REDUCING TEST APPLICATION TIME THROUGH TEST SUB-PATTERN RE-USE

Kwame Osei BOATENG
Faculty of Electrical and Computer Engineering, College of Engineering
Kwame Nkrumah University of Science and Technology, Kumasi, Ghana
TEL.: +233(0)203154862 FAX: +233(0)322063730
boat.soe@knust.edu.gh

Kwasi Adu-Boahen OPARE
Network Operations Centre, University Information Technology Services
Kwame Nkrumah University of Science and Technology, Kumasi, Ghana
TEL.: +233(0)208176760 FAX: +233(0)322063730
opare@knust.edu.gh

Abstract: *Test application time is a major factor of the cost associated with the scan technique. On one extreme test application time can be drastically reduced by using built-in self-test (BIST). However, the quality of random test pattern used by BIST is low. On the other extreme, automated test equipment based testing using quality automatic test pattern generation is very time-consuming. Research in the area of economics of test application is directed towards finding a good blend of the two extremes to arrive at a workable optimal hybrid. A novel method for achieving this hybrid is presented in this paper. The approach used consists of a modification of the scan architecture and the development of a matching algorithm. The modified scan architecture involves the addition of a multiple-input signature register (MISR) to the scan chain. This way test responses are not captured in the scan chain but, rather, are compressed in the MISR. The proposed algorithm then takes advantage of repeated patterns in test vectors, by rearranging them such that those with matching patterns are adjacent to each other. An illustrating of how the algorithm works is also presented.*

Key words: *digital circuit testing, scan technique, test application time, multiple-input signature register, pattern matching algorithm.*

1. Introduction.

The Scan Technique is a design for testability (DFT) method used to make sequential circuits easily testable. This is achieved by introducing flip-flops (latches) into the original design of the circuit. These flip-flops are designed and connected such that the circuit operates in two modes; normal and test modes [1, 2]. With such an arrangement, tests are applied to inaccessible lines in sequential circuits by shifting them into the flip-flops.

The use of the scan technique requires that every bit in the test vectors is scanned-in one clock cycle. As a result, test application time increases in proportion to an increase in the number of test vectors and the number of bits per test vector.

Several solutions exist to minimise the test application time in the scan technique. One of such solutions is the use of multiple scan chains [3; 4]. Ordering of scan chains to obtain an optimal arrangement, is another method for minimising test application time. Although, Narayanan *et al.*[5] shows the ordering of registers in a scan chain to be a computationally complex problem, Narayanan[6] has shown that this is possible. The BIST-Aided Scan Test (BAST)[7] is also another method. Aside reducing test application time, it also reduces test data size.

Other techniques for minimising test application time include the use of reconfigurable scan chains [8, 9] and another one presented in Abramovici *et al.*[10]. The latter presents an approach that reduces the number of vectors for which bits are shifted into and out of scan flip-flops. It takes advantage of the possibility of grouping a large number of possible faults into sets with the following properties.

a. Each fault in a set has one or more test vectors that propagate the effects of those faults to the primary output of the circuit-under-test.
b. The state input parts of those test vectors are compatible with those of the vectors for the other faults within the set.

This paper presents a new approach to minimise test application time in scan based circuits. It is called the Pattern Match Approach. It consists of a modification of the scan architecture and an algorithm called the Pattern Match Algorithm (PMA). The algorithm reduces the number of bits in a test vector that are shifted into and out of the scan flip-flops.

In the next section the modified architecture is explained together with how it can be exploited to reduce the test vector scan-in time. Next, the algorithm is presented in Section 3, followed by an illustration (in Section 4) of its application. The paper ends with a conclusion.

2. Scan architecture modification

In the scan architecture, the flip-flops are arranged such that test vectors are scanned into them during the test mode. During normal mode, the scanned-in vectors, together with those from the primary inputs are applied to the circuit under test (CUT). The response is then captured in the flip-flops. In this process, the flip-flops that hold the scanned-in vectors are the same ones that capture the response. Therefore, the scanned-in vectors are cleared by the response from the CUT. This attribute of the scan architecture does not allow the re-use of patterns in a test vector that is already scanned in. As a result, the whole of the next test vector must be scanned in during the test mode.

According to Jha and Gupta[11], for a sequential circuit with m state inputs, if n vectors are needed to test for certain faults, the number of clock cycles taken to scan them in is shown in Equation (1). In addition, it takes the number of clock cycles shown in Equation (2) to scan the last response out. Putting all together, the total number of clock cycles required to apply n test vectors to a circuit with m state inputs is shown in Equation (3).

$$C_{in} = n(m+1) \tag{1}$$
$$C_{out} = m - 1 \tag{2}$$
$$C_{clk} = C_{in} + C_{out} = n(m+1) + m - 1 \tag{3}$$

It can be deduced that if test vectors are scanned in during $\tilde{m}$ instead of m clock cycles, where $\tilde{m} = m - \epsilon$, then savings $(n\epsilon)$ in test application time can be achieved as shown in Equations (4) and (5).

$$\tilde{C}_{in} = n(\tilde{m}+1) \tag{4}$$
$$\tilde{C}_{in} = n\big((m-\epsilon)+1\big) = C_{in} - n\epsilon \tag{5}$$

However, any reduction in the clock cycles required for the scan-in of a given test vector will leave part of the response to the previous test vector in the scan chain. This problem must be overcome in order to develop algorithms for minimising test application time based on the re-use of common patterns in test vectors. This can be dealt with by modifying the scan architecture, as shown in Figure 1.

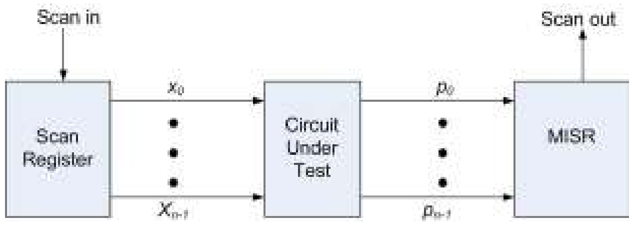

Figure 1: Scan Architecture used in the approach

The modified scan architecture consists of the circuit under test (CUT) with scan circuitry connected to its input. Its output is also connected with a multiple-input signature register (MISR) that has a serial output. The MISR is used to compress the response from the CUT into a signature. The signature is then scanned out at the end of the testing process through the serial output. It can be observed from the figure that only the MISR found in the traditional BIST architecture is incorporated. The pseudorandom binary sequence generator (PRBSG) is not used. Therefore, instead of pseudorandom patterns, deterministic vectors, pre-processed by an algorithm, are scanned into the CUT.

The addition of the MISR in the proposed architecture enables the separation of the response vectors from the scanned-in vectors. Whilst a response vector is captured by the MISR, the scanned-in vector stays in the scan flip-flops to be re-used. This arrangement makes it possible for algorithms, that pre-process test vectors prior to scan-in, to be developed. The pre-processing of vectors enables scan-in time to be minimised, thus, affecting the over-all test application time.

3. The pattern match algorithm.

The algorithm is made up of the Initialising Pivot Procedure, the Pivot Cycle Procedure and the Compare function. All these modules are used in the Main Procedure that implements the algorithm.

3.1. Terms and Variables used in the Algorithm

Initializing vector is the test vector used to set the output of a logic gate to the complement of the logic value expected when testing a fault under a fault model (such as the stuck-open fault) that requires a test pair. It also ensures that the complement logic is propagated to an observable output of the circuit under test. It is always the first vector of a test pair, however, note that there can be a situation where the second vector of a given test pair is also the first vector of the next test pair. An initializing vector is identified with an attribute *i*.

Dependent vector is the second vector of a test pair. A dependent vector is identified with an attribute *d*. If test vector t_k is the second vector of a given test pair and also the first vector of the next test pair, then t_p is both dependent and initializing.

Independent vector is a test vector that constitutes a complete test for a particular fault under a given fault model. An independent vector does not need any initializing vector.
Pivot vector is a vector that serves as the basis of comparison for other vectors.

Starting Pivot Vector is the first test vector to be used as a pivot vector in a cycle of the proposed algorithm.

Pivot Cycle: This refers to the period from the beginning of usage of a pivot vector till when a new one is selected.

$\boldsymbol{A} \equiv [T_j]$, is an $m \times n$ matrix made up of binary elements, $[b_{i,j}]$ that form part of test vectors.

$\boldsymbol{A}_{(p)}$ has the same size and test vectors as in matrix A. The starting pivot vector of $A_{(p)}$ is the vector at column *p* of A.

$\boldsymbol{T_k} = [b_{i,k}]$ of A, for $0 \leq i \leq m-1$ and a constant *k*, at any time**.** It serves as space holder for test vectors—the value of any $b_{i,k}$ depends on the test vector held at T_k and the column operations that have been performed in the current pivot cycle.

$\boldsymbol{t_{k+1}}$ is the identifier of the test vector held at column T_k of A. At any instant of algorithm progression, t_{k+1} may occupy columns other than the *k*-th.

NB: In the algorithm, the name of the column a test vector currently occupies (instead of the actual identifier of that test vector) is used in a generic sense to refer to the test vector.

$\boldsymbol{T_k^{(r)}} \subset T_k$, where r signifies the number of least significant bits (elements) that have been shifted out of T_k. In other words, $T_k^{(r)} \subset T_k$, where $T_k^{(r)} \equiv [b_{i,j}]$, $r \leq i \leq m-1$. Thus $\mathrm{T}_k^{(0)} \equiv \mathrm{T}_k$.

$\boldsymbol{T_k^{(-r)}} \subset T_k$, where $T_k \equiv [b_{i,k}]$, and $0 \leq i \leq m-r-1$.

Skip, *S*: This signifies the length of a matched pattern in two test vectors that must be adjacent to each other (i.e. they must occupy columns *k* and *k+1*) according to the proposed algorithm. It specifies the number of elements in the second of the two test vectors that should be skipped by the ATE (and hence the number of clocked cycles saved) during the scan-in. It is calculated as an integer attribute s, where $s = m - r$.

$\boldsymbol{\tilde{A}_{(p)}}$ is the resultant matrix, or the optimum arrangement for minimising test application time when the test vector $\boldsymbol{t_{p+1}}$ is used as the starting pivot vector. For each *p*, $\tilde{A}_{(p)}$ is associated with an integer attribute ***Ssum*****$_p$, the sum of skips of all test vectors in $\tilde{A}_{(p)}$.

k is the variable used to mark the column occupied by the current pivot vector.
x is the variable that marks the relative position of the column occupied by a test vector, with respect to the column occupied by the pivot vector.
y is the variable that represents the number of dependent vectors directly following an initialising vector.

3.2. The Compare Function

The Compare function takes two arguments1, $T_k^{(r)}$ and $T_{k+x}^{(-r)}$. If no match is found between the arguments, the function returns *false* otherwise it returns *true*. A match exists if,

1 The two arguments are $T_{(k+x+y-1)}^{(r)}$ and $T_{(k+x+y)}^{(-r)}$ when the Compare function is called in an Initialising Pivot Procedure

$\forall b_{i,k}$ in $T_k^{(r)}$, where $r \leq i \leq m-1$, and

$\forall b_{j,k+x}$ in $T_{k+x}^{(-r)}$, where $0 \leq j \leq m-r-1$,

$b_{i,k} = b_{j,k+x}$ or $b_{j,k+x}$ is a don't-care.

3.3. The Main Procedure

This is the *main* procedure of the proposed algorithm. In the algorithm all rotations are counter-clockwise. The following steps describe the flow of the function.

```
Step 1:    START
 1. Set $p = 0$.
Step 2:
 1. Is $p < n$?
   a.      If NO,
        i.  select the resultant matrix,
            $\ddot{A}_{(p)}$ with the lowest value p
            having the highest value of
            $Ssum_p$. The selected $\ddot{A}_{(p)}$ is an
            optimum matrix for reducing
            test application time for the
            test set.
       ii.  Go to step 9.
Step 3:
 1. Is $T_p$ a dependent vector?
   a.      If YES,
        i.  set $Ssum_p$ to null,
       ii.  increment p, and
      iii.  go to Step 2.
Step 4:
 1. Select vector $T_p$.
 2. Associate skip, $S = 0$ with the
    selected test vector, and
 3. initialise variables as follows:
    $k = p, r = 0, x = 0 \text{ and } y = 0$.
Step 5:
 1. Is $T_p$ an initialising vector?
   a.      If YES,
        i.  perform the Initialising
            Pivot Procedure.
Step 6:
 1. If $k > 0$,
   a.      rotate vectors from column 0
    to $k+y$, $y+1$ times horizontally.
   b.      Set $k = y$.
   c.      Increment x, and
   d.      set $y = 0$.
Step 7:
 1. Is $k < n$-1?
   a.      If YES,
        i.  perform the Pivot Cycle
            Procedure, and
       ii.  repeat Step 7.
Step 8:
 1. Compute $Ssum_p$,
 2. associate result with the resultant
    matrix, $\ddot{A}_{(p)}$
 3. increment p, and
 4. go to Step 2.
Step 9:    END
```

3.4. The Initialising Pivot Procedure

The Initialising Pivot Procedure is called when the newly found/selected pivot vector is an initialising vector. When the pivot vector is tested and found to be initialising, this procedure is executed according to the following steps.

```
Step 1:   START
Step 2:
 1. Increment y, and
 2. set $r = 0$.
Step 3:
 1. Is $r < m$? (Are there still some
    elements in the initialising pivot
    vector yet to be shifted out?).
   a.      If NO,
       i.  set $S = 0$, and associate S
           with $T_{k+x+y}$
      ii.  go to Step 6.
Step 4:
 1. Perform the Compare function with
    $T_{k+x+y-1}^{(r)}$ and $T_{k+x+y}^{(-r)}$ as the arguments.
   a.      If the function returns
     false,
       i.  increment r, and
      ii.  go to Step 3.
Step 5:
 1. Compute skip, S, and associate it
    with $T_{k+x+y}$.
Step 6:
 1. Is $T_{k+x+y}$ an initialising vector?
   a.      If YES,
       i.  go to Step 2.
Step 7:   END
```

3.5. The Pivot Cycle Procedure

The Pivot Cycle Procedure is called when the stage has been set for a fresh sequence of comparisons to be carried out after selecting a new starting pivot vector. The following steps are performed in the procedure.

```
Step 1: START
  1. Set $r = y = 0$.
Step 2:
  1. Is $r < m$?
    a.  If NO,
      i.   set $x = 1$
    ii.   set $r = m$, and
    iii.  go to Step 8.
Step 3:
  1. Is $k + x > n$? (In other words, has
     the last vector been compared
     with the pivot vector?)
     a. If YES,
        i.   set $x = 1$,
        ii.  increment $r$, and
        iii. go to Step 2.
Step 4:
  1. Is $T_{(k+x)}$ (the vector being
     compared with the pivot vector)
     a dependent vector?
     a. If YES,
        i.   go to Step 7.
Step 5:
  1. Perform the Compare function
     with $T_k^{(r)}$ and $T_{k+x}^{(\ell-r)}$ as the
     arguments.
Step 6:
  1. Did the execution of the Compare
     function in step 5 return true?
     a. If YES,
        i.   go to Step 8.
Step 7: .
  1. increment $x$, and
  2. go to Step 3.
Step 8:
  1. Compute skip, $S=m-r$, and
  2. associate it with $T_{k+x}$.
Step 9:
  1. Is $T_{k+x}$ an initialising vector?
     a. If NO,
        i.   go to Step 11.
Step 10:
  1. and
  2. perform Initialising Pivot
     Procedure.
Step 11:
  1. If $x > 1$,
     a. Rotate vectors from $k + 1$
        through $k + x + y$, $y + 1$ times
        horizontally.
Step 12:
  1. $k = k + y + 1$,
  2. $x = 1$, and
  3. $y = 0$.
Step 13:     END
```

3.6. The Test Application Process

When a test vector is about to be scanned in, the Automated Test Equipment (ATE) (under test program control) reads the skip value of the vector to determine the number of rows that must be skipped (i.e. the number by which to reduce the number of clock cycles per vector). A skip value of s, tells the ATE to skip the first s bits of the vector and scan in the rest (thereby resulting in $m - s$ shift-ins). Simultaneously the binary data within the scan register is shifted up just enough to accommodate the incoming $m - s$ bits of binary data. The new test vector within the scan register is then applied to the CUT and the response captured by the MISR. In the case of test pairs, the MISR captures the responses produced by the second half-pairs. After all test vectors have been applied, the compressed response (signature) is scanned out of the MISR and compared with the signature of a fault-free circuit.

4. An illustration of application of the algorithm

In this illustration, the twelve test vectors in Table 1 are used. All bits of each vector are assumed to be applied to state inputs of the circuit under test. *Labels* row shows the attributes of vectors that are either initializing or dependent. The number of rows in the *Vectors* section of the table represents the number of elements, *m*, in a test vector. In this test set *m=6*. The number of columns represents the number, *n*, of test vectors. (*n=12*). The test set comprises of test vectors for faults in two fault models (one model requires test pairs and the other requires single test vectors). This mix was chosen to fully demonstrate the complete effect of the proposed algorithm.

4.1. Executing the Algorithm

The Main Procedure starts with the first vector ($T_{p=0}$) in the given matrix, *A* (Table 2) as a valid starting pivot vector, since T_0 is not dependent. The variable k is set to p and so $k = 0$. The variables *x*, *y* and *r* are each set to zero. Thus, $x = y = r = 0$. A skip, $S = 0$ is assigned to T_0, as shown in Table . The vector T_0 is not an initialising vector so the Initialising Pivot procedure is not executed. Since $x \ngtr 1$, no rotation is done. The variable *k* remains unchanged since $y = 0$. Variable *x* is incremented (*x=x+1*). At this stage, $k < (n - 1)$, so the Pivot Cycle procedure is called.

Table 1: Test Vectors for all Targeted Faults (Matrix A)

Test ID	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
Label		i	d/i	d				i	d		i	d
Ve ct	0	1	1	0	1	1	0	1	1	0	0	0
	1	0	0	0	1	0	1	1	0	0	1	0

	1	1	0	0	0	0	0	0	0	0	1	0
	0	0	0	1	1	1	0	1	0	1	0	1
	0	0	1	0	1	1	0	1	1	0	1	0
	1	0	0	0	1	0	0	1	0	1	0	0

Table 2: Matrix $A_{(0)}$

T_k	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
Test ID	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
Label		i	d/i	d				i	d		i	d
Skip	**0**											
Vectors	**0**	1	1	0	1	1	0	1	1	0	0	0
	1	0	0	0	1	0	1	1	0	0	1	0
	1	1	0	0	0	0	0	0	0	0	1	0
	0	0	0	1	1	1	0	1	0	1	0	1
	0	0	1	0	1	1	0	1	1	0	1	0
	1	0	0	0	1	0	0	1	0	1	0	0

Table 3: Starting Pivot shifted up by 1 bit

T_k	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
Test ID	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
Label		I	d/i	d				i	d		i	d
Skip												
Vectors	1	1	1	0	1	1	0	1	1	0	0	0
	1	0	0	0	1	0	1	1	0	0	1	0
	0	1	0	0	0	0	0	0	0	0	1	0
	0	0	0	1	1	1	0	1	0	1	0	1
	1	0	1	0	1	1	0	1	1	0	1	0
	■	0	0	0	1	0	0	1	0	1	0	0

4.2. Executing the Pivot Cycle Procedure with T_0 as the Pivot Vector

In the Pivot Cycle procedure, each of the variables *r* and *y* is set to zero. Thus the condition $r < m \quad (m = 6)$ is true. $k + x \ngtr n$ and T_{k+x} (T_1) is not a dependent vector and so the Compare function is executed with $T_0^{(r)}$ and $T_1^{(-r)}$ as arguments. For $r = 0$, there is no match; thus *x* is incremented to 2. The execution of the Compare function, with T_0 and T_2 as arguments also returns false. The process of incrementing *x* and executing the Compare function continues—ignoring all dependent vectors, i.e. vectors T_2, T_3, T_8 and T_{11} are not used when encountered. At this stage T_0 is shifted up by one bit, thereby incrementing *r* to 1, as shown in Table 3 . The variable *x* is re-set to 1, and the Compare function is executed with the arguments $T_k^{(1)}$ and $T_{k+x}^{(-1)}$, where the latter argument takes values from T_1 to T_{11}. This also does not lead to a match so r is incremented to 2 and *x* is reset to 1. With $r = 2$ and $x = 5$, T_5 is the second argument and the Compare function returns true (see Table 4). A skip, $S = 6 - 2 = 4$ is assigned to t_6 and since t_6 is not an initialising vector, vectors T_1 up to T_5 are rotated once horizontally. This brings the vector t_6 to column 1, as shown in Table 5. Variable *k* is updated to 1, and variables *x* and *y* are also reset to 1 and 0, respectively. The Pivot Cycle procedure ends.

Table 4: A Match Found at T_5 after a 2-bit shifting

T_k	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
Test ID	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
Label		i	d/i	d				i	d		i	d
Skip	0					**4**						
Vectors	**1**	1	1	0	1	**1**	0	1	1	0	0	0
	0	0	0	0	1	**0**	1	1	0	0	1	0
	0	1	0	0	0	**0**	0	0	0	0	1	0
	1	0	0	1	1	**1**	0	1	0	1	0	1
	■	0	1	0	1	1	0	1	1	0	1	0
	■	0	0	0	1	0	0	1	0	1	0	0

Table 5: The Second Pivot Vector

T_k	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
Test ID	t_1	t_6	t_2	t_3	t_4	t_5	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
Label			i	d/i	d			i	d		i	d
Skip	0	4										
Vectors	0	**1**	1	1	0	1	0	1	1	0	0	0
	1	**0**	0	0	0	1	1	1	0	0	1	0
	1	**0**	1	0	0	0	0	0	0	0	1	0
	0	**1**	0	0	1	1	0	1	0	1	0	1
	0	**1**	0	1	0	1	0	1	1	0	1	0
	1	**0**	0	0	0	1	0	1	0	1	0	0

4.3. Executing the Pivot Cycle Procedure with T_1 as the Pivot Vector

Since k<(n-1), the Pivot Cycle procedure is executed again with T_1 as the pivot vector. With $T_1^{(r)}$ and $T_{1+x}^{(-r)}$ as arguments for the Compare function, no match is found for $r = 0$, 1. T_{10} matches T_1 when $r = 2$ and $x = 9$ (see Table 6). A skip, $S = 4$ is assigned to t_{11}, and since T_{10} is an initialising vector, Initialising Pivot procedure is invoked.

Table 6: A Match Found at T_{10} after shifting T_1 2 bits up

T_k	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
Test ID	t_1	t_6	t_2	t_3	t_4	t_5	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
Label			i	d/i	d			i	d		i	d

Skip	0	4									4	
Vectors	0	**0**	1	1	0	1	0	1	1	0	**0**	0
	1	**1**	0	0	0	1	1	1	0	0	**1**	0
	1	**1**	1	0	0	0	0	0	0	0	**1**	0
	0	**0**	0	0	1	1	0	1	0	1	**0**	1
	0	■	0	1	0	1	0	1	1	0	1	0
	1	■	0	0	0	1	0	1	0	1	0	0

4.3.1. Executing the Initializing Pivot Procedure for T_{10}

The Initialising Pivot procedure starts by incrementing y to 1 and setting *r* to 0. Since $r < m$, the Compare function is executed with $T^{(r)}_{k+x+y-1}$ and $T^{(-r)}_{k+x+y}$ (i.e. $T^{(r)}_{10}$ and $T^{(-r)}_{11}$ where T_{11} is the dependent vector of T_{10}) as arguments. No match is found. Variable *r* is also incremented to 1, by shifting up the pivot vector by 1. The Compare function is executed again. This also does not produce a match. A match is, however, found when $r = 5$ (in Table) and the dependent vector, t_{12} (T_{11}), is assigned a skip, $S = 6 - 5 = 1$. Since t_{12} (T_{11}) is not an initialising vector, the Initialising Pivot procedure ends.

Still in Pivot Cycle Procedure, vectors T_2 to T_{11} are rotated $y + 1 = 2$ times horizontally to bring t_{11} to column 2, followed by t_{12} in column 3, as shown in Table 8. Variable *k* is updated to 3 whilst *x* and *y* are reset to 1 and 0, respectively. The Pivot Cycle procedure ends.

Table 7: Comparing Initialising and Dependent Vectors (T_{10} and T_{11})

T_k	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
Test ID	t_1	t_6	t_2	t_3	t_4	t_5	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
Label			i	d/i	d			i	d		i	d
Skip	0	4									4	1
Vectors	0	**0**	1	1	0	1	0	1	1	0	**0**	**0**
	1	**1**	0	0	0	1	1	1	0	0	■	0
	1	**1**	1	0	0	0	0	0	0	0	■	0
	0	**0**	0	0	1	1	0	1	0	1	■	1
	0	■	0	1	0	1	0	1	1	0	■	0
	1	■	0	0	0	1	0	1	0	1	■	0

Table 8: T_3 as Pivot Vector

T_k	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
Test ID	t_1	t_6	t_{11}	t_{12}	t_2	t_3	t_4	t_5	t_7	t_8	t_9	t_{10}
Label			i	d	i	d/i	d			i	d	
Skip	0	4	4	1								
Vectors	0	1	0	**0**	1	1	0	1	0	1	1	0
	1	0	1	**0**	0	0	0	1	1	1	0	0
	1	0	1	**0**	1	0	0	0	0	0	0	0
	0	1	0	**1**	0	0	1	1	0	1	0	1
	0	1	1	**0**	0	1	0	1	0	1	1	0
	1	0	0	**0**	0	0	0	1	0	1	0	1

4.4. Executing the Pivot Cycle Procedure with T_3 as the Pivot Vector

Since $k < (n-1)$, the Pivot Cycle procedure is executed again with T_3 as the pivot vector. Ignoring all dependent vectors, for example T_6, the Compare function with $T^{(r)}_3$ and $T^{(-r)}_{3+x}$ as arguments returns true at T_8 when $r = 2$ and $x = 5$ (see Table 9). A skip, $S = 4$ is assigned to t_7 (T_8) and vectors T_4 up to T_8 are rotated once horizontally, bringing t_7 to column 4, as shown in Table 10. Variable *k* is updated as $k = 5$ whilst *x* and *y* are reset to 1 and 0, respectively. The Pivot Cycle procedure ends.

4.5. Summary of the rest of Algorithm Execution

Still $k < (n-1)$ and so the Pivot Cycle procedure is executed again with T_4 as the pivot vector. $T^{(3)}_4$ matches $T^{(-3)}_{11}$. T_{11} (t_{10}) was assigned a skip $s = m - r = 6 - 3 = 3$. After rotating vectors T_5 through T_{11} once, t_{10} came to occupy column 5 as the pivot.

Table 9: Comparison with T_3

T_k	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
Test ID	t_1	t_6	t_{11}	t_{12}	t_2	t_3	t_4	t_5	t_7	t_8	t_9	t_{10}
Label			i	d	i	d/i	d			i	d	
Skip	0	4	4	1					4			
Vectors	0	1	0	**0**	1	1	0	1	**0**	1	1	0
	1	0	1	**1**	0	0	0	1	**1**	1	0	0
	1	0	1	**0**	1	0	0	0	**0**	0	0	0
	0	1	0	**0**	0	0	1	1	**0**	1	0	1
	0	1	1	■	0	1	0	1	0	1	1	0
	1	0	0	■	0	0	0	1	0	1	0	1

Table 10: T_4 as Pivot Vector

T_k	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
Test ID	t_1	t_6	t_{11}	t_{12}	t_7	t_2	t_3	t_4	t_5	t_8	t_9	t_{10}
Label			i	d		i	d/i	d		i	d	

Skip	0	4	4	1	4							
Vectors	0	1	0	0	**0**	1	1	0	1	1	1	0
	1	0	1	0	**1**	0	0	0	1	1	0	0
	1	0	1	0	**0**	1	0	0	0	0	0	0
	0	1	0	1	**0**	0	0	1	1	1	0	1
	0	1	1	0	**0**	0	1	0	1	1	1	0
	1	0	0	0	**0**	0	0	0	1	1	0	1

Table 12: $Ssum_p$ Values of Resultant Matrices, $\tilde{A}_{(p)}$

$\tilde{A}_{(p)}$	$\tilde{A}_{(0)}$	$\tilde{A}_{(1)}$	$\tilde{A}_{(2)}$	$\tilde{A}_{(3)}$	$\tilde{A}_{(4)}$	$\tilde{A}_{(5)}$	$\tilde{A}_{(6)}$	$\tilde{A}_{(7)}$	$\tilde{A}_{(8)}$	$\tilde{A}_{(9)}$	$\tilde{A}_{(10)}$	$\tilde{A}_{(11)}$
$Ssum_p$	35	34	null	null	35	27	29	29	null	30	32	null

Next, $T_5^{(3)}$ matched $T_6^{(-3)}$, a skip, $S = 3$ is assigned to t_2 (T_6) , and, since t_2 is an initialising vector, the Initialising Pivot procedure is executed. T_6 matches its dependent vector t_3 (T_7) at $r = 2$. t_3 is therefore assigned a skip, $S = 4$. $t_3(T_7)$ is also an initialising vector so *y* is incremented to 2 and *r* reset to 0. T_7 matched its dependent vector T_8 $r = 1$. The dependent vector, t_4 (T_8), is assigned a skip, $S = 5$. Thus, T_8 becomes the next pivot without rotation because $x \ngtr 1$.

For increments of r from 1 through 6 did not result in any matches and so T_9 becomes the next pivot vector with s=m-r=6-6=0.

Now $k = 11$ and the condition, $k < (n-1)$ is false. The sum of all the skip values, $Ssum_0$, is therefore computed and assigned to the resultant matrix, $\tilde{A}_{(p=0)}$, as shown at the foot of Table 11.

Table 11: The Resultant Matrix $\tilde{A}_{(0)}$

T_k	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
Test ID	t_1	t_6	t_{11}	t_{12}	t_7	t_{10}	t_2	t_3	t_4	t_5	t_8	t_9
Label			i	d			i	d/i	d		i	d
Skip	**0**	**4**	**4**	**1**	**4**	**3**	**3**	**4**	**5**	**0**	**6**	**1**
Vectors	0	1	0	0	0	0	1	1	0	1	1	1
	1	0	1	0	1	0	0	0	0	1	1	0
	1	0	1	0	0	0	1	0	0	0	0	0
	0	1	0	1	0	1	0	0	1	1	1	0
	0	1	1	0	0	0	0	1	0	1	1	1
	1	0	0	0	0	1	0	0	0	1	1	0

The sum of skips $Ssum_0 = 35$ for matrix $\tilde{A}_{(0)}$.

T_9 matches T_{10} without shifting (i.e. $r = 0$). t_8 (T_{10}) is therefore assigned a skip, $S = m - r = 6 - 0 = 6$. t_8 is an initialising vector and so the Initialising Pivot procedure is executed matching $T_{10}^{(5)}$ and $T_{11}^{(-5)}$ and assigning $S = m - r = 6 - 5 = 1$ to the dependent vector, t_9 $(\text{at } T_{11})$.

The Main Procedure continues by incrementing *p* to 1. In other words, t_2 (T_1), of matrix A is selected as the next starting pivot vector. Since t_2 is not a dependent vector a skip, $S = 0$, is associated with it as shown in Table 13. At this stage (Step 6 of *Main* procedure) $y = 0$ and $k = p = 1 > 0$, so t_2 (T_1) swaps positions with t_1 (T_0) in a horizontal rotation of the two vectors. The execution of the Main procedure then continues till $Ssum_1$ is computed and assigned to $\tilde{A}_{(1)}$. This process is repeated for *p*=2, 3, …, 11. Note that vectors T_2, T_3, T_8 and T_{11} are dependent vectors and so $Ssum_2 = Ssum_3 = Ssum_8 = Ssum_{11} = NULL$.

Table 13: Selecting T_1 as the next Starting Pivot Vector for Matrix A

T_k	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
Test ID	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
Label		i	d/i	D				i	d		i	d
Skip		0										
Vectors	0	**1**	1	0	1	1	0	1	1	0	0	0
	1	**0**	0	0	1	0	1	1	0	0	1	0
	1	**1**	0	0	0	0	0	0	0	0	1	0
	0	**0**	0	1	1	1	0	1	0	1	0	1
	0	**0**	1	0	1	1	0	1	1	0	1	0
	1	**0**	0	0	1	0	0	1	0	1	0	0

Results obtained from the application of the proposed algorithm (Pattern Match Algorithm) to the example test set, (organized as matrix A), is as summarized in

From the table, the resultant matrices, $\tilde{A}_{(0)}$ and $\tilde{A}_{(4)}$, have the highest $Ssum_p$ = 35. Since of the two $\tilde{A}_{(0)}$ has the lesser value of *p*, it is chosen as the best arrangement of the test set for minimising test scan-

$\tilde{A}_{(p)}$	$\tilde{A}_{(0)}$	$\tilde{A}_{(1)}$	$\tilde{A}_{(2)}$	$\tilde{A}_{(3)}$	$\tilde{A}_{(4)}$	$\tilde{A}_{(5)}$	$\tilde{A}_{(6)}$	$\tilde{A}_{(7)}$	$\tilde{A}_{(8)}$	$\tilde{A}_{(9)}$	$\tilde{A}_{(10)}$	$\tilde{A}_{(11)}$
$Ssum_p$	35	34	null	null	35	27	29	29	null	30	32	null

in time using the proposed method. From Equations 1 and 2, the original test set using standard scan technique (without the Vector Match method) takes 84 clock cycles to scan-in all the vectors and a total of 89 clock cycles for the complete test application process.

From Equation (4), $n\epsilon = Ssum_0$. Therefore it takes $84 - 35 = 49$ clock cycles to scan-in the test vectors when the Pattern Match Algorithm is used. Adding the additional 5 clock cycles to scan-out the test signature brings the total test application time to 54 clock cycles. Evidently, the proposed approach gives better results.

5. Conclusion

After reviewing the scan architecture, it was realised that it does not allow the re-use of patterns in already scanned-in test vectors. This is because the same flip-flops that hold the scanned-in vectors are the ones that capture the response during the test application process. As a result, an already scanned-in vector is cleared by the response. Consequently, every bit in all the test vectors must be scanned in. This makes test application time very long. To solve this problem, a new architecture was proposed.

The new architecture involves the addition of a MISR to the scan chain of the circuit under test (CUT). In test mode, the state outputs are disconnected from the scan chain inputs and connected to the MISR. Thus the MISR is used to capture and compress the test responses. This allows previously scanned-in vector to be held in the flip-flops—paving a way for a possible reuse of the entire vector or part thereof. Next an algorithm, called the Pattern Match Algorithm (PMA), was developed to take advantage of repetitive patterns in test vectors by re-using them. By so doing, it avoids a complete scan-in of whole vectors. Time saving is achieved as a result.

The Pattern Match Algorithm works by rearranging test vectors to achieve the optimum arrangement for reducing test application time. In a cycle of the rearrangement process, patterns in test vectors are compared with that of a vector T_p, called the pivot vector. Among the rest of the vectors (yet to become pivots) one with the longest matching pattern is made to follow T_p directly. The vector so identified becomes the new pivot vector and the process continues to the last vector. A new cycle is then run with a new starting pivot. At the end the cycle with maximum savings in clock cycles determines the final arrangement of test vectors.

References

1. Williams, M., Angell, J.: *Enhancing Testability of Large-scale Integrated Circuits via Test Points and Additional Logic*. In: IEEE Transactions on Computers, 1973.

2. Eichelberger, E. B., Williams, T. W.: *A Logic Design Structure for Design for Testability*. In: Proceedings of Design Automation Conference, 1977.

3. Narayanan, S., Breuer, M. A.: *Asynchronous multiple scan chains*. In: Proceedings of VLSI Test Symposium, 270–276, 1995.

4. Narayanan, S., Gupta, R., Breuer, M. A.: *Optimal configuring of multiple scan chains.*In: IEEE Trans. on Computers, 1121–1131, 1993.

5. Narayanan, S., Njinda, C., Breuer, M. A.: *Optimal sequencing of scan registers*. In: Proceedings of Int. Test Conference,. 293–302, 1992.

6. Narayanan, S.: *Scan Chaining and Test Scheduling in an Integrated Scan Design System*. PhD thesis, University of Southern California, Los Angeles, CA*.,* 1994.

7. Hiraide, T., Boateng, K. O., Konishi, H., Itaya, K., Emori, M., Yamanaka, H., Mochiyama, T.: *BIST-Aided Scan Test - A New Method for Test Cost Reduction*. In: Proceedings of 21st IEEE VLSI Test Symposium, 359, 2003.

8. Morley, S. P., Marlett, R. A.: *Selectable length partial scan: a method to reduce*

vector length. In: Proceedings of Int. Test Conference*,* 385–392, 1991.

9. Narayanan, S., Breuer, M. A.: *Reconfiguration techniques for a single scan chain*. In: *IEEE Trans. on Computer-Aided Design*, 750–765, 1995.

10. Abramovici, M., Rajan, K. B., Miller, D. T.: *Freeze: a new approach for testing sequential circuits*. In: Proc. Design Automation Conference*,* 22–25, 1992.

11. Jha, N., Gupta, S.: *Testing of Digital Systems.* Cambridge University Press, 2003.