YAGLE, a Second Generation Functional Abstractor for CMOS VLSI Circuits

A. Lester, P. Bazargan-Sabet, A. Greiner

LIP6/ASIM Laboratory
Université Pierre et Marie Curie (Paris 6)
4, Place Jussieu, 75252 Paris Cedex 05

E-mail: anthony.lester@lip6.fr

ABSTRACT

This paper presents a new functional abstraction tool for CMOS VLSI. The tool uses a procedure called circuit disassembly in order to extract an oriented gate netlist from a transistor netlist. Logic equations are then generated for these extracted gates in order to produce a VHDL data-flow description for a circuit. This tool combines an advanced functional analysis technique with subgraph isomorphism algorithms in order to handle the widest possible number of circuit styles with a minimum of user intervention.

1. INTRODUCTION

1.1 Context

The continual increase in the complexity of integrated circuits poses a consequently heavier burden on the verification stages of the design process. In addition, the race to put new designs onto the market means that all verification must be performed quickly, reliably, and with a minimum of user intervention. This paper describes a procedure for the post-layout functional verification of CMOS VLSI designs, which satisfies these criteria.

The principle of functional verification is a comparison of the behaviour of the synthesised transistor net-list, obtained by extraction from the layout, with the behaviour of the initial specification. Straightforward switch-level simulation of the transistor net-list rapidly becomes unreasonable for large designs. To resolve this problem, it is necessary to obtain, from the transistor net-list, a set of logic equations, which entirely describe its behaviour. This allows the simulation to be performed at a higher level with corresponding speed gains. In addition, this also opens the door to formal proof comparison methods.

1.2 Objectives

The procedure of functional abstraction can be divided into two stages. The first stage is the extraction of an oriented gate net-list from the unoriented transistor net-list. The second stage is the generation of Boolean logic expressions for each of these gates in order to produce a behavioural description of the entire circuit.

It is the first stage, known variously as gate extraction [1], or circuit disassembly [2], which costs the most computing time. Two basic approaches have been used to tackle this problem. The first approach uses pattern-matching techniques to identify predefined gates within the transistor net-list. This set of predefined gates is usually constituted from a user-defined library [3][4]. This technique has the advantage of flexibility in that the user is able to tailor the tool to handle a particular design style. However, the technique suffers from the disadvantage of not being automatic, and from the fact that the user is required to have a priori knowledge of the circuitry of the implemented design.

The second approach can be defined as a partitioning of the transistor net-list, according to a limited number of generic rules. Each partition represents an extracted gate for which a behavioural description can be deduced [2]. If the rules are well defined, this approach has the important benefit of being as generic as possible without user-intervention. However, the treatment of complex design styles such as precharged logic or pass-transistor logic can use a significant amount of computing time and memory.

This paper presents a method based on the approach presented in [2], with a new algorithm designed to cope with larger and more complex designs. This is combined with functionality based on the first approach in order to handle complex synchronous components and analogue components within a circuit.

2. GENERAL METHOD

2.1. The Cone Model

The partition model used in our approach is the same as used in [2]. The idea is to partition the circuit into extracted gates such that the states of all the circuit nodes connected to at least one transistor gate are defined. In order to achieve this objective, an object referred to as a cone is built for each of these nodes. A cone is defined as the set of paths for current flow between the power supplies and the node on which the cone is built. Figure 1 illustrates this for a simple circuit.

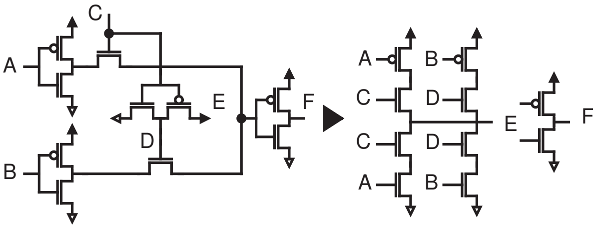

Figure 1: Transistors to Cones

The result of the partitioning of the circuit is a set of cones (completely defining the state of all transistor gates and drivable external connectors); each cone containing a set of branches (paths).

The connectivity of the cone netlist is established by adding two lists of cones to each cone:

- The INCONE list: the set of cones that attack one or more gates of the transistors of the cone.
- The OUTCONE list: the set of cones that contain the cone within their INCONE list.

The branches allow the local functionality of the cone to be readily analysed. This functionality is expressed by a pair of Boolean equations:

$$S_{up} = \sum_{VddPaths} \prod_{TN_i} x_i \prod_{TP_j} \bar{x}_j \qquad (1)$$

which defines the condition for which the output is set to one.

$$S_{down} = \sum_{VssPaths} \prod_{TN_i} x_i \prod_{TP_j} \bar{x}_j \qquad (2)$$

which defines the condition for which the output is set to zero.

where x corresponds to the signal on the transistor gate.

If

$$S_{up} \cdot S_{down} = 0 \qquad (3)$$

then the cone is locally non-conflictual since there exists no input pattern which can activate an up path and a down path simultaneously.

Conversely, if

$$S_{up} + S_{down} = 1 \qquad (4)$$

then the cone is locally non-tristate since there exists no input pattern which activates neither an up path nor a down path.

Note that the non-satisfaction of equations (3) and (4) does not imply that the cone actually corresponds to a high impedance or a conflictual node, since the preceding logic can render impossible the direct input combinations resulting in these states.

2.3. General Disassembly Algorithm

This process is a significant evolution of that presented in [2]. The aim of the disassembly is to generate a fully connected, functionally correct cone netlist for the entire circuit. In order treat more complex circuits in as near as possible linear time, a new, two stage algorithm has been developed.

The two stages can be summarised as follows:

1) Extraction of all simple CMOS dual cones along with initialisation of all remaining cones.
2) Construction of all remaining cones. A cone is built only if the surrounding logic has already been extracted. The surrounding logic is exploited during branch construction to prevent false branches.

Stage 1 - CMOS Dual Extraction

Our target is CMOS VLSI circuits, these invariably contain a high proportion of simple CMOS dual gates. We define simple CMOS as the gates that meet the following criteria.

1) Up branches contain only P-type transistors, and down branches contain only N-type transistors.
2) No internal node of the resulting cone attacks the gate of a transistor.
3) Equations (3) and (4) are satisfied.

Stage 2 - Context Dependant Cone Construction

The fundamental problem to resolve for all the remaining cones is that of false branches. This is illustrated in figure 2.

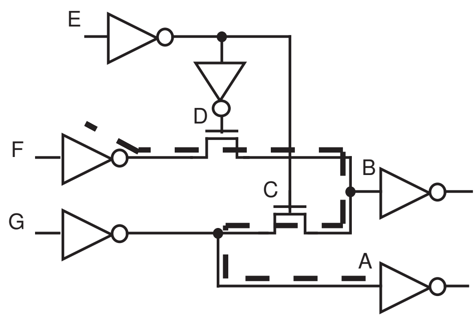

Figure 2: Example of a False Branch

The example of Figure 2 shows a false branch which would be built for the cone A, if knowledge of the mutual exclusion of signals C and D were not taken into account.

In order to exploit the surrounding logic during the cone construction phase, it is first necessary to verify that sufficient logic has already been extracted. This is achieved by the attempted creation of the local dependancy graph (LDG) for the cone under construction. An example of this graph is shown in figure 3.

The LDG is a single root directed acyclic graph, distinct for each cone, it should be noted that:

1) The graph is correct, apart from the direct input arcs to the root node, necessarily an approximation, obtained from an initial traversal of the transistor netlist, since the cone is not yet built.

2) The construction of the graph is limited by a depth parameter, defined, for a node within the graph, as the longest path between that node and the root node.
3) If the construction of the graph is only limited by anything other than a node of an unfinished cone, at less than the maximum depth, then the graph is considered complete.

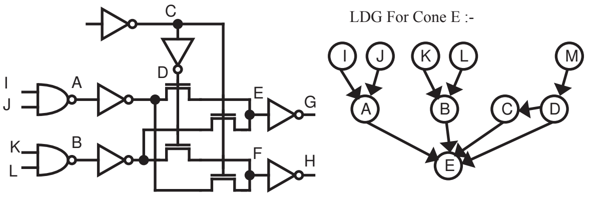

Figure 3: The Local Dependancy Graph

If the LDG is not complete, then construction passes to another cone, the original cone being dealt with in a subsequent iteration as soon as its LDG is complete.

Once a complete LDG has been obtained, the smallest set of variables required to express the state of the root node must be obtained, this is the set of cone primary variables. This set can be readily obtained from the LDG, since it corresponds to the articulation points closest to the root node.

Defn.: An articulation point, in a single root directed acyclic graph, is a node such that no node upon which it depends has a path between itself and the root node which does not pass through this node.

The cone is then built by "growing" the branches recursively, link by link, as illustrated in figure 4. This process is optimised by the creation of the two-level graph shown in figure 4, which relates the cone direct inputs to the primary variables. This graph is maintained dynamically during the branch construction phase.

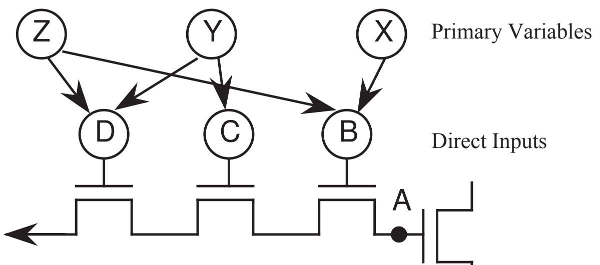

Figure 4: The Construction of a Branch

At each addition of a new link to a branch, the updated graph is consulted to verify any correlations with preceding links. Correlated direct inputs are expressed in terms of the primary variables, and the product of the resulting expressions is calculated. A value of zero for the product indicates that the branch under construction is false. The construction of this branch is therefore abandoned.

The global functionality of the resulting cone is verified by expanding equations (1) and (2) in terms of the primary variables, before verifying the conditions of equations (3) and (4).

3. PATTERN MATCHING

3.1. The Approach

The algorithm of the preceding section works efficiently for combinatorial circuitry. However, any real circuit contains components such as sequential elements or even analog blocks for which the functionality cannot be derived algorithmically.

The number of forms of these types of components is generally limited in any circuit. This fact justifies a pattern-matching approach for which the forms and their associated functionality are user-specified.

To implement this, a development of the SubGemini [5] algorithm was used. The user specifies the transistor netlists of the components to be identified, along with their associated behavioural descriptions. All instances of these components are then identified in the transistor netlist of the circuit, prior to the general disassembly stage. The transistors of each instance of a component are marked as corresponding to an instance of that component, so that they can be ignored during the cone extraction stage.

The overall behaviour of the circuit is therefore composed of a combination of the specified behaviours of the pattern-matched components, with the generated behaviours of the combinatorial components.

3.2. The Identification Algorithm

The detection of a specified transistor netlist within the circuit is a two-phase algorithm based on the SubGemini [5] method. It uses an initial phase to rapidly detect possible instances within the circuit, before employing a more computationally intensive phase to verify each possible instance.

Phase 1:

The first phase is identical to that of SubGemini [5]. It employs a simple iterative relabelling technique to rapidly extract a limited list of candidates for the component being searched. The circuit and subcircuit to be identified are considered graphs as shown in figure 5, where both signals and transistors are nodes.

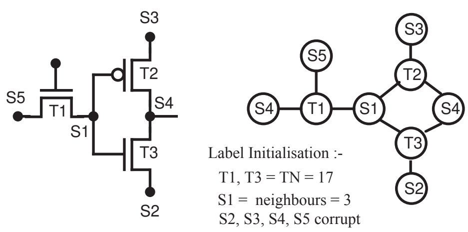

Figure 5: Circuit Graph and Labelling

Node labels are initialised in the circuit and the subcircuit as in figure 5, with the external signals of the subcircuit being marked as corrupt. The nodes are then relabelled iteratively according to equation (5), with neighbours of corrupt nodes in the subcircuit being marked as corrupt.

$$L(X_{i=n+1}) = L(X_{i=n}) + \sum_{all\ N} W_{NX} \cdot L(N_{i=n}) \qquad (5)$$

L(X) represents the label of X.

N represents the neighbours of X.

i represents the iteration count.

W represents a connection weight chosen to maximise the dispersion between different subcircuits.

The iteration is stopped when relabelling would render all subcircuit nodes corrupt. The subcircuit node whose label has the least number of instances in the circuit is retained as the key node, and all circuit nodes with that label are retained as candidates for the second phase.

Phase 2:

The second phase is an original algorithm, which verifies each candidate node, by attempting identical traversal of the circuit and the subcircuit in a similar way to that used in [6]. However, the problem of backtracking from an incorrect traversal is avoided by maintaining a matrix of all possible solutions during the traversal. Possible solutions are created and deleted dynamically as the circuit and subcircuit are traversed simultaneously. If any solutions remain, once all nodes of the subcircuit have been traversed, then the candidate is verified. Note that it is possible for more than one solution to remain, for example in the case of symmetric subcircuits. In this case, the solution choice is arbitrary.

4. RESULTS

The algorithms described have been fully implemented in 'C' in a program distributed for evaluation as part of the ALLIANCE CAD System.

A number of results for this hybrid approach are presented in table 1. The results are from execution on a Sun UltraSparc 1 workstation with 64Mb RAM. Table 2 shows the amount of execution time used for the pattern-matching stage for the same circuits. The behavioural descriptions obtained were validated by simulation.

Circuit	Number of Transistors	Number of Gates	Time
I2CCORE	956	227	0.1s
DLXm	37,794	7,823	1m37s
4x128 RAM	4,510	1,368	54s
4x512 RAM	13,917	4,462	9m27s
PCIDDC	202,264	54,217	40m41s

Table 1: Total execution times

The program is able to cope with significantly sized blocks of over 100,000 transistors in reasonable time. However, the architecture has a significant influence on the execution times, as is shown by the results for the RAM.

For the pattern-matching stage, it can be seen that a relatively small number of models is required to cope with the circuits. The simplicity of the phase 1 SubGemini algorithm renders the subcircuit detection stage extremely rapid.

Circuit	Number of Models	Instances Detected	Detection Time
I2CCORE	6	23	0.1s
DLXm	6	1266	7s
4x128 RAM	3	528	0.8s
4x512 RAM	3	2084	3.3s
PCIDDC	3	6515	50s

Table 2: Time for subcircuit detection

5. CONCLUSION

The tool demonstrates the efficiency of a hybrid approach to functional abstraction. The combination of pattern-matching for the detection of a small number of complex gate types, with automatic treatment of the majority of gates, results in a tool which comes close to satisfying the demand for fully automatic generation of a behavioural description from a transistor netlist. This description must be sufficiently robust to allow formal proof or simulation verification of the synthesised netlist.

References

[1] M. Boehner, "LOGEX - An Automatic Extractor from Transistor to Gate Level for CMOS Technology", *25th ACM/IEEE Design Automation Conference*, pp. 517-522, June 1988.

[2] M. Laurentin, A. Greiner, R. Marbot, "DESB, a Functional Abstractor for CMOS VLSI", *IEEE European Design Automation Conference*, 1992.

[3] T. Kostelijk, B. De Loore, "Automatic Verification of Library-Based IC Designs", *IEEE Journal of Solid-State Circuits*, March 1991.

[4] J.-B. Guignet, "Abstraction Fonctionnelle des Composants VLSI", PhD Thesis, Université Pierre et Marie Curie, March 1998.

[5] M. Ohlrich, C. Ebeling, E. Ginting, L. Sather, "SubGemini: Identifying SubCircuits using a Fast Subgraph Isomorphism Algorithm", *30th Design Automation Conference*, June 1993.

[6] F. Lellau, T. Hoepken, E. Barke, "A Technology Independent Block Extraction Algorithm", *21st Design Automation Conference*, June 1984.

[7] A. Greiner, F. Pécheux, "Alliance: A Complete Set of CAD Tools for Teaching Digital VLSI Design", *3rd Eurochip Workshop on VLSI Design Training*, pp. 230-237, September 1992.