CSE 311 Foundations of Computing I

Lecture 4, Boolean Logic

Autumn 2012

Announcements

- Reading assignments
  - Boolean Algebra
    - 12.1 – 12.3 7th Edition
    - 11.1 – 11.3 6th Edition
    - 10.1 – 10.3 5th Edition
  - Predicates and Quantifiers
    - 1.4 7th Edition
    - 1.3 5th and 6th Edition

Boolean logic

- Combinational logic
  - $output_t = F(input_t)$
- Sequential logic
  - $output_t = F(output_{t-1}, input_t)$
    - 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

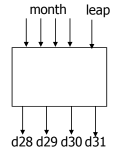
- Calendar subsystem: number of days in a month (to control watch display)
  - used in controlling the display of a wrist-watch LCD screen
  - inputs: month, leap year flag
  - outputs: number of days

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);
   }
}
```


Implementation as a combinational digital system

- 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	–	–	–	–	–

Combinational example (cont.)

- 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–	–	–	–	–	–

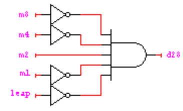
Combinational example (cont.)

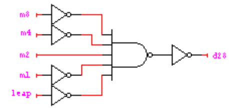
d28 = m8'•m4'•m2•m1'•leap'

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')



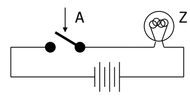


Combinational logic

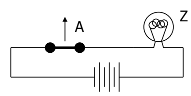
- Switches
- Basic logic and truth tables
- Logic functions
- Boolean algebra
- Proofs by re-writing and by perfect induction

Switches: basic element of physical implementations

- Implementing a simple circuit (arrow shows action if wire changes to “1”):



close switch (if A is “1” or asserted)
and turn on light bulb (Z)



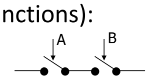
open switch (if A is “0” or unasserted)
and turn off light bulb (Z)

Z ≡ A

Switches (cont.)

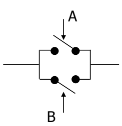
- Compose switches into more complex ones (Boolean functions):

AND



Z ≡ A and B

OR

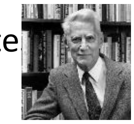


Z ≡ A or B

Transistor networks

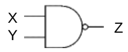
- Modern digital systems are designed in CMOS technology
  - MOS stands for Metal-Oxide on Semiconductor
  - C is for complementary because there are both normally-open and normally-closed switches
- MOS transistors act as voltage-controlled switches
  - similar, though easier to work with than relays.

Multi-input logic gate

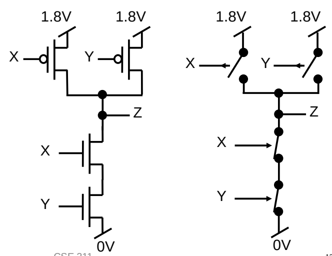


Claude Shannon – 1938

- CMOS logic gates are inverting
  - Easy to implement NAND, NOR, NOT while AND, OR, and Buffer are harder

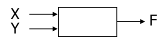


X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0



Possible logic functions of two variables

- There are 16 possible functions of 2 input variables:
  - in general, there are 2**(2**n) functions of n inputs



X	Y	16 possible functions (F_0–F_{15})															
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	X and Y		X		Y	X xor Y	X or Y	X nor Y not (X or Y)	X = Y	not Y		not X		X nand Y not (X and Y)	1

Boolean algebra

George Boole – 1854

- An algebraic structure consists of
  - a set of elements B
  - binary operations { + , • }
  - and a unary operation { ’ }
  - such that the following axioms hold:

1. the set B contains at least two elements: a, b

2. closure:	a + b is in B	a • b is in B
3. commutativity:	a + b = b + a	a • b = b • a
4. associativity:	a + (b + c) = (a + b) + c	a • (b • c) = (a • b) • c
5. identity:	a + 0 = a	a • 1 = a
6. distributivity:	a + (b • c) = (a + b) • (a + c)	a • (b + c) = (a • b) + (a • c)
7. complementarity:	a + a’ = 1	a • a’ = 0

Logic functions and Boolean algebra

Any logic function that can be expressed as a truth table can be written as an expression in Boolean algebra using the operators: ’, +, and ●

X, Y are Boolean algebra variables

X	Y	X • Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X’	X’ • Y
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	0

X	Y	X’	Y’	X • Y	X’ • Y’	(X • Y) + (X’ • Y’)
0	0	1	1	0	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	1

(X • Y) + (X’ • Y’) ➔ X = Y

Boolean expression that is true when the variables X and Y have the same value and false, otherwise

Axioms and theorems of Boolean algebra

identity
1. X + 0 = X — 1D. X • 1 = X

null
2. X + 1 = 1 — 2D. X • 0 = 0

idempotency:
3. X + X = X — 3D. X • X = X

involution:
4. (X’)’ = X

complementarity:
5. X + X’ = 1 — 5D. X • X’ = 0

commutatively:
6. X + Y = Y + X — 6D. X • Y = Y • X

associativity:
7. (X + Y) + Z = X + (Y + Z) — 7D. (X • Y) • Z = X • (Y • Z)

distributivity:
8. X • (Y + Z) = (X • Y) + (X • Z) — 8D. X + (Y • Z) = (X + Y) • (X + Z)

Axioms and theorems of Boolean algebra (cont.)

uniting:
9. X • Y + X • Y’ = X — 9D. (X + Y) • (X + Y’) = X

absorption:
10. X + X • Y = X — 10D. X • (X + Y) = X
11. (X + Y’) • Y = X • Y — 11D. (X • Y’) + Y = X + Y

factoring:
12. (X + Y) • (X’ + Z) = X • Z + X’ • Y — 12D. X • Y + X’ • Z = (X + Z) • (X’ + Y)

consensus:
13. (X • Y) + (Y • Z) + (X’ • Z) = X • Y + X’ • Z — 13D. (X + Y) • (Y + Z) • (X’ + Z) = (X + Y) • (X’ + Z)

de Morgan’s:
14. (X + Y + ...)’ = X’ • Y’ • ... — 14D. (X • Y • ...)’ = X’ + Y’ + ...

Proving theorems (rewriting)

- Using the laws of Boolean algebra:
  - e.g., prove the theorem: X • Y + X • Y' = X

distributivity (8)	X • Y + X • Y' = X • (Y + Y')
complementarity (5)	= X • (1)
identity (1D)	= X

e.g., prove the theorem: X + X • Y = X

identity (1D)	X + X • Y = X • 1 + X • Y
distributivity (8)	= X • (1 + Y)
identity (2)	= X • (1)
identity (1D)	= X

Proving theorems (perfect induction)

- Using perfect induction (complete truth table):
  - e.g., de Morgan's:

(X + Y)' = X' • Y'
NOR is equivalent to AND with inputs complemented

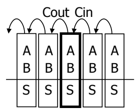
X	Y	X'	Y'	(X + Y)'	X' • Y'
0	0	1	1		
0	1	1	0		
1	0	0	1		
1	1	0	0		

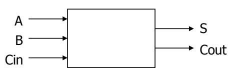
(X • Y)' = X' + Y'
NAND is equivalent to OR with inputs complemented

X	Y	X'	Y'	(X • Y)'	X' + Y'
0	0	1	1		
0	1	1	0		
1	0	0	1		
1	1	0	0		

A simple example: 1-bit binary adder

- Inputs: A, B, Carry-in
- Outputs: Sum, Carry-out





A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S = A' B' Cin + A' B Cin' + A B' Cin' + A B Cin
Cout = A' B Cin + A B' Cin + A B Cin' + A B Cin

Apply the theorems to simplify expressions

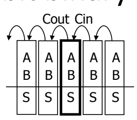
- The theorems of Boolean algebra can simplify expressions
  - e.g., full adder's carry-out function

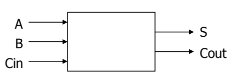
Cout = A' B Cin + A B' Cin + A B Cin' + A B Cin
= A' B Cin + A B' Cin + A B Cin' + A B Cin + A B Cin
= A' B Cin + A B Cin + A B' Cin + A B Cin' + A B Cin
= (A' + A) B Cin + A B' Cin + A B Cin' + A B Cin
= (1) B Cin + A B' Cin + A B Cin' + A B Cin
= B Cin + A B' Cin + A B Cin' + A B Cin + A B Cin
= B Cin + A B' Cin + A B Cin + A B Cin' + A B Cin
= B Cin + A (B' + B) Cin + A B Cin' + A B Cin
= B Cin + A (1) Cin + A B Cin' + A B Cin
= B Cin + A Cin + A B (Cin' + Cin)
= B Cin + A Cin + A B (1)
= B Cin + A Cin + A B

adding extra terms creates new factoring opportunities

A simple example: 1-bit binary adder

- Inputs: A, B, Carry-in
- Outputs: Sum, Carry-out





A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Cout = B Cin + A Cin + A B

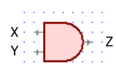
S = A' B' Cin + A' B Cin' + A B' Cin' + A B Cin
= A' (B' Cin + B Cin') + A (B' Cin' + B Cin)
= A' Z + A Z'
= A xor Z = A xor (B xor Cin)

From Boolean expressions to logic gates

- NOT X' $\overline{X}$ ~X X/

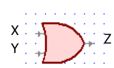
X	Y
0	1
1	0

- AND X • Y XY X ∧ Y



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

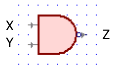
- OR X + Y X ∨ Y



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

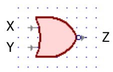
From Boolean expressions to logic gates (cont'd)

- NAND



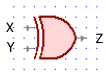
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

- NOR



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

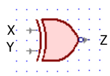
- XOR
  $X \oplus Y$



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

X xor Y = X Y' + X' Y
X or Y but not both
("inequality", "difference")

- XNOR
  X = Y



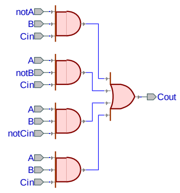
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

X xnor Y = X Y + X' Y'
X and Y are the same
("equality", "coincidence")

Full adder: Carry-out

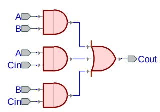
Before Boolean minimization

Cout = A'BCin + AB'Cin
+ ABCin' + ABCin



After Boolean minimization

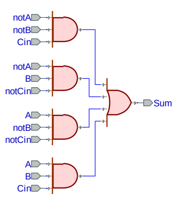
Cout = BCin + ACin + AB



Full adder: Sum

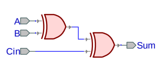
Before Boolean minimization

Sum = A'B'Cin + A'BCin'
+ AB'Cin' + ABCin

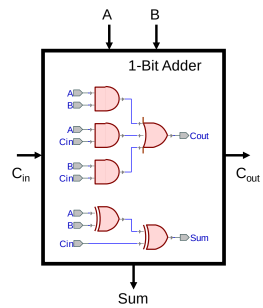


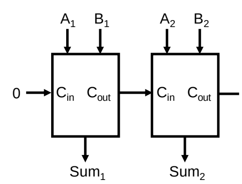
After Boolean minimization

Sum = (A⊕B) ⊕ Cin



Preview: A 2-bit ripple-carry adder





Mapping truth tables to logic gates

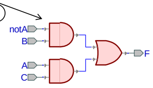
- Given a truth table:
  1. Write the Boolean expression
  2. Minimize the Boolean expression
  3. Draw as gates
  4. Map to available gates

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

1

F = A'BC'+A'BC+AB'C+ABC
2 = A'B(C'+C)+AC(B'+B)
= A'B+AC

3



4