MULTIPLE-OUTPUT SHARED TRANSISTOR LOGIC (MOSTL) FAMILY SYNTHESIZED USING BINARY DECISION DIAGRAM

by

Takayasu Sakurai, Bill Lin, and A. Richard Newton

Memorandum No. UCB/ERL M90/21

16 March 1990

MULTIPLE-OUTPUT SHARED TRANSISTOR LOGIC (MOSTL) FAMILY SYNTHESIZED USING BINARY DECISION DIAGRAM

by

Takayasu Sakurai, Bill Lin, and A. Richard Newton

Memorandum No. UCB/ERL M90/21

16 March 1990

TITLE PAGE

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

MULTIPLE-OUTPUT SHARED TRANSISTOR LOGIC (MOSTL) FAMILY SYNTHESIZED USING BINARY DECISION DIAGRAM

by

Takayasu Sakurai, Bill Lin, and A. Richard Newton

Memorandum No. UCB/ERL M90/21

16 March 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Multiple-Output Shared Transistor Logic (MOSTL) Family Synthesized Using Binary Decision Diagram

Takayasu Sakurai*, Bill Lin and A. Richard Newton

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA94720, U.S.A.
***) On leave from Semiconductor Device Engineering Lab.,**
Toshiba Corporation, Kawasaki, 210, Japan

Abstract

A new type of logic family, Multiple-Output Shared Transistor Logic (MOSTL) family, is defined and a synthesis method for generating MOSTL is described. The MOSTL implements a logic function not by combining logic gates such as NAND's and OR's, but by combining transistors directly as switches. Since the MOSTL has more freedom in realizing a logic function, it offers a smaller and faster circuit than the standard cell based approach. More concretely speaking, in the MOSTL, transistors are shared among several logic functions and thus the number of MOSFET's are reduced and this in turn may reduce delay time. It is best suited for the Sea-Of-Gates designs and a full manual design where designers are permitted to build a circuit at a transistor level.

A synthesis method presented is based on Binary Decision Diagram (BDD) and usually gives a good solution. The method is demonstrated to generate a sneak-path free circuit and in this sense never fails to produce a solution, which is an important feature when applied to real designs.

A MOSTL together with the synthesis method will provide a systematic way to generate a 'clever' circuit, which could only have been built by the ingenuity of experienced circuit designers otherwise.

1. Introduction

Transistor level logic network synthesis has been attracting attentions for a long time[1-5, 14-16] since the early and important work of Shannon[10,11]. Horn et al. [14, 16] in the middle of 50's proposed a symbolic matrix technique to tackle the problem and it is useful in the analysis of a logic switching network but as for the synthesis it was based on the intuition and gave a limited success.

There are two major advantages in using the transistor-level synthesis. One is the use of 'pass variables' or 'pass transistors', a good example of which is a steering logic family introduced in [17]. The other is a sharing of transistors among different switching paths. The former advantage is pursued by recent researches [5, 15] and a big advance has been observed in this area but the latter advantage is not studied well. Wu et al. [3, 4] investigated the sharing problem and a limited success has been reported if the problem is confined to a single-contact, single-output network where one variable can drive only one control gate of a transistor and the number of outputs is one. Even for the general single-output case, the synthesis method is still based on intuition.

In this report, one practically important logic family, namely Multiple-Output Shared Transistor Logic (MOSTL) family, is defined and a systematic way of synthesizing it is described. The MOSTL is a single-stage logic gate and more general than the single-output logic gate. It utilizes both of the pass variables and the transistor sharing and includes usual CMOS complex gates, a steering logic and a barrel shifter.

A synthesis method presented is based on Binary Decision Diagram (BDD)[6,7] and usually gives a good solution. The method is demonstrated to generate a sneak-path free circuit and in this sense never fails to produce a solution, which is an important feature when applied to real designs.

In Section 2, MOSTL is defined and examples are given. A synthesis method based on BDD is described in Section 3, followed by a sneak-path free nature of the the generated circuits is discussed in Section 4. Section 5 and 6 are dedicated for discussions and possible area of future works and conclusions, respectively. In Appendix, a sample program is shown for the BDD-based minimizer.

2. Multiple-Output Sharing Transistor Logic (MOSTL)

The schematic diagram of the MOSTL is shown in Fig.1 and an example is given in Fig.2. In Fig.1, the NMOS and PMOS blocks include a transistor circuit where any number of output terminals are connected to a power line or to pass variable inputs according to the control variables. From the left side of the boxes, control variables are input and from the bottom or the top of the boxes, pass variables are incurred. The NMOS/PMOS block can include non-serial-parallel structure and non-planar structure. Three variations are shown in the figure but several other configurations are also possible.

The salient feature of MOSTL is the exclusion of mixing PMOS's and NMOS's in one circuit block and that the only one power source attached to the NMOS logic part is VSS and the only power line connected to the PMOS logic is VDD. By limiting the structure like this, it is possible to eliminate a multi-stage nature and V_{TH} problems from the synthesis. V_{TH} , threshold voltage of

MOSFET, hinders the output to swing full VDD-VSS range and this degrades circuit margins if it is not treated properly. The multi-stage nature makes the problem intractable. The MOSTL includes most of the practical logic circuits such as CMOS complex gates, a barrel shifter, and a steering logic family.

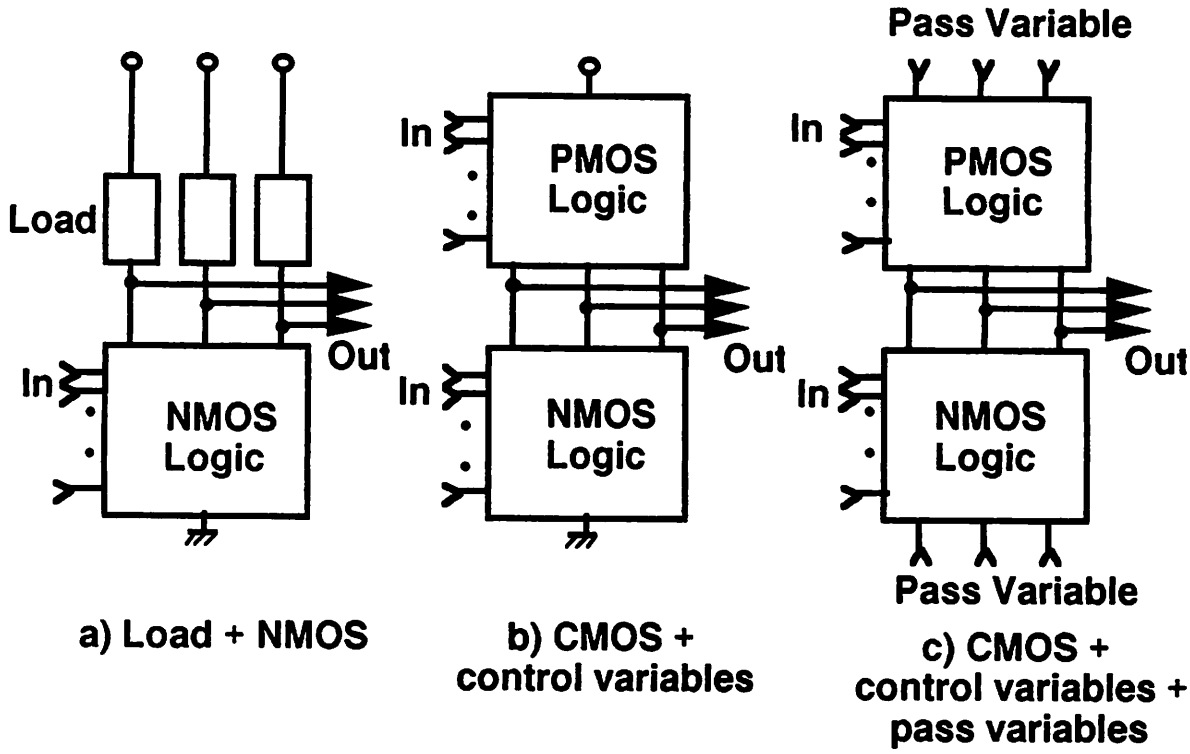

Fig.1 Schematic diagram of MOSTL

The example in Fig.2 is for a parity generator circuit for three input. The functional descriptions are:

$$f = abc' + ab'c + a'bc + a'b'c'$$

$$f' = abc + ab'c' + a'bc' + a'b'c\,.$$

In this expression, prime (') denotes an inverted input. This type of logic function is difficult to minimize by a standard cell approach. Direct implementation of the logic function by a parallel-serial CMOS transistor network needs 48 transistors, while the MOSTL needs 20 or 16 transistors depending on the use of pass variables. If the number of inputs is increased, the advantage becomes more eminent.

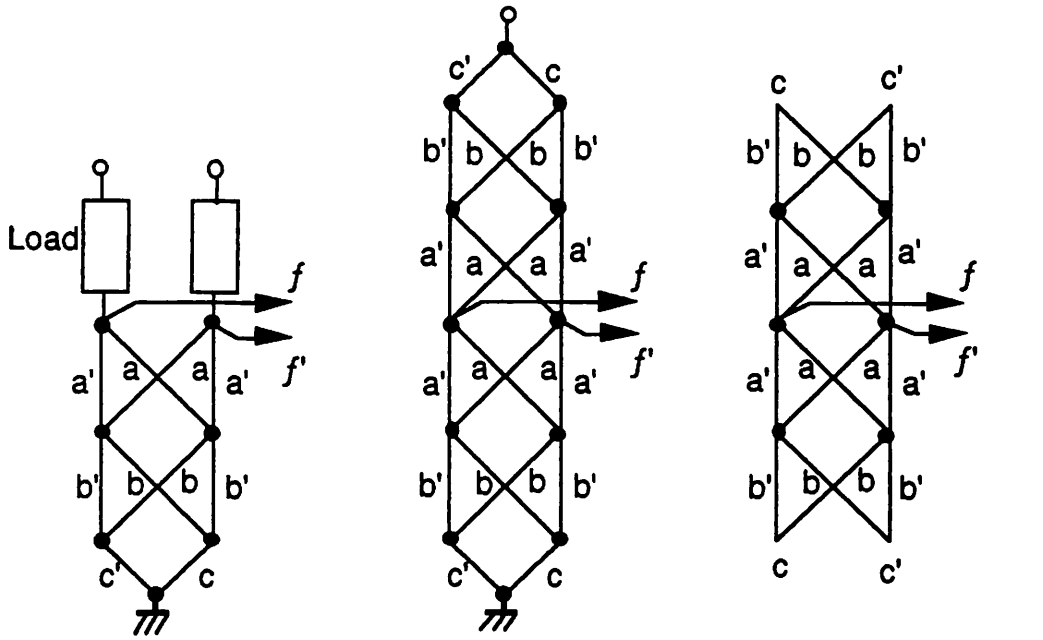

Fig.2 An example of MOSTL (a parity generator circuit)

3. Synthesis of the MOSTL Using Binary Decision Diagram (BDD)

In this section, a synthesis method is described. The synthesis begins by building a BDD. To build the BDD, there are several methods. One method [6] is: first generate logic binary trees for

separate logic functions as shown in Fig.3 and then merge these trees by merging common subtrees from the bottom. In Fig.3, the left-most © in the left tree means that the function goes to '0' when c=1 and goes to '1' when c=0. Consequently, the subgraph which is rooted at the © and the subgraph which is rooted at the second left © in the middle tree is considered to be the same. So the pointer to the second left © can be switched to the left-most © in the left tree. Applying this procedure iteratively, the reduced BDD of the right graph can be obtained. The detailed description of the procedure is found in [6].

The BDD has an important feature that if the input ordering is given, the reduce BDD is unique so that it can be used as a standard form of logic function. The number of edges included in the BDD depends on the ordering and the optimum ordering is difficult to find without an exhaustive search. However, for less than 5~6 inputs, the exhaustive search is possible and since the MOSTL is a single-stage gate, the number of input is small.

Once the BDD is constructed, it is easy to interpret the graph as a transistor circuit. The edges directed to the terminal [1] basically correspond PMOS block MOSFETs and the edges directed to the terminal [0] correspond to NMOS block MOSFETs. When constructing a PMOS block, a=1(0) edge should be converted to a PMOSFET whose gate is controlled by a' (a). For a NMOS block, a=1(0) edge should be converted to a NMOSFET whose gate is controlled by a (a'). A literal whose two children are [1] and [0] may be replaced by a pass variable input.

Further reduction in the number of transistors is possible when checks are made for all edges if the edges can omitted or shorted. The example of this further reduction is explained next using a more complicated example.

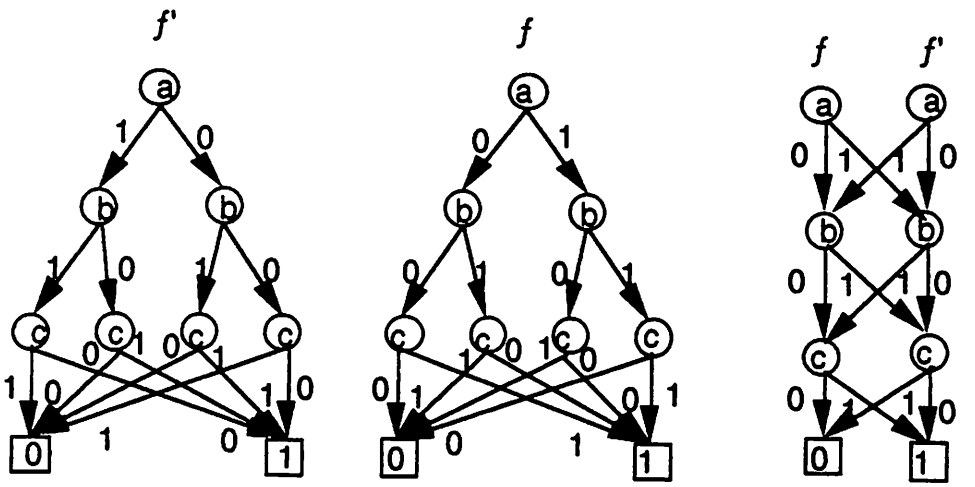

Fig.3 Binary Decision Diagram (BDD) for a parity generator

Figure 4 and TABLE I show a Karnaugh map and a truth table of the more complicated example, respectively. There are three output terminals and the functional description is:

$$f1 = AB'C + A'D' + A'B'C$$

$$f2 = AB'D' + A'B$$

$$f3 = AC + A'BC' + AB'C'D' .$$

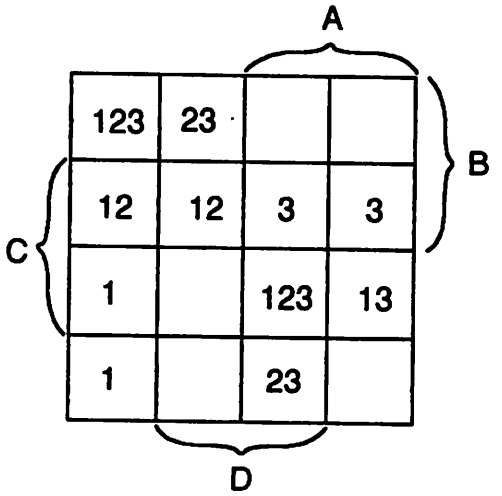

Fig.4 Karnaugh map of 'relay3' example

TABLE I Truth table of 'relay3' example

A	B	C	D	*f*1	*f*2	*f*3
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	0	1	1	0	0	0
0	1	0	0	1	1	1
0	1	0	1	0	1	1
0	1	1	0	1	1	0
0	1	1	1	1	1	0
1	0	0	0	0	0	0
1	0	0	1	0	1	1
1	0	1	0	1	0	1
1	0	1	1	1	1	1
1	1	0	0	0	0	0
1	1	0	1	0	0	0
1	1	1	0	0	0	1
1	1	1	1	0	0	1

After constructing a BDD, the NMOS and PMOS blocks are extracted separately. Then each edge in the graph is tested if it can be omitted or shorted. If it can be omitted or shorted, the transistor can be eliminated. In this example of 'relay3'[1] in Fig.5, five edges are shortable. The shorting process may create a sneak-path (see the next section), so that a careful validity checking of the shorting should be done. One way of doing this is through a simulation, which is adopted in the program listed in Appendix.

For this example, the number of transistors needed is 27 as shown in Fig.5, but a parallel-serial implementation of the logic leads to 42 transistors.

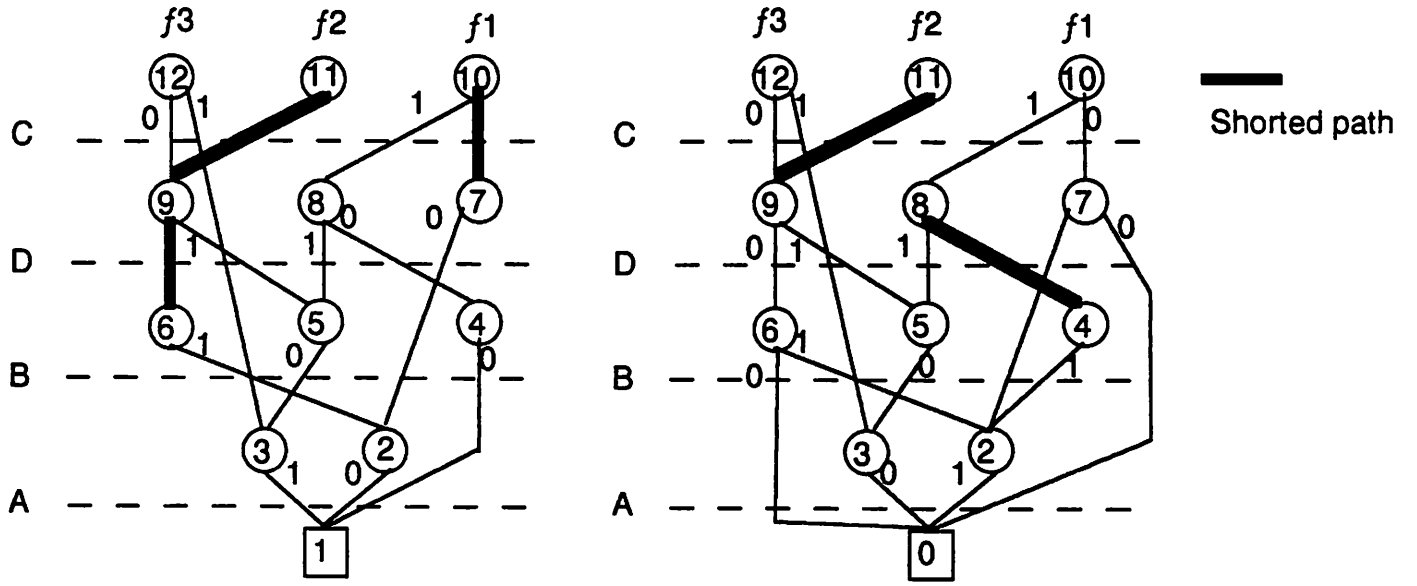

Fig.5 Synthesized MOSTL for 'relay3' example

4. Edge-Merging and Sneak-Path

In the synthesis of MOSTL or more general transistor switching network, a sneak-path is a difficult problem. An example of the sneak-path is shown in Fig.6. Suppose two functions f = a and g = a + b are to be realized. First, the edge controlled by a is connected to f and the edge controlled by b is connected to g realizing that f = a and g = b. Then to make g be a + b, vertices i and j can be connected. Then g becomes correct but f becomes incorrect because there exists a path from [1] to f through b. This is a sneak-path. Usually sneak-paths are not obvious and a critical checking should be employed to reveal the sneak-paths.

The essence of the sneak-path is the existence of contradiction on the assignment of logic values on one vertex. In the example, when a = 0 and b = 1, g expects vertex j to be 1 while *f* expects vertex j to be 0, which is a contradiction.

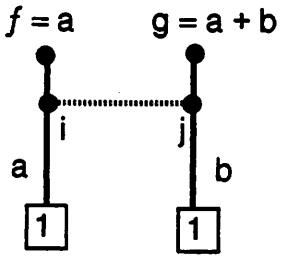

Fig.6 Sneak-Path

A very powerful transformation in constructing MOSTL is 'edge-merging' as shown in Fig.7. The essence of the edge-merging is to merge two edges with one node common controlled by the same variable into one edge. Other than the BDD based method described above, this edge-merging seems promising. The drawback of the edge-merging, however, is the creation of sneak-path.

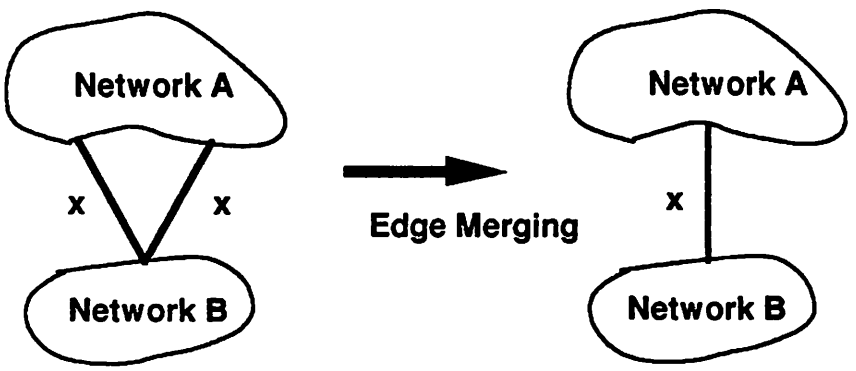

Fig.7 Edge-Merging technique.

It can be demonstrated that the BDD based method generates a network which does not contain sneak-paths. First, separate logic binary trees do not have sneak-paths because only one path is activated at a time which connects an output to the power source. The reduce operation in the BDD reduction scheme does not create any sneak-paths. This latter part is explained in more detail. The reduce operation includes only two kinds of procedures as shown in Fig.8.

One procedure is an elimination procedure and the other is a subgraph sharing procedure. The elimination procedure does not introduce a sneak-path because the only thing this procedure does is to assign one physical vertex instead of two logically shorted vertices. If there exists a sneak-path after the procedure, it must be existed before the procedure.

The subgraph sharing procedure does not introduce any sneak-paths either because whenever vertex j expects 0(1) on vertex n, i also expects 0(1) on the vertex n. So there is no contradiction and thus no sneak-paths.

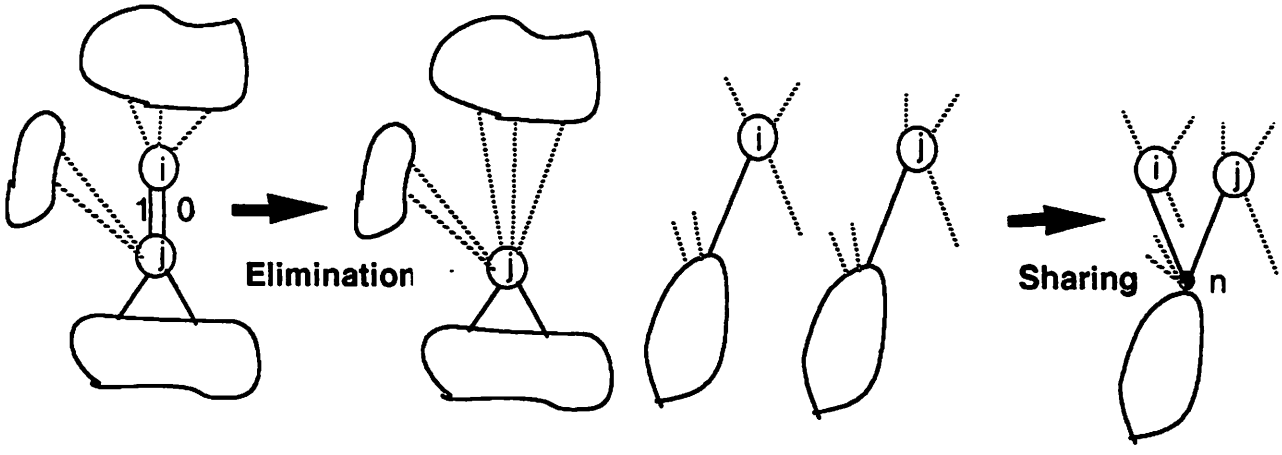

Fig.8 Two basic procedures to reduce BDD. The sneak-path free nature is demonstrated by using this figure.

5. Discussions and Future Work

The synthesis method presented here based on a BDD usually gives a good-quality solution to a MOSTL generation problem as in the example of relay3 circuit. Sometimes it gives the optimum transistor network as in the example of parity generator circuits. However, the method does not always guarantee the optimality so that sometimes the method generates a bad circuit. In this sense, some procedure is preferable to be taken to improve the generated transistor network. Simulated Diffusion or Simulated Annealing can be a choice.

Other than the synthesis method itself, a research as a VLSI synthesis system is of interest. The total system may look like Fig.9. We can make use of a standard logic minimizer[8,9] and the several outputs of logic functions generated by the logic minimizer which share common inputs are bundled together and input to a MOSTL synthesizer. The transistor sharing and the pass variables are treated properly in the MOSTL synthesizer.

The partitioner in Fig.9 and MOSTL generator should be working cooperatively or iteratively so as to optimize the area and speed. A research should also to be carried in this area. That is, multi-stage MOSTL optimization is the important next step. As is mentioned in previous section, the inclusion of don't care condition is another area to look into, although simple inclusion is easy.

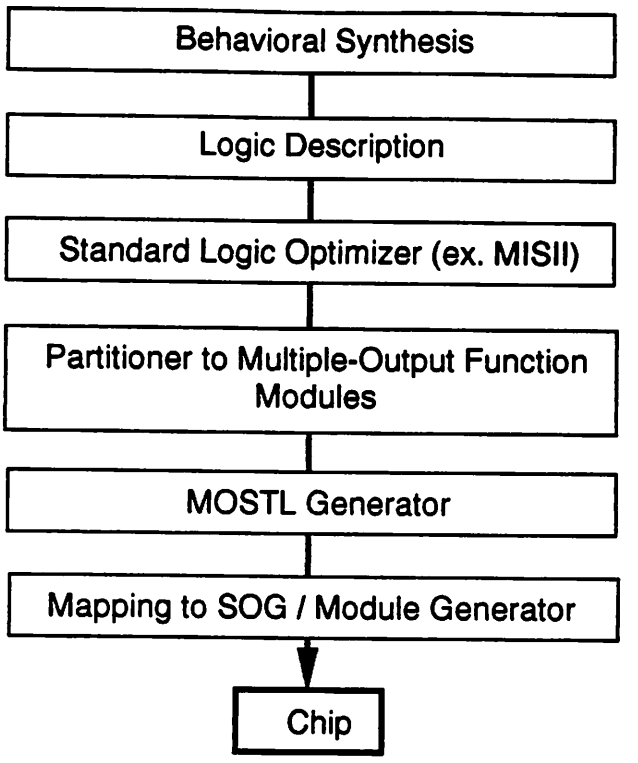

Fig.9 MOSTL generator incorporated in a design system

6. Conclusions

A new family of logic circuit is introduced and a synthesis method is presented based on a BDD. Although this method does not guarantee to give the optimum circuit and some extensions are desirable, it usually gives a good solution. The method is demonstrated to generate a sneak-path free circuit and in this sense never fails to produce a solution, which is an important feature when applied to real designs.

A MOSTL together with the synthesis method will provide a systematic way to generate a 'clever' circuit, which could only have been built by the ingenuity of experienced circuit designers otherwise. Sea-Of-Gates and a fully manual design can be benefitted by the proposed method.

Acknowledgments

The encouragement of Prof. B.Brayton, Prof. A.Sangiovanni-Vincentelli, Y.Unno, Y.Takeishi, H.Yamada and T.Iizuka throughout the course of this work is appreciated. This work was supported by a grant from Toshiba Corporation.

References

[1] H.J.Beuscher, A.H.Budlong, M.B.Haverty, Electronic Switching Theory and Circuits, Section 10, Van Norstrand Reinhold Berkeshire, England.

[2] N.Deo, Graph Theory with Applications to Engineering and Computer Science, Prentice Hall.

[3] M-Y.Wu, I.N.Hajj, "Switching Network Logic Approach to Sequential MOS Circuit Design," IEEE Trans. on CAD, CAD-8, No.7, pp.782-794, Jul.1989.

[4] M-Y.Wu, W.Shu, S-P.Chan, "A Unified Theory for MOS Circuit Design - Switching Network Logic," Int. J. Electronics, Vol.58, No.1, pp.1-33, 1985.

[5] C.Pedron, A.Stauffer, "Analysis and Synthesis of Combinational Pass Transistor Circuits," IEEE Trans. on CAD, CAD-7, No.7, pp.775-785, Jul.1988.

[6] R.E.Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Trans. on Computers, C-35, No.8, pp.677-691, Aug.1986.

[7] S.B.Akers, "Binary Decision Diagrams," IEEE Trans. on Computers, C-27, No.6, pp.509-516, Jun.1978.

[8] R.Rudell, A.L.Sangiovanni-Vincentelli, "ESPRESSO-MV: Algorithm for Multiple Valued Boolean Minimization," CICC'85, May 1985.

[9] R.Brayton et al, "Multiple-level Logic Optimization System," ICCAD'86, pp.356-359, Nov.1986.

[10] C.E.Shannon, "A Symbolic Analysis of Relay and Switching Circuits," AIEE Transactions, vol.57, pp.713-723, 1938.

[11] C.E.Shannon, "The Synthesis of Two-Terminal Switching Circuits," Bell System Tech. J., 28, pp.59-98, 1949.

[12] E.J.McCluskey, "Minimization of Boolean Functions," Bell System Tech. J., pp.1417-1445, Nov.1956.

[13] E.J.McCluskey, "Detection of Group Invariance or Total Symmetry of a Boolean Function," Bell System Tech. J., pp.1445-1453, Nov.1956.

[14] F.E.Horn, L.R.Schissler, "Boolean Matrices and the Design of Combinational Relay Switching Circuits," Bell System Tech. J., pp.177-202, Jan.1955.

[15] D.Radhakrishnan, S.R.Whitaker, G.K.Maki, "Formal Design Procedures for Pass Transistor Switching Circuits," IEEE J. Solid-State Circ., SC-20, No.2, Apr.1985.

[16] D.Lewin, Design of Logic Systems, Section 5, Van Norstrand Reinhold, Berkeshire, England, 1985.

[17] C.Mead, L.Conway, Introduction to VLSI Systems, Addison-Wesley, Massachusetts, 1980.

Appendix A Program Listing

Program source codes are shown in the following pages. The programs are written in QuickBasic Ver.1.0 for Macintosh SE/30. The following is an example of the input to the program.

Input example of BDD synthesizing program

Two numbers in the first line are number of input and output.
The following lines include function description.
Actually they are the output bit pattern of the function corresponding the input

```
0,0,0,0
0,0,0,1
0,0,1,0
0,0,1,1
   •
   •
   •
```

```
---------------------
4,3
1,0,0
0,0,0
1,0,0
0,0,0
1,1,1
0,1,1
1,1,0
1,1,0
0,0,0
0,1,1
1,0,1
1,1,1
0,0,0
0,0,0
0,0,1
0,0,1
```

```
OPTION BASE 0
DEFINT a-w
E = 10000: nleaf = 1: nfunc = 1: nXD = 1: nV = 1
DIM SHARED ibitm(nleaf,nfunc), obitm(nleaf,nfunc)
DIM SHARED XD2YD(nXD), VPat(nV), permV(nV), minPermV(nV)
DIM SHARED XL(nXD), XH(nXD), YL(nXD), YH(nXD), YV(nXD)
DIM SHARED pobitm(nleaf, nfunc), V2XD1(nV), V2XDn(nV), V2YD1(nV),
V2YDn(nV)
DIM SHARED nYFan(nXD), YFan(nXD, INT(nXD / 2)), YFanV(nXD,
INT(nXD / 2)), visited(nXD)
DIM SHARED queue(500)
DIM SHARED YHbuf(nXD), YLbuf(nXD), YVbuf(nXD)
 '--- V : input variable   L : low   H : high
 '--- Pat : Pattern
 '--- X : original data
 '--- Y : reduced data

Windower:
 '---- initialize window 1 ----
WINDOW 2,"Graphics Window", (200,20)-(480, 300),1
WINDOW 1,"Text Window", (0,20)-(200, 300),1
TEXTSIZE 8
TEXTFONT 3
OPEN "scrn:" FOR OUTPUT AS #1

 '---- menu ----
MENU 1,0,1,"File"
MENU 2,0,1,"Edit"
MENU 3,0,1,"BDD"
MENU 4,0,1,"Params"

MENU 1,1,1,"Load"
MENU 1,2,1,"Show Loaded Data"
MENU 1,3,1,"Output Select"
MENU 1,4,0,"Print"
MENU 1,5,0,"-"
MENU 1,6,1,"Quit": cmdkey 1,6,"Q"

MENU 2,1,0,"Copy":cmdkey 2,1,"C"

MENU 3,1,1,"Create & Reduce"
MENU 3,2,1,"Exhaustive Search"
MENU 3,3,1,"Minimize"
MENU 3,4,1,"Show BDD"

MENU 4,1,0,"Params Set"

ON MENU GOSUB Menucheck: MENU ON
Idle:
   GOTO Idle

Menucheck:
   menunumber = MENU(0)
   menuitem = MENU(1)
   MENU
   ON menunumber GOSUB Filer, ClipBoarder, Bdder, Setter
RETURN

Filer:
   ON menuitem GOSUB Loader, Shower, Outer, Quitter, Quitter ,Quitter
RETURN

Quitter:
   CLOSE
   SetCreate "bdd.out", "MSWD"
   WINDOW CLOSE 1
   WINDOW CLOSE 2
END

ClipBoarder:
   ON menuitem GOSUB ClipCopier
RETURN

Bdder:
   showFlag = 111
   ON menuitem GOSUB BDDCreateReducer, BDDExhaustiver,
BDDMinimizer, BDDShower
RETURN

Setter:
   ON menuitem GOSUB Quitter
RETURN

ClipCopier:
   OPEN "CLIP:PICTURE" FOR OUTPUT AS #3
   PRINT#3, image$
   CLOSE #3
RETURN

Eraser:
   WINDOW 2
   WINDOW 1
   image$ = ""
   CLS
RETURN

Loader:
   '---- load data ----
   infile$ = FILESS(1,"TEXT")
   IF (infile$ = "") THEN RETURN
   OPEN infile$ FOR INPUT AS #2
   '---- input # of input & # of output ----
   INPUT #2, nV, nfunc
   nleaf = 2 ^ nV
   ERASE ibitm, obitm
   DIM SHARED ibitm(nleaf,nV), obitm(nleaf, nfunc)
   '---- input output bit pattern ----
   FOR ileaf = 1 TO nleaf
      LINE INPUT #2, inline$
      FOR ifunc = 1 TO nfunc
         obitm(ileaf, ifunc) = VAL(MID$(inline$, 2*(ifunc-1)+1, 1))
      NEXT
   NEXT
   '---- initialize ibitm ----
   FOR ileaf = 1 TO nleaf
      remainder = ileaf - 1
      FOR iV = nV TO 1 STEP -1
         thisbit = remainder MOD 2
         ibitm(ileaf, iV) = thisbit
         remainder = (remainder - thisbit) / 2
      NEXT
   NEXT
   CLOSE #2
RETURN

Shower:
   '---- show loaded data ----
   WINDOW 1
   PRINT #1, "nVar=";nV, "nfunc=";nfunc
   FOR ileaf = 1 TO nleaf
      FOR iV = 1 TO nV
         PRINT #1, ibitm(ileaf, iV);" ";
      NEXT
      PRINT #1,"  ";
      FOR ifunc = 1 TO nfunc
         PRINT #1, obitm(ileaf, ifunc);" ";
      NEXT
      PRINT #1,""
   NEXT
RETURN

Outer:
   '---- output device select ----
   WINDOW 1
   PRINT "Output to screen(0)"
   INPUT "or new file(1) or append to the file(2)"; outdev
   CLOSE #1
   outfile$ = "bdd.out"
   SELECT CASE outdev
   CASE 0
      OPEN "scrn:" FOR OUTPUT AS #1
   CASE 1
      'outfile$ = FILESS(0)
      'IF (outfile$ = "") THEN RETURN
      OPEN outfile$ FOR OUTPUT AS #1
   CASE 2
      'outfile$ = FILESS(1,"TEXT")
      'IF (outfile$ = "") THEN RETURN
      OPEN outfile$ FOR APPEND AS #1
      PRINT #1,"": PRINT #1,""
   CASE ELSE
      OPEN "scrn:" FOR OUTPUT AS #1
   END SELECT
RETURN

BDDInitializer:
```

```
    '---- Initialize BDD ----
    nXD = (2 ^ nV - 1) * nfunc + 1
    '---- define arrarys ----
    ERASE XD2YD, VPat, permV, minPermV, XL, XH, YL, YH, YV
    ERASE pobitm, V2XD1, V2XDn, V2YD1, V2YDn
    ERASE nYFan, YFan, YFanV, visited
    ERASE YHbuf, YLbuf, YVbuf
    DIM SHARED XD2YD(nXD), VPat(nV), permV(nV), minPermV(nV)
    DIM SHARED XL(nXD), XH(nXD), YL(nXD), YH(nXD), YV(nXD)
    DIM SHARED pobitm(nleaf, nfunc), V2XD1(nV), V2XDn(nV),
V2YD1(nV+1), V2YDn(nV+1)
    DIM SHARED nYFan(nXD), YFan(nXD, INT(nXD / 2)), YFanV(nXD,
INT(nXD / 2)), visited(nXD)
    DIM SHARED YHbuf(nXD), YLbuf(nXD), YVbuf(nXD)
    XL(0) = 0: XH(0) = 0: XL(1) = 1: XH(1) = 1
    '---- initialization of V2XD1, V2XDn, I, h ----
    V2XD1(1) = 2: V2XDn(1) = 2 ^ (nV-1) * nfunc + 1
    FOR iV = 2 TO nV
      V2XD1(iV) = V2XDn(iV-1) + 1
      V2XDn(iV) = V2XD1(iV) -1 + 2 ^ (nV - iV) * nfunc
      XL(V2XD1(iV)) = V2XD1(iV-1)
      XH(V2XD1(iV)) = V2XD1(iV-1) + 1
    NEXT
RETURN

BDDCreateReducer:
    '---- BDD create and reduce according to the input perm ----
    GOSUB BDDInitializer
    WINDOW 1
    PRINT "Enter permutation pattern."
    PRINT "There should be";nV;" numbers separated by blank."
    LINE INPUT inlineS
    FOR iV = 1 TO nV
      permV(iV) = VAL(MID$(inlineS, 2*(iV-1)+1, 1))
      'PRINT "permV(";iV;")=";permV(iV)
    NEXT
    GOSUB BDDTreeCreator
    GOSUB BDDReducer
    GOSUB BDDChecker
    GOSUB BDDMinimizer
    GOSUB BDDCoster
RETURN

BDDExhaustiver:
    GOSUB BDDInitializer
    '---- permutation matrix initialize ----
    FOR iV = 1 TO nV
      permV(iV) = iV
    NEXT
    endPermFlag = 0
    startPermFlag = 1
    '---- permutaion loop ----
    minBDDtotalCost = E
    WHILE (endPermFlag = 0)
      GOSUB GeneratePerm
      GOSUB BDDTreeCreator
      GOSUB BDDReducer
      'GOSUB BDDChecker
      GOSUB BDDCoster
      IF (totalCost < minBDDtotalCost) THEN
        minBDDtotalCost = totalCost
        FOR iV = 1 TO nV
          minPermV(iV) = permV(iV)
        NEXT
      END IF
    WEND
    '---- re-generate minimum BDD ----
    FOR iV = 1 TO nV
      permV(iV) = minPermV(iV)
    NEXT
    GOSUB BDDTreeCreator
    GOSUB BDDReducer
    GOSUB BDDChecker
    GOSUB BDDCoster
RETURN

BDDTreeCreator:
    '---- scramble function data according to permutation ----
    FOR ileaf = 1 TO nleaf
      FOR iV = 1 TO nV
        VPat(iV) = ibitm(ileaf, permV(iV))
      NEXT
      target = 1
      FOR iV = 1 TO nV
        target = target + VPat(iV) * 2 ^ (nV - iV)
      NEXT
      FOR ifunc = 1 TO nfunc
        pobitm(target, ifunc) = obitm(ileaf, ifunc)
      NEXT
    NEXT
    '---- create BDD leaf ----
    FOR ileaf = 1 TO nleaf STEP 2
      FOR ifunc = 1 TO nfunc
        iXD = (ifunc-1) * (nleaf/2) + (ileaf-1)/2 + 2
        XL(iXD) = pobitm(ileaf, ifunc)
        XH(iXD) = pobitm(ileaf+1, ifunc)
      NEXT
    NEXT
    '---- initialize XL & XH ----
    FOR iV = 2 TO nV
      FOR iXD = V2XD1(iV)+1 TO V2XDn(iV)
        XL(iXD) = XL(iXD-1) + 2
        XH(iXD) = XL(iXD) + 1
      NEXT
    NEXT
    FOR iYD = 0 TO nXD
      XD2YD(iYD) = iYD
      YL(iYD) = E: YH(iYD) = E
    NEXT
    YL(0) = 0: YH(0) = 0: YL(1) = 1: YH(1) = 1
RETURN

BDDReducer:
    '---- reduce BDD ----
    V2YD1(1) = 2: V2YDn(1) = 2: specialVertex = 0
    FOR iV = 1 TO nV
      FOR iXD = V2XD1(iV) TO V2XDn(iV)
        '---- if high child = low child, eliminate the vertex ----
        IF (XD2YD(XL(iXD)) = XD2YD(XH(iXD))) THEN
          XD2YD(iXD) = XD2YD(XL(iXD))
          IF (iV = nV) THEN
            YL(V2YDn(iV)) = XD2YD(XL(iXD))
            YH(V2YDn(iV)) = XD2YD(XH(iXD))
            YV(V2YDn(iV)) = iV
            'XD2YD(iXD) = iYD
            V2YDn(iV) = V2YDn(iV) + 1
            specialVertex = specialVertex + 1
          END IF
        ELSE
          iYD = V2YD1(iV)
          BDDReduceLoop:
          '---- check for the same subtree which already exists ----
          IF (XD2YD(XL(iXD)) = YL(iYD)) AND (XD2YD(XH(iXD)) = YH(iYD))
THEN
            XD2YD(iXD) = iYD
            GOTO BreakBDDReduceLoop
          ELSE
            IF (iYD >= V2YDn(iV)) THEN
              YL(V2YDn(iV)) = XD2YD(XL(iXD))
              YH(V2YDn(iV)) = XD2YD(XH(iXD))
              YV(V2YDn(iV)) = iV
              XD2YD(iXD) = iYD
              V2YDn(iV) = V2YDn(iV) + 1
              GOTO BreakBDDReduceLoop
            END IF
          END IF
          iYD = iYD + 1
          GOTO BDDReduceLoop
          BreakBDDReduceLoop:
        END IF
      NEXT
      '---- set first and last iYD for the next level ----
      V2YD1(iV+1) = V2YDn(iV)
      V2YDn(iV+1) = V2YDn(iV)
    NEXT
    nYD = V2YDn(nV) - 1
RETURN

BDDCoster:
    '---- cost calculation and output ----
    zeroedge = 0: oneedge = 0
    FOR YD = 2 TO nYD
      IF YL(YD) = 0 OR YH(YD) = 0 THEN zeroedge = zeroedge + 1
      IF YL(YD) = 1 OR YH(YD) = 1 THEN oneedge = oneedge + 1
    NEXT
    ncost = 2 * (nYD - 1 - specialVertex) - oneedge
```

```
    pcost = 2 * (nYD - 1 - specialVertex) - zeroedge
    totalCost = ncost + pcost
    PRINT #1, "perm= ";
    FOR iV = 1 TO nV
      PRINT #1, permV(iV);
    NEXT
    PRINT #1,""
    PRINT #1, USING "nMOS=###  pMOS=###  T=###"; ncost, pcost,
totalCost
RETURN

BDDChecker:
    '---- checking the validity ----
    FOR ifunc = 1 TO nfunc
      '---- scan every output function ----
      FOR ileaf = 1 TO nleaf
        iYD = nYD - (nfunc - ifunc)
        '---- scan every leaves ----
        FOR iV = 1 TO nV
          'VPat(iV) = ibitm(ileaf, permV(iV))
          VPat(iV) = ibitm(ileaf, iV)
          IF (YV(iYD) = nV - iV + 1) THEN
            IF (VPat(iV) = 0) THEN
              iYD = YL(iYD)
            ELSE
              iYD = YH(iYD)
            END IF
          END IF
        NEXT
        IF (iYD <> pobitm(ileaf, ifunc)) THEN
          '--- check failed ---
          PRINT #1, "Check failed at ifunc, ileaf", ifunc, ileaf, iYD, pobitm(ileaf,
 ifunc), obitm(ileaf, ifunc)
          PRINT #1, "info on perm= ";
          FOR iV = 1 TO nV
            PRINT #1, permV(iV);
          NEXT
          PRINT #1,""
          PRINT #1, "info on VPat= ";
          FOR iV = 1 TO nV
            PRINT #1, VPat(iV);
          NEXT
          PRINT #1,""
        END IF
      NEXT
    NEXT
RETURN

GeneratePerm:
    '---- generate permutation one by one in lexicographic order ----
    IF (startPermFlag = 1) THEN
      startPermFlag = 0
      RETURN
    END IF
    '---- find the largest i so that p(i) < p(i+1) ----
    i = nV - 1
    WHILE (permV(i) > permV(i+1))
      i = i - 1
      IF (i = 0) THEN
        endPermFlag = 1
        GOTO PermLoopEnd
      END IF
    WEND
    '---- find the smallest pj so that i < j and pi < pj ----
    pi = permV(i)
    pj = nV + 1
    FOR jin = i+1 TO nV
      IF (pi < permV(jin)) AND (permV(jin) < pj) THEN
        j = jin
        pj = permV(jin)
      END IF
    NEXT
    '---- swap p(i) < p(j) ----
    SWAP permV(i), permV(j)
    '---- reverse the order following pj ----
    itemp = nV
    iSwapEnd = INT((nV - (i+1) + 2) / 2)
    FOR iV = i+1 TO i+iSwapEnd
      SWAP permV(iV), permV(itemp)
      itemp = itemp - 1
    NEXT
    PermLoopEnd:
RETURN

BDDShower:
    sf1 = INT (showFlag /100): showFlag = showFlag - 100 * sf1
    sf2 = INT (showFlag / 10): showFlag = showFlag - 10 * sf2
    sf3 = INT (showFlag / 1)
    IF (sf1 = 1) THEN
      '---- BDD info display ----
      'print #1, "# input= ";nV,"# output= ";nfunc
      PRINT #1, "perm= ";
      FOR iV = 1 TO nV
        PRINT #1, permV(iV);
      NEXT
      PRINT #1,""
      PRINT #1, USING "nMOS=###  pMOS=###  T=###"; ncost, pcost,
totalCost
      PRINT #1, CHR$(13)+"oneFlag=";oneFlag
      FOR iYD = 0 TO nYD
        PRINT #1, USING "ID=##  L=#####  H=#####  V=##"; iYD, YL(iYD),
YH(iYD), YV(iYD)
        BDDShowerLoop:
          IF (MOUSE(0) <> 0) GOTO BDDShowerLoop
      NEXT
    END IF
    '---- display minimized switching network ----
    IF (sf2 = 1) THEN
      oneFlag = 0: GOSUB MatShower
    END IF
    IF (sf3 = 1) THEN
      oneFlag = 1: GOSUB MatShower
    END IF
RETURN

MatShower:
    PRINT #1, CHR$(13)+"oneFlag=";oneFlag
    FOR iYD = 0 TO nYD
      PRINT #1, USING "ID=##  L=#####  H=#####  V=##"; iYD, Y01L(oneFlag,
 iYD), Y01H(oneFlag, iYD), Y01V(oneFlag, iYD)
      MatShowerLoop:
        IF (MOUSE(0) <> 0) GOTO MatShowerLoop
    NEXT
RETURN

BDDMinimizer:
    short = 1000: termOpen = 2000
    '---- consider one tree and zero tree separately ----
    FOR oneFlag = 0 TO 1
      FOR iYD = 0 TO nYD
        '---- store YH, YL, YV into buffer ----
        YHbuf(iYD) = YH(iYD): YLbuf(iYD) = YL(iYD): YVbuf(iYD) = YV(iYD)
        '---- open special edges ----
        IF (oneFlag = 0) THEN
          IF (YH(iYD) = 1) THEN YH(iYD) = termOpen
          IF (YL(iYD) = 1) THEN YL(iYD) = termOpen
        ELSE
          IF (YH(iYD) = 0) THEN YH(iYD) = termOpen
          IF (YL(iYD) = 0) THEN YL(iYD) = termOpen
        END IF
      NEXT
      '---- making shorts ----
      FOR iYD = 2 TO nYD
        '---- for low edge ----
        oldY = YL(iYD)
        '---- skip terminal open edge ----
        IF (oldY <> termOpen) THEN
          YL(iYD) = YL(iYD) + short
          GOSUB ShortOpenOK
          IF (retSOOK = 0) THEN YL(iYD) = oldY
        END IF
        '---- for high edge ----
        oldY = YH(iYD)
        '---- skip terminal open edge ----
        IF (oldY <> termOpen) THEN
          YH(iYD) = YH(iYD) + short
          GOSUB ShortOpenOK
          IF (retSOOK = 0) THEN YH(iYD) = oldY
        END IF
      NEXT
      '---- making opens ----
      FOR iYD = 2 TO nYD
        '---- for low edge ----
        oldY = YL(iYD)
        '---- skip terminal open edge ----
        IF (oldY <> termOpen) THEN
```

```
        YL(iYD) = iYD
        GOSUB ShortOpenOK
        '--- if not openable, resume the edge ---
        IF (retSOOK = 0) THEN YL(iYD) = oldY
      END IF
      '--- for high edge ---
      oldY = YH(iYD)
      '--- skip terminal open edge ---
      IF (oldY <> termOpen) THEN
        YH(iYD) = iYD
        GOSUB ShortOpenOK
        '--- if not openable, resume the edge ---
        IF (retSOOK = 0) THEN YH(iYD) = oldY
      END IF
    NEXT
    '--- store separate BDD ---
    FOR iYD = 0 TO nYD
      Y01H(oneFlag, iYD) = YH(iYD)
      Y01L(oneFlag, iYD) = YL(iYD)
      Y01V(oneFlag, iYD) = YV(iYD)
      '--- restore YH, YL, YV ---
      YH(iYD) = YHbuf(iYD): YL(iYD) = YLbuf(iYD): YV(iYD) = YVbuf(iYD)
    NEXT
  NEXT
RETURN

MakeFanMatrix:
  '--- YFan example ---
  '            v (stored in nYFan)
  '--- 0 (YD):  3 (# of fans)  4  15  3 (fans) ---
  '--- 1 (YD):  2 (# of fans)  4   7   (fans) ---
  '--- 2 (YD):  1 (# of fans)  6       (fans) ---
  '--- YFanV example ---
  '--- 0 (YD):                2  -1  0
  '--- The YFanVs equals to 0 is shorted path ---
  '--- The YFanVs < 0 is low child path ---
  '--- initialize nYFan ---
  showFlag = 100: GOSUB BDDShower
  FOR iYD = 0 TO nYD
    nYFan(iYD) = 0
  NEXT
  FOR iYD = 0 TO nYD
    FOR iHL = 0 TO 1
      '--- for high and low edge ---
      IF (iHL = 0) THEN jYD = YL(iYD) ELSE jYD = YH(iYD)
      '--- skip open path ---
      IF (jYD <> iYD) AND (jYD <> termOpen) THEN
        '--- add low or high child to YFan ---
        PRINT #1, "nYFan, iYD, jYD, iHL, child";nYFan(iYD);iYD;jYD;iHL
        nYFan(iYD) = nYFan(iYD) + 1
        YFan(iYD, nYFan(iYD)) = jYD
        IF (iHL = 1) THEN YFanV(iYD, nYFan(iYD)) = YV(iYD) ELSE
YFanV(iYD, nYFan(iYD)) = -YV(iYD)
        '--- consider short path ---
        IF (jYD < short) THEN
        PRINT #1, "nYFan, iYD, jYD, iHL, non-s";nYFan(iYD);iYD;jYD;iHL
          nYFan(jYD) = nYFan(jYD) + 1
          YFan(jYD, nYFan(jYD)) = iYD
          IF (iHL = 1) THEN YFanV(jYD, nYFan(jYD)) = YV(iYD) ELSE
YFanV(jYD, nYFan(jYD)) = -YV(iYD)
        ELSE
          jYD = jYD - short
        PRINT #1, "nYFan, iYD, jYD, iHL, short";nYFan(iYD);iYD;jYD;iHL
          nYFan(jYD) = nYFan(jYD) + 1
          YFan(jYD, nYFan(jYD)) = iYD
          YFanV(jYD, nYFan(jYD)) = 0
        END IF
      END IF
    NEXT
  NEXT
  GOSUB FanShower:
RETURN

FanShower:
  '--- show fanout matrix ---
  FOR iYD = 0 TO nYD
    PRINT #1, "YFan (YFanV): ";
    FOR jYD = 1 TO nYFan(jYD)
      PRINT #1, USING "## (##)  ";YFan(iYD, jYD);YFanV(iYD,jYD);
    NEXT
    PRINT #1, ""
  NEXT
RETURN

ShortOpenOK:
  '--- check the validity of shorts and opens ---
  '--- initialize fanout matrix and visited array ---
  retSOOK = 1
  GOSUB MakeFanMatrix
  FOR iYD = 0 TO nYD
    visited(iYD) = 0
  NEXT
  '--- scan every leaves ---
  FOR ileaf = 1 TO nleaf
    '--- start from root ---
    IF (oneFlag = 0) THEN iYD = 0 ELSE iYD = 1
    nq = 0: iq = 0
    queue(iq) = kYD
    shortDetected = 0
    '--- initialize value matrix & VPat ---
    FOR iYD = 2 TO nYD
      valYD(iYD) = E
    NEXT
    FOR iV = 1 TO nV
      VPat(iV) = ibitm(ileaf, iV)
    NEXT
    SOOKLoop:
      '--- take one from the queue if possible, otherwise break ---
      IF (iq > nq) THEN GOTO BreakSOOKLoop
      kYD = queue(iq)
      iq = iq + 1
      '--- see if the node is visited ---
      IF (visited(kYD) = 1) THEN GOTO SOOKLoop
      visited(kYD) = 1
      '--- load fanouts in the queue which is connected to this node ---
      FOR jYD = 1 TO nYFan(kYD)
        '--- if shorted or conducted, add to the queue ---
        iV = YFanV(kYD, jYD)
        IF ((iV < 0) AND (VPat(ABS(iV)) = 0)) OR ((iV > 0) AND
(VPat(ABS(iV)) = 1)) THEN conducted = 1 ELSE conducted = 0
        IF (iV = 0) OR (conducted = 1) THEN
          nq = nq + 1
          queue(nq) = YFanV(kYD, jYD)
        END IF
      NEXT
      BreakSOOKLoop:
      '--- by now, visited nodes are connected to source ---
      '--- check if all one(zero) output are visited from source 1(0)
      '--- and all zero(one) output are not visited from source 0(1)
      FOR ifunc = 1 TO nfunc
        iYD = nYD - (nfunc - ifunc)
        shortFlag = 0
        openFlag = 0
        IF (visited(iYD) = 1) AND (oneFlag <> pobitm(ileaf, ifunc)) THEN
shortFlag = 1
        IF (visited(iYD) = 0) AND (oneFlag = pobitm(ileaf, ifunc)) THEN
openFlag = 1
        '--- report short or open ---
        IF (shortFlag = 1) OR (openFlag = 1) THEN
          retSOOK = 0
          '--- check failed ---
          IF (shortFlag = 1) THEN
            PRINT #1, "Short at ifunc, ileaf"; ifunc; ileaf
          END IF
          IF (openFlag = 1) THEN
            PRINT #1, "Open at ifunc, ileaf"; ifunc; ileaf
          END IF
          PRINT #1, "pobitm,  obitm="; pobitm(ileaf, ifunc); obitm(ileaf, ifunc)
          PRINT #1, "info on perm= ";
          FOR iV = 1 TO nV
            PRINT #1, permV(iV);
          NEXT
          PRINT #1,""
          PRINT #1, "info on VPat= ";
          FOR iV = 1 TO nV
            PRINT #1, VPat(iV);
          NEXT
          PRINT #1,""
        END IF
      NEXT
    NEXT
  RETURN
```