CDA 3103: Study Set 2

TRUTH TABLES, BOOLEAN ALGEBRA, CANONICAL FORMS, INCOMPLETELY SPECIFIED FUNCTIONS, LOGIC MINIMIZATION, KARNAUGH MAPS, LOGIC GATES, COMBINATIONAL LOGIC DESIGN

Review: Voltage

- Physically, each bit is an individual transistor-wire in the system.
- The value of the bit is determined by its voltage.
- Low voltage is associated with 0s.
- High voltage is associated with 1s.

Review: Truth Tables

- Truth tables are used to completely specify a combinational hardware element
  - (commonly called a “logic block”).
- A logic block with *n* inputs has 2^n entries in the truth table.

Example: Truth Tables

Given: Complete a truth table that has 3 inputs (A, B, and C) and one output (F). F is asserted whenever B is asserted but A is not. F is also asserted whenever C is asserted. F is deasserted in all other cases.

Partial Credit 1: A truth table with 3 inputs should have 8 entries. List every possible combination of inputs in the input columns. Using binary values 0-7 for entries 0-7 is a quick way to make sure each entry is different. There is also one output column.

Solution 1: Set up the truth table with the correct number of entries, input columns, and output columns. Fill in the inputs.

Truth Table			
A	B	C	F
0	0	0	____
0	0	1	____
0	1	0	____
0	1	1	____
1	0	0	____
1	0	1	____
1	1	0	____
1	1	1	____

Example: Truth Tables

Given: Complete a truth table that has 3 inputs (A, B, and C) and one output (F). F is asserted whenever B is asserted but A is not. F is also asserted whenever C is asserted. F is deasserted in all other cases.

Partial Credit 2: Consider the first statement: F is asserted whenever B is asserted but A is not.

Solution 2: Fill in the outputs for this statement

Truth Table			
A	B	C	F
0	0	0	____
0	0	1	____
0	1	0	__1__
0	1	1	__1__
1	0	0	____
1	0	1	____
1	1	0	____
1	1	1	____

Example: Truth Tables

Given: Complete a truth table that has 3 inputs (A, B, and C) and one output (F). F is asserted whenever B is asserted but A is not. F is also asserted whenever C is asserted. F is deasserted in all other cases.

Partial Credit 3: Then, the following statement: F is also asserted whenever C is asserted

Solution 3: Fill in the outputs for this statement

Truth Table			
A	B	C	F
0	0	0	____
0	0	1	__1__
0	1	0	__1__
0	1	1	__1__
1	0	0	____
1	0	1	__1__
1	1	0	____
1	1	1	__1__

Example: Truth Tables

Given: Complete a truth table that has 3 inputs (A, B, and C) and one output (F). F is asserted whenever B is asserted but A is not. F is also asserted whenever C is asserted. F is deasserted in all other cases.

Partial Credit 4: Finally: F is deasserted in all other cases.
Any output entry without a 1 should have a 0.

Solution 4: Fill in the outputs for this statement

Truth Table			
A	B	C	F
0	0	0	__0__
0	0	1	__1__
0	1	0	__1__
0	1	1	__1__
1	0	0	__0__
1	0	1	__1__
1	1	0	__0__
1	1	1	__1__

Review: Boolean Algebra

- Boolean Algebra is also used to describe combinational hardware elements. Each logic block has at least one corresponding algebraic function, but it may have more than one as many functions are equivalent.
- Boolean Algebra may use any number of variables, but only two values (0 and 1) and three operators (NOT, AND, and OR).
- In addition to describing a logic block, both truth tables and Boolean algebra can be used to prove that two functions are equivalent. This is useful for minimizing our functions.

Example: Proof by Perfect Induction

Given: Use the following truth table to prove that the specified Uniting Law is true.
Uniting Law: XY + XY' = X

Partial Credit 1: In proof by perfect induction we need to show that the column for x and the column for XY + XY' have the same values. This proves that they are equivalent. We use the other columns to complete each Boolean operator in X*Y + X*Y'. This helps us ensure we have the correct final values for the equation.

Truth Table					
X	**Y**	**X*Y**	**Y'**	**X*Y'**	**X*Y+X*Y'**
0	0	___	___	___	___
0	1	___	___	___	___
1	0	___	___	___	___
1	1	___	___	___	___

Example: Proof by Perfect Induction

Given: Use the following truth table to prove that the specified Uniting Law is true.
Uniting Law: XY + XY' = X

Partial Credit 1: In proof by perfect induction we need to show that the column for x and the column for XY + XY' have the same values. This proves that they are equivalent. We use the other columns to complete each Boolean operator in X*Y + X*Y'. This helps us ensure we have the correct final values for the equation.

Solution 1: Fill in the values for X*Y (X AND Y)

Truth Table					
X	**Y**	**X*Y**	**Y'**	**X*Y'**	**X*Y+X*Y'**
0	0	_0_	___	___	___
0	1	_0_	___	___	___
1	0	_0_	___	___	___
1	1	_1_	___	___	___

Example: Proof by Perfect Induction

Given: Use the following truth table to prove that the specified Uniting Law is true.
Uniting Law: XY + XY' = X

Solution 2: Fill in the values for Y' (NOT Y)

Truth Table					
X	**Y**	**X*Y**	**Y'**	**X*Y'**	**X*Y+X*Y'**
0	0	_0_	_1_	___	___
0	1	_0_	_0_	___	___
1	0	_0_	_1_	___	___
1	1	_1_	_0_	___	___

Example: Proof by Perfect Induction

Given: Use the following truth table to prove that the specified Uniting Law is true.
Uniting Law: XY + XY' = X

Solution 3: Fill in the values for XY' (X AND NOT Y)

Truth Table					
X	**Y**	**X*Y**	**Y'**	**X*Y'**	**X*Y+X*Y'**
0	0	_0_	_1_	_0_	___
0	1	_0_	_0_	_0_	___
1	0	_0_	_1_	_1_	___
1	1	_1_	_0_	_0_	___

Example: Proof by Perfect Induction

Given: Use the following truth table to prove that the specified Uniting Law is true.
Uniting Law: XY + XY' = X

Solution 4: Fill in the values for XY + XY'
(X AND Y) OR (X AND NOT Y)
by performing an OR operation on the XY column and the XY' column.

Truth Table					
X	**Y**	**X*Y**	**Y'**	**X*Y'**	**X*Y+X*Y'**
0	0	_0_	_1_	_0_	_0_
0	1	_0_	_0_	_0_	_0_
1	0	_0_	_1_	_1_	_1_
1	1	_1_	_0_	_0_	_1_

Example: Proof by Rewrite

Given: Consider the following proof by rewrite. For each step, identify which Boolean axiom has been applied.

F = A’B’C’ + A’BC’ + AB’C + ABC’ + ABC

F = A’B’C’ + A’BC’ + AB’C + ABC’ + ABC + A’BC’

F = AC + A’B’C’ + A’BC’ + ABC’ + A’BC’

F = AC + BC’ + A’B’C’ + A’BC’

F = AC + BC’ + A’C’

Example: Proof by Rewrite

Given: Consider the following proof by rewrite. For each step, identify which Boolean axiom has been applied.

F = A’B’C’ + A’BC’ + AB’C + ABC’ + ABC

Solution 1:

F = A’B’C’ + A’BC’ + AB’C + ABC’ + ABC + A’BC’ Idempotency Law

F = AC + A’B’C’ + A’BC’ + ABC’ + A’BC’

F = AC + BC’ + A’B’C’ + A’BC’

F = AC + BC’ + A’C’

Example: Proof by Rewrite

Given: Consider the following proof by rewrite. For each step, identify which Boolean axiom has been applied.

Solution 2:

F = A’B’C’ + A’BC’ + AB’C + ABC’ + ABC

F = A’B’C’ + A’BC’ + AB’C + ABC’ + ABC + A’BC’ — Idempotency Law

F = AC + A’B’C’ + A’BC’ + ABC’ + A’BC’ — Uniting Law

F = AC + BC’ + A’B’C’ + A’BC’

F = AC + BC’ + A’C’

Example: Proof by Rewrite

Given: Consider the following proof by rewrite. For each step, identify which Boolean axiom has been applied.

Solution 3:

F = A’B’C’ + A’BC’ + AB’C + ABC’ + ABC	
F = A’B’C’ + A’BC’ + AB’C + ABC’ + ABC + A’BC’	Idempotency Law
F = AC + A’B’C’ + A’BC’ + ABC’ + A’BC’	Uniting Law
F = AC + BC’ + A’B’C’ + A’BC’	Uniting Law
F = AC + BC’ + A’C’	

Example: Proof by Rewrite

Given: Consider the following proof by rewrite. For each step, identify which Boolean axiom has been applied.

F = A’B’C’ + A’BC’ + AB’C + ABC’ + ABC	
F = A’B’C’ + A’BC’ + AB’C + ABC’ + ABC + A’BC’	Idempotency Law
F = AC + A’B’C’ + A’BC’ + ABC’ + A’BC’	Uniting Law
F = AC + BC’ + A’B’C’ + A’BC’	Uniting Law
Solution 4: F = AC + BC’ + A’C’	Uniting Law

Example: Proof by Rewrite

Given: Use the axioms and laws of Boolean Algebra to prove that A'B'C' + A'B'C + AB'C' + AB'C + ABC' is equivalent to AC' + B'.

Partial Credit 1: In proof by rewrite we apply the axioms and laws of Boolean Algebra to manipulate our starting algebraic statement into new forms. There are often several ways to do this, so credit is given to any valid manipulation that makes progress towards the end statement. Since our given end statement has fewer terms and each term has fewer literals than the given start statement, we should assume that the uniting law will be used several times. Let's determine which groups can be combined with the uniting law.

Solution 1: A'B'C' + A'B'C + AB'C' + AB'C + ABC' -> A'B'C' + A'B'C + AB'C' + AB'C + ABC'

A'B'C' + A'B'C + AB'C' + AB'C + ABC'

Since AB'C' participates in two different groupings, we should make a copy of it.

Example: Proof by Rewrite

Given: Use the axioms and laws of Boolean Algebra to prove that A'B'C' + A'B'C + AB'C' + AB'C + ABC' is equivalent to AC' + B'.

Partial Credit 2: Now that we have determined which groups can be combined with the uniting law, we can make a duplicate of AB'C' and unite until we reach the target equation.

Solution 2:

A'B'C' + A'B'C + **AB'C'** + AB'C + ABC' → A'B'C' + A'B'C + **AB'C'** + AB'C + ABC' + **AB'C**

Idempotency Law

A'B'C' + A'B'C + AB'C' + **AB'C** + ABC' + **AB'C'** → A'B'C' + A'B'C + AB'C' + **AB'** + ABC'

Uniting Law

A'B'C' + **A'B'C** + AB'C' + AB' + ABC' → **A'B'** + AB'C' + AB' + ABC'

Uniting Law

A'B' + **AB'C'** + AB' + **ABC'** → A'B' + **AC'** + AB'

Uniting Law

A'B' + AC' + **AB'** → AC' + **B'**

Uniting Law

Review: Canonical Forms

- Canonical forms are a unique algebraic signature for logic blocks specified by truth tables.
- A logic block can have multiple equivalent equations, but will only have two canonical equations.

Review: Canonical Forms

- Sum-of-Products form is a summation of product terms that describes the function
  - Other names: minterm expansion, disjunctive normal form
  - A “minterm” is a shorthand notation for product terms based on the entry (m0 for entry 0 (000), m4 for entry 4 (100), m7 for entry 7 (111), etc.)
  - To construct: examine the output column of the truth table. Each entry that has a 1 has a minterm in the output equation. Minterms are written based on the values of the input. A 0 corresponds to a negated literal, but a 1 is a natural literal. Multiply these literals to form a product term. If your inputs are A, B, and C then entry 101 is minterm m5 = AB’C. Sum each minterm to create a final sum-of-products equation.

Review: Canonical Forms

- Product-of-Sums form is a product of sum terms that describes the function
  - Other names: maxterm expansion, conjunctive normal form
  - A “maxterm” is a shorthand for sum terms
    (M0 for entry 0 (000), M4 for entry 4 (100), M7 for entry 7 (111), etc.)
  - To construct: examine the output column of the truth table. Each entry that has a 0 has a maxterm in the output equation. Maxterms are written based on the values of the input. A 1 corresponds to a negated literal, but a 0 is a natural literal. Add these literals together to form a sum term. If your inputs are A, B, and C then entry 101 is maxterm M5 = A’ + B + C’. Multiply each maxterm to create a final product-of-sums equation.
  - Maxterms are inverses of minterms.

Example: Canonical Forms

Given: Given the following truth table, determine the sum-of-products and product-of-sums representation.

Partial Credit 1: Sum-of-products is the more common of the two canonical forms. To construct, look for 1's in the output column. Each corresponds to a minterm – product term – in the output.

Solution 1:

$$F = \sum m(0, 1, 2, 4, 6)$$
$$= m0 + m1 + m2 + m4 + m6$$
$$= A'B'C' + A'B'C + A'BC' + AB'C' + ABC'$$

Truth Table			
A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Example: Canonical Forms

Given: Given the following truth table, determine the sum-of-products and product-of-sums representation.

Partial Credit 2: To construct a product-of-sums equation, look for 0's in the output column. Each corresponds to a maxterm – sum term – in the output. Remember that maxterms are inverses of minterms!

Solution 2:

$$F = \prod M(3, 5, 7)$$
$$= M3 + M5 + M7$$
$$= (A + B' + C') * (A' + B + C') * (A' + B' + C')$$

Truth Table			
A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Review: Incompletely Specified Functions

- The number of entries in a truth table is always a power of two (1, 2, 4, 8, 16, etc.). But some logic blocks do not require every entry. In our class example we represented decimal digits with binary. Decimal digits range from 0-9. To represent 9 we need to use four bits. Four inputs means we will have 16 entries, where we only need 10 of them. Entries 10-15 will not be used and do not require specification. We use X to denote that they are not specified.
- This adds a new set to our function, called the “don’t-care set”. The set of 1’s is called the “on set” and the set of 0’s is called the “off set”. If a function has “don’t cares” in the output, we need to specify 2 of the three sets in the canonical function.
  - If creating a minterm expansion, use on set (m) and don’t care set (d)
  - If creating a maxterm expansion, use off set (M) and don’t care set (D)
  - D and d represent the same set, the change in case is just convention (upper case for maxterm expansions and lower case for minterm expansions)

Example: Incompletely Specified Functions

Given: Consider the following truth table where F is incompletely specified. Determine the minterm and maxterm expansions for F.

Partial Credit 1: To construct the expansions, we need to include the don't care set along with either the on-set (minterm expansion) or the off-set (maxterm expansion).

Solution 1: The minterm expansion for F = $\sum[m(2, 3, 4) + d(0, 7)]$

The maxterm expansion for F = $\prod [M(1, 5, 6) + D(0, 7)]$

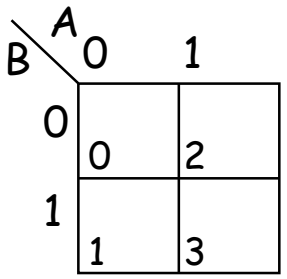
Truth Table			
A	B	C	F
0	0	0	X
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	X

Review: Logic Minimization

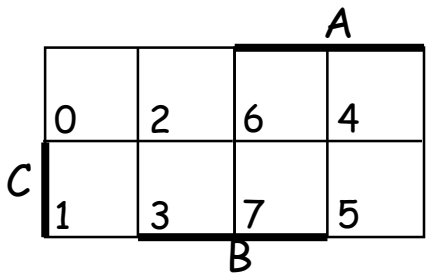
- The goal of logic minimization is to reduce our equations in order to reduce the amount of hardware required to build the system. We want to reduce the number of operators, the number of terms, and the number of literals.
- Our method of reduction is called a Karnaugh map. This “map” allows us to visualize locations where we can apply the uniting theorem to reduce the equation.

Review: Logic Minimization

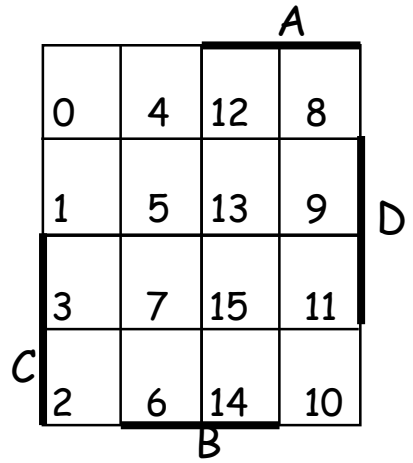
- The maps are rigid in form and based on the number of entries in the truth table:



2-input map



3-input map



4-input map

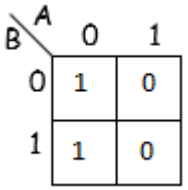
- The numbers shown in the boxes identify which entry each box corresponds to. When filling the map, you take the output from the corresponding entry. Each output column requires its own separate map.

Review: Logic Minimization

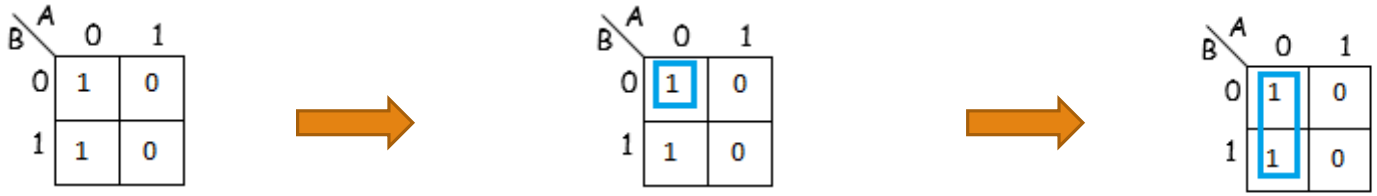
- When the ouputs are placed in the map we can follow a simplification algorithm:
  - Step 1: Choose any element of the ON-set
  - Step 2: Find the maximal groupings of 1s and Xs adjacent to that element
    - consider top/bottom row, left/right column, and corner adjacencies
    - this forms prime implicants
    - groupings must be a size that is a power of 2
  - Repeat Steps 1 and 2 to find all prime implicants
  - Step 3: Revisit the 1s in the K-map
    - if covered by single prime implicant that implicant is essential and must participate in the final cover
    - 1s covered by essential prime implicant do not need to be revisited
  - Step 4: If there are any remaining 1s not covered by essential prime implicants, then choose the smallest number of prime implicants that cover the remaining 1s
- Construct the equation based on the groups. If a literal (A, B, C, D, etc.) changes value anywhere in the group it can be discarded. If a literal always has the value 1 within the group, it participates in its natural form. If a literal always has the value 0 within the group, it participates in its negated form. Multiply all of the literals that participate from the group. This forms a term. Each group contributes one term. Sum all of the terms to create the minimized sum of products form.

Example: Karnaugh Maps

Given: Consider the following Karnaugh map and determine the equation it represents:



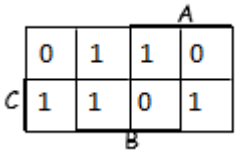
Partial Credit 1: Chose any element of the on set and create the largest possible group while maintaining a size that is a power of 2.



Solution 1: After making this first group, all of the elements of the on set (all of the 1s) are covered. This group is the only group we need. B has the value 0 in the first square and the value 1 in the second square. Since B changes value within the group, it is discarded. A has the value 0 in the first square and the value 0 in the second square. Since it always has the value 0, A' will be in the term. There are no other literals, so the term is A'. There are no other groups, so there are no other terms. **The final equation is A'.**

Example: Karnaugh Maps

Given: Consider the following Karnaugh map and determine the equation it represents:

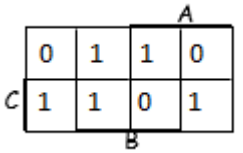


Partial Credit 1: Chose any element of the on set and identify all of the prime implicants at this location.

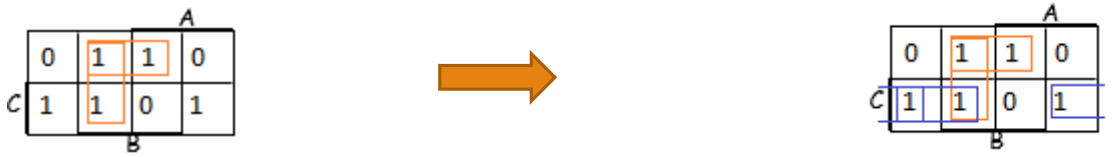
Solution 1: There are two prime implicants around this element. Since some members of the on-set are not covered, we do not know yet which implicants will participate in the minimal cover. We need to identify the rest of the prime implicants.

Example: Karnaugh Maps

Given: Consider the following Karnaugh map and determine the equation it represents:



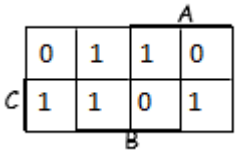
Partial Credit 2: Identify the remaining prime implicants.



Solution 2: Now, all elements of the on-set are covered. It is common for some elements to be in more than one prime implicant.

Example: Karnaugh Maps

Given: Consider the following Karnaugh map and determine the equation it represents:

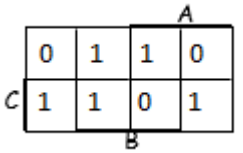


Partial Credit 3: Identify which prime implicants are essential.

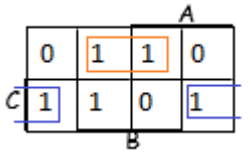
Solution 3: Prime implicants are essential if they contain a 1 that is not covered by any other implicant. These implicants must participate in the final cover. Our final equation must contain the term from the orange group (BC') and the term from the blue group (B'C). We also need to include the final member of the on set.

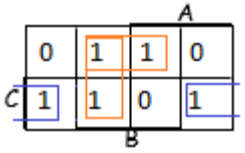
Example: Karnaugh Maps

Given: Consider the following Karnaugh map and determine the equation it represents:

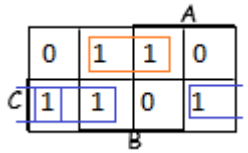


Partial Credit 4: Construct the final equation. Use one of the prime implicants that was not deemed essential.





OR



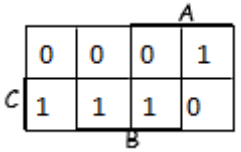
Solution 4: The uncovered 1 is potentially covered by 2 groups, neither of which is essential. We can choose either group to participate in the final cover. This creates two possible correct answers:

Equation 1: BC’ + B’C + A’B

Equation 2: BC’ + B’C + A’C

Example: Karnaugh Maps

Given: Consider the following Karnaugh map and determine the equation it represents:

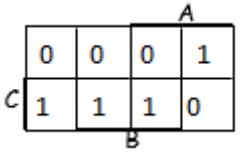


Partial Credit 1: Chose any element of the on set and identify all of the prime implicants at this location.

Solution 1: There is only one prime implicant at this location. Remember! You cannot make a group of size 3. Only powers of 2 will work. We still need to identify the rest of the prime implicants.

Example: Karnaugh Maps

Given: Consider the following Karnaugh map and determine the equation it represents:

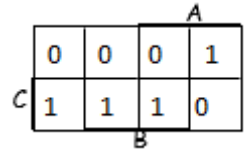


Partial Credit 2: Identify the remaining prime implicants.

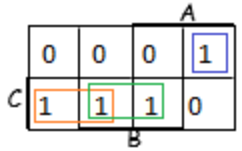
Solution 2: It is sometimes necessary to have a group of size 1. This is still a power of 2: $2^0 = 1$. All of these prime implicants are essential, because each one uniquely covers an element of the on set. We can now construct the final equation.

Example: Karnaugh Maps

Given: Consider the following Karnaugh map and determine the equation it represents:



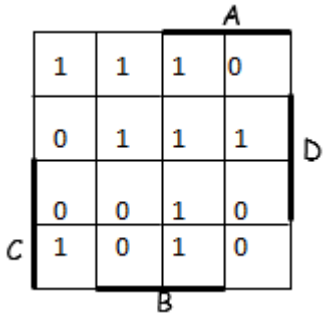
Partial Credit 3: Construct the final equation



Solution 3:
Orange group: A'C
Green group: BC
Blue group: AB'C'
Final Equation: A'C + BC + AB'C'

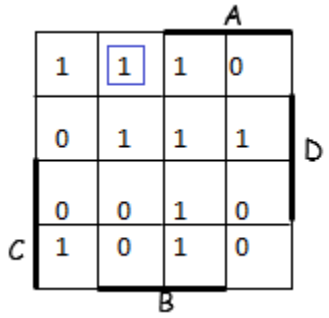
Example: Karnaugh Maps

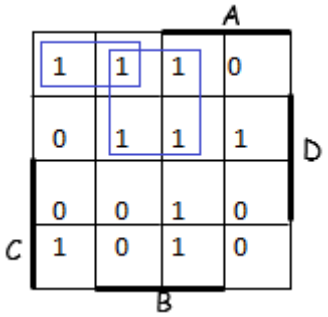
Given: Consider the following Karnaugh map and determine the equation it represents:



Partial Credit 1: Chose any element of the on set and identify all of the prime implicants at this location.

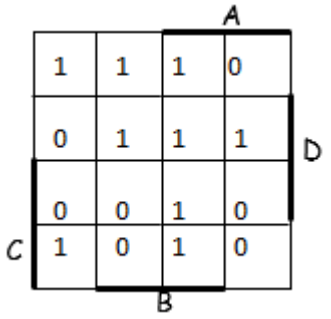
Solution 1:





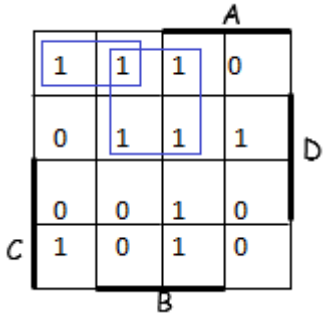
Example: Karnaugh Maps

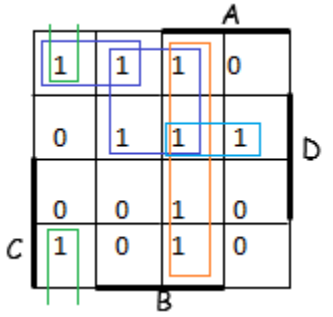
Given: Consider the following Karnaugh map and determine the equation it represents:



Partial Credit 2: Identify the remaining prime implicants.

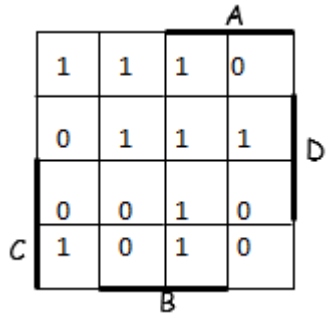
Solution 2:





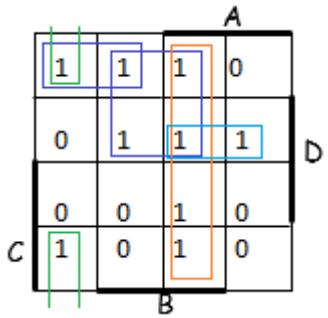
Example: Karnaugh Maps

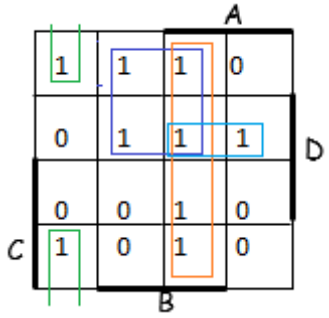
Given: Consider the following Karnaugh map and determine the equation it represents:



Partial Credit 3: Identify which groups are essential.

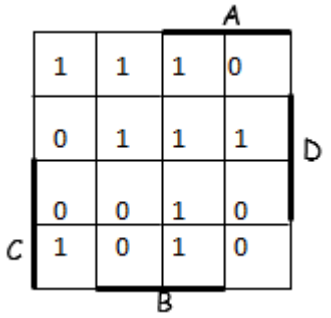
Solution 3:





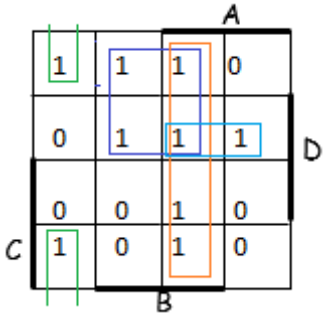
Example: Karnaugh Maps

Given: Consider the following Karnaugh map and determine the equation it represents:



Partial Credit 4: Construct the final equation.

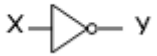
Solution 4: Orange group: AB
Green group: A'B'D'
Dark Blue group: BC'
Light Blue group: AC'D
Final Equation: AB + BC' + A'B'D' + AC'D



Review: Logic Gates

- To obtain the hardware for our system, we need to represent our equations using a gate diagram. With Boolean algebra, we only have three operators: AND, OR, and NOT. We can represent these equations with the corresponding gates:
  - AND Gate 

  - OR Gate 

  - NOT Gate (inverter): 

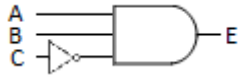
- Visually, instead of Z = X * Y we have an AND gate with X and Y as the inputs and Z as the output.

Example: Logic Gates

Given: Create a gate diagram for the following equation: E = ABC’

Partial Credit 1: Generally, each operator corresponds with one gate. AB implies A*B- following standard algebraic notation – and BC’ implies A*C’; therefore, ABC’ implies A*B*C’. Technically, we have two AND operations here. Since AND is a commutative operation it doesn’t matter which order these are multiplied in, so we combine these with a single AND gate with 3 inputs, called “fan-ins”. This also applies to OR operations.

Solution 1:

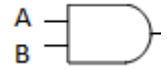


Example: Logic Gates

Given: Create a gate diagram for the following equation: F = AB + C

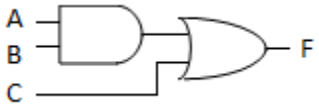
Partial Credit 1: AB happens first in the order of operations and the convention is to place that gate distinctly before any future operations.

Solution 1:



Partial Credit 2: Then, C is added to the result of AB. So the output of the AND gate is used as the input for the following OR gate along with the literal C. The output of the OR gate is the result of the function F.

Solution 2:

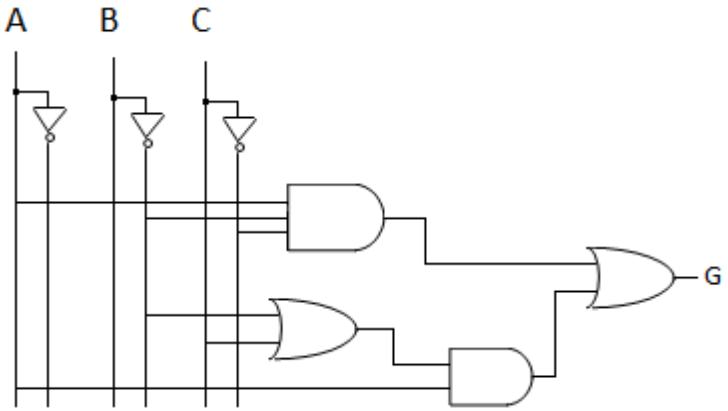


Example: Logic Gates

Given: Create a gate diagram for the following equation: G = AB'C' + A(B'+C)

Partial Credit 1: Gate diagrams should only show each input once. If we need to use an input multiple times, we start with a single input line and add offshoots to each location the input is needed. Similar to before, gates that need to complete their action to provide an intermediate result are placed neared the input lines. Gates that can proceed in parallel are shown on the same level. In this case, AB'C' and B'+C can be calculated at the same time, but A(B'+C) must be calculated before the final OR operation.

Solution 1:



Review: Designing Hardware

The design process is both creative and iterative; you may need to revisit any given step several times before arriving at a workable solution and many more times to arrive at an efficient solution.

1. Describe the system you are creating. What should it do? What are the inputs to the system? What are the intended outputs? How many of each will be necessary? Are any inputs or outputs more than one bit wide? Split these into individual bits. Draw a block diagram to represent the system with each bit of input and output.
2. Create a truth table with one column for each bit of input. This will determine the number of rows ("entries") that the truth table will need. For n inputs, you will need to specify 2^n entries. Fill in the input entries with every possible combination of input values. An easy way to do this is with binary values from 0 to 2^n -1.

Review: Designing Hardware

3. Add output columns for each bit of output.
   a) Determine if your system has any unreachable states that might result in an incompletely specified function. Based on how the system should work, are any of the input combinations impossible? Place X's in the output section of these entries.
   b) For all other entries, add the corresponding output based on what the system should do. Common examples are arithmetic operations, logical operations, selections, and incrementing/decrementing.
4. Construct a Karnaugh Map for each output column. The size of the map depends on the number of entries in the truth table (4, 8, or 16). Place the output values from the truth table into the map based on the numbered squares.
5. Use the Karnaugh Map to reduce the output equations. Each 1 in the map is consider an implicant. Group adjacent implicants into groups such that each group's size is a power of 2 and each group is as large as possible. Every 1 should belong to at least one group. Choose the smallest number of these groups that covers every 1 to create your minimum cover.
6. Identify your output equations based on the minimum cover. These should be minimized sum-of-product form equations.
7. Construct the gate diagram(s) to match your minimized equations.

Example: Design a Counter (Version 1)

Given:

Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. The system cannot be given odd numbers; assume this is verified before the inputs are given to the system.

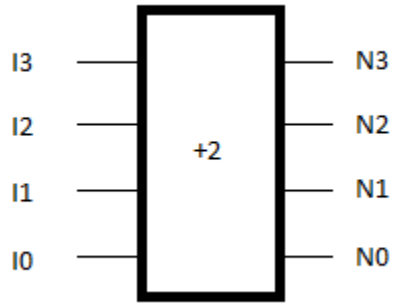
Partial Credit 1:

We are given a description of the system and how it should behave. Based on this description we can determine the number of inputs and outputs for the system. We are told that the largest number the system will accept is 14. 14 in binary is 1110, so this value requires 4 bits to represent. This tells us we will need 4 inputs: one for each bit of input we accept. The system then produces the next even number. Essentially, the system takes the input value and adds two to it. We know that the largest value accepted is 14 and that the system only takes non-negative integers. We can infer then that it takes even values 0-14. If the system is given 12, it should produce 14. As previously mentioned, 14 require 4 bits which tells us we also need 4 bits of output.

Example: Design a Counter (Version 1)

Given: Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. The system cannot be given odd numbers; assume this is verified before the inputs are given to the system.

Solution 1: This is the block diagram. The input bits are labeled I0-I3 and the output bits are labeled N0-N3 (I is common for input and O is common for output, but O0 – the least significant output bit – can be difficult to interpret).



Example: Design a Counter (Version 1)

Given:

Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. The system cannot be given odd numbers; assume this is verified before the inputs are given to the system.

Partial Credit 2:

Now we can build the truth table. The number of necessary columns is shown in the block diagram. There are four inputs, so we need four input columns. There are four outputs, so we also need four output columns.

Solution 2:

Truth Table							
I0	I1	I2	I3	N0	N1	N2	N3
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

Example: Design a Counter (Version 1)

Given:

Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. The system cannot be given odd numbers; assume this is verified before the inputs are given to the system.

Partial Credit 2:

Now we can build the truth table. In this version, we are told that odd inputs are invalid and these are checked somewhere before the inputs arrive at the system. Since we will not receive any odd inputs we do not need to specify any output for those situations, resulting in don't-cares for those entries.

Solution 2:

Truth Table							
I0	I1	I2	I3	N0	N1	N2	N3
0	0	0	0				
0	0	0	1	X	X	X	X
0	0	1	0				
0	0	1	1	X	X	X	X
0	1	0	0				
0	1	0	1	X	X	X	X
0	1	1	0				
0	1	1	1	X	X	X	X
1	0	0	0				
1	0	0	1	X	X	X	X
1	0	1	0				
1	0	1	1	X	X	X	X
1	1	0	0				
1	1	0	1	X	X	X	X
1	1	1	0				
1	1	1	1	X	X	X	X

Example: Design a Counter (Version 1)

Given:

Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. The system cannot be given odd numbers; assume this is verified before the inputs are given to the system.

Partial Credit 2:

Now we can build the truth table. The remaining entries have as output the next even number. For example, when given 0, the system should produce 2. When given 8, the system should produce 10. Remember from the description: when the system is given 14, it produces 0.

Solution 2:

Truth Table							
I0	I1	I2	I3	N0	N1	N2	N3
0	0	0	0	0	0	1	0
0	0	0	1	X	X	X	X
0	0	1	0	0	1	0	0
0	0	1	1	X	X	X	X
0	1	0	0	0	1	1	0
0	1	0	1	X	X	X	X
0	1	1	0	1	0	0	0
0	1	1	1	X	X	X	X
1	0	0	0	1	0	1	0
1	0	0	1	X	X	X	X
1	0	1	0	1	1	0	0
1	0	1	1	X	X	X	X
1	1	0	0	1	1	1	0
1	1	0	1	X	X	X	X
1	1	1	0	0	0	0	0
1	1	1	1	X	X	X	X

Example: Design a Counter (Version 1)

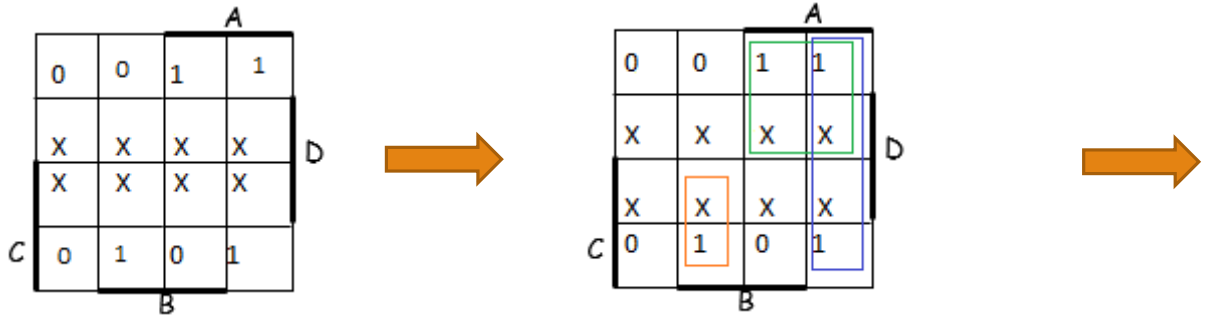
Given:

Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. The system cannot be given odd numbers; assume this is verified before the inputs are given to the system.

Partial Credit 3:

Now, create a Karnaugh map for each bit of output. This will require 4, 4-input Karnaugh maps. First, add the values for N0 into a map. Then, create the groupings. Remember, we get to choose how to treat don't-cares, so use them to make bigger groups where effective!

Solution 3:



Create the minimized sum-of-products equation:
N0 = AC' + AB' + A'BC

In our truth table, A corresponds to I0, B to I1, C to I2, and D to I3:

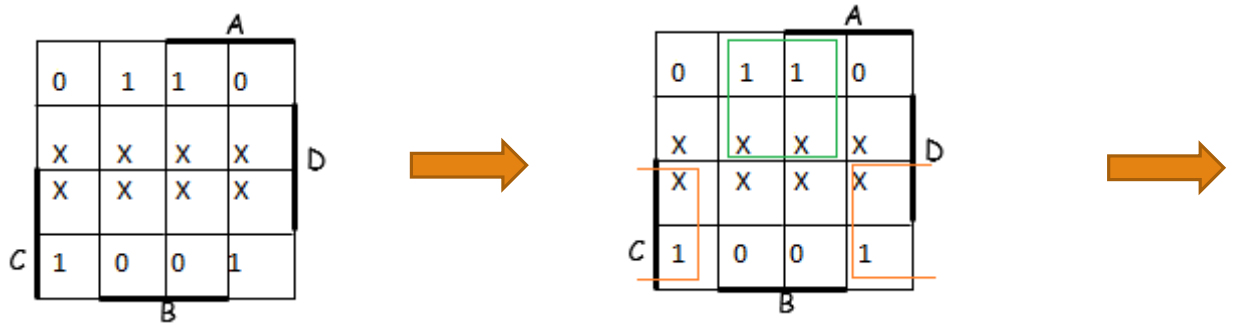
N0 = I0I2' + I0I1' + I0'I1I2

Example: Design a Counter (Version 1)

Given: Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. The system cannot be given odd numbers; assume this is verified before the inputs are given to the system.

Partial Credit 4: Repeat for N1. Remember, adjacencies can wrap around sides, top/bottom, and even corners.

Solution 4:



Create the minimized sum-of-products equation:
N1 = BC' + CB'

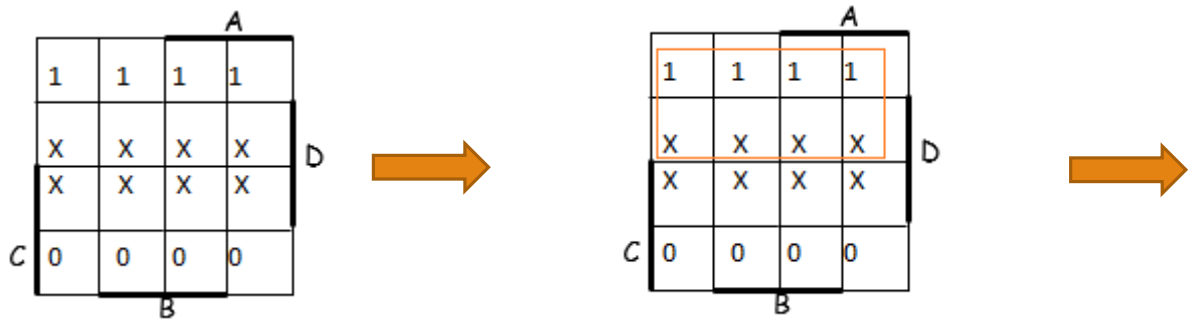
In our truth table, A corresponds to I0, B to I1, C to I2, and D to I3.
N1 = I1I2' + I2I1'

Example: Design a Counter (Version 1)

Given: Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. The system cannot be given odd numbers; assume this is verified before the inputs are given to the system.

Partial Credit 5: Repeat for N2. Use the don't-care set to maximize the size of our group.

Solution 5:



Create the minimized sum-of-products equation:
N2 = C'

In our truth table, A corresponds to I0, B to I1, C to I2, and D to I3.
N2 = I2'

Example: Design a Counter (Version 1)

Given:

Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. The system cannot be given odd numbers; assume this is verified before the inputs are given to the system.

Partial Credit 6:

N3 is unique in that there are no elements in the on-set. Regardless of the input value, N3 is either 0 or don't care. Making a group of don't cares would add to the hardware we would need to specify N3.

Solution 6:

Example: Design a Counter (Version 1)

Given:

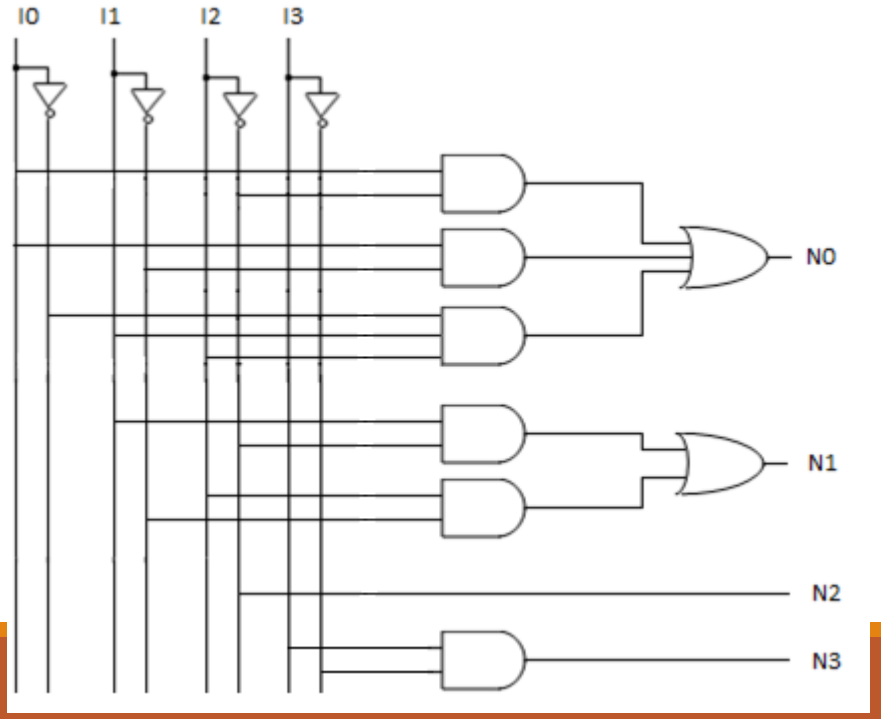
Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. The system cannot be given odd numbers; assume this is verified before the inputs are given to the system.

Partial Credit 7:

Now that we have all of our equations, we can create the final gate diagram for our hardware.

Solution 7:

Note: Since 0 is not an input to the system, we set N3 to 0 with I3 AND I3'. Any input and it's complement could be used for this output.

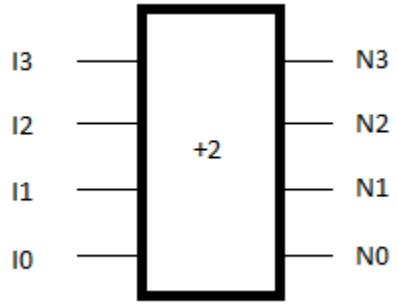


Example: Design a Counter (Version 2)

Given: Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. When the system is given an odd number, it will also return 0.

Partial Credit 1: The specification for this version is very similar to the first one. The only difference is in the way odd values are handled.

Solution 1: The block diagram is the same for both systems.



Example: Design a Counter (Version 2)

Given:

Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. When the system is given an odd number, it will also return 0.

Partial Credit 2:

We will start to see differences starting with the truth table. In this version, we are told that odd inputs should reset the system to zero.

Solution 2:

Truth Table							
I0	I1	I2	I3	N0	N1	N2	N3
0	0	0	0				
0	0	0	1	0	0	0	0
0	0	1	0				
0	0	1	1	0	0	0	0
0	1	0	0				
0	1	0	1	0	0	0	0
0	1	1	0				
0	1	1	1	0	0	0	0
1	0	0	0				
1	0	0	1	0	0	0	0
1	0	1	0				
1	0	1	1	0	0	0	0
1	1	0	0				
1	1	0	1	0	0	0	0
1	1	1	0				
1	1	1	1	0	0	0	0

Example: Design a Counter (Version 2)

Given:

Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. When the system is given an odd number, it will also return 0.

Partial Credit 2:

We will start to see differences starting with the truth table. The remaining entries have as output the next even number. For example, when given 0, the system should produce 2. When given 8, the system should produce 10. Remember from the description: when the system is given 14, it produces 0.

Solution 2:

Truth Table							
I0	I1	I2	I3	N0	N1	N2	N3
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0
0	0	1	0	0	1	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	1	1	0
0	1	0	1	0	0	0	0
0	1	1	0	1	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	1	0	1	0
1	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0
1	0	1	1	0	0	0	0
1	1	0	0	1	1	1	0
1	1	0	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0

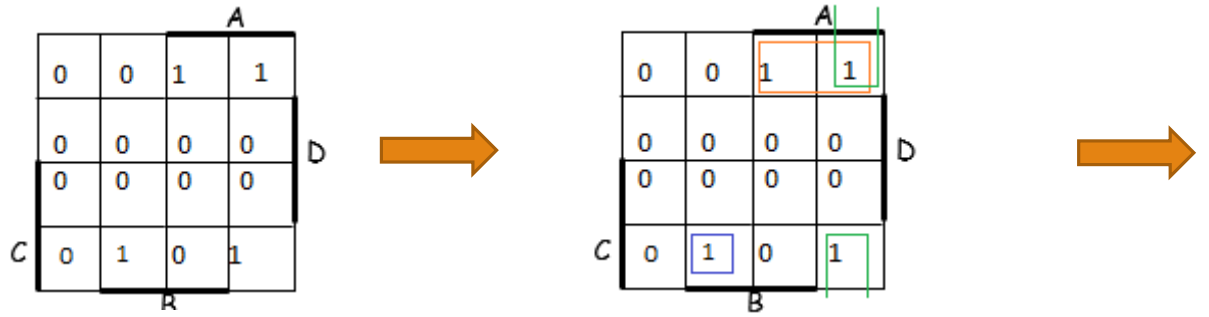
Example: Design a Counter (Version 2)

Given: Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. When the system is given an odd number, it will also return 0.

Partial Credit 3: Now, create a Karnaugh map for each bit of output. Since there are no don't cares we won't be able to use those in our groupings. What effect do you think that will have on the system?

First, add the values for N0 into a map. Then, create the groupings.

Solution 3:



Create the minimized sum-of-products equation:
N0 = AC'D' + AB'D' + A'BCD'

In our truth table, A corresponds to I0, B to I1, C to I2, and D to I3.
N0 = I0I2'I3' + I0I1'I3' + I0'I1I2I3'

Example: Design a Counter (Version 2)

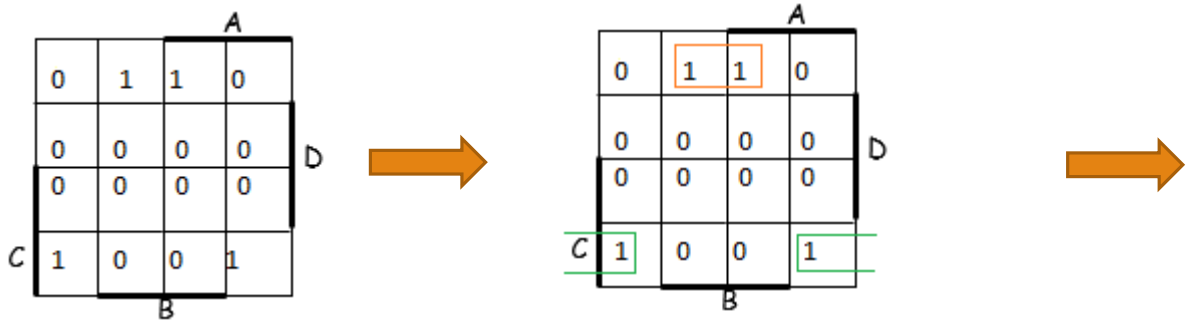
Given:

Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. When the system is given an odd number, it will also return 0.

Partial Credit 4:

Now, create a Karnaugh map for each bit of output. Remember, we form the "minimal cover" by identifying the smallest number of the largest groups to cover the on-set.

Solution 4:



Create the minimized sum-of-products equation:
N1 = BC'D' + B'CD'

In our truth table, A corresponds to I0, B to I1, C to I2, and D to I3.
N1 = I1I2'I3' + I1'I2I3'

Example: Design a Counter (Version 2)

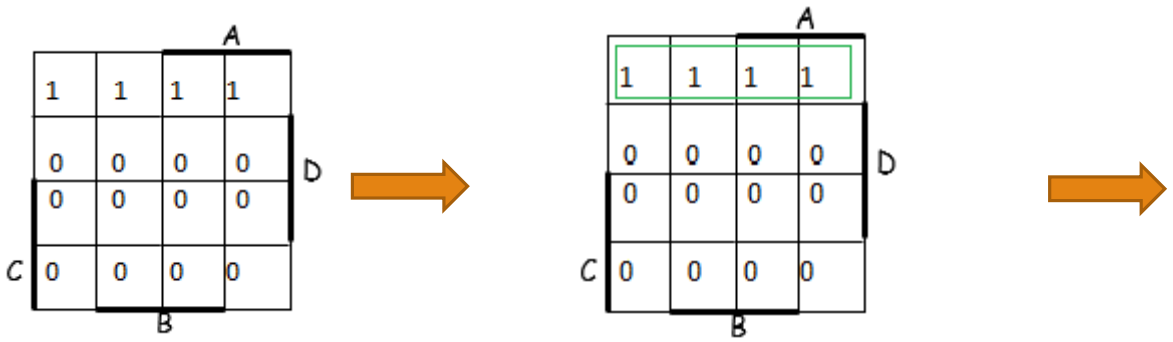
Given:

Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. When the system is given an odd number, it will also return 0.

Partial Credit 5:

Now, create a Karnaugh map for each bit of output. Form the minimal cover for N2:

Solution 5:



Create the minimized sum-of-products equation:
N2 = C'D'

In our truth table, A corresponds to I0, B to I1, C to I2, and D to I3.
N2 = I2'I3'

Example: Design a Counter (Version 2)

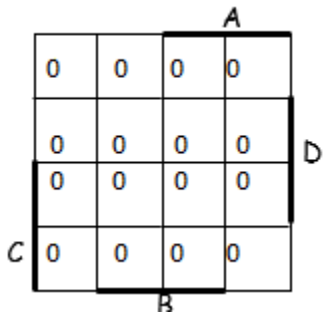
Given:

Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. When the system is given an odd number, it will also return 0.

Partial Credit 6:

Now, create a Karnaugh map for each bit of output. N3 is unique in that there are no elements in the on-set. Regardless of the input value, N3 is 0.

Solution 6:

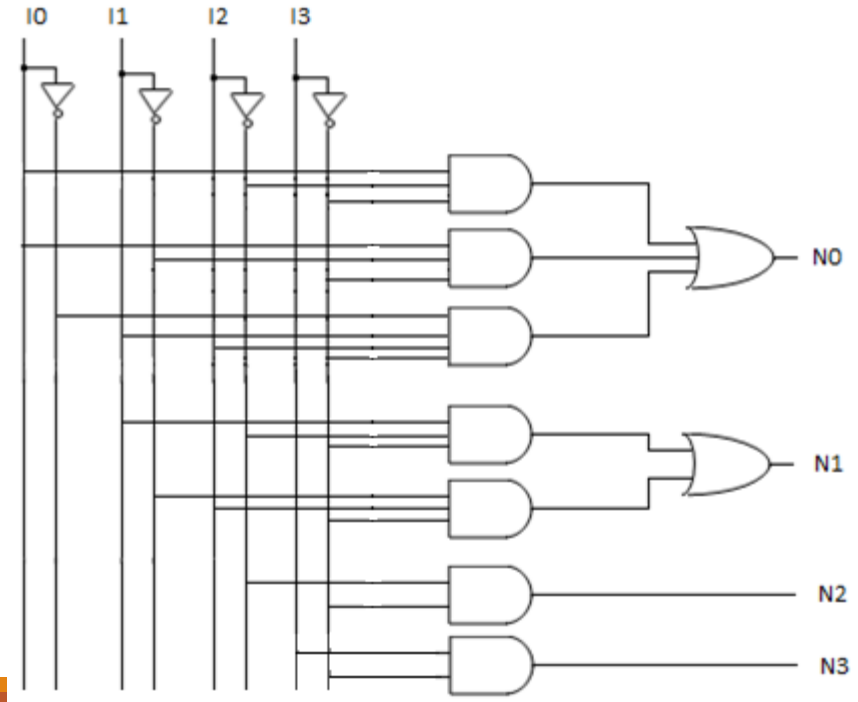


N3 = 0.

Example: Design a Counter (Version 2)

Given: Design a combinational hardware element that counts even numbers. The system must be given a non-negative even number as input and will return as output the next even number. For example, when given the number 2 the system will output the number 4. All numbers will be in binary. The largest number the system can accept is 14, but it will return 0 in this case. When the system is given an odd number, it will also return 0.

Partial Credit 7: Now that we have all of our equations, we can create the final gate diagram for our hardware.



Solution 7: Note: Since 0 is not an input to the system, we set N3 to 0 with I3 AND I3'. Any input and it's complement could be used for this output.

Review: Design Analysis

How is the gate diagram for Counter (Version 1) different from Counter (Version 2)? How do the don't cares affect the development of the system? Which version requires more hardware? Is one better than the other? Describe a situation where the additional hardware might be more beneficial.

The Version 2 diagram requires one additional gate and several gates with additional fan-ins. Adding fan-ins will increase the hardware cost as well as adding another gate, so the second version of the system requires more hardware than the first. In general, the one with less hardware will be preferred. However, we are only told that the odd values are checked before they arrive at the system we built. This check will also require some logic and hardware to implement. The second system is more beneficial if the checking hardware is more complex than the additions made for version 2.