Notes on Logic Synthesis and NAND/NOR Networks

1 Synthesis Using AND, OR, and NOT Gates

1.1 Introduction

The process whereby we begin with a description of the desired functional behaviour and then generate a cirucit that realizes theis behavious is called *synthesis*. Logic circuit synthesis involves designing a circuit that implements a given Boolean function using fundamental logic gates: AND, OR, and NOT. We can express the required behaviour of the function using truth tables.

1.2 Sum-of-Products and Product-of-Sums

If a function f is specified in the for of a truth table, then an expression that realizes f can be obtained by considering either the rows in the table for which $f = 1$ or rows for which $f = 0$.

Minterms - For a function of n variables, a product term in which each of the n variables appears once is called Minterm.

Sum-of-Products form - A function f can be represented by an expression that is a sum of minterms, where each minterm is ANDed with the value of f for the corresponding valuation of input variables. This form of representation of function is called Sum-of-Products form. If each product term is a minterm, then the expression is called *Canonical sum-of-products*. After deriving the Canonical form of function we can manipulate the expression whose corresponding circuit has the lowest cost.

Cost - A good indication of cost of a logical circuit is total number of gates plus the total number of inputs to all gates in the circuit.

Maxterm - The complements the minterms are called maxterms.

Product-of-Sums - The expression which represents a function f in the form of product of terms is in Product-of-Sums form. If the terms are maxterms, then it is called *Canonical product-of-sums* form.

1.3 Logic Minimization

Logic expressions obtained directly from truth tables may not always be optimal in terms of gate count or complexity. Boolean algebra and Karnaugh maps are commonly used to simplify expressions and reduce circuit complexity.

2 NAND and NOR Logic Networks

2.1 Introduction

NAND and NOR gates are obtained by complementing the output of AND and OR gates respectively. NAND and NOR gates are widely used in digital logic design due to their simplicity in implementation and their universality—they can be used to construct any Boolean function.

2.2 NAND as a Universal Gate

A NAND gate performs the operation:

$$\text{NAND}(x_1, x_2) = \overline{x_1 x_2}$$

By DeMorgan's theorem:

$$\overline{x_1 x_2} = \overline{x_1} + \overline{x_2}$$

This shows that a NAND gate of variables x_1 and x_2 is equivalent to first complementing each of the variables and then ORing them.

2.3 NOR as a Universal Gate

A NOR gate performs:

$$\text{NOR}(x_1, x_2) = \overline{x_1 + x_2}$$

By DeMorgan's theorem:

$$\overline{x_1 + x_2} = \overline{x_1 x_2}$$

This shows that a NOR gate of variables x_1 and x_2 is equivalent to ANDing both the inputs and then complementing them.

2.4 Converting Logic Networks

Expressing a function in terms of either sum-of-products or product-of-sums format leads to a logic circuit that have either AND-OR or an OR-AND structure. Any such network can be implemented using only NAND gates or NOR gates respectively. This can be done by replacing each connection between AND gate and an OR gate by a connection that includes two inversions of the signal,

one inversion at the output of the AND gate and the other at the input of OR gate. It would have no effect on out put as complement of complement of the given signal is the signal itself. As we are inverting the output of the AND gate, it becomes a NAND gate and we feeding the inverted inputs to the OR gate, which makes it a NAND gate. A similar process is following for NOR gate. This transformation is useful for hardware optimization since NAND and NOR gates are easier to fabricate.