Optimal-Time Multipliers and C-Testability

Bernd Becker
Fachbereich 20 – Informatik
Johann Wolfgang Goethe-Universität
D-6000 Frankfurt/Main
West Germany

Joachim Hartmann
Lehrstuhl Prof. Hotz
Fachbereich 14 – Informatik
Universität des Saarlandes
D-6600 Saarbrücken
West Germany

Abstract

After a brief review on testability aspects of parallel arithmetical units we focus on n-bit multipliers and especially consider a class of Wallace tree multipliers made suitable for VLSI design by Vuillemin and Luk [VuLu].

It is shown that for these circuits both optimal running time and optimal test complexity can be obtained. A complete test set according to the single cellular fault model is presented. (In this case, the single cellular fault model is superior to the classical single stuck-at model.) The proposed test only consists of 17 patterns for all n. Hence, the multiplier is C-testable, i.e. it can be tested by a number of input combinations which is independent of the number of cells in the circuit.

The extra test hardware is very small. Only two additional ports and $n-2$ internal connections are necessary.

Keywords: Testability of regular structures, parallel multipliers, test complexity, C-testability.

1 Introduction

Many parallel architectures result in regular designs, since they often consist of a large number of similiar subcircuits with small, simple interfaces. Moreover, regular structures have become more and more attractive with the advent of very large scale integration (VLSI). In many cases their geometrical regularity facilitates placement and routing problems, and their hierarchy may allow a reduction in the immense number of data that occur during the design process of a large circuit.

As indicated above regularity offers a lot of advantages in *designing* circuits. We now turn to the question how it can be used for *testing*. The examination of this issue yields test strategies for whole classes of circuits instead of some patterns for one special implementation as is the case in usual automatic test pattern generation. Well-known examples for regular structures are iterative logic arrays (ILAs), programmable logic arrays (PLAs), and arithmetic units. For these often used circuits testability aspects have been considered in [Fr,SrHa,ChPa] (ILAs), [Fu,RoRa] (PLAs), and [FeSh,BhHa,Be,BeSp,ChAb,Ho] (arithmetic units).

Here, we are interested in parallel architectures realizing arithmetic functions. For these circuits two important quality measures are performance and test size. In previous papers, efforts were made to optimize one of these measures, but optimality in both was not obtained. So the question arises whether this trade off between running time and number of test patterns is inherent or is caused by "unskillful" implementations. As we will see in this paper, it is possible to combine optimal running time with optimal test size for a parallel multiplier.

Good testability for adders and multipliers can be achieved by combining full-adders to ILAs. In this case, the known test techniques for ILAs can easily be applied. By this strategy, Ferguson and Shen have succeeded in constructing a C-test for a n-bit array multiplier ([FeSh]). That means, the number of test patterns is a constant independent of n.

Other examples of circuits that are C-testable and have time delays linear in n are given in [ChAb].

Hong has also proposed a multiplier with running time $O(n)$ ([Ho]). This circuit requires $3n + 60$ test patterns while it manages with the minimum number of adder cells if the extra test hardware is not considered.

However, the major disadvantage of these structures is the suboptimal running time that is at least linear with the operands' bit width n.

High-speed realizations of arithmetic functions take time $O(\log n)$. Battacharya and Hayes have introduced

fast and easily testable arithmetic units that are based on tree-like structures ([BhHa]). By providing extra test hardware in each basic cell they have been able to apply a level-by-level test strategy that results in tests of size $O(\log n)$.

In a previous paper ([Be]), a fast n-bit multiplier has been examined. It is a modified Wallace tree ([Wa]), which has been adapted for VLSI design by Vuillemin and Luk ([VuLu]). The test presented consists of $4 \log n + 3$ input combinations and only requires slight hardware modifications.

In this paper, we propose a C-test for this multiplier. 17 patterns form a complete test according to the single cellular fault model (which is superior to the single stuck-at fault model in this case [Be]). Thus we succeed in combining (asymptotically) optimal running time with C-testability in the case of a multiplier.

The very regular design of the multiplier in [VuLu] is achieved by introducing redundant parts with redundant primary inputs. The C-test is developed under the assumption that these redundant inputs and certain carry inputs are accessible. During normal operation these inputs have to be set to zero. Thus in a design that need not supply testability they could be connected to ground. An analysis of the employed patterns reveals that it is possible to dispense with all but two of these inputs if $n-2$ internal connections are provided. This test overhead is very small compared to other easily testable multipliers ([FeSh]: n full-adders and 7 extra inputs, [ChAb]: $n-1$ XOR gates and 5 extra inputs, [Ho]: n OR gates, n^2 transfer gates, and 1 extra input, [BhHa]: extra hardware in each basic cell).

This paper is structured in the following way:

In section 2, the logic design of the tree multiplier is considered.

Section 3 defines the fault model and gives the main ideas for the construction of the test. (The detailed discussion of the test patterns can be found in the appendix.) One of the key ideas is to use a sufficient criterion for C-testability in tree-like structures which is developed in Section 3.2.

2 The multiplier

2.1 Logical structure

For the construction of the tests knowledge of the multiplier's structure is necessary. So we start with a logical description of the circuit.

Let $a_n \cdots a_1$ and $b_n \cdots b_1$ be the binary representations[1] of the two numbers a and b. For simplicity we assume n to be a power of 2. The product of a and b is

[1] a_n and b_n are the most significant bits

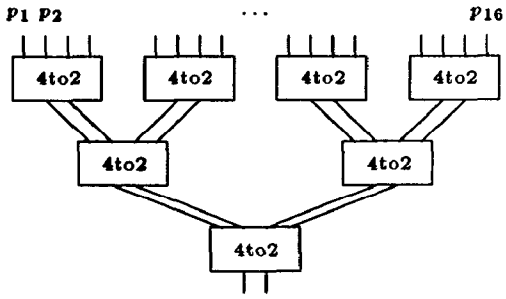

Figure 1: Reduction part of a 16-bit multiplier

equivalent to the sum of the *partial products* $p_1, \ldots, p_n$, which are given by $p_i = a \cdot b_i \cdot 2^{i-1}$. Fast multiplication is accomplished by summing up the partial products by tree-like structures. For this we use *4to2-reductions*[2] which receive four $2n$-bit numbers as inputs and compute two $2n$-bit numbers such that the sum of the inputs is equal to the sum of the outputs. They are arranged as a binary tree of depth $\log_2(n) - 1$, which is fed with the partial products (Figure 1). Thus the circuit computes two $2n$-bit numbers whose sum equals the product $a \cdot b$. In the sequel, the subcircuit containing the 4to2-reductions is said to be the *reduction part*. Since the delay of a 4to2-reduction is independent of n and the computation of the partial products can be performed in constant time, the circuit takes the (asymptotically) optimal time $O(\log(n))$. We will investigate its testability in section 3.

A final $2n$-bit adder has to be connected to the multiplier to deliver the result of the multiplication represented as one number. This should be a fast adder with running time $O(\log n)$ (see e.g. [BrKu], [LaFi], [BeKo], [BeSp]) in order to preserve the asymptotically optimal delay of the multiplier.

To make the multiplier's construction clearer, we will describe in the next two sections how the partial products are generated and how the reduction part is built up.

2.2 Generation of the partial products

Figure 2 gives the detailed structure of the part that generates the partial products. In the diagram the bullets represent the logical AND of the signals encountering there. The binary representation of p_i can be found in the i^{th} row of the AND array, it is $p_{i,2n} \cdots p_{i,1}$. Formally, $p_{i,j}$ is given by $p_{i,j} = b_i \cdot a_{j-i+1}$. The a-inputs $a_{2n}, \ldots, a_{n+1}$ and $a_0, a_{-1}, \ldots, a_{-n+2}$ are so-called *redundant* inputs, which are set to zero during normal operation. They are introduced to allow a very regular design that can easily be adapted to a generator for VLSI design systems, as it was done for CADIC ([HBKMO]) and VENUS ([HNS]).

[2] A 4to2-reduction consists of two carry save adders.

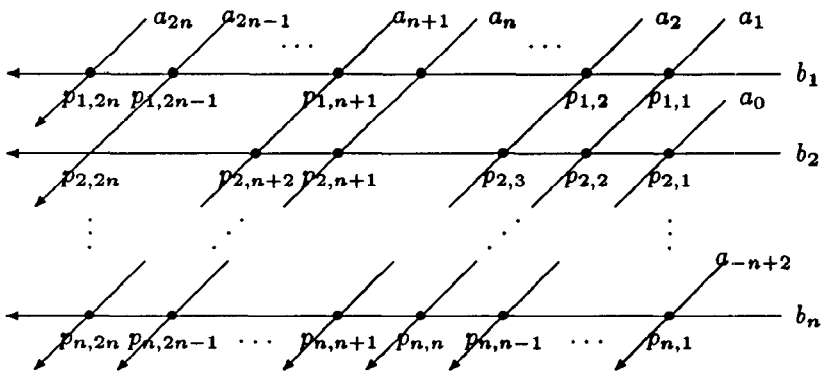

Figure 2: Generation part

2.3 The reduction part

We build 4to2-reductions for k-bit numbers using 1-bit reductions which take carries into account. These basic cells are called *4to2-cells* and consist of two full-adders each (Figure 3). A full-adder (FA) produces a sum bit and a carry bit of three input bits. We call FA_1 in Figure 3 the *upper* full-adder and FA_2 the *lower* one. If we arrange k 4to2-cells as a chain by linking the carries, the result will be a k-bit 4to2-reduction. We call the primary carry inputs of a 4to2-reduction *initial* and the primary carry outputs *terminal* carry lines. When the initial carries are set to zero, the circuit reduces four numbers to two. Note that each signal at most passes two full-adders. Thus the computation performed by a k-bit 4to2-reduction takes the time of two full-adder delays and does not depend on k. As was already mentioned in section 2.1, we construct the multiplier's complete reduction part by arranging $2n$-bit 4to2-reductions as a tree. From the view of bit slices, there are $2n$ cascaded single trees (*reduction trees*) of depth $\log_2(n) - 1$. A reduction tree of depth 1 consists of one 4to2-cell. Inductively, a reduction tree of depth $k+1$ is defined as follows: its root is a 4to2-cell whose left and right input bit pairs are fed by the output bit pair of a reduction tree of depth k each. The output bit pair of a reduction tree is the output bit pair of its root. In the sequel, we look upon the root's level as the lowest one and the leaves' level as the highest one. Figure 4 shows the cascadation of the reduction trees in a perspective manner. The horizontal (carry) leads are omitted. The last tree (which produces the terminal carries) computes the most significant bits of both output numbers. We denote it by $tree_{2n}$ and, analogeously, the tree for the j^{th} output bit pair by $tree_j$ for $j = 1, \ldots, 2n$. The values of the reduction trees' vertical inputs can also be seen in Figure 4. The input bits $p_{i,j}$ correspond to the entries $p_{i,j}$ in the array of Figure 2. Identification of the signals shows that the j^{th} column of the AND array computes the input vector of $tree_j$.

A planar design can be accomplished by projecting this 'three dimensional structure' in a plain. Figure 5 shows the result. Each column (bit slice) corresponds to a reduction tree augmented by the AND gates that produce its input vector. The square cells are the AND gates while the other ones are the full-adders (for a formal description of the layout see [Be]).

3 Construction of a C-test

3.1 The fault model

We assume that a single basic cell (AND cell or full-adder) in the circuit can permanently fail in an arbitrary manner as long as its behavior remains combinational (*single cellular fault model*). With regard to this assumption, it is necessary to apply all possible input combinations to each cell and to propagate any faulty signal produced at the cell's outputs to a primary output of the multiplier.

It can be shown (see [Be]) that in the case of the presented multiplier the single cellular fault model is more general than the often used single stuck-at fault model.

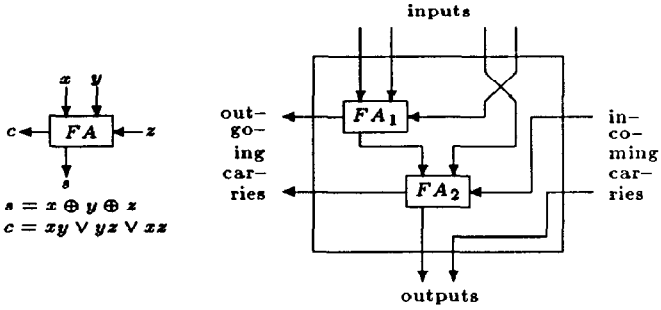

Figure 3: 4to2-cell

3.2 Construction of the test

The next two sections give the main ideas for the construction of the C-test. In the appendix there is a detailed description of the patterns.

We will proceed in the following way:

First of all, we show that fault propagation is very simple for the multiplier. By investigating constant testing in tree structures we succeed in constructing a C-test for the multiplier's reduction part. This test, however, cannot be applied by setting the primary inputs. Therefore, we examine how to modify the test to get an applicable one.

We concentrate on tests for the full-adders because the AND gates can easily be controlled by setting the multiplier's inputs.

Now we start with the details:

Lemma 1 ([Be]) *If exactly one basic cell fails for any input combination of the circuit, then at least one faulty signal is propagated to a primary output of the multiplier on applying this input pattern.*

Proof: We only give a sketch of the proof. When a full-adder receives exactly one faulty signal, it propagates this fault by its sum output (s) which computes the parity of the three inputs. The circuit's wiring ensures, in fact, that for every wrong behavior of a basic cell, there exists a full-adder in the next level which gets exactly one faulty input. Inductively, it follows that the fault propagates until it reaches a primary output. ■

Lemma 1 is very useful. It asserts that to test a basic cell for any assignment to its inputs, it suffices to choose any pattern that applies this assignment to the cell. It is not necessary to take a special input combination for propagating the fault to a primary output.

To develop a sufficient criterion for C-testability in tree structures, we consider a cell C realizing a Boolean function $f : B^{2k} \rightarrow B^k$ where $k \in \mathbf{N}$ and $B = \{0, 1\}$. The inputs of C are assumed to be separated in k left and k right ones. Now it is possible to construct a binary tree of several cells of type C by connecting the outputs of cells to left or right inputs of other cells. We denote a transition of C by $t = (L, R, O) \in B^k \times B^k \times B^k$ where L (R) is applied to the left (right) inputs, and O is the output implied. For illustration we will sometimes use a symbolic representation of a transition as is shown in Figure 6.

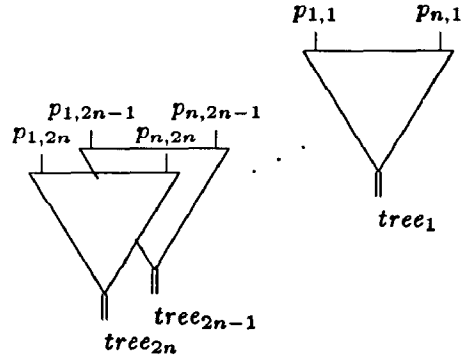

Figure 4: Reduction trees forming the multiplier's reduction part

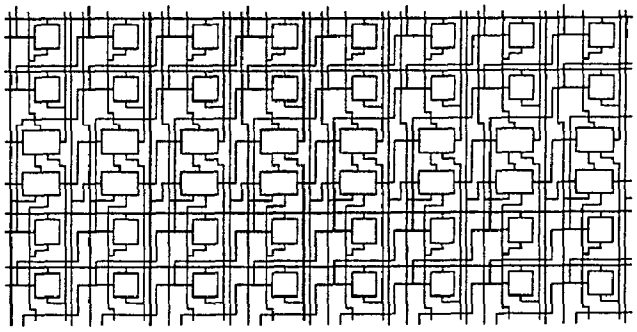

Figure 5: Layout of a 4-bit multiplier

We are now able to formulate the next lemma:

Lemma 2 *Let $T = \{t_1, t_2, \ldots, t_m\}$ with $t_i = (L_i, R_i, O_i)$ for $i = 1, \ldots, m$ be a set of transitions of C.*

If there are permutations $l, r : \{1, \ldots, m\} \rightarrow \{1, \ldots, m\}$ which fulfill

$$L_i = O_{l(i)} \text{ and } R_i = O_{r(i)} \text{ for } i = 1, \ldots, m$$

(i.e. the output values permutatedly occur on the left and right inputs), then the transitions $t_1, \ldots, t_m$ can be applied to all cells of any binary tree of cells C by m patterns.

Proof: First of all, we present a scheme to construct an assignment to the nodes of the tree that only uses transitions of T. We apply any $t \in T$, say t_1, to the tree's root. In this case, L_1 (R_1) is the output of the left (right) subtree over the root. Let $x = l(1)$ and $y = r(1)$, then $L_1 = O_x$ and $R_1 = O_y$. So t_x (t_y) can be applied to the root's left (right) parent node. An assignment to all nodes can be made by going on in this way.

Beginning with $t_2, \ldots, t_m$ instead of t_1 as value of the root we are able to construct m different assigments to

Figure 6: Cell C and symbolic representation of a transition t

the tree. We show that for any node the m patterns apply m different transitions. Obviously this is true for the root. This implies m different output values of the root's left and right parent node. Thus it inductively follows for all nodes that the m patterns imply m different transitions. Since only the transitions $t_1, \ldots, t_m$ are used, all transitions are applied to each node. ∎

Example 1: Let C be a cell that computes the logical exclusive OR (XOR) of two bits. For C there exist the following four transitions: $t_1 = (0,1,1)$, $t_2 = (1,0,1)$, $t_3 = (0,0,0)$, and $t_4 = (1,1,0)$. The permutations l and r with $l(1) = 4, l(2) = 2, l(3) = 3, l(4) = 1, r(1) = 1, r(2) = 4, r(3) = 3$ and $r(4) = 2$ fulfill the conditions of Lemma 2. Since every fault is propagated to the primary output, a tree of XOR cells is C-testable by four patterns. The four assignments of Figure 7, which are constructed according to the proof of Lemma 2, test every XOR cell in a tree of depth 3 for all input combinations. ∎

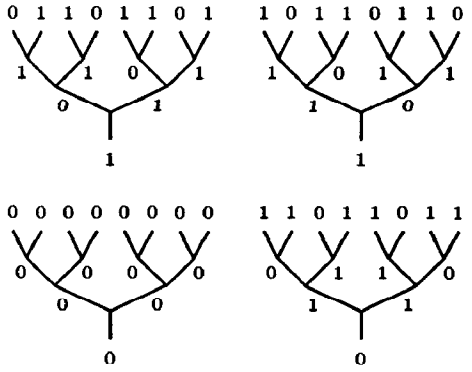

Figure 7:

Now we try to apply the above lemma to the multiplier's reduction part. Here the cell C is a 4to2-cell. To denote the carry lines, we define a transition of a 4to2-cell as a tupel $t = (L, R, O, C_{in}, C_{out}) \in (B \times B)^5$ where L, R, and O are as introduced before, and C_{in}, C_{out} stand for incoming and outgoing carries (see Figure 8). It is clear that the last lemma can be applied to the 4to2-cell if the carry lines are ignored.

To keep the test set's cardinality low, we want to apply certain transitions not only in one reduction tree but simultaneously in many trees. This can be done by (preliminary) confining to such transitions whose incoming and outgoing carries are the same, i.e. transitions $t = (L, R, O, C_{in}, C_{out})$ where $C_{in} = C_{out}$. They are called *1-repeatable transitions*

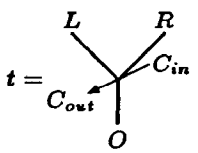

Figure 8: Transition of a 4to2-cell

Example 2: Consider the 1-repeatable transitions given in Figure 9. They satisfy the conditions of Lemma 2 if the permutation l and r are $l(1) = 1$, $l(2) = 2$, $r(1) = 2$ and $r(2) = 1$. For the values of L, R, O, C_{in} and C_{out}, we use the abbreviations $T_0 = (0,0)$, $T_1 = (0,1)$, $T_2 = (1,0)$ and $T_3 = (1,1)$ ($T_i = (x,y)$ iff (x,y) is the binary representation of i for $i = 0,1,2,3$).

The method presented in the proof of Lemma 2 allows applying the transitions t_1 and t_2 to all 4to2-cells of one single tree by two patterns. ∎

Figure 9:

On condition that the initial carry lines are conveniently set, the same assignment can simultaneously be applied to all reduction trees by means of 1-repeatable transitions. In this way, we succeed in generating some but not all input combinations to the full-adders by a number of patterns independent of the multiplier's size. It can be proven that $\Omega(\log n)$ patterns will be necessary to test a n-bit multiplier if only 1-repeatable transitons are used (see [Ha]).

In [Be], $4\log(n)+3$ patterns are shown to be sufficient for a complete test. For this a level-by-level strategy is used. That means, all full-adders of one level are tested by applying the same transition to all 4to2-cells of this level. This is done for each level.

To eliminate the test size's dependence on n, we consider *2-repeatable* transitions. These are pairs of transitions having the property that the outgoing carries of one transition are the same as the other's incoming ones and vice versa, i.e. transitions $t = (L, R, O, C_{in}, C_{out})$ and $t' = (L', R', O', C'_{in}, C'_{out})$ where $C_{in} = C'_{out}$ and $C_{out} = C'_{in}$. 2-repeatable transitions can be regarded as one *double transition*, whose left and right input value are (L, L') and (R, R') respectively and whose output value is (O, O').

Example 3: The three double transitions of Figure 10 again fulfill Lemma 2. Hence, they can be applied to all the 4to2-cells of two adjacent reduction trees by three patterns. As the transitions are 2-repeatable, it is possible to repeat the same assignment in each second tree. If the assignments in odd and even subscripted reduction trees are exchanged, each single transition of the double transitions will be brought to all 4to2-cells of the multiplier. ∎

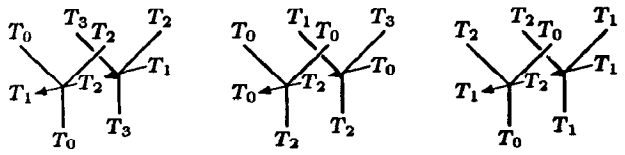

Figure 10:

A detailed analysis shows that for each input combination of a full-adder, there exists a 1- or 2-repeatable transition that brings this combination to the full-adder and that belongs to a set of transitions as described in Lemma 2. But the problem of generating the corresponding patterns via the AND gates remains. This problem is considered in the next section.

3.3 Generation of the test patterns

For the proofs of the following lemmas, it is necessary to know which input vectors a reduction tree receives. In Section 2.3, it was already mentioned that the vertical inputs of $tree_j$ are the entries of the j^{th} column in Figure 2.

In the sequel, we speak of *1- and 2-periodic patterns.* These are the values of the reduction trees that will be caused by applying 1- and 2-repeatable transitions respectively, if the assignment of $tree_i$ equals that one of $tree_{i+1}$ for $i = 1, \ldots, 2n-1$ (1-periodic patterns) and if the assignments of $tree_i$ and $tree_{i+2}$ are the same for $i = 1, \ldots, 2n-2$ (2-periodic patterns) respectively.

Lemma 3 ([Be]) *Any 1-periodic pattern can be achieved by setting the multiplier's primary inputs.*

Proof: 1-periodic patterns result in the same assignment for all reduction trees. It follows that all columns of the AND array have the same values. Thus the binary representation of p_i must be either $0\cdots0$ or $1\cdots1$ for $i = 1, \ldots, n$. These patterns can be generated by setting all a inputs to 1 and $b_i = 0$ if p_i's representation is $0\cdots0$ and $b_i = 1$ otherwise. ■

The generation of 2-periodic patterns is more difficult. There are even patterns that are not achievable at all by assigning the primary inputs.

A successful idea is to generate the required transitions a constant number of levels beneath the leaves. Then the remaining levels can be tested by the level-by-level method. To do that, we classify the generatable 2-periodic patterns. In Figure 11, an achievable assignment of the AND array is depicted. It is obtained by setting all b inputs to 1 and the a inputs to 0 and 1 alternately. For this input combination, all 4to2-cells in the highest level receive T_1 and T_2 resp. as input pairs.

Lemma 4 *Given a 2-periodic pattern. Let* tree *and* tree' *be two adjacent trees. If the inputs for the 4to2-cells in the highest level of* tree *are from the set* $\{T_0, T_1\}$

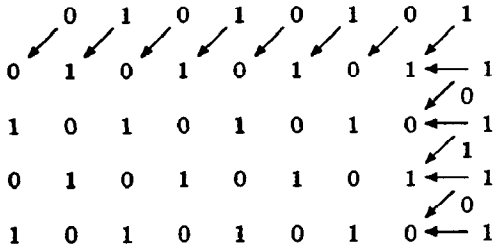

Figure 11: Generatable values by the AND array

and those of tree' *are from* $\{T_0, T_2\}$*, then the pattern is generatable by appropriately setting the primary inputs.*

Proof: The case that *tree* and *tree'* have no T_0 as inputs of 4to2-cells in the highest level is already considered (see Figure 11). The other cases are constructed from the above one by setting some b_i's to zero. ■

If a 2-periodic pattern is not generatable, we can try to apply a pattern that satisfies the condition of the above lemma and yields the desired transitions from some deeper level on.

Example 4: The double transitions of Example 3 have (T_0, T_3), (T_2, T_2) or (T_0, T_1) as left and right inputs. The double transitions of Figure 12 have these values as outputs. Thus these transitions "fit" to those of Example 3 and can be put above them.

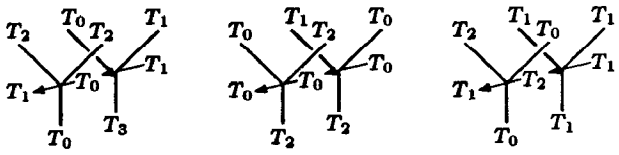

Figure 12:

Furthermore, by conveniently setting the initial carry lines a generatable pattern of period 2 is produced. This follows from Lemma 4 and the fact that the 4to2-cells in the highest level of two adjacent trees now have T_0, T_2 and T_0, T_1 resp. as inputs.

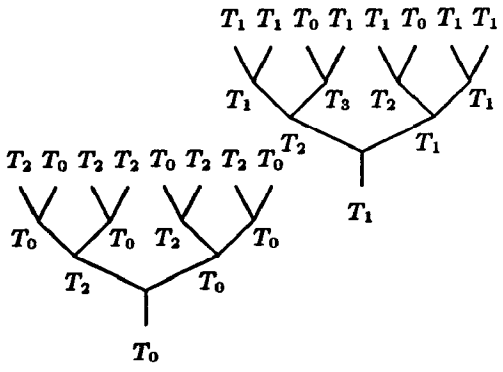

Figure 13: A generatable 2-periodic pattern

Figure 13 shows the situation for two reduction trees where the carries are omitted. The nodes in the two lower levels are labelled by transitions of Example 3. The corresponding values are not generatable by the AND array. Extending by the transitions of Figure 12, however, results in an achievable 2-periodic pattern. ■

These strategies allow proving:

Theorem 5 *The multiplier is C-testable according to the single cellular fault model. 17 patterns are sufficient for a complete test.*

The detailed proof of this theorem is given in the appendix. The 1- and 2-repeatable transitions in the appendix, which exhaustively test the full-adders and fulfill Lemma 2, were found by a computer program.

3.4 Reduction of redundant inputs

So far, we have assumed that the redundant a inputs and the initial carry lines are controllable. This need not be required. Since each second a input gets the same value during testing, the redundant a inputs with even subscripts can be connected. This is also possible for those with odd subscripts. In this way, the redundant a inputs are combined to two new primary inputs, which are set to zero during normal operation.

The presented test patterns force the terminal carry lines to have the same assignment as the initial ones should have. Therefore, it is possible to connect them without affecting the applicability of the test. This does not cause an error during normal operation because the product of two n-bit numbers is a $2n$-bit number. Thus all terminal carries (and the initial ones with them) are set to zero during normal multiplication.

In a design that need not supply testability the redundant a inputs and initial carry lines would be connected to ground.

Corollary 6 *Eliminating* $3(n-1)$ *redundant primary inputs of the multiplier by using* $n-2$ *new internal connections and 2 additonal ports does not affect the multiplier's testability. It remains C-testable with 17 patterns.*

The proposed modifications are shown in Figure 14. In this diagram the new wiring requires more area than necessary. The picture is given in this way for the sake of clarity. Compact symbolic designs which were made for the original and modified version of the multiplier with the chip design system CADIC (see [HBKMO]) show that the hardware overhead in chip area is only about 3%.

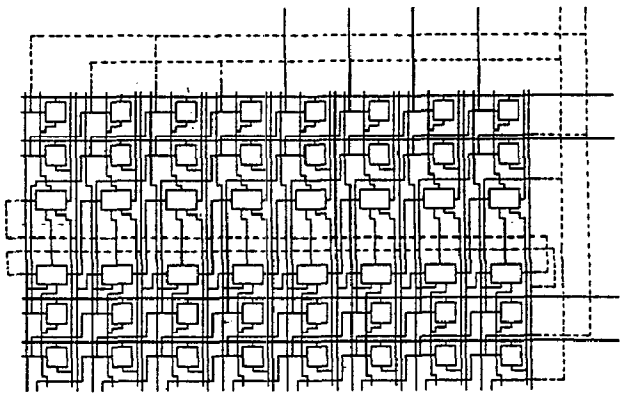

Figure 14: Modified 4-bit multiplier (additional lines are dashed)

4 Summary and concluding remarks

The testability according to the single cellular fault model of a fast modified Wallace tree multiplier has been investigated. A new sufficient criterion for C-testability in tree structures has been developed and applied to the multiplier. It has been shown that the multiplier is C-testable by 17 patterns. Compared to a design that need not supply testability, only some internal wiring and two additional ports are provided.

Further work ([Ha]) has been done for constructing an efficient C-test of 49 patterns for single stuck-open faults.

5 Appendix: Proof of Theorem 5

It has to be shown that all input combinations can be brought to all AND gates and all full-adders by 17 patterns. For this we use the transitions given in Table 1.

The sets $\mathcal{T}_1 = \{t_1\}$, $\mathcal{T}_2 = \{t_2\}$ and $\mathcal{T}_3 = \{t_3, t_4\}$ consist of 1-repeatable transitions and fulfill Lemma 2. So it follows along with Lemma 3 that t_1, t_2, t_3 and t_4 can be applied to all 4to2-cells by four patterns. The corresponding assignments to the full-adders can be seen in Table 1. These are in the notation (x, y, z) (see Figure 3) for the upper full-adders $(0,0,0)$, $(1,1,1)$, $(0,1,0)$, $(1,0,1)$ and for the lower full-adders $(0,0,0)$, $(1,1,1)$, $(1,1,0)$, $(0,0,1)$.

The sets $\mathcal{T}_4 = \{(t_5, t_6), (t_7, t_8), (t_9, t_{10})\}$ (Example 3) and $\mathcal{T}_5 = \{(t_{11}, t_{12}), (t_{13}, t_{14}), (t_{15}, t_{16}), (t_{17}, t_{18})\}$ of 2-repeatable transitions also satisfy the condition of Lemma 2. Unfortunately, it is not possible to obtain these transitions by setting the primary inputs. So we use the strategy described in Example 4. We put the double transitions of $\mathcal{T}_6 = \{(t_{19}, t_{20}), (t_{21}, t_{22}), (t_{23}, t_{24})\}$ above those of $\mathcal{T}_4$. Thus

transition	$(L, R, O, C_{in}, C_{out})$	combinations applied to upper FA	lower FA
t_1	$(T_0, T_0, T_0, T_0, T_0)$	(0,0,0)	(0,0,0)
t_2	$(T_3, T_3, T_3, T_3, T_3)$	(1,1,1)	(1,1,1)
t_3	$(T_1, T_2, T_1, T_1, T_1)$	(0,1,0)	(1,1,0)
t_4	$(T_2, T_1, T_2, T_2, T_2)$	(1,0,1)	(0,0,1)
t_5	$(T_0, T_2, T_0, T_2, T_1)$	(0,0,0)	(0,1,1)
t_6	$(T_3, T_2, T_3, T_1, T_2)$	(1,1,0)	(0,1,0)
t_7	$(T_0, T_0, T_2, T_2, T_0)$	(0,0,0)	(0,0,1)
t_8	$(T_1, T_3, T_2, T_0, T_2)$	(0,1,1)	(0,1,0)
t_9	$(T_2, T_0, T_0, T_2, T_1)$	(1,0,0)	(1,0,1)
t_{10}	$(T_2, T_1, T_1, T_1, T_2)$	(1,0,1)	(0,0,0)
t_{11}	$(T_1, T_0, T_3, T_1, T_0)$	(0,1,0)	(1,0,0)
t_{12}	$(T_0, T_3, T_0, T_0, T_1)$	(0,0,1)	(1,1,0)
t_{13}	$(T_3, T_0, T_1, T_1, T_2)$	(1,1,0)	(0,0,0)
t_{14}	$(T_0, T_1, T_0, T_2, T_1)$	(0,0,1)	(1,0,1)
t_{15}	$(T_0, T_3, T_0, T_0, T_1)$	(0,0,1)	(1,1,0)
t_{16}	$(T_1, T_0, T_3, T_1, T_0)$	(0,1,0)	(1,0,0)
t_{17}	$(T_0, T_1, T_0, T_2, T_1)$	(0,0,1)	(1,0,1)
t_{18}	$(T_3, T_0, T_1, T_1, T_2)$	(1,1,0)	(0,0,0)
t_{19}	$(T_2, T_2, T_0, T_0, T_1)$	(1,0,0)	(1,1,0)
t_{20}	$(T_0, T_1, T_3, T_1, T_0)$	(0,0,1)	(1,0,0)
t_{21}	$(T_0, T_2, T_2, T_0, T_0)$	(0,0,0)	(0,1,0)
t_{22}	$(T_1, T_0, T_2, T_0, T_0)$	(0,1,0)	(1,0,0)
t_{23}	$(T_2, T_0, T_0, T_2, T_1)$	(1,0,0)	(1,0,1)
t_{24}	$(T_1, T_1, T_1, T_1, T_2)$	(0,1,1)	(0,0,0)
t_{25}	$(T_2, T_0, T_3, T_1, T_0)$	(1,0,0)	(1,0,0)
t_{26}	$(T_0, T_3, T_0, T_0, T_1)$	(0,0,1)	(1,1,0)
t_{27}	$(T_2, T_2, T_0, T_0, T_1)$	(1,0,0)	(1,1,0)
t_{28}	$(T_1, T_0, T_3, T_1, T_0)$	(0,1,0)	(1,0,0)
t_{29}	$(T_0, T_0, T_1, T_1, T_0)$	(0,0,0)	(0,0,0)
t_{30}	$(T_1, T_2, T_0, T_0, T_1)$	(0,1,0)	(1,1,0)
t_{31}	$(T_2, T_0, T_0, T_2, T_1)$	(1,0,0)	(1,0,1)
t_{32}	$(T_1, T_1, T_1, T_1, T_2)$	(0,1,1)	(0,0,0)
t_{33}	$(T_0, T_0, T_2, T_2, T_0)$	(0,0,0)	(0,0,1)
t_{34}	$(T_1, T_1, T_0, T_0, T_2)$	(0,1,1)	(0,0,0)
t_{35}	$(T_2, T_0, T_0, T_2, T_1)$	(1,0,0)	(1,0,1)
t_{36}	$(T_1, T_1, T_1, T_1, T_2)$	(0,1,1)	(0,0,0)
t_{37}	$(T_0, T_0, T_0, T_0, T_0)$	(0,0,0)	(0,0,0)
t_{38}	$(T_0, T_1, T_2, T_0, T_0)$	(0,0,1)	(1,0,0)
t_{39}	$(T_2, T_2, T_2, T_2, T_1)$	(1,0,0)	(1,1,1)
t_{40}	$(T_1, T_1, T_1, T_1, T_2)$	(0,1,1)	(0,0,0)
t_{41}	$(T_2, T_2, T_0, T_0, T_1)$	(1,0,0)	(1,1,0)
t_{42}	$(T_1, T_0, T_3, T_1, T_0)$	(0,1,0)	(1,0,0)
t_{43}	$(T_3, T_2, T_1, T_3, T_3)$	(1,1,0)	(0,1,1)
t_{44}	$(T_0, T_1, T_2, T_0, T_0)$	(0,0,1)	(1,0,0)

Table 1: Used transitions

the 4to2-cells in the highest level receive the transitions of $\mathcal{T}_6$ while in the remaining levels the transitions of $\mathcal{T}_4$ are applied. Exchanging the assignments of adjacent reduction trees leads to six patterns that bring (among others) $(1,1,0)$, $(0,1,1)$, $(1,0,0)$ to the upper full-adders in lower levels, $(0,1,1)$, $(0,1,0)$, $(1,0,1)$ to the lower full-adders in lower levels, $(1,0,0)$, $(0,0,1)$, $(0,1,1)$ to the upper full-adders in the highest level, and $(0,1,0)$, $(1,0,0)$, $(1,0,1)$ to the lower full-adders in the highest level.

In a similiar way the transitions of $\mathcal{T}_7 = \{(t_{25}, t_{26}), (t_{27}, t_{28}), (t_{29}, t_{30}), (t_{31}, t_{32})\}$ are put on those of $\mathcal{T}_5$. The transitions of $\mathcal{T}_7$ don't fulfill Lemma 4. So a further extension is required. This is done by the transitions of $\mathcal{T}_8 = \{(t_{33}, t_{34}), (t_{35}, t_{36}), (t_{37}, t_{38}), (t_{39}, t_{40}), (t_{41}, t_{42})\}$. Now we get generatable 2-periodic patterns. The assignments of adjacent trees don't have to be exchanged since the patterns derived from $\mathcal{T}_5$ bring the same set of single transitions to adjacent reduction trees. Hence, four patterns are sufficient to apply $(0,0,1)$ to the upper full-adders and $(1,0,0)$ to the lower ones in all levels except the two highest ones. Note that the same four patterns also bring the combination $(1,0,0)$ to all lower full-adders in the second highest level by the transitions t_{25} or t_{28}.

So far, all but the following input combinations are brought to the full-adders: $(0,0,1)$ to the upper full-adders in the second highest level, $(1,1,0)$ to the upper full-adders in the highest level, and $(0,1,1)$ to the lower full-adders in the highest level.

The remaining assignments for the highest level are achieved by simultaneously applying t_{43} to all 4to2-cells in this level (one pattern). This leads to a 1-periodic pattern that is generatable according to Lemma 3.

For the lacking combination in the second highest level, we use t_{44}. This transition in all 4to2-cells of this level can be forced by applying t_1 and t_3 resp. to the nodes in the highest level. Again Lemma 3 guarantees the generatability.

We have not yet considered the AND gates. One can easily see that $\mathcal{T}_1$ applies $(0,0)$ when both a and b inputs are set to 0, $\mathcal{T}_2$ applies $(1,1)$, and $\mathcal{T}_3$ applies $(1,0)$. (The first components of the bit pairs stand for the a inputs and the second ones for the b inputs of the AND cells.) The remaining input $(0,1)$ is obtained by one more pattern (set all a inputs to 0 and all b inputs to 1).

So it follows that 17 patterns apply all input combinations to all basic cells which completes the proof.

6 References

[BhHa] D. Bhattacharya, J.P. Hayes: 'Fast and Easily Testable Implementation of Arithmetic Functions', Proc. 1986 Int. Symp. Fault-Tolerant Computing, pp. 324-329

[Be] B. Becker: 'An Easily Testable Optimal-Time VLSI-Multiplier', Acta Informatica 24, 1987, pp. 363-380

[BeKo] B. Becker, R. Kolla: 'On the Construction of Optimal Time Adders', Proc. of the 5th STACS, 1988, Lecture Notes in Computer Science 294, Springer, pp. 18-28

[BeSp] B. Becker, U. Sparmann: 'A Uniform Test Approach for RCC-Adders', Proc. of the 3rd Aegean Workshop of Computing, 1988

[BrKu] R.P. Brent, H.T. Kung: 'A Regular Layout for Parallel Adders', IEEE Trans. on Comp., Vol. C-31, 1982, pp. 260-264

[ChAb] A. Chatterjee, J.A. Abraham: 'Test Generation for Arithmetic Units by Graph Labelling', Proc. 1987 Int. Symp. Fault-Tolerant Computing, pp. 284-289

[ChPa] Wu-Tung Cheng, J.P. Patel: 'Multiple-Fault Detection in Iterative Logic Arrays', Proc. 1985 Int. Test Conf., pp. 493-499

[FeSh] J. Ferguson, J.P. Shen: 'The Design of Easily-Testable Array Multipliers', IEEE Trans. on Comp., Vol. C-33, 1984, pp. 554-560

[Fr] A.D. Friedman: 'Easily Testable Iterative Systems', IEEE Trans. on Comp., Vol. C-22, 1973, pp. 320-323

[Fu] H. Fujiwara: 'A New PLA Design for Universal Testability', IEEE Trans. on Comp., Vol. C-33, 1984, pp. 745-750

[Ha] J. Hartmann: 'Ein C-Test für einen schnellen Multiplizierer', T.R., 2/1988, SFB 124, Saarbrücken 1988

[Ho] Sung Je Hong: 'An Easily Testable Parallel Multiplier', Proc. 1988 Int. Symp. Fault-Tolerant Computing, pp. 214-219

[HBKMO] G. Hotz, B. Becker, R. Kolla, P. Molitor, H.G. Osthof: 'Hierarchical design based on a calculus of nets', Proc. 24th Design Automation Conference, 1987, pp. 649-653

[HNS] E. Hörbst, M.Nett, H. Schwärtzel: 'VENUS: Entwurf von VLSI-Schaltungen', Springer Verlag, 1986

[LaFi] R.E. Ladner, M.J. Fischer: 'Parallel Prefix Computation', J. Ass. Comp. Mach., Vol. 27, 1980, pp. 839-849

[RoRa] M. Robinson, J. Rajski: 'An Algorithmic Branch and Bound Method For PLA Test Pattern Generation', Proc. 1988 Int. Test Conf., pp. 784-795

[SrHa] T. Sridhar, J.P. Hayes: 'Design of Easily Testable Bit-Sliced Systems', IEEE Trans. on Comp., Vol. C-30, 1981, pp. 324-336

[VuLu] J. Vuillemin, W.K. Luk: 'Recursive Implementation of Optimal Time VLSI Integer Multipliers', IFIP Proc. VLSI'83, 1983, pp. 155-168

[Wa] C.S. Wallace: 'A Suggestion for a Fast Multiplier', IEEE Trans. on Electronic Computers, EC-13, 1964, pp. 14-17