2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)

Implementing the Unique Existential Quantifier in Digital Logic Design

1st Nesreen A. Hamad
Department of Computer Science. Faculty of Science and Information Technology
Al-Zaytoonah University of Jordan
Amman, Jordan
nesreen.hamad@zuj.edu.jo

2nd Maher A. Nabulsi
Department of Computer Science. Faculty of Science and Information Technology
Al-Zaytoonah University of Jordan
Amman, Jordan
nabusli@zuj.edu.jo

Abstract— **Nowadays, digital circuits are widely applied in different equipment such as computers, cell phones, digital watches, etc. As a result, new approaches to implement digital circuits are needed. Applying predicate logic is one way for achieving that. In particular, quantification (which is a commonly studied topic in predicate logic) can be used in the translation of a given logical statement which would assist in designing digital circuits. As a result, this paper proposes a new approach to implement the unique existential quantifier in digital circuits.**

Keywords—Predicate Logic, Quantifiers, Digital logic implementation, Decoder circuit, Data Bus, Multiplexer

I. Introduction

Digital Logic is considered the backbone of electronic systems such as robotics, computers, cell phones and other electronic applications. Indeed, logic plays a significant role in the design, programming, and use of computers. In digital logic, one of the main goals is to design digital circuits with less cost and less power usage [1],[2]. Applying discrete mathematics would assist in achieving this goal and would help in finding other alternatives in designing digital circuits. In particular, digital circuits may be represented with logical formulas by using propositional logic and its extension “predicate logic” [3]. Along with designing digital circuits, predicate logic is used for reasoning about programs, dynamic systems and used in building artificial intelligence reasoning systems [4].

The correspondence between digital logic circuits and propositional logic has been known for a long time. Such a relation can be summarized in Table I.

TABLE I. Digital logic circuits and propositional logic correspondence

Digital design	Propositional Logic
Logic gate (and, or , not)	Propositional connective ($\wedge$, $\vee$, $\neg$)
Circuit	Formula
Voltage level	Truth value

In propositional logic, the term “proposition” is defined as an assertion that is either true or false. On the other hand, a predicate is defined as an assertion that contains a finite number of variables and becomes a proposition when specific values are substituted for the variables from the domain of the predicate (set of all values that may be substituted in place of the variable) or when these variables are bound using quantifiers [5]. Quantification is a concept that specifies the quantity of cases in the domain that satisfies an open formula [6]. There are three forms of quantification: the universal quantifier ($\forall$), the existential quantifier ($\exists$) and the unique existential quantifier ($\exists!$). The meaning of each quantifier is as follows:

- $\forall$x P(x) means: " for all values of x, the predicate P(x) is true ".
- $\exists$x P(x) means: " for some values of x (at least one value), the predicate P(x) is true " .
- $\exists!$x P(x) means: " for one and only one value of x, the predicate P(x) is true ".

The implementation of the universal, existential and unique existential quantifiers in digital logic design based on their meanings is as follows:

Let U represents the universe of discourse of the predicate P (x), or in another word the domain of the predicate. For example, if U = { 1, 2, 3 }, then:

- $\forall$x P (x) is true if [P (1) $\wedge$ P (2) $\wedge$ P (3)] is true, this can be implemented with multiple-input AND-gate [7] as shown in Fig. 1.

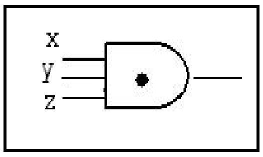

Fig. 1. Multiple-input AND-gate.

- $\exists$x P (x) is true if [P (1) $\vee$ P (2) $\vee$ P (3)] is true, this can be implemented with multiple-input OR-gate [7] as shown in Fig. 2.

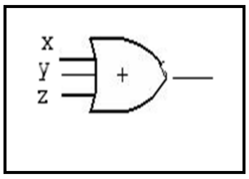

Fig. 2. Multiple-input OR-gate.

- $\exists!x\, P(x)$ is true if $[\,[P(1) \wedge \neg P(2) \wedge \neg P(3)]$
$\vee [\neg P(1) \wedge P(2) \wedge \neg P(3)]$
$\vee [\neg P(1) \wedge \neg P(2) \wedge P(3)]\,]$

is true, this can be implemented with two-level AND-OR gates as shown in Fig. 3.

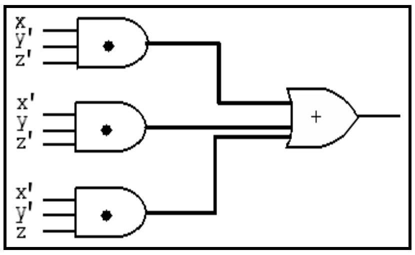

Fig. 3. Two-level AND-OR gates.

Where in the above figures: x, y and z represent P (1), P (2) and P (3) respectively, and x′, y′ and z′ represent ¬ P (1), ¬ P (2) and ¬ P (3) respectively.

An application for $\exists!x\, p(x)$ is presented in [8], where the relation between the $\exists!x\, p(x)$ and the X-OR was presented and had shown that the unique existential quantifier is equal to the X-OR only when the number of elements in the universe is two, which can be used to implement a one-selecting variant of a transition system. However, they are not equal when the number of elements in the universe is more than two.

In this paper, another alternative for implementing the unique existential quantifier with two and more variables in the predicate ($\exists!x\, \exists!y\, P(x, y)$, $\exists!x\, \exists!y\, \exists!z\, P(x, y, z)$, etc.) is presented which can be applicable and beneficial for other applications.

The rest of this paper is organized as follows: Section II presents our proposed method. The drawn conclusions and the planned future work are discussed in Section III.

II. METHODOLOGY

In this section, some definitions that are related to our work are presented. Then, our proposed implementations of $\exists!x\, \exists!y\, P(x, y)$ and $\exists!x\, \exists!y\, \exists!z\, P(x, y, z)$ are discussed.

Definition 1. A ($n * 2^n$) decoder is a circuit with *n* inputs, and 2^n outputs, where only one output is active at any given time [1].

Definition 2. A ($2^n * 1$) Multiplexer (MUX) is a circuit with 2^n inputs and a single output, where *n* is the number of selection lines ($S_0, S_1, S_2, \ldots, S_n$), that select which input to send to the output [1].

Definition 3. A data bus is a communication system that is used to transfer data between registers in a multiple-register configuration. It can be constructed with MUXs or with three-state buffers [1].

A. Implementing $\exists!x\, \exists!y\, P(x, y)$

In predicate logic, a domain (universe of discourse) must be chosen for each predicate. Thus, in this paper, the domain that is chosen is the set of 0 and 1, as these two numbers represent the binary digits of the computer.

Firstly, in order to understand how the proposition: $\exists!x\exists!y\, P(x, y)$ is implemented, we should find its equivalence when U = {0, 1} as follows:

Formula 1.

$$\exists!x\, \exists!y\, P(x, y) \Leftrightarrow$$
$$[P(0,0) \wedge \neg P(0,1) \wedge \neg P(1,0) \wedge \neg P(1,1)]$$
$$\vee\ [\neg P(0,0) \wedge P(0,1) \wedge \neg P(1,0) \wedge \neg P(1,1)]$$
$$\vee\ [\neg P(0,0) \wedge \neg P(0,1) \wedge P(1,0) \wedge \neg P(1,1)]$$
$$\vee\ [\neg P(0,0) \wedge \neg P(0,1) \wedge \neg P(1,0) \wedge P(1,1)]$$

The equivalence in Formula 1 means that for only one value of x and only one value of y, P(x, y) is true. When this is satisfied, only one bracket on the right-hand side of the equivalence will be true and all other brackets will be false. The OR (∨) between all the brackets will result in a true value for the proposition.

On the other hand, to implement $\exists!x\, \exists!y\, P(x, y)$ in the digital circuits , a 2 * 4 decoder is used, which is a circuit with 2 inputs (x , y) and 4 outputs (D_0, D_1, D_2, D_3), where only one output is active at any given time , which is similar to the meaning of $\exists!x\, \exists!y\, P(x, y)$ (only one value of x and one value of y satisfy the proposition). As a result, $\exists!x\, \exists!y\, P(x, y)$ is implemented by a 2 * 4 decoder. The truth table of a 2 * 4 decoder is presented in Table II.

TABLE II. THE TRUTH TABLE OF A 2 * 4 DECODER

X	Y	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

From Table II, $D_0 = x'y'$, $D_1 = x'y$, $D_2 = xy'$, $D_3 = xy$, where x' is the negation of x and y' is the negation of y. The circuit of a 2 * 4 decoder is presented in Fig. 4.

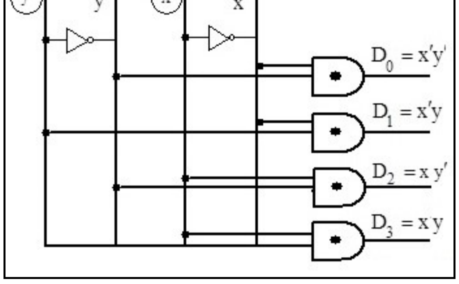

Fig. 4. 2* 4 decoder's circuit.

As a result, D_0, D_1, D_2, D_3 correspond to P(0,0), P(0,1), P(1,0), P(1,1) respectively, which means that $\exists! x\, \exists! y\, P(x, y)$ is implemented in a 2*4 decoder circuit.

Another implementation for $\exists! x\, \exists! y\, P(x, y)$ can be done using a data bus with 4 * 1 MUXs, which have two selection lines (S_1, S_0), and 4 registers (RA, RB, RC, RD) that are the inputs to the MUXs. Only one register must be selected to the bus lines, while the other registers must be inhibited from the bus lines depending on the values of the selection lines, this is the same as unique existential quantifier where only one value of x and only one value of y satisfy the proposition. The truth table for the suggested data bus is shown in Table III and its circuit with four registers each of four bits is presented in Fig. 5, where RA, RB, RC, RD correspond to P(0,0), P(0,1), P(1,0), P(1,1) respectively.

TABLE III. THE TRUTH TABLE OF A DATA BUS WITH 4 * 1 MUXS

S_1	S_0	RA	RB	RC	RD
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

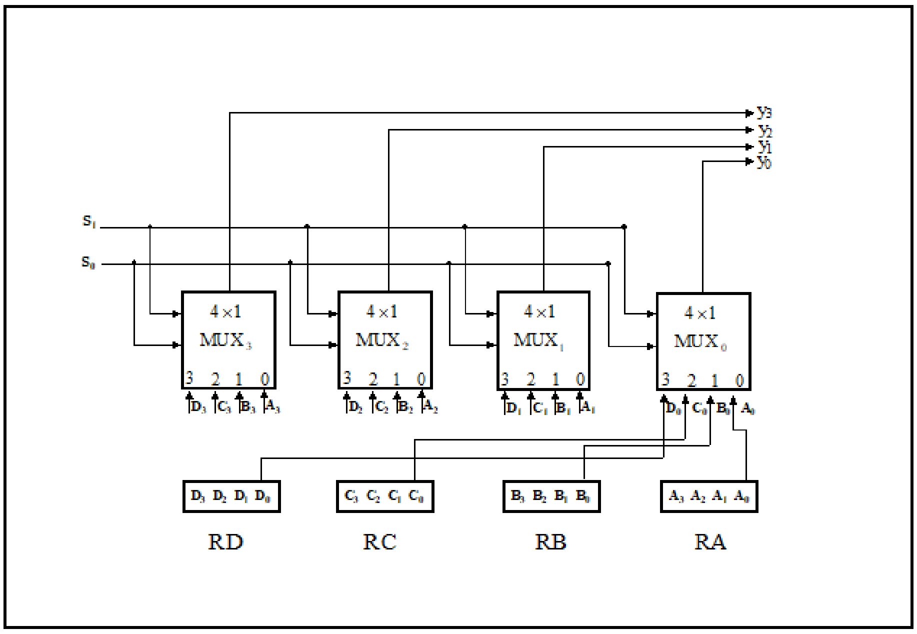

Fig. 5. Data bus with 4 * 1 MUXs.

B. *Implementing $\exists! x\, \exists! y \exists! z\, P(x, y, z)$*

Firstly, in order to understand how the proposition $\exists! x \exists! y \exists! z\, P(x, y)$ is implemented, we should find its equivalence when U = {0,1} as follows:

Formula 2.

$$\exists! x\, \exists! y\, \exists! z\, P(x, y, z) \Leftrightarrow$$

$$[P(0,0,0) \wedge \neg P(0,0,1) \wedge \neg P(0,1,0) \wedge \neg P(0,1,1) \wedge$$
$$\neg P(1,0,0) \wedge \neg P(1,0,1) \wedge \neg P(1,1,0) \wedge \neg P(1,1,1)]$$

$$\vee$$

$$[\neg P(0,0,0) \wedge P(0,0,1) \wedge \neg P(0,1,0) \wedge \neg P(0,1,1) \wedge$$
$$\neg P(1,0,0) \wedge \neg P(1,0,1) \wedge \neg P(1,1,0) \wedge \neg P(1,1,1)]$$

$$\vee$$

$$[\neg P(0,0,0) \wedge \neg P(0,0,1) \wedge P(0,1,0) \wedge \neg P(0,1,1) \wedge$$
$$\neg P(1,0,0) \wedge \neg P(1,0,1) \wedge \neg P(1,1,0) \wedge \neg P(1,1,1)]$$

$$\vee$$

$$[\neg P(0,0,0) \wedge \neg P(0,0,1) \wedge \neg P(0,1,0) \wedge P(0,1,1) \wedge$$
$$\neg P(1,0,0) \wedge \neg P(1,0,1) \wedge \neg P(1,1,0) \wedge \neg P(1,1,1)]$$

$$\vee$$

$$[\neg P(0,0,0) \wedge \neg P(0,0,1) \wedge \neg P(0,1,0) \wedge \neg P(0,1,1) \wedge P(1,0,0)$$
$$\wedge \neg P(1,0,1) \wedge \neg P(1,1,0) \wedge \neg P(1,1,1)]$$

$$\vee$$

$$[\neg P(0,0,0) \wedge \neg P(0,0,1) \wedge \neg P(0,1,0) \wedge \neg P(0,1,1) \wedge \neg P(1,0,0)$$
$$\wedge P(1,0,1) \wedge \neg P(1,1,0) \wedge \neg P(1,1,1)]$$

$$\vee$$

$$[\neg P(0,0,0) \wedge \neg P(0,0,1) \wedge \neg P(0,1,0) \wedge \neg P(0,1,1) \wedge \neg P(1,0,0)$$
$$\wedge \neg P(1,0,1) \wedge P(1,1,0) \wedge \neg P(1,1,1)]$$

$$\vee$$

$$[\neg P(0,0,0) \wedge \neg P(0,0,1) \wedge \neg P(0,1,0) \wedge \neg P(0,1,1) \wedge$$
$$\neg P(1,0,0) \wedge \neg P(1,0,1) \wedge \neg P(1,1,0) \wedge P(1,1,1)]$$

The equivalence in Formula 2 means that for only one value of x, only one value of y and only one value of z, P(x, y, z) is true. When this is satisfied, only one bracket will be true and all other brackets will be false. The (∨) between all the brackets will result in a true value for the proposition.

To implement ∃!x∃!y∃!z P (x, y, z) in the digital circuits, a 3 * 8 decoder is used for the implementation, which is a circuit with 3 inputs (x, y, z) and 8 outputs (D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7). Only one output is active at any given time which is similar to the meaning of ∃!x ∃!y∃!z P (x, z, y) (only one value of x, one value of y and one value of z satisfy the proposition). As a result, ∃!x ∃!y ∃!z P (x , y, z) is implemented by a 3 * 8 decoder. The truth table of a 3* 8 decoder is presented in Table IV, where D_0, D_1, …, D_7 correspond P(0,0,0), P(0,0 ,1), …, P(1,1,1) respectively. The circuit for a 3*8 decoder is not illustrated, because it is based on the same idea as in a 2*4 decoder.

TABLE IV. THE TRUTH TABLE OF A 3 * 8 DECODER

X	Y	Z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

It can also be implemented using a data bus with 8 * 1 MUXs, which have three selection lines (S_2 , S_1, S_0), and 8 registers (RA, RB, RC, RD, RE, RF, RG, RH). Only one register must be selected to the bus lines at any given time, this is the same as unique existential quantifier where only one value of x, only one value of y and only one value of z satisfy the proposition. The truth table for the suggested bus is presented in Table V, where RA, RB, …, RH correspond to P(0,0,0), P(0,0,1), …, P(1,1,1) respectively. The circuit for the data bus with 8*1 MUXs is not illustrated, because it is based on the same idea as in the data bus with 4*1 MUXs.

TABLE V. THE TRUTH TABLE OF A DATA BUS WITH 8 * 1 MUXS

S_2	S_1	S_0	**RA**	**RB**	**RC**	**RD**	**RE**	**RF**	**RG**	**RH**
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

III. CONCLUSION AND FUTURE WORKS

In this paper, an implementation of the unique existential quantifier in some digital circuits was presented. It was shown that ∃!x ∃!y P (x, y) is implemented by a 2 * 4 decoder and in a data bus with 4 * 1 MUXs. Furthermore, ∃!x ∃!y ∃!z P (x, y, z) is implemented by a 3 * 8 decoder and in a data bus with 8 * 1 MUXs. As a result, we can conclude that the unique existential quantifier with *n* variables can be implemented by a ($n * 2^n$) decoder and in a data bus with ($2^n * 1$) MUXs. In future, we will investigate other new applications for the quantifiers.

REFERENCES

[1] M. Mano, C. Kime and T. Martin, Logic and Computer Design Fundamentals, Global Edition, 5th ed. NOIDA: Pearson Education Limited, 2016.

[2] J. Gibson, Electronic logic circuits, 3rd ed. Abingdon, Oxon: Routledge, 2011.

[3] T. Jenkyns and B. Stephenson, Fundamentals of Discrete Math for Computer Science: A Problem-Solving Primer, 2nd ed. Cham: Springer International Publishing, 2018.

[4] C. Hall and J. O'Donnell, Discrete mathematics using a computer, 2nd ed. London: Springer-verlag, 2013.

[5] M. Nabulsi and N. Hamad, “Proofs of Implications Involving Quantifiers Distribution Over Logical Operators”, Journal of Theoretical and Applied Information Technology, vol. 95, no. 12, pp. 2824 -2829, 2017.

[6] K. H. Rosen, Discrete mathematics and its applications, 7th ed. New York: McGraw-Hill, 2012

[7] M. Nabulsi, “Some Propositions, Quantified Assertions and their uses in Computer Hardware Design”, An-Najah University Journal for Research - A, vol. 14, pp. 157 – 168, 2000.

[8] M. Nabulsi and A. Abdalla, " The relationship between Exclusive- or and the Unique Existential Quantifier ", Journal of Computer Science, vol. 4, no. 9, pp. 741 – 743, 2008.