Inadmissible Class of Boolean Functions under Stuck-at Faults

Debesh K. Das[1], *Debabani Chowdhury*[1], *Bhargab B. Bhattacharya*[2], *Tsutomu Sasao*[3]
[1]Computer Sc. & Engg. Dept., Jadavpur University, Kolkata -700 032, India
[2]ACM Unit, Indian Statistical Institute, Kolkata 700 108, India
[3]Dept. of Computer Science, Meiji University, Kawasaki, Kanagawa 214-8571, Japan
{(*debeshkdas, debabani.chowdhury, bhargab.bhatta*)@*gmail.com, sasao@cs.meiji.ac.jp*}

Abstract: Several underlying structural and functional factors that determine the fault behavior of a network are not yet well understood. In this paper, we show that there exists a large class of Boolean functions, called *root functions*, which can never appear as faulty response in an irredundant two-level circuit even when any arbitrary multiple stuck-at faults are injected. Conversely, we show that any other Boolean function can appear as a faulty response in an irredundant realization of some root function under certain stuck-at faults. We characterize this new class of functions and show that for *n* variables, their number is exactly equal to the number of independent dominating sets (Harary and Livingston, *Appl. Math. Lett.,* 1993) in a Boolean *n*-cube. Similar properties are observed for multiple-valued logic functions as well. Finally, we discuss its application to logic design and point out some open problems.

Keywords: Boolean functions, Multiple-valued functions, hypercube, redundancy, stuck-at faults, testing

1. Introduction

Although test generation and design-for-testability (DFT) issues have been studied extensively over a few decades, characterization of the impossible class of faulty functions (ICFF) [1], i.e., the set of Boolean functions that cannot appear as faulty response in a combinational circuit, is not well understood. Several structural factors determine the output functions in the presence of faults. Some early results regarding fault behavior under stuck-at faults were reported by Hayes [2], [3], Fujiwara [4], and by Das et al. [5].

In this paper, we introduce a new class of Boolean functions, called as *root functions*, and show that no such function can appear as a faulty response in any two-level irredundant AND-OR circuits under stuck-at faults. We prove that the set of all root functions and the set of their possible faulty response functions are not only mutually exclusive but their union collectively exhausts all 2^{2^n} Boolean functions of *n* variables for any *n*. More interestingly, we show that the number of such root functions is exactly equal to the number of independent domination in a hypercube of dimension *n* [6]. The concept of root functions is then extended to multiple-valued logic functions.

2. Preliminaries

Let $F(x_1, x_2, x_3, \ldots, x_n)$ be a switching function of *n* variables. A *literal* is a variable or its complement. A *minterm* is a product of literals in which every literal appears once [7]. A minterm (*m*) is a *true (false) minterm* if *F*(*m*) = 1 (0). The *Hamming distance between two minterms* is the number of bits in which they differ. A minterm is said to be adjacent to another minterm if their Hamming distance is unity. A Boolean function may be defined by the set of true minterms and can be expressed as a sum-of-products (s-o-p) form. An *n*-variable Boolean function *F* is said to be *non-vacuous* if *F* cannot be expressed with fewer than *n* variables.

Definition 1: [7] A sum-of-products (s-o-p) expression of a Boolean function is called *irredundant* if no term or literal can be deleted from the expression without altering the function.

An *implicant* is a product of literals covering one or more true minterms. A *prime implicant* of a function is an implicant that cannot be covered by another implicant with fewer literals. A Boolean function can be pictorially represented using a *Karnaugh map (K-map)* for small values of *n* [7].

Definition 2: [8] Two functions are said to be *equivalent* if one can be transformed to the other by (i) negation (*N*) of one or more input variables, and/or (ii) permutation (*P*) of two or more input variables. The functions that are equivalent under operation (i) form the *N-equivalent* class and those under operation (ii) form the *P-equivalent* class, and those equivalent under operations (i) and/or (ii) form the *NP-equivalent* class.

3. Root Functions in Binary Logic

In this section, we present new properties of root functions in the context of fault behavior, some results on their bounds, and their relationship to hypercubes.

3.1 Preliminaries and properties

Definition 3 [9]: A Boolean function *F* is said to be *isolated* if every true minterm of *F* is a prime implicant by itself, *i.e.*, no two true minterms of *F* are adjacent.

A true minterm is said to *dominate* a false minterm if the latter is adjacent to it.

A function F is called *maximal* if for every false minterm m_i of F, there exists at least one true minterm which dominates m_i.

Definition 4 [5]: A Boolean function F is a *root function* if F is non-vacuous, isolated and maximal.

Example 1: In Boolean functions of two variables, there are only two root functions, shown in Figure 1.

$x_2 \backslash x_1$	0	1
0	1	
1		1

(a)

$x_2 \backslash x_1$	0	1
0		1
1	1	

(b)

Figure 1: Two root functions for $n = 2$

(a)

$x_3 \backslash x_2x_1$	00	01	11	10
0	1			
1			1	

(b)

$x_3 \backslash x_2x_1$	00	01	11	10
0	1		1	
1		1		1

Figure 2: Two root functions for $n = 3$

Example 2: Figure 2 shows two examples of root functions of three variables.

Lemma 1: Let $N(R, n)$ be the number of true minterms in a root function $R(n)$ with n-variables. Then $N(R, n) \leq 2^{n-1}$.
Proof: See [17].

Lemma 2: $N(R, n) \geq \lceil 2^n/(n+1) \rceil$.
Proof: See [17].

Definition 5: A root function that contains the maximum number of true minterms is called *max-root*. A root function that contains the minimum number of true minterms is called *min-root.*
Corollary 1: For any n, there exist exactly two max-root functions R_1 and R_2 such that $N(R_1, n) = N(R_2, n) = 2^{n-1}$.
This follows from the construction used in the proof of Lemma 1. Clearly, $R_1 = x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_{n-1} \oplus x_n$ and $R_2 = 1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_{n-1} \oplus x_n$ are two max-root functions. In other words, the parity function and its complement are the only max-root functions. Also, the upper bound as stated in Lemma 1 is tight.

Proof: See [17].

Example 3: From Lemma 2, it follows that $N(R, n) \geq 2$, 2, 4, 6, 10 and 16 for n = 2, 3, 4, 5, 6 and 7 respectively. However, the lower bound is not always achievable for min-root functions. For n = 2, each of the functions shown in Figure 1(a) and 1(b) is a min-root and a max-root. For n =3, it is easy to check that the function shown in Figure 2(a) is a min-root, whereas that in Figure 2(b) is a max-root. For n =4, Figure 3 shows a min-root function where the lower bound on $N(R, n)$ is satisfied as $N(R, 4) \geq 4$. However, we will show next that for n = 5, a min-root function will consist of at least 8 true minterms, and thus the lower bound of $N(R, 5) \geq 7$ is not achievable.

Lemma 3: Let $R_1(n)$ be a root function of n variables such that $N(R_1, n) = k$. Then any function $R_2(n)$ that is obtained by complementing a literal in $R_1(n)$, is also a root function. In other words, if $R_2(n)$ is N-equivalent to $R_1(n)$, then $R_2(n)$ is a root function implying $N(R_1, n) = N(R_2, n) = k$.
Proof: See [17].

Theorem 1: Let $R(n)$ be a root function of n variables. If $N(R, n) = k$, then there exists a root function $R(n + 1)$ of $(n+1)$ variables such that $N(R, n+1) = 2k$.
Proof: See [17].

Example 4: Consider the root function in Figure 3(a) with two true minterms for n = 3. Based on this, we can construct a root function with four true minterms (0000, 0111, 1101, 1010) for n = 4, as shown in Figure 4. Similarly, we can construct a root function of five variables with eight true minterms (00000, 00111, 01101, 01010, 10001, 10110, 11100, 11011).

$x_4x_3 \backslash x_2x_1$	00	01	11	10
00	1			
01			1	
11		1		
10				1

Figure 3: A root function for n = 4

Remark 1: Although by Lemma 2, we have $N(R, 5) \geq 6$, for n = 5, the lower bound is not achievable. It is easy to prove from essential domination argument that the minimum number of true minterms for a root function of 5 variables is 8, an example of which is given earlier.

Example 5: A root function for n = 6 variables can be constructed by using Theorem 1 on the above example for n =5. This yields a root function with 16 true minterms: (000000, 000111, 001010, 001101, 010001, 010110, 011011, 011100, 100001, 100110, 101011, 101100, 110000, 110111, 111010, 111101). However, a min-root function for n = 6 consists of 12 true minterms: (000000, 000111, 001100, 011010, 011001, 010100, 101011, 100110, 100101, 110011, 111000, 111111). Thus, for n = 6, the lower bound for min-root function is not achievable.

Example 6: For n = 7, the lower bound is achievable as there exists a min-root of 16 true minterms: (0001 011, 0001 100, 0010 010, 0010 101, 0111 000, 0111 111, 0100 001, 0100 110, 1000 000, 1000 111, 1011 001, 1011 110, 1110 011, 1110 100, 1101 010, 1101 101).

Root functions other than min- or max-root

Lemma 1 and Lemma 2 provide an upper and a lower bound on the number $N(R, n)$ of true minterms in a root function $R(n)$ for n-variables. While the upper bound is always achievable for any n, the lower bound may not be achievable as demonstrated earlier by several examples. Here, we show that even between the two achievable bounds, not every value is feasible for a root function. For example, $N(R, 4)$ = 4 for a min-root (Figure 3) and $N(R, 4)$ = 8 for a max-root (4-variable parity function). However, for n =4, no root function exists with 6 or 7 true minterms.

Example 7: An example of a root function for four variables with five true minterms is shown in Figure 4.

x_2x_1 / x_4x_3	00	01	11	10
00	1			
01			1	
11		1		1
10			1	

Figure 4: A root function for n = 4 with $N(R, 4)$ = 5

3.2 Impossible class of faulty functions (ICFF)

In this subsection, we will show that the set of root functions plays an important role in partial characterization of ICFF in combinational circuits under the stuck-at fault model (single/multiple). A combinational circuit is said to be *irredundant* of all faults are detectable at the output, else it is called *redundant*. A single-output two-level AND-OR circuit will be irredundant only if it implements an irredundant s-o-p expression of a Boolean function. Since, in a redundant circuit ICFF is unpredictable [9], we consider irredundant realizations only.

Theorem 2: Let C be a single-output two-level irredundant combinational circuit realizing an n-variable Boolean function F_0. Let F_f denote the faulty response function under any stuck-at fault in C. Then F_f cannot be an n-variable root function.
Proof: See [17].
Theorem 3: Let F_f be a function of n variables. Then, there exists a root function $R(n)$ in whose irredundant two-level realization C, F_f will appear as a faulty response on injection of some stuck-at faults in C.
Proof: See [17].
From Theorems 2 and 3, we have the following:
Corollary 2: The set of all root functions and the set of their possible faulty response functions under stuck-at faults are mutually exclusive and collectively exhaustive over all Boolean functions of n variables for any n, when single-output two-level irredundant circuit realizations are considered.

3.3 Relation to independent dominating set

In this section, we show that the set of n-variable root functions has a one-to-one correspondence to the independent dominating sets in a hypercube of dimension n [6, 10].

Let $G = (V, E)$ denote a hypercube of dimension n, *i.e.*, $|V| = 2^n$ and there is an edge between two nodes in G if the Hamming distance between their binary representations is unity. A subset $S \in V$ is said to dominate G if every node in $V - S$ is adjacent to at least one node in S. If no two elements in S are adjacent in G, then S is called an independent dominating set. By $Q_{ind}(G)$ is meant the *minimum cardinality* of the independent dominating sets of G. Computation of minimum-size independent dominating set is a hard problem in graphs, although very good heuristic algorithms have recently been reported [11].

Since a Boolean function can be thought as a collection of 1-nodes (true minterms) in a hypercube, the following result is immediate.

Theorem 4: An n-variable Boolean function $R(n)$ is a root function if and only if $R(n)$ corresponds to an independent dominating set in a hypercube of dimension n.

Thus, $Q_{ind}(G)$ represents the number of true minterms in a min-root function. It has been reported [6] that $Q_{ind}(G)$ is 1, 2, 2 4, 8, 12, for n = 1, 2, 3, 4, 5, 6 respectively, which match with our findings. However, until now not much is known about $Q_{ind}(G)$ for higher values of n. Also, enumeration of total number of root functions (or the number of independent dominating sets in a hypercube) is not yet studied. In the next subsection, we present our simulation results.

3.4 Results and discussion

We have performed exhaustive enumeration of root functions for up to 6 variables and the results are reported in Table 1.

Table 1: Number of root functions for different numbers of true minterms in an *n*-variable function

# of variables *n*	# true minterms in root functions	# of root functions	Total number of root functions
2	2	2	2
3	2	4	6
3	4	2	
4	4	24	42
4	5	16	
4	8	2	
5	8	1140	1670
5	9	320	
5	10	176	
5	12	32	
5	16	2	
6	12	320	1,281,402
6	14	9600	
6	15	25920	
6	16	736440	
6	17	337920	
6	18	116320	
6	19	40320	
6	20	8320	
6	21	3840	
6	22	1856	
6	24	480	
6	27	64	
6	32	2	

Our experiments revealed many interesting results. For certain values of the integer k lying within the range of min-root size and 2^{n-1}, a root function with k true minterms may not exist. For example, when $n = 5$, no root function exists for four values of k: 11, 13, 15, and 14. Similarly, for $n = 6$, no root functions exist with 13, 23, 25, 26, 28, 29, 30, 31 true minterms. For each n, we enumerate the root functions with a possible number of true minterms (shown in the second column of the table). Also, all the root functions that are *NP*-equivalent, are counted in one group of the table.

3.5 Root functions as universal logic modules

As a consequence of Theorem 3, the root functions can serve as universal logic modules for realizing any Boolean function. Given two-level realizations of root functions, all other non-root functions can be produced at the outputs on injection of certain stuck-at faults. We illustrate this principle for $n = 3$.

For three variables, there are six root functions (Table 1). These functions can be classified into two NP-equivalence classes, which are shown in Figure 2(a) (consisting of two true minterms) and in Figure 2(b) (consisting of four minterms). We realize only these two functions by two-level AND-OR circuits, say C_1 and C_2 respectively. The remaining four root functions can be realized by permuting or complementing the input literals. We have checked that all the remaining 250 non-root functions can be produced from C_2 by injecting faults and by applying input permutations/literal complementation. Further, 118 functions can be produced from C_1 as well. Hence, the realizations of only two root functions are sufficient to produce all 256 functions of three variables. Similarly, three root functions are sufficient for producing all 65,536 four-variable functions.

4. Root Functions in Multiple-Valued Logic

When the scope of switching algebra is extended to the domain D: $\{0, 1, \ldots, (B\text{-}1)\}$ with B discrete levels, $B > 2$, it can be used to describe the behavior of Multiple-Valued Logic (MV Logic) circuits.
Let the MV logic variables be denoted as x_i, and the constant values are denoted with lower case letters such as *i, j, k*.

4.1 The operators and logic gates

The generated closed MV Logic algebra is the ordered set D consisting of the unary successor operator, two binary Maximum and Extended AND operators ($+$, $*^i$), and two elements 0 and (B-1).
Definition 6: The successor operator denoted by p^1, is defined as $p^1 = q$, with q being the next element after the element p in the (cyclic) ordered set (0, 1, 2…., B-1) [12]. The gate implementing the successor operator is known as a successor gate.
The successor operator when applied to variable x_i for p times is denoted by x_i^p. Note that $x_i^p = (x_i + p)$ *mod* B, where *mod* stands for the modulo operator and the symbol "+", in this case, stands for arithmetic addition. The "+" and "*" symbols stand for the Maximum and Minimum operator respectively, unless otherwise specified. We also call them as OR and AND operator respectively.

Definition 7: The maximum or OR operator denoted by $x_i + x_j$, is defined as $x_i + x_j = x_i$ if $x_i \geq x_j$, otherwise $x_i + x_j = x_j$ [25]. The gate performing this operation is known as OR gate.
Definition 8: The minimum or AND operator denoted by $x_i * x_j$, is defined as $x_i * x_j = x_i$ if $x_i \leq x_j$, otherwise $x_i * x_j = x_j$ [13]. The gate performing this operation is known as AND gate.

When the minimum operation is performed on some literals or successors of literals, we get the product term that is obtained at the output of the AND gate. Thus, the expression $x_i * x_j$ is a product term, where each of x_i, x_j can be a literal or a successor of a literal.
Definition 9: The extended AND operator (eAND) denoted by $x_i *^p x_j$, is defined as $x_i *^p x_j = p$ if $x_i = x_j = p$ otherwise $x_i *^p x_j = 0$ [14]. The gate performing this operation is known as an Extended AND gate. The expression $x_i *^p x_j$ is called an extended product term appearing at the output of the extended AND gate.

Figure 5 represents the gates performing the above-mentioned operations.

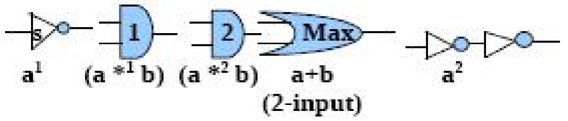

Figure 5: Representation of gates: Successor1, eAND^1, eAND^2, Maximum, Successor2

Definition 10: The EXOR operator (XOR) denoted by $x_i \oplus x_j$, is the value $(x_i + x_j)$ *mod* B, where '+' stands for arithmetic addition.
Definition 11: The canonical extended sum-of-products expression is represented as the application of maximum (OR) operation on product or extended product terms.
Definition 12: Given a canonical extended sum-of-products expression, if the product (extended product) terms realized by AND (eAND) gates are OR-ed, and if the inputs to AND or eAND gates represent literals or literals followed by successors, we obtain a two-level AND-OR multiple-valued circuit.

4.2 Faults in multiple-valued circuit and the roots

We will consider only stuck-at faults in a multiple-valued logic circuit [15]. A line h_i in a network is said to be stuck-at-q if the line is permanently fixed at the value q, where $0 \leq q \leq B$-1. This fault is denoted by $h;/q$. Clearly, for a k-line network, there are $(B+1)^k$ -1 possible faults in the network. We will consider only two-level AND-OR multiple-valued logic circuits.
Definition 13: A *root function* in multiple-valued logic is a function that can never appear as a faulty response in any irredundant two-level multiple-valued AND-OR circuit in the presence of a fault.
For a multiple-valued logic function of n variables with the domain D: {0, 1, . . ., (B-1)}, there are N (= B^n) possible input combinations and the total number of possible functions is B^N. Thus, in ternary logic where B=3, for n input literals, there are N (= 3^n) possible input combinations and the total number of possible functions is 3^N. For example, if n=2, there are 3^9 (=19683) possible functions.

Assume a two-level irredundant AND-OR realization of each of these B^N $(N=B^n)$ functions, and the presence of faults (single or multiple) therein; for each fault, there will be a corresponding faulty function. Thus, there are at least B^N different two-level AND-OR circuits (some functions may have more than one two-level irredundant realizations). We ask the following question: "does there exist any function that can never appear as a faulty response under a fault in any of these two-level realizations?" If it exists, such a function will be a root function in multiple-valued logic. The following theorem answers this question.

Theorem 5: The function $x_1 \oplus x_2 \oplus \ldots . \oplus x_n$ is a root function in multiple-valued logic.

Proof: If we realize a logic circuit representing the canonical extended sum-of-products expression of $x_1 \oplus x_2 \oplus \ldots . \oplus x_n$, each product term in this expression becomes a minterm where each literal exists ether in its original form or in the form of its successor. As a result of injecting a stuck-at fault, this product term either becomes a constant having a value among (0, 1, 2,…, B-1), or it grows to cover some adjacent minterms producing a different function. Moreover, in the resulting function, no two adjacent minterms can have the same value. Keeping everything else fixed, if we change the value of one literal from 0 to (B-1), we get different values of B in the output, and this property will be observed for every literal. Any fault in the circuit will render at least two adjacent minterms to have the same value. Thus, it is not possible to reach $x_1 \oplus x_2 \oplus \ldots . \oplus x_n$ as a faulty response. □
We conjecture that as in binary logic, there exist several root functions in multiple-valued logic. We also conjecture that it is possible to generate any non-root function by injecting suitable faults in a realization of some root functions.

4.3 Example of root functions

Let us consider the ternary logic functions with the base of (0, 1, 2).
The canonical extended sum-of-products expression of the function $a \oplus b$ is given by $a \oplus b = a_*^2 b^2 + a^1{}_*^2 b^1 +$

$a^2{*}^2b + a{*}^1b^2 + a^1{*}^1b + a{*}^1b^1$, where each product term represents a minterm. This function is a root function and hence no two-level irredundant circuit realizing any of the remaining 19682 functions can produce this function in the presence of a fault. A two-level irredundant ternary AND-OR circuit realizing this function is shown in Figure 6. We have observed that 823 functions (including three constant functions 0, 1 and 2) can be generated by injecting faults in this circuit.

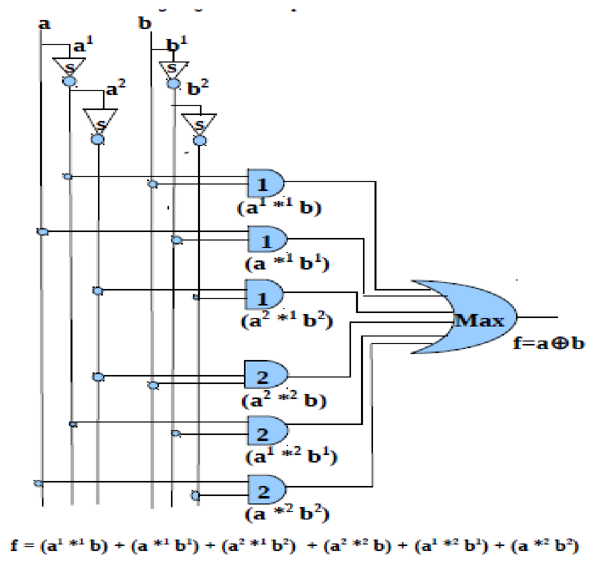

Figure 6: Two-level irredundant AND-OR circuit realizing the ternary logic function $a \oplus b$

Similarly, the complement of $a \oplus b$ will behave as a root function, which can also produce 823 different functions as faulty response. It has been observed that there are 73 common functions that appear in these two sets of faulty responses.

Note that for binary logic, i.e., when $n = 2$, there exist exactly two root functions that can generate all the remaining 14 Boolean functions. However, in ternary logic, $a \oplus b$ and its complement cannot exhaust the remaining set of 19681 functions. These functions may be generated by other root functions.

5. Conclusion

We have introduced a new class of functions called root functions, both for Boolean and multiple-valued logic. The rationale behind naming them as "roots" lies in the fact that none of these functions can appear as a faulty response in a single-output two-level irredundant circuit under any stuck-at fault. In binary logic, they can produce any other non-root functions by suitable injection of faults in their irredundant realizations, and we conjecture that the result will hold for multi-valued logic as well. Thus, these root functions can be used as universal logic modules to realize any function. For binary logic, we have shown that a root function is equivalent to an independent domination set in a hypercube and derived several new properties and bounds. We have enumerated the number of root functions up to six variables in Binary logic. There are many open problems related to this research, such as characterization of the inadmissible number of minterms in a root function and derivation of a closed formula for enumeration. Very recently, some authors are studying a special class of Boolean functions called "good functions", which are closely related to the notion of root functions [16].

References

1. B. B. Bhattacharya and B. Gupta, "On the impossible class of faulty functions in logic networks under short circuit faults," *IEEE Trans. Comput.,* vol. C-35, no. 1, pp. 85-90, Jan. 1986.
2. J.P. Hayes, "On realizations of Boolean functions requiring a minimal number of tests," *IEEE Trans. Comput.,* vol. C-20, pp. 1506-1513, Dec. 1971.
3. J.P. Hayes, "Transition count testing of combinational circuits," *IEEE Trans. Comput.,* vol. C-25, pp. 613-620, June 1976.
4. H. Fujiwara, "On closedness and test complexity of logic circuits," *IEEE Trans. Comput.,* vol. C-30, pp. 556-562, Aug. 1981.
5. D. K. Das, S. Chakraborty, and B. B. Bhattacharya, "Boolean algebraic properties of fault behavior in logic circuits," in *Proc. of Int. Workshop on Boolean Problems*, pp. 143-150, Sept., 2000.
6. F. Harary and M. Livingston, "Independent domination in hypercubes," *Appl. Math. Lett.,* vol. 6, no. 3, pp. 27-28, 1993.
7. Z. Kohavi, *Switching and Finite Automata Theory.* McGraw-Hill, Inc., 1970
8. S. L. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic.* Academic Press, 1985.
9. D. K. Das, S. Chakraborty, and B. B. Bhattacharya, "Interchangeable Boolean functions and their effects on redundancy in logic circuits," in *Proc. ASP-DAC*, pp. 469-474, 1998.
10. F. Harary, J.P. Hayes, and H.-J. Wu, "A survey of the theory of hypercube graphs," *Comput. Math. Applic.* vol. 15, no. 4, pp. 277-289, 1988
11. C. Laforest and R. Phan, "Solving the minimum independent domination set problem in graphs by exact algorithm and greedy heuristic," *RAIRO - Operations Research*, vol. 47, issue 03, pp. 199-221, July 2013.
12. K. C. Smith, "Multiple-Valued logic: A tutorial and appreciation," *IEEE Trans. Comput,* vol. 21, no. 4, pp. 17–27, April 1988.
13. G. Epstein, G. Frieder, and D. C. Rine, The Development of Multiple-Valued Logic as Related to Computer Science. In D. C. Rine, ed., *Computer Science and Multiple-Valued Logic: Theory and Applications*, pages 81–101, North-Holland, Amsterdam, 1977.
14. M. E. R. Romero, E. M. Martins, and R. R. Santos, "Multiple-valued logic algebra for the synthesis of digital circuits," *Proc. 39th Int. Symp. Multiple-Valued Logic,* pp. 262-267, 2009.
15. T. Raju Damarla, "Fault detection in multiple-valued logic circuits," *Proc. 12th Int. Symp. Multiple-Valued Logic,* pp. 69-74, 1990.
16. S. Maitra and E. Pasalic, "Some ideas about "good" functions," *Manuscript*, 2013.
17. D. K. Das, D. Chowdhury, and B. B. Bhattacharya, CSE Dept. Technical Report 6/2013, *Jadavpur University*, 2013.