HOPLA - PLA OPTINIZATION AND SYNTHESIS

S. Wimer, N. Sharfman

NATIONAL SEMICONDUCTOR INC.
P.O. Box 3007, Hertzeliya B 46104, ISRAEL

ABSTRACT

A system that automates Programmable Logic Array optimization and synthesis for VLSI design is described. PLA logic is defined via a high level Hardware Definition Language. After translation to table representation comes the logic optimization phase, carried out according to a user defined optimization criterion. The geometrical optimization phase follows, supplemented by a manual interactive graphic PLA editor. The system outputs symbolic Layout of the PLA which can be translated into polygon-level layout.

INTRODUCTION

While the ultimate Silicon Compiler solution for design automation in VLSI is being pursued by the academic community, the industry cannot afford to wait until such tools are economically feasible for general purpose, high volume VLSI chips. Seeking an interim solution, we are faced with alternatives, such as:

- Automatic parts of the design process.
- Achieve full automation for a selective "breed" of logic.

Natural condidates for all-out automation are regular structures : Micro-code ROM implementations lend themselves easily to automation, from high level definition (via microcode assemblers) down to the final layout. Next in line are Programmable Logic Arrays (PLA's). Those however, due to their greater flexibility, present a grander challenge - primarily in logic and geometrical optimization.

Recent past trends clearly indicate that an increasing percentage of VLSI chip area is occupied by PLA's. Design by means of those structures is easier, faster and less error prone along the whole life cycle of a VLSI chip i.e. - logic design, layout, debug and possible upgrades.

HOPLA is an integrated system for PLA optimization and synthesis. Figure1 presents its block diagram : The user specifies his logic thru the high level Hardware Definition Language. The translation phase generates a Logic Array which serves as input to the Expansion Module. The Logic Array is then preprocessed to resolve ambiguities and add information. At the Logic Optimization phase which follows, the user selects a "cost function" which defines the optimization criterion. The Geometric Optimization module deals with the actual implementation of the PLA. It provides an interactive graphic PLA editor thru which the layout of the PLA can be manipulated at the symboliclevel. An automatic optimization algorithm can be applied in addition to manual functions.

The output of the Geometry phase is a symbolic layout data base. It is a straight - forward task to convert this format to polygon-level layout.

We shall now describe the various phases of the system in more detail.

SYSTEM INPUT

The primary input to HOPLA is the specification of the logic thru a Hardware Definition Language. This language also serves as input to our simulation facility. This enables the user to simulate his logic, within its environment prior to implementation. After he verifies it, the same description is used for HOPLA.

The following example for logic description via the Hardware Definition Language, was originally specified in terms of a Logic Array:

```
BLOCK ADDER
; Perform two bit addition Z : = X++Y

DEFINE REG 2 X "FIRST 2 BIT OPERAND"
DEFINE REG 2 Y "SECOND 2 BIT OPERAND"
DEFINE REG 3 Z "3 BIT RESULT"
DEFINE LOCAL C_IN, C_OUT
; The local variables serve to store the carry
; bits
; Define bit 0
Z(0) : = X(0) # Y (0)
; Handle carry
C-OUT(0) : =  X (0) * Y (0)
C_IN(1) : = C_OUT (0)
; Define bit 1
Z(1) : = X (1) # Y (1) # C_IN (1)
; Handle carry
C_OUT(1) : = 'C_IN(1) *X(1) *Y(1) +
 %           C-IN(1)* (X(1) +Y(1) )
C_IN (2) : =C_OUT(1)
; Define bit 2
Z (2) : = C_IN (2)
END BLOCK
```

As can be seen from the above example, the operators that can be used in logic specification are:

+ (or), *(and), #(xor), '(complement) and parentheses.

Intermediate variables may be used to enhance readability. Those are defined as LOCAL variables. They will not be implemented as PLA input/output.

As an alternative to the symbolic HDL specification of logic, it is possible to input a Logic Array directly to HOPLA.

TRANSLATION AND PREPROCESSING

The first phase of HOPLA translates the HDL specification of the logic to table format[2]. This table represents a "Sum of Products (SOP)" realization of the logic.

The 2 bit adder example specified in the previous section thru the Hardware Definition Language, assumes the following table representation after translation.
(Don't care situations for inputs are denoted by X) :

X(0)	Y(0)	X(1)	Y(1)	Z(0)	Z(1)	Z(2)
0	1	X	X	1	0	0
1	0	X	X	1	0	0
1	1	1	1	0	1	0
1	1	0	0	0	1	0
0	X	0	1	0	1	0
1	0	0	1	0	1	0
0	X	1	0	0	1	0
1	0	1	0	0	1	0
1	1	1	X	0	0	1
1	1	0	1	0	0	1
0	X	1	1	0	0	1

Note that the equations or the SOP table do not always include the full description of the logic. This is the case whenever not all possible input combinations are specified. In the 2 bit adder example, the output for the input combination 0000 is not specified. Since the table entries define all cases where at least one of the outputs is a logical "1", two possibilities exist for such cases:

a) The output values for those input combinations not defined by the user are logical "0".

b) The output values are "don't cares".

In the preprocessing phase, the user is asked to determine the output values for those input combinations. If they are "don't cares" (a situation that might occur whenever those combinations can never be realized), the subsequent Logic Optimization can be improved.

LOGIC OPTIMIZATION

This module manipulates the Logic Array produced in the previous phase in order to optimize it according to a user defined optimization criterion. Natural candidates for optimization criteria are related to PLA area, power consumption, delay considerations and so on. Those considerations are realized thru minimization of the number of product terms, literals etc.

Considerable efforts have gone into logic optimization in the past. The prioneering MINI[3] and PRESTO[4] programs, served as a basis for other PLA optimization systems[5]. These programs however always use a fixed optimization criterion. Their algorithm is heuristic. As a result absolute minimum is not guaranteed.

Our optimization algorithm is non-heuristic, thus the solution is always the best possible. This is achieved by using linear and integer programming techniques.

In fact, we trade off more general and accurate optimization procedures against execution time. Since we do not encourage implementation of logic with very large PLAs, because area utilization becomes inefficient, this price is worth while. We can realistically handle tens of input/output lines and hundreds of implicants.

To illustrate some performance characteristics of HOPLA on VAX 11/780:

The 2 bit adder optimization on the number of product terms has taken less than 1 CPU second. The resulting table is:

X(0)	Y(0)	X(1)	Y(1)	Z(0)	Z(1)	Z(2)
0	1	X	X	1	0	0
1	0	X	X	1	0	0
1	1	0	0	0	1	0
0	X	0	1	0	1	0
0	X	1	0	0	1	0
1	1	1	1	0	1	0
1	1	1	X	0	0	1
X	0	0	1	0	1	0
X	0	1	0	0	1	0
X	X	1	1	0	0	1
1	1	X	1	0	0	1

The Mead and Conway light controller example[7], initially had 10 product terms, with 5 inputs and 7 outputs. It took 2 CPU seconds to reduce to 9 product terms. The same logic was reported to have taken 1/2 CPU hour on a DEC-10 to achieve the same result[6].

Logic requiring 8 inputs, 31 outputs and 49 product terms was reduced to minimum product terms in 13 seconds and to a minimum literals in 44 seconds.

Another table with 13 inputs, 20 outputs and 129 product terms, was optimized in 15 minutes.

Note that the system performance is not a function of the number of input/outputs and product terms only, but also depends upon the intrinsics of the logic.

GEOMETRIC PLA MANIPULATION

This phase is aimed specifically at minimizing the area occupied by the PLA. Its output is a symbolic layout (Sticks Diagram), which is a high

level representation of the layout.

The starting point is the classical representation of PLA, defined by Mead and Conway [7]. This is obtained directly from the Logic Array and is drawn on screen upon entry to the Geometrical Reduction phase. A variety of commands to improve the geometry of the PLA automatically and/or manually is provided.

Automatic editing commands perform input/output or impliant folding. Folding is a well known area reduction technique [8]. We use heuristic graph theoretical methods to represent and implement it.

Manual command repertoire includes information commands, such as HELP (get list of commands) and SIZE (get current area), SAVE command, PLOT for producing hard copy and so on. The most important manual commands are MOVE IO and MOVE IMP. The user is prompted with a cursor on the screen and is asked to point at an input, output or product term and its destination. These commands can be used for manual folding, creating diagonal layout, segmentation, etc.

The CHECK command performs a symbolic level design rule check.

Figure 2 displays a symbolic layout for the Mead and Conway light controller. Input, output and product terms folding are demonstrated. The O's and X's represent transistors in the AND and OR planes respectively. A 37% area reduction is achieved on top of the optimized regular layout, under the constraint of keeping every input adjacent to its complement.

No command in the Geometric Manipulation phase can destroy the logic as it was defined by the Hardware Definition Language description. The only way to change the logic is by going back to this description (or edit the logic array).

A simple conversion program transforms the final layout into actual polygon-level representation in the appropriate format (CIF,CALMA,etc.)

CONCLUSIONS

HOPLA is a "programmable" mini silicon-compiler for PLA's. It provides an answer to one of the more accute problems in Design Automation - integration. It is connected to logic simulation on one hand and to traditional layout systems on the other. The various modules of HOPLA behave like a system, namely the output of one serves as input to the next.

Flexibility is provided via user defined optimization criteria, manual geometrical editing functions etc.

Logic integrity is preserved throughout the system. Accidental logic bugs are thus avoided.

The system is not capable of processing huge PLA's. We do not consider this is a major handicap since such structures tend to be inefficient. A future logic segmentation module will divide large logic blocks into loosely coupled sub-blocks. Each of these sub-blocks will in turn serve as input to HOPLA.

REFERENCES

1) H. Fleischer, L.I. Maissel, "An Introudction to Array Logic", IBM J. Res. Devlop.,Vol 19, pp. 98-109, Mar.1975.
2) D.L. Dietmeyer, "Connection Arrays from Equations", Journal of Design Automation and Fault-Tolerant Computing, April, 1979, pp.109-125.
3) S.J. Hong, R.G. Cain, D.L. Ostapko, "MINI: A heuristic approach for logic minimization", IBM J. of Research and Development, Sept. 1974, pp. 443-458.
4) D.W. Brown, "A state machine synthesizer - SMS", Eighteenth DA Conference, ACM,IEEE,1981, pp. 301-305.
5) S. Kang, W.M. Van Cleenput, "AutomaticPLA Synthesis from DDL-p Description", Eighteenth DA Conference, ACM, IEEE, 1981, pp. 391-397
6) B. Teel, D. Wilde, "A Logic Minimizer for VLSI PLA Design", Nineteenth DA Conference, ACM, IEEE 1982, pp. 156-162.
7) C. MEAD, L. CONWAY Introduction to VLSI systems, Addison Wesley, 1980.
8) G.D. Hachtel, A.R. Newton, A.L. Sangiovanni-Vincentelli, "An algorithm for optimal PLA folding", IEEE transactions on CAD of Integrated Circuits and Systems, April, 1982, pp.63-77 and "Techniques for Programmable Logic Arrays Folding", Nineteenth DA Conference ACM,IEEE,pp. 147-155.

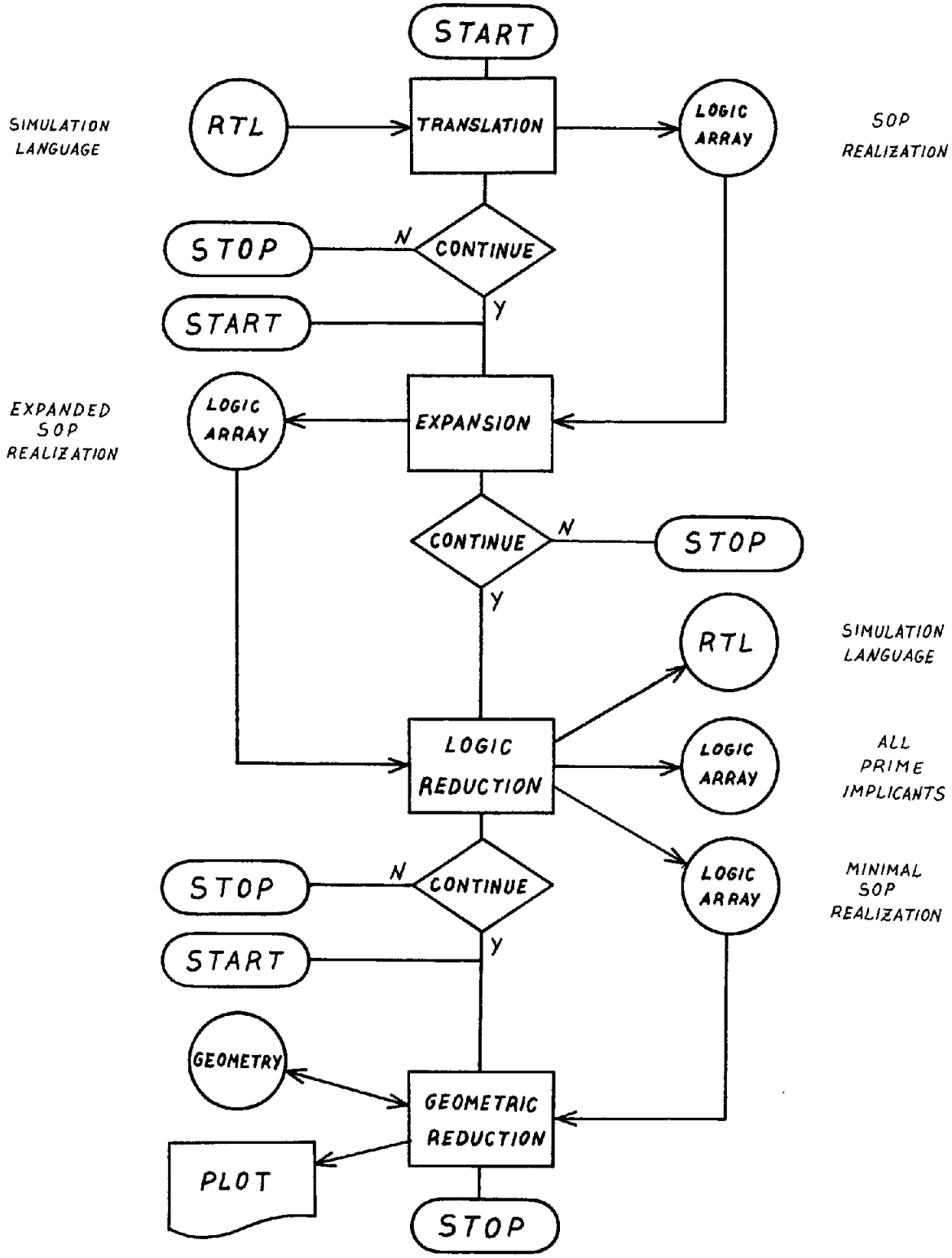
START
SIMULATION LANGUAGE
RTL
TRANSLATION
LOGIC ARRAY
SOP REALIZATION
STOP
N
CONTINUE
START
Y
EXPANDED SOP REALIZATION
LOGIC ARRAY
EXPANSION
CONTINUE
N
STOP
Y
RTL
SIMULATION LANGUAGE
LOGIC REDUCTION
LOGIC ARRAY
ALL PRIME IMPLICANTS
STOP
N
CONTINUE
LOGIC ARRAY
MINIMAL SOP REALIZATION
START
Y
GEOMETRY
GEOMETRIC REDUCTION
PLOT
STOP

FIGURE 1

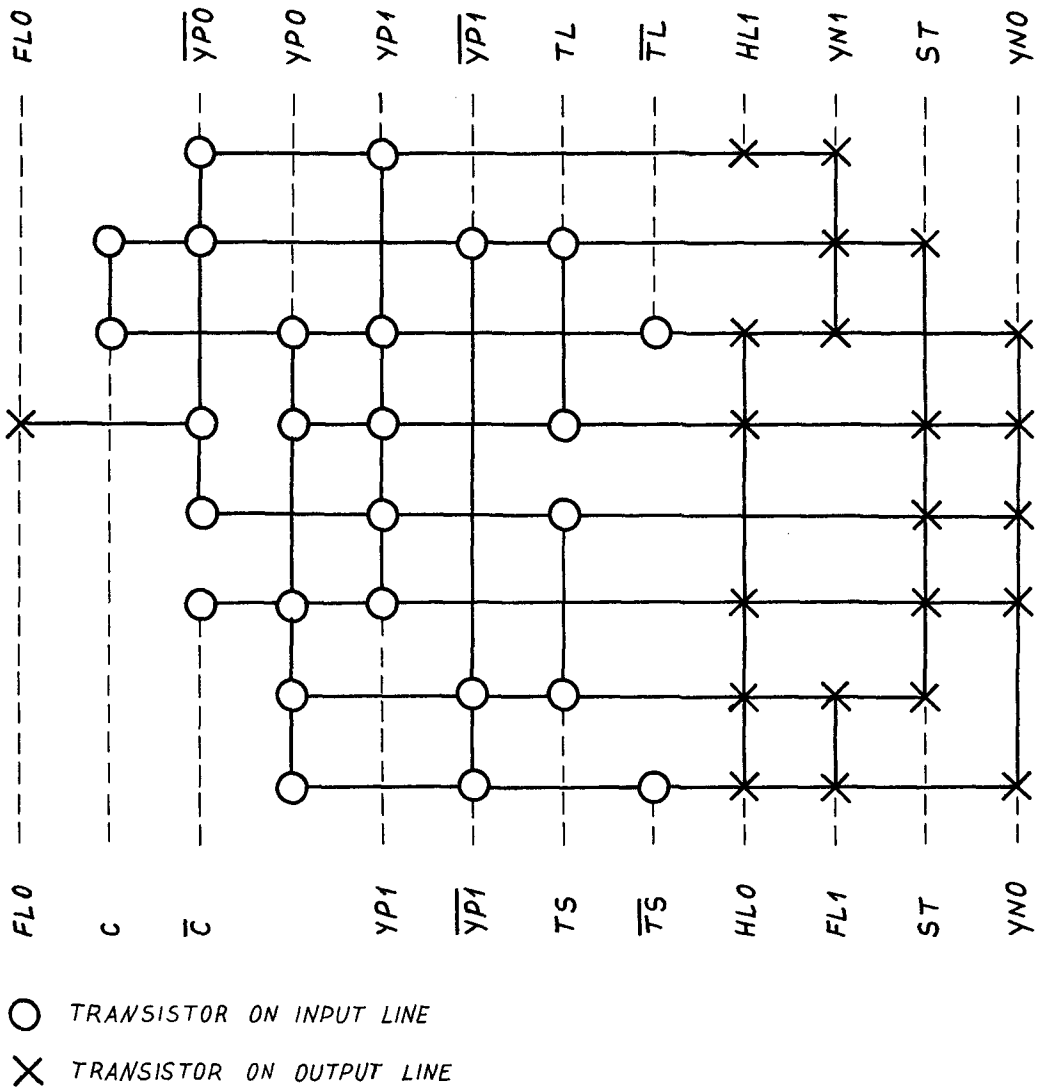
FL0
$\overline{YP0}$
YP0
YP1
$\overline{YP1}$
TL
$\overline{TL}$
HL1
YN1
ST
YN0
FL0
C
$\overline{C}$
YP1
$\overline{YP1}$
TS
$\overline{TS}$
HL0
FL1
ST
YN0
TRANSISTOR ON INPUT LINE
TRANSISTOR ON OUTPUT LINE

FIGURE 2