ABSTRACT
This paper presents a unifying approach to the testing of fine-grained VLSI arrays. The approach covers a wide range of regular arrays and leads directly to test pattern generation. The approach includes the often overlooked but non-trivial problem of establishing fault propagation paths. The approach can be extended to encompass a multiple fault model and in this case the problem of fault effect propagation becomes more complex.

Cheng [1] highlights an important difference between cell verification and array verification. In cell verification, each cell in the array must be verified against the chosen fault models (at the cell level and at the array level) but in array level verification it is only necessary to verify the function of the complete array. The principal difference is that it can arise that some states that might exist in a cell are precluded by the array interconnections. In this case a requirement for cell verification would conclude that the array is untestable. Since this situation often arises with restricted access to the boundary of an array, we consider only array verification in which we only need to test for detectable faults.

Fault Models
The key to testing regular arrays is the ability to examine individual cells at a detailed level and then to generalise the results to the array level. It is therefore necessary to select appropriate fault models both for the cell and also at the array level. Typical cell level fault models would be a functional model, a 'stuck-at' model or a technology dependent model such as CMOS transistor faults. Typical array level fault models would be either a single faulty cell model or else a multiple faulty cell model.

Cell Level Fault Model
In this paper we assume a functional fault model for the cell. The functional fault model [2] is independent of the technology chosen to implement the circuit design. A fault under this model manifests itself as an arbitrary change to the logical operation of the circuit. The assumption of this model is that there is no increase in the number of states in a cell. Thus, provided that in a sequential cell the initial state is known, the model requires just a single vector for each fault. Nevertheless the entire truth table (or transition table) must be verified, i.e. all possible cell inputs must be applied.

Array Level Fault Model
When a fault exists in a cell the fault effect must be propagated to the boundary of the array where it can be detected. The choice of fault model at the array level determines the complexity of establishing a fault propagation path. In this paper we consider only the simplest array level fault model where it is assumed that at most one cell of the array is faulty. The requirements for establishing fault propagation paths are discussed later. The approach can be extended to encompass a multiple fault model and in this case the problem of fault effect propagation becomes more complex.

Data Flow
In regular arrays the data can flow in different directions and at different speeds. A general method used to represent the data flow in regular arrays is the Data Dependence Graph (DDG) [3]. The DDG is a graph that shows the computations that occur in an array and the dependency of the different data streams on each other. When associated with a particular hardware implementation it also dictates the sequence of computation and the propagation of data with time. Each node in the DDG represents a computation executed by an array cell at a specific time. Each edge of the graph represents a data value and demonstrates the data dependence between computations. Figure 1 shows a 1-D array and its corresponding DDG. The $i$ direction represents the movement of $S^i$ data across the array. The $(i+2j)$ direction represents the counter flow of the $S^j$ data in the array. The $j$ direction represents the sequence of $S^j$ values applied to the array.

The DDG always has one more dimension than the
corresponding physical array so that the common case of two
dimensional arrays involve the use of three dimensional DDGs.
Counter-flow data or different speed unilateral flows merely
result in dependencies in different directions in the DDG. The
DDG is therefore a useful formalism for representing a large
class of arrays and furthermore it is already in use in the design
environment. In solving the test problem it is convenient to
consider first the propagation of each of the different data
streams. Formally with the DDG we examine test vector
propagation and fault effect propagation along each of the
distinct directions of the DDG which are connected by data
dependencies.

Test Vector Propagation

Since the DDG describes the data flow, the precise nature of the
dependence can be described for each distinct direction in the
DDG. This is achieved by using a propagation graph for each
distinct signal direction Sj in which data propagates through the
DDG. The dependence is a function of the cell logic which alters
the data as it passes through the cell. A propagation graph is
defined as \( G_{S_j} = (V, F_{S_j}) \) where the set of nodes \( V = \{V_1, V_2, \ldots, V_n\} \) are the possible states of a cell (i.e. potential test
vectors) and the arcs \( F_{S_j} \) represent the possible propagations of
the vectors in the direction \( S_j \). An arc exists from \( V_j \) to \( V_k \) when
a vector \( V_j \) at a node of the DDG is compatible with a vector \( V_k \)
at the next node in signal direction \( S_j \) if we had free choice of the
data along the other signal directions.

The cells in the array in Figure 1 have the truth table of Table 1.
If vector 2 is applied to a cell, the output on signal \( S_1 \) is a logic
1. Therefore only the vectors 4, 5, 6 and 7 can be applied to the
next cell along the \( S_1 \) signal direction. The output on signal \( S_2 \)
is a logic 0 and only the vectors 0, 1, 4 and 5 can be applied to
the next cell along that direction. The complete propagation
graphs for signals \( S_1 \) and \( S_2 \) are shown in Figure 2.

By examining the propagation graphs we can determine a
classification of the array. If every vector can be included in
compatible closed paths or "cycles" in each of the signal
directions, then the array is C-testable [4]. If a vector is in a
cycle then it can be applied to a series of cells in that signal
direction for a single test application and therefore the test set
required to apply that vector to every cell in the array is not
dependent on the size of the array. If a vector cannot be
included in compatible cycles then a linear test set must be
employed to propagate the vector to every cell in the array and
the size of the test set grows with the array size. In Figure 2
vectors 2 and 4 are not included in a cycle for signal direction
\( S_2 \). Therefore successive cycles of vectors 6 and 7 are required
to propagate the vectors to every cell in the array [5].

Table 1. truth table

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>( S_1 )</th>
<th>( S_2 )</th>
<th>( S_3 )</th>
<th>Outputs</th>
<th>( S_1 )</th>
<th>( S_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Fault Effect Propagation

A fault propagation path must be created to the boundary of the
array such that the existence of a fault at the output of the cell
under test can be detected. The DDG again allows each signal
direction in the array to be considered separately for fault effect
propagation. There are many choices available in establishing a
fault propagation path from the output of the cell under test to the
array boundary. A simple and sufficient approach is to establish
a fault propagation path from the faulty cell output along that
distinct signal direction to the array boundary. The choice of
multiple or single faulty cells will lead to the creation of different
types of fault propagation paths, since the multiple case requires
the effects of fault masking due to other erroneous cells to be
considered [1]. In this paper we will only consider the single
faulty cell case.
At the binary bit level a fault will manifest itself as the inverse of the expected bit. For example consider an array of the cells of Figure 1 with the propagation graphs of Figure 2. If the vector \( \overline{3} \) is applied to a cell the correct response in the \( S^1 \) signal direction is a logic 1. Any fault that is detected by vector \( \overline{3} \) on the \( S^1 \) line will manifest itself as the inverse of the correct value, which is logic 0. Therefore to establish a fault propagation path to the array boundary along the \( S^1 \) signal direction we must be able to discriminate between the correct and the faulty cell response to the test vector. In order to distinguish between the correct and faulty response of the cell under test to the vector \( \overline{3} \), the inputs to the next and subsequent cells along the \( S^1 \) signal direction must be \( S^2, S^3 = 0, \) or \( 1, 1 \). Examination of Table 1 shows that these are the only input combinations which propagate the fault through a cell to the next \( S^1 \) value. These inputs with the correct \( S^1 \) input give the vectors \( S \) or \( Z \). The possible vector propagations which establish a fault propagation path along each signal direction are shown in Figure 3 which we term fault effect propagation graphs. Note that whereas for test propagation we require solutions which are compatible for each signal direction represented by the propagation graphs, in propagating a fault effect to an array output it is sufficient to propagate the fault effect from each cell output in one direction only. Indeed we may be able to propagate the fault effect out through a mixture of directions as we show in a later example. However, efficient testing is dependent on efficient fault effect propagation as much as an efficient test vector propagation; for example the advantages of C-testability will be eroded if the fault effects cannot also be propagated in compatible cycles.

**Test Sets**

The problem of combining vectors into test sets such that the required test vectors are applied to every cell in the array and the fault effects are propagated to the array boundary can be solved by using the DDG along with the propagation graphs. It can be expressed as a pattern matching problem in which each node of the DDG is instantiated with specific vectors which simultaneously satisfy the constraints of the propagation graphs.

We term the resulting graph a Test Vector Dependence Graph (TVDG) [6]. For example consider the TVDG of Figure 4 where the vectors \( V_1, V_2, \) and \( V_3 \) are defined and the choice of vector \( V_4 \) must be determined. From the TVDG \( V_4 \) is constrained by the signals \( S^a, S^b, \) and \( S^c \) and is a member of the set of vectors which obeys the propagation of \( V_3 \) along signal \( S^a \), the propagation of \( V_2 \) along signal \( S^b \) and the propagation of \( V_1 \) along signal \( S^c \).

We wish the test sets to be efficient by which we mean that we aim at 100% fault coverage in the minimum time. Therefore we try to propagate test vectors in cycles where more than one cell is tested by the vectors of a test set. Consider the array of Figure 1 as an example and the functional fault model which requires the application of all vectors to every cell. Examining the test vector propagation graphs of Figure 2, the only vectors which can be included in compatible cycles for test vector application are 0, 1, and \( S \). By looking at Figure 3 we see that of these, only vector \( S \) can be included in a fault-effect propagation cycle. Therefore only the vector \( S \) can be included in a cycle (self propagating) which will test every cell at one application.

The other test vectors will have to be included in linear test sets whose size and test time are dependent on the size of the array.

**Vector Propagation Cycle** → **Test** → **Fault Effect Propagation Cycle**.

Where the vector propagation cycle or "flushing vectors" propagate the required test to every cell in the array and the fault effect propagation cycle propagates the result to the boundary of the array where any fault can be detected. An example of a linear test set which propagates test vectors \( 4 \) and \( 7 \) to the 3rd cycle;

\[ S \rightarrow S \rightarrow 4 \rightarrow 7 \rightarrow 1 \rightarrow 1 \]

where \( S \) is the "flushing vector" and \( 1 \) is the fault effect.

**Boundary Restrictions**

A critical and often overlooked assumption made in all the work on testing regular arrays is that there are no constraints on the data access at the boundary of the array i.e. all data lines at the array boundary are completely controllable and observable. This is not always the case and we consider here the effect of boundary restrictions on array inputs and array outputs.

**Array Input Boundary Restrictions**

There are two possible consequences of restrictions on the inputs at the boundary of the array. The first is the addition of another signal direction in the DDG to model the constraint. The second is the creation of a boundary condition in the DDG with the boundary cells only affected.

**Additional signal restrictions.** This type of restriction will occur when a data value is stored in the cell and cannot be altered on every clock cycle or data is entered to the array through a register.
loading operation as shown in Figure 5. This restriction can be modeled by an additional signal in the J direction of the DDG since the data value is kept constant for a specific time until a

**DDG boundary restrictions.** This type of restriction is very common where a data input at the boundary is tied to a specific logic value. For example in the case of the array of Figure 1 the signal SI represents a partial result accumulation and may be tied to logic 0 at the input to the first cell. This is modeled in the DDG by tying the data inputs at the specific boundary cells to the appropriate logic value and the vector propagations are then constrained implicitly. With SI tied to logic 0 for the first cell, the vectors applicable to it are 0, 1, 2, and 3. Examining the propagation graph for signal SI (Figure 2) shows that applying these vectors to the first cell still allows all other possible vectors to be applied to the next cell.

**Array Output Boundary Restrictions**

In general the only array outputs that are sure to be observable are the computational results. Therefore we are often restricted in the choice of fault propagation paths to those that propagate a fault effect to the observable computation outputs. For example in the case of the array of Figure 1, SI may be the only observable output signal and a fault propagation path must be established from the SI and S2 outputs of the cell under test to the array S1 output. The vector propagation constraints of Figure 3 establish a fault propagation path along the SI signal direction. The vector propagation constraints of Figure 6 establish a fault propagation path from the S2 output of the cell under test to the SI array output along the SI signal direction.

**Correlator Example**

We present here a case study to illustrate the test strategy. It is based on a commercially available correlator consisting of a two dimensional bit-level systolic array [7, 8]. The correlator computes the correlation y(k) between 64 reference coefficients a(i) and a data sequence x(k+i), expressed mathematically as:

\[
y(k) = \sum_{i=0}^{63} a(i) \cdot x(k + i)
\]

The array is a 64 stage bit slice with 4-bit data and 1-bit reference. The result can grow up to 10 bits so that the array consists of 10 rows by 64 columns of single bit multiply/accumulate cells. With 1-bit reference data then the function of each processor is the gated addition of a data word x(j) to the partial result y. Each bit-level cell (Figure 7) performs a gated full-addition and then propagates 4 data streams (x: input data, a: reference data, c: carry to more significant bit, y: partial result) to the neighboring elements in a pipelined manner. A common 2 phase clock controls the flow of data between neighboring cells. Each cell is active on one of these phases. A processor active on one phase will have its nearest neighbour active on the alternate phase.

The testing "goal" is to apply all the vectors for testing a single cell, as required by the fault model, to every cell in the array. The test vector propagation method must satisfy the requirements of propagating all fault effects, such that they can be detected.

**Fault Model**

At the array level a single faulty cell is assumed but the process of array rather than cell verification is used. In the cell all the possible faults are modeled using the functional fault model. Thus all 16 possible input combinations must be applied to test the cell and exhaustively verify the truth table. The fault effects of this model are rather general in that a fault can manifest itself as any alteration to the truth table.

**Data Flow**

The array has a bilateral data flow in the horizontal direction with the input data x and the partial result y flowing in different directions. Because the array is two dimensional, the data dependence graph (Figure 7) has three dimensions but since the reference coefficient a and the carry c have the same direction
and speed, they can be represented as a single direction. The direction of the DDG represents the sequence of data input to the array. The i direction represents the flow of data along each row in the array and the k direction represents the flow of data down each column. Since 2 phase clocking is used with each processor active on alternating clock cycles then time increases from node to node in half cycles along the i and k directions and in full clock cycles along the j direction.

![Propagation Graph for y](image)

![Propagation Graph for x](image)

![Propagation Graph for a](image)

**Figure 7.** Correlator Cell and DDG

**Test Vector Propagation Functions**

The truth table for the correlator cell is given in Table 2. The propagation graphs for each distinct signal directions are given in Figure 8. Examining the graphs reveals that all the vectors can be included in cyclic test sets in the row direction i.e. the data input x and result y signal directions. In the column signal direction the vectors 2 and A cannot be included in cycles and a linear test set using the vectors 3 and B as flushing vectors is required.

**Table 2.** Correlator Truth Table

<table>
<thead>
<tr>
<th>State</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>x a c y</td>
<td>\hat{y} \hat{y}</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 1 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>3</td>
<td>0 1 1 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>4</td>
<td>1 0 0 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>5</td>
<td>0 0 1 1</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>6</td>
<td>0 1 1 1</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>7</td>
<td>1 0 0 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>8</td>
<td>1 0 1 1</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>9</td>
<td>1 1 0 0</td>
<td>1 1 0 1</td>
</tr>
<tr>
<td>A</td>
<td>1 1 0 1</td>
<td>1 1 1 0</td>
</tr>
<tr>
<td>B</td>
<td>1 1 1 0</td>
<td>1 1 1 0</td>
</tr>
<tr>
<td>C</td>
<td>1 1 1 1</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

**Boundary Restrictions**

A block diagram of the correlator is shown in Figure 9. The following restrictions on access to the boundary of the array exist.

1. The input data x is a 4 bit word with the remaining 6 bits given by sign extension of the most significant bit.
2. The carry input c to the top row of the array is tied to logic 0.
3. The reference coefficients a at the top of array are supplied to the array through a shift register.
4. The input to the partial results path, y, is tied to logic 0 at the left hand side of the array.

5. Access is only available to 4 bits of the data x after it has passed through the array.

6. Access to the correlation result y is through a barrel shifter and an adder circuit (for accumulating multiple chip results).

7. No access is available to the carry value of the input data from row to row, but for the remaining restrictions 5-7 affect the propagation of any fault effect.

Restrictions 1-4 affect the propagation of test vectors and restrictions 5-7 affect the propagation of any fault effect.

Restrictions on the carry inputs. The carry input c to the first row of the array is tied to logic 0 and this places a constraint on the vectors which can be applied. Only 2 out of 4 possible combinations of the reference data a and the carry in c are therefore possible, allowing only 8 out of the 16 vectors to be applied (0, 1, 4, 5, 8, 9, C, D). Examining the ac propagation graph of Figure 10 shows that 12 of the 16 vectors can be applied to row 2 (0, 1, 4, 5, 6, 7, 8, 9, C, D, E, F). In fact the 4 remaining vectors can never occur in the array with this restriction on the carry c. Since the test "goal" is the verification of the array under the functional fault model, then only those applicable vectors need be applied to each cell and we can satisfy our goal without applying the 4 vectors 2, 3, A and B.

Restrictions on the reference coefficient inputs. The reference data is loaded into the correlation array via a holding register. This consists of a serial shift register through which the reference data passes a(0) first. A similar shift register is used to gate the reference into the array by passing a single enable bit in the opposite direction to the reference data. As a reference bit meets the enable bit, it is passed into the reference register and thereafter this value is retained as the input to the array. This restriction is modelled by the addition of the signal a in the j direction of the DDG since the register ensures that the value of a is kept constant until a new enable signal is applied.

Restrictions on the partial result. At the right hand column of the array the partial results input y is tied to logic 0. This allows only 8 out of 16 vectors to be applied to the first column (0, 1, 2, 4, 6, 8, A, C, E). Examining the y propagation graph of Figure 10 shows that the remaining 8 vectors can be propagated to column 2 and the rest of the array.

Restrictions on access to array outputs. The only observable output for the lower 6 rows of the array is the result y. Therefore a fault propagation path must be established from the cell outputs to the Y output. Applying any vector establishes the path for a fault on the y and a cell outputs. For the y and a cell outputs the fault propagation graphs of Figure 10 show the constraints on applicable vectors. Since the array output y is only observable through the barrel shift and adder circuits then the inputs to these circuits must be set to establish a fault propagation path through them.

**Figure 9. Correlator Block Diagram**

**Restriction on the data input.** The input data x is a 4 bit input with sign extension of the remaining bits. Therefore for the first four rows of the array there is complete freedom in the choice of value of the input data from row to row, but for the remaining rows there is a restriction since the value of the input data is determined by the value applied to row 4. Therefore in the column direction the value of x on successive clock phases is constant. This restriction can be modelled by the addition of the signal x in the column direction (k on the DDG). This additional signal has the same speed and direction as c and a and the propagation graph for that direction must therefore be altered.

**Restrictions on the carry inputs.** The carry input c to the first row of the array is tied to logic 0 and this places a constraint on the vectors which can be applied. Only 2 out of 4 possible combinations of the reference data a and the carry in c are therefore possible, allowing only 8 out of the 16 vectors to be applied (0, 1, 4, 5, 8, 9, C, D). Examining the ac propagation graph of Figure 10 shows that 12 of the 16 vectors can be applied to row 2 (0, 1, 4, 5, 6, 7, 8, 9, C, D, E, F). In fact the 4 remaining vectors can never occur in the array with this restriction on the carry c. Since the test "goal" is the verification of the array under the functional fault model, then only those applicable vectors need be applied to each cell and we can satisfy our goal without applying the 4 vectors 2, 3, A and B.

**Restrictions on the reference coefficient inputs.** The reference data is loaded into the correlation array via a holding register. This consists of a serial shift register through which the reference data passes a(0) first. A similar shift register is used to gate the reference into the array by passing a single enable bit in the opposite direction to the reference data. As a reference bit meets the enable bit, it is passed into the reference register and thereafter this value is retained as the input to the array. This restriction is modelled by the addition of the signal a in the j direction of the DDG since the register ensures that the value of a is kept constant until a new enable signal is applied.

**Figure 10. Fault Effect Propagation Graph**
following will be required: (1) access to the A output, this need not involve an extra 64 connections, but could use an appropriately configured AND-OR chain as suggested in Reference [9]; (2) a change to the sign extension circuit to allow different values of x to be propagated eliminating the need for x in the k direction of the DDG; or (3) the use of a more specific fault model at the cell level - for example if a "stuck-at" fault model is assumed, stuck-at faults on the a data line can be detected by the vectors 8, 9, 2, 4, 6, 5 and 7.

Due to the restrictions only the vectors 8 and 9 can be included in cycles in all signal directions. Therefore linear test sets of the type described in section 6 will be required.

This paper has presented a unified approach to the testing of fine grained arrays and has shown that the often overlooked effects of fault propagation and boundary value restrictions can have a significant impact on the testing problem. We have illustrated the kind of problems that can arise and have shown the way our test approach handles them. We are currently refining CAD tools to exploit this approach and give more rapid generation of test vectors. Besides the ability to generate test vectors for a specified design and/or to pinpoint critical difficulties in test vector or fault effect propagation as we have discussed, the availability of rapid test generation allows the designer to make rational decisions about the need to improve the boundary access. For example he can evaluate the effectiveness of improving the accessibility (eg. to the carry inputs or the partial results of the correlator array) in terms of reducing the test time.

Acknowledgement

This work has been undertaken as part of the UK Alvey project ARCH 013 whose support is gratefully acknowledged. We are also grateful to GEC and Plessey for permission to publish the material.
References


