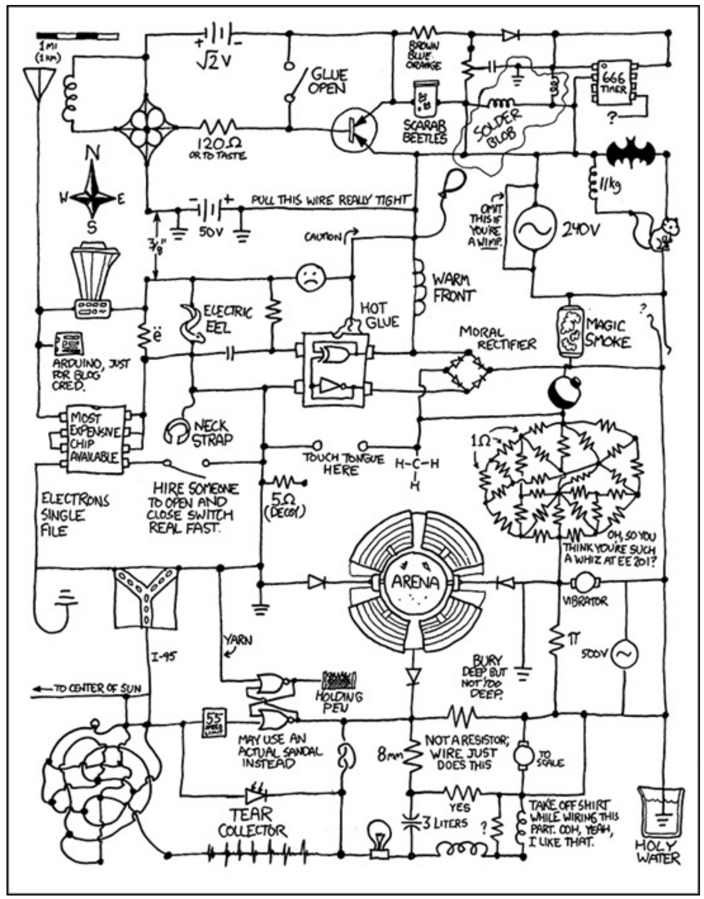
CSE 311: Foundations of Computing

Lecture 5: DNF, CNF and Predicate Logic



Warm-up Exercise

- Create a Boolean Algebra expression for C below in terms of the variables $\boldsymbol{a}$ and $\boldsymbol{b}$

$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{C(a,b)}$
1	1	0
1	0	1
0	1	1
0	0	0

$$ab' + a'b$$

Warm-up Exercise

- Create a Boolean Algebra expression for “c” below in terms of the variables a and b

$$c = ab' + a'b$$

- Draw this as a circuit (using AND, OR, NOT)

Combinational Logic Example

```
case SUNDAY or MONDAY:
    return isLecture ? 3 : 1;
case TUESDAY or WEDNESDAY:
    return isLecture ? 2 : 1;
case THURSDAY:
    return isLecture ? 1 : 1;
case FRIDAY:
    return isLecture ? 1 : 0;
case SATURDAY:
    return isLecture ? 0 : 0;
```

Weekday		isLecture	c_0	c_1	c_2	c_3
SUN	000	0	0	1	0	0
SUN	000	1	0	0	0	1
MON	001	0	0	1	0	0
MON	001	1	0	0	0	1
TUE	010	0	0	1	0	0
TUE	010	1	0	0	1	0
WED	011	0	0	1	0	0
WED	011	1	0	0	1	0
THU	100	-	0	1	0	0
FRI	101	0	1	0	0	0
FRI	101	1	0	1	0	0
SAT	110	-	1	0	0	0
-	111	-	1	0	0	0

Truth Table to Logic (Part 3)

	$d_2d_1d_0$	L	c_0	c_1	c_2	c_3
SUN	000	0	0	1	0	0
SUN	000	1	0	0	0	1
MON	001	0	0	1	0	0
MON	001	1	0	0	0	1
TUE	010	0	0	1	0	0
TUE	010	1	0	0	1	0
WED	011	0	0	1	0	0
WED	011	1	0	0	1	0
THU	100	-	0	1	0	0
FRI	101	0	1	0	0	0
FRI	101	1	0	1	0	0
SAT	110	-	1	0	0	0
-	111	-	1	0	0	0

Now, we do c_1:

$$c_3 = d_2' \cdot d_1' \cdot d_0' \cdot L + d_2' \cdot d_1' \cdot d_0 \cdot L$$

$$c_2 = d_2' \cdot d_1 \cdot d_0' \cdot L + d_2' \cdot d_1 \cdot d_0 \cdot L$$

Truth Table to Logic (Part 3)

	$d_2d_1d_0$	L	c_0	c_1	c_2	c_3
SUN	000	0	0	1	0	0
SUN	000	1	0	0	0	1
MON	001	0	0	1	0	0
MON	001	1	0	0	0	1
TUE	010	0	0	1	0	0
TUE	010	1	0	0	1	0
WED	011	0	0	1	0	0
WED	011	1	0	0	1	0
THU	100	-	0	1	0	0
FRI	101	0	1	0	0	0
FRI	101	1	0	1	0	0
SAT	110	-	1	0	0	0
-	111	-	1	0	0	0

Now, we do c_1:

$d_2' \cdot d_1' \cdot d_0' \cdot L'$

$d_2' \cdot d_1' \cdot d_0 \cdot L'$

$d_2' \cdot d_1 \cdot d_0' \cdot L'$

$d_2' \cdot d_1 \cdot d_0 \cdot L'$

???

$d_2 \cdot d_1' \cdot d_0 \cdot L$

$c_3 = d_2' \cdot d_1' \cdot d_0' \cdot L + d_2' \cdot d_1' \cdot d_0 \cdot L$

$c_2 = d_2' \cdot d_1 \cdot d_0' \cdot L + d_2' \cdot d_1 \cdot d_0 \cdot L$

Truth Table to Logic (Part 3)

	$d_2d_1d_0$	L	c_0	c_1	c_2	c_3
SUN	000	0	0	1	0	0
SUN	000	1	0	0	0	1
MON	001	0	0	1	0	0
MON	001	1	0	0	0	1
TUE	010	0	0	1	0	0
TUE	010	1	0	0	1	0
WED	011	0	0	1	0	0
WED	011	1	0	0	1	0
THU	100	-	0	1	0	0
FRI	101	0	1	0	0	0
FRI	101	1	0	1	0	0
SAT	110	-	1	0	0	0
-	111	-	1	0	0	0

Now, we do c_1:

$d_2' \cdot d_1' \cdot d_0' \cdot L'$

$d_2' \cdot d_1' \cdot d_0 \cdot L'$

$d_2' \cdot d_1 \cdot d_0' \cdot L'$

$d_2' \cdot d_1 \cdot d_0 \cdot L'$

$d_2 \cdot d_1' \cdot d_0'$

$d_2 \cdot d_1' \cdot d_0 \cdot L$

No matter what L is, we always say it's 1. So, we don't need L in the expression.

$$c_3 = d_2' \cdot d_1' \cdot d_0' \cdot L + d_2' \cdot d_1' \cdot d_0 \cdot L$$
$$c_2 = d_2' \cdot d_1 \cdot d_0' \cdot L + d_2' \cdot d_1 \cdot d_0 \cdot L$$

Truth Table to Logic (Part 4)

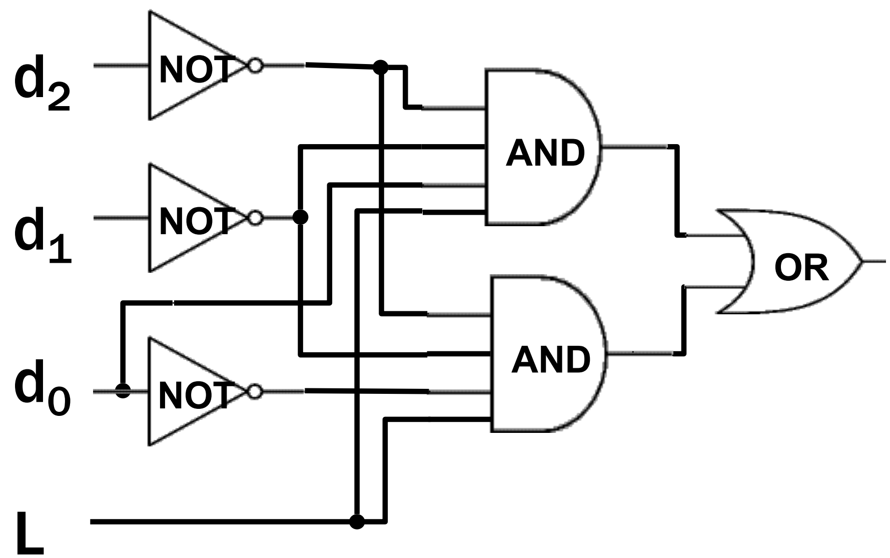
$c_0 = d_2 \cdot d_1' \cdot d_0 \cdot L' + d_2 \cdot d_1 \cdot d_0' + d_2 \cdot d_1 \cdot d_0$

$c_1 = d_2' \cdot d_1' \cdot d_0' \cdot L' + d_2' \cdot d_1' \cdot d_0 \cdot L' + d_2' \cdot d_1 \cdot d_0' \cdot L' + d_2' \cdot d_1 \cdot d_0 \cdot L' + d_2 \cdot d_1' \cdot d_0' + d_2 \cdot d_1' \cdot d_0 \cdot L$

$c_2 = d_2' \cdot d_1 \cdot d_0' \cdot L + d_2' \cdot d_1 \cdot d_0 \cdot L$

$c_3 = d_2' \cdot d_1' \cdot d_0' \cdot L + d_2' \cdot d_1' \cdot d_0 \cdot L$

Here's c_3 as a circuit:



Important Corollaries of this Construction

- **$\neg$, $\wedge$, $\vee$ can implement any Boolean function**

  we didn't need any others to do this

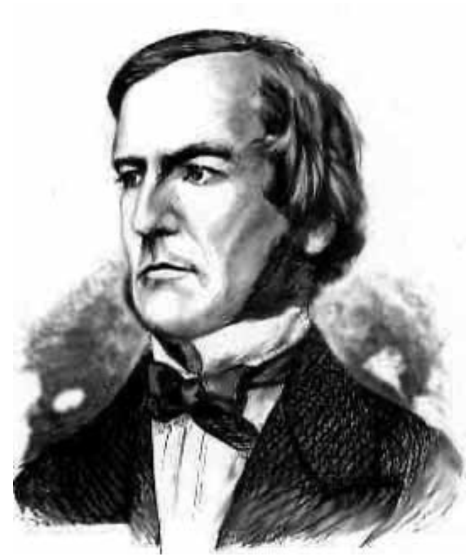
- **Actually, just $\neg$, $\wedge$ (or $\neg$, $\vee$) are enough**

  follows by De Morgan's laws

- **Actually, just NAND (or NOR)**

Boolean Algebra

- Usual notation used in circuit design
- Boolean algebra
  - a set of elements B containing {0, 1}
  - binary operations { + , • }
  - and a unary operation { ’ }
  - such that the following axioms hold:



For any a, b, c in B:

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

Simplification using Boolean Algebra

uniting:

10. $a \bullet b + a \bullet b' = a$

10D. $(a + b) \bullet (a + b') = a$

absorption:

11. $a + a \bullet b = a$

11D. $a \bullet (a + b) = a$

12. $(a + b') \bullet b = a \bullet b$

12D. $(a \bullet b') + b = a + b$

factoring:

13. $(a + b) \bullet (a' + c) = a \bullet c + a' \bullet b$

13D. $a \bullet b + a' \bullet c = (a + c) \bullet (a' + b)$

consensus:

14. $(a \bullet b) + (b \bullet c) + (a' \bullet c) = a \bullet b + a' \bullet c$

14D. $(a + b) \bullet (b + c) \bullet (a' + c) = (a + b) \bullet (a' + c)$

de Morgan's:

15. $(a + b + \ldots)' = a' \bullet b' \bullet \ldots$

15D. $(a \bullet b \bullet \ldots)' = a' + b' + \ldots$

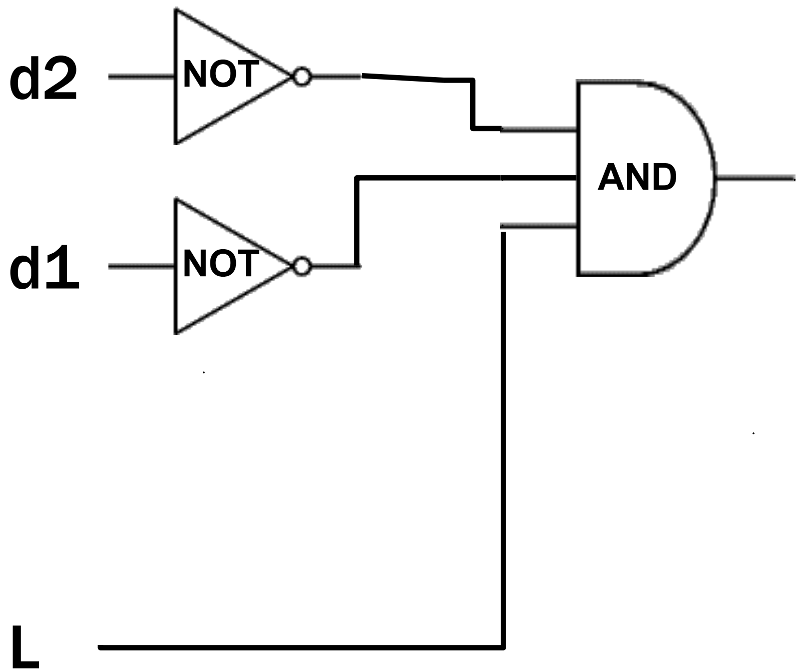
Simplifying using Boolean Algebra

c3 = d2’•d1’•d0’•L + d2’•d1’•d0•L

= d2’•d1’•(d0’ + d0)•L

= d2’•d1’•1•L

= d2’•d1’•L



1-bit Binary Adder

A	$0 + 0 = 0$ (with $C_{OUT} = 0$)
$\underline{+\ B}$	$0 + 1 = 1$ (with $C_{OUT} = 0$)
S	$1 + 0 = 1$ (with $C_{OUT} = 0$)
(C_{OUT})	$1 + 1 = 0$ (with $C_{OUT} = 1$)

1-bit Binary Adder

A	$0 + 0 = 0$ (with $C_{OUT} = 0$)
<u>+ B</u>	$0 + 1 = 1$ (with $C_{OUT} = 0$)
S	$1 + 0 = 1$ (with $C_{OUT} = 0$)
(C_{OUT})	$1 + 1 = 0$ (with $C_{OUT} = 1$)

Idea: chain these together to add larger numbers

Recall from elementary school:

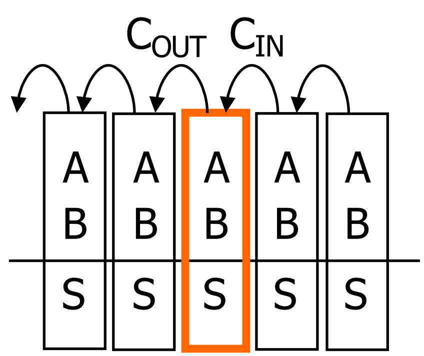
$$\begin{array}{r} 2\ 4\ 8 \\ \underline{+\ 3\ 7\ 5} \end{array}$$

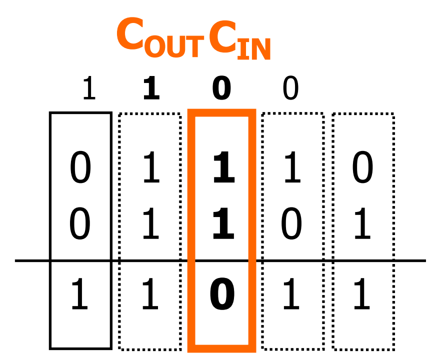
1-bit Binary Adder

A	$0 + 0 = 0$ (with $C_{OUT} = 0$)
+ B	$0 + 1 = 1$ (with $C_{OUT} = 0$)
S	$1 + 0 = 1$ (with $C_{OUT} = 0$)
(C_{OUT})	$1 + 1 = 0$ (with $C_{OUT} = 1$)

Idea: These are chained together with a carry-in

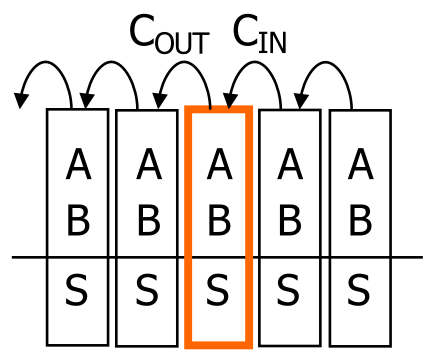
(C_{IN})
A
+ B
S
(C_{OUT})





1-bit Binary Adder

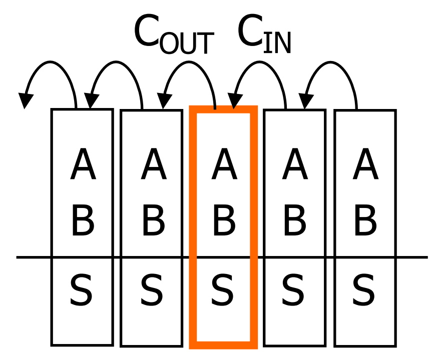
- **Inputs:** A, B, Carry-in
- **Outputs:** Sum, Carry-out



A	B	C_{IN}	C_{OUT}	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

1-bit Binary Adder

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



A	B	C_{IN}	C_{OUT}	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

Derive an expression for S

$A' \cdot B' \cdot C_{IN}$

$A' \cdot B \cdot C_{IN}'$

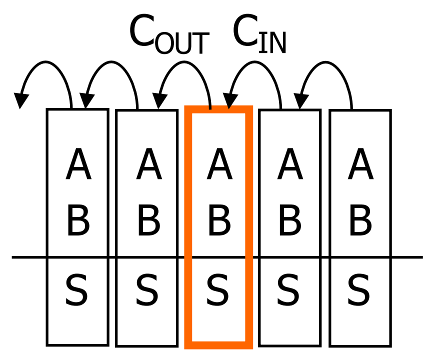
$A \cdot B' \cdot C_{IN}'$

$A \cdot B \cdot C_{IN}$

$$S = A' \cdot B' \cdot C_{IN} + A' \cdot B \cdot C_{IN}' + A \cdot B' \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

1-bit Binary Adder

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



A	B	C_{IN}	C_{OUT}	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

Derive an expression for C_{OUT}

$A' \cdot B \cdot C_{IN}$

$A \cdot B' \cdot C_{IN}$

$A \cdot B \cdot C_{IN}'$

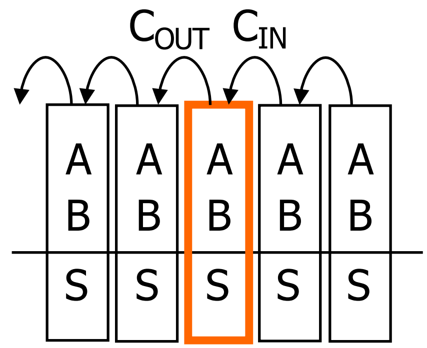
$A \cdot B \cdot C_{IN}$

$$C_{OUT} = A' \cdot B \cdot C_{IN} + A \cdot B' \cdot C_{IN} + A \cdot B \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

$$S = A' \cdot B' \cdot C_{IN} + A' \cdot B \cdot C_{IN}' + A \cdot B' \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

1-bit Binary Adder

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



A	B	C_{IN}	C_{OUT}	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

$$S = A' \cdot B' \cdot C_{IN} + A' \cdot B \cdot C_{IN}' + A \cdot B' \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

$$C_{OUT} = A' \cdot B \cdot C_{IN} + A \cdot B' \cdot C_{IN} + A \cdot B \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

Apply Theorems to Simplify Expressions

The theorems of Boolean algebra can simplify expressions

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

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

Apply Theorems to Simplify Expressions

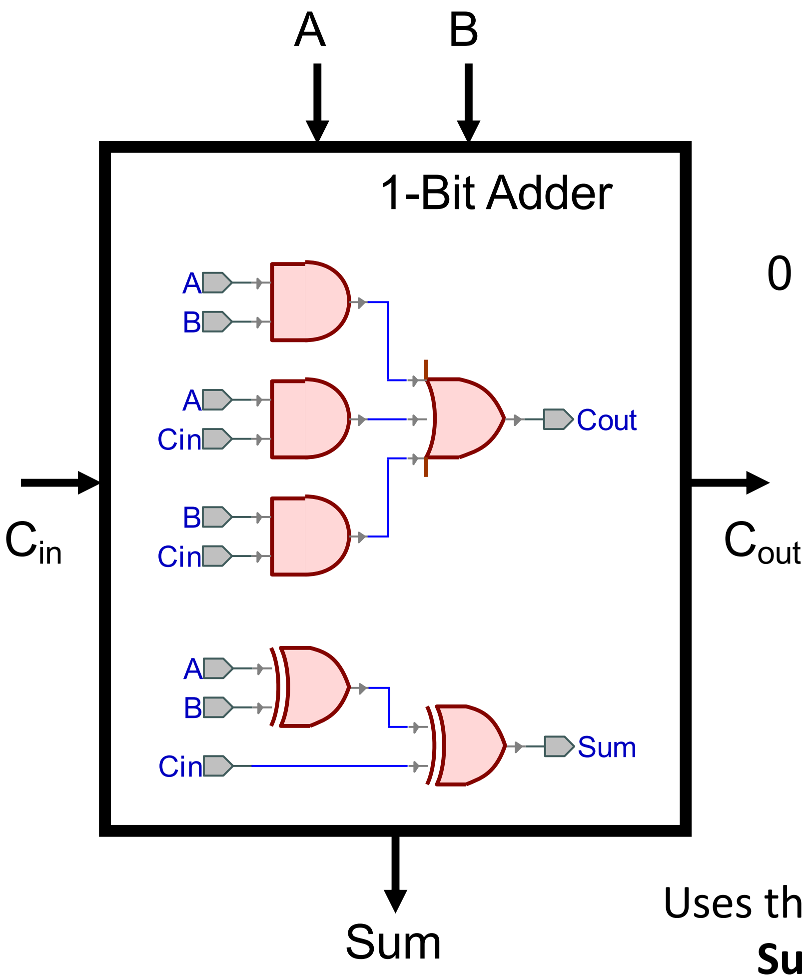
The theorems of Boolean algebra can simplify expressions

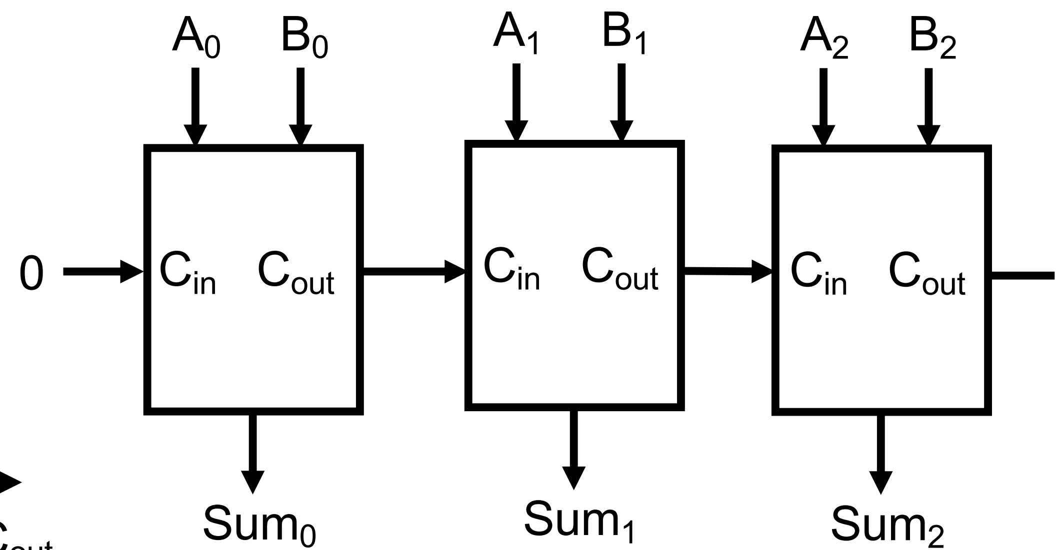
- **e.g., full adder's carry-out function**

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

adding extra terms
creates new factoring
opportunities

A 2-bit Ripple-Carry Adder





Uses the fact that

$$\mathbf{Sum} = A' \cdot B' \cdot C_{IN} + A' \cdot B \cdot C_{IN}' + A \cdot B' \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

is equivalent to $\mathbf{Sum} = (A \oplus B) \oplus C_{IN}$

Mapping Truth Tables to Logic Gates

Given a truth table:

1. **Write the output in a table**
2. **Write the Boolean expression**
3. **Minimize the Boolean expression**
4. **Draw as gates**
5. **Map to available gates**

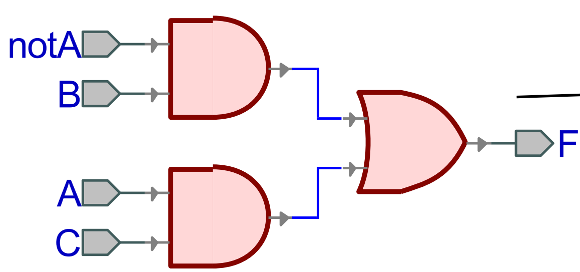
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

2

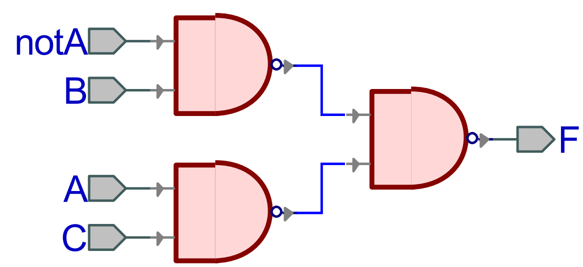
$$F = A'BC'+A'BC+AB'C+ABC$$
$$= A'B(C'+C)+AC(B'+B)$$
$$= A'B+AC$$

3

4



5



Canonical Forms

- Truth table is the **unique signature** of a 0/1 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 produce the same expression

Sum-of-Products Canonical Form

- AKA **Disjunctive Normal Form (DNF)**
- AKA **Minterm Expansion**

3 Add the minterms together

F= A′B′C + A′BC + AB′C + ABC′ + ABC

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

1 Read T rows off truth table	2 Convert to Boolean Algebra
001 →	A′B′C
011 →	A′BC
101 →	AB′C
110 →	ABC′
111 →	ABC

F

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

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

Product-of-Sums Canonical Form

- AKA **Conjunctive Normal Form (CNF)**
- AKA **Maxterm Expansion**

4 Multiply the maxterms together

F =

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

1 Read F rows off truth table

2 Negate all bits

3 Convert to Boolean Algebra

F

Product-of-Sums Canonical Form

- AKA **Conjunctive Normal Form (CNF)**
- AKA **Maxterm Expansion**

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

1 Read F rows off truth table

2 Negate all bits

3 Convert to Boolean Algebra

$000 \rightarrow 111 \rightarrow A + B + C$

$010 \rightarrow 101 \rightarrow A + B' + C$

$100 \rightarrow 011 \rightarrow A' + B + C$

F

4 Multiply the maxterms together

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

Product-of-Sums: Why does this procedure work?

Useful Facts:

- We know (F')' = F
- We know how to get a **minterm** expansion for F'

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

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

Product-of-Sums: Why does this procedure work?

Useful Facts:

- We know (F')' = F
- We know how to get a **minterm** expansion for F'

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

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

Taking the complement of both sides…

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

And using DeMorgan/Comp.…

$$F = (A'B'C')' \ (A'BC')' \ (AB'C')'$$

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

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 $\neq$ minimal form

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