

Bit-Modular Defect/Fault-Tolerant Convolvers

Luigi Dadda, Vincenzo Piuri

Dipartimento di Elettronica e Informazione, Politecnico di Milano

P.za Leonardo da Vinci 32, I-20133 Milano, Italy

phone +39-2-2399.3510, +39-2-2399.3606, fax +39 2 2399.3411

email: dadda@elet.polimi.it, piuri@elet.polimi.it

Abstract

Design of a family of high-throughput modular convolvers is discussed, with particular reference to the defect and fault tolerance issues. The proposed architecture is based on parallel computation for the individual operands' bits with final merging of the partial results. Different degrees of defect/fault tolerance are considered for different production and operational environments. Modularity is exploited to support also functional adaptability.

1. Introduction

Convolution is one of the most used operations in digital signal/image processing [1]. The use of dedicated VLSI/WSI devices (*convolvers*) is mandatory in applications having hard real-time constraints on latency and throughput (e.g., system control and aerospace applications). Advances in integration technologies allow physical realizability of large dedicated convolvers, but defect probability may lead to a poor or even unacceptable yield. Suited techniques [2] must therefore be adopted for yield enhancement. Mission-critical applications impose to certify result correctness: besides, with real-time constraints, fault tolerance must be provided concurrently to the nominal operation. High reliability needs also concurrent error correction, while high availability implies minimization of the repair time, possibly with degraded functional capabilities.

Several convolver architectures were presented in the literature (e.g., [3, 4]), dealing with specific aspects of the implementation constraints (e.g., circuit complexity, latency, throughput), but they usually do not consider the regularity of the structure, the defect/fault-tolerance issues, and the functional adaptability (defect/fault tolerance is obtained by overposing some techniques on the basic structure).

In this paper, we present modular high-throughput convolvers with contemporaneous capabilities of functional adaptability, defect/fault tolerance, and graceful degradation. These goals are achieved by partitioning the input samples at bit level and by using separate subconvolvers each dealing with a different samples' bit. In [5], the preliminary study of the arithmetic operations was presented and the possible use of fault tolerance techniques was only suggested. In the present paper, detailed architectures implementing this strategy are discussed and evaluated. A wide range of defect/fault tolerance strategies and techniques are applied at different architectural levels to protect the whole system by exploiting the intrinsic modularity: both space redundancy based on data coding and time redundancy are considered and evaluated with respect to performance and circuit complexity. Design guidelines are provided to identify the best architectural approach as trade-off between these characteristics. Convolver adaptation to different computational requirements as well as graceful degradation are discussed by exploiting the architectural supports used for defect/fault tolerance.

2. The basic schemes of the bit-modular convolver

The computation of the convolution outputs $Y_j = C(W^N; X_j) = \sum_{i=0..N-1} W_i X_{j-i}$ can be partitioned in disjoint subsets of operations on the individual samples' bits, being W^N the N -cardinality convolution weights' vector and X_j the samples' vector at time j . Each n -bit sample X_j is partitioned in the individual bits $x_{j,i}$; all sample's bits are presented in parallel to the convolver, while the N samples are presented serially. The nominal convolution becomes:

$$Y_j = \sum_{i=0..N-1} W_i (\sum_{k=0..n-1} x_{j-i,k} 2^k) = \sum_{k=0..n-1} 2^k (\sum_{i=0..N-1} W_i x_{j-i,k}) = \sum_{k=0..n-1} 2^k C(W^N; x^{j,k})$$

The convolution with the same weights is applied to each samples' bit (bit-convolutions) and the partial results are merged at the proper weighted position. All bit-convolutions are computed in parallel by different circuits (called *bit-convolvers*), each sample's bit being used in the corresponding bit-convolution with the same weights. The name *bit-modular* refers to the modular convolver architecture composed by identical modules each dealing with one bit.

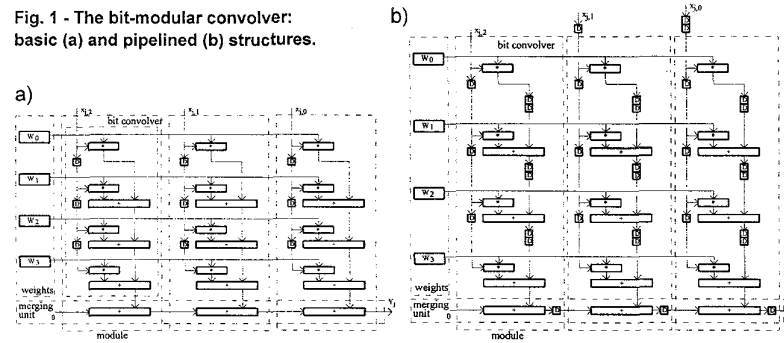
The basic structure can be obtained by mapping all arithmetic operations onto dedicated units, as shown in Fig. 1a. Generation of each term $W_i x_{j-i,k}$ in the weighted summation $C(W^N; x^{j,k})$ within the individual bit-convolver is a simplified multiplication in which the multiplier is one-bit long, and is implemented by the partial product generator of a traditional multiplier. Convolution weights are stored in registers and distributed to all bit-convolvers. The bit-convolutions $C(W^N; x^{j,k})$ are available at the same time: they are added by the merging unit at the proper weight 2^k . Bit-convolutions are generated starting from the one of the most-significant sample bit: they are added in this order so that reconfiguration can be supported efficiently both for defect/fault tolerance and graceful degradation. The linear adder in the merging unit can be replaced by a tree of two-input adders (having logarithmic instead of linear latency), but modularity is greatly reduced so as to prevent reconfiguration.

In traditional convolvers, the individual multiplications are obtained by adding the partial products corresponding to the same convolution's weight (i.e., by adding the terms horizontally in Fig. 1a, instead of vertically). The final convolution is generated by adding the multiplications' results. The circuit complexity both of traditional and bit convolvers are identical. Parallelism is better exploited in bit-convolvers since usually the number N of convolution's weights is smaller than the number n of sample's bits. Thus, a smaller number of additions is performed in each bit-convolver, being higher the total number of bit-convolvers. Conversely, the bit-convolver merging adders limit the convolver speed since they are longer than corresponding adders in the traditional convolver.

Throughput is greatly increased by pipelining (see Fig. 1b), despite a circuit complexity increase. The input operands of the merging unit must be skewed in time (all bits of the same operand are presented in parallel). This is obtained by using a set of registers delaying the bit-convolvers outputs or by skewing the convolver's samples. No additional circuit is required if the convolver input bits are presented in the skewed form since the bit-convolvers generate bit-parallel outputs which are skewed one with respect to the other bit convolver output.

Independently from the implementation technology, we evaluate the circuit complexity, the latency and the throughput by using the traditional gate count to provide architectural-level design guidelines. Evaluations are given in Fig. 7 by considering ripple-carry, carry-look-ahead (CLA), and carry-save adders. Among the non pipelined structures, CLA adders provide the highest throughput and the lowest latency, despite a higher circuit complexity; among the pipelined architectures, carry-save adders maximize the throughput, while CLA adders gives the smallest latency, being the complexity about equivalent. In the carry-save solutions, carry propagation is in the merging unit final stage only, but modularity is reduced.

Fig. 1 - The bit-modular convolver:
basic (a) and pipelined (b) structures.



3. The defect tolerance issue

High regularity and modularity of the architectures presented in Section 2 are well suited to support strategies for defect tolerance. To obtain an acceptable yield for the complete convolver, architectural supports and techniques must be introduced to exclude the defective components from the active computation. Hardware configuration of the interconnections between units [2] is achieved by segmenting the interconnection paths and introducing configurable interconnections between segments. *Permanent configuration* is realized through laser cutting, electrical programming of connecting fuses, or interconnection deposition; *dynamic reconfiguration* through programmable switches allows a more flexible approach.

To apply these techniques, spare components are required to replace the defective ones. Configuration is successful if the convolver can be created by connecting fault-free components. For some distributions of defects, even if the nominal convolver cannot be configured, the fault-free components may be connected to create a convolver with reduced functional capabilities with respect to the nominal ones, i.e., with a smaller number of weights in the kernel or with a smaller number of samples' bits.

Confinement of the defective components and configuration of the fault-free system are obtained by applying the following operations: defect detection, identification of the faulty components, identification of a feasible configuration of the fault-free components, configuration of the system. Detection and location may be based on any traditional technique for end-of-production testing and diagnosis, possibly by introducing the necessary hardware supports in the nominal architecture of the convolver. The feasible configuration is identified according to the capabilities of the interconnection structure and of the reconfiguration technology and applied.

Fig. 2 shows the architecture with switched-bus interconnections. As usual, interconnection switches and buses are assumed fault-free [6] and constitute the hard core of the architectures. The skewing registers for the pipelined architecture in Fig. 2b are hard core too; the scheme in Fig. 2c protects some of these registers by introducing few additional switches and registers. The basic module for reconfiguration is composed by one bit-convolver and the corresponding adder stage in the merging unit. A defective module is excluded from the computation by configuring the interconnections between modules so that it is by-passed (e.g., by using the Diogene's approach [6]). Unused spares remain disconnected from the active computation.

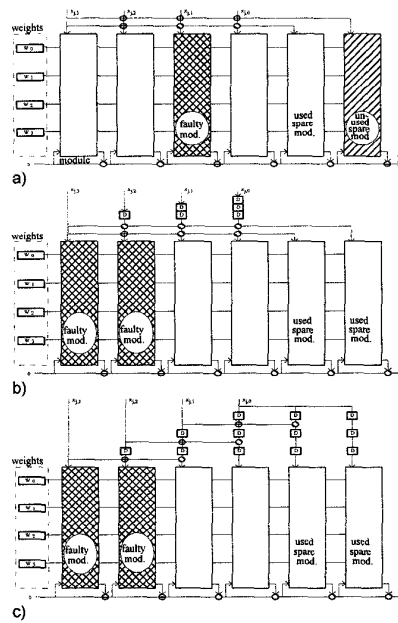


Fig. 2 - The switched-bus architecture for defect tolerance: non-pipelined (a), pipelined (b), and skewing (c) schemes.

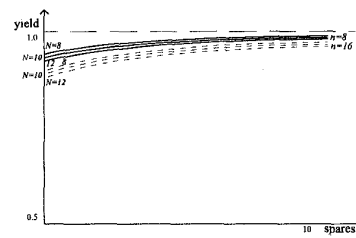


Fig. 3 - Yield.

With respect to the basic architectures, the latency increase and the throughput decrease are null or not relevant, according with the reconfiguration technology adopted (no delay by using laser cutting, fuses, or deposition; a very limited technology-dependent delay for switches along the switched buses).

The additional circuit complexity is due to the spares and the interconnection network (e.g., for switched buses, redundant interconnection paths and switches): as shown in Fig. 7, it increases with the number of spare modules. The yield Y is given by $Y = q^{F(n-1)} \sum_{f=0, \dots, F} \binom{F}{f} (p)^{n+F-f} (1-p)^f$, being F the number of spare units, p and q the probabilities that the module and the interconnection switches are fault free, respectively (see Fig. 3).

Configurability at the level of the individual multiplier/adder within each of the modules considered above is not efficient from the points of view both of circuit complexity and yield. The complexity of the n -bit interconnections for samples and weights is too relevant with respect to the protected circuits (one n -bit adder and - in the case of the pipelined convolver - one n -bit register); with respect to the architectures of Fig. 1, it is about 50% or 26% for the non-pipelined and pipelined cases, respectively. Then, we must add the spare modules, the n -bit switched buses within the modules, and the interconnection structure between modules. As a consequence, the yield greatly decreases since the switches are hard core. Complexity reduction and yield increase can be obtained by grouping the arithmetic units within a module into blocks each dealing with k weights: the circuit complexity of the switches becomes about $25+25/k$ % and $13+13/k$ %.

A tree structure for the merging unit requires a very complex interconnection structure and much more spares to guarantee the same protection. Each tree level is treated as an independent linear array. As many spare adders must be introduced as the tolerated faults; a switched bus routes the outputs of the previous level and another switched bus moves the adders outputs back to the nominal position for the subsequent level. Reconfiguration is performed within the individual level. The complexity of the interconnection structure and the relevance of the hard core prevent the use of this approach.

4. Strategies and techniques for fault tolerance

The basic step of any fault-tolerance strategy is detection of the errors induced by faults; fault location is required to provide system survival through interconnection reconfiguration. When the application is so critical that consistent results has to be guaranteed continuously, error correction must be provided.

Different approaches can be adopted to achieve *error detection*. Off-line approaches can be applied periodically by using traditional testing techniques, as for defect detection. The same architectural supports introduced for defects are used, but the fault absence is certified only at given time intervals and the nominal activities must be suspended.

If each result must be verified immediately, concurrent techniques must be adopted. *Modular redundancy* with output comparison [2] is simple in principle, but hardware redundancy is very high since copies identical to the nominal structure and the checking circuit are required.

In the literature, the use of *time redundancy* was proposed to minimize the increase of circuit complexity [7]. All these techniques repeat the same computation several times by using the system components in a different way for each repetition; then results are compared to detect the errors due to faults. *Recomputation with shifted (rotated) operands* performs the recomputation step by shifting (rotating) the operands for a given number of bits. In *alternating logic*, the nominal operation is applied to the complement of the nominal operands. In *recomputation with duplication with comparison*, the nominal operation is applied in parallel to one half of the operands at a time by using two identical copies of the circuits: carries are propagated from the first to the second part of the computation. Due to the bit-modular nature of our architecture, the most efficient approach from the points of view of circuit and interconnection complexities is the recomputation with rotated operands: operands (weights in this specific case) are splitted in two parts and these suboperands are exchanged during the nominal and the checking phases. The resulting non-pipelined architecture is presented in Fig. 4a. The arithmetic units are modified to support the nominal and the repeated steps. Registers in the merging unit are introduced to hold the output of the first step during generation of the checking result; checking in each module is required to localize the faulty module. Hardware redundancy is very low (about 10%) and decreases linearly as N increases; latency is often practically twice the nominal value and throughput is half the nominal one. Error coverage is very high, also for multiple-bit errors.

Another class of approaches is based on *data coding* [8], in particular AN codes [9] and residue codes [10]. In *AN codes*, the

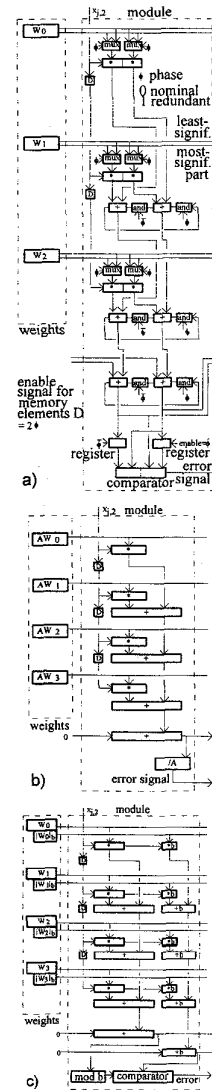


Fig. 4 - Concurrent error detection by RERO (a), AN code (b), and residue code (c).

nominal inputs are coded by a linear transformation and convolution is performed in the coded space: an error is detected whenever results are not divisible by the code generator A (A^2 must be considered if both operands are coded). The resulting module architecture is given in Fig. 4b for the case of the non-pipelined structure. If the convolver input bits are skewed, application of AN codes to weights protects the whole convolver, but the delay units. No encoder is required; the architecture is identical to the nominal one, but a higher number of bits for weights and operations must be considered. If parallel inputs are assumed, AN code for weights does not protect the skewing circuit, but avoids encoding. Arithmetic units and registers have a higher number of bits with respect to the nominal structure as in the case of skewed inputs. If AN code is applied to the convolver inputs, an encoder is required at the convolver inputs; delay units are protected, while weights are not. The architecture has a higher complexity since the additional modules are introduced to deal with the additional input bits of the encoded representation. Application of AN codes to both operands is effective to protect the whole architecture, but the circuit complexity is relevant since both additional modules and larger-operand arithmetic units must be adopted. For single error detection, the $3N$ code is completely effective for the protected components (as in [9]), even if error masking may occur in the presence of multiple faults. The percentage of redundant circuit complexity is about given by $100(2/n+1/N)$ or $100(4/n+1/N)$, when one or two operands are coded, respectively (for $n=8$, it is about 25% or 50%, respectively, while it drops to 12% or 25% for $n=16$). The circuit complexity of the delay units can be practically neglected with respect to the redundant circuit complexity required by coding both operands for their protection; thus, it is convenient - both for circuit complexity and fault probability - to consider the delay units as hard core and by coding only the convolution weights. The latency increase percentage and the throughput decrease percentage are about $200/(n+N)$ or $400/(n+N)$, for one or two coded operands, respectively; for $N=8$ and $n=8$, they are about 25% and 50%, respectively, while they decrease to 9% and 18% for $n=16$.

By using the *residue codes*, the codeword is obtained by cascading the residue of the nominal operand with respect to a given base to the nominal operand itself. Nominal operations are performed on the nominal operands, while module operations (corresponding to the nominal ones) are applied to the residues. Nominal results are coded and compared to the results of module operations: an error is detected whenever such residues are not equal. The non-pipelined module is shown in Fig. 4c. For single errors, error detection at minimum costs is obtained by using the residue code with base 3, but error masking may occur for multiple faults. Hardware redundancy is very high with respect to the nominal architecture since it is given by $100(3/n+3/N)$; it is about 75% and 50% when $N=8$, for $n=8$ or 16, respectively. Latency increase percentage and throughput reduction percentage are given about by $100 \log_3 n/(n+\log_3 n+N)$; for $N=8$ and $n=8$, it is about 16% while it becomes 14% for $n=16$.

To overcome the errors, it is necessary to restore the nominal behavior of the architecture by confining the fault effects, i.e., by excluding the faulty components from the active computation. To achieve this goal, it is necessary to identify the faulty components in the architecture (*localization*). The same techniques adopted for testing can be effectively used, but the nominal convolver operation must be suspended to allow for off-line diagnosis.

When high availability is required, diagnosis and repair time must be minimized. If a concurrent technique has been adopted to detect the presence of errors in modules, information generated by it can be exploited also to localize - concurrently with the nominal operations - the fault in a module; no additional hardware support is required.

Fault confinement can be achieved by *reconfiguring* the interconnection network [5]. The same techniques adopted for end-of-production configuration are used for reconfiguration, if

they have not created the interconnection topology permanently; a switched network can be directly exploited without any additional hardware support, even if it is hard core. Reconfiguration can be applied to the architectures with error detecting capabilities presented above since we have taken care of preserving the regularity and the modularity of the convolver when we introduced the concurrent detection techniques. The new configuration is created on a host computer and, then, downloaded on the convolver by positioning the interconnection switches.

Due to the linear array architecture of the convolver, the reconfiguration algorithm is very simple (Diogene's algorithm) and can be implemented also on-board by using the error signal generated by each component to drive the interconnection switches (Fig. 5).

When consistent outputs must be delivered continuously, the techniques adopted for concurrent detection need to be enhanced to deal with concurrent *error correction*. Since it was shown that modular redundancy with voting [2], time redundancy [7], and data coding [8-10] can be effectively used in regular arithmetic structures, these approaches constitute feasible solutions also for the convolver. Costs are higher than the concurrent error detection due to the higher redundancy required to regenerate the correct expected outputs. Time-redundancy techniques have lower costs in terms of silicon area, while data coding have higher time performance.

5. Functional adaptability and graceful degradation

Modularity and regularity of the previous architectures can also be exploited to support functional adaptability of the convolver, i.e., the capability of configuring the convolver to perform convolutions having different kernel cardinalities or sample bit numbers. As the number of sample bits is concerned, the configurable structure of the bit-convolvers, which is required for defect/fault tolerance, can be directly used to customize the number of sample bits. Configuration implies simply to connect the desired number of fault-free modules in the linear array and by excluding the faulty or non-necessary modules by operating on the interconnection switches.

As kernel cardinalities is concerned, we need to modify the basic convolver structure seen in Section 2. A bit-convolution can be splitted in s bit-subconvolutions by partitioning the complete convolution kernel $\mathbf{W}^{N*}$ in $\lceil N*/N \rceil$ sub-kernels composed by N weights each. The nominal convolution can be rewritten as

$$\begin{aligned} C(\mathbf{W}^{N*}; \mathbf{x}_{j,k}) &= \sum_{i=0..N*-1} \mathbf{W}_i \mathbf{x}_{j-i,k} = \\ &= \sum_{i=0..N-1} \mathbf{W}_i \mathbf{x}_{j-i,k} + \sum_{i=N..2N-1} \mathbf{W}_i \mathbf{x}_{j-i,k} + \dots + \sum_{i=(\lceil N*/N \rceil - 1)N..N*-1} \mathbf{W}_i \mathbf{x}_{j-i,k} \end{aligned}$$

The bit-subconvolutions can be computed in parallel and, then, the merging unit adds all bit-subconvolutions to generate the complete convolution at the convolver output. Latency gain due to parallelism of the bit-subconvolutions is reduced by the higher number of stages in the merging unit.

Also in this case, the configurable architecture for defect/fault tolerance can be exploited to customize the length of the bit-convolvers (i.e. the kernel cardinality); the architecture is given

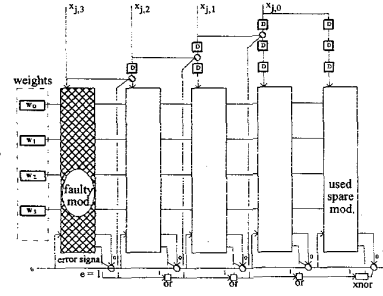


Fig. 5 - Self-reconfiguring architecture.

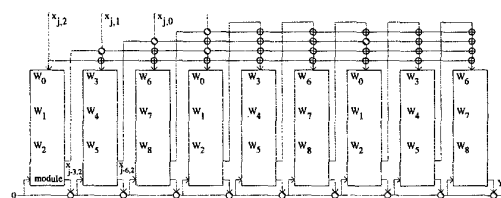


Fig. 6
The modified
architecture
for functional
adaptability.

in Fig. 6. The convolution weights cannot be stored in a unique set of registers for all bit-convolvers as in the basic architecture of Fig. 1, since they need to be changed for the specific interconnection configuration (i.e., for the specific kernel's cardinality). To allow for adaptability, a set of registers is inserted in each bit-convolver. Interconnections are introduced to propagate the input bits from one bit-convolver towards the adjacent one so as to support the increase of kernel's cardinality.

Configuration is performed according with the specific implementation adopted. Laser cutting or programmable fuses can be used both to exclude the defective modules and to define the length of the bit-convolvers. In the case of switched connections, switches are programmed to include only the desired number of fault-free modules.

Functional adaptability can also be used to introduce additional defect/fault tolerance capabilities by means of *functional graceful degradation* of the nominal behavior of the convolver. When there are no enough modules at the end of production to configure all the fault-free bit-convolvers required by the application, we can create a convolver with reduced functional capabilities with respect to the nominal algorithm by decreasing the cardinality of the convolution kernel and, possibly, by changing the weight values. Alternately, we can decrease the precision by truncating the least significant bits of the inputs samples.

In the basic approach discussed in Section 2, the system is declared faulty if there are no enough fault-free modules: with functional graceful degradation, we increase the yield even if precision and computational capabilities are reduced with respect to the nominal algorithm. When subsequent life-time faults occur, we can again apply this approach by gracefully degrading the behavior of the convolver, one step at a time. The system will be assumed faulty only when the computational capability becomes lower than the minimum value that can be accepted for the specific application.

References

- [1] Oppenheim A.V., Schaffer R.W., *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975
- [2] Siewiorek D.P., Swarz R.S., *The Theory and Practice of Reliable System Design*, Digital Press, 1982
- [3] *VLSI Signal Processing*, Cappello P.R. et alii eds., IEEE Books, 1984
- [4] *VLSI and Modern Signal Processing*, Kung S.Y., Whitehouse H.J., Kailath T. eds., Prentice-Hall, Englewood Cliffs, NJ, 1984
- [5] Dadda L., Sami M., High-Speed Parallel Input-Output Bit-Sliced Fault-Tolerant Convolver, *IEEE Proc. Int'l Workshop on defect and fault tolerance in VLSI systems*, Dallas, TX, Nov. 1992
- [6] Negrini R., Sami M.G., Stefanelli R., *Fault-Tolerance Through Reconfiguration of VLSI and WSI Arrays*, MIT Press, 1989
- [7] Johnson B.W., *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, 1989
- [8] Rao T.R.N., *Error Coding for Arithmetic Processors*, Academic Press, NY, 1974
- [9] Piuri V., Fault-tolerant array processors: an approach based upon AN codes, *IEEE Proc. Int. Conf. on Circuits and Systems (ISCAS88)*, Helsinki, Finland, pp. 199-203, June 1988
- [10] Piuri V., Fault-Tolerant Systolic Arrays: an Approach Based upon Residue Arithmetic, *IEEE Proc. Int. Symp. on Computer Arithmetic (ARITH8)*, Como, Italy, pp. 230-238, May 1987

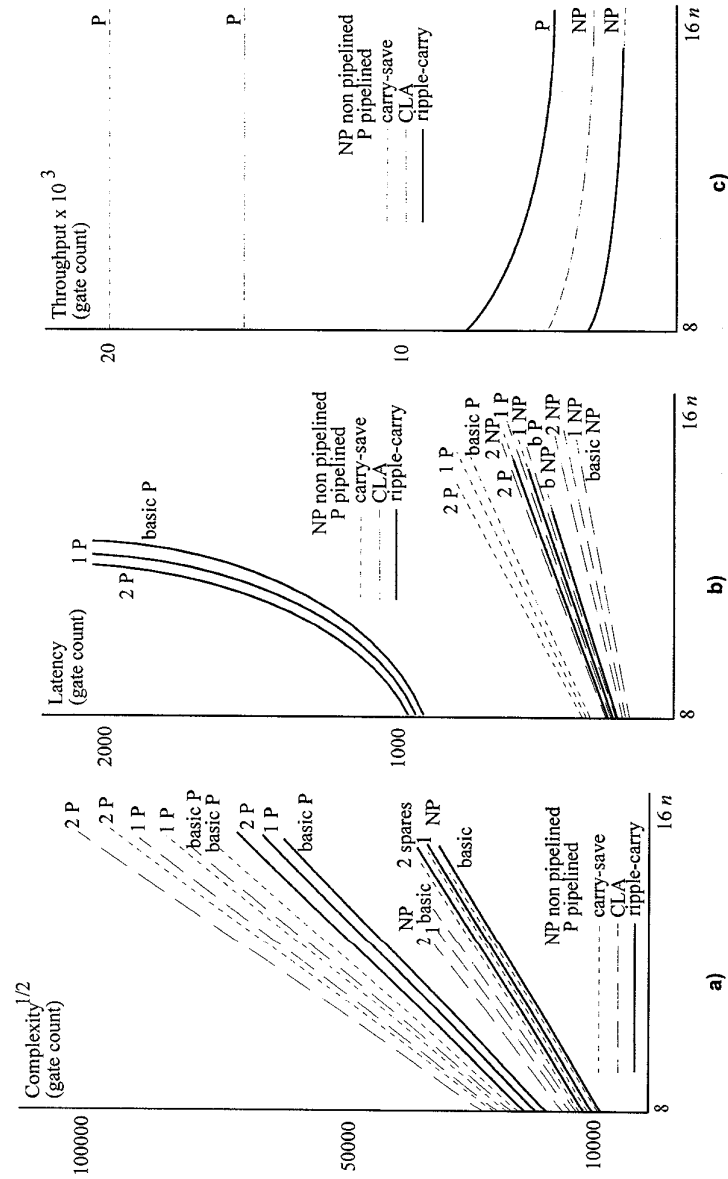


Fig. 7 - Evaluations (for $N=8$): circuit complexity (a), latency (b), and throughput (c).