Regular Paper

An Efficient Self-reconfiguration Algorithm for Degradable Processor Arrays

Masaru Fukushi† and Susumu Horiguchi†

This paper considers the issue of reconfiguring degradable processor arrays implemented in VLSI/WSI (Wafer Scale Integration). As well as the efficiency of the reconfiguration algorithms, it has become necessary to develop a *built-in self-reconfiguration* mechanism which can automatically reconfigure a faulty processor array. However, so far, self-reconfiguration algorithms have not been proposed for degradable processor arrays. In this paper, we propose an efficient self-reconfiguration algorithm for degradable processor arrays based on simple schemes of column bypass and row rerouting. The proposed row rerouting scheme allows flexible inter-connections of neighboring processors since there is no limitation on the maximum distance. Furthermore, the rerouting scheme is performed in parallel using information from neighboring processors, and this property makes a hardware implementation of the proposed algorithm much easier. The performance of the proposed algorithm is compared with previous studies and indicates that although the maximum physical distance is longer, the proposed algorithm achieves better results in terms of harvest, degradation, and time complexity. Finally, the hardware implementation of the proposed algorithm is also shown to achieve self-reconfigurable systems.

1. Introduction

Massively parallel systems consisting of hundreds or thousands of processing elements (PEs) are expected to provide the capability for high performance computing. Recent advances in VLSI and WSI (Wafer Scale Integration) technologies allow the massively parallel systems to be implemented on a chip or a silicon wafer, consequently, high performance and small sizes of systems can be realized.

A mesh-connected processor array (mesh array) is a type of massively parallel system and is expected to be used as an architecture for various tasks such as image processing, signal processing, and so on. Since the mesh array has regular and simple structures, it is quite suitable for VLSI/WSI implementations. However, one of the major issues in designing such massively parallel systems is an efficient reconfiguration strategy to avoid defects/faults on a system.

There are two approaches to reconfiguring mesh arrays, namely the *redundancy approach* and the *degradation approach*. In the redundancy approach, some PEs are dedicated as spares to replace faulty PEs. The reconfiguration problem is to find an assignment of spare PEs to all faulty PEs, so a fault-free array whose size is fixed is obtained. For this approach, many reconfiguration algorithms have been proposed to date [1)~12)]. On the other hand, in the degradation approach, no PE is dedicated as spare. All PEs are treated in a uniform way and a fault-free array whose size is flexible is derived from a faulty array. Usually the problem is to derive a fault-free array of the maximum size under the constraint of the minimum dimension which is dependent on applications. This approach differs essentially from the redundancy approach in the reconfiguration strategies and the size of resultant arrays. In the redundancy approach, if all faulty PEs cannot be replaced, the wafer is regarded as faulty and will be discarded. In this sense, the degradation approach can contribute to enhance the yields where fault-free arrays of a fixed size are not strictly required. However, compared to the redundancy approach, little research has been reported on the reconfiguration problem of the degradation approach.

Kuo, et al. [13)] studied the reconfiguration problems for degradable mesh arrays under three switching and routing constraints, namely (*i*) *row and column bypass*, (*ii*) *row* (*column*) *bypass and column* (*row*) *rerouting* and (*iii*) *row and column rerouting*, and proposed some heuristic algorithms for these problems. Low, et al. proposed efficient heuristic algorithms for the second and the third problems in Refs. 14) and 15) respectively by employing

† School of Information Science, Japan Advanced Institute of Science and Technology

greedy rerouting. As well as the efficiency of the reconfiguration algorithms, it has become necessary to develop *built-in self-reconfiguration* mechanisms which can automatically reconfigure a faulty processor array [5),9),16)]. However, so far, self-reconfiguration algorithms have not been proposed for degradable processor arrays.

In this paper, we propose an efficient self-reconfiguration algorithm for degradable processor arrays based on the second problem of column bypass and row rerouting, and show a hardware implementation using FPGA (Field Programmable Gate Array). The proposed row rerouting scheme allows flexible interconnections of neighboring PEs since there is no limitation on the maximum distance. Furthermore, the rerouting scheme is performed in parallel using information from neighboring PEs, and this property makes a hardware implementation of the proposed algorithm much easier.

The rest of this paper is organized as follows: Section 2 describes the architecture of a degradable mesh array. Section 3 presents the proposed reconfiguration algorithm for the column bypass and row rerouting problem, and the reconfiguration performances are compared with a previous algorithm in Section 4. Section 5 briefly mentions a possibility of applying the proposed algorithm for the reconfiguration problem of row and column rerouting. The hardware implementation of the proposed algorithm is shown in Section 6. Section 7 is devoted to the conclusions.

2. Degradable Mesh Array

2.1 Architecture

In order to avoid faulty PEs, additional hardware such as switches and redundant interconnections are generally incorporated in mesh arrays. **Figure 1** shows the architecture of a degradable mesh array which has M rows and N columns of PEs and one vertical track between every two consecutive columns. A switch is allocated at each intersection point between tracks and links in order to change the interconnection of adjacent PEs. Each switch has four functions as shown in Fig. 1 and the *EW* function is the initial function for all switches. With respect to the location of the switches and tracks, this architecture is the same structure as presented in Refs. 13) and 14).

For convenience of explanation, addresses for all PEs and switches are defined as follows.

Definition 1 All PEs and switches have

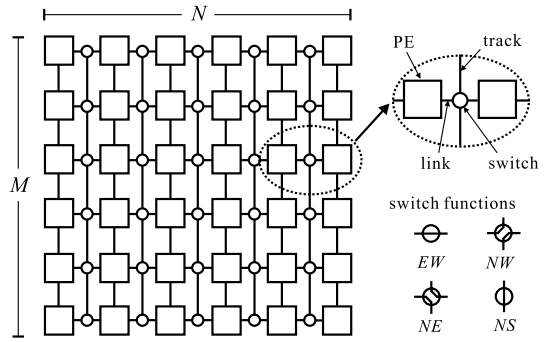


Fig. 1 Degradable mesh architecture.

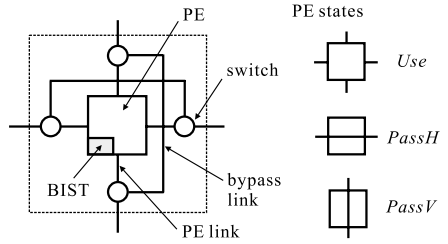


Fig. 2 Internal PE structure and its states.

row and column indices and those in the i-th row and j-th column are denoted as PE$[i, j]$ $(1 \leq i \leq M,\ 1 \leq j \leq N)$ and SW$[i, j]$ $(1 \leq i \leq M,\ 1 \leq j \leq N-1)$, respectively.

An internal structure of the PE is illustrated in **Fig. 2** where four switches are allocated to four links. Since each switch in Fig. 2 has only two functions (connect to either PE link or bypass link), the hardware structure of the switch is simpler than that of Fig. 1. All PEs with internal switches have three states, *Use*, *Pass Vertically* (*PassV*) and *Pass Horizontally* (*PassH*). The *PassV* and *PassH* states correspond to bypassing the PEs vertically and horizontally respectively. It is assumed that each PE$[i, j]$ has a Built-In-Self-Test (BIST) circuit to detect its own faults, though the hardware circuit is out of the scope of this paper. The output of the BIST is defined as follows to describe our reconfiguration algorithm.

Definition 2 The output of the BIST in PE$[i, j]$ is defined as $fault[i, j]$ which has a value of either 0 (fault-free) or 1 (faulty).

In our architecture, only PEs which include their BIST circuits are assumed to be faulty and all switches, links and tracks are assumed to be fault-free. Moreover the circuits required to implement the reconfiguration algorithm are also assumed to be fault-free. Such a fault model is assumed widely [3)~15)] and can be justified for

the following reason. The switches, tracks and links use much less hardware compared to the PE, thus their probabilities of being faulty are much lower.

2.2 Reconfiguration Problem

A *physical array* is an array after fabrication which includes some faulty PEs, and a *logical array* is a fault-free array after reconfiguration. Rows and columns included in a physical (logical) array are called *physical* (*logical*) *rows* and *physical* (*logical*) *columns*, respectively. The reconfiguration problem for degradable mesh arrays is formalized as follows.

> For a given $M \times N$ physical array and integer r and c, find an $m \times n$ logical array under the constraint of $m \geq r$ and $n \geq c$.

Generally, row (column) bypass schemes or row (column) rerouting schemes are employed to avoid faulty PEs. In the bypass scheme, one row (column) is removed from a physical array and adjacent two rows (columns) of the bypassed row (column) are directly connected as shown in **Fig. 3** (a). The states of PEs in bypassed columns are changed to *PassH*, hence this scheme requires no external switch to avoid faulty PEs. In the rerouting scheme, a faulty PE is removed from an array in such a way that inter-connections of neighboring fault-free PEs are rerouted in order not to connect with the faulty PE using external switches and tracks. Figure 3 (b) shows an example of row rerouting in which the PE$[i, j]$ is connected with PE$[i+1, j+1]$ to avoid faulty PE$[i, j+1]$. The states of faulty PEs are changed to *PassV*.

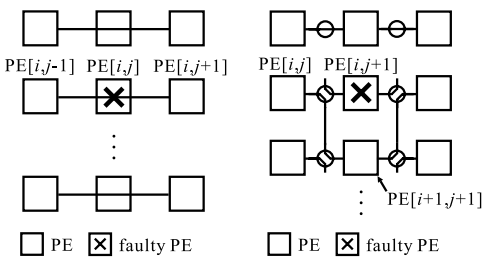
Here, *maximum row distance* is defined to be the largest possible difference between the row indices of two connected fault-free PEs in a logical row. To be defined more precisely, it corresponds to the maximum number of $|i - i'|$ between two logically connected PE$[i, j]$ and PE$[i', j']$ on the same logical row. In a typical row rerouting scheme, if T tracks are placed between every two consecutive columns, the maximum row distance is T and PE$[i, j]$ can connect with either PE$[i - T, j + 1]$, $\cdots$, PE$[i, j + 1]$ $\cdots$ or PE$[i + 1, j + 1]$.

It is obvious that row and column rerouting schemes are more flexible. However, the reconfiguration problem turns out to be very difficult since rerouting in both row and column directions are considered simultaneously. The algorithms proposed for the row and column rerouting problem [13)~15)] are still based on the idea of bypass and rerouting schemes. That is why we also concentrate on the column bypass and row rerouting scheme in developing our algorithm.

3. Column Bypass and Row Rerouting

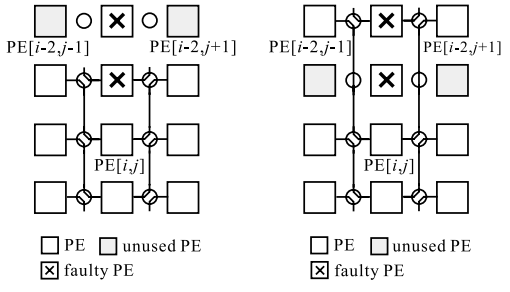
3.1 New Rerouting Scheme

Our proposed reconfiguration algorithm described later is based on a column bypass and a row rerouting scheme. An efficient rerouting scheme is employed in the proposed algorithm to realize more flexible inter-connections and it allows the maximum row distance to be more than one. **Figure 4** shows the difference between a typical row rerouting scheme and the proposed new row rerouting scheme. In Fig. 4 (a), PE$[i, j]$ cannot connect to both PE$[i-2, j-1]$ and PE$[i-2, j+1]$ due to the limitation of the maximum row distance. In this case both PE$[i - 2, j - 1]$ and PE$[i - 2, j + 1]$ cannot be used in a logical array since they have no valid inter-connections. On the other hand, the new rerouting scheme allows such prohibited inter-connections as shown in Fig. 4 (a). In Fig. 4 (b), PE$[i, j]$ can connect with both PE$[i - 2, j - 1]$ and PE$[i - 2, j + 1]$ using the switch function NS, and those PEs can be included in a logical array. However, both PE$[i - 1, j - 1]$ and PE$[i - 1, j + 1]$ become unused PEs since a track cannot be used for two



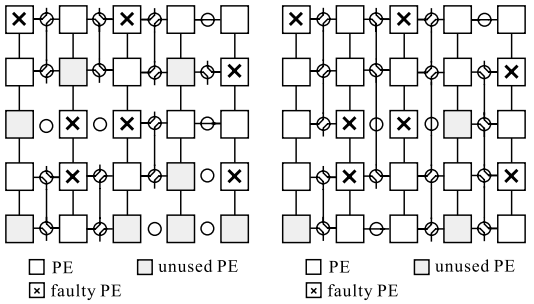
(a) Column bypass. (b) Row rerouting.

Fig. 3 Column bypass and row rerouting.



(a) Typical scheme. (b) Proposed scheme.

Fig. 4 Row rerouting schemes.



(a) Reconfiguration with typical scheme. (b) Reconfiguration with new scheme.

Fig. 5 Reconfiguration examples with row rerouting scheme.

distinct inter-connections.

The advantage of the new rerouting scheme is the efficient utilization of PEs. **Figure 5** illustrates examples of reconfiguration with a typical row rerouting scheme and the proposed row rerouting scheme. In Fig. 5 (a), at most two rows are obtained by row rerouting and many fault-free PEs become unused due to the limitation of maximum row distance. On the other hand, as in Fig. 5 (b), three rows can be obtained by the new rerouting scheme. The proposed new rerouting scheme is realized by two procedures of deactivating PEs and changing switch functions, and they are described as follows.

3.2 Deactivating PEs

The unused PEs shown in Fig. 4 (b) should be distinguished from other used PEs and deactivated in the logical array. The objective of this procedure is to determine which PEs should be unused and to deactivate them. The PEs which are suspected of being unused will receive signals from neighboring PEs. In Fig. 4 (b), PE$[i-1, j]$, which suspects the existence of such PEs, transfers signals to both PE$[i-1, j-1]$ and PE$[i-1, j+1]$, then they determine themselves whether they should be deactivated or not. Some variables are defined to describe the deactivation more precisely.

Definition 3 A variable $deact[i, j]$ is defined in PE$[i, j]$ and has a value of 0 (not deactivated) or 1 (deactivated).

Definition 4 A variable $unused[i, j]$ in PE$[i, j]$ is defined by '$fault[i, j]$ OR $deact[i, j]$'.

Definition 5 $F_{in}[i, j]$ and $F_{out}[i, j]$ are an input value and an output value of PE$[i, j]$ respectively and defined as follows to calculate the number of unused PEs in each column.

Procedure Deactivation(C)

$step := 0$;
for all $1 \le i \le M$ and $1 \le j \le N$ **do**
  if $j \in C$ **then**
    $F_{out}[i, j] := 0$;
    while $step < 2M$ **do**
      /* count of $F_{out}[i, j]$ */
      Transfer $F_{out}[i, j]$ to the PE$[i+1, j]$;
      /* transfer signals */
      if $unused[i, j] = 1$ and $F_{in}[i, j] \ge 1$ **then**
        Transfer signals to PE$[i, l(j)]$ and PE$[i, r(j)]$;
      end if
      /* decision for deactivation */
      if signal from PE$[i, r(j)]$ **then**
        if $F_{in}[i, r(j)] - F_{in}[i, j] \ge 1$ **then**
          $flag\_r := 1$; **else** $flag\_r := 0$;
        end if
      end if
      if signal from PE$[i, l(j)]$ **then**
        if $F_{in}[i, l(j)] - F_{in}[i, j] \ge 1$ **then**
          $flag\_l := 1$; **else** $flag\_l := 0$;
        end if
      end if
      $deact[i, j] := flag\_r$ OR $flag\_l$;
      $step := step + 1$;
    end while
    $m' := \min_{j \in C}[M - F_{out}[M, j]]$;
    if $deact[i, j] = 1$ or $i - F_{in}[i, j] > m'$ **then**
      Change state of PE$[i, j]$ to $PassV$;
    end if
  end if
end for
return m';

Fig. 6 Procedure for deactivating PEs.

$$F_{in}[i, j] = \begin{cases} 0 \quad (i = 1). \\ \sum_{k=1}^{i-1} unused[k, j] \\ \qquad (2 \le i \le M), \end{cases} \quad (1)$$

$$F_{out}[i, j] = F_{in}[i, j] + unused[i, j]. \quad (2)$$

It is clear that the $F_{out}[M, j]$ represents the total number of unused PEs in the j-th column.

Definition 6 $r(j)$ and $l(j)$ are defined as the physical column indices of the logical right and left columns of the j-th column respectively, where $l(j) < j < r(j)$.

For a given physical array, let C denote the set of logical column indices. **Figure 6** shows the procedure for deactivating PEs, where three tasks of 'count of $F_{out}[i, j]$', 'transfer signals' and 'decision for deactivation' are performed in each PE in parallel. These tasks are repeated just $2M$ times to determine all unused PEs.

Lemma 1 The Deactivation takes at most $2M$ time steps to decide all unused PEs, where M is the number of physical rows.

Proof: Suppose that PEs which are faulty and transfer signals to adjacent PEs are *source*

PEs. As can be seen easily, the source PE, i.e., PE$[i,j]$ never transfers signals to PEs in the i'-th row, where $1 \leq i' < i$. The signals from source PE$[i,j]$ propagate only in the same physical row. For example, suppose, source PE$[i,j]$ transfers signals to PE$[i,r(j)]$ and deactivates it. Then PE$[i,r(j)]$ begins to transfer signals to PE$[i,r(r(j))]$ in the next step if $F_{in}[i,r(j)]$ is greater than 1. Note that as in Fig. 6, these tasks are performed step by step.

Now consider the worst case in the i-th row. If all PE$[i',j]$ in the j-th column are faulty ($1 \leq i' \leq i$), then signals from the source PE$[i,j]$ may be propagated as far as PE$[i,j']$ in the same physical row, where $|j'-j| \leq i-1$, and it takes $(i-1)$ steps. Note that the $F_{in}[i,j]$ are also accumulated step by step, so it may take $(i-1)$ steps for PE$[i,j]$ to begin to transfer signals to PE$[i,j \pm 1]$. Thus, it takes at most $(i-1)+(i-1)$ steps for the source PE$[i,j]$ to decide all PE$[i,j']$ to be deactivated or not, where $|j'-j| \leq i-1$. Even if more than two source PEs exist in the same physical row, the above tasks are performed concurrently, hence, it takes at most $(i-1)+(i-1)$ steps in the i-th rows to decide all unused PEs.

From the above discussion the mesh array which has M rows takes at most $(M-1)+(M-1)$ steps to decide all unused PEs, and the number of steps is less than $2M$. □

After determining the unused PEs, the states of unused PEs are changed to *PassV*. Furthermore, fault-free PEs are also deactivated if they satisfy the condition $i-F_{in}[M,j] > m'$, because m' PEs in every column of C are used to construct a logical array.

3.3 Changing switch functions

All PEs which do not have valid interconnections have been deactivated in the previous procedure, and m' fault-free PEs remain in every column of C. In this procedure, the inter-connections of remaining PEs are rerouted by changing switch functions using a simple rule. One of the important features of the proposed algorithm is that each switch can change its own function automatically using the states of neighboring PEs. Therefore all switches are changed in parallel within one step. **Figure 7** shows a proposed rule to change all switch functions in which only *unused*$[i,j]$, *unused*$[i,r(j)]$, $F_{in}[i,j]$ and $F_{in}[i,r(j)]$ are used to decide the desired function of SW$[i,j]$. Again, we depict the same reconfiguration example as in Fig. 5 (b) to show

Procedure Change_Switch_Functions(C)

for all $1 \leq i \leq M$ and $1 \leq j \leq N-1$ **do**
  if $j \in C$ **then**
    $l :=$ *unused*$[i,j]$;
    $r :=$ *unused*$[i,r(j)]$;
    if $l = 1$ and $r = 1$ **then**
      Function of SW$[i,j] := NS$; $\cdots$ (1)
    else if $F_{in}[i,j] > F_{in}[i,r(j)]$ **then**
      Function of SW$[i,j] := NW$; $\cdots$ (2)
    else if $F_{in}[i,j] < F_{in}[i,r(j)]$ **then**
      Function of SW$[i,j] := NE$; $\cdots$ (3)
    else if $F_{in}[i,j] = F_{in}[i,r(j)]$ **then**
      if $l = 0$ and $r = 0$ **then**
        Function of SW$[i,j] := EW$; $\cdots$ (4)
      else if $l = 1$ and $r = 0$ **then**
        Function of SW$[i,j] := NW$; $\cdots$ (5)
      else if $l = 0$ and $r = 1$ **then**
        Function of SW$[i,j] := NE$; $\cdots$ (6)
      end if
    end if
  end if
end for

Fig. 7 A rule for changing switch functions.

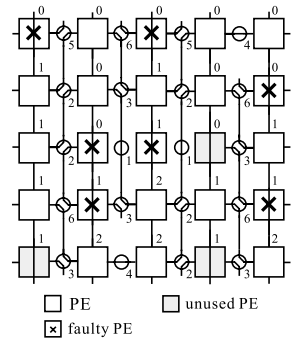


Fig. 8 An example of changing switch functions.

an example of changing switch functions with this rule. In **Fig. 8** the value on each PE$[i,j]$ corresponds to that of $F_{in}[i,j]$ and the number under each SW$[i,j]$ corresponds to the number within parenthesis shown in Fig. 7. For example, the value 5 subscribed in the upper left switch in Fig. 8 means that the switch function was changed according to condition (5) in Fig. 7.

Lemma 2 Suppose that m' available PEs in the u-th column are labeled as $u_1, u_2, \cdots, u_{m'}$ in order of physical row index. In two adjacent logical columns, u and v, u_i can be connected to v_i ($1 \leq i \leq m'$) using the proposed rule in Fig. 7, and there are no contradictions such that each PE has a valid inter-connection and any two inter-connections never intersect with each other.

Proof: First, an inter-connection of u_1 and v_1

is considered. Suppose that $u_1 = \mathrm{PE}[i_{u1}, u]$ and $v_1 = \mathrm{PE}[i_{v1}, v]$, where i_{u1} and i_{v1} indicate the physical row indices of u_1 and v_1, respectively. There are three cases between i_{u1} and i_{v1}; (i) i_{u1} and i_{v1} are equal, (ii) i_{u1} is smaller than i_{v1}, (iii) i_{u1} is greater than i_{v1}. In case (i), it is clear that u_1 and v_1 are connected according to condition (4) in Fig. 7, because $F_{in}[i_{u1}, u]$ and $F_{in}[i_{v1}, v]$ are equal. In case (ii), the function of $\mathrm{SW}[i_{u1}, u]$ is determined to be NE according to condition (6), since v_1 is the first fault-free PE in column v. The function of $\mathrm{SW}[i_{v1}, u]$ is determined to be NE according to condition (3), since $F_{in}[i_v, v]$ is greater than $F_{in}[i_u, u]$. If $(i_{v1} - i_{u1})$ is equal to 1, u_1 is connected to v_1 obviously. Even if $(i_{v1} - i_{u1})$ is greater than 1, u_1 is connected to v_1 for the following reason. Since all $\mathrm{PE}[i, v]$ $(1 \le i < i_{v1})$ are faulty or deactivated, all $\mathrm{PE}[i', u]$ $(i_{u1} + 1 \le i' < i_{v1})$ will be deactivated even if they are fault-free. Then the function of $\mathrm{SW}[i', u]$, which are placed beside $\mathrm{PE}[i', u]$, are changed to be NS according to condition (1), consequently, $\mathrm{SW}[i_{u1}, u]$ and $\mathrm{SW}[i_{v1}, u]$ are connected. Thus u_1 and v_1 are also connected when $(i_{v1} - i_{u1})$ is greater than 1. The proof in case (iii) is omitted since this case is symmetrical with case (ii). From the above, it is shown that u_1 and v_1 are connected to each other.

Secondly, an inter-connection of u_2 and v_2 is considered. Suppose that physical row indices of u_2 and v_2 are i_{u2} and i_{v2}, respectively. By considering the relationship among i_{u1}, i_{v1}, i_{u2} and i_{v2} in above each case of (i), (ii) and (iii) respectively, the same discussions are valid for u_2 and v_2 as for u_1 and v_1, therefore it is proved easily that u_2 and v_2 are connected to each other and the connection never intersects with that of u_1 and v_1.

The whole procedure above can be applied sequentially in following rows. □

Theorem 1 A logical mesh array of size $m' \times n'$ is constructed by Deactivation and Changing_Switch_Functions, where m' is obtained from the Deactivation and n' is equal to the number of columns of C.

Proof: As mentioned in the previous subsection, all n' columns included in C have m' fault-free PEs after Deactivation. According to Lemma 2, the m' PEs have valid interconnections between adjacent columns, hence m' rows are constructed correctly. □

Note that each label i of u_i and v_i $(1 \le i \le m')$ in Lemma 2 corresponds to a logical row index, and can be calculated by $i - F_{in}[i, j]$ $(j \in C)$.

3.4 Bypassing a column

It is reported in Ref. 5) that bypassing is an efficient strategy for clustered fault model where a large number of faults concentrate on parts of a wafer. The problem is how to determine which column should be bypassed. Obviously, the column which has the maximum number of faulty PEs should be bypassed with high priority, since more unavailable PEs are removed from the array. To represent the number of faulty PEs, the following variables are defined.

Definition 7 Both $f_{in}[i, j]$ and $f_{out}[i, j]$ are defined by replacing $unused[i, j]$ in equations (1) and (2) of Definition 5 by $fault[i, j]$ respectively.

The procedure Bypass_a_Column is described in **Fig. 9**. In this procedure, a search from leftmost to rightmost columns is executed twice in order to bypass one column. In the first search, the maximum value of $f_{out}[M, j]$ is obtained (denoted by f_{max} in Fig. 9) and in the second search one column is bypassed. In order to bypass a column in the area where the most faulty PEs are clustered, not only $f_{out}[M, j]$ but also $a[j]$ is taken into consideration as in Fig. 9, if there are more than two columns

```
Procedure Bypass_a_Column(C)

f_max = a_max := 0;
if j ∈ C then
  for all 1 ≤ j ≤ N do
    a[j] := f_out[M, l(j)] + f_out[M, j] + f_out[M, r(j)];
            (if columns l(j) or r(j) exist)
  end for

  /* obtain maximum number of f_out[M, j] */
  for j=1 to N do
    if (f_out[M, j] > f_max) or (f_out[M, j] = f_max
        and a[j] > a_max) then
      f_max := f_out[M, j];
      a_max := a[j];
    end if
  end for

  /* bypass a column */
  for j=1 to N do
    if f_out[M, j] = f_max and a[j] = a_max then
      for all 1 ≤ i ≤ M do
        State of PE[i, j] := PassH;
        Function of SW[i, j] := EW;
      end for
      C := C − {j};
      f_max := −1;
    end if
  end for
end if
```

Fig. 9 Procedure for bypassing one column.

Algorithm DBC(M, N, r, c)

C (set of indices of logical columns) := $\{1, \cdots, N\}$;
B (set of indices of bypassed columns) := ϕ;
$m = n := 0$;

for $col = N$ to c **do**
  row := Deactivation(C);
  if $row \times col > m \times n$ **then**
    $m := row$;
    $n := col$;
    add all indices of bypassed columns to B;
  end if
  Bypass_a_Column(C);
  $col := col - 1$;
end for

if $m > r$ and $n > c$ **then**
  $C := \{1, \cdots, N\}$; (Initialize array)
  Bypass all columns included in B;
  $C := C - B$;
  Deactivation(C);
  Change_Switch_Functions(C);
  Return 'Logical array of size $m \times n$ found';
else
  Return 'reconfiguration failed';
end if

Fig. 10 The algorithm DBC.

whose $f_{out}[M, j]$s are equal to f_{max}.

3.5 Proposed Reconfiguration Algorithm

A reconfiguration algorithm for degradable mesh arrays consisting of the procedures Deactivation, Change_Switch_Functions, and Bypass_a_Column is shown in **Fig. 10**. The proposed algorithm is characterized by two procedures, bypassing columns and changing switch functions automatically, hence it is named *Degradation algorithm based on Bypass and Change* (DBC). Firstly, in this algorithm all N columns of a physical array are included in a logical array, and the maximum number of rows (denoted by *row* in Fig. 10) is obtained by Deactivation. At this point, according to Theorem 1, a logical array of size $row \times N$ can be obtained and becomes a candidate for the final logical array. Then, after bypassing a column by procedure Bypass_a_Column, the *row* is renewed by Deactivation and a logical array of size $row \times (N - 1)$ is provably obtained. If the size is larger than the previous one, this array becomes the new candidate for the final logical array. These procedures are repeatedly applied until $(N - c)$ columns are bypassed.

4. Performance Comparisons

To evaluate the efficiency of DBC synthetically, we employ four figures of merit; harvest, degradation, time complexity, and the maximum physical distance, which are in relation of tradeoff, in general. So far, there has been no better algorithm for column bypass and row rerouting than MAXCOR [14], therefore we compare those of DBC with those of MAXCOR. In each comparison, we assume the case that an $m \times n$ logical array is obtained from an $M \times N$ physical array under the constraint of $m \geq r$ and $n \geq c$.

Table 1 Comparison of harvest and degradation between DBC and MAXCOR.

		MAXCOR		DBC	
physical array	PE yield	harvest (%)	degradation (%)	harvest (%)	degradation (%)
16×16	0.95	87.63	15.87	88.75	15.74
16×16	0.90	83.34	25.23	84.08	24.45
16×16	0.85	78.74	32.95	80.45	31.49
16×16	0.80	73.24	41.36	76.70	38.60
16×16	0.75	67.66	49.26	73.32	45.01
32×32	0.95	89.75	14.72	90.06	14.42
32×32	0.90	83.58	24.74	84.87	23.59
32×32	0.85	77.05	34.54	80.16	31.89
32×32	0.80	70.23	43.82	75.94	39.26
32×32	0.75	62.72	52.96	72.07	45.94

4.1 Harvest and Degradation

Algorithm DBC has been implemented in C and simulated using large numbers of randomly generated datasets. Here we introduce the definitions of harvest and degradation, which are commonly used to evaluate the efficiency of algorithms in the degradation approach [13)~15)].

$$harvest = \frac{N_{logical}}{N_{fault\text{-}free}} \times 100\%,$$

$$degradation = \frac{N_{physical} - N_{logical}}{N_{physical}} \times 100\%,$$

where $N_{physical}$, $N_{logical}$ and $N_{fault\text{-}free}$ correspond to the number of PEs in a physical array, the number of PEs in a logical array, and the number of fault-free PEs in a physical array, respectively. The harvest represents how effective the fault-free PEs are utilized in constructing a logical array from a physical array and the degradation measures the degree of potential performance loss due to a smaller logical array than the physical array.

Table 1 shows the harvest and degradation for physical arrays of size 16×16 and 32×32 when the minimum size of a logical array is 1×1. This size means that both algorithms never fail to reconfigure unless all PEs are faulty. The harvest and degradation in Table 1 are the average of 10000 simulation results. Note that MAXCOR stops reconfiguring if the obtained logical array is larger than $r \times c$. Therefore MAXCOR is modified to find a logical array of

the maximum size in order to compare fairly.

It is clear from Table 1 that DBC consistently outperforms MAXCOR in terms of the percentage of harvest and degradation for both sizes of physical arrays. The difference is particularly noticeable under conditions of low PE yield.

4.2 Time complexity

The time complexity of DBC is $O((N - c)(t_1 + t_2))$, where t_1 is the time required to perform Deactivation and t_2 is the time required by Bypass_a_Column. The times of other processes, including the procedure Change_Switch_Functions, are neglected since they take at most a small number of steps. It is easy to see that t_1 is $O(M)$ and t_2 is $O(N)$. Note that $f_{out}[M, j]$ in Fig. 9 can be accumulated when $F_{out}[M, j]$ is accumulated in Fig. 6. Then the time complexity of the proposed DBC is $O((N - c)(M + N))$.

On the other hand, the time complexity of MAXCOR is $O((N - c)F)$, where F is the number of fault-free PEs [14]. Taking into account the fact that F is close to $M \times N$, the time complexity of DBC is much smaller than MAXCOR.

4.3 Maximum Physical Distance

The maximum physical distance between logically adjacent PEs decides the maximum signal latency of reconfigured arrays. The physical distance between logically adjacent PE$[i, j]$ and PE$[i', j']$ is defined to be $|i - i'| + |j - j'|$. The maximum row distance of DBC under the assumed case is $(M - m)$, therefore the maximum physical distance of DBC is $(M - m) + (N - n + 1)$ when PE$[i, j]$ and PE$[i + (M - m), j + (N - n + 1)]$ are connected logically.

On the other hand, the maximum row distance is limited to 1 in MAXCOR. The worst case is that PE$[i, j]$ and PE$[i+1, j+(N-n+1)]$, or PE$[i, j]$ and PE$[i + (M - m + 1), j]$ are connected logically. Then the maximum physical distance is $max[(N - n + 2),\ (M - m + 1)]$. Hence, the maximum physical distance of DBC is almost double compared to that of MAXCOR in regular arrays.

4.4 Comparison with Row and Column Rerouting Scheme

As mentioned earlier, the reconfiguration with row and column rerouting schemes is a very difficult problem in the degradation approach, and efficient algorithms have not been proposed yet. The methodology of divide and conquer is employed commonly [13)~15)] to apply the algorithm for column bypass and row rerouting to the reconfiguration problem of row and column rerouting. In this method, a reconfiguration algorithm based on bypass and rerouting schemes is applied twice in both row and column directions. Then the largest array which is obtained by these two algorithms is taken as the final logical array. Note that even if these two algorithms are applied, rerouting in both row and column directions are not considered simultaneously.

Table 2 Comparison of harvest and degradation between DBC and RCRoute.

		RCRoute		DBC	
physical array	PE yield	harvest (%)	degradation (%)	harvest (%)	degradation (%)
16×16	0.95	90.44	14.15	90.45	14.14
16×16	0.90	85.80	22.90	86.05	22.68
16×16	0.85	82.08	30.10	82.65	29.62
16×16	0.80	78.04	37.51	79.24	36.55
16×16	0.75	74.51	44.11	76.02	42.98
32×32	0.95	90.83	13.69	91.03	13.50
32×32	0.90	85.32	23.18	86.20	22.39
32×32	0.85	79.80	32.20	81.75	30.54
32×32	0.80	74.44	40.47	77.77	37.80
32×32	0.75	69.34	47.99	73.80	44.65

Low [15] proposed a new scheme which enables fault-free PEs in bypassed rows (columns) to replace adjacent faulty PEs under the constraint where the maximum row (column) distance is one. The algorithm RCRoute [15] is the combination of this scheme and MAXCOR, where a divide and conquer method is also employed. The performance of DBC is compared with that of RCRoute under the following two assumptions as in Refs. 13)~15). (*i*) Switches and a track are also placed between every two adjacent rows. (*ii*) The methodology of divide and conquer is also employed in DBC.

Table 2 shows the harvest and degradation in the same fashion as Table 1. From Table 2, DBC shows higher percentages of harvest and lower percentages of degradation than those of RCRoute. Thus it is shown that DBC can also be applied as an efficient strategy for the row and column rerouting problem. Note that the time complexity and the maximum physical distance of RCRoute is almost the same as MAXCOR.

5. Self-reconfigurable Mesh Array on FPGA

To show the possibility of achieving self-reconfigurable systems, which is an important feature for massively parallel systems, we give a hardware implementation of the proposed

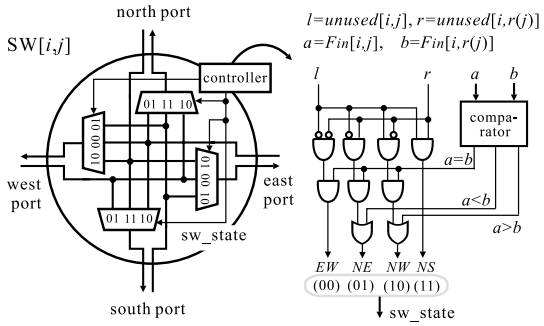


Fig. 11 Switch circuit.

DBC.

5.1 Circuit Design

Each component such as switch, PE, and controller is designed in VHDL using design tool "MAX+PlusII 9.6 (ALTERA)". **Figure 11** illustrates a switching circuit that consists of four multiplexers and one controller. The rule in Fig. 7 is implemented in this controller to determine switch function automatically. According to the determined function, actual connections of all ports are established by using multiplexers. For example, if the controller determines the switch function EW, then the value 00 is input to all multiplexers to connect the east and west port with each other.

Each PE is designed to execute only the functions required to perform DBC, though the detailed circuit is omitted here. Each PE$[i,j]$ consists of four internal switches, an accumulator for $F_{out}[i,j]$, a signal generator to deactivate PE$[i,r(j)]$ and PE$[i,l(j)]$, a decision maker circuit driving PE deactivation, flip-flops to hold $fault[i,j]$, $unused[i,j]$, and data selectors. By these circuits, procedure Deactivation is performed automatically synchronized to the system clock. The data selectors have the following important function when the PE is bypassed. Suppose that the PE$[i,j]$ is bypassed and both PE$[i,l(j)]$ and PE$[i,r(j)]$ are connected to each other. In this case, PE$[i,l(j)]$ requires the information in PE$[i,r(j)]$ instead of that in PE$[i,j]$ to perform both Deactivation and Change_Switch_Functions correctly, and PE$[i,r(j)]$ also requires the information in PE$[i,l(j)]$. The information held in PE$[i,r(j)]$ such as $unused[i,r(j)]$, $F_{in}[i,r(j)]$, and the signal wire for transferring the deactivation signal will be supplied to PE$[i,l(j)]$ by the selectors.

To control the DBC and column bypass, main controller and bypass controllers are designed as in **Fig. 12**. The main controller in Fig. 12

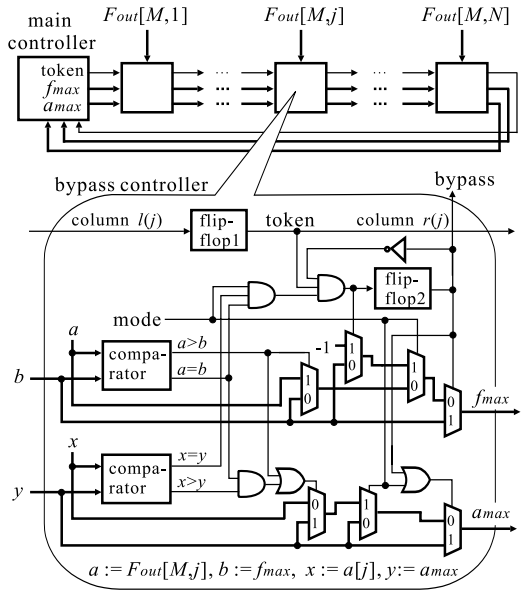


Fig. 12 Controller for DBC and column bypass.

controls DBC, though the detailed circuits are omitted here, and a bypass controller is allocated at the bottom of each column to perform column bypass. Flip-flop 1 in Fig. 12 configures a token ring by connecting to other controllers in order to provide tokens. Only one column which gets the token can perform the processes of column bypass. The values f_{max} and a_{max} are input from the previous $l(j)$-th column to this j-th column and they are compared with $F_{out}[i,j]$ and $a[j]$ respectively. According to the conditions in the bypassing algorithm, a new f_{max} and a_{max} are selected using multiplexers and they are output to the next $r(j)$-th column. They are input finally to flip-flops in the main controller. The signal "mode" in Fig. 12 denotes the first search (mode = "Low") or the second search (mode = "High") as presented in sub-section 3.4. In the second search, if $F_{out}[i,j]$ and $a[j]$ equals to f_{max} and a_{max} respectively, the column which is not bypassed yet is selected for bypass. Once the column is bypassed, flip-flop 2 shown in Fig. 12 continues to output "High" and inputs f_{max} and a_{max} will pass through the column.

5.2 Hardware Implementation

Using the components presented in the previous sub-section, a prototype system of self-reconfigurable mesh array of size 6×6 is implemented in FPGA boards, MEB200-A250 and MU200-EA10 (Mitsubishi Electronic Microcomputer Application Software Co. LTD.). The

former, which has EPF10K250AGC599-3 device (ALTERA), is for the system and the latter is for the data I/O. In this implementation, we assume that each PE link is 2-bits wide, namely 1-bit input and 1-bit output. Some fault patterns are also embedded into PEs and we check all I/O ports, switch states, and PE states. With these checks, we observed the expected results according to the embedded fault patterns and verified that the proposed DBC can be implemented on FPGA where the clock frequency is 20 MHz. Since the frequency is the maximum output of the FPGA board, our system seems to work with higher clock frequencies.

The hardware cost for one switch, one PE, the controller in Fig. 12 and a 6×6 self-reconfigurable mesh array shows 27 LCs (Logic Cells), 34 LCs, 402 LCs, and 2,395 LCs, respectively. Since 1LC corresponds to about 20 gates, entire cost for the 6×6 array is about 47,900 gates. Note that this estimate is very rough because 20 gates are not always implemented in 1LC. They will become much simpler if they are implemented in customized IC.

6. Conclusion

The reconfiguration process to avoid faulty PEs is one of the most important issues for designing massively parallel systems in VLSI/WSI. In this paper, we proposed an efficient self-reconfiguration algorithm for degradable mesh arrays based on a column bypass scheme and a row rerouting scheme. The proposed algorithm named DBC allows maximum row distance to be more than one in single track architectures, and it realizes an efficient utilization of PEs. Furthermore, the rerouting scheme is performed in parallel using information from neighboring PEs, and this property makes the hardware implementation of the proposed algorithm much easier. The performance of the proposed algorithm is compared with previous studies and indicates that although the maximum physical distance is longer than previous studies due to the flexible rerouting, the proposed algorithm achieves better results in terms of harvest, degradation, and time complexity. Finally, we showed the hardware implementation of the proposed algorithm using FPGA and confirmed its correct behavior.

Future work is to investigate the possibility of developing more efficient reconfiguration algorithms such that both row and column rerouting are performed simultaneously.

Acknowledgments This research was supported in part by Grant-in-Aid for Scientific Research No.11558032, Ministry of Education and Science of Japan.

References

1) Kung, S.Y., Jean, S.N. and Chang, C.W.: Fault-Tolerant Array Processors Using Single-Track Switches, *IEEE Trans. Comput.*, Vol.38, No.4, pp.501–514 (1989).
2) Jean, S.N., Fu, H.C. and Kung, S.Y.: Yield Enhancement for WSI Array Processors using Two-and-Half-Track Switches, *Proc. Int'l. Conf. on Wafer Scale Integration*, pp.243–250 (1990).
3) Roychowdhury, V.P., Bruck, J. and Kailath, T.: Efficient Algorithms for Reconfiguration in VLSI/WSI Array, *IEEE Trans. Comput.*, Vol.39, No.4, pp.480–489 (1990).
4) Varvarigou, T.A., Roychowdhury, V.P. and Kailath, T.: A Polynomial Time Algorithm for Reconfiguring Multiple-Track Models, *IEEE Trans. Comput.*, Vol.42, No.4, pp.385–395 (1993).
5) Numata, I. and Horiguchi, S.: A Self-Reconfiguration Architecture for Mesh Arrays, *Proc. Int'l. Workshop on Defect and Fault Tolerance in VLSI Systems*, pp.212–220 (1994).
6) Numata, I. and Horiguchi, S.: Wafer-scale Integration Implementation of Mesh-Connected Multiprocessor Systems, *Syst. and Comput. in Japan*, Vol.26, No.1, pp.1–10 (1995).
7) Takanami, I., Kurata, K. and Watanabe, T.: A Neural Algorithm for Reconstructing Mesh-Connected Processor Arrays Using Single-Track Switches, *Proc. Int'l. Conf. on Wafer Scale Integration*, pp.101–110 (1995).
8) Horita, T. and Takanami, I.: A Built-In Self-Reconstruction Approach for Partitioned Mesh-Arrays Using Neural Algorithm, *IEICE Trans. Inf. and Syst.*, Vol.E79-D, No.8, pp.1160–1167 (1996).
9) Horita, T. and I., T.: An Efficient Method for Reconfiguring the $1\frac{1}{2}$ Track-Switch Mesh Array, *IEICE Trans. Inf. and Syst.*, Vol.E82-D, No.12, pp.1545–1553 (1999).
10) Horita, T. and I., T.: Fault-Tolerant Processor Arrays Based on the $1\frac{1}{2}$-Track Switches with Flexible Spare Distributions, *IEEE Trans. Comput.*, Vol.49, No.6, pp.542–552 (2000).
11) Shigei, N., Miyajima, H., Ishizaka, T. and Murashima, S.: On Methods for Reconfiguring Processor Arrays, *IEICE Trans. Inf. and Syst.*, Vol.E79-D, No.8, pp.1139–1146 (1996).
12) Chen, Y.Y., Upadhyaya, S.J. and Cheng, C.H.: A Comprehensive Reconfiguration Scheme for Fault-Tolerant VLSI/WSI Array Processors, *IEEE Trans. Comput.*, Vol.46, No.12,

pp.1363–1371 (1997).
13) Kou, S.Y. and Chen, I.Y.: Efficient Reconfiguration Algorithms for Degradable VLSI/WSI Arrays, *IEEE Trans. Computer-Aided Design*, Vol.11, No.10, pp.1289–1300 (1992).
14) Low, C.P. and Leong, H.W.: On the Reconfiguration of Degradable VLSI/WSI Arrays, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.16, No.10, pp.1213–1221 (1997).
15) Low, C.P.: An Efficient Reconfiguration Algorithm for Degradable VLSI/WSI Arrays, *IEEE Trans. Comput.*, Vol.49, No.6, pp.553–559 (2000).
16) Smith, M.D. and Mazumder, P.: Generation of Minimal Vertex Covers for Row/Column Allocation in Self-Repairable Arrays, *IEEE Trans. Comput.*, Vol.45, No.1, pp.109–115 (1996).

(Received August 31, 2001)
(Accepted December 18, 2001)

Masaru Fukushi received the B.S. and M.S. degrees in information science from Hirosaki University in 1997 and 1999, respectively. He is currently working towards the Ph.D. degree at Graduate School of Information Science in JAIST (Japan Advanced Institute of Science and Technology). His research interests are fault-tolerant systems and multi-processor systems.

Susumu Horiguchi received the M.E and D.E degrees from Tohoku University in 1978 and 1981, respectively. He was a faculty of Department of Information Science at Tohoku University from 1981 to 1992. He was a visiting scientist of IBM Thomas J. Watson Research Center from 1986 to 1987 and a visiting professor of The Center for Advanced Studies at the University of Southwestern Louisiana and Department of Computer Science, Texas A&M University summer in 1994 and 1997. He is currently a Full Professor of the Graduate School of Information Science at JAIST (Japan Advanced Institute of Science and Technology). He has been conducting his research group as the chair of Multi-Media Integral System Laboratory at JAIST. He has been involved in organizing many international workshops, symposia and conferences sponsored by IEEE, IEICE and IPS. His research interests have been mainly concerned with interconnection networks, parallel computing algorithm, massively parallel processing, parallel computer architecture, VLSI/WSI architecture, and Multi-Media Integral System. Dr. Horiguchi is a senior member of IEEE Computer Society, and members of IPS and IASTED.