A LOGIC HAZARD COVER ALGORITHM

Richard S. Sandige and Bogdan M. Wilamowski*

Abstract

This paper presents three techniques for obtaining logic hazard covers. The first technique illustrates the standard Karnaugh map technique, which is useful for perhaps up to six variables and then becomes very time consuming to draw as well as difficult to use. The second technique shows a different approach using a hand calculation process for the new logic hazard cover algorithm presented in this paper. The advantage of the logic hazard cover algorithm over the Karnaugh map technique is that any number of input variables can be used, thus allowing Boolean specifications of just a few or of many input variables. The third technique illustrates the use of a software implementation of the logic hazard cover algorithm. The software tool removes the tedium associated with the Karnaugh map and hand calculation methods by automating the design process, yielding a fast, efficient, and convenient technique for identifying required logic hazard covers for the combinational logic circuits that are primarily used in asynchronous sequential designs.

Key Words

Algorithm, logic hazard, glitch, asynchronous, software tool, logic hazard covers

1. Introduction

Logic hazards can lead to spurious output signals, called glitches [1, 2], in combinational logic circuits. These types of glitches can be eliminated by product terms called logic hazard covers (LHCs). When these product terms are added to a minimized SOP (sum of products) Boolean function, the function becomes a logic-hazard-free function with a glitch-free SOP form of circuit implementation. Logic-hazard-free circuits are an essential requirement for the design of asynchronous (level mode) sequential circuits.

A logic hazardous function, that is, a function devoid of logic hazard covers [3], results in a glitchy SOP form of circuit implementation. A logic glitch occurs when a single input variable changes and the output is expected to remain constant but does not due to a difference in the delay times of two or more signal paths leading to the output.

The realization of circuits using logic-hazard-free functions results in the removal of spurious outputs or glitches. To demonstrate this, a software schematic capture and timing simulation tool, B^2 Logic [4], is used to show the implementation and timing simulation of logic hazardous functions (those without logic hazard covers) and hazard-free functions (those with logic hazard covers). The following techniques are used to obtain logic hazard covers for several Boolean function specifications: the Karnaugh map approach, a hand calculation approach using the logic hazard cover algorithm, and a software tool approach that automates the logic hazard cover algorithm.

The software tool allows up to 50 logical variables for a minimized SOP Boolean function. By reapplying the software tool with all logic hazard covers included in the function, one can verify the removal of all logic hazards. This tool provides designers with an automated approach to generating logic-hazard-free functions for large as well as smaller logic circuits.

2. Generation of Logic Hazard Covers via a Karnaugh Map

To illustrate the process of obtaining logic hazard covers, consider the following Boolean specification represented by equation (1).

$$F1(A,B,C) = \Sigma m(1,5,6,7) \qquad (1)$$

The three-variable Karnaugh map in fig. 1 illustrates this plotted specification. Equation (2) shows a minimized SOP equation for the ones of the function $F1$.

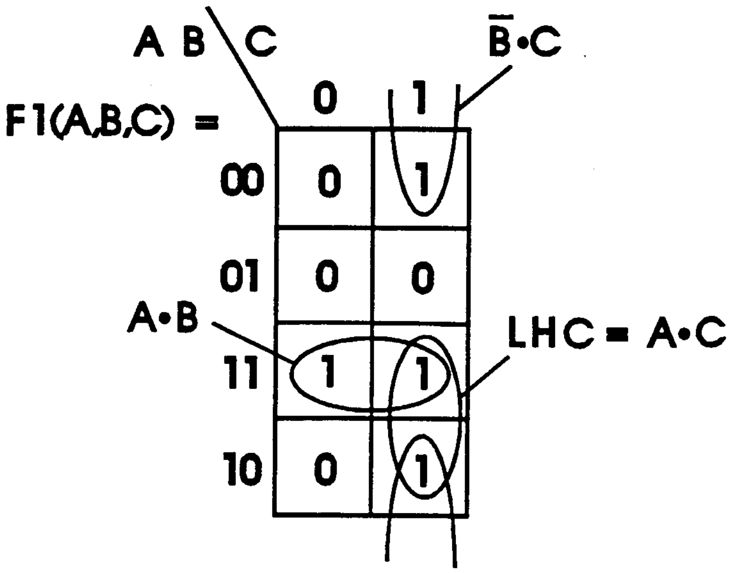


Figure 1. Karnaugh map for Boolean specification represented by (1).

* Department of Electrical Engineering, University of Wyoming, Laramie, WY 82071 USA E-mail: sandige@uwyo.edu

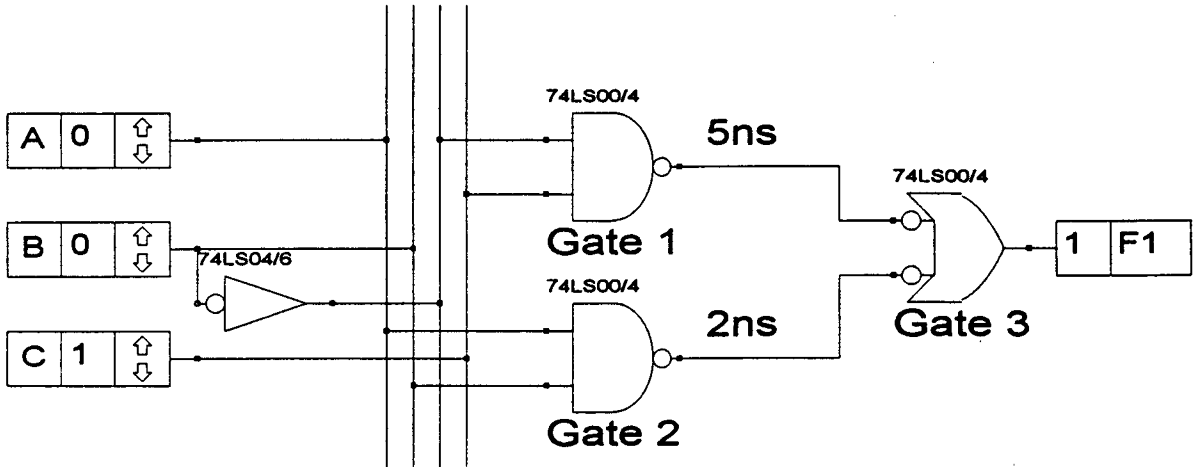
(paper no. 202-1006)

$$F1 = \overline{B} \cdot C + A \cdot B \qquad (2)$$

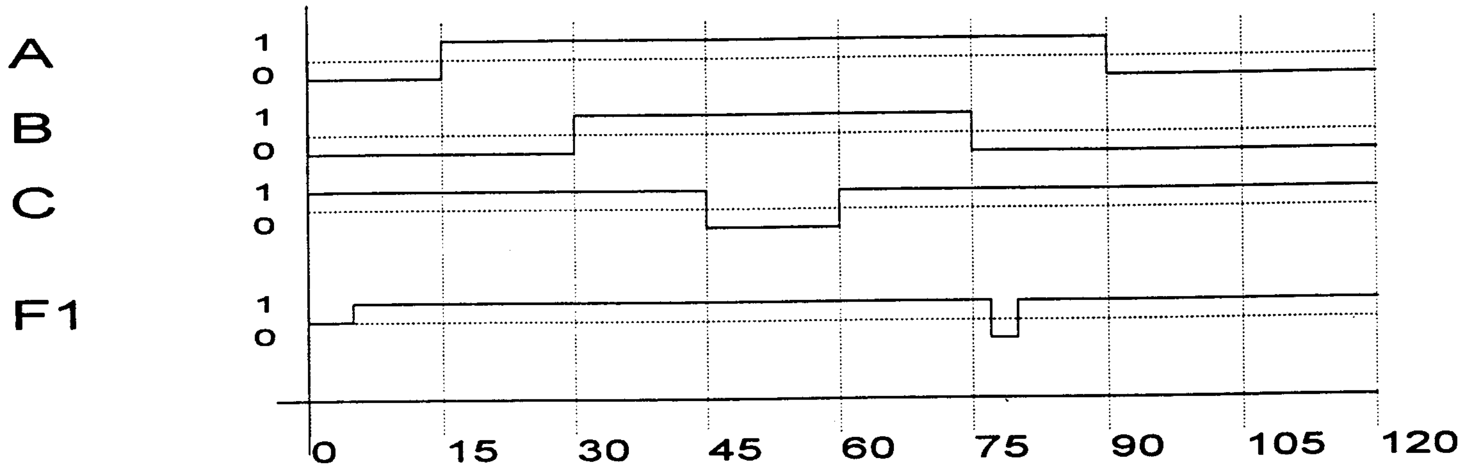
A static 1 logic hazard [5] can be detected in the Karnaugh map in fig. 1 by observing adjacent ones not covered in the minimized SOP form of the function, that is, not linked together. Notice that only one variable changes from 0 to 1 or from 1 to 0, as in the case of the adjacent ones in cells 5 and 7. For example, the function in the Karnaugh map is represented by inputs ABC, and as these inputs change from 101 to 111 or from 111 to 101 (variable B changes from 0 to 1 or from 1 to 0), a logic hazard exists in the circuit; that is, the output $F1$ of the combinational logic circuit in SOP form that is used to implement (2) can change from 1 to 0 then back to 1. Covering (linking) the adjacent ones in cells 5 and 7 to form the redundant prime implicant or hazard cover $A \cdot C$ and then logically adding this product term to (2) effectively eliminates the static 1 logic hazard between cells 5 and 7.

Fig. 2(a) shows a combinational logic circuit in SOP form for (2) using the software schematic capture tool, B^2 Logic. The simulation timing diagram for the circuit is shown in fig. 2(b). By inspecting the timing diagram in fig. 2(b), one can see that a logic 0 glitch occurs, that is, $F1$ momentarily goes to a 0, when variable B changes from 1 to 0 because the propagation delay $\Delta t1$ (5ns) is greater than $\Delta t2$ (2ns). Delays $\Delta t1$ and $\Delta t2$ represent the propagation delays that exist from the input signal line B (in this case) to the outputs of gates 1 and 2 respectively. When variable B changes from 1 to 0, gate 2 tries to make output $F1$ go to a 0 and gate 1 tries to make output $F1$ go to a 1. If gate 2's output is faster than gate 1's output, $F1$ momentarily goes to a 0. Changing the propagation delays such that $\Delta t1$ is less than $\Delta t2$, that is, gate 1 is faster than gate 2, causes a logic 0 glitch to occur when variable B changes from 0 to 1.

No other logic hazards exist between cells in fig. 1. Equation (3) has the same functionality as (2), only the logic hazard cover has been added to make (3) a logic-hazard-free (LHF) function.



(a)

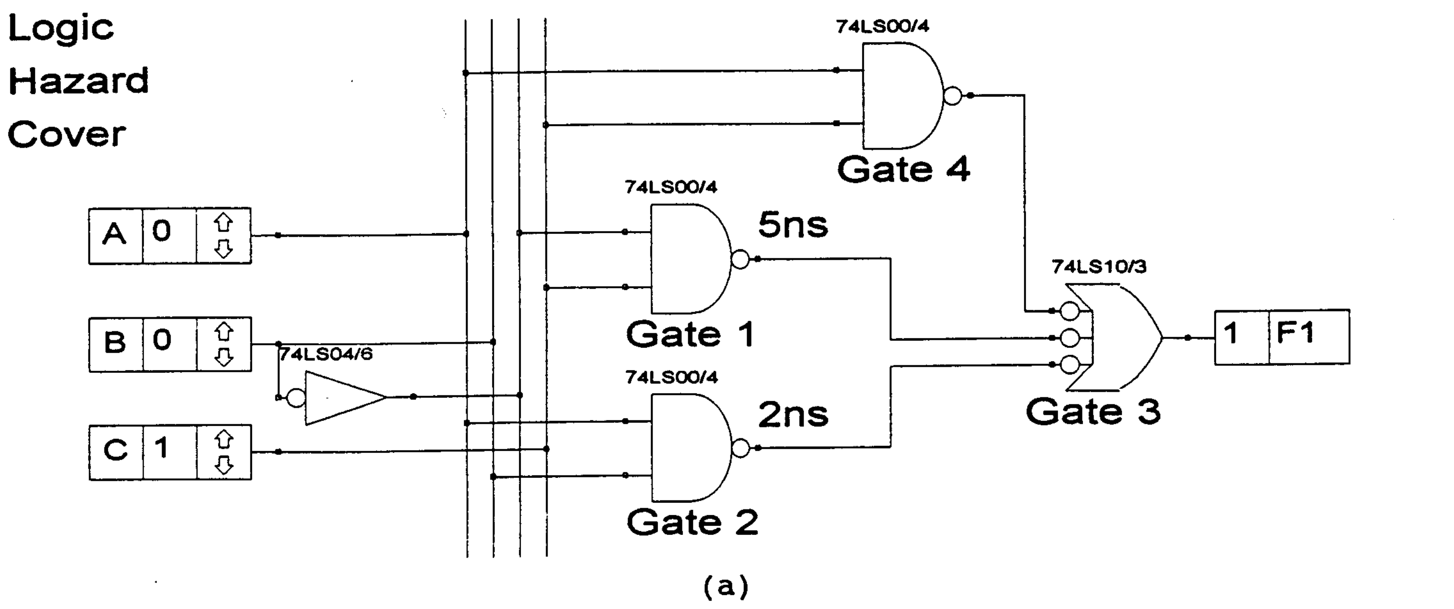


(b)

Figure 2. (a) Schematic for (2) using the software package B^2 Logic. (b) Simulation timing diagram showing a logic 0 glitch.

$$F1_{LHF} = \overline{B} \cdot C + A \cdot B + A \cdot C \tag{3}$$

The logic hazard cover in (3) insures that the output $F1_{LHF}$ will stay at a value of 1 when a single input variable changes from 0 to 1 or from 1 to 0, or output $F1_{LHF}$ will stay at a value of 0 when a single input variable changes from 0 to 1 or from 1 to 0 [5, 6]. Fig. 3(a) shows the circuit for the function $F1_{LHF}$ in (3).



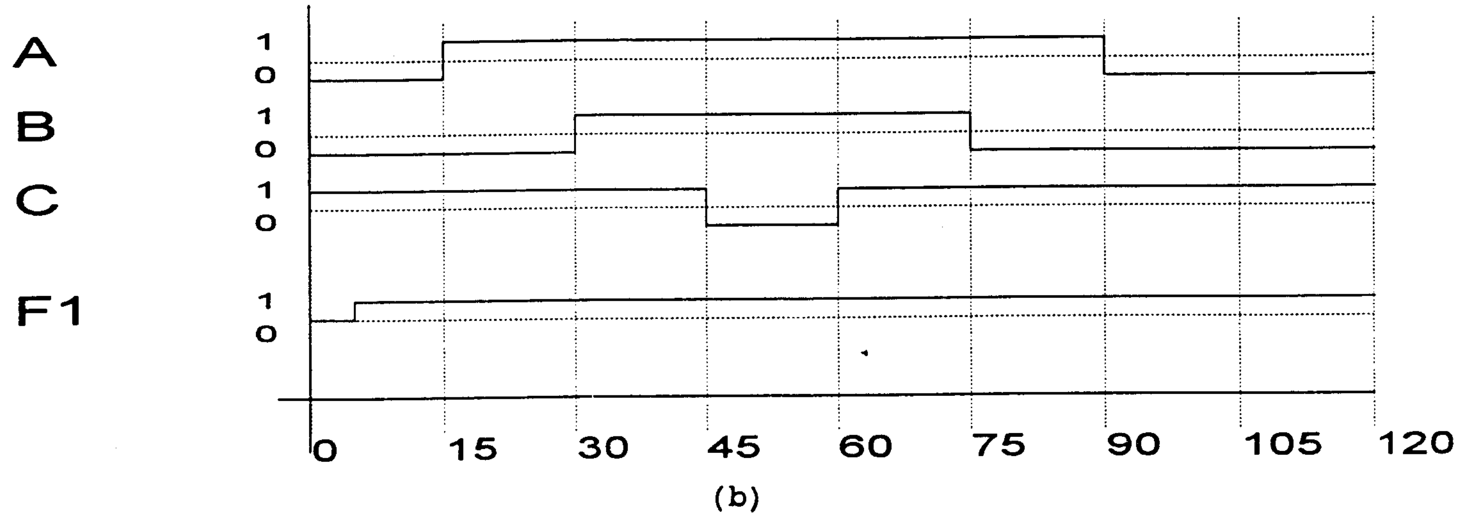


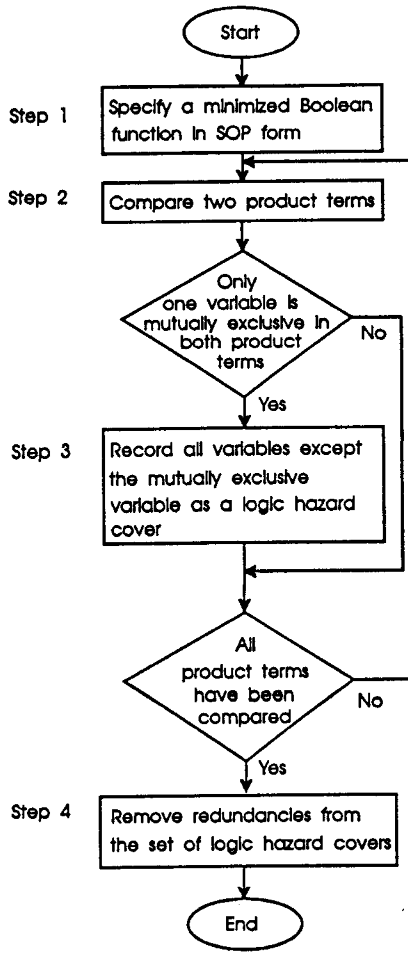
Figure 3. (a) Schematic for Equation 3 using the software package B^2 Logic, (b) Simulation timing diagram showing removal of the glitch.

The circuit in fig. 3(a) that includes the logic hazard cover has the same propagation delays as the circuit in fig. 2(a). Notice in the simulation timing diagram fig. 3(b) that the logic 0 glitch between cells 5 and 7 has been removed. Further testing also shows that for function $F1_{LHF}$ all logic 0 and logic 1 glitches resulting from static 1 and static 0 logic hazards have been removed.

3. Generation of Logic Hazard Covers via the Logic Hazard Cover Algorithm

For large or small functions, a systematic way of generating logic hazard covers for minimized SOP functions is represented by the logic hazard cover algorithm flow chart illustrated in fig. 4. The logic hazard cover algorithm allows one to identify all logic hazard covers without using a Karnaugh map. In general, Karnaugh maps are useful for perhaps up to six variables and then become very time consuming to draw and difficult to use. The advantage of using the logic hazard cover algorithm is that it can be used for any number of variables by either hand calculation or software synthesis.

Fig. 5(a) shows the hand calculation technique for computing the logic hazard covers for function F1 (equation (2)), and figs. 5(b) and (c) show the results of using the software tool, HAZARD, to calculate the logic hazard covers for function F1. The software tool is very useful for obtaining logic hazard covers and easily confirms the results obtained from either hand calculations or Karnaugh maps.



Start
Step 1
Specify a minimized Boolean function in SOP form
Step 2
Compare two product terms
Only one variable is mutually exclusive in both product terms
No
Yes
Step 3
Record all variables except the mutually exclusive variable as a logic hazard cover
All product terms have been compared
No
Yes
Step 4
Remove redundancies from the set of logic hazard covers
End

Figure 4. Logic hazard cover algorithm.

HAZARD, can be rerun with all logic hazard covers included with the minimum SOP part of the function, as shown in fig. 6(a). The result will indicate that there are no additional logic hazard covers required verifying the removal of all logic hazards, as illustrated in fig. 6(b). The tool can also be used in a "what if" mode by appending one or more logic hazard covers to the minimum SOP part of the function. Logic hazard covers not appended will be shown in the result.

$$F1 = \underbrace{\bar{B}\cdot C + A\cdot B}_{LHC = A\cdot C}$$

(a)

```
/BC
AB
```

(b)

```
*   input data:
/BC
AB
*  logic hazard covers:
AC
```

(c)

Figure 5. (a) Hand calculation technique for computing logic hazard covers for function F1 (equation (2)). (b) File generated using a text editor with the extension .LIN (Logic INput) for function F1. (c) Result of using HAZARD to calculate the logic hazard covers for function F1.

```
/BC
AB
AC
```

(a)

```
*   input data:
/BC
AB
AC
*  no logic hazard covers
```

(b)

Figure 6. (a) File generated using a text editor with the extension .LIN (Logic INput) for F1 in (3), a logic-hazard-free function. (b) Result of rerunning HAZARD to verify the removal of all logic hazards for function F1 in (3).

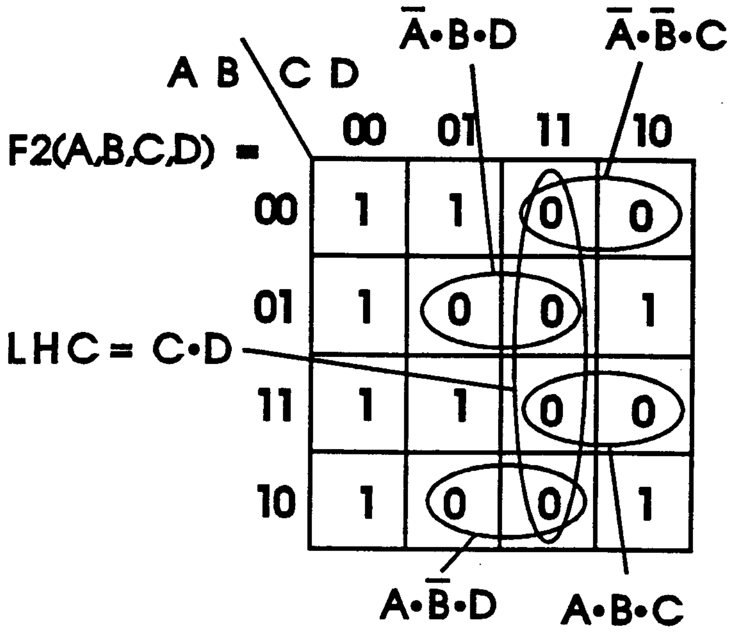


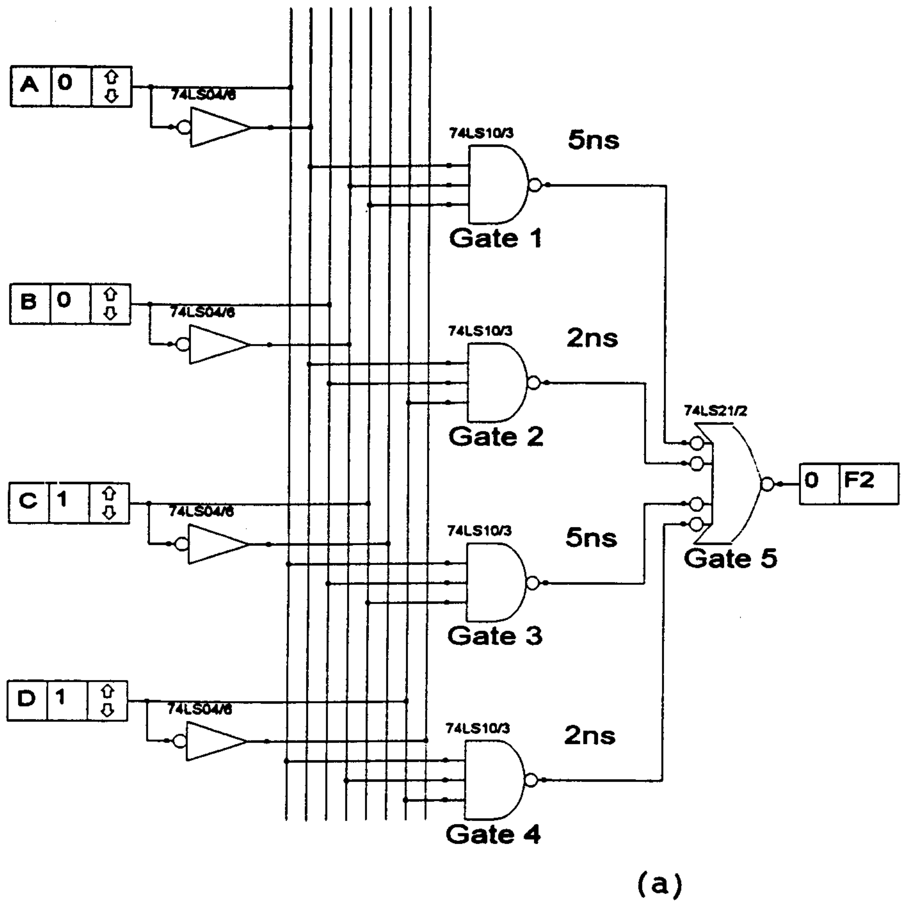
Figure 7. Karnaugh map for Boolean specification represented by (4).

4. Additional Examples

The following examples are used to further illustrate the methods of obtaining logic hazard covers. In each of the following functions it will be demonstrated that the most convenient approach is the software tool called HAZARD. This tool automates the search process by utilizing the logic hazard cover algorithm shown in fig. 4. Consider the following Boolean specification represented by (4):

$$F2(A,B,C,D) = \Pi M(2,3,5,7,9,11,14,15) \quad (4)$$

The four-variable Karnaugh map in fig. 7 illustrates this plotted specification. Equation (5) shows a minimized SOP equation for the *zeros* of the function $F2$.



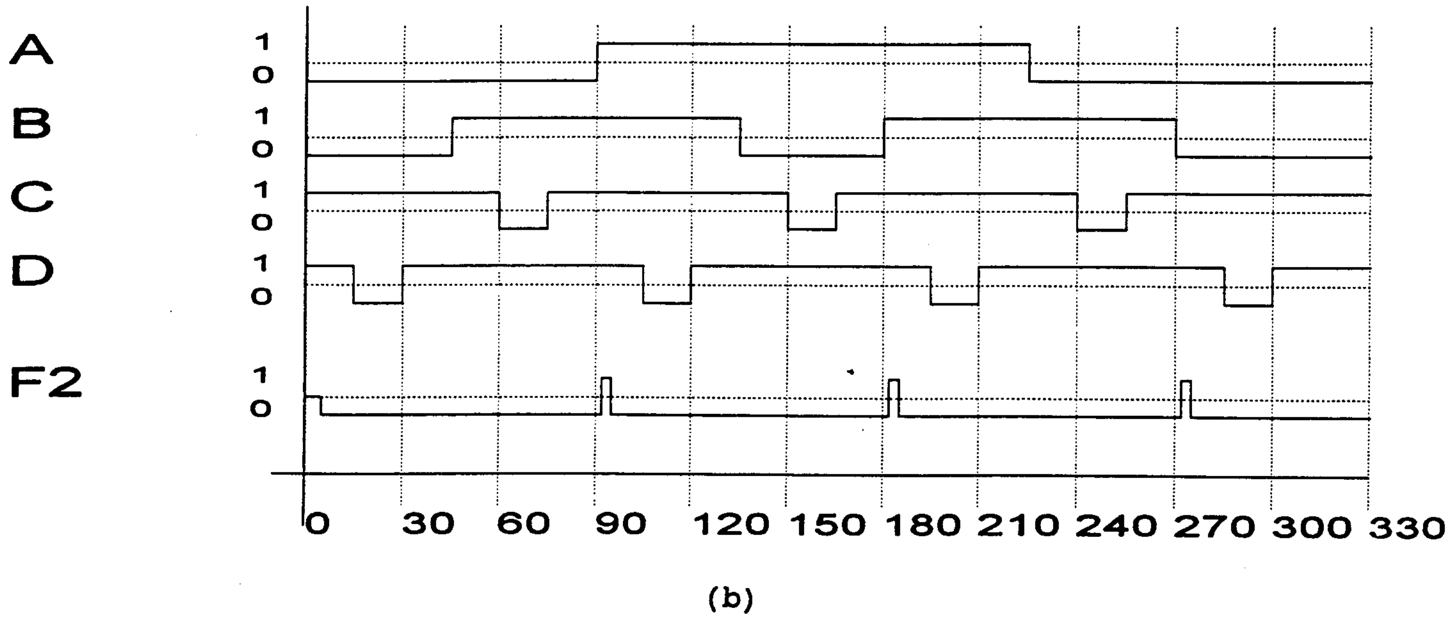


Figure 8. (a) Schematic for (5) using the software package B^2 Logic. (b) Simulation timing diagram showing three logic 1 glitches.

$$\overline{F2} = \overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot B\cdot D + A\cdot B\cdot C + A\cdot\overline{B}\cdot D \quad (5)$$

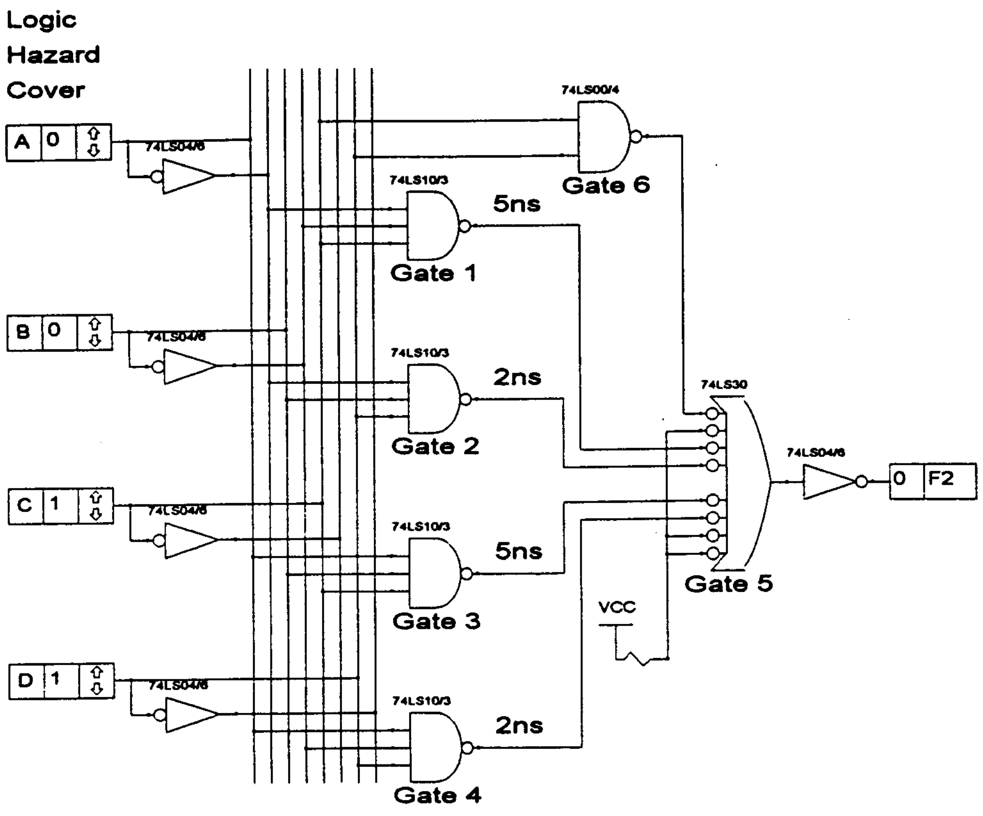
Four static 0 logic hazards can be detected in the Karnaugh map in fig. 7 by observing adjacent zeros not covered in the minimized SOP form of the function, that is, not linked together. Note that only one variable changes from 0 to 1 or from 1 to 0, as in the case of the adjacent zeros in cells 3 and 7, cells 7 and 15, cells 15 and 11, and cells 11 and 3, and the function $F2$ should maintain the value of 0. Fig. 8 shows a circuit implementation for (5) in SOP form along with a simulation timing diagram. Observe that the output $F2$ in the simulation timing diagram contains three logic 1 glitches that are due, in each case, to different delays in two or more signal paths and a change of only one input signal.

Logic hazard covers for (5) are obtained by simply covering the adjacent zeros in the cells that cause the three static 0 logic hazards in fig. 7 and writing the resulting product terms as follows:

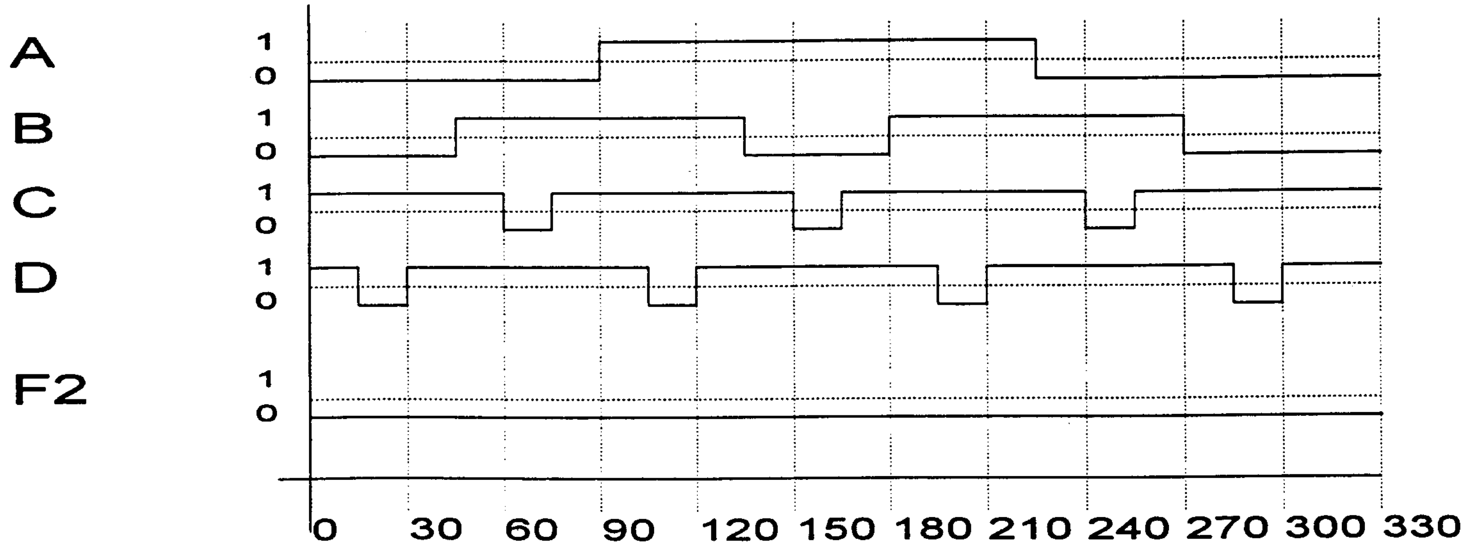
$$\overline{A}\cdot C\cdot D \quad B\cdot C\cdot D \quad A\cdot C\cdot D \quad \overline{B}\cdot C\cdot D.$$

Because all four of the zeros in the cells that cause the three static 0 logic hazards are adjacent to each other, only a single logic hazard cover $C \cdot D$ is required, as observed in fig. 7. Logically adding this logic hazard cover to (5) results in the logic-hazard-free function represented by (6).

$$\overline{F2}_{LHF} = \overline{A}\cdot\overline{B}\cdot C+\overline{A}\cdot B\cdot D+A\cdot B\cdot C+A\cdot\overline{B}\cdot D+C\cdot D \quad (6)$$



(a)



(b)

Figure 9. (a) Schematic for (6) using the software package B^2 Logic. (b) Simulation timing diagram showing the removal of the three logic 1 glitches.

$$\overline{F2} = \bar{A}\bullet\bar{B}\bullet C + \bar{A}\bullet B\bullet D + A\bullet B\bullet C + A\bullet\bar{B}\bullet D$$

$$LHC = \bar{A}\bullet C\bullet D$$

none

$$LHC = \bar{B}\bullet C\bullet D$$

$$LHC = B\bullet C\bullet D$$

none

$$LHC = A\bullet C\bullet D$$

Step 4: $\bar{A}\bullet C\bullet D + \bar{B}\bullet C\bullet D + B\bullet C\bullet D + A\bullet C\bullet D$
$= C\bullet D\bullet(\bar{A} + \bar{B} + B + A) = C\bullet D\bullet 1 = C\bullet D$

so, $LHC = C\bullet D$

(a)

```
/A/BC
/ABD
ABC
A/BD
```

(b)

```
*   input data:
/A/BC
/ABD
ABC
A/BD
*  logic hazard covers:
CD
```

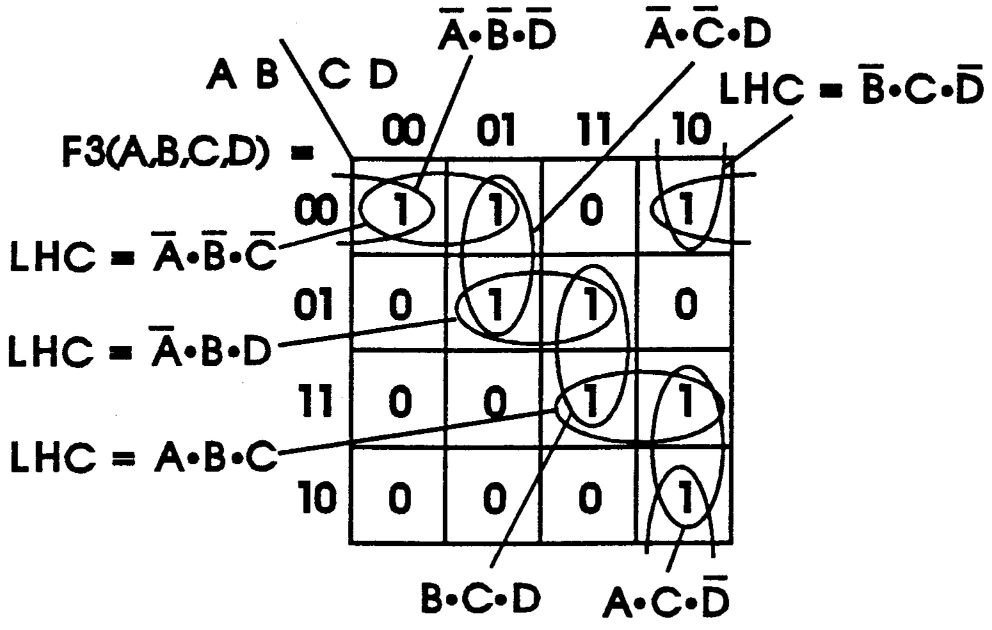
(c)

Figure 10. (a) Hand calculation technique for computing logic hazard covers for function F2 (equation (5)). (b) File generated using a text editor with the extension .LIN (Logic INput) for function F2. (c) Result of using HAZARD to calculate the logic hazard covers for function F2.

The logic hazard cover $C \cdot D$ is a redundant prime implicant and is not required except to make function $F2$ logic hazard free as illustrated in the simulation timing diagram shown in fig. 9(b). The circuit implementation for $F2$ in (6) is shown in fig. 9(a), and the resulting timing simulation in fig. 9(b) shows the removal of all three logic 1 glitches by the addition of the logic hazard cover term $C \cdot D$.

The hand calculation technique for computing the logic hazard covers for function $F2$ (equation (5)) is illustrated in fig. 10(a). Observe that step 4, "Remove redundancies from the set of logic hazard covers," in the logic cover algorithm means to minimize the logic hazard covers followed by eliminating any remaining logic hazard covers that are included in the set of product terms of the minimum SOP form of the function. Applying step 4 provides the single logic hazard cover $C \cdot D$. Figs. 10(b) and (c) show the results of using HAZARD to calculate the logic hazard covers for function $F2$. Observe in fig. 10(c) that the logic hazard covers obtained by HAZARD are the same as those obtained by both the Karnaugh map and the hand calculation techniques.

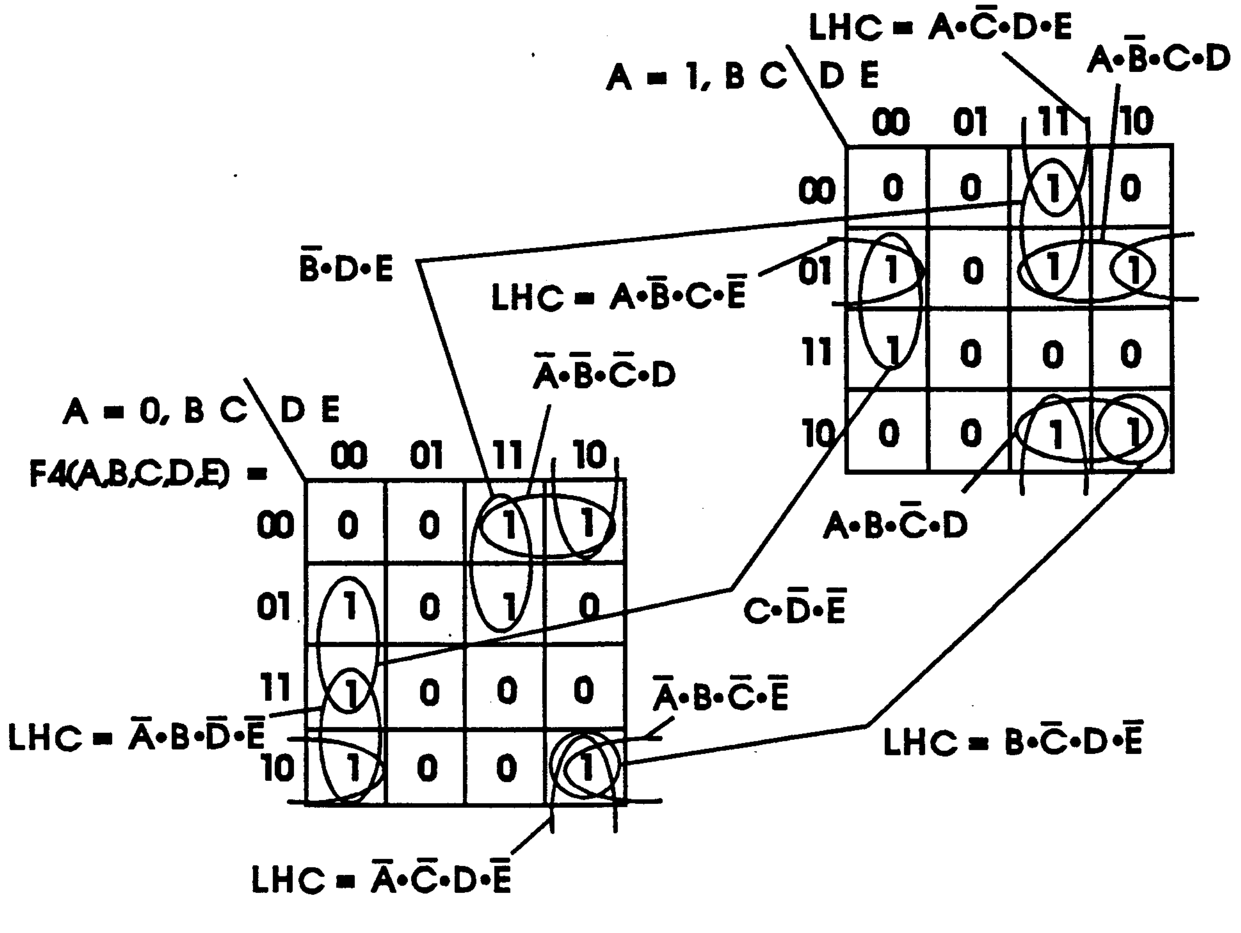
Figs. 11 and 12 show plotted Karnaugh maps for four and five variable functions $F3$ and $F4$ respectively. Each map shows a set of product terms for a minimized SOP



$\bar{A} \cdot \bar{B} \cdot \bar{D}$
$\bar{A} \cdot \bar{C} \cdot D$
A B \ C D
LHC = $\bar{B} \cdot C \cdot \bar{D}$
F3(A,B,C,D) =
00 01 11 10
LHC = $\bar{A} \cdot \bar{B} \cdot \bar{C}$
LHC = $\bar{A} \cdot B \cdot D$
LHC = $A \cdot B \cdot C$
$B \cdot C \cdot D$
$A \cdot C \cdot \bar{D}$

	00	01	11	10
00	1	1	0	1
01	0	1	1	0
11	0	0	1	1
10	0	0	0	1

Figure 11. Karnaugh map for four-variable function F3.



LHC = $A \cdot \bar{C} \cdot D \cdot E$
$A \cdot \bar{B} \cdot C \cdot D$
A = 1, B C \ D E
$\bar{B} \cdot D \cdot E$
LHC = $A \cdot \bar{B} \cdot C \cdot \bar{E}$
$\bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D$
$A \cdot B \cdot \bar{C} \cdot D$
A = 0, B C \ D E
F4(A,B,C,D,E) =
$C \cdot \bar{D} \cdot \bar{E}$
$\bar{A} \cdot B \cdot \bar{C} \cdot \bar{E}$
LHC = $\bar{A} \cdot B \cdot \bar{D} \cdot \bar{E}$
LHC = $B \cdot \bar{C} \cdot D \cdot \bar{E}$
LHC = $\bar{A} \cdot \bar{C} \cdot D \cdot \bar{E}$

A = 1	00	01	11	10
00	0	0	1	0
01	1	0	1	1
11	1	0	0	0
10	0	0	1	1

A = 0	00	01	11	10
00	0	0	1	1
01	1	0	1	0
11	1	0	0	0
10	1	0	0	1

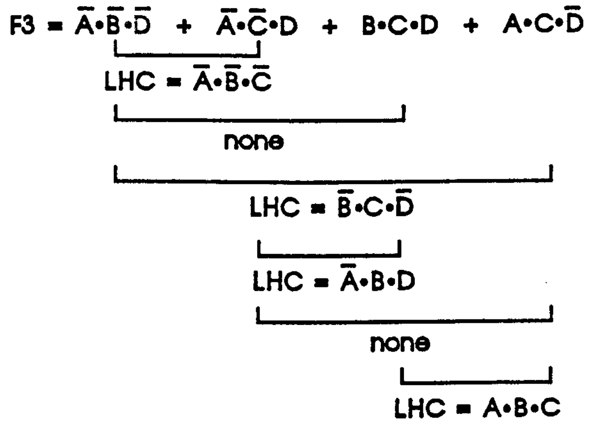
Figure 12. Karnaugh map for five-variable function F4.

equation for the *ones* of the respective function as well as the logic hazard covers for the minimized SOP equation.

Four static 1 logic hazards can be detected in fig. 11 by observing adjacent ones in the map not covered in the minimized SOP form of the function, that is, not linked together. In each case function $F3$ should maintain the value of 1 when a single input changes, but due to unequal delays along two or more signal paths in the implementation of the circuit for $F3$, a logic 0 glitch may occur.

Five static 1 logic hazards can be detected in fig. 12 by observing adjacent ones in the map not covered in the minimized SOP form of the function, that is, not linked together. A logic 0 glitch may occur as a result of each of these static 1 logic hazards.

The hand calculation technique for computing the logic hazard covers for function $F3$ is illustrated in fig. 13(a). Figs. 13(b) and (c) show the results of using HAZARD to calculate the logic hazard covers for function $F3$.



$F3 = \overline{A}\bullet\overline{B}\bullet\overline{D} + \overline{A}\bullet\overline{C}\bullet D + B\bullet C\bullet D + A\bullet C\bullet\overline{D}$
LHC = $\overline{A}\bullet\overline{B}\bullet\overline{C}$
none
LHC = $\overline{B}\bullet C\bullet\overline{D}$
LHC = $\overline{A}\bullet B\bullet D$
none
LHC = $A\bullet B\bullet C$

(a)

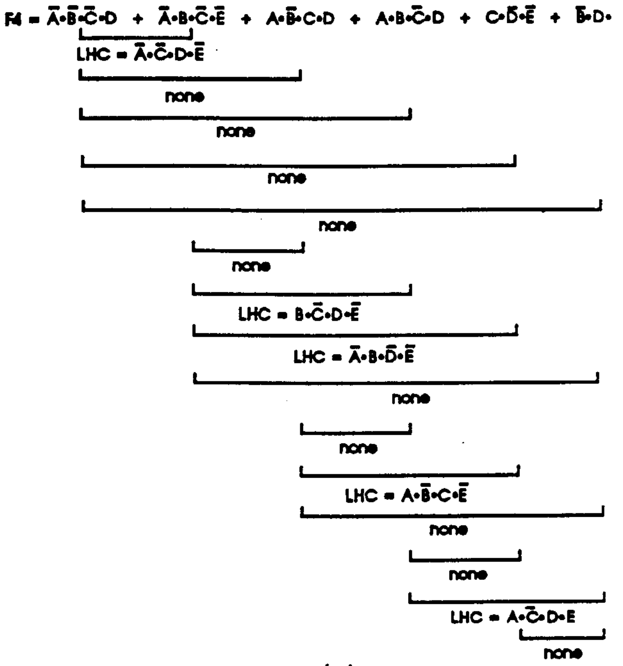
```
/A/B/D
/A/CD
BCD
AC/D
```

(b)

```
*   input data:
/A/B/D
/A/CD
BCD
AC/D
*  logic hazard covers:
/A/B/C
/BC/D
/ABD
ABC
```

(c)

Figure 13. (a) Hand calculation technique for computing logic hazard covers for function F3. (b) File generated using a text editor with the extension .LIN (Logic INput) for function F3. (c) Result of using HAZARD to calculate the logic hazard covers for function F3.



$F4 = \overline{A}\bullet\overline{B}\bullet\overline{C}\bullet D + \overline{A}\bullet B\bullet\overline{C}\bullet\overline{E} + A\bullet\overline{B}\bullet C\bullet D + A\bullet B\bullet\overline{C}\bullet D + C\bullet\overline{D}\bullet\overline{E} + \overline{B}\bullet D\bullet$
LHC = $\overline{A}\bullet\overline{C}\bullet D\bullet\overline{E}$
none
none
none
none
none
LHC = $B\bullet\overline{C}\bullet D\bullet\overline{E}$
LHC = $\overline{A}\bullet B\bullet\overline{D}\bullet\overline{E}$
none
none
LHC = $A\bullet\overline{B}\bullet C\bullet\overline{E}$
none
none
LHC = $A\bullet\overline{C}\bullet D\bullet E$
none

(a)

```
/A/B/CD
/AB/C/E
A/BCD
AB/CD
C/D/E
/BDE
```

(b)

```
*   input data:
/A/B/CD
/AB/C/E
A/BCD
AB/CD
C/D/E
/BDE
*  logic hazard covers:
/A/CD/E
B/CD/E
/AB/D/E
A/BC/E
A/CDE
```

(c)

Figure 14. (a) Hand calculation technique for computing logic hazard covers for function F4. (b) File generated using a text editor with the extension .LIN (Logic INput) for function F4. (c) Result of using HAZARD to calculate the logic hazard covers for function F4.

Observe in fig. 13(c) that the logic hazard covers obtained by the software tool are the same as those obtained by both the Karnaugh map technique in fig. 11 and the hand calculation technique in fig. 13(a).

As the number of variables in a Boolean representation increases, obtaining logic hazard covers becomes more difficult, as one would expect. The hand calculation technique for computing the logic hazard covers for function $F4$ is illustrated in fig. 14(a).

The logic hazard covers for function $F4$ are shown in figs. 14(c) as calculated by HAZARD. In each example provided in this paper the same logic hazard covers were obtained using three different techniques. The advantage of using the software tool for calculating logic hazard covers should be primarily the time saved. This is especially true when the software tool is compared to either the Karnaugh map or hand calculation method for a Boolean specification such as $F4$, which contains a large number of variables.

Function $F3$ can be written as a logic-hazard-free function as represented by (7).

$$F3_{LHF} = \overline{A}\cdot\overline{B}\cdot\overline{D}+\overline{A}\cdot\overline{C}\cdot D+B\cdot C\cdot D+A\cdot C\cdot\overline{D}$$
$$+\overline{A}\cdot B\cdot D+\overline{A}\cdot\overline{B}\cdot\overline{C}+A\cdot B\cdot C+\overline{B}\cdot C\cdot\overline{D} \quad (7)$$

Equation (8) is one representation of a logic-hazard-free function for the Boolean specification provided in fig. 12.

$$F4_{LHF} = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D+\overline{A}\cdot B\cdot\overline{C}\cdot\overline{E}+A\cdot\overline{B}\cdot C\cdot D+A\cdot B\cdot\overline{C}\cdot D$$
$$+C\cdot\overline{D}\cdot\overline{E}+\overline{B}\cdot D\cdot E+\overline{A}\cdot\overline{C}\cdot D\cdot\overline{E}+B\cdot\overline{C}\cdot D\cdot\overline{E}$$
$$+\overline{A}\cdot B\cdot\overline{D}\cdot\overline{E}+A\cdot\overline{B}\cdot C\cdot\overline{E}+A\cdot\overline{C}\cdot D\cdot E \quad (8)$$

Rerunning HAZARD with the logic-hazard-free function $F4$ in (8) verifies that no additional logic hazard covers are required in the function, as shown in fig. 15(b). This implies that a circuit implementation for function $F4_{LHF}$ in SOP form should result in the elimination of all logic 0 and logic 1 glitches.

It should be noted that the software tool can be used to generate the logic hazard covers for either the ones for a Boolean specification, as in the cases of $F1$, $F3$, and $F4$ in this paper, or the zeros for a Boolean specification, as in the case of $F2$.

5. Conclusion

This paper has presented three techniques for the identification of logic hazard covers that are used to obtain logic-hazard-free functions. The standard Karnaugh map technique was shown to be very easy to use for perhaps up to four variables. Both the hand calculation technique and the software tool technique use the logic hazard cover algorithm. The hand calculation technique is time consuming but generally can be used more easily than the Karnaugh map technique for Boolean specifications that contain a larger number of input variables. The software tool HAZARD provides the most flexibility, is the most convenient, and utilizes less time for calculating logic hazard covers. The timing simulator (B^2Logic) was used to show the presence of glitches caused by logic hazardous

```
/A/B/CD
/AB/C/E
A/BCD
AB/CD
C/D/E
/BDE
/A/CD/E
B/CD/E
/AB/D/E
A/BC/E
A/CDE
```

(a)

```
*   input data:
/A/B/CD
/AB/C/E
A/BCD
AB/CD
C/D/E
/BDE
/A/CD/E
B/CD/E
/AB/D/E
A/BC/E
A/CDE
*  no logic hazard covers
```

(b)

Figure 15. (a) File generated using a text editor with the extension .LIN (Logic INput) for F4 in (8), a logic-hazard-free function. (b) Result of rerunning HAZARD to verify the removal of all logic hazards for function F4 in (8).

functions and to verify the removal of glitches for logic-hazard-free functions. HAZARD was also used as a verification tool to show that minimized SOP forms of Boolean expressions that contain appropriate hazard covers do not contain logic hazards.

References

[1] R. S. Sandige, *Modern Digital Design* (New York: McGraw-Hill, 1990).

[2] C. Innes, "Avoiding Programmable Logic Hazards," *National Anthem, National Semiconductor,* January/February 1988, 5.

[3] R. F. Tinder, *Digital Engineering Design: A Modern Approach* (Englewood Cliffs, NJ: Prentice-Hall, 1991).

[4] J. A. Engelbert, *User Manual for B^2 Logic v2.2,* Beige Bag (B^2) Software, Ann Arbor, MI, 1990.

[5] S. H. Unger, *The Essence of Logic Circuits* (Englewood Cliffs, NJ: Prentice-Hall, 1989).

[6] E. J. McCluskey, *Logic Design Principles with Emphasis of Testable Semicustom Circuits* (Englewood Cliffs, NJ: Prentice-Hall, 1986).