Logic and Computer Design Fundamentals

Chapter 2 – Combinational Logic Circuits

Part 3 – Additional Gates and Circuits

Charles Kime & Thomas Kaminski

Overview

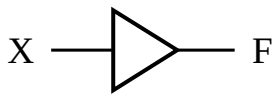
- **Part 1 – Gate Circuits and Boolean Equations**
  - **Binary Logic and Gates**
  - **Boolean Algebra**
  - **Standard Forms**
- **Part 2 – Circuit Optimization**
  - **Two-Level Optimization**
  - **Map Manipulation**
  - **Multi-Level Circuit Optimization**
- **Part 3 – Additional Gates and Circuits**
  - **Other Gate Types**
  - **Exclusive-OR Operator and Gates**
  - **High-Impedance Outputs**

Other Gate Types

- **Why?**
  - **Implementation feasibility and low cost**
  - **Power in implementing Boolean functions**
  - **Convenient conceptual representation**
- **Gate classifications**
  - **Primitive gate - a gate that can be described using a single primitive operation type (AND or OR) plus an optional inversion(s).**
  - **Complex gate - a gate that requires more than one primitive operation type for its description**
- **Primitive gates will be covered first**

Buffer

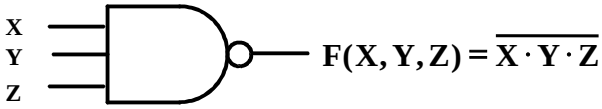
- **A buffer is a gate with the function F = X:**



- **In terms of Boolean function, a buffer is the same as a connection!**
- **So why use it?**
  - **A buffer is an electronic amplifier used to improve circuit voltage levels and increase the speed of circuit operation.**

NAND Gate

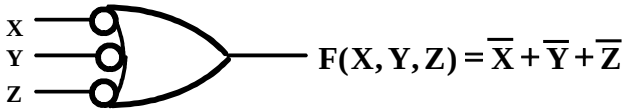
- **The basic NAND gate has the following symbol, illustrated for three inputs:**
  - **AND-Invert (NAND)**



- **NAND represents NOT AND, i. e., the AND function with a NOT applied. The symbol shown is an AND-Invert. The small circle ("bubble") represents the invert function.**

NAND Gates (continued)

- **Applying DeMorgan's Law gives Invert-OR (NAND)**



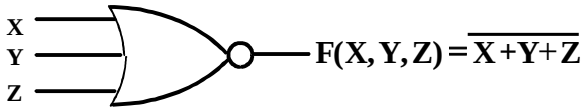
- **This NAND symbol is called Invert-OR, since inputs are inverted and then ORed together.**
- **AND-Invert and Invert-OR both represent the NAND gate. Having both makes visualization of circuit function easier.**
- **A NAND gate with one input degenerates to an inverter.**

NAND Gates (continued)

- **The NAND gate is the natural implementation for the simplest and fastest electronic circuits**
- ***Universal gate* - a gate type that can implement any Boolean function.**
- **The NAND gate is a universal gate as shown in Figure 2-30 of the text.**
- **NAND usually does not have a operation symbol defined since**
  - **the NAND operation is not associative, and**
  - **we have difficulty dealing with non-associative mathematics!**

NOR Gate

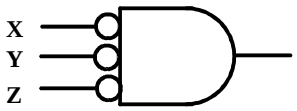
- **The basic NOR gate has the following symbol, illustrated for three inputs:**
  - **OR-Invert (NOR)**



- **NOR represents NOT - OR, i. e., the OR function with a NOT applied. The symbol shown is an OR-Invert. The small circle ("bubble") represents the invert function.**

NOR Gate (continued)

- **Applying DeMorgan's Law gives Invert-AND (NOR)**



- **This NOR symbol is called Invert-AND, since inputs are inverted and then ANDed together.**
- **OR-Invert and Invert-AND both represent the NOR gate. Having both makes visualization of circuit function easier.**
- **A NOR gate with one input degenerates to an inverter.**

NOR Gate (continued)

- **The NOR gate is another natural implementation for the simplest and fastest electronic circuits**
- **The NOR gate is a universal gate**
- **NOR usually does not have a defined operation symbol since**
  - **the NOR operation is not associative, and**
  - **we have difficulty dealing with non-associative mathematics!**

Exclusive OR/ Exclusive NOR

- **The *eXclusive OR* (*XOR*) function is an important Boolean function used extensively in logic circuits.**
- **The XOR function may be;**
  - **implemented directly as an electronic circuit (truly a gate) or**
  - **implemented by interconnecting other gate types (used as a convenient representation)**
- **The *eXclusive NOR* function is the complement of the XOR function**
- **By our definition, XOR and XNOR gates are complex gates.**

Exclusive OR/ Exclusive NOR

- **Uses for the XOR and XNORs gate include:**
  - **Adders/subtractors/multipliers**
  - **Counters/incrementers/decrementers**
  - **Parity generators/checkers**
- **Definitions**
  - **The XOR function is:** $X \oplus Y = X\overline{Y} + \overline{X}Y$
  - **The eXclusive NOR (XNOR) function, otherwise known as *equivalence* is:** $\overline{X \oplus Y} = XY + \overline{X}\,\overline{Y}$
- **Strictly speaking, XOR and XNOR gates do no exist for more that two inputs. Instead, they are replaced by odd and even functions.**

Truth Tables for XOR/XNOR

- **Operator Rules: XOR XNOR**

X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

X	Y	$\overline{(X \oplus Y)}$ or X≡Y
0	0	1
0	1	0
1	0	0
1	1	1

- **The XOR function means:**

  X OR Y, but NOT BOTH

- **Why is the XNOR function also known as the *equivalence* function, denoted by the operator ≡?**

XOR/XNOR (Continued)

- **The XOR function can be extended to 3 or more variables. For more than 2 variables, it is called an *odd function* or *modulo 2 sum* (*Mod 2 sum*), not an XOR:**

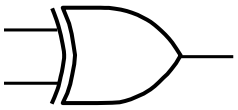
$$X \oplus Y \oplus Z = \overline{X}\,\overline{Y}\,Z + \overline{X}\,Y\,\overline{Z} + X\,\overline{Y}\,\overline{Z} + X\,Y\,Z$$

- **The complement of the odd function is the even function.**
- **The XOR identities:**

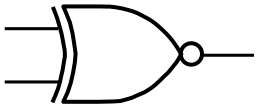
$$X \oplus 0 = X \qquad X \oplus 1 = \overline{X}$$
$$X \oplus X = 0 \qquad X \oplus \overline{X} = 1$$
$$X \oplus Y = Y \oplus X$$
$$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z$$

Symbols For XOR and XNOR

- **XOR symbol:**



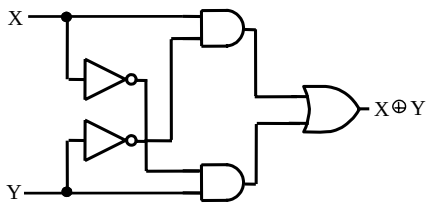
- **XNOR symbol:**



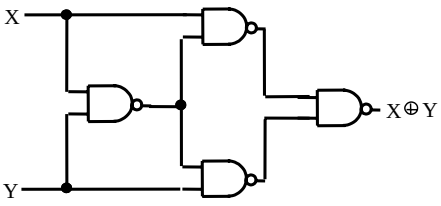
- **Symbols exist only for two inputs**

XOR Implementations

- **The simple SOP implementation uses the following structure:**



- **A NAND only implementation is**:

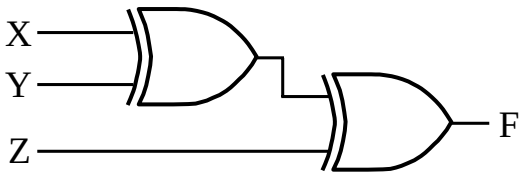


Odd and Even Functions

- The odd and even functions on a K-map form "checkerboard" patterns.
- The 1s of an odd function correspond to minterms having an index with an odd number of 1s.
- The 1s of an even function correspond to minterms having an index with an even number of 1s.
- Implementation of odd and even functions for greater than 4 variables as a two-level circuit is difficult, so we use "trees" made up of :
  - 2-input XOR or XNORs
  - 3- or 4-input odd or even functions

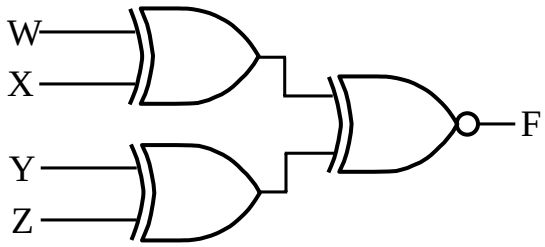
Example: Odd Function Implementation

- Design a 3-input odd function $F = X \oplus Y \oplus Z$ with 2-input XOR gates
- Factoring, $F = (X \oplus Y) \oplus Z$
- The circuit:



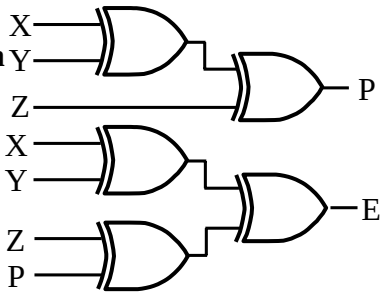
Example: Even Function Implementation

- **Design a 4-input odd function F = W⊕X⊕Y⊕Z with 2-input XOR and XNOR gates**
- **Factoring, F = (W⊕X)⊕(Y⊕Z)**
- **The circuit:**



Parity Generators and Checkers

- **In Chapter 1, a parity bit added to n-bit code to produce an n + 1 bit code:**
  - **Add odd parity bit to generate code words with even parity**
  - **Add even parity bit to generate code words with odd parity**
  - **Use odd parity circuit to check code words with even parity**
  - **Use even parity circuit to check code words with odd parity**
- **Example: n = 3. Generate even parity code words of length 4 with odd parity generator:**
- **Check even parity code words of length 4 with odd parity checker:**
- **Operation: (X,Y,Z) = (0,0,1) gives (X,Y,Z,P) = (0,0,1,1) and E = 0. If Y changes from 0 to 1 between generator and checker, then E = 1 indicates an error.**



Hi-Impedance Outputs

- **Logic gates introduced thus far**
  - **have 1 and 0 output values,**
  - **cannot have their outputs connected together, and**
  - **transmit signals on connections in only one direction.**
- **Three-state logic adds a third logic value, Hi-Impedance (Hi-Z), giving three states: 0, 1, and Hi-Z on the outputs.**
- **The presence of a Hi-Z state makes a gate output as described above behave quite differently:**
  - **"1 and 0" become "1, 0, and Hi-Z"**
  - **"cannot" becomes "can," and**
  - **"only one" becomes "two"**

Hi-Impedance Outputs (continued)

- **What is a Hi-Z value?**
  - **The Hi-Z value behaves as an open circuit**
  - **This means that, looking back into the circuit, the output appears to be disconnected.**
  - **It is as if a switch between the internal circuitry and the output has been opened.**
- **Hi-Z may appear on the output of any gate, but we restrict gates to:**
  - **a 3-state buffer, or**
  - **a transmission gate,**

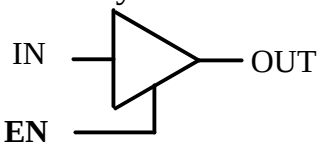
  each of which has one data input and one control input.

The 3-State Buffer

- **For the symbol and truth table, IN is the data input, and EN, the control input.**
- **For EN = 0, regardless of the value on IN (denoted by X), the output value is Hi-Z.**
- **For EN = 1, the output value follows the input value.**
- **Variations:**
  - **Data input, IN, can be inverted**
  - **Control input, EN, can be inverted**

  by addition of "bubbles" to signals.

Symbol



Truth Table

EN	IN	OUT
0	X	Hi-Z
1	0	0
1	1	1

Resolving 3-State Values on a Connection

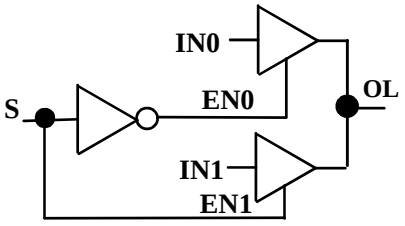
- **Connection of two 3-state buffer outputs, B1 and B0, to a wire, OUT**
- **Assumption: Buffer data inputs can take on any combination of values 0 and 1**
- **Resulting Rule: At least one buffer output value must be Hi-Z. Why?**
- **How many valid buffer output combinations exist?**
- **What is the rule for *n* 3-state buffers connected to wire, OUT?**
- **How many valid buffer output combinations exist?**

Resolution Table		
B1	B0	OUT
0	Hi-Z	0
1	Hi-Z	1
Hi-Z	0	0
Hi-Z	1	1
Hi-Z	Hi-Z	Hi-Z

3-State Logic Circuit

- **Data Selection Function: If s = 0, OL = IN0, else OL = IN1**
- **Performing data selection with 3-state buffers:**

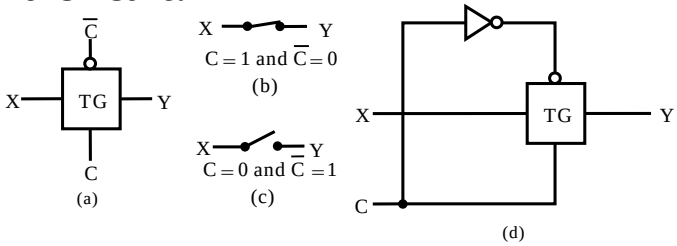
EN0	IN0	EN1	IN1	OL
0	X	1	0	0
0	X	1	1	1
1	0	0	X	0
1	1	0	X	1
0	X	0	X	X



- **Since EN0 = $\overline{S}$ and EN1 = S, one of the two buffer outputs is always Hi-Z plus the last row of the table never occurs.**

Transmission Gates

- **The transmission gate is one of the designs for an electronic switch for connecting and disconnecting two points in a circuit:**

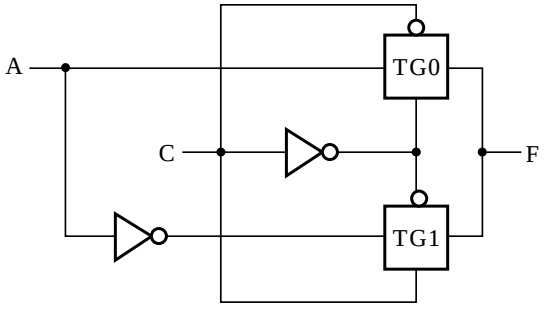


Transmission Gates (continued)

- In many cases, X can be regarded as a data input and Y as an output. C and $\overline{C}$, with complementary values applied, is a control input.
- With these definitions, the transmission gate, provides a 3-state output:
  - C = 1, Y = X (X = 0 or 1)
  - C = 0, Y = Hi-Z
- Care must be taken when using the TG in design, however, since X and Y as input and output are interchangeable, and signals can pass in both directions.

Circuit Example Using TG

- Exclusive OR $F = A \oplus C$



(a)

A	C	TG1	TG0	F
0	0	No path	Path	0
0	1	Path	No path	1
1	0	No path	Path	1
1	1	Path	No path	0

(b)

- The basis for the function implementation is TG-controlled paths to the output

More Complex Gates

- The remaining complex gates are SOP or POS structures with and without an output inverter.
- The names are derived using:
  - A - AND
  - O - OR
  - I - Inverter
  - Numbers of inputs on first-level “gates” or directly to second-level “gates”

More Complex Gates (continued)

- Example: AOI - AND-OR-Invert consists of a single gate with AND functions driving an OR function which is inverted.
- Example: 2-2-1 AO has two 2-input ANDS driving an OR with one additional OR input
- These gate types are used because:
  - the number of transistors needed is fewer than required by connecting together primitive gates
  - potentially, the circuit delay is smaller, increasing the circuit operating speed

Terms of Use

- Return to Title Page