October 1986

UILU-ENG-86-2237
CSG-58

COORDINATED SCIENCE LABORATORY
College of Engineering

ON THE C-TESTABILITY OF GENERALIZED COUNTERS

Abhijit Chatterjee
Jacob A. Abraham

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS none
2a. SECURITY CLASSIFICATION AUTHORITY not applicable	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE not applicable	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CSG-58 UILU-ENG-86-2237	5. MONITORING ORGANIZATION REPORT NUMBER(S) none
6a. NAME OF PERFORMING ORGANIZATION University of Illinois Coordinated Science Laboratory 6b. OFFICE SYMBOL (If applicable) n.a.	7a. NAME OF MONITORING ORGANIZATION 1. General Electric Company 2. Semiconductor Research Corporation
6c. ADDRESS (City, State and ZIP Code) 1101 West Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State and ZIP Code) 1. 1 River Road, Schenectady, NY 12345 2. P.O. Box 12053, Research Triangle Park, NC 27709
8a. NAME OF FUNDING/SPONSORING ORGANIZATION see 7a. 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER 1. none 2. 84-06-049-4
8c. ADDRESS (City, State and ZIP Code) see 7b.	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: n.a. PROJECT NO.: n.a. TASK NO.: n.a. WORK UNIT NO.: n.a.
11. TITLE (Include Security Classification) On the C-Testability of Generalized Counters	
12. PERSONAL AUTHOR(S) Abhijit Chatterjee and Jacob A. Abraham	
13a. TYPE OF REPORT technical 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) October, 1986 15. PAGE COUNT 41
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	C-testability, counters, iterative logic arrays, test sets

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report investigates the testability of a class of circuits, called counters, that perform the addition of sets of input bits of equal arithmetic weight. These circuits consist of full- and half-adders interconnected in an iterative manner defined by the counting process. The general class of counter circuits contain reconvergent fanout and are structurally not as regular as one- or two-dimensional iterative logic arrays.

A model for analyzing the structure of counter circuits is proposed. Several schemes for generating test sets that exploit the iterative structure of counter circuits are presented. The testability of such circuits is enhanced by imposing certain design constraints on them. Some methods for generating easily testable counter structures are proposed. It is shown that counters can be always designed to be testable with either eight or nine tests, irrespective of the input size.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code) 22c. OFFICE SYMBOL

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

ON THE C-TESTABILITY OF GENERALIZED COUNTERS

Abhijit Chatterjee

Jacob A. Abraham

Computer Systems Group
Coordinated Science Laboratory
University of Illinois
Urbana, IL 61801

ABSTRACT

This paper investigates the testability of a class of circuits called counters, that perform the addition of sets of input bits of equal arithmetic weight. These circuits consist of full and half adders interconnected in an iterative manner defined by the counting process. The general class of counter circuits contain reconvergent fanout and are structurally not as regular as 1 or 2 dimensional Iterative Logic Arrays.

A model for analyzing the structure of counter circuits is proposed. Several schemes for generating test sets that exploit the iterative structure of counter circuits are presented. The testability of such circuits is enhanced by imposing certain design constraints on them. Some methods for generating easily testable counter structures are proposed. It is shown that counters can be always designed to be testable with either 8 or 9 tests, irrespective of the input size.

* This research was supported by the General Electric Company and the Semiconductor Research Corporation under contract SRC RSCH 84-06-046.
A preliminary version of this paper will be presented at the Int. Conf. on Computer Aided Design, Nov. 1986.
A. Chatterjee is on leave from the General Electric Research and Development Laboratories, Schenectady, N.Y. 12301.

1. Introduction

In the testing of iterative logic, the test size can be substantially reduced by exploiting the repetitive nature of the hardware. Patterns of test vectors are applied, that simultaneously test the circuit modules [1-4]. In the case of 1-D Iterative Logic Arrays (ILAs), the signal flow is in one direction and the inputs to each of the modules are directly controllable. In the case of 2-D ILAs, the signal flow is in two perpendicular directions and the controllability issue is more complicated. In each of the above cases, there does not exist any reconvergent fanout and the interconnection between the modules is fairly rigid. There exists a class of iterative circuits called counters which are not as regular as the circuits described above. A counter performs the summation of sets of input bits of equal arithmetic weight. A binary encoding of this summation is generated on an appropriate number of output lines. For any given set of inputs, several counter circuits can be designed to perform the above summation. Further, these circuits contain reconvergent fanout.

A model for the counting process is first presented. Some properties of counters are derived and it is shown that a subset of the class of counter circuits can be designed without the use of half adders. A recursive construction for the above subset of the class of counter circuits is presented. Counters based on this construction are shown to be C-testable with a test length of 8.

A labelling scheme for a tree based representation of counter circuits is developed. This scheme is used to generate tests for counters. It is shown that by a process of 'growing' the trees in this tree based representation, any counter can be designed to be testable in 8 or 9 tests. Two such schemes are presented and timing issues are discussed. Finally, the testability of different configurations of counter circuits is discussed and a branch and bound algorithm for generating tests for any counter configuration is presented.

2. Previous Work

The problems of testing 1-D Iterative Logic Arrays are almost solved [1-5]. The testing of 2-D ILA's has been widely investigated and is well understood. Conditions for the C-testability of 2-D ILA's are presented in [6]. The testing of Exclusive-OR tree structures is discussed in [7]. In

[8], a recursive procedure for generating an O(n) test set for an n input general tree structure is presented. It has been shown [9] that if each of the paths from the inputs to the root node is C-testable, then the entire tree can be tested with a test length proportional to the logarithm of the number of inputs. However, there do not exist any results on generalized counters.

3. The Counting Process : A Model and it's Description

A unary counter of full and half adders sums n bits of equal weight and generates a weighted binary sum on $\lfloor \log_2 n \rfloor + 1$ output lines [10-11]. A tree model for counter circuits and some related properties are first discussed.

In the following, a half or full adder is referred to as an *addition module*. An input or output of an addition module of the counter, hereby referred to as a *counter node*, can be reached from one of the counter inputs by traversing a sequence of addition modules $a_{m_1}, a_{m_2}, \ldots a_{m_k}$. The traversal through an addition module is from one of it's inputs to either it's sum or carry output.

DEFINITION 1: The *carry count* of a traversal from an input of the counter to a counter node is defined to be the number of addition modules in the sequence $a_{m_1}, a_{m_2}, \ldots a_{m_k}$, that are traversed from an input to the corresponding carry output.

In the following, the arithmetic weight of a counter node is referred to simply as the *weight* of the node.

LEMMA 1: If the weight of an input to the counter is 2^k and the carry count of a traversal from this input to a counter node is C, then the weight of the node is 2^{k+C}.

PROOF: The weight of the carry output of an addition module is twice the weight of the inputs to the module. If the carry count is C, the weight of the counter node reached by the corresponding traversal must be $2^C 2^k = 2^{k+C}$. □

3.1. The Tree Model for Counters

Consider the addition of 7 bits of equal weight by repeated addition of groups of 3 bits. This can be represented by a set of trees as shown in Figure 1.2 for the 7 input unary counter of Figure 1.1. The nodes of the trees represent the full adders of the counter and are assigned the same labels as the full adders of Figure 1.1 to which they correspond. The output branch of each node of a tree corresponds to the sum output of the associated full adder. The carry outputs of the full adders represented by the nodes of the tree of Figure 1.2a are connected to the inputs of the tree of Figure 1.2b, which are labelled accordingly. This process is recursive and leads to Figure 1.2c, until no more carries are generated. The values on the output branches of the root nodes of these trees represent the weighted binary count of the number of 1's at the inputs to the tree of Figure 1.2a.

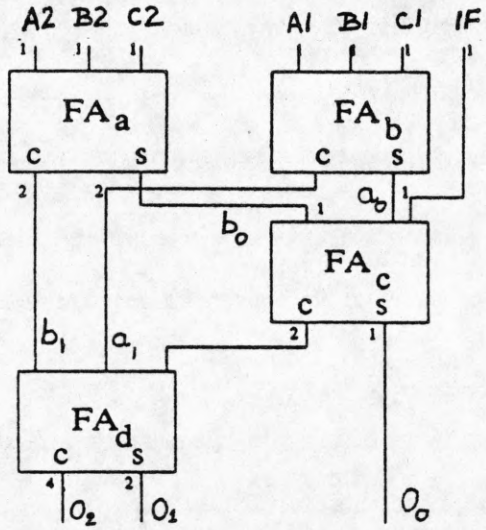

Figure 1.1: 7 input unary counter

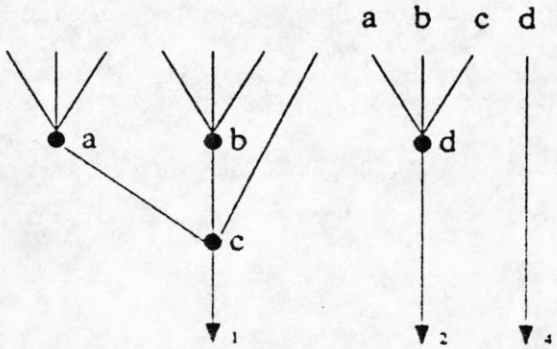

Figure 1.2: Tree representation for 7 input counter

In general, n bits can be summed by repeated addition of groups of 2 or 3 bits, $n \geq 3$. This can be represented by a set of trees, where each node of a tree has 2 or 3 input branches respectively. If a node of a tree has 2 input branches, it represents a half adder. If it has 3 input branches, it represents a full adder. Each tree represents a bit slice of the counter. Each node of a tree representing a full adder, reduces 3 branches to 1 branch, while each node representing a half adder reduces 2 branches to 1 branch. This is said to be a *3 to 1* or *2 to 1 reduction of branches* respectively. It is always possible to do 3 to 1 reductions of branches until either 2 or 3 branches are left to be reduced by the root node of the tree. In the former case, the root node corresponds to a half adder, while in the latter it corresponds to a full adder. In our design methodology, a half adder, if at all necessary, is placed at the root node of a tree. Such trees as above are called *reduction trees* and a reduction tree of input size k is denoted by R_k.

DEFINITION 2: Of the set of reduction trees that define the counter structure, the *fundamental tree* is defined to be the one whose inputs are all controllable inputs to the counter.

DEFINITION 3: A *derived subtree* is defined as a tree which has one or more of it's inputs fed by carries from another bit slice of the counter. The tree from which these carries are derived is called the *parent tree* of the *derived subtree*.

In general, there are many ways of forming the reduction trees that constitute each bit slice of the counter. Further, for each construction of the above set of reduction trees that define the counter, there are many ways that the carries in between these trees can be interconnected. Different possibilities as above, represent different hardware realizations of a counter with regard to the manner in which the addition modules are interconnected.

The output of every tree in the set of reduction trees for a counter represents the parity of it's corresponding inputs. If the weight of the input bits to such a parity tree is 2^i, then this tree is said to belong to bit slice i and the *weight* of this bit slice is 2^i.

LEMMA 2: Every hardware realization of a counter can be uniquely represented by a set of reduction trees with the following properties:

Property 1: There exists a unique mapping between the full and half adders in the hardware realization of a counter and the nodes of the reduction trees in it's corresponding tree model.

Property 2: The physical interconnection of the carries between two reduction trees in bit slices i and $i+1$, defines a one to one correspondence between the nodes of the reduction tree of bit slice i and the inputs of the reduction tree of bit slice $i+1$ to which the carries from the addition modules represented by the above nodes are connected.

PROOF: The carry count is used to assign a weight to every counter node. The interconnection of the addition modules of the counter through counter nodes of the same weight represent trees where the nodes of the trees correspond to addition modules and the branches of the trees correspond to counter nodes of the same weight. This one to one mapping verifies Property 1. Property 2 can then be trivially verified. □

EXAMPLE 1: Consider the 9 input unary counter of Figure 2. The tree representation of this counter is depicted in Figure 3. The labels on the nodes of the trees correspond to the addition modules they represent. The labels on the inputs to the reduction trees represent addition modules whose carries are connected to those inputs. The inputs to reduction trees without any labels represent controllable inputs to the counter.

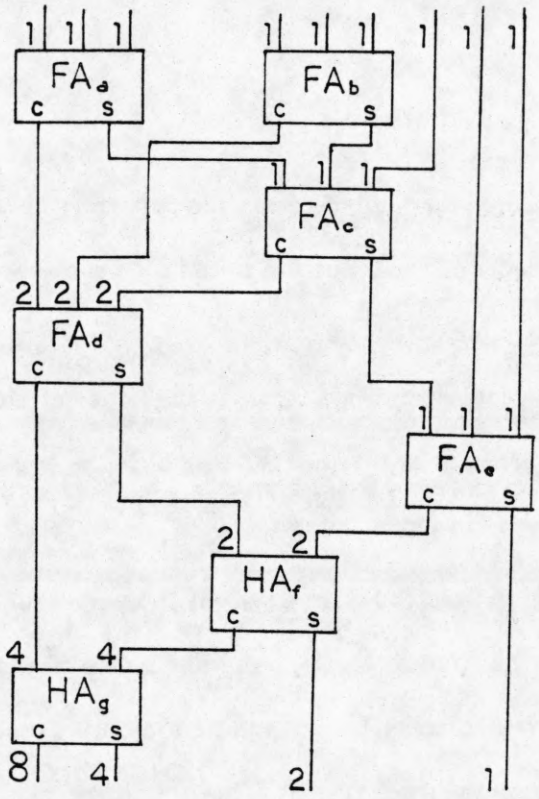

Figure 2: 9 input unary counter

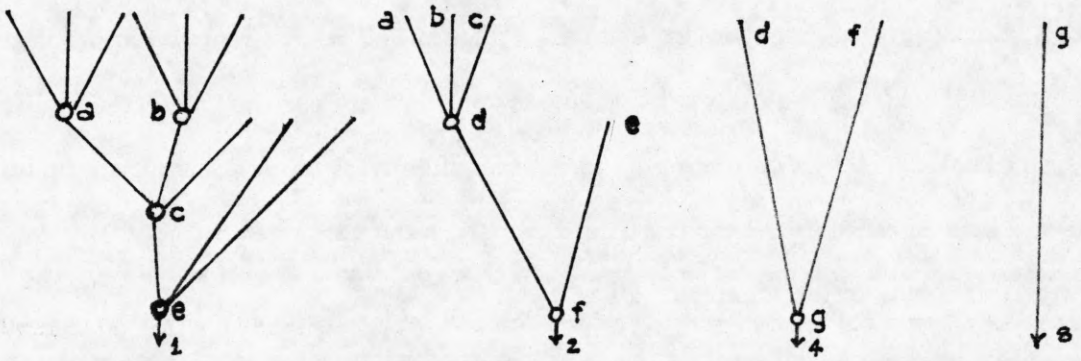

Figure 3: Tree representation for 9 input counter

3.2. Some Properties of Unary Counters

A subset of the class of unary counter circuits can be constructed with full adders only. In our design methodology, the minimum number of half adders necessary to construct a counter are used.

DEFINITION 4: A counter that can be constructed without any half adders is defined to be *complete*. Any other counter is an *incomplete* counter.

THEOREM 1: All unary counters of input size $N = 2^n - 1$, $n \geq 2$, are complete. All other unary counters are not complete.

COROLLARY 1: If the bit of weight 2^i in the binary representation of N is 0 then the bit slice of weight 2^i of the counter must contain at least 1 half adder.

PROOF: Consider a reduction tree R_I in which every node correspond to a full adder. Define a simple graph G, with the vertex set $\{\{V_{input}\},\{V_{internal}\},v_0\}$, as in Figure 4, where $\{V_{input}\}$ is a set of vertices that represents the inputs to the reduction tree, $\{V_{internal}\}$ is the set of adder nodes and v_0 is a vertex representing the output of the reduction tree. The edges of G are the same as that of the reduction tree.

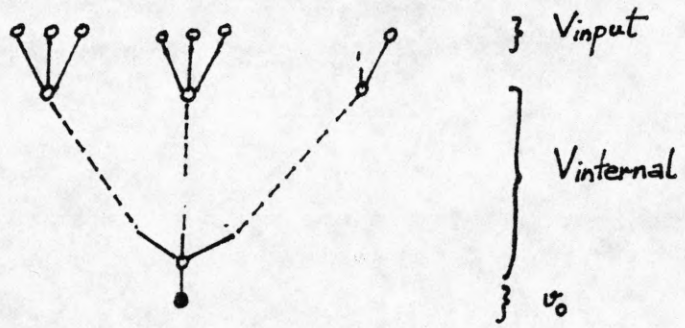

Figure 4: Graph G for reduction tree

Let α be the number of vertices in the set $\{V_{input}\}$ and let β be the number of vertices in the set $\{V_{internal}\}$. If d_{v_i} is the degree of the vertex i, ϵ is the number of edges and n the number of vertices in G, then from a result in graph theory [12]:

$$\sum_{i=0}^{i=n} d_{v_i} = 2\epsilon \tag{1}$$

Applying (1) to G, we get $4\beta + \alpha + 1 = 2(3\beta + 1)$ or

$$\alpha = 2\beta + 1 \tag{2}$$

Now consider that the root node of the reduction tree corresponds to a half adder. Applying (1) to G, we get $4(\beta - 1) + 3 + \alpha + 1 = 2(3(\beta - 1) + 2 + 1)$ or

$$\alpha = 2\beta \tag{3}$$

There are two cases:

Case 1: Remainder of $\frac{\alpha}{2}$ is 1

In this case there are $\left\lfloor \frac{\alpha}{2} \right\rfloor$ carries into the next bit slice and no half adders are necessary to construct the reduction tree R_I.

Case 2: Remainder of $\frac{\alpha}{2}$ is 0

In this case there are $\left\lfloor \frac{\alpha}{2} \right\rfloor$ carries into the next bit slice and a minimum of one half adder is necessary to construct the reduction tree. This half adder is placed at the root node of R_I.

If R_{n_i} and $R_{n_{i+1}}$ are reduction trees for bit slices i and i+1 respectively, then from cases 1 and 2:

$$n_{i+1} = \left\lfloor \frac{n_i}{2} \right\rfloor \tag{4}$$

Let $n_0 = N$ initially. The set of remainders obtained from the division $\frac{n_i}{2}$ in the recursive computation of n_{i+1} from n_i, represents the binary encoding of N. A 0 in the bit position of weight 2^k, $k \geqslant 0$, in this binary encoding implies that the remainder of $\frac{n_k}{2}$ is 0. By case 2, the root node of R_{n_k} must be a half adder. Hence the binary encoding of N must not contain any 0's. Since this implies that the binary encoding of N must be of the form 1111.....1, hence for a complete counter, N must be of the form $2^n - 1$, for $n \geqslant 2$ (n=1 is trivial).

Also, if N is not of the form $2^n - 1$, then it's binary encoding contains at least one 0 and the corresponding reduction tree contains a half adder. Hence it is not complete. □

3.2.1. Fault Observability in Counters

It can be easily shown that under the Single Cell Faulty Model (SCFM) [9], any error in the sum or carry outputs of an addition module will always cause an error in the observed counter

output. Consider an addition module whose inputs are of weight 2^i. An error in the sum output of this addition module causes the output of the i'th bit slice to be incorrect. This is because this output is the parity of it's inputs. An error in the carry output of the addition module causes an error in the output of some bit slice j, $j > i$. If both the sum and carry outputs of the addition module are incorrect, then the output of bit slice i is incorrect although there might not be any error in a bit slice j, $j > i$, due to fault masking. Hence an error in the outputs of an addition module is always propagated to one of the outputs of the counter.

4. An Empirical Approach to Test Generation for Complete Counters

In the following discussion, the counters referred to are unary counters. The following procedure constructs a $2^m - 1$ input counter from two counters of input size $2^{m-1} - 1$. The construction can also be used to prove by induction, that all counters of input size $2^m - 1$ can be constructed with full adders only.

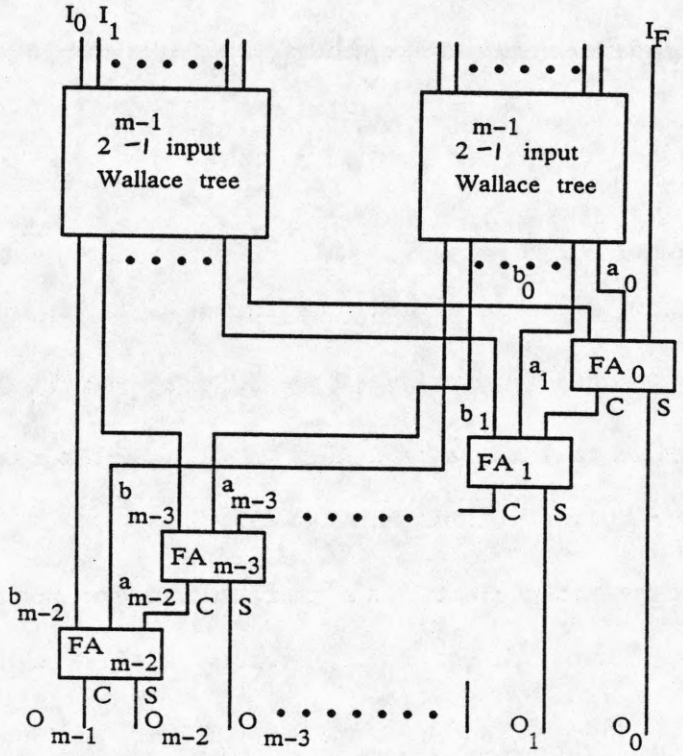

Figure 5: Recursive construction of unary counters

Construction 1: Consider the identity

$$2^m - 1 = 2^{m-1} - 1 + 2^{m-1} - 1 + 1$$

Hence, two 2^{m-1} input counters and an extra input can be used to construct a 2^m-1 input counter as in Figure 5. With regard to Figure 5, the blocks FA_i are full adders used to perform the summation of two output lines of equal weight from each of the 2^{m-1} input counters. Notice that the full adders FA_i are serially connected and that this serial chain is C-testable [9]. It is easy to see that any path in the above circuit consists of a string of serially connected full adders. This serial chain is through either the sum or carry output of the full adder and is C-testable.

In the following, conditions are derived for the C-test to be applied to the chain of full adders FA_i, in Figure 5. In Figure 5, the vectors A = $a_0\, a_1 ... a_{m-1}$ and B = $b_0\, b_1 ... b_{m-1}$ are the outputs of the two 2^{m-1} input counters. Input IF is the only directly controllable input line for the full adder chain FA_i. The objective is to establish the patterns of vectors A and B, such that all the full adders in the chain are exhaustively tested by controlling IF for each such combination of patterns.

Let C and D be two arbitrary vectors of length m. It can be verified that the set of vectors A and B in Table 1, is a C-test of cycle length 8 for the above chain of full adders. Call this test set the *C8* test set (an acronym for constant 8). The set of vectors applied at either $a_0\, a_1 ... a_{m-1}$ or $b_0\, b_1 ... b_{m-1}$ in applying the C8 test set are the same and are referred to as the C8 input set. The vectors listed in either of the columns A or B of Table 1 constitute the C8 input set.

In Table 1, C and D can be chosen to be the same. Tests 1-4 apply all possible vectors with $a_i = b_i$ and the rest apply all possible vectors with $a_i \neq b_i$.

DEFINITION 5: Define the *characteristics* of the the *C8* input set of length 8 by the following properties:

Property 1: The test set must contain the vectors 000..0,111..1,0101.... and 1010.....

Property 2: For every test vector A, different from those identified in *Property 1*, the complemented vector $\bar{A}$ must also appear in the test set.

Table 1.

Test	A	B	IF
Test 1	0000..0	0000..0	0
Test 2	1111..1	1111..1	1
Test 3	0101...	0101...	1
Test 4	1010...	1010...	0
Test 5	C	$\bar{C}$	0
Test 6	$\bar{C}$	C	0
Test 7	D	$\bar{D}$	1
Test 8	$\bar{D}$	D	1

Let O(1),O(2),....,O(8) represent the vectors obtained at the output terminals of the full adder chain $FA_0,FA_1,...,FA_{m-1}$ in Figure 5. The vector O(i) is the vector $o_0,o_1,...,o_m$ obtained in applying the i'th test to the inputs of the full adder chain.

LEMMA 3: In applying the *C8* test set to the full adder chain of Construction 1, the set of vectors {O(1),O(2),....,O(8)} retain the characteristics of the *C8* input set.

PROOF: Consider the *C8* test set and the output vector O(i) = $o_0\, o_1...o_m$ for the i'th test, listed for the tests 1-8 of Table 1 in Table 2. The output O(1) = 0000..0 when test 1 is applied and the output O(2) = 1111..1 when test 2 is applied. The application of test 3 gives O(3) = 1010... and the application of test 4 gives O(4) = 0101... . Hence property 1 is satisfied.

Table 2.

Test	Vector O(i)
Test 1	0000..0
Test 2	1111..1
Test 3	1010...
Test 4	0101...
Test 5	1111..0
Test 6	0000..0
Test 7	0000..1
Test 8	0000..1

For each of the tests 5-8, either $a_i = 1$ and $b_i = 0$ or $a_i = 0$ and $b_i = 1$, for FA_i, where $0 \leqslant i \leqslant m-1$. If IF = 0 as in test 5 and test 6, then every full adder in the full adder chain has one of it's inputs set to 1. Hence all the sum bits for all the FA_i are 1, except for the most significant output bit which is the carry from FA_{m-1}. If IF = 1, then it can be seen that all the sum and carry bits of the above full adders are complemented. Hence property 2 is satisfied.

Since both properties 1 and 2 are satisfied, the set {O(0),O(1),....,O(m)} has the characteristics of the *C8* input set. □

LEMMA 4: In applying all possible input vectors to a full adder, the set of output vectors obtained (C_{out} , S) satisfy the characteristic properties of the *C8* input set.

Lemma 4 is easily verified. Hence, by Lemma 3 and Construction 1, the test set for the 7-input counter of Figure 1.1 is derived in Table 3. The vector A represents the values $a_0\, a_1$. Similarly for the vector B.

THEOREM 2: Unary counters of input size 2^n -1 based on Construction 1, where $n \geqslant 2$, are C-testable with a minimal C-test cycle of length 8.

PROOF: (By induction) We have already seen that 3 and 7 input counters can be tested with a C-test cycle of length 8.

Table 3.

Test	A1	B1	C1	A2	B2	C2	A	B	IF	O0	O1	O2
Test1	0	0	0	0	0	0	00	00	0	0	0	0
Test2	1	1	1	1	1	1	11	11	1	1	1	1
Test3	0	1	1	0	1	1	01	01	1	1	0	1
Test4	1	0	0	1	0	0	10	10	0	0	1	0
Test5	0	0	1	1	1	0	10	01	0	1	1	0
Test6	1	1	0	0	0	1	01	10	0	1	1	0
Test7	0	1	0	1	0	1	10	01	1	0	0	1
Test8	1	0	1	0	1	0	01	10	1	0	0	1

Assume that the application of a set of tests T to the inputs of a $2^{m-1}-1$ input counter results in a set of output vectors O(1)-O(8), which have the characteristics of the C8 input set. Then a combination of test vectors from the set T can be applied to the inputs of each of the two $2^{m-1}-1$ input counters of construction 1, such that the C8 test set is applied to the corresponding full adder chain. By Lemma 3, the set of output vectors O(1)-O(8) of the 2^m-1 input counter retains the characteristics of the C8 input set. Hence, by the induction hypothesis, a test set can always be constructed for any complete unary counter based on construction 1. □

The above theorem also defines a method for deriving the test set for any complete unary counter based on construction 1.

5. Test Generation Based on the Tree Model

Test generation for unary counters is simplified by virtue of the iterative structure of the trees concerned. Consider the reduction trees of each bit slice individually, with the inputs to the tree considered as externally controllable inputs.

LEMMA 5: Every reduction tree in the set of reduction trees that represent a counter, is individually C-testable.

PROOF: Consider the vectors $V_1 = 00001111$, $V_2 = 00110011$, $V_3 = 01010101$ and $V_4 = 01101001$. It can be verified that $V_i \oplus V_j \oplus V_k = V_l$, where $i \neq j \neq k \neq l$ and $1 \leqslant i, j, k, l \leqslant 4$. Also, any 3 of these vectors contain all possible combinations of 3 bit values.

The root node of a tree can be labelled such that it's 3 or 2 input branches and it's output branch have different labels, according to whether it represents a full or half adder respectively. Since every node has 4 incident branches by construction, the labelling can be recursively applied to all the nodes whose output branches have been assigned labels.[1] □

In the above procedure, if the label V_i is associated with the input i, V_j with the input j, upto V_n with input n, then the i'th test vector is obtained by applying the i'th element of V_i to i, the

[1]The proof is an extension of that presented in [7] for 2 input EX-OR trees.

i'th element of V_j to j and so on, upto the i'th element of V_n to n. Since the length of each of the vectors V_i is 8, $1 \leqslant i \leqslant 4$, 8 tests are obtained.

Lemma 5 is a necessary condition for the overall counter to be C-testable. It remains to be shown that the C-tests for each of the reduction trees taken together, can be applied simultaneously.

Consider the set of vectors $V=\{V_1,V_2,V_3,V_4\}$, where each of the vectors V_i are the same as defined in Lemma 5. Further, let $C=\{C_1,C_2,C_3,C_4\}$ be a set of vectors where each of the vectors C_i are defined as follows: $C_1 = 01110001$, $C_2 = 01001101$, $C_3 = 00101011$, $C_4 = 00010111$. Let V_i be the i'th vector in the set V where $1 \leqslant i \leqslant 4$ and define C_i similarly for the set C. Also, define the functions g( ) and f( ) as follows:

(1) $g(V_i,V_j,V_k) = V_i \oplus V_j \oplus V_k$

(2) $f(V_i,V_j,V_k) = V_i.V_j + V_i.V_k + V_j.V_k$

The function g( ) is the bitwise summation of the vectors V_i, V_j and V_k, while the function f( ) is the bitwise carry produced in the above summation.

The following relations can be verified:

(1) $g(V_i,V_j,V_k) = V_l$

(2) $f(V_i,V_j,V_k) = C_l$

(3) $g(C_i,C_j,C_k) = C_l$

(4) $f(C_i,C_j,C_k) = V_l$

(5) $g(V_i,V_j,\overline{V}_k) = \overline{V}_l$

(6) $f(V_i,V_j,\overline{V}_k) = C_k$

(7) $g(V_i,\overline{V}_j,\overline{V}_k) = V_l$

(8) $f(V_i,\overline{V}_j,\overline{V}_k) = \overline{C}_i$

(9) $g(\nabla_i, \nabla_j, \nabla_k) = \nabla_l$

(10) $f(\nabla_i, \nabla_j, \nabla_k) = \bar{C}_l$

where $i \neq j \neq k \neq l$ and $1 \leq i, j, k, l \leq 4$, in each of the above 10 equations.

The above equations which define the manner in which the sum and carry bits are generated at each addition module, satisfy the following property:

Property 1: Each of the equations 1-10 can be written with the 'V' and 'C' variables interchanged.

The sets of vectors V and C are seen to satisfy the following property:

Property 2: Any combination of 3 vectors V_i, V_j, V_k or C_i, C_j, C_k, $i \neq j \neq k \neq l$ and $1 \leq i, j, k, l \leq 4$ and combinations of complemented or uncomplemented sets of the same, contain all possible combinations of 3 bit values.

Branch Labelling Problem: The problem of finding a minimum test set of length 8 reduces to the problem of labelling each of the branches of the tree representation of the given counter with the labels V_1-V_4,∇_1-∇_4 or C_1-C_4,$\bar{C}_1$-$\bar{C}_4$. By property 2, the indexes 'i' of the vectors V_i, ∇_i or $C_i, \bar{C}_i$, applied at each of the inputs to a node must be different from each other. Further, the index j of the output branch of that node is uniquely determined by the equations 1-10. A *consistent* labelling for a tree is one that is determined by the above conditions. In considering the labelling for a set of reduction trees, the labelling for the fundamental tree determines the labels on the inputs to it's derived subtree. This set of input labels must define a consistent labelling for the derived subtree. In the same manner, a labelling for any tree in the set of reduction trees for a counter must define a consistent labelling on it's derived subtree. Such a labelling is a *consistent* labelling for the entire set of reduction trees.

By Property 1, the fundamental tree is labelled with V's, it's derived subtree with C's, the next derived subtree with V's and so on in an alternating manner.

DEFINITION 6: A *simple* labelling set consists of only the uncomplemented labels V_1-V_4 and C_1-C_4. Any other labelling set is *complex*.

For simple labelling only equations 1-4 are relevant and the label on the carry from an addition module is the same as the label on the output branch of the node representing that module.

Consider the reduction tree set for a complete unary counter. The reduction tree set for a 3 input counter is simply $\{ R_3 \}$. The carry from R_3 is assumed to be implicit in R_3. The reduction tree set for a 7 input counter is $\{ R_7,R_3\}$. For a counter of input size 2^n-1, $n \geqslant 2$, the reduction tree set is $\{R_{2^n-1},R_{2^{n-1}-1},\ldots R_{2^2-1}\}$.

Let us associate the names *a,b,c,d,* with the branches incident on a full adder node of a reduction tree in the following manner. The output branch of a node is called the *a* branch of the node and the 3 input branches are called the *b,c* and *d* branches of the node in a clockwise manner (Figure 6).

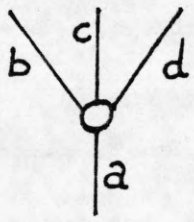

Figure 6: Branch names for a node

Consider the reduction tree R_j. the *external* branches of R_j are those which are inputs to R_j and the *internal* branches of R_j are those which join two nodes of R_j. *External nodes* of R_j are those which have one or more inputs that are external branches of R_j. The rest are *internal* nodes. Further, *external* and *internal* labels are those obtained on the external and internal branches of R_j, by a consistent labelling in R_j. The unordered set of internal and external labels with repetition, form the *internal* and *external* labelling sets of R_j respectively.

5.1. Tree Growing

Consider the full adder node of the reduction tree R_3 of a 3 input counter. The reduction tree R_7 can be constructed by connecting the outputs of 2 full adder nodes to the b and c input branches of the full adder node of R_3. Similarly, the reduction tree R_{15} can be constructed by connecting the output of a full adder node to every input of R_7 that is the b or c input branch of an external node of R_7. Such a process of replicating a reduction tree R_{k_1} and adding full adder nodes to the inputs of R_{k_1} to construct a reduction tree R_{k_2}, where $k_2 > k_1$, is called *tree growing*. tree R_{k_2} is said to have been grown from R_{k_1}. Beginning with R_3, a set of reduction trees is grown recursively, at each step growing R_{k_2} from R_{k_1}, $k_2 > k_1$.

Case 1: Consider $k_1 = 2^n - 1$ for some $n \geq 3$. Then the reduction tree R_{k_2}, $k_2 = 2^{n+1} - 1$ is obtained from R_{k_1} by adding a full adder node to every input of R_{k_1} that is the b or c input branch of an external node of R_{k_1}. If the largest reduction tree generated by the above process is $R_{i_{max}}$, where $i_{max} = 2^j - 1$, for some $j \geq 3$, then the set of reduction trees obtained, represent that for a unary counter of input size $2^j - 1$

EXAMPLE 2: The reduction tree R_7 for the 7 input counter of Figure 1 can be obtained by tree growing as above, from R_3.

Case 2: Consider the reduction tree R_{k_1} where k_1 is odd. Further, let every node of R_{k_1} satisfy the property that if it's b input branch is connected to the output of a full adder node, so is it's c input branch. This is a necessary condition for the tree growing method of case 2 to be applied. Let $k_2 = 2k_1 + 1$ and we are to grow R_{k_2} from R_{k_1} (i.e there exists 1 full adder in R_{k_2} for every input of R_{k_1}). Since k_2 is odd, an even number of full adder nodes must be added to the inputs of R_{k_1}. Hence $(\frac{k_2 - k_1}{2})$ pairs of full adder nodes are added to pairs b and c of input branches of external nodes of R_{k_1} to generate R_{k_2}. The reduction tree R_{k_2} meets the necessary condition for case 2 for tree growing as described earlier. Hence trees of larger size can be grown by applying the process recursively to R_{k_2}.

EXAMPLE 3: Figure 7 shows how R_{23} is grown from R_{11}.

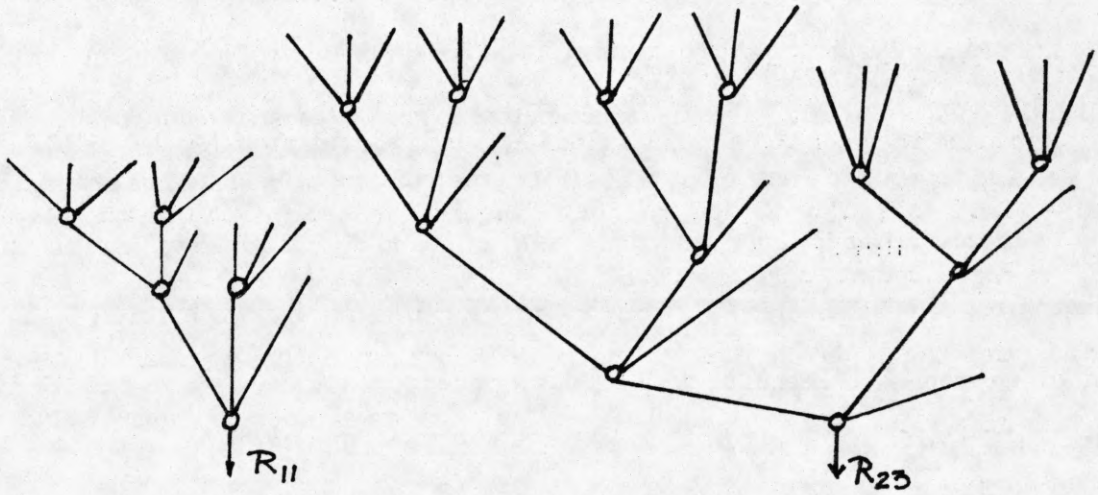

Figure 7: Tree growing for case 2

Tree growing as in case 1 is a special case of tree growing as in case 2. In the following discussion, *tree growing* refers to that of case 2 and the corresponding definitions of R_{k_1} and R_{k_2} are retained.

DEFINITION 7: Define a *sequence of branches* from the root node to a node n of a reduction tree, as the sequence b,c of branches, (ignoring the labels on the output branches of nodes) that must be followed from the root node to reach that node.

EXAMPLE 4: In Figure 7a, the sequence {bc} of branches must be followed to get to the node S from the root node N of R_{15}.

Let $M(n) \rightarrow n'$ be a partial mapping that maps a node n of the reduction tree R_{k_2} to a node n' of it's derived subtree R_{k_1}, where R_{k_2} is grown from R_{k_1}.

DEFINITION 8: M(n) → n' if it is possible to reach n' from the root node of R_{k_1} by following the same sequence of branches that it takes to reach n from the root node of R_{k_2}. M(n) → NULL otherwise.

EXAMPLE 5: In Figure 7a, M(P) → M .

DEFINITION 9: Define an operation R( ) such that m = R(n), if the output branch of a node n in a reduction tree is connected to one of the inputs of the node m in the same tree.

EXAMPLE 6: In Figure 7a, N = R(P) in R_{15}.

By the manner in which R_{k_2} is grown from R_{k_1}, it is clear that if M(n) → NULL, then $M(R(n)) \neq NULL$. The following construction defines the manner in which the carries from the reduction tree R_{k_2} are connected to the inputs of R_{k_1}.

Construction 2: Let n be a full adder node in the reduction tree R_{k_2}, whose carry is to be connected to the inputs of the reduction tree R_{k_1}. There are 2 cases.

Case 1: M(n) → NULL.

In this case if the output branch of n is connected to the b or c input branch of R(n), then the carry from n is connected to the c or b input branch of n' respectively, where M(n) → n'.

Case 2: M(n) → n', where n' is a node in R_{k_1}.

In this case, the carry from n is connected to the d input branch of n'. □

5.2. Partitioned Labelling

There are many ways of labelling a set of reduction trees for a counter. In the following, we discuss a labelling scheme that is later shown to be consistent with Constructions 1 and 2. The labels V and C are dropped for convenience and only the subscripts {1,2,3,4} are retained.

Partition the labelling set {1,2,3,4} into 2 partitions L_1={2,3} and L_2={1,4}. Partition the set of branches {a,b,c,d} incident on each full adder node into 2 partitions, B_1={b,c} and B_2={a,d}.

DEFINITION 10: A *partitioned* labelling is defined as one in which the labels in the set L_i, i=1,2, are applied either to the branches in B_1 or the branches in B_2, but not both.

EXAMPLE 7: Figure 8 depicts a partitioned labelling of the set of reduction trees for a 15 input unary counter.

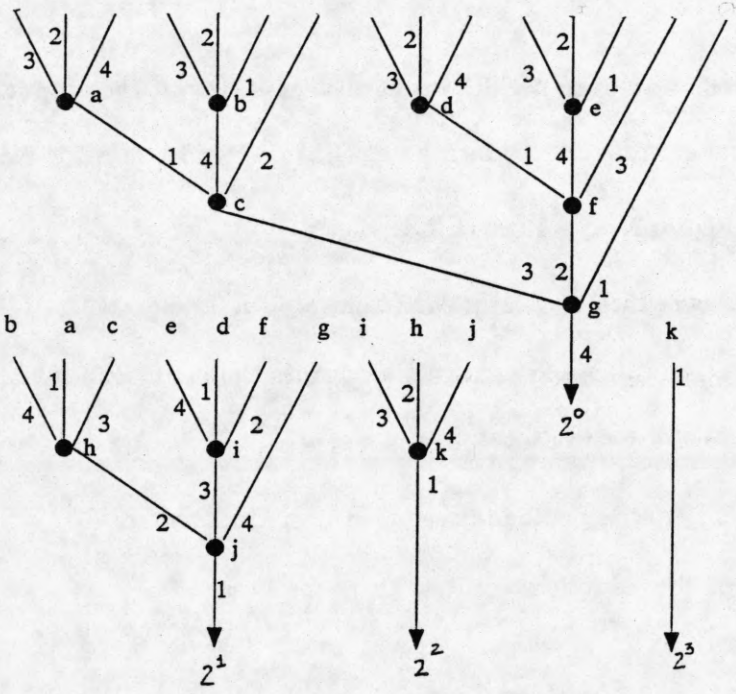

Figure 8: Partitioned labelling for 15 input unary counter

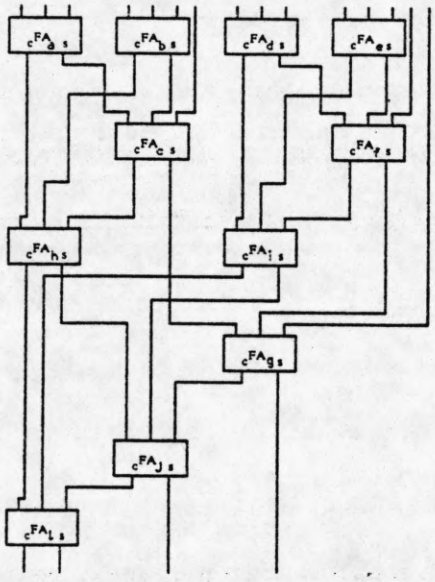

Figure 9: 15 input unary counter

Let SWITCH( ) be an operator on the sets L_i, i=1,2, such that SWITCH(L_i), exchanges the labels in the specified set L_i, on all the branches of a reduction tree. Consider a partitioned labelling of R_{k_2}, where R_{k_2} is grown from R_{k_1}. The labels on the branches a,b,c and d of a node n of R_{k_2} are duplicated on the node n' of R_{k_1} if M(n) → n' and $n' \neq NULL$. This process is repeated for every

node n of R_{k_2}. The resultant labelling of the nodes of R_{k_2} is called the *duplicated labelling* of R_{k_2} on R_{k_1}.

Let R_{k_1} and R_{k_2} be defined as in the tree growing method of case 2. Further, let the carries between these two trees be interconnected as in construction 2.

THEOREM 3: A consistent labelling between R_{k_2} and R_{k_1} is obtained by applying the operators SWITCH(L_1) and SWITCH(L_2) to the duplicated (partitioned) labelling of R_{k_2} on R_{k_1}.

COROLLARY 2: A consistent labelling between R_{k_2} and R_{k_1} defined as in the tree growing method of case 1, can be similarly found.

PROOF: For every node n in R_{k_2}, either M(n) = NULL or $M(n) \neq NULL$.

Case 1: M(n) = NULL.

From Construction 1, the output branch of n is connected to either the b or c input branch of R(n). Let it be connected to the b input branch and let the label on this branch be i. Then for simple labelling, the label on the carry signal from n is also i. By Construction 2, this carry is connected to the c input branch of n', where M(n) → n'. By the SWITCH operation on the duplicated labelling of R_{k_2} on R_{k_1}, the label on the carry from n and the label on the c input branch of n' is i. Hence the labelling is consistent.

Case 2: $M(n) \neq NULL$

Consider the node n' of R_{k_1}, where M(n) → n'. By Construction 1, the input branch d of n' is an input to R_{k_1}. By the SWITCH operation on the duplicated partitioned labelling of R_{k_2} on R_{k_1}, the label i of the output branch of n is the label of the input branch d of n'. Since i is also the label of the carry from node n and by Construction 2, the carry from n is connected to the d input branch of n', the labelling is consistent.

Since the labelling of each of the reduction trees R_{k_2} and R_{k_1} is consistent and the carries between the trees are consistent with this labelling, the labelling for both the trees is consistent.

□

In the above, the branches b,c and d of every node are permutable. However, the naming convention is adopted for the sake of formalism.

Consider the reduction tree set RTS = $\{R_{2^n-1}, R_{2^{n-1}-1}, \ldots R_3\}$ obtained for a 2^n-1 input unary counter by tree growing as in case 1. The interconnection of the carries between the trees is based on construction 2.

THEOREM 4: A consistent labelling of the reduction tree set RTS is obtained by a partitioned labelling on the fundamental tree of the counter.

COROLLARY 3: A simple labelling set is sufficient to consistently label the reduction tree set of complete unary counters based on Constructions 1 and 2 and defines a C-test of length 8.

PROOF: By corollary 2, a consistent labelling between the fundamental tree R_{2^n-1} and $R_{2^{n-1}-1}$ can be found.

Now consider the set of reduction trees $\{R_{a_0}, R_{a_1}, R_{a_2}, \ldots, R_{a_i}\}$, where $a_i = 2^{n-i}-1$. If a consistent labelling for the above set for $i < n$ exists, then a consistent labelling for the set $\{R_{a_0}, \ldots, R_{a_{i+1}}\}$ can be found by applying corollary 2 to find a consistent labelling between R_{a_i} and it's derived subtree $R_{a_{i+1}}$. By the induction hypothesis then, a consistent labelling for the complete set of reduction trees can always be found for any n. This labelling defines 8 tests for the counter.

□

EXAMPLE 8: The partitioned labelling of Figure 8 defines 8 tests for the corresponding 15 input counter of Figure 9.

5.3. Test Generation for Incomplete Counters

A method for finding which bit slice of a counter must have a half adder is described in the proof of theorem 1. These half adders are placed at the root nodes of the corresponding reduction trees. In the worst case, all but one bit slice can have half adders at the above root nodes. This

worst case ripple carry chain of half adders considered as a sub circuit of the counter, can be tested in O(m) tests for m half adders in the circuit. Since there are $O(\log_2 n)$ half adders for an n input counter, these can be tested in $O(\log_2 n)$ tests. By our earlier hypothesis, if the full adders in the circuit can be tested in 8 tests, then it should be possible to find an $O(\log_2 n)$ test for an n input incomplete counter in general. In the following, we discuss how full adder trees that can be always tested in 8 tests are constructed. A design for testability procedure is also proposed that substitutes each half adder at the root node of a reduction tree by a full adder and by the addition of some extra circuitry, makes it possible to test all incomplete counters in at most 9 tests.

5.4. Generalized Counters

Generalized counters perform the summation of sets of input bits ,each set corresponding to a different weight. *Complete* and *incomplete* generalized counters are defined in a manner similar to unary counters.

DEFINITION 11: A $C(i_k, i_{k-1},...,i_1,i_0;\text{count})$ weighted counter is defined to be one that adds i_0 bits of weight 2^0 to i_1 bits of weight 2^1, and so on upto i_k bits of weight 2^k, where $k \geq 0$ and *count* is the number of output bits produced.

THEOREM 5: All generalized counters of input size N, where

$$N = \sum_{i=0}^{i=k} i_k 2^k = 2^n - 1 \qquad (1)$$

where $n \geq 2$ are complete.

COROLLARY 4: If the bit of weight 2^i in the binary representation of N is 0, then the bit slice of weight 2^i of the counter must contain at least 1 half adder.

PROOF: The addition of i_0 bits of weight 2^0 to i_1 bits of weight 2^1 and so on, upto i_k bits of weight 2^k is equivalent to the summation of N bits of weight 2^0 in the following manner. If a bit of weight 2^j ,$0 \leq j \leq k$, is 0 or 1, then it is equivalent to setting 2^j bits of weight 2^0 to 0 or 1 respectively. Hence the argument of theorem 1 can be applied to the N bits of weight 2^0. The proof then follows.

□

For the generalized counter $C(i_k, i_{k-1}, \ldots i_1, i_0; \text{count})$, the number of input bits to each bit slice is computed from the recurrence

$$c_{j+1} = \left\lfloor \frac{c_j + i_j}{2} \right\rfloor \tag{2}$$

where $c_j = 0$ for j=0. The number of input bits to bit slice j is $N_j = c_j + i_j$ and c_j is the number of carries into bit slice j from bit slice j-1.

5.4.1. Restricted Tree Growing with Partitioned Labelling

There exists a subset of the class of generalized counters on which the partitioned labelling scheme can be used effectively. Consider the reduction tree $R_{N_{j+1}}$, where $N_{j+1} = c_{j+1} + i_{j+1}$ as before. If we impose the restriction that i_{j+1} must be even, then $\frac{i_{j+1}}{2}$ pairs of external inputs to $R_{N_{j+1}}$ can be formed. Pairs {b,c} of input branches of external nodes of $R_{N_{j+1}}$ can then be selected to which these pairs of external inputs can be connected. If the d input branch of a node n is connected to an external input, then the d input branch of the node n', where R(n)=R(n'), must also be similarly connected.

In order to grow R_{N_j} from $R_{N_{j+1}}$, full adder nodes are added to the inputs of the replicated tree R'_{N_j} of R_{N_j}, as in the tree growing of case 2. However, full adder nodes are not added to those inputs of R'_{N_j} to which inputs i_{j+1} are connected in the corresponding tree R_{N_j}. The growing of R_{N_j} is said to be based on the input restrictions of $R_{N_{j+1}}$.

It should be mentioned that the number of external inputs i_k to the derived subtree of the fundamental tree need not be even, since one does not need to apply the process recursively to the fundamental tree.

5.4.2. Design Methodology 1 for Generalized Counters

(1) Compute the number of input bits N_j to each bit slice j of the generalized counter.

(2) Construct the reduction tree R_{N_j} for each bit slice j, by tree growing based on the input restrictions of $R_{N_{j+1}}$ recursively, beginning with R_3.

(3) Interconnect the carries between the trees as in Construction 2.

The external inputs of each of the reduction trees that are not connected to the carries from full adder nodes of other reduction trees are the inputs to the generalized counter.

5.4.3. Test Generation for Counters Based on Design Methodology 1

The testing procedure is an extension of that for unary counters. Consider the mapping M(n) $\rightarrow$ n' between the nodes n of R_{N_j} and the nodes n' of $R_{N_{j+1}}$. The set of nodes of $R_{N_{j+1}}$ obtained by performing the above mapping for every node n of R_{N_j} defines a subtree ST_{j+1} of $R_{N_{j+1}}$. By theorem 3, a consistent labelling between R_{N_j} and ST_{j+1} is obtained by a partitioned labelling of R_{N_j}. The resultant labelling of ST_{j+1} is then extended to $R_{N_{j+1}}$ to obtain the labels on the input branches to $R_{n_{j+1}}$ that are external inputs to the counter.

By the above process, starting with the fundamental tree, a consistent labelling can be obtained for all pairs of bit slices recursively. The set of labels obtained at the inputs to the counter define a test set of length 8.

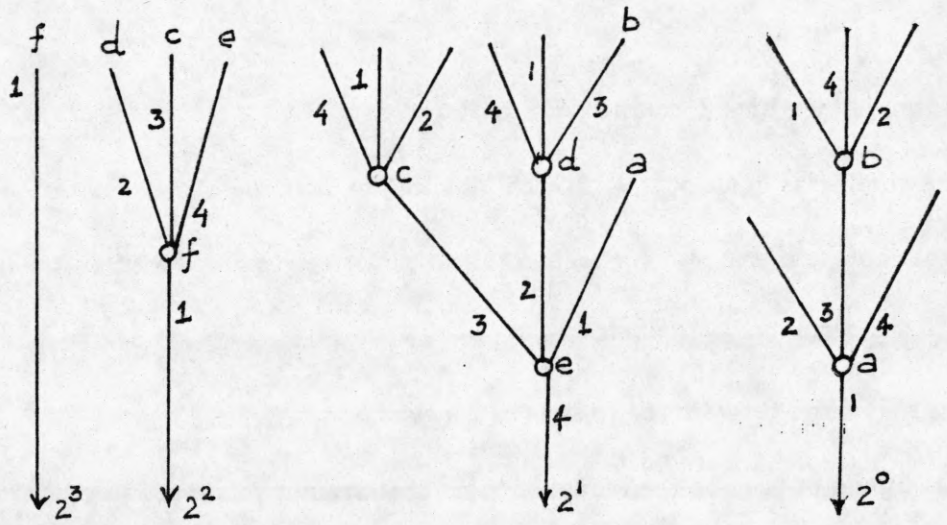

Figure 10: Labelling for C(5,5:4) counter

EXAMPLE 9: Figure 10 depicts the reduction tree set and labelling obtained by tree growing, for C(5,5:4). Figure 11 depicts it's corresponding hardware implementation.

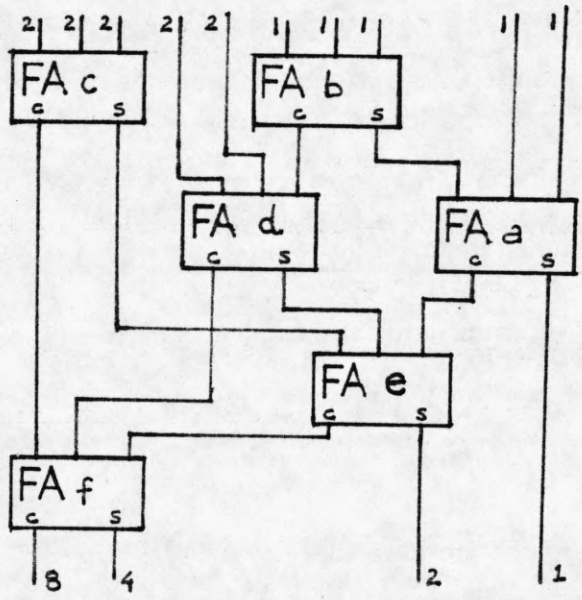

Figure 11: C(5,5:4) schematic

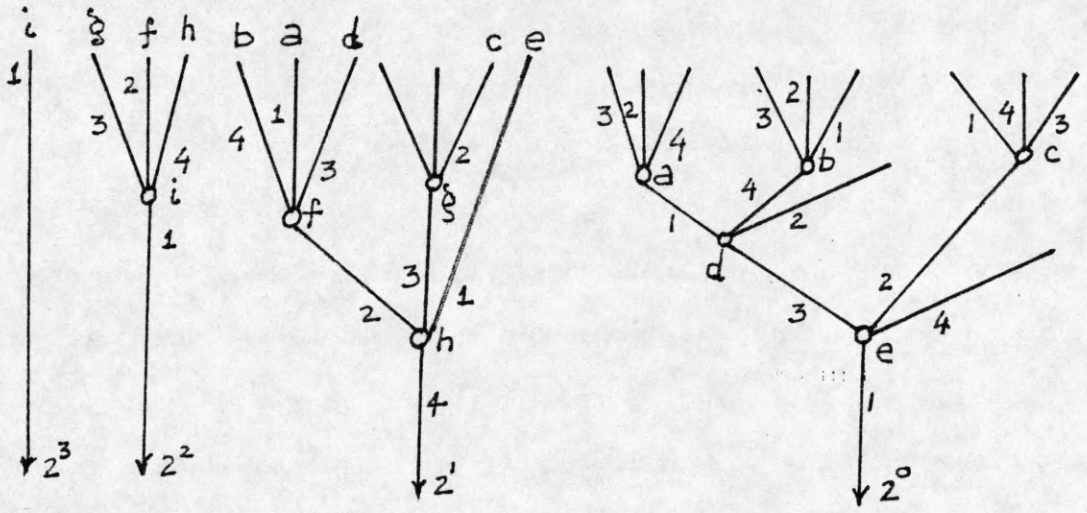

Figure 12: Reduction tree set and labelling for C(2,11:4)

EXAMPLE 10: Figure 12 depicts the reduction tree set and labelling for C(2,11:4).

5.5. Unrestricted Tree Growing

In unrestricted tree growing, the structure of the the reduction trees of each bit slice and the way they are interconnected, is determined by the manner in which the trees are grown.

Consider a consistent labelling on R_j. Let E be the external labelling set of R_j, every node of which represents a full adder. By equation 1, j must be odd. The labels belonging to the set E can be obtained from the carry signals from each of j nodes of a reduction tree R_k, if such a reduction tree can be found, where $k = 2j + 1$. Since k is also odd, all the nodes of such a reduction tree R_k. can be designed to be full adder nodes. Let I be the internal labelling set of R_k for a consistent labelling on R_k.

THEOREM 6: For a consistent labelling on every reduction tree R_j, there exists a consistent labelling on a reduction tree R_k, $k = 2j + 1$, such that the set E of labels of R_j is the same as the set I of labels for R_k.

PROOF: Let i,j,k,l be 4 labels such that $i \neq j \neq k \neq l$ and $1 \leqslant i, j, k, l \leqslant 4$. Every node of R_j has 4 distinct labels on each of it's 4 incident branches. This must also be true for R_k if the labelling is to be consistent.

The procedure used to construct R_k, is to grow it from it's root node, labelling it consistently at each step (This tree growing is different from that described earlier). Consider a partially grown tree R_t, with t inputs, where $3 \leqslant t \leqslant k$ and t is odd. Let the internal labelling set of R_t be I_t, where I_t is a proper subset of E. We pick a label l from the set $\{E - I_t\}$. If this label l is an external label for R_t, then a node n is added to the input of R_t with the label l as in Figure 13, to obtain R_{t+2}. R_{t+2} is grown from R_t. The internal labelling set I_{t+2} of R_{t+2} becomes $I_{t+2} = I_t + l$. We have to show that it is possible to grow R_{t+2} from R_t for all $3 \leqslant t \leqslant k$, such that if I_t is a subset of E, then I_{t+2} is also a subset of E and I_t is contained in I_{t+2}.

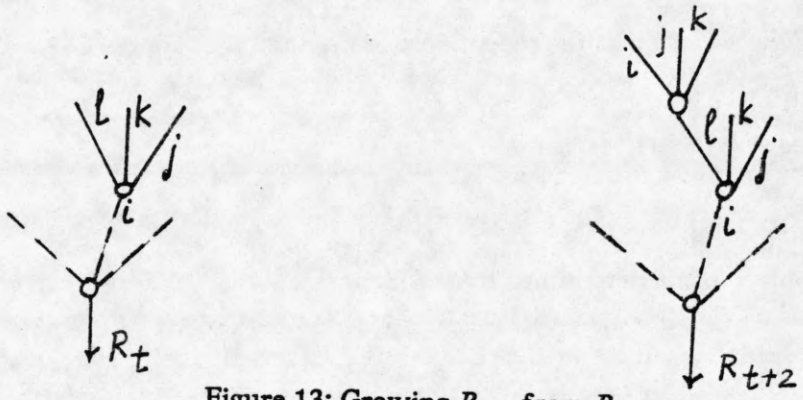

Figure 13: Growing R_{t+2} from R_t

Consider the reduction tree R_j. The external labels in the set E come from 3 types of external nodes of R_j as in Figure 14.

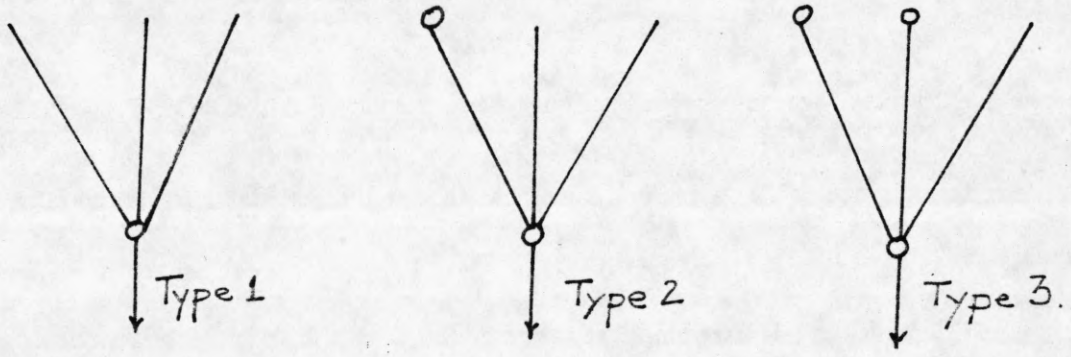

Figure 14: Types of external nodes

All 3 input branches to a node of type 1 are external branches. A node of type 2, has 1 internal and 2 input external branches. A type 3 node has 2 internal and 1 external input branch.

Assume that the label l picked from the set $\{E-I_t\}$ is also an external label for R_t. Then l must be the label of the output branch of the node n which is used to construct R_{t+2} from R_t.

It must also be the external label of a node of type 1,2 or 3 of R_j.

Case 1: 1 is the external label of a node n' of R_j of type 1.

In this case, there must be 2 other labels i,j where $i \neq j \neq l$, such that i,j and l are the labels on the 3 inputs to n'. The labels i,j must appear on the inputs to node n of R_k because l is the label on the output branch of n. The branches with the labels i,j of n are made the output branches of 2 nodes as in Figure 15a and these nodes are assigned consistent labels. The output of the node n is connected to the external branch of R_t with label l to yield R_{t+6}. The labels l,i,j are deleted from E and added to the set I. Hence I_{t+6} is a subset of E.

Case 2: 1 is the external label of a node n' of R_j of type 2.

In this case, there must be 1 other label i, where $i \neq l$, such that i,l are the labels on the 2 external inputs to n' of R_j. The label i must appear on the inputs to node n because l is the label on the output branch of n. The branch with the label i is made the output branch of a node which is labelled consistently as in Figure 15b. The labels i,l are deleted from E and added to I. The output of the node n is connected to the external branch of R_t with label l to yield R_{t+4}. Again, I_{t+4} is a subset of E.

Case 3: 1 is the external label of a node n' of R_j of type 3.

In this case, the output of the node n is connected to the external branch of R_t with label l to yield R_{t+2} as in Figure 15c. I_{t+2} is a subset of E.

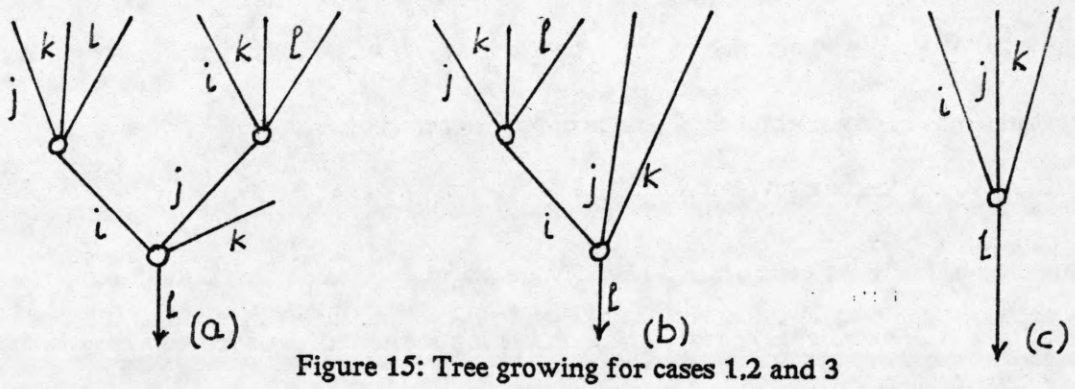

Figure 15: Tree growing for cases 1,2 and 3

Initially a label is picked from E and the construction of one of the cases 1,2 or 3 is applied to obtain R_7,R_5 or R_3 respectively. Any reduction tree must have at least one node *n1* of type 1. For any node of types 1 and 2 in R_j, there must exist at least 1 external label for that node which is the same as one of the external labels of *n1*. Hence the external labels of nodes of types 1 and 2 can always be made the internal labels of R_t, by applying the constructions of cases 1 and 2.

To complete the proof, we have to show that if a label in E comes from a node n' of R_j of type 3, it can always be made an internal label of R_k.

Consider a node n of type 2 in Figure 16a. Deletion of this node results in the configuration of Figure 16b. Delete all the nodes of R_j of type 2 as above.

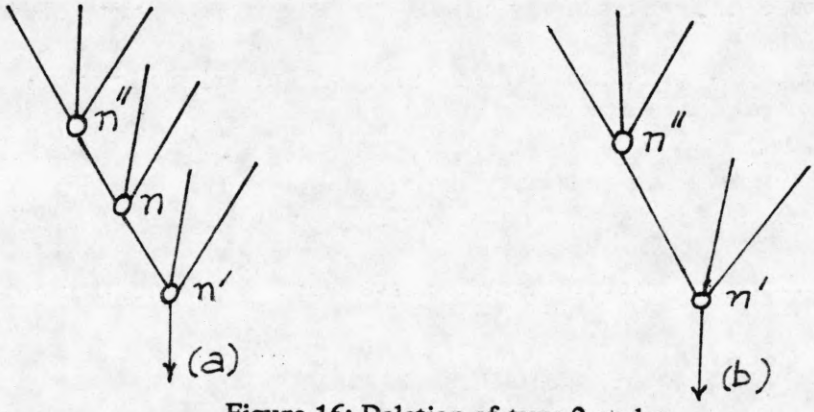

Figure 16: Deletion of type 2 nodes

The resultant tree $R_{j'}$ contains nodes of types 1 and 3 only. Consider that all but the external nodes of $R_{j'}$ are of type 1. Then from a property of binary trees [1], the number of nodes of type 3 is one more than the number of nodes of type 1. If the internal nodes of R_j are also of type 3, then

in general

Number of nodes of type 3 > *Number of nodes of type* 1 (1)

Hence, it is possible to select a label from E which corresponds to a node n' of R_j of type 1 and grow R_t based on the construction of case 1. This construction yields all the labels 1,2,3,4 as external labels of R_{t+6}. This process can be repeated alternately for cases 1 and 3. Now if there exists any node of type 3 in R_j, the corresponding label l can be made an internal label of R_t by applying the construction for case 3. By (1), the correctness of the procedure is guaranteed. □

5.6. The Design Methodology and Testing Strategy

Consider the generalized counter C($i_k, i_{k-1}, \ldots i_2, i_0$:count). The number of input bits to each bit slice is computed by the recurrence

$$c_{j+1} = \left\lfloor \frac{c_j + i_j}{2} \right\rfloor$$

where $c_j = 0$ for j=0 . The number of input bits to bit slice j is $N_j = c_j + i_j$.

5.6.1. Tree Growing for the Unrestricted Case

Since $N_{j+1} = c_{j+1} + i_{j+1}$, c_{j+1} of the external labels of $R_{N_{j+1}}$ must be internal labels of R_{N_j}. This is easily achieved, as the process of growing R_{N_j} as in theorem 6, consists of picking a label at a time from the set E of R_{N_j} and adding it to the set I_t of internal labels of the partially grown tree R_t. When c_{j+1} labels have been picked, $t = N_j$ and the tree R_{N_j} is designed (t as in theorem 6). The i_{j+1} labels remaining in the set E define i_{j+1} inputs to the reduction tree $R_{N_{j+1}}$, which are inputs to the counter. The growing of R_{N_j} is said to have been *based* on c_{j+1} labels of $R_{N_{j+1}}$.

5.6.2. Design Methodology 2 for Generalized Counters

5.6.2.1. Designing the Reduction Tree Set

(1) Compute the number of inputs N_j to each bit slice j of the counter.

(2) Construct and label the reduction tree R_3 for bit slice *k-1*, where bit slice *k* is the most significant bit slice of the counter (the carry from the R_3 tree of bit slice *k-1*).

(3) Let i=2.

Find the reduction tree of input size N_{k-i} for bit slice *k-i*, by tree growing based on the c_{j+1} labels of the subtree $R_{c_{k-i+1}}$ of the reduction tree $R_{N_{k-i+1}}$. If N_{k-i} is even, add an extra controllable input EI_{k-i} to bit slice *k-i* and grow the reduction tree $R_{N_{k-i}+1}$. Preferably, EI_{k-i} is made an external branch of the root node of $RN_{k-i}+1$. If N_{k-1} is even and the associated half adder corresponds to the root node of R_{k-1}, then an extra input is not added to bit slice *k-1*.

(4) Set $i=i+1$ and repeat from step 3 until all the reduction trees upto R_{i_0}, based on a consistent labelling have been found.

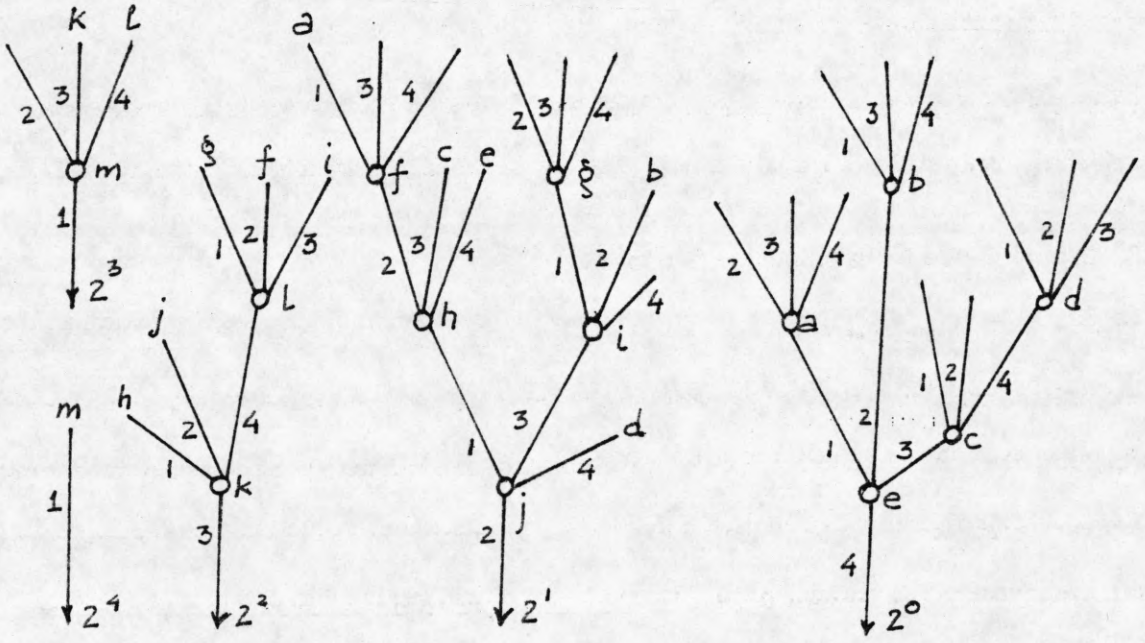

Figure 17: Reduction tree set and labelling for C(1,0,6,11:5)

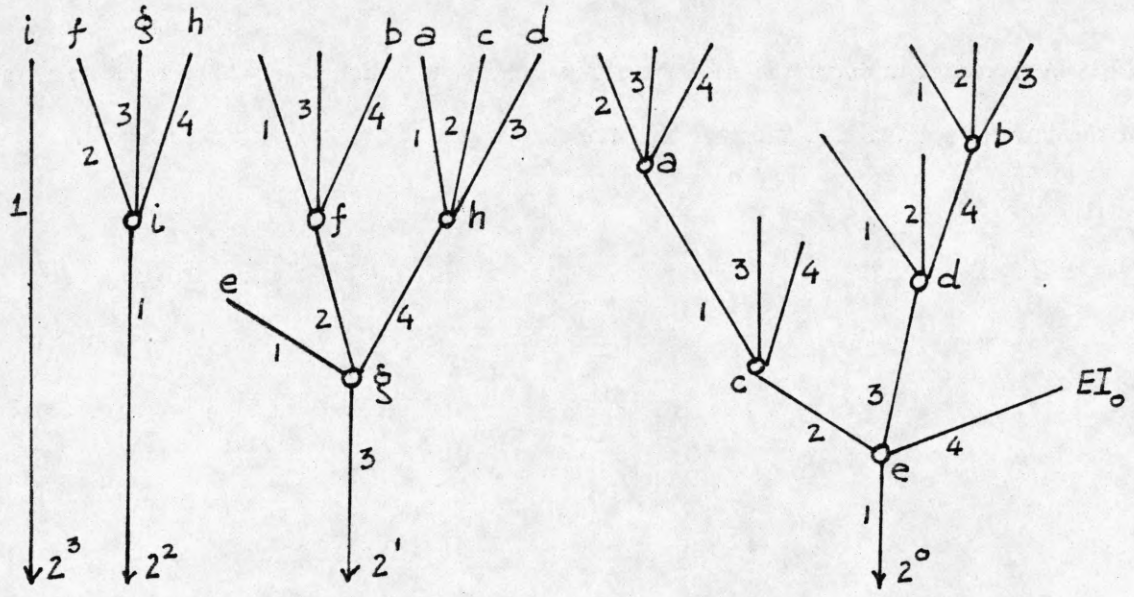

Figure 18: Reduction tree set and labelling for C(2,10:4)

EXAMPLE 11: Figure 17 shows the reduction tree set for the complete generalized counter C(1,0,6,11:5). The numbers at the outputs of the trees represent the corresponding weights of the bit slices. The reduction trees are obtained by tree growing and the timing heuristic is used.

EXAMPLE 12: Figure 18 shows the reduction tree set for the incomplete generalized counter C(2,10:4). The even number of inputs in bit slice 0 is made odd by adding the extra input EI_0.

5.6.2.2. Mapping the Reduction Tree Set into Hardware

The 1-1 mapping between every reduction tree and the corresponding full adder tree is trivial. We need only discuss how the carries between the trees are interconnected. Consider the reduction trees R_{N_j} and $R_{N_{j+1}}$ of bit slices j and $j+1$ respectively. The carries from the nodes whose output branches correspond to the set of labels I of R_{N_j} are connected to the inputs corresponding to the set of labels E of $R_{N_{j+1}}$, where $0 \leqslant j < k$. This interconnection is defined by the 1-1 mapping between the elements of I and E.

For every bit slice j in which a half adder has been substituted by a full adder , an extra input EI_j is created. Connect the output an AND gate to the input EI_j for every such bit slice j. Connect an input of this gate to any branch of a suitable reduction tree with the same label as EI_j.

Connect the other input of the AND gate to a TEST line, which is set to 0 during normal operation of the counter and is set to 1 during test mode.

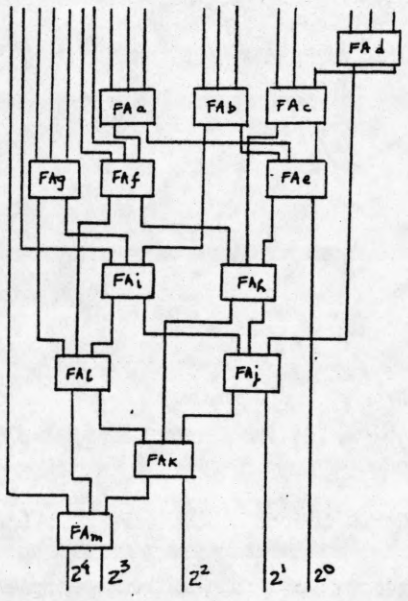

Figure 19: Hardware for C(1,0,6,11:5)

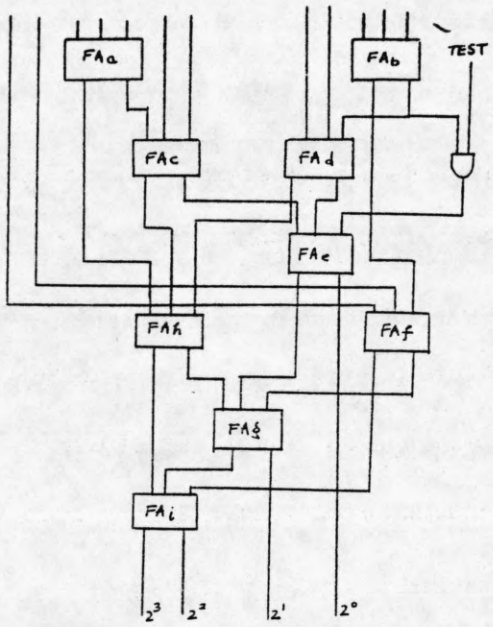

Figure 20: Hardware for C(2,10:4)

EXAMPLE 13: Figure 19 shows the hardware corresponding to the reduction tree set for C(1,0,6,11:5). The labels of the full adders and the labels of the nodes of the reduction tree set of

Figure 16 depict the 1-1 mapping between the two.

EXAMPLE 14: Figure 20 shows the hardware corresponding to the reduction tree set for C(2,10:4). The labels of the full adders and the labels of the nodes of the reduction tree set of Figure 17 depict the 1-1 mapping between the two.

5.6.3. The Testing Procedure

If the counter is complete, the 8 tests are defined by the labels on the controllable inputs. If the counter is incomplete, 9 tests for the counter are obtained as follows.

(1) With TEST=1, apply the 8 tests defined by the labels on the controllable inputs to the counter.

(2) With TEST=0 apply the test 11111,...1 to the controllable inputs to the counter.

THEOREM 7: All generalized counters constructed by design methodology 1 can be tested in 8 or 9 tests.

PROOF: Let bit slice k be the most significant bit slice of the counter.

Case 1: The generalized counter is complete.

The R_3 or R_2 tree of bit slice *k-1* is tested exhaustively. By theorem 1,a consistent labelling between $R_{N_{j+1}}$ and R_{N_j} is obtained by construction. Now every full adder in $R_{N_{j+1}}$ is tested exhaustively by test vectors on it's inputs defined by it's external labels. By virtue of the consistent labelling, it is possible to apply the above test vectors to the inputs of $R_{N_{j+1}}$ by applying a set of test vectors to the inputs of R_{N_j}. This latter set of test vectors test each full adder in R_{N_j} exhaustively. Hence, by the induction hypothesis, 8 tests can be found for the complete counter.

Case 2: The generalized counter is not complete.

With TEST=1, all the full adders in the circuit are tested as in case 1. This applies the combinations 11 and 10 to the respective AND gates (the first bit is the TEST bit). With TEST=0, the input vector 1111,..1 applies the test 01 to all the AND gates. This tests all the AND gates. Hence 9

tests are required. □

THEOREM 8: A complete generalized counter can always be designed to be testable in 8 tests. Also, incomplete generalized counters can always be designed, with the addition of AND gates as in design methodology 2, to be testable in 9 tests.

PROOF: The proof follows from theorem 6 and design methodology 2. □

6. Some Timing Issues

The computation speed of the counters designed, depends on the height of the reduction trees on which the designs are based. In constructing these trees, it is desirable to generate balanced trees as much as possible.

With regard to design methodology 1, consider the growing of R_{N_j} based on the input restrictions of $R_{N_{j+1}}$. Pairs of external inputs must be connected to pairs of branches {b,c} of external nodes of $R_{N_{j+1}}$ in such a manner that these nodes are at the largest height values in $R_{N_{j+1}}$. This reduces the height of the overall set of tree structures and is more time efficient.

With regard to design methodology 2, consider the partially grown tree R_t of theorem 6. The growing of this tree is based on the reduction tree $R_{N_{j+1}}$. At each step of the tree growing process, a label chosen from the set {E-I_t} is made an internal label of R_t. In order that R_t is a balanced tree, a proper choice of this label must be made at each step of the iteration. The following heuristic is used in order to minimise the depth of the counter circuit represented by the reduction tree set.

Timing Heuristic: In choosing labels from the set {E - I_t} to grow R_t, those labels are chosen first that come from branches of $R_{N_{j+1}}$ closest to the root node.

Designs using the above heuristic are shown in Figures 17 and 18.

7. Complex Labelling

Given an arbitrary design for a counter it is not always possible to find a consistent labelling for it with a simple labelling set. Consider the 15 input counter of Figure 21. Let the node names

represent one of four labels that can be assigned to the output branch of that node. For a consistent labelling, the following set of inequalities are obtained:

$$A \neq B \neq F \neq G \quad (1)$$

$$C \neq D \neq E \neq F \quad (2)$$

$$A \neq B \neq C \neq I \quad (3)$$

$$D \neq E \neq F \neq H \quad (4)$$

$$I \neq H \neq G \neq J \quad (5)$$

$$I \neq H \neq J \neq K \quad (6)$$

These lead to the inequality $A \neq B \neq H \neq I \neq J$. Therefore a simple labelling set is insufficient to solve the above set of inequalities. A solution with a complex labelling set is indicated in Figure 21. The following is the sketch of a branch and bound method used to generate test sets for counter circuits which are not amenable to simple labelling.

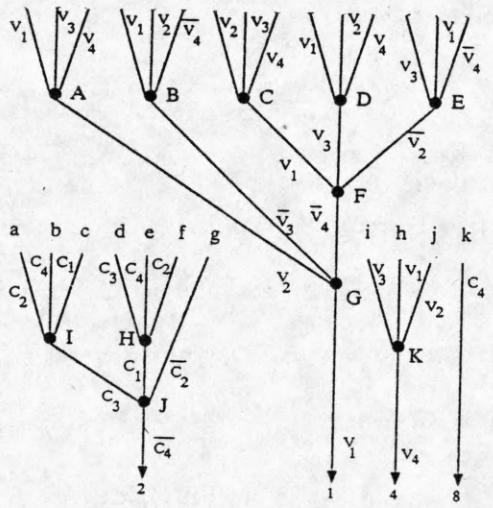

Figure 21: Complex labelling

Procedure Branch and Bound:

(1) Label the root node of the fundamental tree with the labels V_i, V_j, V_k and V_l on it's three input and one output branch, where $1 \leqslant i, j, k, l \leqslant 4$ and $i \neq j \neq k \neq l$.

(4) If the output branch of a node in the fundamental tree has been labelled, assign labels to it's input branches such that a consistent labelling is obtained.

(5) Repeat until all the nodes of the fundamental tree have been labelled.

The above method has been used to generate tests for several counter circuits. An example is presented in Figure 20.

8. Conclusions

This paper presents a new model for the testability of iterative circuits for which classical models such as the flow table are inadequate. It is seen that by exploiting some properties of full addition, the issue of reconvergent fanout is sidetracked. The major result is a set of 10 equations on which the entire proposition is based. The testing of counter circuits has been shown to reduce to the problem of labelling the branches of a tree based representation of the counter structure in a systematic manner. It is shown that all counter circuits can always be designed to be testable in either 8 or 9 tests.

ACKNOWLEDGEMENTS

We would like to thank Subhasis Laha who suggested the method of Construction 1 and Professor Prithviraj Banerjee of the University of Illinois for the many valuable discussions and helpful remarks. We would like to thank Dr. Sung Je Hong of GE R&D Schenectady, N.Y for his helpful discussions especially with regard to the corollary of Theorem 1. We also acknowledge the help and support of Dr. Manuel D'Abreu of GE R&D Schenectady.

REFERENCES

[1] A.D. Friedman, "Easily Testable Iterative Systems," *IEEE Transactions on Computers*, Vol C-22, No 12, Dec 1973.

[2] Francisco J.O. Dias, "Truth Table Verification of an Iterative Logic Array," *IEEE Transactions on Computers*, Vol C-25, No 6, June 1976.

[3] W.H. Kautz, "Testing for Faults in Cellular Logic Arrays," *Proc, 8'th Annu. Symp. Switching and Automata Theory*, 1967.

[4] John Paul Shen and Joel Ferguson, "Easily Testable Array Multipliers," *Proc, 13'th Int'l Symp. on Fault Tolerant Computing*, June 1983.

[5] R. Parthasarthy and S.M. Reddy, "A Testable Design of ILA's," *IEEE Transactions on Computers*, Vol C-31, pp 833-841, Nov 1981.

[6] Hasan Elhuni, Anastasios Vergis, Larry Kinney, "C-Testability of Two-Dimensional Arrays of Combinational Cells," *Proc, Int'l Conference on Computer Aided Design*, Nov 1985.

[7] S.C. Seth, K.L. Kodandapani, "Diagnosis of Faults in Linear Tree Networks," *IEEE Transactions on Computers*, Vol C-26, pp 29-33, Jan 1977.

[8] J.A. Abraham and D.D. Gajski, "Design of Testable Structures Defined by Simple Loops," *IEEE Transactions on Computers*, Vol C-33, pp 875-884, Nov 1981.

[9] Wu Tung Cheng, "Testing and Error Detection in Iterative Logic Arrays," *Ph.D Thesis*, University of Illinois at Urbana-Champaign, 1985.

[10] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Freq*, pp 349-356, May 1965.

[11] Kai Hwang, "Computer Arithmetic, Principles, Architecture and Design," *John Wiley and Sons*, 1979.

[12] Edward Reingold, Jurg Nievergelt, Narsing Deo, "Combinatorial Algorithms: Theory and Practice," *Prentice Hall Inc*, 1977.