A Novel Three Value Logic for Computing Purposes

Ali Soltani and Saeed Mohammadi

***Abstract*—The aim of this article is to suggest a new three-valued logic named SADEGH logic instead of the binary logic for designing digital systems. The methodology employed in this logic involves using three operators 1, 0 and -1 for explaining the meanings of correctness, incorrectness, and undefined. By using some defined relations, systems could be designed which have the advantages of increasing addressing model, control, precision of sampling and decreasing calculating operations in addition to being compatible with the binary logic [1]. The results of this study suggest the possibility of applying the SADEGH logic to designing combinational and sequential circuits.**

***Index Terms*—Dominant operand, identity operand, undefined operand, three-valued logic, SADEGH three-valued logic.**

I. Introduction

Traditionally, logical calculi are bivalent. There are only two possible truth values: true and false. The law of the excluded middle is one of the foundations of the classical two valued logic: a proposition P is either true or false, there is no other choice. At the beginning of the year 1920, the first non-classic illogical calculation became possible from a third value. Later on, at the of the year 1930s, logical three values were introduced for the analysis of partial-recursive predicates. In these logics, three values true (T) undefined (U) and false (F) are presented. Among the studies done about the three-value logic we can refer to an article introducing not3, ext3, and3, or3, imp3, equ3, xor3, les3, gre3, leq3, geq3 operators [2]. In another article [3], the three-value logic is considered further and the single and double input operators are defined as the following.

TABLE I: Defined Operands

F	G	$\neg F$	$F \wedge G$	$F \vee G$	$F \leftarrow_K G$	$F \leftrightarrow_K G$	$F \leftrightarrow_C G$	$F \leftarrow_L G$	$F \leftrightarrow_L G$
$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\bot$	$\bot$	$\top$	$\bot$
$\top$	u	$\bot$	u	$\top$	$\top$	u	$\bot$	$\top$	u
$\bot$	$\top$	$\top$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	u	$\top$	$\bot$	u	u	u	$\bot$	u	u
u	$\top$	u	u	$\top$	u	u	$\bot$	u	u
u	$\bot$	u	$\bot$	u	$\top$	u	$\bot$	$\top$	u
u	u	u	u	u	u	u	$\top$	$\top$	$\top$

To design digital systems, an interpolation relation is required for development. In the articles cited before, it is not possible to develop systems based on the three-value logic.

Manuscript received November 14, 2012; revised December 23, 2012.

The authors are with Alghadir center of higher education Zanjan, Iran (e-mail: ali.s@roozbeh.ac.ir,saeedm@roozbeh.ac.ir).

II. Sadegh Three-value Logic

A. The Primitive Operators in This Logic Are Defined as the Following

1) Definition 1:

The double-input PG1 operator indicated by the symbol (+). In this operator, 1, 0 and -1 are defined as the dominated, identity and undefined operands respectively as presented in Table II.

TABLE II: PG1 Table

(+)	-1	0	1
-1	-1	-1	1
0	-1	0	1
1	1	1	1

2) Definition 2:

The double-input PG0 operator indicated by the symbol In this operator, 1, 0 and -1 are defined as the undefined, dominated and identity operands respectively as presented in Table III.

TABLE III: PG0 Table

(.)	-1	0	1
-1	-1	0	1
0	0	0	0
1	1	0	1

3) Definition 3:

The double-input PG-1 operator indicated by the symbol. (<>), In this operator, 1, 0 and -1 are defined as the identity, undefined and dominated operands respectively as presented in Table IV.

TABLE IV:PG-1 Table

(<>)	-1	0	1
-1	-1	-1	-1
0	-1	0	0
1	-1	0	1

TABLE V: $SG^{\wedge}$

X	$X^{\wedge}$
-1	0
0	0
1	1

TABLE VI: $SG^{\backslash}$

X	$X^{\backslash}$
-1	-1
0	0
1	0

TABLE VII: SG^{*}

X	X^{*}
-1	-1
0	1
1	1

Similarly, we defined the single-input operators as follows: in the Tables V, VI, VII, VIII, IX, X, XI, XII, XIII, column X is the input that can take the three values 1, 0 and -

1 and second column shows the results after the values of X have been affected by the single-input operators.

TABLE VIII: $SG^{>}$

X	$X^{>}$
-1	1
0	0
1	1

TABLE IX: $SG^{<}$

X	$X^{<}$
-1	-1
0	-1
1	1

TABLE X: $SG^{\sim}$

X	$X^{\sim}$
-1	0
0	-1
1	1

TABLE XI: $SG^{'}$

X	$X^{'}$
-1	1
0	0
1	-1

TABLE XII: SG^{-}

X	X^{-}
-1	-1
0	1
1	0

TABLE XIII: $SG^{\%}$

X	$X^{\%}$
-1	-1
0	0
1	-1

B. The basic rules of the SADEGH logic.

X is a variable that can take the three values 1, 0 and -1.

The relations of the PG1 operator:

$$X + -1 = X^{<}$$
$$X + 0 = X$$
$$X + 1 = 1$$

In the PG1 operator, the operand -1 was introduced as undefined. But since the combination of this operand with the other two operands and with itself results in $X^{<}$, -1 loses its undefined quality.

The relations of the PG0 operator:

$$X \,.\, -1 = X$$
$$X \,.\, 0 = 0$$
$$X.\, 1 = X^{>}$$

In the PG0 operator, the operand 1 was introduced as undefined. But since the combination of this operand with the other two operands and with itself results in $X^{>}$, 1 loses its undefined quality.

The relations of the PG-1 operator:

$$X <> -1 = -1$$
$$X <> 0 = X^{\backslash}$$
$$X <> 1 = X$$

In the PG-1 operator, the operand 0 was introduced as undefined. But since the combination of this operand with the other two operands and with itself results in $X^{\backslash}$, 0 loses its undefined quality.

TABLE XIV: NUMBERS IN BASE-3

Base-3			**Base-10**
-1	-1	-1	-13
-1	-1	0	-12
-1	-1	1	-11
-1	0	-1	-10
-1	0	0	-9
-1	0	1	-8
-1	1	-1	-7
-1	1	0	-6
-1	1	1	-5
0	-1	-1	-4
0	-1	0	-3
0	-1	1	-2
0	0	-1	-1
0	0	0	0
0	0	1	1
0	1	-1	2
0	1	0	3
0	1	1	4
1	-1	-1	5
1	-1	0	6
1	-1	1	7
1	0	-1	8
1	0	0	9
1	0	1	10
1	1	-1	11
1	1	0	12
1	1	1	13

III. NUMBER BASE CONVERSIONS

A. Conversion from Base-3 to Base-10:

Numbers 1, 0 and -1 are used in three-value systems and they refer to one, zero and minus one as mathematical quantities. To represent bigger or smaller numbers, we put the three numbers successively following some specified rules. In the three-value systems, each number can be written as the following in the three-value system [4]:

$$X=(a_n \cdot 3^n) + (a_{n-1} \cdot 3^{n-1}) + \ldots\ldots + (a_1 \cdot 3^1) + (a_0 \cdot 3^0)$$

Here, each of the factors $a_0, a_1, \ldots a_{n-1}, a_n$ can be 1, 0 or -1. In the SADEGH logic, each of the numbers 1, 0 and -1 is called a "TET" (Ternary Digit).

B. Conversion from Base-10 to Base-3:

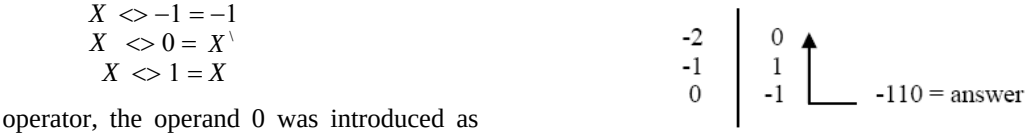
We explain this by point by an example:

$(-6)_{10} = (-110)_3$

-6 / 3 =	-2	+	0	$a_0 = 0$
-2 / 3 =	-1	+	$\frac{1}{3}$	$a_1 = 1$
-1 / 3 =	0	+	$\frac{-1}{3}$	$a_2 = -1$

Therefore, the answer is $(-6)_{10} = (a_2 a_1 a_0)_3 = (-110)_3$

The arithmetic process can be mainpulated more conveniently as follows:



First we divide -6 by 3. As the quotient we write a number the multiplication of which by the divisor produces a number that is different from the dividend by less than 1 or more than -1; i.e. we take -2 as the quotient. In the next

stage, -2 must be divided by 3. Again, we write a number as the quotient so that the product of its multiplication by the divisor differs from the dividend by less than 1 or more than -1, that is -1. We continue the process until the quotient becomes 0, 1 or -1. After reaching this stage, we write the last quotient and the remainders from the end to beginning to get the final result.

IV. Getting the Output Relation

To design a circuit or a block in digital systems, we need to produce the output relation according to the different inputs. We use a truth table to obtain such an output relation and then develop it in the form a logical diagram [5].

In base-2 digital systems, there are two general methods of obtaining the output relation known as the minterm and maxterm methods [6]. To clarify the way we obtain the output relation in SADEGH logic, we first offer an example of how the output relation is produced in the binary logic and then explain how the output relation is obtained in the SADEGH logic.

We first produce a relation for one of the two operand type for operand 1, for example we do as the following:

A	B	F	
0	0	0	
0	1	1	$\overline{A}$.B
1	0	1	A.$\overline{B}$
1	1	1	A.B

Sub-relations

Fig. 1. Example of truth table for binary digital

For each row, we obtain a sub-relation. Since 1 is the identity operand in the AND operator, we use this operator between the two operands A and B, and since in the AND operator only 1.1=1, we need to change each of the operand A and B to 1. To do so the NOT operator is applied. Similarly, we obtain the sub-relation for each of the outputs of 1 and then we put the sub-relation to gather in the general relation using OR due to the fact that 1 is the dominant operand in the OR operator. Thus, if one of the sub-relations becomes 1, the output of the general relation will be certainly 1. Otherwise, it will be equal to 0.

$$F=\overline{A}.B+A.\overline{B}+A.B$$

We follow the same procedure in the SADEGH logic. This procedure consists of the following three major outputs each of which can be used to obtain the output relation:

$$F_{\max} = f_1^{\wedge} + f_{-1}^{\backslash}$$

$$F_{mid} = f_1^{<} . f_0^{\%}$$

$$F_{\min} = f_0^{>} <> f_{-1}^{*}$$

A. the Method of Obtaining $F_{\max}$

$$F_{\max} = f_1^{\wedge} + f_{-1}^{\backslash}$$

We explain this method by an example: in the $F_{\max}$ relation, f_1 refers to the sub-relation for the output of 1. Fig. 3 presents a random-output truth table. For each of the rows whose output is 1, we write the sub-relation as presented in the figure.

A	B	F	
-1	-1	-1	A . B
-1	0	0	$A^{\sim}$ + B
-1	1	0	$A^{\sim}$ + B^{-}
0	-1	-1	$A^{\sim}$. B
0	0	0	A + B
0	1	1	A^{-} <> B
1	-1	1	A <> B'
1	0	0	A^{-} + B
1	1	1	A <> B

Operand (-1) sub-relations
Operand (0) sub-relations
Operand (1) sub-relations

Fig. 2. Random-output truth table

Since 1 is the identity operand in the PG-1 operator, we use this operator between the operands A and B and since in the PG-1 operator only 1<>1=1, we need to change each of the operands A and B to 1. To change 0 and -1 to 1, we use the single-input operators SG^{-} and $SG^{'}$ respectively. For example, in row 7 of the truth table above, B must be changed from -1 to 1 using $SG^{'}$ and in row 6 A must be changed from 0 to 1 using SG^{-}. To obtain f_1, we need to combine the sub-relations by using an operator in which 1 is the dominant operand, that is operator PG1. Thus, the f_1 relation will be as follow:

$$f_1 = (A^{-} <> B) + (A <> B^{'}) + (A <> B)$$

It is not sufficient to use only f_1 to obtain the general relation because f_1 is valid in output of 1 but not in the output of 0 and -1. Therefore, we need a different relation such as f_{-1} or f_0, and since we are applying the $f_{\max}$ relation, we also need to use f_{-1}.

In this relation, f_{-1} refer to the sub-relation for the output of -1. We write the sub-relation for each of the rows whose output is -1.

Since -1 is the identity operand in the PG0 operator, we use this operator between the operands A and B and since in the PG0 operator only -1. -1= -1, we need to change each of the operands A and B to -1. To change 1 and 0 to -1, we use the single-input operators $SG^{'}$ and $SG^{\sim}$ respectively. To obtain f_{-1}, we need to combine the sub-relations by using an operator in which -1 is the dominant operand, which is operator PG-1. Thus, the f_{-1} relation will be as follow:

$$f_{-1} = (\text{A} . \text{B}) <> (A^{\sim} . \text{B})$$

The general relation obtained will be as the following:

$$f_{\max} = ((A^{-} <> B) + (A <> B^{'}) + (A <> B))^{\wedge} + ((A.B) <> (A^{\sim}.B))^{\backslash}$$

According to this general relation, we can design a circuit that is capable of developing the truth table presented above. It is worth mentioning that we could also use the f_{mid} and $f_{\min}$ relations to obtain the general relation. We explain

each of them in the following paragraphs.

B. The Method of Obtaining f_{mid}

We first need to obtain f_1 as explained above.

$$f_1 = (A^- <> B) + (A <> B') + (A <> B)$$

In the f_{mid} relation, f_0 refers to the sub-relation for the output of 0. We obtain the f_0 for each of the rows whose output is 0.

Since 0 is the identity operand in the PG1 operator, we use this operator between the operands A and B and since in the PG0 operator only 0 + 0= 0, we need to change each of the operands A and B to 0. To change 1 and -1 to 0, we use the single-input operators SG^- and $SG^{\sim}$ respectively. To obtain f_0, we need to combine the sub-relations by using an operator in which 0 is the dominant operand, that is operator PG0. Thus, the f_0 relation will be as follow:

$$f_0 = (A^{\sim} + B).(A^{\sim} + B^-).(A + B).(A^- + B)$$

The general relation obtained will be as the following:

$$f_{mid} = ((A^- <> B) + (A <> B') + (A <> B))^{<} . ((A^{\sim} + B).(A^{\sim} + B^-).(A + B).(A^- + B))^{\%}$$

C. The Method of Obtaining $f_{\min}$

To obtain $f_{\min}$ we need the f_0 and f_{-1} relations whose method of obtaining were explained before.

$$f_0 = (A^{\sim} + B).(A^{\sim} + B^-).(A + B).(A^- + B)$$
$$f_{-1} = (A . B) <> (A^{\sim} . B)$$

Thus, the general relation is:

$$f_{\min} = ((A^{\sim} + B).(A^{\sim} + B^-).(A + B).(A^- + B))^{>} . ((A.B) <> (A^{\sim}.B))^{*}$$

D. An Example of Combinational Circuits

Decoder circuits: These circuits convert the data form base-3 to base-10.

A	B	D_{-4}	D_{-3}	D_{-2}	D_{-1}	D_0	D_1	D_2	D_3	D_4
-1	-1	1	0	0	0	0	0	0	0	0
-1	0	0	1	0	0	0	0	0	0	0
-1	1	0	0	1	0	0	0	0	0	0
0	-1	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0
1	-1	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	1	0
1	1	0	0	0	0	0	0	0	0	1

Fig. 3. The decoder circuit truth table

The relations obtained from the above table are as the following:

$$D_{-4} = (A' <> B')^{\wedge} \qquad D_{-3} = (A' <> B^-)^{\wedge}$$
$$D_{-2} = (A' <> B)^{\wedge} \qquad D_{-1} = (A^- <> B')^{\wedge}$$
$$D_0 = (A^- <> B^-)^{\wedge} \qquad D_1 = (A^- <> B)^{\wedge}$$
$$D_2 = (A <> B')^{\wedge} \qquad D_3 = (A <> B^-)^{\wedge}$$
$$D_4 = (A <> B)^{\wedge}$$

V. The Advantages of Developing Digitalize System on Base-3 Instead of Base-2

A. Enhancing Addressing Capability, Sampling Accuracy and Control

If we use 8 bits for addressing in the binary logic, we can only address 2^8 = 256 memory units following the formula M = 2^n in which n is the number of address lines and M refers to the number of the addressable units [5]. However, using the same number of address lines in the SADEGH logic, we can address 3^8 = 6561 memory units following the formula M = 3^n in which n is the number of the address lines and m refers to the number of the addressable units.

Similarly, due to the increase in the number of the signal sampling quantum's in SADEGH, sampling accuracy is enhanced accordingly.

As for the enhancement of control in SADEGH, it should be noted that three processes can be controlled using a tet, which in binary logic this is limited to two processes using a bit.

B. Representing Negative Numbers

In binary digital systems, to represent negative umbers we use an extra bit that is the result of using the sign bit or complementation. In the SADEGH logic; however, due to the coverage of all the three regions of the positive, zero, and negative numbers, there is no need for extra bit.

C. Compatibility with the Binary Digital Systems

Duo to the complete coverage of Boolean algebra [5] in the SADEGH logic, we will also have the right output for each of the input 0 and 1 in the binary systems.

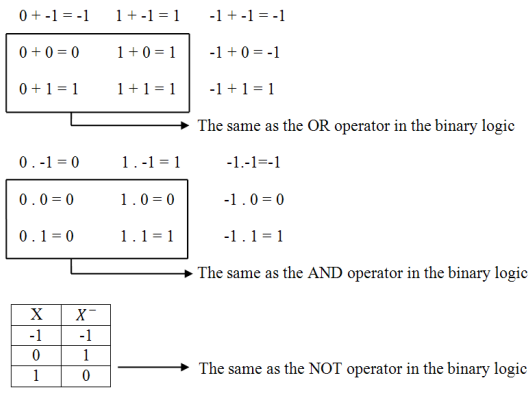


Fig. 4. Compatibility with the binary digital systems

D. Addition and Subtraction Numbers

The result of the addition or subtraction of two numbers on base-10 will be the same as the result of the addition or subtraction of the two numbers on base-3 in the SADEGH

logic. The following is an example:

```
  11     =>    1 1 -1
+ -8     =>  + -1 0 1
  3      =>    0 1 0
```

E. Subtraction of Numbers Using Complementation

For subtracting two numbers, we first make a complement of the subtrahend then add it to the minuend. For complementation, we change every 1 to -1 and every -1 to 1 and we leave the 0's unchanged. The following example illustrates this:

```
  11    =>    1 1 -1   same number    =>   1 1 -1
- 8     =>  - 1 0 -1   complementation =>  + -1 0 1
  3     =>    0 1 0    result          =>    0 1 0
```

In binary digital systems, to make a complement, we use a NOT operator plus an adder circuit [5]. The same action in SADEGH logic is done only by using one single-input SG' operator.

VI. Conclusions

The SADEGH three-value logic can be basis for designing digital systems due to the kind of relations among the operands, complete coverage of the binary base, and the application of all the three regions of numbers (positive, zero and negative). In this logic, it is also possible to obtain the right output for any binary input. This will make a SADEGH-based designed system compatible with binary digital systems. On the other hand, the smallest memory unit in the binary logic (bit) can take only the two values 0 or 1, while the smallest memory unit in the SADEGH logic (tet) can take the three values -1, 0, or 1 that makes it possible to enhance addressing, sampling accuracy, and control. Also, fewer gates will be required for adding and subtracting numbers in the complementation method.

Implications for further studies

Simplifying output relations using simplification formula.

Using decimal numbers in the SADEGH logic to enhance accuracy.

Designing the internal structure of the gates in the SADEGH logic.

Development of sequential circuits.

References

[1] M. M. Mano, *Digital Design*, Prentice Hall, pp. 27-30, 2002.

[2] J. Brandt, "A Three-Valued Logic for HOL Kananaskis," Department of Computer Science. Germany, pp. 3-6, 2008.

[3] S. Hölldobler and C. D. P. K. Ramli, "Logic Programs under Three-Valued Lukasiewicz Semantics," *International Center for Computational Logic*, TU Dresden, 01062 Dresden. Germany, pp. 2-4, 2007.

[4] R. D. Merrill, *Ternary logic in digital computers*, pp. 1-3, 1965.

[5] M. M. Mano, *Digital Design*, prentice hall, pp. 16-48, 2002.