TIME-REDUNDANT FAULT-MASKING IN ALUs

Chwan-Chia Wu , Lih-Ren Cheng
Department of Electrical Engineering and Technology
National Taiwan Institute of Technology
Taipei, Taiwan 10772, R.O.C.
and
Tien-Shou Wu
Department of Electrical Engineering
National Cheng Kung University
Tainan, Taiwan 70101, R.O.C.

ABSTRACT

A new error correction scheme for bit-sliced ALUs is presented. The method adopted for fault location is an extension of a concurrent error detection scheme called RESO (Recomputing with Shifted Operands). The term bit-slice is used in the generic sense and the length of the slice may be one or more bits. The proposed scheme requires two consecutive computation steps for normal operations and the possible locations of faults, if any, can be located provided that the failures are confined to a certain number of adjacent bit-slices. The fault-free bit-slices of the ALU are thus figured out. The final and effective computation is then carried out through those identified fault-free bit-slices. A circular structure is proposed to realize this concept.

I. INTRODUCTION

Today, computers have been used around every corners of the world. Some applications are so critical that a minor errors caused by the computer may induce a disaster. Therefore, some fault tolerance features must be included to detect and/or correct errors whenever they occur[1,2]. Some error detecting codes [3-7], for instance, were developed for checking arithmetic operations. In the mean time, utilizing a fully duplicated logic unit such as Triple Modular Redundancy (TMR) has been widely accepted as an effective method for checking logical operations. Recently, Patel et al. proposed a time-redundant technique, called Recomputing with Shifted Operands (RESO) [8], which can detect all functional errors resulting from failures confined to a small area of the chip, and is thus quite appropriate for the VLSI technology. RESO, however, does not generally provide error correction in arithmetic operations. Correcting errors in arithmetic operations can be achieved by other time-redundant techniques introduced in the literature such as [9]. The shifting-operand approach has further extended to rotation for error detection and correction by several researches [10,11]. In Butner's TTR (Triple Time Redundancy) approach [10], no communication can exist among the identical bit-slices used. This rules out the use of TTR for adder and other arithmetic unit with carry signals. Only with a substantial increase in hardware inside the bit-slices and outside, TTR can be used for some arithmetic units. While the structure proposed by Cheng et al. [11] does not provide error correction capability.

In this paper we propose an error-correction scheme, which is based on a new fault location method [12] to locates the faulty bit-slices and bypasses these bit-slices for recomputation. Here, the term bit-slice is used in the generic sense and the length of the slice may contain one or more bits. The proposed error-correction scheme indeed uses redundancy in time and can correct errors resulting from the failures confined to a small area of the chip. A circular structure of ALU is introduced to cope with the recomputation requirement. We first describe the fault location method in the next section. An overview of the proposed scheme is given in Section 3, which is followed by the sections addressing the error correction capabilities and the computation steps carried out by this scheme.

II. FAULT LOCATION BY USING RESO

In this section, we give a brief review of the fault location method [12] by which the "suspicious" faulty bit-slices can be located.

We assume that during the first computation step the operands input directly to the ALU and after the desired operation has been carried out, $S^0 = (s^0_{n-1}, \dots , s^0_1, s^0_0)$ is resulted. During the second computation step, the operands are first shifted left by k bit-slices, and then input to the ALU to conduct the desired operation. Finally, the result is shifted right by k bit-slices to yield $S^k = (s^k_{n-1}, \dots , s^k_1, s^k_0)$. Here, the width of each bit-slice is r bits, $r \geq 1$. Therefore, for an n-bit long operand, there requires $m = \lceil n/r \rceil$ to cover the entire operand, where the least significant bit-slice is denoted as bit-slice 0 and the most significant bit-slice is denoted as bit-slice m-1. Then we define $FD_m(S^0,S^k)$ as follows.

Definition: Let S^0 and S^k be compared bit-slice by bit-slice, from the least significant bit-slice to the most significant bit-slice. Further, let i $(0 \leq i \leq m-1)$ be the first mismatched bit-slice found by this comparison, then we have $FD_m(S^0,S^k) =$

i. If S^0 equals S^k, then $FD_m(S^0,S^k) = m$.

With this mismatch function FD, we can end up with the following theorems which indeed describe the principle for fault location. The theorems are given directly without proofs. The readers may refer to [12] for the detail of proofs of these theorems.

THEOREM 1: If the failures in a logic unit are confined to two adjacent bit-slices, and S^0 and S^2 are outputs of this logic unit resulting from two computation steps. Then $FD_m(S^0,S^2)=i<m$ implies that the faulty bit-slices must locate from bit-slice max[0, i-1] to bit-slice (i+3).

THEOREM 2: If the failures in a bit-sliced ripple-carry adder or a bit-sliced carry-lookahead adder are confined to a single bit-slice, and S^0 and S^2 are outputs of this particular adder resulting from two computations. Then $FD_m(S^0,S^2)=i<m$ implies that the faulty bit-slice must locate between (including) bit-slice max[0, i-1] and bit-slice (i+2).

Recall that the bit-slice in a carry-lookahead adder which we mention here is somewhat different from the way we mean for the bit-slice in a ripple-carry adder and is defined by [8].

Theorems 1 and 2 can further extend to a more general situation which is described by the next two theorems.

THEOREM 3: If the failures in a bit-sliced logic unit are confined to k adjacent bit-slices, and S^0 and S^k are corresponding to the computational results of the first step and the second step, respectively. Then $FD_m(S^0,S^k)=i<m$ implies that the faulty bit-slices must be located from bit-slice max[0,i-k+1] to bit-slice (i+2k-1).

THEOREM 4: If the failures in a bit-sliced ripple-carry adder or a bit-sliced carry-lookahead adder are confined to (k-1) adjacent bit-slices, and S^0 and S^k are outputs of this particular adder resulting from two computation steps. Then $FD_m(S^0,S^k)=i<m$ implies that the faulty bit-slices be located from bit-slice max[0,i-k+1] to bit-slice (i+2k-2).

Based on the theorems given above we may came up with a very interesting remark. That is, if the failures are confined to k adjacent bit-slices of a logic unit, then through conducting two consecutive operations as described above and by evaluating $FD_m(S^0,S^k)$, a total of no more than (3k-1) "suspicious" faulty bit-slices can be located. Similarly, if the failures are confined to (k-1) adjacent bit-slices in a bit-sliced ripple carry adder or a bit-sliced carry-lookahead adder, a total of (3k-2) "suspicious" faulty bit-slices can be located by conducting the computations addressing above.

III. ERROR CORRECTION IN BIT-SLICED ALUs

In order to bypass the faulty bit-slices we propose an error correction scheme which is achieved by recomputing with k-bit-slice rotated operands during the second computation step. Specifically, the input operands are first fed into the ALU and the desired computation is conducted. The output result is stored in a register. The second computation step follows by rotating operands left by k bit-slices, performing computation and finally rotating the result back. Comparison of the results from these two steps are then carried out. A mismatch implies errors have been detected and a recomputation is required. Further rotation is needed to bypass the faulty bit-slices and to assure that the recomputation is carried out through fault-free bit-slices.

Clearly, in the fault-free case, the results achieved by step 1 and step 2 should be identical. However, this is true only for bit-wise logical operations when a conventional ALU is used. As for the arithmetic operations, since bit-slices in the arithmetic unit is no longer as isolated as that of the logic unit, a conventional ALU may not guarantee a match all the time under the fault-free situations. The carry signals in an adder, for instance, may be blocked when the fault-free bit-slices are physically separated by a group of consecutive defective bit-slices between them. Such case is amendable if the adder is designed in a circular fashion so that the carry signal generated by the most significant bit in the adder feeds back to the least significant bit as its carry input. Implementations of a circular ripple-carry adder and a circular carry-lookahead adder are illustrated in Fig.1 and Fig.2, respectively. Based on this circular structure, the carry signals can never be blocked even if the fault-free bit-slices are not connected in a consecutive fashion.

When a circular adder is used, two other problems should be concerned and handled properly, they are: (1) carry signals may be erroneously generated by the faulty bit-slices; and (2) the position at which to insert the carry-in signal. To deal with these two problems we need to classify data bit-slices as a combination of information bit-slices and redundant bit-slices. Here, information bit-slices are corresponding to those bit-slices of operands, and the number of these bit-slices is assumed to be m as we used in the previous section. The redundant bit-slices are all set to 0's and the number of which is yet to determine. The data words from two source inputs are fed into the corresponding bit-slices of the ALU for computation. Consider an example as shown in Fig.3, if bit-slices 0 to i and bit-slices (i+h-1) to (m+h-1) are all fault-free bit-slices, then the information bit-slices are all computed through fault-free bit-slices, That is bit-slices 0 to (i-1) and bit-slices (i+h) to (m+h-1). Further, since bit-slice (i+h-1) is fault-free and has been assigned to be a redundant bit-slice, it is clear that the carry signal erroneously generated by the lower order faulty bit-slices (ranging from bit-slice i+1 to bit-slice i+h-2) can never propagate through this bit-slice. The computation result is therefore correct. Next, if a carry-in signal is involved in our computation, we shall determine where to inject this carry-in signal. This is common when cascading a number of ALUs to allow longer data computation. Again, we use the same notion as before, except that this time we insert 1's to both input ports of the bit-slice (i+h-1) when carry-in is "1" and 0's to both input ports of the bit-slice (i+h-1) when carry-in is "0". Such arrangement will again make the erroneous carry signal transparent to the final (m+1)-bit-slice result. Notice that the (m+1)-bit-slice result is taken from the output of bit-slice (i+h) to bit-slice (m+h-1) and bit-slice 0 to bit-slice i. The highest order bit of this computation result, here being the output of bit-slice i, is a carry-out when ADD operation is conducted. In summary, we need 3k redundant bit-slices for ADD operation, among which (3k-2) bit-slices are used to cover the suspicious faulty bit-slices, one bit-slice for carry-in and one bit-slice for carry-out.

IV. ERROR CORRECTION CAPABILITIES

In this section, we discuss error correction capabilities of the proposed scheme. The fault model we assume here is a functional fault model. The failures affect a complete bit-slice or several adjacent bit-slices. Since the basic premise of our scheme is to carry out computation through fault-free bit-slices during the third computation step, one requires to know how many "useable" bit-slices are available in order to perform a correct computation.

By theorem 3 and theorem 4 it is easy to end up with the following conclusions.

THEOREM 5: The proposed error correction scheme can correct all errors in all bit-wise logical operations carried out by a circular logic unit when the failures are confined to k adjacent bit-slices.

Proof: By theorem 3 it is easy to figure out that the number of suspicious faulty bit-slices is (i+2k-1)-(i-k+1)=3k-1 when the failures are confined to k adjacent bit-slices. Since the word length of the logic unit is m+3k bit-slices, the number of fault-free bit-slices is equal to m+1. Furthermore, these fault-free bit-slices are adjacent logically due to the circular structure. Therefore, a bit-wise logical operation can be correctly carried out in the recomputation step.

Q.E.D.

THEOREM 6: The proposed error correction scheme can correct all errors in arithmetic operations in a circular bit-sliced ripple-carry adder or a circular bit-sliced carry-lookahead adder when the failures are confined to (k-1) adjacent bit-slices for k>1.

Proof: Applying theorem 4 we can again figure out that the number of suspicious faulty bit-slices is 3k-2 when the failures are confined to (k-1) adjacent bit-slices. Thus the fault-free bit-slices in a circular bit-sliced ripple-carry adder or a circular bit sliced carry-lookahead adder are adjacent from the circular point of view and the number is m+2, which is just long enough to carry out an effective recomputation for an m-bit-slice long ADD operation.

Q.E.D.

V. COMPUTATION ALGORITHM

Before presenting the computation step, we shall first figure out how many bit-slices should rotate for the recomputation.

Based on the structure we described earlier together with the theorems outlined in the last section, we come up with the following remarks:

(1) For all logical operations in a circular ALU, if the failures are confined to k adjacent bit-slices, and $FD_{m+1}(S^0,S^k)=i < m$. Then bit-slice i plus (2k-1) bit-slices on its left-hand side and (k-1) bit-slices on its right hand side might be faulty.

(2) A circular bit-sliced ripple-carry adder or a circular bit-sliced carry-lookahead adder is chosen in an ALU for arithmetic operations. Furthermore, if the failures are confined to (k-1) adjacent bit-slices. Then $FD_{m+1}(S^0,S^k)$ $=i<m+1$ implies that bit-slice i plus (2k-2) bit-slices on its left-hand side and (k-1) bit-slices on its right-hand side might be faulty.

Here, the terms "left-hand side" and "right-hand side" are both treated in a circular sense. With These two remarks, the following investigation is straightforward. First, let us consider the case for logical operations. If a mismatch occurs and $FD_{m+1}(S^0,S^k)=i<m$ is detected, then referring to remark (1) can locate the suspicious faulty bit-slices. It is clear that all these suspicious faulty bit-slices should be bypassed at the time that the final recomputation is carried out. In this case, one requires to move the least significant information bit into bit-slice (i+2k).

For the case of arithmetic operations, if a circular bit-sliced ripple-carry adder or a circular bit-sliced carry-lookahead adder is used. Moreover, if a mismatch is detected and $FD_{m+1}(S^0,S^k)=i<m+1$ is resulted. It is clear that (from remark (2)) bit-slices (i-k+1) to (i+2k-2) are likely faulty and should be skipped during the final computation in order to achieve an accurate result. Also notice that a fault-free bit-slice, in this case, bit-slice (i+2k-1), should be reserved to generate a correct carry-in signal, as such correct arithmetic operation can be resulted. Consequently, left rotation by (i+2k) bit-slices is required to conduct recomputation.

The required computation algorithm is given below:

Step 1: The input operands are fed into the ALU to conduct the desired logical/arithmetic operation.

Step 2: Rotate the input operands left by k bit-slices and then feed into the ALU. After the desired logical/arithmetic operation is carried out, rotate the result right by k bit-slices. If identical outputs are resulted, the result is correct; otherwise, a further recomputation (step3)is required.

Step 3: If $FD_{m+1}(S^0,S^k)=i$ is detected, rotate the input operands left by (i+2k) bit-slices, conduct the desired computation, and then rotate the result back by (i+2k) bit-slices. Now, the output is a correct result.

Notice that the input operands contains both information bit-slices (which is the effective messege part for actual computation) and redundant bit-slices. Initially, except bit-slice n+3k-1, all the contents of the redundant bit-slices are set to 0's. While the bit contents of bit-slice n+3k-1 are set to the value of carry-in.

VI. CONCLUSIONS

This paper presents a time-redundant technique for concurrent error correction in arithmetic and logic units. The proposed error correction scheme can detect errors and simultaneously locate (suspicious) faulty bit-slices during the first two computation steps. If errors are detected, recomputation is required to achieve accurate result. The basic premise is to bypass these "uncertain" bit-slices and carry out recomputation through those fault-free bit-slices. A circular bit-sliced ALU is proposed to realize this concept. Note that the term bit-slice we use here is not limited by its length. Indeed, a bit-slice may be one or more bits. However, since two extra redundant bit-slices are required in adders (and also other arithmetic units) for carry-in and carry-out, which in terms can be resolved perfectly by two single bits, it is not economical to choose a long bit-slice in our proposed system.

The fault model we used is that the physical failures are confined to a small area of the chip. Further, the precise nature of the faults does not have to be well understood, which is in contrast to many techniques that the traditional stuck-at fault is presumed. As assumming an exact fault model, such as stuck-at fault, is no longer appropriate for VLSI situation [13,14], our model is doubtlessly more suitable for the VLSI environment.

Implementation of this technique requires substantial hardware to support fast shift operations which may require more silicon area compare to other scheme such as TMR. But the application of this concept is not limited to simple units which are very common in use like ripple-carry adder and carry-lookahead adder. Indeed, this error correction scheme can apply to more complex arithmetic processors provided that the system can be partitioned into bit-slices physically or logically. Under such circumstances, the area occupied by the shifters is no longer a dominant factor of the entire circuit. As such our proposed technique may be more attractive than others because it can provide higher availability and reliability for the system.

REFERENCES

[1] D.P.Siewiorek, and R.S. Searz, The Theory and Practice of Reliable System Design, Digital Press, Bedford, Mass., 1982.

[2] D.A. Rennels, "Fault-tolerant computing—concepts and examples," IEEE Trans. Comput., Vol.C-33, pp.1116-1129, Dec. 1984.

[3] A. Avizenis, "Arithmetic codes: cost and effectiveness studies for application in digital systems design," IEEE Trans. Comput., Vol.C-20, pp.1322-1331, Nov. 1971.

[4] H. L. Garner, "Error codes for arithmetic operations," IEEE Trans. Electron. Comput., Vol.EC-15, pp.763-770, May 1966.

[5] A. Avizenis, G.C. Gilley, F.P. Mathur, D.A. Rennels, J.A. Rohr, and D.K. Rubin, "The STAR computer: an investigation of the theory and practice of fault-tolerant computer design," IEEE Trans. Comput., Vol.C-20, pp.1312-1322, Nov. 1971.

[6] J.F. Wakerly, Error Detecting Code, Self-Checking Circuits and Applications, North-Holland, New York, 1978.

[7] F.F. Sellers, M.Y. Hsiao, and L.W. Bearnson, Error Detecting Logic for Digital Computers, McGraw-Hill, New York, 1968.

[8] J.H. Patel, and L.Y. Fung, "Concurrent error detection in ALU's by recomputing with shifted operands," IEEE Trans. Comput., Vol.C-31, pp.589-595, July 1982.

[9] S. Laha, and J.H. Patel, "Error correction in arithmetic operations using time redundancy," Proc. 13th Annual Int'l. Symp. Fault-Tolerant Comput., pp.298-305, June 1983.

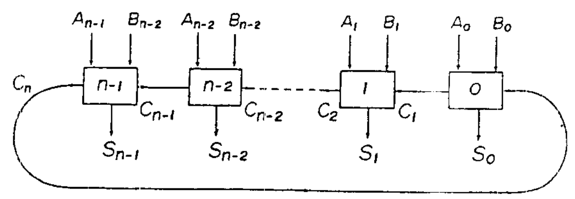
[10] S.E. Butner, "Triple time redundancy, fault-masking in byte-sliced systems," Tech. Rep. CSL TR211, Comput. Syst. Lab., Dept. of Elec. Engr., Standford Univ., Calif., Aug 1981.

[11] W.-T Chang, and J.H. Patel, "Concurrent error detection in iterative logic arrays," Proc. 14th Annual Int'l. Symp. Fault-Tolerant Comput., pp.298-305, June 1984.

[12] C.C. Wu, "Time-redundant fault-location in bit-sliced ALU's," IEEE Trans. Comput.,(To be published).

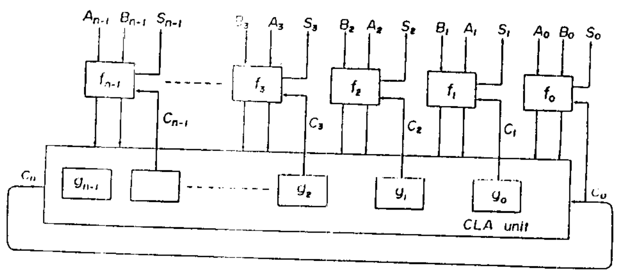
[13] J. Galiary, Y. Grouzet, and M. Vergniqult, "Physical versus logical fault models in MOS LSI circuits: impact on their testability," IEEE Trans. Comput., Vol.C-29, pp.527-531, June 1980.

[14] P. Banerjee, and J.A. Abraham, "Fault characterization of VLSI MOS circuits," Proc. IEEE Int'l. Conf. Circuits and Comput., pp.564-568, Sept. 1982.



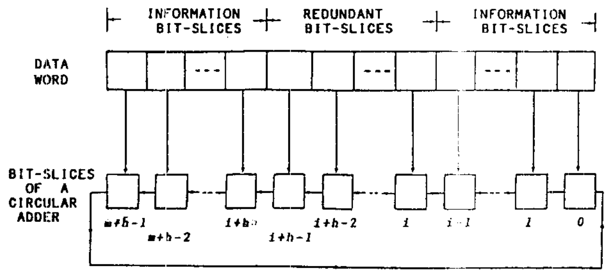
A_{n-1} B_{n-2} A_{n-2} B_{n-2} A_1 B_1 A_0 B_0
C_n
$n-1$
$n-2$
1
0
C_{n-1} C_{n-2} C_2 C_1
S_{n-1} S_{n-2} S_1 S_0

Figure 1. The physical and logical view of a bit-sliced circular ripple-carry adder with r=1.



A_{n-1} B_{n-1} S_{n-1} B_3 A_3 S_3 B_2 A_2 S_2 B_1 A_1 S_1 A_0 B_0 S_0
f_{n-1} f_3 f_2 f_1 f_0
C_{n-1} C_3 C_2 C_1
C_n C_0
g_{n-1} g_2 g_1 g_0
CLA unit

Figure 2. The logical view of a circular carry-lookahead adder with r=1.



INFORMATION BIT-SLICES
REDUNDANT BIT-SLICES
INFORMATION BIT-SLICES
DATA WORD
BIT-SLICES OF A CIRCULAR ADDER
$m+h-1$ $m+h-2$ $i+h$ $i+h-1$ $i+h-2$ i $i-1$ 1 0

Figure 3. Bit-slices of a circular adder and the contents of the associated data word.