PAPER

# A C-Testable 4-2 Adder Tree for an Easily Testable High-Speed Multiplier*

**Nobutaka KITO**[†a)], ***Member***, **Kensuke HANAI**[††], ***Nonmember***, ***and*** **Naofumi TAKAGI**[†], ***Member***

**SUMMARY** A C-testable 4-2 adder tree for an easily testable high-speed multiplier is proposed, and a recursive method for test generation is shown. By using the specific patterns that we call 'alternately inverted patterns,' the adder tree, as well as partial product generators, can be tested with 14 patterns regardless of its operand size under the cell fault model. The test patterns are easily fed through the partial product generators. The hardware overhead of the 4-2 adder tree with partial product generators for a 64-bit multiplier is about 15%. By using a previously proposed easily testable adder as the final adder, we can obtain an easily testable high-speed multiplier.

***key words:*** *multiplier, design for testability, 4-2 adder tree, C-testability*

## 1. Introduction

The growth in the number of logic gates integrated in a VLSI chip has made testing chips more difficult. In order to reduce the cost of a test of a VLSI chip, it is crucial to make its component circuits, such as arithmetic circuits, easily testable. In this paper, we focus on development of an easily testable parallel multiplier.

A parallel multiplier is one of the key component circuits in VLSI systems. Widely used parallel multipliers are categorized into two types. One is an array multiplier, and the other is a tree-type multiplier such as a Wallace multiplier. Although an array multiplier is area-efficient, it does not operate so fast, and its computation time is proportional to its operand size. On the other hand, a tree-type multiplier operates fast, and its computation time is proportional to the logarithm of its operand size.

In this paper, we propose a C-testable 4-2 adder tree (or 4-2 compressor tree) for an easily testable tree-type multiplier and show a generation method of test patterns of the tree through the partial product generators (PPGs) of a multiplier. An arithmetic circuit is said to be C-testable, if it can be tested with a constant number of input patterns independent of its operand size. We adopt the cell fault model [1] and treat basic circuit blocks, such as full adders, as cells.

Although C-testable array multipliers were proposed [2]–[5], no C-testable tree-type multiplier has been proposed so far. In [6] and [7], easily testable multipliers with a 4-2 adder tree were proposed. These multipliers require the number of patterns growing with its operand size. In [8], design methods of C-testable counters were proposed. Although the methods can construct a C-testable Wallace tree, the structure of the obtained tree is complex, and it seems impossible to feed test patterns of the tree through the PPGs of a multiplier. In [9], we proposed a generalized design method for C-testable multipliers which can generate tree-type multipliers. However, no detailed design for specific archtectures were provided.

We use a 4-2 adder tree with special structure which has specific connections between 4-2 adders, and show a method of recursive test generation for it using special patterns which we call 'alternately inverted patterns.' By this method, we always obtain 14 input test patterns for every 4-2 adder tree of this type, regardless of the size of the tree. We show a design of the PPGs which can produce alternately inverted patterns for a test of the 4-2 adder tree.

This paper is organized as follows. In the next section, we briefly review a multiplier with a 4-2 adder tree [10] and describe the cell fault model. In Sect. 3, we propose a C-testable 4-2 adder tree and a recursive test generation method for it. In Sect. 4, we show a design of a C-testable 4-2 adder tree with PPGs for an easily testable multiplier. In Sect. 5, we discuss the hardware overhead and the delay overhead.

## 2. Preliminaries

### 2.1 Multiplier with a 4-2 Adder Tree

We consider an $N$-bit unsigned multiplier. For simplicity, we assume $N$ is a power of 2 and at least 4. We let the multiplicand and the multiplier be $X = [x_{N-1}x_{N-2}\cdots x_0]$ and $Y = [y_{N-1}y_{N-2}\cdots y_0]$, respectively.

In general, a parallel multiplier consists of three parts: partial product generators (PPGs), a partial product compressor, and a final adder. The PPGs generate partial products $P_j = X \cdot y_j \cdot 2^j$ for $j = 0, 1, \cdots, N-1$. The partial product compressor adds up the partial products by carry-save additions. In a multiplier with a 4-2 adder tree, a 4-2 adder (or 4-2 compressor) tree is used as the compressor [10]. The final adder is a carry propagate adder, which adds up the two binary numbers (sum and carry) produced by the compressor and generates the product.

The 4-2 adder tree consists of 4-2 adders as shown in

Manuscript received February 1, 2010.
Manuscript revised June 18, 2010.

[†]The authors are with the Graduate School of Informatics, Kyoto University, Kyoto-shi, 606–8501 Japan.
[††]The author is with Sanyo Semiconductor Co., Ltd., Gunmaken, 370–0596 Japan.

*This work was done while the authors were with the Department of Information Engineering, Nagoya University.

a) E-mail: nkito@i.kyoto-u.ac.jp

DOI: 10.1587/transinf.E93.D.2783

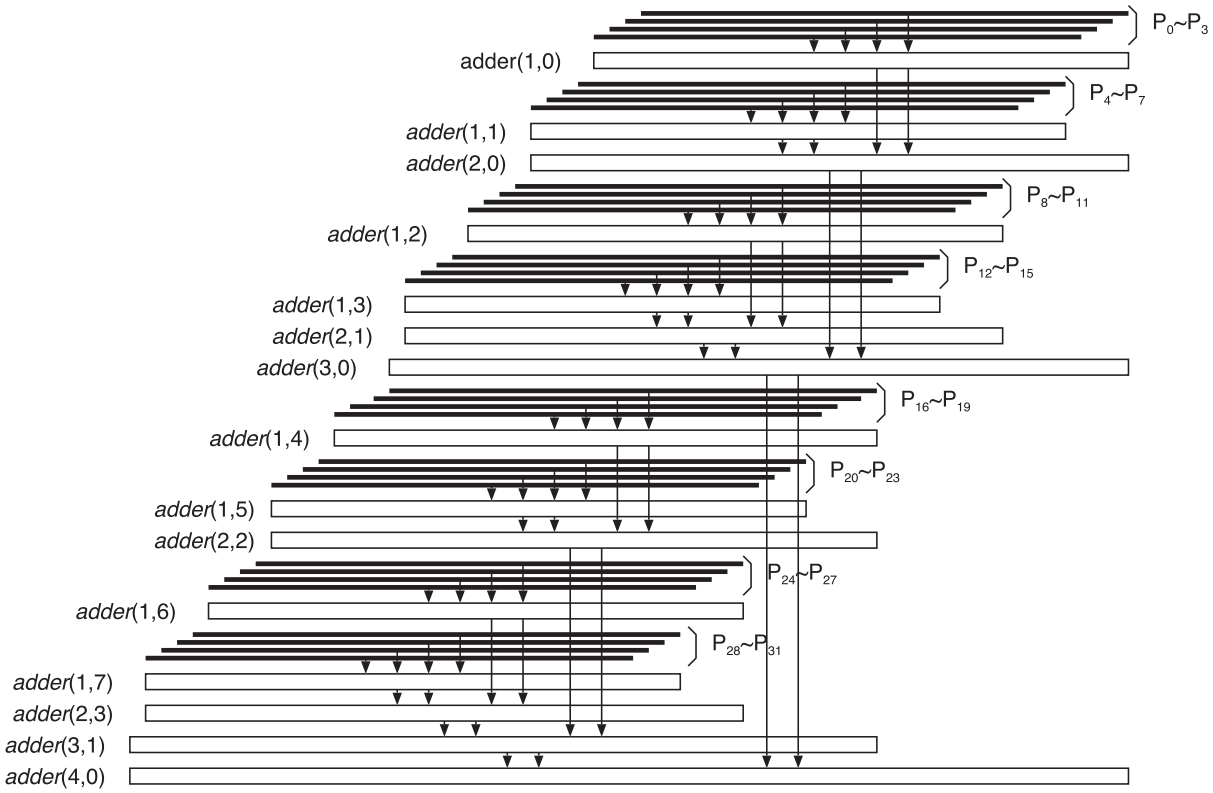



**Fig. 1** A 4-2 adder tree for 32-bit multiplier.

Fig. 1. (The figure shows a 4-2 adder tree for a 32-bit multiplier.) We label each 4-2 adder as $adder(l, m)$, where $l$ denotes the level in the tree to which the adder belongs and $m$ is the number of the adder in the level. We let the level of an adder at a leaf of the tree be 1 and let that of the adder at the root as $L = \log_2 N - 1$. We number the adders in level 1 so that $adder(1, m)$ sums up partial products $P_{4m}$, $P_{4m+1}$, $P_{4m+2}$ and $P_{4m+3}$, and number the adders in level $l$ ($\geq 2$) so that $adder(l, m)$ sums up the output of $adder(l-1, 2m)$ and that of $adder(l-1, 2m+1)$.

A 4-2 adder sums up 4 binary numbers to two binary numbers. We construct a 4-2 adder by connecting 4-2 adder blocks, each of which consists of two full adders (FAs), as shown in Fig. 2. We name the input terminals of a 4-2 adder block as $a$, $b$, $c$, $d$ and $w$, and the output terminals as $e$, $f$ and $u$. Terminal $u$ of an adder block is connected with terminal $w$ of the adder block at the next higher position. We name the internal line in a block as $v$.

Figure 3 shows 4-2 adders in a 4-2 adder tree of a multiplier. Figures 3 (a), (b), and (c) show an 4-2 adder in level 1, one in level 2, and one in level $l$ ($\geq 3$), respectively. In the figure, position $k$ means the bit position with weight $2^k$. Hereafter, we call the adder block at position $k$ as 'adder block $k$'.

Note that there are some positions in the both end parts of each 4-2 adder which have less than 4 input bits. FAs and half adders (HAs) are used as adder blocks for these positions instead of 4-2 adder blocks.

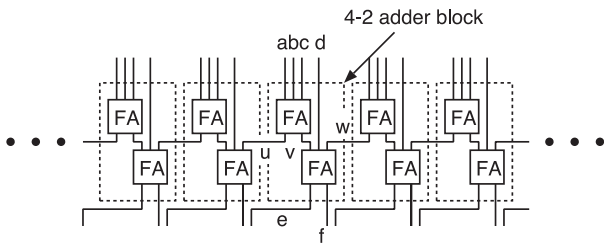



**Fig. 2** A 4-2 adder.

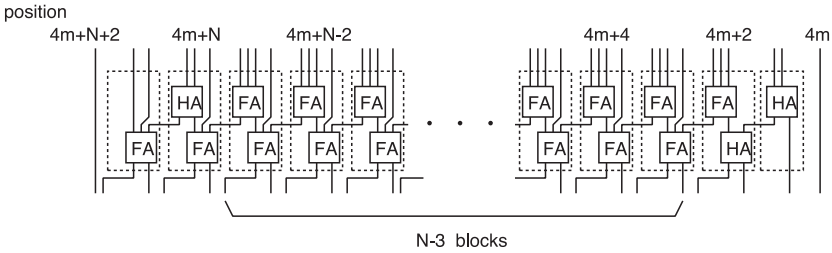
## 2.2 The Cell Fault Model

We adopt the 'cell fault model' [1] as the fault model. In the model, it is assumed that the considered circuit consists of cells. We treat basic circuit blocks, such as full adders, as cells.

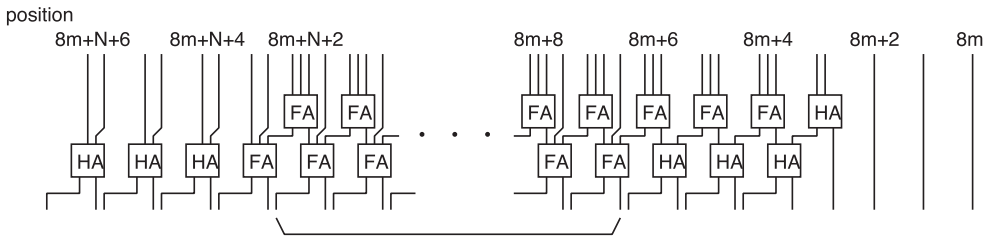
In the model, the followings hold.

- At most one cell can be faulty in the circuit.
- The faulty cell is memoryless. Namely, the faulty cell works as a combinational circuit, and its output is determined by only its present input.
- There is at least one input pattern of the faulty cell that makes the output of the cell incorrect.

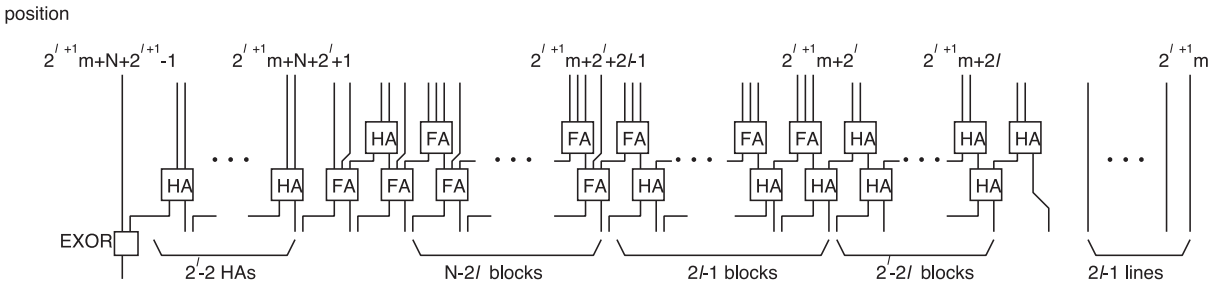
A test set with respect to this model must satisfy the following two conditions. Note that the test set is independent of gate-level implementations of cells.




(a) a 4-2 adder in level 1: $adder(1, m)$




(b) a 4-2 adder in level 2: $adder(2, m)$




(c) a 4-2 adder in level $l$: $adder(l, m)$ ($l \geq 3$)

**Fig. 3** 4-2 adders in a 4-2 adder tree.

- All cells in the circuit must receive exhaustive input patterns when all test patterns in the test set are applied to the circuit.
- The effect of a faulty cell must propagate to at least one of the primary outputs of the circuit.

## 3. Test of a 4-2 Adder Tree by Alternately Inverted Patterns

As stated in Sect. 2.1, the inputs of $adder(l, m)$ ($l \geq 2$) are from $adder(l-1, 2m)$ and $adder(l-1, 2m+1)$. Here, we specify the connections between the 4-2 adders, in order to make the 4-2 adder tree easily testable. In this section, we focus on the middle part, i.e., the positions from $2^{l+1}m + 2^l + 2l - 1$ to $2^{l+1}m + 2^l + N - 2$, which have 4 input bits. We will consider the end parts in the next section.

We connect input terminals $a$, $b$, $c$, and $d$ of the adder block $k$ of $adder(l, m)$ with output terminal $f$ of adder block $k$ of $adder(l-1, 2m)$, $e$ of block $k-1$ of $adder(l-1, 2m)$, $f$ of block $k$ of $adder(l-1, 2m+1)$, and $e$ of block $k-1$ of $adder(l-1, 2m+1)$, respectively, as shown in Fig. 4.

We define an alternately inverted pattern, an ai-pattern in short, $(\alpha, \beta, \gamma, \delta)_{ai}$ ($\alpha, \beta, \gamma, \delta \in \{0, 1\}$) for an input of a 4-2 adder as follows:
Input bits to $a$, $b$, $c$, and $d$ of adder block $k$ are $\alpha$, $\beta$, $\gamma$, and $\delta$, respectively for even $k$'s, and $\bar{\alpha}$, $\bar{\beta}$, $\bar{\gamma}$, and $\bar{\delta}$, respectively for odd $k$'s, where $\bar{\alpha}$ is the logical inverse (complement) of $\alpha$.

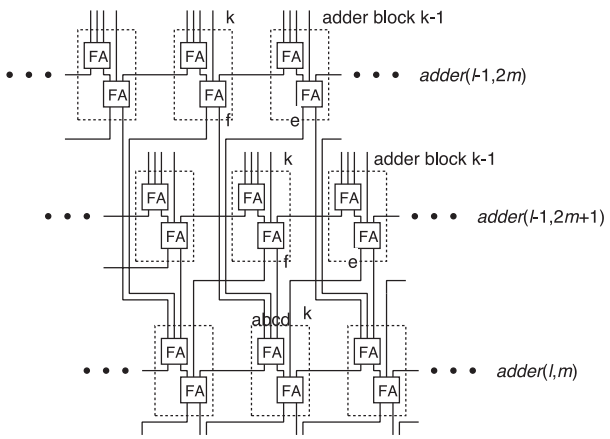



**Fig. 4** Connection between 4-2 adders (Input terminals $a$, $b$, $c$, and $d$ of the adder block $k$ of $adder(l, m)$ are connected with output terminal $f$ of adder block $k$ of $adder(l-1, 2m)$, $e$ of block $k-1$ of $adder(l-1, 2m)$, $f$ of block $k$ of $adder(l-1, 2m+1)$, and $e$ of block $k-1$ of $adder(l-1, 2m+1)$, respectively).

There are 16 ai-patterns. We name them as $ai_0$ to $ai_{15}$,

**Table 1** Signal values of adder blocks fed with ai-patterns.

| | even-numbered adder block | | | | | | odd-numbered adder block | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pattern | $a,b,c,d$ | $w$ | $v$ | $e$ | $f$ | $u$ | $a,b,c,d$ | $w$ | $v$ | $e$ | $f$ | $u$ |
| $ai_0$ | 0,0,0,0 | 1 | 0 | 0 | 1 | 0 | 1,1,1,1 | 0 | 1 | 1 | 0 | 1 |
| $ai_{15}$ | 1,1,1,1 | 0 | 1 | 1 | 0 | 1 | 0,0,0,0 | 1 | 0 | 0 | 1 | 0 |
| $ai_1$ | 0,0,0,1 | 1 | 0 | 1 | 0 | 0 | 1,1,1,0 | 0 | 1 | 0 | 1 | 1 |
| $ai_{14}$ | 1,1,1,0 | 0 | 1 | 0 | 1 | 1 | 0,0,0,1 | 1 | 0 | 1 | 0 | 0 |
| $ai_3$ | 0,0,1,1 | 1 | 1 | 1 | 1 | 0 | 1,1,0,0 | 0 | 0 | 0 | 0 | 1 |
| $ai_{12}$ | 1,1,0,0 | 0 | 0 | 0 | 0 | 1 | 0,0,1,1 | 1 | 1 | 1 | 1 | 0 |
| $ai_5$ | 0,1,0,1 | 1 | 1 | 1 | 1 | 0 | 1,0,1,0 | 0 | 0 | 0 | 0 | 1 |
| $ai_{10}$ | 1,0,1,0 | 0 | 0 | 0 | 0 | 1 | 0,1,0,1 | 1 | 1 | 1 | 1 | 0 |
| $ai_7$ | 0,1,1,1 | 0 | 0 | 0 | 1 | 1 | 1,0,0,0 | 1 | 1 | 1 | 0 | 0 |
| $ai_8$ | 1,0,0,0 | 1 | 1 | 1 | 0 | 0 | 0,1,1,1 | 0 | 0 | 0 | 1 | 1 |

**Table 2** Relation between input patterns of adjacent 4-2 adders in a 4-2 adder tree.

| $adder(l-1,2m)$ | $adder(l-1,2m+1)$ | $adder(l,m)$ |
|---|---|---|
| $ai_1$ | $ai_8$ | $ai_0$ |
| $ai_{14}$ | $ai_7$ | $ai_{15}$ |
| $ai_8$ | $ai_{12}$ | $ai_1$ |
| $ai_7$ | $ai_3$ | $ai_{14}$ |
| $ai_{15}$ | $ai_0$ | $ai_3$ |
| $ai_0$ | $ai_{15}$ | $ai_{12}$ |
| $ai_{10}$ | $ai_{10}$ | $ai_5$ |
| $ai_5$ | $ai_5$ | $ai_{10}$ |
| $ai_3$ | $ai_1$ | $ai_8$ |
| $ai_{12}$ | $ai_{14}$ | $ai_7$ |

by regarding $(\alpha\beta\gamma\delta)$ as a binary number. For example, $ai_5 = (0,1,0,1)_{ai}$. We use 10 ai-patterns, $ai_0$, $ai_{15}$, $ai_1$, $ai_{14}$, $ai_3$, $ai_{12}$, $ai_5$, $ai_{10}$, $ai_7$, and $ai_8$, for a test of the 4-2 adder tree.

Table 1 shows the signal values of adder blocks fed with ai-patterns. We can see that by these 10 ai-patterns, the two FAs in a 4-2 adder block, i.e., the one with inputs $a$, $b$ and $c$, and the one with inputs $v$, $d$, and $w$, are both fed with all the 8 patterns. We can also see that the output patterns of the 4-2 adder are alternately inverted. This is because the sum and the carry of an FA are both self-dual functions.

Using the fact that the output pattern of a 4-2 adder is alternately inverted when it is fed with an ai-pattern, we can test a 4-2 adder tree efficiently. We can feed an ai-pattern to $adder(l,m)$, by feeding appropriate ai-patterns to $adder(l-1,2m)$ and $adder(l-1,2m+1)$. For example, we can feed $ai_0$ to $adder(l,m)$, by feeding $ai_1$ and $ai_8$ to $adder(l-1,2m)$ and $adder(l-1,2m+1)$, respectively.

Table 2 shows the relation between input patterns of adjacent 4-2 adders in a 4-2 adder tree. The relation holds at any level in the tree. As we can see in the table, the three input pattern sets of adjacent 4-2 adders are identical. Therefore, we can test a 4-2 adder tree by only 10 patterns, regardless of the size of the tree.

We can obtain a test pattern set of a 4-2 adder tree by using the relation shown in Table 2 recursively. Each of the 10 test patterns exclusively corresponds to one of the 10 ai-patterns fed to the adder at the root of the tree. Let us consider a 4-level 4-2 adder tree. We can feed $ai_0$ to $adder(4,0)$ by feeding $ai_1$ and $ai_8$ to $adder(3,0)$ and $adder(3,1)$, respectively. We can feed $ai_1$ to $adder(3,0)$ and $ai_8$ to $adder(3,1)$ by feeding $ai_8$, $ai_{12}$, $ai_3$, and $ai_1$ to $adder(2,0)$, $adder(2,1)$, $adder(2,2)$, and $adder(2,3)$, respectively. Finally, we obtain the test pattern ($ai_3$, $ai_1$, $ai_0$, $ai_{15}$, $ai_{15}$, $ai_0$, $ai_8$, $ai_{12}$) fed to $adder(1,0)\ldots adder(1,7)$. We can obtain the other 9 test patterns in the same way.

Note that we have considered only the middle part of the 4-2 adders in this section. We will consider the both end parts in the next section.

## 4. C-Testable 4-2 Adder Tree with PPGs

### 4.1 C-Testable 4-2 Adder Tree

As stated in Sect. 2.1, in a 4-2 adder tree, there are some positions in the both end parts of each 4-2 adder which have less than 4 input bits. As shown in Fig. 3, these parts consist of FAs and HAs instead of 4-2 adder blocks. In order to adopt the recursive method of test generation described in the previous section, we modify the 4-2 adders so that their end parts also consist of 4-2 adder blocks.

Figure 5 shows a modified 4-2 adder in level 1. The modified parts are drawn by bold lines. An 'S' cell and a 'C' cell are reductions of an FA, which compute only the sum and the carry, respectively. The lowest part of $adder(1,0)$ is a bit simpler, and the part surrounded by the broken lines in Fig. 5 (a) is replaced by the one shown in (b). This is because the outputs of this part are directly connected to the final adder.

We need several extra input bits at the input terminals at the end parts shown by $\circ$ in the figure. We discuss this matter later.

Figure 6 shows a modified 4-2 adder in level 2. The lowest part of $adder(2,0)$ is simpler and shown in (b). Figure 7 shows a modified 4-2 adder in level $l$ ($l \geq 3$). The lowest part of $adder(l,0)$ is simpler and shown in (b). In $adder(L,0)$, i.e., the adder at the root, since its outputs are directly connected to the final adder, we need not modify its upper part. This part is shown in (c).

We connect the input terminals of $adder(l,m)$ ($l \geq 2$) with the output terminals of $adder(l-1,2m)$ and $adder(l-1,2m+1)$ in the way stated in the previous section. Then, there are several input terminals at the end parts shown by $\circ$ in Figs. 6 and 7 which are not connected with any output terminal. (In $adder(3,m)$, terminal $c$ at the most significant position, i.e., position $16m+N+15$ and terminal $a$ at position $16m+N+7$ shown by $\bullet$ in Fig. 7 are also not connected

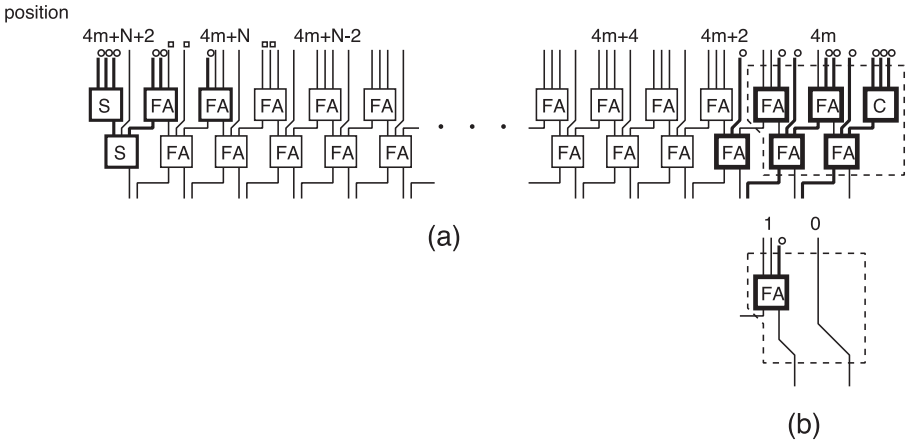



**Fig. 5** Modified 4-2 adder in level 1: (a) $adder(1, m)$, (b) Lowest part of $adder(1, 0)$.

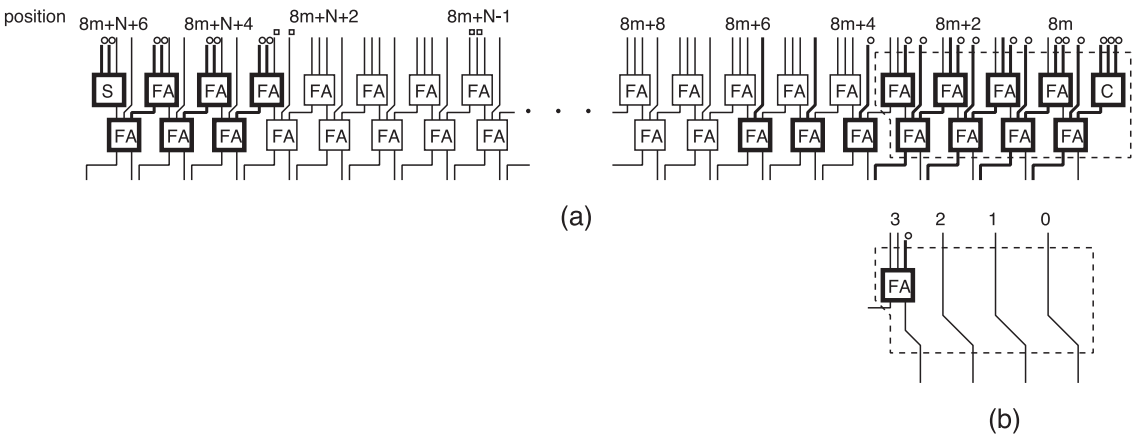



**Fig. 6** Modified 4-2 adder in level 2: (a) $adder(2, m)$, (b) Lowest part of $adder(2, 0)$.

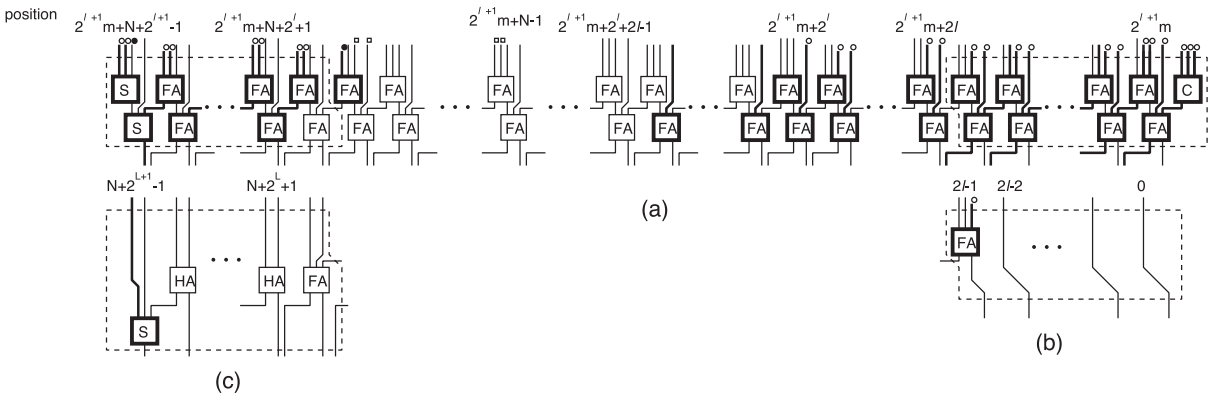



**Fig. 7** Modified 4-2 adder in level $l$ ($l \geq 3$): (a) $adder(l, m)$, (b) Lowest part of $adder(l, 0)$ (c) Upper part of $adder(L, 0)$.

with any output terminal.) We need extra input bits to these terminals, as in the case of the adders in level 1.

Each extra bit must be 0 in normal operation and must have an appropriate value in test mode. Since each 4-2 adder is fed with an ai-pattern in test, we can use a copy (or its inverse) of an input bit to the corresponding terminal at a middle position as an extra bit. In $adder(l, m)$, we use the input bits at position $2^{l+1}m + N - 1$ for the extra bits to terminals $a$ and $b$, and use the input bits at position $2^{l+1}m + N + 2^l - 1$ for those to terminals $c$ and $d$. Note that these input bits are from the highest position of the middle part of the preceding adders. Since $2^{l+1}m + N - 1$ and $2^{l+1}m + N + 2^l - 1$ are odd, we use copies of the input bits for odd numbered positions and their inverses for even numbered positions. Positions of

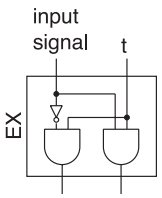



**Fig. 8** An EX cell.

bits to be copied are shown by □ in Figs. 5, 6, and 7.

In order to produce a copy and its inverse of an input bit in test mode, we introduce an 'EX' cell which is shown in Fig. 8. An EX cell takes AND of the input bit and signal $t$, as well as AND of the inverse of the input bit and $t$, where $t$ is an additional external input signal for test and is 0 in normal operation. In normal operation, since $t = 0$ and therefore the extra bits are 0, the adder tree consisting of the modified 4-2 adders works as same as the one consisting of the original 4-2 adders shown in Fig. 3.

### 4.2 Generation of Alternately Inverted Patterns by PPGs

Now we consider the partial product generators (PPGs) and the connection between PPGs and the 4-2 adder tree.

Each PPG generates a partial product $P_j = X \cdot y_j \cdot 2^j$ in normal operation. We label each PPG as $PPG(j)$ according to the corresponding multiplier bit $y_j$. Each PPG consists of $N$ 'PG' cells which takes AND of a multiplicand bit and a multiplier bit. The PG cell at position $k$ of $PPG(j)$ takes AND of $x_{k-j}$ and $y_j$.

We connect the output terminals of the PG cells at position $k$ of $PPG(4m)$, $PPG(4m + 1)$, $PPG(4m + 2)$, and $PPG(4m + 3)$ $(0 \leq m \leq \frac{N}{4} - 1)$ to input terminals $a$, $b$, $c$, and $d$ of adder block $k$ of $adder(1, m)$, respectively.

In order to produce ai-patterns in test mode, we separate the PG cells of $PPG(j)$ into two groups, i.e., the PG cells receiving $x_i$ of even $i$ and those receiving $x_i$ of odd $i$. In test mode, we feed the former with $y_j$, while the latter with the inverse of $y_j$. To produce the inverse of $y_j$ in test mode, we introduce a 'CI' (controlled inverter) cell which takes EXOR of $y_j$ and the mode signal $t$. Figure 9 shows the modified PPG.

When we intend to generate ai-pattern $(\alpha, \beta, \gamma, \delta)_{ai}$ ($\alpha$, $\beta$, $\gamma$, $\delta \in \{0, 1\}$) for $adder(1, m)$, we let $X$ be $[11 \cdots 1]$ (all 1) and let $y_{4m}$, $y_{4m+1}$, $y_{4m+2}$, and $y_{4m+3}$ be $\alpha$, $\bar{\beta}$, $\gamma$, and $\bar{\delta}$, respectively.

Let us consider the 4-level 4-2 adder tree with PPGs of a 32-bit multiplier as an example. The test pattern that feeds $ai_0$ to $adder(4, 0)$ is $X$ = [1111 1111 1111 1111 1111 1111 1111 1111], $Y$ = [1001 1011 1010 0101 0101 1010 0010 0110] and $t = 1$.

### 4.3 Test Set and Fault Propagation of a 4-2 Adder Tree with PPGs

We adopt the cell fault model and treat FAs, C cells, S cells, PG cells, CI cells and EX cells as cells.

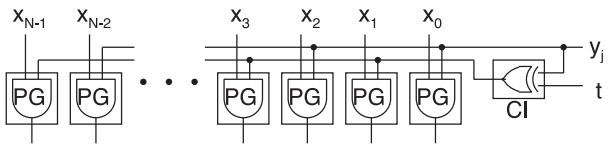



**Fig. 9** Modified PPG.

**Table 3** Test set of the 4-2 adder tree with PPGs of a 32-bit multiplier.

| $X$ | $Y$ (= $[y_{31} \ldots y_0]$) | $t$ |
|---|---|---|
| all 1 | 1001 1011 1010 0101 0101 1010 0010 0110 | 1 |
| all 1 | 0110 0100 0101 1010 1010 0101 1101 1001 | 1 |
| all 1 | 0100 1101 1011 0010 1001 1011 1010 0101 | 1 |
| all 1 | 1011 0010 0100 1101 0110 0100 0101 1010 | 1 |
| all 1 | 0010 0110 1001 1011 1101 1001 0110 0100 | 1 |
| all 1 | 1101 1001 0110 0100 0010 0110 1001 1011 | 1 |
| all 1 | 1111 1111 1111 1111 1111 1111 1111 1111 | 1 |
| all 1 | 0000 0000 0000 0000 0000 0000 0000 0000 | 1 |
| all 1 | 1010 0101 1101 1001 0100 1101 1011 0010 | 1 |
| all 1 | 0101 1010 0010 0110 1011 0010 0100 1101 | 1 |
| all 0 | 0000 0000 0000 0000 0000 0000 0000 0000 | 0 |
| all 0 | 1111 1111 1111 1111 1111 1111 1111 1111 | 0 |
| all 1 | 0000 0000 0000 0000 0000 0000 0000 0000 | 0 |
| all 1 | 1111 1111 1111 1111 1111 1111 1111 1111 | 0 |

FAs, C cells and S cells are fed with their all input patterns by the 10 test patterns considered above. We need four more test patterns for feeding PG cells, CI cells and EX cells with their all input patterns. They are ($X = [00 \cdots 0]$ (all 0), $Y = [00 \cdots 0]$ (all 0) and $t = 0$), ($X = [00 \cdots 0]$ (all 0), $Y = [11 \cdots 1]$ (all 1) and $t = 0$), ($X = [11 \cdots 1]$ (all 1), $Y = [00 \cdots 0]$ (all 0) and $t = 0$), and ($X = [11 \cdots 1]$ (all 1), $Y = [11 \cdots 1]$ (all 1) and $t = 0$). Table 3 shows the test set (14 patterns) for the 4-2 adder tree with PPGs of a 32-bit multiplier.

Now, we show that all faults are observed at the output of the multiplier.

The effect of a fault in a CI cell propagates to $\frac{N}{2}$ positions of the corresponding PPG and produces an erroneous value in the partial product except the cases of $X = [00 \cdots 0]$ (all 0). It is obvious that the error in the value propagates to the final adder and can be observed at the output of the multiplier. Note that all CI cells are fed with all input patterns by only the test patterns with $X = [11 \cdots 1]$ (all 1).

A fault in a cell of the other types (PG, FA, C, S and EX) causes an erroneous value in the corresponding partial product or the output of the corresponding 4-2 adder. The error is to propagate toward the final adder. Because of the tree structure of the 4-2 adder tree, it propagates through only one 4-2 adder in each level. Let us consider that the error has propagated to $adder(l, m)$.

When $t = 0$ or the effect of the fault has not reached the EX cells, the error propagates to the succeeding adder in level $l + 1$. Here, the effect of the fault is seen as opposite values on signal lines than the case without the fault, while the error is the difference in the value of the output of the 4-2 adder. When a value is represented in carry-save form, there is a possibility that the opposite values in signal lines do not change the value. Thus, we distinguish the effect of

the fault from the error.

When the effect of the fault has reached the EX cells for $c$ and $d$ inputs, it propagates to the lowest part of the adder. The considerd EX cells are at position $2^{l+1}m + N + 2^l - 1$. The arrived error is larger than $2^{(2^{l+1}m+N+2^l-2l)}$, because the effect forwards at most two positions in each adder. On the other hand, the value produced by the EX cells is less than $2^{(2^{l+1}m+2^l+1)}$. Therefore, the error cannot be canceled and propagates to the succeeding adder. Furthermore, the effect having propagated to the lowest part cannot reach any EX cell of the succeeding adders, because the delivered positions are much lower than positions of EX cells of the succeeding adders.

When the effect has reached the EX cells for $a$ and $b$ inputs, it propagates to the highest and the lowest part of the adder. Therefore, the error cannot be canceled and propagates to the succeeding adder. The effect having propagated to the lowest part cannot reach any EX cell of the succeeding adders. The effect having propagated to the highest part also cannot affect any EX cell of the succeeding adders, because the delivered positions are higher than positions of EX cells of the succeeding adders.

Thus, the error cannot be canceled at any level. The error propagates to the final adder and can be observed at the output of the multiplier.

Therefore, the proposed 4-2 adder tree with PPGs is testable with 14 patterns regardless of its operand size under the cell fault model. Note that the easily testable 4-2 adder tree proposed in [6] requires $4\log_2 N + 3$ patterns, and that proposed in [7] requires $4\log_2 N + 6$ patterns.

### 4.4 Easily Testable Multiplier

Here, we consider an easily testable multiplier using the proposed C-testable 4-2 adder tree with PPGs. Although we could construct a C-testable multiplier by using a ripple carry adder as the final adder as in [7], we do not think this is a good way because it increases the time complexity of the whole multiplier to $O(N)$ and spoils the merit of using the 4-2 adder tree. As mentioned in [6], we should use a fast adder with time complexity of $O(\log N)$ as the final adder.

We may construct an easily testable high-speed multiplier by using a previously proposed easily testable high-speed adder [11]–[16] as the final adder. Unfortunately, we do not have an excellent way to feed test patterns to the final adder through the 4-2 adder tree. Therefore, we insert extra isolation hardware (multiplexor) at the inputs of the final adder, so that we can feed test patterns directly. Of course, this solution will not only increase the hardware overhead but also will increase the delay of the multiplier. Development of an excellent way to feed test patterns to the final adder through the 4-2 adder tree with PPGs, as well as development of a more efficient easily testable high-speed adder, is left as a future work.

## 5. Hardware Overhead and Delay Overhead

We estimate hardware overhead of the 4-2 adder tree with PPGs by an equivalent number of 2-input NAND gates. For example, in CMOS technology without transmission gates, a 2-input EXOR gate and a 2-input NAND gate can be realized by 10 transistors and 4 transistors, respectively. Therefore, we consider an CI cell is 2.5 gate equivalents. We use gate equivalents of cells in Table 4 for overhead estimation.

We show the estimation of hardware overhead in Table 5. In the table, "Original 4-2 adder tree with PPGs" denotes the gate equivalents of an original 4-2 adder tree with PPGs. "Proposed 4-2 adder tree with PPGs" denotes the gate equivalents of the 4-2 adder tree with PPGs shown in this paper. As the operand size $N$ increases, the ratio of hardware overhead decreases. Hardware overhead of the 4-2 adder tree with PPGs for a 64-bit multiplier is about 15%.

We also estimate delay overhead of the proposed 4-2 adder tree with PPGs. Compared to an original PPGs (that generate partial products only by PG cells), the delay of the PPGs is increased by the delay of one CI cell (EXOR gate). In normal operation, the worst delay time of the tree is almost the same as that of an original one, because their logic levels are the same. Therefore, delay overhead of the 4-2 adder tree with PPGs is about one EXOR gate delay.

To aquire actual overhead by the proposed modification, we synthesized design of the proposed 4-2 adder tree with PPGs and design of the original one. We used Synopsis design compiler to synthesize designs and adopted the Rohm 0.18 $\mu m$ CMOS process cell library provided by Kyoto University. Table 6 shows synthesis results. When synthesising designs, we used timing constraint options so that both designs have the same delay time for each operand size $N$. Even under the timing constraints, ratio of hardware overhead is relatively close to that of hardware overhead in Table 5 except operand size $N = 16$. Note that when strong timing constraints are used, because of one CI cell delay in

**Table 4** Gate equivalent of each cell for overhead estimation.

| Cell | Gate equivalents |
|---|---|
| FA | 8.5 |
| HA | 4.0 |
| C | 3.5 |
| S | 5.0 |
| PG | 1.5 |
| CI | 2.5 |
| EX | 3.0 |

**Table 5** Hardware overhead estimation of the 4-2 adder tree with PPGs.

| $N$ | C-testable 4-2 Adder Tree with PPGs (#of gates) | Original 4-2 Adder Tree with PPGs (#of gates) | Overhead |
|---|---|---|---|
| 16 | 2996.0 | 2346.0 | 27.7% |
| 32 | 12254.0 | 10103.0 | 21.3% |
| 64 | 47315.0 | 41336.0 | 14.5% |
| 128 | 181184.0 | 165985.0 | 9.2% |

**Table 6** Synthesis results of the 4-2 adder tree with PPGs.

| $N$ | C-testable 4-2 Adder Tree with PPGs (Area [$\mu m^2$] / Delay [ns]) | Original 4-2 Adder Tree with PPGs (Area [$\mu m^2$] / Delay [ns]) | Hardware Overhead |
|---|---|---|---|
| 16 | 36014 / 2.00 | 24676 / 2.00 | 45.9% |
| 32 | 116031 / 3.00 | 97545 / 3.00 | 19.0% |
| 64 | 437524 / 4.00 | 393668 / 4.00 | 11.1% |
| 128 | 1692769 / 5.00 | 1589592 / 5.00 | 6.5% |

the proposed PPGs, hardware overhead becomes larger than that in Table 5 to achieve the same delay time to the original design.

The hardware overhead is much smaller than that of the easily testable 4-2 adder tree proposed in [6] where all 4-2 adders in the tree are of $2N$-bit in length. By the mechanism of producing the extra bits required at the end parts of each 4-2 adder, we can reduce the hardware overhead drastically. (The hardware overhead of the easily testable 4-2 adder tree proposed in [7] is not clear, because the method to treat the positions in the both end parts of each 4-2 adder that have less than 4 input bits is not shown.)

As stated in the end of the previous section, insertion of extra isolation hardware at the inputs of the final adder will further increase the hardware overhead and delay overhead.

## 6. Conclusion

We have shown a C-testable 4-2 adder tree with PPGs for a high-speed multiplier. The proposed 4-2 adder tree has recursive structure, and an efficient test set for it is obtained by recursive generation process using special patterns. By using a previously proposed easily testable adder as the final adder, we can construct an easily testable high-speed multiplier. Development of an excellent way to feed test patterns to the final adder through the 4-2 adder tree with PPGs, as well as development of a more efficient easily testable high-speed adder, is left as a future work.

Although we assumed that operand size $N$ is a power of 2, the proposed method can be applied to any $N$. Note that when $N$ is odd, we need a dummy partial product.

## Acknowledgment

The authors thank Associate Professor Kazuyoshi Takagi of Nagoya University for his comments and discussions.

This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc.

### References

[1] W.H. Kautz, "Testing for faults in cellular logic arrays," Proc. Eighth Ann. Symp. Switching and Automata Theory, pp.161–174, 1967.
[2] J.P. Shen and F.J. Ferguson, "The design of easily testable VLSI array multipliers," IEEE Trans. Comput., vol.C-33, no.6, pp.554–560, June 1984.
[3] A. Chatterjee and J.A. Abraham, "Test generation, design-for-testability and built-in self-test for arithmetic units based on graph labeling," J. Electronic Testing, vol.2, pp.351–372, Nov. 1991.
[4] D. Gizopoulos, D. Nikolos, A. Paschalis, and C. Halatsis, "C-testable modified-booth multipliers," J. Electronic Testing, vol.8, no.3, pp.241–260, June 1996.
[5] K.O. Boateng, H. Takahashi, and Y. Takamatsu, "Design of C-testable modified-booth multipliers," IEICE Trans. Inf. & Syst., vol.E83-D, no.10, pp.1868–1878, Oct. 2000.
[6] B. Becker, "An easily testable optimal-time VLSI-multiplier," Acta Informatica, vol.24, pp.363–380, 1987.
[7] P. Zeng, Z. Mao, Y. Ye, and Y. Deng, "Test pattern generation for column compression multiplier," Proc. Seventh Asian Test Symposium, pp.500–503, 1998.
[8] A. Chatterjee and J.A. Abraham, "On the C-testability of generalized counters," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.CAD-6, no.5, pp.713–726, Sept. 1987.
[9] N. Kito and N. Takagi, "A design method of easily testable multipliers with various structures of partial product adder," IEICE Trans. Inf. & Syst. (Japanese Edition), vol.J91-D, no.10, pp.2478–2486, Oct. 2008.
[10] J. Vuillemin, "A very fast multiplication algorithm for VLSI implementation," Integration the VLSI Journal, vol.1, pp.39–52, 1983.
[11] B. Becker, "Efficient testing of optimal time adders," IEEE Trans. Comput., vol.37, no.9, pp.1113–1121, Sept. 1988.
[12] B. Becker, R. Drechsler, and P. Molitor, "On the generation of area-time optimal testable adders," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.14, no.9, pp.1049–1066, Sept. 1995.
[13] R.D. Blanton and J.P. Hayes, "Testability of convergent tree circuits," IEEE Trans. Comput., vol.45, no.8, pp.950–963, Aug. 1996.
[14] W.R. Moore, "Minimal C-testable tests for block-CLA adders," Int. J. Electron., vol.85, pp.611–628, 1998.
[15] R.D. Blanton and J.P. Hayes, "On the design of fast, easily testable ALU's," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.8, no.2, pp.220–223, April 2000.
[16] D. Gizopoulos, M. Psarakis, A. Paschalis, and Y. Zorian, "Easily testable cellular carry lookahead adders," J. Electronic Testing, vol.19, pp.285–298, June 2003.

**Nobutaka Kito** received the B.E., M.I.S. and Dr. of Information Science degrees in information engineering from Nagoya University, Nagoya, Japan, in 2004, 2006, and 2009, respectively. His current interests include design for testability, computer arithmetic, and CAD algorithms.

**Kensuke Hanai** received the B.E. and M.E. degrees in information engineering from Nagoya University, Nagoya, Japan, in 2001 and 2003, respectively.

**Naofumi Takagi** received the BE, ME, and PhD degrees in information science from Kyoto University, Kyoto, Japan, in 1981, 1983, and 1988, respectively. He joined Kyoto University as an instructor in 1984 and was promoted to an associate professor in 1991. He moved to Nagoya University, Nagoya, Japan, in 1994, and promoted to a professor in 1998. He returned to Kyoto University in 2010. His current interests include computer arithmetic, hardware algorithms, and logic design. He received Japan IBM Science Award and Sakai Memorial Award of the Information Processing Society of Japan in 1995, and The Commendation for Science and Technology by the Minister of Education, Culture, Sports, Science and Technology of Japan in 2005.