社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2009-59,DC2009-46(2009-11)

Increasing Yield Using Partially-Programmable Circuits

山下　　茂[†]　　吉田　浩章[††]　　藤田　昌宏[††]

[†] 立命館大学 情報理工学部　〒 525-8577 草津市 野路東 1-1-1
[††] 東京大学 大規模集積システム設計教育研究センター (VDEC)
〒 113–8656 東京都文京区本郷 7–3–1
E-mail: †ger@cs.ritsumei.ac.jp, ††hiroaki@cad.t.u-tokyo.ac.jp, †††fujita@ee.t.u-tokyo.ac.jp

あらまし　本論文では，回路の一部を LUT に置き換えた *Partially-Programmable Circuits*（PPCs）と名付けた回路を用いることで製造歩留まりを向上させる手法を提案する．PPC の中のいくつかの LUT の機能を変更させせすることにより，ある結線が冗長となる場合, その結線は「故障に強い」と考えられる．その理由は，その結線にいかなる故障が起こってもいくつかの LUT の内部論理の変更により回路を正常に動作させることが可能だからである．そのような故障に強い結線を増やすために，SPFD や CSPF といった論理関数の自由度を利用して冗長な結線を回路に追加する手法を提案する．簡単な予備実験から，我々のアプローチは有望であると確認できた.

キーワード　yield, LUT, Partially-Programmable Circuit (PPC), SPFD

Increasing Yield Using Partially-Programmable Circuits

Shigeru YAMASHITA[†], Hiroaki YOSHIDA[††], and Masahiro FUJITA[††]

[†] College of Information Science and Engineering, Ritsumeikan University
1-1-1 Noji Higashi, Kusatsu, Shiga 525-8577, Japan
[††] VLSI Design and Education Center(VDEC), University of Tokyo
7–3–1 Hongo, Bunkyo-ku, Tokyo, 113–8656, Japan
E-mail: †ger@cs.ritsumei.ac.jp, ††hiroaki@cad.t.u-tokyo.ac.jp, †††fujita@ee.t.u-tokyo.ac.jp

Abstract　This paper proposes to use *Partially-Programmable Circuits* (PPCs) which are obtained from conventional logic circuits by replacing their sub-circuits with LUTs. If a connection in an PPC becomes redundant by changing the functionality of some LUTs, the connection is considered to be robust to defects because even if there are some defects at the connection, the circuit works properly by changing the functionality of some LUTs appropriately. To increase the number of such robust connections, we add some redundant connections to LUTs. We find such redundant connection by using functional flexibility represented by SPFDs and/or CSPFs. From the result of our preliminary experiments, we consider our approach is promising.

Key words　yield, LUT, Partially-Programmable Circuit (PPC), SPFD

1 Introduction

It has been considered that there will be a sharp increase in manufacturing defect levels in future electronic technologies [1] [2]. Thus, there has been a considerable interest in practical techniques to increase the yield of LSIs [3]〜[5]. At the logic design level, a general method is to add *space* redundancy, e.g., Triple Modular Redundancy (TMR). If we are considering only manufacturing defects, another possible way to increase the yield is Double Modular Redundancy (DMR); we select a module without defect between the two identical modules after the LSI test. If we use FPGAs, we can have much more ways to bypass the defects after the LSI manufacture [6].

However, the above methods obviously have disadvantages: area overhead and/or performance degradation. In this paper, we propose a totally different approach to increase the yield with (possibly) lower overhead: we make a logic circuit from conventional logic circuits by replacing its sub-circuits with Look-Up Tables (LUTs) and Multiplexers (MUXs). Then, if we detect some defects (by the LSI test) in it, we reconfigure the functionality of some of LUTs and MUXs to bypass the defects. We call such circuits *Partially-Programmable Circuits* (PPCs) since some of their parts are programmable. Compared to the DMR scheme, our approach does not duplicate a target logic circuit entirely, but

Copyright ©2009　by　IEICE

only replaces some parts of the circuit into LUTs. Therefore, the area and performance overhead could be small. Obviously we cannot use a PPC like an FPGA to reconfigure the functionality at any time; a PPC can only be reconfigured to bypass some (not all) defects, and thus the area/performance overhead would be lower than an FPGA.

In this paper, we suppose a single defect at a connection, e.g., stuck-at-0/1 fault. Then, we say a connection is *robust* to stuck-at-0 or stuck-at-1, if we can bypass the fault (by reconfiguring the circuit) to stuck-at-0 or stuck-at-1, respectively. Note that if a connection is robust to both stuck-at-0 and stuck-at-1, we can deal with any situation where the logic value at the connection becomes incorrect value, and we call such a connection simply robust. Note also that we can bypass any defect at a logic gate if its fanout connections are all robust. In this paper, we mainly consider the ratio of robust connections in a designed circuit because the ratio is obviously related to the yield. The ratio is 100% for the above DMR scheme since one of the duplicated modules should work correctly if there is only a single fault in the circuit. On the other hand, since our scheme does not use as much overhead as the DMR scheme, we cannot expect such a high ratio; even so we can expect that our scheme achieve a reasonably high ratio with relatively low overhead as our preliminary experiments show.

To increase the ratio of robust connections in a designed PPC, our proposed method is to add some redundant connections in advance. To find appropriate redundant wires, our circuit transformation utilizes the notion of SPFD (Set of Pairs of Functions to be Distinguished) [7] and CSPFs (Compatible Sets of Permissible Functions) [8].

In the reminder of this paper, first we review the notion of SPFDs and CSPFs in Section 2. Next we provide an overview of our scheme in Section 3 by using a motivational example. Then we propose a circuit transformation method in Section 4. We then provide some preliminary experimental results in Section 5 followed by our conclusions in Section 6.

2 Preliminaries

2.1 Basic Terminologies

We use the following notation in this paper.

- N_i represents a node in a network.
- $C_{[i,j]}$ represents the connection from N_i's output to one of N_j's inputs. If there is $C_{[i,j]}$, N_i is called a *fanin* of N_j, and N_j is called a *fanout* of N_i. Additionally, if there is a path from node N_i to node N_j, node N_i is called a *transitive fanin* of node N_j, and node N_j is called a *transitive fanout* of node N_i.
- f_i: represents the logic function at the output of N_i with respect to the primary inputs of the network. This is sometimes called the "global" function of the node.

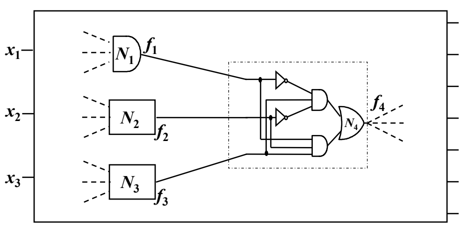


Figure 1 A Boolean network to explain CSPFs/SPFDs.

We use the "functional flexibility" of logic functions:

- The condition in which an alternative function can replace the global function at the output of N_i is called the "*functional flexibility* of N_i."
- The condition in which an alternative function can replace the global function used for an input to N_j, which corresponds to $C_{[i,j]}$, is called the "*functional flexibility* of $C_{[i,j]}$."

Note that the functional flexibility of $C_{[i,j]}$ generally differs from that of $C_{[i,k]}$ although the global logic functions corresponds to $C_{[i,j]}$ and $C_{[i,k]}$ are the same as f_i.

To represent the functional flexibility for a conventional gate, we usually use an incompletely specified function whose ON-Set, OFF-Set and DC (Don't Care)-Set correspond to the input patterns, where the function must become 1, 0, and don't care. There are many calculation methods for such functional flexibility, e.g., *satisfiability don't cares (SDC)* [9], *observability don't cares (ODC)* [9], *compatible observability don't cares (CODC)* [10], [11], and *compatible sets of permissible functions (CSPFs)* [8]. In this paper, we use CSPFs among them, but our procedure can be modified to work with any one of methods to calculate functional flexibility.

To represent the functional flexibility for LUTs, we can have much more freedom because we can change the functionality of LUTs. To utilize such flexibility, we use *Sets of Pairs of Functions to be Distinguished (SPFDs)* [7].

The following notations are used for them:

- $SPFD_i$ ($CSPF_i$) is an SPFD (CSPF) that represents the functional flexibility of N_i, respectively.
- $SPFD_{[i,j]}$ ($CSPF_{[i,j]}$) is an SPFD (CSPF) that represents the functional flexibility of $C_{[i,j]}$, respectively.

Note that $SPFD_i$ and $SPFD_{[j,i]}$ implicitly assume that N_i is an LUT since we do not use SPFDs for input connections of conventional gates. $SPFD_i$ ($CSPF_i$) and $SPFD_{[i,j]}$ ($CSPF_{[i,j]}$) are sometimes referred to simply as "the SPFD (CSPF) of the node" and "the SPFD (CSPF) of the connection," respectively.

In the following, we review the basic concept of CSPFs and SPFDs.

2.2 CSPF

Figure 2 shows an example for CSPFs [8] for the network shown in Figure 1. For simplicity, we suppose that the net-

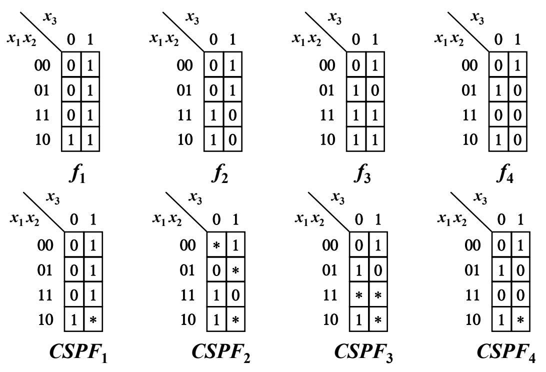


Figure 2 Functional flexibility by CSPFs.

work has only three primary inputs x_1, x_2, and x_3. If f_1, f_2, and f_3 are shown as in Figure 2, internal logic of N_4 is f_4 is calculated by $(f_1 \cdot f_2 \cdot f_3 + \overline{f_1} \cdot \overline{f_2} \cdot f_3)$ as shown in Figure 2. Then, if the functional flexibility of N_4 is represented by $CSPF_4$ in Figure 2, the functional flexibilities of C_1, C_2, and C_3 are calculated as $CSPF_1, CSPF_2$, and $CSPF_3$ in Figure 2, respectively, by the method in the paper [8]. In the example, as long as f_1, f_2, and f_3 change within the flexibilities represented by $CSPF_1, CSPF_2$, and $CSPF_3$, it is guaranteed that f_4 changes within the flexibility represented by $CSPF_4$.

In this paper, we represent a CSPF as (f_{ON}, f_{OFF}) when its ON-Set and OFF-Set are f_{ON} and f_{OFF}, respectively.

2.3 SPFD

To discuss SPFDs, we need to define what it means **to distinguish a pair of functions**.

[Definition 1] For two logic functions f and g, if $f(X)$ always becomes 1 for all primary input vectors X where $g(X)$ becomes 1, f is said to **include** g, also written $g \leqq f$, or $g \Rightarrow f$. Note that "f includes g" $\Leftrightarrow g \cdot \overline{f} = 0$.

[Definition 2] If either one of the following two conditions is satisfied, a function f is said to **distinguish** a pair of functions g_1 and g_2.

- f includes g_1, and $\overline{f}$ includes g_2. ($g_1 \leqq f \leqq \overline{g_2}$.)
- f includes g_2, and $\overline{f}$ includes g_1. ($g_2 \leqq f \leqq \overline{g_1}$.)

In other words, f is said to distinguish g_1 and g_2 when one of ON-Set and OFF-Set of f includes g_1 and the other includes g_2.

Now, we can formalize an SPFD as follows.

[Definition 3] **SPFD (Set of Pairs of Functions to be Distinguished)** is a set of pairs of functions that can be represented as $\{(g_{1a}, g_{1b}), (g_{2a}, g_{2b}), \cdots, (g_{na}, g_{nb})\}$.

We also define the following terminology.

[Definition 4] For an SPFD $= \{(g_{1a}, g_{1b}), (g_{2a}, g_{2b}), \cdots, (g_{na}, g_{nb})\}$, a function f is said to **satisfy the SPFD's condition** or simply **satisfy the SPFD** if f distinguishes g_{ia} and g_{ib} for all i.

Note that it is implicitly assumed that $(g_{ia} \cdot g_{ib} = 0)$ in the above SPFD; otherwise, no function can distinguish g_{ia} and g_{ib}.

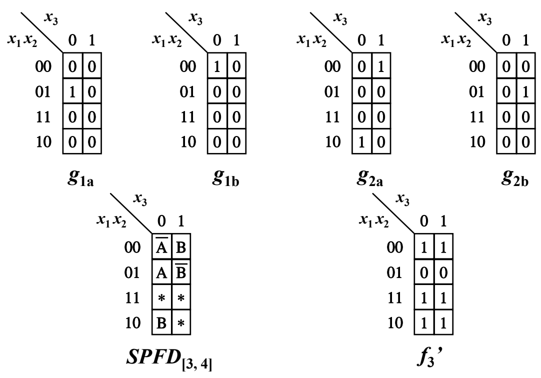


Figure 3 Functional flexibility by an SPFD.

In the example shown in Figures 1 and 2, suppose we implement the sub-circuit shown as the dotted rectangle into one LUT which is called N_4. Let also the functional flexibility of N_4 be the same as $CSPF_4$ in Figure 2. Then, the algorithm in [7] calculates $SPFD_{[3,4]}$ as $\{(g_{1a}, g_{1b}), (g_{2a}, g_{2b})\}$, where g_{1a}, g_{1b}, g_{2a} and g_{2b} are as shown in Figure 3.

The functions that satisfy $SPFD_{[3,4]}$ are functions whose intuitive truth tables are as follows.

- the values corresponding to A and $\overline{A}$ in the truth table shown as "$SPFD_{[3,4]}$" in Figure 3 must be different (one is 1 and the other is 0).
- the values corresponding to B and $\overline{B}$ in the truth table shown as "$SPFD_{[3,4]}$" in Figure 3 must be different (one is 1 and the other is 0).

Therefore, we can see that f_3' in Figure 3 satisfies $SPFD_{[3,4]}$. Since f_3' is not included in either $CSPF_{[3,4]}$ in Figure 2 or the simple negation of $CSPF_{[3,4]}$, we cannot replace f_3 with f_3' if we use $CSPF_{[3,4]}$ to represent the functional flexibility.

2.4 Calculation of SPFDs and CSPFs

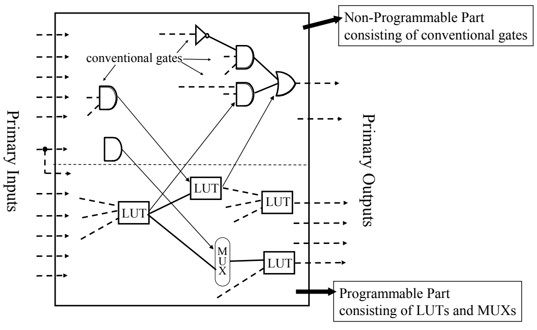
In our method, we calculate SPFDs and CSPFs by the original methods [7] and [8], respectively. They are calculated from the primary outputs toward the primary inputs as follows:

- For a primary output node N_i, set $SPFD_i$ as $\{(f_i, \overline{f_1})\}$, and $CSPF_i$ as $(f_i, \overline{f_1})$.
- For node N_i, we can calculate $SPFD_i$ ($CSPF_i$) by merging all the functional flexibility of the fanout connections of N_i. We refer the readers to the original papers [7] and [8] for the detail.
- From $SPFD_i$ ($CSPF_i$), we can calculate the SPFDs (CSPFs) of the fanin connections of N_i by the methods in [7] and [8], respectively.

3 Overview of our proposed scheme

3.1 A motivational example

In our proposed scheme, we implement a combinatorial logic circuit that has the following two parts: (1) Non-programmable part consisting of conventional gates, and (2) programmable part consisting of LUTs and MUXs. We call


Non-Programmable Part consisting of conventional gates
conventional gates
Primary Inputs
Primary Outputs
LUT
LUT
LUT
LUT
MUX
Programmable Part consisting of LUTs and MUXs
Figure 4 A partially-programmable circuit

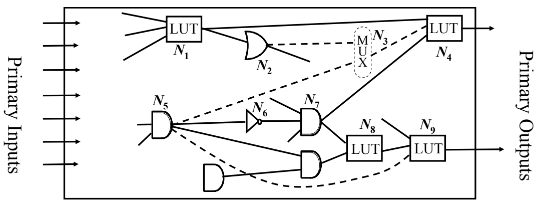


Figure 5 A Boolean network.

this kind of circuits specifically as *Partially-Programmable Circuits* (PPCs). Figure 4 shows an example. Some of the primary inputs may be connected to the both parts, and each primary output can be from either part. Also there may be connections from the programmable part to the non-programmable part, and/or vice versa.

A PPC can be described as a Boolean network (as shown in Figure 5) where we can have the following three types of nodes:

- Conventional logic gates, e.g., AND/OR/NOT.
- LUTs whose internal functionality can be reconfigured.
- MUXs whose select lines are controlled by programmable memory cells.

Suppose we can notice the following facts for the circuit (without dotted lines) in Figure 5. (We will explain how we can know that below.)

- If we add a connection from N_5 to N_9, and then reconfigure the functionality of LUTs N_8 and N_9 appropriately, the connection from $C_{[7,8]}$ becomes redundant.
- If we add a connection from N_5 to N_4, and then reconfigure the functionality of LUT N_4 appropriately, the connection from $C_{[7,4]}$ becomes redundant.
- If we add a connection from N_2 to N_4, and then reconfigure the functionality of LUT N_4 appropriately, the connection from $C_{[1,4]}$ becomes redundant.

Therefore, in the above situation, our scheme adds the dotted connections and MUX N_3 as shown in the figure. Then, we can bypass one of the following defects.

- If there is a defect at the connection from $C_{[7,8]}$, we can bypass the defect by reconfiguring the functionality of N_8 and N_9 appropriately.
- If there is a defect at the connection from $C_{[7,4]}$, we can bypass the defect by selecting the connection from N_5 at N_3, and reconfiguring the functionality of N_4 appropriately. A stuck-at-fault occurs at the same time on the both $C_{[7,8]}$ and $C_{[7,4]}$ because they are connected. Nevertheless, we assume a defect may occur on one of the connections independently, because there is, for example, an open fault.
- If there is a defect at any input of N_7 (or even a defect in N_7 itself), we can bypass the defect by the above two adjustments.
- If there is a defect at the connection from $C_{[1,4]}$, we can bypass the defect by selecting the connection from N_2 at N_3, and reconfiguring the functionality of N_4 appropriately.

Accordingly, in the above circuit (with the added dotted lines), $C_{[7,8]}$, $C_{[7,4]}$, $C_{[1,4]}$ and the input connections to N_7, $C_{[5,6]}$ and $C_{[6,7]}$ as well as the added connections ($C_{[5,9]}$, $C_{[5,3]}$, $C_{[2,3]}$ and $C_{[3,4]}$). Note that we do not need the two input connections of N_4 from N_2 and N_5 at the same time because we suppose a single defect in the circuit. Therefore, we do not increase the number of inputs of N_4 which results to an area penalty; we just put a MUX before N_4 to select a necessary input between the two inputs depending on a defect.

3. 2 Intuitive difference from DMR

Note that all the robust connections in our scheme are not redundant at the same time, because we can bypass only a single fault at one of the robust connections at one time, i.e., we may not deal with multiple faults at the same time and thus they cannot be redundant at the same time. On the other hands, all the connections in one of the duplicated modules are all redundant at the same time in nature in the DMR scheme. This is a key observation of the difference between our scheme and the DMR scheme; the overhead of our scheme may be much lower because we considers only a single fault in a circuit. Even so we can expect to increase the yield greatly because the probability of having a single fault should be much larger than having multiple faults in standard LSI manufacturing. Thus, we expect our scheme may provide a better trade-off point between the area/performance overhead and the yield than the simple DMR scheme.

3. 3 Problem Formulation

Now it should be clear what we want to do in this paper:

Our Problem.

Given a conventional combinatorial circuit, our problem is to design a PPC of the same functionality so that the ratio of robust connections is as high as possible.

4 Our procedure to increase robust connections

4. 1 Conversion between SPFDs and CSPFs

Unlike the previous SPFD-based circuit transformation method [7], we need to calculate both SPFDs and CSPFs at the same time because our circuits contain LUTs and conventional gates. Thus we need to convert between SPFDs and CSPFs during the above calculation. This can be done as follows:

SPFD to CSPF. Suppose N_j is an LUT, and one of its fanin, N_i, is a conventional gate. Then, we can calculate the functional flexibility of $C_{[i,j]}$ as an SPFD, but in order to propagate it toward the inputs of N_i we need to convert it to a CSPF. Without loss of generality, let $SPFD_{[i,j]} = \{(g_{a_1}, g_{b_1}), (g_{a_2}, g_{b_2}), \cdots, (g_{a_m}, g_{b_m})\}$, where f_i includes g_{a_k} and $\overline{f_i}$ includes g_{b_k} for all k. Then, we convert the above SPFD to $CPFD_{[i,j]} = (f_{ON}, f_{OFF})$ where $f_{ON} = \sum_m g_{a_m}$ and $f_{OFF} = \sum_m g_{b_m}$. Note that any function included in $CPFD_{[i,j]}$ satisfies $SPFD_{[i,j]}$.

CSPF to SPFD. If N_j is a conventional gate, the functionality of each fanin connection of N_j is calculated as a CSPF. Then, if one of its fanin node, N_i, is an LUT, we need to convert $CPFD_{[i,j]} = (f_{ON}, f_{OFF})$ to an SPFD to proceed the above calculation of the functional flexibility in a circuit. This is easy; we just let $SPFD_{[i,j]} = \{(f_{ON}, f_{OFF})\}$.

4. 2 Our proposed method to generate PPCs

Now we are ready to explain our proposed procedure to increase robust connections in a given circuit.

In our procedure, we first generate an initial PPC from a given circuit as follows. We assume to use k-input LUTs in the final implementation.

Generation of an initial PPC.

Step. 1 Map the given circuit into a $(k-1)$-input LUT networks. This can be done by any technology mapper.

Step. 2 We select some of the LUTs by some heuristics. Then, we restore the original conventional gates for unselected LUTs.

It is desirable that LUTs in a PPC have the following features:

Feature 1: An LUT should be useful to increase the number of robust connections, i.e., some connections becomes redundant if we reconfigure the functionality of the LUT.

Feature 2: An LUT does not degrade the circuit performance too much.

Considering Feature 1, we consider it to be appropriate to select the following LUTs:

- An LUT which is near to the primary outputs.
- An LUT which is on a re-convergent point in the circuit.

Considering Feature 2, we can consider a heuristic should not select an LUT which is on a critical path.

After generating initial PPC (without MUXs), we add redundant connections to increase robust connections in the PPC by the procedure as shown in Figure 6. We explain the procedure by using Figure 7. In the procedure, we try to make each connection, $C_{[i,j]}$, in the given circuit robust in the outer loop (line 1 to 25). Suppose we are now proceeding $C_{[i,j]}$ in the circuit in Figure 7. In the loop between lines 2 and 10, we calculate CSPFs/SPFDs of the transitive fanout nodes of N_j (the nodes in the dotted area in the figure). We select each node, N_m, in the area from the primary outputs towards N_j. Then, if N_m is an LUT as the figure, we try to add a redundant connection at lines 4 and 5. By adding an connection, we expect $C_{[i,j]}$ becomes redundant. After processing all the transitive fanout nodes of N_j, we finish the loop between lines 2 and 10.

```
1: for $C_{[i,j]} \leftarrow$ each connection in the PPC do
2:   for $N_m \leftarrow$ the transitive fanout of $N_j$ in the order of
     SPFD/CSPF calculation do
3:     if $N_m$ is an LUT then
4:       Find $N_p$ such that $f_p$ contributes to make the functional
         flexibility of $C_{[l,m]}$ larger at the calculation of SPFDs where
         $N_l$ is a fanin of $N_m$ and a transitive fanout of $N_j$.
5:       Add a redundant connection $C_{[p,m]}$.
6:       Propagate SPFDs through $N_m$.
7:     else
8:       Propagate CSPFs through $N_m$.
9:     end if
10:  end for
11:  if $C_{[i,j]}$ is redundant by judging from the calculated functional
     flexibility by the above then
12:    for $C_{[q,r]} \leftarrow$ the added connections during the above loop for
       $C_{[i,j]}$ do
13:      if $C_{[q,r]}$ is redundant by judging from the calculated func-
         tional flexibility then
14:        Remove $C_{[q,r]}$.
15:      end if
16:    end for
17:  else
18:    Remove all the added connections during the above.
19:  end if
20: end for
21: for $N_m \leftarrow$ each LUT in the PPC do
22:   if the number of fanin of $N_m > k$ then
23:     Add a MUX by which all the added connections to $N_m$ are
        selected to one input to $N_m$.
24:   end if
25: end for
```

Figure 6 A pseudo code of the proposed procedure.

Even by the above loop, if we cannot make $C_{[i,j]}$ redundant, we should remove all the added connections in the above loop at line 18. Even if $C_{[i,j]}$ becomes redundant, there may be some added connections which do not contribute to make $C_{[i,j]}$ redundant. Such connections are removed at line 14.

Finally, if the number of fanin connections exceeds k at an LUT by adding connections, we add a MUX before the LUT to select a necessary input between the added (redundant) inputs. This is done in the loop between lines 21 to 25. Note that we add a redundant connection to one LUT at most once in the loop between lines 2 and 10 for processing $C_{[i,j]}$ in the

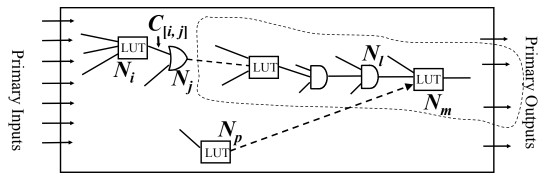


Figure 7 A Boolean network to explain the proposed procedure.

outer loop. However, we may add a connection to the same LUT when we are processing a different $C_{[i',j']}$ in the outer loop. Thus, we may need a MUX. We would like to remind readers of the example in Section 3, where we use a MUX. This means that we can use only one added input connection to each LUT when we try to fix a defect. Thus, when we propagate SPFDs (CSPFs) at line 6 (at line 8), respectively, we ignore the (previously) added input connections.

We would like to note that we can consider many other procedure to transform PPCs; the procedure in Figure 6 is just one of them.

5 Preliminary experiments

To evaluate how our method can increase the number of robust connections, we did the following preliminary experiments.

Step 1: We map some of MCNC benchmark circuits [12] into k-input LUT networks. To do so, we use the recommended SIS (a system for sequential circuit synthesis of UC Berkeley) [13] technology mapper commands.

Step 2: We generate initial PPCs such that the LUTs at the primary outputs are selected.

Step 3: For each LUT, we add $(6-k)$ connections so that some of the original inputs to the LUT become redundant.

Unlike our transformation method (described in Section 4) where we utilize MUXs, we just use 6-input LUTs (without MUXs) for the final implementation in order for the simplicity of the experiments. For example, in the case of $k=4$, we can add two connections to each LUT.

After the above circuit transformation, we count the original input connections to all the LUTs of the following two types: (a) connections that can become redundant by reconfiguring the functionality of LUTs, and (b) the other connections.

We report the number (a) and (b) for the three cases: $k=3,4,5$ in Table 1. In the table, "-" indicates we cannot complete the transformation because of the explosion of BDDs. From the table, we can observe that we indeed make some connections robust by (even) the above simple transformation.

6 Conclusions

We propose to use *Partially-Programmable Circuits* (PPCs) to increase the yield of LSIs with low overhead compared to previous approaches. We expect that our proposed

Table 1 Results: (a) robust, and (b) non-robust con.

Circuit	$k=3$		$k=4$		$k=5$	
	(a)	(b)	(a)	(b)	(a)	(b)
C1355	2	62	0	128	0	128
C1908	6	51	0	94	3	63
C2670	25	89	29	105	-	-
C3540	13	30	2	45	10	54
C432	10	9	12	12	10	21
C499	3	77	0	128	0	144
C880	5	69	-	-	-	-
alu2	5	11	5	13	4	16
apex6	21	257	24	343	22	403
apex7	18	80	8	110	11	126
b9	24	33	33	40	30	47

circuit transformation method increases the number of *robust* connections in PPCs, which apparently results in higher yield. Our preliminary experiments justify our expectation; we found that some connections indeed become redundant in some benchmark circuits.

Our future work is obviously to implement our circuit transformation method, and to evaluate its performance.

References

[1] G. Bourianoff: "The future of nanocomputing", IEEE Computer, **36**, 8, pp. 44–53 (2003).

[2] Semiconductor Industry Association: "International Technology Roadmap for Semiconductors 2007 Edition" (2007).

[3] V. K. R. Chiluvuri and I. Koren: "Layout-synthesis techniques for yield enhancement", IEEE Transactions on Semiconductor Manufacturing, **8**, 2, pp. 178–187 (1995).

[4] A. Nardi and A. L. Sangiovanni-Vincentelli: "Logic synthesis for manufacturability", IEEE Design & Test of Computers, **21**, 3, pp. 192–199 (2004).

[5] C. He and M. Jacome: "Defect-aware high-level synthesis targeted at reconfigurable nanofabrics", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, **26**, 5, pp. 817–833 (2007).

[6] A. Doumar and H. Ito: "Detecting, diagnosing, and tolerating faults in SRAM-based field programmable gate arrays: a survey", IEEE Trans. on Very Large Scale Integration (VLSI) Systems, **11**, 3, pp. 386–405 (2003).

[7] S. Yamashita, H. Sawada and A. Nagoya: "SPFD: A New Method to Express Functional Permissibilities", IEEE Trans. on CAD, **19**, 8, pp. 840–849 (2000).

[8] S. Muroga, Y. Kambayashi, H. C. Lai and J. N. Culliney: "The Transduction Method - Design of Logic Networks Based on Permissible Functions", IEEE Trans. Comput., **38**, 10, pp. 1404–1424 (1989).

[9] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. Sangiovanni-Vincentelli and A. R. Wang: "Multi-level Logic Minimization Using Implict Don't Cares", IEEE Trans. on CAD, **7**, 6, pp. 723–740 (1988).

[10] H. Savoj and R. K. Brayton: "The Use of Observability and External Don't Cares for Simplification of Multi-Level Networks", Proc. DAC, pp. 297–301 (1990).

[11] H. Savoj, R. K. Brayton and H. J. Touati: "Extracting Local Don't Cares for Network Optimization", Proc. ICCAD, pp. 514–517 (1991).

[12] S. Yang: "Logic synthesis and optimization benchmarks user guide version 3.0", MCNC (1991).

[13] E. M. S. et al.: "SIS: A System for Sequential Circuit Synthesis", Technical Report UCB/ERL M92/41, Univ. of California, Berkeley (1992).