

A FAST PIPELINED COMPLEX MULTIPLIER: THE FAULT TOLERANCE ISSUES

Luca Breveglieri Vincenzo Piuri Donatella Sciuto

Dip. di Elettronica - Politecnico di Milano
P.zza Leonardo Da Vinci n° 32 - 20133 Milano - Italy
ph: +39 (0)2 2399 3405 fax: +39 (0)2 2399 3411
e-mail: breveglieri/piuri/sciuto@ipmel2.elet.polimi.it

Abstract

A comprehensive discussion of a dedicated device for serial complex multiplication is presented, covering architectural, reliability and fault tolerance properties. The pipelined architecture is briefly described. It is optimized w.r.t. several figure of merits: clock rate, external pipelining and pipeline filling degree. Testability features are analyzed under functional fault models by means of graph-theoretic methods, showing full testability of the device. Error detection is introduced by means of arithmetic codes and the tradeoff between error detection and cost is evaluated. Eventually on-line reconfiguration is introduced through the Diogenes approach and the tradeoff between fault tolerance and cost is also discussed. Discussion are based on analytic interpolation, software simulation and the evaluation of prototypal layouts in CMOS technology.

1 Introduction

Considerable interest has been given in recent years to the definition of serial input multipliers, which are well-suited to VLSI/WSI implementations for fast massive computations, as required by signal and image processing applications. In fact, even though serial multipliers are considered apparently slower than parallel schemes, they are characterized by a reduced number of input and output pins, a simplified internal interconnection structure, hence a high clock rate, a high throughput, a reduced silicon area and an easy testability.

Several serial-input serial-output multipliers have been presented in the literature [1,2,3]. In particular, in [4] a new pipelined architecture has been proposed, characterized by a very simple logic scheme with a latency of $n - 1$ clock cycles, where n is the number of bits used to represent the factors, and a clock period upper bounded only by the propagation delay of a single full adder. Moreover, consecutive operands need not be separated by wait intervals, i.e. the structure reaches full external pipelining. Partially based on this scheme, a new approach in designing serial complex multipliers has been derived and presented in [5]. In such multiplier the real and imaginary parts of the operands are represented in full-fractional, two's complement notation. This architecture is optimized by compacting the real products into which the complex computation is decomposed. Overlapping and compression of these operations allow to optimize computation time and silicon area. The resulting architecture is a semisystolic array of bit-slices.

This paper presents the basic architecture of such serial complex multiplier and mainly details its testability, diagnosability and fault tolerance features. All these properties are evaluated with reference to analytical derivation, software simulation and prototypal implementations of the involved devices.

The multiplier is characterized by an easy testability, allowing to achieve a complete fault coverage with respect to the single stuck-at fault model in a time linear with respect to the number of bit-slices of the multiplier, without any additional gate and/or signal to increase testability. Furthermore, to achieve on-line fault tolerance, on-line error detection has been introduced by means of data coding, in order to limit the area required for error detection and to reduce fault latency. An arithmetic code, in this case a $3N$ code, has been chosen and the encoder and decoder circuits necessary for such error detection technique have been added to the basic multiplier. These circuits, together with the evaluation of the added area and the fault coverage, will be discussed in section 2. Then the problem of the localization and the reconfiguration of faulty elements is considered; in this case a spatial redundancy approach is introduced. It is possible either to adopt an off-line reconfiguration procedure or design an on-line self-reconfiguring multiplier. Self-reconfiguration can be achieved by duplication of the bit-slices, comparison of results to localize the fault and then by addition of a reconfiguration circuit to replace the faulty bit-slice with a spare fault-free one. Comparisons between nominal, self-detecting and self-reconfiguring multipliers are discussed throughout the paper.

2 Architecture of the complex multiplier

The product of two complex numbers can be reduced to four real products and two sums/subtractions. In fact, denoting the two operands $x = a + ib$ and $y = c + id$, the product is given by $z = x * y = (a + ib) * (c + id) = (ac - bd) + i(ad + bc)$.

The multiplier here proposed works with bit-serial operands and result. The real and imaginary parts of the operands are supplied to the multiplier simultaneously on separate input lines. Similarly, the real and imaginary parts of the result are output serially from two separate output lines. Both inputs and outputs are represented in full-fractional two's complement LSB-first arithmetic. For instance, $a = -a_n 2^{-1} + \sum_{i=1}^{n-1} a_{n-i} 2^{-(1+i)}$, that is $0.5 < a \leq -0.5$, and so on for $b, c, \dots$, hence any product ab still ranges over the same interval $(0.5, -0.5]$. Both the real and the imaginary parts of the factors are two's complement integers, represented over $n \geq 1$ bits. Both the real and the imaginary parts of the product are two's complement integers, represented over $2n$ bits. Hence, no overflow problems in the multiplication need be taken into account.

The architecture computes the serial/parallel to serial algorithm, since it is shown in literature [1] that such approach represents the simplest and most efficient way to compute a product. Such algorithm, however, requires that the first operand is presented in parallel while the second one is inserted serially; a parallel to serial conversion is performed by the architecture, but without additional costs.

The basic idea in the design of the multiplier derives from the following observation:

- According to the definition, the real part of the complex product, $ac - bd$, is computed by subtracting the two real products ac and bd . Each one of these, say ac , is in turn computed by shifting and adding the rows $a_{n-i}c2^{-(1+i)}$ ($0 \leq i \leq n-1$) of the partial product real matrix ac .
- Instead of computing separately the partial product matrices ac and bd , adding first their rows and then subtracting the two products ac and bd , one can merge the summation of the rows with the subtraction of the products. The operation performed is now: $(a_{n-i}c - b_{n-i}d)2^{-(1+i)}$ ($0 \leq i \leq n-1$).

The same can be done for the imaginary part of the result, $ad + bc$; it is even easier, as all operations are additions: $(a_{n-i}d + b_{n-i}c)2^{-(1+i)}$ ($0 \leq i \leq n-1$). By compression of such subtractions and sums into a single, though more complex, operation, high performances can be achieved. Figure 1a shows the detailed structure of the multiplier, composed of three sections. The real and the imaginary parts of the factor $a + ib$ are stored in

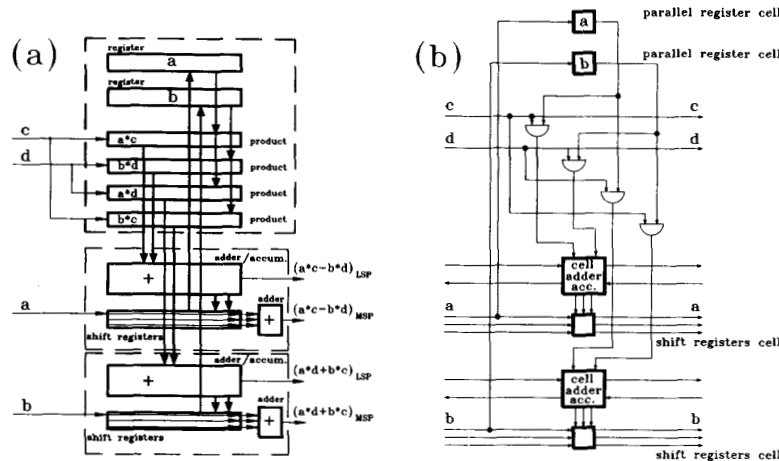


Figure 1: (a) The complex multiplier. (b) Bit-slice of the multiplier.

advance in parallel form. This means that the input of the factor $a + ib$ must precede n clock periods the input of the factor $c + id$. When the factor $c + id$ is input, the first section generates the partial product matrices ac , bd , ad and bc , row by row, while the other two sections compute the real and imaginary parts of the result. Figure 2 gives the complete time diagram of the data flow; it shows that the overlapping of output delivering and input acquisition can be performed without introducing idle clock cycles between the operands (i.e. full external pipelining is obtained), thus fully exploiting the structure without wasting computation time.

The adder sections of the multiplier compute the LSP (Least Significant Part) and the MSP (Most Significant Part) of the real and imaginary parts of the result. LSP's are computed and output by two pipelined adder/accumulators, while the factor $c + id$ is input. At the end of the introduction of $c + id$, the contents of the adder/accumulators, which represent the MSP's of the product, are transferred in parallel into shift register (2 or 3 register per each MSP, depending on the implementation), where they are shifted and added by a serial adder placed at the outputs, during the subsequent n clock periods. Thus, a full precision result is computed, though its output is overlapped to the introduction of the new factors and to the output of the new product, as figure 2 shows.

The two sets of shift registers are progressively emptied as the computation goes on, hence they are used to perform a serial to parallel conversion of the factor $a + ib$, which, as already mentioned, need be supplied in advance. At the end of the introduction of the factor $c + id$, $a + ib$ is transferred in parallel to the product generators.

The whole multiplier is a semisystolic array of n bit slices; figure 1b shows a bit-slice. Each bit-slice computes one bit for each one of the four real partial product matrices ac , bd , ad and bc , one bit for each one of the two accumulated partial sums and one bit for each one of the two sets of shift registers.

Three possible logic solutions for the cells of the adder/accumulator arrays are shown in figure 3. They are derived by using distinct schemes for the propagation of the carries generated by the addition (subtraction) of the partial products. Prototypal implementations have shown that solution 3b3 is the optimal one, hence it is the only one to be used

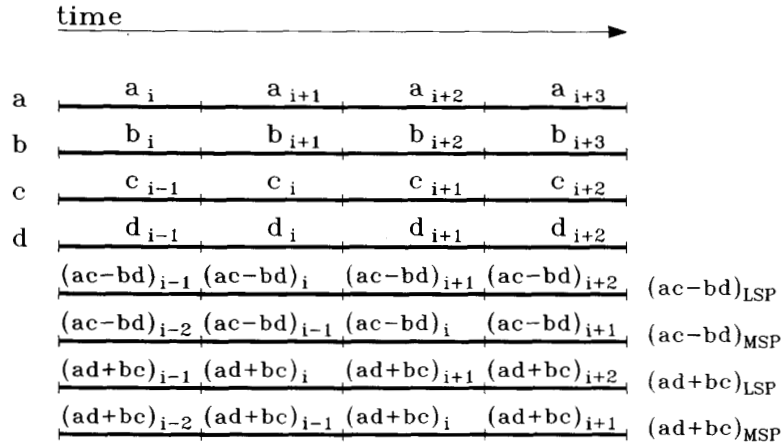


Figure 2: Time diagram of the complex multiplier.

in the following (a wider discussion can be found in [5]) whenever implementation data are reported. This also implies the the two sets of shift registers consist of precisely two shift registers per set.

A prototypal multiplier has been implemented, in CMOS 2μ technology, in semi-automatic way, by means of the SOLO 1400 development tool (ES2). Table 1 shows the main physical characteristics of the device. Maximum clock frequency

Complex Multiplier Characteristics	
Format	$(8 + i8) * (8 + i8) = 16 + i16$ bits
Technology	CMOS 2μ (ES2) - Semi-Automatic Silicon Compilation
Dimensions	$2.97 \text{ mm} * 2.23 \text{ mm} = 6.63 \text{ mm}^2$
Timing	71.75 MHz best - 35.65 MHz worst

Table 1: Characteristics of a complex multiplier, with $n = 8$.

has been evaluated by simulation in the best and in the worst case. Best case idealizes the device by assuming standard fixed delays and no degradation due to power, temperature, etc., whereupon worst case corresponds to the military worst conditions of commercial logic-temporal simulators.

3 Testability

The testability of the bit-slices has been studied considering the approach presented in [6]. This methodology approaches the testing problem by modeling the structure as an array of (identical) finite state machines and by deriving a test sequence from their transition diagrams, under a functional fault model. In this fault model three types of faults are considered, namely:

- An incorrect change of the output of some transition.

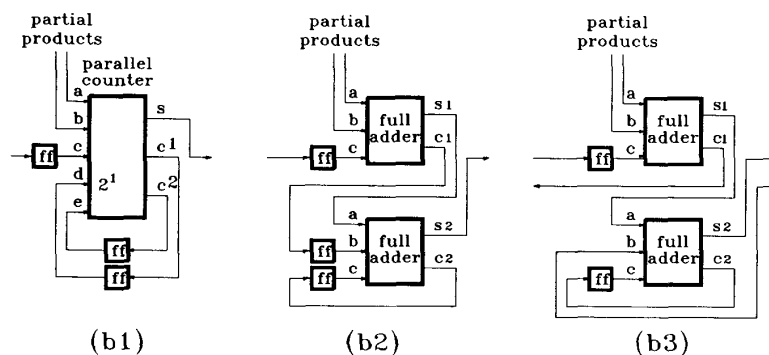


Figure 3: Three solutions for the implementation of the cells of the adder/accumulator array.

- An incorrect change of the final state of some transition.
- Both the final state and the output of some transition different from the correct ones.

Multiple faults are taken into account considering all possible combinations of these three classes of faults, for distinct transitions. This fault model has been verified to map completely onto the stuck-at fault model [10].

Each transition of the finite state machine is verified by checking its outputs and its next state. This check is performed by concatenating the input label of the transition under observation a specific input sequence, named Unique Input/Output Sequence [7], or a set of UIO sequences if only one cannot be found. A UIO sequence for a state is an input/output behavior which is not exhibited by any other state. hence a UIO sequence allows an univocal identification of the final state of a transition. Not all states are characterized by a UIO. In this case a set of UIOs which allow to partially distinguish the state from a subset of all other states of the FSM are identified and the transition must be applied a number of times equal to the number of UIOs identified to verify the correctness of the final state against all other states of the machine.

In principle for each state transition the test sequence can be generated as follows:

- Applying a reset input such that the initial state of the machine is known.
- Applying a set of transitions allowing to drive the machine into the initial state of the transition that is to be tested.
- Applying the transition under test concatenated to its UIO sequence.

Obviously the first part of the test sequence is not necessary if we are sure that we are already in the correct state. Therefore, considering only the test subsequences constituted by a transition concatenated to the corresponding UIO, for each transition, one can try, after eliminating all subsequences completely contained in others, to connect all test subsequences together, either by concatenation or by overlapping, in order to minimize the overall test length. Different methods have been proposed, the most frequent ones based on the solution of the Chinese Postman Tour problem on a graph constructed with the test subsequences.

In the case of an array of sequential cells, it is not sufficient to build the test sequence, since some input sequences cannot be propagated through the array and/or their results cannot be propagated to the observable output, without error masking. Hence, a controllability procedure and an observability procedure have been identified to verify some sufficient conditions to guarantee that any sequence can be applied, while maintaining the necessary characteristics for controllability and observability. However, if such properties are not satisfied, then it is still possible to build a test sequence by verifying at each step of concatenation or overlapping that the resulting input sequence is controllable and observable. This is performed by checking whether the input sequence being created belongs to the output language of the dependent inputs produced by the finite state machine, for controllability, and that the resulting output sequence belongs to the language accepted by the distinguishability graph, for observability.

The sufficient conditions for controllability allow to verify if it is possible to drive each test sequence to any bit-slice (controllability problem). This check can be performed by deriving a controllability graph from the diagram of the finite state machine [6]. Topological properties on such graph allow to identify, given the test sequence, if it can be generated at each step by the finite state machine, until the primary input is reached. In the present case, since the controllability graph of the bit-slices is strictly connected, any test sequence can be driven to any internal bit-slice.

To analyze observability, a distinguishability graph can be derived from the finite state machine [6], in order to verify propagation of the test results without error masking until the primary output is reached. The distinguishability graph is constituted by the same set of states of the original finite state machine, connected only by those transitions which allow to maintain distinguishability of the output values, given the same independent input. If the obtained graph is strictly connected, then observability is guaranteed for any bit-slice.

For the first two unilateral bit-slices in figure 3b1 and 3b2 here considered, the distinguishability graph coincides with the diagram of the finite state machine, hence it is possible to propagate the test results from any bit-slice to the main outputs without masking effects. If the third bit-slice in figure 3b3 is considered, the analysis performed is not sufficient, since this is a simple bilateral array. Note, however, that the propagation from right to left is limited to a depth of two cells. In this case three different graphs based on the distinguishability graph have to be analyzed in order to verify the influence in terms of controllability and observability of the left-to-right data flow with respect to the opposite one. The sufficient conditions are still based on the connectivity of such graphs and in the case of this bit-slice they are satisfied, thus granting for both controllability and observability.

4 On-line error detection

A widely used class of arithmetic codes, namely the AN code, is adopted to allow on-line error detection [8]; in particular, the $3N$ code has been implemented to detect errors under the single stuck-at fault model. This coding technique, under the traditional single stuck-at fault model, is known to be efficient for parallel multipliers, while it shows poor performances in traditional bit-serial architectures, due to the dispersion of the errors over a large number of dependent bits, caused by the serial operation of the device. In fact, a fault in a parallel structure, which computes a whole product in a single clock cycle, only affects the bit of the result which is generated in the same spatial position where the fault itself is located. A fault in a serial structure, which distributes the computation of the result over a time interval of several clock cycles, affects all bits of the result which will cross the spatial position of the fault itself. Thus, a single fault in a parallel multiplier causes single errors, whereupon it causes multiple errors in serial multipliers. The presence of multiple errors may induce error masking effects, which make error detection a difficult problem.

Since the present structure shares some characteristics of parallel architectures and others typical of bit-serial ones, it is not possible to achieve maximum performances in detecting errors by means of the AN code, as for parallel structures. However, simulations have shown that a significantly high error detecting capability can be achieved by means of the $3N$ code, at reasonable costs in terms of additional computational time, data redundancy and silicon area required by the error detection circuitry.

The $3N$ encoder introduces redundancy into the operands through an arithmetic linear transformation, namely multiplication of both factors by the code generator 3. Figure 4a shows the $3N$ encoder. $3N$ coding is obtained in serial arithmetic by shifting each factor and adding it to the unencoded factor itself; hence the resulting serial $3N$ encoder is very simple. This method works also for two's complement integers, without significant modifications. The encoded factors should be represented over $n + 2$ bits (where $n \geq 1$

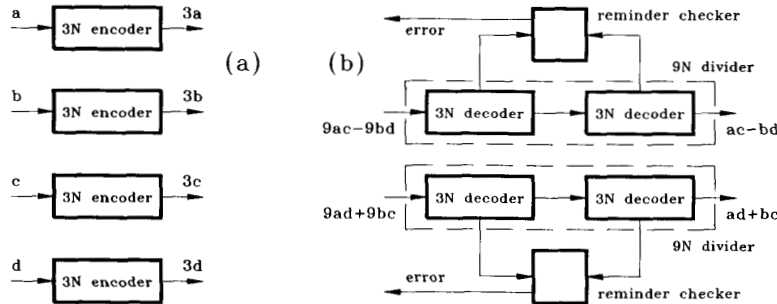


Figure 4: (a) $3N$ serial encoder. (b) $3N$ serial decoder (actually a divider by 9).

is the length of unencoded factors). However, in order to simplify the encoding circuitry, the serial encoder cannot widen the operands of two bits, since this is incompatible with the achievement of full external pipelining, therefore the original unencoded input factors, of n bits each one, need already be sign extended over two additional redundant sign bits. The two bits sign extension is ignored by the encoder and the encoded factors still range over n bits.

The $3N$ decoder eliminates redundancy from the result by means of another arithmetic transformation, namely division by $3 \times 3 = 9$ of the result. This is done in serial arithmetic by means of a recursive operation: the decoded product is shifted by 3 positions, i.e. it is multiplied by 8, and then it is subtracted from the unencoded product, yielding recursively the decoded product. Figure 4b shows the $3N$ decoder.

The $2n$ bits of the encoded result are output partitioned into two halves of n bits on two different lines, the least significant half preceding the most significant half. Hence the decoders need be duplicated in order to receive both flows separately. The decoded result still ranges over $2n$ bits. Its four most significant bits are redundant and they should represent a sign extension. If this is not verified, then the result does not belong to the $3N$ code and therefore an error is detected. A specific circuit checking the four bits is cascaded to the output of the multiplier.

The additional area required by the encoding and decoding elements, with respect to the total area of the multiplier, has been evaluated as a function of the number of bit-slices. The resulting curve is shown in figure 5 and is obtained by the gate counting of different instances of the multiplier, for various numbers of bits. The overall area of the error detection circuitry consists of a constant term, independent of the number of bit-slices n , plus a slowly increasing term, dependent on n . The constant term causes the

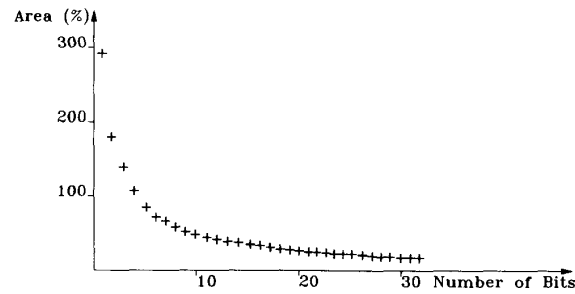


Figure 5: Area comparisons of the error detection circuitry with the nominal multiplier, parameterized w.r.t. the number of bits of the operands.

area of the error detection circuitry to be much larger than area of the nominal multiplier, but its effect is nullified quickly when n increases, only leaving the effect of the variable term, which is however not relevant.

Of course, the maximum clock frequency of the redundant multiplier remains equal to the nominal one, as the structure is a semisystolic array and the encoder/decoder is a simple and fast sequential machine.

Finally, we consider the number of undetected errors, i.e. of all those errors, which generate as product a multiple of 9. Simulations have shown that in the worst case a 3% of stuck-at faults will be undetected. Table 2 shows the fault coverages, obtained by simulation, under the single stuck-at fault model, for a multiplier of 10 bit-slices.

Undetected Faults (in %) for each Bit- Slice of a 10 Bit Multiplier										
Bit-Slices Positions										Ext. Adders
1	2	3	4	5	6	7	8	9	10	
2.4%	2.7%	1.6%	2.4%	2%	2.6%	2.4%	2.5%	3.1%	2.8%	3.4%

Table 2: Percentages of undetected faults for each bit-slice, under the single stuck-at fault model.

5 Reconfiguration

Reconfiguration can be performed in an off-line fashion through an approach similar to the Diogenes approach [9]. Switches and interconnections are added to the basic structure between bit-slices; after localizing the fault the involved bit-slice is switched off by adjusting properly the interconnection switches. This however requires suitable fabrication technologies, which allow to modify or even to undo part of the production process.

A self-reconfiguration scheme can also be envisioned. For run-time fault localization the bit-slices can be duplicated; both bit-slices of each pair work on the same data. EXOR gates are introduced between each pair of bit-slices, allowing to compare the results; the comparison is stored for future use. One bit for each pair of bit-slices suffices. If a mismatch is found the pair of faulty bit-slices is identified by this bit. When the MSB (Most

Significant Bit) of the product is computed, the presence of possible errors is detected by the error detection circuitry. Through the scanning of the stored comparisons, the faulty bit-slice is uniquely localized, under the single stuck-at fault model. The faulty bit-slice should be bypassed by means of a set of redundant interconnections and switches, logically similar to the ones used for off-line reconfiguration. These switches are controlled by a chain of finite state machines, which manage the actual localization of faults and activate the reconfiguration switches when the $3N$ decoder detects an error. Reconfiguration can also be introduced at a higher granularity level, allowing to disconnect groups of adjacent bit-slices, rather than individual ones. This allows to save switches and interconnections. A parameter C can be used to characterize this granularity. Case $C = 1$ coincides with a pure double modular redundancy. Case $C = n$ indicates the presence of n switches, i.e. the ability to disconnect individual faulty bit-slices. Intermediate cases correspond to switches controlling stacks of $\lceil \frac{n}{C} \rceil$ adjacent bit-slices each one.

These possible on-line reconfiguration schemes have been compared in terms of timing effect and additional silicon area as a function of the number of bits and of the number of reconfiguration elements (the parameter C). Figure 6a shows the percentage area increase of the on-line reconfigurable multiplier w.r.t. the nominal one, while 6b shows the same curve w.r.t. the multiplier equipped with error detection circuitry, but not on-line reconfiguration circuitry. The curves are parameterized by C ; three possibilities $C = 1, 3, 5$ are reported. The major factor in the additional silicon area derives from duplication; in fact

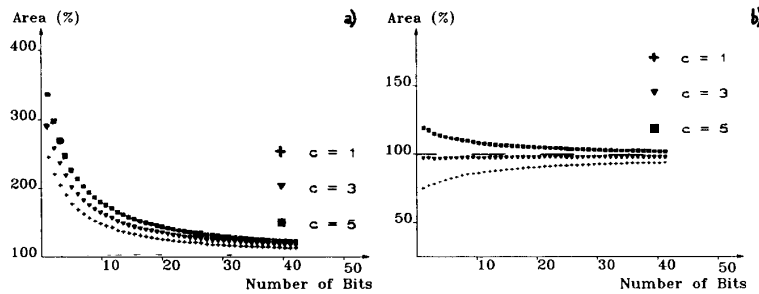


Figure 6: (a) Area increase due to error detection and on-line reconfiguration capability w.r.t. the number of bits. (b) Area increase due to on-line reconfiguration, w.r.t. the number of bits. Both curves are parameterized by C .

the curves tend asymptotically to twice the nominal area. The number of reconfiguration elements does not actually affect the area when increasing the number of bit-slices. Figures 9a and 9b represent respectively the comparison between the self-reconfiguring multiplier and the original one and the comparison between the multiplier with only error-detection and the self-reconfiguring one.

6 Conclusion

A complete analysis of the architectural design and the implementation of a serial complex multiplier has been presented. The multiplier exhibits different architectural interesting properties: high clock frequency, full testability, diagnosability and reconfigurability.

Such multiplier structure has proved to be a building block for more complex operation, mainly serial complex convolution [11,12]. Future work may be directed to testability,

diagnosability and reconfigurability for serial convolvers; part of this research is already active [11,12]. Other research directions are the extension of the present architecture and its properties to numeric domains other than the complex field.

References

- [1] L. Dadda, D. Ferrari, Digital Multipliers: a unified Approach, in *Alta Frequenza*, vol. 37, n. 11, November 1968
- [2] J.T. Scanlon, W.K. Fuchs, High Performances Bit-serial Multiplication, in *IEEE Transactions on Communications*, 1986
- [3] E. E. Swartzlander, The quasi-serial Multiplier, in *IEEE Transactions on Computers*, vol. C-22, April 1973
- [4] L. Dadda, On serial Input Multipliers for two's Complement Numbers, in *IEEE Transactions on Computers*, 1987
- [5] L. Breveglieri, V. Piuri, D. Sciuto, Fast pipelined Multipliers for Bit-serial Complex Numbers, Proceedings of COMPEURO, 1991
- [6] G. Buonanno, F. Lombardi, D. Sciuto, Testability Conditions for linear sequential Arrays, Proceedings of IEEE PCCC, 1991
- [7] K. K. Sabnani, A. T. Dahbura, A Protocol Test Generation Procedure, in *Computer Networks*, vol. 15, n. 4, 1988
- [8] *Error Coding for arithmetic Processors*, T. R. N. Rao, Academic Press, N.Y., 1974
- [9] A. L. Rosenberg, The Diogenes Approach to testable Fault tolerant Arrays of Processors, in *IEEE Transactions on Computers*, October, 1983
- [10] F. Lombardi, D. Sciuto, Y.-N. Shen, Evaluation and Improvement of Fault Coverage for Verification and Validation of Protocols, in *IEEE Proc. Int. Symposium on Parallel and Distributed processing*, December 1990
- [11] L. Breveglieri, L. Dadda, A Bit-sliced Convolver, Proceedings of ICCD '88, New York, U.S.A., 1988
- [12] L. Dadda, Polyphase Convolvers, in *Journal of VLSI Signal Processing*