Fault Identification in Reconfigurable Carry Lookahead Adders Targeting Nanoelectronic Fabrics *

Wenjing Rao
UC San Diego
CSE Department
wrao@cs.ucsd.edu

Alex Orailoglu
UC San Diego
CSE Department
alex@cs.ucsd.edu

Ramesh Karri
Polytechnic Univ
ECE Department
rkarri@poly.edu

ABSTRACT

Online repair through reconfiguration is a particularly advantageous approach in the nanoelectronic environment since reconfigurability is naturally supported by the devices. However, precise identification of faulty locations is of critical importance for fine-grain repairs.

A CLA is mainly composed of: (1) carry generation blocks and (2) g, p signal generation blocks. In this paper we propose two schemes for fault identification in these two parts correspondingly. For carry generation blocks, an inherently redundant computation path is exploited to identify the faulty block with high precision. As a time redundancy approach, recomputation with rotated operands (RERO) has been utilized in online fault detection for CLA's [13]. For g, p generation blocks, we exploit the RERO scheme to achieve precise fault identification.

A comprehensive analysis is provided for the aliasing in the proposed fault identification approach. It is shown that both the amount of repair hardware overhead and the fault coverage loss for the proposed scheme are very low. Overall, the proposed scheme can perform fast and precise identification of faults in the CLA components with low area overhead, thus facilitating the development of powerful and efficient fault tolerance schemes through online repair for nanoelectronic systems.

1. INTRODUCTION

As CMOS is facing its physical limits in shrinking below today's 90nm scale, nano devices have been actively researched and proposed as the basis for the next generation electronic systems [1, 2]. However, due to their small scale, nanoelectronics are projected to be highly unreliable [2, 3]. Consequently, aggressive fault tolerance schemes are necessary.

One of the significant advantages of nanoelectronic systems is the abundant hardware resources, which support not only massive parallelism and high performance, but also the use of hardware redundancy for fault tolerance purposes. Typical hardware redundancy based fault tolerance schemes include fault masking and online repair schemes.

Fault masking approaches require multiple redundant copies of a computation and tolerate online faults through a majority vote. It is shown that the application of simple fault-masking schemes, such as NMR and N-MUX, necessitates immense hardware redundancies to tolerate the large fault rates in a nanoelectronic system [4, 5].

Online repair schemes, on the other hand, depend on the online detection and identification of a fault, leading to a subsequent repair process to ensure correctness. In comparison to the fault masking

schemes, online repair schemes require significantly less hardware overhead [4], especially for regular systems where replacement of faulty components can be performed with common backup units [6, 7]. In fact, as bottom-up fabrication, the expected mode of future nano circuit assembly, can only produce regular structures [8, 9], a reconfiguration capability is necessary in any nanoelectronic system for the purpose of functionality formation in a post-fabrication stage, thus supporting repair online as well [10].

The repair process can be performed either after online detection of a fault or after a field test carried out once in a while in the system. Such online repair schemes, however, hinge on the fault identification granularity. If a fault can be precisely localized within a small component, then the repair procedure can replace the small faulty component. A coarse-grain fault identification results in the replacement of a big chunk of hardware. Therefore, fine-grain fault identification is crucial to the effectiveness of online repair based fault tolerance.

It is well known that an adder constitutes the fundamental arithmetic building block. Since fault occurrence in the current CMOS systems is extremely rare, existing fault tolerance schemes for adders can typically afford to utilize coding based approaches or fault masking approaches [11, 12]. Related previous work also includes online detection of faults in adders [13, 14, 15, 16, 17]; yet online identification of the faulty locations has not been tackled in CMOS adders.

Since the fault rate in the nanoelectronic environment is dramatically higher, conventional fault tolerance schemes either fail to provide sufficient fault tolerance, or consume prohibitively expensive resources for doing so. Based on the reconfigurability of nanoelectronics, online repair based schemes can provide powerful fault tolerance capabilities with less hardware overhead, thus positioning them as a promising choice in fault tolerant adder design.

In adder designs, the various candidates provide multiple trade-off points between the main optimization criteria of area and delay. In the particular environment of nanoelectronics, due to the new characteristics such as the abundant hardware and the pervasive unreliability, a re-evaluation of the various existing adder design candidates under the set of new challenges and benefits is necessitated.

Because of the abundance in hardware and massive parallelism supported by nano devices, the area constraint diminishes in relative importance in the nanoelectronic environment. Therefore, parallel adders, providing an optimal performance with $O(\log n)$ latency, are promising in that the carries are computed in parallel so as to avoid the performance loss due to carry rippling. Among the parallel adders, carry lookahead adders (CLA) show particular advantages for the nanoelectronic environment with its regular structure, which not only fits for the regular structure based nanofabric,

*The work of the first two authors is supported in part by NSF Grant 0082325.

Figure 1: Hierarchical implementation of a 64-bit CLA

but also support efficient online repair based fault tolerance. Since a CLA is constructed with regular blocks with inputs disjoint across the blocks, precise component-level fault identification within the adder can be approached with low hardware overhead.

An aliasing analysis is provided at the end of this paper for the proposed fault identification scheme. Through the analysis of the repair hardware overhead and the fault coverage loss ratio caused by fault identification aliasing in the proposed scheme, it is shown that high fault identification resolution can be achieved at a significantly reduced amount of repair hardware with small fault coverage loss.

2. THE CARRY LOOKAHEAD ADDER

In order to avoid the delay of the rippling carries, a carry-lookahead adder (CLA) [18] uses the carry-in bit and the generate (g) and propagate (p) signals to predict the most significant carry bits. Since the prediction hardware of carries at the most significant bits becomes exceedingly expensive when the width of a CLA is large, the CLA is typically organized hierarchically with multiple levels.

Figure 1 shows a typical 64-bit hierarchical CLA composed of 3 levels. The g, p signals of various blocks are generated hierarchically: initially, at the lowest level, the g, p signals of each bit are calculated; then, at the second level, the g, p signals of the 4-bit blocks are generated using the one-bit g, p signals; finally, the g, p signals of the 16-bit block are generated using the g, p signals of the four 4-bit blocks.

The carry bits are also calculated in 3 stages from the lookahead carry generators (LCG). Initially, the highest level LCG uses the lowest carry C_0 and the 4 highest level block g, p signals to generate the carry-in bits, C_{16} , C_{32} , and C_{48} , for the lower level LCGs. Then these carry-in bits and C_0 are used by the lower level LCGs to further generate 12 carry-in bits for the lowest level LCGs. In the third stage, the 16 LCGs at the lowest level produce the remaining 48 carry-in signals, thus completing the carry generation for all the bits.

As is shown in Figure 2, a 4-bit LCG takes as inputs the carry-in of the least significant bit and the g, p signals generated from 4 lower-level LCG blocks. The outputs of the 4-bit LCG include

Figure 2: A 4-bit Lookahead Carry Generator (LCG) circuit

Figure 3: Carry generation with internal redundancy

block g, p signals for the current level and the carry-in bits for the 4 lower-level LCG blocks.

3. FAULT IDENTIFICATION IN CLA

Fault identification relies on the differentiability among the manifestations of various faults, known as the *fault syndromes*. When distinct faulty components produce identical fault syndromes, *fault aliasing* occurs, resulting in an *ambiguity set* requiring consequently the replacement of all the possibly faulty components in the ambiguity set. Therefore, fault aliasing generally leads to extra hardware overhead in the repair process. Under a multiple fault occurrence scenario, aliasing may even lead to incorrect identification of faulty components, resulting in loss of fault coverage.

A CLA contains a number of LCGs as its constituent components. All the LCGs are identical in their internal structure. Owing to their regularity, reconfiguration can be easily supported by replacing faulty LCGs with spare ones. Their regularity also helps construct low-cost, high-resolution fault identification schemes.

As is shown in figure 2, an LCG is composed of a g, p signal generation block and a carry-bit generation block. For the carry generation blocks, the existence of inherent redundancy enables us to perform precise fault identification. For the g, p signal generation blocks, we take advantage of the characteristic that different blocks use disjoint input signals and utilize a *REcomputing with Rotated Operands* (RERO) scheme to identify faulty components.

3.1 Carry generation block fault identification

In a hierarchical CLA, the carry-in to a LCG block is generated by the higher level LCG for performance purposes. However, the same carry-in signal can also be generated through another path at the same level with a constant delay.

Consider the 4-bit LCG block shown in figure 3. The three carry generation blocks, C_3 , C_2 , and C_1 , independently generate the carry-in bits c_3 , c_2 , and c_1 , for the lower level LCG numbered 4, 3 and 2, respectively. In fact, another copy of c_3 (indicated as c_3' in figure 3), can be generated with a small latency, from the carry-in of c_2 and the g, p signals of the lower LCG block 3. Specifically, c_3' is generated in the same way that c_{out} is generated in the higher level LCG by using $c_{out} = g + p \cdot c_0$.

Since two copies of the same signal are generated using disjoint hardware blocks, faults can be precisely identified by comparing these carry-bit pairs. Figure 4 shows the fault identification process in the carry generation blocks. For instance, carry bits c_3' and c_2 are generated using block C_2 . If block C_2 is faulty, both the carry bit pairs of (c_3, c_3') and (c_2, c_2') conflict. The conflicting results on both sets of computation implicate c_2 as the faulty component. The syndromes of all the faulty components as well as the combinations of multiple faulty components are shown in the syndrome in figure 4.

It is worth noticing that all single-faults, double-faults and triple-faults can be precisely identified without aliasing since the corresponding syndromes are all distinct. Basically, each hardware block participates in a combination of comparisons, thus forming a *participation signature* vector. In the carry generation process, all the signature vectors of the hardware blocks are linearly independent. Specifically, C_3 participates in 1-out-of-3 comparisons and

Figure 4: Fault identification by redundant carry comparison

can be represented by a participation vector of (1,0,0). Similarly, the participation vector for $C2$ is (1,1,0) and for $C1$ is (0,1,1).

A fault syndrome observed in this case is a vector formed by the results of the three comparisons, which is mathematically the linear combination of the participation vectors of the faulty blocks. Since all the participation vectors constitute the basis of a vector space, any fault syndrome vector can be uniquely represented by a linear combination of all the participation vectors. Therefore, all the fault syndromes are distinct and aliasing is prevented inherently.

Overall, the above analysis shows how the inherent redundancy in CLA carry generation can be exploited to provide precise fault identification resolution. Since the redundant computations reuse the existing data paths in a CLA, it entails very little hardware overhead. The additional hardware for each LCG for fault identification in the carry generation part is basically a few exclusive-or gates plus the logic implementing the syndrome table. The hardware overhead can be traded-off for fault identification precision by implementing a reduced subset of the syndrome table for the cases when it is not necessary to consider multiple faults.

3.2 Fault identification in g,p blocks with RERO

The g,p generation block in a CLA does not bear an inherent redundancy as the carry generation block. Therefore, redundancy needs to be explicitly added for fault identification. Similar to the well-known recompute with shifted operands (RESO) approach, recompute with rotated operand (RERO) is a time redundancy based technique to detect faults online [19, 13, 20].

Typically, RERO for CLA requires significant hardware even for online fault detection since it is constrained by the dependencies among the input bits due to the carry propagation [13]. In this paper we achieve the precise identification of the faulty blocks among the g,p generation components by exploiting the absence of dependencies among inputs of the g,p generation blocks.

When components are performing the same type of computation with independent inputs, the faulty component can be identified by recomputing using a different permutation of the inputs and comparing the results. A simple form of permutation is rotation. In a CLA, the g,p generation blocks at the same level use disjoint input signals; therefore, a rotated recomputation can be used to perform fault identification. In figure 5, we show two possible RERO scenarios. Numbers 1 to 4 correspond to computations while letters A to D denote the components. Each computation is carried out twice by distinct components at different time slots and the results compared. The outcomes of the comparisons are the fault syndromes.

RERO using a 2-bit rotate is shown in figure 5a. Computation 2 is executed by component B in time slot $t1$ and is re-computed by component D in time slot $t2$. Similarly, computation 4 is executed by D in $t1$ and by B in $t2$. When component B is faulty, ($=$, $\neq$, $=$, $\neq$) can be seen to be the fault syndrome, which is identical to the syndrome of a faulty D . This results in a fault aliasing effect between B and D . Similar aliasing exists between the faults in A and C as well.

Figure 5: Fault aliasing in RERO by rotation amount

In order to precisely identify faulty components under a single-fault assumption, in an n -bit computation, the rotation amount r should be chosen such that $GCD(n, s) = 1$. As is shown in figure 5b, for RERO with 1-bit rotate, each faulty component exhibits a distinct syndrome, thus avoiding aliasing.

3.2.1 Generalizing RERO to hierarchical CLA's

If only a single level is considered, the selection of the rotation amount in RERO only needs to consider the aliasing criterion. However, in a hierarchical CLA, the rotation amount for RERO needs to consider the g,p blocks at multiple levels. During RERO the original and the duplicate computations need to use the same set of computations with different permutations over the components. A higher level g,p block takes inputs from 4 consecutive blocks at the lower level. Therefore, in order to guarantee that the same set of computations is applied at this higher level during RERO, the rotation amount should be chosen such that the same subset of consecutive inputs is not split across the two computations.

In a 3 level 64-bit CLA, the first level consists of 16 g,p generator blocks, each processing 4 bits from the input operands. The second level consists of 4 g,p generator blocks, each taking the output of 4 consecutive first level blocks. The third level has only 1 block, taking the output from the 4 blocks in the second level. RERO can identify a faulty component in the second level using a one-bit rotate. This in turn corresponds to RERO with a 4-bit rotate in the first level.

In general, in a hierarchical CLA, faulty blocks at all the levels except for the single block at the highest level¹ can be precisely identified by setting the rotation amount top down starting from the second-to-highest level. Since a higher level g,p block uses 4 lower level g,p blocks, a 1-bit rotate at the second-to-highest level translates into an $n/4$ -bit rotate at the lowest level where n is the number of bits in the operands for the CLA.

3.2.2 Half fault manifestation & field test vector selection

Faulty units can be precisely identified by RERO when a faulty component manifests its fault in both the computation and the re-computation. However, since a component executes the two different computations in two different time slots, it is very likely that the fault manifests in only one of the two computations. We denote such a partial fault manifestation as *Half Fault Manifestation* (HFM). Typically, the manifestation of a faulty component depends on the input to the computation. At run time, transient faults and single event upsets can also result in HFM's. HFM precludes precise fault identification due to the aliasing of faulty syndromes. Figure 6 shows aliasing due to half fault manifestation.

¹Since the highest level of a CLA is composed of a single block only, RERO is neither applicable nor necessary for the fault identification of this specific block. We recommend the use of simple fault masking schemes for the fault tolerance of this block.

Figure 6: Half fault manifestation (HFM) of component B

During online fault identification, the HFM caused by different inputs of the two computations leads to an indistinguishable fault syndrome between the two possibly faulty components. Therefore, the subsequent repair process needs to use double the amount of backup hardware to replace both faulty component candidates. During field test, however, test vectors are pre-generated and stored to be applied once in a while during system execution, to identify faulty components and perform repair. Therefore, when RERO is used in field test, test vectors can be selected in such a way that HFM's are avoided.

For an r -bit RERO on an n -bit computation, the input bits should be partitioned into equal-sized groups of size $GCD(n, r)$. Field test should guarantee the application of identical inputs for every group for each test vector. Within each group, test vectors can be generated deterministically or pseudo-randomly using BIST. Applying RERO during field test avoids storing the test responses. This also significantly reduces the amount of fault identification hardware.

3.2.3 Fault identification hardware

Figure 7 shows the fault identification hardware for 1-bit RERO on a 4-bit computation. An exclusive-or gate is required for each bit pair to be compared. The fault syndrome is generated by the corresponding pair of comparison results; thus the XOR output pair is connected to an AND gate to indicate the faulty bit. In addition, we require two registers, one for the results of the basic computation and the other for the recomputation result. In a 3 level 64-bit hierarchical CLA, the proposed scheme can check the g, p signal generation of the 20 ($= 16+4$) blocks in the two low levels. This translates into two 20-bit registers, 20 exclusive-or gates and 20 AND gates for the syndrome detection circuitry shown in Figure 7.

4. FAULT IDENTIFICATION ANALYSIS

Let us consider $n/4$ -bit RERO without loss of generality, since aliasing introduced by single-faulty-block and two-faulty-blocks scenarios exists only within the four components that form a rotation. Figure 8 shows $n/4$ -bit RERO among four components and will serve as a template for analysis.

4.1 Single-faulty-block scenario

Table 1 shows the faulty syndrome analysis under the single-faulty-block scenario. The computation indices as well as the components refer to the ones in figure 8. In the table, full fault manifestation is represented in upper-case letters. HFM is represented

Figure 7: g, p signal generation block syndrome decision HW

Figure 8: $n/4$ shifted RERO

in lower-case letters, followed by a number indicating the specific time slot of the fault effect manifestation.

For instance, the second column shows that, when computation 1 and 2 conflict under the single-faulty-block scenario, component A is the only faulty candidate and its fault is manifested fully in both computations. Therefore, the faulty component is precisely identified. On the other hand, the last column shows that, when a conflict exists only in computation 4, either component D shows HFM in the first time slot, or component C shows HFM in the second time slot. Therefore, the ambiguity set includes both $d1$ and $c2$. In this case, the repair costs double the amount of hardware since C and D are both replaced due to the ambiguity.

Essentially, HFM results in an enlarged ambiguity set, thus requiring extra hardware to perform repair. In this case, double the amount of hardware is required due to HFM. There is however no loss of fault coverage in the aliasing cases, since a faulty component is always included in the ambiguity set.

Suppose the probability of a fault manifesting on any input is p ; then the probability of full fault manifestation is p^2 and HFM is $2p(1-p)$. The expected amount of hardware to perform the repair is thus $\frac{p^2 \times 1 + 2p(1-p) \times 2}{p^2 + 2p(1-p)} = \frac{4-3p}{2-p}$ the amount of repair hardware with full resolution.

4.2 Two-faulty-blocks scenario

Tables 2 and 3 show the fault syndrome analysis for double-fault scenario among the 4 components. These tables only show the new aliasing introduced by the double-fault manifestation in the 4 components. Otherwise, the effect of double faults can be simply analyzed by treating them as two single faults.

Table 2 shows how aliasing due to the ambiguity in the faulty component candidate set can result in additional hardware to repair. For instance, as shown in the second column, the full manifestation of faults in A and B results in conflicting values in computations 1 and 3 only, since the common manifestation of faults of A and B in computation 2 cancels the fault effect. However, when A has an HFM in time slot $t1$ and B has an HFM in time slot $t2$, it results in the same syndrome. In fact, all the combinations listed in a column result in the identical faulty syndrome. In all the cases shown in table 2, all 4 components need to be replaced.

Table 3 shows that some of the fault syndromes in the presence of two faults are identical to the fault syndromes in the presence of a single fault. These syndromes are in turn interpreted as single faults resulting in a consequent loss in fault coverage.

Based on the type of fault manifestation, namely full and half, there are three scenarios: (i) both faults fully manifested, (ii) both

computations showing conflict	1,2	2,3	3,4	4,1	1	2	3	4
ambiguity set	A	B	C	D	a1 d2	b1 a2	c1 b2	d1 c2
replacing:	A	B	C	D	(AD)	(AB)	(BC)	(CD)

Table 1: Fault aliasing analysis for single-fault assumption

computations showing conflict	1,3	2,4	1,2,3	2,3,4	1,3,4	1,2,4	1,2,3,4
ambiguity set	A,B C,D a1,c1 d2,b2 a1,b2 d2,c1	B,C A,D b1,d1 b2,c2 b1,c2 b2,d1	A,c1 A,b2 B,a1 B,d2	B,d1 B,c2 C,b1 C,a2	C,a1 C,d2 D,c1 D,b2	A,d1 A,c2 D,b1 D,a2	A,C B,D
replacing:	(ABCD)						

Table 2: Aliasing due to ambiguity in the faulty component candidate set doubles the hardware for repair

faults half manifested and (iii) one fully while the other half manifested. Following is a detailed analysis for the required hardware and loss in fault coverage for these three cases.

4.2.1 Both faults fully manifested

As can be seen in table 2, there are altogether 6 possible cases of two fully manifested faults: AB, CD, BC, AD, AC and BD, which cost double the amount of hardware to repair. Considering the case of an $(n/4)$ -bit RERO, there are altogether $(n \times (n-1))/2$ possible combinations of two fully manifested faults. Among these combinations $6 \times n/4 = 3n/2$ cases require double the amount of hardware to repair. Therefore, the expected amount of hardware required for repair is $\frac{n(n-1)/2 + 3n/2}{n(n-1)/2} = \frac{n+2}{n-1}$.

Table 3 also shows that no fault coverage loss is incurred when two fully manifested faults are considered.

4.2.2 One fully and one half manifested fault

There are altogether 16 cases of one fault fully manifested with the other half manifested, as is shown in columns 4-7 of table 2, which result in double the amount of repair hardware. From table 3 another 8 cases are shown in the rightmost 4 columns, which result in the loss of half the fault coverage, i.e., one of the two faults is not identifiable. These disjoint 16 cases and 8 cases constitute all the possible 24 possible combinations of *one fully and one half manifested fault* among among components A, B, C and D. An extra 50% amount of repair hardware is required in this case since the HFM demands a replacement of two components and the full fault manifestation uses only one.

For an $(n/4)$ -bit RERO, there are in total $n(n-1) \times 2$ possible combinations for this fault manifestation case, among which $16 \times n/4 = 4n$ of them result in double the repair hardware, $8 \times n/4 = 2n$ of them result in half fault coverage loss, and the remaining ones require 50% extra amount of repair hardware with no loss in fault coverage. Therefore, the expected hardware requirement in this case is $\frac{(2n(n-1) - 24n/4) \times 1.5 + 2 \times 16n/4}{2n(n-1) - 8n/4} = \frac{3n-4}{2n-4}$ the amount of required repair hardware with full resolution. The expected rate of

computation showing conflict	none	1,2	2,3	3,4	4,1	1	2	3	4
ambiguity set	a1d2 b1a2 c1b2 d1c2	a1b1 d2a2 d2b1	b1c1 a2b2 a2c1	c1d1 b2c2 b2d1	a1d1 d2c2 a1c2	Ab1 Dc2	Bc1 Ad2	Cd1 Ba2	Da1 Cb2
replacing:	∅	A	B	C	D	(AD)	(AB)	(BC)	(CD)
fault coverage	1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
loss ratio	1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	1	1	1	1	1				
	1								

Table 3: Aliasing & fault coverage loss by double-fault

fault manifest	probability	HW amount for repair			fault coverage loss rate		
		Exp	$n = 4$	$n = 16$	Exp	$n = 4$	$n = 16$
full/full	p^4	$\frac{n+2}{n-1}$	2	6/5	0	0	0
full/half	$2p^3(1-p)$	$\frac{3n-4}{2n-4}$	2	11/7	$\frac{1}{2n-2}$	1/6	1/30
half/half	$p^2(1-p)^2$	2	2	2	$\frac{3}{2n-2}$	1/2	1/10

Table 4: HW / fault coverage loss for $n = 4$ and $n = 16$

fault coverage loss is $\frac{2n \times 0.5}{2n(n-1)} = \frac{1}{2n-2}$.

4.2.3 Both faults half manifested

Similar to the analysis in the last subsection, we can observe from tables 2 and 3 that 8 cases of both faults being half manifested result in doubling the amount of repair hardware and 16 cases result in loss of fault coverage. Specifically, among the 16 cases, as is shown in columns 2-6 of table 3, 8 cases result in complete loss of fault coverage and the other 8 cases result in half fault coverage loss. The average fault coverage loss ratio in these 16 cases is 0.75.

When two faults are both half manifested, ambiguity is always incurred regardless of the positioning of the faults. The expected amount of repair hardware under this specific fault manifestation is always double the amount of required repair hardware under full resolution and the expected fault coverage loss ratio is $\frac{0.75 \times 16n/4}{2n(n-1)} = \frac{3}{2n-2}$.

4.2.4 Fault identification analysis for a 64-bit CLA

In the case of a 3-level 64-bit CLA, the fault identification for the lower two levels of g, p generation blocks consists of two RERO, performing $n/4$ -bit rotation. Table 4 shows the required repair hardware amount as well as fault coverage loss in the two RERO cases of $n = 4$ and $n = 16$.

From table 4 we can deduce the following expressions under the two-faulty-block scenario:

$n=4$: HW amount = 2

$n=4$: fault coverage loss ratio = $\frac{2p^3(1-p)(1/6) + p^2(1-p)^2(1/2)}{p^4 + 2p^3(1-p) + p^2(1-p)^2}$
 $= (1/p^2)((1/6)p^4 - (2/3)p^3 + (1/2)p^2)$

$n=16$: HW amount = $\frac{(5/6)p^4 + 2p^3(1-p)(11/7) + 2p^2(1-p)^2}{p^4 + 2p^3(1-p) + p^2(1-p)^2}$
 $= (1/p^2)((2/35)p^4 - (6/7)p^3 + 2p^2)$

$n=16$: fault coverage loss ratio = $\frac{2p^3(1-p)(1/30) + p^2(1-p)^2(1/10)}{p^4 + 2p^3(1-p) + p^2(1-p)^2}$
 $= (1/p^2)((1/30)p^4 - (2/15)p^3 + (1/10)p^2)$

Figures 9(a) and 9(b) respectively plot the repair hardware and fault coverage loss ratios for $n=4$ and $n=16$ as a function of the fault manifestation probability, p . It can be seen from figure 9(a) that when $n=4$, under the scenario of two-faulty-blocks, double the amount of hardware is always required for repair, because there are 4 components in total and any fault position combination leads to aliasing. When $n=16$, aliasing decreases with increasing p , i.e., faults tend to fully manifest. Under the single-fault assumption, the required hardware reaches the lower bound of 1 when $p=1$. In other words, aliasing is entirely caused by HFM. For the double-fault case, however, even under full fault manifestation ($p=1$), aliasing still exists depending on the positions of the two faults. This translates into 20% extra repair hardware.

The loss in fault coverage is due to a combination of HFM and the relative positions of the two faults. It can be seen from figure 9(b) that under the single-fault assumption, there is no loss in fault coverage. Under the double-fault assumption, there is no loss in fault coverage when there are no HFM's ($p=1$).

Overall, when faults are fully manifested, precise identification can be achieved and the faults can be repaired with almost zero

Figure 9: HW & fault loss curves

hardware overhead. Consequently, for the proposed RERO in field test, fine-grain online fault identification and precise repair can be achieved. For the online identification of the faults that manifest rarely, the resolution is less precise due to possible aliasing, but still within acceptable upper bounds for various n . In the worst case, double the amount of hardware is required to perform a repair, which is still quite efficient since the components are of fine-grain size in nanoelectronic CLA design.

When n is small the fault loss ratio is higher, costing relatively more hardware to repair. Effectiveness of fault identification, both in terms of repair hardware and loss in the fault coverage, is better for larger n . In fact, when n is small, only a few components are involved; thus the likelihood of having multiple faults occurring among the components is also small. A large n naturally demands more powerful and precise fault identification since the likelihood of faults occurring is much higher. Particularly, in the nanoelectronic environment where massive parallelism is supported and fault rates are dramatically high, the proposed scheme is promising due to its capability of providing powerful fault identification with low repair hardware overhead for large CLA's.

5. CONCLUSIONS

We propose in this paper a fault identification approach for nanoelectronic carry lookahead adders (CLA). The proposed approach is shown to be the crucial part in performing any reconfiguration-based component-level online repair scheme for the fundamental arithmetic elements. Such fault tolerance schemes are advantageous in nanoelectronics because: 1) reconfigurability is a required characteristic in nanofabric, and 2) it is very efficient due to the utilization of the least amount of hardware to perform the repair.

We deal with the two main parts of a CLA by proposing two distinct approaches. By utilizing time redundancy for recomputation exploiting the inherent redundancy, the hardware overhead to implement these approaches is quite small. The fault identification capability of the carry generation blocks is proven to be perfect with zero-aliasing. For the g, p signal generation blocks, a thorough analysis is provided to show that very high resolution can be achieved with low aliasing and small loss of fault coverage. The proposed approach sets up the basis upon which efficient online reconfiguration repair can be applied for the basic arithmetic blocks to provide powerful fault tolerance capability for nanoelectronic systems.

6. REFERENCES

- [1] ITRS, *International Technology Roadmap for Semiconductors Emerging Research Devices*, 2004.
- [2] European Commission, *Technology Roadmap for Nanoelectronics*, 2001.
- [3] P. Beckett and A. Jennings, "Towards Nanocomputer Architecture", in *Asia-Pacific Computer System Architecture Conference*, pp. 141–150, 2002.
- [4] K. Nikolic, A. Sadek and M. Forshaw, "Architectures for Reliable Computing with Unreliable Nanodevices", in *IEEE-NANO*, pp. 254–259, 2001.
- [5] J. Han, J. Gao, Y. Qi, P. Jonker and J. A. B. Fortes, "Toward Hardware-Redundant, Fault-Tolerant Logic for Nanoelectronics", *IEEE Design and Test of Computers*, vol. 22, n. 4, pp. 328–339, July-August 2005.
- [6] I. Kim, Y. Zorian, G. Komoriya, H. Pham, F. P. Higgins and J. L. Lewandowski, "Built in Self Repair for Embedded High Density SRAM", in *ITC*, pp. 1112–1119, 1998.
- [7] C. H. Lee, M. A. Perkowski, D. V. Hall and D. S. Jun, "Self-Repairable EPLDs II: Advanced Self-Repairing Methodology", in *IEEE Congress on Evolutionary Computation*, pp. 616–623, 2001.
- [8] R. Rubin and A. DeHon, "Nanowire-based Sublithographic Programmable Logic Arrays", in *FPGA*, pp. 123–132, 2004.
- [9] D. B. Strukov and K. K. Likharev, "CMOL FPGA: A Reconfigurable Architecture for Hybrid Digital Circuits with Two-terminal Nanodevices", *Nanotechnology*, vol. 16, pp. 888–900, Apr 2005.
- [10] S. C. Goldstein, M. Budiu, M. Mishra and G. Venkataramani, "Reconfigurable Computing and Electronic Nanotechnology", in *ASAP*, pp. 132–143, 2003.
- [11] M. Alderighi, S. D'Angelo, G. R. Sechi and C. Metra, "Novel Fault-Tolerant Adder Design for FPGA-Based Systems", in *IOLTW*, pp. 54–58, 2001.
- [12] E. G. Sogomonyan, V. Ocheretnij and M. Gossel, "A New Code-Disjoint Sum-Bit Duplicated Carry Look-Ahead Adder for Parity Codes", in *ATS*, pp. 365–370, 2001.
- [13] J. Li and E. E. Swartzlander Jr, "Concurrent Error Detection in ALUs by Recomputing with Rotated Operands", in *DFT*, pp. 109–116, 1992.
- [14] B. K. Kumar and P. K. Lala, "On-line Detection of Faults in Carry-Select Adders", in *ITC*, pp. 912–918, 2003.
- [15] A. Walker and P. K. Lala, "On-line Error Detectable Carry-Free Adder Design", in *DFT*, pp. 66–71, 2001.
- [16] J. G. G. Langdon and C. K. Tang, "Concurrent Error Detection for Group Look-ahead Binary Adders", *IBM J. Res. Develop*, pp. 563–573, Sep 1970.
- [17] 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*, pp. 208–215, February 1988.
- [18] B. Parhami, *Computer Arithmetic Algorithms and Hardware Designs*, Oxford University Press, 2000.
- [19] J. H. Patel and L. Y. Fung, "Concurrent Error Detection in ALUs by Recomputing with Shifted Operands", *IEEE Transactions on Computers*, vol. 31, pp. 589–592, December 1982.
- [20] A. Majumdar, C. S. Raghavendra and M. A. Breuer, "Fault Tolerance in Linear Systolic Arrays Using Time Redundancy", *IEEE Transactions on Computers*, vol. 39, pp. 269–276, February 1990.
- [21] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1990.