A Graphical System for Hierarchical Specifications and Checkups of VLSI Circuits*

B. Becker Th. Burch G. Hotz D. Kiel R. Kolla P. Molitor H.G. Osthof G. Pitsch U. Sparmann

Fachbereich Informatik
Universität des Saarlandes
D-6600 Saarbrücken, FRG

ABSTRACT

The two most important frontend components of the VLSI design system CADIC are presented. The first one allows graphical specification of recursively defined circuits. The other one allows the designer to navigate across the synthesized layout following the hierarchical specification to check e.g. CADIC's hierarchical optimizations or to control the outcome of test (generation) algorithms.

1. Introduction

In this paper we introduce the two most important frontend components of the VLSI design system CADIC, one allowing graphical specification of recursively defined circuits and the other allowing the designer to navigate across the layout (automatically generated by the CADIC system) and to easily extract information (statistics) about the synthesized subcircuits, e.g. whether the hierarchical optimizations work well in a specified subcircuit or in which subcircuits there are untestable faults.

CADIC is a VLSI design system (see figure 1) based on an algebraic approach well suited for hierarchical design of integrated circuits. Other main features in addition to the graphical components described in this paper are the compact internal representation of large regular design objects and the hierarchical processing of each individual step of the automated synthesis (e.g. layer assignment [14,17,19], generation of power supply nets and their minimal area sizing [12], routing [13,15]). A circuit description on CADIC's design level, which is briefly introduced in the next section, consists of behavioral as well as topological aspects without touching physical details. This design level admits – in contrast with RELACS [3] e.g. which is oriented to purely geometrical operations – recursive design operations in a topological framework and so makes it possible for a hardware designer to *easily* specify regular circuits in a very compact way (see section 3).

In today's VLSI design systems a circuit is specified by its logic plan. The input of the logic plan is realized graphically on a CAD workstation. To reach a higher degree of flexibility, some design systems make some special cell generators available to the designer. The idea of generators is that cells can be parameterized by its input width or by its function and the design system generates them automatically. Characteristic representatives of such cells are counters, PLAs, RAMs and ROMs. However, in these systems designers have not the possibility of parameterizing their own logic plans, i.e. building generators of their own (see e.g. the Siemens design system VENUS [10] or the ES2 design system [†]).

CADIC's component for graphical circuit specifications allows – in addition to descriptions of nonregular logic – recursive definitions of families of regular circuits, which are based on simultaneous cell- and edge-replacement operations. The graphical description of a family of 2^k-bit multipliers will be introduced for illustration in section 3. Because CADIC is based on a well studied algebraic calculus (see [18]) it is guaranteed that there are no ambiguities in how to refine nets. This is the main difference to other such systems as for instance ESCHER [5], which is the only system but CADIC, known to the authors, allowing graphical recursive definitions of circuits.

Subsequent to automated synthesis the layout of the specified circuit is analyzed. In parts this is done automatically (e.g. design rule check), in parts it must be done 'by hand' (e.g. checkups of optimization results, if the outcome of a heuristic is not satisfac-

*This work was supported by the Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 124 "VLSI Entwurfsmethoden und Parallelität"

[†] see Document ES2-014-0013/0066-0068 published by ES2 European Silicon Structures

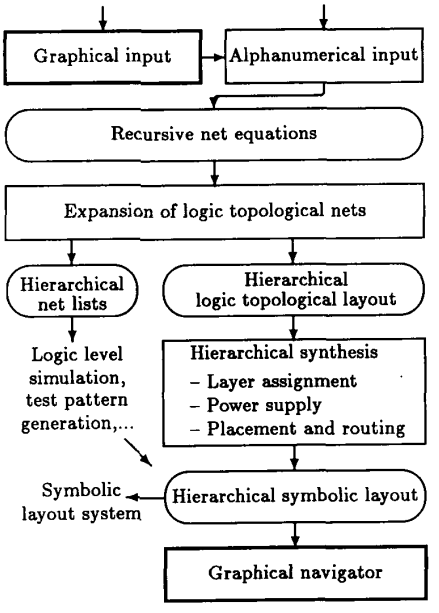

Figure 1 *The CADIC system*

2024/90/0000/0174$01.00©1990 IEEE

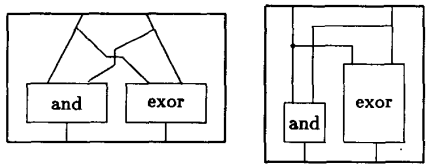

Figure 2 *Two logic topographical nets which belong to the same lt-net*

tory). The manual checkups are supported by CADIC's graphical navigator which allows going step by step top down in the layout following the hierarchical specification of the circuit. In each step the frames of the different subcircuits are fitted with information and statistics so that the designer easily finds the crucial positions in the layout. A more precise presentation of this graphical component is given in section 4.

CADIC has been developed at the Universität des Saarlandes. It is implemented on VAX/UNIX and SIEMENS/BS2000. APOLLO DN3000 and MX2 APC9732 serve as frontend hardware. The whole software written in COMSKEE and C comprises of about 80000 lines of code of which about 20000 lines fall to the both graphical components. CADIC is exhaustively used during practical studies in VLSI design (see e.g. [6,8,9]). A generator for floating point multipliers [6] which has been specified using the above graphical input component will serve as illustration in this paper. Given m the length of the mantissa and e the number of bits in the exponent an (m,e)-bit floating point multiplier is generated. The (24,8)-bit floating point multiplier which contains about 70000 transistors has been generated and synthesized. Test patterns were generated. Since the fault coverage was not satisfactory, the results of the test pattern generator had to be analyzed. With the aid of the graphical navigator it was very easy to find the crucial part of the design (i.e. the subcircuit containing most of the untested faults).

In this paper we concentrate on CADIC's graphical components. The system itself and its theoretical background will only be presented as far as necessary. More detailed information about it can be found in [2,11,15,18].

2. The design level of CADIC

Circuits are laid out into a rectangle R. In order to suppress geometrical and physical details of manufacturing processes and to become independent of technology that way, we forget the width and the layer of wires. While doing so, wires become simple lines, which may branch and cross each other. Furthermore, we suppose that the circuit is constructed of cells computing digital values. Assuming that these cells have a physically correct design, we forget their internal structure and size and only maintain the order of external connectors on their boundary. By considering crossings and junctions of wires as cells, which perform crossings and junctions of signals, this abstraction results in a planar arrangement of cells in the plane whose interconnections consist of crossing-free non-overlapping lines (see figure 2). We call such a structure *logic topographical* net.

To suppress precise geometrical relations of this abstract layout, we consider two such layouts to be equivalent iff they can be transformed into each other by a sequence of deformations such as deformation of wires, translation and stretching of cells, and order preserving translation of connectors along the side of a rectangle, i.e. deformations which maintain the planar topological structure of the layout. Figure 2 shows two abstract layouts which we consider to be equivalent. A maximal set of equivalent layouts is called *lt-net* (*logic topological* net).

Each wire segment is related to a type $t \in T$. For an lt-net α we denote the sequence of the types of the connectors on the northern and southern side of α (from left to right) by $N(\alpha)$ and $S(\alpha)$, the sequences on the eastern and the western side (from top to bottom) are denoted by $E(\alpha)$ and $W(\alpha)$. If the cardinality of T is equal 1, $N(\alpha)$, $S(\alpha)$, $E(\alpha)$ and $W(\alpha)$ denote the number of pins on the northern, southern, eastern and western side. A type is an attribute of a wire, e.g. a wire of type $\underline{7}$ could represent 7 parallel wires (of type $\underline{1}$).

3. Graphical design of recursively described lt-nets

By allowing simultaneous cell- and edge-replacement operations, large regular lt-nets can be described in a way, which can be surveyed easily. We will demonstrate this fact by recursively specifying a family of multipliers for binary numbers which is part of the (m,e)-bit floating point multipliers already mentioned in the introduction. This type of multiplier is a tree-multiplier, which is a modified version of a Wallace tree multiplier [22] made suitable for VLSI design by Luk and Vuillemin [16].

A first step in the design is the following one: Let $a = (a_{n-1},\ldots,a_0)$, $b = (b_{n-1},\ldots,b_0)$ be two binary numbers. The product of a and b is equal to the sum of the n binary numbers of length $2n$ represented by the n lines of the following matrix P_n, where a_ib_j denotes the *logical and* of the bits a_i and b_j, if $a_{2n-1} = a_{2n-2} = \ldots = a_\cdot = a_{-1} = a_{-2} = \ldots = a_{-n+1} = 0$:

$$\begin{pmatrix} a_{2n-1}b_0 & a_{2n-2}b_0 & \cdots & a_0b_0 \\ a_{2n-2}b_1 & a_{2n-3}b_1 & \cdots & a_{-1}b_1 \\ \vdots & \vdots & \vdots & \vdots \\ a_nb_{n-1} & a_{n-1}b_{n-1} & \cdots & a_{-n+1}b_{n-1} \end{pmatrix}$$

Matrix P_n gives us the idea to compose an n-bit multiplier by $2n$ identical columns (or n identical rows). In this way, we obtain the first three recursive equations. Note that all the equations are described by figures and can be fed as figures in the computer.

(1) An n-bit multiplier is given by the circuit $c[n,2n]$ (see figure 3), where $c[n,i]$ consists of i columns of an n-bit multiplier.

(2) To simplify presentation let us assume that $n = 2^k$ for some $k \geq 1$. Then $c[n,i]$ can be composed of two instances of $c[n,\frac{i}{2}]$ and is graphically described as shown in figure 4.

It follows that parallel to the replacement of cell $c[n,i]$ in a circuit each wire segment of type $k[i]$ is replaced by two wire segments of type $k[\frac{i}{2}]$.

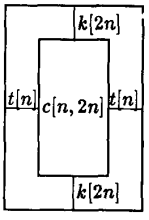

Figure 3

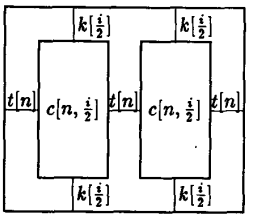

Figure 4

(3) The equation that $c[n,2]$ consists of 2 columns of an n-bit multiplier which are denoted by $col[n]$ serves as base of the induction (figure 5).

The multiplier we design is a partial one, i.e. the output of the circuit will be two $2n$-bit numbers whose sum is equal to the product of a and b. Since the semantics of the multiplier construction is exhaustively described in [1] and is not the main point in this context, we only give a vague idea of it and we concentrate on the topological description of a column, which can be described by the following three productions. In order to understand the construction better, we start with the description of a column of a 2-bit multiplier.

(4) Since matrix P_2 consists of only two rows defining two 4-bit numbers whose sum is equal to the product of a and b, one has only to generate this matrix, so that a column $col[2]$ of a 2-bit multiplier is given e.g. by the lt-net shown in figure 6. (The outputs of the cells are marked by thick lines on the border.) Wire type **a** (**b**) represents a wire belonging to the a-operand (b-operand) and **r** represents a result. The replacement of cell $col[2]$ is accompanied by the replacement of each wire of type $t[2]$ by four wires of type **b**, **a**, **b** and **a** and the replacement of the wires of type $k[1]$ by three wires of type **a**, **r** and **r**.

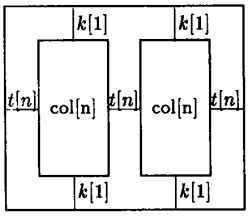

Figure 5

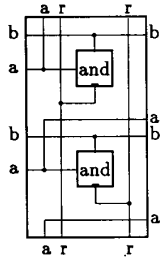

Figure 6

(5) Note that each row of matrix P_n represents a $2n$-bit number. The idea of the tree-multiplier is to reduce the total number of rows in each step by half. Locally this is done by the following lt-net CSA4to2 (The carry output of the fulladder cell FA is located on the western side of its border) (see figure 7). A row of $2n$ such cells is used to reduce four numbers to two.

(6) Now, a column $col[n]$ of an n-bit multiplier can be recursively defined as shown in figure 8.

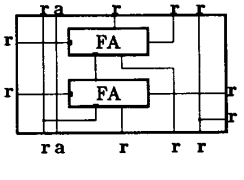

Figure 7

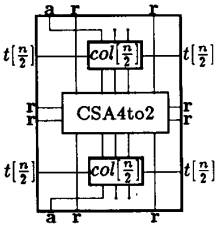

Figure 8

CADIC is fed with these figures. In a first step they are automatically converted into a more suitable form for further processing, namely into 'algebraic net expressions' which serve as interface between the graphical input and the remaining part of CADIC. (An exact description of the system for graphical circuit specifications and its facilities is given in [4].)

Theoretical background and resulting problems

It is obvious that the parallel replacement of cells and edges is not allowed in any case. The necessary conditions which must be fulfilled are easily drawn from the algebraic base exhaustively investigated in [18] so that there are no ambiguities in how to refine nets.

Owing to limited space, we will only give a brief outline of the problems arising in this context. Let $NET(A,T)$ be the set of the lt-nets composed of cells from the cell library A and of wire segments of types from T. We can define two partial binary operations ' $\oplus$ ' and ' $\ominus$ ' on $NET(A,T)$. The operation ' $\oplus$ ' (' $\ominus$ ') constructs a new lt-net by placing the nets upon each other (side by side), if and only if the specifications of the pins on the relevant sides match, i.e. $\alpha \ominus \beta$ is defined iff $E(\alpha) = W(\beta)$, and $\alpha \oplus \beta$ is defined iff $S(\alpha) = N(\beta)$.

A refinement of a lt-net formally corresponds to a homomorphism $\varphi_2 : NET(A,T) \to NET(A,T)$ with $\varphi_2(\alpha \circ \beta) = \varphi_2(\alpha) \circ \varphi_2(\beta)$ ($\circ \in \{\ominus, \oplus\}$) and a monoidhomomorphism $\varphi_1 : T^* \to T^*$. φ_1 defines the refinement of the signal types, φ_2 the refinement of the lt-nets themselves. The condition which must be fulfilled is that the refinement of each cell $x \in A$ is compatible with the refinement of the wire types, i.e. $N(\varphi_2(x)) = \varphi_1(N(x))$, $S(\varphi_2(x)) = \varphi_1(S(x))$, $W(\varphi_2(x)) = \varphi_1(W(x))$ and $E(\varphi_2(x)) = \varphi_1(E(x))$.

One problem which now has to be solved in this context is the following: Let $t \in T$ be a (nonterminal) wire type, and let $\varphi_1(t) = t_1 t_2$ be its refinement. Then the refinement of a southwest knee of type t is obviously as shown in figure 9, because the pins of a lt-net are enumerated in the north and south from left to right and in the east and west from top to bottom. (A different arrangement about the enumeration of pins results in difficulties with the refinement of other 'wire segments'.) Often such a refinement of a wire segment containing a crossing is not intended. Therefore for any signal type t we introduce the reversed type $\bar{t}$, and we allow to connect wires one having type t and the other having type $\bar{t}$. Under the arrangement that the refinement of $\bar{t}$ is the reversed refinement of type t, the construction given in figure 10 is allowed. By this arrangement refinements of junctions can be controlled, too.

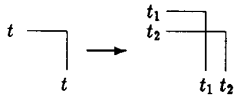

Figure 9

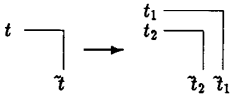

Figure 10

4. Graphical navigator

Next to the graphical design of a family of circuits, e.g. the family of the n-bit tree multipliers, the designer tells the system the circuit he wants to construct, e.g. the 32-bit multiplier. The system *expands* this special circuit following the recursive net equations. It results in a *hierarchical* logic topological layout (see figure 1). CADIC's main feature is that all components work *hierarchically*. What does this mean? Consider e.g. the above recursive specification! A 32-bit multiplier is composed of 64 col[32]-subcircuits, 128

col[16]-subcircuits, 256 col[8]-subcircuits, 512 col[4]-subcircuits and 1.024 col[2]-subcircuits. The circuit CSA4to2 occurs about 1.000 times. In order to take advantage of hierarchy given by systems of net equations the notion of a *box* is introduced. Each variable on the left side of a net equation can represent a box. In the above example CSA4to2, *col*[2], *col*[4], ..., $c[2,2]$, $c[2,4]$,... are such candidates. Now, if such a variable is declared to be a box it occurs only once in the internal representation of the whole circuit, e.g. if *col*[8] is box then the internal representation of *multiplier*[32] does not contain the *col*[8]-subcircuit 256 times but only once. Let $B(R)$ be the set of the boxes of the internal representation R. Then ***hierarchical processing*** means that attempts are being made to handle all occurrences of a box $b \in B(R)$ in the same manner, e.g. they are identically optimized. By this means, the internal representation only increases by a constant factor during synthesis which allows further (hierarchical) processing.
It is obvious that the quality of the optimizations during synthesis is impaired by hierarchical processing. There are trade-offs between 'compactness of the hierarchy' and 'quality of the result', i.e. the compacter the representation of the circuit, the worse the quality of the hierarchical optimizations. This trade-off has been investigated for the via minimization problem during layer assignment [14,17]. It has turned out that • the 'quality of the result' is not inversely proportional to the size of $B(R)$, • the trade-off is 'discontinuous', i.e. it can happen that a hierarchical optimization works *considerably* better after having manipulated $B(R)$ only a little bit, e.g. by removing only one box out of $B(R)$, • there often exists a very compact representation requiring only a few more vias than the 'flat circuit' ($B(R) = \emptyset$) does.
So in general, to obtain good optimization results, the designer has to look for the crucial boxes of the internal representation and soften up the hierarchy at these positions.
Parts of the analysis of the synthesized layout have to be done manually. In addition to checkups of hierarchical optimizations we will speak here about checkups of fault coverages during test pattern generation. The graphical navigator whose features will be illustrated with the example of the 8-bit tree multiplier in the next paragraph effectually supports this job.

Features of the navigator

After the navigator has been activated, the frame of the 8-bit multiplier appears on the screen. Now there are several operations possible. The 3 most important ones are illustrated here by examples.

Partial Expansion With the aid of this operation the designer can descend step by step top down in the layout following the hierarchical specification. An arbitrary box (hierarchical view of a subcircuit) can be selected by ticking it off and expanded a specified number of steps. Figure 11 (upper left corner) shows the view of the 8-bit multiplier after expanding it 3 times. In this case the four dashed boxes represent the four subcircuits $c[8,4]$. The boxes can be fitted with information and statistics about optimizations and/or test coverage.

Zooming If the designer wants to check details of the layout, the navigator offers the possibility of blowing up parts of the layout. Then he can go on descending top down in the layout. Figure 11 (on the right) shows the picture we get by blowing up the rightmost $c[8,4]$-subcircuit and expanding it 3 times. (Note that the Zooming operation is not restricted to the borders of a box but can be applied to an arbitrary part of the current view of the circuit.)

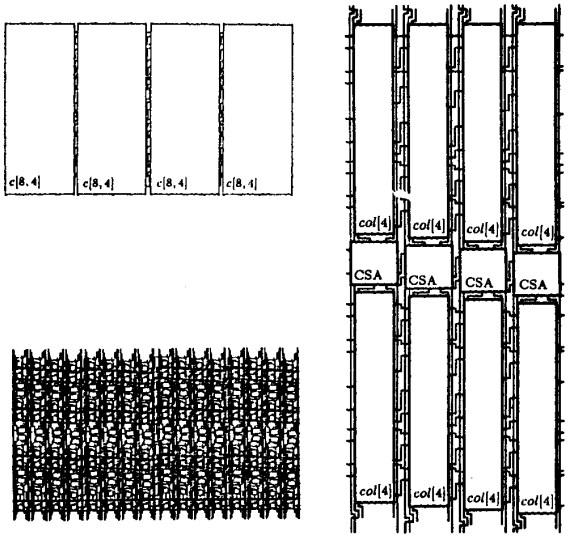

Figure 11 *Illustration of the features of the navigator*

Full expansion This operation expands the specified box completely. The full expansion of the 8-bit multiplier is shown in the lower left corner of figure 11. Details of this circuit can be viewed by using the Zooming operation.
At any moment the navigator knows the references between the layout drawn on the screen and the internal representation of the complete circuit. It follows that if the designer ticks off a cell in the layout, the navigator can tell the path in the initial specification of the lt-net and/or in the internal representation.

An application: Checkups of heuristics

We take up again the trade-off 'compactness of the hierarchy' and 'quality of result' for the via minimization problem during layer assignment. It turned out that by softening up a little bit the internal representation of a circuit considerably better optimization results can be obtained. So assume that for a circuit a lot of vias have been generated by heuristics for layer assignment, one now has with the aid of the navigator the possibility of easily checking the positions where vias have been placed. Experiments show that most of the vias are needed on the joints to the instances of boxes. These crucial positions can easily be found by the navigator. By ticking them off, the system can automatically soften up the hierarchy in the right way and layer assignment can be redone.
We illustrate this with an 'extreme' example. Consider the circuit family recursively described in figure 12. (For a precise definition of $M[k]$: Assume that $M[1]$ is a given basic cell with two pins of type **a** on the northern, southern and western boundary and three pins of type **a**, **r** and **a** on the eastern side. n^{κ} denotes net n reflected on the vertical axis, n^{i} represents net n rotated counterclockwise by 90°. Note that the equations $i \cdot i = -1$, $-1 \cdot i = -i$ and $-i \cdot i = 1$ hold. The topology of an lt-net $M[k]$ corresponds to an H-tree, which can be used e.g. as ROM. For this, subcircuit C has to be defined as shown in figure 12 where type **a** represents a wire segment belonging to the address and where a wire of type **r** represents a result. The boundary of

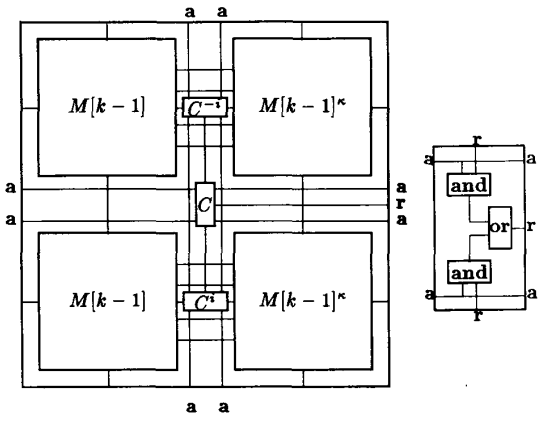

Figure 12 *On the left the recursive definition of the lt-net $M[k]$ and on the right the refinement of subcircuit C*

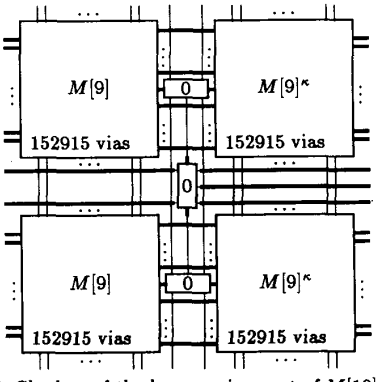

Figure 13 *Checkup of the layer assignment of $M[10]$ The boxes are fitted with the number of the generated vias.*

$M[k]$ ($k \geq 1$) is given by $W(M[k]) = w[k]$, $N(M[k]) = S(M[k]) = t[k]$ and $E(M[k]) = p[k] \cdot \mathbf{a} \cdot \mathbf{r} \cdot \mathbf{a} \cdot p[k]$.) Now let R_k be the internal representation of $M[k]$ and assume that the set $B(R_k)$ of the boxes is given by $\{C, M[i] \mid 1 < i < k\}$. Then a possible heuristic for layer assignment (in the case of two available layers) is to first embed box C in two layers, then box $M[2]$, $M[3]$ and so on (bottom-up pass). Obviously an optimal layer assignment with respect to via minimization needs no via for box C (if the layers of the terminals of the basic cells are not preassigned). Thus by induction it follows that the resulting layer assignment to $M[k]$ needs $7 \sum_{i=0}^{k-2} 4^i$ vias. With the aid of the navigator one can easily find out that all the vias occur on the joints to the instances of box C (see figure 13) and it is easy to see that the removal of C out of $B(R_k)$ decreases the number of vias considerably. In fact, in that case no via will be generated during layer assignment. (Notice also that the internal representation remains very compact after this manipulation of $B(R_k)$.)

A further application: Checkups of fault coverages

Another interesting application of the graphical navigator is in test preparation. Consider for example test generation programs. These programs usually try to compute a test for every stuck-at fault in a given circuit. Since the problem of generating a test for a given stuck-at fault is NP-complete [7] the time (number of backtracks) spent for one fault can be bounded by the user of the program in order to get a result in a reasonable amount of time. Thus besides a test set for the testable faults, test generation programs output as information • the list of *redundant faults*, i.e. the faults for which the program proved that there exists no test for them and • the list of *aborted faults*, i.e. the faults for which no test could be generated in the specified time limit and which could not be proved redundant.

This information is normally output in textual form. Thus it is a very tiresome task for the designer to find the critical parts (with a large number of aborted faults) of his design. In CADIC this task is facilitated by the graphical navigator which visualizes the output of the test generation program (at the moment we use Siemens SOCRATES [20,21]). After the test information has been loaded, the graphical navigator outputs the following additional information: • Each box (hierarchical view of a subcircuit) is marked with the number of aborted and redundant faults within it. (Figure 14 shows a subcircuit and its fault statistics zoomed out of the (24,8)-bit floating point multiplier.) • The inputs and outputs of elementary gates are marked with the aborted and redundant faults on them. For example the fully expanded subcircuit of *adder*[48] shown in figure 15 contains an aborted stuck-at 1 fault (marked with *a*1 at the corresponding line).

Thus by descending through the design guided by the statistics inside the boxes it is very easy to find the critical parts of the design which can be exactly analyzed by using the Zooming and Full Expansion operation. The designer can now improve the test quality, by • making hardware modifications in order to improve the controlability and observability of parts with insufficient test coverage or • using designer insight to construct tests for aborted faults or classify them as redundant.

As an example let us consider again the (24,8)-bit expansion of the floating point multiplier. For the original circuit SOCRATES achieved only a fault coverage of 93.09% (with a backtrack limit of 10). (The fault coverage is defined as the percentage of classified faults, i.e. faults which have been proved redundant or for which tests have been generated.) Rising the backtrack limit improves the fault coverage gradually, but also increases the running time of the algorithm. Thus sufficient fault coverage could not be reached in reasonable time. The results of test generation were analyzed

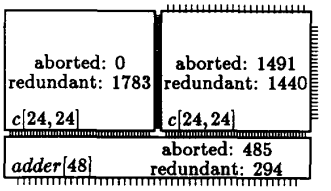

Figure 14 *Fault statistics of the 48-bit integer multiplier inside the (24,8)-bit floating point multiplier*

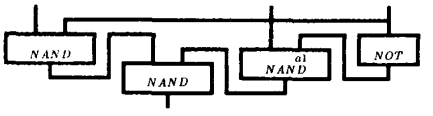

Figure 15 *Aborted stuck-at 1 fault in a subcircuit of adder*[48]

by using the graphical navigator. It turned out that most of the aborted faults (1976) were located inside the 24-bit integer multiplier (the remaining logic contained only 59 aborted faults). Figure 14 gives the fault statistics of the integer multiplier itself. It shows that all of the aborted faults inside the partial multiplier are located in its right half. Propagation of faults from this part is impeded by the rounding operation which rounds the 48-bit mantissa product to 24-bit precision. Thus small hardware modifications were made to improve the observability of this part resulting in a fault coverage of 99,12% by SOCRATES (again the number of backtracks was bounded by 10). (Note that our version of SOCRATES does not incorporate the features presented in [20] which will further improve the fault coverages given here.)

5. Conclusion

Two frontend components of the VLSI design system CADIC are presented in this paper. The main features of a graphical system for hierarchical specifications and of a graphical navigator as well are introduced and the effectiveness of both components is demonstrated with the help of examples.

Graphical circuit specification here is not only understood as the specification of a specific (possibly non regular) circuit, but also allows the definition of parameterized circuits and thus offers the possibility of building one's own generators. (In section 3 this is illustrated by the definition of a multiplier generator.) The generators' definitions are based on simultaneous cell- and edge-replacement operations, whose correctness and non-ambiguity is guaranteed automatically by the system. Problems which had to be solved in this context are indicated. All in all, the graphical specification here profits from the algebraic calculus being the heart of CADIC.

In the same way as graphical circuit specification offers a very comfortable tool to shorten and simplify specification time, the second component, the graphical navigator, helps during the remaining phases of the design. The navigator allows the designer to navigate across the synthesized layout using several operations as *Partial Expansion, Zooming, Full Expansion,* .... This is especially important to control the outcome of algorithms or heuristics working hierarchically. Slight changes in hierarchy may imply dramatic changes in the quality of an optimization. Layer assignment is used as an example to show how the navigator helps in this case. A second example demonstrating the importance of the navigator is the visualization of the results obtained during test pattern generation. Here the navigator is used for analyzing the outcome of a non-hierarchical algorithm and replaces 'exhaustive search' in a net list. Both examples help to understand the importance of the graphical support provided by the navigator. A lot of further applications can be imagined for the future. Since at any time the references between the parts of the layout drawn on the screen and the internal representation of the complete circuit are known, parts of the design can be 'selected' by ticking off the right position in the layout. We are planning to use this facility of the navigator for interactive guidance of algorithms. As indicated above, this may lead to significant improvements in hierarchical layer assignment. Another example is interactive control of test generation algorithms. The test generation algorithm can be restarted with a larger number of backtracks on a set of aborted faults just by ticking them off. In the same way the designer can classify aborted faults as redundant. Also in this context the graphical interface is much more convenient than the textual interfaces used up to now.

In summary, we think that a graphical specification system (including parameterized circuit specification) and a graphical navigator provide facilities which essentially ease the design of integrated circuits and in this sense are indispensable tools for today's design systems.

REFERENCES

[1] B. Becker. An easily testable optimal time VLSI-multiplier. *ACTA INFORMATICA*, 24:363–380, 1987.

[2] B. Becker, G. Hotz, R. Kolla, P. Molitor, and H.G. Osthof. Hierarchical design based on a calculus of nets. In *Proceedings of DAC87*, pages 649–653, 1987.

[3] L. Budach, E. Giessmann, H. Grassmann, B. Graw, and Ch. Meinel. Relacs – a recursive layout computing system. In *Proceedings of the 'Parallel Algorithms and Architectures' Workshop*, pages 86–88, Suhl, GDR, 1987.

[4] Th. Burch. *Ein grafisches Eingabesystem für CADIC.* Master's thesis, Universität des Saarlandes, 1988.

[5] E. Clarke and Y. Feng. *ESCHER - A Geometrical Layout System For Recursively Defined Circuits.* In *Proceedings of DAC86*, pages 650–653, 1986.

[6] Th. Fettig, U. Sparmann, G. Wannemacher, and W. Weber. Entwurf von kombinatorischen Schaltkreisen für schnelle Gleitkommaarithmetik. Documentation of CADIC Designs 1989.

[7] H. Fujiwara and S. Toida. The complexity of fault detection problems for combinational logic circuits. *IEEE Transactions on Computers*, C-31, 1982.

[8] B. Grünewald, B. Grande, R. Kolla, and J. Schnabel. Entwurf eines Stringspeichers für COMSKEE. Documentation of CADIC Designs 1989.

[9] R. Hahn, R. Krieger, U. Sparmann, and B. Becker. *Strukturbasierte Selbsttests für arithmetische Schaltkreise.* Documentation of CADIC Designs 1988.

[10] E. Hörbst, M. Nett, and H. Schwärtzel. *VENUS: Entwurf von VLSI-Schaltungen.* Springer Verlag, 1986.

[11] G. Hotz, R. Kolla, and P. Molitor. On network algebras and recursive functions. In *Proceedings of the 3rd International Workshop on Graph Grammars and Their Applications to Computer Science*, pages 250–261, Springer Verlag, LNCS 291, 1986.

[12] R. Kolla. A dynamic programming approach to the power supply net sizing problem. In *Proceedings of EDAC90*, 1990.

[13] R. Kolla. *Spezifikation und Expansion logisch topologischer Netze.* PhD thesis, Universität des Saarlandes, 1986.

[14] R. Kolla and P. Molitor. A note on hierarchical layer assignment. *INTEGRATION, the VLSI journal*, 7:213–230, 1989.

[15] R. Kolla, P. Molitor, and H.G. Osthof. *Einführung in den VLSI-Entwurf.* B.G. Teubner Verlag, 1989.

[16] W.K. Luk and J. Vuillemin. Recursive implementation of optimal time VLSI integer multipliers. In *Proceedings IFIP Congress 83*, pages 155–168, 1983.

[17] P. Molitor. Constrained via minimization for systolic arrays. *IEEE Transactions on CAD/ICAS.* (Will appear in 1990.)

[18] P. Molitor. Free net algebras in VLSI-theory. *Fundamenta Informaticae*, XI:117–142, 1988.

[19] P. Molitor. On the contact minimization problem. In *Proceedings of STACS87*, pages 420–431, 1987.

[20] M.H. Schulz and E. Auth. Advanced automatic test pattern generation and redundancy identification techniques. In *18th Symposium on Fault-Tolerant Computing*, pages 30–35, 1988.

[21] M.H. Schulz, E. Trischler, and T.M. Sarfert. Socrates: a highly efficient automatic test pattern generation system. In *Proceedings of 1987 International Test Conference*, pages 1016–1026, 1987.

[22] C.S. Wallace. A suggestion for a fast multiplier. *IEEE*, 13:14–17, 1964.