On Network Algebras and Recursive Equations

Günter Hotz, Reiner Kolla, Paul Molitor
Fachbereich 10, Universität des Saarlandes
D-6600 Saarbrücken, FRG

ABSTRACT. By means of an example, we will present a description language for regular VLSI layouts. This language is a network calculus able to deal with recursive equations. These recursive equations can be understood as graph grammars. The solution of a recursive system of equations can be obtained by the iteration of a homomorphism of the net algebra. In a certain sense, the class of the layouts defined by a system of equations can also be understood as Lindenmayer-Rozenberg-system.

Key words: computer-aided design, grammar types, hardware description languages, recursion schemes, very large scale integration.

CONTENTS

1. Introduction

We are going to present a calculus whose objects are planar computation graphs layed out into a rectangle R. The external connectors (input/output pins) of these nets are placed on the boundary of R. We call such a formation logic topological net. Two logic topological nets are equal, if they "mainly have the same planar topological structure", i.e., if there is a sequence of elementary deformations in the plane that transforms one net into the other without producing nonplanar intermediate embeddings.

There are two partial binary operations "$\oslash$" and "$\ominus$" defined on logic topological nets. The operation "$\oslash$" constructs a new net by placing the nets above each other, the operation "$\ominus$" by placing the nets side by side, if and only if the number and the specification (input/output) of the pins on the relevant sides match. Moreover, there are unary operations, namely 90°-rotation and reflection of logic topological nets.

By factorizing the algebra according to suitable relations , we get away from planar graphs and we consider abstract graphs and graph transformations, which do not change the functional properties of the computation graph, suitable for optimizing nets.

In order to test the practicability of this approach we have designed the system CADIC for the hierarchical design of integrated circuits. The system's kernel is based on the above calculus. CADIC has been designed at the Department of Computer Science of the Universität des Saarlandes. The system runs on SIEMENS/BS2000 and VAX/UNIX. In its present stage of implementation it is possible to generate symbolic layouts of circuits with several millions of elementary units. The circuits are described by compact specifications, namely recursive net equations defined on the basis of our algebra.

In this paper we shall give an impression of the theory by presenting some examples. These examples also show that the theory can be integrated into the theory of graph grammars.

2. Planar and spacial nets

The example of Artin's plaits illustrates that by factorizing the algebra of planar logic topological nets we obtain spacial nets:

Let us consider a planar net F (see figure 2.1), whose edges are not labeled. The knots are labeled by $\sigma =$  or by $\sigma' =$ 

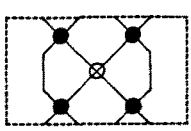


Figure 2.1

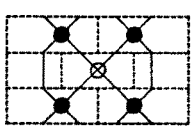


Figure 2.2

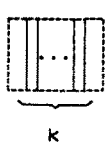


Figure 2.3

Let the symbol "$|_k$" ($k \in \mathbf{N_0}$) denote the net shown in figure 2.3, then with the aid of the operations "$\ominus$" and "$\oslash$" we can represent the net F by the expression

$$(a) \quad F = (\sigma \ominus \sigma) \oslash (|_1 \ominus \sigma' \ominus |_1) \oslash (\sigma \ominus \sigma)$$

The broken lines in figure 2.2 show how to cut up the net in order to obtain the above expression.

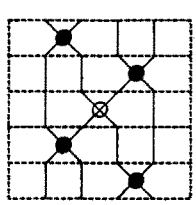


Figure 2.4

Figure 2.4 also shows the net F, but with another geometrical appearance which leads to the following expression:

$$(b) \quad F = (\sigma \ominus |_2) \oslash (|_2 \ominus \sigma) \oslash (|_1 \ominus \sigma' \ominus |_1) \oslash (\sigma \ominus |_2) \oslash (|_2 \ominus \sigma)$$

The two expressions (a) and (b) represent different geometrical embeddings of the same net. The embeddings can be transformed into each other by planar deformations.

Generally the following equation holds:

$$(|_{k_1} \ominus \tau \ominus |_{n_1}) \oslash (|_{k_2} \ominus \rho \ominus |_{n_2}) = (|_{k_2} \ominus \rho \ominus |_{n_2}) \oslash (|_{k_1} \ominus \tau \ominus |_{n_1})$$

if $k_1 + 2 \leq k_2$ or $k_2 + 2 \leq k_1$ and $\tau, \rho \in \{\sigma, \sigma'\}$.

Now we give a special interpretation to σ and σ' such that the planar graph becomes a spacial one (figure 2.5).

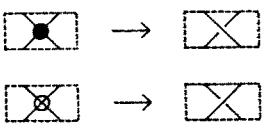


Figure 2.5

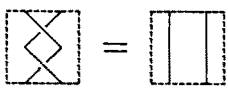


Figure 2.6

This interpretation is characterised by the relations

$$\sigma \oslash \sigma' = |_2$$

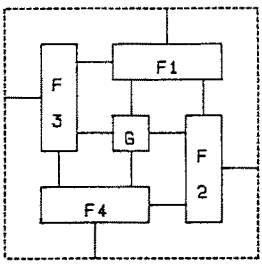
and

$$(\sigma \ominus |_1) \oslash (|_1 \ominus \sigma) \oslash (\sigma \ominus |_1) = (|_1 \ominus \sigma) \oslash (\sigma \ominus |_1) \oslash (|_1 \ominus \sigma).$$

If the number n of inputs is fixed, we, obviously, obtain Artin's plait group with n threads.

By this we have demonstrated that we can easily detach ourselves from planar nets, which offer the advantage of a simple calculus, by factorizing the algebra of planar nets according to a suitable system of relations, e.g. by adding the relation $\sigma \oslash \sigma = |_2$ (see figure 2.6) we obtain a representation of abstract graphs.

Let us return to the logic topological nets. The first question we have to answer is whether each net can be described by the operations "$\oslash$", "$\ominus$" and by nets which contain at most one knot. We call such nets elementary nets.



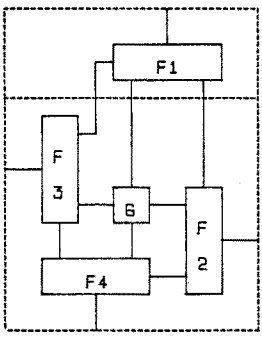


Figure 2.7 Figure 2.8

The example given in figure 2.7 illustrates that there are problems if the knots of the graph are not points in the usual sense, but objects with geometrical dimensions as is the case in VLSI-design. In figure 2.7 the rectangles are knots representing physical elements which cannot be cut up because they represent "atomic functions", i.e., it is not possible to assign functions to the parts of a decomposition of such a knot so that the function of the knot can easily be computed with the aid of the functions of the parts.

The graph shown in figure 2.7 can, however, be cut up, if we first deform it as described in figure 2.8. In our calculus this deformation corresponds to the introduction of the relation

It is clear that there is no algebraic expression in our calculus which represents the formation of figure 2.7. By applying planar deformations to this embedding we, however, obtain configurations which can be represented by algebraic expressions. In general, each logic topological net can be represented by an algebraic expression. Moreover, expressions describing the same logic topological net can be transformed into each other by applying our rules ([MO86]).

3. Semantics-preserving transformations of nets

The most natural way to define the behaviour of nets is to compute it with the aid of an algebraic expression and the functions of the elementary nets, e.g., by using homomorphisms. This can be done if the flow of information is "monotone", e.g., from top to bottom. Generally that is not possible this way. An approach of treating the general case is given in [BU86], [KO87b], [MO86].

If we assume that each wire is unidirectional and that the information flows from top to bottom the nets of figure 3.1 and 3.2 obviously compute the same functions (independent of the function performed by the elementary cell A).

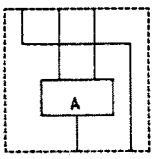
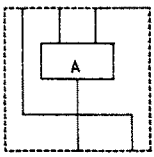


Figure 3.1



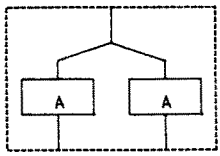
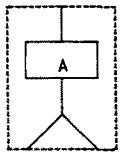


Figure 3.2



For such nets there is a complete system of correct relations if there are fixed cells for reversing a list of signals, for branching signals and, for cutting off unnecessary wires. In this connection a relation is called correct, if the application of this relation to a net n does not change the behaviour of n. A system S of relations is called complete if any two logic topological nets which perform the same function for any interpretation of the remaining elementary cells can be transformed into each other by applying S ([HO65]).

If we add additional "fixed" cells, the set of the necessary relations is, in general, not enumerable ([MO86], [MO87]).

If we confine ourselves to boolean functions, then there is a complete set of correct relations which transform any two realizations of the same function into each other ([HO65], [CL70]).

Thus, we have a calculus, in which many design problems can easily be specified and solved, consisting of different algebras interconnected by homomorphisms.

There are a lot of problems arising in the design of VLSI circuits. First, we have to specify the problem exactly. This specification is mostly independent of the planar realization. It has not been possible until now to solve design problems by starting only with the specification because the arrangement of communication lines and basic cells plays a prominent part in the design process. In our calculus, however, the designer has the possibility to include geometrical properties in the circuit description. During the design process the solution of the design problem is embedded in lower and lower levels of the calculus.

It is very difficult to take advantage of the relations to find good layouts. We have done it in order to solve the layer assignment problem by using the simulated annealing method. The results are rather good ([MO85]).

In the following chapter we shall illustrate these ideas more precisely.

4. A 2^n-bit adder

We extend our algebras by inserting an infinite set $\{A_n;\ n \in \mathbf{N_0}\}$ of variables. First, we only know the specification of the pins of these nets. For this we use an infinite set of signal-types which we denote by nonnegative integers, e.g. $\underline{7}$. A wire of type $\underline{n}$ represents 2^n parallel binary wires.

A_n denotes the 2^n-bit adder. On the northern side of A_n there are two inputs of type $\underline{n}$ and in the south two outputs of type $\underline{n}$ and one output of type $\underline{1}$.

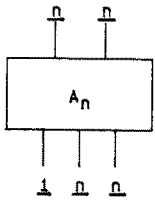


Figure 4.1

We use the following notation:

$$N(A_n) = \underline{n} \cdot \underline{n},\ S(A_n) = \underline{1} \cdot \underline{n} \cdot \underline{n},\ W(A_n) = \varepsilon,\ E(A_n) = \varepsilon.$$

The symbol "·" denotes the concatenation in the free monoid $\mathbf{N}_0^*$.

The adder we describe is of depth $log(n)$ and is wellknown as "conditional sum adder" ([SK60]). It simultanously computes the sum x+y and the sum x+y+1 if the operands are x and y.

Now, suppose that we have already designed a $\frac{n}{2}$-bit adder which computes the sum and the sum + 1 of its inputs simultanously. Then, for n-bit numbers x and y, x+y and x+y+1 can be computed by two $\frac{n}{2}$-bit adders and by a multiplexer stage S_n, which selects the correct result (see figure 4.2). By combining this idea with the divide and conquer technique we obtain an adder of depth n for 2^n-bit operands.

The wire of type $\underline{1}$ in the south of A_n contains both carry bits, that of x+y and that of x+y+1. The western output of type $\underline{n}$ denotes the sum x+y+1 modulo 2^{2^n} and the eastern one denotes the sum x+y modulo 2^{2^n}.

Now, we will illustrate how to construct A_{n+1} by using A_n. In order to do this we refine wires of type $\underline{n+1}$ to two wires of type $\underline{n}$. The left wire represents the most significant 2^n bits and the right one the least significant bits:

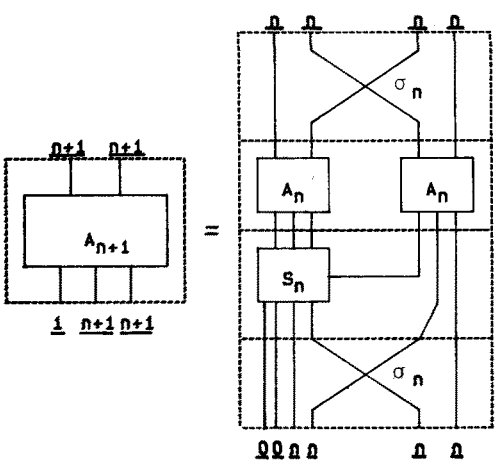


Figure 4.2

σ_n is defined in the following way.

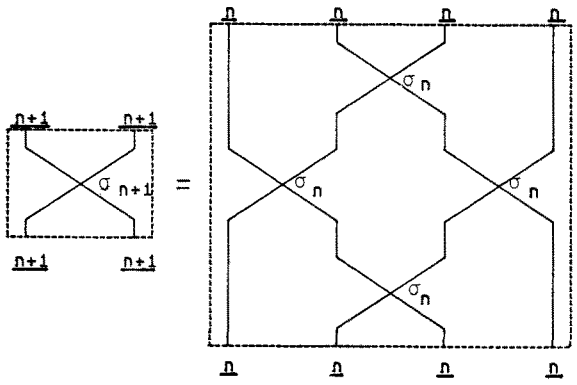


Figure 4.3

To shorten matters, we have omitted the exact description of the multiplexer stage S_n.

In our calculus we can specify the structure of A_{n+1} and of σ_{n+1} by

$$A_{n+1} = (|_1 \ominus \sigma_n \ominus |_1) \oslash (A_n \ominus A_n) \oslash (S_n \ominus \lrcorner \ominus |_2) \oslash (|_3 \ominus \sigma_n \ominus |_1)$$
$$\sigma_{n+1} = (|_1 \ominus \sigma_n \ominus |_1) \oslash (\sigma_n \ominus \sigma_n) \oslash (|_1 \ominus \sigma_n \ominus |_1).$$

The refinement of a 1-bit adder is given by

$$A_0 = (\curlywedge \ominus \curlywedge) \oslash (|_1 \ominus \sigma_0 \ominus |_1) \oslash (\curlywedge \ominus \curlywedge \ominus EXOR)$$
$$\oslash (|_1 \ominus \sigma_0 \ominus |_2) \oslash (OR \ominus AND \ominus \curlywedge) \oslash (|_2 \ominus NOT \ominus |_1).$$

σ_0 denotes a crossing of two wires of type $\underline{0}$. The type of the communication wires of the elementary nets $\curlywedge$, $\lrcorner$, $|_1$ is implicitely given by the context.

In this connection A_n and σ_n are defined for all $n \in \mathbf{N_0}$ by a recursion. This recursion can be solved by stepwise iteration of the functor $\varphi = (\varphi_1, \varphi_2)$ which is induced by the recursive equations ([KO87a]). $\varphi_1 : \mathbf{N_0^\star} \to \mathbf{N_0^\star}$ is the monoid homomorphism defined by $\varphi_1(\underline{n+1}) = \underline{n} \cdot \underline{n}$ and $\varphi_1(\underline{0}) = \underline{0}$. $\varphi_2(A_{n+1})$ and $\varphi_2(\sigma_{n+1})$ are defined as the nets given in the recursion. Furthermore, we define

$$\varphi_2(AND) = AND,\ \varphi_2(OR) = OR,$$
$$\varphi_2(EXOR) = EXOR,\ \varphi_2(NOT) = NOT,$$
$$\varphi_2(\curlywedge) = \curlywedge,\ \varphi_2(\sigma_0) = \sigma_0.$$

With that, the functor φ is an endomorphism on our netalgebra. If we apply φ until the result does not change anymore, we obtain the circuit which realizes the adder.

If we define

$$\varphi^{(n+1)} = \varphi^{(n)} \circ \varphi,\ \varphi^{(0)} = id$$

and

$$\varphi^* = \lim_{n \to +\infty} \varphi^{(n)},$$

then $\varphi^*(A_n)$ is the solution of the recursion. We call $\varphi^*(A_n)$ the expansion of A_n.

It is clear that $\varphi^*(A_n)$ is not a geometrical layout, but a whole class of layouts of which we have to pick out the one which is the "best" with respect to the design criterions. Because these optimizations are not dealt with during this workshop, we only remark upon the interesting aspect.

In general, the size of a circuit grows exponentially with the number of refinements, e.g. a 32-bit Wallace-tree multiplier (i.e. n=5) contains about 300000 elementary units. This, obviously, leads to problems concerning the computing time of optimization algorithms.

However, by treating each variable only once (independent of the fact that the variable occurs several times in the whole net) this problem can be solved ([CADIC86b], [CADIC87]). By using this hierarchical approach paging is no longer necessary. This greatly reduces the effective computing time because a nonhierarchical algorithm in particular has to page in and out of memory portions of the circuit if it is too large to be completely kept in memory. Clearly, there are difficulties at the "seams" between the black boxes, i.e., there is a trade-off between "computing time" and "quality of result" ([KM86]).

The calculus has been developed to handle both logical and geometrical information in an easy and comfortable way. It is clear that the geometrical properties included in the circuit description by the designer are present in the layout. It has, however, turned out that unfavorable designs can, mostly, be improved by making minor modifications to the recursive equations. Moreover, there are probabilistic algorithms to improve such designs ([KO87a]).

5. Some remarks on CADIC

CADIC is the name of our design system, the kernel of which is the calculus presented in this paper. CADIC runs on SIEMENS/BS2000 and will run on VAX/UNIX in the near future. A version of the system for a personal computer is in development.

The system is mainly written in COMSKEE, which is a programming language for linguistic applications. COMSKEE integrates a data bank for document retrieval ([ME84]).

Finally we would like to demonstrate the power of CADIC. The system of the net-equations given in figure 5.1 is a compact specification of a family of fast multipliers based on a modified Wallace tree. "+" denotes a crossover of a horizontal wire and a vertical wire. The member $MULTREE[k, k+1]$ represents the 2^k-bit multiplier (for more details see [BE85]). A layout of the 8-bit (resp. 16-bit) multiplier which was generated by our system is shown in figure 5.2 (resp. 5.3). For the 16-bit multiplier the running time was speeded up by a factor of about 50 by using hierarchical algorithms.

It is clear, that the above net equations may be specified by using graphical methods similar to those proposed in [CF85].

$$
\begin{aligned}
+_k &= \oslash_{j=1}^{k} + \\
MU2 &= (+ \ominus + \ominus \top \ominus +) \oslash (+ \ominus \dashv \ominus \mid \ominus \mid) \oslash \\
&\quad (\mid \ominus AND \ominus \vdash \ominus +) \oslash (\vdash \ominus \lrcorner \ominus \mid \ominus \mid) \oslash \\
&\quad (+ \ominus + \ominus \top \ominus +) \oslash (+ \ominus \dashv \ominus \mid \ominus \mid) \oslash \\
&\quad (\mid \ominus AND \ominus \mid) \oslash (\mid \ominus \llcorner \ominus \dashv) \oslash (\mid \ominus \vdash \ominus +) \\
CSA2 &= (\mid \ominus \mid \ominus \llcorner \ominus + \ominus \neg \ominus \mid) \oslash (+ \ominus FA \ominus + \ominus \neg \ominus \mid) \oslash \\
&\quad (\mid \ominus \mid \ominus \vdash \ominus + \ominus + \ominus +) \oslash (+ \ominus FA \ominus \neg \ominus \mid \ominus \mid \ominus \vdash) \oslash \\
&\quad (\vdash \ominus \lrcorner \ominus \mid \ominus \mid \ominus \mid \ominus \mid) \\
CSA &= (\mid \ominus \vdash \ominus + \ominus \lrcorner) \oslash (FA \ominus + \ominus \neg) \\
MULTREE[i,j] &= MULTREE[i,j-1] \ominus MULTREE[i,j-1] \\
MULTREE[i,0] &= BMULTREE[i] \\
BMULTREE[i] &= (\mid \ominus \uparrow \ominus \mid \ominus \uparrow \ominus \mid) \oslash \\
&\quad (+_{3 \cdot 2^{i-1}-2} \ominus BMULTREE[i-1] \ominus +_{3 \cdot 2^{i-1}-2}) \oslash CSA2 \oslash \\
&\quad (+_{3 \cdot 2^{i-1}-2} \ominus BMULTREE[i-1] \ominus +_{3 \cdot 2^{i-1}-2}) \oslash \\
&\quad (\mid \ominus \downarrow \ominus \mid \ominus \downarrow \ominus \mid) \oslash \\
BMULTREE[1] &= MU2
\end{aligned}
$$

Figure 5.1: Specification of a family of fast multipliers

6. Acknowledgement

We would like to thank Dr. Bernd Becker, Ursula Becker, Ursula Fissgus and Hans-Georg Osthof who have taken an active part in the design of optimization algorithms and of the system itself. Dr. Jan Messerschmidt, Manfred Ries and Jörg Schütz have supported us in the area of COMSKEE. Moreover, we express our thanks to the students who have done a good job during practical trainings.

7. References

[BE85] B.Becker:
"An easily testable optimal-time VLSI-multiplier"
Proceedings of EUROMICRO 85, pp.401-411, Brussels, North-Holland.

[BU86] H.Bühler:
"Korrektheitsbeweise von rekursiv beschriebenen logisch-topologischen Netzen"
Diplomarbeit, Saarbrücken 1986.

[CADIC86a] G.Hotz, B.Becker, R.Kolla, P.Molitor:
"Ein logischer-topologischer Kalkül zur Konstruktion von integrierten Schaltkreisen"
INFORMATIK: Forschung und Entwicklung, 1986, Heft 1 und 2, Springer Verlag 1986.

[CADIC86b] B.Becker, G.Hotz, R.Kolla, P.Molitor, H.G.Osthof:
"CADIC: A System for Hierarchical Design of Integrated Circuits"
TR07/1986, SFB124, Saarbrücken 1986.

[CADIC87] B.Becker, G.Hotz, R.Kolla, P.Molitor, H.G.Osthof:
"Hierarchical Design based on a Calculus of Nets"
will appear in the Proceedings of the 24^{th} Design Automation Conference, Miami 1987.

[CF85] E.Clarke, Y.Feng:
"ESCHER - A geometrical Layout System for Recursively Defined Circuits"
CMU-CS-85-150, Carnegie-Mellon University, Pitsburgh 1985.

[CL70] V.Claus:
"Ebene Realisierungen von Schaltkreisen"
Dissertation, Saarbrücken 1970.

[HO65] G.Hotz:
"Eine Algebraisierung des Syntheseproblems für Schaltkreise"
EIK 1, 1965, pp.185-205, 209-231.

[KM86] R.Kolla, P.Molitor:
"A note on hierarchical layer-assignment"
TR08/1986, SFB124, Saarbrücken 1986.

[KO87a] R.Kolla:
"Über Fragen im Zusammenhang mit der Spezifikation und Expansion großer logisch topologischer Netze"
Dissertation, Saarbrücken 1987.

[KO87b] R.Kolla:
"Proving correctness of logic topological nets"
will appear as technical report, SFB124, Saarbrücken 1987.

[ME84] J.Messerschmidt:
"Linguistische Datenverarbeitung mit COMSKEE"
Stuttgart: B.G. Teubner 1984

[MO85] P.Molitor:
"Layer Assignment by Simulated Annealing"
Microprocessing and Microprogramming 16 (1985), pp.345-350, North-Holland.

[MO86] P.Molitor:
"Über die Bikategorie der logisch topologischen Netze und ihre Semantik"
Dissertation, Saarbrücken 1986.

[MO87] P.Molitor:
"Free Net Algebras in VLSI-Theory"
in preparation, Saarbrücken 1987.

[SK60] J.Sklansky:
"Conditional-sum addition logic"
IRE-EC 9, pp.226-231, 1960.

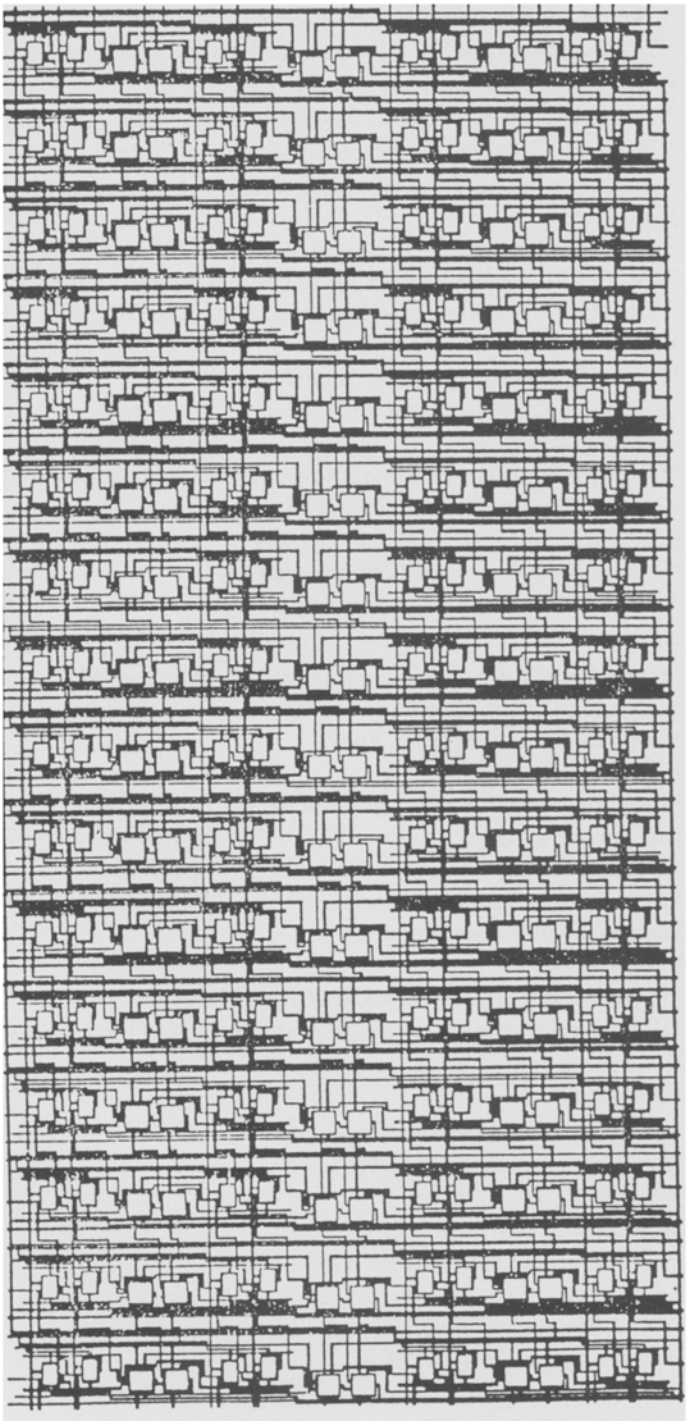


Figure 5.2: 8-bit multiplier

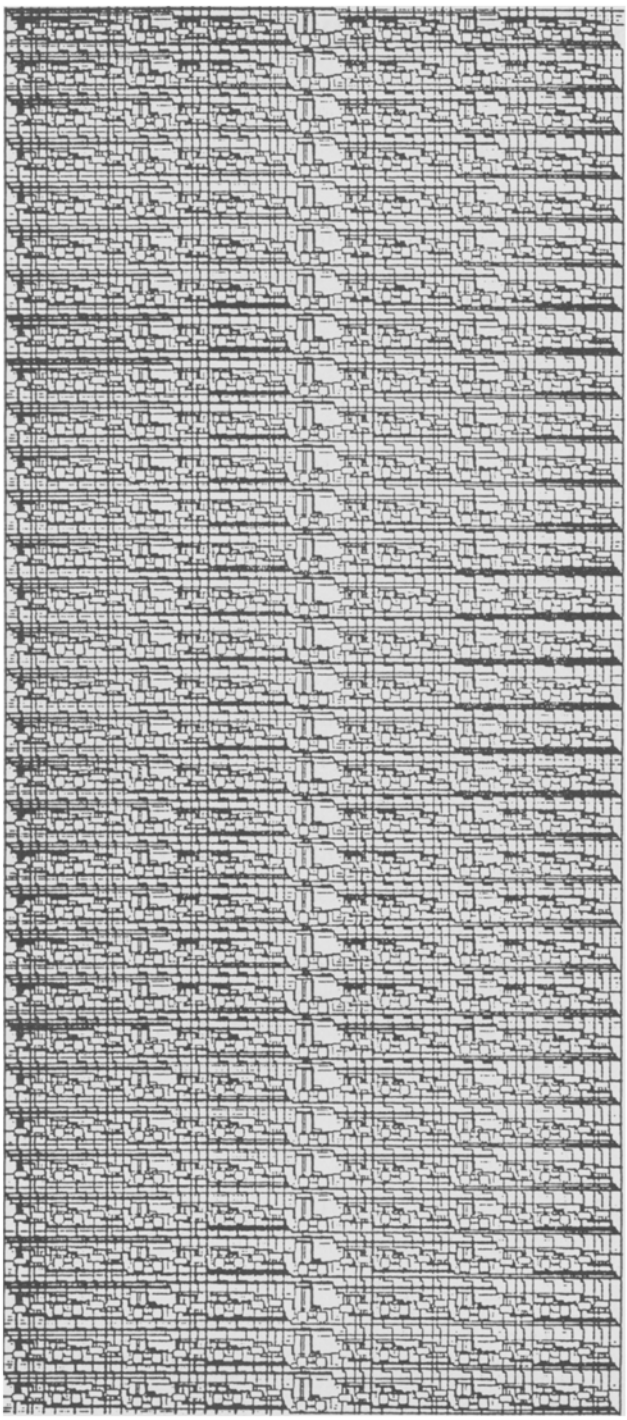


Figure 5.3: 16-bit multiplier