To appear in the *Proceedings of the VLSI 81 International Conference*, Edinburgh, Scotland, 18-21 August 1981.

AN ALGEBRAIC APPROACH TO VLSI DESIGN

Luca Cardelli and Gordon Plotkin

Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, UK

1. INTRODUCTION

We propose an algebraic view of the hierarchical approach to VLSI design developed by Mead, Conway and others (Mead and Conway, 1980). VLSI networks are described by expressions of a many-sorted *n*MOS algebra, and the algebraic operators are designed to support a structural methodology. The algebra can be embedded in a programming language as an abstract data type (Goguen *et al.*, 1978), and since the emphasis is on expressions (they denote networks) it is natural to use an applicative language (Burge, 1975).

These ideas are really very general and should be useful wherever a structural approach is needed for graphical or geometrical information; in particular there is no difficulty with other technologies such as *c*MOS. The ideas largely originate with Milner (1979) and the small differences in the choice of operators are motivated by programming convenience and the fact that 1-1 connection is more natural for VLSI than many-many connection.

Our *n*MOS expressions will have various interpretations whether as graphs, geometric configurations, behaviours or other networks; in section 2 we give an informal one as pictures composed of coloured lines (*sticks*) and their intersections (*stones*). Each network will interface with its environment via a set of named coloured *ports* which determine its *sort*. The unary operators *renaming* and *restriction* manipulate the interface, and the more interesting binary operator *composition* joins smaller networks together into larger ones.

By adopting various abbreviations an extended notation is developed in section 3 which merges naturally into an applicative language. The control structures in the language

provide conditional assemblies of networks and parameterizations, and a specialized iteration construct allows the convenient expression of the most common VLSI subsystems. We thus obtain what might be best called a high level chip assembly language.

The flexibility of parameterization of textual expressions has no graphical parallel and so we prefer the textual expression of networks. However, graphical tools should be used as much as possible especially for interactive visual feedback, and we believe that an algebraic approach might still be useful for hierarchical input and editing.

In conclusion, our algebraic notation can help the integration of the graphical and textual aspects of network design, because it can reflect the intended structure of networks and keep the connectivity information local and well-organized.

2. *n*MOS NETWORK ALGEBRAS

Our algebras are many-sorted, the idea being that the sort of a network determines its interface with its environment. We view networks as interfacing through a set of named ports (one name to each port), each on some layer. Formally, let {green,red,blue} be the set of *types* and let PNames be an infinite set of *(port) names* (and we use a,b to range over names and A,B,C to range over finite sets of names). Then a *sort* is a map s: A -> Types (and we put $|s| = A$); s(a) shows the layer of the port (named) a.

The elementary network components form the set, Γ, of *constants* of our algebra (and are ranged over by c); every constant, c, has sort σ(c). Here is one reasonable choice for Γ giving the sort in an evident notation; Fig.1 pictures some of these elementary components.

Contacts
- Green GCon: {gn,gs,ge,gw:green}
- Red RCon: {rn,rs,re,rw:red}
- Blue BCon: {bn,bs,be,bw:blue}
- Green-Red GRCon: {gn,gs:green; re,rw:red}
- Red-Blue RBCon: {rn,rs:red; be,bw:blue}
- Blue-Green BGCon: {bn,bs:blue; ge,gw:green}

Transistors
- Enhancement ETran: {source,drain:green; gate,gate':red}
- Depletion DTran: {source,drain:green; gate,gate':red}

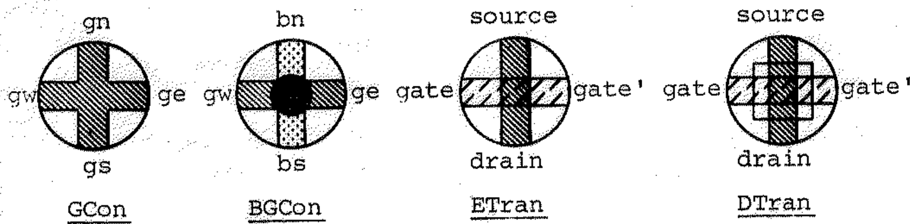

Fig.1 Some elementary components.

There are two unary operators for manipulating port names thereby changing the interface. The *restriction* operator, \a, removes the name a from the sort of a network; we make it postfix and often abbreviate $\setminus a_1 \ldots \setminus a_n$ to $\setminus a_1, \ldots, a_n$. Fig.2 pictures DTran' = DTran\gate'. Formally for every s and a we have a unary operator \a: s -> s' (where |s'| = |s|\{a} and s'(a') = s(a') for a' in |s'|); thus if e is an algebraic expression of sort s then e\a is one of sort s'.

The *renaming* operator {r} (where r: A -> A' is a bijection) renames the ports of a network according to r; we make it postfix and often write $\{a_1 \setminus b_1, \ldots, a_n \setminus b_n\}$ for r when $\{a_i\} \subseteq A$ and $b_i = r(a_i)$ and for a not in $\{a_i\}$ we have r(a) = a. Fig.2 pictures a power supply, ground and a butting-contact denoted, respectively, by the expressions,

VDD = BGCon\gw {ge\high,bs\vdde,bn\vddw} and
GND = BGCon\ge {gw\low,bn\gnde,bs\gndw} and
BuCon= GRCon\gn,rw {gs\green,re\red}.

They have respective types VDD: {high:green; vdde,vddw:blue} and GND: {low:green; gnde,gndw:blue} and BuCon: {green:green; red:red}. Formally for every s and bijection r: |s| -> A' we have a unary operator {r}: s -> s' where $s' = s \circ r^{-1}$.

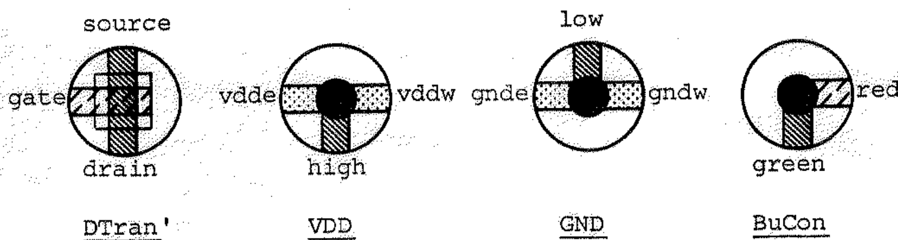

Fig.2 Some building-blocks.

The binary *composition* operator, [r] (where r: A -> A' is a bijection) composes two networks together according to r; the composition is allowed only if corresponding ports have

the same type and there are no two ports with the same name among those not joined via r. We make [r] an infix, associating to the left and often write it as $[a_1\text{--}b_1,\ldots,a_n\text{--}b_n]$ where $\{a_i\} = A$ and $b_i = r(a_i)$. Fig.3 shows how to make a resistor. First compose DTran" = DTran' {source⌄in} with BuCon via [gate--red] obtaining DTran" [gate--red] BuCon : {in,drain,green:green}; then compose the result with GCon' = GCon⌄ge {gs⌄out} via [drain--gn,green--gw] obtaining Res: {in,out:green} where

Res = DTran" [gate--red] BuCon [drain--gn,green--gw] GCon'

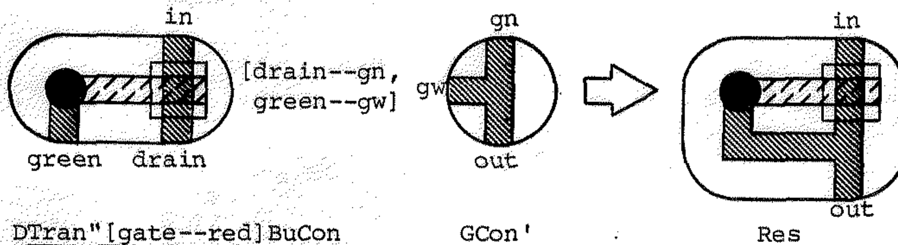

Fig.3 Building a resistor.

Formally for every s,s' and bijection r: A -> A' where $A \subseteq |s|$ and $A' \subseteq |s'|$ and s(a) = s'(r(a)) for every a in A (type restriction) and such that $B \cap B' = \emptyset$ where $B = |s| \setminus A$ and $B' = |s'| \setminus A'$ (names refer to one port) we have a binary operator [r]: s,s' -> s" where $|s''| = B \cup B'$ and s"(b) is s(b) if b is in B and s'(b) otherwise.

As another example, an inverter of sort {in:red; out:green; vdde,vddw,gnde,gndw:blue} can be built as

Inv = (VDD[high--in]Res[out--gn](GCon⌄gw)[gs--source]
ETran[drain--low]GND) {gate⌄in,ge⌄out}

Composition cells in the sense of Rowson (1980) can be regarded as repeated applications of our composition with some restriction, renaming and constants.

The pictures we have used above will now be formalised as appropriate kinds of graphs; they can be regarded as stick diagrams where connectivity is the only topological information retained. Algebraically, what we have done above is to give the *signature* of *n*MOS algebras and we now turn to the most important algebra.

<u>Definition</u> An *n*MOS *network* is a quintuple $N = \langle V,\gamma,A,\pi,E\rangle$ where V is a non-empty finite set and $\gamma: V \to \Gamma$ (and we put $P = \{\langle v,b\rangle \mid b \in \sigma(\gamma(v))\}$) and A is a finite set of names and

π: A -> P is 1-1 and $E \subseteq P \times P$ subject to the following conditions (where $type(v,b) = \sigma(\gamma(v))(b)$):

1. E is symmetric and a partial function.
2. If $\langle v,b\rangle E \langle v',b'\rangle$ then $type(v,b) = type(v',b')$ and $v \neq v'$.
3. No $\langle v,b\rangle$ is both $\pi(a)$ for some a and in the domain of E.

The *sort* of N is s where $|s| = A$ and $s(a) = type(\pi(a))$.

Intuitively, V is the set of vertices (or nodes) and $\gamma(v)$ is the the elementary component at v and A is the set of port names and P is the set of ports and $\pi(a)$ is the port that a alone names and E is the connection relation between ports. The first condition ensures that connection is symmetric and any port is connected to at most one other; the second ensures type-consistency and that there are no self-loops; the third ensures no port is both named and connected. We identify isomorphic networks when the isomorphism is the identity on port names.

Now the *carriers* of the algebra are the networks of sort s, for each s. Any constant, c, of sort s denotes the network $\langle \{c\}, c \mapsto c, |s|, a \mapsto \langle c,a\rangle, \emptyset\rangle$. The operations can be given by simple definitions.

Restriction $\langle V,\gamma,A,\pi,E\rangle \setminus a = \langle V,\gamma,A',a' \mapsto \pi a',E\rangle$ where $A' = A \setminus \{a\}$.

Renaming $\langle V,\gamma,A,\pi,E\rangle\{r\} = \langle V,\gamma,r(A),\pi \circ r^{-1},E\rangle$

Composition $\langle V,\gamma,A,\pi,E\rangle[r]\langle V',\gamma',A',\pi',E'\rangle =$ $\langle V \cup V',\gamma \cup \gamma',B \cup B',\pi'',E \cup E' \cup E''\rangle$ where we assume that $V \cap V' = \emptyset$ and where $r: C \to C'$ and $B = A \setminus C$ and $B' = A \setminus C'$ and $\pi''(b)$ is $\pi(b)$ if b is in B and $\pi'(b)$ otherwise and where $E'' = \{\langle \pi(a),\pi'(r(a))\rangle, \langle \pi'(r(a)),\pi(a)\rangle \mid a \in C\}$.

It can easily be shown that every network is denoted by some algebraic expressions, confirming the power of our notation; also it can be decided in polynomial time whether two expressions denote isomorphic networks, but we lack a good upper bound. We can also axiomatise the equalities between expressions by variants of Milner's laws of flow. In the following x,y and z range over arbitrary expressions such that the equations are between well-sorted expressions of the same sort; we write $\sigma(x)$ for the sort of x and $id_A: A \to A$ is the identity on A.

Composition $(x[id_A]y)[id_{A' \cup A''}]z = x[id_{A'' \cup A}](y[id_{A'}]z)$
 if $A'' \cap A = A \cap A' = A' \cap A'' = \emptyset$

$x[r' \circ r]y = x\{r \cup id_B\}[r']y$
 where $B = |\sigma(x)| \setminus A$ where $r: A \to A'$

$x[r]y = y[r^{-1}]x$

<u>Restriction</u> $x \setminus a = x \quad (\text{if } a \notin |\sigma(x)|)$

$x \setminus a \setminus b = x \setminus b \setminus a$

$(x[r]y) \setminus a = (x \setminus a)[r](y \setminus a) \quad (\text{if } a \in |\sigma(x[r]y)|)$

<u>Renaming</u> $x\{id_A\} = x \quad (\text{where } A = |\sigma(x)|)$

$x\{r\}\{r'\} = x\{r' \circ r\}$

$x\{r\} \setminus b = (x \setminus a)\{r \setminus \{<a,b>\}\} \quad (\text{where } b = r(a))$

$(x[r]y)\{r' \cup r''\} = x\{r' \cup id_A\}[r]y\{id_{A'} \cup r''\}$

$(\text{where } r: A \to A')$

It can be shown that all these laws are true in the above network algebra (consistency) and that any true equation in the algebra can be proved from the laws (completeness). It follows from this and the above remarks on the power of the notation (definability) that our algebra is initial in the class of algebras satisfying the laws. Any semantics for *n*MOS networks should be such an algebra.

3. GEOMETRIC INTERPRETATION

Net Algebras can be embedded in a programming language by adding an abstract data type *picture* together with some special syntax for constants and operators. We illustrate this process in the case of a geometric interpretation of Net Algebras, obtaining a language capable of directly expressing geometric layouts. The host language considered here is Edinburgh ML (Gordon *et al.*, 1979) and the resulting language is called Sticks & Stones. An implementation has been carried out on the ERCC DEC-10 at Edinburgh University (Cardelli, 1981) which makes interactive use of a colour graphics display and also compiles pictures into CIF files.

In the geometric interpretation, a picture is a configuration of coloured geometric *figures* (generally rectangles or polygons) and ports which are now named *vectors* with a size and an orientation. We use *compound* port names like g.east, a or a.b.1. Both figures and vectors have a fixed displacement from a conventional origin which is local to each picture; configurations are identified up to translation and rotation. The geometric interpretation can be made formal, but we concentrate here on other issues.

There is an infinite supply of constants, given by the syntax exemplified in Fig.4. The elementary picture GCon is a green box with lower left corner at 0^0 and upper right corner at 2^2. It has four ports. For example, s is a green port which is a vector starting from the point 0^0 and going in direction 0 degrees counterclockwise from the x-axis for a length of 2. Note that the names n (north), s (south) etc.

are only conventional; pictures have no predefined orientation. The definition of constants requires precise geometric information, and we would prefer graphical input to the system.

```
GCon = form (n: green port [2^2,180,2];
             s: green port [0^0,  0,2];
             e: green port [2^0, 90,2];
             w: green port [0^2,270,2])
       with green box [0^0,2^2].
ETran= form (n: red   port [2^2,180,2];
             s: red   port [0^0,  0,2];
             e: green port [2^0, 90,2];
             w: green port [0^2,270,2])
       with green box [0^0,2^2]
       and  red   box [0^0,2^2].
DTran= form (n: red   port [3^4,180,2];
             s: red   port [1^0,  0,2];
             e: green port [4^1, 90,2];
             w: green port [0^3,270,2])
       with green box [0^1,4^3]
       and  red   box [1^0,3^4]
       and  yellow box [0^0,4^4].
```

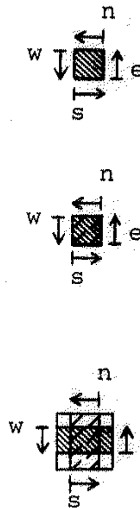

Fig.4 Some elementary geometric components.

Restriction and renaming operate in the way described in section 2, but some abbreviations are introduced. There is a *pattern matching* feature on compound port names. A name like a.!.b.? represents all the names in a sort which match the pattern, where ! matches any single atomic part and ? matches a (possibly empty) list of atomic parts of a compound name. The use of ? is restricted to the end of a pattern to avoid ambiguities; the pattern ? matches all the ports. Hence \!.east means forget all the east ports and {green.?\?} means rename all the green ports by dropping the prefix green. Pattern matching is an abbreviation for the appropriate enumeration of all the ports matching the pattern.

A renaming of the form {a\b move 2} is a case of *geometric renaming*; the port a is renamed b and it is moved by 2 towards the east of the port (the north being the tip of its vector). During this movement the port leaves a trail of its passage, which is a polygon of the same colour as the port. Geometric renaming is an abbreviation for the composition of a suitable "trail" form. It is possible to move, rotate and change the size of ports, and to compose these actions.

A combination of pattern matching and geometric renaming is shown in Fig.5.

```
out = GCon~e {?~? move 2}
pos = ETrans {n~ctrl.n,s~ctrl.s,
              e~data.e,w~data.w} {?~? move 2}
neg = DTrans {n~ctrl.n,s~ctrl.s,
              e~data.e,w~data.w} {?~? move 1}
```

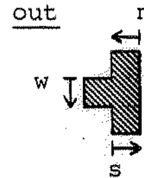

Fig.5 Some restrictions and renamings.

The T-shaped out is obtained from a green square GCon, first forgetting the e port, and then moving all the remaining ports outwards. The pictures pos and neg (not shown) have a cross shape.

Our final example (Fig.6 and 7) is a function taking a parameter n and producing a selector with n control inputs, n complemented control inputs and 2^n inputs. Some standard programming language features are used without explanation.

```
 1.  let sel n =
 2.  for i in 1::exp(2,n)
 3.  iter (for j in n::1
 4.        iter if bit(i-1,j-1) = 0
 5.             then pos[data.e--data.w]neg{ctrl.?~ctrl'.?}
 6.             else neg[data.e--data.w]pos{ctrl.?~ctrl'.?}
 7.        with [data.e--data.w])
 8.       [data.e.!--out.w] out
 9.  with [ctrl.s.!--ctrl.n.!,ctrl'.s.!--ctrl'.n.!,
           out.s--out.n]
10.  where rec bit(i,j) =
11.  if j=0 then i mod 2 else bit(i div 2,j-1)
```

Fig.6 A selector generator.

The circuit shown in Fig.7 is the result of the evaluation of sel 2 (selector with two control inputs). This selector program takes advantage of a specialized iteration construct:
<u>for</u> <variable> <u>in</u> <list> <u>iter</u> <picture> <u>with</u> <connection>
where the <connection> is used between the <picture>'s produced as the iteration <variable> ranges through the <list> of values. Iteration also applies an automatic indexing appending the number n to all the ports produced at the n-th iteration (e.g. a becomes a.3), thus avoiding name clashes.

The selector is obtained by two nested iterations, first building the rows and then joining them up into an array. At the core of the double loop (lines 4-6) we have to choose between a pair pos-neg' and a pair neg-pos' (where pos' and neg' are pos and neg with their ctrl ports renamed to ctrl'); this is done using a function bit (defined in lines 10,11). The inner loop (lines 3-7) connects all these pairs into a row, with the variable j ranging from n to 1 (line 3). At the end of the inner loop, an out element is added to the right of the row (line 8). In the outer loop the variable i ranges from 1 to 2^n (line 2) while all the rows are connected from south to north (line 9). The exclamation marks in lines 8 and 9 take care of the indexes added to the ports during the inner iterations.

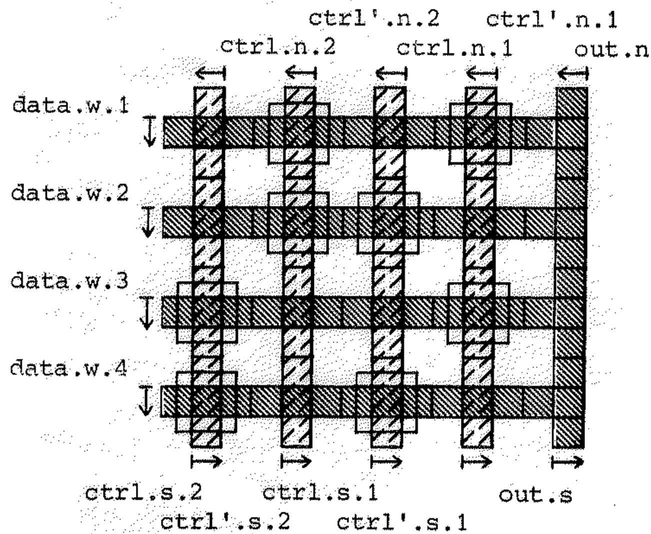

Fig.7 A selector.

It should be emphasised that the program in Fig.6 contains no explicit geometric information, and this is to be expected for many common VLSI subsystems. The double loop (array) pattern is also very common in structural design, and many other interesting examples can be produced by the use of parameterization and recursion.

REFERENCES

Buchanan,I. (1980). "Modelling and Verification in Structured Integrated circuit Design", PhD thesis, Dept. of Computer Science, University of Edinburgh.

Burge,W.H. (1975). "Recursive Programming Techniques", Addison-Wesley, Reading, Mass.

Cardelli,L. (1981). "Sticks & Stones: An Applicative VLSI Design Language", Internal Report CSR-85-81, Dept. of Computer Science, University of Edinburgh.

Goguen,J.A., Thatcher,J.W. and Wagner,E.G. (1978) "An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types", in "Current Trends in Programming Methodology", Vol IV (ed. R.T.Yeh), Prentice-Hall.

Gordon,M.J., Milner,R. and Wadsworth,C.P. (1979). "Edinburgh LCF", Lecture Notes in Computer Science, N°78, Springer-Verlag.

Mead,C. and Conway,L. (1980). "Introduction to VLSI Systems", Addison-Wesley.

Milner,R. (1979). "Flowgraphs and Flow Algebras", Journal of the ACM 26(4).

Rowson,J.A. (1980). "Understanding Hierarchical Design", PhD thesis, California Institute of Technology, Dept. of Computer Science.

Williams,J.D. (1977). "Sticks; A New Approach to LSI Design", Master's thesis, Massachussets Institute of Technology, Dept. of Computer Science.