1 23 4

On the Generation of Area-Time Optimal Testable Adders

Bernd Becker, *Member, IEEE*, Rolf Drechsler, *Member, IEEE*, and Paul Molitor, *Member, IEEE*

Abstract— **We present a performance driven generator for integer adders which has the following interesting feature: The generator is parametrized in the operands' bitlength n, the delay of the addition t_n and the fault model *FM*. *FM* may in particular be chosen as the classical stuck-at fault model, the cellular fault model or the robust path delay fault model. The output of the generator is a** *performance oriented conditional sum type adder*, **i.e., an area-minimal n-bit adder of the "conditional sum type" with delay $\leq t_n$ (if it exists) together with a small complete test set with respect to the chosen fault model *FM*.**

Keywords— **Circuit design, design for testability, area-time optimal adders, multidimensional dynamic programming.**

I. Introduction

Since 1982, various VLSI designs for fast addition have been proposed [1], [2]. The delays of the adders presented are optimal from the asymptotic point of view.

However, the actual optimal structure of a realistic adder depends on the cell library used (see e.g. [3]). Moreover, designers often do not want to obtain a fastest area-minimal adder but an area-minimal one in the class of adders with a delay less than some bound t. As an example consider floating point multipliers. Due to the delay of the multiplication of the mantissas, the circuit performing the addition of the exponents has not to be time optimal. This leads to *constraint driven adder design*, especially *time driven adder design*. Generators parameterized by the operands's bitlength n and by an upper bound t_n for the delay computing an area-minimal n-bit adder with delay $\leq t_n$ (if it exists) are demanded.

Recently, Wei and Thompson [4] and Chan et al. [5] formulated the area-time optimal adder design as a dynamic programming problem. Their approach is based on Ladner and Fischer's *parallel prefix computation* [6], and is essentially a look-ahead addition. Unfortunately, Wei and Thompson do not take into account the testability (with respect to some fault model) of the area-time optimal adders constructed by their generator. However, as a result of technological improvements testability aspects gain more and more importance and should therefore be considered as early as possible.

Modern VLSI electronic circuitry may contain hundreds of thousands of transistors on a single silicon chip. Even if the chips are correctly designed, i.e., design verification has been done successfully, a non negligible fraction of them will have physical defects caused by imperfections occurring during the manufacturing process (e.g., open connections induced by dust particles). Therefore, there has to be a test phase in which production verification is performed, i.e., in which the *good* chips are sorted from the *bad* ones. For a detailed treatment of the topic see [7].

Due to the variety of possible defects restrictions on a subset of the possible faults are necessary; these simplifying assumptions based on experience of many years are manifested in fault models. Since tests are generated to test for the fault mechanisms described by the assumptions, the reliability of the chip is at least partially determined by the accuracy and effectiveness of the fault model (measured, e.g., in detected physical failures).

The classical stuck-at fault model (SAFM) is well-known and used throughout the industry [8]. It assumes that a defect causes a basic cell input or output to be fixed to either logic 0 or logic 1. Thus, all failures with this effect will be detected by tests for stuck-at faults, but there exist other static faults that are not covered by using this simple model [9], [10], [11], [12]. Thus, other fault models are necessary to overcome this problem, e.g., flexible fault models which verify the correct static behavior of a combinational circuit based on inductive fault analysis [13], [14]. The strongest cell-based fault model to control the correct static behavior of a combinational circuit is the cellular fault model (CFM), which tries to verify the function of each basic cell in the circuit completely [15], [16]. However, it has been observed by many authors that a large amount of defects typical of today's VLSI technologies are not covered by static fault models like the stuck-at or cellular fault model. In many cases dynamic fault models which allow to model stuck-open faults or timing issues [17], [18], [19] are necessary. In particular, it is natural (and necessary) to check delays in a circuit that has been constructed by a time driven design methodology, since physical failures can also influence the timing behavior of a circuit. The generator can only use the information from the library, but not the actual delay behavior. We therefore also consider the robust path delay fault model [18] in this paper. The robust path delay fault model is a very powerful fault model whose objective is to check the propagation delay of every path in the circuit. The more powerful a fault model is the more faults are detected in the circuit, e.g., a complete robust path delay fault test detects many multiple stuck-at faults [20].

With increasing complexity of VLSI circuits, the costs

Manuscript received June 22, 1993; revised April 12, 1994. This work was supported in part by DFG grant Be 1176/4-1 and SFB124. This paper was recommended by Associate Editor K. Keutzer.

B. Becker and R. Drechsler are with the Department of Computer Science, J.W.Goethe-University Frankfurt, D-60054 Frankfurt/Main, Germany. E-mail: <name>@kea.informatik.uni-frankfurt.de.

P. Molitor is with the Institute of Computer Science, Martin-Luther University Halle, D-06099 Halle/Saale, Germany. E-mail: molitor@infsparc2.informatik.uni-halle.de.

IEEE Log Number

for the test phase have increased dramatically; at least 25% and up to 60-70% of the total product costs are due to testing [21], [22]. Due to the increasing costs of the test phase, specialists in the field of testing agree that testability issues have to be considered from the very beginning of the design process to control the test costs and to guarantee the testability of the circuit at the end of the manufacturing process. This requires a detailed analysis of the relation between structural properties of the circuit under test and testability properties. Successful steps in this direction can be observed: Relevant subclasses of circuits, which, due to their structural properties, allow efficient test generation algorithms and as a result yield complete test sets of small size, have to be classified and investigated. Typical examples of classes which have been successfully treated by this method are PLAs, memories, arithmetical units and tree-like structures (see e.g. [23], [24], [15], [25], [26], [27], [28]).

In this paper we present a realistic and systematic method for constructing area-time optimal parallel adders together with efficient complete tests with respect to a chosen fault model. Our approach to generate adders is based on the conditional sum scheme [29]. The delay model used is the *intrinsic-plus-fanout delay model* given in lots of industrial cell libraries (see e.g. [30]) and also used by the logic synthesis and optimization benchmarks of the Microelectronics Center of North Carolina [31]. For the computation of the area, we restrict ourselves to the *sum of the cell areas* because the adder design presented in the following is done on a symbolic design level. The lengths of the wires cannot be estimated in general, since they largely depend on the technology used. For some design processes this simplifying assumption is realistic, i.e., *channelless gate arrays* or *sea-of-gates* [32], [33]. As in [5], [4], the area-time optimal adder design itself is formulated as a dynamic programming problem. We present two methods for this first phase. The first one was presented in [34] and is based on a straightforward dynamic program. However, this algorithm is impracticable for large n-bit adders ($n > 64$), because the memory requirement is too large[1]. The second algorithm solves the problem of area-time optimal adder design by dynamic programming where each step is based on *point location* [35]. This algorithm decreases the actual runtime and the memory requirements of the first one. The technique presented can also be applied to other circuit structures, e.g., to the look-ahead addition principle. The algorithms have been implemented, examples are given in Section IV. The experimental results show that the second algorithm is applicable also for large n although its asymptotic worst case behavior is worse. An efficient complete test with respect to a chosen fault model is generated after the *performance oriented conditional sum type adder* (POCSTA) has been constructed. The testability results concentrate on the generation of complete test sets for the stuck-at fault model, the cellular fault model and the robust path delay fault model. When we use a static fault model we assume that there is at most one fault in the circuit (*single fault assumption*). This is a usual assumption made in literature and by industry. For static fault models the size of the test set is linear in the depth of the circuit (stuck-at fault model) or linear in the size of the circuit (cellular fault model). For the robust path delay fault model the size of the test set is bounded by $O(n^2 \cdot depth)$, where *depth* denotes the depth of the circuit. The quality of the test sets is analysed by giving lower bounds for their sizes which only slightly differ from the upper bounds provided by the construction. In many cases the tests are even optimal from the asymptotic point of view. For the stuck-at fault model test we generalize a test strategy given in [36] for the classical conditional sum adder (CSA). The construction method for the cellular fault model can easily be modified to any static cell based fault model between the cellular fault model and the stuck-at fault model. To obtain a fully testable adder using the robust path delay fault model the original adder has to be modified.

The worst case runtime of the first phase, i.e., of the generation of the n-bit POCSTA with delay $\leq t_n$, is $O(n^2 \cdot t_n^2 \cdot \log t_n)$, where t_n is measured, e.g., in units of $0.1ns$. The runtime of the second phase, i.e., of the generation of a complete efficient test, is bounded by $O(n^2)$ for static fault models and is bounded by the size of the test set for the robust path delay fault model.

The paper is structured as follows: Adders of the conditional sum type (CST) are described in Section II. Section III provides an introduction to basic notations and concepts of testing. The fault models used are introduced. The testability of the CST adders is examined and we obtain lower and upper bounds for the sizes of the test sets. The observations on testability are valid for all CST adders independent of the structure of the specific adder. So we guarantee that all generated adders are testable; thus the designer does not have to worry about testability. The formulation of the dynamic programming problem performing area-time optimal adder design with respect to a given library is presented in Section IV where the computation models used are also described. We finish with a resume of the results in Section V.

II. Conditional sum type adders

In this section the class of the CST-adders considered in this paper is described.

CST-adders are a generalization of the adder presented in [29]. They are defined in the following way: Let $a = \sum_{i=0}^{n-1} a_i 2^i$, $b = \sum_{i=0}^{n-1} b_i 2^i$ be two n-bit numbers, and for any $k \in \{1, \ldots, n-1\}$ let $a^{(k,1)} := \sum_{i=k}^{n-1} a_i 2^{i-k}$ be the number represented by the most significant $n-k$ bits of a, and $a^{(k,0)} := \sum_{i=0}^{k-1} a_i 2^i$ be the number represented by the least significant k bits. $b^{(k,1)}$, $b^{(k,0)}$ are defined analogously. Now, suppose that we already have designed a k-bit adder and an $(n-k)$-bit adder computing simultaneously both, the sum and the sum plus 1 of their inputs. Then $a+b$ and $a+b+1$ can be computed by a k-bit adder, an $(n-k)$-

[1] Analogous problems arise during the implementation of the generator of Chan, Schlag, Thomborson, and Oklobdzija [5]. Their generator requires e.g. 60 Mbyte memory to generate the trade-off of the 84-bit adder.

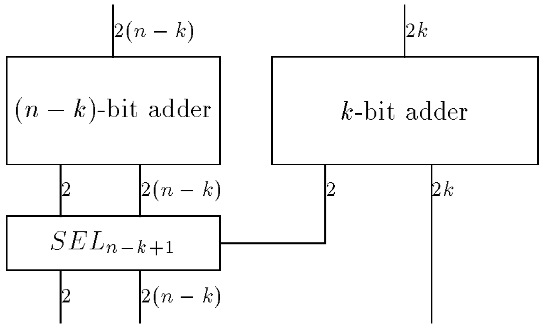

Fig. 1. Recursive structure of a CST-adder

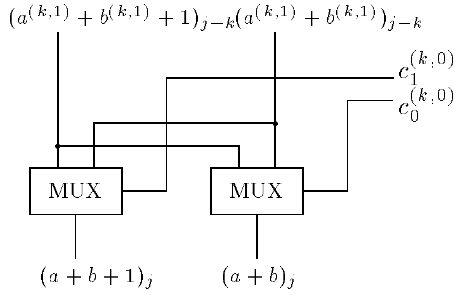

Fig. 2. Graphical description of a *SEL*-cell

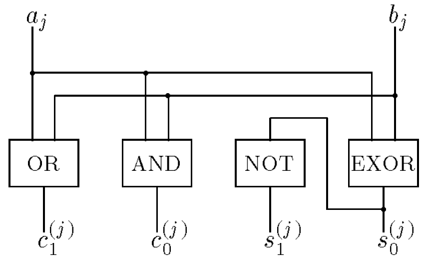

Fig. 3. Description of the 1-bit adder A_1

bit adder and a multiplexer stage as follows (see Fig. 1): Compute in parallel $a^{(k,1)}+b^{(k,1)}$, $a^{(k,1)}+b^{(k,1)}+1$, and $a^{(k,0)}+b^{(k,0)}$, $a^{(k,0)}+b^{(k,0)}+1$ by an $(n-k)$-bit adder and a k-bit adder, respectively. Obviously, the least significant k bits of $a+b$ and $a+b+1$ are identical to the least significant k bits of $a^{(k,0)}+b^{(k,0)}$ and $a^{(k,0)}+b^{(k,0)}+1$, respectively. In order to complete the addition, select the most significant $n-k+1$ bits of $a+b$ and $a+b+1$ by

$$(a+b)_j = \begin{cases} (a^{(k,1)}+b^{(k,1)})_{j-k} & \text{if } c_0^{(k,0)}=0 \\ (a^{(k,1)}+b^{(k,1)}+1)_{j-k} & \text{if } c_0^{(k,0)}=1 \end{cases}$$

and

$$(a+b+1)_j = \begin{cases} (a^{(k,1)}+b^{(k,1)})_{j-k}, & \text{if } c_1^{(k,0)}=0 \\ (a^{(k,1)}+b^{(k,1)}+1)_{j-k}, & \text{if } c_1^{(k,0)}=1 \end{cases}$$

($\forall j \in \{k,\ldots,n\}$) where $c_0^{(k,0)}$, $c_1^{(k,0)}$ denote the carry of $a^{(k,0)}+b^{(k,0)}$ and $a^{(k,0)}+b^{(k,0)}+1$, respectively, and $(\sum x_i 2^i)_j$ is given by x_j.

The selection stage is denoted by SEL_{n-k+1} and is called a *SEL-chain* or *SEL-stage* of size $n-k+1$. It consists of $n-k+1$ *SEL-cells*. The leftmost *SEL*-cell of a *SEL*-chain is denoted by *carry-SEL*. The remaining ones are denoted by *block-SELs*. The right input signals of a *SEL*-chain are connected to the right input pins of each *SEL*-cell in this chain. The *SEL*-cell itself is described as shown in Fig. 2. Here, a MUX-cell is a multiplexer, where the select input is on the right. If logic 1 (0) is applied to the select input, the cell outputs the value applied to the leftmost (rightmost) upper pin.

The specification of the recursively defined adder is completed by description of the 1-bit adder A_1 (see Fig. 3). To simplify matters in the sequel, we will call a *A_1-cell to be at bit position j*, if a_j and b_j are applied to it. A *SEL-cell* is said *to be at bit position j* if its upper input pins are connected either to two output pins of the A_1-cell being at bit position j or to the two output pins of a *SEL*-cell being at bit position j.

For notational convenience we use symbolic values to describe the functional behavior of the adders, namely, $A=(0,0)$, $G=(1,1)$, $P=(1,0)$, and $R=(0,1)$. The abbreviations A (*absorb*), G (*generate*), P (*propagate*), and R (*redundant*) originate from the carry computation process (for details see [15]). Consider a *SEL*-cell (see Fig. 2). Then the upper (right) input lines receive values from the set $M := \{A, P, G, R\}$ and *SEL* computes the binary operation $\circ$ given in the left diagram of Table I. Rows (columns) correspond to fixed values of the upper (right) input lines. $\circ$ defines a monoid operation on M with *neutral* element P and *left-stable* elements A and G, i.e., $P\circ X = X\circ P = X$ and $A\circ X = A$, $G\circ X = G$ ($\forall X \in M$). The monoid is known as the *conditional carry monoid* from [15]. The right diagram of Table I shows the symbolic function computed by the A_1-cell (see Fig. 3).

TABLE I
FUNCTIONS COMPUTED BY *SEL* AND A_1

$\circ$	A	P	G	R
A	A	A	A	A
P	A	P	G	R
G	G	G	G	G
R	G	R	A	P

	A	P	G	R
(c_1, c_0)	A	P	G	P
(s_1, s_0)	P	R	P	R

For the following it will be helpful to look at CST-adder-circuits as tree-like structures computing products over M and thus *lift* our testability discussions on a symbolic level. Obviously, the choices of k determine the tree-like structure of the adder and, as a detailed analysis in the following will show, influence its performance.

The tree-like structure is best reflected in the *structure tree* of a CST-adder. We will see that the structure tree uniquely determines a CST-adder A_n and vice versa. The structure tree of A_n is a labeled binary tree with n leaf nodes. Each inner node corresponds to exactly one recursive composition step by a chain of *SEL*-cells. The labels at the

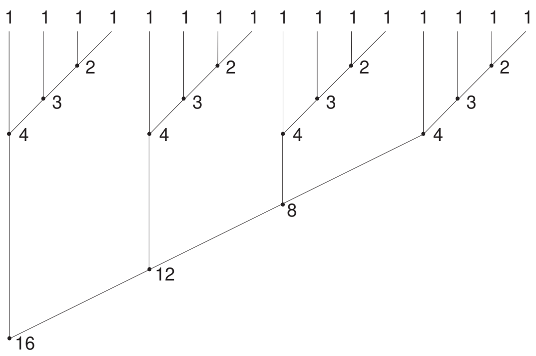

Fig. 4. Structure tree of the 16-bit CST-adder $A_{16}(9.3ns)$.

nodes define the bit size of the adder resulting from the composition step: Each leaf node has label 1, a non leaf node v has label $k_1 + k_2$, if the left child (right child) of v has label k_1 (k_2). The length of the SEL-chain at node v is then given by $k_1 + 1$. The leftmost SEL-cell is the *carry-SEL* of the chain at v. Notice that the structure tree of A_n is contained as a subcircuit in A_n: Consider the tree built by the carry-SELs. For an illustration see Fig. 4.

A SEL-chain is a *boundary chain*, if the outputs of the corresponding block-SELs are directly connected to primary output ports. Block-SELs (carry-SELs) in a boundary chain are called *boundary block-SELs* (*boundary carry-SELs*). It easily follows that the number of boundary block-SELs is exactly $n-1$ for any CST-adder A_n independent of the structure of A_n. We will use boundary chains for the construction of complete test sets and the estimation of the quality of the sets as well.

Two other notions are important: The *left-depth* $d_{left}(A_n)$ of a CST-adder A_n is the maximum number of left (= *vertical*) edges on a path from a leaf to the root of the structure tree. d_{left} varies from 1 to $n-1$ in the class of n-bit CST-adders. The left-depth of the classical n-bit conditional sum adder ($n = 2^k$) is $\log n = k$ (= depth of the corresponding structure tree), since in this case the structure tree is a complete balanced binary tree of depth k. In the example of Fig. 4 we have $d_{left} = 2$. The *height* i of the structure tree contains all nodes whose distance to the root of the structure tree is i. A SEL-cell in A_n has *height* i, if its SEL-chain corresponds to a node of height i in the structure tree. Thus, the height of a cell measures the distance from the primary outputs, whereas the usual depth d of a cell measures the distance from the primary inputs. The height of A_n $h_{tree}(A_n)$ is the maximum height of a node in A_n. Clearly, $h_{tree}(A_n) = d(A_n)$, which denotes the usual depth of the circuit A_n.

III. Testability of CST-adders

In this section we study the testability of CST-adders with respect to the stuck-at fault model (SAFM), the cellular fault model (CFM) and the robust path delay fault model (RPDFM). In Section III-A we provide some basic notions and concepts of testing which are important for the derivation of the testability results in Sections III-B–III-D. In III-B–III-C we give lower and upper bounds for the complexity of complete test sets of an n-bit adder A_n with respect to the SAFM and CFM. In case of the RPDFM we first describe a modification of the CST-adder to obtain robust path delay fault testability for all paths and then derive bounds for a complete test also in this fault model (Section III-D). All upper bounds are based on the effective construction of complete test sets. For an overview Table II summarizes in asymptotic form the results that will be proved in this section:

TABLE II
Results on lower and upper bounds

fault model	complete test set	
	lower bound	upper bound
SAFM	$\Omega(d_{left}(A_n))$	$O(d(A_n))$
CFM	$\Omega(n)$	$O(n^2)$
RPDFM	$\Omega(n^2)$	$O(n^2 d_{left}(A_n))$

The proof methods are independent of the choices of the parameter k, which determine the structure of the adders. All test sets can be generated from the structure tree of the adder. Thus, the test sets can be constructed efficiently after the generation of the area-time optimal adders (see Section IV).

A. Notations and definitions

As mentioned before we consider combinational logic circuits realizing the addition of two binary numbers. In general, a *combinational logic circuit* (CLC) is defined over a fixed library and modeled as a directed acyclic graph $C = (V, E)$ with some additional properties: Each vertex $v \in V$ is labeled with the name of a basic cell or with the name of a primary input (PI) or primary output (PO). The collection of basic cells available is given by the fixed library. There is an edge (u, v) in E from vertex u to v, iff an output pin of the cell associated to u is connected to an input pin of the cell associated to v, i.e., edges contain additional information to specify the pins of the source and sink node they are connected to. Vertices have exactly one incoming edge per input pin. Nodes labeled as PI (PO) have no incoming (outcoming) edges.

In this paper we use two libraries, the library $LIB_{CST} := \{SEL, A_1\}$ and a standard library STD consisting of the 2-input, 1-output *AND, OR* gate and the 1-input, 1-output inverter NOT^2. The CST-adder-circuits have been introduced in Section II as circuits over LIB_{CST}. See Fig. 2 and 3 for a definition of the cells SEL and A_1 as subcircuits over STD. The $EXOR$-gate in Fig. 3 can easily be replaced by its basic cell realization. A CST-adder-circuit C_1 over LIB_{CST} can easily be interpreted as a circuit C_2 over STD just by replacing each basic cell by its standard cell realization. C_2 is then called an *expansion* of C_1.

[2] The results obtained in the following can be directly transfered to the case where other libraries are given, e.g., NAND-, NOR-, NOT-based libraries that are most often used in CMOS design processes.

We are now ready to give a short description of the fault models (FM) considered in this article.

A fault in the *stuck-at fault model (SAFM)* [37] causes exactly one input or output pin of a node in the circuit C to have a fixed constant value (0 or 1) independent of the values applied to the PIs of the circuit. A *stuck-at fault* with fault location v therefore is completely defined by a tuple $(v[i], \epsilon)$ or $([i]v, \epsilon)$. $v[i]$ $([i]v)$ denotes the i-th input (output) pin of v, $\epsilon \in \{0, 1\}$ is the fixed constant value. Usually, the stuck-at model is only considered on the gate level, i.e., for circuits over the library STD. Here, we have given an obvious generalization to circuits over arbitrary libraries.

In the *cellular fault model (CFM)* [15], [16] it is assumed that a fault modifies the behavior of exactly one node v in a given CLC C and that the modified behavior is still combinational. Since this fault can be detected by observing the incorrect output values of v for one suitable input combination, it suffices to test for faults of the following kind. A *cellular fault* in C is a tuple $(v, I, X/Y)$, where v is the faulty node (= fault location), I is the input for which v does not behave correctly, and X (Y) is the output of the correct (faulty) node on input I.

In general, a *cell based static fault model (CBFM)* is obtained from the CFM by a restriction of the fault set that has to be considered. Any function $g_{v,faulty}$ which is unequal to the correct behavior g_v at node v defines a possible fault in the CFM. In many cases it is too expensive to check this large variety of possible faults, and methods like inductive fault analysis may indicate that satisfying results are obtainable, if only a certain subset F_v of the possible faults at node v is checked. The stuck-at fault model defined above is a well-known example of a CBFM.

We finish the discussion of CFM, CBFM and SAFM with some general definitions and remarks on the relation between the fault models. For this let C be any CLC over a fixed library and FM a fault model as defined above. A fault f in FM is *testable*, iff there exists a test t for this fault, i.e. iff the output values of C on applying t in the presence of f are different from the output values of C in the fault free case. The goal of any test pattern generation process is a *complete* test set for the circuit under test in the considered fault model FM, i.e., a test set that contains a test for each testable fault. It easily follows from the definitions that, given a fixed circuit C, a complete test set in CFM is also complete in any CBFM and thus also in SAFM. The cellular fault model therefore is stronger than all other static fault models.

A widely accepted criterion for the quality of a complete test set is the size of the set, since it determines the test application time. We therefore define the *test complexity* of a circuit C with respect to a static FM by $TC_{FM}(C) := \min\{|\mathcal{T}|;\ \mathcal{T}$ is complete for C in FM$\}$, where $|\mathcal{T}|$ denotes the size of the test set, i.e., the number of tests.

Now, consider a circuit C_2 which results from a circuit C_1 by "expansion". Then, one can easily show that a complete test set for C_1 in CFM is also a complete test set for C_2 in CFM. Thus, the CFM is more powerful, if the size of the basic cells increases. We call this property the *completeness property* of CFM. Notice, that in SAFM there is a trend in the opposite direction. This is the reason why in general the strongest version of SAFM, i.e., SAFM for circuits over STD, is considered.

The other fault model we consider in this paper is the *robust path delay fault model (RPDFM).* The RPDFM [18] is a dynamic FM which checks whether the propagation delays of all paths in a given combinational logic circuit are less than the system clock interval. For our discussion of the path delay fault model the CST-adders are considered as combinational circuits over STD. A *path* π is given by an alternating sequence of nodes and edges $(v_0, e_0, v_1, \ldots, v_n, e_{n+1}, v_{n+1})$ starting at a PI v_0 and ending at a PO v_{n+1}. Inputs of nodes on the path where no edge e_i of the path ends are called *side inputs.* A *rising* $(0 \rightarrow 1)$ or *falling* $(1 \rightarrow 0)$ *transition propagates along* π, if a sequence of transitions $t_0, t_1, \ldots, t_{n+1}$ occur at the nodes $v_0, v_1, \ldots, v_{n+1}$, such that t_i occurs as a result of t_{i-1}. π has a *path delay fault* for the rising (falling) transition, if the actual propagation delay of the transition along π exceeds the system clock interval. For the detection of a path delay fault a pair of patterns (I_1, I_2) is required rather than a single pattern as in the *cellular fault model* and the *stuck-at fault model*: The *initialization vector* I_1 is applied and all signals of the circuit C are allowed to stabilize; then the *propagation vector* I_2 is applied and after the system clock interval the outputs of C are controlled. A two-pattern test is called a *robust test* for a path delay fault on π, if it detects that fault *independently* of *all other delays* in the circuit and *all other delay faults* not located on π.

In Section III-D of this paper we concentrate on robust testing of path delay faults. A *controlling* value at the input of a node is the value that completely determines the value at the output, e.g., 1 (0) is the controlling value for *OR* (*AND*) and 0 (1) is the non-controlling value for *OR* (*AND*). It turns out that the construction of tests with the following property is possible: For each path delay fault there exists a robust test (I_1, I_2) which sets all side inputs to the non-controlling values on application of I_1 and remains stable during application of I_2, i.e., the values on the side inputs are not invalidated by hazards or races. These tests are called *strong robust PDF tests.* In the following we only use such tests for the RPDFM, but for simplicity we call them robust PDF tests, too. For a detailed classification of path delay faults see [38].

B. Testability of CST-adders in the SAFM

Next, we discuss the testability of the CST-adders in the fault models introduced before. In this section the weakest fault model, the SAFM, is considered. It is shown that a lower bound for the test complexity is given by twice the left-depth of the CST-adder. An upper bound of $O(depth)$ is obtained by the effective construction of a complete test set. For this, two construction phases "generation at the lower leaves" and "generation to the higher leaves" are introduced which allow to efficiently apply methods presented in [36] for the test of CST-adders with a complete balanced binary structure tree.

We start with the lower bound.

Theorem 1 (lower bound) Let A_n be an n-bit CST-adder defined over the library LIB_{CST}. Then we have
$$TC_{SAFM}(A_n) \geq 2 \cdot d_{left}(A_n).$$

Proof: For the lower bound we specify a set $\mathcal{F}$ of $2 \cdot d_{left}(A_n)$ stuck-at faults, such that each fault in $\mathcal{F}$ requires an extra test. At first, consider a path π' with maximal left-depth in the structure tree. Look at the corresponding path π of carry-*SEL*s in A_n. Then there are exactly $d_{left}(A_n)$ carry-*SEL*s on π whose upper input lines correspond to a left edge on π'. Among these cells exactly the carry-*SEL* which is nearest to the destination of π is a boundary carry-*SEL*. The remaining $d_{left}(A_n)-1$ cells are non boundary carry-*SEL*s. Then the following two classes of stuck-at faults at the right input lines of these carry-*SEL*s define $\mathcal{F}$: The stuck-at 1 fault at the upper right input line and the stuck-at 0 fault at the lower right input line, if the cell is a non boundary carry-*SEL*; the stuck-at 1 fault and the stuck-at 0 fault at the lower right input line, if the cell is a boundary carry-*SEL*.

The following observations now lead to the lower bound of $2 \cdot d_{left}(A_n)$:

As mentioned earlier, only the values A, P, G can be generated at the inputs of carry-*SEL*s. This can be easily proved by induction. Thus, the application of P at the upper input is necessary for the detection of any stuck-at fault at the right input lines, because A and G are left-stable elements.

It can easily be seen that a stuck-at 1 fault at the upper right input line and a stuck-at 0 fault at the lower right input line of a non boundary carry-*SEL* are testable and require the application of the values A and G at the right input lines, respectively. Therefore tests which apply PA and PG to the non boundary carry-*SEL*s of π must be contained in any complete test set.

Let x be the boundary carry-*SEL* on π. It follows by inspection of Table I that the stuck-at 1 fault at the upper right input line is redundant, the stuck-at 0 fault at the lower right input line fault remains testable and therefore requires the application of PG to this cell. A stuck-at 1 fault at the lower right input line is testable and requires the application of PA or PP to the cell inputs.

If P is to be generated at the upper input of a carry-*SEL* x on π, then the whole subtree of carry-*SEL*s rooted at x, especially the carry-*SEL*s of π below x, must receive PP as inputs and cannot receive PA or PG which would be necessary to test a fault at carry-*SEL*s below x. ■

We now come to the construction of complete test sets, which proves that the lower bound is also an upper bound from the asymptotic point of view, for adders with non differing depth and left-depth. A complete test in the SAFM for the classical conditional sum adder, where the carry tree is a complete binary tree, has already been constructed in [36]. We show how this construction can be generalized to CST-adders. At first some notions for the description of test patterns have to be introduced: It turns out to be helpful to consider the structure tree, if patterns are to be applied to the cells of the circuit. Let v be a node whose left (right) child v_1 (v_2) is labeled with k_1 (k_2). Then the left incoming, the right incoming, and the outgoing edge of v correspond to k_1+1, k_2+1 and $k := k_1+k_2+1$ uniquely defined pairs of signal lines in A_n, respectively. A valid assignment of the left incoming edge (right incoming edge) of v is a string $S_1 := X^1Y_1^1Y_2^1\cdots Y_{k_1}^1 \in M^{k_1+1}$ ($S_2 := X^2Y_1^2Y_2^2\cdots Y_{k_2}^2 \in M^{k_2+1}$). X^i denotes the output value of the carry-*SEL* at node v_i, the Y_j^i denotes the value of the j-th pair of signal lines (numbered from left to right). If (S_1, S_2) are applied to node v, the resulting output assignment S at node v is given by

$$(X^1 \circ X^2)(Y_1^1 \circ X^2)(Y_2^1 \circ X^2)(Y_{k_1}^1 \circ X^2)Y_1^2 \cdot \ldots \cdot Y_{k_2}^2.$$

Since S results from S_1, S_2 by repeated application of $\circ$, we write $S = S_1 \circ S_2$. Now consider the patterns $P_1^k := G^k \cdot P$, $P_2^k := A^k \cdot P$, $P_3^k := P \cdot R^k$, $P_4^k := A \cdot P \cdot R^{k-1}$, and $P_5^k := G \cdot P \cdot R^{k-1}$ which are valid assignments for a node with label k. For simplicity of notation we often omit the superscript k and write P_i instead of P_i^k. Using this and the composition rules $P_1 = P_1 \circ P_1$, $P_2 = P_2 \circ P_2$, $P_3 = P_3 \circ P_3$, $P_4 = P_4 \circ P_3$, and $P_5 = P_5 \circ P_3$ we can easily conclude:

Lemma 1: Let v be any node in the structure tree of a CST-adder A_n. Then P_i for $i = 1, \ldots, 5$ can be generated at the output edge of v by setting the inputs of A_n appropriately.

The lemma above is a generalization of a corresponding lemma in [36] and demonstrates that the composition laws for P_i do not depend on the length of the *SEL*-chains. A complete test set in the SAFM for CST-adders with complete balanced binary structure tree (CBBST) is now given by the following lemma, which is a summary of the testability properties derived in [36].

Lemma 2 (upper bound for CBBST) A CST-adder A_n with CBBST is completely testable over the library LIB_{CST} in the SAFM with $6 \log n + 2$ ($= 6d_{left}(A_n) + 2$) test patterns, as defined in Fig. 5. Furthermore, the test set fulfills the following propagation properties:

– Let F be a testable fault at the input lines of a cell x of height j. Then F is tested by a pattern constructed in step (1) or a pattern constructed in step (2j), …, (6j) or (7j).

– If x is a block-SEL at bit position k the faulty difference is propagated by the uniquely determined sequence of block-SELs to the output port for bit position k. Along this path the faulty difference is only combined from the right with output values of carry-SELs.

– If x is a carry-SEL at bit position n, the faulty difference is propagated by the uniquely determined sequence of carry-SELs to the output port for the carry bit of the adder A_n. Along this path the faulty difference is only combined from the right with output values of carry-SELs.

– If x is a carry-SEL at bit position $k < n$, the faulty difference is propagated to an output port for a bit position $k' > k$. k' uniquely defines a sequence of carry cells followed

(1) For $i = 1, 2$ apply (P_i, P_i) to all nodes.
For any height $j = 0, \ldots, \log n - 1$
(2j) Apply (P_3, P_4) to all nodes of height j
(3j) Apply (P_3, P_5) to all nodes of height j
(4j) Apply (P_1, P_2) to all nodes of height j
(5j) Apply (P_2, P_1) to all nodes of height j
(6j) Apply (P_3, P_4) and (P_3, P_5) alternately to the nodes of height j starting with (P_3, P_4) at the rightmost node
(7j) Apply (P_3, P_5) and (P_3, P_4) alternately to the nodes of height j starting with (P_3, P_5) at the rightmost node

Fig. 5. Complete test set for CST-adders with CBBST

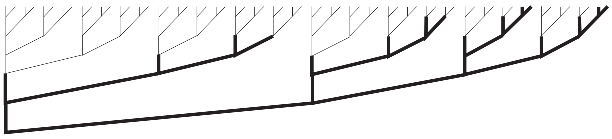

Fig. 6. Pseudo structure tree corresponding to the 16-bit CST-adder shown above. The bold part corresponds to the original 16-bit CST-adder $A_{16}(9.3ns)$)

by a sequence of block cells which propagates the difference. The first block cell x' in this sequence is the only cell of the whole sequence where the faulty difference is not combined with the output value of a carry-SEL but with the value P at the upper input lines of x'. Now consider the SEL-stage of x'. Assume x'' is another block cell in this SEL-stage that receives P at the upper input lines. Let k'' be the bit position of x''. Then the faulty difference is also propagated to an output port for a bit position $k'' > k$.

For the proof of the lemma we refer to the original paper [36]. We rather show that a (non trivial) generalization to CST-circuits is possible. For this, consider the structure tree of a CST-adder-circuit A_n, that is not necessarily complete and balanced. Now, consider the smallest complete balanced binary tree which can be pruned to the structure tree of A_n, i.e., the smallest complete binary tree obtained by expanding the structure tree of A_n. We call it the *pseudo structure tree* of A_n. For illustration see Fig. 6. The non-bold part does not belong to the original structure tree. The pseudo structure tree corresponds to a larger CBBST-adder, in which A_n is embedded.

Now, a complete test set for A_n is constructed in the following way: Compute a *pseudo* complete test set for this pseudo structure tree by using Lemma 1 and 2. For each test pattern this leads to a labeling with the P_i's of the pseudo structure tree, values are in particular assigned to the outputs of all carry-*SEL*s. Use this information to modify any pseudo test pattern $\mathcal{P}_{pseudo}$ to a test pattern $\mathcal{P}$ of A_n as follows: Consider the leaf nodes of the structure tree of A_n one by one. If the leaf node considered is also a leaf node in the pseudo structure tree, maintain the labeling of the node. If the leaf node corresponds to an inner node of the pseudo tree, replace the label by one of P_1^1, P_2^1, P_3^1, such that the first element (= output value of the carry-*SEL*) remains unchanged. This uniquely defines the new labels, if one takes into account that the equations $P_1^1 = P_5^1$ and $P_2^1 = P_4^1$ hold.

Of course, the modification defines an input pattern for A_n. Now, look at a cell x_{pseudo} in the pseudo adder, that stands for a "real" cell x in A_n. Assume that x_{pseudo} and x have height j and that the pseudo test pattern $\mathcal{P}_{pseudo}$ is the test constructed for a stuck-at fault at cell x_{pseudo}. Then the resulting test pattern $\mathcal{P}$ for A_n is a test for this stuck-at fault at x because of the following reasons: Due to the special patterns considered and the *generation laws* given in the proof of Lemma 1, $\mathcal{P}$ and $\mathcal{P}_{pseudo}$ exactly generate the same values at the inputs of x and x_{pseudo}, respectively. Furthermore, our modification of $\mathcal{P}_{pseudo}$ to $\mathcal{P}$ guarantees that the values on outputs of carry-*SEL*s remain unchanged. Look at Fig. 5 and the definition of the P_i's to see that $\mathcal{P}_{pseudo}$ and $\mathcal{P}$ apply P to the upper inputs of at least one block-*SEL* in any *SEL*-chain with height $< j$. Now apply the propagation properties of Lemma 2 to obtain that $\mathcal{P}$ is a test for the considered fault.

We end with a complete test set for A_n in SAFM. The construction is conceptually simple, but it may require exponential time, since the pseudo tree is possibly exponentially larger than the structure tree of the circuit considered. In the proof of the next theorem we show, that this disadvantage can be avoided. On the whole, we get the following theorem:

Theorem 2 (SAFM-test) Let A_n be an n-bit CST-adder. Then a complete test set $\mathcal{T}_{SAFM}$ for A_n in SAFM with $|\ \mathcal{T}_{SAFM}\ | = 6 \cdot d(A_n) + 2$ can be constructed in time $O(n \cdot d(A_n))$.

Proof: Note that the depth of the pseudo structure tree is equal to the depth of the structure tree of A_n. Therefore, only the time bound remains to be proved. For this look at the test patterns of Fig. 5. The construction has to be done efficiently for all seven types of patterns defined there. For type (1) there is nothing to prove.

Therefore consider a fixed height j and the pattern of type (2j). (P_3, P_4) is easily generated at the inputs of any non leaf node of height j by choosing the leaf inputs of the leaves in height k with $k > j$ appropriately according to the composition laws given in Lemma 1. We call this phase of the construction "generation to the higher leaves" There may be leaf nodes in a height $k \leq j$. Remember, that it was important for the construction of the test patterns from the pseudo test patterns, that the value at the output of the carry-*SEL*s remained unchanged. Application of (P_3, P_4) in height j generates carry values A in the lower heights. We therefore place P_2^1 at the leaves in heights k with $k \leq j$. This phase of the construction is called "generation at the lower leaves" This completes the construction for patterns of type (2j).

Type (3j), (4j), (5j) are handled analogously. P_1, P_1, P_2, respectively, is used for the generation at the lower leaves.

The remaining types (6j) and (7j) are slightly more complicated. We first define the *parity* of a node v in the structure tree as follows: If the outgoing edge of v is a right input edge of a node in the next (lower) height, the parity of v is *even*. Otherwise the parity is *odd*.

Now consider type (6j). In a first phase, the "generation to the higher leaves" we generate (P_3, P_4) $((P_3, P_5))$ at the even (odd) non leaf nodes of height j. The "generation at the lower leaves" works as follows:

Even (odd) leaf nodes of height j are labeled with P_2 (P_1). (Even (odd) nodes of height j have to produce A (G) at the outputs of their carry-*SEL*s!) Leaf nodes in heights k with $k < j$ are labeled with P_1. (At the nodes in heights $k < j$ the outputs of carry-*SEL*s have to be G.) The construction for type (7j) is done analogously.

It follows from the construction that we obtain the same complete test set as with the help of the pseudo structure tree. Furthermore it can easily be seen that the time complexity can be bounded from above by $O(n \cdot d(A_n))$. ■

Thus, we have given the construction of a complete test set whose size is linear in the depth of the circuit. From Theorem 1 we know that the left-depth is a lower bound, i.e., for circuits with non differing depth and left-depth we have constructed an optimal test from the asymptotic point of view. The algorithm described above has been implemented to generate test sets for CST-adders.

Our test for the SAFM is based on the library LIB_{CST}. A complete test based on a library with primitives of lower complexity often is desirable in the SAFM to obtain a set of patterns covering many potential physical defects. From [36] it is known that a $(\log n)$-test is possible for the classical conditional sum adder viewed as a circuit over STD. Since the same methods are used as before, a generalization along the lines given above is possible for the CST-adders. We omit the detailed construction and switch over to the construction of a complete test set for a more powerful static fault model.

C. Testability of CST-adders in the CFM

CFM is a more powerful fault model than the SAFM. We show that for CST-adders this leads to a lower bound of n for the test complexity. We construct complete test sets whose worst case size is quadratic in n. We start again with the proof of a lower bound.

Theorem 3 (lower bound) Let A_n be an n-bit CST-adder over the library LIB_{CST}. Then we have $TC_{CFM}(A_n) \geq n$.

Proof: For the lower bound consider the $n-1$ boundary block-*SEL*s and the output of the rightmost A_1-cell. Notice that for each cell there exists a testable cellular fault that requires the output P in the correct case. Fix the boundary block-cell x at a bit position i. P as value at the output requires P on the upper and right input (R cannot be generated at the right input!). Now, it follows by induction on n that the right and upper input of any boundary block-cell with bit position $< i$ necessarily have value P and R, respectively, and that the two rightmost output pins of the rightmost A_1-cell have value R. In particular neither a boundary block-*SEL* to the right of x nor the rightmost A_1 can have output P at the same time as x. ■

The theorem above shows that in CFM there is no hope to find a complete test set which has a sublinear size compared to the number of input bits. In the SAFM this is possible and can really be done as we have shown above. In this sense the CFM is computationally harder than the SAFM. A lower bound for a CBFM cannot be given in a general form. It strongly depends on the subset of faults that have to be tested.

For the construction of a complete test set it is useful to determine the input combinations which can be generated at *SEL*-cells and the differences that can be propagated from the output of *SEL*-cells to the primary outputs. For this consider a *SEL*-cell x. The upper input of x, the right input of x and the right inputs to the *SEL*-cells on the path from x to the output are pairwise independent from each other, since they all correspond to disjoint subcircuits. The value R can never be generated at an input of a carry-*SEL*. A and G can never be generated at the right output of A_1-cells. All the remaining input combinations can be generated at the inputs of *SEL*-cells. We therefore obtain the following lemma for any CST-adder A_n.

Lemma 3 (generation for CFM)

– At the inputs of carry-SELs 9 different input combinations described by XY with $X, Y \in \{A, P, G\}$ can be generated.

– At block-SELs that receive their upper input from an A_1-cell 6 different input combinations given by XY with $X \in \{R, P\}$and $Y \in \{A, P, G\}$ can be generated.

– At the remaining SEL-cells 12 different input combinations described by XY with $X \in \{A, P, G, R\}, Y \in \{A, P, G\}$ can be generated.

It turns out that the differences X/Y with $X/Y \neq A/P, P/A, G/R, R/G$ can be made visible at the primary outputs independent of the fault location from where they have to be propagated. We therefore call them *easy* for the following. For the remaining differences propagation may be impossible. They are called *hard*. Note that hard differences are exactly the differences which lead to a faulty result only on the sum+1 line. More precisely, we obtain the following lemma whose proof follows from the overall structure of CST-adders and Table I:

Lemma 4 (propagation for CFM)

– Hard differences at the rightmost outputs of the rightmost A_1-cell and at the outputs of boundary block or boundary carry cells cannot be made visible. (The final sum plus 1 is not of interest!)

– If an easy difference has to be propagated from the output of a carry-SEL, consider the tree of carry-SELs and propagate the difference along the uniquely determined path to the output of this tree by combining it with the neutral element P.

– If a hard difference has to be propagated from the output of a carry-SEL, that is not a boundary carry cell, again consider the tree of carry-SELs and the corresponding path to the output. There exists at least one side input on this path, which is a right edge. Set this input to G and thus map the hard difference to an easy one. The remaining side inputs are set to P.

– If an easy difference has to be propagated from the bottom output of a block-SEL, consider the uniquely determined path to the primary output at the same bit position and set the side inputs, which all are outputs of carry-SELs, to P.

– If a hard difference has to be propagated from the bottom output of a block-SEL, which is not a boundary block cell, again consider the uniquely determined path to the primary output at the same bit position. Set the first side input to G, the remaining ones to P.

For an n-bit CST-adder A_n define n_{SEL} as the number of SEL-cells in A_n, n_{carry} as the number of carry-SELs in A_n, $n_{lowblock}$ as the number of block-SELs that receive their upper input from an A_1-cell, and $n_{highblock} = n_{SEL} - n_{carry} - n_{lowblock}$. Note that the equations $n_{carry} = n - 1$ and $n_{lowblock} = n - 1$ hold. We combine the lemmas above to obtain the following theorem:

Theorem 4 (CFM-test) Let A_n be an n-bit CST-adder. Then a complete test set $\mathcal{T}_{CFM}$ for A_n in CFM with

$$\begin{aligned} |\mathcal{T}_{CFM}| &= 8n + 18n_{carry} + 12n_{lowblock} + 24n_{highblock} \\ &= 24n_{SEL} - 10n + 18 \\ &= O(|A_n|) \end{aligned}$$

can be constructed in time $O(n^2)$.

Proof: For each cell x and each input combination which can be generated at x, two test patterns are required, namely one to propagate an easy difference and one to propagate a hard difference from the output line of x. Thus, 8 patterns suffice for the complete test of an A_1-cell. The remaining number of tests suffices for the test of the SEL-cells according to Lemma 3 and 4. Since $O(|A_n|) = O(n^2)$, the time bound follows, if we make sure that each test requires constant construction time. We use again Lemma 3 and 4. From the lemmas it follows that we only have to store the values at special positions for the generation and the propagation, but not all input values. Therefore we perform a bottom-up computation and store all information in the structure tree of the adder. Thus, each node in the tree is visited only once. ■

From an asymptotic point of view the test is not optimal in many cases, e.g., for the classical conditional sum adder. At the moment it is open, whether there exists a test with test complexity $O(n)$ for any n-bit CST-adder. It should be mentioned at this point that the test set can easily be adapted to a general cell based static fault model: Patterns that test for cellular faults which are not contained in the fault model are simply deleted.

D. *Testability of CST-adders in the RPDFM*

We now focus on a powerful dynamic fault model. We first show that the adders described in Section II are not fully robust path delay fault (RPDF) testable and so a reconstruction is given to obtain a testable circuit. The modifications described are similar to the one applied in [39] to design testable adders. The construction of the test sets is completely different, since the structures of the adders are different. For the modified adders we prove that they are fully RPDF testable with a test set of size $O(n^2 d_{left}(A_n))$. Furthermore, we show that this is optimal up to the factor of $d_{left}(A_n)$.

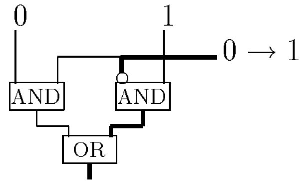

Fig. 7. Untestable path in a multiplexer

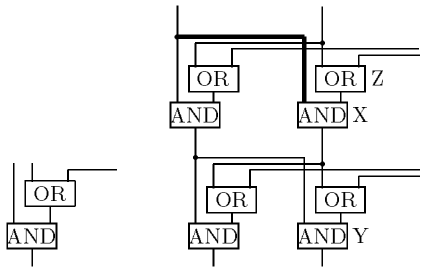

Fig. 8. New cell and resulting carry-stage

Construction of testable adders

We start with a consideration of the carry-tree of the CST adders. The carry-computation is realized by multiplexers. It is easy to see that MUXes are not RPDF testable without the application of value R=(0,1) at the northern input (see Fig. 7), since for a robust test we must apply non-controlling values at the side inputs along the path to be tested. But R=(0,1) can never be generated at the northern inputs of the carry-tree. One posibility to overcome this difficulty is to modify the function on the *don't care set*, so that the same function is realized on the applicable values.

A cell for such a function is shown on the left side in Fig. 8. If we replace the multiplexers of the carry computation by the cell on the left side in Fig. 8 we obtain a circuit

TABLE III
NEW *SEL*-CELL

$\circ$	A	P	G	R
A	A	A	A	A
P	A	P	G	R
G	G	G	G	G
R	A	A	A	A

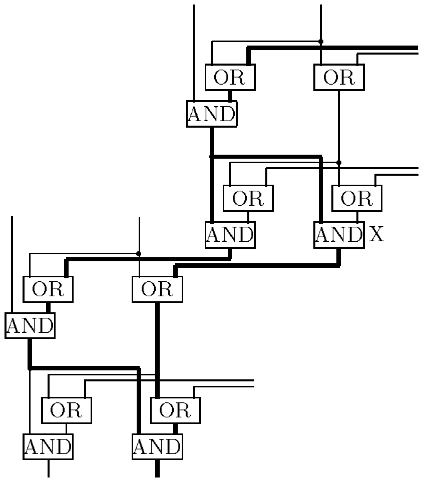

Fig. 9. Carry-tree with reconvergency

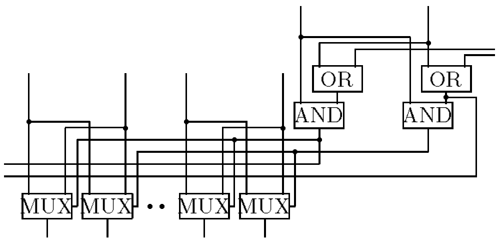

Fig. 10. SEL-stage carry-output

computing the function described by Table III. This is the same operation on the applicable values A, P and G as shown in Table I, i.e., such a replacement will not change the correct functional behavior of the circuit.

The new SEL-cell is "locally" RPDF testable, but the composition of two such cells (illustrated in Fig. 8 on the right) still has some untestable paths. When we have a look at the path starting at the upper left input and then going to the AND-gates X and Y, it is not difficult to check that this path cannot be robustly tested, since we have a reconvergency at Y. By examining the thick line in Fig. 8 we can see, that a stuck-at 1 fault on this signal cannot be propagated to the outputs: A test for a stuck-at 1 fault requires a 0 at the upper left input and this determines the values of the outputs. Thus, we can remove the thick line from the circuit and also the AND-gate X. We have to directly connect the output of the OR-gate Z with the former output of gate X. It can be checked that the resulting circuit has no untestable paths with respect to the RPDFM. By induction we obtain that the vertical composition of n such cells, called carry stage, is RPDF testable without the application of the value R.

Next, we have a look at some carry-stages simultaneously, so that the output of the first one is the side input of the next one (see Fig. 9). It is shown that this can lead again to an untestable path due to a reconvergence (thick

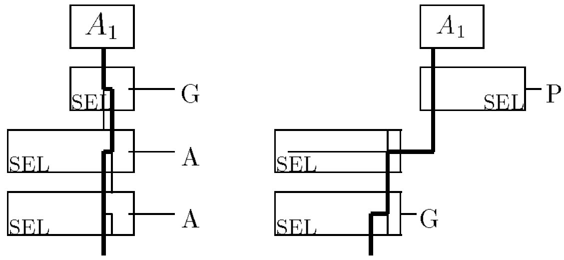

Fig. 11. Sum paths and carry-sum paths

lines) on the next stage, if the next stage is a vertical composition of at least two cells.

It turns out that the idea of "path doubling" and subsequent removal of stuck-at redundancies as it is presented in [40] for the removal of PDF untestabilities can also be applied in the case of reconverging paths: The AND-gate X in Fig. 9 has to be doubled. One AND-gate is then used for the carry-computation and the other one for the multiplexers. On the left input of the AND-gate for the carry-computation is an untestable stuck-at 1 fault. So it can be removed as in Fig. 8. Again this construction can easily be generalized to the complete carry-tree and results in the following modifications: Carry cells whose output lines correspond to a vertical edge in the carry tree are replaced by the upper right cells in Fig. 9. Carry cells whose output lines correspond to a non-vertical edge in the carry tree are replaced by the cell given in Fig. 10. Boundary cells are left unchanged.

All in all, we obtain modified adders of the CST with identical functional behavior, that, as we will see in the following, are fully RPDF testable. These testable adders are called RCST-adders (= robust CST-adder).

Testability and test complexity

In this subsection we prove the RPDF testability of all paths in the adders obtained from the modifications described in the last subsection. We assume $k \in \{\frac{n}{2}, \ldots, n-1\}$. This assumption is based on a heuristic optimizing the generator with respect to runtime (see Section IV). It simplifies the construction of the test patterns and the calculation of the sizes of the test sets.

We divide the paths to be tested into three classes:

Sum paths: Paths, that begin at the s(um)-outputs $s_1^{(j)}$ and $s_0^{(j)}$ of the A_1-cells (left thick path in Fig. 11)

Carry paths: Paths in the carry tree, i.e., the paths start at the c(arry)-outputs $c_1^{(j)}$ and $c_0^{(j)}$ of the A_1-cells and then run to the outputs of the carry-SEL in the last stage

Carry-sum paths: Paths, beginning at the c(arry)-outputs $c_1^{(j)}$ and $c_0^{(j)}$ of the A_1-cells and then running through the AND-gates in the multiplexers in the SEL-cells (right thick line in Fig. 11)

The test patterns for the test of the paths will be denoted by the values at the output of the A_1-cells, using the coding introduced in Section II. The number of tests obtained in this way has to be multiplied by a factor ≤ 4 to determine the test complexity of the whole adder. (It is easy to see that all paths in the A_1-cells are RPDF testable.)

The patterns are denoted as indicated by the following example: PR,..,PR,PR/GP,PR,..,PR,GP,PR,..,PR means, that all A_1-cells keep their value except one, which has the PR/GP-change. This notion is only used in Lemma 5. All the other proofs use arguments on the structure of the adders to describe the test sets. If we have a bound for the number of paths, this directly implies a bound for the number of test patterns, too.

Lemma 5 (sum paths) All sum paths $S(A_n)$ of an RCST-adder are RPDF testable by $\leq 2 \cdot n^2$ test patterns.

Proof: At the s-outputs of the A_1-cells we only have the values R or P (consider the EXOR-NOT-combination and Table I). The signals directly run through the SEL-stages to the outputs (see Fig. 11 at the left). At the SEL-stages we use all combinations of A and G at the side inputs, since the right (left) upper input of a multiplexer in a SEL-cell is tested using the value A (G). This is sufficient to sensitize a sum path and thus we get a number of tests exponential in the number of SEL-stages where the path runs through. The number is limited by $\log n$, since $k \in \{\frac{n}{2}, \ldots, n-1\}$. A detailed analysis shows that we obtain patterns of the form: PR,..,PR,PR/GP,PR,..,PR,GP,PR,..,PR,AP,PR,..,PR.

The positions where the values AP and GP must be applied can be directly derived from the structure tree. We get the upper bound by using this construction for all A_1-cells separately. ■

The RCST adders were constructed in such a way, that all paths are robust PDF testable. In the following we omit the detailed construction of the test patterns. Instead, we prove upper and lower bounds for the size of the test set. For this, we use the structure of the adders as described in Section II. We first give an upper bound on the number $C(A_n)$ of carry paths depending on the left-depth $d_{left}(A_n)$ of the specific adder A_n. For this, we determine $C^i(A_n)$, which describes the number of carry paths in an n-bit RCST-adder without boundary SELs, i.e., A_n is a subcircuit of a larger adder corresponding to a subtree of the structure tree, whose root node is not a boundary cell.

Lemma 6 (carry paths) Let A_n be an n-bit RCST adder. Then $C^i(A_n) \leq c \cdot d_{left}(A_n) \cdot n$ and $C(A_n) \leq 2 \cdot c \cdot d_{left}(A_n) \cdot n$ hold for all $c \geq 3$.

Proof: It can directly be seen that the first proposition holds for $n = 2$, since $C^i(A_2)$ is equal to 5. The general case is proven by induction. The number $C^i(A_n)$ of "inner" carry paths is given by

$$C^i(A_n) \quad = \quad C^i(A_k) + C^i(A_{n-k}) + n - k. \qquad (1)$$

Equation (1) can easily be derived from the recursive definition of the adders in Section II and the modification given above. (It follows by induction, that exactly $n-k$ carry paths end at the right carry output of such an inner $(n-k)$-bit adder A_{n-k}. Thus, only $n-k$ carry paths of A_{n-k} continue in two ways in the bottom stage.) We assume that the first proposition of the lemma holds for all adders of bit size $< n$. Thus, using $d_{left}(A_n) = max\{d_{left}(A_k), d_{left}(A_{n-k}) + 1\}$ we distinguish three cases:

Case 1: $n = 2$ and $k = 1$

$$C^i(A_n) = 5$$

Case 2: $k \geq 2$ and $n - k = 1$

$$\begin{aligned} C^i(A_n) &\leq 3 + c \cdot d_{left}(A_k) \cdot k \\ &\leq c \cdot d_{left}(A_n) \cdot n \end{aligned}$$

Case 3: $k \geq 2$ and $n - k > 1$

$$\begin{aligned} C^i(A_n) &= C^i(A_k) + C^i(A_{n-k}) + n - k \\ &\leq c \cdot d_{left}(A_k) \cdot k + c \cdot d_{left}(A_{n-k}) \cdot (n-k) + n - k \\ &< c \cdot (d_{left}(A_k) \cdot k + (d_{left}(A_{n-k}) + 1) \cdot (n-k)) \\ &\leq c \cdot d_{left}(A_n) \cdot (k + n - k) \\ &= c \cdot d_{left}(A_n) \cdot n \end{aligned}$$

This proves the first proposition. For $C(A_n)$ we have:

$$C(A_n) \quad = \quad 2 \cdot C^i(A_{n-k}) + C(A_k) \qquad (2)$$

which is less than

$$\begin{aligned} &2 \cdot c \cdot d_{left}(A_{n-k}) \cdot (n-k) + 2 \cdot c \cdot d_{left}(A_k) \cdot k \\ &= 2 \cdot c \cdot d_{left}(A_n) \cdot n. \end{aligned}$$

This finishes the proof. ■

Using the result of Lemma 6 an upper bound for the number of carry-sum paths can be proven.

Lemma 7 (carry-sum paths) Let A_n be an n-bit RCST adder. Then we have for the number of carry-sum paths $CS(A_n) \leq 12 \cdot d_{left}(A_n) \cdot n^2$

Proof: The number of carry-sum paths is given by

$$\begin{aligned} CS(A_n) \quad &= \quad CS(A_k) + 2 \cdot CS(A_{n-k}) \\ &\quad +2 \cdot (n-k) \cdot C(A_k). \end{aligned} \qquad (3)$$

Again the formula can be derived from Section II. We obtain the factor $2 \cdot CS(A_{n-k})$, because in the new SEL-chain each path can continue in two ways. $2 \cdot C(A_k) \cdot (n-k)$ describes all carry paths of A_k that can branch out in the SEL-stage. The proposition holds for $n = 2$, since $CS(A_2)$ is equal to 8. d denotes a constant that will be determined in the following. Thus we obtain:

$$\begin{aligned} &CS(A_n) \\ &= CS(A_k) + 2 \cdot CS(A_{n-k}) + 2 \cdot (n-k) \cdot C(A_k) \\ &\leq d \cdot d_{left}(A_k) \cdot k^2 + 2 \cdot d \cdot d_{left}(A_{n-k}) \cdot (n-k)^2 \\ &\quad +4 \cdot c \cdot d_{left}(A_k) \cdot k \cdot (n-k) \\ &\leq d_{left}(A_n) \cdot (d \cdot k^2 + 4 \cdot c \cdot (n-k) \cdot k \\ &\quad +2 \cdot d \cdot (n-k)^2) \\ &\leq d \cdot d_{left}(A_n) \cdot n^2 \end{aligned}$$

The last estimation holds for $d \geq 4c = 12$ and $k \geq \frac{n}{2}$. This can be seen by a straightforward computation. ■

We resume the results of Lemmas 5-7.

Theorem 5 (upper bound) All paths in an n-bit RCST-adder A_n are robust PDF testable by a test set of size $O(n^2 \cdot d_{left}(A_n))$.

Corollary 1: The complete test set is bounded by $8 \cdot n^2 + 24 \cdot d_{left}(A_n) \cdot n \cdot (2 \cdot n + 1)$.

Now, we give a general lower bound for the size of the test set for an arbitrary member of the class of RCST-adders.

Theorem 6 (lower bound) Each test set of an adder of the RCST has size $\Omega(n^2)$.

Proof: We have a look at the inputs from a_1 to $a_{\frac{n}{4}}$ and to the outputs of the adder from the $\frac{3}{4}n$-th to the n-th position. We only examine the carry outputs of the A_1-cells. Each a_i can influence each s_j and thus there must exist a carry-sum path for each pair (a_i, s_j). Thus all in all $\frac{n^2}{16}$ carry-sum paths must be tested.

The RCST adders are defined recursively and by the assumption $k \in \{\frac{n}{2}, \ldots, n-1\}$ it follows that all the input signals a_i and all the output signals s_j are only connected by two carry signals (see Fig. 1).

We now examine the carry-sum paths in more detail. The paths starting at the c-outputs of the A_1-cells must run through the multiplexers in the SEL-chains. Therefore a P must be present at the northern inputs of the multiplexer. To generate a P we must use a G or an A at the inputs of the A_1-cells (see Table I), but this destroys a parallel test since A and G are *left-stable* elements (see Table III).

Thus, the paths enumerated above must be tested separately and we obtain the lower bound. ■

We finish with an examination of two special adders of the RCST and prove optimal asymptotic bounds for their test sets: The *classical conditional sum* adder (CCSA), where k is recursively chosen as $\frac{n}{2}$, and the *carry ripple* adder (CRA), where k is recursively chosen as $n-1$.

We prove the lower bounds for these adders using the equations (1), (2) and (3).

The carry paths of the CCSA $C(A_n)$, with $k = \frac{n}{2}$, are given by:

$$\begin{aligned} C(A_n) &= C(A_k) + 2 \cdot C^i(A_{n-k}) \\ &\geq 2 \cdot C^i(A_{n-k}) \\ &\geq \frac{n}{4} \cdot log\frac{n}{2} \end{aligned}$$

Thus, the number of carry-sum paths can be calculated by equation (3):

$$\begin{aligned} CS(A_n) &\geq 3 \cdot CS(A_k) + n \cdot log\frac{n}{4} \cdot \frac{n}{8} \\ &= \Omega(n^2 \cdot logn) \end{aligned}$$

So the size of the test set is optimal from the asymptotic point of view, since the left-depth of the CCSA is $d_{left}(CCSA) = logn$.

The same considerations for the CRA show, that the number of carry-sum paths are bounded by $\Omega(n^2)$. This is also optimal, since it has left-depth $d_{left}(CRA) = 1$.

IV. Generator

In this section we focus on the problem of finding an area-time optimal adder of the CST. First a definition of the computation models for area and time is given (Section IV-A). Then area-time optimal adder design is formulated as a dynamic programming problem (Section IV-B). We start with a description of a straightforward implementation. Unfortunately, this generation method is impracticable for large bitlengths because the memory requirement is too large. Therefore we integrate point location into the multi-dimensional dynamic programming approach which makes the method applicable for practice because it reduces the memory requirement. The computational complexity of the algorithm proposed is investigated in Section IV-D. We close Section IV by presenting some heuristics to speed-up the computation (Section IV-E). Statistical experiments illustrate and motivate the different steps shown.

We do not focus on the construction of RCST adders, as they have been defined in the last section. But the methods presented can be directly transfered.

A. Computation models

The delay model is based on the *intrinsic-plus-fanout delay model* which is also used by many industrial VLSI design systems, e.g., *VENUS S-S, Semicustom Design System, 1.0µm* from SIEMENS, Munich [30], and by the logic synthesis and optimization benchmarks of the Microelectronics Center of North Carolina [31]. In this model, the description of a cell x with one output pin contains [41]

– the *intrinsic delay times* $t_{up}(x,i)$ and $t_{down}(x,i)$ of the ith input of cell x, i.e., the propagation delay times of x with respect to the rising edge i and the falling edge i, respectively, if the capacitive load CL on the output of x is equal to 0,

– the *capacitance* $c_{in}(x,i)$ *of the ith input of* x measured in *load units* (LU), where one load unit corresponds to the input capacitance of a two-input NAND gate,

– the *load dependences* $ld_{up}(x)$ and $ld_{down}(x)$ of the output of x measured in ns/LU.

For our purpose we only have to consider basic cells with one output pin. Since $t_{up}(x,i)$, $t_{down}(x,i)$, $t_{up}(x,j)$ and $t_{down}(x,j)$ differ from each other only a little bit for $i \neq j$ and $x \in \{AND, OR, EXOR, MUX\}$, we consider $t(x) := \max_i\{t_{up}(x,i), t_{down}(x,i)\}$ as the intrinsic delay time $t(x)$ of cell x in order to simplify matters. Furthermore, we consider $ld(x) := \max\{ld_{up}(x), ld_{down}(x)\}$ as the load dependence $ld(x)$ of cell x.

The timing of a CMOS cell depends on the capacitive load on the cell output, i.e., the input capacitances of the following CMOS cells and the wiring. The delay time $d(x)$ of cell x for a given capacitive load CL can be approximately determined by the equation $d(x) = t(x) + ld(x) \cdot CL$ where $CL := C_{cells} + C_{wires}$ holds and C_{cells} denotes the sum of all the capacitances of the cell input pins at the driven network. C_{wires} denotes the sum of the wiring capacitance. Since the design presented is done on a symbolic design level and since the physical wire lengths are unknown, we assume $C_{wires} = 0$ for this paper. Note that the wire lengths, i.e., the wire capacitances, can easily be approximated if the physical design is done along the lines of the symbolic design. The cell characteristics used in our examples are taken from the SIEMENS library mentioned above. For each cell x except the MUX-cell the capacitances of the inputs of x differ from each other only a little bit so that we can confine ourselves to the maximum of these values. The capacitance of the select input s of the MUX-cell is much larger than the capacitance of the remaining data inputs (denoted by d) so that we differentiate between these two values.

The area-time optimal adder design presented in the following section is done on a symbolic design level. Therefore, we approximate the area of a circuit by the *cell area*, i.e, we do not take the interconnect area into account. The lengths of the wires cannot be estimated in general, since they largely depend on the technology used. For some design processes this simplifying assumption is realistic , i.e., *channelless gate arrays* or *sea-of-gates* [33], [32]. Note once more that the wire length can easily be approximated if the physical design is done along the lines of the symbolic design.

B. Formulation as dynamic programming problem

The problem can be described as follows:

Given the number n of bits and the delay t_n (e.g. measured in units of $0.1ns$) of the n-bit addition.

Construct an area-minimal n-bit adder of the conditional sum type with delay $\leq t_n$ if it exists.

We denote such an adder by *performance oriented conditional sum type adder of type 1* (POCSTA$_1$).

Now, assume that for any $m < n$ and $t_m \leq t_n$ an m-bit POCSTA$_1$ $A_m(t_m)$ of delay $\leq t_m$ is already designed, if it exists. Then, the problem to solve is to find any $k \in \{1, \ldots, n-1\}$ such that there exists

– an $(n-k)$-bit POCSTA$_1$ $A_{n-k}(t_{n-k})$ of delay less than t_{n-k} with

$$t_{n-k} := t_n - t(MUX) - 2 \cdot c_{in}(MUX, d) \cdot \max\{ld(x)\},$$

where $\max\{ld(x)\}$ equals the maximum of $ld(AND)$, $ld(OR)$, $ld(NOT)$, $ld(EXOR)$, $ld(MUX)$. Obviously, the last two terms only depend on the characteristics of the library L used, i.e., $t_{n-k} = t_n - f_1(L)$, where f_1 is a function depending only on the library L chosen.

– a k-bit POCSTA$_1$ $A_k(t_k)$ of delay less than t_k with

$$t_k = t_n - f_2(L, n-k+1),$$

where f_2 is a function depending only on the library L chosen and the size $n-k+1$ of the selection stage, which causes a delay due to the input capacitances of the SEL-cells and to the drivers which have to be inserted if the fanout given by $n-k+1$ becomes too large. This fanout problem can be solved by a heuristic algorithm, e.g., that of Singh and Sangiovanni-Vincentelli [41]. The results shown in this paper are based on a straightforward solution of the fanout problem, namely by simply cascading the drivers.

If k is not uniquely determined by the constraints above, we choose any one minimizing the resulting area.

Now, consider an array $AREA$ of n rows and t_n columns. Assume that for all $m < n$ and $t_m \leq t_n$ the entry $AREA[m, t_m]$ is equal to $+\infty$ if no m-bit adder of the conditional sum type with delay $\leq t_m$ exists, and equal to the area of an m-bit POCSTA$_1$ $A_m(t_m)$ with delay $\leq t_m$, otherwise.

```
main()
{ init;
  for (m = 3; m <= n; m++)
    for (t_m = t_2; t_m <= t_n; t_m++) find1(m, t_m);
  print_POCSTA_1(n, t_n);
}
```

Fig. 12. Dynamic program for optimal adder design

Thus, we have to solve the following *dynamic programming problem*:

Find $k \in \{1, \ldots, n-1\}$ such that

$$\begin{aligned} &AREA[n-k, t_n - f_1(L)] \\ &+AREA[k, t_n - f_2(L, n-k+1)] \\ &+2(n-k+1) \cdot area(MUX) + f_3(L, n-k+1) \end{aligned}$$

is minimal, where f_3 is a function which only depends on the library L used and the size $n-k+1$ of the selection stage. f_3 specifies the area of the drivers which have to be inserted in the case that $n-k+1$ is too large.

Let $find1(n, t_n)$ be the function computing k as well as the area of the corresponding POCSTA$_1$ and storing these values in the arrays K and $AREA$ given the entries of $K[m, t_m]$, and $AREA[m, t_m]$ for all $m < n$ and $t_m \leq t_n$. Then the program given in Fig. 12 computes a POCSTA$_1$ for given n and t_n. Procedure *init* initializes the arrays, namely the entries for $m = 1$ and $m = 2$ because there is no area-time trade-off for the 1-bit POCSTA$_1$ and the 2-bit POCSTA$_1$. The term t_2 denotes the delay of the 2-bit POCSTA$_1$. The runtime of the function $find1(m, t_m)$ is $\leq c' \cdot m$. Thus, the runtime of the dynamic program above performing area-time optimal adder design is $\leq c \cdot n^2 \cdot t_n$. The constant c is small.

Theorem 7: Let n and t_n be given. An n-bit POCSTA$_1$ with delay $\leq t_n$ can be generated in time $O(n^2 \cdot t_n)$ and in space $8 \cdot n \cdot t_n + O(1)$ bytes if there exists such an n-bit adder.

Proof: Because of the discussion above we only have to prove that $8 \cdot n \cdot t_n + O(1)$ bytes suffice. Each entry of the array $AREA$ occupies 4 bytes, that is $4 \cdot n \cdot t_n$ bytes for the whole array. After having computed the area of the area-minimal adder which satisfies the delay requirement, the corresponding adder has to be constructed. This can be done top-down because of array K which occupies $4 \cdot n \cdot t_n$ bytes memory, too. ■

The algorithm has been applied to the 16-bit POCSTA$_1$s. There are 21 different POCSTA$_1$s by using the SIEMENS library mentioned above and a maximum fanout up to 20 LU. The area of the slowest adder (with delay $17.7ns$) is two thirds of the area of the fastest one (with delay $6.9ns$). There is no area-time optimal adder with tree depth equal to 4, i.e., the adder computing the carries through a total balanced tree is not area-time optimal with respect to the library used. Fig. 13 shows an example of a 16-bit

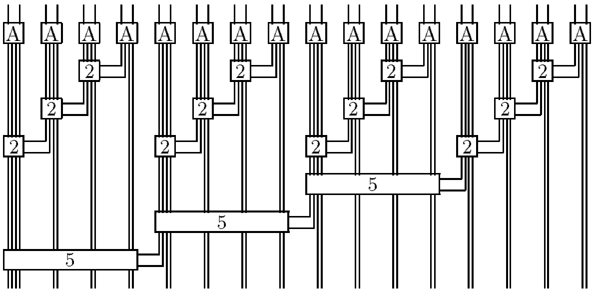

Fig. 13. Schematic structure of the 16-bit POCSTA_1 $A_{16}(9.3\mathrm{ns})$ (A and $x \in \{2, 5\}$ stand for A_1 and SEL_x, respectively)

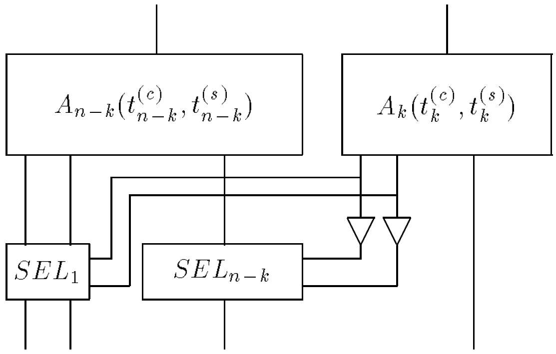

Fig. 14. Tuning up the adders.

POCSTA_1 generated for the input parameter $t_n = 9.3ns$. The cell area of the circuit obtained is $291548 \mu m^2$. Fig. 16 on the left illustrates the area-time trade-off. (The exact values of the trade-off are given in [42].) Here, we have recorded the area of the minimum area POCSTA_1 which meets the given delay.

Looking at Fig. 1 we observe that the $2 \cdot k$ rightmost output pins of the k-bit adder are directly connected to the output pins of the whole n-bit adder whereas the 2 carry output pins of the k-bit adder have to drive the multiplexers of the selection stage. Thus, in order to obtain better area-time optimal adders, it seems to be reasonable to differentiate between the signal delays of paths going from the input pins of an adder to the carry output pins and the signal delays of those paths going from the input pins to the remaining output pins.

As in Section III we denote paths going from the input pins to the carry output pins of an adder by *carry-paths* and paths going from the input pins to the remaining output pins of an adder by *sum-paths*. The delays of the carry-paths and of the sum-paths are denoted by *carry-delay* and *sum-delay*, respectively.

Differentiating between carry-delays and sum-delays has the consequence that a generator parametrized by the operands' bitlength n, by an upper bound $t_n^{(c)}$ for the carry-delay, and by an upper bound $t_n^{(s)}$ for the sum-delay computing an area minimal n-bit adder with carry-delay $\leq t_n^{(c)}$ and sum-delay $\leq t_n^{(s)}$ is demanded.

In any adder constructed by the generator proposed in the last paragraph the carry-delay always equals the sum-delay (see Fig. 1). Thus, to realize the idea of differentiating between carry-delays and sum-delays we have to "disconnect" the carry-paths and the sum-paths from each other. This can be done by directly routing the carries driving the selection stage to the carry-SELs and by disconnecting these paths from the sum-paths by introducing non-inverting buffers (denoted by $\triangle$) in front of the block-SELs (see Fig. 14). Of course, the buffers are not inserted if the cut is at position $k \geq n - \alpha$, where α is a constant which has to be determined dependent on the library used. Note that this modification does not influence the test complexity and testability.

Now, the problem is given as follows:

Given the number n of bits and the delays $t_n^{(c)}$ and $t_n^{(s)}$ (e.g. measured in units of $0.1ns$) of the n-bit addition.

Construct an area-minimal n-bit adder of the CST with carry-delay $\leq t_n^{(c)}$ and sum-delay $\leq t_n^{(s)}$.

We denote such an adder by *performance oriented conditional sum type adder of type 2* (POCSTA_2). Analogously to the solution presented above, this problem can be formulated as a dynamic programming problem. Assume that for any $m < n$, $t_m^{(c)} \leq t_n^{(c)}$, and $t_m^{(s)} \leq t_n^{(s)}$ an m-bit POCSTA_2 $A_m(t_m^{(c)}, t_m^{(s)})$ of carry-delay $\leq t_m^{(c)}$ and sum-delay $\leq t_m^{(s)}$ is already designed, if it exists. Then, the problem to solve is to find $k \in \{1, \ldots, n-1\}$ such that there exists

– an $(n-k)$-bit POCSTA_2 $A_{n-k}(t^{(c)}_{n-k}, t^{(s)}_{n-k})$ of carry-delay less than $t^{(c)}_{n-k}$ and sum-delay less than $t^{(s)}_{n-k}$ with

$$t^{(c)}_{n-k} := t_n^{(c)} - t(MUX) - 2 \cdot c_{in}(MUX, d) \cdot \max\{ld(x)\}$$

and

$$t^{(s)}_{n-k} := t_n^{(s)} - t(MUX) - 2 \cdot c_{in}(MUX, d) \cdot \max\{ld(x)\}.$$

Note that the equations $t^{(c)}_{n-k} = t_n^{(c)} - g_1(L)$ and $t^{(s)}_{n-k} = t_n^{(s)} - g_1(L)$ hold where g_1 is a function depending only on the library L chosen.

– a k-bit POCSTA_2 $A_k(t_k^{(c)}, t_k^{(s)})$ of carry-delay $t_k^{(c)}$ and sum-delay $t_k^{(s)}$ with $t_k^{(s)} := t_n^{(s)}$ and $t_k^{(c)} := \min\{tc, ts\}$ with

$$\begin{aligned} tc = t_n^{(c)} &- t(MUX) - (c_{in}(MUX, s) \\ &+ c_{in}(DRIVER)) \cdot \max\{ld(x)\}, \end{aligned}$$

i.e, $tc = t_n^{(c)} - g_2(L)$, and

$$ts = t_n^{(s)} - g_3(L, n-k),$$

where g_3 is appropriately defined.

If k is not uniquely determined by the constraints above, we choose any one minimizing the resulting area.

Now, consider a 3-dimensional array $AREA$ of size $n \cdot t_n^{(c)} \cdot t_n^{(s)}$. Assume that for all $m < n$, $t_m^{(c)} \leq t_n^{(c)}$, and $t_m^{(s)} \leq t_n^{(s)}$ the entry $AREA[m, t_m^{(c)}, t_m^{(s)}]$ is equal to $+\infty$ if no m-bit adder of the conditional-sum type with carry-delay $\leq t_m^{(c)}$ and sum-delay $\leq t_m^{(s)}$ exists, and equal to the area of an m-bit POCSTA_2 $A_m(t_m^{(c)}, t_m^{(s)})$ with carry-delay $\leq t_m^{(c)}$ and sum-delay $\leq t_m^{(s)}$, otherwise.

```
main()
{ init; t_n^(c) = t_n; t_n^(s) = t_n;
  for (m = 3; m ≤ n; m++)
    for (t_m^(c) = t_2^(c); t_m^(c) ≤ t_n^(c); t_m^(c)++)
      for (t_m^(s) = t_2^(s); t_m^(s) ≤ t_n^(s); t_m^(s)++)
        find2(m, t_m^(c), t_m^(s));
  print_POCSTA2(n, t_n);
}
```

Fig. 15. Improved dynamic program for optimal adder design

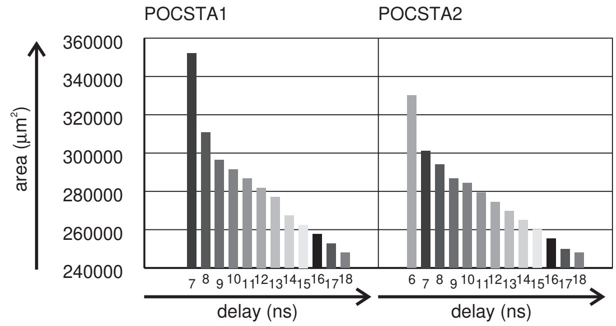

Fig. 16. Area-time trade-offs

Thus, we have to solve the following dynamic programming problem:

Find $k \in \{1, \dots, n-1\}$ such that

$$\begin{aligned}&AREA[n-k, t_n^{(c)} - g_1(L), t_n^{(s)} - g_1(L)]\\&+AREA[k, g_4(L, n-k, t_n^{(c)}, t_n^{(s)}), t_n^{(s)}]\\&+2 \cdot area(DRIVER) + 2(n-k+1) \cdot area(MUX)\\&+g_5(L, n-k)\end{aligned}$$

is minimal, where $g_4 = \min\{t_n^{(c)} - g_2(L), t_n^{(s)} - g_3(L, n-k)\}$ and g_5 is a function depending on the library L used and the size $n-k$ of the remaining part of the selection stage. g_5 specifies the area of the drivers which have to be inserted in the case that $n-k$ is too large.

Let $find2(n, t_n^{(c)}, t_n^{(s)})$ be the function computing k as well as the area of the corresponding POCSTA$_2$ and storing these values in the arrays K and $AREA$ given the entries $K[m, t_m^{(c)}, t_m^{(s)}]$ and $AREA[m, t_m^{(c)}, t_m^{(s)}]$ for all $m < n$, $t_m^{(c)} \le t_n^{(c)}$, and $t_m^{(s)} \le t_n^{(s)}$. Then the program given in Fig. 15 realizes the generator required. Note that only during the generation process we differentiate between carry-delays and sum-delays. The designer himself only asks for an area minimal n-bit adder with delay less than some bound t_n. The runtime of function $find2(m, t_m^{(c)}, t_m^{(s)})$ is $\le c'' \cdot m$. Thus, the runtime of the program is $\le C \cdot n^2 \cdot t_n^2$. The constant C is small. The memory requirement increases from $8 \cdot n \cdot t_n$ bytes to $8 \cdot n \cdot t_n^2$ bytes.

Theorem 8: Let n and t_n be given. An n-bit POCSTA$_2$ with carry-delay $\le t_n$ and sum-delay $\le t_n$ can be generated in time $O(n^2 \cdot t_n^2)$ and in space $8 \cdot n \cdot t_n^2$ bytes if it exists.

We have applied this new algorithm to the 16-bit POCSTA$_2$s, too. As it was hoped, the trade-off consisting of 31 different POCSTA$_2$s is much finer than the previous one which consists of only 21 different POCSTA$_1$s. The fastest POCSTA$_2$ is $0.9ns$ (about 13 percent) faster than the fastest POCSTA$_1$ although its area is 5 percent smaller. Once again there is no POCSTA$_2$ of tree depth equal to 4. Fig. 16 illustrates the area-time trade-off of 16-bit POCSTA$_2$s and the comparison to the adders of type 1. See [42] for the exact values.

C. Reduction to point location

In Section IV-B, we have shown how to obtain a very fine area-time trade-off for adders of the conditional sum type. But, there remain the following problems: The generator proposed always requires about $8 \cdot n \cdot t_n^2$ bytes of memory. Of course, the memory requirement can be decreased to $4 \cdot n \cdot t_n^2$ bytes by not storing the cut positions, i.e., by not computing the entries of array K. (Note that they are only needed during the computation of $print\_POCSTA_2(n, t_n)$. The entries needed can be computed in a top-down phase.) Nevertheless, this requirement is too high. Consider Table IV where the memory requirement is shown for the two cases. The first column denotes the operands' bitlength, the second one denotes an approximation of the delay of the slowest n-bit POCSTA$_2$ measured in $0.1ns$, and the third and fourth column denote the memory the generator requires. So to generate the trade-off of the 64-bit adders, where the delay of the slowest one is about $70.4ns$, the generator requires $4 \cdot 64 \cdot 704^2 = 126877696$ bytes or $8 \cdot 64 \cdot 704^2 = 253755392$ bytes memory.

TABLE IV
MEMORY REQUIREMENTS OF THE GENERATOR COMPUTING POCSTA$_2$S

n	t_n $[0.1ns]$	$8nt_n^2$ $[MByte]$	$4nt_n^2$ $[MByte]$
16	171	4	2
32	352	32	16
64	704	254	127
128	1408	2030	1015
256	2816	16240	8120
512	5632	129922	64961

In the remainder of this section we show how to refine the dynamic program in order to solve this problem. There is an important feature of our problem which has not yet been exploited, namely that tightening up the delay requirements results in larger adders. Let $\mathbf{N} := \{1, 2, \dots\}$, $\mathbf{N}_q := \{1, \dots, q-1, q\}$, $\mathbf{N}_{+\infty} := \mathbf{N} \cup \{+\infty\}$, and $\mathbf{N}_\infty := \mathbf{N} \cup \{-\infty, +\infty\}$. Then, the following statement follows:

Lemma 8: The function $AREA[m, ., .] : \mathbf{N}_{t_n^{(c)}} \times \mathbf{N}_{t_n^{(s)}} \to \mathbf{N}_{+\infty}$ is monotone decreasing in both variables.

Proof: Obviously, if we decrease the delay time we can only construct larger adders, since more computations must be performed in parallel. The case that we can construct smaller adders with smaller delay can never occur, since in this case we can also choose the smaller adder for the position in the matrix where more area is available. ■

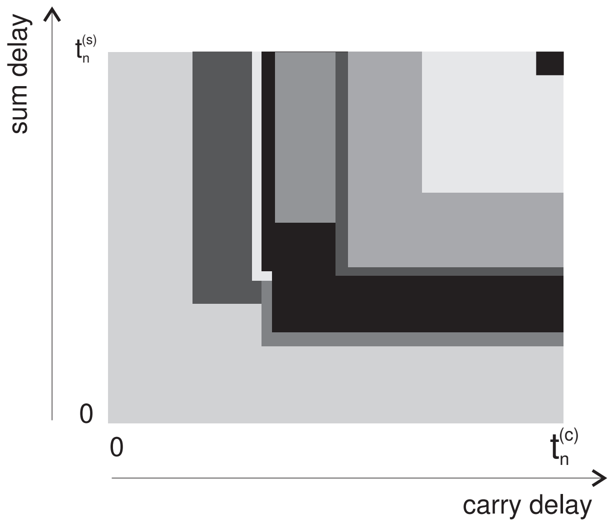

Fig. 17. Monotony of function $AREA[8,.,.]$

The lemma is also valid, if we take the interconnection area into account, although the resulting adders might be different. Note that the proof of the lemma is independent of the structure of the circuits considered. Thus, the statement also holds for other structures like other adder types, multipliers and so on.

Fig. 17 illustrates this statement for $AREA[8,.,.]$. The carry-delays and the sum-delays are given at the x-axis and at the y-axis, respectively. We observe that there are 10 different faces. The face adjacent to the left bottom corner denotes those requirements not met by any 8-bit adder. Each other face denotes POCSTA$_2$s having the same area and meeting the corresponding requirements. The monotony of the function $AREA[m,.,.]$ is expressed by the fact that $AREA[m,x_1,y_1] \leq AREA[m,x_2,y_2]$ if $p_1 = (x_1,y_1)$ *dominates* $p_2 = (x_2,y_2)$, i.e., if $x_1 \geq x_2$ and $y_1 \geq y_2$. We denote this relation by $p_1 \geq p_2$. The relation $\geq$ can be extended to faces in a canonical manner. A face α dominates a face β ($\alpha \geq \beta$) if there is a point $p_1 \in \alpha$ which dominates a point $p_2 \in \beta$. Because of the monotony the statement $(\alpha \geq \beta \; and \; \beta \geq \alpha) \Rightarrow \alpha = \beta$ holds. Note that the relation $\geq$ is not total.

Obviously, only the entries at the "lower" boundaries of the faces are of interest because changing the time requirements of an entry inside a face does not affect the area of the resulting adder. Thus, we associate the following directed acyclic graph $G_m = (V_m, E_m)$ to the function $AREA[m,.,.]$: A pair $(x,y) \in \mathbf{N}_\infty \times \mathbf{N}_\infty$ is in V_m if the entry $AREA[m,x,y]$

– differs from the entries at its left ($AREA[m,x-1,y]$) and its bottom side ($AREA[m,x,y-1]$), or

– is equal to the entries at its left and its bottom side, but differs from the entry $AREA[m,x-1,y-1]$, or

– differs from the entries at its bottom and its right side ($AREA[m,x+1,y]$), or

– differs from the entries at its left and its top side ($AREA[m,x,y+1]$).

Assume here $AREA[m,x,y] = -\infty$, if $x \leq 0$, $y \leq 0$, $x > t_n^{(c)}$, or $y > t_n^{(s)}$.

A vertex $(x,y) \in V_m$ is called a *vertex of type* α if α is the area of the POCSTA$_2$ constructed by our generator for the input parameters m, x, and y. The vertices $s = (-\infty,-\infty)$ and $t = (+\infty,+\infty)$ complete V_m. We call s the *source of* G_m and t the *target of* G_m.

Let (x_1,y_1) and (x_2,y_2) be two vertices of type α. Then, there is an edge from (x_1,y_1) to (x_2,y_2) if and only if $x_1 = x_2$ and $y_1 > y_2$ or $x_1 < x_2$ and $y_1 = y_2$. The graph is completed by

– a directed edge from source s to vertex $(x,y) \in V_m$ if and only if $y = t_n^{(s)}$,

– a directed edge from vertex $(x,y) \in V_m$ to target t if and only if $x = t_n^{(c)}$,

– a directed edge from $(x,y+1)$ to (x,y), if $(x,y) \in V_m \setminus \{s,t\}$ is a vertex of indegree 0. If $(x,y+1)$ is not in V_m yet, add it. In this case, $(x,y+1)$ lies on a directed edge which has to be broken. Note that the edge starts from a vertex which dominates (x,y),

– a directed edge from (x,y) to $(x+1,y)$, if $(x,y) \in V_m \setminus \{s,t\}$ is a vertex of outdegree 0. If $(x+1,y)$ is not in V_m yet, add it and break the edge on which this new vertex lies. Note that the edge ends in a vertex which dominates (x,y).

In order to realize function $find2(n, t_n^{(c)}, t_n^{(s)})$, it suffices to store the graphs G_m for all $m < n$. If this is done, the problem to be solved is *point location* [35] for G_m. In general, point location is defined as follows: Given a subdivision of the plane induced by a planar graph G and a test point p, identify the region which contains p.

In general point location can be solved by the following lemma:

Lemma 9 ([43]) Point location in an N-vertex planar subdivision can be effected in $O(\log N)$ time using $O(N)$ storage, given $O(N \log N)$ preprocessing time.

In the next subsection we show that the application of the lemma does not improve the generator on the assumption that every entry of $AREA[m,.,.]$ is a vertex of G_m. However, experiments showed that this worst case seems to never occur. In order to present these experiments, we have to work out the key idea of the proof of the lemma above. This leads to the introduction of notions we need in the following. The exact proof of the lemma can be found in [43], [35].

The crucial point of this lemma is the construction of a *monotone complete set of paths in* G. A set M of paths is called *complete* if the union of all the paths contains the graph. A set M of paths is called *monotone* if for any two paths C_i and C_j of M, the vertices of C_i which are not vertices of C_j "lie on the same side" of C_j, i.e., the paths are ordered. Thus, one can apply the bisection principle, i.e., binary search, to G. If there are r paths in M and the longest path has p vertices, the search time uses $O(\log p \cdot \log r)$ in the worst-case. The search time can be reduced to $O(\log N)$ by the *bridged chain method* introduced by Edelsbrunner, Guibas and Stolfi [43].

TABLE V

MEMORY REQUIREMENT OF THE IMPROVED ALGORITHM ($|V_n|$ ($|Faces_n|$) DENOTES THE NUMBER OF VERTICES (FACES) OF G_n)

n	t_n [0.1ns]	$\mid Faces_n \mid$	$\mid V_n \mid$	$\mid V_n \mid / t_n$
20	220	56	227	1.03
25	275	87	330	1.20
30	330	117	460	1.39
35	385	150	608	1.57
40	440	177	739	1.67
45	495	199	766	1.54
50	550	232	914	1.66
55	605	270	1066	1.76
60	660	313	1211	1.83
65	715	355	1343	1.87
70	770	387	1502	1.95
75	825	416	1581	1.91
80	880	436	1658	1.88
85	935	476	1821	1.94
90	990	518	1976	1.99

We now come to the application of the lemma to our problem. Note that the G of the lemma above corresponds to G_m in our application. It is easy to see that there exists a monotone complete set M_m of paths in G_m of size r, where r is the amount of different faces of G_m. Each path corresponds to a face α, connects source s to target t, and contains the vertices of type α. The only difficulty occurring during the construction of these paths is due to those faces α where there is no vertex of type α on the northern or on the eastern border of the diagram.

This difficulty can be solved by sorting the faces first with respect to "domination" (topological sorting). This results in a list $\alpha_1, \ldots, \alpha_r$ such that face α_k does not dominate α_j for all $j \in \{1, \ldots, r-1\}$ and $k \in \{j+1, \ldots, r\}$. We construct the paths along the lines of this sorted list. A path which corresponds to a face α where there is no vertex of type α on the northern side or on the eastern side nestles to the path previously constructed. Because of the two last cases of the definition of E_m there always exists such a path for any such face.

Thus, the construction of the monotone complete set of paths in G_m is much easier than in the general case, where the graph has to be "regularized"

D. *Complexity of the algorithm*

In the worst case estimation we have to assume that each entry of $AREA[m,.,.]$ is a vertex of G_m, i.e., $N = t_n^2$ and that there are $O(N)$ different faces in $AREA[m,.,.]$. Thus, the generator would require at least time $O(n^2 \cdot t_n^2 \cdot \log t_n)$ and space $O(n \cdot t_n^2)$. This estimation claims that the refined algorithm is slower than the first one and that the memory requirement has not decreased.

However, the experiments made (see Table V) show that for $n \leq 90$ the size of V_n equals $f(n) \cdot t_n$ where $f(n)$ is inbetween 1 and 2. The size of V_{50}, e.g., is 914 whereas the delay of the slowest 50-bit POCSTA$_2$ (measured in $0.1ns$) is about 550, i.e., $f(50) = 1.66$. Roughly estimated the improved program has to store about 75000 vertices altogether to compute the trade-off of the 84-bit adders. Note furthermore that the bottom border of each face consists of about 4 vertices in the average so that there are about 19000 paths to be considered. Thus, the data structure of the bridged chain method (see [35] pp.61–62) contains about $\frac{d}{d-1} \cdot (75000 + 2 \cdot 19000)$ entries, where $d \geq 2$ can be chosen arbitrary and where the actual runtime increases proportional to d. Each entry contains 4 additional pointers (see [35] pp.61–62). Thus, choosing d between 2 and 5, the generator requires between 3 and 5.5 MByte memory to compute the trade-off of the 84-bit adders. Note that the generator of Chan et. al requires about 60 MByte memory [5]. This raises hope that for practical n the size of V_n is $\leq c \cdot t_n$ where c is a small constant. Working on this principle the following lemma holds:

Lemma 10: If the size of V_n is $\leq c \cdot t_n$, then the algorithm proposed has runtime $O(n^2 \cdot t_n^2 \cdot \log t_n)$ and memory requirement $c' \cdot n \cdot t_n$ bytes with c' being a small constant.

Proof: Because of the discussion above, we only have to show how to construct the graphs G_m $(m < n)$.

To compute whether an entry $AREA[m,x,y]$ is a vertex of V_m, only nine entries have to be known, namely $AREA[m,i,j]$ with $x-1 \leq i \leq x+1$ and $y-1 \leq j \leq y+1$. By computing the array $AREA[m,.,.]$ from left to right, starting with columns $AREA[m,0,.]$, $AREA[m,1,.]$, $AREA[m,2,.]$, …, the graph G_m can be computed by storing only three columns at any moment of the computation. No column has to be computed twice.

Thus, the whole data structure consists of the graphs G_i $(i = 1, \ldots, n)$ and three columns of size $8 \cdot t_n$. As each graph G_i has size $O(t_n)$, the data structure for G_i (used by the bridged chain method) has size $O(\frac{d}{d-1} \cdot t_n)$ (see [35]). Thus, the generator requires storage space in the order of $n \cdot t_n$.

The runtime of point location is in the order of $\log t_n$. To compute an entry of $AREA[m,.,.]$, the procedure solving point location has to be called $m-1$ times. Thus, the total runtime of the generator is $O(n^2 \cdot t_n^2 \cdot \log t_n)$. ■

E. *Heuristics*

Of course, the runtime of the generator can be heuristically speeded up by using the monotony property of function $AREA[m,.,.]$. Clearly

$$(*)\ AREA[m,x,y+i] = AREA[m,x,y] \quad (i=0,\ldots,p)$$

if $AREA[m,x,y]$ equals $AREA[m,x,y+p]$. Thus, it is reasonable to start with the computation of the entries

$$AREA[m,x,\min\{i \cdot q_1, t_m\}]$$

for a constant q_1 and $i = 0, 1, \ldots, \lceil \frac{t_m}{q_1} \rceil$. If

$$AREA[m,x,j \cdot q_1] = AREA[m,x,\min\{(j+1) \cdot q_1, t_m\}]$$

TABLE VI

Amount of n-bit POCSTA$_2$s where (**) does not hold.

n	[20..50]	[50..70]	[70..90]
amount of n-bit POCSTA$_2$s where (**) does not hold	1	4	1

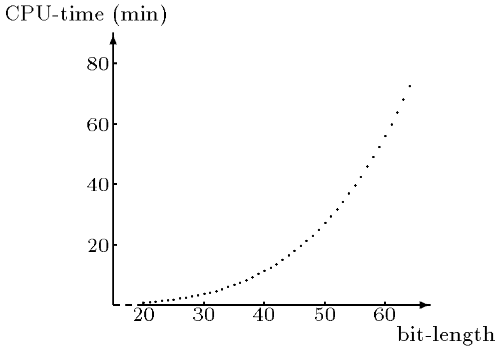

Fig. 18. CPU-time for various bitlengths

the entries $AREA[m,x,u]$ ($u = j \cdot q_1 + 1, \ldots, \min\{(j+1) \cdot q_1, t_m\} - 1$) have not to be computed because of $(*)$. Otherwise, we compute the entries

$$AREA[m, x, j \cdot q_1 + \min\{k \cdot q_2, q_1 - 1\}]$$

for a constant $q_2 \leq \frac{q1}{2}$ and $k = 1, \ldots, \lceil \frac{q_1}{q_2} \rceil - 1$ and so on.

A further improvement is motivated by the observation that the bitlength $n-k$ of the left adder $A_{n-k}(t^{(c)}_{n-k}, t^{(s)}_{n-k})$ is less than the bitlength k of the right adder $A_k(t^{(c)}_k, t^{(s)}_k)$ of an n-bit POCSTA$_2$ in most cases. Table VI shows the amount of n-bit POCSTA$_2$s where the inequation

$$(**) \quad k \geq \frac{n}{2} - 1$$

does not hold ($20 \leq n \leq 90$) using the SIEMENS library mentioned above. Thus, it seems to be tolerable to restrict the algorithm to cuts $k \geq \frac{n}{2} - 1$. This modification of the algorithm has only small influence upon the trade-off itself. However it speeds up the actual runtime by a factor 2.

V. Conclusion

We presented a realistic and systematic method for constructing area-time optimal adders of the CST together with efficient complete test sets with respect to a chosen fault model *FM*.

FM can be chosen as a static (SAFM, CFM) or dynamic (RPDFM) fault model. We have proven that all generated adders can be tested using small test sets. The optimal test set for an adder of the (R)CST can be directly generated from the structure tree of the adder.

We presented various methods to generate the area-time optimal adders. The complexity analysis of the generators and the experimental results showed that it is also possible to generate adders of large bit-lengths. In Fig. 18 the time for the computation on a Sparc 2+ with 32 MB of memory is presented.

The idea of the generation is based on point location. This technique can also be applied to other circuit structures, not only CST-adders. One key point is the monotony of the function $AREA$ in Section IV-C. This property is also valid, if we take the interconnection area into account, although the resulting adders might be different. Finally to speed up the runtime additionally some heuristics were presented.

References

[1] R.P. Brent and H.T. Kung, "A regular layout for parallel adders", *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260–264, 1982.

[2] B. Becker and R. Kolla, "On the construction of optimal time adders", *FUNDAMENTA INFORMATICAE*, vol. XII, pp. 205–220, 1989.

[3] S. Turrini, "Optimal group distribution in carry-skip adders", in *Proceedings of the 9th Symposium on Computer Arithmetic*, 1989, pp. 96–103.

[4] B.W.Y. Wei and C.D. Thompson, "Area-time optimal adder design", *IEEE Transactions on Computers*, vol. C-39, no. 5, pp. 666–675, 1990.

[5] P.K. Chan, D.F. Schlag, C.D. Thomborson, and V. Oklobdzija, "Delay optimization of carry-skip adders and block carry-lookahead adders using multidimensional dynamic programming", *IEEE Transactions on Computers*, vol. C-41, no. 8, pp. 920–930, 1992.

[6] R.E. Ladner and M.J. Fischer, "Parallel prefix computation", *Journal of ACM*, vol. 27, no. 4, pp. 831–838, 1980.

[7] M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[8] M.A. Breuer and A.D. Friedman, *Diagnosis and reliable design of digital systems*, Computer Science Press, 1976.

[9] P. Banerjee and J.A. Abraham, "Generating tests for physical failures in MOS logic circuits", in *Proceedings of IEEE International Test Conference*, 1983, pp. 554–559.

[10] C.C. Beh, K.H. Aray, C.E. Radke, and K.E. Torku, "Do stuck-at fault models reflect manufacturing defects?", in *Proceedings of IEEE International Test Conference*, 1982, pp. 34–42.

[11] J. Galiay, Y. Crouzet, and M. Vergniault, "Physical versus logical fault models MOS LSI circuits: impact on their testability", *IEEE Transactions on Computers*, vol. C-29, no. 6, pp. 527–531, 1980.

[12] P. Lamoureux and V.K. Agarwal, "Non-stuck-at fault detection in NMOS circuits by region analysis", in *Proceedings of IEEE International Test Conference*, 1983, pp. 129–137.

[13] J. Ferguson and J.P. Shen, "Extraction and simulation of realistic CMOS faults using inductive fault analysis", in *Proceedings of IEEE International Test Conference*, 1988, pp. 475–484.

[14] W. Maly, "Realistic fault modeling for VLSI testing", in *Proceedings of the 24th ACM/IEEE Design Automation Conference DAC'87*, 1987, pp. 173–180.

[15] B. Becker and U. Sparmann, "A uniform test approach for RCC-adders", in *Proceedings of the 3rd Aegean Workshop on Parallel Computation and VLSI Theory*, 1988, pp. 288–300.

[16] A.D. Friedman, "Easily testable iterative systems", *IEEE Transactions on Computers*, vol. C-22, no. 12, pp. 1061–1064, 1973.

[17] E. Lindbloom, J.A. Waicukauski, B. Rosen, and V. Iyengar, "Transition fault simulation by parallel pattern single fault propagation", in *Proceedings of IEEE International Test Conference*, 1986, pp. 542–549.

[18] G.L. Smith, "A model for delay faults based upon paths", in *Proceedings of IEEE International Test Conference*, 1985, pp. 342–349.

[19] R.L. Wadsack, "Fault modeling and logic simulation of CMOS and MOS integrated circuits", *The Bell System Technical Journal*, vol. 57, 1978.

[20] S. Devadas, K. Keutzer, and S. Malik, "A synthesis-based test generation and compaction algorithm for multi-faults", *JETTA*, vol. 4, pp. 91–104, 1993.

[21] B. Milne, "Testability, 1985 technology forecast", *Electronic Design*, vol. 10, pp. 143–166, 1985.

[22] T.W. Williams, W. Daehn, M. Grützner, and C.W. Starke, "Comparison of aliasing errors for primitive and non-primitive

polynomials", in *Proceedings of IEEE International Test Conference*, 1986, pp. 282–288.

[23] M.S. Abadir and H.K. Reghbati, "Functional testing of semiconductor random access memories", *Computing Surveys*, vol. 15, pp. 1983, 175-198.

[24] B. Becker and J. Hartmann, "Optimal-time multipliers and c-testability", in *Proceedings of the 2nd Annual Symposium on Parallel Algorithms and Architectures*, 1990, pp. 146–154.

[25] B. Becker and U. Sparmann, "Computations over finite monoids and their test complexity", in *Proceedings of the 19th International Symposium on Fault-Tolerant Computing*, 1989, pp. 299–306.

[26] W. Daehn and J. Mucha, "A hardware approach to self-testing of large PLA's", *IEEE Transactions on Circuits and Systems*, vol. CAS-28, 1981.

[27] J. Ferguson and J.P. Shen, "The design of easily-testable VLSI array multipliers", *IEEE Transactions on Computers*, vol. C-33, no. 6, pp. 554–560, 1984.

[28] H. Fujiwara and K. Kinoshita, "A design of programmable logic arrays with universal tests", *IEEE Transactions on Computers*, vol. C-30, no. 11, 1981.

[29] J. Sklansky, "Conditional sum addition logic", *IRE-EC*, vol. 9, pp. 226–231, 1960.

[30] SIEMENS-München, "VENUS S-S, Semicustom Design System, ACMOS 4H", December 1990.

[31] S. Yang, "Logic synthesis and optimization benchmarks user guide (version 3.0)", Tech. Rep., Microelectronics Center of North Carolina, Research Triangle Park, NC 27709, January 1991.

[32] R.L. Geiger, P.E. Allen, and N.R. Strader, *VLSI-Design Techniques for Analog and Digital Circuits*, McGraw Hill International Editors, 1990.

[33] R. Kolla, P. Molitor, and H.G. Osthof, *Einführung in den VLSI-Entwurf*, B.G. Teubner Stuttgart, 1989.

[34] B. Becker and P. Molitor, "A performance driven generator for efficient testable conditional-sum adders", in *Proceedings of the European Design Automation Conference EURODAC'92*, 1992, pp. 370–375.

[35] F.P. Preparata and M.I. Shamos, *Computational Geometry*, Springer Verlag, 1985.

[36] B. Becker, "Efficient testing of optimal-time adders", *IEEE Transactions on Computers*, vol. C-37, no. 9, pp. 1113–1121, 1988.

[37] R.D. Eldred, "Test Routines Based on Symbolic Logical Statements", *Journal of ACM*, vol. 6, no. 1, pp. 33–36, 1959.

[38] A.K. Pramanick and S.M. Reddy, "On the design of path delay fault testable combinational circuits", in *Proceedings 1990 IEEE International Fault Tolerant Computing Symp.*, 1990, pp. 374–381.

[39] B. Becker and R. Drechsler, "A time-optimal robust path delay fault self-testable adder", in *Proceedings of the European Design Automation Conference EURODAC'92*, 1992, pp. 376–381.

[40] P. Ashar, S. Devadas, and K. Keutzer, "Testability properties of multilevel logic networks derived from binary decision diagrams", in *Advanced Research in VLSI*, 1991.

[41] K.J. Singh and A. Sangiovanni-Vincentelli, "A heuristic algorithm for the fanout problem", in *Proceedings of the 27th ACM/IEEE Design Automation Conference DAC'90*, 1990, pp. 357–360.

[42] B. Becker, R. Drechsler, and P. Molitor, "On the implementation of an efficient performance driven generator for conditional-sum-adders", in *Proceedings of the European Design Automation Conference EURODAC'93*, 1993, pp. 402–407.

[43] E. Edelsbrunner, L.J. Guibas, and J. Stolfi, "Optimal point location in a monotone subdivision", *SIAM Journal on Computing*, vol. 15, no. 2, pp. 317–340, 1986.

[44] R. Drechsler, B. Becker, and P. Molitor, "A performance oriented generator for robust path delay fault testable adders", in *Proceedings of the 4th ITG/GI-Workshop 'Testmethoden und Zuverlässigkeit von Schaltungen und Systemen'*, 1992.

Bernd Becker studied mathematics and computer science at University of Saarland, Germany, from 1973 to 1982. He received the Dipl.-Math, Dr.rer.nat. and the Dr.habil. degrees in 1979, 1982, and 1988, respectively.
Between 1979 to 1988, he was with the Sonderforschungsbereich "Electronic Speech Recognition" (79-81), Institute for Computer Science and Applied Mathematics (81-83), and Sonderforschungsbereich "VLSI Design Methods and Parallelism" (84-88), all at University of Saarland. Presently, he is a professor for "Complexity Theory and Efficient Algorithms "at the J.W.Goethe-University Frankfurt.
His research interests include data structures and efficient algorithms (for circuit design), design, test and verification of VLSI circuits.
Dr. Becker is a member of IEEE and the ACM.

Rolf Drechsler received the diploma from the J.W.Goethe-University Frankfurt, Germany, in 1992 where he is currently working towards his Ph.D. degree at the Department of Computer Science.
His research interests include logic synthesis, testing, and genetic algorithms.

Paul Molitor received the Diplom, Ph.D., and Habilitation degrees in 1982, 1986, and 1992, respectively, from the University of Saarland, Germany.
Between 1982 to 1992, he was with the Sonderforschungsbereich "VLSI Design Methods and Parallelism" at the University of Saarland. After being visiting professor at the University of Halle (1992/93) and the University of Freiburg i.Br. (1993), he was an associate professor in the Computer Science Department of the Humboldt-University Berlin in 1993/94. Since 1994, he is a full professor at the Martin-Luther University Halle, Germany, where he is the chairman of the Institute of Computer Science.
His research interests include logic and physical synthesis, hierarchical VLSI design, algorithms, and data structures.
Dr. Molitor is a member of IEEE and the ACM.