CS 135: Computer Architecture I

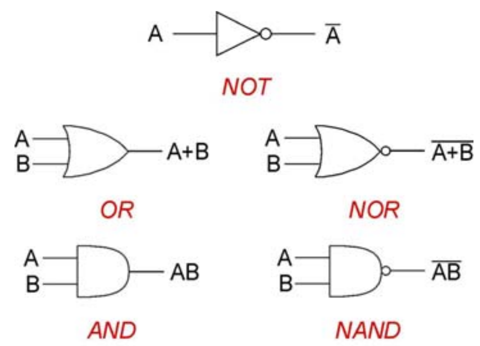
Boolean Algebra

Instructor: Prof. Bhagi Narahari

Dept. of Computer Science

Course URL: www.seas.gwu.edu/~bhagiweb/cs135/

Basic Logic Gates



Digital Logic Circuits

- We saw how we can build the simple logic gates using transistors
  - N-type: send 1 to gate to close ‘switch’
  - P-type: send 0 to gate to close ‘switch’
- Use these gates as building blocks to build more complex combinational circuits
  - Decoder: based on value of n-bit input control signal, select one of 2^N outputs
  - Multiplexer: based on value of N-bit input control signal, select one of 2^N inputs.
  - Adder: add two binary numbers
  - …any boolean function

Theory of Combinational Logic Design ?

- Is there a well grounded theory behind design of boolean logic circuits/functions ?
- Equivalent circuits ?
- Efficient design ?
  - Fewest gates used

Boolean Algebra

- To describe behavior of combinational circuit
  - Truth table
  - Boolean algebraic expressions
  - Digital logic circuit/diagram
- Algebraic expression written according to laws of boolean algebra specifies not only what a combinational circuit does, but also how it does it!

Truth Table to Digital Circuit design...

- Look at all rows with a 1 in the output
  - ALL conditions in the input must hold for the output to be 1 = AND of all the input conditions
  - Any row can be 1 = do an OR of all the row conditions
- When is $x_1 = 1$?
  - Look at all rows where $x_1 = 1$
  - What are the values of inputs for each of these rows ?

Canonical Boolean expressions with Minterms

- Each row in a truth table specifies values of all the input variables
  - What is the value of each input variable – 0 or 1
  - i.e., y=1 or y'=1 for each input variable y
- For the specific output, what row(s) are we interested ?
- When is the value of the output =1
  - When the value in the row of the truth table =1
  - What are the values of the input variables ?
- Canonical boolean expression
  - Any boolean expression can be converted to an equivalent two level AND-OR expression
    - an OR of AND terms,
    - each AND term corresponds to a 1 in the truth table row,
    - each AND term contains all input variables exactly once – i.e., a *minterm*
  - Canonical expression: OR of minterms

Boolean Algebra – Definitions..recall from CS123

- Boolean algebra has three operations defined over boolean variables:
  - OR (+), AND (.) and complement (')
- Recall fundamental properties of Boolean algebra
  - These apply to anything that is a boolean algebra
    - Sets, digital logic circuits, …

Boolean Algebra– Fundamental Properties

- Commutative:
  - $x+y = y+x$ $\quad$ $x.y = y.x$
- Associative
  - $(x+y)+z = x+(y+z)$ $\quad$ $(x.y).z = x.(y.z)$
- Distributive
  - $x+(y.z) = (x+y).(x+z)$ $\quad$ $x.(y+z)=(x.y)+(x.z)$
- Identity
  - $x=0 = x$ $\quad$ $x.1 = x$
- Complement
  - $x + (x') = 1$ $\quad$ $x.(x') = 0$

Laws of Boolean algebra

- Duality property: each boolean property has a dual property
  - Exchange + and . Exchange 1 and 0
- Many useful properties/theorems can be proved from the 10 fundamental properties

Example: Idempotent Property

- Prove: $x + x = x$
- Proof: use only the 10 fundamental laws
- $x+x = (x+x).1$; From identity property
- $(x+x).1 = (x+x).(x+x')$; complement
- $(x+x).(x+x')= (x.x) + (x.x')$; distributive
- $x + (x.x')= x +0$; complement
- $x+0 = x$; identity property
  - QED
- The duality property is: $x.x =x$

Some useful properties. . .

- Zero theorem
  - $x+1 =1$ $\quad$ $x.0 = 0$
- Absorption property
  - $x + x.y = x$ $\quad$ $x.(x+y) = x$
- De Morgan's law
  - $(a.b)' = a' + b'$ $\quad$ $(a+b)' = a' . b'$
  - $(a.b.c)' = a'+b'+c'$ $\quad$ $(a+b+c)'= a'.b'.c'$
- Complement
  - $(x')' = x$
  -

Two Level Circuits

- Every boolean expression can be transformed to an AND-OR expression
  - Resulting in a 2 level circuit
- Advantage of 2 level circuit ?
  - Gate delays
    - Go through only two levels/layers of gates

Why this discussion of Boolean Algebra...

- Every boolean expression has a corresponding logic circuit diagram; and every logic circuit diagram has a corresponding boolean expression
  - One to one correspondence
- But a given truth table can have several corresponding implementations
- How to map from truth table to boolean expression ?
  - How to pick the "best" boolean expression ?

Simplification of boolean expressions

- The boolean expression/function *x(a,b,c,d) = a'bd' + a'c'd' + a'bc'd'*
- Can be simplified using absorption property to

  a'bd' + (a'c'd')+(a'c'd')b = a'bd'+ a'c'd'

Combinational Circuit Design: Truth Tables and Boolean expressions

- Given truth table we want to find an "efficient" implementation (i.e., circuit)
  - Efficient in speed
  - Efficient in number of gates
  - Simplicity of design
- Canonical boolean expression
  - Any boolean expression can be converted to an equivalent two level AND-OR expression
    - an OR of AND terms,
    - each AND term corresponds to a 1 in the truth table row,
    - each AND term contains all input variables exactly once – i.e., a ***minterm***
  - Canonical expression: OR of minterms
- Graphical method for designing two level circuits with 3 or 4 variables using minimum possible number of gates
  - What is this method ????

Canonical Expressions

- Consider boolean expression *x,* where *x(a,b,c)= abc + a'bc +ab*
  - First two are minterms since they contain all three input variables
- *abc +a'bc +ab = abc + a'bc + ab(c+c')*
  = abc +a'bc + abc + abc'
  = abc + a'bc + abc'

Truth table?

Example

a	b	c	x
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Transformation of the Boolean expression

- *abc + a'bc + abc'*
  = (a +a')bc + abc'
  = bc + abc'
- *a'bc + abc + abc'*
  = a'bc + ab(c + c')
  = a'bc + ab
- *a'bc + abc + abc'*
  = a'bc + abc + abc + abc'
  = (a'+a)bc + ab (c+c')
  = bc + ab

Truth table in 2-dimensions

bc

b (b=1)

a \ bc	00	01	11	10
0	0	0	1	0
1	0	0	1	1

(a (a=1)

c (c=1)

x_1 = *bc* x_2 =*ab*

Therefore, x = a'bc + abc + abc' = bc + ab

Distance between minterms

- Concept of "distance" between two minterms (Hamming distance):
  - Number of variables that are different
  - Distance(abc, abc')=1 only c and c' different
  - Distance(abc, a'bc')=2 both a and c are different
- Arrange 2-d truth table so that values in consecutive columns(rows) differ in one bit position

Karnaugh Maps

- Graphical way to represent boolean functions
  - Based on concept of distance
- Recognizing adjacent minterms is key to minimization of AND-OR expression
- K-map is a tool to minimize a two level circuit that it makes it easy to spot adjacent minterms
- Karnaugh Map is a truth table arranged so that adjacent entries represent minterms that differ by one.

Grouping minterms in K-Map

- Group 'cells' in K-map that are adjacent and have a value of 1 in the cell
  - Group of 2 cells in 3 variable K-map: is an AND of two variables
  - Group of 4 cells in 3 variable K-map: is single variable

Minimization using K-Maps

- Minimization procedure : determine best set of groups that will cover all the 1's in the K-map
  - "best" means the set that corresponds to a two-level circuit with the least number of gates and the least number of inputs per gate.
  - The number of groups equals number of AND gates
  - We want the smallest number of groups with each group as large as possible such that the groups cover all the 1's

Example

bc

b (b=1)

a	00	01	11	10
0	1	1	0	0
1	0	1	1	0

(a (a=1)

c (c=1)

Example

bc

b (b=1)

a	00	01	11	10
0	1	1	0	0
1	0	1	1	0

(a (a=1)

c (c=1)

Example

bc

b (b=1)

a	00	01	11	10
0	1	1	0	0
1	0	1	1	0

(a (a=1)

c (c=1)

a’b’ + b’c + ac

Example

bc

b (b=1)

a	00	01	11	10
0	1	1	0	0
1	0	1	1	0

(a (a=1)

c (c=1)

a’b’ + ac

4 variable Karnaugh map

ab \ cd	00	01	11	10
00	1			
01	1		1	
11	1	1	1	
10	1			

b a d c

Summary of Combinational Logic

- Combinational device/circuit: any circuit built using the basic gates
- Expressed as
  - Truth table
  - Digital circuit
  - Boolean function
- Any boolean function can be expressed as two level function
- Minimization procedure: Karnaugh Map
  - Try to minimize the number of gates, and inputs to gates, in a two level circuit

Combinational vs. Sequential

- Combinational Circuit
  - always gives the same output for a given set of inputs
    - ex: adder always generates sum and carry, regardless of previous inputs
- Sequential Circuit
  - stores information
  - output depends on stored information (state) plus input
    - so a given input might produce different outputs, depending on the stored information
  - *example: vending machine*
    - Current total increases when you insert coins
    - output depends on previous state
  - useful for building “memory” elements and “state machines”

Vending Machine

- Problem: Read input coins, Keep track of current total.
  - If total equal to (or greater than) 75 cents then process request
- What do we need to keep track of ?
  - Current total (of coins fed into the machine)
  - What is a "state" of the machine
    - Current total!
  - Assume only 5c, 10c, 25cent coins
- 'transition' between states ?
  - Current state 10 goes to
    - next state 15 if input = 5c
    - Next state 20 if input =10c
    - Next state 35 if input=25c

Next . . . Sequential Logic

- Build a device, using combinational logic devices, to store a value
  - RS Latch (also called SR Latch)
- Implement concept of memory
- Methodology behind design of sequential logic circuits
  - Finite State Machines
- Combine sequential and combinational logic devices to "assemble" a simple processor!

Sequential Circuits

- Combinational logic circuits are perfect for situations when we require the immediate application of a Boolean function to a set of inputs.
- There are other times, however, when we need a circuit to change its value with consideration to its current state as well as its inputs.
  - These circuits have to "remember" their current state.
- *Sequential logic circuits* provide this functionality for us.