CSE 311

Foundations of Computing I

Pre-Lecture Problem

Do it! Do it now! What are you waiting for? ☺

Use Logical Equivalences to show

$$p \wedge \big((p \rightarrow q) \vee (p \rightarrow r)\big) \equiv (r \vee q) \wedge p$$

Identity

$p \wedge \mathsf{T} \equiv p$

$p \vee \mathsf{F} \equiv p$

Domination

$p \vee \mathsf{T} \equiv \mathsf{T}$

$p \wedge \mathsf{F} \equiv \mathsf{F}$

Idempotency

$p \vee p \equiv p$

$p \wedge p \equiv p$

Commutativity

$p \vee q \equiv q \vee p$

$p \wedge q \equiv q \wedge p$

Associativity

$(p \vee q) \vee r \equiv p \vee (q \vee r)$

$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$

Distributivity

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

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

Absorption

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

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

Negation

$p \vee \neg p \equiv \mathsf{T}$

$p \wedge \neg p \equiv \mathsf{F}$

DeMorgan's Laws

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

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

Double Negation

$\neg\neg p \equiv p$

Law of Implication

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

Contrapositive

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

A Combinational Logic Example

Sessions of Class:

We would like to compute the number of lectures or quiz sections remaining *at the start* of a given day of the week.

- **Inputs:** Day of the Week, Lecture/Section flag
- **Output:** Number of sessions left

Examples: Input: (Wednesday, Lecture) Output: 2

Input: (Monday, Section) Output: 1

Implementation in Software

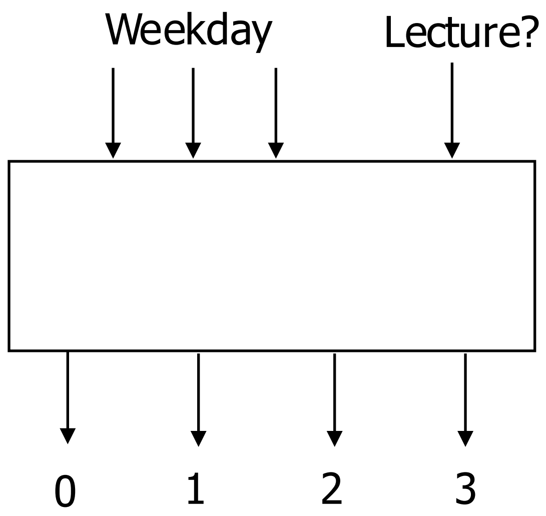
```
public int classesLeftInMorning(weekday, lecture_flag) {
    switch (weekday) {
        case SUNDAY:
        case MONDAY:
            return lecture_flag ? 3 : 1;
        case TUESDAY:
        case WEDNESDAY:
            return lecture_flag ? 2 : 1;
        case THURSDAY:
            return lecture_flag ? 1 : 1;
        case FRIDAY:
            return lecture_flag ? 1 : 0;
        case SATURDAY:
            return lecture_flag ? 0 : 0;
    }
}
```

asks like a series of if statements.
IF weekday == sunday: do this
EXCEPT as soon as switch is turned on, ALL cases activate
until method is broken (by break statement or return statement)

Implementation with Combinational Logic

Encoding:

- How many bits for each input/output?
- Binary number for weekday
- One bit for each possible output



Defining Our Inputs!

Weekday Input:

- Binary number for weekday
- Sunday = 0, Monday = 1, …
- We care about these in binary:

Weekday	Number	Binary
Sunday	0	$(000)_2$
Monday	1	$(001)_2$
Tuesday	2	$(010)_2$
Wednesday	3	$(011)_2$
Thursday	4	$(100)_2$
Friday	5	$(101)_2$
Saturday	6	$(110)_2$

Converting to a Truth Table!

1 = lecture
0 = quiz
- = same for both

number of classes of that type (0,1,2,3)

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

Weekday		Lecture?	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				
MON	001	1				
TUE	010	0				
TUE	010	1				
WED	011	0				
WED	011	1				
THU	100	$\boxed{-}$	0	1	0	0
FRI	101	0				
FRI	101	1				
SAT	110	-				
-	111	-				

Converting to a Truth Table!

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

Weekday		Lecture?	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 1)

	$d_2d_1d_0$	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

Let's begin by finding an expression for $\mathbf{c_3}$. To do this, we look at the rows where $\mathbf{c_3} = \mathbf{1}$ (true).

Truth Table to Logic (Part 1)

	$d_2d_1d_0$	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

e_1

DAY == SUN && L == 1

DAY == MON && L == 1

e_2

$e_1 \vee e_2$

Truth Table to Logic (Part 1)

	$d_2d_1d_0$	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

$d_2d_1d_0$ == 000 && L == 1

$d_2d_1d_0$ == 001 && L == 1

Substituting DAY for the binary representation.

Truth Table to Logic (Part 1)

	$d_2d_1d_0$	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

d_2 d_1 d_0 L

$d_2 == 0$ && $d_1 == 0$ && $d_0 == 0$ && $L == 1$

$d_2 == 0$ && $d_1 == 0$ && $d_0 == 1$ && $L == 1$

Splitting up the bits of the day;
so, we can write a formula.

Truth Table to Logic (Part 1)

	$d_2d_1d_0$	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

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

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

Replacing with Boolean Algebra...

Truth Table to Logic (Part 1)

	$d_2d_1d_0$	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

$d_2' \bullet d_1' \bullet d_0' \bullet L$

$d_2' \bullet d_1' \bullet d_0 \bullet L$

Either situation causes c_3 to be true. So, we "or" them.

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

Truth Table to Logic (Part 2)

	$d_2d_1d_0$	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

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

d2 *

Now, we do 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	$d_2' \cdot d_1' \cdot d_0' \cdot L'$
SUN	000	1	0	0	0	1	
MON	001	0	0	1	0	0	$d_2' \cdot d_1' \cdot d_0 \cdot L'$
MON	001	1	0	0	0	1	
TUE	010	0	0	1	0	0	$d_2' \cdot d_1 \cdot d_0' \cdot L'$
TUE	010	1	0	0	1	0	
WED	011	0	0	1	0	0	$d_2' \cdot d_1 \cdot d_0 \cdot L'$
WED	011	1	0	0	1	0	
THU	100	-	0	1	0	0	??? $(d_2 \cdot d_1' \cdot d_0)' \cdot \cancel{(L')}$
FRI	101	0	1	0	0	0	
FRI	101	1	0	1	0	0	$d_2 \cdot d_1' \cdot d_0 \cdot L$
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:

$d_2' \bullet d_1' \bullet d_0' \bullet L'$

$d_2' \bullet d_1' \bullet d_0 \bullet L'$

$d_2' \bullet d_1 \bullet d_0' \bullet L'$

$d_2' \bullet d_1 \bullet d_0 \bullet L'$

$d_2 \bullet d_1' \bullet d_0'$

$d_2 \bullet d_1' \bullet d_0 \bullet L$

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

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

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

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	$d_2' \bullet d_1' \bullet d_0' \bullet L'$
SUN	000	1	0	0	0	1	
MON	001	0	0	1	0	0	$d_2' \bullet d_1' \bullet d_0 \bullet L'$
MON	001	1	0	0	0	1	
TUE	010	0	0	1	0	0	$d_2' \bullet d_1 \bullet d_0' \bullet L'$
TUE	010	1	0	0	1	0	
WED	011	0	0	1	0	0	$d_2' \bullet d_1 \bullet d_0 \bullet L'$
WED	011	1	0	0	1	0	
THU	100	-	0	1	0	0	$d_2 \bullet d_1' \bullet d_0'$
FRI	101	0	1	0	0	0	
FRI	101	1	0	1	0	0	$d_2 \bullet d_1' \bullet d_0 \bullet L$
SAT	110	-	1	0	0	0	
-	111	-	1	0	0	0	

Now, we do c_1:

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

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

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

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

Truth Table to Logic (Part 4)

	$d_2d_1d_0$	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

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

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

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

Finally, we do c_0:

$d_2 \bullet d_1' \bullet d_0 \bullet L'$

$d_2 \bullet d_1 \bullet d_0'$

$d_2 \bullet d_1 \bullet d_0$

Truth Table to Logic (Part 4)

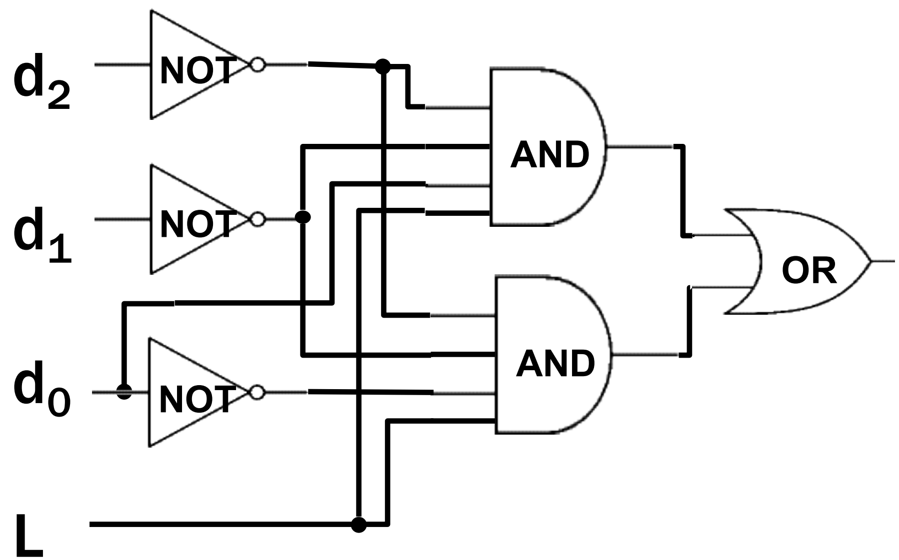
$c_0 = d_2 \bullet d_1' \bullet d_0 \bullet L' + d_2 \bullet d_1 \bullet d_0' + d_2 \bullet d_1 \bullet d_0$

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

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

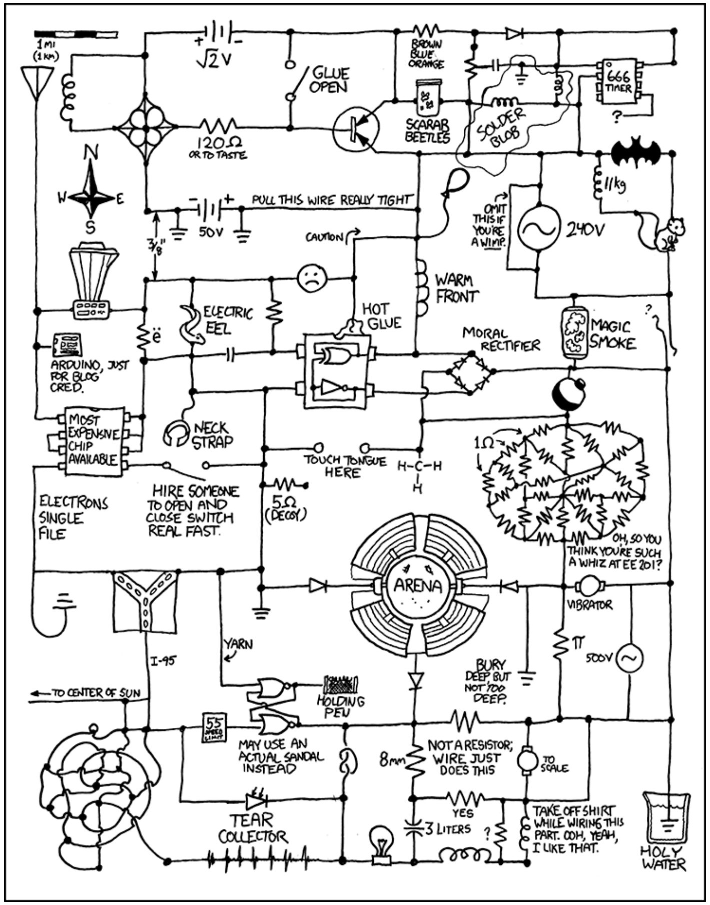
$c_3 = d_2' \bullet d_1' \bullet d_0' \bullet L + d_2' \bullet d_1' \bullet d_0 \bullet L$

Here's c_3 as a circuit:



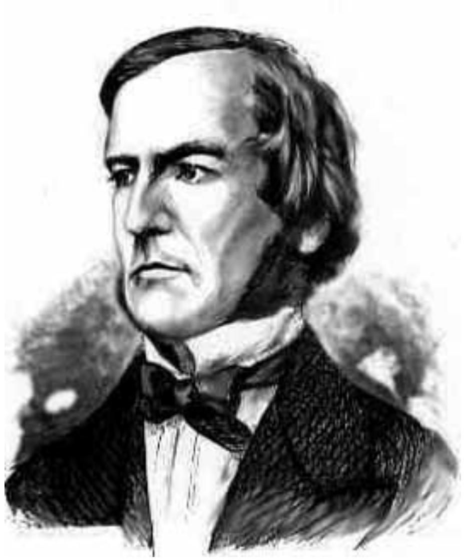
CSE 311: Foundations of Computing

Lecture 4: Boolean Algebra, Circuits, Canonical Forms



Boolean Algebra

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



1. the set B contains at least two elements: 0, 1

For any a, b, c in B:

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

Axioms and Theorems of Boolean Algebra

identity:

1. $X + 0 = X$ 1D. $X \cdot 1 = X$

null:

2. $X + 1 = 1$ 2D. $X \cdot 0 = 0$

idempotency:

3. $X + X = X$ 3D. $X \cdot X = X$

involution:

4. $(X')' = X$

complementarity:

5. $X + X' = 1$ 5D. $X \cdot X' = 0$

commutativity:

6. $X + Y = Y + X$ 6D. $X \cdot Y = Y \cdot X$

associativity:

7. $(X + Y) + Z = X + (Y + Z)$ 7D. $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$

distributivity:

8. $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$ 8D. $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$

Axioms and Theorems of Boolean Algebra

uniting:

9. $X \bullet Y + X \bullet Y' = X$

9D. $(X + Y) \bullet (X + Y') = X$

absorption:

10. $X + X \bullet Y = X$

10D. $X \bullet (X + Y) = X$

11. $(X + Y') \bullet Y = X \bullet Y$

11D. $(X \bullet Y') + Y = X + Y$

factoring:

12. $(X + Y) \bullet (X' + Z) = X \bullet Z + X' \bullet Y$

12D. $X \bullet Y + X' \bullet Z = (X + Z) \bullet (X' + Y)$

consensus:

13. $(X \bullet Y) + (Y \bullet Z) + (X' \bullet Z) = X \bullet Y + X' \bullet Z$

13D. $(X + Y) \bullet (Y + Z) \bullet (X' + Z) = (X + Y) \bullet (X' + Z)$

de Morgan's:

14. $(X + Y + \ldots)' = X' \bullet Y' \bullet \ldots$

14D. $(X \bullet Y \bullet \ldots)' = X' + Y' + \ldots$

Proving Theorems (Rewriting)

Using the laws of Boolean Algebra:

prove the theorem: $X \bullet Y + X \bullet Y' = X$

$$X \bullet Y + X \bullet Y' = X \cdot (\cancel{Y + Y'})$$

$$= X \cdot 1$$

$$= X$$

prove the theorem: $X + X \bullet Y = X$

$$X + X \bullet Y =$$

Proving Theorems (Rewriting)

Using the laws of Boolean Algebra:

prove the theorem: $X \bullet Y + X \bullet Y' = X$

distributivity (8)	$X \bullet Y + X \bullet Y' = X \bullet (Y + Y')$
complementarity (5)	$= X \bullet (1)$
identity (1D)	$= X$

prove the theorem: $X + X \bullet Y = X$

identity (1D)	$X + X \bullet Y = X \bullet 1 + X \bullet Y$
distributivity (8)	$= X \bullet (1 + Y)$
uniting (2)	$= X \bullet (1)$
identity (1D)	$= X$

Proving Theorems (Truth Table)

Using complete truth table:

For example, de Morgan's Law:

$(X + Y)' = X' \bullet Y'$

NOR is equivalent to AND with inputs complemented

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

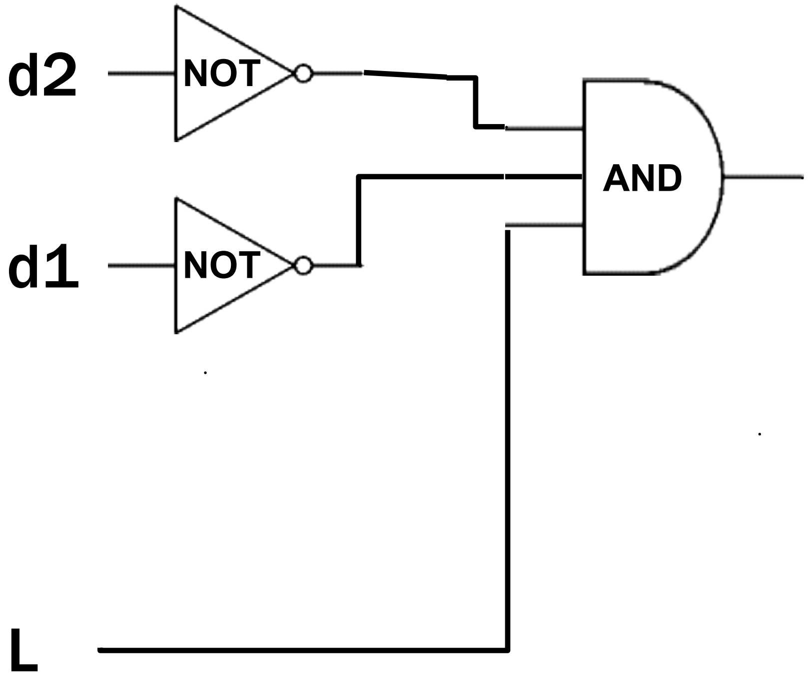
$(X \bullet Y)' = X' + Y'$

NAND is equivalent to OR with inputs complemented

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

Simplifying using Boolean Algebra

$$c3 = (d2' \bullet d1' \bullet d0' \bullet L) + (d2' \bullet d1' \bullet d0 \bullet L)$$
$$= d2' \bullet d1' \bullet (d0' + d0) \bullet L$$
$$= d2' \bullet d1' \bullet (1) \bullet L$$
$$= d2' \bullet d1' \bullet L$$



1-bit Binary Adder

Let's make a circuit that adds three single bit inputs together into a single binary number

				1
A	0	1	1	1
+ B	+ 0	+ 1	+ 0	+ 1
+ C	+ 0	+ 0	+ 1	+ 1
S_C S	0 0	1 0	1 0	1 1

1-bit Binary Adder

Let’s make a circuit that adds three single bit inputs together into a single binary number

A	0	1	1	1
+ B	+ 0	+ 1	+ 0	+ 1
+ C	+ 0	+ 0	+ 1	+ 1
$S_C S$	00	10	10	11

1-bit Binary Adder

- **Inputs:** A, B, C
- **Outputs:** Two-bit Sum

A	B	C	S_C	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

Larger Sum?

1

1 1 0

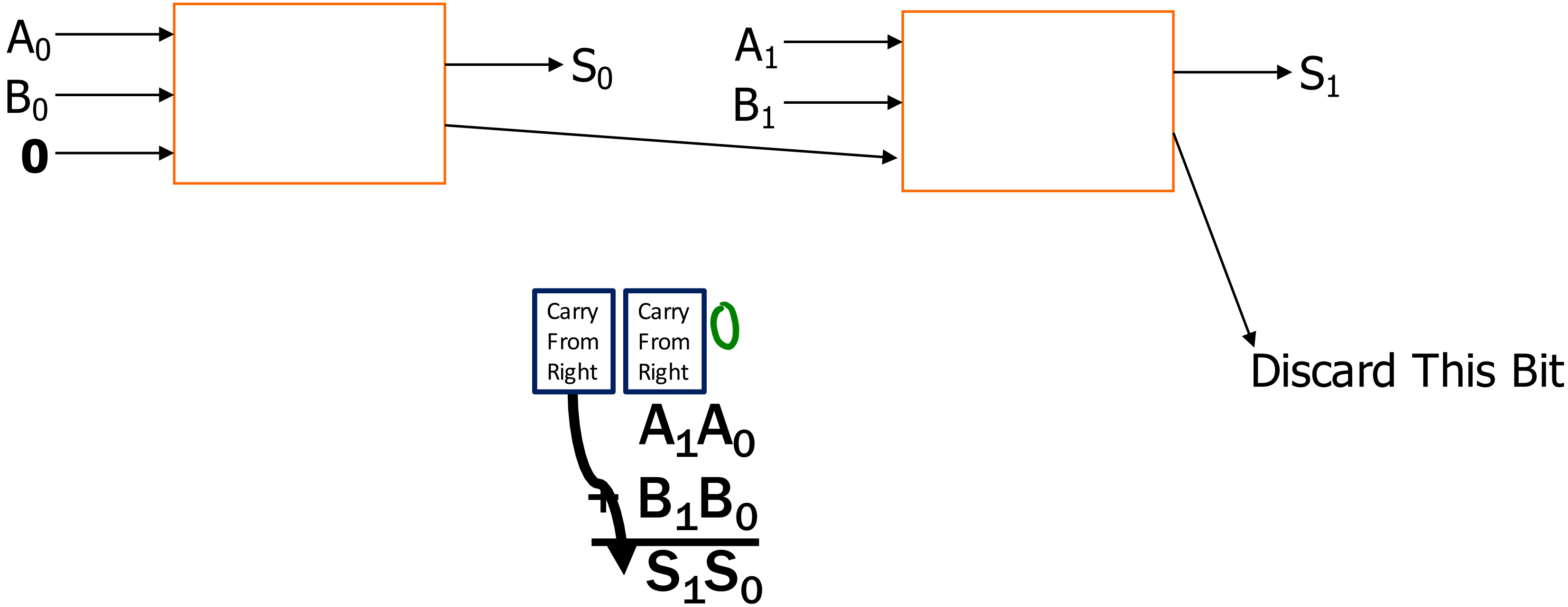
11

+ 01

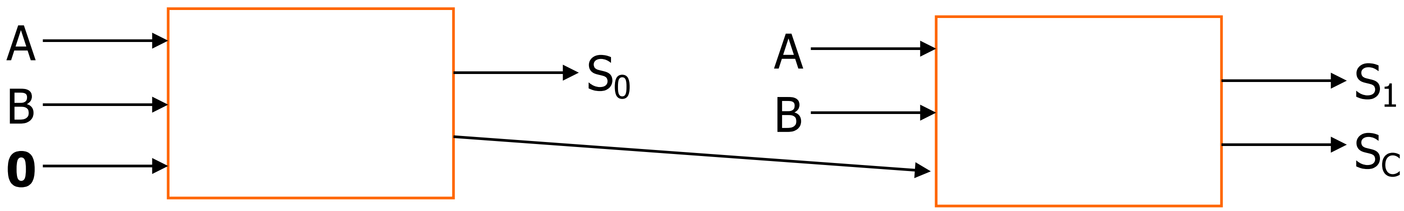
100

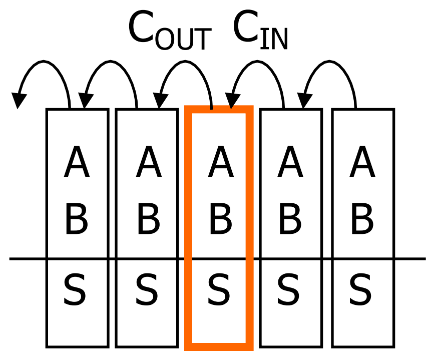
To get a sum of numbers with more bits, we can **combine** two of our original circuit together!

Larger Sum?


A0
B0
0
S0
A1
B1
S1
Discard This Bit
Carry From Right
Carry From Right
0
A1A0
+ B1B0
S1S0

Larger Sum?

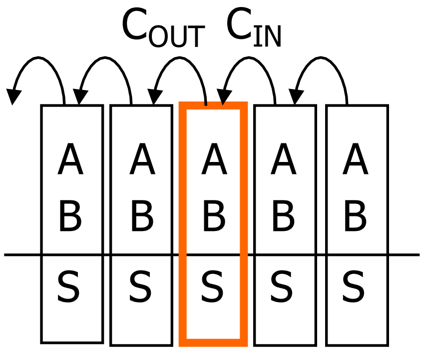




To support this idea, we rename our inputs/outputs.

1-bit Binary Adder

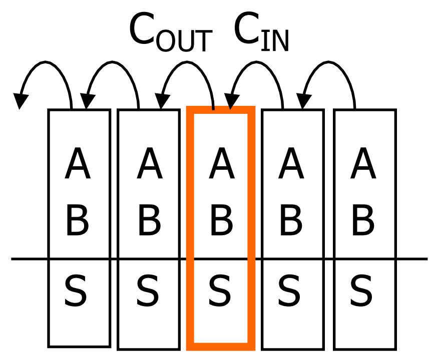
- **Inputs:** A, B, Carry-in
- **Outputs:** Sum, Carry-out



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

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

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

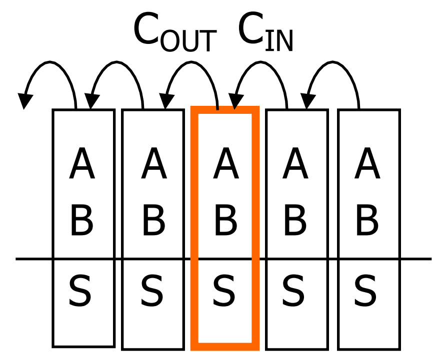
$A \cdot B' \cdot C_{IN}'$

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

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

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}

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

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

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

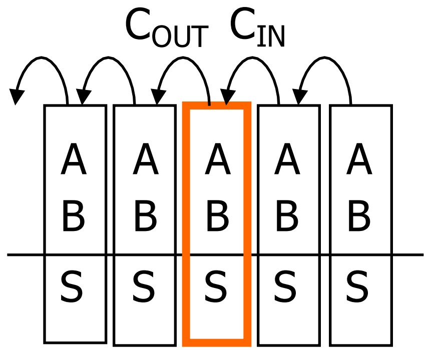
$A \cdot B \cdot C_{IN}$

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

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

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

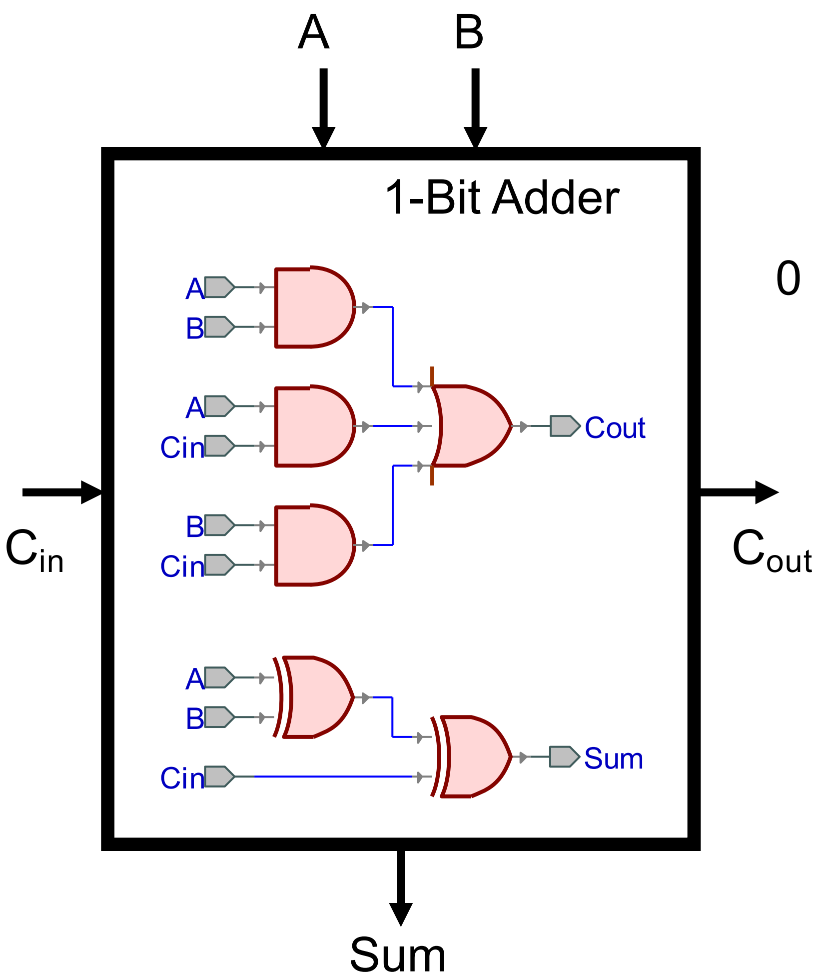
The theorems of Boolean algebra can simplify expressions

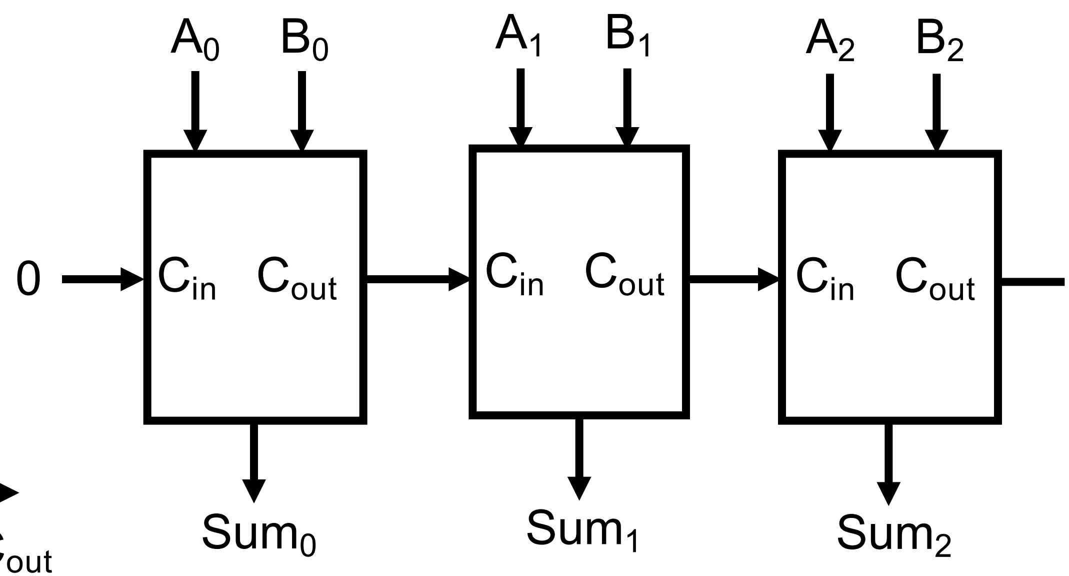
– **e.g., full adder's carry-out function**

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

adding extra terms creates new factoring opportunities

A 2-bit Ripple-Carry Adder





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

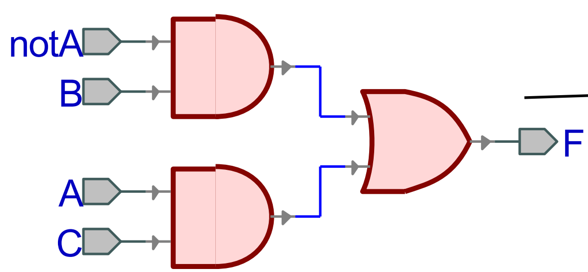
$F = A'BC' + A'BC + AB'C + ABC$

2

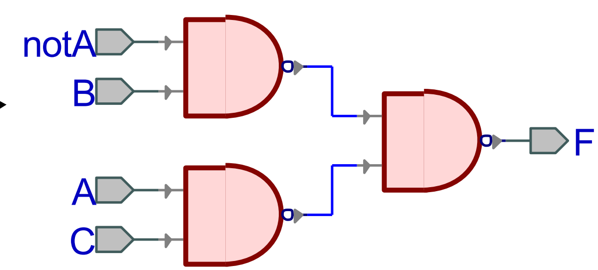
$= A'B(C'+C) + AC(B'+B)$

$= A'B + AC$

3



4



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

- 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

$A + C$

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

$(A \wedge B) \vee C$