A PARTITIONING APPROACH TO DESIGN FAULT-TOLERANT ARITHMETIC ARRAYS

Thou-Ho Chen , Liang-Gee Chen and Yeu-Shen Jehng

Department of Electrical Engineering, National Taiwan University
Taipei, Taiwan 10764, R.O.C.

ABSTRACT An alternative fault-tolerant design in VLSI-based arithmetic arrays using the partitioning technique is presented in this paper. The basic concept behind this study is that the arithmetic array can be divided into m parts and its operation can be completed through m iterative calculations with some one part. By taking three such parts with majority-voting technique at each iteration, error correction in whole can be achieved through m-step computations. This leads to the same capability of fault tolerance as the triple modular redundancy (TMR). The overheads of chip area and operation time are just only introduced by multiplexers, latches and voters, and can be reduced by selecting an appropriate value m . Based on AT^2 (where A is denoted by the chip area and T is the operation time) measure of VLSI performance, the proposed design is shown to be superior to the general TMR method. In addition, some application-specified tradeoffs between speed performance and area cost are also presented.

I. Introduction

Arithmetic has played important roles in computing systems for a long time. Due to the advances in VLSI technology, large integration of processing elements has been found to be feasible in achieving high-speed computation. Consequently, more and more sophisticated arithmetic processors have become standard hardware features for today's high-performance digital computer systems. Since any fault may seriously damage the operations of the processors, high reliability will become a very important issue in future VLSI design. Conventionally, system reliability has been achieved by two basic strategies: fault avoidance and fault tolerance [1][2]. In the case of fault avoidance, the reliability can be improved by the use of high-reliability components which are provided through the sophisticated testing schemes. Those schemes can identify permanent faults only, but not transient faults. This inadequacy has motivated the development of fault-tolerance techniques, which can tolerate all permanent and transient faults during the system operation.

In general, fault-tolerance in arithmetic operations can be achieved through space or time redundancy. Space redundancy uses coding techniques or module-redundant schemes with voting to perform the corrective action. In coding techniques, the complexity of the decoding process for error correction and the necessity to provide some levels of error-free operation in the course of encoding and decoding may complicate the design, and the limited fault model based on the traditional stuck-at fault, which is inappropriate for VLSI [3][4], will decrease the error correction capability. Triple modular redundancy (TMR) can easily obtain the correct result despite any existing fault in one of the three modules, but it requires at least 200% more hardware overhead. The time redundancy approach utilizes only one module to provide error correction by repeating the operation at least three times. For error correction in adders and multipliers [5], the structure is physically divided into three or four identical sub-units to carry out three computation steps with the same set of operands. It can take a lower hardware cost, but the degenerated throughput arising from extra recomputing time will make it unsuitable for the computation-intensive work and real-time computing systems. Judged by the VLSI performance measure of AT^2 (where A is the chip area and T is the operation time) [6], time redundancy is rather a high price to pay.

In this paper, an alternative design of adders and array multipliers with error correction using the partitioning technique is proposed. The basic concept is that the arithmetic array can be divided into m parts. If every part is identical, error correction can be achieved through m-step computations by organizing the TMR scheme with three parts. If they are not the same, it is available that some one part of them can be utilized to build the TMR scheme with three copies for achieving fault tolerance. Since the computing time of each part is about $1/m$ times that of the original structure, the time overhead in the whole operation consists only in the accumulation of the switching, latching and voting time; but the hardware overhead results from multiplexers, latches and voters. Both time and hardware overheads can be reduced significantly by selecting an appropriate$m$ of partitions. In order to demonstrate the effectiveness of the proposed design, a comparison with the general TMR scheme in terms of

CH3129-4/92/0000-0432 $3.00 © 1992 IEEE

hardware, time and area-time measure AT^2 is made and the optimal m can be derived. In addition, some interesting tradeoffs with moderate hardware overhead and little performance penalty among partitioning sets may be attractive for some special-purpose applications. The comparisons provided in this paper are based on the evaluation of area requirement and operation time using the standard cell library of CMOS technology in the Lcells with the GDT tools. However, for lack of considering the interconnection delay and area, only a rough estimate is available.

II. Partitioning Methodology

Many partitioning approaches in arithmetic units have been developed for achieving concurrent error detection such as REDWC [7], BIDO [8], and error correction [5]. In REDWC, the adder, as well as the operands, is divided into the lower and upper halves which are used to perform two identical half-additions. Two results of both half-adders at each half-addition are compared for achieving error detection and one result is then stored to represent the corresponding half of the final result. BIDO employs the inherent idle full adders in each half-multiplier of array structure to perform the repeated operation. As a result, both the normal and repeated operations are carried out in two half-multipliers alternately to do concurrent error detection. Based on the partitioning concept, this section will describe a novel fault-tolerant design in the adder to illustrate our methodology. The area requirement and performance penalty will also be evaluated.

Given the data dependency, an n-bit adder can be divided into two $(n/2)$-bit adders to carry out n-bit addition [2][7]. Each $(n/2)$-bit adder can obtain a complete result through two half-additions. In the first half-addition, we use three $(n/2)$-bit adders with lower-half operands to perform the TMR operation to secure a correct lower-half result. This half result is then stored to represent the lower-half of the final result. To acquire the upper-half result, those three $(n/2)$-bit adders are again utilized with upper-half operands to perform the TMR operation during second half-addition. At last, a correct complete result of addition is generated. Fig. 1(a) shows operations of the proposed fault-tolerant addition. Based on the same deduction, an n-bit addition can be divided into four $(n/4)$-bit adders, because the result of n-bit addition can be obtained by calculating four times with an $(n/4)$-bit adder. By using three $(n/4)$-bit adders to construct the TMR structure, a correct result of n-bit addition can be derived through four-step computations. Operations of the scheme is illustrated in Fig. 1(b). For the same reason, the fault-tolerant n-bit addition can be achieved on three (n/m)-bit adders by performing m times of TMR operation. In making the addition, selection of the appropriate operands is accomplished by utilizing the m to 1 multiplexer of (n/m)-bit. In the course of the operation, the partial result must be stored at each computation-step and the carry-out must also be latched in order to provide the carry-in in the subsequent partial addition. Since the carry-out of the (n/m)-bit adder may be erroneous, it is also required to be voted. The block diagram of an n-bit adder with fault tolerance is shown in Fig. 2.

The adder used in our design must satisfy the condition that the carry-out propagation among sub-adders should transfer ripply from the lower sub-adder to the upper sub-adder when the original structure is divided. But within each sub-adder architecture, any type of addition is allowable. Since the error corrective action of the proposed design is identical to that of the general TMR, our fault assumption is confined to the single faulty module and the fault-tolerant capability is the same as TMR's. Most modular-redundant approaches are also applicable for our method, among them being the NMR, NMR with spares, TMR with triplicated voters, self-purging approach and sift-out modular approach [1][2].

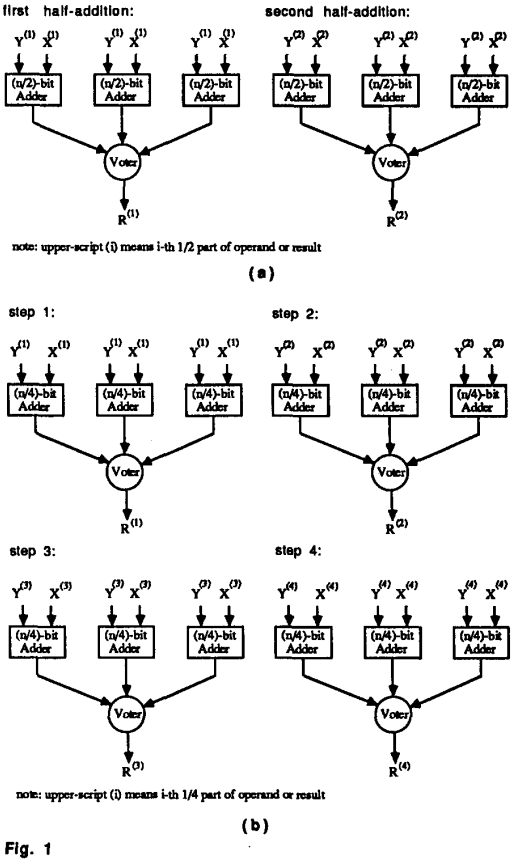
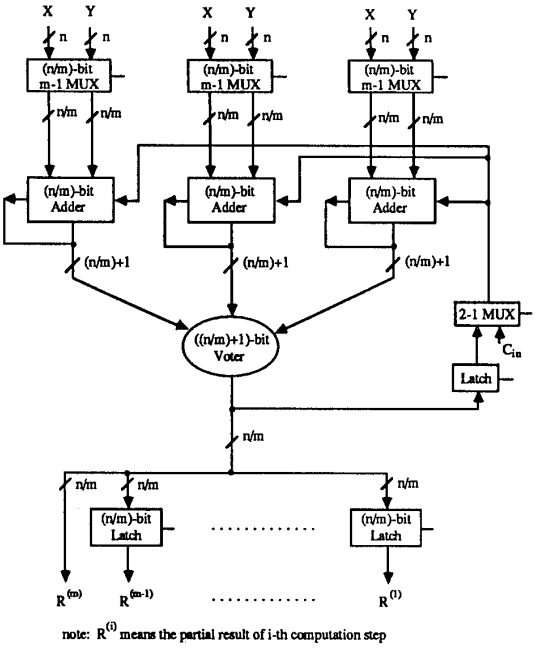


Fig. 1

(a) Two-step computations in n-bit fault-tolerant adder

(b) Four-step computations in n-bit fault-tolerant adder



note: $R^{(i)}$ means the partial result of i-th computation step

Fig. 2 The fault-tolerant scheme of n-bit adder using m partitions

In this study, we utilize the standard cell library of CMOS technology in the Lcells generator with the GDT tools to implement the proposed design and use the Lsim simulator [9] to do all simulations of the proposed design. For moderate evaluation of the operation time and area requirement, we define τ as the unit P-N-transistor pair delay corresponding to one level of the logic circuit and α as the unit area taken in the pair. Both τ and α will depend on the implemented technology. The P-N-transistor pair counts and time delays of those typical cells and modules used to realize the proposed design are listed in Table 1.

For simplicity, the following representations of variables are used throughout this paper:

t_{FA} = time delay of 1-bit full adder
$t_{4\text{-CLA}}$ = time delay of 4-bit carry lookahead adder
t_{voter} = time delay of 1-bit voter
t_{latch} = time delay of 1-bit latch
$t_{m\text{-1 MUX}}$ = time delay of 1-bit m to 1 multiplexer
$t_{1\text{-2 DMUX}}$ = time delay of 1-bit 1 to 2 demultiplexer
t_{AND} = time delay of AND gate
t_{XOR} = time delay of XOR gate
a_{FA} = area of 1-bit full adder
$a_{4\text{-CLA}}$ = area of 4-bit carry lookahead adder
a_{voter} = area of 1-bit voter
$a_{m\text{-1 MUX}}$ = area of 1-bit m to 1 multiplexer
$a_{2\text{-1 MUX}}$ = area of 1-bit 2 to 1 multiplexer
a_{latch} = area of 1-bit latch
a_{AND} = area of 1-bit AND gate
$a_{1\text{-2 DMUX}}$ = area of 1-bit 1 to 2 demultiplexer

From the schematic diagram of Fig. 2, the time T and area A requirements of an n-bit ripple carry adder (RCA) are

$$T_{RCA} = (n)*t_{FA} + (m)*t_{voter} + (m)*t_{m\text{-1 MUX}} + (m\text{-}1)*t_{latch}$$

$$A_{RCA} = (3n/m)*a_{FA} + ((n/m) + 1)*a_{voter} + (6n/m)*a_{m\text{-1 MUX}} + a_{2\text{-1 MUX}} + ((n(m\text{-}1)/m) + 1)*a_{latch}$$

For carry lookaheasd adder (CLA), the gates are partitioned to keep the number of inputs at or below four, generally. This is typical of the carry lookahead adder that would be used in a gate array or the standard cell design [10]. For example, a 16-bit adder can be built through serial connection using four 4-bit carry lookahead adders where each adder possesses a lookahead-carry capability but the carry-propagation is rippled from cell to cell [7]. The evaluation of time and area requirements is as follows:

$$T_{CLA} = (n/4)*t_{4\text{-CLA}} + (m)*t_{voter} + (m)*t_{m\text{-1 MUX}} + (m\text{-}1)*t_{latch}$$

$$A_{CLA} = (3n/4m)*a_{4\text{-CLA}} + ((n/m) + 1)*a_{voter} + (6n/m)*a_{m\text{-1 MUX}} + a_{2\text{-1 MUX}} + ((n(m\text{-}1)/m) + 1)*a_{latch}$$

Table 1: Cells used in partitioning design for evaluation of area and time

Cell or Module	NOT	2-input NAND	2-input NOR	2-input AND	2-input OR	2-input XOR	2-input XNOR	1-bit Latch	2-1 MUX	1-bit FA*	1-bit Voter*	4-bit CLA*	4-1 MUX*	8-1 MUX*	16-1 MUX*	32-1 MUX*	64-1 MUX*	1-2 DMUX*
P-N count (α)	1	2	2	3	3	5	5	3	6	16	9	91	15	36	75	156	327	7
Delay (τ)	1	1	1	2	2	3	3	2	2	5	2	8	2	2	4	4	6	2

* Modules constructed with the Lcells Standard Cell Library

Table 2: Evaluation in fault-tolerant adders

Scheme / Item		Ripple-Carry Adder						Carry Lookahead Adder					
		16-bit	32-bit	64-bit	% increase * *			16-bit	32-bit	64-bit	% increase * *		
					16-bit	32-bit	64-bit				16-bit	32-bit	64-bit
m = 0 *	T	80	160	320	0	0	0	32	64	128	0	0	0
	A	256	512	1024	0	0	0	364	728	1456	0	0	0
m = 1 *	T	82	162	322	3	1	1	34	66	130	6	3	2
	A	912	1824	3648	256	256	256	1236	2472	4944	240	240	240
m = 2	T	90	170	330	13	6	3	42	74	138	31	16	8
	A	785	1553	3089	207	203	202	947	1877	3737	160	158	157
m = 4	T	102	182	342	28	14	7	54	86	150	69	34	17
	A	641	1265	2513	150	147	145	722	1427	2837	98	96	95
m = 8	T	126	206	366	58	29	14	-	110	174	-	72	36
	A	605	1193	2369	136	133	131	-	1274	2531	-	75	74
m = 16	T	206	286	446	158	79	39	-	-	254	-	-	98
	A	569	1121	2225	122	119	117	-	-	2450	-	-	68

In the above the unit in area evaluation is α , in time evaluation is τ .
* m = 0 denotes the original adder, m = 1 the general TMR scheme
* * the difference over that of the original adder

In the time T of the above formulas, delay of the 2-1 MUX is exchanged due to overlapping with the operation of the m-1 MUX.

Table 2 summarizes both time and area taken in the RCA and CLA for 16-bit, 32-bit and 64-bit designs, as well as their percentage increase over that of the original adder. From those partitions, it has been shown that the optimal values of m resulting from AT^2 measure can be provided in order to obtain a lower area-time cost design. The proposed fault-tolerant design is compared with the TMR (where m = 1), the latter being used as the base in percentage calculation to find out the design improvement, which is defined as the percentage difference in AT^2. It can be observed that for a 32-bit RCA the optimal value of m is 4 and the improvement is about 13% and for a 64-bit RCA the improvement increases to 22% or so though the optimal value of m is still 4. In the case of the CLA, a 32-bit scheme may yield 5% improvement when the optimal m is 4 but a 64-bit scheme will produce 24% improvement when the optimal m remains the same. Fig. 3 shows the AT^2 costs calculated when m = 1, 2, 4, 8 and 16 for 16-bit, 32-bit and 64-bit wordlength systems. From these experimental results, it is clear that the proposed design is better than the general TMR especially in a large wordlength scheme. Nevertheless, the AT^2 cost is not reduced with increasing m. This is because when m is increased the operation time becomes much longer but the required area decreases only slightly. The phenomenon is due to the fact that the multiplexer takes a heavy weight on area evaluation in larger m. If the wordlength increases, the effect of the multiplexer will be reduced significantly. It should be pointed out that the partitioning is ineffective for the 16-bit adder because the machine has limited operation time and small area requirement. In other words, if the original structure is large in both area and operation time, the multiplexer will affect less.

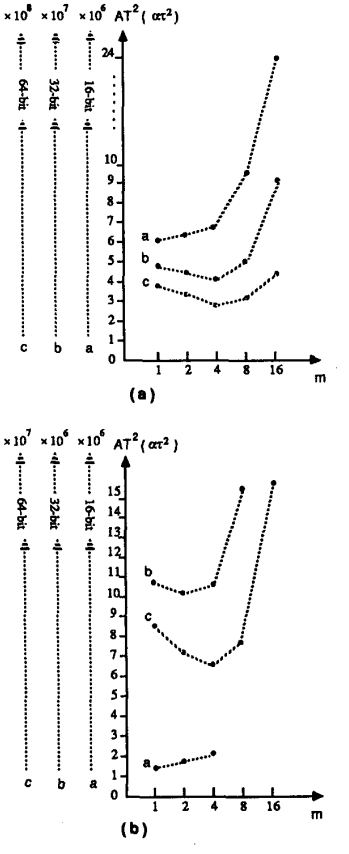


Fig. 3(a) AT^2 cost distribution in RCA
(b) AT^2 cost distribution in CLA

On the other hand, the partitioning can incur a lower hardware cost without significantly sacrificing speed. Consider the RCA in Table 2. When m = 2, the percentage increase for the 16-bit scheme is 13% in time and 207% in area; for the 32-bit scheme, 6% in time and 203% in area; for the 64-bit scheme, only 3% in time and 202% in area. If m = 4, for the 32-bit scheme the speed penalty is 14% and the area increase is 147% and for the 64-bit scheme the speed penalty is 7% and the area increase is 145%. The above fact point up the superiority of the proposed design over the general TMR where the area increase is 256% and speed penalty is up to 3%. Results of other tradeoff analysis between time and area can be observed from Table 2, and such tradeoffs may be attractive for some special-purpose applications.

III. Design of Fault-Tolerant Array Multiplier

To implement a concurrent error-correctable array multiplier with the proposed method, some modifications of the original structure are required. For the purpose of reducing extra computing time during iterative operations, the error correction strategy needs to be different from that of the adder described above.

Now consider an n-by-n unsigned carry-save array multiplier (CSM), also called the Pezaris array multiplier [11]. The n - 1 upper rows will perform carry-save operations but the last row forms a ripple-carry adder. For obtaining a good partitioning, the two substructures are required to have different processing in designing the fault-tolerant scheme. For achieving the integrality of partitioning and deriving much more sets of partitions, the order m of partitions is selected as an even number since the wordlength is often an even number. This leads that one extra carry-save row needs to be added on the upper substructure. Fig. 4(a) shows a modified 6-by-6 CSM where only the input data of the first row is different from the original scheme. When using the partitioning with m = 2, two computation steps are needed for completing the multiplication in this modified multiplier, as shown in Fig. 4(b). In step 1, all intermediate results must be latched as input data for the subsequent step and only product bits P_0, P_1

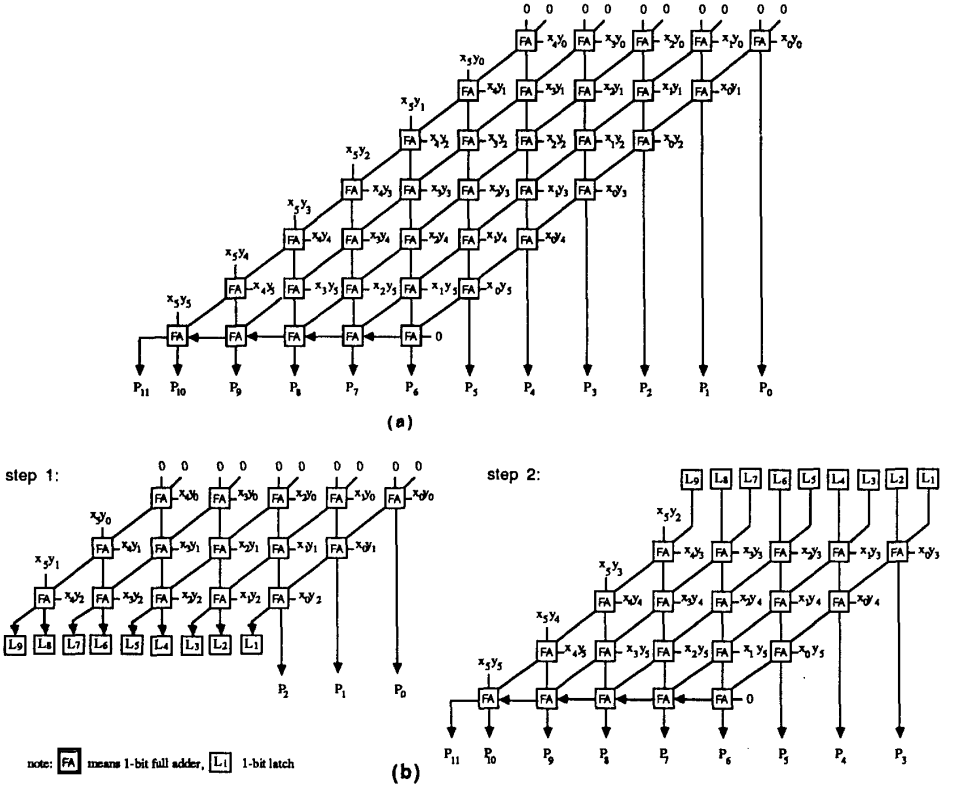


Fig. 4(a) A modified 6-by-6 carry-save array multiplier (CSM)
(b) Two computation steps of 6-by-6 CSM with error correction

and P_2 are voted for the final result. This partitioning type aims at reducing hardware cost but it makes both computation steps unequal in operation time. In step 2, those latched data are calculated to generate the remaining product bits P_3~P_{11}, which are again voted for the final result.

In the multiplication of two n-bit numbers, the result could be $2n$ bits or n bits long, depending on the precision desired. In this study, we settle on the n-bit result to design fault-tolerant multiplication. Fig. 5 shows the schematic diagram of an n-by-n CSM with concurrent error correction, using partitioning of m. In the partitioning, the part within the dashed-line block is n/m carry-save rows and their function is to carry out m iterative calculations. At each iteration, the data come from two multiplexers and the computed result is latched as partial input data for the iteration. In the two multiplexers, the first one is to select the appropriate partial product terms $x_i y_j$ for the corresponding iteration and hence it needs to bear m to 1 selecting capability. For the second multiplexer with 2 to 1 selecting capability, data zero is taken as input data only in the first iteration and in the remaining iterations it selects the latched data resulting from the previous iteration as another input data of the current iteration. The 1 to 2 demultiplexer receives the intermediate results and transfers them to the latch at each of the previous m - 1 iterations. But in the last iteration, the demultiplexer transfers the last intermediate result to the lowest row, a ripple-carry adder, to compute the final result. It should be pointed out that to reduce extra time delay for the whole operation, only one vote taking for three final results is called for, a requirement unlike that of the adder, which needs m times of voting.

Just like the adder, the time T and area A requirements of the multiplier are as follows:

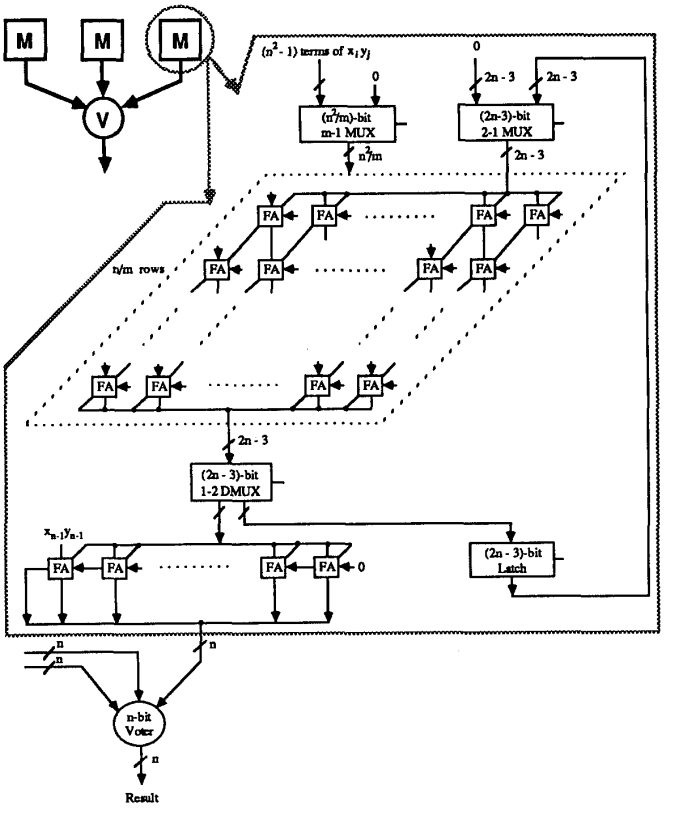


Fig. 5 The fault-tolerant scheme of n-by-n CSM using m partitions

$$T_{CSM} = 2(n\text{-}1)*t_{FA} + (m)*t_{m\text{-}1\ MUX} + (m)*t_{1\text{-}2\ DMUX} + t_{voter} + (m\text{-}1)*t_{latch} + t_{AND}$$

$$A_{CSM} = (3n^2)*a_{AND} + 3(n(n\text{-}1)/m + n - 1)*a_{FA} + 3(2n\text{-}3)*a_{2\text{-}1\ MUX} + (3n^2/m)*a_{m\text{-}1\ MUX} + 3(2n\text{-}3)*a_{1\text{-}2\ DMUX} + 3(2n\text{-}3)*a_{latch} + (n)*a_{voter}$$

With Table 1, the above two formulas can be applied to the evaluation of the area and time involved in the 16-bit, 32-bit and 64-bit fault-tolerant schemes. The evaluations as well as the percentage increase over that of the original multiplier are summarized in Table 3. A comparison with the general TMR reveals that the partitioning technique proposed can take a good tradeoff between time and area for other specific applications. Specifically, when m = 4, the percentage increase in the area required is 124%, 97%, and 84% but the sacrifice in speed is only 16%, 8%, and 4% for the 16-, 32-, and 64-bit schemes, respectively. When the bit number is increased, the partitioning is much more effective. This can also be proved in the area-time complexity where the AT^2 cost distribution of three wordlength systems are plotted in Fig. 6. It also indicates that the optimal m is 4 for the 16-bit and 32-bit schemes and 8 for the 64-bit scheme. If compared with the general TMR in regard to on AT^2 cost, the design improvement over the TMR is 3% for the 16-bit scheme with m = 4, 25% for the 32-bit scheme with m = 4 and 38% for the 64-bit scheme with m = 8. As is the case with the adder, the multiplier does not suffer a decline in AT^2 cost with the increasing m because both m to 1 multiplexer and AND-gates of $x_i y_j$ have a heavy weight on area evaluation, especially whren m is large.

Table 3: Evaluation in fault-tolerant CSM

Item \ Scheme		16-bit	32-bit	64-bit	% increase** 16-bit	32-bit	64-bit
m = 0*	T	152	312	632	0	0	0
	A	4608	18944	76800	0	0	0
m = 1*	T	154	314	634	1	1	~0
	A	13968	57120	230976	203	202	201
m = 2	T	164	324	644	8	4	2
	A	12624	46944	180096	174	148	135
m = 4	T	176	336	656	16	8	4
	A	10320	37344	140928	124	97	84
m = 8	T	200	360	680	32	15	8
	A	9456	33696	125952	105	78	64
m = 16	T	-	440	760	-	41	20
	A	-	31296	116160	-	65	51
m = 32	T	-	-	920	-	-	46
	A	-	-	112416	-	-	46

In the above the unit in area evaluation is α , in time evaluation is τ .
* m = 0 denotes the original multiplier, m = 1 the general TMR scheme
** the increase over that of the original multiplier

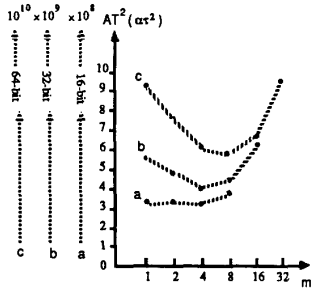


Fig. 6 AT^2 cost distribution in CSM

The design concept of the modified CSM can be similarly applied to other FA-based multiply arrays, such as the carry propagate array multiplier [8], the two's complement bi-section array multiplier and the Baugh-Wooley two's complement array multiplier [11]. Besides, the proposed partitioning technique can also be applied to array dividers. Due to the good regularity and cellular structure in divide arrays, such as the nonrestoring array divider, restoring array divider and carry-lookahead array divider [11], the fault-tolerant scheme can be constructed straightforward and the performance is much better than others mentioned above. Fig. 7 and Fig. 8 shows the fault-tolerant scheme of n-by-n nonrestoring array division (NRD) using m partitioning and its AT^2 cost, respectatively. Different from the adder and the multiplier discussed previously, cost in the fault-tolerant divider decreases with increasing m significantly. This is attributed to the good regularity of the array structure, simple intermediate data dependency to be divided, large area requirements and operation time in itself.

IV. Conclusions

In this paper we have presented a fault-tolerant design in arithmetic units using the partitioning technique. Some interesting designs with moderate hardware overhead and little performance degradation can be derived by selecting an appropriate m. Based on the VLSI area-time measure AT^2, the comparison with the general TMR reveals the effectiveness of our design, especially for the structure where both its area requirement and operation time are large. In reality, different architectures for the same standard cell as well as the technology implemented will result in different evaluations. Nevertheless, from the results presented has been shown that the partitioning will obtain a good tradeoff between area and performance in fault-tolerant arithmetic units for some special-purpose applications. In view of the superiority over the general TMR, the proposed method will be very attractive in future fault-tolerant design. We hope that the partitioning technique will be further applied to other array structures while showing cost-effective fault-tolerance.

References

[1] D. K. Pradhan, *Fault-tolerant computing theorey and techniques. Vol. I, II.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1986.

[2] B. W. Johnson, *Design and analysis of fault tolerant digital systems.* Addison-Wesley, 1989.

[3] J. H. Patel and L. Y. Fung, "Concurrent error detection in ALUs by recomputing with shifted operands," *IEEE Trans. Computers*, Vol. C-31, pp.589-595, July 1982.

[4] J. A. Abraham and W. K. Fuchs, "Fault and error models for VLSI," *Proc. of IEEE*, pp.639-654, May 1986.

[5] L. Subhasis and J. H. Patel, "Error correction in arithmetic operations using time redundancy," in *Proc. 13th IEEE Fault Tol. Comp. Symp.* (FTCS), pp.298-305, 1983.

[6] C. L. Wey and T. Y. Chang, "Design and analysis of VLSI-based parallel multipliers," *IEE Proc.*, Vol. 137, Pt. E, pp.328-336, July 1990.

[7] B. W. Johnson, J. H. Aylor and H. H. Hana, "Efficient use of time and hardware redundancy for concurrent error detection in a 32-bit VLSI adder," *IEEE Journal of Solid-State Circuits*, Vol. 23, pp.208-215, Feb. 1988.

[8] L. G. Chen and T. H. Chen, "Computation with simultaneously concurrent error detection using bi-directional operands," in *Proc. IEEE Int. Conf. Comp. Design (ICCD)*, pp.128-131,1989.

[9] *Lcells Generator Library User Guide, Generator Development Tools*, Ver. 3, Silicon Compiler Systems Co.

[10] N. Weste and K. Eshraghian, *Principles of CMOS VLSI design.* Addison-Wesley, 1985.

[11] K. Hwang, *Computer arithmetic: principles, architecture, and design.* Wiley, New York, 1979.

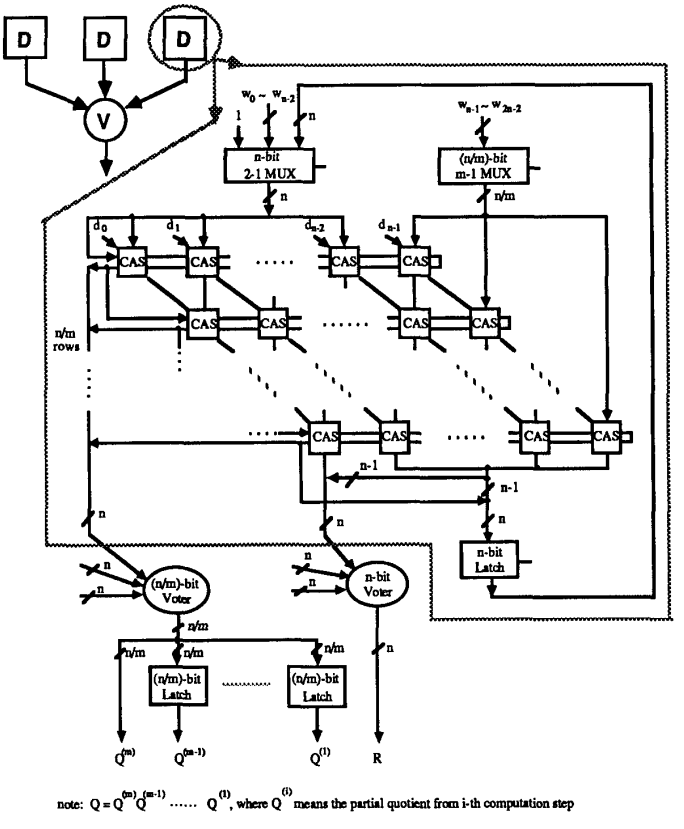


Fig. 7 The fault-tolerant scheme of n-by-n NRD using m partitions

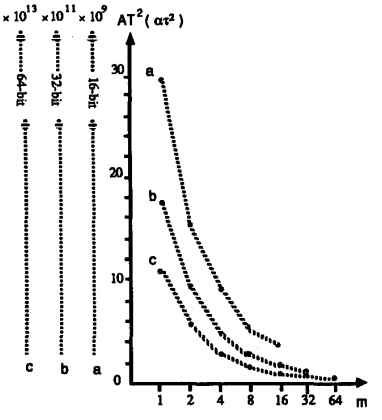


Fig. 8 AT^2 cost distribution in NRD