An algorithm to implement a boolean function using only NAND's or only NOR's.*

Katherine Fletcher

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 2.0†

Abstract

An algorithm to implement a boolean function as a gate network using only NAND's or only NOR's is presented. Any boolean function can be implemented straightforwardly using AND's, OR's, and NOT gates. Using DeMorgan's Law in different forms gates in the network can be successively converted to use only NAND's or only NOR's.

NAND's and NOR's are the most common basic logic circuit element in use because they are simpler to build than AND and OR gates, and because each is **logically complete** . Many logical functions are expressed using AND's, OR's, and Inverters (NOT), however, because an implementing circuit can be constructed straightforwardly from the truth table expression of a logical function and because **Karnaugh Map's** can be used to minimize AND, OR, INVERTER networks.

This document is adapted, with permission, from algorithms and examples given in *Dr. Jump's Elec326 course notes.* [1]

Below a simple algorithm is given for converting a network with AND gates, OR gates and INVERTERS to one with NAND gates or NOR gates exclusively. First the boolean function is represented using AND's, OR's, and NOT gates. Then, using DeMorgan's Law in various forms, the AND, OR, INVERTER network is converted step-by-step to use only NAND gates or only NOR gates.

DeMorgan's Law using Boolean Algebra

OR to NAND	AND to NOR
$a \vee b \equiv \neg(\neg a \wedge \neg b)$	$a \wedge b \equiv \neg(\neg a \vee \neg b)$

Table 1

*Version 1.6: Sep 4, 2009 10:51 am +0000
†http://creativecommons.org/licenses/by/2.0/

DeMorgan's Laws Illustrated Using Logic Gates

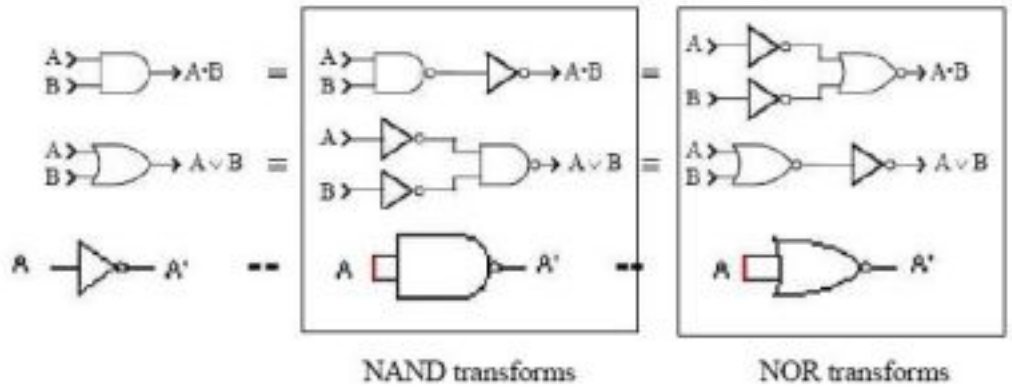

Figure 1: The figure is adapted with permission from *Dr. Robert Jump's Elec326 lecture notes* [1]. The first two rows of the figure above illustrate DeMorgan's Law using gates. The third row illustrates how to eliminate any inverters with either NAND or NOR gates.

Conversion Algorithm

1. Draw AND, OR, INVERTER implementation. First draw out an implementation of the boolean function using AND gates, OR gates and INVERTERS. Any implementation that uses only those three gate types will work. One way to implement a boolean function using AND's, OR's and INVERTERS is to build the **Disjunctive Normal Form** of the boolean function from the truth table that describes the function. Disjunctive Normal Form, is also called **Sum of Products** form. Propositional Logic: Normal Forms [1] gives a succinct treatment of normal forms and of how to go from a truth table to Disjunctive Normal Form.
2. Apply DeMorgan's Law. Apply DeMorgan's Law to the circuit by using the equivalences in the first two rows of the Figure above. To create a NAND only circuit, use the transforms in the left box, and for a NOR only circuit use the transforms in the right-hand box.
3. Remove redundant inverters: Any time that two inverters are in series (an inverted output goes directly in to an inverted input), remove both of them, since they cancel each other out.
4. Replace remaining inverters. Replace any remaining inverters with the equivalent NAND or NOR implementation (the third row of the Figure).

Example 1

[1] "Propositional Logic: normal forms" <http://cnx.org/content/m12075/latest/>

Conversion Example

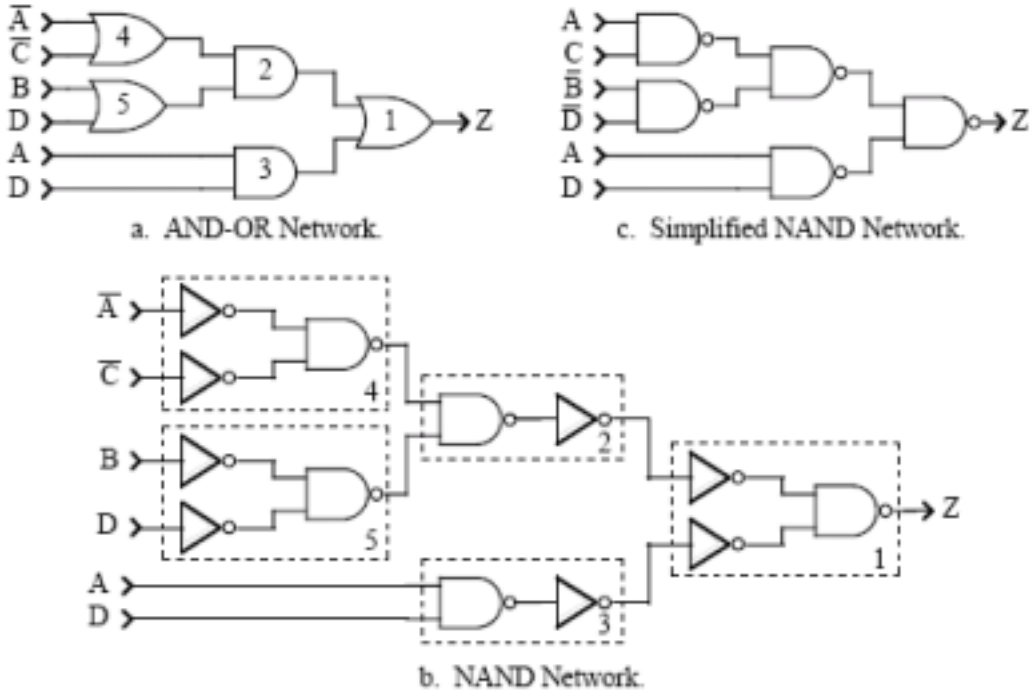

Figure 2: Example conversion to a NAND only network.

Note that in step c. the final elimination of inverters isn't quite done since B and D are inverted into one of the NAND's.

Glossary

Definition 1: Karnaugh Map
A visual map of the truth table of a boolean expression and an algorithm for removing redundant elements to realize a minimized boolean expression.

Definition 2: logically complete
A set of circuit gates or logical elements is logically complete if any boolean function representable by a truth table can be realized using only gates or elements from that set.

Example
AND, OR, and NOT is a logically complete set. NAND is logically complete. NOR is logically complete.

References

[1] Robert Jump. *NAND/NOR Networks.* Unpublished course notes from Elec 326, Rice University, Department of Electrical and Computer Engineering, Houston TX 77251, 2004.