On the Inadmissible Class of Multiple-Valued Faulty-Functions under Stuck-at Faults

Debabani Chowdhury and Debesh K. Das
Computer Sc. & Engg. Dept.
Jadavpur University, Kolkata 700 032, India
Email: *debabani.chowdhury@gmail.com*; *debeshd@hotmail.com*

Bhargab B. Bhattacharya
ACM Unit, Indian Statistical Institute
Kolkata 700 108, India
Email: *bhargab.bhatta@gmail.com*

Tsutomu Sasao
Dept. of Computer Science
Meiji University, Kawasaki
Kanagawa 214-8571, Japan
Email: *sasao@cs.meiji.ac.jp*

***Abstract*— There exists a class of Boolean functions, called root-functions, which can never appear as faulty response in irredundant two-level AND-OR combinational circuits even when any arbitrary multiple stuck-at faults are injected. However, for multi-valued logic circuits, root-functions are not yet well understood. In this work, we characterize some of the multiple-valued root-functions in the context of irredundant two-level AND-OR multiple-valued circuit realizations. As in the case of binary logic, such a function can never appear as a faulty-function in the presence of any stuck-at fault. We present here a preliminary study on multiple-valued root-functions for ternary (3-valued) logic circuits, and identify a class of n-variable ternary root-functions using a recursive method called *concatenation*. Such an approach provides a generalized mechanism for identifying a class of root-functions for other p-valued ($p > 3$), n-variable, two-level AND-OR logic circuits. Furthermore, we establish an important connection between root-functions and the classical latin-square functions.**

Index Terms- Latin-square functions, multiple-valued logic, stuck-at faults, ternary functions, root-functions

I. Introduction and Preliminaries

In a recent work [1], it has been shown that there exists a class of Boolean functions, called root-functions, which never appear as faulty response when an arbitrary single or multiple stuck-at faults are injected in an irredundant two-level AND-OR circuit realization of a Boolean function. Root-functions also play an important role in the characterization of *Impossible Class of Faulty-Functions* (ICFF) [3] under various test models. However, for multiple-valued functions, very little is known about the existence of such root-functions.

The scope of switching algebra can be extended to the domain $D = \{0, 1, ..., (p-1)\}$ of p discrete levels, $p > 2$, to describe the behaviour of Multiple-Valued Logic (MVL) circuits. We consider here single-output, two-level AND-OR MVL circuits, and study the properties of multiple-valued root-functions in such context. As in the case of binary logic, we define MVL root-functions as those, which can not appear as faulty-functions when an arbitrary single or multiple stuck-at faults are injected in irredundant two-level AND-OR MVL realizations of a multiple-valued function. Some preliminary concept of ternary-valued root-functions, i.e. for $p = 3$, were introduced in an earlier work [1]. Here, we explore, in-depth, the underlying properties of root-functions for MVL and their connections to another interesting class of functions known as latin-square functions [12].

Note that for any multiple-valued logic function of n variables with the domain $D = \{0, 1, ..., (p-1)\}$, there are $N = p^n$ possible input combinations, and the total number of possible functions is p^N. Thus, for the ternary domain where $p = 3$, for n input variables, there are $N = 3^n$ possible input combinations and the total number of possible functions will be 3^N. For example, when $n = 2$, there are $3^9 = 19683$ possible functions. Thus, there are at least p^N different two-level AND-OR circuits (some functions may have more than one two-level irredundant realizations). Assume a two-level irredundant AND-OR realization for each of these $p^N (N = p^n)$ functions. Now consider the presence of single or multiple faults in the circuits. For each fault, there will be a corresponding faulty-function (denoting the output function when the fault is injected). The question is, whether there exists any function that can never appear as a faulty-function for any possible fault in any of the two-level realizations. If it exists, then that function is a root-function. It is conjectured that there would also exist several root-functions for multiple-valued logic circuits as in the case of Boolean functions [1]. The identification of some of these root-functions among the set of all p^N functions may help characterization of impossible class of multiple-valued faulty-functions, as well. In this paper, we show that there exists a multitude of MVL functions that behave as root-functions and hence, establish that the earlier conjecture [1] is indeed true.

Root-functions in the ternary domain $D = \{0, 1, 2\}$ are named as ternary root-functions. In this paper, we show that many root-functions do exist in the context of two-level implementation of ternary logic as in the binary $B = \{0, 1\}$ domain [1]. We also establish an interesting connection of ternary root-functions with the classical ternary latin-square functions [12] and show that the latter set is a subset of the set of former type.

In order to facilitate the identification of ternary root-functions, we propose a recursive procedure called *concatenation* that allows us to construct an n-variable ternary root-function from three $(n-1)$-variable ternary root-functions. We generalize the method of concatenation in connection to root-functions to make it suitable for the Boolean, ternary, or for other higher p-valued functions, i.e., where $D = \{0, 1, 2, ..., (p-1)\}$, $p \geq 2$. We show that an n-variable Boolean (ternary) function can be

0195-623X/16 $31.00 © 2016 IEEE
DOI 10.1109/ISMVL.2016.55

IEEE computer society

constructed from two (three) $(n-1)$-variable Boolean (ternary) functions. We generate all 2-variable and 3-variable ternary latin-square functions [12], that are ternary root-functions, as well; also each of them is a max-root-function, i.e., it includes the maximum number of product terms in its minimal sum-of-product expression.

II. Background

Let $X = \{x_1, x_2, ..., x_n\}$ be a set of n-variables, where x_i takes on values from $D = \{0, 1, ..., p-1\}$. A function $F(X)$ is a mapping $F : D^n \rightarrow D$. Specifically, $F(X)$ is said to be an n-variable p-valued function. For $p = 3$, function $F(X)$ is said to be a ternary function. A function value $F(x)$ corresponding to a specific assignment of values x to variables in X is called a minterm.

Example 1: Fig. 1 shows an example of a 2-variable ternary function $F(x, y)$ having three minterms with value 1 and five minterms with value 2.

	$y^0 = 0$	$y^1 = 1$	$y^2 = 2$
$x^0 = 0$	2	1	2
$x^1 = 1$	1	2	0
$x^2 = 2$	2	2	1

Fig. 1. An example of a 2-variable ternary function $F(x, y)$ in its map-representation

Definition 1: The unary operator on the p-valued variable x, called a literal, is denoted by $x^b = p-1$ when $x = b$, otherwise 0.

Definition 2: Max operator $(\vee)$ returns the maximum of its two operands. Let a and b be two operands; then max function $a \vee b = max(a, b)$.

Definition 3: Min operator (.) returns the minimum of its two operands. Let a and b be two operands; then min function $a \wedge b = (a.b) = min(a, b)$.

Definition 4: A sum-of-product expression for function $F(X)$ is minimal if there is no other expression for $F(X)$ with fewer product terms or literals.

The minimal sum-of-product expression for a 2-variable ternary function F in Fig. 1 is $F(x, y) = (1.x^0.y^1) \vee (1.x^1.y^0) \vee (1.x^2.y^2) \vee (x^0.y^2) \vee (x^1.y^1) \vee (x^2.y^0)$.

Definition 5: Stuck-at-fault (s-a-f) [5]: A line h_i in a network is said to be stuck-at-q if a fixed logic value q set at this line, models the effect of the fault at the circuit output, where $q \in \{0, 1, \cdots, p-1\}$. This fault is denoted by h_i/q. Clearly, in a circuit with k lines, there are $(p+1)^k - 1$ possible faults in the network.

Definition 6: [13] A combinational circuit is said to be irredundant if all stuck-at faults, single or multiple, are detectable by input-output experiments.

Definition 7: [7]: A Boolean root-function is a logic function that can never appear as a faulty response in any irredundant two-level AND-OR logic circuit in the presence of any arbitrary (single or multiple) stuck-at faults.

Example 2: Fig. 2 shows an example of 4-variable Boolean root-function f with true vectors (0000,0111,1100,1001,1010). With respect to stuck-at faults, the root-functions for multiple-valued logic is defined as follows.

Definition 8: 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 any arbitrary (single or multiple) stuck-at faults.

	$x_3'.x_4' = 00$	$x_3'.x_4 = 01$	$x_3.x_4 = 11$	$x_3.x_4' = 10$
$x_1'.x_2' = 00$	1	0	0	0
$x_1'.x_2 = 01$	0	0	1	0
$x_1.x_2 = 11$	1	0	0	0
$x_1.x_2' = 10$	0	1	0	1

Fig. 2. 4-variable Boolean root-function $f(x_1, x_2, x_3, x_4)$ in its map-representation with five true minterms

Definition 9: The Boolean root-functions that contain the maximum number of true minterms are called max-root-functions.

Example 3: Fig. 3 shows an example of a 3-variable Boolean max-root-function $M(x_1, x_2, x_3)$ with maximum (four) number of true minterms.

	$x_2'.x_3' = 00$	$x_2'.x_3 = 01$	$x_2.x_3 = 11$	$x_2.x_3' = 10$
$x_1' = 0$	1	0	1	0
$x_1 = 1$	0	1	0	1

Fig. 3. 3-variable Boolean max-root-function $M(x_1, x_2, x_3)$ in its map-representation with maximum (four) number of true minterms

Obviously, a max-root-function contains maximum number of product terms in their minimal sum of-products expression. In this light, we can define the max-root-functions for ternary logic as follows.

Definition 10: An n-variable ternary root-function that has the maximum number of product terms in their minimal sum-of-products expression is a ternary max-root-function.

Example 4: Fig. 4 shows 2-variable ternary max-root-function $H(x, y)$ with maximum (six) number of product terms.

	$y^0 = 0$	$y^1 = 1$	$y^2 = 2$
$x^0 = 0$	0	1	2
$x^1 = 1$	1	2	0
$x^2 = 2$	2	0	1

Fig. 4. An example of a 2-variable ternary max-root-function $H(x, y)$ in its map-representation

Definition 11: [12] A permuter functions $P(x)$ of a p-valued variable x is a function such that for no two distinct values of x, the function assumes the same value.

Definition 12: [12] A latin-square function $f(x_1, x_2, ..., x_n)$ is a function that satisfies the following property: $\forall i = 0, 1, \cdots, n,\ f(a_1, a_2, ..., a_{i-1}, x_i, a_{i+1}, ..., a_n) = g(x_i)$ is a permuter function on x_i for any assignment of values $(a_1, a_2, ..., a_{i-1}, a_{i+1}, ..., a_n)$ to $(x_1, x_2, ..., x_{i-1}, ..., x_{i+1}, ..., x_n)$.

Example 5: Fig. 4 shows an example of a 2-variable ternary latin-square function $H(x, y)$ in its map-representation.

III. Method of Concatenation in Binary Logic

The concatenation operation on Boolean functions can be used recursively to construct new functions with a larger number of variables [9], [10]. The method of concatenation was used earlier for the construction of resilient Boolean functions in a different context [9], [10]. In fact, for binary logic, an n-variable (for even n) Maiorana-McFarland type of bent function can be constructed by concatenating $2^{\frac{n}{2}}$ distinct affine functions on $\frac{n}{2}$ variables. Later, such ideas were used to construct Boolean functions with versatile cryptographic properties [9], [10]. For an illustration of this method, let us consider two $(n-1)$-variable Boolean functions, g, h. For an instance, an n-variable Boolean function f_1 can be generated from g, h by appending 0 with every true vector of g and appending 1 with every true vector of h, and then selecting those appended vectors as true vectors of f_1. Again another n-variable Boolean function f_2 can be generated from g, h by appending 1 with every true vector of g and appending 0 with every true vector of h, and then selecting those appended vectors as true vectors of f_2. Thus, the concatenation between g and h can be expressed as: $f_1 = x_n^{'} g \vee x_n h$, $f_2 = x_n^{'} h \vee x_n g$. We use this concatenation technique to construct larger root-functions from basic root-functions as follows.

Procedure Root-through-Concatenate(n)

1. Consider two root-functions R_1 and R_2 of $(n-1)$-variables $(x_{n-1}, x_{n-2}, ..., x_2, x_1)$ where $R_1 \cap R_2 = \varnothing$.
2. Append x_n with each of R_1 and R_2 with values 0(1) and 1(0) respectively to construct functions $f_1(f_2)$ of n-variables. Let appended x_n with 0(1) be represented as $x_n^0(x_n^1)$. Then, $f_1 = x_n^0 R_1 \vee x_n^1 R_2$ and $f_2 = x_n^1 R_1 \vee x_n^0 R_2$.

Example 6: Figures 5(a) and 5(b) show 3-variable Boolean root-functions g and h with true vectors 000, 111 and 001, 110 in their map-representations, respectively. The function f_1 can be generated from g and h by the method of concatenation by appending 0 with every true vector of g as shown in Fig. 6, and that obtained by appending 1 with every true vector of h is shown in Fig. 7. The root-function obtained in this manner for $n = 4$ is shown in Fig. 5(c).

1	0	0	0
0	0	1	0

(a) $g(000, 111)$

0	1	0	0
0	0	0	1

(b) $h(001, 110)$

1	0	1	0
0	0	0	0
0	1	0	1
0	0	0	0

(c) $f_1(0000, 0011, 1101, 1110)$

Fig. 5. 4-variable Boolean root-functions f_1 generated by concatenation between two 3-variable root-functions g and h, where $f_1 = x_n^{'} g \vee x_n h$

vectors of g before appending 0	vectors of g after appending 0	vectors in g with values	vectors in f_1 with values
000	0000	$g(000) = 1$	$f_1(0000) = 1$
001	0010	$g(001) = 0$	$f_1(0010) = 0$
011	0110	$g(011) = 0$	$f_1(0110) = 0$
010	0100	$g(010) = 0$	$f_1(0100) = 0$
100	1000	$g(100) = 0$	$f_1(1000) = 0$
101	1010	$g(101) = 0$	$f_1(1010) = 0$
111	1110	$g(111) = 1$	$f_1(1110) = 1$
110	1100	$g(110) = 0$	$f_1(1100) = 0$

Fig. 6. Vectors in f_1 with values produced from true vectors in g

vectors of h before appending 1	vectors of h after appending 1	vectors in h with values	vectors in f_2 with values
000	0001	$h(000) = 0$	$f_1(0001) = 0$
001	0011	$h(001) = 1$	$f_1(0011) = 1$
011	0111	$h(011) = 0$	$f_1(0111) = 0$
010	0101	$h(010) = 0$	$f_1(0101) = 0$
100	1001	$h(100) = 0$	$f_1(1001) = 0$
101	1011	$h(101) = 0$	$f_1(1011) = 0$
111	1111	$h(111) = 0$	$f_1(1111) = 0$
110	1101	$h(110) = 1$	$f_1(1101) = 1$

Fig. 7. vectors in f_1 with values produced from vectors in h

IV. Root-Functions in Ternary Logic

We adopt the concatenation technique to produce ternary root-functions. The method is also applicable for a general multiple-valued logic system with a slight modification. Here, we identify only those root-functions that satisfy the conditions for being max-root as well as latin-square.

A. 1-Variable Ternary Root-Functions

The total number of 1-variable ternary logic functions is 3^{3^1} = 27. Among them, we show functions $g_{(1,1)}$, $g_{(1,2)}$, $g_{(1,3)}$, $g_{(1,4)}$, $g_{(1,5)}$, $g_{(1,6)}$, $g_{(1,7)}$, $g_{(1,8)}$, $g_{(1,9)}$ in Fig. 8; each of these nine functions is also a root-function.

Function			
(a) $g_{(1,1)}$	0	1	2
(b) $g_{(1,2)}$	1	2	0
(c) $g_{(1,3)}$	2	0	1
(d) $g_{(1,4)}$	0	2	1
(e) $g_{(1,5)}$	1	0	2
(f) $g_{(1,6)}$	2	1	0
(g) $g_{(1,7)}$	1	2	2
(h) $g_{(1,8)}$	2	1	2
(i) $g_{(1,9)}$	2	2	1

Fig. 8. All 1-variable ternary root-functions $g_{(1,1)}$, $g_{(1,2)}$, $g_{(1,3)}$, $g_{(1,4)}$, $g_{(1,5)}$, $g_{(1,6)}$, $g_{(1,7)}$, $g_{(1,8)}$, $g_{(1,9)}$ in their map- representation

All 1-variable ternary constant functions $f_{000} = 0$, $f_{111} = 1$ and $f_{222} = 2$ are reachable from these nine functions. Besides that, table 1 shows all other faulty-functions reachable from these nine functions. Thus, all other functions are reachable from these nine functions when suitable stuck-at faults are injected in their two-level irredundant AND-OR MVL circuit realization. It can be also shown that none of these nine functions are reachable from any function under a faulty-condition. Hence, these nine functions are root-functions. Moreover, each of these functions has two product terms in their minimal sum-of-product expression. For $n = 1$, the number of maximum product terms in minimal sum-of-product expression of a ternary function is also two. Thus, for $n = 1$, there does not exist any other root-function other than the max-root-functions. Among these nine functions, six functions $g_{(1,1)}$, $g_{(1,2)}$, $g_{(1,3)}$, $g_{(1,4)}$, $g_{(1,5)}$, $g_{(1,6)}$ are latin-square functions.

Root-Functions (R)	Faulty-Functions Reachable From Corresponding R			
0 1 2	0 0 2	1 1 2	1 1 0	0 1 0
0 2 1	0 2 0	1 2 1	1 0 1	0 0 1
1 0 2	0 0 2	1 1 2	1 1 0	0 0 0
1 2 0	0 2 0	1 2 1	1 0 1	1 0 0
2 0 1	2 0 0	2 1 1	0 1 1	0 0 1
2 1 0	2 0 0	2 1 1	0 1 1	0 1 0
1 2 2	2 0 0	2 1 1	0 2 2	1 0 0
2 1 2	0 2 0	1 2 1	2 0 2	0 1 0
2 2 1	0 0 2	1 1 2	2 2 0	0 0 1

TABLE I

Reachability from 1-variable root-functions to faulty-functions

B. Construction of Root-Function in Ternary Logic

We use the concatenation procedure to construct root-functions when $n > 1$. In binary logic, two binary root-functions are required for every concatenation. In the case of ternary logic, three ternary max-root-functions are required for every concatenation. In general, for p-valued logic, p different logic functions are required for every concatenation.

For an illustration, let $g_{(n-1,1)}$, $g_{(n-1,2)}$, $g_{(n-1,3)}$ be three $(n-1)$-variable ternary functions. Now, an n-variable ternary function $f_{(n,1)}$ can be generated by appending 0 with every true vector of $g_{(n-1,1)}$, and appending 1 with every true vector of $g_{(n-1,2)}$ and similarly, by appending 2 with every true vector of $g_{(n-1,3)}$, and finally, by selecting those appended vectors as vectors of $f_{(n,1)}$ with the same value as in $g_{(n-1,1)}$, $g_{(n-1,2)}$ and $g_{(n-1,3)}$. Such a concatenation operation with $g_{(n-1,1)}$, $g_{(n-1,2)}$ and $g_{(n-1,3)}$ is denoted by $f_{(n,1)} = x^0 g_{(n-1,1)} \vee x^1 g_{(n-1,2)} \vee x^2 g_{(n-1,3)}$. In ternary logic, three $(n-1)$-variable ternary root-functions are required for performing concatenation. A set of such triple functions is called a *concatenable triplet*.

Definition 13: The concatenable triplet is formed by three distinct ternary latin-square functions $\{g_{(n-1,1)}, g_{(n-1,2)}, g_{(n-1,3)}\}$, where for every $(n-1)$-variable minterm x, $g_{(n-1,i)}(x) \cap g_{(n-1,j)}(x) = \varnothing$ for $\forall\ (i,j)$, $1 \leq (i,j) \leq 3$ and $i \neq j$.

The number of n-variable ternary functions that can be generated from each concatenable triplet is 3! = 6. For example, a concatenable triplet $\{g_{(n-1,1)}, g_{(n-1,2)}, g_{(n-1,3)}\}$ can generate six functions $f_{(n,1)}$, $f_{(n,2)}$, $f_{(n,3)}$, $f_{(n,4)}$, $f_{(n,5)}$, $f_{(n,6)}$ as given below:

$f_{(n,1)} = x^0 g_{(n-1,1)} \vee x^1 g_{(n-1,2)} \vee x^2 g_{(n-1,3)}$
$f_{(n,2)} = x^0 g_{(n-1,1)} \vee x^2 g_{(n-1,2)} \vee x^1 g_{(n-1,3)}$
$f_{(n,3)} = x^1 g_{(n-1,1)} \vee x^0 g_{(n-1,2)} \vee x^2 g_{(n-1,3)}$
$f_{(n,4)} = x^1 g_{(n-1,1)} \vee x^2 g_{(n-1,2)} \vee x^0 g_{(n-1,3)}$
$f_{(n,5)} = x^2 g_{(n-1,1)} \vee x^0 g_{(n-1,2)} \vee x^1 g_{(n-1,3)}$
$f_{(n,6)} = x^2 g_{(n-1,1)} \vee x^1 g_{(n-1,2)} \vee x^0 g_{(n-1,3)}$.

C. Concatenation Procedure for Ternary Logic

Procedure 1: Multi-valued-root(number of variables n)

1. Identify the set of all concatenable triplets for $(n-1)$-variable $(x_{n-1}, x_{n-2}, ..., x_1)$.

2. For each triplet $\{g_(n-1,1), g_(n-1,2), g_(n-1,3)\}$, do the following:

3. Consider a triplet $\{g_{(n-1,1)}, g_{(n-1,2)}, g_{(n-1,3)}\}$. Execute Step 4 for each possible combination of $\{p_i, p_j, p_k\}$ where $p_i, p_j, p_k \in \{0,1,2\}$ and $p_i \neq p_j \neq p_k$.

4. Append x_n with each of $\{g_{(n-1,1)}, g_{(n-1,2)}, g_{(n-1,3)}\}$ with values p_i, p_j, p_k, respectively and construct the function $f_{(n)}$. Let x_n appended with p_i be denoted as $x_n^{p_i}$. Hence, $f_n = x_n^{p_i} g_{(n-1,1)} \vee x_n^{p_j} g_{(n-1,2)} \vee x_n^{p_k} g_{(n-1,3)}$, which is an n-variable ternary root-function.

5. return.

Procedure 2: Generate-root(number of variables n)

1. for (variable = 2; variable $\leq n$; variable++)

Call Procedure 1 Multi-valued-root(variable).

2. end.

D. 2-Variable Ternary Root-Function

1) Generation of All 2-variable Ternary Latin-Square Max-root-Functions: Starting from the set of 1-variable ternary latin-square functions, we construct 2-variable ternary root-functions as follows. We know that there are six ternary 1-variable latin-square max-root-functions $g_{(1,1)}$, $g_{(1,2)}$, $g_{(1,3)}$, $g_{(1,4)}$, $g_{(1,5)}$, $g_{(1,6)}$ shown in Fig. 8. We find two concatenable triplets $\{g_{(1,1)}, g_{(1,2)}, g_{(1,3)}\}$ and $\{g_{(1,4)}, g_{(1,5)}, g_{(1,6)}\}$ where each of concatenable triplet can generate 3! = 6 different 2-variable ternary latin-square max-root-functions. Thus, a total of twelve 2-variable ternary latin-square max-root-functions can be constructed.

Example 7 : Let us choose $\{g_{(1,1)}, g_{(1,2)}, g_{(1,3)}\}$ as a concatenable triplet. Function $g_{(2,1)}$ is 2-variable ternary root-function generated from $\{g_{(1,1)}, g_{(1,2)}, g_{(1,3)}\}$, i.e. $g_{(2,1)} = x^0 g_{(1,1)} \vee x^1 g_{(1,2)} \vee x^2 g_{(1,3)}$ where 10, 01, 22 are 1-valued vectors, 20, 11, 02 are 2-valued vectors and 00, 21, 12 are 0-valued vectors. Fig. 9, Fig. 10, and Fig. 11 illustrate the method for generating vectors of $g_{(2,1)}$ from vectors of $g_{(1,1)}$, $g_{(1,2)}$ and $g_{(1,3)}$, respectively, by appending 0 with vectors of $g_{(1,1)}$, appending 1 with vectors of $g_{(1,2)}$ and appending 2 with vectors of $g_{(1,3)}$. Notice that $g_{(1,1)}$ is also a latin-square function. Similary, other 2-variable ternary latin-square max-root-functions can be generated from 1-variable latin-square max-root-functions $g_{(1,1)}$, $g_{(1,2)}$, $g_{(1,3)}$, $g_{(1,4)}$, $g_{(1,5)}$, $g_{(1,6)}$ as in Fig. 8 by the method of concatenation:

$g_{(2,2)} = x^0 g_{(1,3)} \vee x^1 g_{(1,1)} \vee x^2 g_{(1,2)}$
$g_{(2,3)} = x^0 g_{(1,2)} \vee x^1 g_{(1,3)} \vee x^2 g_{(1,1)}$
$g_{(2,4)} = x^0 g_{(1,1)} \vee x^1 g_{(1,3)} \vee x^2 g_{(1,2)}$
$g_{(2,5)} = x^0 g_{(1,2)} \vee x^1 g_{(1,1)} \vee x^2 g_{(1,3)}$
$g_{(2,6)} = x^0 g_{(1,3)} \vee x^1 g_{(1,2)} \vee x^2 g_{(1,1)}$
$g_{(2,7)} = x^0 g_{(1,4)} \vee x^1 g_{(1,5)} \vee x^2 g_{(1,6)}$
$g_{(2,8)} = x^0 g_{(1,6)} \vee x^1 g_{(1,4)} \vee x^2 g_{(1,5)}$
$g_{(2,9)} = x^0 g_{(1,5)} \vee x^1 g_{(1,6)} \vee x^2 g_{(1,4)}$
$g_{(2,10)} = x^0 g_{(1,4)} \vee x^1 g_{(1,6)} \vee x^2 g_{(1,5)}$
$g_{(2,11)} = x^0 g_{(1,5)} \vee x^1 g_{(1,4)} \vee x^2 g_{(1,6)}$
$g_{(2,12)} = x^0 g_{(1,6)} \vee x^1 g_{(1,5)} \vee x^2 g_{(1,4)}$.

These functions are shown in Fig. 12.

Vectors of g_1' before appending 0	Vectors of g_1' after appending 0	Vectors in g_1' with values	Vectors in g_1 with values
0	00	$g_1'(0) = 0$	$g_1(00) = 0$
1	10	$g_1'(1) = 1$	$g_1(10) = 1$
2	20	$g_1'(2) = 2$	$g_1(20) = 2$

Fig. 9. Vectors in g_1 with values produced from vectors in g_1'

Vectors of g_2' before appending 1	Vectors of g_2' after appending 1	Vectors in g_2' with values	Vectors in g_1 with values
0	01	$g_2'(0) = 1$	$g_1(01) = 1$
1	11	$g_2'(1) = 2$	$g_1(11) = 2$
2	21	$g_2'(2) = 0$	$g_1(21) = 0$

Fig. 10. Vectors in g_1 with values produced from vectors in g_2'

Vectors of g_3' before appending 2	Vectors of g_3' after appending 2	Vectors in g_3' with values	Vectors in g_1 with values
0	01	$g_3'(0) = 2$	$g_1(02) = 2$
1	11	$g_3'(1) = 0$	$g_1(12) = 0$
2	21	$g_3'(2) = 1$	$g_1(22) = 1$

Fig. 11. Vectors in g_1 with values produced from vectors in g_3'

	$y^0 = 0$	$y^1 = 1$	$y^2 = 2$
$x^0 = 0$	00	01	02
$x^1 = 1$	10	11	12
$x^2 = 2$	20	21	22

(a) Map-representation of 2-variable ternary function

0	1	2
1	2	0
2	0	1

(b) $g_{(2,1)}$

2	0	1
0	1	2
1	2	0

(c) $g_{(2,2)}$

1	2	0
2	0	1
0	1	2

(d) $g_{(2,3)}$

0	2	1
1	0	2
2	1	0

(e) $g_{(2,4)}$

1	0	2
2	1	0
0	2	1

(f) $g_{(2,5)}$

2	1	0
0	2	1
1	0	2

(g) $g_{(2,6)}$

0	1	2
2	0	1
1	2	0

(h) $g_{(2,7)}$

2	0	1
1	2	0
0	1	2

(i) $g_{(2,8)}$

1	2	0
0	1	2
2	0	1

(j) $g_{(2,9)}$

0	2	1
2	1	0
1	0	2

(k) $g_{(2,10)}$

1	0	2
0	2	1
2	1	0

(l) $g_{(2,11)}$

2	1	0
1	0	2
0	2	1

(m) $g_{(2,12)}$

Fig. 12. All 2-variable ternary latin-square max-root-functions

The total number of 2-variable ternary logic functions is 3^{3^2} = 19683. Among them, we could construct only twelve root-functions by the method of concatenation. These functions are latin-square functions as well [12]. Moreover, these twelve functions satisfy the properties of max-root-functions, where number of product terms is maximum, i.e. 6 = $(2.3^{n-1}, n = 2)$ [12].

E. 3-Variable Ternary Root-Function

From Fig. 12, notice that the number of 2-variable concatenable triplets is four, and these triplets are $\{g_{(2,1)}, g_{(2,2)}, g_{(2,3)}\}$, $\{g_{(2,4)}, g_{(2,5)}, g_{(2,6)}\}$, $\{g_{(2,7)}, g_{(2,8)}, g_{(2,9)}\}$, $\{g_{(2,10)}, g_{(2,11)}, g_{(2,12)}\}$. From each of this triplet, we obtain 3! = 6 different 3-variable ternary max-root-functions. Hence, the number of 3-variable ternary latin-square max-root-functions generated by concatenation is 24. The total number of 3-variable ternary logic functions is 3^{3^3}. Among them, we could identify only these 24 functions as root-functions. Again for each of them, the number of product terms are $2.3^{n-1}, n = 3$. Hence, all of these are ternary 3-variable max-root-functions. Fig. 13 shows the map-representation for 3-variable ternary functions. All 3-variable ternary latin-square max-root-functions $R_{(3,1)}, R_{(3,2)}, ..., R_{(3,24)}$ have been identified and shown in Appendix.

F. Number of Latin-square Max-root-Functions

In ternary logic, we have 6, 12, or 24 latin-square, max-root-functions for 1-variable, 2-variable, or for 3-variable, respectively. In ternary, the number of n-variable latin-square max-root-functions = 2 $\times$ number of $(n-1)$-variable latin-square max-root-functions.

G. Number of Product Terms in Max-Root-Functions

For each 1-variable or 2-variable ternary max-root-function, the number of product terms in its minimal sum-of-products expression is 3 and 6, respectively. All max-root-functions have the maximum number of product terms in their minimal sum-of-product expressions. For n-variable ternary max-root-functions, the number of product terms will be equal to 2.3^{n-1}. In general, for p-valued system, an n-variable max-root-function will have $(p-1).p^{n-1}$ number of product terms in its minimal sum-of-product expression.

H. Relation Among Root, Max-Root, and Latin-Square Functions

We have observed earlier that all max-root-functions constructed by the concatenation method are also latin-square functions. The question is: Whether there exists any other max-root-functions, which are not latin-square functions. For $n = 1$, we have seen that there are nine max-root-functions among which three ($g_{(1,7)}, g_{(1,8)}$ and $g_{(1,9)}$ in Fig. 8) are not latin-square functions. Therefore, in general, the set of latin-square functions is a subset of the set of max-root-functions. However, for $n = 2$ and 3, we could not identify any max-root-function that is not a latin-square function. Nevertheless, we believe such functions indeed exist. Also, for $n = 1$, we have identified nine root-functions, and all of them are max-root-functions. In binary logic, for $n = 1$ and 2, every root-function is a max-root-function. Note that in binary logic, for $n > 2$, there exist root-functions, which are not max-root-functions. Fig. 2 shows an example of a 4-variable root-function, which is not a max-root-function. Unfortunately, in ternary logic, we could not construct any such root-function for $n = 2$ or 3. We, however, believe that such functions do exist. This discussion leads to the following observation.

Observation: For any n, $S_L \subset S_M \subset S_R$ where S_L, S_M and S_R denote the set of all ternary latin-square functions, ternary max-root-functions, and ternary root-functions, respectively.

V. Conclusion

We have identified a few multiple-valued root-functions and studied some of their attributes. Some special root-functions are classified as being max-root, and a subset of the latter consists of as latin-square functions. We have described a concatenation-based procedure for constructing n-variable latin-square functions recursively from $(n-1)$-variable functions for multiple-valued logic. We have identified all 1-variable ternary max-root-functions, and among them, six are observed to be latin-square functions. We have also identified all ternary 2- and 3-variable latin-square functions by the method of concatenation. We noticed that such ternary latin-square functions exhibit certain regular patterns in their map-representations. However, the mechanism for identifying ternary root-functions that are not max-root-functions, is yet to be investigated. Also, exploring the attributes of other ternary non-max-root-functions requires further study.

References

[1] D. K. Das, D. Chowdhury, B. B. Bhattacharya, T. Sasao, "Inadmissible class of Boolean Functions under Stuck-at Faults," in *Proc., IEEE* 44^{th} *International Symposium on Multiple-Valued Logic* (ISMVL 2014, 19-21 May), vol. 1, pp. 237-242, 2014.

[2] M. E. R. Romero, E. M. Martins, and R. R. Santos, "Multiple-valued logic algebra for the synthesis of digital circuits," *In Proceedings, 39th International Symposium on Multiple-Valued Logic*, pp. 262-267, 2009.

[3] 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.

[4] G. Epstein, G. Frieder, and D. C. Rine, "The Development of Multiple-Valued Logic as Related to Computer Science," In D. C. Rine, editor, *Computer Science and Multiple-Valued Logic: Theory and Applications*, pages 81-101, North-Holland, Amsterdam, 1977.

[5] T. Raju Damarla, "Fault detection in multiple-valued logic circuits," *In Proceedings, Twentieth International Symposium on Multiple-Valued Logic*, pp. 69-74, 1990.

[6] Z. Kohavi, "Switching and Finite Automata Theory," *McGraw-Hill, Inc.*, 1970.

[7] D. K. Das, S. Chakraborty and B. B. Bhattacharya, "Boolean algebraic properties of fault behavior in logic circuits," *In Proc., Int. Workshop on Boolean Problems*, pp. 143-150, Sept., 2000.

[8] 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.

[9] S. Maitra and E. Pasalic, "A Maiorana-McFarland type construction for resilient Boolean functions on n-variables (n even) with nonlinearity $> 2^{n-1} - 2^{n/2} + 2^{n/2-2}$," *Discrete Applied Mathematics*, 154(2): 357-369 (2006).

[10] P. Sarkar and S. Maitra, "Construction of Nonlinear Resilient Boolean Functions using "Small" Affine Functions," *IEEE Transactions on Information Theory*, vol. 50, no. 9, pp. 2185-2193, 2004.

[11] Damarla, T.R., "Fault detection in multiple valued logic circuits," *In Proceedings, Twentieth International Symposium on Multiple-Valued Logic*, pp. 69-74, 1990.

[12] P. Tirumalai and J. T. Butler, "On the Realization of Multiple-valued Logic Functions Using CCD PLA's," *In Proc., IEEE International Symposium on Multiple-Valued Logic*, pp. 33-42, 1984.

[13] S. Chakraborty, D. K. Das, B. B. Bhattacharya, "Logical Redundancies in Irredundant Combinational Circuits," *Journal of Electronic Testing : Theory and Applications*,4(2):125-130, May 1993.

	$x_1^0 = 0$	$x_1^1 = 1$	$x_1^2 = 2$
$x_2^0 = 0$	000	100	200
$x_2^1 = 1$	010	110	210
$x_2^2 = 2$	020	120	220

(a) $x_3^0 = 0$

	$x_1^0 = 0$	$x_1^1 = 1$	$x_1^2 = 2$
$x_2^0 = 0$	001	101	201
$x_2^1 = 1$	011	111	211
$x_2^2 = 2$	021	121	221

(b) $x_3^1 = 1$

	$x_1^0 = 0$	$x_1^1 = 1$	$x_1^2 = 2$
$x_2^0 = 0$	002	102	202
$x_2^1 = 1$	012	112	212
$x_2^2 = 2$	022	122	222

(c) $x_3^2 = 2$

Fig. 13. Map-representation for 3-variable ternary function with variables (x_1, x_2, x_3).

Appendix

Map-representation of $R_{(3,1)} = x^0 g_{(2,1)} \vee x^1 g_{(2,2)} \vee x^3 g_{(2,3)}$

0	1	2	2	0	1	1	2	0
1	2	0	0	1	2	2	0	1
2	0	1	1	2	0	0	1	2

Map-representation of $R_{(3,2)} = x^0 g_{(2,1)} \vee x^1 g_{(2,3)} \vee x^3 g_{(2,2)}$

0	1	2	1	2	0	2	0	1
1	2	0	2	0	1	0	1	2
2	0	1	0	1	2	1	2	0

Map-representation of $R_{(3,3)} = x^0 g_{(2,2)} \vee x^1 g_{(2,1)} \vee x^3 g_{(2,3)}$

2	0	1	0	1	2	1	2	0
0	1	2	1	2	0	2	0	1
1	2	0	2	0	1	0	1	2

Map-representation of $R_{(3,4)} = x^0 g_{(2,2)} \vee x^1 g_{(2,3)} \vee x^3 g_{(2,1)}$

1	2	0	0	1	2	2	0	1
2	0	1	1	2	0	0	1	2
0	1	2	2	0	1	1	2	0

Map-representation of $R_{(3,5)} = x^0 g_{(2,3)} \vee x^1 g_{(2,1)} \vee x^3 g_{(2,2)}$

1	2	0	2	0	1	0	1	2
2	0	1	0	1	2	1	2	0
0	1	2	1	2	0	2	0	1

Map representation of $R_{(3,6)} = x^0 g_{(2,3)} \vee x^1 g_{(2,2)} \vee x^3 g_{(2,1)}$

2	0	1	1	2	0	0	1	2
0	1	2	2	0	1	1	2	0
1	2	0	0	1	2	2	0	1

Map-representation of $R_{(3,7)} = x^0 g_{(2,4)} \vee x^1 g_{(2,5)} \vee x^3 g_{(2,6)}$

0	2	1	1	0	2	2	1	0
1	0	2	2	1	0	0	2	1
2	1	0	0	2	1	1	0	2

Map-representation of $R_{(3,8)} = x^0 g_{(2,4)} \vee x^1 g_{(2,6)} \vee x^3 g_{(2,5)}$

0	2	1	2	1	0	1	0	2
1	0	2	0	2	1	2	1	0
2	1	0	1	0	2	0	2	1

Map-representation of $R_{(3,9)} = x^0 g_{(2,5)} \vee x^1 g_{(2,4)} \vee x^3 g_{(2,6)}$

2	1	0	0	2	1	1	0	2
0	2	1	1	0	2	2	1	0
1	0	2	2	1	0	0	2	1

Map-representation of $R_{(3,10)} = x^0 g_{(2,5)} \vee x^1 g_{(2,6)} \vee x^3 g_{(2,4)}$

1	0	2	0	2	1	2	1	0
2	1	0	1	0	2	0	2	1
0	2	1	2	1	0	1	0	2

Map-representation of $R_{(3,11)} = x^0 g_{(2,6)} \vee x^1 g_{(2,4)} \vee x^3 g_{(2,5)}$

1	0	2	2	1	0	0	2	1
2	1	0	0	2	1	1	0	2
0	2	1	1	0	2	2	1	0

Map-representation of $R_{(3,12)} = x^0 g_{(2,6)} \vee x^1 g_{(2,5)} \vee x^3 g_{(2,4)}$

2	1	0	1	0	2	0	2	1
0	2	1	2	1	0	1	0	2
1	0	2	0	2	1	2	1	0

Map-representation of $R_{(3,13)} = x^0 g_{(2,7)} \vee x^1 g_{(2,8)} \vee x^3 g_{(2,9)}$

0	2	1	1	0	2	2	1	0
1	0	2	2	1	0	0	2	1
2	1	0	0	2	1	1	0	2

Map-representation of $R_{(3,14)} = x^0 g_{(2,7)} \vee x^1 g_{(2,9)} \vee x^3 g_{(2,8)}$

0	2	1	2	1	0	1	0	2
1	0	2	0	2	1	2	1	0
2	1	0	1	0	2	0	2	1

Map-representation of $R_{(3,15)} = x^0 g_{(2,8)} \vee x^1 g_{(2,7)} \vee x^3 g_{(2,9)}$

2	1	0	0	2	1	1	0	2
0	2	1	1	0	2	2	1	0
1	0	2	2	1	0	0	2	1

Map-representation of $R_{(3,16)} = x^0 g_{(2,8)} \vee x^1 g_{(2,9)} \vee x^3 g_{(2,7)}$

1	0	2	0	2	1	2	1	0
2	1	0	1	0	2	0	2	1
0	2	1	2	1	0	1	0	2

Map-representation of $R_{(3,17)} = x^0 g_{(2,9)} \vee x^1 g_{(2,7)} \vee x^3 g_{(2,8)}$

1	0	2	2	1	0	0	2	1
2	1	0	0	2	1	1	0	2
0	2	1	1	0	2	2	1	0

Map-representation of $R_{(3,18)} = x^0 g_{(2,9)} \vee x^1 g_{(2,8)} \vee x^3 g_{(2,7)}$

2	1	0	1	0	2	0	2	1
0	2	1	2	1	0	1	0	2
1	0	2	0	2	1	2	1	0

Map-representation of $R_{(3,19)} = x^0 g_{(2,10)} \vee x^1 g_{(2,11)} \vee x^3 g_{(2,12)}$

0	2	1	1	0	2	2	1	0
2	1	0	0	2	1	1	0	2
1	0	2	2	1	0	0	2	1

Map-representation of $R_{(3,20)} = x^0 g_{(2,10)} \vee x^1 g_{(2,12)} \vee x^3 g_{(2,11)}$

0	2	1	2	1	0	1	0	2
2	1	0	1	0	2	0	2	1
1	0	2	0	2	1	2	1	0

Map representation of $R_{(3,21)} = x^0 g_{(2,11)} \vee x^1 g_{(2,10)} \vee x^3 g_{(2,12)}$

2	1	0	0	2	1	1	0	2
1	0	2	2	1	0	0	2	1
0	2	1	1	0	2	2	1	0

Map-representation of $R_{(3,22)} = x^0 g_{(2,10)} \vee x^1 g_{(2,12)} \vee x^3 g_{(2,10)}$

1	0	2	0	2	1	2	1	0
0	2	1	2	1	0	1	0	2
2	1	0	1	0	2	0	2	1

Map-representation of $R_{(3,23)} = x^0 g_{(2,12)} \vee x^1 g_{(2,10)} \vee x^3 g_{(2,11)}$

2	1	0	1	0	2	0	2	1
1	0	2	0	2	1	2	1	0
0	2	1	2	1	0	1	0	2

Map-representation of $R_{(3,24)} = x^0 g_{(2,12)} \vee x^1 g_{(2,11)} \vee x^3 g_{(2,10)}$

1	0	2	2	1	0	0	2	1
0	2	1	1	0	2	2	1	0
2	1	0	0	2	1	1	0	2