CSE 675.02: Introduction to Computer Architecture

Basics of Digital Logic Design

Presentation D

Study: B.1, B2, B.3

Slides by Gojko Babić

From transistors to chips

- Chips from the bottom up:
  - Basic building block: the transistor = “on/off switch”
    - Digital signals – voltage levels high/low
  - Transistors are used to build logic gates
  - Logic gates make up functional and control units
  - Microprocessors contain several functional and control units
- This section provides an introduction into digital logic
  - Combinatorial and sequential logic
  - Boolean algebra and truth tables
  - Basic logic circuits:
    - Decoders, multiplexers, latches, flip-flops
    - Simple register design

Signals, Logic Operations and Gates

- Rather than referring to voltage levels of signals, we shall consider signals that are logically 1 or 0 (or asserted or de-asserted).

Logic operation

NOT

A	$\overline{A}$
0	1
1	0

AND

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

XOR

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Gates

Output is 1 iff:

- NOT: Input is 0
- AND: Both inputs are 1s
- OR: At least one input is 1
- XOR: Inputs are not equal

- Gates are simplest digital logic circuits, and they implement basic logic operations (functions).
- Gates are designed using transistors.
- Gates are used to build more complex circuits that implement more complex logic functions.

Classification of Logic Functions/Circuits

- Combinational logic functions (circuits):
  - any number of inputs and outputs
  - outputs y_i depend only on current values of inputs x_i

  Logic equations may be used to define a logic function.

  Example: A logic function with 4 inputs and 2 outputs

  $y_1 = (x_1 + (x_2 * x_3)) + ((\overline{x_3} * x_4) * x_1)$ "*" used for "and", "+" used for "or"

  $y_2 = (\overline{x_1} + (x_2 * x_4)) + ((x_1 * x_2) * \overline{x_3})$
- For sequential functions (circuits):
  - outputs depend on current values of inputs and some internal states.
- Any logic function (circuit) can be realized using only and, or and not operations (gates).
- nand and nor operations (gates) are universal.

Basic Laws of Boolean Algebra

- Identity laws: $A + 0 = A$
  $A * 1 = A$
- Inverse laws: $A + \overline{A} = 1$
  $A * \overline{A} = 0$
- Zero and one laws: $A + 1 = 1$
  $A * 0 = 0$
- Commutative laws: $A + B = B+A$
  $A * B = B * A$
- Associative laws: $A + (B + C) = (A + B) + C$
  $A * (B * C) = (A * B) * C$
- Distributive laws : $A * (B + C) = (A * B) + (A * C)$
  $A + (B * C) = (A + B) * (A + C)$
- DeMorgan's laws: $(\overline{A + B}) = \overline{A} * \overline{B}$
  $(\overline{A * B}) = \overline{A} + \overline{B}$

Simple Circuit Design: Example

Given logic equations, it is easy to design a corresponding circuit

$$y_1 = (x_1 + (x_2 * x_3)) + ((\overline{x_3} * x_4) * x_1) = x_1 + (x_2 * x_3) + (\overline{x_3} * x_4 * x_1)$$

$$y_2 = (\overline{x_1} + (x_2 * x_4)) + ((x_1 * x_2) * \overline{x_3}) = \overline{x_1} + (x_2 * x_4) + (x_1 * x_2 * \overline{x_3})$$

Truth Tables

- Another way (in addition to logic equations) to define functionality
- Problem: their sizes grow exponentially with number of inputs.

x_1	x_2	x_3	y_1	y_2
inputs			**outputs**	
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

What are logic equations corresponding to this table?

$$y_1 = x_1 + x_2 + x_3$$

$$y_2 = x_1 * x_2 * x_3$$

Design corresponding circuit.

Logic Equations in Sum of Products Form

- Systematic way to obtain logic equations from a given truth table.

x_1	x_2	x_3	y_1	y_2
inputs			**outputs**	
0	0	0	1	1
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

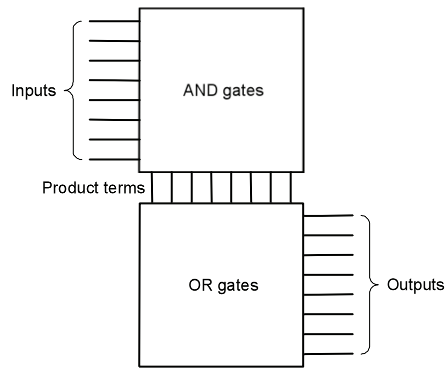
- **A product term is included for each row where y_i has value 1**
- **A product term includes all input variables.**
- **At the end, all product terms are ored**

$$y_1 = \overline{x_1}*\overline{x_2}*\overline{x_3} + \overline{x_1}*\overline{x_2}*x_3 + \overline{x_1}*x_2*x_3 + x_1*x_2*x_3$$

$$y_2 = \overline{x_1}*\overline{x_2}*\overline{x_3} + \overline{x_1}*\overline{x_2}*x_3 + \overline{x_1}*x_2*\overline{x_3} + x_1*\overline{x_2}*\overline{x_3} + x_1*\overline{x_2}*x_3$$

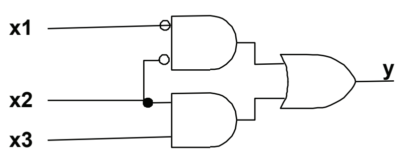
Programmable Logic Array - PLA

• PLA – structured logic implementation



Circuit → Logic Equation → Truth Table

• For the given logic circuit find its logic equation and truth table.



$$y = \overline{x_1} * \overline{x_2} + x_2 * x_3$$

x_1	x_2	x_3	y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

• Note that y column above is identical to y_1 column Slide 8.

• Thus, the given logic function may be defined with different logic equations and then designed by different circuits.

Minimization Using Boolean Laws

- Consider one of previous logic equations:

$$y_1 = \overline{x_1}*\overline{x_2}*\overline{x_3} + \overline{x_1}*\overline{x_2}*x_3 + \overline{x_1}*x_2*x_3 + x_1*x_2*x_3$$

$$= \overline{x_1}*\overline{x_2}*(\overline{x_3} + x_3) + x_2*x_3*(\overline{x_1} + x_1)$$

$$= \overline{x_1}*\overline{x_2} + x_2*x_3$$

But if we start grouping in some other way we may not end up with the minimal equation.

Gray codes

- a.k.a. reflected code – binary numeral system in which two successive values differ by only one digit

2-bit Gray code

00
01
11
10

3-bit Gray code

000
001
011
010
110
111
101
100

4-bit Gray code

0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000

Minimization Using Karnaugh Maps (1/4)

- Provides more formal way to minimization
- Includes 3 steps

1. **Form Karnaugh maps from the given truth table. There is one Karnaugh map for each output variable.**
2. **Group all 1s into as few groups as possible with groups as large as possible.**
3. **each group makes one term of a minimal logic equation for the given output variable.**

Forming Karnaugh maps – using "Gray code"

- **The key idea in forming the map is that horizontally and vertically adjacent squares correspond to input variables that differ in one variable only. Thus, a value for the first column (row) can be arbitrary, but labeling of adjacent columns (rows) should be such that those values differ in the value of only one variable.**

Minimization Using Karnaugh Maps (2/4)

Grouping **(This step is critical)**

When two adjacent squares contain 1s, they indicate the possibility of an algebraic simplification and they may be combined in one group of two. Similarly, two adjacent pairs of 1s may be combined to form a group of four, then two adjacent groups of four can be combined to form a group of eight, and so on. In general, the number of squares in any valid group must be equal to 2^k. Note that one 1 can be a member of more than one group and keep in mind that you should end up with as few groups as possible, which are as large as possible.

Finding Product Terms

The product term that corresponds to a given group is the product of variables whose values are constant in the group. If the value of input variable x_i is 0 for the group, then $\overline{x_i}$ is entered in the product, while if x_i has value 1 for the group, then x_i is entered in the product.

Minimization Using Karnaugh Maps (3/4)

Example 1: Given truth table, find minimal circuit

x_1	x_2	x_3	y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

x3 \ x1 x2	00	01	11	10
0	1	0	0	0
1	1	1	1	0

$$y = \overline{x_1} * \overline{x_2} + x_2 * x_3$$

Minimization Using Karnaugh Maps (4/4)

Example 2:

x_3 \ $x_1 x_2$	00	01	11	10
0	1	1	0	1
1	1	0	0	1

$$y = \overline{x_1} * \overline{x_3} + \overline{x_2}$$

Example 3:

$x_3 x_4$ \ $x_1 x_2$	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	0	1	1	0
10	0	0	0	0

$$y = \overline{x_1} * \overline{x_2} * \overline{x_3} + \overline{x_1} * x_2 * x_4 + x_2 * x_3 * x_4$$

Example 4:

$x_3 x_4$ \ $x_1 x_2$	00	01	11	10
00	0	0	1	1
01	0	1	0	0
11	1	0	0	1
10	0	0	1	1

$$y = x_1 * \overline{x_4} + \overline{x_2} * x_3 * x_4 + \overline{x_1} * x_2 * \overline{x_3} * x_4$$