COMP 122/L Lecture 19

Mahdi Ebrahimi

Slides adapted from Dr. Kyle Dewey

Overview

- Circuit minimization
  - Boolean algebra
  - Karnaugh maps

Circuit Minimization

Motivation

- Unnecessarily large programs: bad
- Unnecessarily large circuits: Very Bad™
    - Why?

Motivation

- Unnecessarily large programs: bad
- Unnecessarily large circuits: Very Bad™
  - Why?
    - Bigger circuits = bigger chips = higher cost (non-linear too!)
    - Longer circuits = more time needed to move electrons through = slower

Simplification

- Real-world formulas can often be simplified, according to algebraic rules
  - How might we simplify the following?

```
R = AB + !AB
```

-How might we simplify this?

Simplification

- Real-world formulas can often be simplified, according to algebraic rules
  - How might we simplify the following?

```
R = AB + !AB
R = B(A + !A)
R = B(true)
R = B
```

Simplification Trick

- Look for products that differ only in one variable
  - One product has the original variable (`A`)
  - The other product has the other variable (`!A`)

```
R = AB + !AB
```

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD
```

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD

BCD(A + !A) + !AB!CD + AB!CD
```

Additional Example 1

!ABCD + ABCD + !AB!CD + AB!CD

BCD(A + !A) + !AB!CD + AB!CD

BCD + !AB!CD + AB!CD

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD

BCD(A + !A) + !AB!CD + AB!CD

BCD + !AB!CD + AB!CD

BCD + B!CD(!A + A)
```

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD

BCD(A + !A) + !AB!CD + AB!CD

BCD + !AB!CD + AB!CD

BCD + B!CD(!A + A)

BCD + B!CD
```

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD

BCD(A + !A) + !AB!CD + AB!CD

BCD + !AB!CD + AB!CD

BCD + B!CD(!A + A)

BCD + B!CD

BD(C + !C)
```

Additional Example 1

```
!ABCD + ABCD + !AB!CD + AB!CD

BCD(A + !A) + !AB!CD + AB!CD

BCD + !AB!CD + AB!CD

BCD + B!CD(!A + A)

BCD + B!CD

BD(C + !C)
BD
```

Additional Example 2

```
!A!BC + A!B!C + !ABC + !AB!C + A!BC
```

Additional Example 2

`!A!BC + A!B!C + !ABC + !AB!C + A!BC`

`!A!BC + A!BC + A!B!C + !ABC + !AB!C`

Additional Example 2

```
!A!BC + A!B!C + !ABC + !AB!C + A!BC

!A!BC + A!BC + A!B!C + !ABC + !AB!C

!BC(A + !A) + A!B!C + !ABC + !AB!C
```

Additional Example 2

```
!A!BC + A!B!C + !ABC + !AB!C + A!BC

!A!BC + A!BC + A!B!C + !ABC + !AB!C

!BC(A + !A) + A!B!C + !ABC + !AB!C

!BC + A!B!C + !ABC + !AB!C
```

Additional Example 2

```
!A!BC + A!B!C + !ABC + !AB!C + A!BC

!A!BC + A!BC + A!B!C + !ABC + !AB!C

!BC(A + !A) + A!B!C + !ABC + !AB!C

!BC + A!B!C + !ABC + !AB!C

!BC + A!B!C + !AB(C + !C)
```

Additional Example 2

```
!A!BC + A!B!C + !ABC + !AB!C + A!BC

!A!BC + A!BC + A!B!C + !ABC + !AB!C

!BC(A + !A) + A!B!C + !ABC + !AB!C

!BC + A!B!C + !ABC + !AB!C

!BC + A!B!C + !AB(C + !C)

!BC + A!B!C + !AB
```

De Morgan's Laws

Also potentially useful for simplification

De Morgan's Laws

Also potentially useful for simplification

```
!(A + B) = !A!B
```

De Morgan's Laws

Also potentially useful for simplification

```
!(A + B) = !A!B

!(AB) = !A + !B
```

De Morgan Example

`!(X + Y)!(!X + Z)`

De Morgan Example

```
!(X + Y)!(!X + Z)

   !A        !B
```

Has the overall form of !A!B

De Morgan Example

`!(X + Y)!(!X + Z)`

`!A` `!B`

Has the overall form of !A!B

De Morgan Example

`!(X + Y)!(!X + Z)`

`!A` `!B`

From De Morgan's Law:

`!(A + B) = !A!B`

Has the overall form of !A!B

De Morgan Example

`!(X + Y)!(!X + Z)`

`!A` `!B`

From De Morgan's Law:

`!(A + B) = !A!B`

`!(X + Y + !X + Z)`

Has the overall form of !A!B

De Morgan Example

`!(X + Y)!(!X + Z)`

`!A` `!B`

From De Morgan's Law:

`!(A + B) = !A!B`

`!(X + Y + !X + Z)`

`!(X + !X + Y + Z)`

Has the overall form of !A!B

De Morgan Example

```
!(X + Y)!(!X + Z)

    !A          !B
```

From De Morgan's Law:

```
!(A + B) = !A!B
!(X + Y + !X + Z)
!(X + !X + Y + Z)
!(true + Y + Z)
```

Has the overall form of !A!B

De Morgan Example

`!(X + Y)!(!X + Z)`

`!A` `!B`

From De Morgan's Law:

`!(A + B) = !A!B`

`!(X + Y + !X + Z)`

`!(X + !X + Y + Z)`

`!(true + Y + Z)`

`!(true)`

Has the overall form of !A!B

De Morgan Example

`!(X + Y)!(!X + Z)`

`!A                 !B`

From De Morgan's Law:

`!(A + B) = !A!B`

`!(X + Y + !X + Z)`

`!(X + !X + Y + Z)`

`!(true + Y + Z)`

`!(true)`

`false`

Has the overall form of !A!B

Boolean Operators (Truth Table)

AND   X Y

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

OR   X + Y

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Complement (opposite) $\overline{X}$

X	NOT X
0	1
1	0

Boolean Algebra

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

Scaling Up

- Performing this sort of algebraic manipulation by hand can be tricky
- We can use *Karnaugh maps* to make it immediately apparent as to what can be simplified

Karnaugh Maps - Rules of Simplification:
http://www.ee.surrey.ac.uk/Projects/Labview/minimisation/karrules.html

Example

```
R = AB + !AB
```

-Start with the sum of products

Example

R = AB + !AB

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

–Build the truth table

Example

R = AB + !AB

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

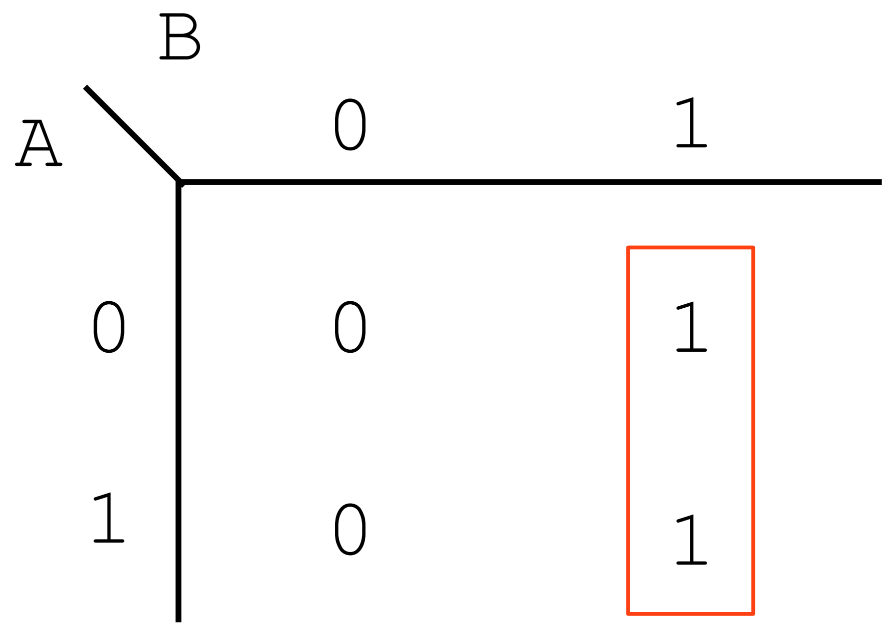
A \ B	0	1
0	0	1
1	0	1

-Build the K-map

Example

R = AB + !AB

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

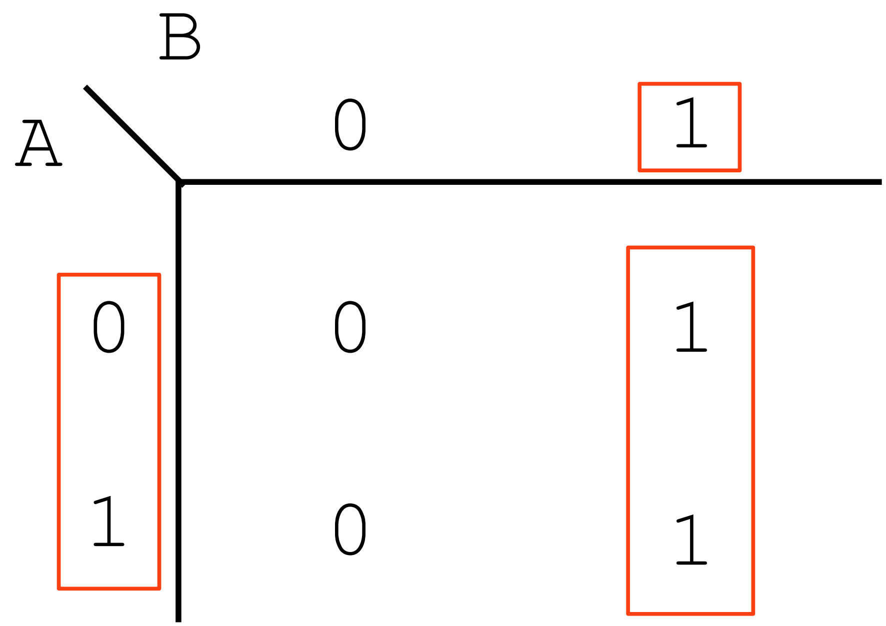


-Group adjacent (row or column-wise, NOT diagonal) 1's in powers of two (groups of 2, 4, 8...)

Example

R = AB + !AB

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1



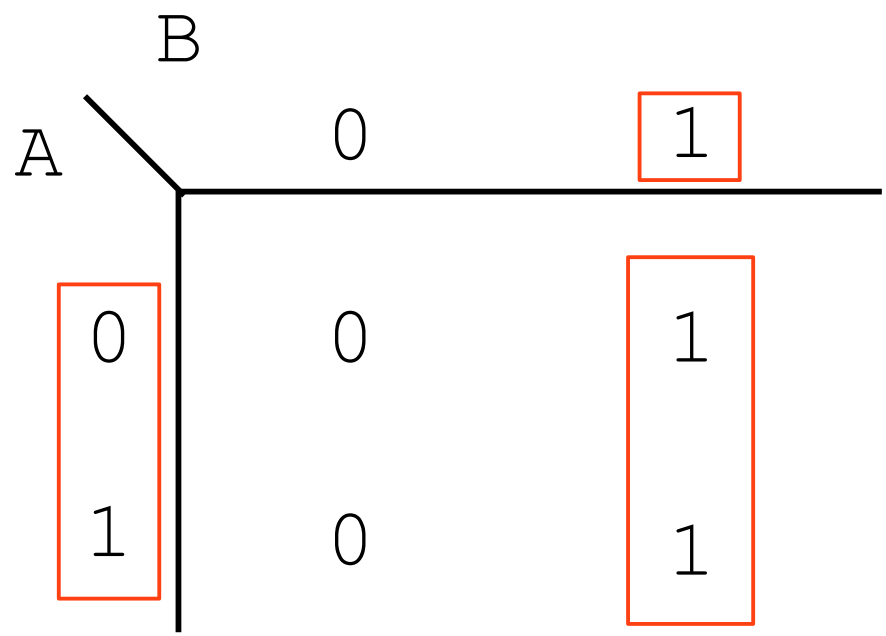
-The values that stay the same are saved, the rest are discarded
-This works because this means that the inputs that differ are irrelevant to the final value, and so they can be removed

Example

R = AB + !AB

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

R = B



-The values that stay the same are saved, the rest are discarded
-This works because this means that the inputs that differ are irrelevant to the final value, and so they can be removed

Three Variables

- We can scale this up to three variables, by combining two variables on one axis
- The combined axis must be arranged such that only one bit changes per position

A \ BC	00	01	11	10
0	?	?	?	?
1	?	?	?	?

Three Variable Example

R = !A!BC + !ABC + A!BC + ABC

-Start with this formula

R = !A!BC + !ABC + A!BC + ABC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

-Build the truth table

R = !A!BC + !ABC + A!BC + ABC

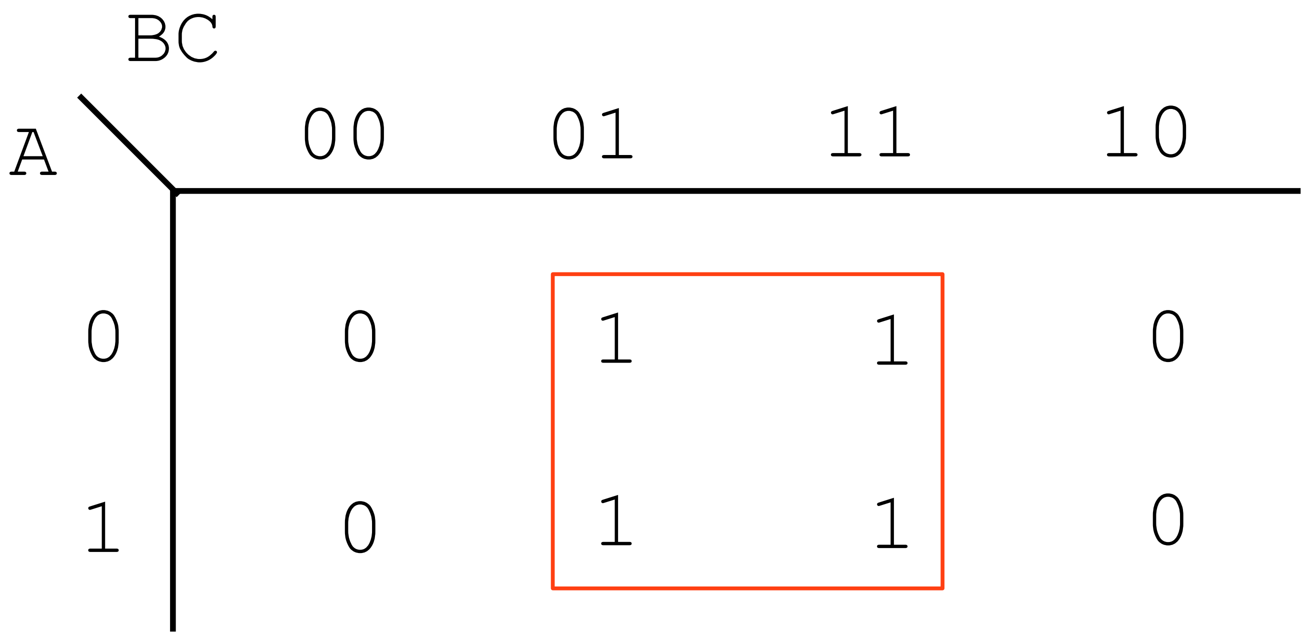
A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

A \ BC	00	01	11	10
0	0	1	1	0
1	0	1	1	0

-Build the K-map

R = !A!BC + !ABC + A!BC + ABC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



-Select the biggest group possible, in this case a square
-In order to get the most minimal circuit, we must always select the biggest groups possible

R = !A!BC + !ABC + A!BC + ABC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

R = C

A \ BC	00	01	11	10
0	0	1	1	0
1	0	1	1	0

–Save the ones that stay the same in a group, discarding the rest

Another Three Variable Example

R = !A!B!C + !A!BC + !ABC +
!AB!C + A!B!C + AB!C

-Start with this formula

R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

–Build the truth table

R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

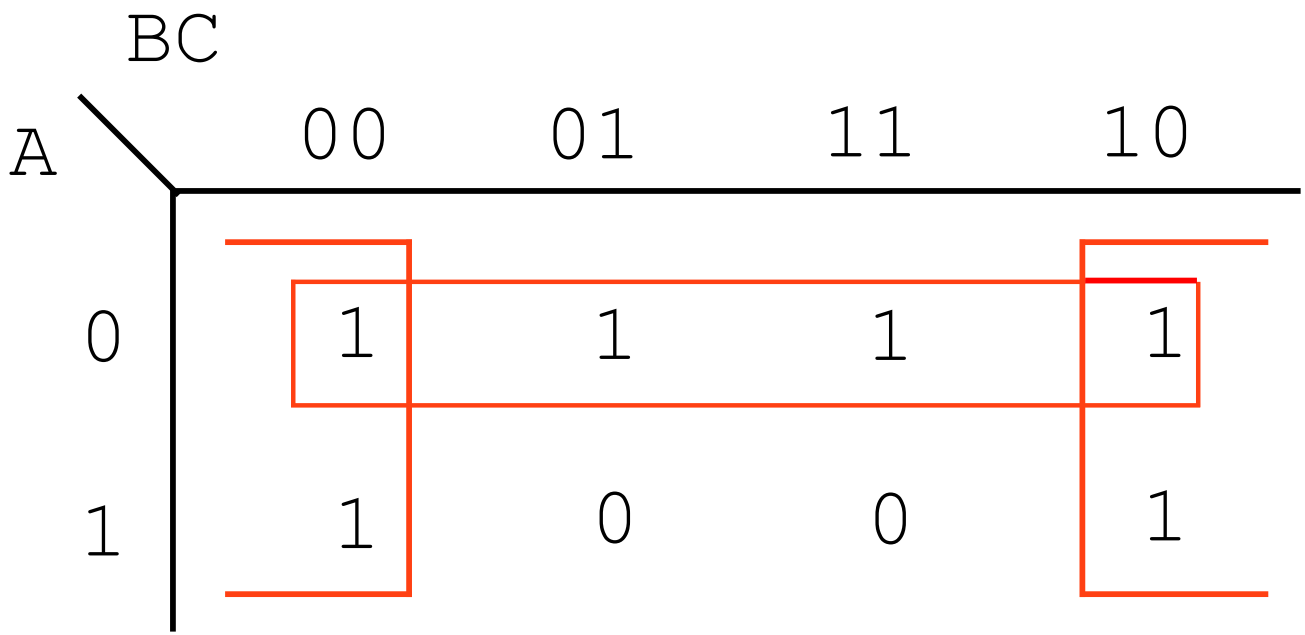
A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

A \ BC	00	01	11	10
0	1	1	1	1
1	1	0	0	1

-Build the K-map

R = !A!B!C + !A!BC + !ABC +
!AB!C + A!B!C + AB!C

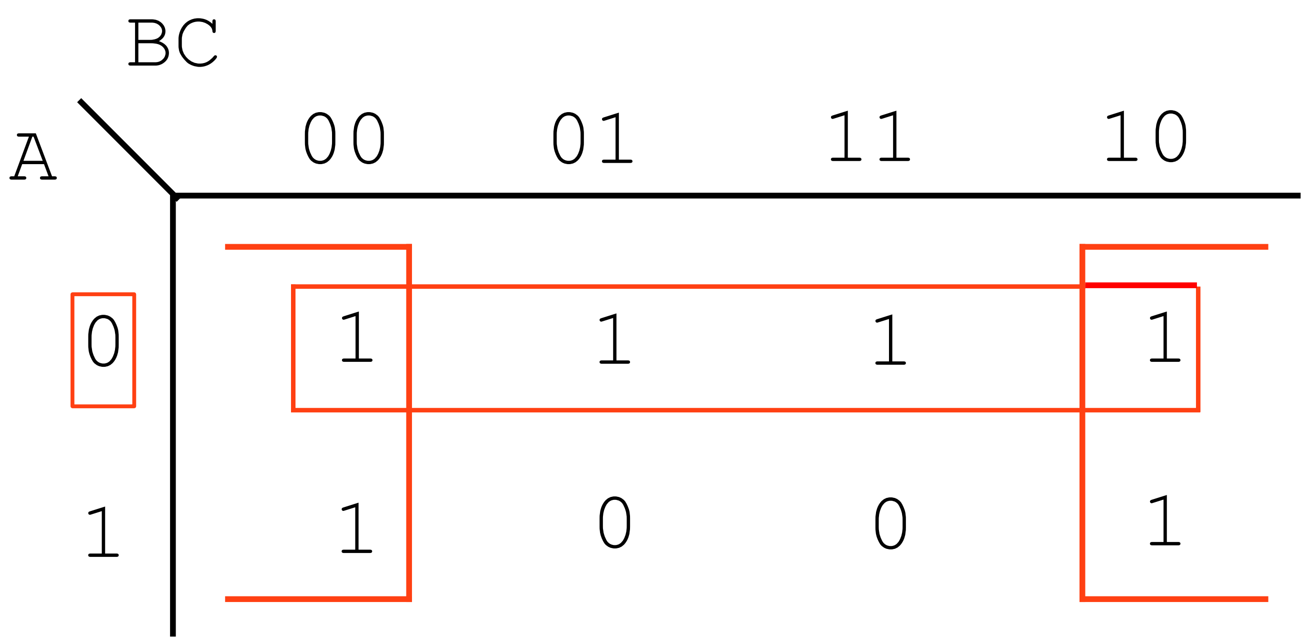
A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



-Select the biggest groups possible
-Note that the values “wrap around” the table

R = !A!B!C + !A!BC + !ABC +
!AB!C + A!B!C + AB!C

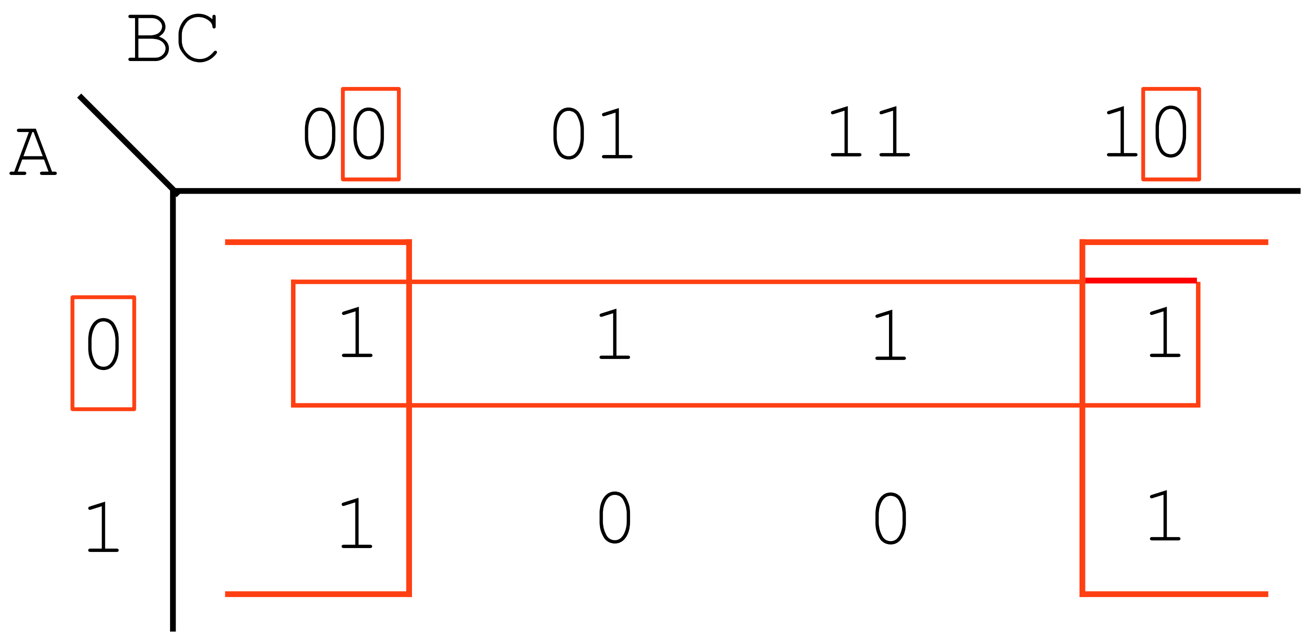
A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



-Save the ones that stay the same in a group, discarding the rest
-This must be done for each group

R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

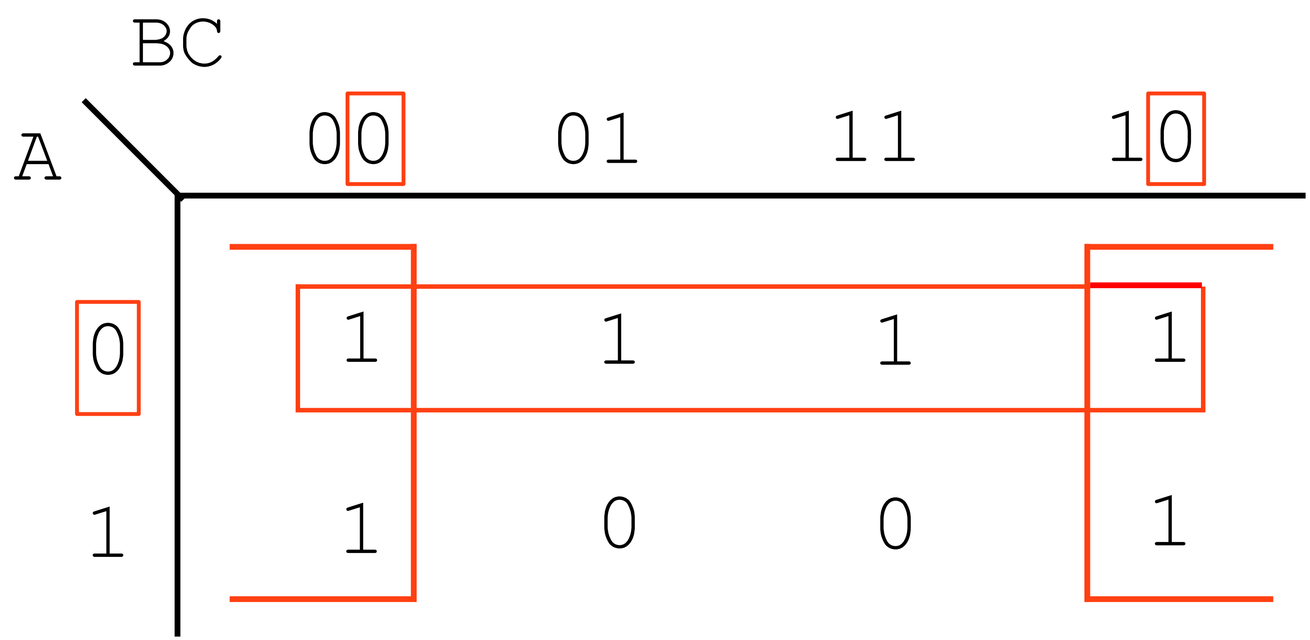


-Save the ones that stay the same in a group, discarding the rest
-This must be done for each group

R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

R = !A + !C



-Save the ones that stay the same in a group, discarding the rest
-This must be done for each group

Four Variable Example

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

–Take this formula

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

-For space reasons, we go directly to the K-map

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

- Group things up
- The edges logically wrap around!
- Groups may overlap each other

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

R =!B!C

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

-Look at the bits that don’t change
-First for the cube

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

R =!B!C + !B!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

- Look at the bits that don't change
- Second for the cube on the edges

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

R =!B!C + !B!D + !AC!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

- Look at the bits that don't change
- Third for the line

K-Map Rules in Summary (1)

- Groups can contain only `1`s
- Only `1`s in adjacent groups are allowed (no diagonals)
- The number of `1`s in a group must be a power of two (1,2,4,8...)
- The groups must be as large as legally possible

K-Map Rules in Summary (2)

- All `1`s must belong to a group, even if it's a group of one element
- Overlapping groups are permitted
- Wrapping around the map is permitted
- Use the fewest number of groups possible

Revisiting Problem

```
!A!BC + A!B!C + !ABC + !AB!C + A!BC
```

Revisiting Problem

R = !A!BC + A!B!C + !ABC + !AB!C + A!BC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Revisiting Problem

R = !A!BC + A!B!C + !ABC + !AB!C + A!BC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

A \ BC	00	01	11	10
0	0	1	1	1
1	1	1	0	0

Revisiting Problem

R = !A!BC + A!B!C + !ABC + !AB!C + A!BC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

R = !AC

A \ BC	00	01	11	10
0	0	1	1	1
1	1	1	0	0

Revisiting Problem

R = !A!BC + A!B!C + !ABC + !AB!C + A!BC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

R = !AC + A!B

A \ BC	00	01	11	10
0	0	1	1	1
1	1	1	0	0

Revisiting Problem

R = !A!BC + A!B!C + !ABC + !AB!C + A!BC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

R = !AC + A!B + !AB!C

A \ BC	00	01	11	10
0	0	1	1	1
1	1	1	0	0

Difference

- Algebraic solution: `!BC + A!B!C + !AB`
- K-map solution: `!AC + A!B + !AB!C`
- Question: why might these differ?

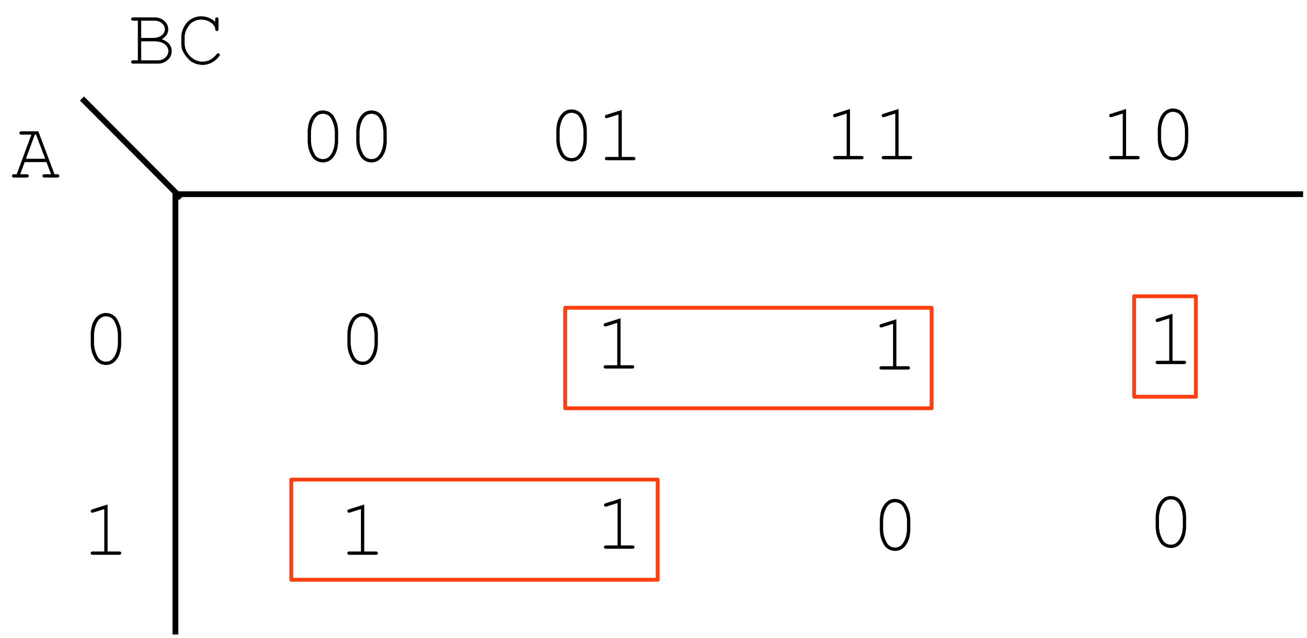
Difference

- Algebraic solution: `!BC + A!B!C + !AB`
- K-map solution: `!AC + A!B + !AB!C`
- Question: why might these differ?
  - Both are *minimal*, in that they have the fewest number of products possible
  - Can be multiple minimal solutions

Difference

Algebraic solution: !BC + A!B!C + !AB

K-map solution: !AC + A!B + !AB!C

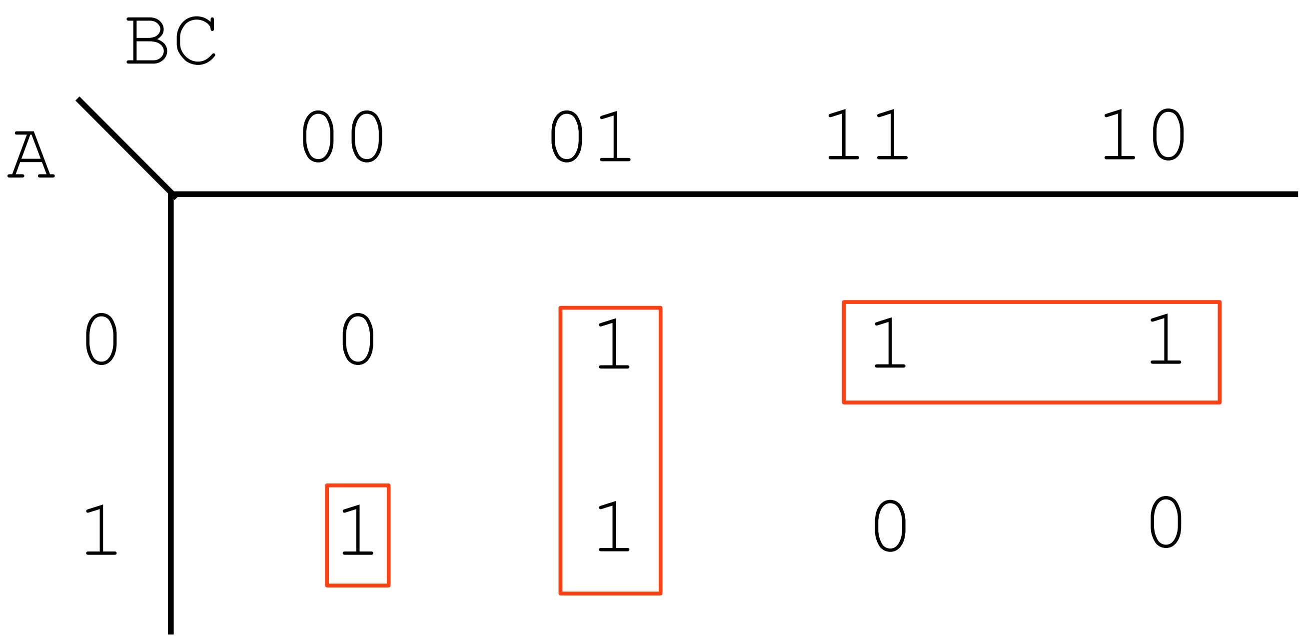


–If we take our k-map from before with the grouping we chose, we get this particular solution

Difference

Algebraic solution: !BC + A!B!C + !AB

K-map solution: !BC + A!B!C + !AB



–If, however, we choose a different (also valid) grouping, we get the same solution as we did algebraically

Don’t Cares

Don’t cares in a Karnaugh map, or truth table, may be either **1**s or **0**s, as long as we don’t care what the output is for an input condition we never expect to see. We plot these cells with an asterisk, *, among the normal **1**s and **0**s.

When forming groups of cells, treat the don’t care cell as either a **1** or a **0**, or ignore the don’t cares. This is helpful if it allows us to form a larger group than would otherwise be possible without the don’t cares. There is no requirement to group all or any of the don’t cares.

Only use them in a group if it simplifies the logic.

A \ BC	00	01	11	10
0	0	0	0	X
1	0	1	X	X