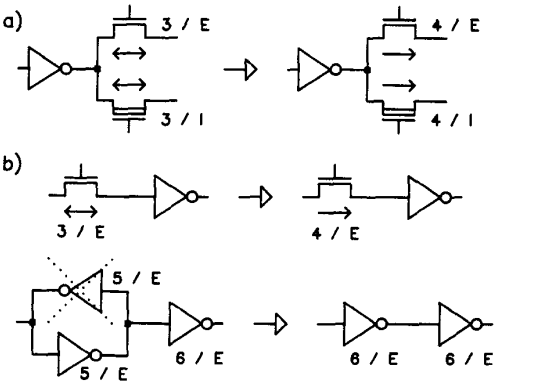

Fig. 8: Conversion of bidirectional branches to unidirectional ones by universal rules.

According to the first universal rule, terminals of bidirectional branches connected to an inverter output or a gate output (branch type 6 or 7) have to be signal inputs (fig. 8a). The second rule defines a terminal of a single bidirectional

branch connected to the inputs of one or more inverters or gates or to control terminals of other branches to be a signal output (fig. 8b). Conversely, if more than one bidirectional branch is connected to these inputs, their signal flow directions stay bidirectional.

To find the direction of signal flow in the remaining bidirectional branches, some special rules may be activated as mentioned before. By these rules the connected elements on both signal terminals of the respective branch are considered. Different kinds of terminals are equipped with different signal strengthes building up seven terminal classes. Although these signal strengthes are not derived from physical transistor data but from a logical point of view, they represent realistic relations in almost all design styles. The terminal classes in decreasing order of signal strength and their codes are as follows:

A : output always active (inverter, gate),
B : terminal of an inverter fedback by another inverter (combination of class A and class E),
AZ: tri-state output of GND-branch or VDD-branch,
BZ: signal terminal of bidirectional branch (type 3),
BA: BZ changed to tri-state output by a direction determining rule,
BE: BZ changed to input by a direction determining rule,
E: input of an inverter or a gate; control input of other branch.

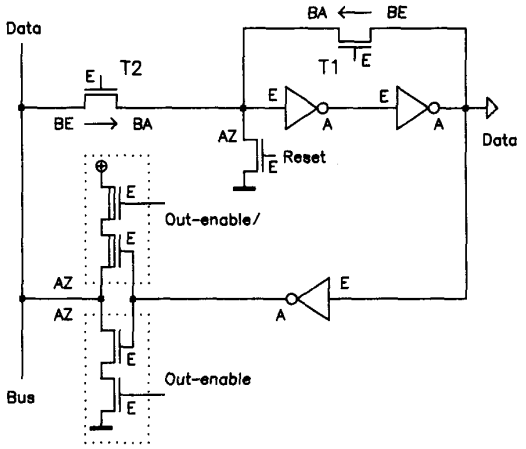

Fig. 9: Determining the direction of signal flow in bidirectional branches by logically assigned signal strengthes.

Terminal classes A and E are used within the two universal rules to change terminals with classes B to classes A and E, and to change terminals with classes BZ to classes BA and BE. The remaining rules are based mainly on the connections of terminals with class BZ to those with class AZ and E. In fig. 9 the signal flow of the load transistor T2 of a resetable latch is defined by having one signal terminal connected to a tri-state inverter (twice class AZ) being stronger than the reset transistor (only once class AZ).

7. Finding Pins and Busses

LOGEX recognizes three categories of pins:

1. Nodes connected only to a terminal of an output class (A, AZ, BA) or to a terminal of an input class (E, BE), respectively, are always identified as pins. If a terminal of a bidirectional class (B, BZ) is connected to a node, this node is declared to be potentially a pin. If a terminal of class BZ is the only connection, then the node will become a bidirectional pin.

2. Nodes will become pins if the corresponding attributes (input, output, bidirectional) are assigned to those nodes by control statements. All pin attributes are examined by LOGEX to be consistent with the attributes LOGEX itself would assign. In case of any inconsistency an appropriate message will be output to the protocol file.

3. All node names defined by the layout designer in the layout - distinguishable by special naming conventions from automatically generated names - are treated as pins by a control statement. LOGEX assigns pin attributes according to terminal classes connected to a pin. Output pins always result from terminal class A, even in combination with other terminal classes, or from terminals of classes AZ and/or BA, if no terminal of an input class (E, BE) is connected to that node. If a terminal of a tri-state output class (AZ, BA) is combined together with a terminal of an input class (E, BE), LOGEX finds a bidirectional pin. The same holds for terminals of class BZ with or without a terminal of an input class (E, BE). Input pins are defined, if only terminal classes E and/or BE are connected to a node.

LOGEX distinguishes three types of busses
1: normal busses, consisting of nodes connected to more than one tri-state terminal (AZ, BZ, BA),
2: pull-up busses, consisting of nodes connected at least to a pull-up branch,
3: pull-down busses, as the reverse of pull-up busses.

After the extraction of pins and busses, LOGEX outputs an analysis of the circuit structures connected to those pins found without the assistance of control statements (pin category 1). This helps the designer in identifying solitary nodes due to possible layout faults like missing or false interconnections. (The analysis module of LOGEX may be used after the extraction process in an interactive manner, enabling the user to analyse arbitrary parts of the layout by specifying

node names or logic functions as a starting point.)

The entire extracted circuit structure is stored in a text file in a coded and easily interpretable form for further processing, even by other computers.

8. Transfer to Logic Simulation

Since the extracted structure contains only primitive logic functions, although hierarchically organized to arbitrary complex overall functions, it is relatively easy adaptable to gate level logic simulators. If the simulator also supports a hierarchically organized simulation model, then in a first step all different cells according to the LOGEX cell catalog - broken down to the lowest hierarchy level - will be supplied once. In the second step the interconnections between the cells together with the cell calls are adapted to the simulator. If the simulator only supports a flat simulation model every cell extracted by LOGEX must be broken down to their lowest level and transfered on this level to the simulator, thus increasing the size of the input data file drastically. In both cases the brake down procedure is carried out by LOGEX.

Thus the only simulation elements a logic simulator has to know are:
- AND, OR, NAND, NOR gates with arbitrary number of inputs,
- inverters and buffers,
- unidirectional (and possibly bidirectional) tri-state buffers,
- pull-up and pull-down resistors or equivalent constructs,
- busses,
- input, output, and bidirectional pins.

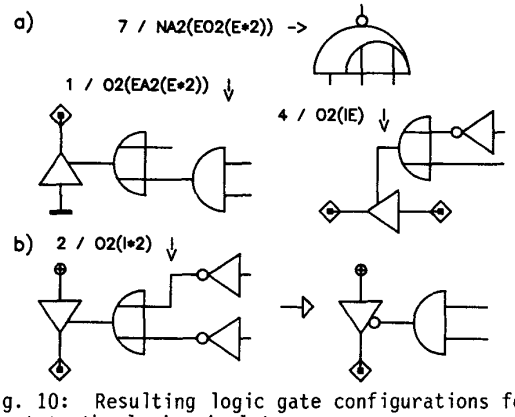

Fig. 10: Resulting logic gate configurations for input to the logic simulator.

Fig. 10 shows some examples for the transfer of cells from the logic function description used in LOGEX to a gate representation for a logic simulator. As can be seen in fig. 10b, it is possible to reduce the total number of gates by using the "de Morgan" theorem in cases where the inputs of a cell are mainly inverting inputs.

If the simulator is able to process timing information, then the primitive gate of the highest hierarchy level of a cell is supplied with the propagation delay. All primitive gates of lower hierarchical level are supplied with zero propagation delay.

For a realistic logic simulation of dynamic CMOS circuits an adequate modelling of the node storage effect is necessary which has to be assured by the logic simulator.

In our case LOGEX is coupled to PROSIM, a proprietary simulation system of our company. PROSIM is able to handle only two hierarchy levels where the lower level is attached to the library. Thus the library is only used to hold the primitive logic functions listed above. The cells extracted by LOGEX are input to PROSIM in a flat manner as described above. PROSIM processes the timing information from LOGEX and models the node storage effect. It has the capability to handle up to 200,000 primitive gates sufficient for todays VLSI circuits.

9. Conclusion

A computer program has been described, which is able to extract a gate level logic description out of a transistor netlist for CMOS VLSI circuits. The extraction process can be handled fully autonomously without any information from a cell library or from the user. All necessary information is directly derived from the transistor connectivity. Nevertheless, the user may interact with the program, mainly in defining pins and in activating rules to determine the direction of signal flow in physically bidirectional branches.

The program is optimized with respect to computer memory consumption and execution speed. VLSI circuits up to 110,000 transistors have been extracted using an addressing space of 8 Mbyte including the program code. The run time for 110,000 transistors on a Siemens 7571 mainframe is 1,700 CPU sec for the whole extraction process, from reading the transistor netlist to writing the gate level description for the logic simulator.

References

[1] A.L. Spickelmier, R.A. Newton: "WOMBAT: A New Netlist Comparison Program", ICCAD-83, Santa Clara, 1983

[2] C. Ebeling, O. Zajicek: "Validating VLSI Circuit Layout by Wirelist Comparison", ICCAD-83, Santa Clara, 1983

[3] R.E. Bryant: "MOSSIM: A Switch-Level Simulator for MOS-LSI", 18th DAC, Nashville, 1981

[4] W. Nebel, R.W. Hartenstein: "Functional Design Verification by Register Transfer Net Extraction", COMPEURO '87, Hamburg, 1987