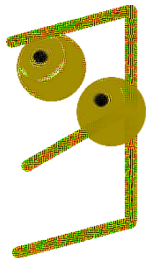
CSE 311



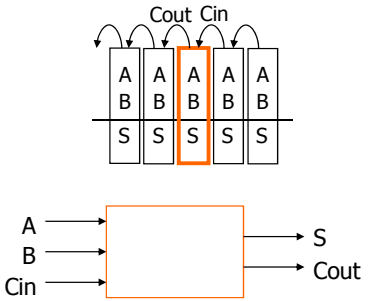
Foundations of Computing I

Fall 2014

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 Theorems to Simplify Expressions

The theorems of Boolean algebra can simplify expressions

- **e.g., full adder's carry-out function**

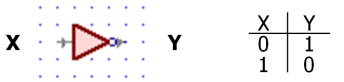
Cout = A' B Cin + A B' Cin + A B Cin' + A B Cin
= A' B Cin + A B' Cin + A B Cin' + **A B Cin + A B Cin**
= A' B Cin + A B Cin + A B' Cin + A B Cin' + A B Cin
= (A' + A) B Cin + A B' Cin + A B Cin' + A B Cin
= (1) B Cin + A B' Cin + A B Cin' + A B Cin
= B Cin + A B' Cin + A B Cin' + **A B Cin + A B Cin**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

Gates Again!

NOT

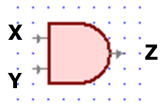
X' $\bar{X}$ $\neg X$



X	Y
0	1
1	0

AND

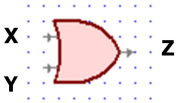
$X \cdot Y$ XY $X \wedge Y$



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

OR

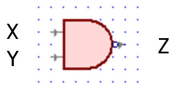
$X + Y$ $X \vee Y$



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

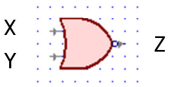
More Gates!

NAND
$\neg(X \wedge Y)$ $(XY)'$



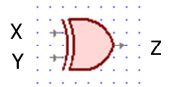
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

NOR
$\neg(X \vee Y)$ $(X + Y)'$



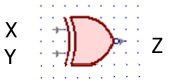
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

XOR
$X \oplus Y$



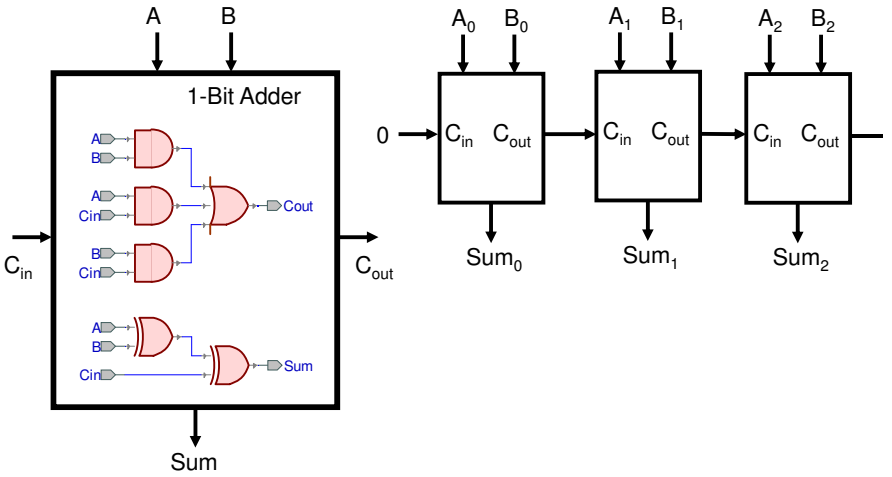
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

XNOR
$X \leftrightarrow Y$



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

A 2-bit Ripple-Carry Adder



Mapping Truth Tables to Logic Gates

Given a truth table:

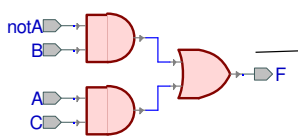
1. Write the Boolean expression
2. Minimize the Boolean expression
3. Draw as gates
4. Map to available gates

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

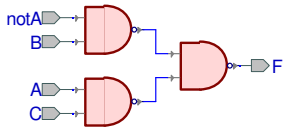
① (from truth table to expression)

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

③



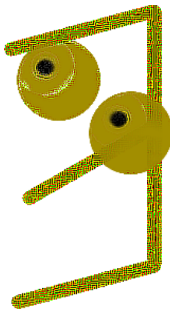
④



CSE 311: Foundations of Computing

Fall 2014

Lecture 5: Canonical Forms, Predicate Logic

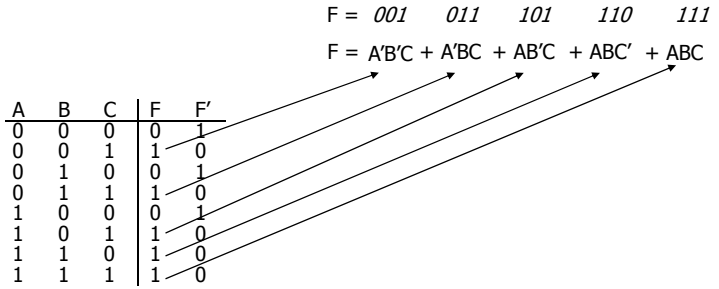


Canonical Forms

- Truth table is the unique signature of a Boolean function
- The same truth table can have many gate realizations
  - –we've seen this already
  - –depends on how good we are at Boolean simplification
- Canonical forms
  - standard forms for a Boolean expression
  - we all come up with the same expression

Sum-of-Products Canonical Form

- also known as Disjunctive Normal Form (DNF)
- also known as minterm expansion



Sum-of-Products Canonical Form

Product term (or minterm)

- ANDed product of literals – input combination for which output is true
- each variable appears exactly once, true or inverted (but not both)

A	B	C	minterms
0	0	0	A'B'C'
0	0	1	A'B'C
0	1	0	A'BC'
0	1	1	A'BC
1	0	0	AB'C'
1	0	1	AB'C
1	1	0	ABC'
1	1	1	ABC

F in canonical form:

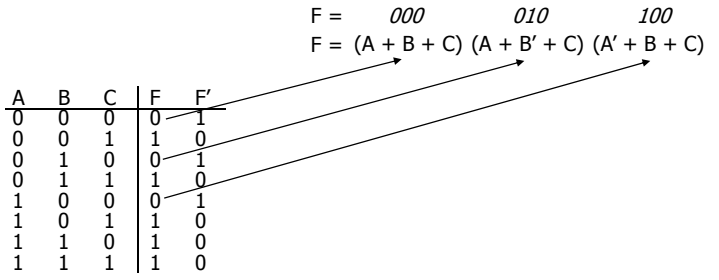
F(A, B, C) = A'B'C + A'BC + AB'C + ABC' + ABC

canonical form ≠ minimal form

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

Product-of-Sums Canonical Form

- Also known as Conjunctive Normal Form (CNF)
- Also known as maxterm expansion



s-o-p, p-o-s, and de Morgan's theorem

Complement of function in sum-of-products form:

– F′ = A′B′C′ + A′BC′ + AB′C′

Complement again and apply de Morgan's and get the product-of-sums form:

– (F′)′ = (A′B′C′ + A′BC′ + AB′C′)′

– F = (A + B + C) (A + B′ + C) (A′ + B + C)

Product-of-Sums Canonical Form

Sum term (or maxterm)

– ORed sum of literals – input combination for which output is false

– each variable appears exactly once, true or inverted (but not both)

A	B	C	maxterms
0	0	0	A+B+C
0	0	1	A+B+C′
0	1	0	A+B′+C
0	1	1	A+B′+C′
1	0	0	A′+B+C
1	0	1	A′+B+C′
1	1	0	A′+B′+C
1	1	1	A′+B′+C′

F in canonical form:

F(A, B, C) = (A + B + C) (A + B′ + C) (A′ + B + C)

canonical form ≠ minimal form

F(A, B, C) = (A + B + C) (A + B′ + C) (A′ + B + C)
= (A + B + C) (A + B′ + C)
(A + B + C) (A′ + B + C)
= (A + C) (B + C)

Predicate Logic

- **Propositional Logic**
  - If the tortoise walks at a rate of one node per step, and the hare walks at a rate of two nodes per step, …
- **Predicate Logic**
  - If the tortoise is on node x, and the hare is on node 2x, then …

Predicate Logic

- **Propositional Logic**
  - Allows us to analyze complex propositions in terms of their simpler constituent parts joined by connectives
- **Predicate Logic**
  - Lets us analyze them at a deeper level…

Predicate Logic

Predicate or **Propositional Function**

– A function that returns a truth value, e.g.,

"x is a cat"

"x is prime"

"student x has taken course y"

"x > y"

"x + y = z" or Sum(x, y, z)

"5 < x"

Predicates will have variables or constants as arguments.

Domain of Discourse

We must specify a "domain of discourse", which is the possible things we're talking about.

"x is a cat"
(e.g., mammals)

"x is prime"
(e.g., numbers)

student x has taken course y"
(e.g., students and courses)

Quantifiers

$\forall x\ P(x)$

P(x) is true for every x in the domain

read as "for all x, P of x"

$\exists x\ P(x)$

There is an x in the domain for which P(x) is true

read as "there exists x, P of x"

Statements with Quantifiers

Domain:
Positive Integers

Even(x)
Odd(x)
Prime(x)
Greater(x,y)
(or "x>y")
Equal(x,y)
(or "x=y")
Sum(x,y,z)

- $\exists x$ Even(x)
- $\forall x$ Odd(x)
- $\forall x$ (Even(x) $\vee$ Odd(x))
- $\exists x$ (Even(x) $\wedge$ Odd(x))
- $\forall x$ Greater(x+1, x)
- $\exists x$ (Even(x) $\wedge$ Prime(x))

Statements with Quantifiers

- $\forall x \exists y$ Greater (y, x)
- $\forall x \exists y$ Greater (x, y)
- $\forall x \exists y$ (Greater(y, x) $\wedge$ Prime(y))
- $\forall x$ (Prime(x) $\rightarrow$ (Equal(x, 2) $\vee$ Odd(x))
- $\exists x \exists y$ (Sum(x, 2, y) $\wedge$ Prime(x) $\wedge$ Prime(y))

Domain:
Positive Integers

Even(x)
Odd(x)
Prime(x)
Greater(x,y)
(or "x>y")
Equal(x,y)
(or "x=y")
Sum(x,y,z)

English to Predicate Logic

- **"Red cats like tofu"**
- **"Some red cats don't like tofu"**

Cat(x)
Red(x)
LikesTofu(x)

Negations of Quantifiers

- **not every positive integer is prime**
- **some positive integer is not prime**
- **prime numbers do not exist**
- **every positive integer is not prime**

Negations of Quantifiers

- $\forall x$ PurpleFruit(x)
  - **"All fruits are purple"**
- $\neg\forall x$ PurpleFruit(x)
  - **"Not all fruits are purple"**
- **How about** $\exists x$ PurpleFruit(x)**?**
  - **"There is a purple fruit"**
  - **If it's the negation, all situations should be covered by a statement and its negation**
  - **Consider the domain** {Orange}**: Neither statement is true!**
- **How about** $\exists x \neg$PurpleFruit(x)**?**
  - **"There is a fruit that isn't purple"**

Domain:
Fruit

PurpleFruit(x)

De Morgan's Laws for Quantifiers

$$\neg\forall x\ P(x) \equiv \exists x\ \neg P(x)$$
$$\neg\exists x\ P(x) \equiv \forall x\ \neg P(x)$$

de morgan's laws for quantifiers

$$\neg\forall x\ P(x) \equiv \exists x\ \neg P(x)$$
$$\neg\exists x\ P(x) \equiv \forall x\ \neg P(x)$$

"There is no largest integer"

$$\neg\exists x\,\forall y\ (x \geq y)$$
$$\equiv \forall x\,\neg\forall y\ (x \geq y)$$
$$\equiv \forall x\ \exists y\,\neg(x \geq y)$$
$$\equiv \forall x\ \exists y\ \ (y > x)$$

"For every integer there is a larger integer"

scope of quantifiers

example: Notlargest(x) $\equiv \exists$ y Greater (y, x)
$\equiv \exists$ z Greater (z, x)

truth value:

doesn't depend on y or z "bound variables"

does depend on x "free variable"

quantifiers only act on free variables of the formula they quantify

$\forall$ x ($\exists$ y (P(x,y) $\rightarrow \forall$ x Q(y, x)))

Scope of Quantifiers

$\exists x\ (P(x) \wedge Q(x))$ **vs.** $\exists x\ P(x) \wedge \exists x\ Q(x)$