

APES: AN INTEGRATED SYSTEM FOR BEHAVIORAL DESIGN, SIMULATION AND EVALUATION OF ARRAY PROCESSORS

Fausto DISTANTE, Vincenzo PIURI

Department of Electronics - Politecnico di Milano
Piazza L. da Vinci 32, I-20133, Milano, Italy

ABSTRACT

Design, simulation and evaluation of array processor architectures is a complex task which may require a set of sophisticated tools. APES is an integrated CAD environment particularly designed to assist the array processor designer at the behavioral level.

The array architecture can be defined by using a behavioral description language or a graphic editor: suited routines provide conversion from textual to graphic representations and vice versa. The array behavior and performances can be evaluated by simulating its computation and observing the data flow at different abstraction levels (functional unit, processing element, array).

Fault-tolerance capabilities are often required in array processors: APES allows to study fault-tolerant array architectures and methodologies by simulating the behavior of the system when faults occur: the kind and the distribution of faults can be defined by the designer. A diagnostic tool is integrated in APES to evaluate the fault-detection and error-correction capabilities of the system under observation. Another tool allows to perform and evaluate the array reconfiguration after fault occurrence by adopting a user-defined strategy.

1. INTRODUCTION

Different architectures have been proposed to satisfy throughput requirements of particular applications, e.g. signal and image processing, computer simulation of VLSI devices and matrix operations. *Array processors* are an effective solution for massive computation. They consist of a number (usually very high) of processing elements (PEs) connected by a regular interconnection network [6] [7]. Recent advances in VLSI and WSI integration technologies allow their implementation at reasonable costs, making these architectures more and more attractive to solve a wide class of high-computing problems.

A simple, fast and effective CAD environment is thus necessary: designers need in fact to verify the correctness of their algorithms (once mapped onto the array [6] [7] [14]) by means of a detailed evaluation of their behavior before proceeding towards the logic and physical design.

Considerations on production yield [5] [9] and on the kind of critical applications to which such architectures are most often devoted lead the designer to introduce fault-tolerant capabilities in the architecture to allow both *end-of-production* restructuring [8] [12] and *run time* fault tolerance [2] [10]. Such additional features may be achieved by means of data coding [2] [11] or by physical redundancy [4] [13]: in any case, the designer needs to verify the efficiency of the adopted technique and, possibly, to compare different solutions by examining (at least at a first approximation) computational delay, throughput, overhead in silicon area, fault latency, error detection probability, survival probability and so on.

After these phases, further design steps and simulations are required at lower levels, from logic to electrical levels and to layout verification. Powerful traditional programs are already available to perform

these activities: problems are in the definition of the interfaces among them and in managing the whole project without an integrated environment.

All previous characteristics are at present hardly available in a single CAD tool; in particular no simple, effective design support has never been implemented for the first, more difficult and delicate phase: behavioral design and evaluation of array processors.

In this paper we present a new integrated CAD environment for design, simulation and evaluation of array processors: **APES**, **A**rray **P**erformance **E**valuation **S**ystem. After a brief description of the overall architecture of this package, we get into detail of some subsystems: in particular we will deal with the behavioral description of the array (textual and graphic data entry), the simulation engine, the definition of possible fault models and their injection, the evaluation of fault-tolerance capabilities (error detection, error correction, restructuration/reconfiguration policies).

2. APES' ARCHITECTURE

APES consists in a set of subsystem devoted to perform specialized subtasks: correct interaction among them is assured by a *shell* which monitors and coordinates the behavior of the whole system (fig. 1).

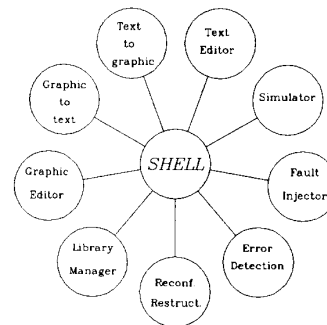


Fig. 1 - APES' architecture

The *shell* of APES presents a simple and homogeneous guided interaction with the environment for behavioral design, verification and evaluation. The user is aided in its work by a friendly graphic interface based upon context sensitive pull-down menus and multiple windows: as soon as a selection is entered through the keyboard or the mouse, currently available alternatives and options are presented.

The shell provides correct synchronization and coordination among the different user activities by properly running the system tasks (the editor, the simulator, the fault injector, the reconfiguration manager, and so on). Such tasks are executed by concurrent processes spawned and controlled by the APES shell.

This module simplifies designer's work: for example, the user can define a set of windows on the screen in which the system will run the different tasks. Multiple windows are usually welcome by designers since they allow:

- to execute parallel tasks without leaving one to begin another,
- to show immediately interactions among circuits and dependencies on their structures and on input data,
- to adjust interactively the behavior of each device in order to match the desired performances or behavior.

Each different design activity at the behavioral level is managed by the proper subsystem. Presently, the following subsystem have been defined and developed:

- 1) The *graphic editor* allows a simple and fast design of the array architecture. Both PEs' structure and interconnection network can be defined by using a set of predefined or user-defined primitives.
- 2) The *graphic-to-text translator* transforms the graphic data entry in a textual form which can be easily exported out of APES into other CAD tools for further design stages till layout drawing. The textual representation of the array is written in VHDL [1], a well-known hardware description language.
- 3) The *text-to-graphic translator* translates the textual description of the array into its graphic form.
- 4) The *library manager* allows to store and retrieve definition of functions, PEs and interconnections. These libraries may be a powerful aid in designing a new array starting from previous tested experiences.
- 5) The *simulation engine* performs the simulation of the behavior of the array by using the definitions of PEs and interconnections and the input data supplied by the user. This task computes the output results of the whole array, of each PE and of each functional unit; timing performances are also evaluated. Functions are available for tracing, for inserting break-points, for step-by-step execution and for selecting data that must be displayed.
- 6) The *fault injector* introduces faults in the basic array architecture according with the fault model adopted by the user. The effect of faults onto the computation can be observed by examining the output of simulator.
- 7) The *diagnostic evaluator* allows automatic checks on outputs of each functional unit and production of statistics for fault-tolerant architectures under given fault distributions.
- 8) The *restructuration/reconfiguration manager* applies to the architecture one of the user-defined restructuration/reconfiguration algorithms (e.g. [4] [10] [13]) and evaluates its characteristics with respect to a set of given figures of merit.

3. BEHAVIORAL DESCRIPTION LANGUAGE AND GRAPHIC EDITOR

In APES the architecture of an array processor and the structure of the processing elements can be defined in two different ways: by a textual definition or by a graphic schematic entry. The latter is easy to understand and use due to its interactivity and to the common aspects with the physical layout, while textual definition can be manipulated with more freedom and can be simply translated into further design phases.

3.1 Behavioral description language

Textual definition is based upon a subset of VHDL [1], a well known hardware description language developed by the U.S. Department of Defence. In fact, in the nearest future this language will become a standard reference for all design environments. At the moment, we have implemented a large subset of VHDL declarations and instructions for

behavioral description of PEs and interconnections: further enhancements will cover the circuit level for integrating subsequent design steps till layout design and verification into APES. In the sequel we discuss the limitations and the use of VHDL for describing our particular application.

VHDL describes a circuit at behavioral level by using a procedural approach, which is very similar to Pascal programming language. In particular, it consists in a *declarative* part and in a *configuration* part. The declarative part defines the elements involved in the definition of the circuit, while configuration instantiates and connects the previously declared elements.

Each circuit can be built starting from *design entities*, i.e. from abstract subsystems with a predefined behavioral interface which defines both the input/output communication channels and the internal functionalities. For array processors we have implemented only three kind of input/output channels, instead of the original five foreseen by VHDL: the *In*, *Out* and *Inout* gates. As body declaration of design entities is concerned, we have implemented the behavioral model according with the standard syntax; structural and mixed descriptions are not presently supported.

APES allows to design also *procedures* and *functions* to simplify the description of large complex circuits and to split into modules the overall system. *Packages* supply the syntactical structure to represent collections of procedures and functions and to build design libraries for basic computing elements.

The concept of data type has been defined in VHDL to characterize the signals in the circuit: this can be useful to perform design verifications and simulations. We implemented *integer* and *enumerative* (*bit*, *boolean*, *character*, *severity level*) types, without decreasing the power of the overall system: physical and floating-point types will be considered in future releases. Numeric subtypes, as *natural* and *positive* types, and composed types, as *string* and *bit vector*, are also supported. As in VHDL, definitions of predefined types are collected in the package STANDARD, which can be directly referenced avoiding the use of the *with construct*.

Sequential circuits are described by means of *static* variables, i.e. variables which must store the state of the system. The syntax and the semantic of the definition of sequential machines has been completely implemented in APES.

The behavior of any circuit is defined by *assignment* operations in the *configuration body* of procedures, functions and design entities. The classes of operators allowed are, in increasing order of priority:

- *logic operators* (and, or, nand, nor, xor),
- *relational operators* (=, /, <, <=, >=, >),
- *arithmetic operators of class 4* (+, -, &),
- *arithmetic operators of class 3* (*, /),
- *arithmetic operators of class 2* (*, /, mod, rem),
- *arithmetic operators of class 1* (**, abs) and the *boolean operator not*.

To introduce the concept of time in PE and array definition (essential figure for timing characterization and simulation of circuits) we implemented the physical type *time* with related units (*fs*, *ps*, *ns*, *us*, *ms*, *s*, *min*, *hr*) and functions *tmin* and *tmax*. Delays in signal propagation can be managed by including the standard clause *after* in VHDL statements.

3.2 Graphic schematic editor

An alternative way to introduce PEs' description and array architecture is given by the graphic editor: this is an interactive tool which allows to draw the schematic view of PEs' functional blocks, to connect such blocks within PEs and to design the interconnection network.

The user interface has been implemented with particular accuracy for a fast and simple learning and use. It is based upon pull-down menus and windowing. Dynamic presentation of the available commands and options allows the user to manage effectively his activities by concentrating his attention on design problems, not on syntactical problems related to textual description or to an old-style command language for the graphic editor. On the other hand, a guided choice of commands

avoids a large number of typing errors improving design speed and correctness.

Like all modern interactive systems for CAD environments, better design performances for the user and a more appealing interface has been achieved by supporting an extensive use of simple and user-friendly pointing devices, as mouse and tablet, not only in drawing, but also in operation selection.

The editor allows to create a PE by selecting the basic functions or user-defined subsystems from a menu and by properly connecting the inputs and the outputs of these items. Besides, it is possible to delete, copy or move selected components of an existing drawing. All these operations take a great advantage from the use of efficient pointing devices because elements can be easily selected only by pointing them on the screen. Filing facilities are also available for saving a design or to load a previously-defined drawing.

To simplify circuit design, a set of simple elements (e.g. arithmetic and logic units, buffers, latches, switches, multiplexers, counters) has been introduced in a parametrized standard *cell library*. All elements included in such library are defined in a parametric way: when the user instantiates an element, he must specify the values of characteristic parameters, such as the number of bits in an adder. A system facility can be used to extend the standard library or to build and use other libraries for custom applications.

As soon as PEs have been defined, assembling of the array architecture can be afforded. The graphic editor provides a set of commands to help the user in this phase: in particular, the selection of PEs can be done through an interactive menu, which allows to display and to check the structure of each selected PE before definitely inserting it in the architecture. Array generation can be obtained by means of the suited command specifying the number of elements and the planar disposition. Interconnections are automatically drawn by the system by connecting the output signals to the input signal with the same name. For particular applications, it is also possible to modify this default behavior by specifying connections explicitly.

The array architecture can be modified, saved and reloaded at any time in APES by means of the suited filing facilities. Also the whole array can be inserted in a library for future uses or for documentation purposes.

3.3 Equivalence of graphic and textual descriptions and translation

The concurrent availability of the two previously described data entry methodologies allows the designer to switch from one to the other depending on his personal attitudes or on the particular phase of design he is affording. The dynamic switch between the two views is provided by a procedure which translates the textual representation of an array processor into the corresponding schematic view and vice versa. Obviously, inverse translation cannot introduce hierarchical relationships between simple components of each processing element starting from their schematic entry. In fact it is very difficult to design and implement a program for automatic extraction of functionalities of different subsystems (e.g. of arithmetic units): this is often a hard task also for an expert system.

The extensive use of windowing in human interface of APES allows to open different windows on textual definition and on graphic description by enhancing synergy of contemporaneous use of both kinds of data entries. For example it is possible to view and zoom the schematic of a part of the circuits and, in an other window, observe the corresponding translation in textual form. The user can modify the structure of PEs or the array architecture by working in the window he prefers: all changes will be reported also in the other description.

4. SIMULATION ENGINE

The *simulation engine* performs the simulation of the behavior of the array by using the definitions of PEs and interconnections and the input data supplied by the user.

The user can interact with the simulation engine through a graphic interface based upon pull-down menus. Available commands and options are dynamically displayed as soon as any command is selected.

When the simulator is started, a synoptic view of the whole array processor is shown in a window on the screen. During simulation actual input and output data are displayed on the graphic representation to give a comprehensive view of the system and its behavior. The designer can modify this default view to point out more details than those allowed by the current display area. Commands are available to zoom into a particular zone and to open other windows on different parts of the array architecture. These features are useful to observe the effect of inputs onto computation results in different points of the array.

Data can be displayed in different formats according to user requirements and interests: the default form is the notation corresponding to each data type, e.g. the decimal representation for integer data or the ASCII standard for characters. Binary and hexadecimal formats are available for all data types, even if readability may be considerably reduced by the great amount of information displayed on the screen. Graphic data presentation versus time is also supported as in logic analyzers.

To increase execution speed and the readability of simulation outputs, the amount of displayed information can be reduced. To such purpose, the designer can select the data that must be shown at runtime: other data are computed and stored by the simulation engine for subsequent operations, but they are not presented to the user.

Data concerning simulation session can be stored in a textual format into a file for subsequent use or for comparisons. The designer can choose the file name and the time or set of conditions which will enable or disable the recording process. This file stores a trace of the behavior of the array processor under the given inputs: different information about the state of the array processor and the computation are recorded, e.g. inputs, outputs, timing performances, intermediate results among PEs, values of user-defined conditions under trace.

Input data at the array borders must be supplied by the user of the simulator. Two alternatives are allowed: the user can enter data directly for each input through the keyboard and the pointing device or by using an input data file. Such textual file is divided in sections: each section is time-stamped and contains the values of the inputs at that time. To reduce the size the input data file, inputs whose value is not changed with respect to the time of the previous section can be omitted.

As already said, default options for all previous commands may be modified interactively by the user. To simplify the use of APES and the allow to tailor the simulation environment according to user requirements, to project characteristics and to the characteristic under study, a user-defined sequence of commands can be stored in the startup command file of the simulator. It is executed as soon as the simulation environment has been selected: after the simulator has been set up, it will accept direct commands from the user. Complex or long sequences of commands can be embedded in command files to simplify the user activities: command files are processed by the command interpreter as commands given through the keyboard.

Debug and observation of the system under simulation may be managed by introducing break points as in usual debugging environments for programming languages. Execution can be automatically stopped when a given condition occurs; for example:

- when an input or output data has a given value,
- when an input or output line has a given value (without regarding data typing),
- when input and/or output data satisfy a boolean condition,
- when input and/or output lines satisfy a boolean condition (without considering data typing),
- when a given number of PEs activations has been performed,
- when one step has been completed (step-by-step execution),
- when a given time has been spent,
- when a given input or output value has overcome a maximum num-

ber of occurrences,

- when a boolean condition on inputs and/or outputs has been satisfied for a given number of times.

The simulation engine represents the array architecture by means of different data structures:

- PEs, switches for data routing among PEs, input and output buffers are modeled by using *transformation entities*,
- interconnections are described in the *interconnection matrix*.

Each transformation entity contains the complete behavioral description of the corresponding unit. It consists in input and output data and in the *transfer function*, which produces outputs from the actual state of the entity and the inputs currently stored in the entity itself. Additional storage is provided for local variables of hardware description. The computational delay, introduced by the entity, is also stored to evaluate timing performances.

The interconnection matrix has as many rows and columns as entities in the array. Elements in each row state whether the entity of the corresponding column is reachable from the entity of the row itself: 1 means that the entity is reachable, while 0 means that no connection is provided.

Simulation is performed by setting up the inputs of the array into the input buffers. Outputs of such elements are computed by applying the corresponding transfer function, i.e. by copying inputs into outputs. Data stored in each input buffer are transferred into the entities connected to it. Connections can be deduced from interconnection matrix.

Transfer function of each entity is applied to generate the outputs of the corresponding entity and interconnection matrix is used to move outputs of an entity to the others connected to it. These operations are iteratively performed until the output buffers are loaded. Then the identity transfer function of output buffers moves the final results of array computation onto the external pins at the array borders.

The execution order of transfer functions, i.e. the activation of entities (both PEs and switches), depends on the kind of array under simulation. PEs can compute their functions after the enable signals have been activated. In *wavefront arrays*, the enable signal of a PE is related on the completion of the execution of PEs connected to PE's inputs [3] [7]; while in *systolic arrays*, it depends only on the master clock of the whole array [6]. Enabled PEs must execute in parallel their computation, at least for an external observer. This behavior has been simulated on a sequential computer by using the storage previously described for inputs and outputs in each entity: it decouples the updating order of the inputs of a PE from the outputs of PEs connecting to them. This implies that any execution order of enabled PEs is allowed if result transfer is performed after sequential computation of all enabled PEs at the time initially considered.

5. FAULT INJECTOR

When fault tolerance is a basic requirement of the array processor, the designer needs to simulate the occurrence of faults and to observe the behavior of the faulty system. This can be useful to identify critical parts and to develop effective techniques for error detection and, possibly, correction.

The *fault injector* has been designed to achieve these goals. It allows to model the hardware failures and to inject them into the array architecture. To model a faulty array we need:

- a model for each possible fault of any PE or switch, i.e. the modified behavior of the faulty entity,
- the fault distribution both in time and in space.

By using these information, the fault injector could generate the fault distribution on the physical array at each moment simulating the life-time degradation of the hardware components.

In APES the array designer is allowed to model any kind of fault or defect. This great flexibility is due to the fact that they can be modeled at behavioral level simply by changing the transfer function of the related entity. In other words, the user must supply different descriptions for each entity: the basic fault-free description and a set of other

descriptions each related to a particular fault occurrence. Considered faults and corresponding behaviors may be linked in APES through a direct interaction with the user or a text command file.

When behavioral simulation of the system is running, the fault injection is executed in parallel with the simulation itself. Whenever a fault arises in the array, the behavior of the newly faulty entity changes: to simulate this event, the injector substitutes the transfer function previously associated to that entity with the transfer function corresponding to the fault just occurred. The simulation engine continues its work without modifications, even if a transfer function has been changed in the array. Fault occurrence is in fact perfectly transparent to the simulation engine which produces output results of each entity by applying the transfer function to inputs and moves data between adjacent entities.

Space and time distribution of each kind of fault can be obtained from the probability distribution of fault occurrence, which is in general a function of time and place in the array. The description of these probabilities and the evaluation of the actual fault distributions is a complex task not yet supported by APES. At the moment, the list of faulty entities in the array at each time cannot be automatically generated in APES, but it can be prepared off-line and passed to the injector through a text file.

To observe particular fault configurations, the user is allowed also to inject the faults interactively. The list of faults can be stored in a text file, like that of the previous case.

The most interesting feature offered by the fault injector is the on-line simulation of the life-time degradation of circuits' behavior: the effects of faults (if not masked) may be directly observed as errors into the output results.

6. DIAGNOSTIC EVALUATOR

The *diagnostic evaluator* is the second tool provided by APES for evaluating the fault-tolerance capabilities of an array processor. In particular, it allows to study the possibilities of error detection and correction in a given architecture and to evaluate automatically some characteristic parameters.

This module is able to detect the errors due to fault occurrences by comparing results produced by the faulty array with ones of the corresponding fault-free array. The user must select the outputs that must be automatically compared through an interactive menu or a text command file.

If the array architecture foresees error-detection capabilities, the user can evaluate them by using the statistic support of the diagnostic evaluator. For each given fault distribution, *detection capability* is given by the ratio between the number of errors detected by the fault-tolerant array under test and the total number of faults occurred when a random inputs have been provided. The diagnostic evaluator ends computation for a given fault configuration when that ratio has become reasonably stable, i.e. when it does not change more than a user-definable percentage. Final statistics are the average value and the variance of detection capability versus the number of faults.

If error correcting circuits are introduced in the architecture, the diagnostic evaluator allows to evaluate the performances of the adopted technique. As for error detection, the outputs of the fault-tolerant array with error correction capabilities are compared with the outputs of the corresponding fault-free array. *Correction capability* is given by the ratio between the number of correct modifications of erroneous outputs and the total number of faults, when random inputs are supplied. The average value and the variance of correction capability versus the number of faults is computed.

The third parameter that can be evaluated is *fault latency*, i.e. the time that passes between fault occurrence and detection. This figure is obtained by observing and recording the time of these events for a given fault distribution. The average value and the variance of fault latency are computed versus the number of faults in the array.

At present, design of this module has been completed, while coding and testing of the prototype version are at the final stage.

7. RESTRUCTURING/RECONFIGURATION MANAGER

This module has been developed to assist the designer of array processors in the study and evaluation of architectures and algorithms for restructuring and reconfiguration. These techniques have been proposed in literature to improve production yield and to allow system survival after fault occurrences [4] [10] [13]. The basic concept exploited by such techniques is that of *redundancy*: faulty units are excluded from the computation and their task is demanded to usually un-used spare units (*space redundancy*) or to already active units which double their work (*time redundancy*). In any case additional circuits have to be introduced into the array and interconnections among PEs must be redefined to include all possibilities due to redundancy.

Physical interconnection matrix holds all possible interconnections among entities in the array processor: for example, the output of a switch has physical connections to more than one PE or switch to allow alternative data routing for fault tolerance. The *interconnection matrix*, as defined in section 4, gives only the active connections due to positions of switches or multiplexers.

Whichever fault-tolerant technique has been adopted by the user, the resulting architecture must be simulated and evaluated to find the optimal matching between fault-tolerance requirements and implementation costs in terms of area occupation and computation delay. In particular, the designer needs to verify the survival capabilities of the redundant system. Since this can be only a statistical figure, it is necessary to run the user-defined restructuring/reconfiguration algorithm upon a number of defective systems. By properly choosing the study cases, it is possible to produce meaningful statistics: a user-friendly approach consists in preparing these cases by using the fault injector described above and by saving distributions of faults in a textual file.

A restructuring/reconfiguration algorithm is a user-defined executable program written in a programming language available on the computer running APES. Such program must accept the faulty configuration of entities (i.e. the list of faulty entities produced by the fault injector) and the physical interconnection matrix. By means of these information, the algorithm computes, if it is possible, the new connections between entities which allow the system survival: the output of restructuring/reconfiguration program is the new interconnection matrix for the simulation engine.

The restructuring/reconfiguration manager applies the user-defined algorithm to the study cases of the array under test by spawning the user-defined program as an auxiliary process of APES. Different statistics are evaluated on the results produced by the restructuring/reconfiguration program and by running some functions of the simulation engine. The more interesting figure which can be obtained is the probability of survival (vs. number of faults) of the array. Other evaluation parameters are strictly related to the performances of the reconfigured system, as the average and maximum interconnection length, the average and maximum computation delay, the clustering of working PEs. Such statistics may be displayed, plotted or stored in a textual file for each array architecture according with the number of faults.

8. CONCLUDING REMARKS

In this paper the APES system for design and evaluation of VLSI or WSI array processors has been presented. Features concerning data entry (via a graphic editor or a hardware description language), the

simulation engine, the fault injector, the diagnostic evaluator and the restructuring/reconfiguration manager have been discussed.

Further capabilities can be integrated into the system considered here to allow an easy management of the whole array design, from the behavioral to the electric level and to final production masks. For example, traditional well-known packages for subsequent steps from logic design to layout verification can be included in the shell of APES. Besides, suited interfaces can translate the input/output format of each of these programs to show an homogeneous, interactive, friendly interface to the user. On the other hand, a data base can be inserted in APES to trace the development of the design phases and to store libraries of tested VLSI subsystem.

Implementation of the shell and the behavioral subsystems of APES is currently at the final stage: the prototypal version consists in about 10,000 lines of source code written in C language under Unix 4.2Bsd operating system. An other experimental release is being developed for DOS operating system on personal computers based upon Intel 386 microprocessor.

REFERENCES

- [1] IEEE Design and Test of Computers, Special Issue on VHDL, April 1986
- [2] J.A. Abraham, P. Banerjee, C.Y. Chen, W.K. Fuchs, S.Y. Kuo, A.L. Reddy, "Fault tolerance techniques for systolic arrays", IEEE Comp. Magazine, July 1987
- [3] F. Distanto, M. Sami, "A protocol for asynchronous wavefront-computation arrays", Systolic Arrays, W. Moor Ed., Adam Hilger, 1987
- [4] F. Distanto, R. Negrini, V. Piuri, "Simulated Annealing for defect tolerance in two-dimensional arrays", IFIP Workshop on Wafer Scale Integration, 1987
- [5] P. Franzon, "Yield modeling for fault-tolerant arrays", Systolic Arrays, Adam Hilger Ed., 1987
- [6] S.Y. Kung, K.S. Arun, R.J. Gal-Ezer, D.V. Bhaskar Rao, "Wavefront array processor: language, architecture and applications", IEEE Trans. on Computers, Nov. 1982
- [7] H.T. Kung, S.C. Lo, S.N. Jean, J.N. Kwang, "Wavefront array processors - Concepts and Implementation", IEEE Comp. Magazine, July 1987
- [8] F.T. Leighton, C.E. Leiserson, "Wafer scale integration of systolic arrays", Proc. 23th Symp. Found. Computer Sci., 1982
- [9] R.M. Mangir, A. Avizienis, "Fault tolerant design for VLSI: effect of interconnection requirements on yield improvement of VLSI design", IEEE Trans. on Computers, July 1982
- [10] R. Negrini, M. Sami, R. Stefanelli, "Fault tolerance in supercomputers: problems and solutions for array processors", IEEE Comp. Magazine, Feb. 1986
- [11] V. Piuri, "Fault-tolerant array processors: an approach based upon A*N codes", Proc. ISCAS'88, Helsinki, 1988
- [12] J.I. Raffel, "The RVLSI approach to wafer scale integration", Proc. Workshop on Wafer Scale Integration, 1985
- [13] M. Sami, R. Stefanelli, "Reconfigurable architectures for VLSI processing arrays", Proc. IEEE, May 1986
- [14] L. Snyder, "Introduction to the configurable, highly parallel computer", IEEE Comp. Magazine, Jan. 1982