Journal of Kerbala University , Vol. 6 No.4 Scientific. 2008

Design and Implementation for Intelligent Logic Simplification Program

Hawrra H.Abbas Al-Rubiae
Computer Engineering.
The University of Kerbala

Riyadh M. Naife
Electronic and communication Engineering.
The University of Kerbala

Abstract :
Before built any complex logic system we must find all the possible way to establish this system with minimum logic components so, in this paper we design and implement intelligent program (using DELPHE programming language), this program work on simplifying and drawing any logic equation according to knowledge base built from Boolean algebra rules**.**

1-Introduction
Artificial intelligence leads to constructing systems to assist people in difficult tasks in many branches of industry, construction, engineering design, agriculture, environment protection, commercial services such as: economy, banking and financing, information technology, medicine, education, scientific research, space exploration, military technology, and others.[1 Miroslav]
Designing an intelligent agent capable of reasoning, planning and acting in a changing environment is one of the important research areas in the field of AI. Such an agent should have knowledge about the domain in which it is intended to act and its capabilities and goals which in this case logic design system. Logic design is determining how to interconnection basic logic building blocks to perform a specific function. An example of logic design is determining the interconnection of logic gates and flip-flops required to perform binary addition.[7 Roth c] It is important to distinguish between combinational logic expression and logic gate networks. A combinational logic expression is a mathematical formula which is to be interpreted using the lows of Boolean algebra: given the expression (a+b), for example, its truth value for any given values of a and b can be computed.
The goal of logic design is to find a network of logic gates which together compute the designed combinational logic function.[8 Wakerly J.] In this paper we are interested in agents which simplified any logic equation and give the optimal result to represent these circuits at the gate –level of abstraction. So this system contain the following two categories

- knowledge representation to accept and express incoming information (logic function) , understand it, confront with already possessed knowledge, and use it efficiently
- reasoning to draw correct and necessary conclusions from acquired data, and make decisions about further actions.[2 John S]

2-Binary Logic
Binary logic deals with variables that take two discrete values and operations that assume logical meaning. The two values the variable take may be called by different names (true and false, yes and no, etc), Binary logic consists of binary variables and logical operations. The variable having two and only two distinct possible values (1 and 0).There are three basic logical operations (AND, OR, and NOT). [4 MANO]

3-Digital Logic Gate
Gate originally got their names from their function of allowing or retarding "gating" the follow of digital information. In general, a gate has one or inputs and produces an output that is a function of the current input value(s). The gates are blocks of hardware that produce the equivalent of logic 1 or logic 0 output signal if input logic requirements are satisfied.Since Boolean functions are

expressed in terms of AND , OR , and NOT operations, it is easier to implement a Boolean function with these types of gates. [8 Wakerly J.]

4-Boolean Algebra

Boolean logic is a complete system for logical operations. It was named after George Boole, who first defined an algebraic system of logic in the mid 19th century. Boolean logic has many applications in electronics, computer hardware and software, and is the base of digital electronics. In 1938, Claude Shannon showed how electric circuits with relays were a model for Boolean logic. This fact soon proved enormously consequential with the emergence of the electronic computer.[3 Wikipedia] A Boolean algebra is a set *A*, supplied with two binary operations $\wedge$(called AND), $\vee$ (called OR), a unary operation $\neg$(called NOT) and two distinct elements 0 (called zero) and 1 (called one), such that, for all elements *a*, *b* and *c* of set *A*, the following axioms hold:

$a \vee (b \vee c) = (a \vee b) \vee c$	$a \wedge (b \wedge c) = (a \wedge b) \wedge c$	associatively
$a \vee b = b \vee a$	$a \wedge b = b \wedge a$	commutatively
$a \vee (a \wedge b) = a$	$a \wedge (a \vee b) = a$	absorption
$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	distributives
$a \vee \neg a = 1$	$a \wedge \neg a = 0$	complements

The first three pairs of axioms above: associatively, commutative and absorption, mean that (*A*, $\wedge$, $\vee$) is a lattice. If *A* is a lattice and one of the above distributive laws holds, then the second distributive law can be proven. Thus, a Boolean algebra can also be equivalently defined as a distributive lattice. From these axioms, one can show that the smallest element 0, the largest element 1, and the complement ¬*a* of any element *a* are uniquely determined. For all *a* and *b* in *A*, the following identities also follow:

$a \vee a = a$	$a \wedge a = a$	idempotence
$a \vee 0 = a$	$a \wedge 1 = a$	bounded ness
$a \vee 1 = 1$	$a \wedge 0 = 0$	
$\neg 0 = 1$	$\neg 1 = 0$	0 and 1 are complements
$\neg(a \vee b) = \neg a \wedge \neg b$	$\neg(a \wedge b) = \neg a \vee \neg b$	De Morgan's laws
$\neg\neg a = a$		involution

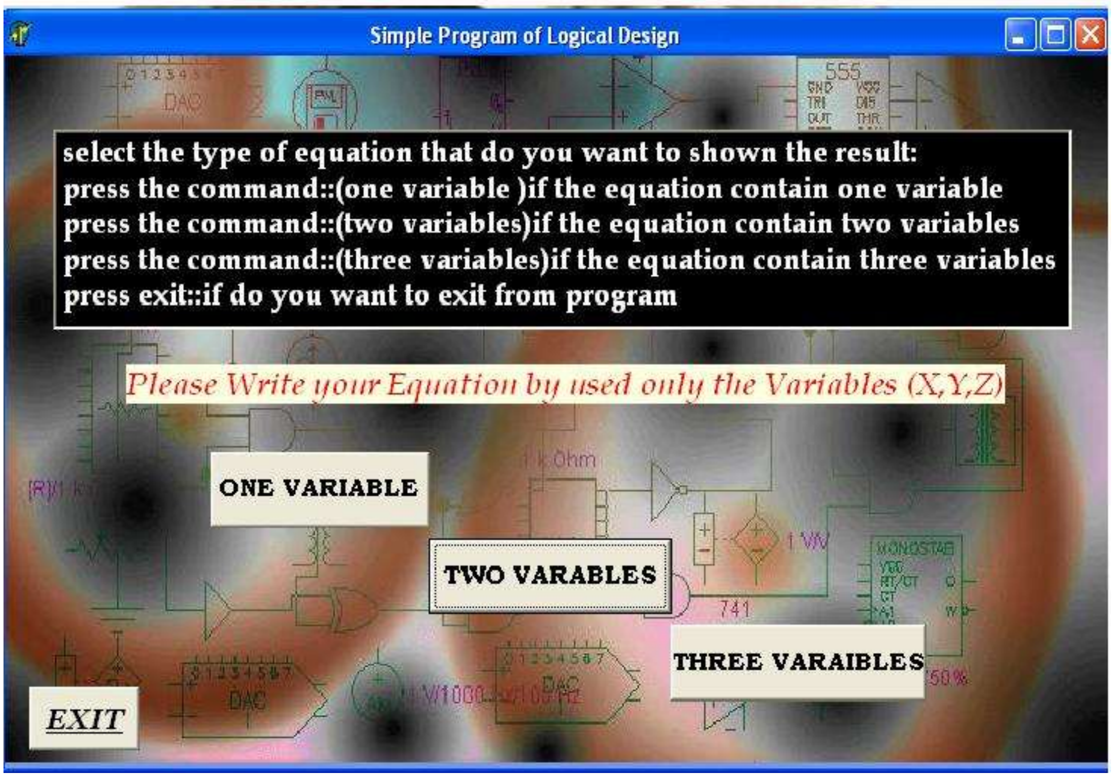
The two-element Boolean algebra is also important in the general theory of Boolean algebras, because an equation involving several variables is generally true in all Boolean algebras if and only if it is true in the two-element Boolean algebra (which can always be checked by a trivial brute force algorithm). This can for example be used to show that the following laws (*Consensus theorems*) are generally valid in all Boolean algebras: [4 MANO][5 L.Floyd][6 Stephen]

(*a* □□*b*) □□(¬*a* □□*c*) □□(*b* □□*c*) □□(*a* □□*b*) □□(¬*a* □□*c*)

(*a* □□*b*) □□(¬*a* □□*c*) □□(*b* □□*c*) □□(*a* □□*b*) □□(¬*a* □□*c*)

5- Implementation and Result

Figure (1) the program main form which contain some note to the program user and let the user select the type of equations that he will simplified it.

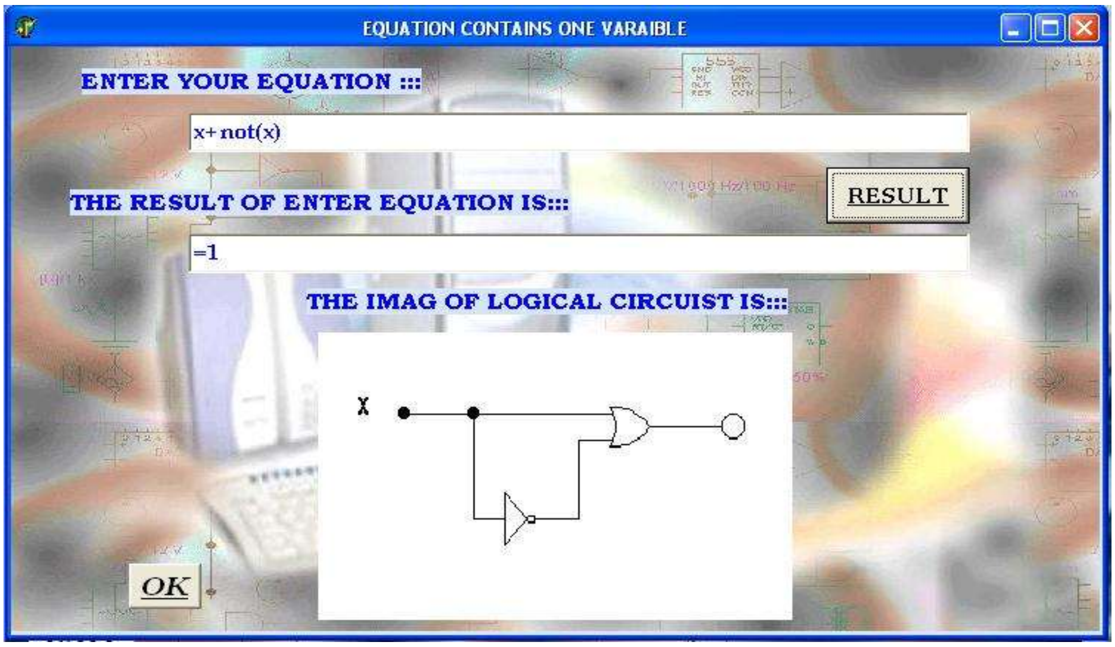


select the type of equation that do you want to shown the result:
press the command::(one variable)if the equation contain one variable
press the command::(two variables)if the equation contain two variables
press the command::(three variables)if the equation contain three variables
press exit::if do you want to exit from program

Please Write your Equation by used only the Variables (X,Y,Z)

Figure (1) the main form

Figure (2) shows the result of simplification for logic equation with one variable only (with logical design for the result circuit), while figure (3) shows the result of simplification for logic equation with two variable also we can used this program to simplify logic equations with thee variable in the same way.



ENTER YOUR EQUATION :::

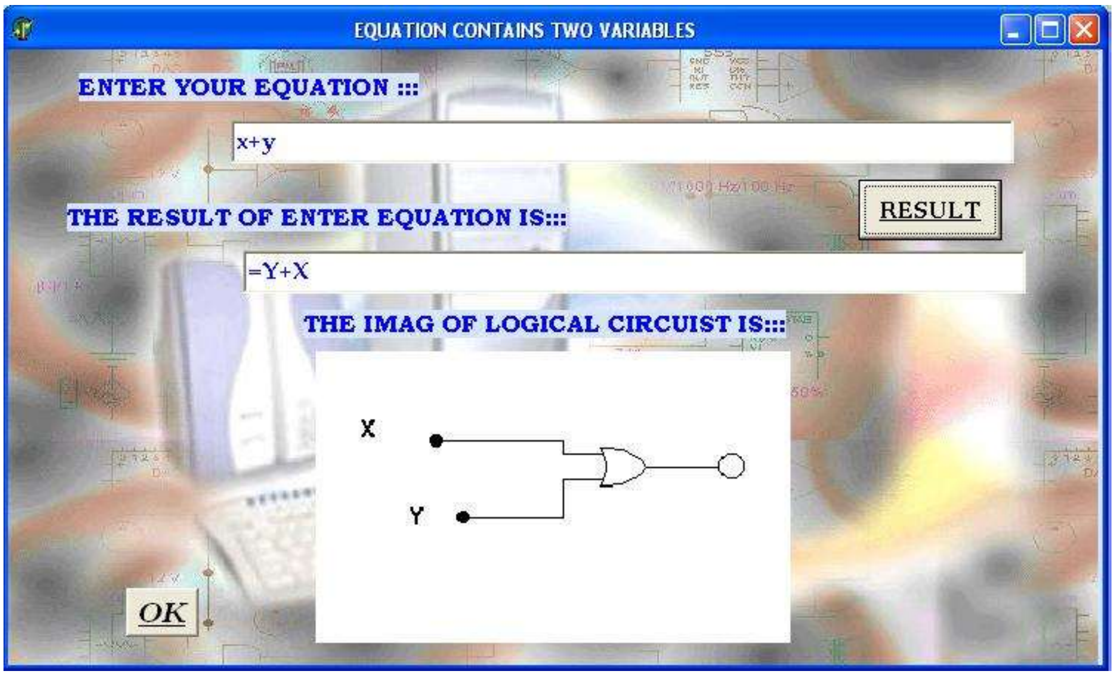
x+not(x)

THE RESULT OF ENTER EQUATION IS:::

=1

THE IMAG OF LOGICAL CIRCUIST IS:::

Figure (2) one variable form


EQUATION CONTAINS TWO VARIABLES

ENTER YOUR EQUATION :::

x+y

THE RESULT OF ENTER EQUATION IS:::

=Y+X

THE IMAG OF LOGICAL CIRCUIST IS:::

Figure (3) two variable form

Conclusion

To built any practical logic or electronic circuit, we must first design and implement it theoretically to get it with minimum number of hardware components .This program tested with some students (in logic circuit lab.) and help them to simplify and understand any logic equation with maximum three variables and give them the logical drawing for the implemented circuit with minimum logical components.

Future work

To modify this program:-

- Built knowledge base used to design the electric circuits
- Let the program modified more complex logical equations (with four and five variables for example).
- Use Karnaugh Map to build the program knowledge base instead of the Boolean algebra.
- Use minimizing algorithms such as the genetic algorithm to design the logic circuits and draw the minimized logic circuit.

References

[1] Miroslav Beličák**,** "Open Design Architecture and Artificial Intelligence Agent application in Information Systems Practice", Department of Computer and Informatics, Technical University of Košice Letná 9, 051 20 Košice, Slovak Republic,miroslav.belicak@tuke.sk

[2] John S Gero," RESEARCH IN DESIGN COMPUTING: AN ARTIFICIAL INTELLIGENCE FRAMEWORK"**,** Key Centre of Design Computing Department of Architectural and Design Science University of Sydney NSW 2006 Australia email: john@arch.usyd.edu.au

[3]"http://en.wikipedia.org/wiki/Boolean_algebra"Wikipedia®is a registered trademark of the Wikimedia Foundation, Inc., a US-registered.

[4] MANO, "Digital Design", Prentice/Hall International, book, 1986

[5]L.Floyd "Digital Fundamental", seven editions, Prentice Hall, book, 2000.

[6] Stephen Brown and Zvonko Varnish," Fundamental of Digital Logic", Amazan.com, Book, 2001.

[7] Roth C,JR.,"Fundamental of Logic Design',PWS Publishing Company,1995.

[8] Wakerly J.,"Digital Design:Principle &Practices",Prentice Hall of India ,2005.