Herbert Grünbacher
Reiner W. Hartenstein (Eds.)

Field-Programmable Gate Arrays:

Architectures and Tools for Rapid Prototyping

Second International Workshop
on Field-Programmable Logic and Applications
Vienna, Austria, August 31 - September 2, 1992
Selected Papers

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
D-76131 Karlsruhe, Germany

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Volume Editors

Herbert Grünbacher
Institut für Technische Informatik, Technische Universität Wien
Treitlstr. 3/182.2, A-1040 Vienna, Austria

Reiner W. Hartenstein
Fachbereich Informatik, Universität Kaiserslautern
Postfach 30 49, D-67653 Kaiserslautern, Germany

CR Subject Classification (1991): B.6-7

ISBN 3-540-57091-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57091-8 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera ready by author
45/3140-543210 - Printed on acid-free paper

Preface

This book contains papers first presented at the 2nd International Workshop on Field-Programmable Logic and Applications (FPL'92), held in Vienna, Austria, from August 31 to September 2, 1992.

The growing importance of field-programmable devices, especially of field-programmable gate arrays, was demonstrated by the increased number of papers submitted in 1992. For the workshop in Vienna 70 papers were submitted. It was pleasing to see the high quality of these papers and their international character with contributions from more than 20 countries. The following list shows the distribution of origins of the papers submitted to FPL'92 (some papers were written by an international team):

Australia:	1	Austria:	4
Canada:	1	Czechoslowakia:	1
Finland:	3	France:	2
Germany:	16	Italy:	3
Japan:	3	Norway:	3
Russia:	1	Spain:	4
Sweden:	4	Switzerland:	1
United Kingdom:	4	USA (incl. Hawaii):	21

From the 70 submitted papers, 23 were selected for this book. The first three papers discuss strategic issues and give surveys. Three papers deal with new FPGA architectures and five papers introduce methods for tools. The last twelve papers report applications focusing on rapid prototyping or new FPGA-based computer architectures.

We would like to thank the members of the technical program committee for reviewing the papers submitted to the workshop. Our thanks go also to the authors who wrote the extended papers for this issue and the reviewers for their timely work on all manuscripts. Especially we would like to thank Helmut Reinig for managing the reviewing process and handling the manuscripts and program planning.

Thanks to the sponsors of the workshop: Universität Kaiserslautern, Technische Universität Wien, IFIP Working Groups 10.2 and 10.5, Wirtschaftsförderungsinstitut der Bundeswirtschaftskammer, Wien and Bundesministerium für Wissenschaft und Forschung, Wien. We also gratefully acknowledge all the work done at Springer-Verlag in publishing this book.

June 1993

Herbert Grünbacher,
Reiner Hartenstein

Program Committee

U. Baitinger (University of Stuttgart, Germany)
P. K. Chan (University of California, Santa Cruz, U.S.A.)
B. Courtois (INPG, Grenoble, France)
C. Ebeling (University of Washington, U.S.A.)
M. Glesner (Technical University of Darmstadt, Germany)
J. P. Gray (Algotronix Ltd., Edinburgh, U.K.)
H. Grünbacher (Technical University of Vienna, Austria)
R. W. Hartenstein (University of Kaiserslautern, Germany)
F. Haq (XILINX, San Jose, U.S.A.)
D. Hill (AT&T Bell Labs, Murray Hill, U.S.A.)
W. Luk (University of Oxford, U.K.)
G. De Micheli (University of Stanford, U.S.A.)
W. R. Moore (University of Oxford, U.K.)
K. D. Müller-Glaser (University of Erlangen-Nürnberg, Germany)
P. O'Leary (Johanneum Research, Graz, Austria)
J. Oldfield (University of Syracuse, U.S.A.)
E. Pasero (University of Torino, Italy)
F. Pirri (University of Firenze, Italy)
J. Rose (University of Toronto, Canada)
M. Sami (Politechnico di Milano, Italy)
A. L. Sangiovanni-Vincentelli (University of California, Berkeley, USA)
M. J. S. Smith (University of Hawaii, U.S.A.)

Organizing Committee

H. Grünbacher (Technical University of Vienna, Austria), general chairman
R. W. Hartenstein (University of Kaiserslautern, Germany), program chairman

Table of Contents

Overview of Complex Array-Based PLDs

Günter Biehl

ISDATA GmbH - Daimlerstr. 51 - W 7500 Karlsruhe 21 - Germany

Abstract. The first PLDs based on sum-of-products arrays were PLAs and PALs. Both are special cases of the more general PML, a fed back NAND array. Some CPLDs take up the idea of the PML for product term expansion, others use various methods of product term allocation. The multiple array architecture is the way to increase the pin count of PLDs. Their logic design requires to partition the logic in consideration of the interconnect matrix of the PLD. Limited interconnect is the reason for an additional placement problem.

1 Types of Programmable Logic

Field programmable logic devices can be divided into three main groups:

- *Ramdomly addressable memories:* PROM, EPROM, EEPRPOM, RAM
- *Array logic:* e.g. PLD, PAL, GAL, PLM, FPLA, EPLD, multiple array PLD;
- *Programmable gate arrays:* e.g. ACTEL, XILINX, Quicklogic.

Implementation of logic in memories is not a problem, provided that the number of address pins of the memory is sufficient. Logic design for programmable gate arrays is similar to synthesis for gate arrays. This paper concentrates on features of array logic.

2 Programmable Array Logic

2.1 PLA and PAL

The ancestor of array logic is the very flexible *PLA* structure illustrated in figure 1a). It consists of a programmable AND-array and a programmable OR-array. It allows the allocation of any number of product terms of the AND-array for each output function. The first field programmable devices on the market were PLAs, offered by Signetics and Intersil in 1975.

Real mass applications of programmable array logic was achieved by Monolithics Memories (MMI) 1978 by a restriction of the general structure: In the *PAL* (later GAL, PLD) structure illustrated in figure 1b) the product terms are allocated to the output functions according to a fixed scheme. Therefore the OR array is replaced by a much simpler hard wiring.

In a full PLA, common product terms of different output functions need to be implemented only once and may be shared by the individual functions. In a PAL structure each function must implement it's own copy of the common product term. For the implementation of a design of a certain logic complexity, a PAL therefore generally needs more product terms than a PLA.

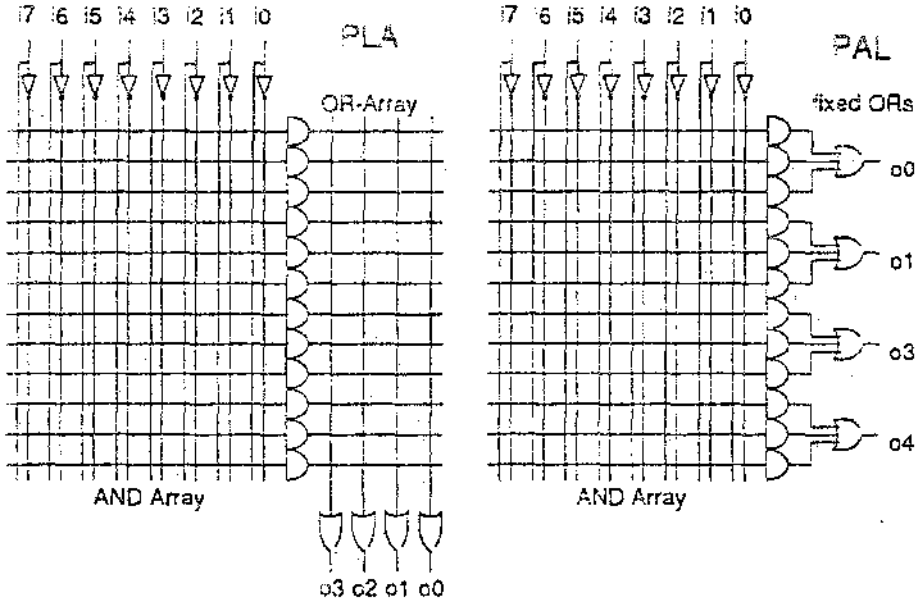

Figure 1: a) PLA structure, b) PAL structure

PLA and PAL enable the two level implementation of any boolean function of the inputs in sum of products form (AND/OR form), provided that the device has enough product terms.

Example 1: y is function of 5 inputs a, b, c, d und e:
y = a b d + a /b e + b /c d + /b /c e + b d f + /b e f
This sum of products form consists of six product terms.

2.2 PML and ERASIC

Even more general than the PLA structure is the PML (programmable macro logic), which was indroduced by Signetics years ago (figure 2). Instead of a separated AND and OR array, the PML has one homogeneous NAND array. A small fraction of NAND outputs feed the output pins, most of the NANDs are fed back to the array. Thus the PML consists of very wide NAND gates, which may be arbitrarily connected via the feedback path.This structure enables NAND networks of any

number of logic levels. The PML contains the AND/OR structure of the PLA as a special case: As we know from boolean algebra, a two level NAND/NAND network is equivalent to a AND/OR network.

Therefore PMLs offer the highest degree of freedom for logic synthesis, but unfortunately the highest degree of difficulty for a synthesis software too. Logic synthesis for PMLs is more related to multi level synthesis for gate logic than to PLD design. Unlike gate synthesis, the sole cost criterion for PMLs is the number of NANDs, as the cost of a NAND does not depend on it's number of inputs.

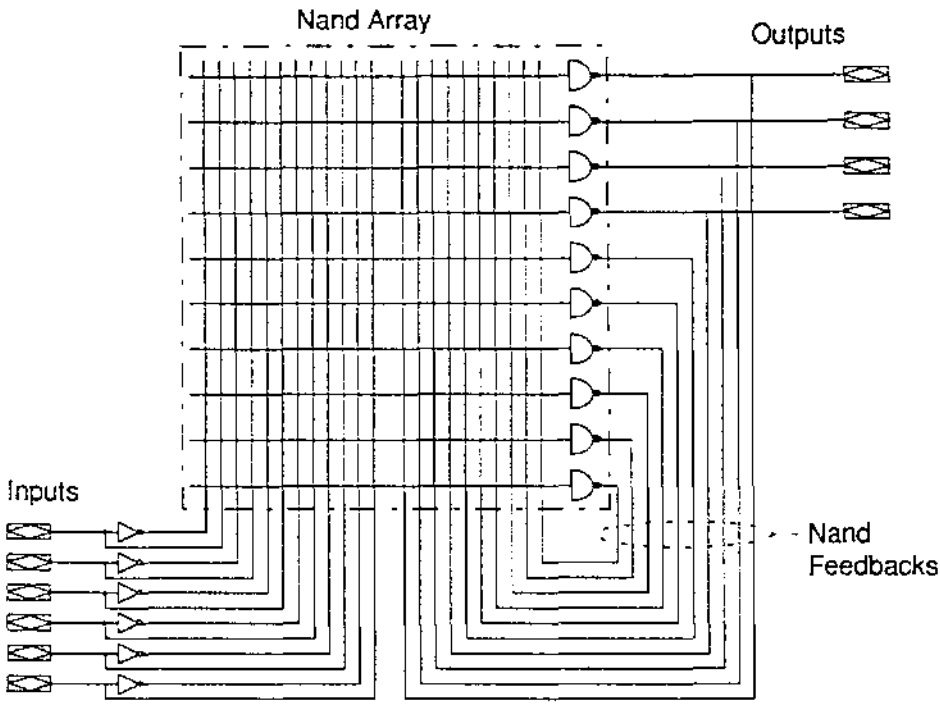

Figure 2: PML as the most general logic array structure

Example 2: the function y of example 1 can be transformed into a representation consisting of four NAND terms instead of six product terms:

$$y = a\,b\,d + a/b\,e + b/c\,d + /b/c\,e + b\,d\,f + /b\,e\,f$$

1. by factoring out 'a' out of the first two terms
$$y = a\,(b\,d + /b\,e) + b/c\,d + /b/c\,e + b\,d\,f + /b\,e\,f$$

2. by factoring out '/c' out of terms 2 and 3
$$y = a\,(b\,d + /b\,e) + /c\,(b\,d + /b\,e) + b\,d\,f + /b\,e\,f$$

3. by factoring out 'f' out of terms 3 and 4

$$y = a\,(b\,d + /b\,e) + /c\,(b\,d + /b\,e) + f\,(b\,d + /b\,e)$$

4. by factoring out '(b d + /b e)' out of the three terms

$$y = (b\,d + /b\,e)\;(a + /c + f)$$

5. introducing the intermediate variable z = (a + /c + f):

$$y = (b\,d + /b\,e)\;z$$
$$y = (b\,d\,z + /b\,e\,z)$$

6. transformed into NAND:

$$z = /(/a\,c\,/f)$$
$$y = /(\;/(b\,d\,z)\;\;/(/b\,e\,z))$$

Hence the implementation of y (and z) in the PML requires only 4 NAND terms according to figure 3, whereas a PLA or PAL implementation requires 6 product terms. Notice that the NAND form has increased delay because it requires three levels of logic.

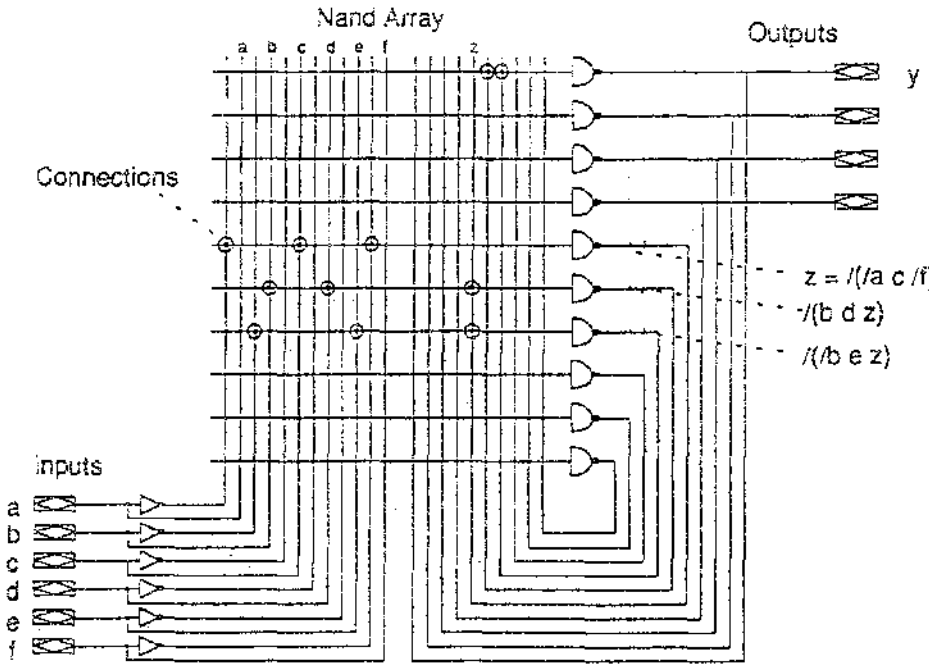

Figure 3: NAND implementation of function y: only 4 NANDs instead of 6 product terms

Up to now no vendor independent PLD compiler offers an optimizing NAND synthesis for PMLs. This might be the reason, why the ideal PML stucture is so rarely used today.

The ERASIC devices of EXEL are closely related to the PML. Instead of the NAND array the ERASICs consist of a fed back NOR array. In the same way as we can implement a two level sum-of-products in NANDs, we can implement a two level product-of-sums in NORs. The real benefit of the ERASIC's NOR array can only be made available by a multi level logic synthesis. The lack of powerful synthesis software might be main reason for the ERASIC's unsuccessfulness.

3 Increasing the Pin Count by Multiple Array Devices

Today PLD designers require PLDs with up to 100 - 200 I/O pins. When trying to increase the complexity of PLDs without sacrificing speed, PLD manufacturers obviously cannot increase the arrays homogeneously. The number of product terms is also limited by the speed required for most PLD applications.

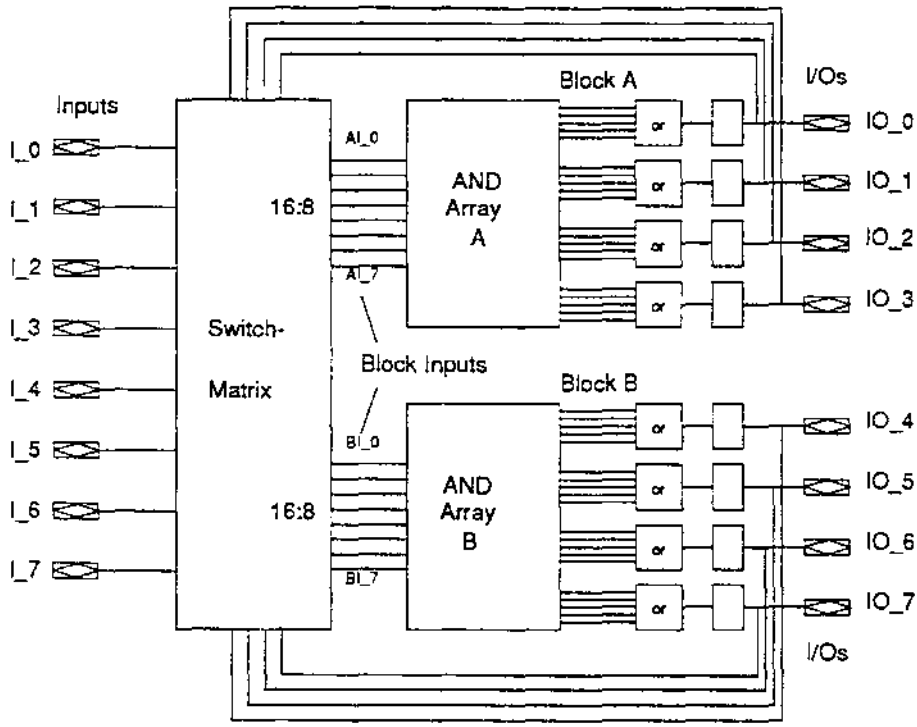

Figure 4: hypothetical example of a multiple-array-PLD consisting of two PLD blocks

Therefore all the new architectures of array based PLDs make compromises. Instead of a big homogeneous logic array, which would be most desirable from the designers point of view, they contain 2 - 32 smaller PLD blocks, which can be connected via a

programmable interconnect array (or switch matrix). Figure 4 illustrates the principle using a hypothetical device which consists of the two PLD blocks A and B. The eight inputs of each block can be fed by a selection of the total 16 signal sources of the device (8 inputs and 4 feedbacks from each block).

Even if todays multiple array PLDs have much more than 100 signal sources (inputs and feedbacks) the individual arrays have only 20 - 40 inputs. Thus very large devices can be build which operate at high speed. The additional delay which comes from the interconnect array is moderate and does not depend on the particular signal path. Because of this regular structure the delay is easily predictable.

The details of the switch matrix depend on the particular device type. The best case is switch matrices which are able to route any selection of signal sources to any PLD block. For this flexible type of switch matrix the design method may be rather simple: Equations which share input variables are preferrably grouped together in one block, as long as the total number of block inputs is not exceeded. This problem is known as block partitioning.

On the other side there are switch matrix architectures which have strong impact on the logic design process. If the switch matrix consists only of small multiplexers, only few combinations of signal sources can be routed to a PAL block. In this case the good block partitioning is not enough. Moreover it is necessary to pin the signals such that the required feedback paths are available in the interconnect array.

Examples of multiple array PLDs are: the MAX families of Altera, the MACH family of AMD, Xilinx (PlusLogic) Hyper-family, and the pLSI-family of Lattice.

4 Architectural Tricks to Increase Logic Complexity

Very often not only the pin count, but also the product term count available per output, is a bottleneck of PLD devices. Therefore all the different architectures of newer complex PLDs offer very different compromises to allow logic complexity while maintaining high speed.

4.1 Increasing the number of product terms for some outputs

The simplest method makes use of the fact, that almost never do all equations of a PLD have the same complexity. Therfore the number of product terms varies for different outputs of a PLD. Thus the total number of product terms of a device is moderate and nevertheless some few functions may be very complex. Certainly this feature is one of the main reasons for the sucess of AMD's 22V10 architecture with 8 to 16 product terms per output.

Newer device architectures avoid delay problems which come from large numbers of product terms. They contain only few (3 - 7) private product terms per macrocell, but there are different expansion mechanismes:

One of those is used in AMD's MACH family and Lattice's pLSI family. Their *product-term-allocation* enables a macrocell to allocate the product terms of its neighbours if needed. A first approach of this method was was used years ago in MMI's product-term-sharing PAL series 20RS10.

Figure 5 shows as an example the product term allocation array of the pLSI 1032. The programmable 4x4 array enables an arbitrary allocation of the 4 product term groups to the outputs of the logic block.

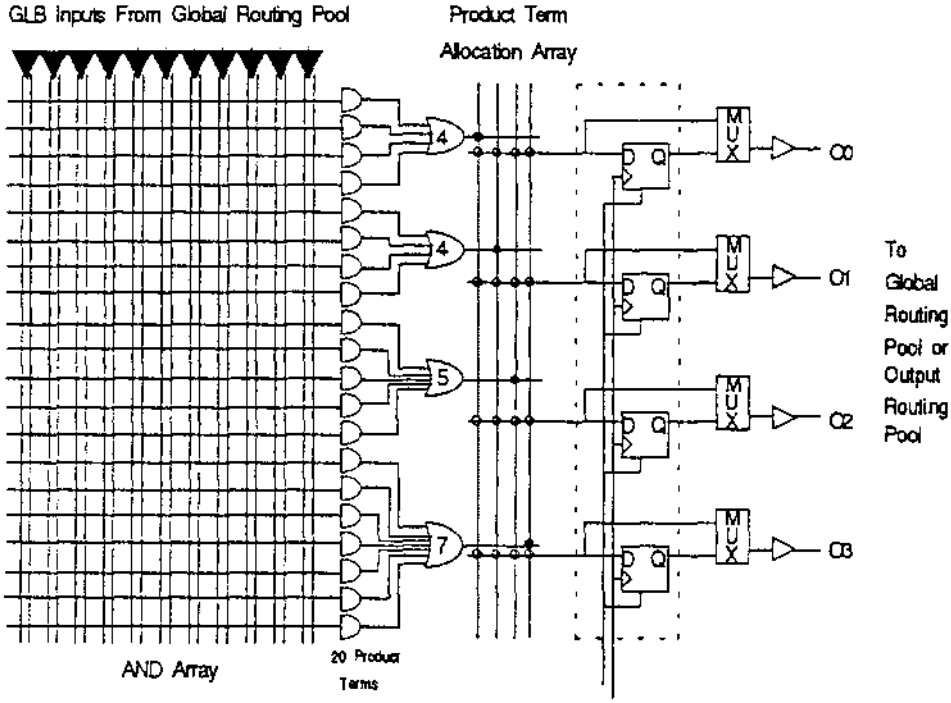

Figure 5: Example for Product-Term-Allocation: pLSI 1032

One can configure

- one function of 20 product terms or
- two functions of 12 and 8 product terms or
- three functions of 9, 7, and 4 product terms or
- four functions of 7, 5, 4, and 4 product terms

in one block.

The product term allocator of AMD's MACH family is very similar. Four product terms are available per macrocell. Each macrocell can borrow the product terms of up to three of it's neighbours and thus functions may use up to 16 product terms.

4.2 Multi level logic

In Altera's MAX family, each function has even only three dedicated product terms. At the first glance this seems to be few, but it is compensated by a very clever expansion mechanism. As illustrated in figure 6, each macrocell brings in two expander terms, which can be allocated by any other macrocell of the PLD block. The logic type of all the terms is NAND.

The three private NAND terms are tied to a NAND in the macrocell. This NAND/NAND architecture is equivalent to a AND/OR form as known from Boolean algebra. Thus this part of the MAX device implements the sum-of-products common to all PALs.

The expander terms are directly fed back to the NAND array, just as the NAND feedbacks of Signetics' PML family. All the terms of the PMLs can be considered as expander terms. Thus the expander terms enable the implementation of multi level logic as described above for the PMLs. Furthermore, by cross cuppling of NAND terms one can build asychronous filpflops.

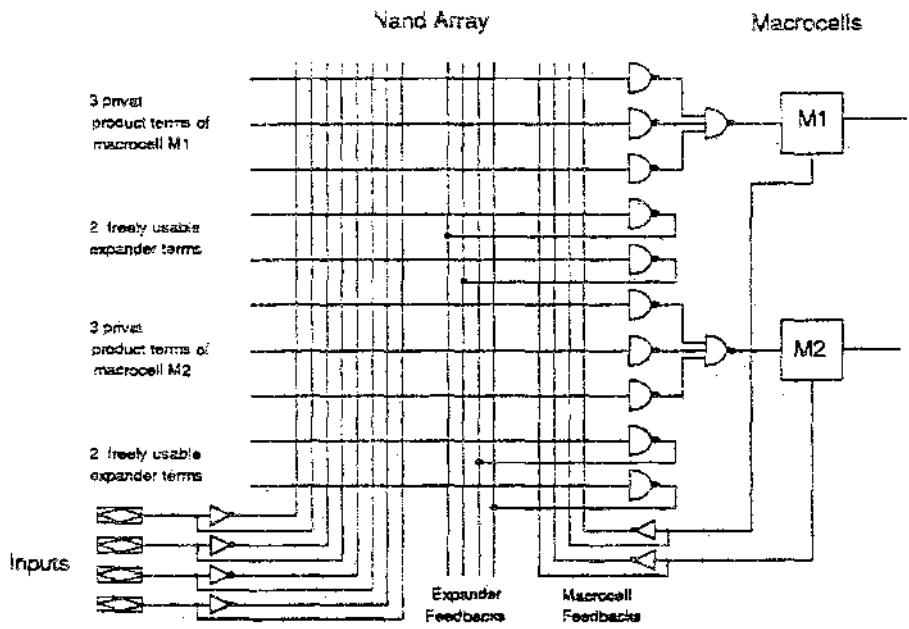

Figure 6: Product terms and expanders of the MAX macrocell

Unlike other PLDs, when designing logic for MAX devices, the product term count of the sum-of-products form is not sufficent to decide wether a macrocell can implement a function or not. Each function of more than three product terms can be implemented by use of the expander terms. In this case the objective of an optimization procedure is the use of as few expanders as possible, as the total number of available expanders is limited.

The drawback of the expanders is increased delay which is dependent on the levels of expanders. A second objective of an optimization procedure is to keep the level of logic as low as possible. Both criteria are considered in the MAX fitter of ISDATA's LOG/iC Compiler.

4.3 Partitioning of the product term array in the time domain

A completely new aproach speeding up sequential devices was introduced by National Semiconductor. The MAPL family of NS restricts the general PLA structure. The product terms of the AND-array are not allocated to the functions as in PALs, but are allocated to the states of a FSM. The consequences of this partitioning are as follows.

The product term blocks of the MAPL as shown in figure 7 must not be confused with the PAL blocks of multiple array devices. For the MAPL, in each clock cycle of the FSM only one single block of product terms is active. Of course this does not reduce the chip area of the whole array, but it reduces the power consumption of the device. The product terms of the current block are enabled by decoding of the select bits. It is the contents of the select register which decide on the selection of the block.

By an appropriate state asignment of the FSM and by the corresponding placement of the product terms within the AND array one can activate, at any time, only the product terms necessary for the actual state transition. Even if, at any time, only one block is active, functions of more product terms than available in one block can be built. Of course this architecture is for sequential logic only.

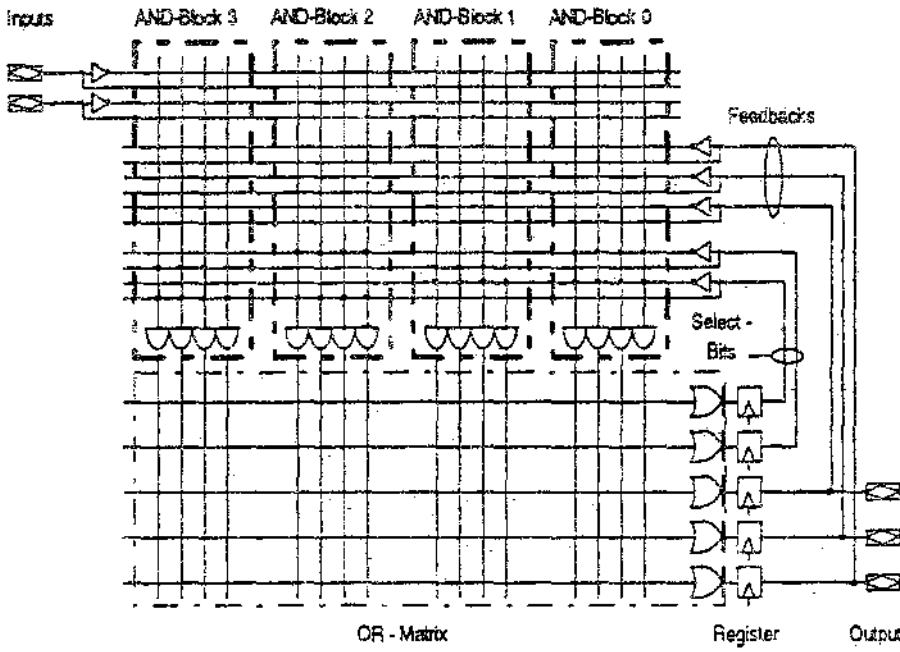

Figure 7: Architecture of the MAPL (simplified)

Extensions of the MAPL devices are available which combine the AND blocks with a classical PAL block, thus enabling combinatorial logic too.

5 Conclusion

The ideal architecture of array based PLDs was a huge NAND array (or NOR) of 100-200 inputs and at least as many fed back NANDs. Because this array is not feasible, compromises in the array size and architecture are necessary in order to achieve the speed requirements. The PLD vendors offer solutions based on very different compromises. The selection of the best solution depends on the application and can not be decided generally.

Technologies and Utilization of Field Programmable Gate Arrays

Jouni Isoaho[1], Arto Nummela[1] and Hannu Tenhunen[2]

[1] Tampere University of Technology, Signal Processing Laboratory
P.O.Box 553, SF-33101 Tampere, Finland
Tel. 358-31-3161876, Fax 358-31-3161857
[2] The Royal Institute of Technology, Institute of Electronic System Design
S-10044 Stockholm, Sweden. Tel. 46-8-7907810, Fax 46-8-103925

Abstract. Advanced CMOS technologies provide continuously more dense and complex integrated circuits, increasing the possibility to utilize Field Programmable Gate Arrays (FPGAs) as a competitor of Mask Programmable Gate Arrays (MPGAs) and as a prototyping device. In this paper, we will overview different FPGA technologies, design systems, application areas, and future trends. The emphasis is on looking these issues from the designer and application point of view instead of technology. The denotion FPGA is used to cover both the FPGAs and complex Programmable Logic Devices (PLDs).

1 Introduction

Traditionally, MPGAs are used for rapid prototyping of ASICs to speed up considerably the design process compared to full-custom design [20]. As the utilization of low power CMOS technologies made it possible to integrate large amounts of logic into a single chip, the programmable devices became a reasonable alternative to MPGAs in several applications, like in consumer electronics, industrial control systems and even in communication. In late 1980s, the size of FPGAs reached several thousands of usable gate array gates making them a reasonable solution for small size and volume ASICs. The development also started the boom of rapid prototyping of ASICs. Due to the fast technological and architectural development of FPGAs, the economical break-even point between MPGA and FPGA for the project is not simply anymore. The selection of the prototyping approach also depends on the application, the stability of the specification and the type of prototyping needed.

Because there are no standards for comparing the different types of FPGAs, and a wide diversity of FPGA technologies [3] and development tools are available, the selection of a suitable FPGA solution (FPGA technology and tools) for the application is not straightforward and general rules cannot be provided. To facilitate the selection and to form good quidelines for it, updated technological information of devices and supporting design tools are needed. Also the trends of future development should be known in order to create the long range strategy for the design group.

[3] over 30 vendors if both large and small devices is taken into account

2 Technology Overview

The comparison of the technology development of memories, MPGAs and FPGAs is presented in Figure 1 [18] [1] [12]. Memories, as a state of the art application, is used to present the technological limits from the beginning of FPGA history. Today, FPGAs and MPGAs are processed using the same size of dimensions and they are all the time coming closer to the design rules used in memory devices. In the price competition, it has to be remembered that FPGAs are high volume standard products.

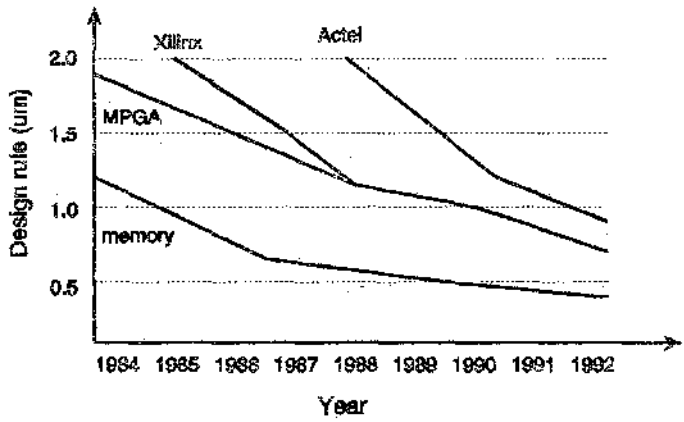

Fig. 1. Smallest feature sizes of memories, MPGAs and FPGAs.

As compared to MPGAs, FPGAs provide faster inhouse prototyping (implementation) with smaller NRE costs, but at the same time, they have lower operating speed and logic density on silicon. System speeds up to 30 to 60 MHz are possible using the present devices according to the FPGA vendors (the real range is dependent on application). In addition, the maximum system speed depends also on the results of the placement and routing tool in several device families. The largest FPGA devices available are about 10 kgates at present, when MPGAs provides more than 10 times higher usable logic capacity. Due to the considerably smaller gate capacity in FPGA devices, large designs have to be split into several devices decreasing the manageability of the design process and making the resulting system less reliable. For this reason, efficient partitioning algorithms and tools are needed. FPGAs provide usually quite a large number of I/O pins, several FPGA families much more than 100, which is quite useful, when implementing large systems and the design has to be split into several FPGAs. Especially with bit-parallel architectures, a large number of additional I/O pins might be needed due to the partitioning.

The FPGA implementation of the design is the matter of hours or days if no Printed Circuit Board (PCB) is needed to process. The corresponding processing

time of MPGAs is normally several weeks (2 - 8 weeks) [12]. Also the hard competition between MPGA foundries and the fast turnaround Multi Project Chip (MPC) and Multi Product Wafer (MPW) type of services has lowered the NRE costs of silicon products adding pressure to FPGA market and making the selection between FPGA and MPGA more unclear in many cases. Basic technologies and FPGA versus MPGA cost comparisons are presented e.g. by Smith in [21].

In FPGA devices, the *programming technology* defines whether the device is programmable only once (antifuse), several times (EPROM, EEPROM) or "infinite" number of times (SRAM). The division of FPGAs according to the programming technology is presented with some examples in Table 1 [1]. Within the FPGAs, antifuse solutions, dielectric and amorphous silicon ones, provide remarkably better area effectiveness than the reprogrammable memory elements, but the total gate density cannot be stated exactly using purely the properties of programming elements, because the size of logic elements, the rest of routing area and the configuration logic also add the total area and they vary much in different architectures. The antifuses provide also the fastest interconnections due to the smallest internal resistance and capacitance, but the length of routings and the speed of actual logic element affects the overall performance. The volatile static RAM based FPGA needs an external memory device or computer cable for loading the configuration data unlike the other technologies, which can be used as standalone circuits.

	SRAM	Dielectric anti-fuse	Amorphous silicon anti-fuse	EPROM	EEPROM
Volatile	yes	no		no	
Programmable	infinite times	one time		several times	
Resistance (ohms)	500	250-500	25-100	1000	1000
Capacitance (pF)	50	2	1	10	20
Approximate area (μm^2)	50	1.5	1.0	10	20
Examples	Algotronix Plessey Xilinx	Actel TI *	Crosspoint QuickLogic	Altera Plus Logic Atmel	AMD NS Lattice

Table 1. Programming technology overview (* second source).

2.1 FPGA Architecture Overview

The FPGA architectures can be classified in two different ways, by the routing architecture and, by the granularity (size and flexibility) of logic cell. There exist roughly two different kinds of optimization goals. If programming elements are small then the use of the logic capacity in the architecture should be optimized, and vice versa, if the routing element is large, the architectural optimization has