Test Generation for Triangular Davio Lattice Structures

Scott Guthridge and Marek Perkowski
Department of Electrical and Computer Engineering
Portland State University
Portland, OR 97207, USA
mperkows@ece.pdx.edu

Abstract

Integrated circuits with a highly regular, repeated structure outperform IC's with complex logic interconnects, in transistor density, gate delay time, and testability. They will be even more important for future technologies such as single electron transistors. These advantages drive the search for highly regular structures that can be used to replace complex logic. This paper studies synthesis and test of regular lattice structures built of Davio gates. Tools useful for analysis and synthesis of the triangular logic lattice structures are given. Finally, universal tests for the structures are developed.

1. Introduction

Integrated circuits with a highly regular, repeated structure outperform IC's with complex logic interconnects in transistor density, gate delay time, and testability. They will be even more important for future technologies such as single electron transistors [9], Chemically Assembled Electronic Nanotechnology [7], quantum dots [13] and others [4]. These advantages drive the search for highly regular structures that can be used to replace complex logic.

Over the last thirty years, the transistor density of memory I.C's, which have a highly regular structure, has increased at a rate of about 1.5 times per year. In comparison, transistor density in microprocessors, which have complex interconnects, has increased at the lower rate of 1.25 times per year. The reason for this difference is that a significant fraction of the chip area in the later case is consumed by the interconnects, thus the overall transistor density is lower, [6] , pp. 2-3.

Further, in submicron CMOS technologies, interconnect delay dominates gate delay within the I.C. This is caused by parasitic capacitances in the interconnect wires, especially, the capacitance due to fringing fields, [6], pp. 222-226. Finally, in a highly regular structure, there's opportunity to find universal test patterns for the structure, whereas in complex logic circuits, test vectors must be tailored to the particular circuit, and may be difficult to generate.

This paper is a study on highly regular lattice structures and diagrams, based on Davio gates, focusing both on logic synthesis and universal test. Such diagrams, introduced in [8] and further studied in [10] have from nanotechnology realization point of view even more interesting properties than Shannon-expansion-based (mux) Akers Arrays [1] and lattice diagrams from [2,9]. The reason for this is that EXOR gate from Davio gate is easily realizable in several future technologies, especially single-electron-transistor, and that Davio gate is a part of Toffoli and Generalized Toffoli Gates that are the base of reversible and quantum logic. All symmetric functions are well-realizable in triangular structures. Most of non-symmetric benchmark functions are either realizable with small number of variable repetitions or a decomposable to triangular arrays, each with small number of variable repetitions. The presented method is thus, in general , used in conjunction with functional decomposition of the Boolean function.

1.1 . Davio Gate

The Davio gate consists of a two-input AND gate and an XOR gate realizing function $F = AB \oplus C$. These gates can be used to construct lattice structures such as the triangular structure illustrated in Figure 1.

The **b** (bottom) inputs of the lattice are connected to memory cells within the IC and are used to program the function that the lattice is to implement. The **s** (side) inputs are the external inputs to the circuit. Finally, **f** is the desired function output. Both the triangular lattice structure and a rectangular structure, discussed later, can be used to implement any symmetric functions of the ***s*** inputs. Before we can discuss these structures in detail, we need to introduce symmetric logic.

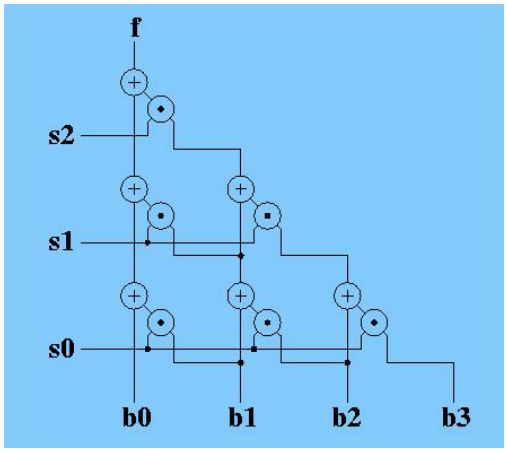

Figure 1. Triangular Lattice Structure using positive Davio gates. It can realize any symmetric function of three variables.

2. Symmetric Logic

Symmetric logic functions are those in which the output depends only on the number of inputs set to one. In other words, it's only the number, not which particular inputs are set. Many familiar logic functions are symmetric: n-input AND, OR, XOR, NAND, NOR, XNOR. We can describe any symmetric function as a set of S(**i**) values, where S(**i**), indicates that the output (**f**) should be equal to one whenever **i** inputs are equal to one. To illustrate this notation, consider the following K-map (Table 1) which describes all symmetric functions in four variables.

Table 1. Karnaugh map illustrating symmetric indices for describing all symmetric functions of four variables

K-Map of all 4-input Symmetric Functions

	s2's3'	s2's3	s2 s3	s2 s3'
s0's1'	S(0)	S(1)	S(2)	S(1)
s0's1	S(1)	S(2)	S(3)	S(2)
s0 s1	S(2)	S(3)	S(4)	S(3)
s0 s1'	S(1)	S(2)	S(3)	S(2)

This K-Map shows an example of what we call “Flower Pattern” due to its resemblance to a flower about S(4) (compare levels of grey). Note that in each case, the value, **i**, in S(**i**) indicates the number of inputs that are set to one. Each S(**i**) can be independently set to zero or one. For example, to implement the AND function, the S values would have the following values: S(0) = 0, S(1) = 0, S(2) = 0, S(3) = 0, S(4) =1. For the XNOR function, the S(**i**) would be set as: S(0) =1, S(1) = 0, S(2) = 1, S(3) = 0, S(4) = 1. A more compact way to write the S(**i**) values is to represent them as a column vector, S, where the top element is S(0). Thus, the XOR function can be written: S = [0][1] [0] [1] [0]. Table 2 below illustrates how each 2-input symmetric function can be realized using the triangular lattice structure above.

Table 2.Explanation on indices of all symmetric functions of two variables to be realized in a triangular lattice structure

Two-Input Symmetric Functions

Number of Side Inputs Set	S vector			b vector			Function Name
	S(0)	S(1)	S(2)	b0	b1	b2	
n/a	0	0	0	0	0	0	false
0	1	0	0	1	1	1	nor
1	0	1	0	0	1	0	xor
0, 1	1	1	0	1	0	1	nand
2	0	0	1	0	0	1	and
0, 2	1	0	1	1	1	0	xnor
1, 2	0	1	1	0	1	1	or
0, 1, 2	1	1	1	1	0	0	true

The "**b**" vector is simply the $\mathbf{b_0, b_1, b_2, ...}$ b_columns values at the base of the lattice structure. The "S" vector specifies the symmetric function desired at output f0. The components of this vector, S(0), S(1), S(2), ... S(rows) specify that $\mathbf{f_0}$ should be 1 when 0, 1, 2, ... rows, respectively side inputs (**s0, s1, s2,** ...) are asserted. How the **b** vectors for these functions are synthesized is discussed in later sections.

2.1. Implementing Arbitrary Logic Using Symmetric Logic

By repeating inputs, i.e. wiring an external input to two or more **s** inputs into the lattice, it's possible to implement any logic function via symmetric logic. This can be done in a similar fashion to variable repetition in Universal Akers Arrays [1] and Shannon-type Lattices [2,7,8,10]. The general lattice diagram fabric found with time several variants, such as Triangular and Rectangular Lattices. As shown by Akers, in the worst case the total number of repetitions in triangular array is exponential, but experimental results from [2,10] show that the total number of repetitions for standard benchmark functions is much smaller, and 2.5 on the average. Of course, some functions require more variable repetitions and in such cases the function should be first partitioned to separate triangular arrays.

3. Triangular Lattice Architecture

The triangular Davio lattice structure was introduced in the previous section. This section studies the triangular lattice in detail. The triangular lattice lattice can be extended to any number of rows by adding stages on the right hand side. The number of columns at the base of the lattice is always one greater than the number of rows.

3.1 Transformation by a Lattice Row

When all **s** inputs are zero, the AND gates prevent signals from passing between the columns of the lattice, and the control signals pass upward without modification. Only the leftmost control input, $\mathbf{b_0}$, passes through to the lattice output, **f**. If any **s** input is set to one, however, the corresponding row transforms the control signals passing upward through it by exclusive OR'ing the row of signals with a left-shifted copy of the row. Mathematically, we can express this transformation as:

$\mathbf{b_j' = b_j \oplus b_{j+1}}$, for all **j** where $\mathbf{b_{j+1}}$ exists.

Where, the ⊕ operator represents XOR, or equivalently, addition in Galois Field(2). For two or more **s** inputs set to one, we can apply the above rule recursively:

$\mathbf{b_j' = b_j \oplus b_{j+1}}$

$\mathbf{b_j'' = b_j' \oplus b_{j+1}' = (b_j \oplus b_{j+1}) \oplus (b_{j+1} \oplus b_{j+2}) =}$

$\mathbf{= b_j \oplus b_{j+2}}$

$\mathbf{b_j''' = b_j'' \oplus b_{j+1}'' = (b_j' \oplus b_{j+1}') \oplus (b_{j+1}' \oplus b_{j+2}') = b_j' \oplus b_{j+2}'}$

$\mathbf{= (b_j \oplus b_{j+1}) \oplus (b_{j+2} \oplus b_{j+3}) =}$

$\mathbf{= b_j \oplus b_{j+1} \oplus b_{j+2} \oplus b_{j+3}}$

...

Where $\mathbf{b_j}$ is the transformation for S(0), $\mathbf{b_j'}$ is the transformation for S(1), $\mathbf{b_j''}$ is the transformation for S(2), etc. The subscript **j** identifies the lattice column. When **j** = 0, these formulas give the relationship between the control inputs and the output.

Let's take another look at the transformations above, but this time let **j** = 0, and don't optimize out the duplicate terms:

$\mathbf{b_0' = b_0 \oplus b_1}$

$\mathbf{b_0'' = b_0 \oplus b_1\ \hat{}2 \oplus b_2}$

$\mathbf{b_0''' = b_0\ \oplus b_1\ \hat{}3 \oplus b_2\ \hat{}3 \oplus b_3}$

$\mathbf{b_0'''' = b_0\ \oplus b_1\ \hat{}4 \oplus b_2\ \hat{}6 \oplus b_3\hat{}4 \oplus b_4}$

$\mathbf{b_0''''' = b_0\ \oplus b_1\ \hat{}5 \oplus b_2\ \hat{}10 \oplus b_3\ \hat{}10 \oplus b_4\hat{}5 \oplus b_5}$

...

By **b^v** we denote symbol **b** exored **v** times**.** Another pattern emerges. Notice that the exponents of the terms are the coefficients of the binomial expansion! Therefore, we can calculate the exponents directly using the formula:

$$\frac{i!}{j!\,(i-j)!}$$

Where, **i** is the number of primes listed after $\mathbf{b_j}$, and **j** is the column. Recall that in Galois Field(2), a term raised to an even power disappears while a term raised to an odd power is the same as the term itself. Thus we can determine if a term is present or absent on the right-hand side by calculating the above expression and testing whether the result is odd or even.

The extreme case of the row transformations occurs when all **s** inputs are equal to one. Studying this case can provide additional insight into what's happening in the Davio lattice structure. When all **s** inputs equal one, it's as if the AND part of the Davio gates have been replaced by jumpers, effectively changing the structure into a triangular lattice of XOR gates. Figure 2 illustrates this degenerate Davio lattice structure. All parts of the circuit that have even numbers of paths to the root (and thus have no effect on the output) have been removed from the drawing. Missing XOR gates in the figure correspond to missing terms in the row transformation formulas above. Note the recursive triangular pattern in the figure, where the top three elements are repeated twice below, then the resulting nine element structure is repeated twice below that, the resulting twenty-seven element structure is repeated twice below that, etc. This recursive triangular pattern is important to Davio regular structures.

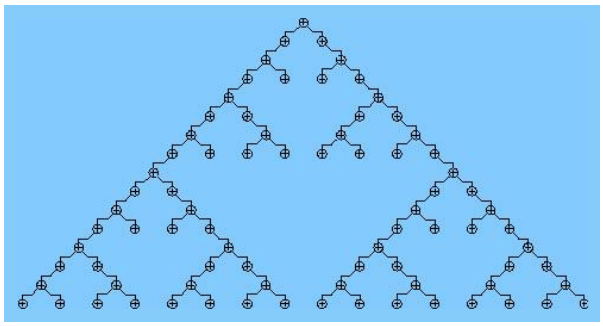

Figure 2. Illustration of a recursive XOR tree realized within a triangular lattice which is helpful to understand methods from this paper.

3.2 Logic Synthesis

Summarizing the section above, the output is related to the control inputs and the number of **s** inputs set by the following relationship:

for S(0): $\mathbf{f = b_0}$

for S(1): $\mathbf{f = b_0' = b_0 \oplus b_1}$
for S(2): $\mathbf{f = b_0'' = b_0 \oplus b_2}$
for S(3): $\mathbf{f = b_0''' = b_0 \oplus b_1 \oplus b_2 \oplus b_3}$
for S(4): $\mathbf{f = b_0'''' = b_0 \oplus b_4}$
...
To generate the **b** vector that makes **f** = 1 for a given set of S(**i**) values, all we need to do is substitute S(**i**) for **f** in each row and then solve the resulting system of linear equations in Galois Field(2).
S(0) = $\mathbf{b_0}$
S(1) = $\mathbf{b_0 \oplus b_1}$
S(2) = $\mathbf{b_0 \oplus b_2}$
S(3) = $\mathbf{b_0 \oplus b_1 \oplus b_2 \oplus b_3}$
S(4) = $\mathbf{b_0 \oplus b_4}$
...
Because the last term of each row, **i**, is always a new element, $\mathbf{b_i}$, the rows are orthogonal and the system always has a solution. A later section details the use of matrices to do this synthesis mechanically.

3.3. Universal Test

One of the motivations behind using a regular lattice structure is that in concept, we should be able to take advantage of the regular structure to develop a universal test. For the triangular lattice structure, however, this has proven to be rather challenging.

3.3.1. Difficulty in Testing the Triangular Lattice

To illustrate why this is difficult, let's try to derive a test sequence for the for the lattice in which the number of test vectors is linearly related to the number of rows in the table. For this to be possible, we have to find ways of testing several gates at once. Begin with a single Davio gate. The following test vectors, Table 3, are sufficient to find all single stuck-at faults in the gate.

Table 3.

Test Vectors for
Single Davio Gate

	s	b0	b1	f
1	1	0	0	0
2	1	0	1	1
3	1	1	1	0
4	0	1	1	1

All Davio gates in a particular lattice row share a common **s** input. What we'd like to do is find **b** vector patterns that simultaneously test the whole row of gates. For the case of **s** = 0, a **b** vector of all ones covers all gates at once. For the cases where **s** = 1, there are a couple different possibilities involving four repeating **b** patterns, Table 4, Table 5.

Table 4

Test Vectors for a Row of Davio Gates (Option 1)

	s	b0	b1	b2	b3	b0'	b1'	b2'
1	1	0	0	0	0	0	0	0
2	1	1	1	1	1	0	0	0
3	1	0	1	0	1	1	1	1
4	1	1	0	1	0	1	1	1
5	0	1	1	1	1	1	1	1

The Table 4 option is better because it, at least, provides output vectors that can be used on the rows above. As may also be apparent from Tables 4 and 5, a 1-0-1-0- alternating vector becomes all ones 1-1-1- when

passing through a row with **s** = 1, and the all ones pattern 1-1-1-1- becomes all zeros 0-0-0- when passing through a row s = 1. Let's try to use this property to do testing in multiple rows simultaneously. The strategy is this:

1. Preset the top-most row with the 0-1 pattern (and for consistency, also present it with 1-0). This can be done from the bottom of the lattice by setting the s inputs of all rows below to zero.
2. Present the second row with the 0-1-0 and 1-0-1 patterns. Take advantage of the fact that the outputs from this row will be 1-1-1 patterns, which allows us to present the top row with test vectors 2 and 5 in the Option 1 table above.
3. Present the third row with the 0-1-0-1 and 1-0-1-0. This will create 1-1-1 patterns going up, and if **s1** = 1, it will create 0-0 patterns to the top row, allowing us to complete the testing of the top row.
4. Continue on through the lattice, testing adjacent rows three at a time. At the end, we need to manually generate the 1-1-1-1- and 0-0-0-0- patterns to complete the testing of the bottom rows.

Table 5.

Test Vectors for a Row of Davio Gates (Option 2)

	s	b0	b1	b2	b3	b0'	b1'	b2'
1	1	0	0	1	1	0	1	0
2	1	0	1	1	0	1	0	1
3	1	1	1	0	0	0	1	0
4	1	1	0	0	1	1	0	1
5	0	1	1	1	1	1	1	1

For the three-level lattice shown in Figure 1, our test vectors are given in Table 6. The asterix columns illustrate the test patterns from Table 4 above for the bottom, middle and top rows, respectively. Using the ``print fault table option'' (-f) of the generic ATPG program from [14] and slicing rows from the table by hand, we can show that for the 3-level triangular lattice, the above test vectors cover 63 out of 66 single stuck-at faults.

Why doesn't it cover all faults? The are two main reasons. First, even though all gates are being driven through their test patterns, and all **s** inputs above are set to one, many of the gate outputs propagate to the lattice output by more than one path. If the number of paths from a gate output to the lattice output is even, then the lattice output does not change when the gate output changes.

Table 6

Test Vectors for Three-Level Triangular Lattice

s0	s1	s2	b0	b1	b2	b3		b0'	b1'	b2'		b0''	b1''		f
0	0	1	0	1	0	1		0	1	0		0	1	*	1
0	0	1	1	0	1	0		1	0	1		1	0	*	1
0	1	1	0	1	0	1		0	1	0	*	1	1	*	0
0	1	0	1	0	1	0		1	0	1	*	1	1	*	1
1	1	1	0	1	0	1	*	1	1	1	*	0	0	*	0
1	0	1	1	0	1	0	*	1	1	1	*	1	1	*	0
1	1	1	1	1	1	1	*	0	0	0	*	0	0		0
0	1	1	1	1	1	1	*	1	1	1		0	0		0
1	1	1	0	0	0	0	*	0	0	0		0	0		0

Second, because we're testing more than one row at a time, it's possible that a higher-level row masks a fault detected by a lower row, i.e. the data going to the higher row isn't what it's supposed to be, but ends up creating the correct response by the time it reaches the lattice output.

For small lattice structures, the generic ATPG program from [14] does find relatively short solutions that detect *all* single stuck-at faults. The following three tables, Table 7, Table 8, and Table 9 show some examples.

Table 7

Test Vectors to Detect All
Single Stuck-At Faults in Triangle(3)

	s0	s1	s2	b0	b1	b2	b3
1	1	1	1	0	0	0	1
2	1	1	1	0	1	1	1
3	0	0	1	1	1	1	1
4	1	1	1	1	0	1	0
5	1	0	1	1	1	1	0
6	0	1	1	0	0	0	1
7	1	1	0	0	0	1	1
8	1	0	1	0	0	0	0

Table 8

Test Vectors to Detect All Single Stuck-At Faults in Triangle(2)

	s0	s1	b0	b1	b2
1	1	1	0	0	1
2	1	1	1	1	1
3	0	1	1	0	1
4	1	1	0	1	0
5	0	1	0	1	0
6	1	0	0	0	1

These solutions suggest that an Order(n) solution probably always exists, however it might not be universal. After much study of these solutions (and many others), we was not been able to find a universal test that can be executed in linear time that covers all faults. What about a less aggressive goal? Can we find a simple, easy to prove, universal test method that executes in quadratic, $O(n^2)$ time?

3.3.2. A Simple Order(n^2) Solution

Using the method in this section, we can detect all single stuck-at faults in the triangular lattice. Additionally, we can gather enough information to make it possible to localize most faults. Finally, we can also detect most stuck-open faults within the Davio gates. The down side is that it requires on the order of $\mathbf{n}^2$ test vectors, where n is the number of lattice rows.

The high-level strategy is to work from the top of the lattice down the first column, testing one gate at a time. We then test the second column, starting from the top and working down, continuing with subsequent columns until all lattice columns have been covered. For each gate, we present the test vectors shown in Table 3, with the addition that vector 1 is repeated at the end. The four test vectors shown in Table 3 are sufficient to find stuck-at faults in the Davio gate. Presenting these patterns in the order shown and repeating vector 1 at the end detects most stuck-open faults within the Davio gate. The Table 3 test vectors applied to each gate are marked in blue in the example below.

Table 9

Test Vectors to Detect All Single Stuck-At Faults in Triangle(7)

	s0	s1	s2	s3	s4	s5	s6	b0	b1	b2	b3	b4	b5	b6	b7
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1
2	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1
3	1	1	1	1	1	1	1	0	1	1	0	1	1	0	0
4	0	1	1	1	1	1	1	0	1	1	0	1	1	1	1
5	1	0	1	1	1	1	1	1	1	0	0	0	1	0	1
6	0	1	0	1	1	1	1	0	0	1	0	0	0	1	0
7	0	0	0	1	1	0	1	0	0	1	1	1	1	0	0
8	1	0	0	0	1	1	1	0	0	1	1	1	0	1	0
9	1	1	1	0	0	0	1	0	1	0	0	1	0	0	0
10	1	1	1	1	1	1	0	0	0	0	0	0	1	1	0
11	1	1	1	1	0	1	1	0	1	0	1	1	0	1	0
12	1	1	1	0	1	0	1	0	0	0	0	0	1	0	1
13	1	1	1	0	0	1	1	0	0	0	1	0	1	0	0
14	1	0	0	0	1	1	1	0	0	0	1	0	0	1	0
15	1	1	1	0	1	1	1	0	1	1	0	0	0	1	0
16	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
17	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0

When testing other than the first column we have to turn on the **s** inputs for the columns above the triangle under test in order to propagate the value from the column under test to the lattice output. Putting it more succinctly, When testing column **j**, the first **j s** inputs (at the top of the lattice) must be set to one. These ones are marked in yellow in the example below.

One final complication: When table 3, test vector 3 is applied, columns left of the column under test contribute 1's which are effectively XOR'ed into the lattice output. If the number of 1's contributed by these

columns is odd, then the lattice output for vector 3 is the inverse of the output of the gate under test. Using the formulas in section 2.1 above, we can see that the number of 1's contributed by the columns to the left is equal to the number of columns on the left. Thus, the expected lattice output for test vector 3 must be inverted on all odd-numbered columns. These inverted cases are marked in red in the example below.

Table 10 is an example of this technique applied to the three-level shown in Figure 1.

Following is an implementation of the test pattern generation program for the triangular lattice. The program is written in Octave, the public domain version of MATLAB:

```
%
% Size of the Lattice in Rows
%
N = 3;

%
% Test Patterns for Each Gate
% Columns: s b0 b1 f
%
patterns = [
   1 0 0 0
   1 0 1 1
   1 1 1 0
   0 1 1 1
   1 0 0 0
];

%
% Generate Test Vectors
%
for col = 1:N                    % for each column
   for row = 1:N                 % for each row
      for i = 1:5                % for each pattern
         %
         % Initialize s and b vectors.
         %
         s = zeros(N, 1);
         b = zeros(1, N+1);

         %
         % Fill in the appropriate
         % `s' and `b' values
         % from the test pattern.
         %
         s(row) = patterns(i, 1);
         b(col:col+1) = patterns(i, 2:3);

         %
         % Get the expected `f' value.
         % If we're on test pattern 3
         % and an odd-numbered column,
         % invert the expected output.
         % Note that
         % octave's arrays begin with
         % index 1 where
         % our arrays begin with index 0, so the
```

```
            % logic really has to to check for even
            % instead of odd in this case.
            %  Note that
            % rem() is remainder.
            %
            f = patterns(i, 4);
            if ((i == 3) && (rem(col, 2) == 0))
               f = !f;
            endif

            %
            % Finally, fill in the `s'
            % values necessary
            % to propagate results from
            % the column under
            % test to the output.
            %
            s(1:(col-1)) = ones(col-1, 1);

            %
            % Combine values and print.
            % The order
            % of the output columns is:
            %       s0, s1, s2, ... b0, b1, b2, ... f.
            %
            vec = [fliplr(s') b f]
          endfor
        endfor
      endfor
```

4. Conclusion

We presented triangular lattice diagrams that use Davio gates. Fast algorithm for realizing symmetric functions has been presented as well as quadratic complexity algorithm for test generation. Using the method from section 2.3.2 we can detect all single stuck-at faults in the triangular lattice. Additionally, we can gather enough information to make it possible to localize most faults. Finally, we can also detect most stuck-open faults within the Davio gates. It is an open problem to investigate how many and what types of stuck-open faults are not detectable using this method.

These results are also useful for arbitrary (non-symmetric) functions realized in Positive Davio Lattices by repeating input variables in levels. The presented diagrams have been extended to diagrams from (reversible) Toffoli gates and the good testability properties of Davio Lattices are also inherited by Reversible Davio Lattices, making them thus useful to several future nano-technologies. Efficient testing with universal tests will be even more important for future technologies than it is for current ones. The triangular planes can be surrounded by special deterministic BIST circuits created especially for this technology.

5. References

1. S. B. Akers. A Rectangular Logic Array*, IEEE Trans. on Computers*. Vol. C-21, pp. 848-857, 1972
2. M. Chrzanowska-Jeske, Y. Xu, M. Perkowski, Logic Synthesis for a Regular Layout. *VLSI Design: An International Journal of Custom-Chip Design, Simulation, and Testing*, 1999.
3. P. Farm and E. Dubrova, Technology Mapping for Chemically Assembled Electronic Nanotechnology, *Proc. IWLS '02.*

4. S. Goldstein and M. Budiu, Nanofabrics: Spatial computing using molecular electronics, *Proc. 28th Annual International Symposium on Computer Architecture*, (Gothenborg, Sweden), June 2001.
5. H. Hasegawa, A. Ito, Ch. Jiang, and T. Muranaka, Atomic Assisted Selective MBE Growth of InGaAs Linear and Hexagonal Nanowire Networks For Novel Quantum Circuits, *Proc. 4th Intern Workshop on Novel Index Surfaces (NIS'01),* Sept 16-20, Apset France.
6. Sung-Mo Kang and Y. Leblebici, CMOS Digital Integrated Circuits, 2nd ed: McGraw Hill, 1999
7. P. Lindgren, R. Drechsler, B. Becker, Synthesis of Pseudo-Kronecker Lattice Diagrams. *Proc. of Intl. Worshop of Applications of Reed-Muller Expantions to Circuit Synthesis, 1999*, Victoria, B. C., Canada, pp. 197 - 204.
8. M. Perkowski, and E. Pierzchala, New Canonical Forms for Four-Valued Logic, *Report, Electrical Engineering Department, PSU* . 1993.
9. E. Pierzchala, and M. Perkowski, *Patent #5,959,871,* September 28, 1999. *Programmable Analog Array Circuit.*
10. M. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, Lattice Diagrams Using Reed-Muller Logic. *Proc. of Intl. Worshop of Applications of Reed-Muller Expansions to Circuit Synthesis, 1997*, Oxford Univ., U.K., pp. 85 - 102.
11. M. Perkowski, A. Al-Rabadi, P. Kerntopf, A. Mishchenko and M. Chrzanowska-Jeske, Three-Dimensional Realization of Multiple-Valued Functions using Reversible Logic, Invited Talk, *Proc. Workshop on Post-Binary Ultra-Large Scale Integration Systems (ULSI),* , pp. 47 - 53, May 21, 2001, Warsaw, Poland.
12. G. Strang, Linear Algebra and its Applications, 3rd ed: Harcourt, 1988, pp. 1-116.
13. T. Yamada, Y. Kinoshita, S. Kasai, H. Hasegawa, Y. Amemiya, Quantum Dot Logic Circuits Based on Shared Binary-Decision Diagram*, Jpn. J. Appl. Phys.* Vol. 40, 2002, pp. 4485-4488, Part1, No. 7, July 2001.
14. Scott's Guthridge Webpage. http://063-170-214108.dslnorthwest.net/scooter/ece510td/project/index.html

Table 10.