Self-Reconfigurable Mesh Array System on FPGA *

Masaru Fukushi Susumu Horiguchi
Graduate School of Information Science, JAIST
mfukushi@jaist.ac.jp, hori@jaist.ac.jp

Abstract

Massively parallel computers consisting of thousands of processing elements are expected to be high-performance computers in the next decade. One of the major issues in designing massively parallel computers is the reconfiguration strategy in order to provide an efficient fault tolerance mechanism to avoid defective processors in such large scale systems. This paper develops a self-reconfigurable mechanism of mesh array for easy hardware implementation using local defect information. Compared to those of previous reconfigurable architecture, the proposed self-reconfigurable mechanism achieves almost the same system yields using only local defect information. A prototype of this self-reconfigurable array is implemented on FPGA and the hardware complexities are also discussed.

1. Introduction

Stacked-silicon-planes is one of the attractive technology to implement massively parallel computers with the recent progress of VLSI technology. The development of such large scale systems will make numerous applications such as multi-media, computer vision, and modeling physical phenomena possible. However, one of the major issues in designing large scale systems is a reconfiguration strategy to provide an efficient fault tolerance mechanism to avoid defects on silicon planes. To achieve the reconfiguration of massively parallel computers many previous studies have been proposed for mesh array.

Kung $et\ al.$ [1] proposed a $1\frac{1}{2}$ track switch model which has a row and a column of spare processing elements (PE) around an $N \times N$ mesh array and switching circuits and tracks placed between PEs. Defective PEs can be removed from the array by changing each switch function. In this model, an $N \times N$ array is realized by assigning each faulty PE to a compensation path with a spare PE using the graph theory. Since this model has low hardware overhead involved when changing each inter-connection of adjacent PEs, many studies have focused on this model [2], [3]. However, these reconfigurable strategies require a host computer to calculate the combination of the compensation paths using the global information of faulty PEs.

For massively parallel computers, an effective self-reconfiguring mechanism using local defect information is very attractive. Numata and Horiguchi [4] proposed a Bypass and Shift (BS) method using only local defect information. The BS method, however, does not guarantee a fast reconfiguration since it uses recursive signals to avoid deadlocks. Takanami $et\ al.$ [6] proposed a neural algorithm using a Hopfield-type neural network. Although many reconfiguration methods have been proposed, these methods are not hardware oriented because they require complex algorithms or procedures to change all the switch functions.

* This work has been supported in part by #1155802 Grant-in-Aid for scientific research.

 0-7695-0719-0/00 $10.00

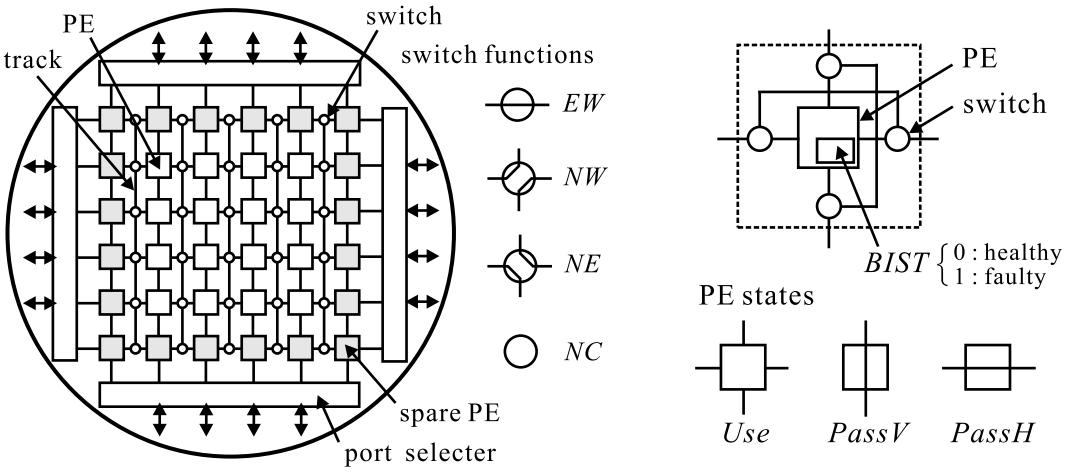

Figure 1. Self-reconfigurable mesh array and PE internal structure.

This paper proposes an efficient self-reconfiguration mechanism for mesh arrays which uses only local defect information. A simple rule used to change each switch function in the proposed algorithm enables hardware implementation to be made more easily. Comparing the reconfiguration performance of the proposed method with previous studies, it is shown that the proposed method achieves the same system yields and reconfiguration speeds are much faster. The hardware complexities of the self-reconfigurable system are also evaluated by implementing a prototype self-reconfigurable system on FPGA chips.

2. Self-reconfigurable Mesh Array

Reconfiguration in array processors aims to obtain an $N \times N$ array from an $(N+R) \times (N+R)$ array by replacing faulty PEs with spare PEs using switches and redundant inter-connections. R rows and R columns of spare PEs are added to the $N \times N$ array. Figure 1 shows the self-reconfigurable mesh array when N=4 and R=2. Spare PEs are identical to normal PEs in terms of function and performance. The physical address of each PE is denoted as PE[i,j] ($1 \leq i, j \leq N+R$), where the i-th row and the j-th column are ordered from the upper-left. T vertical tracks are placed between each column to connect adjacent PEs in a horizontal direction. This paper will focus on a T=1 type architecture and it will be described later.

Switches are allocated at intersection points and have four functions as shown in Fig.1. The EW function is the initial function of all switches. Each switch has a physical address described as SW[i,j] where $1 \leq i, j \leq N+R$-1. The port selector in Fig.1 selects N I/O ports from possible $N+R$ ports. The self-reconfigurable architecture is named as an N-R-T type, then Fig.1 shows a 4-2-1 type.

Figure 1 also shows the PE structure where four switches are allocated to each I/O port to achieve three PE states; *Use*, *PassV* and *PassH*. The *PassV* and *PassH* states correspond to bypassing PEs vertically and horizontally. All faulty PEs change their states into the *PassV* state and become connecting elements. The *PassH* state is used in bypassing. It is also assumed that each PE[i,j] has a Built-in-Self-Test (BIST) circuit to detect faults, denoted as $BIST$[i,j] which has a value of 1 (faulty) or 0 (healthy), though its circuit is not discussed in this paper.

Compared to $1\frac{1}{2}$ track switch model, the N-R-T type architecture has following characteristics.

Kung's model has switches in both horizontal and vertical tracks because it changes switch status by selecting a compensation path among four directions in order to find a spare PE. On the other hand, N-R-T type architecture doesn't need switches in the horizontal tracks because a logical address given to component PE of the array is shifted toward south direction as well as BS method [4]. The number of spare rows and columns are limited in the $1\frac{1}{2}$ track switch model where a T track model can treat just the T spare rows and columns of spare PEs. The proposed N-R-T type, however, chooses any number of rows and columns of spare PEs.

3. Reconfiguration Algorithm

To obtain an $N \times N$ array from an $(N+R) \times (N+R)$ array including any faulty PEs, the proposed reconfiguration algorithm consists of the following three steps. *Step 1* is to bypass the R columns in order of the number of faulty PEs. *Step 2* is to deactivate all the PEs which disturb proper inter-connection between adjacent PEs. *Step 3* is to change all the switch functions at the same time. The proposed reconfiguration comprises of two tasks, bypassing columns and changing the switch functions, hence it is called the Bypass and Change (BC) method.

3.1. Bypassing columns

Step 1 of BC method is to bypass the R columns of an $(N+R) \times (N+R)$ array. Bypassing a column is defined as removing the column from an array and is realized by changing all PE states in the column into *PassH*. It seems that bypassing is an efficient strategy for clustered faults where many faults concentrate on one column. After bypassing the R columns, the $(N+R) \times (N+R)$ array is reduced to an $(N+R) \times N$ array. To explain the bypassing algorithm more clearly, some definitions are necessary.

Definition 1 $F_{in}[i,j]$ and $F_{out}[i,j]$ are defined as follows, which are an input value and an output value of PE$[i,j]$ respectively.

$$\begin{aligned} F_{in}[i,j] &= \begin{cases} \sum_{k=1}^{i-1} BIST[k,j] & \text{where } 2 \le i \le N+R, \\ 0 & \text{where } i = 1. \end{cases} \\ F_{out}[i,j] &= F_{in}[i,j] + BIST[i,j]. \end{aligned}$$

It is clear that $F_{out}[N+R,j]$ corresponds to the total number of faulty PEs on the j-th column. Then the bypassing algorithm is shown as follows.

1. Obtain the total number of faulty PEs $F_{out}[N+R,j]$ at every columns.
2. Bypass the R columns in order of $F_{out}[N+R,j]$ as follows.
   - 2.1 Bypass columns including $F_{out}[N+R,j] > R$.
   - 2.2 Let $REMAIN$ be the number of remaining columns. PE$[N+R,j]$ accumulates 1 to $REMAIN$ if the column is not bypassed and transfers $REMAIN$ to the adjacent PE$[N+R,j+1]$. Then the lower-right PE$[N+R,N+R]$ obtains the total number of remaining columns.
   - 2.3 According to $REMAIN$, the following steps are performed.
     - For $REMAIN = N$, the bypass algorithm is terminated with success.
     - For $REMAIN < N$, the reconfiguration fails.
     - For $REMAIN > N$, bypass just $(REMAIN - N)$ columns repeatedly in order of $F_{out}[N+R,j]$ as shown in Figure 2.

```
begin
  num := REMAIN − N;
  comp := R;
  while num > 0 do
    begin
      for j := 1 to N + R do
        if F_out[N + R, j] = comp then
          begin
            Bypass the column;
            comp := −1
          end;
        if comp = −1 then
          num := num − 1;
        else
          comp := comp − 1
    end
end.
```

Figure 2. The bypassing algorithm of process 2.3.

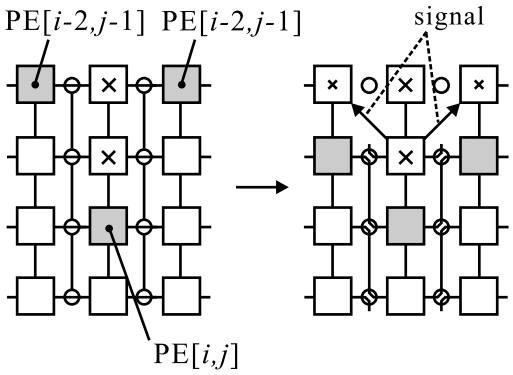

Figure 3. Deactivating PEs by signals.

3.2. Deactivating PEs

Faulty PE[i,j]s must be deactivated and replaced by other healthy PEs. By replacing faulty PEs with healthy PEs, some healthy PEs also are deactivated as shown in Figure 3. In Fig.3, the PE[i,j] is unable to connect with the PE[i-2,j-1] and the PE[i-2,j+1] for T=1. In this case, PE[i-2,j-1] and PE[i-2,j+1] are deactivated and compensated by PE[i-1,j-1] and PE[i-1,j+1] in order to connect with PE$[i, j]$. The objective of $step2$ is to remove all such disconnection. In Fig.3, PE[i-1,j] detects the disconnection and transfers deactivation signals to both PE[i-2,j-1] and PE[i-2,j+1]. Each PE has a variable $DEACT$[i,j] which has a value of 1 (deactivated) or 0 (not deactivated). Now that the variable $DEACT$[i,j] has been defined, all PEs which have the variable $FAULT$[i,j] = $BIST$[i,j] OR $DEACT$[i,j] = 1 are treated as faulty.

Definition 2 The variable S_{in}[i,j] and S_{out}[i,j] are defined by replacing $BIST$[i,j] in Definition 1 with $FAULT$[i,j] to take $DEACT$[i,j] into account.

The algorithm of *step 2* of the BC method performs the following two processes in all PEs independently.

- PE[i, j] transfers a signal to both PE[i-1,j-1] and PE[i-1,j+1] with following condition.
  S_{in}[i,j] > 1 and $FAULT$[i,j]=1.
- PE[i,j] which received signals changes $DEACT$[i,j] into 1 in order to be deactivated with following conditions.
  S_{in}[i+1,j-1] - S_{in}[i,j] > 1 (received signal from PE[i+1, j-1]),
  S_{in}[i+1,j+1] - S_{in}[i,j]> 1 (received signal from PE[i+1,j+1]).

3.3. Changing switch functions

After bypassing and deactivating, each switch function can be changed from the initial function EW into another function at the same time. Figure 4 shows the rule used to change a switch function using the local information of $FAULT$[i,j], $FAULT$[i,j+1], S_{in}[i,j] and S_{in}[i,j+1]. This rule enables the BC method to be implemented into hardware easily, namely Fig.4 can be designed using simple circuits (shown in Figure 9 later).

The example of all switch functions of Fig.4 are illustrated in Figure 5 which shows the example

```
begin
  if S_in[i,j] > S_in[i,j+1] then
    SW[i,j] := NW; ··· (1)
  else if S_in[i,j] < S_in[i,j+1] then
    SW[i,j] := NE; ··· (2)
  else if S_i[i,j] = S_in[i,j+1] then
    begin
      (a, b) := (FAULT[i,j],FAULT[i,j+1]);
      if (a, b) = (0, 0) then
        SW[i,j] := EW; ··· (3)
      else if (a, b) = (1, 0) then
        SW[i,j] := NW; ··· (4)
      else if (a, b) = (0, 1) then
        SW[i,j] := NE; ··· (5)
      else if (a, b) = (1, 1) then
        SW[i,j] := NC ··· (6)
    end
end.
```

Figure 4. The rule to determine switch state.

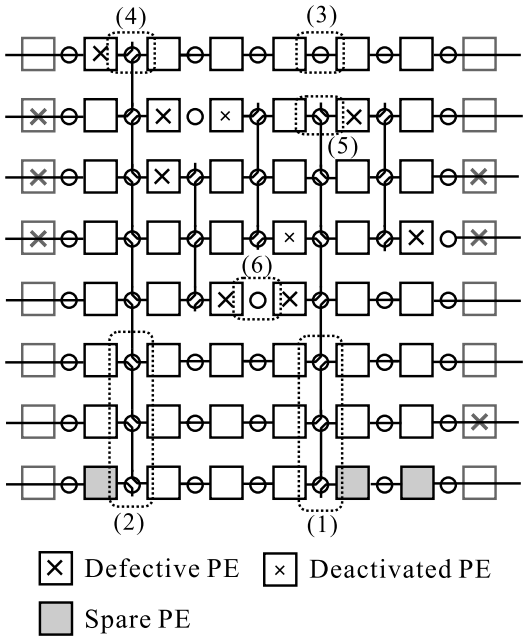

Figure 5. The reconfiguration example for 6-2-1 type.

of reconfiguration for a 6-2-1 type architecture. Each number in Fig.5 corresponds to the number in Fig.4 and shows example situations. In this reconfiguration example, two columns on the far right side and left side are bypassed.

4. Performance Evaluations

There are several performance measures to evaluate the effectiveness of fault tolerance systems. This paper concentrates on the system yield and the reconfiguration time as figures of merit. We assume the following similar to previous studies [1]-[6], that each PE fails with the same probability and the tracks, switches and detection circuit are fault free.

4.1. Array yield

A random fault model is commonly used in previous studies using the assumption that each PE fails with the same probability, PE yield. Array yield is defined by the probability to obtain an $N \times N$ array from an $(N+R) \times (N+R)$ array using reconfiguration methods. Figure 6 shows the array yields performance of the 20-R-1 type architecture as a function of PE yield. Here array yields of BC and BS are obtained for R=2, 4, 6. Each array yield is the average of 1000 simulation results. The BC method achieves almost the same array yields as the BS method with recursive shifts in each type of architecture. The solid line in Fig.6 indicates the yield of a $1\frac{1}{2}$ track switch model for 20$\times$ 20 which is obtained by Kung[1] under the same assumption. Kung's method achieves higher array yields than the other two methods when PE yield is low. Taking into account that Kung's method obtains the complete solution, the BC method can achieve the same yield of Kung's method using only local defect information.

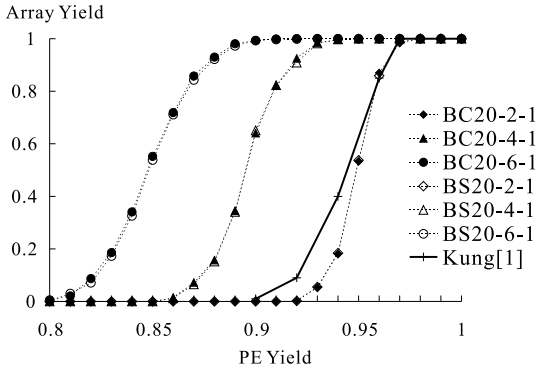

Figure 6. Array yields for 20-R-1 mesh arrays.

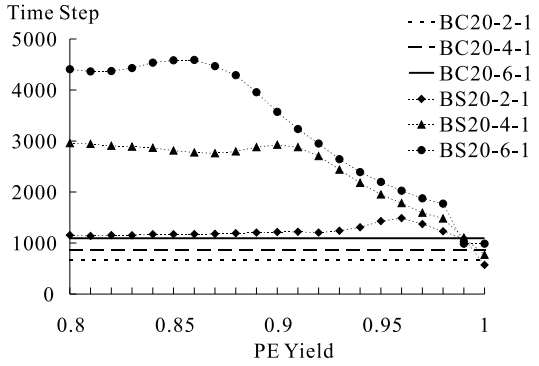

Figure 7. Reconfiguration times for BC and BS methods.

The BC method as well as the BS method can enhance its array yields by increasing the number of spare rows R. The array yield of the BC method is improved significantly for the cases of R=4 and R=6. On the other hand, since the number of spare rows R in Kung's method depends on the number of tracks T, it can treat only R=1 when T=1.

4.2. Reconfiguration time.

Next we discuss the execution times involved in reconfigure mesh arrays by the BC method and the BS method. The reconfiguration time is defined as the total steps to reconfigure the $N \times N$ array from the $(N+R) \times (N+R)$ array. First, we estimate the worst reconfiguration time of the BC method consisting of bypassing(*step 1*), deactivating(*step 2*) and changing the switches(*step 3*). *Step 1* is performed even if there are no defects in the array. In this case, $(N+R)$, $(N+R)$, $(N+R) \times 2R$ time steps are required in process 1, 2.2, 2.3 of bypassing respectively. Hence the worst time of bypassing for the BC method is estimated as $(N+R)(2R+2)$ time steps. In $Step2$ of the BC method, if one signal deactivates all PEs (such a situation would never occur), the maximum cost is $(N+R) \times (N+R)$ time steps. $Step3$ changes all switch functions at a time, so it requires only 1 time step for any fault distributions. According to the above estimation the worst reconfiguration time of the BC method $T_{BC}(N, R)$ is estimated as follows.

$$T_{BC}(N, R) = (N+R)(2R+2)+(N+R)(N+R)+1.$$

On the other hand the reconfiguration time of the BS method depends on defect distributions, so it is evaluated by simulation experiments assuming that each signal used in the method can be processed within 1 time step ideally. Figure 7 plots the time $T_{BC}(20, R)$ estimated above and the reconfiguration time of the BS method for the same array size in Figure 6. As the number of spare rows R increases, the reconfiguration time of the BS method increases significantly, however, the increase of $T_{BC}(20, R)$ is small. It is shown that the BC method performs reconfiguration much faster then the BS method.

5. A self-reconfiguration system on FPGA

5.1. System Board

The self-reconfigurable mesh array using BC method has been designed and simulated by using the MAX+PlusII, ALTERA. The system was implemented in the EPF10K250AGC599-3 (250000

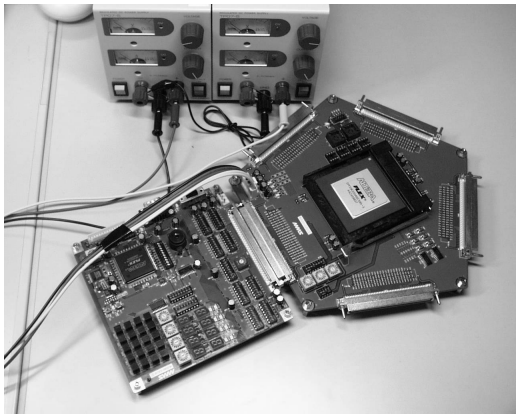

Figure 8. The hardware instruments.

gates) on the MEB200-A250 and MU200-EA10, Mitsubishi Electronic Micro-computer Application Software Co. LTD. These instruments are shown in Figure 8.

5.2. Designed hardware circuits

5.2.1. Switch circuits: Figure 9 shows the switch circuit in which control part performs to determine switch functions in Fig.4. According to the desired function, each port (8bit width) in Fig.9 is connected to one of three other ports (excluding north port and south port). The hardware cost of this circuit is evaluated as 85 LCs (Logic Cell) using the MAX+plusII design tool. Since 1 LC corresponds to about 20 logical gates [7], its hardware cost is estimated as about 1700 gates.

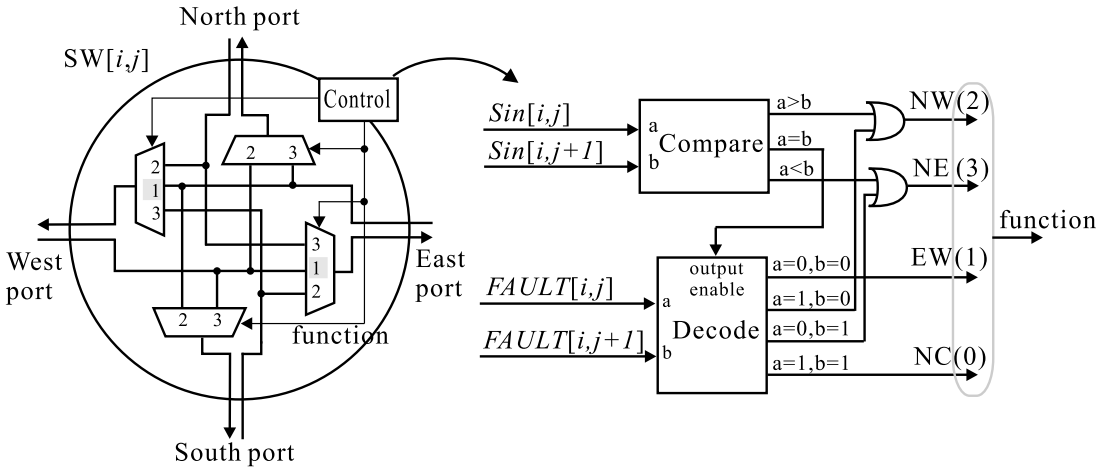

Figure 9. A switch circuit.

5.2.2. PE: The outline of PE was shown in Fig 1. To evaluate the self-reconfigurable mesh array, PE is designed to execute the functions required in the BC method. PE includes four switch circuits, a circuit to accumulate $S_{out}[i,j]$, a circuit to generate signals, a circuit to deactivate PE and so on. Flip-flops to hold $BIST[i,j]$, $FAULT[i,j]$, physical address and logical address are

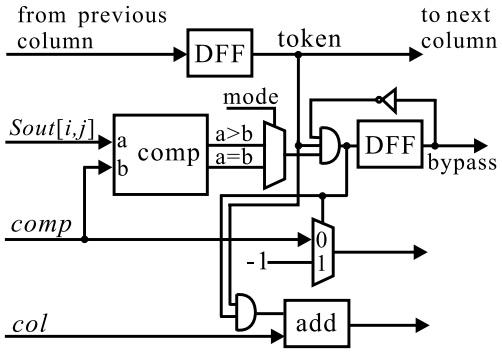

Figure 10. A component of bypass control circuit.

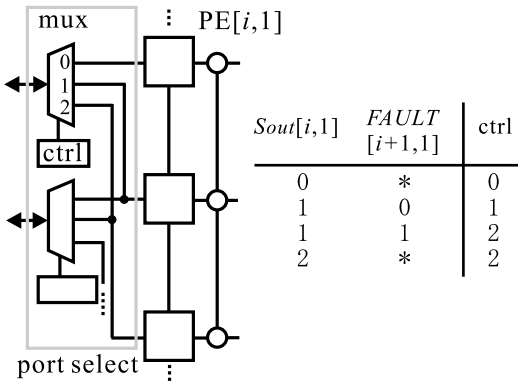

Figure 11. Multiplexers in port select.

also required. These circuits are designed using 108 LCs, about 2160 gates.

5.2.3. Bypass control circuit: Figure 10 shows a component of the bypass control circuit. The prototype consists of $(N+R)$ components and allocate at the bottom of each column. Since bypassing is performed by a synchronizing token, an upper DFF(D-type flip flop) configures a shift-register in order to provide tokens. A comparator compares $F_{out}[N+R,j]$ with $comp$ (explained in Fig.3) and executes the bypass algorithm. A component of the bypass control circuit includes 26 LCs, that is about 520 gates.

5.2.4. Port select circuit: Figure 11 shows a port select circuit consisting of N multiplexers and it is used to connect all logical rows (columns) with proper I/O ports. In Fig.11, PE[i,1] on the most west column can be shifted into PE[$i+R$,1], so one multiplexer selects one port from $(R+1)$ ports. A port select circuit is designed using 37 LCs, that is about 740 gates.

5.2.5. 6-2-1 Type Mesh Array: A 6-2-1 type system is designed and simulated in order to estimate the hardware complexities and the speed of reconfiguration. Some defect patterns are also embedded into PEs. The following three methods can be selected to observe if reconfiguration is successful or not.

1. Check the output data at all I/O points due to the input data.
2. Check all the switch functions.
3. Check the mapping between all physical addresses and logical addresses.

In all three steps, the expected results are observed for each defect patterns. The BC method can be implemented on FPGA using the system clock 20MHz.

5.3. Discussion of hardware overhead

The hardware overheads of the reconfigurable mesh array are discussed here. The Hardware costs of each component are already shown above. Let $G(N, R, P)$ be the entire hardware cost of the $(N+R) \times (N+R)$ mesh array. $G(N, R, P)$ is estimated as follows.

$$G(N,R,P) = 1700(N+R-1)(N+R)+(2160+P)(N+R)(N+R)+520(N+R)+740\times 4N,$$

Table 1. Overhead rate of switches for 6-2-1 type.

PE (gates)	array (gates)	$O_{switch}(6,2,P)$
5000	581320	0.45
10000	901320	0.29
50000	3461320	0.075
100000	6661320	0.039

where P is the number of gates for a PE.

Let's define $O_{switch}(N,R,P)$ as a hardware overhead rate of the switching circuits to the redundant mesh array.

$$O_{switch}(N,R,P) = \frac{G(N,R,0)}{G(N,R,P).}$$

The $O_{switch}(N,R,P)$ includes the overhead of control circuits, port selects and switches in PE etc. Table 1 shows the overhead rate of switches $O_{switch}(N,R,P)$ for a 6-2-1 type. The $O_{switch}(N,R,P)$ is a small when the hardware of a PE is relative large (P=50000, 100000), while it is unacceptable rate when the hardware of a PE is relative small size. Note that the $O_{switch}(N,R,P)$ is evaluated under the assumption of the bus 8 bits width. If the bus width is reduced to 1 bit, the $O_{switch}(N,R,P)$ will show lower rates.

6. Conclusion

The reconfiguration process to avoid defective processors is one of the most important technological issues in the design of massively parallel computers. This paper proposed the self-reconfigurable architecture of mesh array and the self-reconfigurable algorithm (BC Method). The BC method changes each switch function at a time using only local information. It is seen by the performance evaluation that the BC method achieves much higher array yield than Kung's method using sufficient spare PEs and much shorter reconfiguration times than those of BS method. A prototype system is also designed and implemented on FPGA chips and its hardware complexities are evaluated.

References

[1] S. Y. Kung, S. N. Jean and C. W. Chan, "fault-tolerant array processors using single-track switches", IEEE Trans. Computers, Vol.38, No.4, pp.501-514, 1989.

[2] V. P. Roychowdhury, J. Bruck, and T. Kailath, "Efficient algorithms for reconstruction in VLSI/WSI array", IEEE Trans. Computers, Vol.39, No.4, pp.480-489, 1989.

[3] T. A. Varvarigou, V. P. Roychowdry, T. Kailath, "A polynomial time algorithm for reconfiguring multiple-track models", IEEE Trans. Computers, Vol.42, No.4, pp.385-395, 1993.

[4] I. Numata and S. Horiguchi, "A Self-Reconfiguration Architecture for Mesh Arrays", International Workshop on Defect and Fault Tolerance in VLSI Systems, pp.212-220, Oct, 1994.

[5] I. Numata and S. Horiguchi, "Wafer-scale Integration Implementation of Mesh-Connected Multiprocessor Systems", Systems and Computers, Vol.26, No.1, 1995.

[6] I. Takanami, K. Kurata, and T. Watanabe, "A neural algorithm for reconstructing mesh-connected processor arrays using single-track switches", Int'l Conf. on WSI, pp.501-514, Jan, 1995.

[7] ALTERA data sheet, "FLEX 10K Embedded Programmable Logic Family Data Sheet, ver4.01", 6, 1999.