CSE 311: Foundations of Computing

Fall 2013

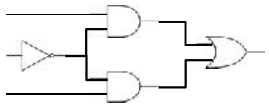
Lecture 3: Logic and Boolean algebra

announcements

Reading assignments

- Propositional Logic
  - 1.1 -1.3 7th Edition
  - 1.1 -1.2 6th Edition
- Boolean Algebra
  - 12.1 – 12.3 7th Edition
  - 11.1 – 11.3 6th Edition

combinational logic circuits



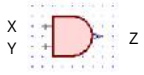
Design a circuit to compute the majority of 3 bits.

What about majority of 5 bits?

some other gates

NAND

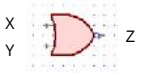
$\neg X \wedge \neg Y$



X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

NOR

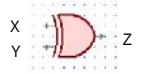
$\neg X \vee \neg Y$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

XOR

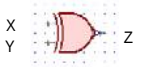
$X \oplus Y$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

$X \leftrightarrow Y, X = Y$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

review: logical equivalence

Terminology: A compound proposition is a
Tautology if it is always true
Contradiction if it is always false
Contingency if it can be either true or false

p Ę Ė p

$p \oplus p$

$(p \rightarrow q) \wedge p$

$(p \wedge q) \vee (p \wedge \neg q) \vee (\neg p \wedge q) \vee (\neg p \wedge \neg q)$

review: logical equivalence

p and *q* are *logically equivalent* if and only if
$p \leftrightarrow q$ is a tautology
i.e. p and *q* have the same truth table

The notation $p \equiv q$ denotes *p* and *q* are logically equivalent

Example: $p \equiv$ Ė Ė p

p	Ė *p*	Ė Ė *p*	*p* é Ė Ė *p*

review: De Morgan's laws

$$\neg (p \wedge q) \equiv \neg p \vee \neg q$$
$$\neg (p \vee q) \equiv \neg p \wedge \neg q$$

What are the negations of:

The Yankees and the Phillies will play in the World Series.

It will rain today or it will snow on New Year's Day.

review: De Morgan's laws

Example: $\neg (p$ ė $q) \equiv (\neg p \vee \neg q)$

p	*q*	Ė *p*	Ė *q*	Ė *p* Ę Ė *q*	*p* ė *q*	Ė (*p* ė *q*)	Ė (*p* ė *q*) é (Ė *p* Ę Ė *q*)
T	T						
T	F						
F	T						
F	F						

review: Law of Implication

$$(p \rightarrow q) \equiv (\neg p \vee q)$$

p	q	p ì q	Ė p	Ė p Ę q	(p ì q) é (Ė p Ę q)
T	T				
T	F				
F	T				
F	F				

understanding connectives

- Reflect basic rules of reasoning and logic
- Allow manipulation of logical formulas
  - Simplification
  - Testing for equivalence
- Applications
  - Query optimization
  - Search optimization and caching
  - Artificial Intelligence
  - Program verification

review: properties of logical connectives

- Identity
- Domination
- Idempotent
- Commutative
- Associative
- Distributive
- Absorption
- Negation
- De Morgan's Laws

Textbook: 1.3 7th Edition/1.2 6th Edition, Table 6

some equivalences related to implication

$$p \rightarrow q \equiv \neg p \vee q$$
$$p \rightarrow q \equiv \neg q \rightarrow \neg p$$
$$p \vee q \equiv \neg p \rightarrow q$$
$$p \wedge q \equiv \neg (p \rightarrow \neg q)$$
$$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$$
$$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$$
$$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$$
$$\neg (p \leftrightarrow q) \equiv p \leftrightarrow \neg q$$

some equivalences related to implication

$p \rightarrow q \quad \equiv \quad \neg q \rightarrow \neg p$

logical proofs

To show P is equivalent to Q

- Apply a series of logical equivalences to subexpressions to convert P to Q

To show P is a tautology

- Apply a series of logical equivalences to subexpressions to convert P to **T**

prove this is a tautology

$(p \wedge q) \rightarrow (p \vee q)$

prove this is a tautology

$(p \wedge (p \rightarrow q)) \rightarrow q$

proving non-equivalence

$(p \rightarrow q) \rightarrow r$ $\qquad$ $p \rightarrow (q \rightarrow r)$

boolean logic

Combinational logic
- $\text{output}_t = F(\text{input}_t)$

Sequential logic
- $\text{output}_t = F(\text{output}_{t-1}, \text{input}_t)$
  - output dependent on history
  - concept of a time step (clock)

An algebraic structure consists of
- a set of elements B = {0, 1}
- binary operations { + , • } (OR, AND)
- and a unary operation { ' } (NOT)

a quick combinational logic example

Calendar subsystem:

# of days in a month (to control watch display)

- used in controlling the display of a wrist-watch LCD screen
- **inputs:** month, leap year flag
- **outputs:** number of days

Example: (March, non-leap year) → **31**

implementation in software

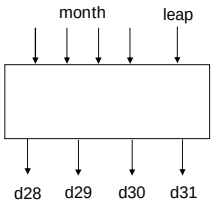
```
integer number_of_days (month, leap_year_flag){
   switch (month) {
      case 1: return (31);
      case 2: if (leap_year_flag == 1) then
              return (29) else return (28);
      case 3: return (31);
      ...
      case 12: return (31);
      default: return (0);
   }
}
```

20

implementation with combinatorial logic

Encoding:

- how many bits for each input/output?
- binary number for month
- four wires for 28, 29, 30, and 31



month	leap	d28	d29	d30	d31
0000	–	–	–	–	–
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
0101	–	0	0	0	1
0110	–	0	0	1	0
0111	–	0	0	0	1
1000	–	0	0	0	1
1001	–	0	0	1	0
1010	–	0	0	0	1
1011	–	0	0	1	0
1100	–	0	0	0	1
1101	–	–	–	–	–
1110	–	–	–	–	–
1111	–	–	–	–	–

implementation with combinatorial logic

Truth-table to logic to switches to gates

d28 = "1 when month=0010 and leap=0"

d28 = m8'•m4'•m2•m1'•leap'

d31 = "1 when month=0001 or month=0011 or … month=1100"

d31 = (m8'•m4'•m2'•m1) + (m8'•m4'•m2•m1) +
… + (m8•m4•m2'•m1')

d31 = can we simplify more?

month	leap	d28	d29	d30	d31
0000	–	–	–	–	–
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
…					
1100	–	0	0	0	1
1101	–	–	–	–	–
111–	–	–	–	–	–

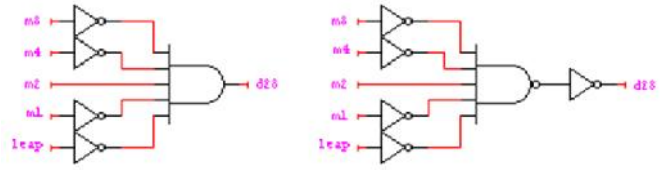
implementation with combinatorial logic

d28 = m8'•m4'•m2•m1'•leap'

d29 = m8'•m4'•m2•m1'•leap

d30 = (m8'•m4•m2'•m1') + (m8'•m4•m2•m1') +
(m8•m4'•m2'•m1) + (m8•m4'•m2•m1)
= (m8'•m4•m1') + (m8•m4'•m1)

d31 = (m8'•m4'•m2'•m1) + (m8'•m4'•m2•m1) +
(m8'•m4•m2'•m1) + (m8'•m4•m2•m1) +
(m8•m4'•m2'•m1') + (m8•m4'•m2•m1') +
(m8•m4•m2'•m1')



combinational logic

- Switches
- Basic logic and truth tables
- Logic functions
- Boolean algebra
- Proofs by re-writing and by truth table