VLSI architecture

Edited by
B. Randell and **P.C. Treleaven**
University of Newcastle upon Tyne
England

Contributions by
Speakers at the Advanced Course on VLSI Architecture held at the University of Bristol, 1982.

Prentice/Hall PHI International

Englewood Cliffs, New Jersey London Delhi
Rio de Janeiro Singapore Tokyo Toronto Wellington

Library of Congress Cataloging in Publication Data

Main entry under title:

VLSI architecture.
Includes bibliographies and index.
1. Electronic digital computers--Circuits--Addresses essays, lectures. 2. Integrated circuits--Very large scale integration--Addresses, essays lectures.
3. Computer architecture--Addresses, essays, lectures
I. Randell, Brian, 1936- II. Treleaven, Philip
III. Title: V.L.S.I. architecture.
TK7888.4.V57 1983 621.3819'5835 82-24040
ISBN 0-13-942672-8

British Library Cataloguing in Publication Data

VLSI architecture.
1. Electronic digital computers—Circuits
I. Randell, Brian II. Treleaven, Philip C
621.3819'58'35 TK7888.4

ISBN 0-13-942672-8

ISBN 0-13-942672-8

PRENTICE-HALL INTERNATIONAL, INC., London
PRENTICE-HALL OF AUSTRALIA PTY., LTD., Sydney
PRENTICE-HALL CANADA, INC., Toronto
PRENTICE-HALL OF INDIA PRIVATE LIMITED, New Delhi
PRENTICE-HALL OF JAPAN, INC., Tokyo
PRENTICE-HALL OF SOUTH EAST ASIA PTE. LTD., Singapore
PRENTICE-HALL, INC., Englewood Cliffs, New Jersey
PRENTICE-HALL DO BRASIL, LTDA., Rio de Janeiro
WHITEHALL BOOKS LIMITED, Wellington, New Zealand

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ON THE LUCIFER SYSTEM

The Lucifer language

Lucifer is a language for describing circuits at the masks level based on juxtaposition of rectangles. A Lucifer expression <exp> is caracterised by an origin and a bounding box. It can be:

(mask x y w h)
(UN <exp1> <exp2> ... <expn>)
(JX <exp1> <exp2> ... <expn>)
(JY <exp1> <exp2> ... <expn>)
(TB m n <exp>)
(DS <symbol>)
(TR x y <exp>)
(ROT n <exp>)
(MX <exp>)
(MY <exp>)

A Lucifer program is a sequence of declarations and an expression. Declarations of the form

(DEFS <symbol> <exp>)

permit definition of symbols. The semantics of an expression is better described by Fig 1. The mask statement is for defining atomic rectangles. UN means union, i.e. overlapping of several expressions by identifying the origins of each expression. JX means juxtaposition along the X coordinates. Thus the origin is that of the first component, and juxtaposition is achieved by putting aside the bounding boxes of all components. Similarly, JY is juxtaposition along the Y axis. Thus juxtaposed expressions are ensured not to overlap. TB is a construct for a two dimensional array. Symbols are used to give names to an expression. TR, ROT, MX, MY are the standard transformations for making translations, rotations (of a multiple of 90 degrees), and mirror operations.

Translators from Cif to Lucifer and from Lucifer into Lucie exist. Two circuits have already been translated into Lucie which is presently our language for sending to process: an 8 bitn adder which is a small version of a sophisticated one designed by L. Guibas and J. Vuillemin [GV 82] and which has been coded in Lucifer by C. Heintz, and a circuit for computing transitive closures coded by B. Serlet.

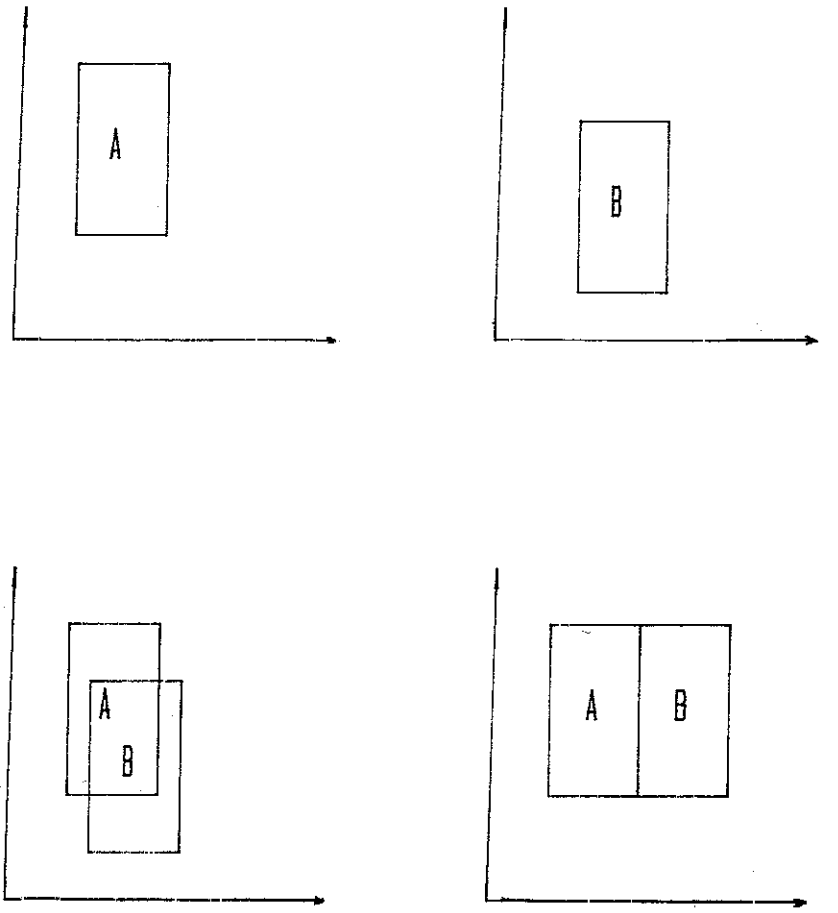

Figure 1

ON THE LUCIFER SYSTEM

The CEYX symbolic structured editor

This subsystem has been implemented by J-M Hullot. It contains an extension of Lisp and a corresponding structured editor Extensions of Lisp contain the *deftree* construct for defining trees The CEYX editor works on any kind of trees defined by this construct Moreover, the editor can work under Emacs Thus, in one Emacs buffer, in the Ceyx mode, keys are defined for moving in the tree structure with holophrasts [Mentor 80] to hide subtrees. Editing is also achieved by special keys These tree extensions permit more elegant ways of programming, and this is the way in which the Lucifer implementation is written This editor is thus strongly inspired by [Mentor 80] with a video environment

Generally, programs for translating an abstract tree structure into Lucifer or for naming wires are written by the user. The Lucifer structure is stored in attributes fields of each tree node, which permits editing on the original structure without retranslating the whole expression. The Lucifer structure can be generated without exact dimensions This permits floorplan layouts where bounding boxes are approximated by possible extra proportions attributes. Note that floorplan pictures can be generated on the Colorix graphics system from Ceyx at time of edition. Furthermore, an other processor allows outputs of floorplan on an HP722 plotter through the FLIP system [Kahn 81]

JEAN-JACQUES LEVY

The graphic editor Luciole

Luciole is inspired both by Icarus [FR 80] and Emacs. It has been implemented by G Baudet and J-J Levy. Interaction is achieved by a standard alphanumeric terminal or a Bitpad tablet with the help of a four button mouse. The implementation language is Lisp except for some basic operations which are coded in assembly language. When calling the editor, one is under the lisp environment. This allows easy writing of editing macros and the calls of different processors for generating Lucifer structures by programs or for checking expressions when editing them

The basic Luciole commands are the following ones:

select-buffer kill-buffer, list-buffers. read-file, find-file, save-file write-file for manipulating the different editing buffers or files. The user is prompted for a buffer name or file name

pan-west,east,north,south, center-screen are used for panning the displays memory The last command promts for two absolute coordinates and display the window centered on that point

zoom-1 2,4,8,+,- flush the screen at the indicated zoom The screen is centered on the cursor +,- mean increasing or decreasing the current zoom by one unit. The zoom command can be used for quick pans when zoom is greater than 1

flip-grid, redisplay. layer-none,all,metal poly diff,implant are several commands for display. *flip-grid* flushes or erases a grid. Grids steps are functions of the current zoom. *redisplay* cleans the screen The layer commands add a new layer on the already flushed layers.

Movements of the cursor is generally achieved by moving the mouse. However two commands permit some more directed ways: *show-coordinates* gives the absolute coordinates of the cursor, *move-cursor* prompts for two values for a relative move of the cursor

reset-mark, set-mark set-mark-father, son right-brother, left-brother, show-type-of-mark, set-mark-org, exchange-mark-and-cursor are different commands for marking a Lucifer expression The set-mark command finds the first expression in tree preorder containing the cursor. Notice that matching takes care of the currently displayed masks selected by the layer command Other commands move the mark along the Lucifer structure At any momment, an orange frame appears around the selected expressions Several expressions can be marked, provided they are brothers in the Lucifer structure. Moreover, an origin can be associated with the mark.

copy-mark, wipe-mark yank, input-eval, input-metal poly, diff, implant, cut prev-in-kill-ring next-in-kill-ring are the basic editing commands The *wipe* command deletes the marked expression, and puts a copy relative to the mark origin in the Luciole kill ring This ring is a garbage with 10 entries *copy-mark* does the same without deleting The *yank* command inserts the top of the kill-ring after the marked Lucifer expression plugging the origin of the marked expression and the one of the yanked expression The inputs command inserts a rectangle after the marked expression. One corner of the rectangle is the origin of the mark, an other one is the current position of the cursor. The command *input-eval* is as yank, but a lisp expression is prompted instead of inserting from the kill ring This expression must generate a Lucifer structure This is a very simple way of mixing interactive inputs and expressions generated by program. (Notice that at time of editing, one has the full lisp environment Luciole can be called under Emacs, and this is a very convenient way for debugging macro-generation programs)

begin-macro-learn. end-macro-learn, execute-la-macro. show-la-macro, save-macro are various commands for handling simple editor macros When starting recording of a macro, one hits in the menu the begin-macro-learn command Then all the commands, inputs and moves of the cursor are recorded until the end-macro-learn command One can then execute it or save it in the menu Note that the macro is executed when defining it This feature, borrowed from Emacs, allows better debugging

The EXI node extractor

The task of a node extractor is very similar to the one of a disassembler in programming languages It takes as input the layout description and produces an electric network formed by a list of triples (grid, drain, source) for each transistor. Each component of a triple is a wire, i e a merge of several elements in the mask description Thus, the main work of the extractor is to find equipotential components in the geometry of a circuit. Naming of wires is achieved by taking care of the names already given in the Lucifer structure and generates new names for unlabelled components

Our extractor, which has been implemented in Lisp by C Heintz, is structured on the Lucifer language and is parameterised on the MOS technology Structure is

easily achieved on juxtaposition nodes by keeping information on the perimeter of bounding boxes. The technology is table driven in order to define contacts or transistors The implementation depends heavily on a rectangles intersection program in order to find pieces which are connected. The program manages a sweeping line across the layout with a data structure described in [BW 80] for describing the section of the circuit Transistors are recognised in the usual way by looking at intersections of poly, diffusion and poly Furthermore, a second pass looks for pullup nodes. The algorithm is linear in the number of elements in the geometry if the circuit is well structured. It takes on Multics 1 second of cpu time for 50 unstructured rectangles.

The MOSSIM simulator

MOSSIM is a very simple simulator which is due to [Bryant 80] It has been implemented on Multics by G Huet amd J-M. Hullot. Its main characteristics are to be a non directed simulator at the transistor level and to be very efficient Time is not taken into account. Input of the simulator is the output of the node extractor Simulation is performed by giving ordered forces to each wire For instance an input wire is stronger than a VDD or ground line, which dominates pullup wires, which is greater than a normal wire At any moment, when values are given to grids, the stronger element in a connected class gives its value to three class Also, there could be undefined wires and an easy law giving the maximum of two values in some straightforward lattice structure produces the value

The problem with MOSSIM is to make it structured in the Lucifer language This is not so easy, mainly because of its non-oriented behaviour. However, experiments have shown that often, for Lucifer nodes, input and output lines can be pointed out by the extractor For instance, input lines are the one connected to some grid. Thus, an hybrid simulator mixing RTL and MOSSIM simulation is under development [WW 78].

The VRD design rules checker

Design rule checking works presently on the layout geometry. It is very similar to a pattern matching problem. About twenty 4 x 4 lambdas window patterns are checked. Moreover the VRD checker is structured with respect to the Lucifer language by keeping a 3 lambdas width boundary for each Lucifer node. This is clearly enough for checking juxtaposition nodes The checker has been implemented in Pascal and recoded in Lisp by L Gallot Furthermore, a new implementation of it taking care of the output of the extractor is under design.

Conclusion

The Lucifer system is nowadays a system still working at a pure geometrical level. It is strongly embedded in the Lisp environment, which permits easy interactions between circuits generated by programs and manual design. Further developments will naturally lead to more symbolic methods Presently, there is already work in this direction. For instance, B. Serlet is implementing some elementary channel routing algorithms, which allows the user to define routing boxes implicitely when juxtaposing two Lucifer nodes Another example is to define some sticks formalism with the use of a simple but manageable compactor (as in [Weste 81] for example) Notice that lot of the programs developped for Lucifer will then still be of use with small variations Furthermore, the Ceyx editor will be of great use for implementation. Also, higher level languages for describing circuits like Plastick [Cardelli 82], which can be compiled in a sticks framework, are of interest Finally, feasible electric simulation seems to be a rather hard but valuable problem.

References

[Kahn 81] G Kahn, *»FLIP User s Manual»*, Tech Report 2, INRIA, Jun 1981

[*Weste 81*] *N H E. Weste, »MULGA-An Interactive Symbolic Layout System for the Design of Integrated Circuits»*, Bell Journal, Vol 60, No 6, Jul-Aug 1981

[MC 80] C. Mead, L Conway, *»Introduction to VLSI systems»*, Addison Wesley, 1980, pp 115-127

[*lucie*] *A. Guyot, A Jerraya, J Raymond »LUCIE Langage Universitaire de Conception de circuits Integres pour l' Enseignement»*, Tech Report, IMAG, Grenoble, 1979

[*GV 82*] *L Guibas, J Vuillemin, »On fast binary addition in MOS technologies»*, 1982, in preparation

[*FR 80*] *D G Fairbain , J. A Rowson, »ICARUS. An Interactive Integrated Circuit Layout Program»*, 15th DAC Conf., IEEE, June 1978, pp 188-192 Also in Mead and Conway, pp 109-115

[*BW 80*] *J L. Bentley, D Wood, »An Optimal Worst Case Algorithm for Reporting Intersection of Rectangles»*, IEEE Iransactions on Computers, Vol C-29, July 1980, pp 571-576

[Bryant 80] R E Bryant, *»An Algorithm for MOS Logic Simulation»*, LAMBDA, Vol 1, No 3, Fourth Quarter 1980.

[*WW 78*] *J M. McWilliams, L C. Widdoes, »The SCALD Physical Design Subsystem»*, Tech Report 153, Dept of EE and Comp Sc, Stanford University, March 1978

[*Mentor 80*] *V Donzeau-Gouge, G Huet, G Kahn, B. Lang, »Programming environments based on structured editors the Mentor experience»*, INRIA Report No 26, July 1980.

[Cardelli 82] L Cardelli, *»Plastick»*, PhD thesis, Univ of Edinburgh, Feb 1982