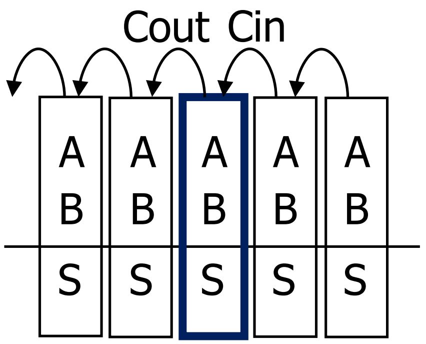
# … gradescope?


≪ e-mail invitations

# adding numbers in binary

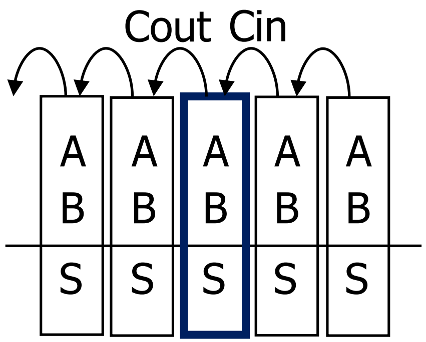
$$317 + 422 = (100111101)_2 + (110100110)_2$$

```
  1 0 0 1 1 1 1 0 0
   100111101
   110100110
 ------------
  1011100011
```



# 1-bit binary adder

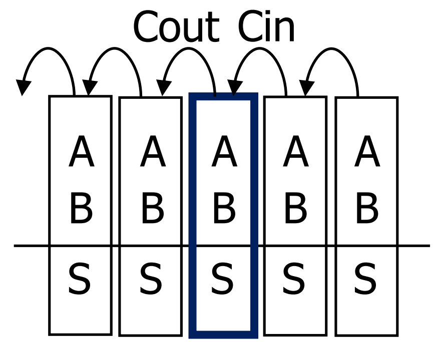
- **Inputs:** A, B, Carry-in
- **Outputs:** Sum, Carry-out




| A | B | Cin | Cout | S |
|---|---|---|---|---|
| 0 | 0 | 0 | | |
| 0 | 0 | 1 | | |
| 0 | 1 | 0 | | |
| 0 | 1 | 1 | | |
| 1 | 0 | 0 | | |
| 1 | 0 | 1 | | |
| 1 | 1 | 0 | | |
| 1 | 1 | 1 | | |

# 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

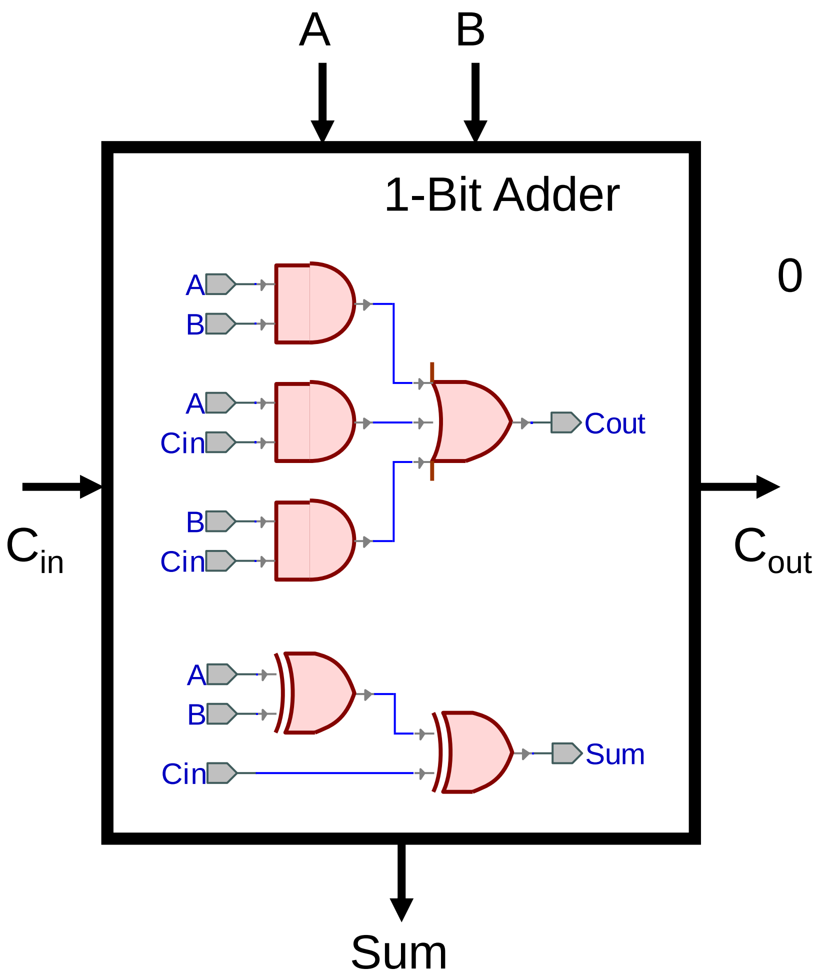
$$
\begin{aligned}
\text{Cout} &= A' \, B \, \text{Cin} + A \, B' \, \text{Cin} + A \, B \, \text{Cin}' + A \, B \, \text{Cin} \\
&= A' \, B \, \text{Cin} + A \, B' \, \text{Cin} + A \, B \, \text{Cin}' + \boxed{A \, B \, \text{Cin} + A \, B \, \text{Cin}} \\
&= A' \, B \, \text{Cin} + A \, B \, \text{Cin} + A \, B' \, \text{Cin} + A \, B \, \text{Cin}' + A \, B \, \text{Cin} \\
&= (A' + A) \, B \, \text{Cin} + A \, B' \, \text{Cin} + A \, B \, \text{Cin}' + A \, B \, \text{Cin} \\
&= (1) \, B \, \text{Cin} + A \, B' \, \text{Cin} + A \, B \, \text{Cin}' + A \, B \, \text{Cin} \\
&= B \, \text{Cin} + A \, B' \, \text{Cin} + A \, B \, \text{Cin}' + \boxed{A \, B \, \text{Cin} + A \, B \, \text{Cin}} \\
&= B \, \text{Cin} + A \, B' \, \text{Cin} + A \, B \, \text{Cin} + A \, B \, \text{Cin}' + A \, B \, \text{Cin} \\
&= B \, \text{Cin} + A \, (B' + B) \, \text{Cin} + A \, B \, \text{Cin}' + A \, B \, \text{Cin} \\
&= B \, \text{Cin} + A \, (1) \, \text{Cin} + A \, B \, \text{Cin}' + A \, B \, \text{Cin} \\
&= B \, \text{Cin} + A \, \text{Cin} + A \, B \, (\text{Cin}' + \text{Cin}) \\
&= B \, \text{Cin} + A \, \text{Cin} + A \, B \, (1) \\
&= B \, \text{Cin} + A \, \text{Cin} + A \, B
\end{aligned}
$$

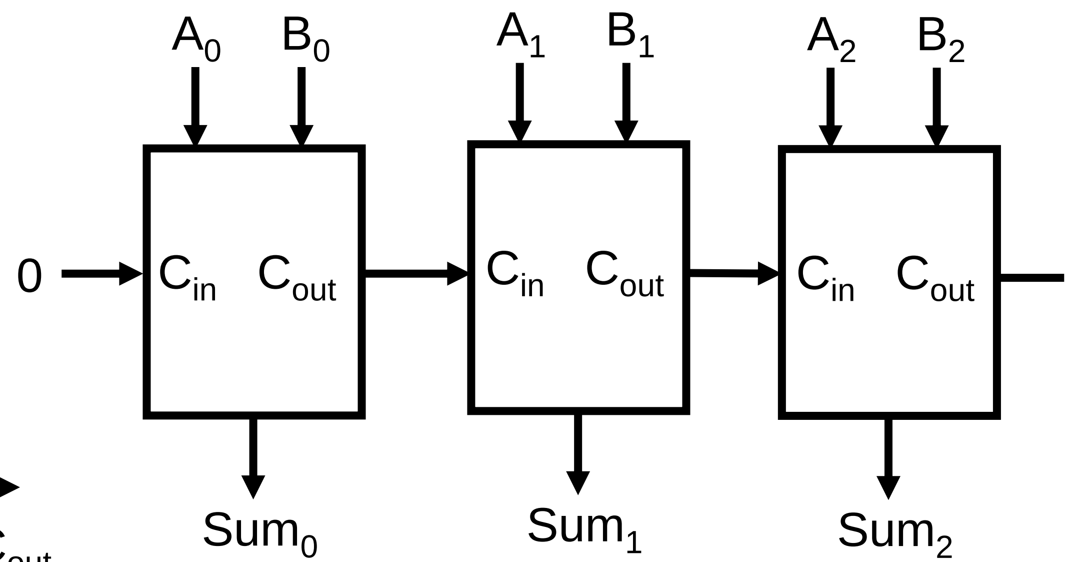
adding extra terms creates new factoring opportunities

Spring 2015

# Lecture 5: Canonical forms and predicate logic

# 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

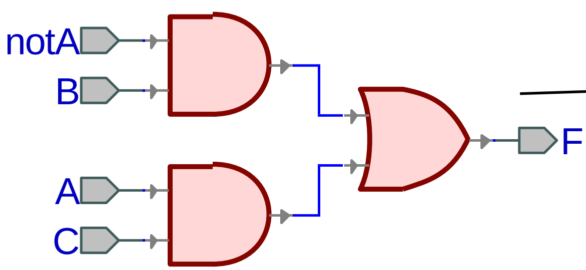
| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

1

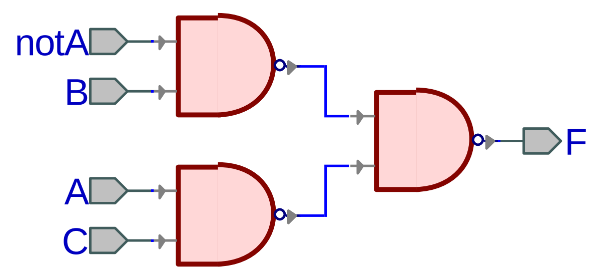
2

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

3




4



# 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

F = *001* *011* *101* *110* *111*

$F = A'B'C + A'BC + AB'C + ABC' + ABC$

| A | B | C | F | F′ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 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 | 0 |

$F = A' + B + C$

# 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

F = *000* *010* *100*

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

| A | B | C | F | F′ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 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 | 0 |

$A'B'C' + A'BC'$

$+ AB'C' = F'$ ✓

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:

- $(F')' = (A'B'C' + A'BC' + AB'C')'$
- $F = (A + B + C)(A + B' + C)(A' + B + C)$

$= (A+B+C)(A+B'+C)(A'+B+C)$

# 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

$$\begin{aligned} F(A, B, C) &= (A + B + C)\,(A + B' + C)\,(A' + B + C) \\ &= (A + B + C)\,(A + B' + C) \\ &\quad (A + B + C)\,(A' + B + C) \\ &= (A + C)\,(B + C) \end{aligned}$$

- Propositional Logic
  - If Pikachu doesn't wear pants, then he flies on Bieber's jet unless Taylor is feeling lonely.

- Predicate Logic
  - If $x$, $y$, and $z$ are positive integers, then $x^3 + y^3 \neq z^3$.

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., positive whole 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

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

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)

# 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)

"x+y=z"

~~y=x+z~~

- $\forall x\ \exists y$ Greater (y, x) T
- $\forall x\ \exists y$ Greater (x, y) F

  x=1
- $\forall x\ \exists y$ (Greater(y, x) $\wedge$ Prime(y)) T
- $\forall x$ (Prime(x) $\rightarrow$ (Equal(x, 2) $\vee$ Odd(x)) T

  5 7
- $\exists x\ \exists y$ (Sum(x, 2, y) $\wedge$ Prime(x) $\wedge$ Prime(y))

  3 = 1·1·1·1·1 3 = 1·3

# statements with quantifiers

prev: now:

- $\forall x \, \exists y$ Greater (y, x) T T
- $\forall x \, \exists y$ Greater (x, y) F T

Domain:
**All integers**

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

Domain of quantifiers is important!

# English to predicate logic

- "Red cats like tofu"

$\forall x\,(Cat(x) \wedge Red(x) \rightarrow LikesTofu(x))$

- "Some red cats don't like tofu"

$\exists x\,(Cat(x) \wedge Red(x) \wedge \neg LikesTofu(x))$

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

Domain = cats

Domain = carbon things

Domain = mammals

# 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"
  - What is $\neg\forall x$ 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!
  - No.
- How about $\exists x$ $\neg$PurpleFruit(x)?
  - "There is a fruit that isn't purple"
  - Yes.

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) $\equiv \exists y \text{ Greater}(y, x)$

$\equiv \exists z \text{ Greater}(z, x)$

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

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

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