A New Way of Detecting Reconvergent Fanout Branch Pairs in Logic Circuits*

Shiy Xu
Shanghai University
Shanghai, China 200072, syxu@fudan.ac.cn

E. Edirisuriya
University of Sri Jayewardenepura,
Nugegoda, Sri Lanka

ABSTRACT

Reconvergent fanout has been one of the critical issues for testing of VLSI circuits and design for testability. In this paper we will present a new algorithm, which will detect all sources of reconvergent fanout branch pairs by processing a normal circuit description. The algorithm identifies all the gates at which reconvergence occurs, the reconvergent sites, and lists all the reconvergent fanout branch pairs that are reconvergent at these sites. The automatic detection of such reconvergence can be used for improving the testability analysis or assist test generation of circuits containing such fanout branches.

Key words: Fanout, Fanout branch, Reconvergence, Testability, Testable Design

1. INTRODUCTION

Many authors have advocated the use of testability measures as aids in identifying regions of a circuit, which will be difficult to test [1-5]. In general, there is a correlation between the testability analysis and the test generation effort [6]. However, the information obtained from conventional testability analysis is often misleading in circuits, which contain reconvergent fanouts. Savir [7] gives some examples of the failure of testability measures to cope with reconvergent fanouts. The problem of reconvergence in VLSI circuits is acute. This statement is supported by Ratiu [8] who states that data collected from several existing integrated circuits shows that about half of the nodes in a typical VLSI chip are fanout nodes. Owing to the inherent pin limitation of VLSI circuits it is obvious that the majority of fanout branches must reconverge before reaching the primary outputs.

However, as noted by Ratiu [8] there is a large amount of reconvergence in a VLSI circuit and it would be clearly impractical for a circuit designer to supply this information. Having identified the importance of locating all the reconvergences in logic circuits, Boberts [9] presented an algorithm to find all the reconvergences present in a circuit. But this algorithm is sometimes insufficient and it does not provide any information regarding to the fanout branches. Therefore, in this paper we will present a new algorithm to find reconvergent fanout pairs, and its location of reconvergent in logic circuits.

2. BASIC CONCEPTS

The input to the proposed algorithm is a text file, known as the circuit description file, which specifies the connectivity details of the circuit nodes. This description specifies the full connectivity for every node in the circuit from primary inputs (PIs) to the primary output (POs). A node with one input (stem) and more than one output (branch) is said to be a **fanout point.** A reconvergent fanout is said to be present at a particular node if there is more than one path from any fanout node to that node. The output line that feeds such a node is said to be the "site of the reconvergence".

For each node in the circuit, a level is assigned. This level measures the distance of the node from the PIs. The level of each PI is defined to be 0. And the level of node i, $l(i)$, whose inputs come from node $k_1, k_2, \ldots\ldots k_p$, is given by

$$l(i) = 1 + \max l(k_j), \quad j = 1, 2, \ldots, p \qquad (1)$$

2.1 Fanout-branch lists

Associated with each node in the circuit is a fanout branch list (FOBL). The FOBL of a node contains a list of entries for every fanout branch that directly (or indirectly) feeds that node. Each entry in a FOBL, say the FOBL for node N, consists of two components:

(a) the unique identifier (the name of line) for the fanout branch;
(b) a path count (appeared in a parenthesis followed the name of line) which holds the number of paths between the fanout branch and the node N.

Now, let us consider the following circuit shown in Figure 1 to see how the FOBL of a node looks like:

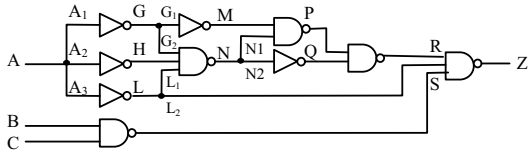

Figure 1. An Example of Expression for FOBL

A typical FOBL of a node, say node P in Fig 1, is represented by:

FOBL (P) = N_1 (1), G_1 (1), G_2 (1), A_1 (2), A_2 (1), A_3 (1),

where N_1, G_1, G_2, A_1, A_2, and A_3 are the fanout branch identifiers. The path count of each fanout branch is indicated in parenthesis to its identifier, e.g. there are 2 paths from fanout branch A_1 to node P.

The entries in a FOBL are held in a definite order. They are held in decreasing value of branch level and within the same level, branches with the same stem are kept adjacent to one another.

Reconvergent fanout branch lists

In addition to the FOBL, each node in the circuit has a reconvergent fanout branch list (RFOBL). The RFOBL of a node contains an entry for every pair of fanout branches that reconverges at that node. The pairs in a RFOBL are also held in a definite order. They are held in decreasing value of fanout branch level and within the same level, fanout branch pairs with the same stem are kept adjacent to one another. A typical RFOBL of a node, say node P in

*This work has been supported by National Natural Science Foundation of China (NSFC) under the grant number 60173029 and the No. 4 Key Project of Shanghai Education Commission

Figure 1, is represented by: RFOBL (P) = ($G_1(1)$, $G_2(1)$), ($A_1(2)$, $A_2(1)$), ($A_1(2)$, $A_3(1)$)

The number within parenthesis is the number of paths from the fanout branch to the node P. These RFOBLs are used to hold the results of the algorithm either for later display, or for the use by another algorithm, such as an improved testability analysis or to assist test generation (TG).

2.2 FOBL & RFOBL operations

Apart from the relative ordering of the entries in the FOBL and RFOBL, they are somewhat similar to the operations of mathematical sets. Several operators, which are analogous to set operators, are defined for the FOBLs and RFOBLs as follows.

- **Union operator**

The union of two FOBLs denoted by "∪", is defined as:

FOBL(C) = FOBL(A) ∪ FOBL(B)

FOBL(C) holds one entry for every entry present in FOBL(A) or in FOBL(B), if the FOBL(C) is the sum of its individual path counts in FOBL(A) and FOBL(B).

- **Intersection operator**

The intersection of two FOBLs, denoted by "∩" operator, is defined as : RFOBL(C) = FOBL(A) ∩ FOBL(B)

Let A_i and A_j ($A_i \neq A_j$) be two entries present in FOBL(A) and FOBL(B) respectively , if both A_i and A_j are fanout branches of the same stem, then RFOBL(C) has an entry of the form (A_i(x), A_j(y)), where x and y are path counts of A_i and A_j respectively . The calculation of path counts (x and y) will be explained latter.

- **Star Union operator**

The star union of RFOBL(A) and RFOBL(B), denoted by "∪*" operator, is defined as:

RFOBL(C) = RFOBL(A) ∪* RFOBL(B)

RFOBL(C) has an entry (A_i(x), A_j(y)), for every entry present in either RFOBL(A) or RFOBL(B). No attention is paid on the path counts in this operation.

- **Difference operator**

The difference of RFOBL (A) and FOBL (B), denoted by "–" operator, is defined as: RFOBL(C) = RFOBL (A) – FOBL (B)

If (A_i(x), A_j(y)) is an entry present in RFOBL(A), and A_j (p) and/or A_j (q) are present in FOBL(B), then the values of x and/or y will be reduced from p and/or q, respectively. As an example, if both A_j (p) and A_j (q) are present in FOBL(B), then RFOBL(C) has an entry (A_i (x-p), A_j (y-q)), and if either A_i or A_j is not in FOBL(B), then the entry in RFOBL(C) is (A_i (x), A_j (y)).

In practice, a more useful version of the difference operator is as follows: RFOBL(C) = RFOBL (A) – (k · FOBL(B)), where k is an arbitrary positive integer which is used to multiply the path count of every entry in FOBL(B). If (A_i(x), A_j(y)) is an entry present in RFOBL (A), and A_j (z) is an entry in FOBL(B), then the path count of A_i in RFOBL (C) will be: path count (A_i / in RFOBL(A)) – k · path count(A_j / in FOBL(B)). i.e., x - k · z.

In this paper we consider only single output devices, however, the procedure can be extended to multiple output devices as well.

2.3 Formation of the FOBL

The FOBL of the output line of a device is formed from the FOBL of the input lines to that device. The FOBL of a line should contain a list of all the fanout branches that feed that line. Hence the FOBL of an output line is simply obtained by forming the union of all the input lines to the device, which feed that output line. There is one additional step in forming the complete FOBL of an output line. This step is required whenever the line is a fanout branch. If this is the case an entry is placed at the head of the newly formed FOBL for that line, this entry holds the identifier for the fanout branch and path count of 1.

2.4 Formation of the RFOBL

For a 2-input device to be the site of a reconvergent of two fanout branches, it is necessary that two branches must have a path to that device via the two inputs to that device. If this is the case, one fanout branch will be present in the FOBL for one input line and other fanout branch will be present in the FOBL, for the other input line. The intersection operator can be used to find all such reconvergences. After locating all the reconvergent pairs (A_i(x), A_j(y)) using intersection operator, the path count x and y are set to the corresponding values of A_i and A_j in the FOBL of the output line. This is repeated for every pair in the RFOBL.

The calculation of the FOBL and RFOBL of the node P in the circuit in Figure 1 are as follows. The FOBLs of the input lines to the node P are:

FOBL (M) = $G_1(1)$, $A_1(1)$

FOBL (N) = $N_1(1)$, $G_2(1)$, $L_1(1)$, $A_1(1)$, $A_2(1)$, $A_3(1)$

To calculate the FOBL (P), we use the union operator, and the union of FOBL(M) and FOBL (N1) will be given as: FOBL (P) = $N_1(1)$, $G_1(1)$, $G_2(1)$, $L_1(1)$, $A_1(2)$, $A_2(1)$, $A_3(1)$

To calculate the RFOBL (P), we have entry $G_1(1)$ in FOBL(M) and entry $G_2(1)$ in FOBL(N_1). Also G_1 and G_2 are fanout branches of the same stem. Therefore, we add an entry ($G_1(1)$,$G_2(1)$) to the RFOBL (P). The path count of 1 in literals of added pair is the corresponding values in the FOBL (P). The complete RFOBL (P) is given by:

RFOBL (P) = ($G_1(1)$,$G_2(1)$), ($A_1(2)$, $A_2(1)$), ($A_1(2)$, A_3 (1))

For an *n*-input device the above seems to suggest that a total of $n(n-1)/2$ intersections would have to be formed. However, the amount of intersections that need to be formed is considerably reduced if the FOBL for the output line is formed at the same time as the RFOBL is formed. If this approach is taken, an n-input gate requires only (n-1) intersection to be formed.

3. ALGORITHMIC PROCEDURES

3.1 Procedure to calculate the RFOBL and FOBL

Let $I_1,I_2,\ldots,I_n$ denote the *n* input lines which are used to generate output Z of a device (function). RFOBL (Z) is the list of fanout branch pairs, which reconverge at the output Z. The procedure used to calculate the FOBL and RFOBL is as follows:

```
Procedure build_fobls(Z);
begin
        RFOBL (Z) = Ø;
        FOBL (Z) =FOBL(I1);
        for x = 2 to n do
        begin
            p= FOBL(Z) ∩ FOBL(Ix);
```

RFOBL(Z) = FOBL(Z) ∪* p;
FOBL(Z)=FOBL(Z) ∪ FOBL(I_x)
end
for every pair in RFOBL(Z)
set the path count of each literal to corresponding values in FOBL(Z);
end;

3.2 Initialization and processing steps

For each node in the circuit, the reach_no counter which is held in a particular data structure is set to the number of inputs lines to that node. This reach_no counter is used to determine when a node can have its FOBL and RFOBL determined via the above-mentioned procedure. The FOBL and RFOBL of a node cannot be determined until the FOBLs of all the input lines that feed that node have been determined. Whenever an input line to a device has its FOBL determined, the reach_no counter for each node fed by that device is decremented. When the reach_no count for a particular node becomes zero, it is known that the node has been 'reached' by all the input lines that feed it, and its FOBL and RFOBL can now be determined.

Two lists are maintained in the algorithm, i.e., the current list (CL) and the next list (NL). The CL holds all the circuit nodes, which are ready to have their FOBL determined, i.e., those nodes that have been reached by all the input lines feeding them. The NL holds all the circuit nodes, which will be processed on the next stage. At the start of the algorithm the CL is initialized to hold all the PIs, while the NL is cleared. The following is a summary of the basic algorithm.

Algorithm I for detecting reconvergent fanout branch pairs with the location of reconvergence is as follows:

Step 1: Read in the circuit description file
Step2: For each node N in the circuit:
reach_no count(N) = number of input lines to the device feeding N
Step 3: NL = list of PIs nodes
Step 4: CL = NL
Step 5: NL = Ø
Step 6: For each node N_1 in the CL:
For each node N_2 fed by node N_1:
reach_no count (N_2) = reached no count(N_2) - 1
If reach_no count (N_2) = 0 then
NL = NL+N_2 (place node on next list)
Step 7: For every N in the CL:
build_fobls (N)
reduce_rfobl (N) (see section 3.3)
Step 8: For each node N in the CL which is a fanout branch:
- generate a unique identifier for the fanout branch
- place identifier (with a path count of 1) at the beginning of the FOBL of node N
Step 9: If NL≠Ø go to Step 4
Step 10: Produce desired output from the RFOBLs
Exit

Although the above algorithm will find all the reconvergent pairs in a circuit, it is also required to identify all the fanout branches that actually reconverge at those sites. This task is harder than the basic reconvergence detection algorithm outlined above. For example, The calculation of the RFOBL for the gate R in Figure 1 should give the pairs (N_1,N_2),(G_1,G_2), (A_1,A_2) and (A_1,A_3) which are reconvergent at gate R. The FOBLs for the input lines to the gate R are: FOBL (P) = $N_1(1)$, $G_1(1)$, $G_2(1)$, $L_1(1)$, $A_1(2)$, $A_2(1)$, $A_3(1)$;

FOBL (Q) = $N_2(1)$, $G_2(1)$, $G_1(1)$, $A_1(1)$, $A_2(1)$, $A_3(1)$

Hence applying the algorithm RFOBL(R), it will give the results as follows: RFOBL(R) = ($N_1(1)$,$N_2(1)$), ($G_1(1)$,$G_2(2)$), ($A_1(3)$,$A_2(2)$), ($A_1(3)$,$A_3(2)$), ($A_2(2)$, $A_3(2)$)

From this result, it seems to indicate that (N_1, N_2), (G_1,G_2) (A_1, A_2) (A_1, A_3), (A_2, A_3) are all reconvergent at the gate R. This is, however, to be incorrect. It is because that, in fact, the pair (A_2, A_3) does not actually reconverge at the gate R. This is one of the advantages we have.

As a matter of fact, in the circuit of Figure 1, there are two paths from fanout branch A_2 to R. But they are not distinct from the two paths from fanout branch A_3 to R. *A path between two nodes is said to be distinct from a path between two other nodes, if there are no nodes in common* (*in between*) *to both path.* To conclude that two fanout branches (obtained by the above algorithm) reconverge at a particular site, it is necessary to establish that there is at least one path from either of these fanout branches that is distinct from the paths from the other fanout branch. In the circuit of Figure 1, for example, the two paths $A_1 \rightarrow G \rightarrow G_1 \rightarrow M \rightarrow P \rightarrow R$ and $A_2 \rightarrow H \rightarrow N \rightarrow N_2 \rightarrow Q \rightarrow R$ are distinct. Therefore, the pair (A_1, A_2) reconverges at the gate R. The two paths $A_2 \rightarrow H \rightarrow N \rightarrow R$ and $A_3 \rightarrow L \rightarrow L_1 \rightarrow R$ are not distinct. As a result, the pair (A_2, A_3) will not be reconverging at the gate R.

The path counts held in the FOBLs are used to accommodate the various circuit topologies. The RFOBL of a particular node is produced in the same manner as outlined previously. Next the RFOBL is reduced so that it will contain only those fanout branch pairs, which actually reconverge at that site. The procedure used to reduce the RFOBL can be summarized as follows.

3.3 RFOBL reduction procedure

The **Algorithm II** of RFOBL reduction process is summarized by the following procedure in which X represents the node that is the site of the reconverge.

Procedure reduce_rfobl(x)
begin
Step 1: New_RFOBL(x) = Ø
Step2: The fanout branch pairs of the same stem are removed from the head of RFOBL(X), and add them to the new_RFOBL(X);
Step 3: If RFOBL(X) = Ø go to step 11;
Step 4: k = sum of the path counts of distinct fanout branches of all pairs added in step 2;
Step 5: Y= stem of the fanout branches added in step 2;
Step 6: Save the RFOBL(X) defined in step 2 for later use;
Step 7: RFOBL(X)= RFOBL(X) – k·FOBL(Y);
Step 8: From the RFOBL(X) found in step 7, identify those pairs whose path counts have zeros in both literal;
Step 9: Restore RFOBL(X) saved in step(f), after removing those pairs identified in step 8;

Step 10: If RFOBL(X) $\neq$ Ø go to step 2; else go to the next step

Step 11: End of procedure

end;

This procedure is applied to the RFOBL produced in step 7 of the algorithm I present in section 3.2.

For further illustration, an example of executing the algorithm will be shown as follows:

Let us apply the procedure of algorithm II of RFOBL reduction process to the node R in Figure 1. As we have seen that the unreduced list of node R is given as

RFOBL(R) = (N_1(1), N_2(1)), (G_1(1), G_2(1)), (A_1(3), A_2(2)), (A_1(3), A_3(2)), (A_2(2), A_3(2))

For saving the space, we omitted the detailed procedure. In summary, the application of above algorithm to the circuit in Figure 1 will produce the results shown in Figure 2.

Reconvergence at gate	Caused by fanout branch pairs
N	(A_1, A_2), (A_1, A_3), (A_2, A_3)
P	(G_1, G_2), (A_1, A_2), (A_1, A_3)
R	(N_1, N_2), (G_1, G_2), (A_1, A_2), (A_1, A_3)
Z	(L_1, L_2), (A_1, A_2), (A_2, A_3)

Figure 2. Located fanout branch pairs from Algorithm II

Note that, if the reconvergent site is not necessary, and the location of the reconvergent fanout branch pairs is only required, then the formation of RFOBL and new_RFOBL is also not necessary. Since the fanout branches feeding the same PO are reconvergent from the FOBL of PO nodes, we can find the reconvergent fanout branch pairs. Thus, the same algorithm can also be used with the elimination of few steps. From the FOBL of PO nodes we can produce the desired output.

4. TIME COMPLEXITY OF THE ALGORITHM

Suppose the circuit is composed of n devices and each of which has two inputs and one output. As each node in the circuit requires one, and only one output line to feed it, there will be a total of ($n + p_i$) nodes in the circuit. The quantity p_i represents the number of PIs to the circuit. In general, the number of PIs will be insignificant and trivial as compared to n, thus the quantity p_i can be ignored. The exact number of fanout nodes in a VLSI circuit is unknown. However, it has been reported [8] that about half of the nodes in a circuit are fanout nodes.

Assuming that the fanout nodes are 'evenly' distributed in the circuit, a FOBL will be fed by an average of n/4 fanout nodes. In general, it can also be assumed that the fanout nodes are distributed evenly among the logic feeding each of the POs making the average number of fanout nodes feeding a FOBL $n/(4 \cdot p_o)$ (where p_o is the number of POs)

For the purpose of explanation, let us assume that each fanout node has k fanout branches. Let m be the average number of fanout nodes feeding a FOBL. To form the FOBL and RFOBL for a 2-input device requires one union and one intersection operation. To form the union of two FOBLs of average length $m \cdot k$ requires at most $2(m \cdot k)$ comparison operations. To form the intersection of two FOBLs of average length $m \cdot k$ requires at most $m \cdot k^2$ comparison operations. The number of comparisons required in difference operator is $m \cdot k^3$. Therefore, to produce the FOBL, RFOBL and new_RFOBL will require at most $(2m \cdot k + m \cdot k^2 + m \cdot k^3)$ comparison operations. Hence, if the average number of fanout nodes feeding a FOBL is $n/(4 \cdot p_o)$, for n nodes in the circuit it requires $(k/2 + k^2/4 + k^3/4) \cdot (n^2/po)$ comparison operations, i.e., the time complexity of the algorithm is $O(n^2 \cdot k^3)$

In digital circuits, the number of fanout branches in a stem varies from one stem to another. Let p be the average number of fanout branches and q the maximum number of the fanout branches in the circuit. Thus in the average case, the time complexity of the algorithm is less than $O(n^2 \cdot p^3)$ and will be $O(n^2 \cdot q^3)$ in the worst case. According to the study we carried out, for ten combinational benchmark circuits, p is in the range of (3-5) and q is in the range of (10-16). Thus, for VLSI circuits, in the average case, the time complexity of the algorithm is less than $O(n^3)$ and in the worst case it is less than $O(n^4)$.

The algorithm has been implemented in C on a SUN-SPARC 630 workstation running on Unix 6.0. The algorithm comprises about 800 source code lines. Execution time of the algorithm for SN74181 arithmetic logic unit (ALU) consisting of 63 gates was 0.7 seconds. For a circuit comprising 4000 gates, the execution time was approximately 5 seconds.

5. CONCLUSION

Reconvergence and fanout issues have been considered as one of the hardest nuts to be dealt with in the areas of testing and testable design. This paper, therefore, proposed an efficient way (algorithm) as a powerful tool for analyzing the scenario of reconvergent fanout branch pairs. The algorithms presented here are dedicatedly designed for identifying all sources of reconvergent fanout branch pairs and sites of reconvergence in a circuit. This is achieved by processing the normal circuit description (net list). Knowledge of the reconvergent structure of a circuit can be used to produce a more accurate testability measure or assist in the test generation process. The complexity of the algorithm has been discussed in order to make it more efficient.

6. REFERENCES

[1] J.E. Stephenson and J. Grason, "A Testability Measure of Register Transfer Level Digital Circuits", Proceedings of IEEE / FTCS'76, Pittsburgh, PA, USA, June 1976, pp.101-107

[2] J. Grason, "TMEAS, A testability Measurement Program", IEEE/ Proceedings of 16th Design Automation Conference, San Diego, CA, USA, June 1979, pp.156-161

[3] L.H. Goldstein "Controllability/observability analysis of Digital Circuits", IEEE Trans. 1979, CAS-26, pp.685-693

[4] P.G. Kovijanic "Computer Aided Testability Analysis". IEEE/ Proceedings of Intl. Automatic Test Conference (Autotestcon'79), Minneapolis. MN, USA, September 1979, pp.292-294

[5] R.G. Bennetts, C.M. Maunder and G..D. Robison,. "CAMELOT: A Computer Aided Measure for Logic Testability", IEE /Proceedings, E, Comput. & Digital Tech., 1981, 128,(5), pp.177-189

[6] V.D. Agrawal and M. Mercer, "Testability Measures What Do They Tell Us?", Proceedings of 1982 IEEE/ Proceedings of Intl. Test conference, pp.391- 396

[7] J. Savir, "Good Controllability and Observability Do Not Guarantee Good Testability ",IEEE Trans on Computers, 1983, C-32, pp.1198-1200

[8] I. M. Ratiu, A. Sangiovanni-Vincentelli and D.O. Pederson, "VICTOR:A Fast VLSI Testability Analysis Program", IEEE/ Proceedings of Intl. Test Conference,1982, pp.391-396

[9] M.W. Robert and P. K. Lala, "Algorithm to Detect Reconvergent Fanout in Logic Circuits", IEEE/ Proceedings, Vol.134, Pt.E.No.2, March 1987

[10] H. Fujiwara, "Computational Complexity of Controllability/Observability Problems for Combinational Circuits", IEEE Trans. on Computers, Vol.39.No.6, 1990, pp.762-767

[11] Shiyi Xu and *E.A.T.A.* Edirisuriya, "Comparison Study of Cost Function", Proceedings of CTW'94, (Chinese Test Workshop, English Version) July, 1994, pp. 42-46