2.1 Two-level combinational logic simplification

Simplifying a sum-of-products expression

Logic simplification (also referred to as ***logic minimization***) means to simplify a Boolean expression before converting to a circuit to yield a smaller circuit.

PARTICIPATION ACTIVITY 2.1.1: Simplifying an expression before converting to a circuit.

Animation captions:

1. A system activates a ringer (r = 1) if motion sensed (a = 1) and daylight detected (b = 1) and no clerk (c = 0), or motion and no daylight and no clerk: r = abc' + ab'c'.
2. The expression can be simplified to ac'(b + b') (distributive), then ac'(1) (complement), and finally ac' (identity).
3. The circuit for ac' is much simpler than for abc' + ab'c'.

PARTICIPATION ACTIVITY 2.1.2: Simplifying a sum-of-products expression.

Consider the example above.

1) How many literals exist in the original expression?
   - ○ 3
   - ○ 6

2) If each AND or OR gate input requires two transistors, how many transistors does the original expression's circuit require? Ignore NOT gates.
   - ○ 8
   - ○ 16

3) If each AND or OR gate input requires two transistors, how many transistors does the simplified expression's circuit require? Ignore NOT gates.
   - ○ 4

16

Seeking i(j + j') opportunities

Given a sum-of-products expression, knowing what to simplify can be hard. To make simplification opportunities more obvious, a common algebraic simplification process is to:

- Convert to sum-of-minterms
- Seek i(j + j') opportunities: ij + ij' = i(j + j') = i

PARTICIPATION ACTIVITY 2.1.3: Seeking i(j + j') opportunities.

Animation captions:

1. Given a sum-of-products expression to simplify by hand, like ac + ab'c, a designer may not see any opportunity to simplify.
2. Thus, given an expression, a common first step is to translate to sum-of-minterms form: ac + ab'c = ac(1) + ab'c = ac(b + b') + ab'c = abc + ab'c + ab'c = abc + ab'c.
3. Then, a designer can seek i(j + j') opportunities: abc + ab'c = ac(b + b') = ac(1) = ac. Such simplification may not have been as obvious on the original equation ac + ab'c.

PARTICIPATION ACTIVITY 2.1.4: Seeking i(j + j') simplification opportunities.

Only type the ? part. Type answers as: ab'

1) y = cd + cd'
y = c(?)

Check Show answer

2) y = c(d + d')
y = c(?)

Check Show answer

3) y = c(1)
y = ?

Check Show answer

4) y = efg + ef'g
y = eg(?)

Check Show answer

5) y = cd' + cd
y = c(?)

Check Show answer

6) y = dc + d'c
y = c(?)

Check Show answer

PARTICIPATION ACTIVITY 2.1.5: First translating to sum-of-minterms, then seeking simplification opportunities.

Simplify. Only type the ? part. Type answers as: ab'

1) y = cd + c
y = cd + c(d + ?)

Check Show answer

2) y = cd + c
y = cd + c(d + d')
y = cd + cd + ?

Check Show answer

3) y = cd + cd + cd'
y = ? + cd'

Check Show answer

4) y = cd + cd'
y = c(?)

Check Show answer

5) y = c(d + d')
y = ?

Check Show answer

Algebraic simplification by hand can be hard

The algebraic simplification process can be hard to do by hand.

PARTICIPATION ACTIVITY 2.1.6: Algebraic simplification can be hard to do by hand.

Animation captions:

1. Simplifying algebraically can be hard. First a designer converts to sum-of-minterms form.
2. Next the designer must create i(j + j') opportunities by replicating a term. The need for such replication may not be obvious.
3. Then the designer must apply i(j + j') simplifications. Again, the places where i(j + j') can be applied aren't obvious. And every step is prone to error.

PARTICIPATION ACTIVITY 2.1.7: Simplifying algebraically can be hard.

Consider the example above.

1) What expression came after: ab + a'b + a'b'

- (a + a')b + a'b'
- ab + a'(b + b')
- ab + a'b + a'b + a'b'

2) How many equations were written during the simplification process?

- 3
- 7
- 21

CHALLENGE ACTIVITY 2.1.1: Boolean algebra simplification.

347136.1658324.qx3zqy7

Start

Simplify the equation:

ab + a'b

= Ex: a (a + a')

1 2 3 4 5

Check Next

2.2 K-maps: Introduction

K-maps

A ***K-map*** is a graphical function representation that eases the simplification process for expressions involving a few variables, by adjacently placing minterms that differ in exactly one variable. K-map is short for ***Karnaugh map***. "Map" is used like how a country map lays out cities next to each other. A K-map lays out minterms instead.

A K-map lays out possible minterms as adjacent cells (boxes). Adjacent minterm cells differ by exactly one variable. Each function minterm cell gets a 1; other cells get 0.

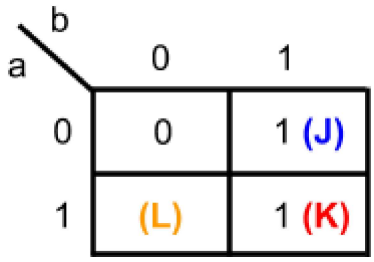
A K-map is a reoriented truth table.

PARTICIPATION ACTIVITY 2.2.1: A 2-variable K-map: Adjacent cells differ in exactly one variable.

Animation captions:

1. The basic K-map structure for 2 variables has two columns (for b = 0 and b = 1), and two rows (for a = 0 and a = 1), totaling 4 cells.
2. Like a truth table, each cell represents a possible minterm.
3. In the K-map, adjacent cells differ by one variable. Ex: a'b and ab are vertically adjacent, and differ in variable a.
4. Like a truth table, a designer insert 1's in certain K-map cells to represent a function's minterms.

PARTICIPATION ACTIVITY 2.2.2: 2-variable K-map basics.



Given function y = ab + a'b, represented in the above figure's K-map.

1) (J) corresponds to which minterm?

Check Show answer

2) (K) corresponds to which minterm?

Check Show answer

3) (L) should have what value (0 or 1)?

Check Show answer

4) Cells (J) and (K) differ in what

variable: a, or b?

Check Show answer

5) Cells (L) and (K) differ in what variable: a, or b?

Check Show answer

6) Cells (L) and (J) differ in how many variables?

Check Show answer

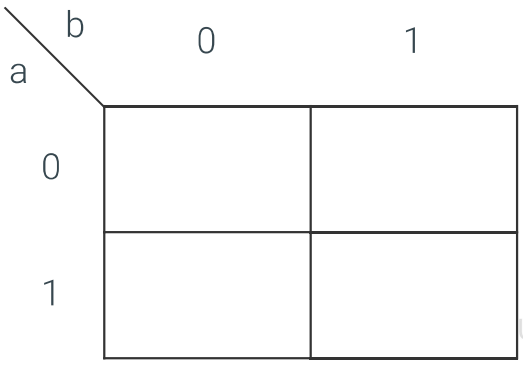
CHALLENGE ACTIVITY 2.2.1: 2-variable K-map basics.

Select the shown minterm(s).

347136.1658324.qx3zqy7

Start

a b



1 2 3 4 5 6

Check Next

Simplifying an expression with a K-map

Because adjacent minterm cells differ in exactly one variable, a K-map's key benefit is to make i(j + j') simplification opportunities obvious: Adjacent 1's are an i(j + j') opportunity. Circling two adjacent 1's graphically represents the algebraic simplification i(j + j') = i(1) = i. After drawing such a circle, a designer can write a product term with the differing variable omitted.

PARTICIPATION ACTIVITY 2.2.3: Simplification with a 2-variable K-map: i(j + j') opportunities are obvious.

Animation captions:

1. Given a sum-of-minterms equation, a designer can just write 1's in the appropriate K-map cells (like for a truth table's rows). y = ab' + ab yields 1's in the bottom two cells.
2. Adjacent 1's are an i(j + j') simplification opportunity. Algebraically, the circle represents ab' + ab = a(b' + b) = a(1) = a.
3. A designer need not write expressions. Instead, after drawing the circle, the designer eliminates the differing variable. Here b differs, a stays 1, so the circle's expression is a.

A powerful feature of a K-map is how easily replicating a minterm is achieved (recall an earlier section's example), merely by circling a cell twice.

PARTICIPATION ACTIVITY 2.2.4: Circling a 1 twice is like replicating a minterm to create i(j + j') opportunities.

Animation captions:

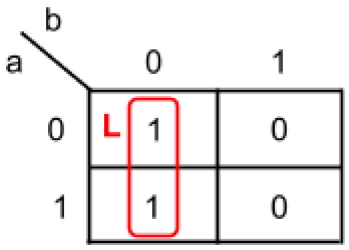
1. Given a sum-of-minterms equation y = ab + a'b + a'b', a designer places 1's in the three cells corresponding to those three minterms.
2. The designer draws a circle around the top two 1's. For that circle, a = 0, and b differs, so the circle represents a'.
3. A second circle around the right two 1's has b = 1, and a differs, so represents b. Note that the 1 for minterm a'b is circled twice, which is like replicating a'b algebraically.
4. Drawing K-map circles is easier than algebraically simplifying. The designer just draws the two circles, and writes a term for each, so y = a' + b. The equations are never written.

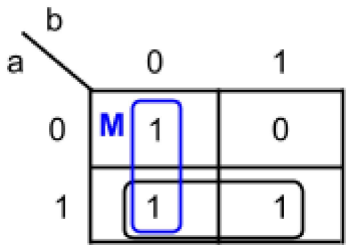
Table 2.2.1: Rules for simplifying a sum-of-minterms expression with a K-map.

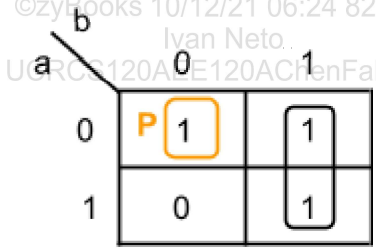
Rule 1:	*Cover every 1* at least once using circles. Add circle's term to expression.

Rule 2:	*Use fewest and largest circles* possible, to achieve simplest expression.

PARTICIPATION ACTIVITY 2.2.5: Basic 2-variable K-map.







Consider the K-maps in the figure above.

1) Circle (L) is what simplified term?

Check Show answer

2) Is circle (M) necessary? Type: yes or no

Check Show answer

3) Is circle (P) a good circle? Type: yes or no

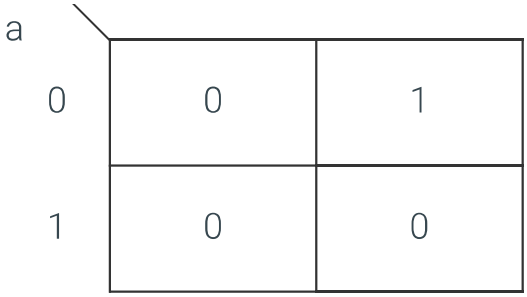
Check Show answer

CHALLENGE ACTIVITY 2.2.2: 2-variable K-map simplification.

Add fewest and largest circles to cover all the 1s.

347136.1658324.qx3zqy7

Start



Add circle Undo

1 2 3

Check Next

Example: Out-of-bed alarm

An example in an earlier section involved sounding an alarm (s = 1) if a person was up from bed (u = 1) and a button pressed (b = 1), or a person was up and button was not pressed. The captured equation was s = ub + ub'. A K-map can be used to simplify the equation.

PARTICIPATION ACTIVITY 2.2.6: Simplifying with a K-map: Out-of-bed alarm.

Animation captions:

1. A designer captures an out-of-bed alarm system as s = ub + ub' (which is already in sum-of-minterms form).
2. The designer places 1's on a 2-variable K-map, one for ub (lower left cell), and one for ub' (lower right cell). The designer places 0's in the other cells.
3. The designer then draws the largest possible circle covering those 1's. That circle is for u = 1, and differs in b, so represents u.
4. The designer adds that term to the simplified equation for s. The K-map helped the designer simplify the expression. The resulting circuit is just a wire.

PARTICIPATION ACTIVITY 2.2.7: Out-of-bed alarm system.

Consider the example above.

1) The designer captured behavior as s = ub + ub', but simplification yielded s = u. Thus, the designer incorrectly captured the original behavior.
   - True
   - False
2) The simplification on the K-map was quite obvious.
   - True
   - False

Example: Motion-sensing light

An earlier section captured a motion-sensing lamp's behavior and then simplified algebraically. That example can more-easily be simplified by a K-map instead.

PARTICIPATION ACTIVITY 2.2.8: Simplifying with a K-map: Motion-sensing light.

Animation captions:

1. For the motion-sensing lamp, simplifying the captured equation i = mt' + tm' + tm required 7 steps, and non-obvious replications of terms to enable simplification.
2. In contrast, simplification with a K-map is straightforward. The designer first fills in 1's in the three cells for minterms mt', m't, and mt.
3. Then, the designer easily draws two circles to cover the 1's, yielding terms m and t. The simplified equation is thus i = m + t. The resulting circuit is just an OR gate.

PARTICIPATION ACTIVITY 2.2.9: Motion-sensing light system.

Consider the example above.

1) How many equations were involved using algebraic simplification?
   - 2
   - 8
2) How many circles were drawn using K-

map simplification?

- 2
- 3

3) K-maps help with simplification by not obeying algebraic properties.

- True
- False

2.3 3- and 4-variable K-maps

3-variable K-map

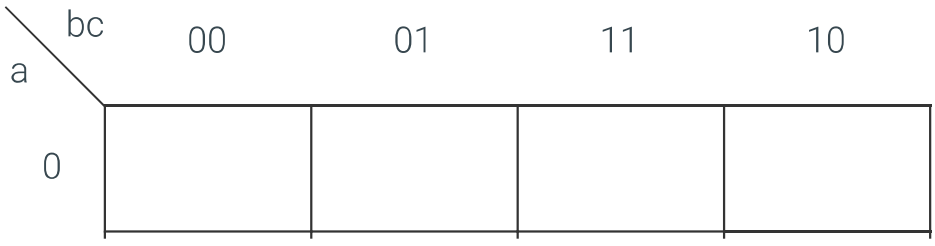
A K-map for three variables has two variables across the top. For adjacent columns to differ in only one variable, note that the columns don't count up in binary (00, 01, 10, 11), but rather are 00, 01, 11, 10. Cells on the far left and far right also differ by one variable so are also "adjacent"; the K-map wraps like a bracelet.

PARTICIPATION ACTIVITY 2.3.1: 3-variable K-map basics.

Animation captions:

1. A 3-variable K-map has 8 cells: Four columns for bc (00, 01, 11, 10) and two rows for a (0, 1). Note the last two columns are NOT 10, 11 (counting up in binary).
2. Like a truth table, each cell represents a possible minterm.
3. Adjacent cells (including left-right) differ in one variable.
4. Like a truth table, a designer inserts 1's in the K-map to represent a function's minterms. The four terms in y = ab'c' + ab'c + a'bc + a'bc' yield four 1's in the K-map.

PARTICIPATION ACTIVITY 2.3.2: Click a cell to show/hide the corresponding minterm.



1

PARTICIPATION ACTIVITY 2.3.3: 3-variable K-map.

a \ bc	00	01	11	10
0	1(J)	0	(L)	(M)
1	0	0	1(K)	0

Given function y = a'b'c' + abc + a'bc, represented in the above figure's K-map.

1) (J) corresponds to which minterm?

Check Show answer

2) (K) corresponds to which minterm?

Check Show answer

3) (L) should have what value (0 or 1)?

Check Show answer

4) (M) should have what value (0 or 1)?

Check Show answer

5) Cells (L) and (K) differ in what variable: a, b, or c?

Check Show answer

6) Cells (L) and (M) differ in what variable: a, b, or c?

Check Show answer

7) Cells (L) and (J) differ in how many variables?

Check Show answer

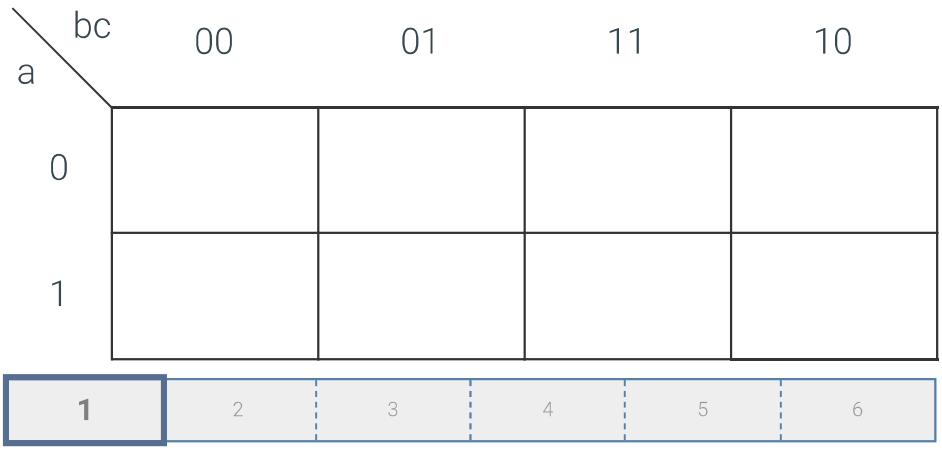
CHALLENGE ACTIVITY 2.3.1: 3-variable K-map basics.

Select the shown minterm(s).

347136.1658324.qx3zqy7

Start

a b c'



Check Next

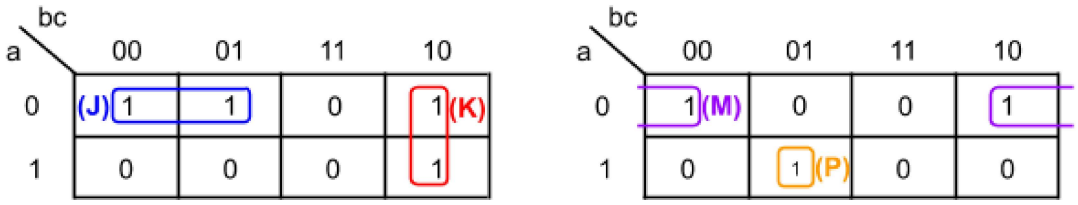
Simplification with a 3-variable K-map

PARTICIPATION ACTIVITY 2.3.4: Simplification with a 3-variable K-map: i(j + j') opportunities are obvious.

Animation captions:

1. The designer first places 1's in the K-map for minterms. For y = ab'c' + ab'c + a'bc + a'bc', four 1's are placed.
2. An adjacent pair of 1's represents an i(j + j') simplification opportunity. The designer circles the two 1's for ab'c', ab'c. Those cells differ in c, yielding ab'.
3. The designer also circles the two 1's for minterms a'bc, a'bc'. Those cells differ in c, yielding a'b.
4. The designer adds the resulting terms to the simplified equation for y, yielding y = ab' + a'b.
5. The algebraic simplifications were shown just to illustrate what each circle represented; the designer never writes those equations.

PARTICIPATION ACTIVITY 2.3.5: Simplification with 3-variable K-maps.



1) Circle (J) corresponds to what simplified term?

Check Show answer

2) Circle (K) corresponds to what simplified term?

Check Show answer

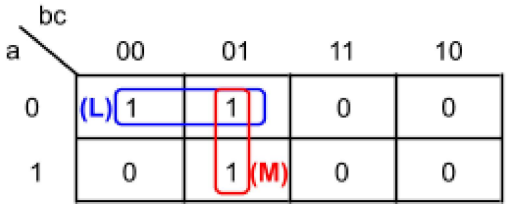
3) Circle (M) corresponds to what simplified term?

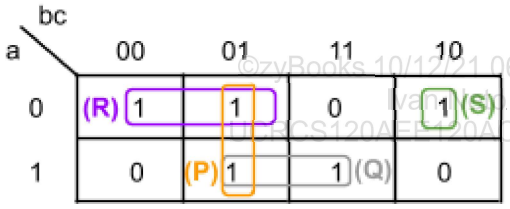
Check Show answer

4) Circle (P) is what term?

Check Show answer

PARTICIPATION ACTIVITY 2.3.6: More simplification with 3-variable K-maps.





1) Circle (L) is what simplified term?

Check Show answer

2) Circle (M) is what simplified term?

Check Show answer

3) Is circle (P) necessary? Type: yes or no

Check Show answer

4) Is circle (Q) necessary? Type: yes or no

Check Show answer

5) Is circle (S) a good circle? Type: yes or no

Check Show answer

PARTICIPATION ACTIVITY 2.3.7: 3-variable K-map simplification.

Click cell to toggle between 1 and 0. Simplified expression shown automatically.

a \ bc	00	01	11	10
0	0	0	0	0
1	0	0	0	0

y = 0

SET ALL TO 0 SHOW MINTERMS

CHALLENGE ACTIVITY 2.3.2: 3-variable K-map simplification.

Add fewest and largest circles to cover all the 1s.

347136.1658324.qx3zqy7

Start

a \ bc	00	01	11	10
0	1	0	0	0
1	0	0	0	0

Add circle Undo

1 2 3 4 5 6 7

Check Next

CHALLENGE ACTIVITY 2.3.3: Write the simplified term or terms.

347136.1658324.qx3zqy7

Start

a \ bc	00	01	11	10
0	0	0	0	0
1	0	1	0	0

1 2 3 4 5 6 7

Check Next

Larger circles

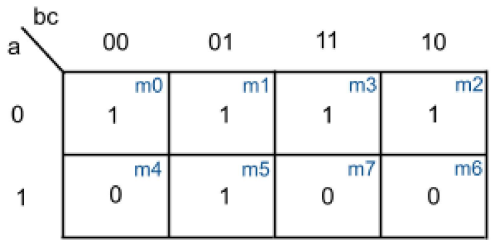
A circle may encompass four adjacent 1's, which removes two variables rather than just one.

PARTICIPATION ACTIVITY 2.3.8: Circling four 1's removes two variables.

Animation captions:

1. Circling four 1's removes two variables. For y = ab'c' + ab'c + abc + abc', the 3-variable K-map has 1's in all four bottom cells. The designer circles those four 1's.
2. Intuitively, all b c combinations appears in that circle (b'c', b'c, bc, bc'), so one MUST be true.
3. Algebraically, a is factored out, yielding a(b'c' + b'c + bc + bc'). Then additional factoring yields a(b'(c'+c) + b(c+c')), which is a(b'(1) + b(1)) = a(b' + b) = a(1) = a.
4. Allowed 4-cell circle shapes are a 1x4, or 2x2. An L shape does not have a corresponding algebraic simplification, so is not allowed.

PARTICIPATION ACTIVITY 2.3.9: Drawing largest circle with 3-variable K-maps.



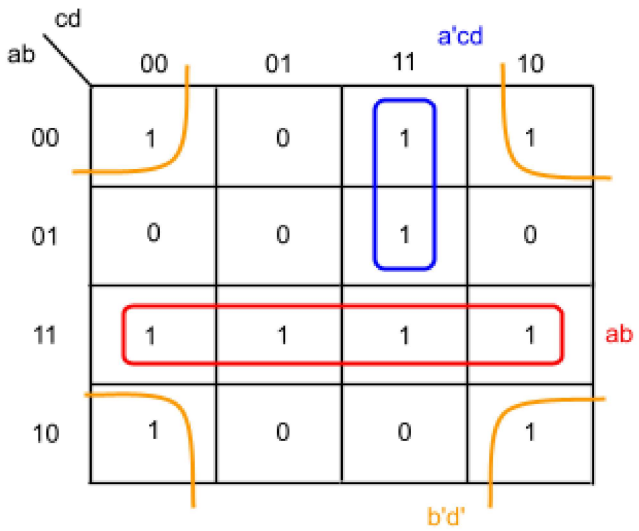
1) A circle covering which cells should be drawn?
   - m0, m1
   - m3, m2
   - m0, m1, m3, m2

2) A circle covering which cells should be drawn?
   - m1, m5
   - m5
   - m5, m7

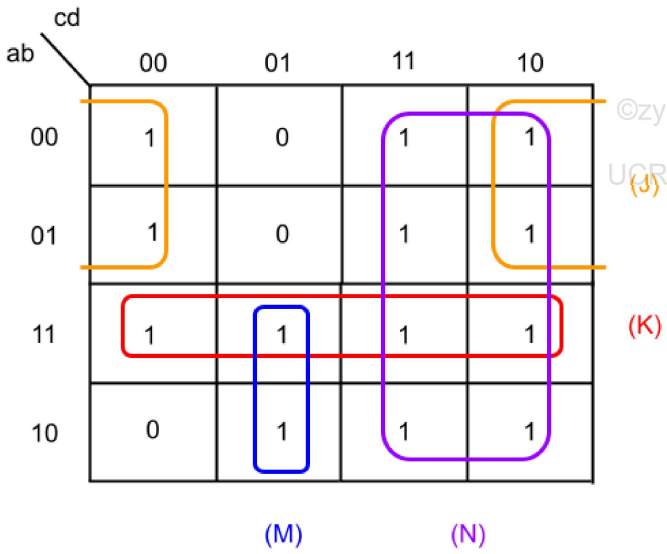
4-variable K-map

A K-map can be drawn for four variables. Top and bottom rows are adjacent (as are left and right columns). Valid circle sizes are 1, 2, 4, 8, or 16 cells.

Figure 2.3.1: Four-variable K-map example.



PARTICIPATION ACTIVITY 2.3.10: Simplification with 4-variable K-maps.



Match the simplified term.

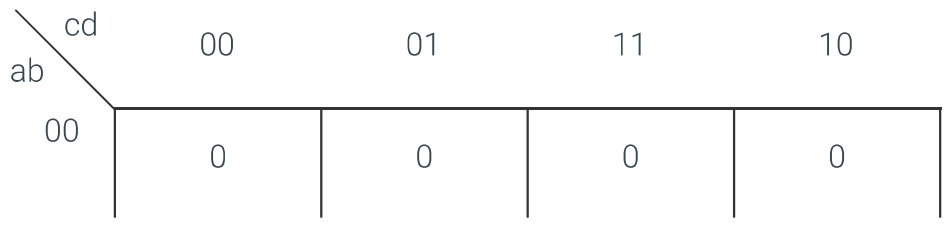
(J) (M) (K) (N)

	a'd'
	ab
	ac'd
	c

Reset

PARTICIPATION ACTIVITY 2.3.11: 4-variable K-map simplification.

Click cell to toggle between 1 and 0. Simplified expression shown automatically.



01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

y = 0

SET ALL TO 0 SHOW MINTERMS

Expression simplification in programming

K-maps usage is not restricted to just digital design. Computer programmers sometimes simplify expressions using K-maps. Ex: A program may make a decision based on an expression: If ((isRed AND !isBlue) OR (!isRed AND isBlue) OR (isRed AND isBlue) then take action X. (! means NOT).

The expression represents a function with two variables. Using a two-variable K-map, the programmer simplifies the expression, yielding: If (isRed OR isBlue) then take action X. The resulting program is easier to read.

Exploring further:

- Online K-map tool (T. Thormaehlen)

2.4 K-map examples

Example: Majority voter circuit

An earlier section captured a majority voter circuit's behavior as a truth table. The resulting equation can be simplified using a 3-variable K-map before creating a circuit, yielding a smaller circuit than for the original unsimplified equation.

PARTICIPATION ACTIVITY 2.4.1: Majority voter circuit, simplified using a K-map.

Animation captions:

1. A designer captured the majority voter behavior using a truth table. To convert to a circuit, the designer first converts to an equation: y = y1'y2y3 + y1y2'y3 + y1y2y3' + y1y2y3.
2. The designer places four 1's on a 3-variable K-map for those four minterms. (Actually, the 1's could have come directly from the truth table; the equation wasn't really necessary).
3. The designer draws three circles, yielding a simpler equation.
4. The final circuit is much simpler than for the starting equation.

PARTICIPATION ACTIVITY 2.4.2: Majority voter circuit simplified using a K-map.

Consider the example above. Type numbers as 2, not two.

1) The K-map had _____ 1's.

Check Show answer

2) The designer drew _____ circles.

Check Show answer

3) _____ circles included the cell for y1y2y3.

Check Show answer

4) If each AND and OR gate input is two transistors, the final circuit requires _____ transistors.

Check Show answer

Arbiter

An ***arbiter*** decides (arbitrates) which of several competing items wins. Ex: If only one button should be pressed at a time but a user presses two or more, an arbiter can decide which pressed button will be recognized. Ex: If two devices simultaneously try to access a resource like a printer, network cable, or storage, an arbiter can decide which device will get access.

PARTICIPATION ACTIVITY 2.4.3: Arbiter circuit, simplified using a K-map.

Animation content:

undefined

Animation captions:

1. A keypad has multiple buttons, only one of which should be pressed at a time, but a user might press two or more.
2. An arbiter decides which button press wins. For inputs a, b, c, output one 1 on s, t, u: a has highest priority, b next, then c. Ex: If abc is 101, output stu is 100.
3. A designer can capture the desired behavior as a truth table. If input a is 1 (regardless of values on b, c), output stu is 100. If b is 1, stu is 010, or if c is 1, stu is 001.
4. The designer can create a K-map for each output. Each K-map is simplified to yield an equation.
5. The designer can create a circuit for each equation.

PARTICIPATION ACTIVITY 2.4.4: Arbiter.

Consider the example above.

1) A keypad may have three buttons but only one should be pressed at a time. An arbiter outputs _____.

- ○ an indication that two buttons have been pressed
- ○ exactly one winner of multiple button presses
- ○ a shock to the user to discourage multiple button presses

2) Suppose a fourth button d were introduced with corresponding output v, and lower priority than c. Based on the equations for s, t, u, what would be the equation for v?

- v = d
- v = a'b'c'd
- Cannot determine

3) Suppose a fourth button d were introduced with corresponding output v, and lower priority than c. Would any of the equations for s, t, u need to be changed?

- Yes
- No

4) What is the word for a component that decides which of multiple inputs "wins" a competition?

- arbitrator
- arbitor
- arbiter

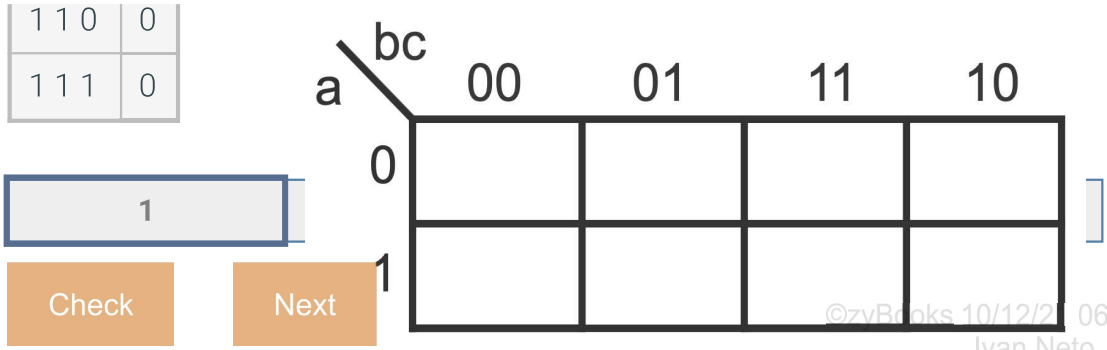
CHALLENGE ACTIVITY 2.4.1: Converting truth tables and equations to K-maps.

347136.1658324.qx3zqy7

Start

Select the K-map cells where f is 1.

a b c	f
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	1
1 0 1	0



2.5 DeMorgan's Law

Basics

DeMorgan's Law is a Boolean algebra property for complementing an expression, coming in two forms: (a + b)' = a'b', and (ab)' = a' + b'. Each literal is complemented, and ANDs / ORs swap.

Table 2.5.1: DeMorgan's Law.

Property	Name	Description
(a + b)' = a'b'	DeMorgan's Law (for OR)	Each literal complemented, ORs become ANDs.
(ab)' = a' + b'	DeMorgan's Law (for AND)	Each literal complemented, ANDs become ORs.

PARTICIPATION ACTIVITY 2.5.1: DeMorgan's Law.

Animation captions:

1. DeMorgan's Law: Complement literals, swap AND / OR. Thus (ab)' = a' + b'.
2. Likewise, (a + b)' = a'b'.

PARTICIPATION ACTIVITY 2.5.2: DeMorgan's Law.

1) (de)' = ?

- d'e'
- d' + e'
- d + e

2) (f + g)' = ?

- f'g'
- fg
- f' + g'

3) (abc)' = ?

- a'b'c'
- a' + b' + c'
- DeMorgan's Law does not apply.

4) A play's cast has an understudy (a substitute). Which is equivalent to saying: If not all cast members show up, the understudy participates.

- If any cast member does not show up, the understudy participates.
- If all cast members show up, the understudy participates.

Examples

PARTICIPATION ACTIVITY 2.5.3: DeMorgan's Law for (ab)': Plane doors.

Animation captions:

1. A plane can take off if both doors are closed (b = 1 and c = 1). That behavior captured as an equation is bc.
2. A warning light should illuminate (y = 1) if NOT both doors are closed. The equation is y = (bc)', which is just the opposite of bc.
3. A designer can apply DeMorgan's Law to convert the equation to sum-of-products: y = (bc)' = b' + c'.
4. The meaning is intuitive: "NOT both doors closed" means the same as "Either door is open".

PARTICIPATION ACTIVITY 2.5.4: DeMorgan's Law for (a + b)': Guards.

Animation captions:

1. If at least one guard is present, d + e, all is good . If NOT, (d + e)', notify manager (y = 1). The designer captures as an equation: y = (d + e)'.
2. The designer applies DeMorgan's Law: y = (d + e)' = d'e'.
3. The law is intuitive: "NOT either guard present" is the same as "both guards absent".

PARTICIPATION ACTIVITY 2.5.5: DeMorgan's Law: Basic examples.

Consider the examples above.

1) The plane can take off if both doors are _____.
   - closed
   - not closed

2) The plane's warning light illuminates if it ____ the case that both doors are closed.
   - is
   - is NOT

3) y = (ab)' _____ in sum-of-products form.
   - is
   - is not

4) y = a' + b' _____ in sum-of-products form.
   - is
   - is not

5) For the guards examples, all is good if any guard is present. The expression for that situation is _____.
   - d + e
   - (d + e)'

6) If NOT the case that any guard is present, the manager should be

notified. The expression for notifying the manager is _____.

- d + e
- (d + e)'

7) Which is in sum-of-products form?

- (d + e)'
- d'e'

More complex cases

The laws apply to expressions beyond just literals. Examples:

- (a'b)' = a'' + b', so a + b'.
- (ab + c)' = (ab)'c'. The law can then be applied again: (a' + b')c'. Multiplying out yields a'c' + b'c'.

PARTICIPATION ACTIVITY 2.5.6: DeMorgan's Law for more complex examples.

1) (d'e)' = ?
Simplify answer.

Check Show answer

2) (f' + g')' = ?

Check Show answer

3) ((d + e)f)' = ?
Apply DeMorgan's Law twice.

Check Show answer

4) (ab + c)' = (?)c'
Apply DeMorgan's Law twice. (Only type the ? part).

( ) c '

Check Show answer

CHALLENGE ACTIVITY 2.5.1: Simplify using DeMorgan's Law.

347136.1658324.qx3zqy7

Start

Apply DeMorgan's Law once to find the inverse of the equation:

(a'b)' = Ex: x + y

1 2 3 4 5

Check Next

DeMorgan's Law in programming

DeMorgan's Law is not just used in digital design, but also by computer programmers. Computer programs use expressions to control decisions. Ex: A program may proceed as long as the input is not 3 or 5. The programmer may write: If NOT(x==3 OR x==5) then proceed.

To simplify the expression, the programmer may apply DeMorgan's Law: If (x != 3 AND x != 5) then proceed. (!= means not equal). The NOT was applied to the two items (so == became != in two places), and the OR was changed to AND. The resulting expression may be simpler to read.

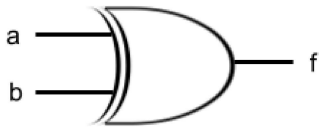
2.6 XOR / XNOR gates

XOR

A two-input ***XOR*** gate (for "exclusive OR") outputs 1 if the input values differ. Thus, y = a XOR b is equivalent to y = ab' + a'b. Digital designers often use the symbol ⊕ for XOR, as in: y = a ⊕ b.

Figure 2.6.1: XOR truth table and gate.

a	b	f
0	0	0
0	1	1
1	0	1
1	1	0



PARTICIPATION ACTIVITY 2.6.1: XOR.

1) 1 XOR 0 = ?
   - 1
   - 0

2) 1 XOR 1 = ?
   - 1
   - 0

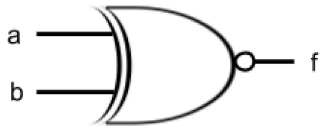
3) 0 XOR 0 = ?
   - 1
   - 0

XNOR

A two-input ***XNOR*** gate outputs 1 if the input values are the same. XNOR is the opposite (NOT) of an XOR gate, hence the "N". y = a XNOR b is equivalent to y = a'b' + ab.

Figure 2.6.2: XNOR truth table and gate.

a	b	f
0	0	1
0	1	0
1	0	0
1	1	1



PARTICIPATION ACTIVITY 2.6.2: XNOR.

1) 1 XNOR 0 = ?
   - 1
   - 0

2) 1 XNOR 1 = ?
   - 1
   - 0

3) 0 XNOR 0 = ?
   - 1
   - 0

4) In contrast to an XOR gate, an XNOR gate's drawing has a _____ drawn at the output.
   - square
   - bubble

Basic XOR and XNOR examples

PARTICIPATION ACTIVITY 2.6.3: XOR example: Factory doors.

Animation captions:

1. For fire safety, a factory's doors must both be locked (a = 1, b = 1) when empty, or both unlocked when people are present. If only one is unlocked, a warning sounds (w = 1).
2. XOR achieves the desired behavior.
3. Warning sounds if only one door is locked, other unlocked.

PARTICIPATION ACTIVITY 2.6.4: XNOR example: Park ride.

Animation captions:

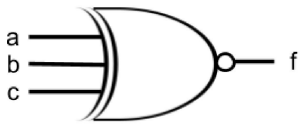
1. For balance, a two-seat amusement park ride is activated (output y = 1) only when there are two riders (a = 1, b = 1), or no riders.
2. y is 1 only when ab is 00 or 11, meaning when ab are the same. XNOR captures the desired behavior: y = a XNOR b.
3. If a, b differ, y is 0. If a, b are the same, y is 1.

Multi-input XOR / XNOR

If XOR has more than two inputs, the output is 1 if the number of input 1's is odd. XNOR's output is 1 if the number of input 1's is even.

Figure 2.6.3: Truth table and gate for a 3-input XOR and and 3-input XNOR.

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

PARTICIPATION ACTIVITY 2.6.5: Multi-input XOR and XNOR.

1) 0 XOR 1 = ?

- 1
- 0

2) 1 XOR 1 = ?

- 1
- 0

3) 0 XOR 1 XOR 0 = ?

- 1
- 0

4) 1 XOR 1 XOR 0 = ?

- 1
- 0

5) 1 XOR 1 XOR 1 = ?

- 1
- 0

6) 1 XOR 0 XOR 1 XOR 1 = ?

- 1
- 0

7) 1 XNOR 1 XNOR 1 = ?

- 1
- 0

8) 1 XNOR 1 XNOR 1 XNOR 1 = ?

- 1
- 0

Example: Parity bit during data transmission

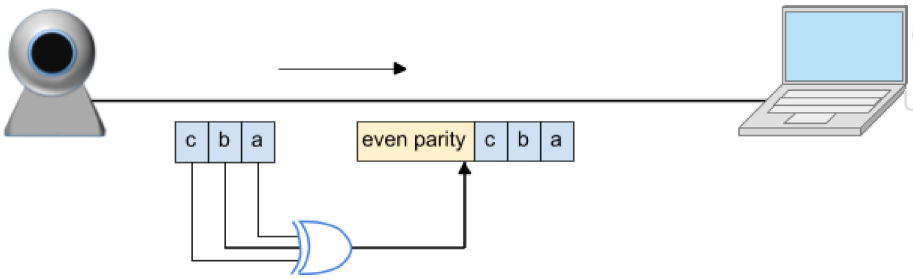
Digital devices commonly communicate bits, such as via a USB cable or via Bluetooth. Ex: A webcam may communicate 010 to a computer. Electrical noise can change a bit from 0 to 1 (or vice-versa), such as 010 changing to 110.

To help a receiver detect an erroneous communication, the sender sends an extra bit, called an ***even parity*** bit, such that the total number of 1's is even. So 010 is sent as 1 010, making the number of 1's even (2). 011 would be sent as 0 011.

An XOR gate quickly computes the desired parity bit.

On the receiving end, another XOR gate detects if the received bits have an odd number of 1's. If odd, the receiver rejects the data.

Figure 2.6.4: Parity during data transmission.



PARTICIPATION ACTIVITY 2.6.6: Parity bits.

1) a = 0, b = 0, c = 0, even parity = ?
   - 1
   - 0

2) a = 0, b = 1, c = 0, even parity = ?
   - 1
   - 0

3) a = 1, b = 0, c = 1, even parity = ?
   - 1
   - 0

Deriving XNOR's expression using DeMorgan's Law

If you've studied DeMorgan's Law, the following shows how XNOR's ab + a'b' can be derived by complementing XOR's a'b + ab'.

(a XOR b)'
(a'b + ab')'
(a'b)' · (ab')' DeMorgan's Law
(a'' + b')(a' + b'') DeMorgan's Law (again)
(a + b')(a' + b)
aa' + ab + b'a' + b'b

0 + ab + a'b' + 0
ab + a'b'
a XNOR b

PARTICIPATION ACTIVITY 2.6.7: DeMorgan's Law and XOR/XNOR.

1) XNOR is the NOT of XOR, so a XNOR b means (a'b + ab')'.
   - ○ True
   - ○ False

2) The expression for XNOR can be derived by applying DeMorgan's Law as follows: (a'b + ab')' = ab' + a'b.
   - ○ True
   - ○ False

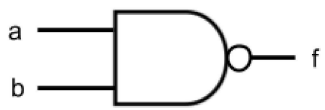
2.7 NAND / NOR (universal gates)

NAND

A ***NAND*** gate is the opposite (the NOT, hence the "N") of an AND gate, outputting 0 if all inputs are 1s; else the output is 1.

Figure 2.7.1: NAND truth table and gate.

a	b	f
0	0	1
0	1	1
1	0	1
1	1	0



PARTICIPATION ACTIVITY 2.7.1: NAND gates.

1) 0 NAND 1 = ?

- 1
- 0

2) 1 NAND 1 = ?

- 1
- 0

3) 0 NAND 0 = ?

- 1
- 0

A NAND gate is a universal gate

NAND gates are popular due to having a simpler CMOS transistor circuit implementation than AND gates: Recall that an AND gate is built from a NAND transistor circuit followed by a NOT circuit.

Furthermore, NAND gates are popular due to being a universal gate. A ***universal gate*** is a single gate type that can implement any combinational circuit. NAND can implement NOT, AND, and OR, as shown below, and is thus universal.

PARTICIPATION ACTIVITY 2.7.2: NAND is a universal gate.

Animation captions:

1. NAND can implement NOT. If input a is fed to both inputs of a 2-input NAND gate, the output is (aa)' = a' + a' (DeMorgan's Law) = a'.
2. NAND can implement AND. A 2-input NAND (ab)' followed by a NOT (also implemented with a 2-input NAND) yields ((ab)')' = (ab)" = ab.
3. NAND can implement OR. A two-input NAND with NOTs at each input yields is (a'b')' = a" + b" (DeMorgan's Law) = a + b.
4. Because NAND can implement NOT, AND, and OR, then any circuit can be implemented with just NANDs. NAND is thus a "universal gate".

PARTICIPATION ACTIVITY 2.7.3: Universal gates.

1) A NAND gate is a universal gate.

- True
- False

2) A NAND gate cannot implement a NOT gate.

- True
- False

3) Inverting the output of a NAND gate produces an AND gate.

- True
- False

4) Inverting the inputs of a NAND gate produces an OR gate.

- True
- False

Converting to NAND gates

NAND being a universal gate enables chip makers to pre-fabricate a chip consisting of millions of NAND gates. Any circuit of NAND gates can be implemented simply by adding wires. Pre-fabricating the chip with AND, OR, and NOT gates would involve complexities like deciding how many of each gate to pre-fabricate, and where to place each gate type. Using NAND is much simpler.

A chip with pre-fabricated gates is sometimes called a ***gate-array ASIC***. ***ASIC*** is short for Application-Specific Integrated Circuit.

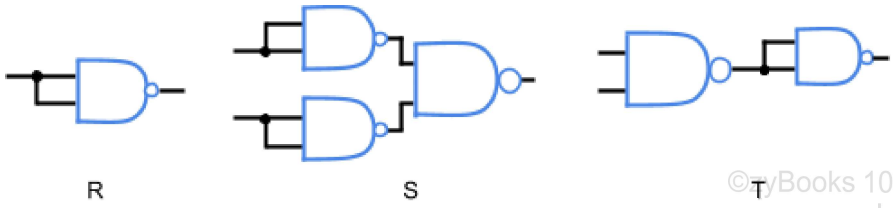
Converting an AND/OR/NOT circuit to a NAND-only circuit enables implementation using fewer transistors as well as enables implementation on a gate-array ASIC. The conversion can be done simply by replacing each AND, OR, and NOT gate by the equivalent structure of NAND gates, then removing double-inversions.

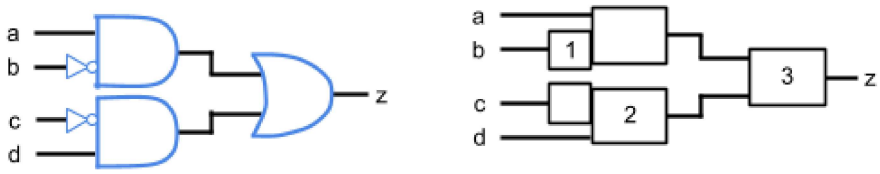
PARTICIPATION ACTIVITY 2.7.4: Converting to a NAND-only implementation.

Animation captions:

1. Given any circuit with AND, OR, and NOT gates, each gate can be replaced by an equivalent NAND circuit. For circuit q = a'b + cd, a designer first replaced the NOT gate by an equivalent 2-input NAND gate.
2. Next, the designer replaces each 2-input AND gate by an equivalent circuit of two NAND gates each.
3. Then, the designer replaces the OR gate by an equivalent three NAND gate circuit.
4. Finally, the designer simplifies the circuit by eliminating any sequence of NOT gate followed by a NOT gate, since such a double inversion yields the original signal (x" = x).

PARTICIPATION ACTIVITY 2.7.5: Converting to NAND gates.





Consider the figure above. Which NAND pattern goes into which box?

1) Box 1

Check Show answer

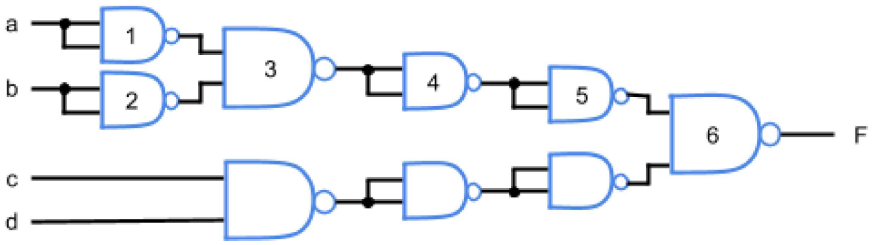
2) Box 2

Check Show answer

3) Box 3

Check Show answer

PARTICIPATION ACTIVITY 2.7.6: Simplifying NAND circuits.



1) Which pair of NAND gates can be eliminated?

- 1, 2
- 3, 4
- 4, 5

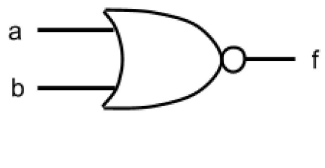
NOR

A ***NOR*** gate is the opposite of an OR gate, outputting 0 if any of the inputs are 1s; else the output is 1.

A discussion analogous to the above NAND discussion exists for NOR. Such discussion is omitted here. Briefly, NOR's transistor structure is simpler than OR's. NOR is also a universal gate. NOT: (a + a)' = a'a' = a' (NOR with inputs tied together). OR: ((a + b)')' = (a + b)" = a + b (NOR followed by NOT). AND: (a' + b')' = a"b" = ab (NOR with each input NOTed).

Figure 2.7.2: NOR truth table and gate.

a	b	f
0	0	1
0	1	0
1	0	0
1	1	0



PARTICIPATION ACTIVITY 2.7.7: NOR gates.

1) 0 NOR 0 = ?

- 1
- 0

2) 1 NOR 1 = ?

- 1
- 0

3) 0 NOR 1 NOR 1= ?

- 1
- 0

4) A NOR gate is a universal gate.

- True
- False

5) An AND gate is a universal gate.

- True
- False

6) A NOT gate is a universal gate.

- True
- False

2.8 Muxes

Basics

A ***multiplexor*** is a combinational circuit that passes one of multiple data inputs through to a single output, selecting which one based on additional control inputs. ***Mux*** is short for multiplexor. A mux's control inputs are called ***select lines***.

Analogy: Due to road construction, four lanes (the data inputs) may be reduced to a single lane (the single output). A policeman (the select inputs) selects which one lane currently passes through by blocking the other lanes.

A ***4x1 mux***, spoken as "4 to 1 mux", has 4 data inputs, 1 data output, and requires 2 select inputs.

PARTICIPATION ACTIVITY 2.8.1: A 4x1 mux.

Animation captions:

1. A 4x1 mux has 4 data inputs (i0-i3), 2 control inputs s1, s0, and output y. When s1s0 = 00, i0 passes through to y.
2. s1s0 = 01 allows i1 to pass.
3. When s1s0 are 01, i1's value will pass through. If i1 is 0, y will be 0. If i1 is 1, y will be 1.
4. s1s0 = 10 passes i2, and s1s0 = 11 passes i3. The truth table is compacted, listing y as i0, i1, i2, or i3 for s1s0 = 00, 01, 10, 11 (respectively).

PARTICIPATION ACTIVITY 2.8.2: Muxes.

Given a 4x1 mux. Assume i3 i2 i1 i0 are 0 1 1 0.

1) If s1s0 = 00, then y = _____ .
   - 1
   - 0

2) If s1s0 = 10, then y = _____ .
   - 1
   - 0

3) s1s0 = _____ allows i3 to pass through to y.
   - 10
   - 11

4) Suppose s1s0 = 11, and i3 is 0, so y = 0. Then, suppose i3 changes from 0 to 1. What will y become?
   - 1
   - 0

Mux equation and circuit

A mux's truth table can be converted to an equation, and then to a circuit.

PARTICIPATION ACTIVITY 2.8.3: A 4x1 mux's behavior can be captured as an equation.

Animation captions:

1. A 4x1 mux's behavior can be captured as an equation. When s1s0 = 00, y should equal i0, so y = s1's0'i0.
2. When s1s0 = 00, the term simplifies to just i0: y = s1's0'i0 = 0'0'i0 = 1*1*i0 = i0.
3. When s1s0 = 01, y should equal i1. That term is s1's0i1. So y = s1's0'i0 + s1's0i1.
4. When s1s0 = 10, y should equal i2, so y = s1's0'i0 + s1's0i1 + s1s0'i2. When s1s0 = 11, y should equal i3, so y = s1's0'i0 + s1's0i1 + s1s0'i2 + s1s0i3.
5. For specific s1s0 values, the expression simplifies to the corresponding data input. Ex: For s1s0 = 11, y = 1'1'i0 + 1'1i1 + 11'i2 + 11i3 = 00i0 + 01i1 + 10i2 + 11i3 = 0 + 0 + 0 + i3 = i3.

PARTICIPATION ACTIVITY 2.8.4: The equation for a 4x1 mux is easily converted to a circuit.

Animation captions:

1. The equation for a 4x1 mux, y = s1's0'i0 + s1's0i1 + s1s0'i2 + s1s0i3, can be converted to a circuit having 4 AND gates and 1 OR gate. Two NOT gates are used to generate s1', s0'.
2. When s1s0 = 00, y should equal i0.
3. The top three AND gates each have at least one s1, s0 value of 0, so will output 0. Only the bottom AND has s1, s0 both 1's, so the bottom AND gate's output will be whatever i0's value is.
4. The OR gate has three 0's from the top three AND gates, and then i0's value. 0 OR x is just x. Thus, the OR gate passes i0's value through. Therefore, when s1s0 = 00, y equals i0.

PARTICIPATION ACTIVITY 2.8.5: Mux design.

1) How many select lines does a 4x1 mux require?

Check Show answer

2) If s1s0 = 01, then y = _____ .
Type: i0, i1, i2, or i3

Check Show answer

3) y = _____ + s1's0i1 + s1s0'i2 + s1s0i3

Check Show answer

Example: Jet cockpit engine status display

A jet cockpit has limited space for switches and lights. A particular jet has four "Engine OK" inputs coming from sensors at each of four engines. The cockpit has a single "Engine temperature OK" light, and two switches for pilots to select among the four Engine OK inputs. A circuit should pass the selected sensor input to the light.

A 4x1 mux readily achieves the desired behavior.

PARTICIPATION ACTIVITY 2.8.6: Mux example: Jet cockpit status light display.

Animation captions:

1. A jet cockpit has limited space. Two switches control which engine's status is displayed via a bulb. If switches = 01, engine B's status is displayed.
2. If the pilot changes the switches to 11, engine D's status is displayed.
3. The system is easily implemented using a 4x1 mux.

PARTICIPATION ACTIVITY 2.8.7: Engine status display.

Consider the example above.

1) If the pilots set the switches to 00, will the light illuminate?
   - Yes
   - No

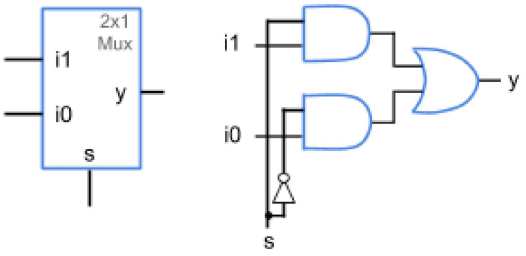
2) If the pilots set the switches to 11, will the light illuminate?
   - Yes
   - No

3) Suppose the switches are set to 11, and Engine D's status input is 0, so the light is not illuminated. Now suppose the pilots fix the problem with Engine D, such that D's status changes from 0 to 1. Will the light illuminate?
   - Yes
   - No

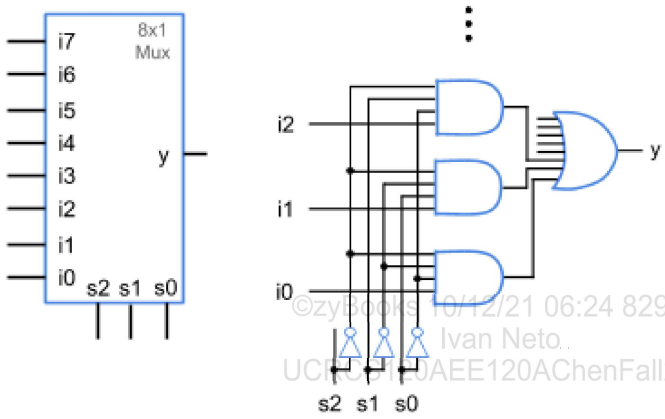
Mux sizes

Mux sizes may be 2x1, 4x1, 8x1, 16x1, etc. For N data inputs, a mux requires $\log_2 N$ select inputs.

Figure 2.8.1: 2x1 and 8x1 muxes.



y = s'i0 + si1



y = s2's1's0'i0 + s2's1's0i1 + s2's1s0'i2 + s2's1s0i3 + s2s1's0'i4 + s2s1's0i5 + s2s1s0'i6 + s2s1s0i7

PARTICIPATION ACTIVITY 2.8.8: Muxes.

1) How many select lines does a 2x1 mux require?

Check Show answer

2) How many select lines does an 8x1 mux require?

Check Show answer

3) How many select lines does a 16x1 mux require?

Check Show answer

4) How many AND gates does a 2x1 mux require?

Check Show answer

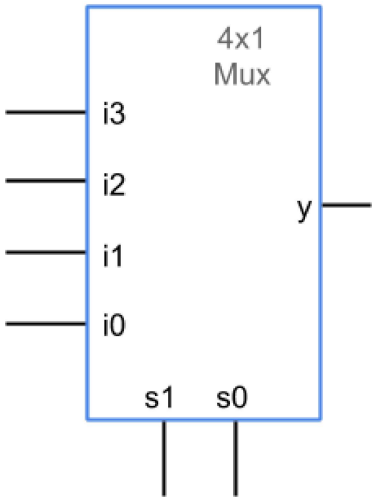
5) How many AND gates does an 8x1 mux require?

Check Show answer

CHALLENGE ACTIVITY 2.8.1: Mux inputs and output.

347136.1658324.qx3zqy7

Start



Which s1s0 allows input i1 to pass through to y? Ex: 10

1 2 3 4

Check Next

2.9 Example: Multiplexed automobile above-mirror display

Many cars have an above-mirror display. A driver can press a button to select one of several options to display, like temperature or average miles per gallon. A mux can be used to design an above-mirror display to reduce the number of wires that need to be connected to the display.

PARTICIPATION ACTIVITY 2.9.1: Above-mirror display using an 8-bit 4x1 mux.

Animation content:

undefined

Animation captions:

1. A display above a car's rear view mirror can display the temperature (T), average mpg (A), instantaneous mpg (I), and miles remaining (M).
2. Sensors in the car measure and send the values to the car's central computer. Each data input is 8-bits wide.
3. A driver presses a button to select what value is displayed from T, A, I and M. An 8-bit 4x1 mux select the desired value, and the mux output connects to the display.
4. Additional inputs x and y follow a sequence - 00, 01, 10, 11 when the button is pressed. xy = 00 displays T, xy = 01 displays A, xy = 10 displays I, and xy = 11 displays M.

PARTICIPATION ACTIVITY 2.9.2: Mux example: Above-mirror display.

1) How many wires are run from the car's central computer to the mux?
   - 4
   - 8
   - 32

2) What mux configuration is required if each sensor value was 32 bits?
   - 8-bit 32x1 mux
   - 32-bit 4x1 mux
   - 32-bit 32x1 mux

3) What mux configuration is required to design a similar display system that displays 6 different 8-bit sensors values?
   - 8-bit 6x1

- 6-bit 8x1
- 8-bit 8x1

2.10 Decoders

Basics

A ***decoder*** is a combinational circuit that converts N inputs to a 1 on one of 2^N outputs. A ***2x4 decoder***, spoken as "2 to 4 decoder", converts two inputs to a 1 on exactly one of four outputs.

PARTICIPATION ACTIVITY 2.10.1: A 2x4 decoder.

Animation captions:

1. A 2x4 decoder outputs exactly one 1, based on the two inputs. i1i0 = 00: y0 = 1 (the other outputs are 0's).
2. i1i0 = 01: y1 = 1 (the other outputs are 0's).
3. i1i0 = 10: y2 = 1 (the other outputs are 0's).
4. i1i0 = 11: y3 = 1 (the other outputs are 0's). A truth table easily captures a 2x4 decoder's behavior, having inputs i1, i0, and outputs y3, y2, y1, y0 (so four functions).

PARTICIPATION ACTIVITY 2.10.2: 2x4 decoder.

Consider a 2x4 decoder.

1) If i1i0 = 00, then y1 = _____ .

Check Show answer

2) If i1i0 = 01, then y1= _____ .

Check Show answer

3) i1i0 = _____ configures the decoder

to output y0 = 0, y1 = 0, y2 = 0, and y3 = 1.

Check Show answer

4) How many outputs are set to 1 at any given time?
Type: 0, 1, 2, 3, or 4

Check Show answer

5) i1i0 = _____ configures the decoder to output y0 = 0, y1 = 0, y2 = 1, and y3 = 1.
Type: 00, 01, 10, 11, or ** if not possible.

Check Show answer

Example: Lawn sprinkler controller

A lawn sprinkler system may have multiple zones. A sprinkler controller activates only one zone at a time, due to limited incoming water. The brain of the controller, typically a small computer, may encode the active zone in binary on output pins, to save pins: If a system has 8 zones, only 3 pins are needed, while 4 zones need only 2 pins. A decoder can convert the binary encoded zone into the activation of the appropriate zone.

PARTICIPATION ACTIVITY 2.10.3: Lawn sprinkler example using a decoder.

Animation captions:

1. A lawn sprinkler system has multiple zones. Due to limited incoming water, only one zone should be activated at a time.
2. The controller's brain has two outputs, encoding the active zone in binary: 00, 01, 10, or 11. Those two outputs should activate one corresponding zone.
3. A decoder easily implements the desired behavior, converting two inputs to a 1 on exactly one of four outputs.

PARTICIPATION ACTIVITY 2.10.4: Lawn sprinkler system with a decoder.

Consider the example above.

1) If the brain outputs 11, what zone is activated?
   - A
   - D

2) What brain output values will activate all zones at once?
   - 11
   - No such values

3) If a system has 32 zones instead of 4, how many outputs would the brain need?
   - 4
   - 5
   - 32

Decoder equation and circuit

A 2x4 decoder has four outputs. Each output's behavior is easily converted to an equation and then to a circuit.

PARTICIPATION ACTIVITY 2.10.5: Each decoder output is easily converted to an equation and circuit.

Animation captions:

1. Each of a decoder's outputs is a unique function. A designer can convert the table to an equation for each output. y0 = i1'i0'.
2. The designer can then convert the equation y0 = i1'i0' to a circuit consisting of one AND gate.
3. Likewise, y1 = i1'i0. y1's circuit also has one AND gate.
4. y2 = i1i0', yielding one AND gate.
5. Finally, y3 = i1i0, yielding one AND gate. The NOT gates are reused, so only two are needed.

PARTICIPATION ACTIVITY 2.10.6: Decoder design.

1) How many inputs does a 2x4 decoder have?

Check Show answer

2) How many outputs does a 2-input decoder have?

Check Show answer

3) How many AND gates does a 2x4 decoder require?

Check Show answer

4) How many OR gates does a 2x4 decoder require?

Check Show answer

Decoder sizes

Other size decoders can be designed similarly.

PARTICIPATION ACTIVITY 2.10.7: Designing other sized decoders.

Animation captions:

1. A 1x2 decoder has 1 input i and 2 outputs y0 and y1. The outputs equations are y0 = i, and y1 = i. The circuit just has a NOT gate and some wires.
2. A 3x8 decoder has 3 inputs i0 i1 i2, and 8 outputs y0-y7. The equation for y0 is i2'i1'i0' (meaning i2i1i0 = 000). The circuit is just an AND gate, with three NOTs at the inputs.
3. y1 = i2'i1'i0. The circuit gets another AND gate for y1. The NOT gates are reused.
4. The remaining outputs are created similarly.

PARTICIPATION

ACTIVITY 2.10.8: Various sized decoders.

1) How many inputs are required for a decoder with 8 outputs?

Check Show answer

2) How many inputs are required for a decoder with 2 outputs?

Check Show answer

3) How many outputs does a 2-input decoder have?

Check Show answer

4) How many outputs does a 4-input decoder have?

Check Show answer

5) How many AND gates does a 3x8 decoder require?

Check Show answer

6) How many OR gates does a 5x32 decoder require?

Check Show answer

©zyBooks 10/12/21 06:24 829162
Ivan Neto
UCRCS120AEE120AChenFall2021

Decoder with enable

Some decoders have an additional input called an ***enable input*** that when 0 sets all outputs to 0s, and when 1 enables the decoder for normal behavior.

A decoder's equations and circuit are easily extended for an enable input by including the enable in each AND.

PARTICIPATION ACTIVITY 2.10.9: A decoder with an enable input.

Animation captions:

1. A 2x4 decoder can be extended with an enable input e, feeding into all AND gates.
2. e = 1 allows normal decoder behavior, because 1 AND x is just x. Thus, when e = 1, each AND gate acts as if e didn't exist.
3. However, when e = 0, all four AND gates output 0, regardless of the values on i1i0, because 0 AND x is 0. The decoder is "disabled".

PARTICIPATION ACTIVITY 2.10.10: Lawn sprinkler example using a decoder.

Animation captions:

1. A sprinkler system may have a manual shutoff switch, perhaps for a homeowner playing with kids on the lawn. In one switch position, the sprinklers behave as usual.
2. But if the switch is moved to the off position, the brain's outputs are ignored, and all zones are deactivated. A decoder with enable easily implements the desired behavior.

PARTICIPATION ACTIVITY 2.10.11: Decoders with enable.

1) If i1i0 = 01 and enable = 1, then y1 = _____ .

Check Show answer

2) If i1i0 = 00 and enable = 1, then y1 = _____ .

Check Show answer

3) If i1i0 = 11 and enable = 0, then y3

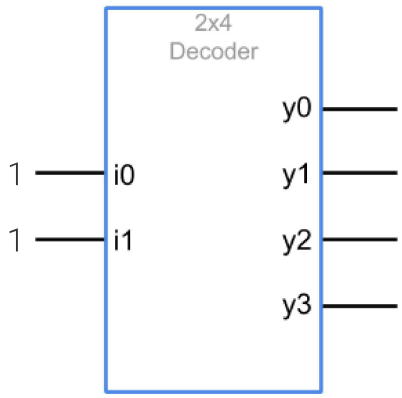
= _____ .

Check Show answer

CHALLENGE ACTIVITY 2.10.1: Decoder input and output.

347136.1658324.qx3zqy7

Start



y3y2y1y0 = Ex: 0001

1 2 3 4 5

Check Next

2.11 Encoders

Encoders

An ***encoder*** is a combinational circuit that converts 1 of N inputs to a binary value using $log(\mathrm{N})$ outputs. An encoder has the reverse functionality of a decoder. A ***4x2 encoder***, spoken as "4 to 2 encoder", outputs a binary value on the two output lines indicating which of the four inputs was 1. An

encoder assumes that exactly one input will be 1. The truth table in the following animation shows the functionality of an encoder with four inputs and two outputs.

PARTICIPATION ACTIVITY 2.11.1: A 4x2 encoder.

Animation content:

undefined

Animation captions:

1. A 4x2 encoder outputs a binary value indicating which input is 1. If d0 is 1, the encoder outputs the binary value for 0, e1e0 = 00.
2. If d1 is 1, the encoder outputs the binary value for 1, e1e0 = 01.
3. If d2 is 1, the encoder outputs the binary value for 2, e1e0 = 10.
4. If d3 is 1, the encoder outputs the binary value for 3, e1e0 = 11.

PARTICIPATION ACTIVITY 2.11.2: Encoder.

1) For a 4x2 encoder, if d3d2d1d0 = 0100, then e1e0 = ____. Type Unknown if unknown.

Check Show answer

2) What values of d3d2d1d0 for a 4x2 encoder will result in an output of e1e0 = 01?

Check Show answer

3) For a 4x2 encoder, if d3d2d1d0 = 0011, then e1e0 = ____. Type Unknown if unknown.

Check Show answer

4) For a 4x2 encoder, if d3d2d1d0 = 0000, then e1e0 = _____. Type Unknown if unknown.

Check Show answer

5) How many inputs does an 8x3 encoder have?

Check Show answer

6) How many outputs does a 16x4 encoder have?

Check Show answer

7) For an 8x3 encoder, what is the output, e2e1e0, if only input d7 is 1? Type Unknown if unknown.

Check Show answer

8) For an 8x3 encoder, what is the output, e2e1e0, if only input d2 is 1? Type Unknown if unknown.

Check Show answer

9) For an 8x3 encoder, which input is 1 if the output e2e1e0 = 000? Type Unknown if unknown.

Check Show answer

10) For a 16x4 encoder, what is the

output if only input d5 is 1?

Check Show answer

Priority encoders

A ***priority encoder*** is a combinational circuit that outputs the binary value for the highest priority input that is 1, where a higher numbered inputs has higher priority Ex: A 4x2 priority encoder has four inputs d3d2d1d0, where d3 has the highest priority and d0 has the lowest.

More than one input of priority encoder can be 1. When multiple inputs are one, the binary value for the highest priority input will be output. Ex: If the input is d3d2d1d0 = 1010, d3 has priority over d1, so the output is e1e0 = 11. A priority encoder assumes that at least one input will be 1.

PARTICIPATION ACTIVITY 2.11.3: A 4x2 priority encoder.

Animation content:

undefined

Animation captions:

1. If d0 = 1, and is the only input that is 1, e1e0 = 00.
2. Input d1 has priority over input d0. If d1 = 1, e1e0 = 01, regardless of the value of input d0.
3. Input d2 has priority over inputs d1 and d0. If d2 = 1, e1e0 = 10, regardless of the value of inputs d1 and d0.
4. The input d3 takes priority than inputs d2, d1 and d0. If d3 = 1, e1e0 = 11, regardless of the value of inputs d2, d1 and d0.

PARTICIPATION ACTIVITY 2.11.4: Priority Encoder.

Consider a 4x2 priority encoder. Given d3d2d1d0, determine the value of e1e0.

1) d3d2d1d0 = 1000

- 01
- 11
- Unknown

2) d3d2d1d0 = 0101

- 00
- 10
- Unknown

3) d3d2d1d0 = 0111

- 00
- 01
- 10

4) d3d2d1d0 = 0000

- 00
- 11
- Unknown

5) d3d2d1d0 = 1111

- 00
- 11
- Unknown

PARTICIPATION ACTIVITY 2.11.5: Larger priority encoders.

1) For an 8x3 priority encoder, what is the output, e2e1e0, if the inputs d0, d3, and d7 are 1? Type Unknown if unknown.

Check Show answer

2) For an 8x3 priority encoder, what is the output, e2e1e0, if inputs d4 and d2 are 1? Type Unknown if unknown.

Check Show answer

3) For an 8x3 priority encoder, how many inputs are 1 if the output

e2e1e0 = 011? Type Unknown if unknown.

Check Show answer

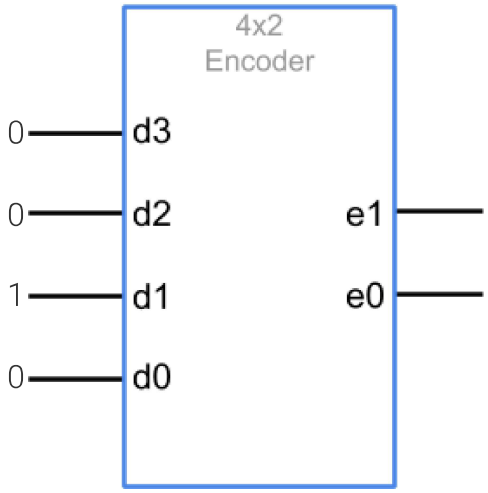
4) For a 16x4 priority encoder, what is the output if inputs d9 and d11 are 1?

Check Show answer

CHALLENGE ACTIVITY 2.11.1: Encoder inputs and outputs.

347136.1658324.qx3zqy7

Start



e1e0 = Ex: 10

1 2 3 4 5 6

Check Next

2.12 Don't cares

Incompletely specified functions

An ***incompletely specified function*** does not define an output value for every input combination. Ex: A 3-position knob may set 2 inputs to 00, 01, or 10. Combination 11 is not possible and thus f is not specified for that combination.

All possible minterms of a function can be divided into an ***on set*** (function outputs 1), ***off set*** (function outputs 0), and ***don't care set*** (function output is not specified). In a truth table or on a K-map, don't care minterms are indicated with an X.

PARTICIPATION ACTIVITY 2.12.1: An incompletely specified function.

Animation captions:

1. This function specifies an output value of 0 for a'b', 1 for a'b, and 1 for ab', but does not specify an output value for ab.
2. Such a situation might arise, for example, if the inputs ab are controlled by a switch having only three positions, thus setting ab with 00, 01, or 10. 11 isn't possible.
3. The possible minterms can be divided into an off set, on set, and don't care set.

PARTICIPATION ACTIVITY 2.12.2: Incompletely specified functions.

Indicate whether the function f having inputs a, b is completely specified.

1) f is 1 for a'b', 1 for a'b, 1 for ab', and 0 for ab.
   - ○ Complete
   - ○ Incomplete

2) f is 1 for a'b', and 0 otherwise.
   - ○ Complete
   - ○ Incomplete

3) f is 1 for a'b', and 1 for ab.
   - ○ Complete
   - ○ Incomplete

4) f is 1 for a'b', 1 for a'b, and 0 for ab'.

- Complete
- Incomplete

5) For a'b', f is 1. For ab, the output value doesn't matter. All other combinations output 0.

- Complete
- Incomplete

Minimizing with don't care minterms

Due to the nature of digital circuits, a circuit will output either 0 or 1 for every input value combination. Thus, even for an incompletely specified function, a designer must still choose whether to output 0 or 1 for each don't care minterm. Commonly, designers make the choice that yields a minimized circuit.

PARTICIPATION ACTIVITY 2.12.3: Minimizing with don't care minterms: Choosing a value that minimizes circuit size.

Animation captions:

1. The X indicates a don't care minterm. The designer must choose to output either 0 or 1.
2. By outputting 0 for that minterm, the minimized expression has two terms with two literals each.
3. By outputting 1, the minimized expression has two terms with only one literal each. With respect to circuit size, outputting 1 is a better choice than 0 for this don't care minterm.

When drawing circles on a K-map, a designer can choose whether an X (don't care) should be 0 or 1. If outputting a 1 allows for a larger circle, then 1 is a better choice, leading to fewer literals in a term. Otherwise, outputting 0 is a better choice, leading to fewer terms.

PARTICIPATION ACTIVITY 2.12.4: For don't cares (X's), designers choose to output 1 if that enables a larger circle (and thus smaller term), else choose to output 0 to yield fewer circles (and thus fewer terms).

Animation captions:

1. Setting the bottom X as 1 allows for a bigger circle, yielding term ab, rather than abc'.
2. Setting the upper X to 1 would just result in an extra term (a'b'c). That X doesn't allow for a larger circle, so X is set to 0.

PARTICIPATION ACTIVITY 2.12.5: Minimizing with don't care minterms.

Indicate whether each X should be chosen to output 0 or 1.

i \ jk	00	01	11	10
0	0	0	Xa	1
1	0	Xb	1	1

i \ jk	00	01	11	10
0	0	Xc	1	0
1	0	Xd	1	0

1) Xa

- 0
- 1

2) Xb

- 0
- 1

3) Xc

- 0
- 1

4) Xd

- 0
- 1

Warning

Many designers recommend avoiding incompletely-specified functions (don't cares) except in rare cases. Even though an input combination should never appear, the combination might possibly appear, even briefly—maybe due to electrical noise, due to glitches while switching, or during startup. If the output really does matter in that case, then the designer should choose a safe value. Ex: For the earlier three-position knob example, if f = 1 turns on a device, then 11 is not really a don't care, because the device should not turn on. So f should be set to 0 for input 11, just in case a 11 accidentally appears.

Example: Electronic die

A die can display values 1-6. When rolled, a circuit lights an LED if a high-number is rolled, defined as 4-6. If the die's value is input to the circuit in binary as 3 bits, then combinations 000 and 111 (0 and 7) can't occur. Thus, 000 and 111 are don't care minterms.

PARTICIPATION ACTIVITY 2.12.6: Electronic die: 0 and 7 are don't care minterms.

Animation captions:

1. This die has six sides, with 1, 2, 3, 4, 5, or 6 dots. A sensor provides the binary value on 3 inputs a, b, c, to a circuit.
2. If the input is 4, 5, or 6, the output should be 1 to light an LED, indicating a high roll. If 1, 2, or 3, the output should be 0.
3. Input combinations 000 and 111 can't occur, so are don't care minterms.
4. A designer places the 1's, 0's, and X's on the K-map. If the X's were both 0's, two circles (and thus two terms) would be needed.
5. By making the X in cell 111 a 1, only one (larger) circle is needed, yielding only one (simpler) term. The X in cell 000 should be 0, else an extra circle would be needed.

PARTICIPATION ACTIVITY 2.12.7: Don't cares example: Electronic die.

Consider the example above.

1) If the designer makes the X in cell 111 a 0, what is f after covering the 1's with circles?
   - ab'
   - ac
   - ab' + ac'

2) If the designer makes the X in cell 111 a 1, what is f after covering the 1's with circles?
   - a'
   - a
   - ab' + ab

3) If the designer makes the X in cell 000

a 1, what additional term would result?

- b'c'
- None

CHALLENGE ACTIVITY 2.12.1: Don't cares.

347136.1658324.qx3zqy7

Start

To minimize the cover, what value should the don't care be set to?

a \ bc	00	01	11	10
0	0	X	1	0
1	1	0	0	0

X = Ex: 0 or 1

1 2 3

Check Next

2.13 Prime implicants and minimal covers

On-sets and implicants

Several mathematical concepts underlie techniques for two-level logic simplification.

- A function's ***on-set*** is the set of minterms that define when the function is 1. Ex: f = a'b'c + a'bc + ab'c' + abc + abc' has 5 minterms in the function's on-set, as listed.

- A term ***covers*** a minterm if the term evaluates to 1 whenever the minterm does. Ex: Term ab covers minterm abc, as well as minterm abc'.
- An ***implicant*** of a function is a term that covers only minterms in the function's on-set. Ex: For the above function, ab is an implicant due to covering abc and abc', both of which are in the function's on-set , and covering no other minterms. a'b' is not an implicant, due to covering minterm a'b'c', which is not in the function's on-set.

On a K-map, each 1 represents a minterm in a function's on-set, and each valid circle (not necessarily largest) is an implicant.

PARTICIPATION ACTIVITY 2.13.1: Implicants of a function. Each valid K-map circle represents an implicant.

Animation captions:

1. A function can be defined as a sum of minterms. Those minterms form the function's on-set. Each minterm is a 1 on a K-map.
2. An implicant of a function is a term that covers only minterms in the on-set. Minterms are obviously implicants. ab is also an implicant.
3. Each implicant is a valid circle on the K-map. Circling one 1 yields an implicant that is a minterm. Circling two 1's also yields an implicant, in this case ab.
4. Each valid K-map circle represents an implicant.

PARTICIPATION ACTIVITY 2.13.2: Identifying implicants of a function.

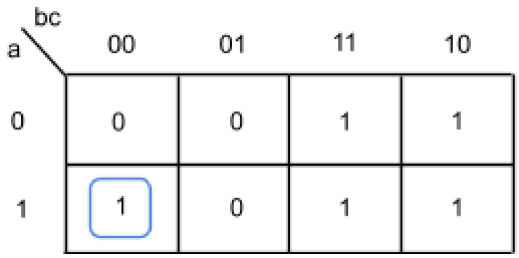
Consider the following K-map for function F = ab'c' + a'bc + abc + a'bc' + abc'.

a \ bc	00	01	11	10
0	0	0	1	1
1	1	0	1	1

1) The function's on-set consists of how many minterms?

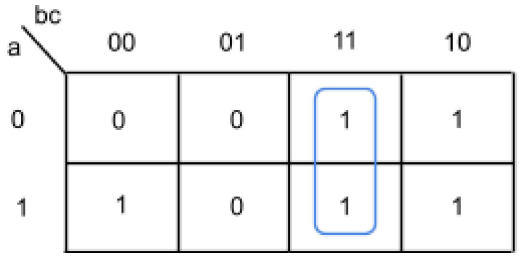
- 2
- 5
- 8

2) What implicant of the function does the shown circle represent?



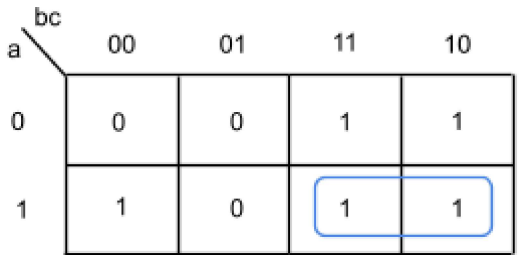
- a
- ab'c'
- abc

3) What implicant of the function does the shown circle represent?



- bc
- a'bc + abc
- Not an implicant

4) What minterms are covered by the shown implicant?



- ab
- abc, abc'
- Not an implicant

5) Is b an implicant of the function?

- Yes
- No

6) Is a' an implicant of the function?

- Yes
- No

7) How many possible implicants does the function have?

- 2
- 5
- 11

Prime implicants

A ***prime implicant*** of a function is an implicant that cannot have a literal removed without becoming a non-implicant. On a K-map representation of a function, each *largest* possible valid circle represents a *prime* implicant.

PARTICIPATION ACTIVITY 2.13.3: Prime implicants: Largest possible K-map circles.

Animation captions:

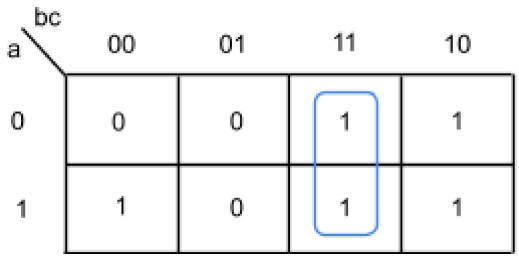
1. a'b'c is an implicant, but is not prime, because a literal (b') can be removed and still yield an implicant (a'c).
2. a'c is a prime implicant. No literal can be removed and still yield an implicant.
3. All the largest circles on a K-map represent all the prime implicants of a function: ac', a'c, ab, and bc.

PARTICIPATION ACTIVITY 2.13.4: Prime implicants.

Consider the following K-map for function F = ab'c' + a'bc + abc + a'bc' + abc'.

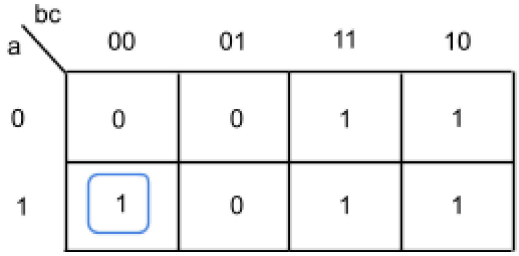
a \ bc	00	01	11	10
0	0	0	1	1
1	1	0	1	1

1) Does this circle represent a prime implicant?



- Yes
- No

2) Does this circle represent a prime implicant?



- Yes
- No

Essential prime implicants and minimal covers

A minimal cover is useful to create the smallest circuit for the function. A function's ***minimal cover*** is an expression for the function, having the fewest terms, each with the fewest literals.

Defining prime implicants (PI's) above was useful because each term in a minimum cover *must* be a prime implicant (else, the cover is not minimal). On a K-map, a minimal cover is a cover using the fewest, largest circles, and largest circles are PI's.

An ***essential prime implicant*** is the only prime implicant to cover a particular minterm in a function's on-set. On a K-map, an essential PI is a largest circle that is the only circle to cover a particular 1. Essential PI's must be in the function's cover.

An algorithm to create a minimal cover is:

1. *Generate PI's:* Find all PI's by drawing all largest possible circles.
2. *Add essential PI's:* Find any essential PI's and add them to the function's cover.
3. *Cover remaining:* Select PI's to cover any 1's not covered by the essential PI's.

More than one minimal cover may exist, as in the animation below (either the purple or brown circle could be used).

PARTICIPATION ACTIVITY 2.13.5: Essential prime implicants and minimal cover.

Animation content:

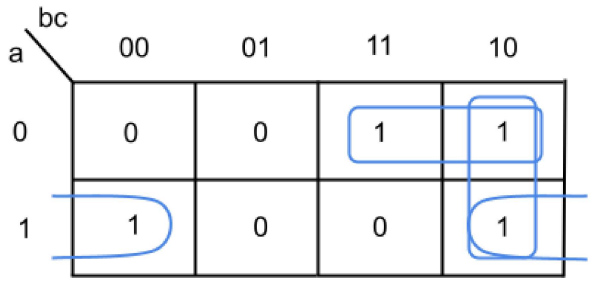
undefined

Animation captions:

1. Each circle is a prime implicant. Goal is to find a minimal cover: The fewest and largest circles.
2. The orange circle is the only circle to cover the bottom left 1, so is essential. The red circle is also essential. Both must be added to the cover.
3. The essential PI's cover four of the five 1's. The remaining 1 should be covered with the fewest largest remaining PI's. Either the green or purple circle can be chosen.

PARTICIPATION ACTIVITY 2.13.6: Essential prime implicants and minimal covers.

Consider the given K-map and prime implicants.



1) Which prime implicant is essential?

- ac'
- bc'
- b

2) Which prime implicant is essential?

- bc'
- a'b
- a'bc

3) After including essential prime implicants in the minimal cover, how many remaining prime implicants must be added?

- 0
- 1

○ 2

K-map minimization usually combines steps

When minimizing a function by hand using a K-map, designers typically combine the above steps, drawing essential PI circles first (covering obviously standalone 1's), then drawing the fewest largest circles to cover remaining 1's. But the above 3 steps are useful when automating the minimization like in the "Quine-McCluskey algorithm" (described in another section), which is especially important for larger functions such as those with 5 inputs or more.

CHALLENGE ACTIVITY 2.13.1: Prime implicants.

347136.1658324.qx3zqy7

Start

Enter all the implicants for the K-map.

a \ bc	00	01	11	10
0	0	0	0	0
1	1	0	0	1

Implicants: Ex: ab, bc'

Enter implicants with literals in alphabetical order. Separate implicants with a comma. Ex: ab, bc'

1 2 3 4

Check Next

2.14 Quine-McCluskey

Basic algorithm

Quine-McCluskey is an algorithm for two-level logic optimization, suitable for computer automation due to using a tabular method (rather than graphical method like K-maps). Given a function's minterms, the algorithm's steps are:

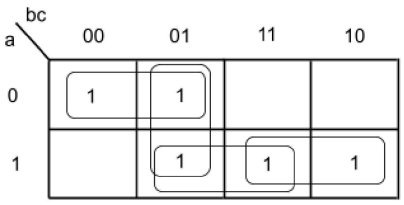
1. *Generate PI's:* Create a table of minterms, then pairwise check minterms for i(j + j') opportunities, combining into new terms in a new column, repeating with new terms until no more combinations can be made. Each term that wasn't combined with another (minterms or new terms) is a prime implicant (PI).
2. *Find essentials:* Draw a table with PI's as rows and minterms as columns, putting a mark to indicate a PI covers a minterm. For any column with only one mark, the PI for that row is essential so is added to the cover. All minterms covered by that PI are also checked off as covered.
3. *Cover remaining:* Select minimal unadded prime implicants to cover remaining minterms.

PARTICIPATION ACTIVITY 2.14.1: Quine-McCluskey algorithm.

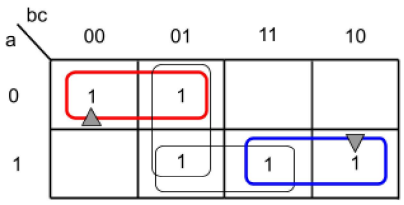
Animation captions:

1. Step 1: Generate all PI's. Create a table of minterms. Check all pairs for i(j + j'), write simpler term.
2. Step 1 (cont): Repeat until no further combinations possible. Uncombined terms are PI's (in this case just second column). Terms list the covered minterms: a'b' (0,1) covers minterms 0 and 1.
3. Step 2. Find essential PI's: Create a table with PI's as rows and minterms as column. Mark column if PI covers minterm.
4. Step 2 (cont): If only one mark exists in a column, covering PI is essential, must add to cover. Ex: Column 0 only has one mark, so a'b' is essential. Check off all minterms covered by that PI as covered.
5. Step 3. Cover remaining: If any minterms remain uncovered, choose fewest/smallest remaining PIs to complete the cover. Here, b'c is arbitrarily choosen (choosing ac instead is also OK).

Figure 2.14.1: Equivalent K-map operations for the above Quine-McCluskey steps.

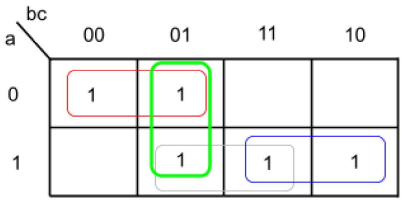


Step 1: Generate PI's



Step 2: Add essential PI's

f = a'b' + ab



Step 3: Cover remaining

f = a'b' + ab + b'c

PARTICIPATION ACTIVITY 2.14.2: Quine-McCluskey: Simple 3-variable example.

Consider the partially-completed Quine-McCluskey algorithm tables below.

On-set: 0, 1, 7

a'b'c' (0) — a'b' (0,1)	
a'b'c (1)	
abc (7)	

Step 1: Generate PI's

	0	1	7
a'b' (0,1)	o	o	
abc (7)			o

PI's

Step 2: Add essential PI's

f = a'b' + abc

Step 3: Cover remaining

1) If all minterm pairs were checked for an i(j + j') opportunity, how many checks were made?

- 1
- 2
- 3

2) How many checks for i(j + j') opportunities are done with a'b'?

- 0
- 1

3) How many PI's were generated?

- 1
- 2

4) How many of the PI's are essential?

- 1
- 2

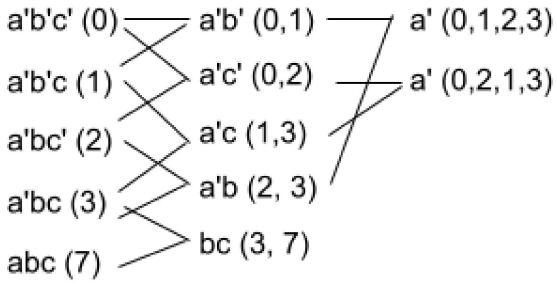
5) How many of the PI's should be added in Step 3 to complete the cover?

- 0
- 1

PARTICIPATION ACTIVITY 2.14.3: Quine-McCluskey: Another 3-variable example.

Consider the partially-completed Quine-McCluskey algorithm tables below.

On-set: 0, 1, 2, 3, 7



Step 1: Generate PI's

PI's	0	1	2	3	7
a' (0,1,2,3)	o	o	o	o	
bc (3,7)				o	o

Step 2: Add essential PI's

1) If all minterm pairs were checked for an i(j + j') opportunity, how many checks were made?

- 5
- 10

- 25

2) How many two-literal prime implicants are generated?
   - 1
   - 2
   - 5

3) The left table lists three implicants that aren't combined with another: bc, a', and a'. Why does the right table only list two PI's?
   - Mistake
   - a' and a' are the same

4) How many PI's are essential?
   - 1
   - 2

5) Is Step 3 necessary?
   - No
   - Yes

Grouping terms by number of uncomplemented literals

Note: The above algorithm checked all minterm pairs (and then all new term pairs in the next column, etc.) for an i(j + j') opportunity. However, for an i(j + j') opportunity to exist, the terms must differ by exactly one literal. As such, an improvement to the algorithm's efficiency is to only check pairs whose number of uncomplemented literals differs by one. Ex: a'b'c has one uncomplemented literal (c) while abc has three (a, b, c). Because the difference is not one, checking that pair can be skipped. Thus, for a given column, the Quine-McCluskey algorithm first sorts terms into a group with zero uncomplemented literals, a group with one, a group with two, etc. Then, for a given term, the algorithm only compares with terms in the next group.

Petrick's method

Note: Step 3 requires an algorithm itself to find the minimum cover. A straightforward approach is called Petrick's method. The method is beyond this material's scope.

Exploring further:

- Online Quine-McCluskey minimizer (T. Thormaehlen)
- Article on Quine-McCluskey (embedded.com)
- Petrick's Method (Wikipedia)