Concurrent Error Detectable Carry Select Adder with Easy Testability

Nobutaka Kito, *Member, IEEE,*
and Naofumi Takagi, *Senior Member, IEEE*

Abstract—A concurrent error detectable adder with easy testability is proposed. The proposed adder is based on a multi-block carry select adder. Any erroneous output of the adder caused by a fault modeled as a single stuck-at fault can be detected by comparing the predicted parity of the sum with the parity of the sum, i.e., the XORed value of the sum bits, and comparing the duplicated carry outputs. The adder is also testable with only 10 input patterns under single stuck-at fault model. This property eases detection of a fault before the occurrence of a second fault. Both the concurrent error detectability to detect erroneous results and the easy testability to find a fault during operation are important for realizing reliable systems. Both the concurrent error detectability and the easy testability of the proposed adder are proven. A 32-bit adder has been designed. Its hardware overhead is about 70%. Its concurrent error detectability and 100% test coverage through the 10 patterns has been confirmed by fault simulation.

Index Terms—Concurrent error detection, carry select adder, design for testability.

1 Introduction

AS the process shrink of integrated circuits advances and the integration density increases, reliability of integrated circuits in field becomes an issue [1], [2], [3]. For critical systems such as server class computers and embedded systems, concurrent detection of errors is important.

Addition is the most basic arithmetic operation, and a lot of adders are used in VLSIs. Reliable adders are important for realizing reliable systems. Adders with concurrent error detectability were proposed [4], [5], [6], [7]. In [4] and [7], parallel prefix adders with concurrent error detectability were shown. Any erroneous output of these adders can be detected based on parity prediction. Namely, they produce a predicted parity of the result, and any erroneous output can be detected by comparing the predicted parity with the parity of the result, i.e., the XORed value of the resultant sum bits. In [5] and [6], carry select adders with concurrent error detectability were shown. Any erroneous output can be detected by comparing two internal values in the carry select adders.

In these previously proposed concurrent error detectable adders, any erroneous output caused by a single fault can be detected. However, an erroneous output caused by multiple faults is not guaranteed to be detected. If a second fault has occurred before the first fault is found through detection of an erroneous output, the concurrent error detectability can be lost. For reliable adders, it is crucial that the first fault is noticed before a second fault occurs. Therefore, it is important for a concurrent error detectable adder that it also has easy testability so that a fault in it can be found easily at maintenance or at reboot, or even during operation. In this brief, hard errors cause by degradation [2], [8], such as electromigration [9], [10] and stress migration [11], [12], are considered. The VLSI chips are assumed to have no fault before a fault occurs and faults appear gradually. Thus, a fault modeled as a stuck-at fault is considered. The same fault model is considered in [4], [5], [6].

In this brief, we propose a concurrent error detectable adder which also has easy testability. Although several easily testable adders were proposed [13], [14], [15], [16], [17], [18], [19], none of them has concurrent error detectability. In those researches, C-testability, i.e., a property to be testable with a test set of constant size regardless of bit-width of a circuit, has been considered as the most powerful property. The adder to be proposed is C-testable. This property is also useful for testing during operation.

The adder to be proposed is based on a multi-block carry select adder. The multi-block carry select adder uses the idea of calculating the sum result by selecting a result from two candidates, one assuming 0 as the carry input and the other assuming 1 as the carry input, according to the actual value of the carry input [20]. The adder to be proposed has concurrent error detectability based on parity prediction. Any erroneous output of the adder caused by a fault modeled as a single stuck-at fault can be detected by comparing its predicted parity output with the parity of its sum result and comparing its duplicated carry outputs. The adder is also testable with only 10 patterns under single stuck-at fault model. Both the concurrent error detectability and the easy testability are proven.

We have designed a 32-bit adder and showed that its hardware overhead is about 70%. We have confirmed its concurrent error detectability by fault simulation with random patterns. We have also confirmed the 100% test coverage through the 10 input patterns by fault simulation.

This brief is organized as follows. In the next section, a multi-block carry select adder is briefly reviewed. In Section 3, a concurrent error detectable adder with easy testability is proposed. Its concurrent error detectability and easy testability are proven. Its gate-level design is also shown. In Section 4, error detectability and easy testability of the proposed adder is confirmed, and its hardware overhead is evaluated. Its power consumption and delay are also estimated. In Section 5, this brief is concluded.

2 Multi-Block Carry Select Adder

Fig. 1 shows a basic design of an n-bit multi-block carry select adder. Its inputs are the augend X: $[x_{n-1} \dots x_0]$ and the addend Y: $[y_{n-1} \dots y_0]$, and its outputs are the sum S: $[s_{n-1} \dots s_0]$ and the carry output c_n. It is composed of b blocks. We number the blocks from the lowest one which is block$_0$. We let the bit-width of the k-th block block$_k$ be n_k. n_k's can differ from each other. block$_k$ has carry input cin_k and carry output $cout_k$. block$_0$ is an n_0-bit ripple carry adder, which consists of a row of full adders. In the figure, each full adder is composed of a half adder (HA) and an incrementer (INC). An INC has three input terminals for a 2-bit binary number and a carry bit, and two output terminals

- *N. Kito is with the School of Engineering, Chukyo University, Toyota, 470–0393, Japan.*
  E-mail: nkito@sist.chukyo-u.ac.jp
- *N. Takagi is with the Graduate School of Informatics, Kyoto University, Kyoto, 606–8501, Japan.*

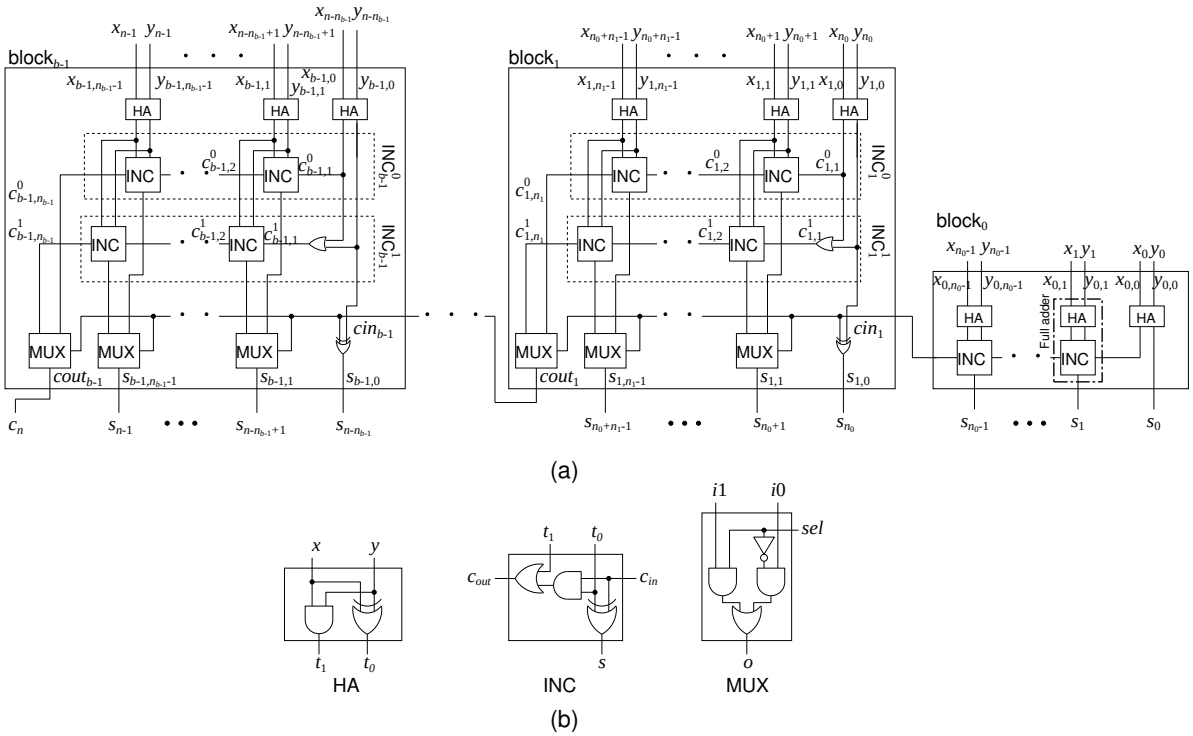

Fig. 1. Multi-block carry select adder (a), and the gate-level designs of HA, INC and MUX (b).

for a 2-bit sum result. Each of the other blocks consists of a row of HAs, two rows of INCs, i.e., INC^0 and INC^1, and a row of 2:1 multiplexers (MUXs). Fig. 1(b) shows gate-level designs of HA, INC, and MUX. In each block except block_0, two candidates of the sum result, i.e., one assuming the carry input cin_k being 0 and the other assuming cin_k being 1 are calculated by INC^0 and INC^1, respectively, and the correct one is selected by the row of MUXs.

In this brief, when a signal name has two subscripts, the first subscript represents the index of the block and the second one represents the position in the block. For example, $x_{k,j}$ and $y_{k,j} (0 \leq j < n_k)$ represent the input bits of the position j in block_k. Here, $x_{k,j} = x_l$ where $l = j + \sum_{m=0}^{k-1} n_m$. When a signal name has a superscript, it represents the row of INCs which the signal belongs to. $s^0_{k,j} (0 \leq j < n_k)$ represents the output of position j of the INC^0 in block_k. $c^0_{k,j}$ and $c^1_{k,j}$ represent the carry signals from position $j-1$ in INC^0_k and INC^1_k, respectively. The operands of block_k are X_k: $[x_{k,n_k-1} \ldots x_{k,0}]$ and Y_k: $[y_{k,n_k-1} \ldots y_{k,0}]$, and the sum result of the block is S_k: $[s_{k,n_k-1} \ldots s_{k,0}]$.

3 Concurrent Error Detectable Adder with Easy Testability

We propose a concurrent error detectable adder with easy testability. In Section 3.1, we show a design of the adder and a configuration of its adder blocks. In Section 3.2 and Section 3.3, we prove concurrent error detectability and easy testability of the adder. In Section 3.4, we discuss hardware overhead of the adder.

3.1 Configuration

Fig. 2(a) shows a design of the proposed adder. In addition to operands X and Y, it receives their parities p_X and p_Y, i.e., the XORed value of X and the XORed value of Y. In addition to sum output S, it produces the predicted parity p_S of the sum output and two carry outputs c_n and c'_n. We restrict the block length n_k to be even.

Any erroneous output of the adder caused by a fault modeled as a single stuck-at fault can be detected by comparing the predicted parity p_S with the parity of sum output S and comparing c_n and c'_n. One inconsistency in the two input pairs, i.e., a pair of X and p_X or a pair of Y and p_Y, can also be detected.

Each addition block except block_0 has two carry inputs. Each addition block has two carry outputs, and produces parity p_{C_k} of carry signals which has the same value as $c_{k,n_k-1} \oplus \cdots \oplus c_{k,1} \oplus cin_k$ in case of correct operation, where $c_{k,n_k-1}, \cdots, c_{k,1}$ are carry bits generated during the addition of X_k, Y_k, and cin_k.

Addition block block_k ($k \geq 1$) of the proposed adder is shown in Fig. 2(b). The addition block receives two operands X_k and Y_k, and produces sum result S_k. In addition to those inputs and output, it receives two carry inputs cin_k and cin'_k, and produces parity of carries p_{C_k} and two carry outputs $cout_k$ and $cout'_k$. $cout_k$ and $cout'_k$ are connected to cin_{k+1} and cin'_{k+1}, respectively.

Fig. 2(c) shows gate-level designs of HA′, INC′, and MUX′. Both of HA′ and INC′ have two carry outputs to obtain concurrent error detectability. One carry bit is used for addition, and the other is used for parity prediction.

HA′ and INC′ are designed so that effects of a single stuck-at fault in an INC′ or its ascendant HA's never appear in both its sum output and one of its carry signals simultaneously because such a faulty operation generates an erroneous sum result and a predicted parity consistent with the erroneous sum result. In MUX′ and INC′, we

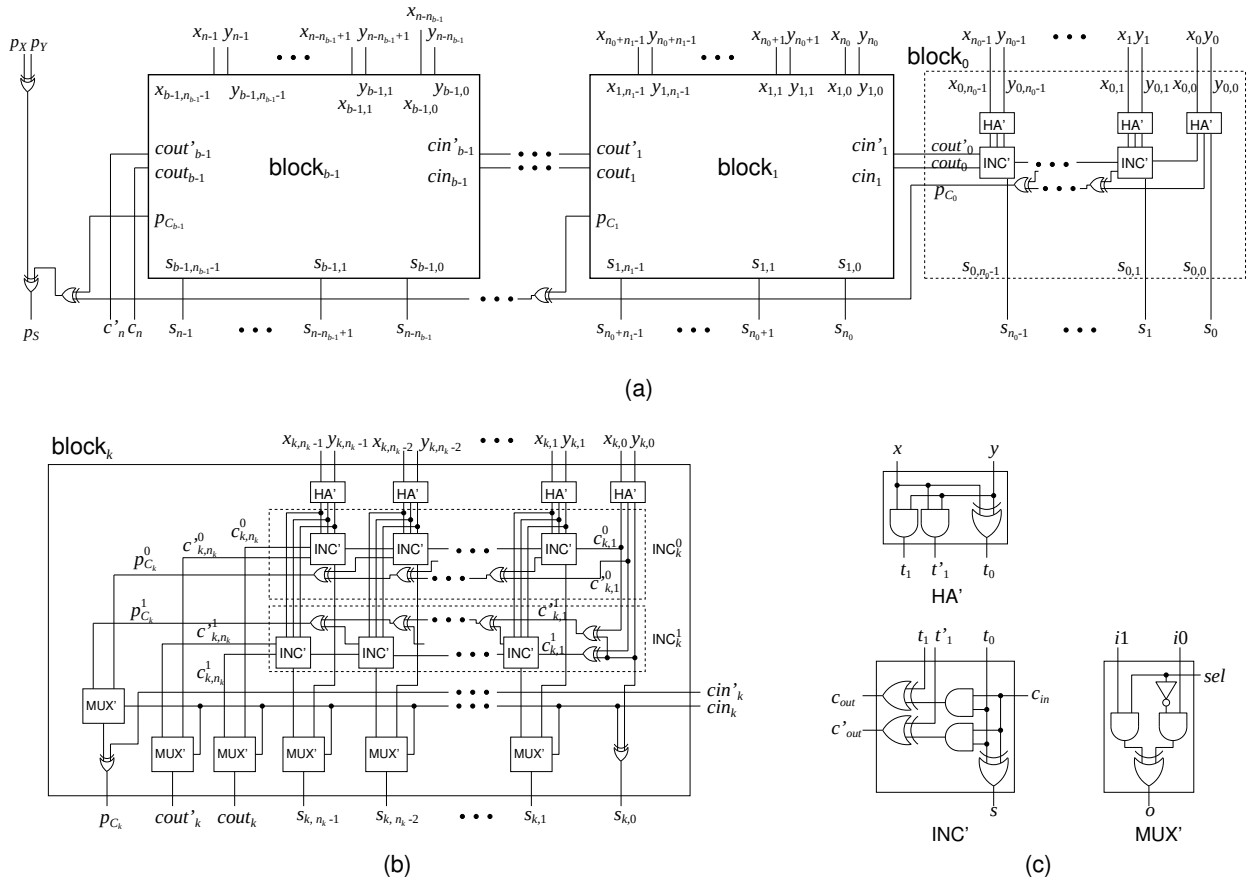

Fig. 2. Concurrent error detectable adder with easy testability (a), the design of block$_k$ ($k \geq 1$) for the adder (b), and the gate-level designs of HA', INC' and MUX' (c).

use XOR gates to improve testability. Because XOR gates prevent masking of effects of a fault, effects of a fault can be observed easily.

In each addition block, an XOR gate is placed for every bit position of INC^0_k and INC^1_k except the most significant position and the least significant position. For parity output p_{C_k}, two intermediate candidates are calculated in the two rows of INCs. One is $p^0_{C_k}$ which is calculated as $c'^0_{k,n_k-1} \oplus \cdots \oplus c'^0_{k,1}$ in INC^0_k, and the other is $p^1_{C_k}$ which is calculated as $c'^1_{k,n_k-1} \oplus \cdots \oplus c'^1_{k,1}$ in INC^1_k. The leftmost MUX' selects one of them according to the value of carry input cin_k, and the XOR gate at the output of the MUX' produces the parity of the carry bits including cin'_k. With the obtained p_{C_k}, the adder calculates predicted parity p_S of the sum as $(p_X \oplus p_Y) \oplus (p_{C_b-1} \oplus \cdots \oplus p_{C_0})$ with XOR gates because s_i is equal to $x_i \oplus y_i \oplus c_i$ in the correct operation.

3.2 Concurrent Error Detectability

Effects of a single stuck-at fault in an INC' or its ascendant HA' never appear in both its sum output and one of its two carry signals simultaneously as described in Section 3.1. Therefore, effect of a fault in an INC' or its ascendant HA' appears on either (1) one of the two carry signals of the INC', (2) the sum signal, or (3) all the three signals (two carries and the sum). In any case, any erroneous result caused by a fault can be detected by comparing the predicted parity p_S with the parity of S and comparing c_n and c'_n.

In case (1), one of the two carry signals of INC' is used for addition, and the other is used for parity prediction. When the carry signal for parity prediction is erroneous, only one-bit of carry bits for the parity prediction is affected and the sum result is correct. Thus, erroneous results caused by the fault can be detected. When the carry signal for addition is erroneous, we let $c_{k,j}$ be the carry that is affected by the fault occurred at an INC'. Then, the error affecting $c_{k,j}$ induces also an error at the sum signal $s_{k,j}$, and if it is not propagated in any subsequent positions, the error is detected because the carry signal affected by the fault is not used for parity prediction. On the other hand, if the erroneous value of $c_{k,j}$ is propagated at q subsequent carry signals $c_{k,j+1}, \cdots, c_{k,j+q}$, they will also induce errors in the q sum signals $s_{k,j+1}, \cdots, s_{k,j+q}$. Thus, the $q+1$ sum signals $s_{k,j}, s_{k,j+1}, \cdots, s_{k,j+q}$ will be erroneous. Here, as the logic generating the carry signals $c'_{k,j+1}, \cdots, c'_{k,j+q}$ is identical to the logic generating the carry signals $c_{k,j+1}, \cdots, c_{k,j+q}$, and they have both the same carry inputs, i.e. the carry signals $c_{k,j}, c_{k,j+1}, \cdots, c_{k,j+q-1}$, the q carry signals $c'_{k,j+1}, \cdots, c'_{k,j+q}$ will be also erroneous. Therefore, $q+1$ sum signals $s_{k,j}, s_{k,j+1}, \cdots, s_{k,j+q}$ will be erroneous, and q carry signals $c'_{k,j+1}, \cdots, c'_{k,j+q}$ will be erroneous. Thus, the predicted parity $(p_X \oplus p_Y) \oplus (p_{C_{b-1}} \oplus \cdots \oplus p_{C_0})$ will be computed by means of q erroneous signals, while the parity of the sum signals will be computed by means of $q+1$ erroneous signals. Therefore, as the number of the erroneous signals used in these two parity computations differ by 1,

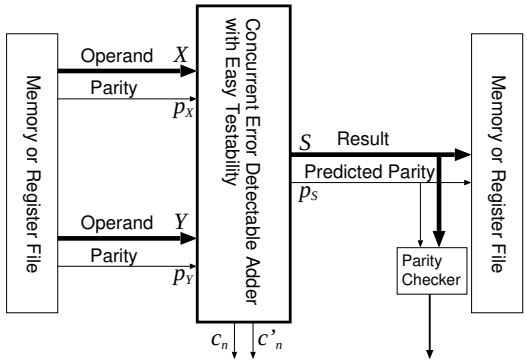

Fig. 3. Example of a datapath circuit with the proposed adder in a system using parity-based error detection.

these two parities will be different and thus the error will be detected.

In case (2), a bit of the sum result is inverted and the parity of the sum result is different from the parity of the correct one. On the other hand, the predicted parity is calculated correctly because all the carry bits used for the prediction are correct. The effect of a fault is detected by comparing the predicted parity with the parity of the sum result.

In case (3), all of the carry bits and the sum bit from the INC' are incorrect. Because both of the two carry bits are incorrect, there is no inconsistency among the obtained sum bits and carry bits used for parity prediction in the upper positions than the position of the faulty INC'. In the lower positions of the INC', though carry bits for parity prediction are correct, the sum bit from the INC' is inverted. Therefore, parity of the sum result is different from the predicted parity.

In the adder, each adder block has two carry inputs cin_k and cin'_k. If there is an error on cin_k then $s_{k,0}$ will be erroneous. Furthermore the error on cin_k can also induce errors on several other sum outputs (let us say at q sum outputs). Also, similarly to the arguments given in case (1), it will also create errors at q carry signals $c'_{k,i}$. Thus, there will be $q+1$ erroneous sum outputs and q erroneous carry signals $c'_{k,i}$, and based to the same arguments as those given in case (1) these errors will also be detected.

The output of an XOR or a MUX' affects only one sum or carry bit or the predicted parity. Therefore, the effect of a fault in them is detected by comparing the predicted parity with the parity and comparing two carry outputs of the adder.

Note that inconsistency of one of the input operands and its parity input causes an incorrect result of the predicted parity because the parity input is used for the parity prediction. Therefore, it is also possible to detect inconsistency of X and p_X or inconsistency of Y and p_Y when there are no faults in the adder.

The proposed adder is suitable for systems using parity-based error detection as shown in Fig. 3. The parity-based error detection of arithmetic circuits was used in real designs [21], e.g. in [22]. Parities fed from a memory or a register file are used as p_X and p_Y for operands X and Y, and the predicted parity obtained by the proposed adder is used for the parity bit of the result. Any erroneous output of the adder is detected by observing the parity checker of the system and comparing two carry outputs c_n and c'_n.

TABLE 1
Input test patterns for block$_0$.

Pattern	$x_{0,j}y_{0,j}$	
	$j \geq 1$	$j = 0$
t_0	01	11
t_1	11	11
t_2	10	10
t_3	00	11
t_4	11	01
t_5	01	11
t_6	11	00
t_7	00	11
t_8	00	11
t_9	00	00

TABLE 2
Input test patterns for block$_k$ ($k \geq 1$) shown in Fig. 2(b).

Pattern	$x_{k,j}y_{k,j}$		cin_k
	$j \geq 1$	$j = 0$	cin'_k
t_0	01	11	0
t_1	11	11	0
t_2	10	10	0
t_3	00	11	0
t_4	11	01	0
t_5	01	01	1
t_6	11	11	1
t_7	10	00	1
t_8	00	01	1
t_9	00	00	1

3.3 Easy Testability

We show that the proposed adder is testable with 10 input patterns under the single stuck-at fault model, through designing test patterns. Each test pattern is obtained by packing (concatenating) input patterns for testing the addition blocks. First, we design test patterns for each addition block.

Table 1 shows the test patterns for block$_0$. In each pattern, the bit values for the positions $j \geq 1$ are the same. In addition to the consideration for concurrent error detection, INC' and HA' are designed carefully considering testability. For an INC' and its ascendant HA', the three patterns derived carefully, $(x\ y\ c_{in}) = 100$, 011, and 111, are sufficient for testing. Note that at least three patterns are necessary for testing inputs and an output of a 2-input AND gate, and the set of those three patterns is smallest for an INC' and its ascendant HA'. Those three patterns are fed to all of INC's and HA's, and effects of a fault appear as erroneous sum results or as erroneous predicted parities through XOR gates. All XOR gates for parity prediction are testable with the 10 patterns.

Table 2 shows the test patterns for block$_k$ ($k \geq 1$). In each pattern, the bit values for the positions $j \geq 1$ are the same. By the patterns in Table 2, each component in block$_k$, such as a MUX', receives patterns shown in Table 3. For a MUX', the two patterns derived carefully, $(i0\ i1\ sel) = 110$ and 111, are sufficient for testing outputs of the internal AND gates and the NOT gate. Testing for its inputs and its output is done as testing for rows of INCs. As shown in Table 3, the two

TABLE 3
Input patterns of components in block$_k$ ($k \geq 1$).

	INC′ and its ascendant HA′ ($x\ y\ c_{in}$)				MUX′ ($i0\ i1\ sel$)				XOR for parity prediction			
	In INC^0		In INC^1		For p_{C_k}	For $cout_k$	For $s_{k,j}$		Receiving $c'^0_{k,j+1}$		Receiving $c'^1_{k,j+1}$	
	$j>1$	$j=1$	$j>1$	$j=1$		and $cout'_k$	$j>1$	$j=1$	$j:even$	$j:odd$	$j:even$	$j:odd$
t_0	011	011	011	011	110	110	000	000	10	11	10	11
t_1	111	111	111	111	110	110	110	110	10	11	10	11
t_2	100	100	101	101	010	010	100	100	00	00	10	11
t_3	000	001	000	001	110	000	000	110	01	01	01	01
t_4	111	110	111	111	010	110	110	010	11	10	10	11
t_5	010	010	011	011	011	011	101	101	00	00	10	11
t_6	111	111	111	111	111	111	111	111	10	11	10	11
t_7	100	100	100	100	001	001	111	111	00	00	00	00
t_8	000	000	000	001	011	001	001	011	00	00	01	01
t_9	000	000	000	000	001	001	001	001	00	00	00	00

TABLE 4
Generation rules of test patterns for the multi-block adder.

Pattern for block$_k$	$cout_k$ (=cin_{k+1})	Pattern for block$_{k+1}$
t_0	1	t_9
t_1	1	t_8
t_2	0	t_2
t_3	0	t_3
t_4	1	t_7
t_5	1	t_5
t_6	1	t_6
t_7	0	t_4
t_8	0	t_1
t_9	0	t_0

patterns are fed to all MUX's. As described before, the three patterns, $(x\ y\ c_{in}) = 100$, 011, and 111, are sufficient for testing an INC′ and its ascendant HA′. Those three patterns are fed to all of INC's and HA's, and the effect of a faulty gate in them propagates to circuit output through MUX's. All XOR gates are testable with the 10 patterns.

We use generation rules in Table 4 defining relations between the patterns of block$_k$ and those of block$_{k+1}$ to pack the patterns for testing addition blocks into test patterns of the whole adder. By using the rules, a test set of 10 patterns can be designed so that the set applies patterns from t_0 to t_9 to all blocks. For each of the test patterns, its bits from ones for block$_0$ to ones for block$_{b-1}$ are designed according to the rules in Table 4. As a result, a set of 10 patterns is obtained for testing the whole adder consisting of multiple blocks.

As an example, a test set for a 32-bit adder consisting of four 8-bit blocks is shown in Table 5. In the table, t_0 to t_9 are arranged in the column for block$_0$. The other columns are derived with the rules in Table 4. For example, we show the derivation of the top row of Table 5. In the top row, t_0 is assigned to block$_0$. By the top row of Table 4, we use t_9 for block$_1$. In the same way, we use t_0 for block$_2$ by the bottom row of Table 4, and use t_9 for block$_3$. The rightmost column of Table 5 shows the corresponding additions to the test patterns in hexadecimal. As we can see from the column, the adder is testable through those 10 additions.

3.4 Hardware Overhead

We estimate circuit size and hardware overhead for the addition block of the proposed adder. In Table 6, the number of gates in the addition block of the proposed adder and that in the addition block of Fig. 1 are shown. In the bottom row of the table, the estimated circuit size in the equivalent number of 2-input NAND gates is presented.

In the estimation, an XOR cell is considered as 2.5 gate equivalents because in CMOS technology without transmission gates, an XOR gate and a 2-input NAND gate can be realized by 10 transistors and 4 transistors, respectively. A NOT gate, a 2-input OR gate, and a 2-input AND gate are considered as 0.5, 1.5, and 1.5 gate equivalents, respectively.

When bit-width n_k is large, hardware overhead of the addition block of the proposed adder is 87.5% (=17.5/20). Note that all AND gates in HA's, INC's, and MUX's except the HA′ at the least significant position of each block can be replaced with NAND gates because XOR is self-anti-dual. Error detectability and testability of the addition block hold after the replacement. When AND gates are replaced with NAND gates, hardware overhead is 67.5% (=13.5/20). Thus, the proposed adder is smaller than duplication which requires overhead more than 100% and was used in real designs such as server processors [23], [24] and controllers for automotive applications [25].

4 Experimental Results

We evaluated error detectability, test coverage of the proposed adder, and its circuit area. We prepared the netlist of the proposed adder by instancing each gate as the corresponding standard cell. Rohm 0.18μm standard cell library provided through VDEC [26] was used for the evaluation. The netlist prepared for the evaluations was a 32-bit design of the proposed adder composed of four 8-bit blocks.

We examined concurrent error detectability of the proposed adder including the error detection unit shown in Fig. 4. The error detection unit consists of a parity generator and a dual-rail checker also known as a two-rail checker [27]. The generator calculates parity of the sum. The checker compares the parity with the predicted parity, and also compares two carry outputs. Two input pairs of the checker are assumed to be complementary, and the pair of its output is kept complementary as long as the assumption is hold. Errors are found by detecting break of the complement in the checker output. Even when a fault occurs in the checker, it does not fall into false negative , i.e., it is fault-secure [27]. We can use the circuit-level design of the checker [28] which considers various types of faults to harden the adder design. For the netlist of the design, we examined error detectability

TABLE 5
Example of test patterns for a 32-bit 4-block adder (where n_3, n_2, n_1, and n_0 were 8).

$block_3$		$block_2$		$block_1$		$block_0$	
$x_{31}y_{31} \dots x_{24}y_{24}$	cin_3	$x_{23}y_{23} \dots x_{16}y_{16}$	cin_2	$x_{15}y_{15} \dots x_8y_8$	cin_1	$x_7y_7 \dots x_0y_0$	$X+Y$
t_9 : 00 ··· 00 00	1	t_0 : 01 ··· 01 11	0	t_9 : 00 ··· 00 00	1	t_0 : 01 ··· 01 11	00010001 + 00ff00ff
t_8 : 00 ··· 00 01	1	t_1 : 11 ··· 11 11	0	t_8 : 00 ··· 00 01	1	t_1 : 11 ··· 11 11	00ff00ff + 01ff01ff
t_2 : 10 ··· 10 10	0	t_2 : 10 ··· 10 10	0	t_2 : 10 ··· 10 10	0	t_2 : 10 ··· 10 10	ffffffff + 00000000
t_3 : 00 ··· 00 11	0	t_3 : 00 ··· 00 11	0	t_3 : 00 ··· 00 11	0	t_3 : 00 ··· 00 11	01010101 + 01010101
t_7 : 10 ··· 10 00	1	t_4 : 11 ··· 11 01	0	t_7 : 10 ··· 10 00	1	t_4 : 11 ··· 11 01	fefefefe + 00ff00ff
t_5 : 01 ··· 01 01	1	t_5 : 01 ··· 01 01	1	t_5 : 01 ··· 01 01	1	t_5 : 01 ··· 01 11	00000001 + ffffffff
t_6 : 11 ··· 11 11	1	t_6 : 11 ··· 11 11	1	t_6 : 11 ··· 11 11	1	t_6 : 11 ··· 11 00	fffffffe + fffffffe
t_4 : 11 ··· 11 01	0	t_7 : 10 ··· 10 00	1	t_4 : 11 ··· 11 01	0	t_7 : 00 ··· 00 11	fefefe01 + ff00ff01
t_1 : 11 ··· 11 11	0	t_8 : 00 ··· 00 01	1	t_1 : 11 ··· 11 11	0	t_8 : 00 ··· 00 11	ff00ff01 + ff01ff01
t_0 : 01 ··· 01 11	0	t_9 : 00 ··· 00 00	1	t_0 : 01 ··· 01 11	0	t_9 : 00 ··· 00 00	01000100 + ff00ff00

TABLE 6
Comparison of gate count.

	Original	Proposed
NOT	n_k	$n_k + 2$
2-AND	$5n_k - 2$	$8n_k$
2-OR	$3n_k - 1$	0
2-XOR	$3n_k - 1$	$10n_k - 1$
Total (# of 2-NAND equivalents)	$20n_k - 7$	$37.5n_k - 1.5$ or $33.5n_k - 0.5$

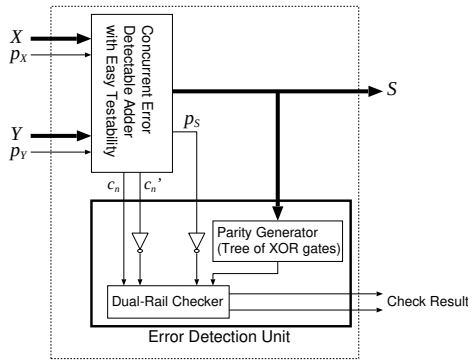

Fig. 4. Error detection unit for evaluation of error detectability.

by simulating every faulty circuit containing a single stuck-at fault with random patterns. 10,000 random patterns were used for each of the simulation. It was confirmed that the checker always reports inconsistency if S is erroneous.

We also confirmed easy testability of the proposed adder. Synopsys TetraMax was employed for evaluation of test coverage. 10 patterns in Table 5 were applied for the 32-bit design. By the evaluation, it was confirmed that effect of any single stuck-at fault appears at the circuit outputs with at least one of the patterns.

We evaluated circuit area of the proposed adder. The netlist of the original design in Fig. 1 was prepared. It was also a 32-bit design composed of four 8-bit blocks. The circuit area of the original design was 7,016μm^2 and that of the proposed design was 11,959μm^2 when NAND gates were used in place of AND gates. Area overhead of the proposed design was about 70.5%, and it was close to the estimation in Section 3.4. We also evaluated circuit area of the proposed adder including the error detection unit in Fig. 4. Its circuit area was 13,038μm^2, and its overhead was still smaller than 100%.

We estimated power consumption and delay of the proposed adder. We used the netlists without the error detection unit and employed Synopsys PrimeTime for the estimation. The estimated power consumption of the original design was 9.51mW, and that of the proposed design was 15.96mW. Therefore, the overhead of the proposed design in power consumption was 67.8%, and was close to the overhead in circuit area. The delay of the original design was 6.92ns. The delay of the proposed design was 7.68ns for the sum result and was 8.41ns for the predicted parity. The overhead in delay time for the sum result was 11.0% because XORs with larger fanout used for carry calculation in the proposed design are slower than ORs in the original one.

For comparison, we evaluated the circuit area of a design using duplication in which the original design in Fig. 1 was duplicated and a dual-rail checker was used for checking the pair of their sum results. Its circuit area was 16,239μm^2, and its area overhead was 131.5%. We also estimated power consumption and delay of the design using duplication. Its estimated power consumption was 25.27mW and the delay of its checker output was 8.52ns. The circuit area of the proposed design was smaller than that of the design using duplication, and the delay of the predicted parity was comparable to the design using duplication.

5 Conclusion

A concurrent error detectable adder with easy testability is proposed. The proposed adder is based on a multi-block carry select adder. It receives parities of two operands in addition to the operands, and produces predicted parity of the sum result and two carry outputs in addition to the sum result. Any erroneous output of the adder by a fault modeled as a single stuck-at fault is detected by parity checking and comparison of the two carry outputs. The adder is also testable with 10 patterns under single stuck-at fault model. This property eases testing of the adder to find a fault before the occurrence of a second fault.

For future intelligent autonomous systems such as autonomous cars, both concurrent error detectability and easy testability for detecting a fault during operation before fatal accidents are crucial. More investigations of circuits with both of concurrent error detectability and testability and coping with various faults such as delay faults and soft errors are desirable for realizing reliable systems.

ACKNOWLEDGMENTS

This work was supported in part by JSPS KAKENHI Grant Number JP16H02795, and by VLSI Design and Education Center (VDEC), The University of Tokyo with the collaboration with Synopsys Corporation.

REFERENCES

[1] J. H. Stathis, "Reliability limits for the gate insulator in CMOS technology," *IBM Journal of Research and Development*, vol. 46, no. 2/3, pp. 265–286, 2002.

[2] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The impact of technology scaling on lifetime reliability," *Proc. International Conference on Dependable Systems and Networks (DSN '04)*, pp. 177–186, June 2004.

[3] D. K. Schroder and J. A. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," *Journal of Applied Physics*, vol. 94, no. 1, pp. 1–18, 2003.

[4] M. Nicolaidis, "Carry checking/parity prediction adders and ALUs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 1, pp. 121–128, Feb. 2003.

[5] B. Kumar and P. Lala, "On-line detection of faults in carry-select adders," *Proc. International Test Conference (ITC '03)*, vol. 1, pp. 912–918, Sep. 2003.

[6] D. Vasudevan and P. Lala, "A technique for modular design of self-checking carry-select adder," *Proc. 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT '05)*, pp. 325–333, Oct. 2005.

[7] N. Kito and N. Takagi, "Low-overhead fault-secure parallel prefix adder by carry-bit duplication," *IEICE Transactions on Information and Systems*, vol. E96-D, no. 9, pp. 1962–1970, Sep. 2013.

[8] J. Rivers, M. Gupta, J. Shin, P. Kudva, and P. Bose, "Error tolerance in server class processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 7, pp. 945–959, July 2011.

[9] J. R. Black, "Electromigration - a brief survey and some recent results," *IEEE Transactions on Electron Devices*, vol. 16, no. 4, pp. 338–347, April 1969.

[10] C. K. Hu, R. Rosenberg, H. S. Rathore, D. B. Nguyen, and B. Agarwala, "Scaling effect on electromigration in on-chip Cu wiring," *Proc. IEEE International Interconnect Technology Conference*, pp. 267–269, May 1999.

[11] E. T. Ogawa, J. W. McPherson, J. A. Rosal, K. J. Dickerson, T. C. Chiu, L. Y. Tsung, M. K. Jain, T. D. Bonifield, J. C. Ondrusek, and W. R. McKee, "Stress-induced voiding under vias connected to wide Cu metal leads," pp. 312–321, April 2002.

[12] H. Matsuyama, T. Suzuki, T. Nakamura, M. Shiozu, and H. Ehara, "Re-think stress migration phenomenon with stress measurement in 12 years," *Proc. IEEE International Interconnect Technology Conference and IEEE Materials for Advanced Metallization Conference (IITC/MAM)*, pp. 307–310, May 2015.

[13] B. Becker, R. Drechsler, and P. Molitor, "On the generation of area-time optimal testable adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 1049–1066, Sep. 1995.

[14] R. D. Blanton and J. P. Hayes, "Testability of convergent tree circuits," *IEEE Transactions on Computers*, vol. 45, pp. 950–963, Aug. 1996.

[15] S. Kajihara and T. Sasao, "On the adders with minimum tests," *Proc. Sixth Asian Test Symposium (ATS '97)*, pp. 10–15, Nov. 1997.

[16] W. R. Moore, "Minimal C-testable tests for block-CLA adders," *International Journal of Electronics*, vol. 85, no. 5, pp. 611–628, 1998.

[17] R. D. Blanton and J. P. Hayes, "On the design of fast, easily testable ALU's," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, pp. 220–223, Apr. 2000.

[18] D. Gizopoulos, M. Psarakis, A. Paschalis, and Y. Zorian, "Easily testable cellular carry lookahead adders," *Journal of Electronic Testing*, vol. 19, pp. 285–298, June 2003.

[19] N. Kito, S. Fujii, and N. Takagi, "A C-testable multiple-block carry select adder," *IEICE Transactions on Information and Systems*, vol. E95-D, no. 4, pp. 1084–1092, Apr. 2012.

[20] O. Bedrij, "Carry-select adder," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 3, pp. 340–346, June 1962.

[21] T. Li, J. A. Ambrose, R. Ragel, and S. Parameswaran, "Processor design for soft errors: Challenges and state of the art," *ACM Computing Surveys*, vol. 49, no. 3, pp. 57:1–57:44, Nov. 2016.

[22] H. Ando, Y. Yoshida, A. Inoue, I. Sugiyama, T. Asakawa, K. Morita, T. Muta, T. Motokurumada, S. Okada, H. Yamashita, Y. Satsukawa, A. Konmoto, R. Yamashita, and H. Sugiyama, "A 1.3GHz fifth generation SPARC64 microprocessor," *Proc. 40th annual Design Automation Conference (DAC '03)*, pp. 702–705, 2003.

[23] T. J. Slegel, R.M. Averill III, M. A. Check, B. C. Giamei, B. W. Krumm, C. A. Krygowski, W. H. Li, J. S. Liptay, J. D. MacDougall, T. J. McPherson, J. A. Navarro, E. M. Schwarz, K. Shum, and C. F. Webb, "IBM's S/390 G5 microprocessor design," *IEEE Micro*, vol. 19, no. 2, pp. 12–23, Mar. 1999.

[24] T. J. Slegel, E. Pfeffer, and J. A. Magee, "The IBM eServer z990 microprocessor," *IBM Journal of Research and Development*, vol. 48, no. 3-4, pp. 295–309, May 2004.

[25] Texas Instruments, "Hercules safety MCUs," *http://www.ti.com/microcontrollers/hercules-safety-mcus/overview.html*, Accessed on Aug. 17, 2018.

[26] VLSI Design and Education Center (VDEC), *http://www.vdec.u-tokyo.ac.jp/*, Accessed on Aug. 17, 2018.

[27] P. K. Lala, *Self-checking and Fault-tolerant Digital Design*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[28] J. C. Lo, "Novel area-time efficient static CMOS totally self-checking comparator," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 2, pp. 165–168, Feb. 1993.