First class

CSEC15 &CSEC25

LOGIC TECHNIQUES

1.FIRST COURSE .

(CSEC15)

COMBINATIONAL SYST

2. **SECOND COURSE .**

(CSEC25)

SEQUENTIAL SYSTEM

FACULTY of Engineering

Dept. of Computer & Software **Engg.**

Ahmed Saleh al-Zuhairi

1.FIRST COURSE .

(CSEC15) COMBINATIONAL SYST

Digital Techniques
Introduction

Outline

- Course, Attendance, Reference Book
- What do we study in this course?
- Why should this be studied?
- How is the course structured?

Logic techniques:
2 Courses Structure

- Logic Gates, Boolean Algebra, Logic Minimization & realization(Karnaugh-Map, Product of sum simplification).
- Combinational Logic Desigin
- Flip-flops, registers, counters, Finite state model
- Synthesis of synchronous sequential circuits
- Number representation: fixed and floating point;

- Addition, subtraction, multiplication & division of numbers.
- Decoders & Encoders
- Multiplexers & Demultiplexers

CS221: Text & References

- Text Book

–Introduction to Logic & computer Design, Alan B. Marcovitz, McGraw-Hill, 2008

–Digital Fundamentals, Thomas L. Floyd, Pearson, 2006.

- *References*

– R. H. Katz & G. Boriello, Contemporary Logic Design, 2/e, Prentice Hall of India, 2009.

– A. P. Malvino, D. P. Leach & G.Saha, Digital Principles & Applications, 7/e, McGraw Hill, 2010.

– S. C. Lee, Digital Circuits & Logic Design, Prentice Hall of India, 2006.

– J. F. Wakerly, Digital Design Principles & Practices, 4/e, Prentice

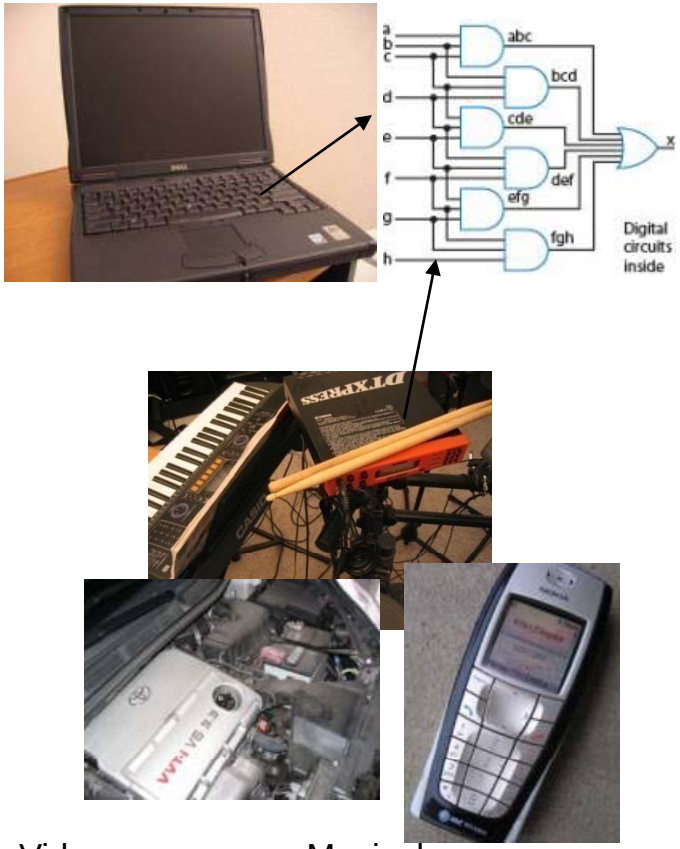
Hall of India, 2008

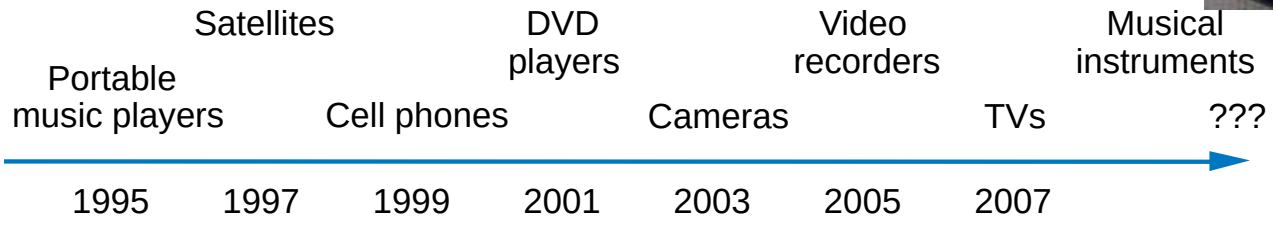
– Vahid Frank, Digital Design, Preview Edition, Wiley India Pvt Ltd, 2010

–M. Morris Mano & M. D. Ciletti, Digital Design, 4/e, Pearson Education, 2007.

Why Study Digital Design?

- **Look "under the hood" of computers**
  - **Solid understanding --> confidence, insight, even better programmer when aware of hardware resource issues**
- **Electronic devices becoming digital**
  - **Enabled by shrinking and more capable chips**
  - **Enables:**
    - **Better devices: Better sound recorders, cameras, cars, cell phones, medical devices,...**
    - **New devices: Video games, PDAs, ...**
  - **Known as "embedded systems"**
    - **Thousands of new devices every year**
    - **Designers needed: Potential career direction**





Digital Design: Motivation

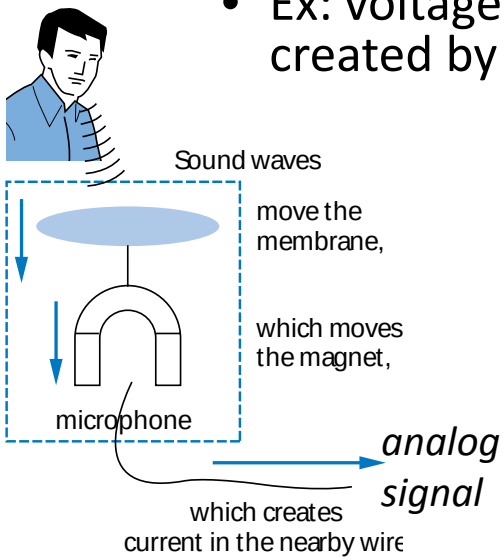
- Implementation basis for modern computing devices
  - Constructing large systems from small components
  - Another view of a computer: **controller + datapath**
- Inherent parallelism in hardware
  - Parallel computation beyond 61C
- Counterpoint to software design
  - Furthering our understanding of computation

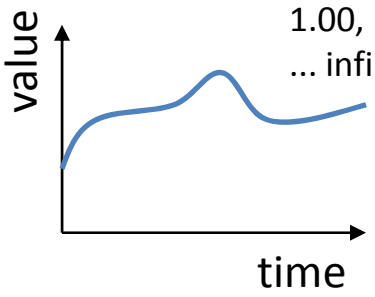
Digital Design

- What is digital ?
  - Digital camera, Digital TV, Digital Watch, Digital Radio, Digital City (e-city), Digital Photo Frame …etc
  - Which gives the things in countable form
  - Scene (analog) to Image (digital)
- Why digital ?
  - Countable form, makes easy to manage
  - Easy management makes more useful and versatile
- What digit ?
  - How to count: Decimal digit: 0 to 9

What Does "Digital" Mean?

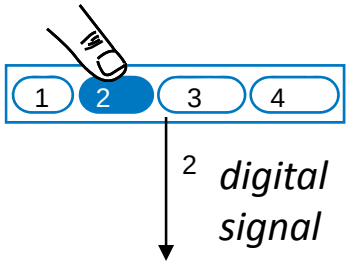
- Analog signal
  - Infinite possible values
    - Ex: voltage on a wire created by microphone

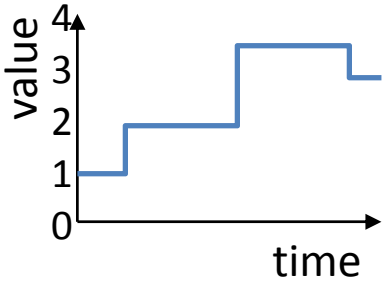




Possible values:
1.00, 1.01, 2.0000009,
... infinite possibilities

- Digital signal
  - Finite possible values
    - Ex: button pressed on a keypad





Possible values:
0, 1, 2, 3, or 4.
That's it.

What Digit? => Number System

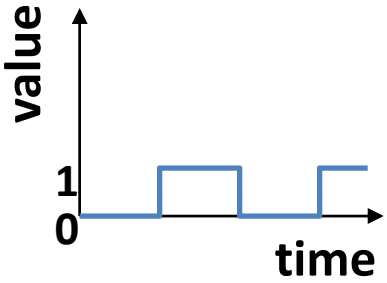
- Famous Number System: Dec, Rom, Bin
- Decimal System: 0 -9
  - May evolves: because human have 10 finger
- Roman System
  - May evolves to make easy to look and feel
  - Pre/Post Concept: (IV, V & VI) is (5-1, 5 & 5+1)
- Binary System, Others (Oct, Hex)
  - One can cut an apple in to two

Design & Logic Design

- What is design?
  - Given problem spec, solve it with available components
  - While meeting quantitative (size, cost, power) and qualitative (beauty, elegance)
- What is logic design?
  - Choose digital logic components to perform specified control, data manipulation, or communication function and their interconnection
  - Which logic components to choose?
    Many implementation technologies (fixed-function components, *programmable devices*, individual transistors on a chip, etc.)
  - Design optimized/transformed to meet design constraints

Digital Signals with Only Two Values: Binary

- ***Binary*** digital signal -- only *two* possible values
  - Typically represented as **0** and **1**
  - One *B*inary dig*it* is a ***bit***
  - We'll only consider *binary* digital signals
  - Binary is popular because
    - Transistors, the basic digital electric component, operate using *two* voltages
    - Storing/transmitting one of *two* values is easier than three or more (e.g., loud beep or quiet beep, reflection or no reflection)

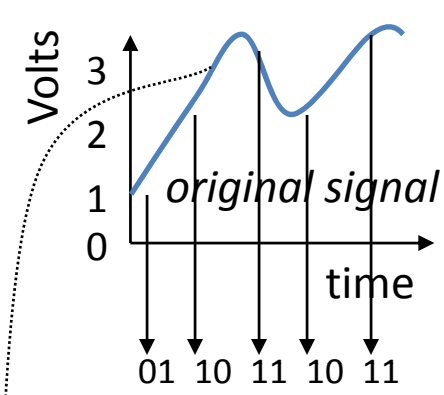


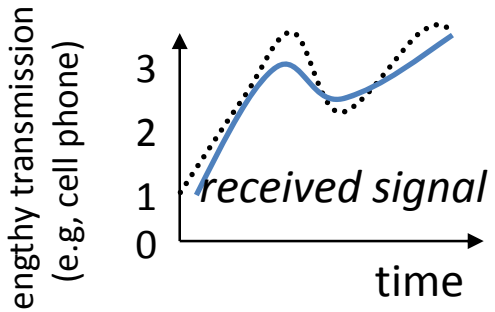
Example of Digitization Benefit

- **Analog signal (e.g., audio) may lose quality**
  - **Voltage levels not saved/copied/transmitted perfectly**
- **Digitized version enables near-perfect save/cpy/trn.**
  - **"Sample" voltage at particular rate, save sample using bit encoding**
  - **Voltage levels still not kept perfectly**
  - **But we can distinguish 0s from 1s**

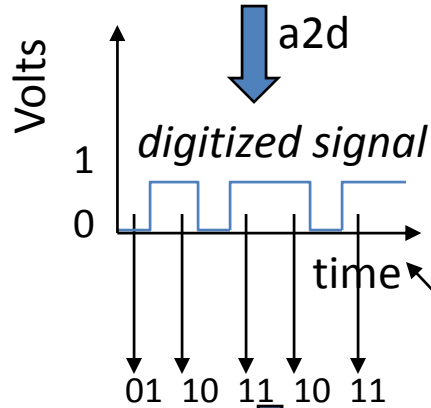
Let bit encoding be:

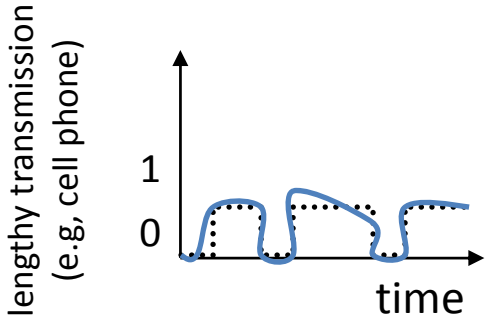
1 V: "01"
2 V: "10"
3 V: "11"



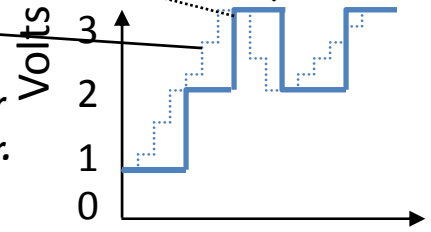


How fix -- higher, lower, ?



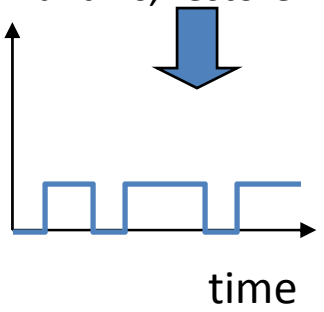


Can fix -- easily distinguish 0s and 1s, restore



Digitized signal not perfect re-creation, but higher sampling rate and more bits per encoding brings closer.

same



Advance Digital Logic Design

- Complicated Circuit (Processor, Memory)
- IC, MSI, LSI, VLSI, ULSI…
- CAD Tool to accelerate
  - Design, Model, Simulate, Validate, Test
  - Placing, Routing, Power distribution
- VHDL/Verilog Simulation
- Schematic Editor
- Embedded System (Area, Power, Cost)

Thanks

Number system

Outline

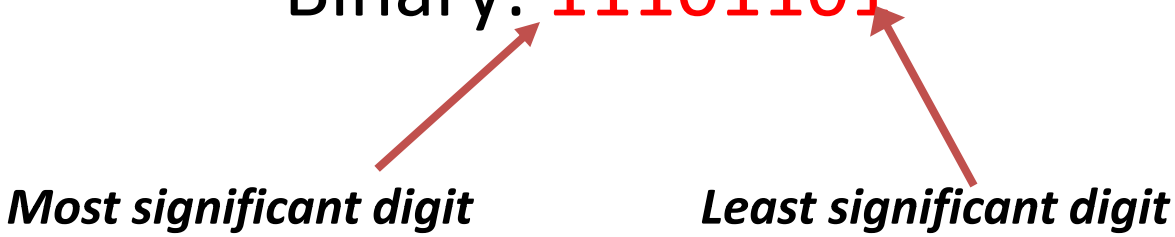
- Number System
  - Decimal, Binary, Octal, Hex
- Conversion (one to another)
  - Decimal to Binary, Octal, Hex & Vice Versa
  - Binary to HEX & vice versa
- Other representation
  - Signed, Unsigned, Complement
- Operation
  - Add, Sub, Mul, Div, Mod
- How to handle real number efficiently ?
  - Float, Double

What Digit? => Number System

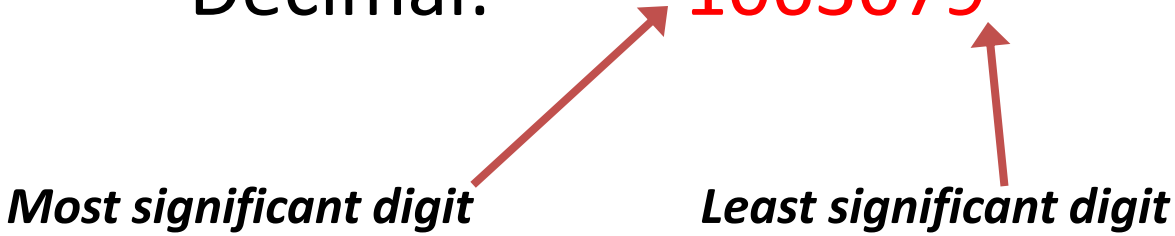
- Famous Number System: Dec, Rom, Bin
- Decimal System: 0 -9
  - May evolves: because human have 10 finger
- Roman System
  - May evolves to make easy to look and feel
  - Pre/Post Concept: (IV, V & VI) is (5-1, 5 & 5+1)
- Binary System, Others (Oct, Hex)
  - One can cut an apple in to two

Significant Digits

Binary: 11101101



Decimal: 1063079



Decimal (base 10)

- Uses positional representation
- Each digit corresponds to a power of 10 based on its position in the number
- The powers of 10 increment from 0, 1, 2, etc. as you move right to left
  - $1{,}479 = 1 * 10^3 + 4 * 10^2 + 7 * 10^1 + 9 * 10^0$

Binary (base 2)

- Two digits: 0, 1
- To make the binary numbers more readable, the digits are often put in groups of 4
  - $1010 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$
    $= 8 + 2$
    $= 10$
  - $1100\ 1001 = 1 * 2^7 + 1 * 2^6 + 1 * 2^3 + 1 * 2^0$
    $= 128 + 64 + 8 + 1$
    $= 201$

How to Encode Numbers: Binary Numbers

- Working with binary numbers
  - In base ten, helps to know powers of 10
    - one, ten, hundred, thousand, ten thousand, ...
  - In base two, helps to know powers of 2
    - one, two, four, eight, sixteen, thirty two, sixty four, one hundred twenty eight
    - Count up by powers of two

2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
512	256	128	64	32	16	8	4	2	1

Octal (base 8)

- Shorter & easier to read than binary
- 8 digits: 0, 1, 2, 3, 4, 5, 6, 7,
- Octal numbers

$$136_8 = 1 * 8^2 + 3 * 8^1 + 6 * 8^0$$
$$= 1 * 64 + 3 * 8 + 6 * 1$$
$$= 94_{10}$$

Hexadecimal (base 16)

- Shorter & easier to read than binary
- 16 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- “0x” often precedes hexadecimal numbers

$$0x123 = 1 * 16^2 + 2 * 16^1 + 3 * 16^0$$
$$= 1 * 256 + 2 * 16 + 3 * 1$$
$$= 256 + 32 + 3$$
$$= 291$$

Counting

Decimal	Binary	Octal	Hexadecimal
0	00000	0	0
1	00001	1	1
2	00010	2	2
3	00011	3	3
4	00100	4	4
5	00101	5	5
6	00110	6	6
7	00111	7	7
8	01000	10	8
9	01001	11	9
10	01010	12	A
11	01011	13	B
12	01100	14	C
13	01101	15	D
14	01110	16	E
15	01111	17	F
16	10000	20	10

Fractional Number

- Point: Decimal Point, Binary Point, Hexadecimal point
- Decimal

  $247.75 = 2x10^{2}+4x10^{1}+7x10^{0}+7x10^{-1}+5x10^{-2}$

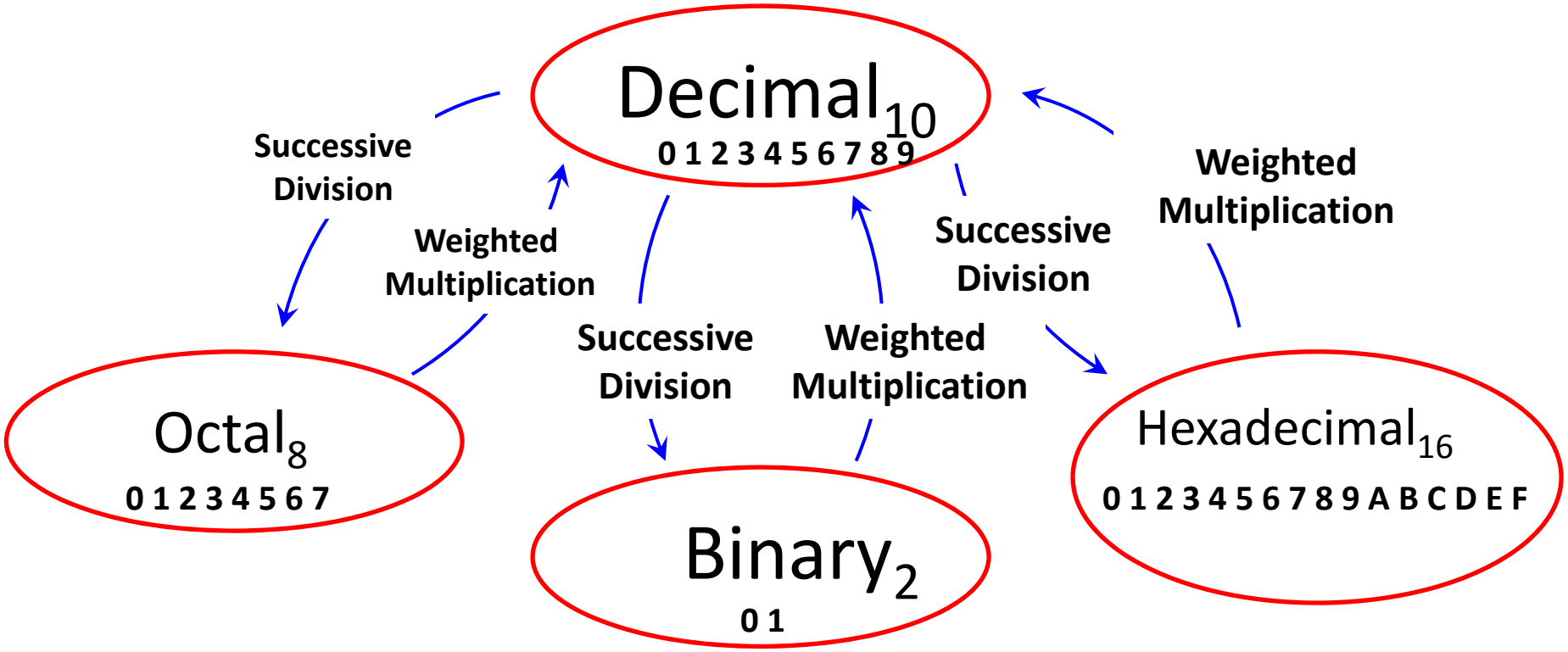
- Binary

  $10.101= 1x2^{1}+0x2^{0}+1x2^{-1}+0x2^{-2}+1x2^{-3}$

- Hexadecimal

  $6A.7D=6x16^{1}+10x16^{0}+7x16^{-1}+Dx16^{-2}$

Converting To and From Decimal



Decimal ↔ Binary

a) Divide the decimal number by **2**; the remainder is the LSB of the **binary** number.

b) If the quotation is zero, the conversion is complete. Otherwise repeat step (a) using the quotation as the decimal number. The new remainder is the next most significant bit of the **binary** number.

a) Multiply each bit of the **binary** number by its corresponding bit-weighting factor (i.e., Bit-0→$\mathbf{2}^0$=1; Bit-1→$\mathbf{2}^1$=2; Bit-2→$\mathbf{2}^2$=4; etc).

b) Sum up all of the products in step (a) to get the decimal number.

Decimal to Binary : Subtraction Method

- Goal
  - Good for human
  - Get the binary weights to add up to the decimal quantity
    - Work from left to right
    - (Right to left – may fill in 1s that shouldn't have been there – try it).

Desired decimal number: **12**

32	16	8	4	2	1	
1						=32 too much
0	1					=16 too much
0	0	1				=8 ok, keep going
0	0	1	1			=8+4=12 DONE
0	**0**	**1**	**1**	**0**	**0**	answer

Decimal to Binary : Division Method

- Good for computer: Divide decimal number by 2 and insert remainder into new binary number.
  - Continue dividing quotient by 2 until the quotient is 0.
- Example: Convert decimal number 12 to binary

12 div 2 = (Quo=6 , Rem=0) LSB
6 div 2 = (Quo=3, Rem=0)
3 div 2 = (Quo=1,Rem=1)
1 div 2 = (Quo=0, Rem=1) MSB

$12_{10} = 1\ 1\ 00\ _2$

Conversion Process Decimal ↔ Base$_N$

a) Divide the decimal number by **N**; the remainder is the LSB of the **ANY BASE** Number .

b) If the quotient is zero, the conversion is complete. Otherwise repeat step (a) using the quotient as the decimal number. The new remainder is the next most significant bit of the **ANY BASE** number.

a) Multiply each bit of the **ANY BASE** number by its corresponding bit-weighting factor (i.e., Bit-0→N^0; Bit-1→N^1; Bit-2→N^2; etc).

b) Sum up all of the products in step (a) to get the decimal number.

Decimal ↔ Octal Conversion

The Process: Successive Division

- Divide number by **8**; R is the LSB of the **octal** number
- While Q is 0
  - Using the Q as the decimal number.
  - New remainder is MSB of the **octal** number.

$$8\overline{)\,94}\quad r = 6 \leftarrow \text{LSB}$$

$$8\overline{)\,11}\quad r = 3$$

$$8\overline{)\,1}\quad r = 1 \leftarrow \text{MSB}$$

$94_{10} = 136_8$

Decimal ↔ Hexadecimal Conversion

The Process: Successive Division

- Divide number by **16**; R is the LSB of the **hex** number
- While Q is 0
  - Using the Q as the decimal number.
  - New remainder is MSB of the **hex** number.

$$16\overline{)\,94}\quad r = E \quad \leftarrow \text{LSB}$$

$$16\overline{)\,5}\quad r = 5 \quad \leftarrow \text{MSB}$$

$94_{10} = 5E_{16}$

Example: Hex → Octal

Example:

Convert the hexadecimal number $5A_H$ into its octal equivalent.

Solution:

First convert the hexadecimal number into its decimal equivalent, then convert the decimal number into its octal equivalent.

5	A
16^1	16^0
16	1
80	10

$80 + 10 = 90_{10}$

$$8 \overline{)\, 90} \quad \begin{matrix} 11 \\ r = 2 \end{matrix} \leftarrow \text{LSB}$$

$$8 \overline{)\, 11} \quad \begin{matrix} 1 \\ r = 3 \end{matrix}$$

$$8 \overline{)\, 1} \quad \begin{matrix} 0 \\ r = 1 \end{matrix} \leftarrow \text{MSB}$$

$\therefore 5A_H = 132_8$

Example: Octal → Binary

Example:

Convert the octal number 132_8 into its binary equivalent.

Solution:

First convert the octal number into its decimal equivalent, then convert the decimal number into its binary equivalent.

1	3	2
8^2	8^1	8^0
64	8	1

64 + 24 + 2 = 90_{10}

$$2\overline{)90}\quad r=0 \leftarrow \text{LSB}$$
$$2\overline{)45}\quad r=1$$
$$2\overline{)22}\quad r=0$$
$$2\overline{)11}\quad r=1$$
$$2\overline{)5}\quad r=1$$
$$2\overline{)2}\quad r=0$$
$$2\overline{)1}\quad r=1 \leftarrow \text{MSB}$$

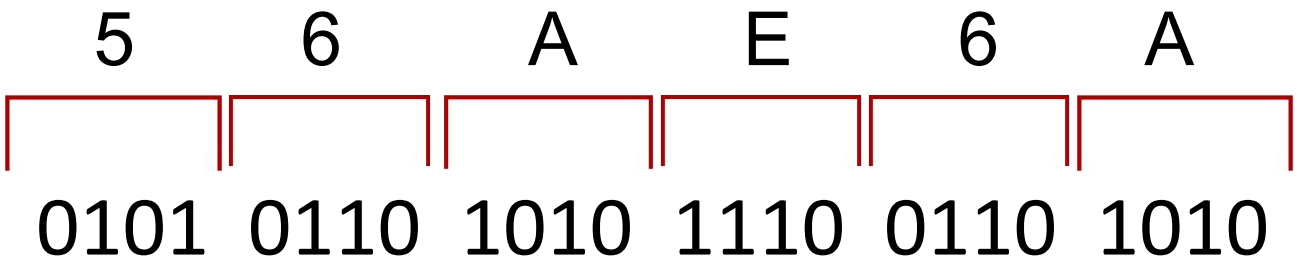
$132_8 = 1011010_2$

Binary ↔ Octal ↔ Hex Shortcut

- Relation
  - Binary, octal, and hex number systems
  - All powers of two
- Exploit (This Relation)
  - Make conversion easier.

Substitution Code

Convert $01010110101011100110101010_2$ to hex using the 4-bit substitution code :



$56AE6A_{16}$

Substitution Code

Substitution code can also be used to convert binary to octal by using 3-bit groupings:

2	5	5	2	7	1	5	2
010	101	101	010	111	001	101	010

$$25527152_8$$

Other Representation

- Signed & Unsigned Number
- Signed number last bit (one MSB) is signed bit

  Assume: 8 bit number

  Unsigned 12 : 0000 1100

  Signed +12 : 0000 1100

  Signed -12 : 1000 1100
- Complement number

  Unsigned binary 12 = 00001100

  1's Complement of 12 = 1111 0011

Thanks

Operations in Number System & Boolean Algebra

Outline

- Number System & Conversion
  - Decimal, Binary, Octal, Hex
- Other representation: Signed, Complement
- Operations in Number System
  - Add, Sub, Mul, Div, Mod
- How to handle real number efficiently
  - Float, Double
- Boolean Algebra: Gates & Theorem

Operation on Numbers

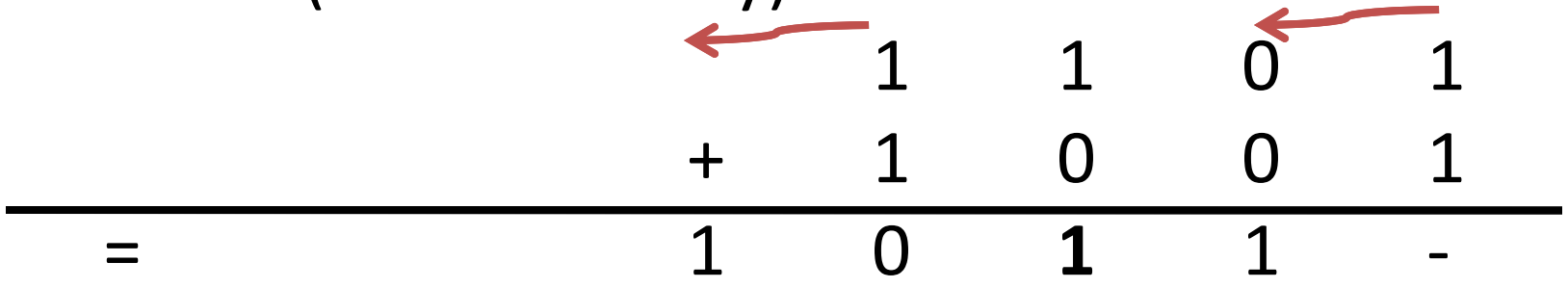
- Addition
- Subtraction
- Multiplication
- Division
- Modulus

Binary Addition

- One bit

0	+	0 =	0	
1	+	0 =	1	
0	+	1 =	1	
1	+	1 =	0	1 (Carry bit)

- Multibit (consider carry)



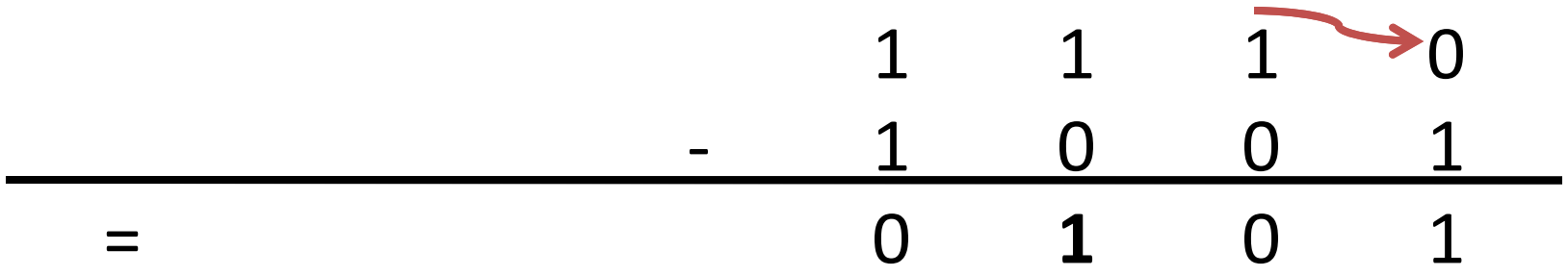
	1	1	0	1	
+	1	0	0	1	
=	1	0	**1**	1	-

Binary Subtraction

- One bit

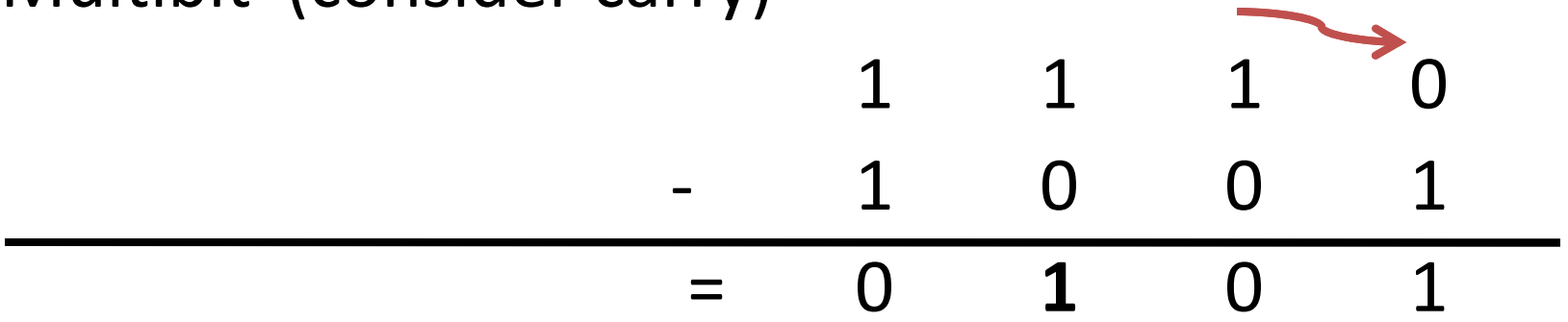
0	-	0 =	0	
1	-	0 =	1	
0	-	1 =	1	1 (Carry bit)
1	-	1 =	0	

- Multibit (consider carry)

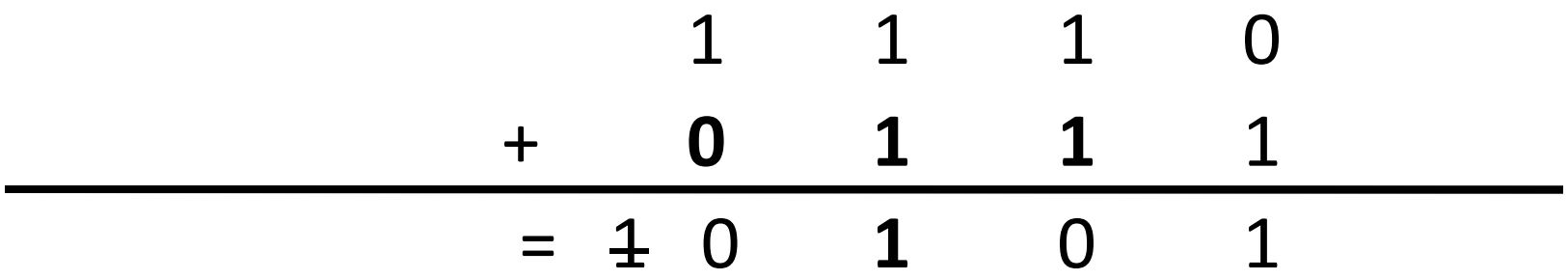


Binary Subtraction

- Multibit (consider carry)



- add 2's complement= (1001)'+1= 0110+1=0111
- Other way (add 2's complement & discard carry)



Binary Multiplication

- Repeated addition
- Many improved technique

			1	0	0	0	$=8_{10}$	
		X	0	1	1	0	$=6_{10}$	
=			0	0	0	0		
		+	1	0	0	0		
	+	1	0	0	0			
+	0	0	0	0				
0	1	1	0	0	0	0	=	48_{10}

Binary Division & Modulus

- $0110010 \div 011 = 50_{10} \div 3_{10}$

```
      011 )  0 1 1 0 0 1 0 (
             0 1 1          ( 1
             0 0 0
      ---------------------------------
               0 0 0        ( 0
               0 0 0
      ---------------------------------
                 0 0 0      ( 0
                 0 0 0
      ---------------------------------
                   0 0 1
                   0 0 0    ( 0          Q=1000=16₁₀
      ---------------------------------
                     0 1 0                R=10= 2₁₀
                     0 0 0    ( 0
```

Hex Addition

- Addition

	1	A	2	B
+	7	C	A	6
=	9	6	D	1

Carry Value
to higher significant
one is 1

- Subtraction

	A	2	B	9
	1	C	F	3
	8	5	C	6

Other Operations to skip

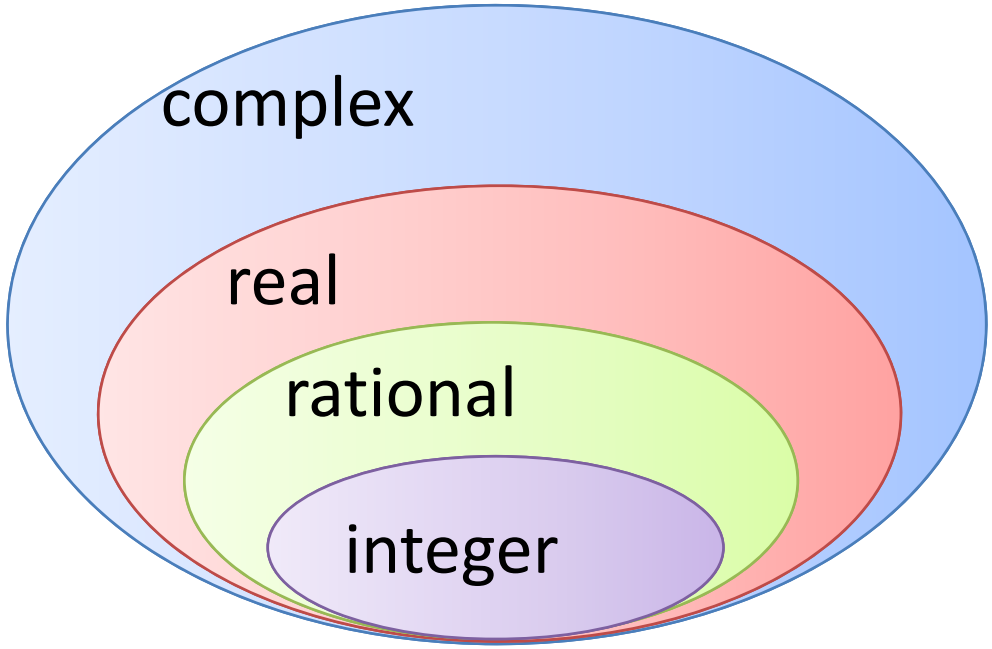
- Hex : Multiplication, Division & Mod
- Oct: Add, Sub, Multiplication, Division & Mod
- Read yourself

Floating Point Numbers

- Fractional Number: 2034.455
- Floating format: 2.03455×10^3
- In Binary: 1.100011×2^7
- Real numbers must be normalized using scientific notation:
  $0.1\ldots \times 2^n$ where n is an integer
- Note that the whole number part is always 0 and the most significant digit of the fraction is a 1 – ALWAYS!

Need to go beyond integers

- integer 7
- rational 5/8
- real $\sqrt{3}$
- complex 2 - 3 i



Extremely large and small values:

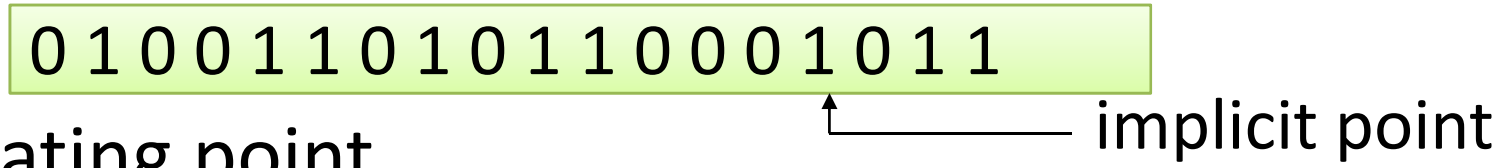
- distance pluto - sun = $5.9\ 10^{12}$ m
- mass of electron = 9.1×10^{-28} gm

Representing fractions

- Integer pairs (for rational numbers)

- Strings with explicit decimal point

- Implicit point at a fixed position



- Floating point

  fraction x base $^{\text{power}}$

Numbers with binary point

$101.11 = 1x2^2 + 0x2^1 + 1x2^0 + \ . \ + 1x2^{-1} + 1x2^{-2}$

$= 4 + 1 + .+ \ 0.5 + 0.25 = 5.75_{10}$

$0.6 = 0.1001100110011001100110011001.....$

$.6 \times 2 = 1 + .2$

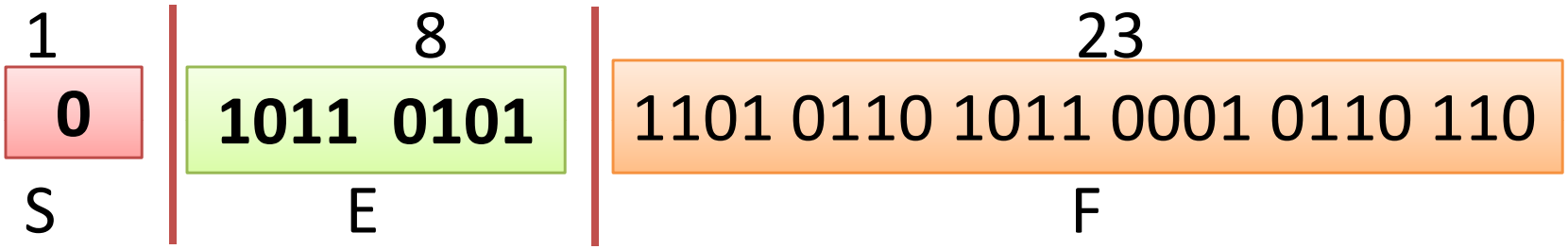
$.2 \times 2 = 0 + .4$

$.4 \times 2 = 0 + .8$

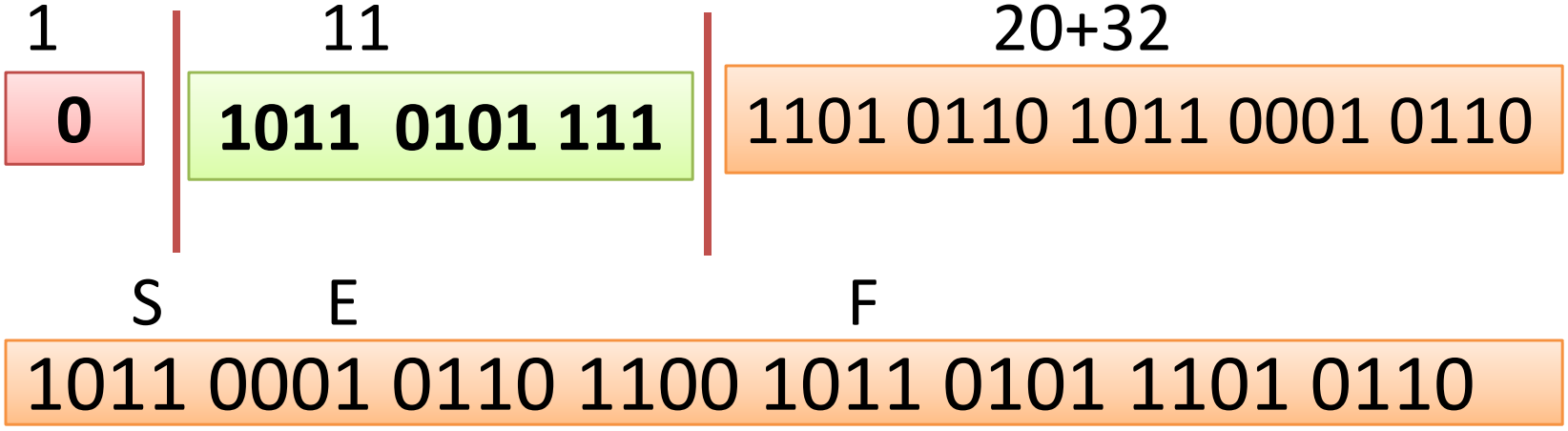
$.8 \times 2 = 1 + .6$

IEEE 754 standard

- Single precision numbers



- Double precision numbers



Float operations

- Add, Sub
  - Make same power, operate, normalize

    $2x10^6 + 3X10^4 = 2x10^6 + 0.03x10^6 = 2.03x10^6$

    $2x10^6 - 3X10^4 = 2x10^6 - 0.03x10^6 = 1.97x10^6$
- Mul, Div
  - Do operation, normalize
  - $2.0x10^6 \times 3.0x10^3 = 2x3 \times 10^{(6+3)} = 6.0x10^9$
  - $2x10^6 \div 3x10^3 = 2/3 \times 10^{(6-3)} = 0.666x10^3 = 6.66x10^2$

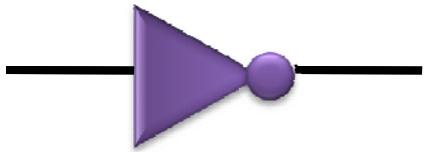
Boolean Algebra

- Computer hardware using binary circuit greatly simply design
- Binary circuits: To have a conceptual framework to manipulate the circuits algebraically
- George Boole (1813-1864): developed a mathematical structure
  - To deal with binary operations with just two values.

Basic Gates in Binary Circuit

- Element 0 : “FALSE”. Element 1 : “TRUE”.
- ‘+’ operation “OR”,‘*’ operation “AND” and ’ operation “NOT”.





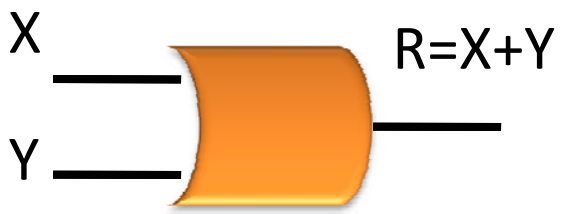
OR	0	1
0	0	1
1	1	1

AND	0	1
0	0	0
1	0	1

NOT	
0	1
1	0

OR Gate

- '+' operation "OR"



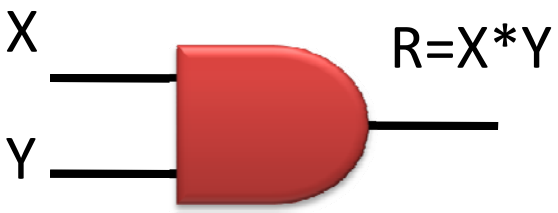
OR	*0*	*1*
0	*0*	*1*
1	*1*	*1*

X	Y	R=X OR Y R= X + Y
0	0	0
0	1	1
1	0	1
1	1	1

1 + Y = 1

AND Gate

- ‘*’ operation “AND”



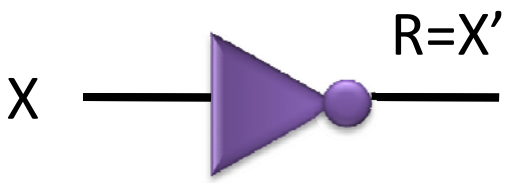
AND	0	1
0	0	0
1	0	1

X	Y	R=X AND Y R= X * Y
0	**0**	**0**
0	**1**	**0**
1	0	0
1	1	1

0 * Y = 0

NOT Gate

- ‘ operation “NOT”



X	R=X’ R= NOT X
0	1
1	0

Boolean Algebra Defined

- Boolean Algebra B : 5-tuple

{B, +, *, ’, 0, 1}

- **+** and ***** are *binary* operators,
- **‘** is a *unary* operator.

Boolean Algebra Defined

- ***Axiom #1: Closure***

  If a and b are Boolean

  (a + b) and (a * b) are Boolean.

- ***Axiom #2: Cardinality***

  if a is Boolean then a' is Boolean

- ***Axiom #3: Commutative***

  (a + b) = (b + a)

  (a * b) = (b * a)

Boolean Algebra Defined

- *Axiom #4: Associative :* **If a and b are Boolean**

$$(a + b) + c = a + (b + c)$$

$$(a * b) * c = a * (b * c)$$

- *Axiom #6: Distributive*

$$a * (b + c) = (a * b) + (a * c)$$

$$a + (b * c) = (a + b) * (a + c)$$

2nd one is Not True for Decimal numbers System

Boolean Algebra Defined

- *Axiom #5: Identity Element :*
  - **B has identity to + and ***

0 is identity element for + : a + 0 = a

1 is identity element for * : a * 1 = a

- *Axiom #7: Complement Element*

a + a' = 1

a * a' = 0

Thanks

Boolean Algebra

Outline

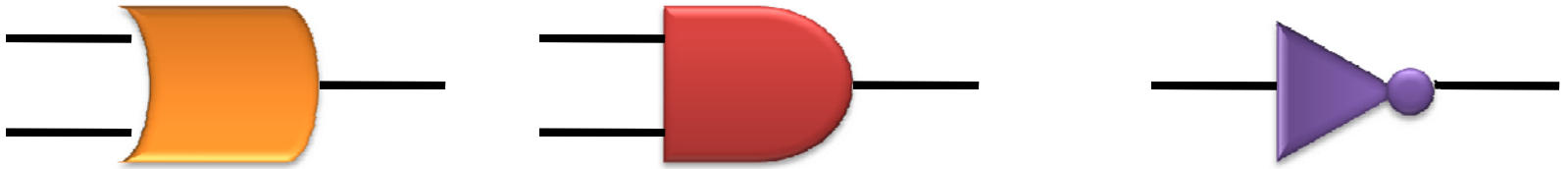
- Basic Gates in Digital Circuit
- Boolean Algebra : Definitions, Axioms
- Named, Simplification & Consensus Theorems
- Duality Principle, Shannon's Expansion
- Proof : Using Truth Table, Using Theorem
- Boolean function: Representation, Canonical form

Boolean Algebra

- Computer hardware using binary circuit greatly simply design
- Binary circuits: To have a conceptual framework to manipulate the circuits algebraically
- George Boole (1813-1864): developed a mathematical structure
  - To deal with binary operations with just two values.

Basic Gates in Binary Circuit

- Element 0 : “FALSE”. Element 1 : “TRUE”.
- ‘+’ operation “OR”,‘*’ operation “AND” and ’ operation “NOT”.



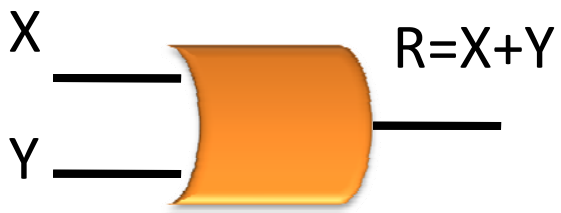
OR	0	1
0	0	1
1	1	1

AND	0	1
0	0	0
1	0	1

NOT	
0	1
1	0

OR Gate

- ‘+’ operation “OR”



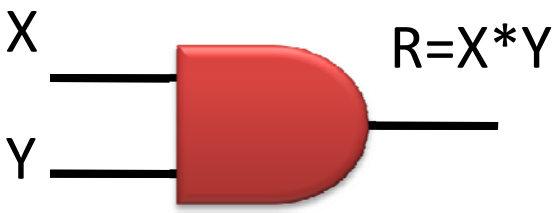
OR	*0*	*1*
0	*0*	*1*
1	*1*	*1*

X	Y	R=X OR Y R= X + Y
0	0	0
0	1	1
1	0	1
1	1	1

1 + Y = 1

AND Gate

- ‘*’ operation “AND”



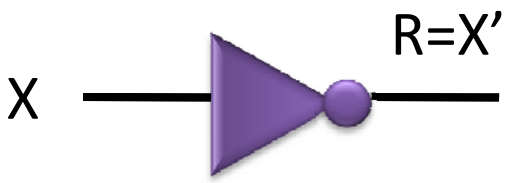
AND	0	1
0	0	0
1	0	1

X	Y	R=X AND Y R= X * Y
0	**0**	**0**
0	**1**	**0**
1	0	0
1	1	1

0 * Y = 0

NOT Gate

- ‘ operation “NOT”



X	R=X’ R= NOT X
0	1
1	0

Boolean Algebra Defined

- Boolean Algebra B : 5-tuple

{B, +, *, ’, 0, 1}

- **+** and ***** are *binary* operators,
- **‘** is a *unary* operator.

Boolean Algebra Defined

- ***Axiom #1: Closure***

  If a and b are Boolean

  (a + b) and (a * b) are Boolean.

- ***Axiom #2: Cardinality***

  if a is Boolean then a' is Boolean

- ***Axiom #3: Commutative***

  (a + b) = (b + a)

  (a * b) = (b * a)

Boolean Algebra Defined

- ***Axiom #4: Associative :*** **If a and b are Boolean**

$$(a + b) + c = a + (b + c)$$

$$(a * b) * c = a * (b * c)$$

- ***Axiom #6: Distributive***

$$a * (b + c) = (a * b) + (a * c)$$

$$a + (b * c) = (a + b) * (a + c)$$

Boolean Algebra Defined

- ***Axiom #5: Identity Element :***
  - **B has identity to + and ***

0 is identity element for + : a + 0 = a

1 is identity element for * : a * 1 = a

- ***Axiom #7: Complement Element***

a + a' = 1

a * a' = 0

Terminology

- **Juxtaposition implies * operation:**

$$ab = a * b$$

- **Operator order of precedence is:**

$$( ) > \ ' \ > * > +$$

$$a+bc = a+(b*c) \neq (a+b)*c$$

$$ab' = a(b') \neq (a*b)'$$

Named Theorems

Idempotent	a + a = a	a * a = a
Boundedness	a + 1 = 1	a * 0 = 0
Absorption	**a + (a*b) = a**	**a*(a+b) = a**
Associative	(a+b)+c= a+(b+c)	(a*b)*c= a*(b*c)

Involution	(a')' = a	
DeMorgan's	(a+b)' = a' * b'	(a*b)'=a' + b'

Simplification Theorem

- Uniting :

  XY + XY' = X X(Y+Y')=X.1=X

  (X + Y)(X + Y') = X XX+XY'+YX+YY'=X+X(**Y+Y'**)+0=X

- Absorption:

  X + XY = X X(1+Y)=X.1=X

  X(X + Y) = X XX+XY=X+XY=X

- Adsorption

  (X + Y')Y = XY, XY' + Y = X + Y XY+YY'=XY+0=XY

Principle of Duality

- Dual of a statement S is obtained
  - By interchanging * and +
  - By interchanging 0 and 1
  - ~~By interchanging for all x by x' also valid (for an expression)~~
- Dual of (a*1)*(0+a') = 0 is (a+0)+(1*a') = 1

(a+b)' = a' * b '

~~(a+b) * 1 = (a'*b')+0~~

Consensus Theorem

- **XY + X'Z + YZ = XY + X'Z**

  XY + X'Z + YZ

  = xy + x'z + (x + x')yz

  = xy + x'z + xyz + x'yz

  = xy + xyz + x'z + x'yz

  = xy(1 + z) + x'z(1 + y)

  = xy + x'z

Consensus (collective opinion) of X.Y and X'.Z is Y.Z

- **(X + Y)(X' + Z)(Y + Z) = (X + Y)(X' + Z)**

Shannon Expansion

- **F(X, Y, Z)= X . F(1,Y,Z) + X'. F(0, Y, Z)**

Example:

XY+X'Z+YZ

=X. (1.Y+0.Z+YZ) + X' (0.Y+1.Z+YZ)

=X.(Y+YZ)+X'(Z+YZ)

=X.(Y(1+Z))+X'(Z(1+Y))

= X(Y.1)+X'(Z.1)

=XY+X'Z

N-bit Boolean Algebra

- Single bit to *n-bit* Boolean Algebra
- Let a = 1101010, b = 1011011
  - a + b = 1101010 + 1011011 = 1111011
  - a * b = 1101010 * 1011011 = 1001010
  - a’ = 1101010’ = 0010101

Proof by Truth Table

- Consider the distributive theorem:

  $a + (b * c) = (a + b)*(a + c)$

  Is it true for a two bit Boolean Algebra?

- Can prove using a truth table
  - How many possible combinations of *a*, *b*, and *c* are there?
- Three variables, each with two values
  - **2*2*2 = 2^3 = 8**

Proof by Truth Table

a	b	c	b*c	a+(b*c)	a+b	a+c	(a+b)*(a+c)
0	*0*	*0*	0	0	0	0	0
0	*0*	*1*	0	0	0	1	0
0	*1*	*0*	0	0	1	0	0
0	*1*	*1*	1	1	1	1	1
1	*0*	*0*	0	1	1	1	1
1	*0*	*1*	0	1	1	1	1
1	*1*	*0*	0	1	1	1	1
1	*1*	*1*	1	1	1	1	1

Proof using Theorems

- Use the properties of Boolean Algebra to proof

$$(x + y)(x + x) = x$$

- *Warning, make sure you use the laws precisely*

$(x + y)(x + x)$	**Given**
$(x + y)x$	**Idempotent**
$x(x + y)$	**Commutative**
x	**Absorption**

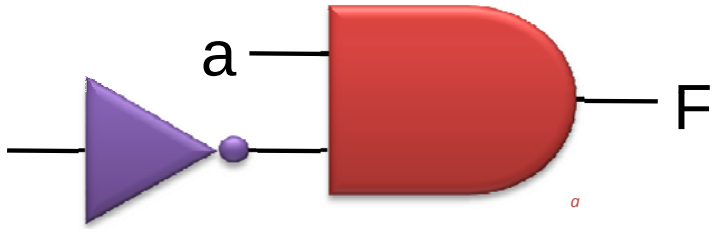
Converting to Boolean Equations

- Convert the following English statements to a Boolean equation
  - Q1. a is 1 and b is 1.
    - Answer: F = a AND b = ab
  - Q2. either of a or b is 1.
    - Answer: F = a OR b = a+b
  - Q3. both a and b are not 0.
    - Answer:
      - (a) Option 1: F = NOT(a) AND NOT(b) = a'b'
      - (b) Option 2: F = a OR b = a+b
  - Q4. a is 1 and b is 0.
    - Answer: F = a AND NOT(b) = ab'

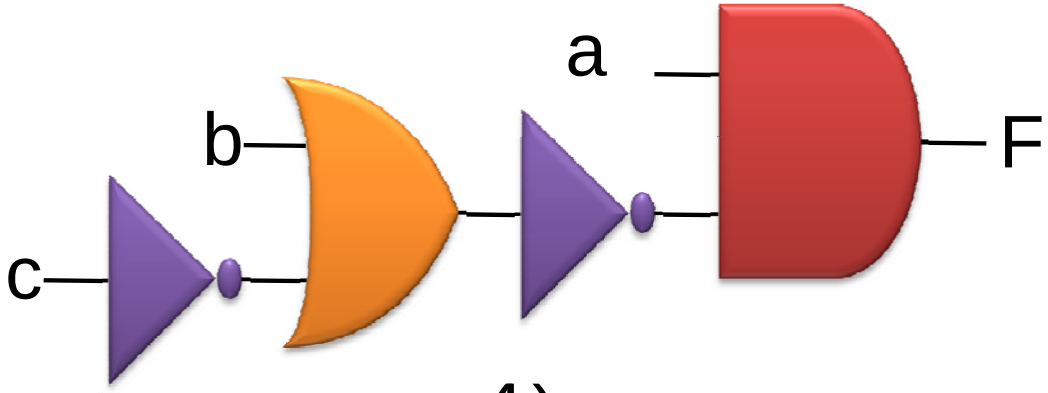
Example: Converting a Boolean Equation to a Circuit of Logic Gates

- Q: Convert the following equation to logic gates:

  F = a AND NOT(b OR NOT(c))

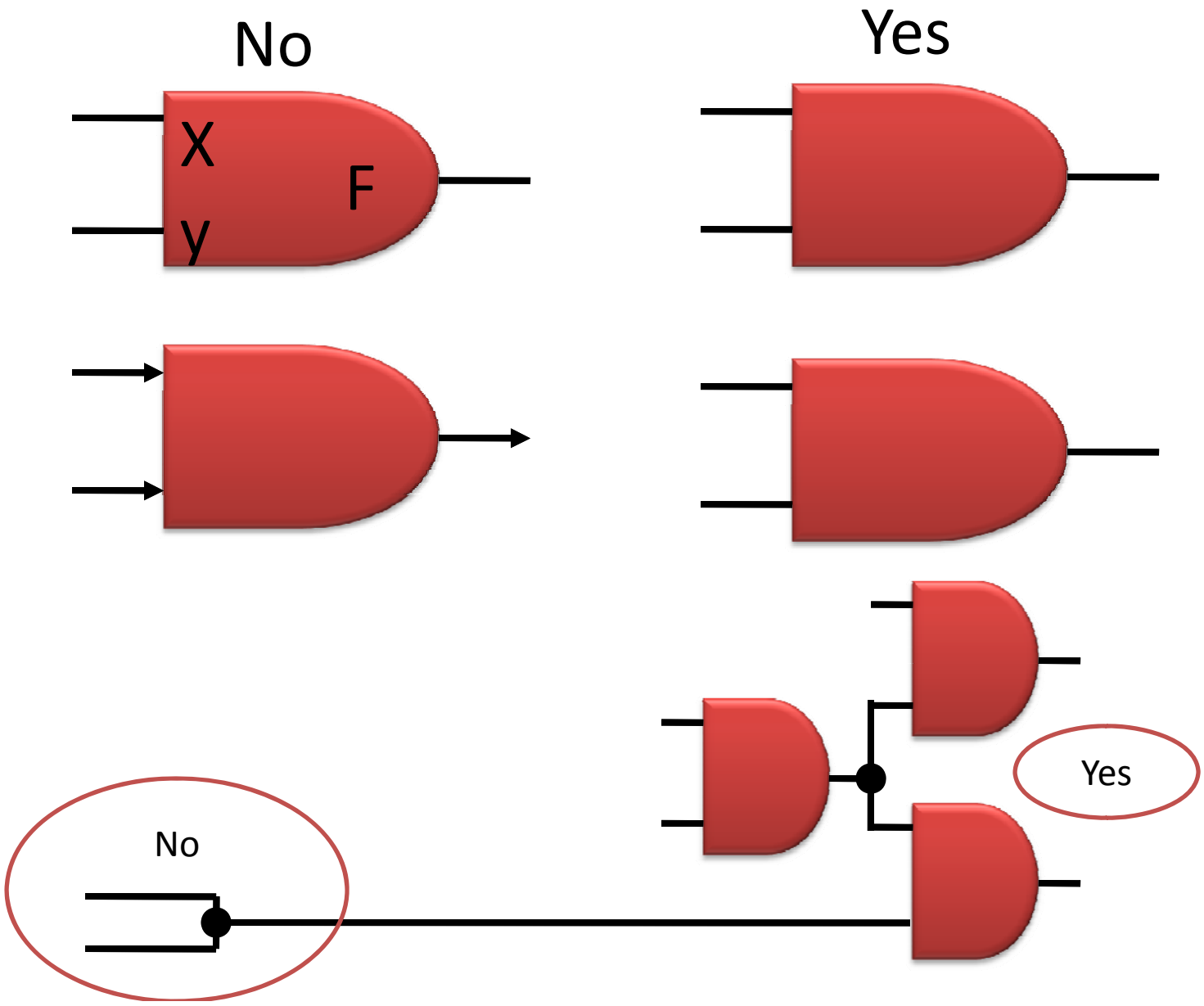


(a)



(b)

Some Circuit Drawing Conventions



Thanks

More Gates & Minterm-Maxterm

Outline

- *Review: Duality Principle*
- Mores gates: XOR,XNOR, NAND, NOR
- Design
  - Min term & Sum of Product
  - Max Term & Product of SUM
- Boolean function: Representation, Canonical form
- Karnough map simplification

Principle of Duality

- Dual of a statement S is obtained
  - By interchanging * and +
  - By interchanging 0 and 1
  - ~~By interchanging for all x by x' also valid (for an expression) only for simple demorgan law~~
- Dual of (a*1)*(0+a') = 0 is (a+0)+(1*a') = 1

  (a+b)' = a' * b '

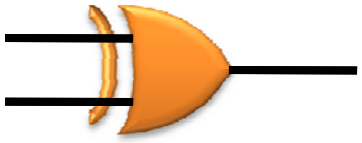
  (a+b) * 1 = (a'*b')+0

Duality examples

- x + 0 = x x.1=x
- X+x’=1 x.x’=0
- A+B’C A. (B’+C)
- A’B’+AB (A’+B’).(A+B)

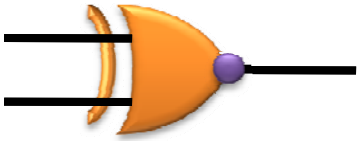
Logic Gates: XOR, XNOR

- XOR



A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

- XNOR



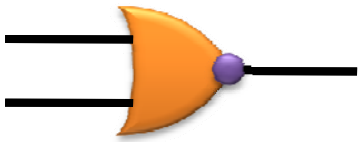
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

Logic Gates: NAND, NOR

- NAND

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

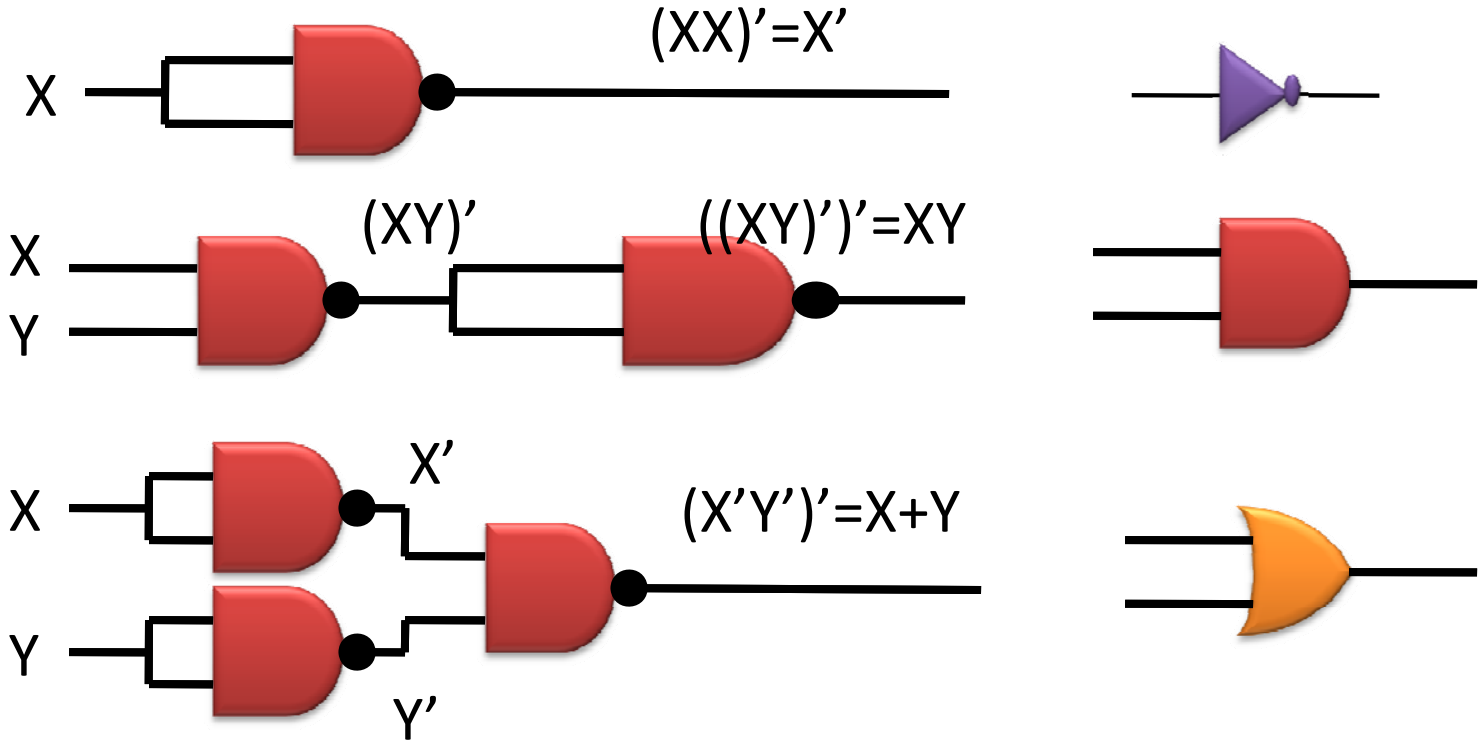
- NOR



A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

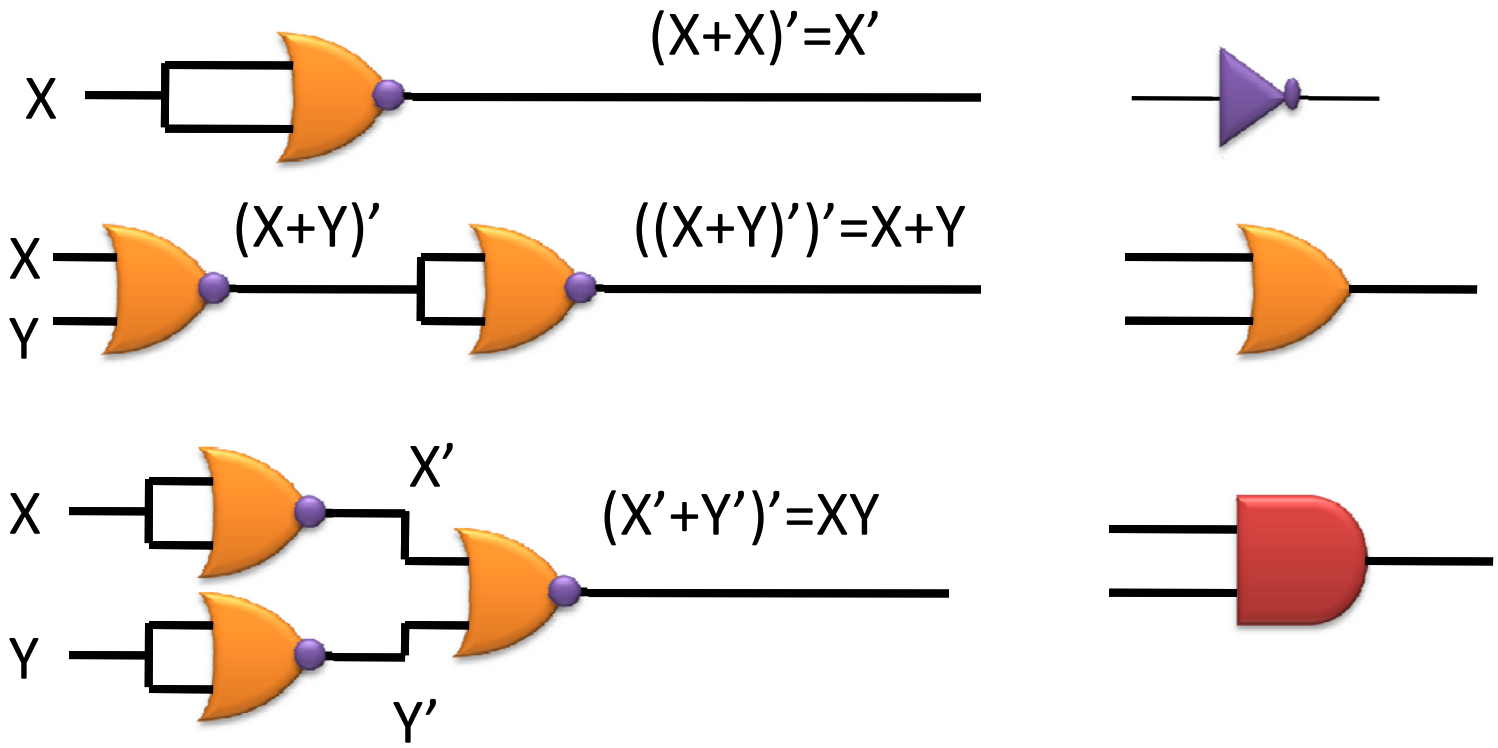
NAND & NOR are universal

- (xx)’ = x’
- ((xy)’)’= xy
- (x’y’)’= x+y

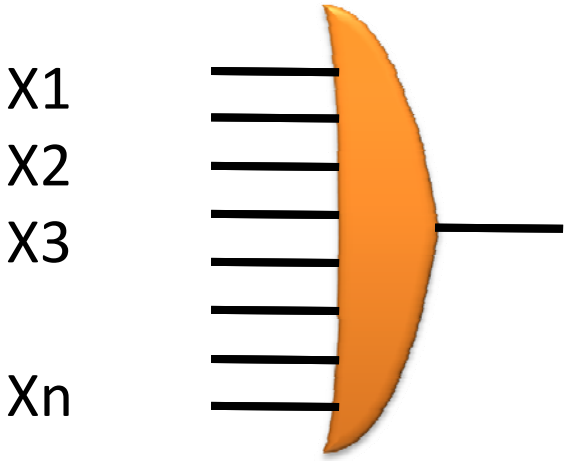


NAND & NOR are universal

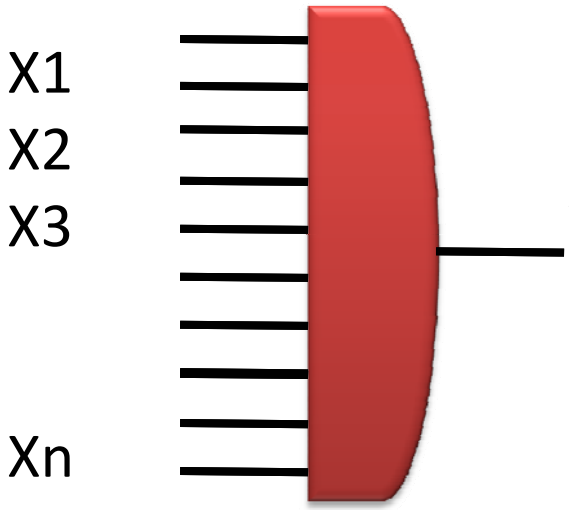
- (x+x)’ = x’
- ((x+y)’)’= x+y
- (x’+y’)’= x.y



Multi-input gate



X1+X2+X3+ … +Xn



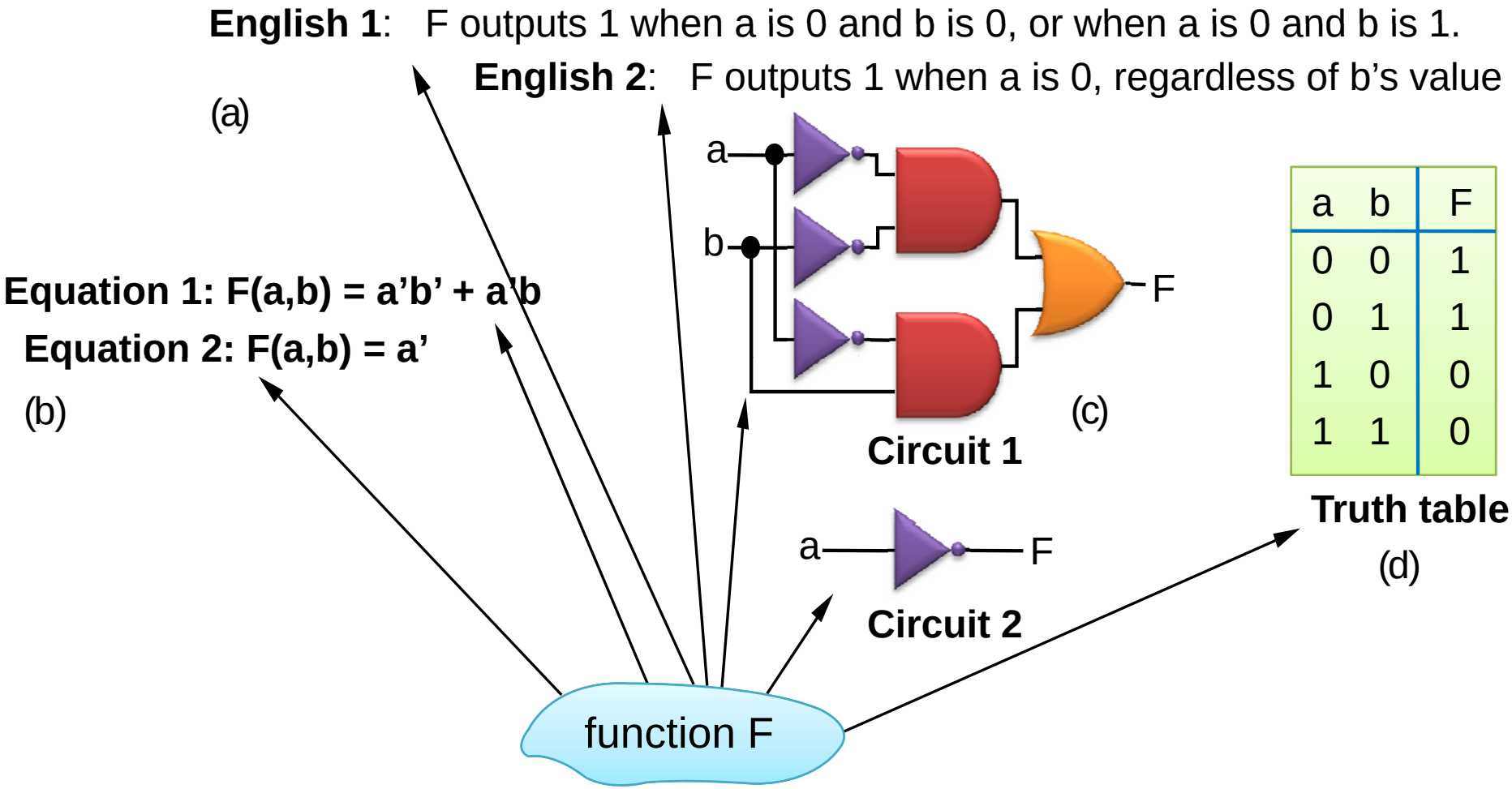
X1 . X2. X3. … .Xn

Boolean Functions:Terminology

F(a,b,c) = a'bc + abc' + ab + c

- Variable
  - Represents a value (0 or 1), Three variables: **a**, **b**, and **c**
- Literal
  - Appearance of a variable, in true or complemented form
  - Nine literals: **a'**, **b, c, a, b, c', a, b,** and **c**
- Product term
  - Product of literals, Four product terms: **a'bc, abc', ab, c**
- Sum-of-products (SOP)
  - Above equation is in sum-of-products form.
  - "F = (a+b)c + d" is not.

Representations of Boolean Functions



- A function can be represented in different ways
  - Above shows seven representations of the same functions F(a,b), using four different methods: English, Equation, Circuit, and Truth Table

Truth Table Representation of Functions

- Define value of F for each possible combination of input values
  - 2-input function: 4 rows
  - 3-input function: 8 rows
  - 4-input function: 16 rows
- Q: Use truth table to define function F(a,b,c) that is 1 when abc is 5 or greater in binary

a	b	F
0	0	
0	1	
1	0	
1	1	

(a)

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	

(b)

a	b	c	d	F
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	

(c)

a	b	c	F
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Converting among Representations

- Can convert from any representation to any other
- Common conversions
  - Equation to circuit
  - Truth table to equation
  - Equation to truth table
    - Easy -- just evaluate equation for each input combination (row)
    - Creating intermediate columns helps

Inputs		Outputs	Term
a	b	F	F = sum of
0	0	1	a'b'
0	1	1	a'b
1	0	0	
1	1	0	

F = a'b' + a'b

Q: Convert to truth table: F = a'b' + a'b

Inputs				Output
a	b	a' b'	a' b	F
0	0	1	0	**1**
0	1	0	1	**1**
1	0	0	0	**0**
1	1	0	0	**0**

Q: Convert to equation

a	b	c	F	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	ab'c
1	1	0	1	abc'
1	1	1	1	abc

F = ab'c + abc' + abc

Standard Representation: Truth Table

- How to determine two functions are the same?
  - Use algebraic methods
  - But if we failed, does that prove *not* equal? No.
- Solution: Convert to truth tables
  - Only ONE truth table representation of given same functions: Standard representation

F = ab + a'

a	b	F
0	0	1
0	1	1
1	0	0
1	1	1

Same

F = a'b' +a'b + ab

a	b	F
0	0	1
0	1	1
1	0	0
1	1	1

Canonical Form -- Sum of Minterms

- Truth tables too big for numerous inputs
- Use standard form of equation instead
  - Known as ***canonical form***
  - Regular algebra: group terms of polynomial by power
    - $ax^2 + bx + c$ ($3x^2 + 4x + 2x^2 + 3 + 1 \rightarrow 5x^2 + 4x + 4$)
  - Boolean algebra: create sum of minterms
    - ***Minterm***: product term with every function literal appearing exactly once, in true or complemented form
    - Just multiply-out equation until sum of product terms
    - Then expand each term until all terms are minterms

Determine if F(a,b)=ab+a' is same function as F(a,b) = a'b'+a'b+ab by to canonical form.

F = ab+a' (already **sum of products**)
F = ab + a'(b+b') (expanding term)
F = ab + a'b + a'b' (it is **canonical form**)

Canonical form or Standard Form

- Canonical forms
  - Sum of minterms (SOM)
  - Product of maxterms (POM)
- Standard forms (may use less gates)
  - Sum of products (SOP)
  - Product of sums (POS)

F = ab+a' (already **sum of products:SOP**)
F = ab + a'(b+b') (expanding term)
F = ab + a'b + a'b' (it is **canonical form:SOM**)

Canonical Forms

- It is useful to specify Boolean functions in a form that:
  - Allows comparison for equality.
  - Has a correspondence to the truth tables
- Canonical Forms in common usage:
  - Sum of Minterms (SOM)
  - Product of Maxterms (POM)

Minterms

- **product term** is a term where literals are ANDed.
  - Example: x'y', xz, xyz, …
- **minterm** : A product term in which all variables appear exactly once, in normal or complemented form
  - Example: F(x,y,z) has 8 minterms

    x'y'z', x'y'z, x'yz', ...
- Function with n variables has 2^n minterms
- A minterm equals 1 at exactly one input combination and is equal to 0 otherwise
  - Example: x'y'z' = 1 only when x=0, y=0, z=0
- A minterm is denoted as m_i where i corresponds the input combination at which this minterm is equal to 1

2 Variable Minterms

- **Two variables (X and Y) produce 2x2=4 combinations**

  XY **(both normal)**

  XY' **(X normal, Y complemented)**

  X'Y **(X complemented, Y normal)**

  X'Y' **(both complemented)**

Maxterms

- **Maxterms are OR terms with every variable in true or complemented form.**

  X+Y (both normal)

  X+Y' (x normal, y complemented)

  X'+Y (x complemented, y normal)

  X'+Y' (both complemented)

Maxterms and Minterms

- **Two variable minterms and maxterms.**

Index	Minterm	Maxterm
0	x’ y’	x + y
1	x ‘y	x + y’
2	x y’	x’ + y
3	x y	x’ + y’

- **The index above is important for describing which variables in the terms are true and which are complemented.**

Minterms

Minterms for Three Variables

X	Y	Z	Product Term	Symbol	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\bar{X}\bar{Y}\bar{Z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\bar{X}\bar{Y}Z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\bar{X}Y\bar{Z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\bar{X}YZ$	m_3	0	0	0	1	0	0	0	0
1	0	0	$X\bar{Y}\bar{Z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$X\bar{Y}Z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$XY\bar{Z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	XYZ	m_7	0	0	0	0	0	0	0	1

Variable complemented if 0
Variable uncomplemented if 1

m_i indicated the i^{th} minterm
i indicates the binary combination
m_i is equal to 1 for ONLY THAT combination

Maxterms

- **Sum term** : A term where literals are ORed.
  - Example: x'+y', x+z, x+y+z, …
- **Maxterm** : a sum term in which all variables appear exactly once, in normal or complemented form
  - Example: F(x,y,z) has 8 maxterms

    (x+y+z), (x+y+z'), (x+y'+z), ...
- Function with n variables has 2^n maxterms
- A maxterm equals 0 at exactly one input combination and is equal to 1 otherwize
  - Example: (x+y+z) = 0 only when x=0, y=0, z=0
- A maxterm is denoted as M_i where i corresponds the input combination at which this maxterm is equal to 0

Maxterms

Maxterms for Three Variables

X	Y	Z	Sum Term	Symbol	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	0	$X+Y+Z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$X+Y+\overline{Z}$	M_1	1	0	1	1	1	1	1	1
0	1	0	$X+\overline{Y}+Z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$X+\overline{Y}+\overline{Z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\overline{X}+Y+Z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\overline{X}+Y+\overline{Z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\overline{X}+\overline{Y}+Z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\overline{X}+\overline{Y}+\overline{Z}$	M_7	1	1	1	1	1	1	1	0

Variable complemented if 1
Variable not complemented if 0

M_i indicated the i^{th} maxterm i indicates the binary combination M_i is equal to 0 for ONLY THAT combination

Expressing Functions with Minterms

- Boolean function can be expressed algebraically from a give truth table
  - Forming sum of ALL the minterms that produce 1 in the function

Example : Consider the function defined by the truth table

$$F(X,Y,Z) = X'Y'Z' + X'YZ' + XY'Z + XYZ$$

$$= m_0 + m_2 + m_5 + m_7$$

$$= \sum m(0, 2, 5, 7)$$

X	Y	Z	m	F
0	**0**	**0**	**m_0**	**1**
0	0	1	m_1	0
0	**1**	**0**	**m_2**	**1**
0	1	1	m_3	0
1	0	0	m_4	0
1	**0**	**1**	**m_5**	**1**
1	1	0	m_6	0
1	**1**	**1**	**m_7**	**1**

Expressing Functions with Maxterms

- Boolean function : Expressed algebraically from a give truth table
- By forming logical product (AND) of ALL the maxterms that produce 0 in the function

Example:

Consider the function defined by the truth table

$F(X,Y,Z) = \Pi\, M(1,3,4,6)$

Applying DeMorgan

$F' = m_1 + m_3 + m_4 + m_6 = \sum m(1,3,4,6)$

$F = F'' = [m_1 + m_3 + m_4 + m_6]'$

$= m_1'.m_3'.m_4'.m_6'$

$= M_1.M_3.M_4.M_6$

$= \Pi\, M(1,3,4,6)$

X	Y	Z	M	F	F'
0	0	0	M_0	1	0
0	**0**	**1**	**M_1**	**0**	**1**
0	1	0	M_2	1	0
0	**1**	**1**	**M_3**	**0**	**1**
1	**0**	**0**	**M_4**	**0**	**1**
1	0	1	M_5	1	0
1	**1**	**0**	**M_6**	**0**	**1**
1	1	1	M_7	1	0

Note the indices in this list are those that are missing from the previous list in $\Sigma m(0,2,5,7)$

Sum of Minterms vs Product of Maxterms

- A function can be expressed algebraically as:
  - The sum of minterms
  - The product of maxterms
- Given the truth table, writing F as
  - $\sum m_i$ – for all minterms that produce 1 in the table, or
  - ΠM_i – for all maxterms that produce 0 in the table
- Minterms and Maxterms are complement of each other.

Example: minterm & maxterm

- Write E = Y' + X'Z' in the form of $\sum m_i$ and $\prod M_i$?
- **Method1**

First construct the Truth Table as shown

$E = \sum m(0,1,2,4,5)$, and

$E = \prod M(3,6,7)$

X	Y	Z	m	M	E
0	0	0	m_0	M_0	**1**
0	0	1	m_1	M_1	**1**
0	1	0	m_2	M_2	**1**
0	1	1	m_3	M_3	0
1	0	0	m_4	M_4	**1**
1	0	1	m_5	M_5	**1**
1	1	0	m_6	M_6	0
1	1	1	m_7	M_7	0

Example (Cont.)

Solution: **Method2_a**

$E = Y' + X'Z'$

$= Y'(X+X')(Z+Z') + 'Z'(Y+Y')$

$= (XY'+X'Y')(Z+Z') + X'YZ'+ X'Z'Y'$

$= Y'Z+X'Y'Z+XY'Z'+X'Y'Z'+ X'YZ'+X'Z'Y'$

$= m_5 + m_1 + m_4 + m_0 + m_2 + m_0$

$= m_0 + m_1 + m_2 + m_4 + m_5$

$= \sum m(0,1,2,4,5)$

To find the form ΠMi, consider the remaining indices

$E = \Pi M(3,6,7)$

Solution: **Method2_b**

$E = Y' + X'Z'$

$E' = Y(X+Z)$

$= YX + YZ$

$= YX(Z+Z') + YZ(X+X')$

$= XYZ+XYZ'+X'YZ$

$E =$

$(X'+Y'+Z')(X'+Y'+Z)(X+Y'+Z')$

$= M_7 . M_6 . M_3$

$= \Pi M(3,6,7)$

To find the form Sm_i, consider the remaining indices

$E = \sum m(0,1,2,4,5)$

Canonical Forms

- The sum of minterms and the product of maxterms forms are known as the **canonical forms** of a function.

Standard Forms

- Sum of Products (SOP) and Product of Sums (POS) are also standard forms
  - AB+CD = (A+C)(B+C)(A+D)(B+D)
- The sum of minterms is a special case of the SOP form, where all product terms are minterms
- The product of maxterms is a special case of the POS form, where all sum terms are maxterms

SOP and POS Conversion

SOP → POS

F = AB + CD

= (AB+C)(AB+D)

= (A+C)(B+C)(AB+D)

= (A+C)(B+C)(A+D)(B+D)

Hint 1: Use id15: X+YZ=(X+Y)(X+Z)

Hint 2: Factor

POS → SOP

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

= (A'+BC)(C+D)

= A'C+A'D+BCC+BCD

= A'C+A'D+BC+BCD

= A'C+A'D+BC

Hint 1: Use id15 (X+Y)(X+Z)=X+YZ

Hint 2: Multiply

Thanks

KMap-Logic Minimization

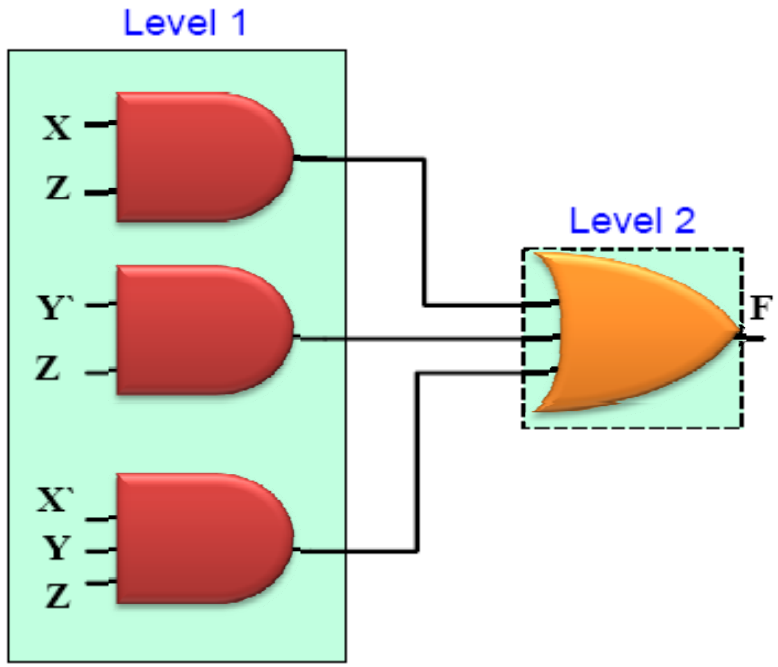
Outline

- SOP & POS Implementation
- Boolean function: Representation
  - SOM (Sum of Minterms), Canonical form
  - SOP (Sum of Product)
- Karnough map simplification
  - 2, 3, 4 variable karnaugh map
  - Don’t care condition
  - Algorithm for better grouping
- Karnaugh map with >= 5 variable

Implementation of SOP

$$F(X,Y,Z) = XZ+Y'Z+X'YZ$$

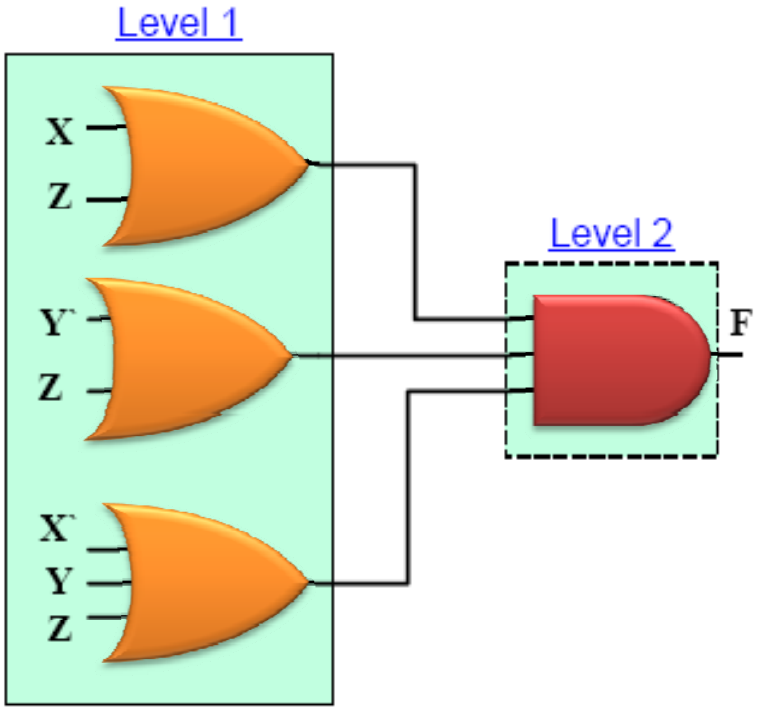
- Any SOP expression can be implemented using 2-levels of gates
- The 1st level consists of AND gates, and the 2nd level consists of a single OR gate
- Also called 2-level Circuit



Implementation of POS

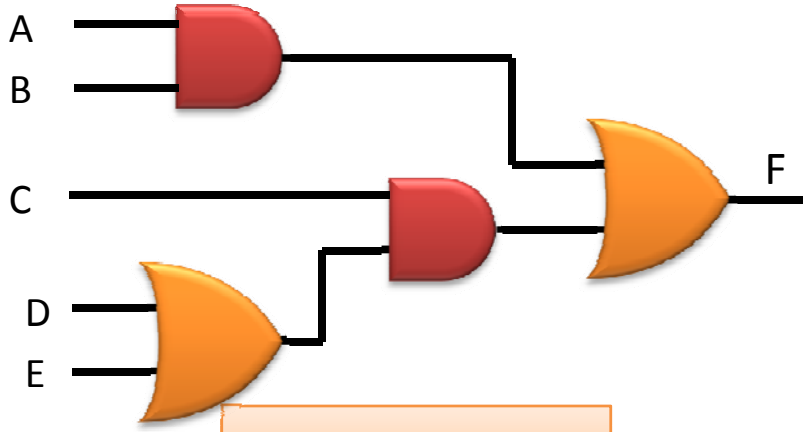
$$F(X,Y,Z) = (X+Z)(Y'+Z)(X'+Y+Z)$$

- Any POS expression can be implemented using 2-levels of gates
- The 1st level consists of OR gates, and the 2nd level consists of a single AND gate
- Also called 2-level Circuit

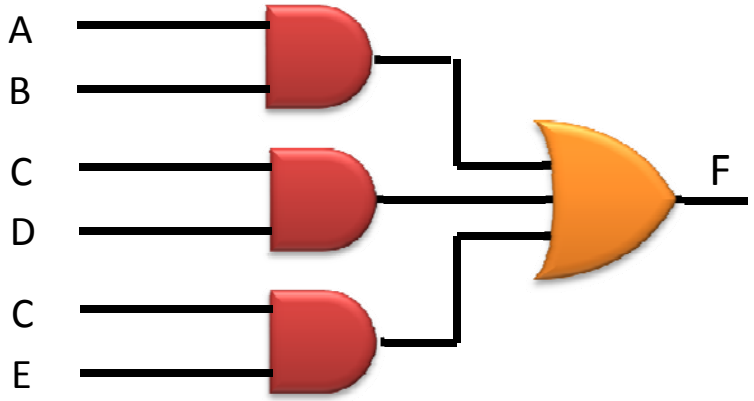


Implementation of SOP

- Consider F = AB + C(D+E)
  - This expression is NOT in the sum-of-products form
  - Use the identities/algebraic manipulation to convert to a standard form (sum of products), as in F = AB + CD + CE
- Logic Diagrams:



3-level circuit



2-level circuit

Canonical Forms

- **It is useful to specify Boolean functions in a form that:**
  - **Allows comparison for equality.**
  - **Has a correspondence to the truth tables**
- **Canonical Forms in common usage:**
  - **Sum of Minterms (SOM)**
  - **Product of Maxterms (POM)**

Simplification: Theorem method

$E = \sum m(0,1,2,4,5)$

$= m_0 + m_1 + m_2 + m_4 + m_5$

$= m_5 + m_1 + m_4 + m_0 + m_2 + m_0$

$= \mathbf{XY'Z+X'Y'Z+XY'Z'+X'Y'Z'+X'YZ' +X'Z'Y'}$

$= (XY'+X'Y')(Z+Z') + X'YZ'+ X'Z'Y'$

$= Y'(X+X')(Z+Z') + X'Z'(Y+Y')$

$= \mathbf{Y' + X'Z'}$

Simplified one: Require less Gates and faster
2 Level

Both are in SOP format : 2 level

Simplification of Boolean Functions

- An implementation of a Boolean Function requires the use of logic gates.
- A smaller number of gates, with each gate (other then Inverter) having less number of inputs, may reduce the cost of the implementation.
- There are 2 methods for simplification of Boolean functions.

Simplification of Boolean Functions: Two Methods

- **Algebraic method** by using Identities & Theorem
- **Graphical method** by using Karnaugh Map method
  - The K-map method is easy and straightforward.
  - A K-map for a function of n variables consists of 2^n cells, and,
  - in every row and column, two adjacent cells should differ in the value of only one of the logic variables.

Karnaugh Map Method

- A graphical method of simplifying logic equations or truth tables.
- Also called a K map
- Theoretically can be used for any number of input variables, but practically limited to 5 or 6 variables.

Karnaugh Map Advantages

- Minimization can be done more systematically
- Much simpler to find minimum solutions
- Easier to see what is happening (graphical)
- Almost always used instead of boolean minimization.

Gray Codes

- Gray code is a binary value encoding in which adjacent values only differ by one bit

2-bit Gray Code
00
01
11
10

Truth Table Adjacencies

F = A’

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

These are *adjacent in a gray code sense*

- they differ by 1 bit

We can apply XY + XY’ = X

A’B’ + A’B = A’(B’+B) = A’(1) = A’

F = B

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

Same idea: A’B + AB = B

Key idea:

Gray code adjacency allows use of simplification theorems

Problem:
Physical adjacency in truth table does not indicate gray code adjacency

Karnaugh Map Method

- The truth table values are placed in the K map.
- **Adjacent K map square differ in only one variable both horizontally and vertically.**
- **The pattern from top to bottom and left to right must be in the form**
- A SOP expression can be obtained by ORing all squares that contain a 1.

A'B', A'B, AB, AB'

00, 01, 11, 01

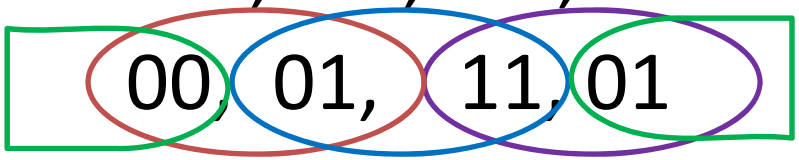
Filling of Karnaugh Map

Why not: A'B', A'B, AB', AB

Only two adjacent can be grouped

Group Reduce a variable: AB'+AB=A(B'+B)=A

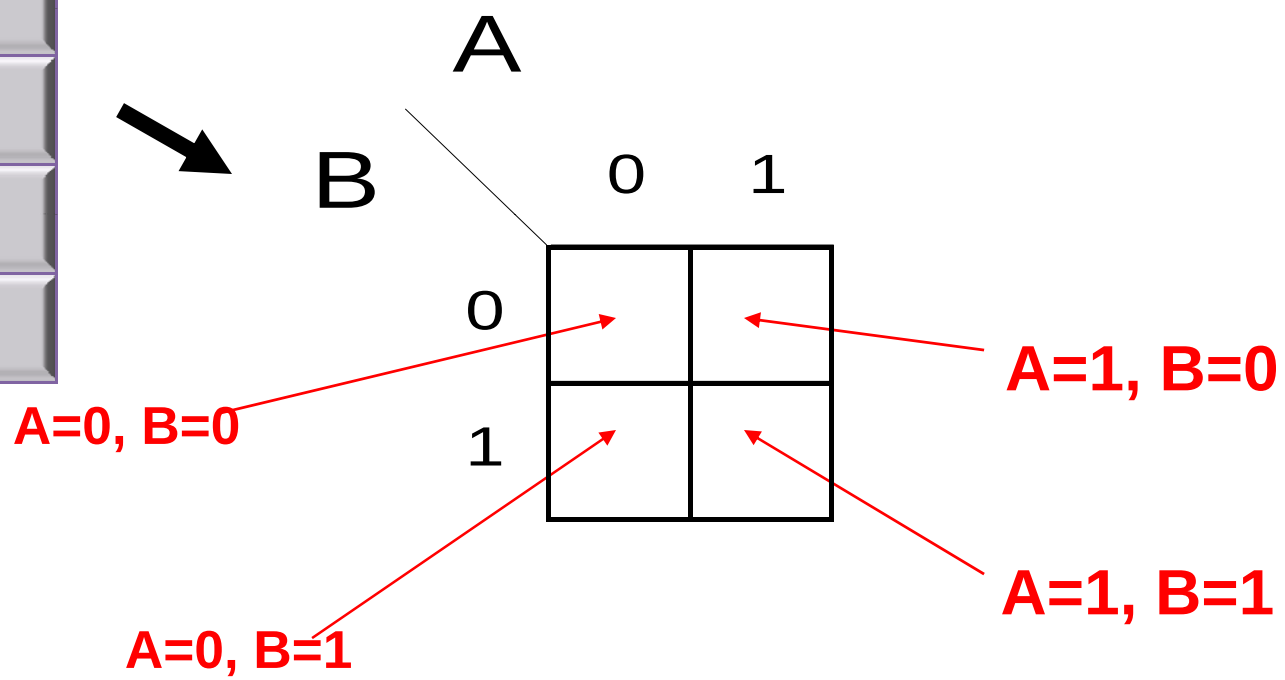
A'B', A'B, AB, AB'



All 4 Adjacent can be grouped

2-Variable Karnaugh Map

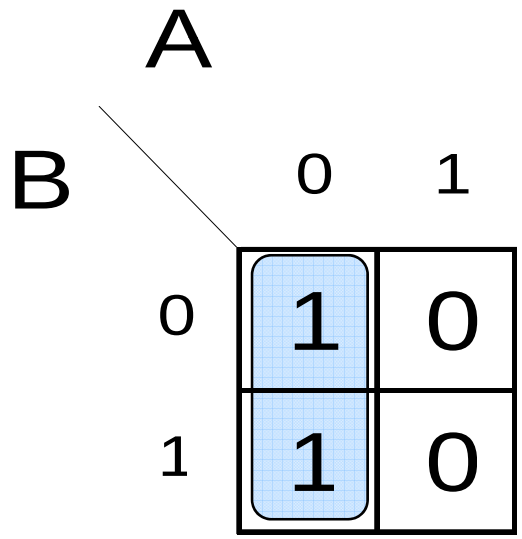
A	B	F
0	0	
0	1	
1	0	
1	1	



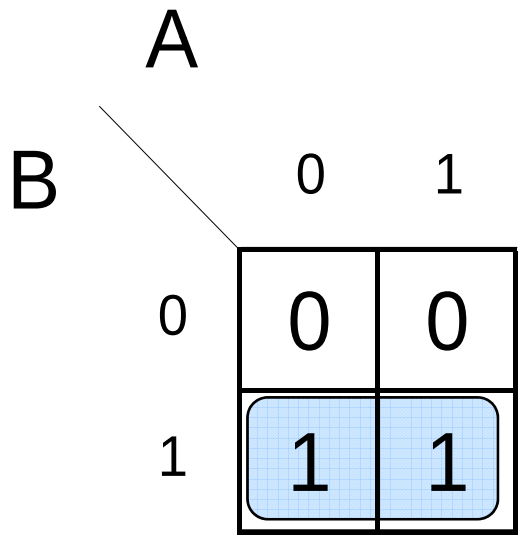
A different way to draw a truth table: by folding it

Karnaugh Map

- In a K-map, physical adjacency **does** imply gray code adjacency



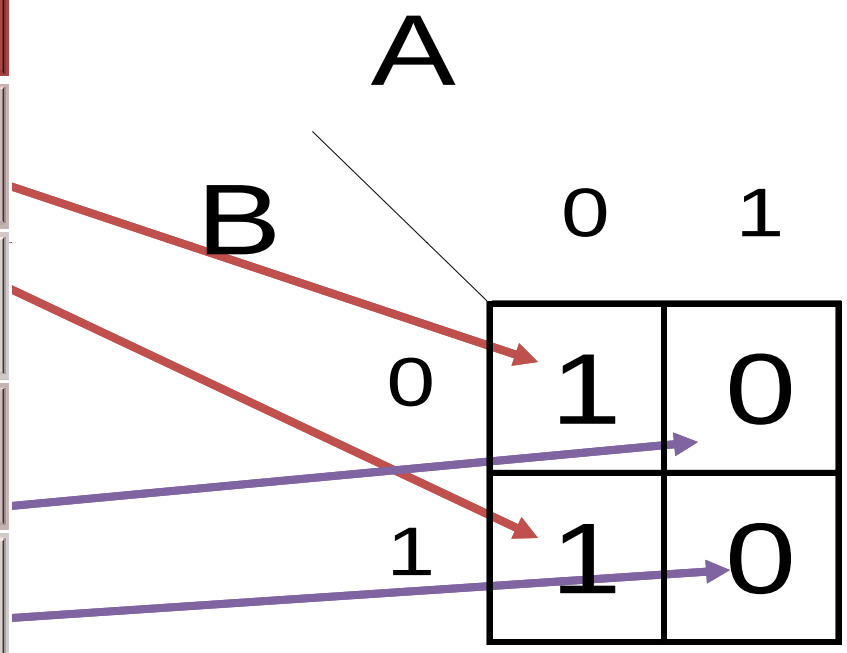
F =A'B' + A'B = A'



F = A'B + AB = B

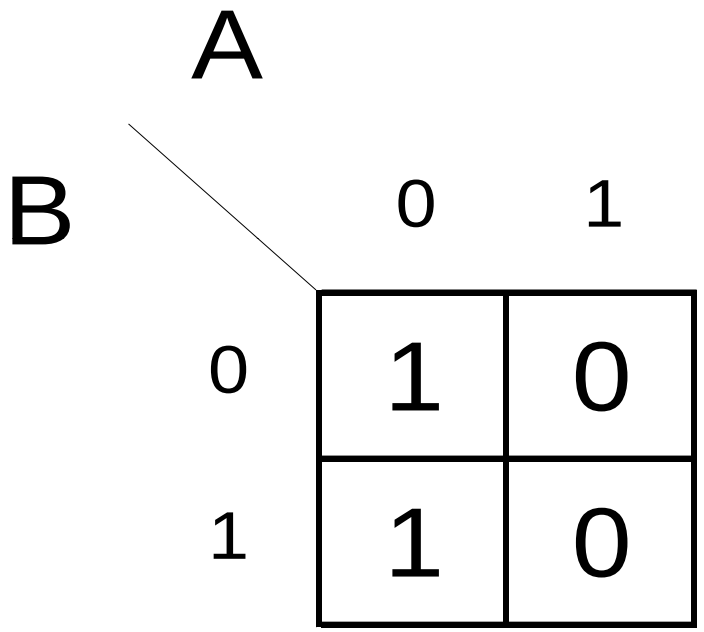
2-Variable Karnaugh Map

A	B	index	F
0	0	0	1
0	1	1	1
1	0	2	0
1	1	3	0



2-Variable Karnaugh Map

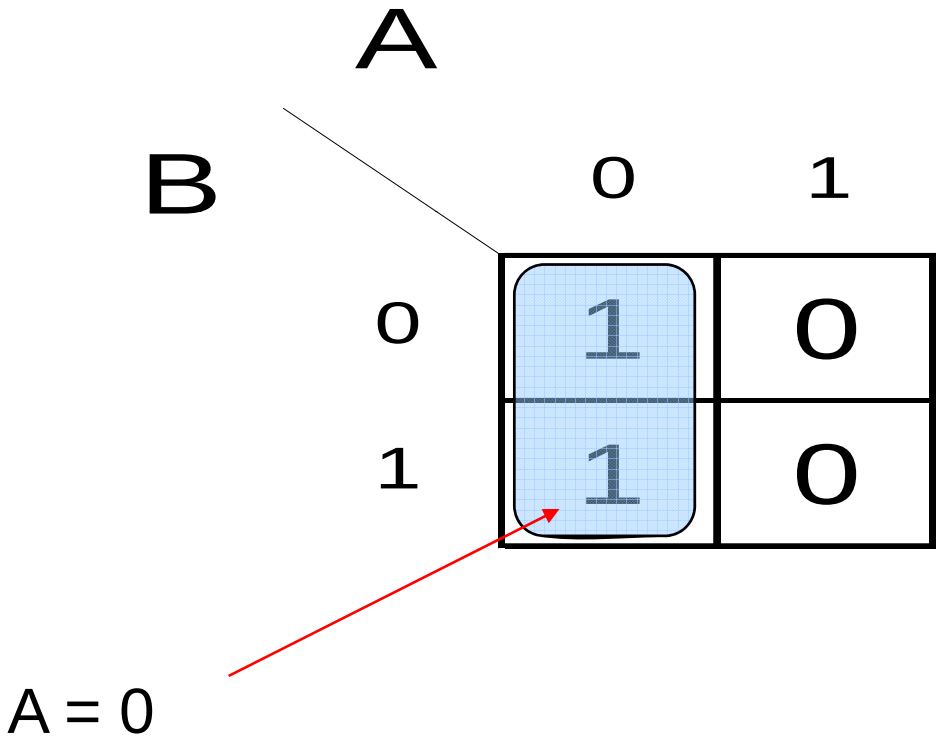
A	B	index	F
0	0	0	1
0	1	1	1
1	0	2	0
1	1	3	0



F = A'B' + A'B = A'

2-Variable K Map: Grouping

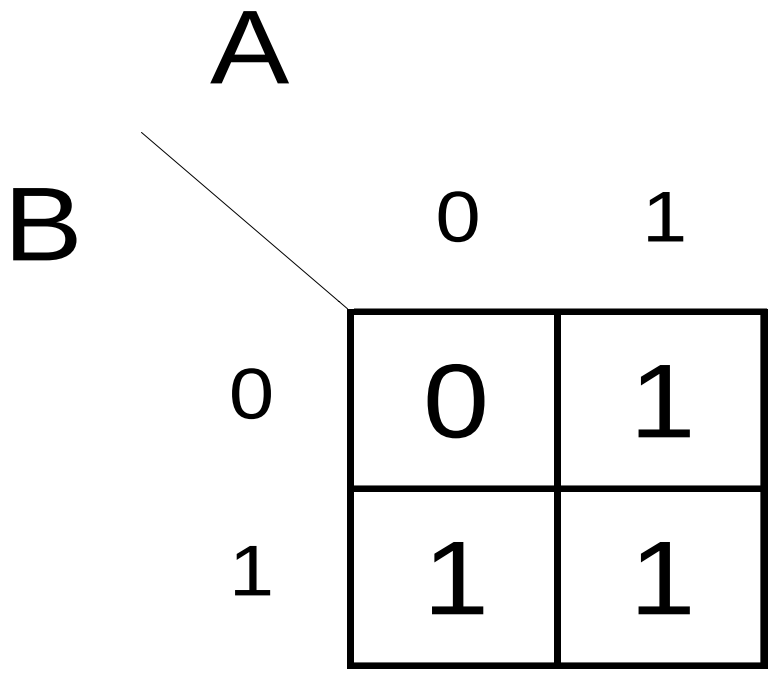
A	B	index	F
0	0	0	1
0	1	1	1
1	0	2	0
1	1	3	0



F = A’

Another Example

A	B	index	F
0	0	0	0
0	1	1	1
1	0	2	1
1	1	3	1



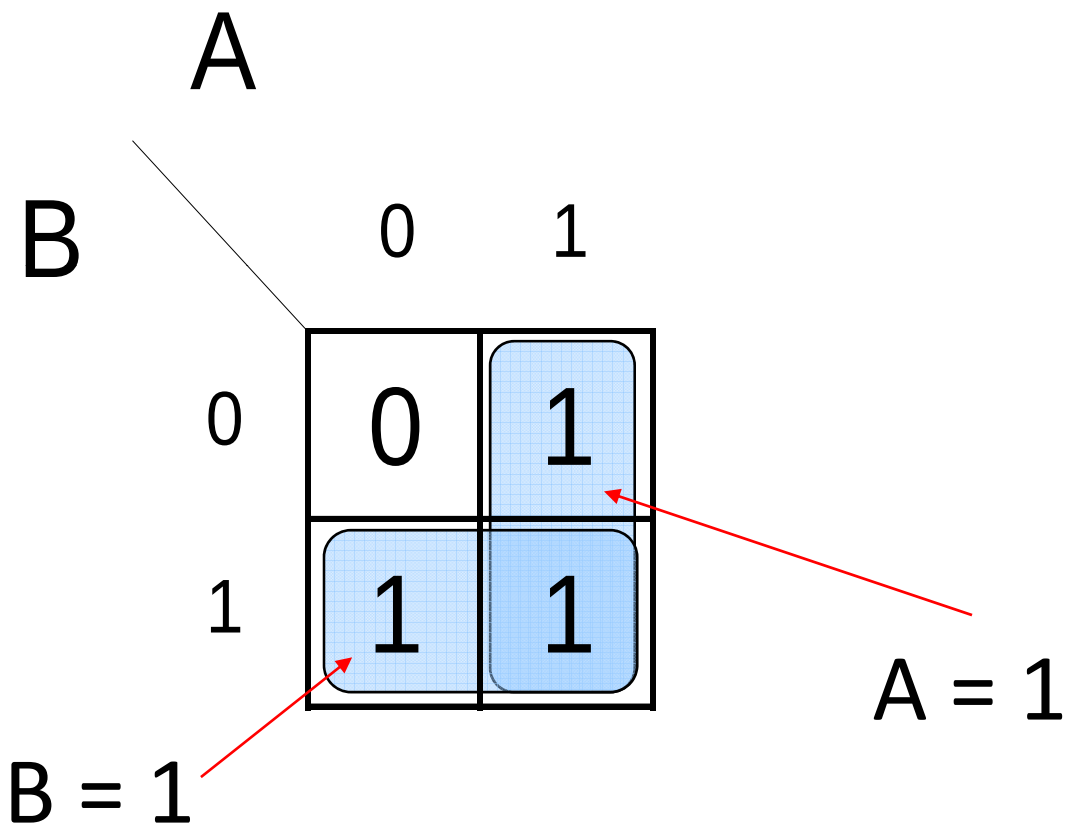
F = A'B + AB' + AB

= (A'B + AB) + (AB' + AB)

= A + B

Another Example

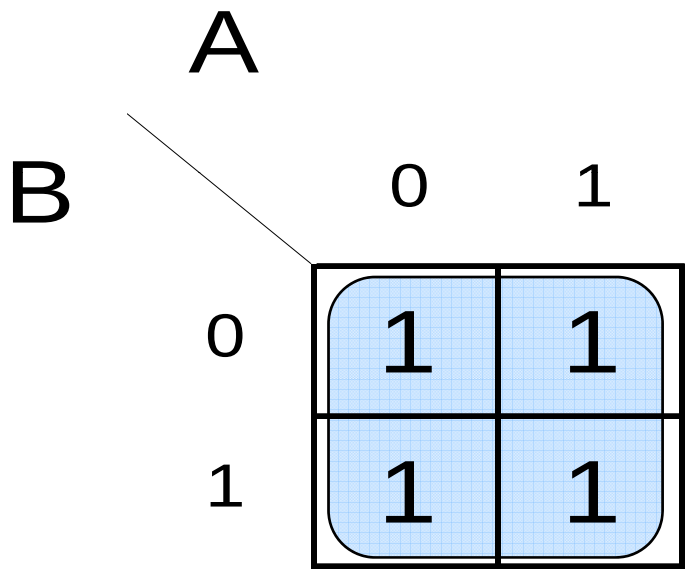
A	B	index	F
0	0	0	0
0	1	1	1
1	0	2	1
1	1	3	1



F = A + B

Yet Another Example

A	B	index	F
0	0	0	1
0	1	1	1
1	0	2	1
1	1	3	1



F = 1

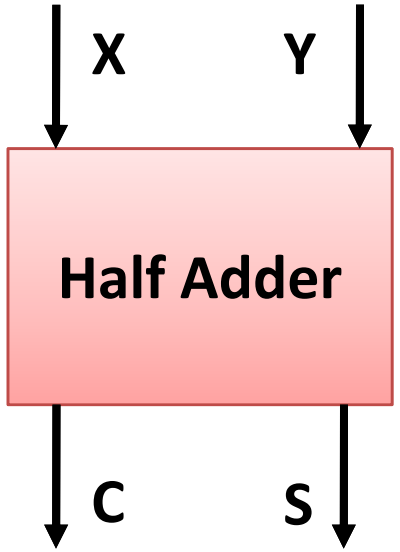
Groups of more than two 1's can be combined

HALF ADDER: One bit adder

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Y \ X	0	1
0	0	1
1	1	0

Y \ X	0	1
0	0	0
1	0	1



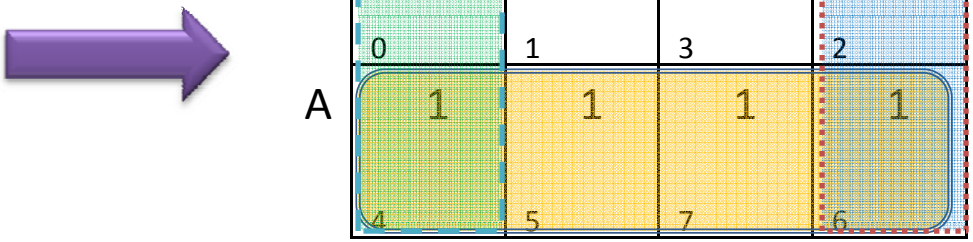
S = A'B + AB'
= A XOR **B**

C = AB

K-Map of three variable

A	B	C	index	F
0	0	0	0	1
0	0	1	1	0
0	1	0	2	1
0	1	1	3	0
1	0	0	4	1
1	0	1	5	1
1	1	0	6	1
1	1	1	7	1

F=∑m(0,2,4,5,6,7)



F= A + B'C' + BC'

Group of 4
m(4,5,7,6)

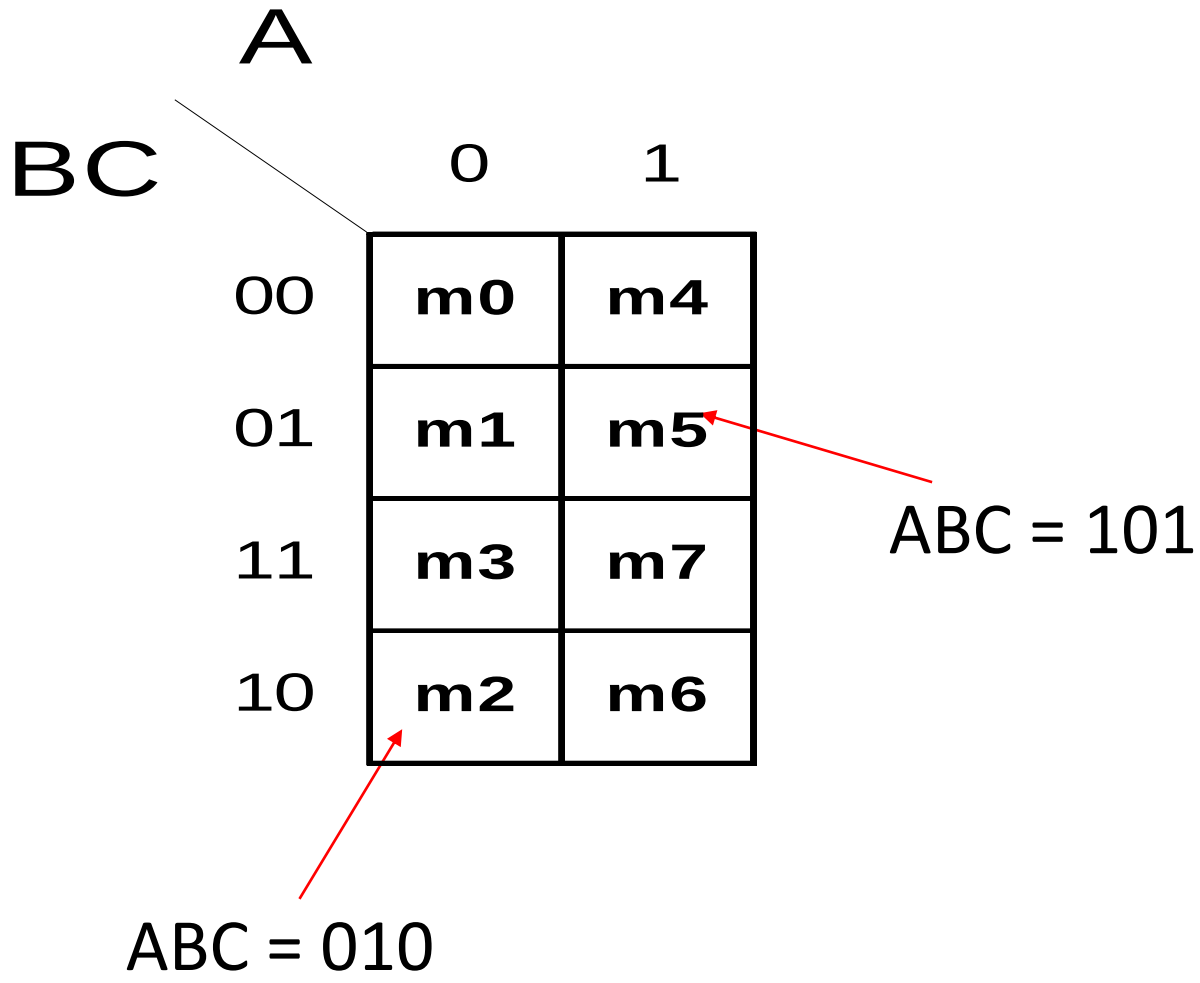
Group of 2
m(2,6)

Group of 2
m(0,4)

3-Variable Karnaugh Map Showing Minterm Locations

Note the order of the B C variables:

0 0
0 1
1 1
1 0



Thanks

KMap-Logic Minimization Contd..

Outline

- Karnough map simplification
  - 2, 3 variable KMap
  - 4 variable karnaugh map
  - Don’t care condition
  - Algorithm for better grouping
- Karnaugh map with >= 5 variable

Karnaugh Map Method

- The truth table values are placed in the K map.
- **Adjacent K map square differ in only one variable both horizontally and vertically.**
- **The pattern from top to bottom and left to right must be in the form**
- A SOP expression can be obtained by ORing all squares that contain a 1.

A'B', A'B, AB, AB'

00, 01, 11, 01

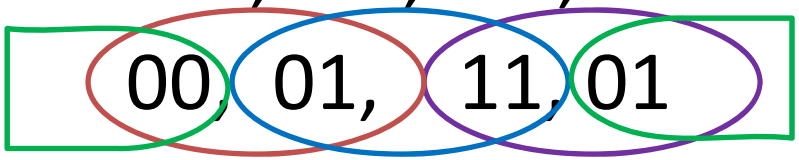
Filling of Karnaugh Map

Why not: A'B', A'B, AB', AB

Only two adjacent can be grouped

Group Reduce a variable: AB'+AB=A(B'+B)=A

A'B', A'B, AB, AB'

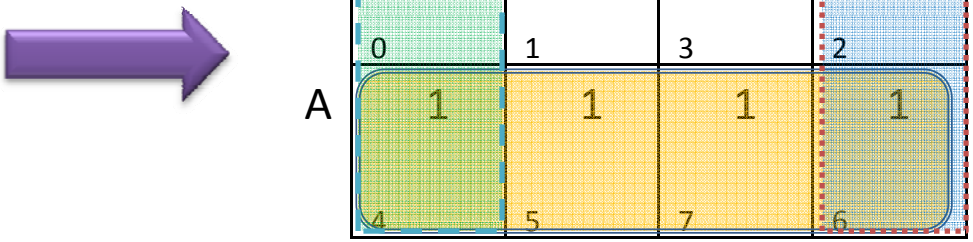


All 4 Adjacent can be grouped

K-Map of three variable

A	B	C	index	F
0	0	0	0	1
0	0	1	1	0
0	1	0	2	1
0	1	1	3	0
1	0	0	4	1
1	0	1	5	1
1	1	0	6	1
1	1	1	7	1

F=∑m(0,2,4,5,6,7)



F= A + B'C' + BC'

Group of 4 m(4,5,7,6)

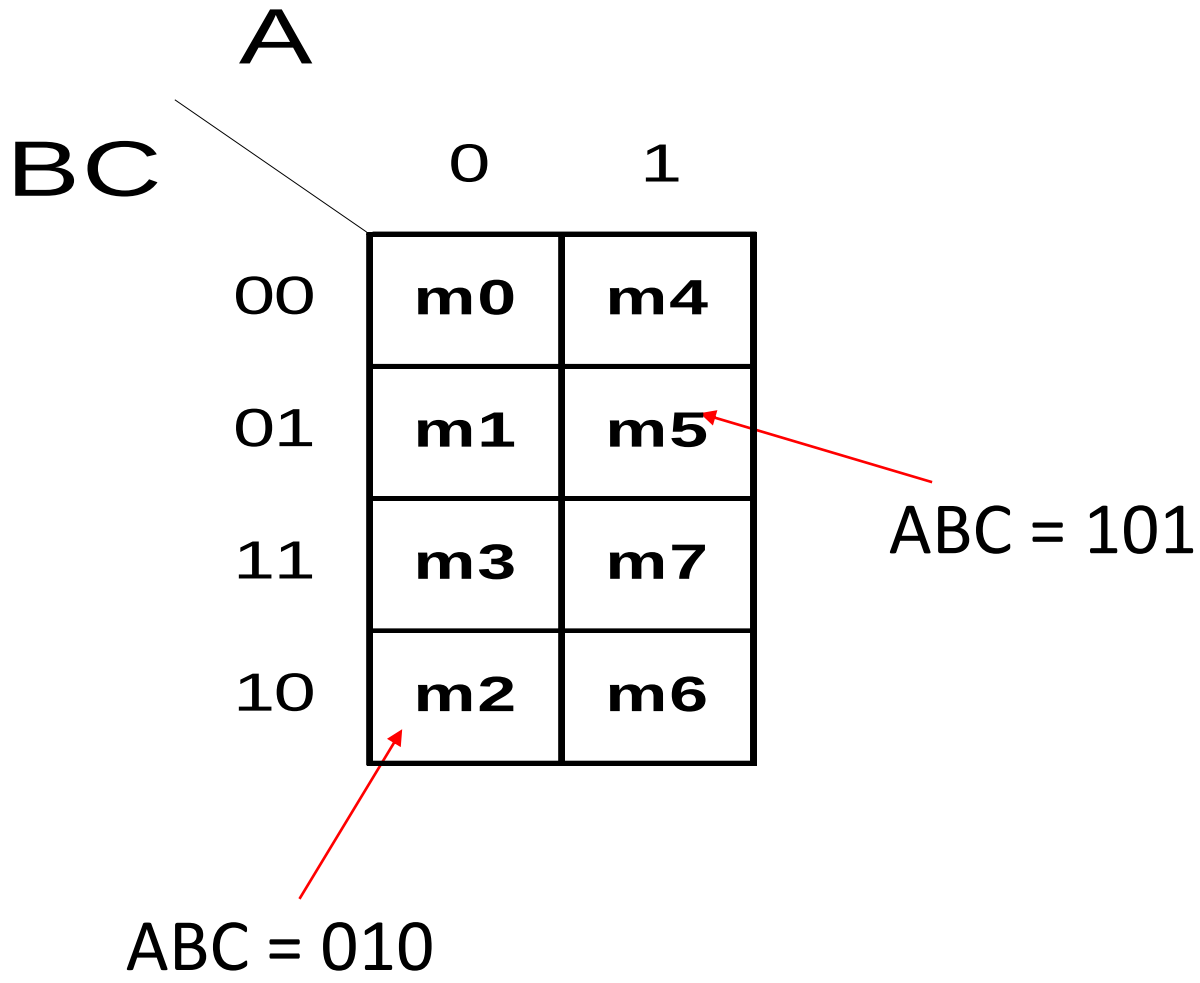
Group of 2 m(2,6)

Group of 2 m(0,4)

3-Variable Karnaugh Map Showing Minterm Locations

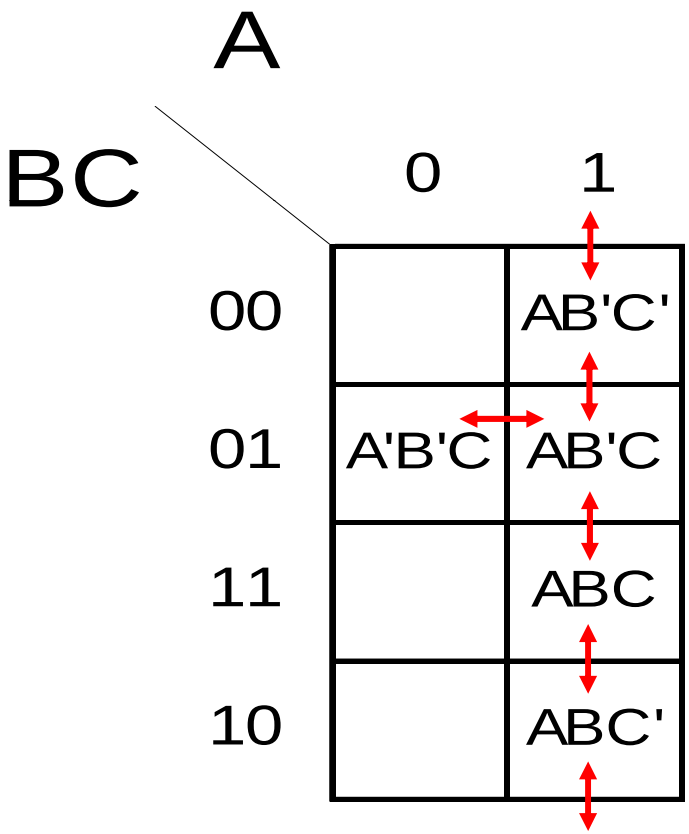
Note the order of the B C variables:

0 0
0 1
1 1
1 0



Adjacencies

- Adjacent squares differ by exactly one variable



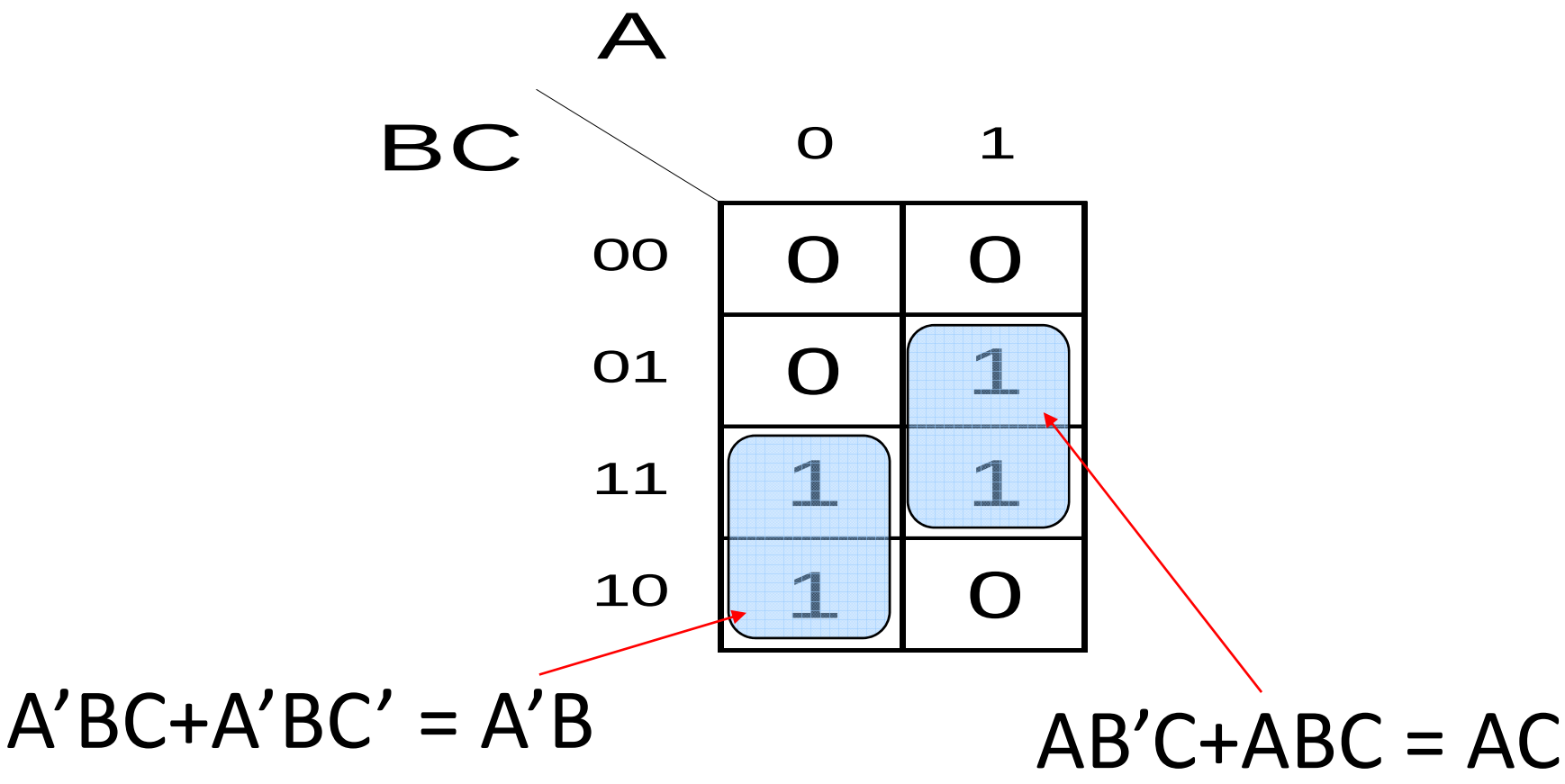
There is wrap-around:
top and bottom rows are adjacent

Truth Table to Karnaugh Map

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

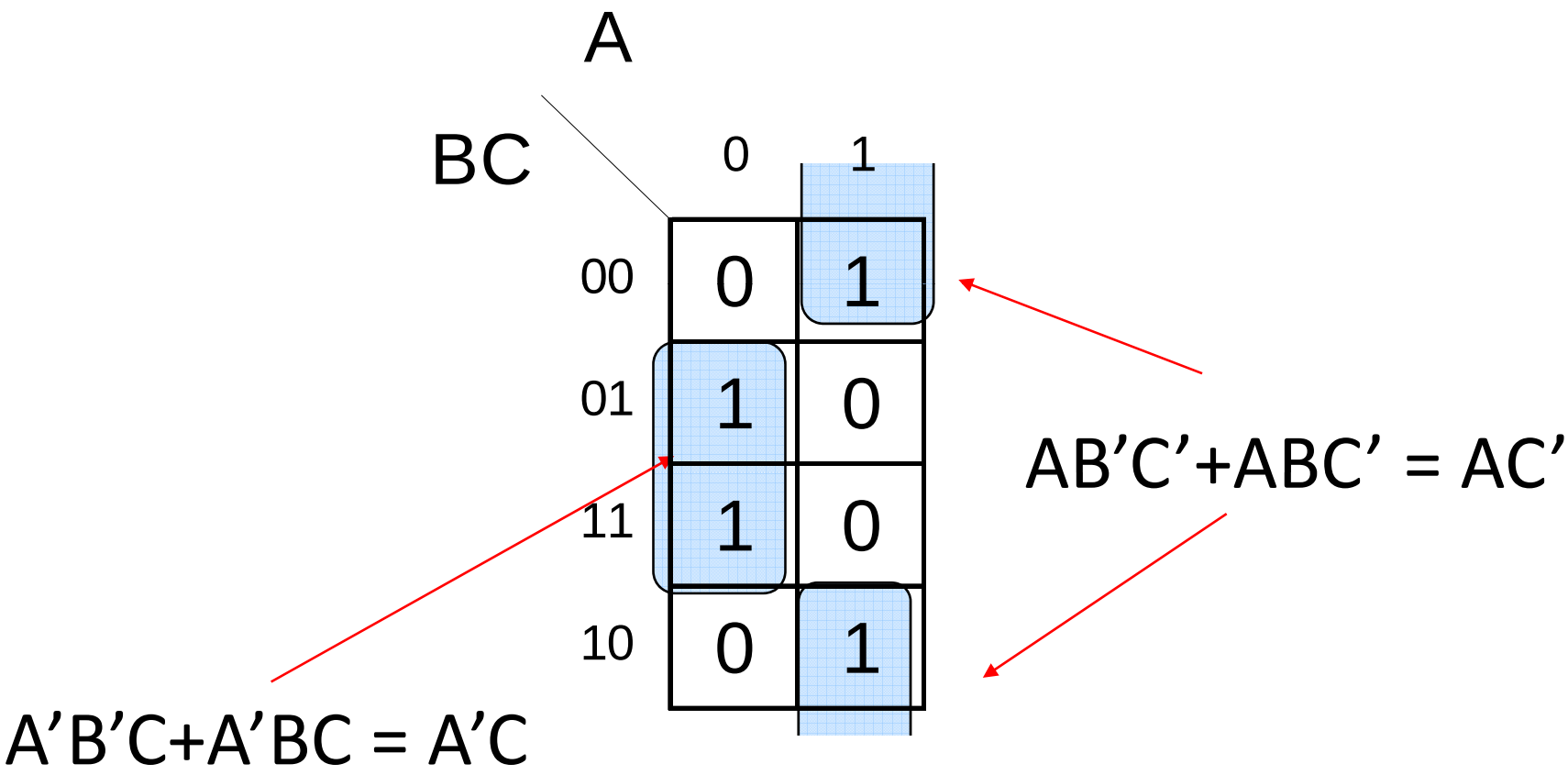
BC \ A	0	1
00	0	0
01	0	1
11	1	1
10	1	0

Minimization Example



F = A'B + AC

Another Example



F = A'C + AC' = A ⊕ C

Minterm Expansion to K-Map

$$F = \sum m(1, 3, 4, 6)$$

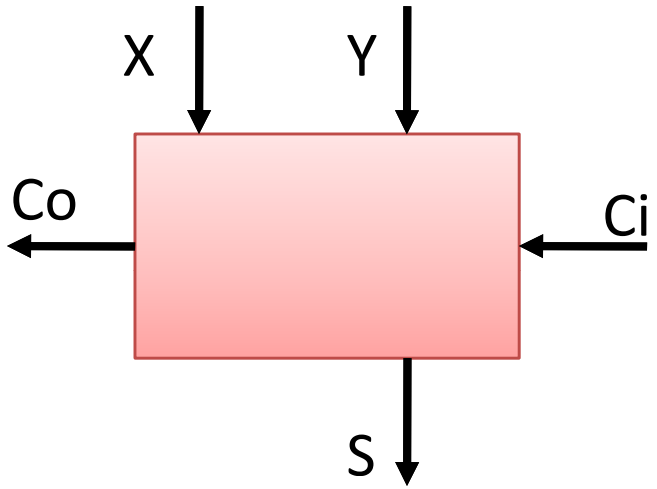
BC \ A	0	1
00	m0	m4
01	m1	m5
11	m3	m7
10	m2	m6

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

Minterms are the 1's, everything else is 0

Full Adder Example: Minterms

Ci	X	Y	index	S	Co
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	2	1	0
0	1	1	3	0	1
1	0	0	4	1	0
1	0	1	5	0	1
1	1	0	6	0	1
1	1	1	7	1	1



$$S = \sum m(1, 2, 4, 7)$$

$$Co = \sum m(3, 5, 6, 7)$$

Full Adder Output

$S = \sum m(1, 2, 4, 7)$

$Co = \sum m(3, 5, 6, 7)$

XY \ Ci	0	1
00	0	1
01	1	0
11	0	1
10	1	0

XY \ Ci	0	1
00	0	0
01	0	1
11	1	1
10	0	1

S=Ci'X'Y+Ci'XY'+CiX'Y'+CiXY Co=XY+CiX+CiY

Maxterm Expansion to KMap

$$F = \prod M(0, 2, 5, 7)$$

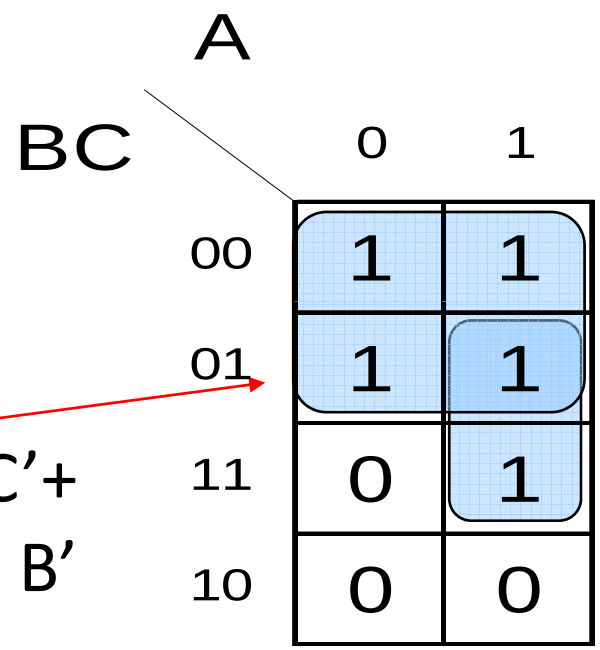
BC \ A	0	1
00	M0	M4
01	M1	M5
11	M3	M7
10	M2	M6

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

Maxterms are the 0's, everything else is 1

Yet Another Example

2^n 1's can be circled
at a time
1, 2, 4, 8, … OK
3 not OK



A'B'C'+AB'C'+
A'B'C+AB'C = B'

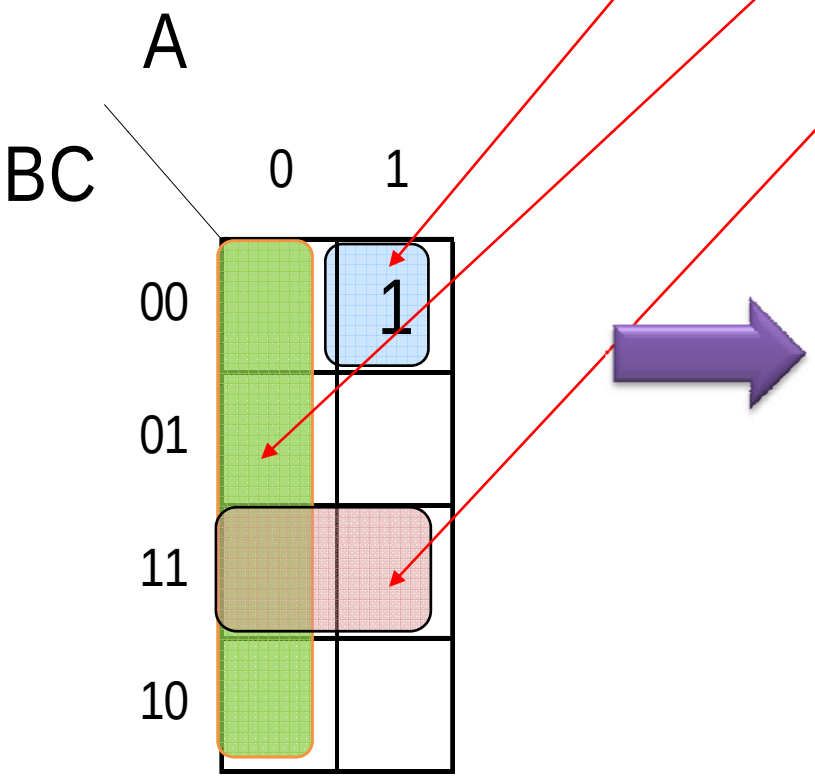
AB'C+ABC = AC

F = B' + AC

**The larger the group of 1's
the simpler the resulting product term**

Boolean Algebra to Karnaugh Map

Plot: ab'c' + a' + bc

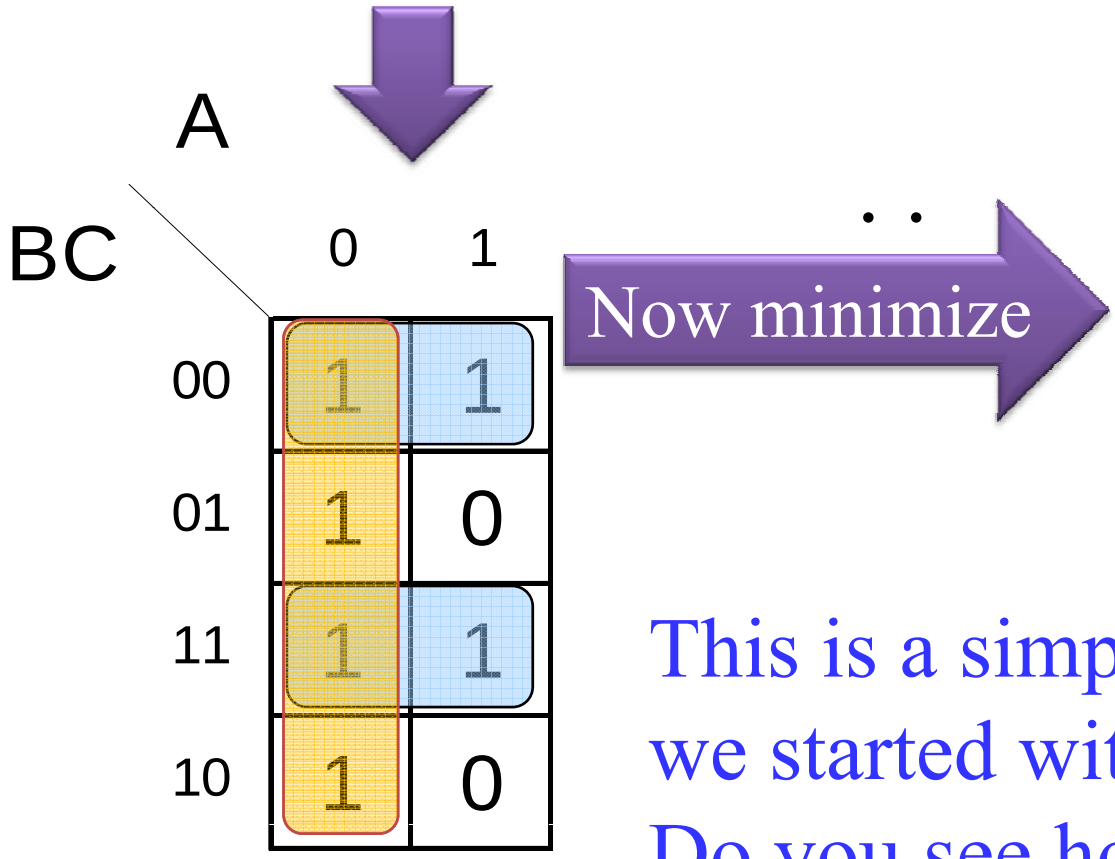


BC \ A	0	1
00	1	1
01	1	0
11	1	1
10	1	0

Remaining spaces are 0

Boolean Algebra to Karnaugh Map

Plot: ab'c' + bc + a'



Now minimize

B'C' + BC + A'

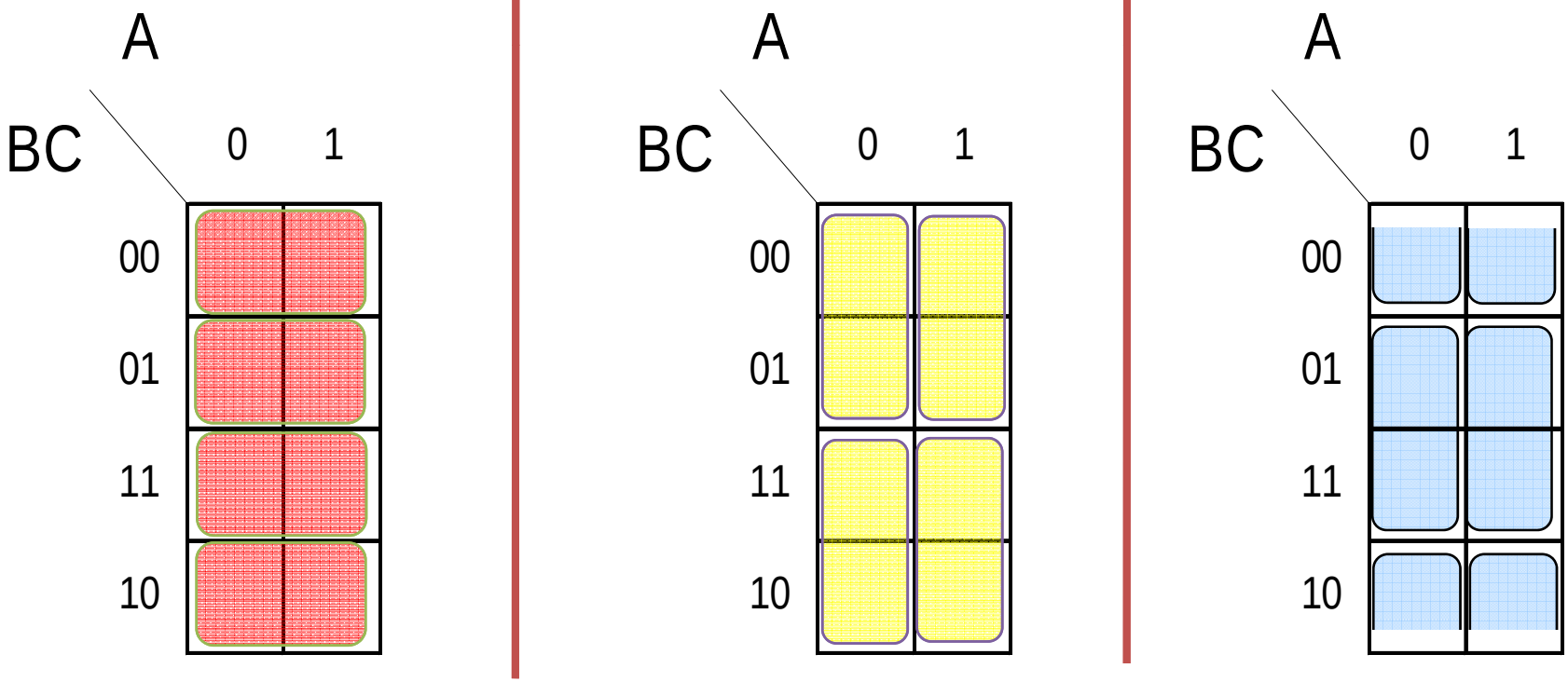
This is a simpler equation than we started with.

Do you see how we obtained it?

Mapping Sum of Product Terms

The 3-variable map has 12 possible groups of 2 spaces

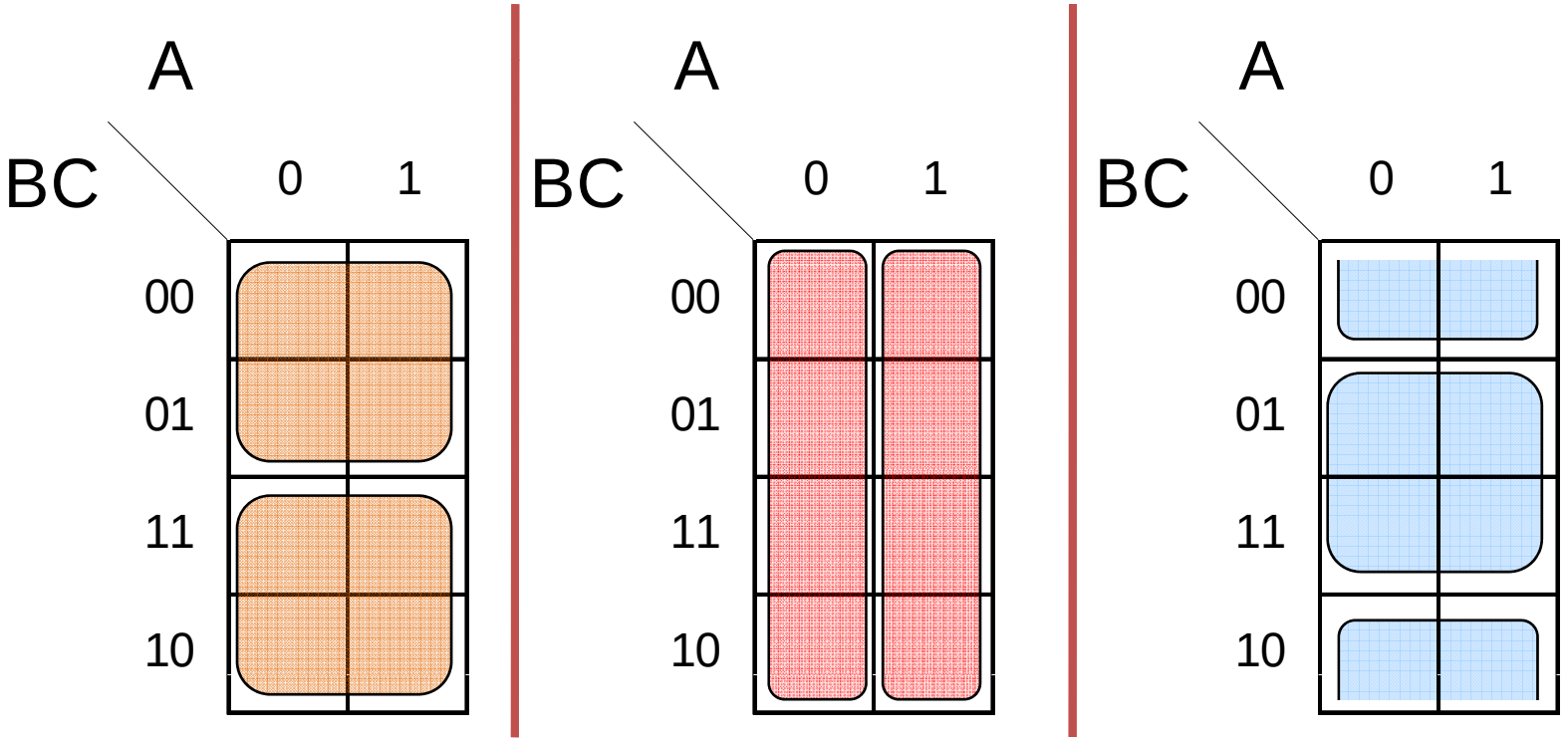
These become terms with 2 literals



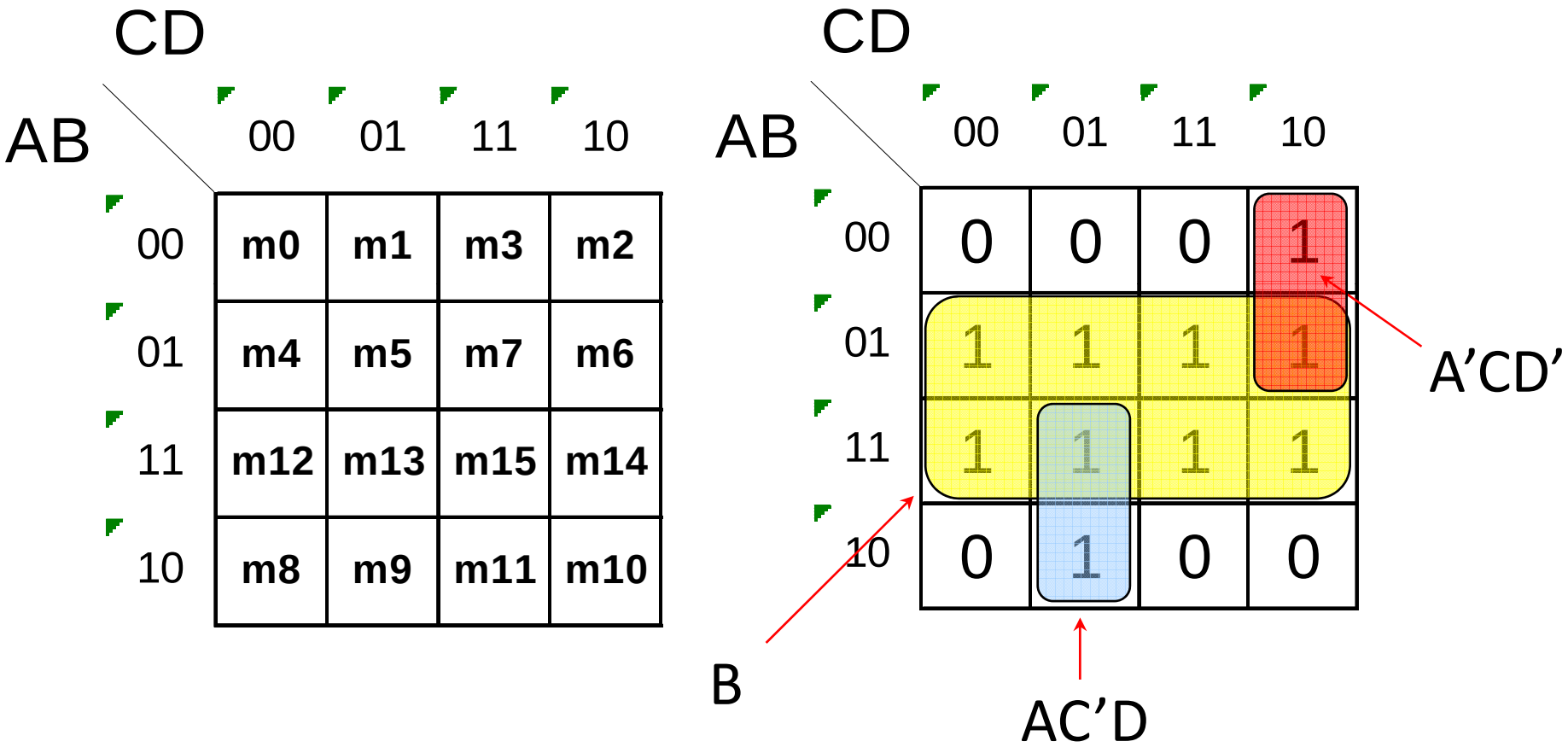
Mapping Sum of Product Terms

The 3-variable map has 6 possible groups of 4 spaces

These become terms with 1 literal



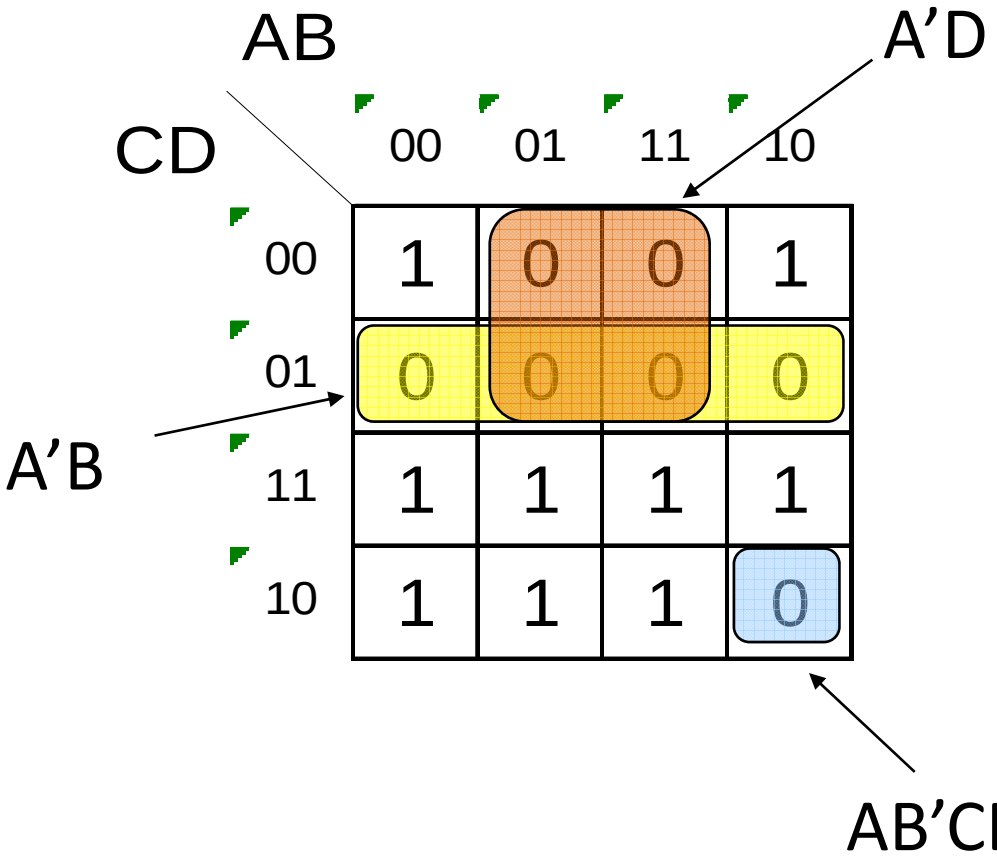
4-Variable Karnaugh Map



$$F = AC'D + A'CD' + B$$

Note the row and column orderings. Required for adjacency

Find a POS Solution



F' = A'D + A'B + AB'CD'

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

Find solutions to groups of 0's to find F'

Invert to get F then use DeMorgan's

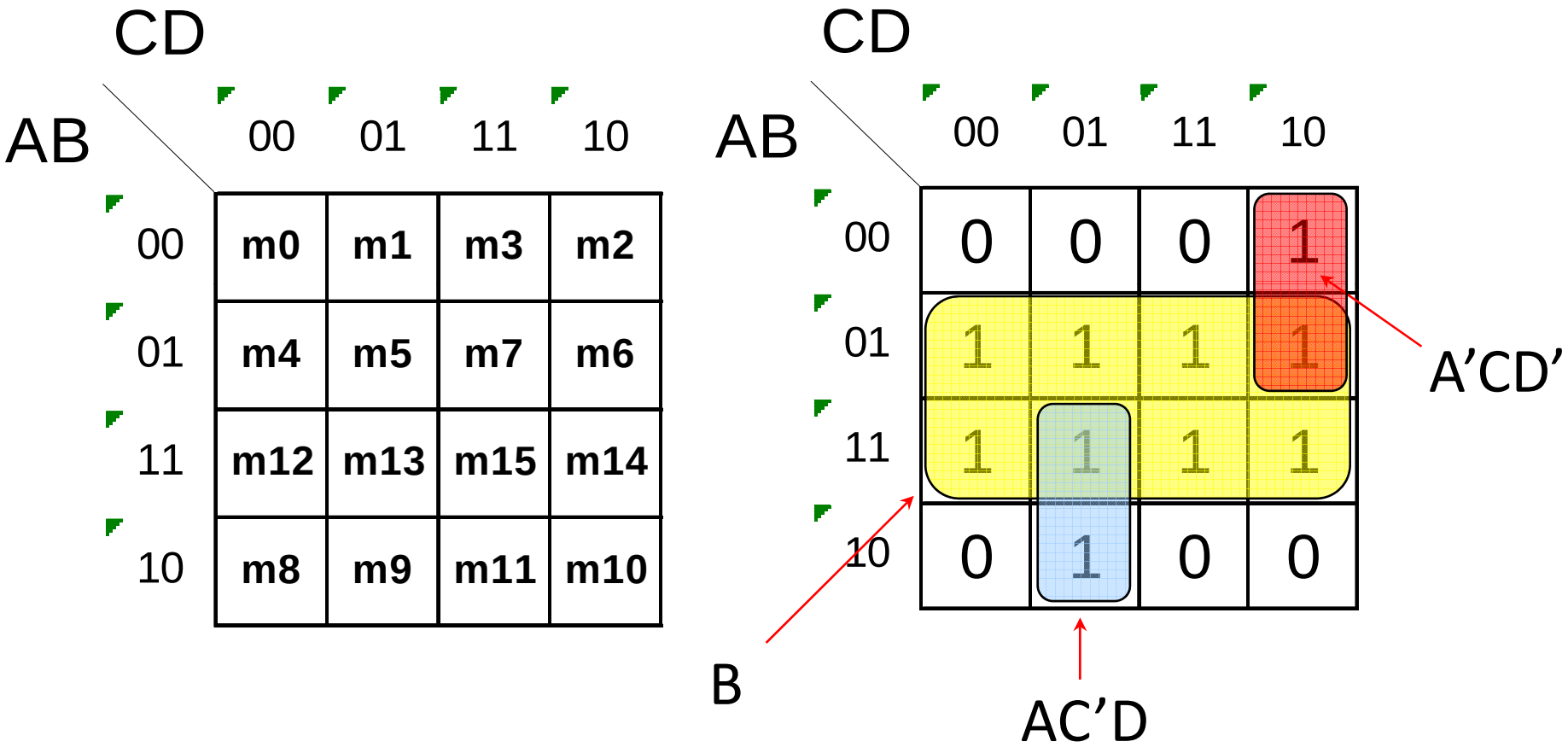
Thanks

KMap-Logic Minimization Contd..

Outline

- Karnough map simplification
  - 4 variable karnaugh map
  - Don’t care condition
  - Algorithm for better grouping
- Karnaugh map with >= 5 variable

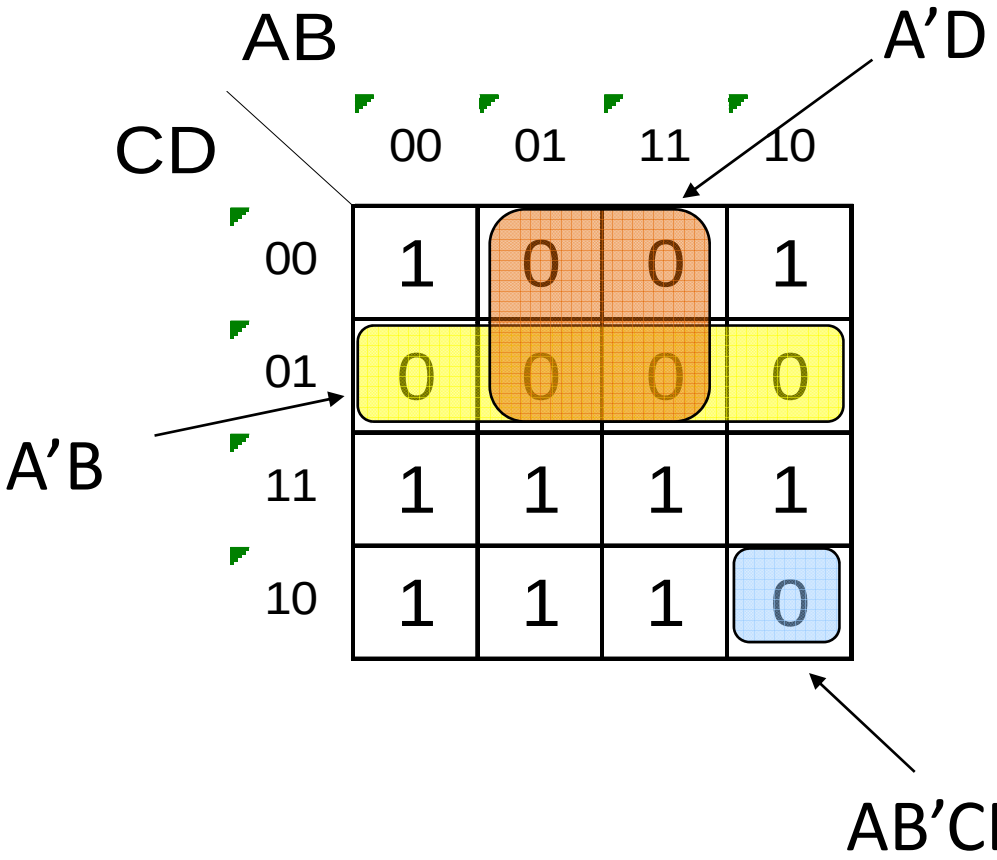
4-Variable Karnaugh Map



$$F = AC'D + A'CD' + B$$

Note the row and column orderings. Required for adjacency

Find a POS Solution



F' = A'D + A'B + AB'CD'

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

Find solutions to groups of 0's to find F'

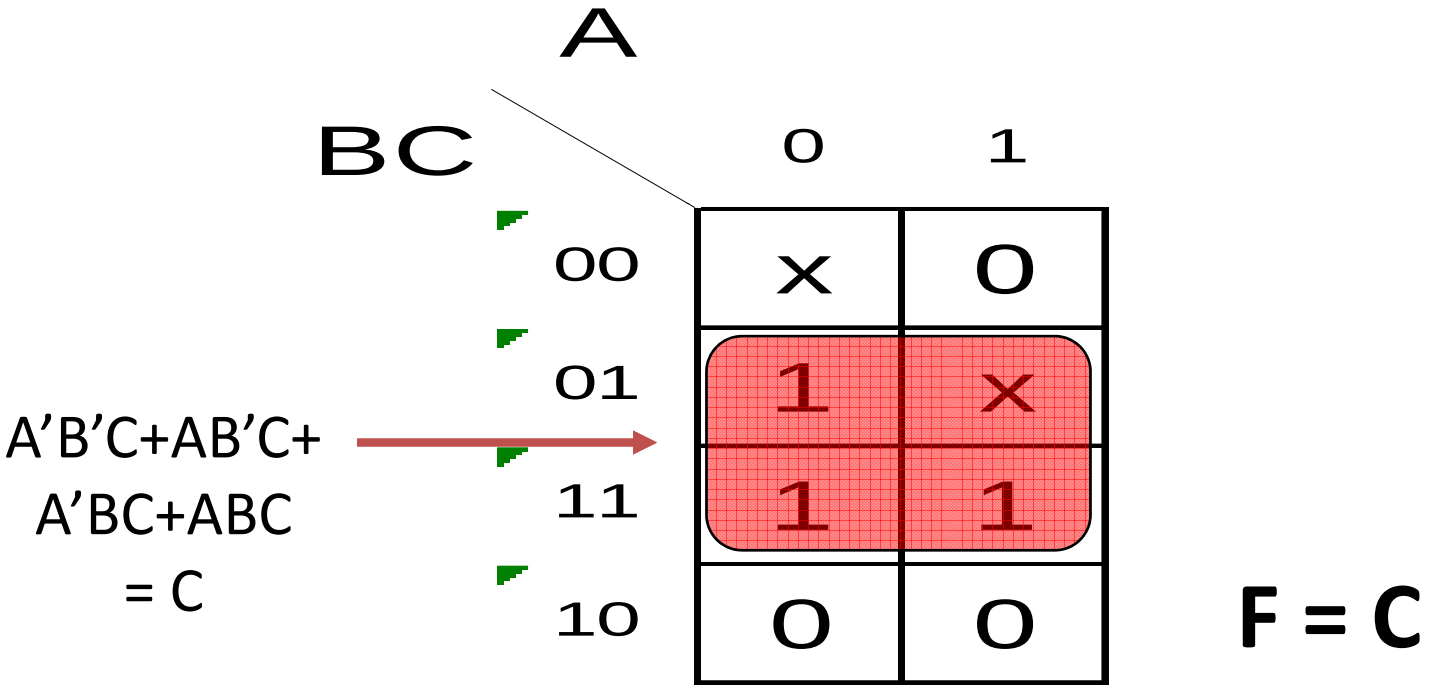
Invert to get F then use DeMorgan's

Don’t Care

- A **don't-care term** is an input to a function that the designer does not care about
- Because that input would never happen
- Example:
  - BCD number (0-9, A-F) are 4 bits, don’t care about input A-F
  - Suppose a system have 5 type of input
    - Unfortunately we can’t have 2 input line
    - Make 3 input line and last 3 sequence as don’t care
    - S0, S1, S2,S3,S4, X,X,X == > 000, 001….,111

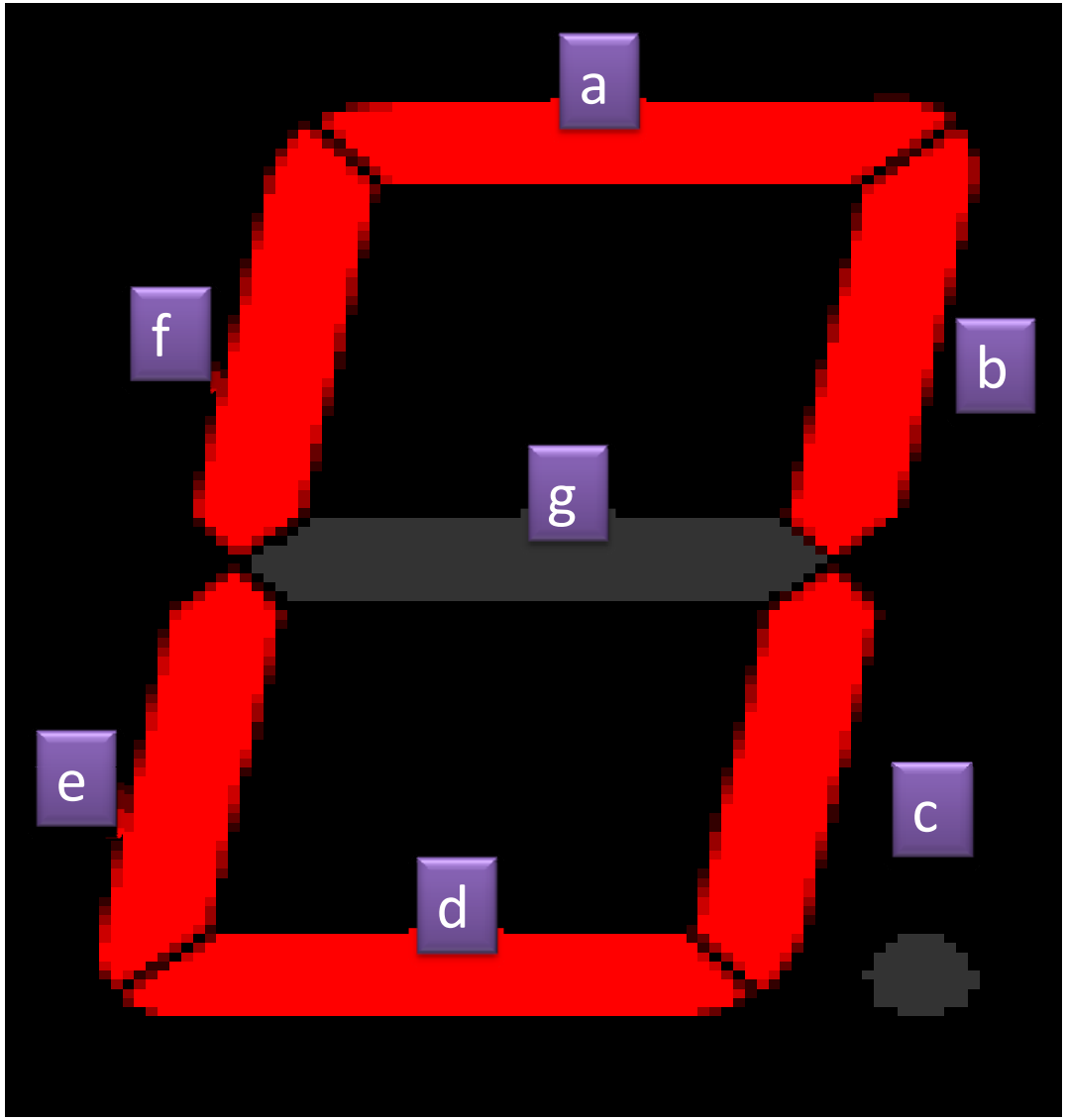
Dealing With Don't Cares

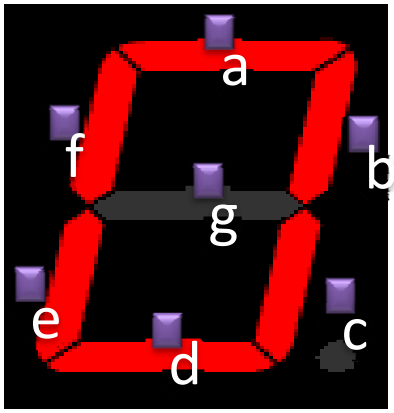
$$F = \Sigma m(1, 3, 7) + \Sigma d(0, 5)$$



Circle the x's that help get bigger groups of 1's
Don't circle the x's that don't

7 Segment Display



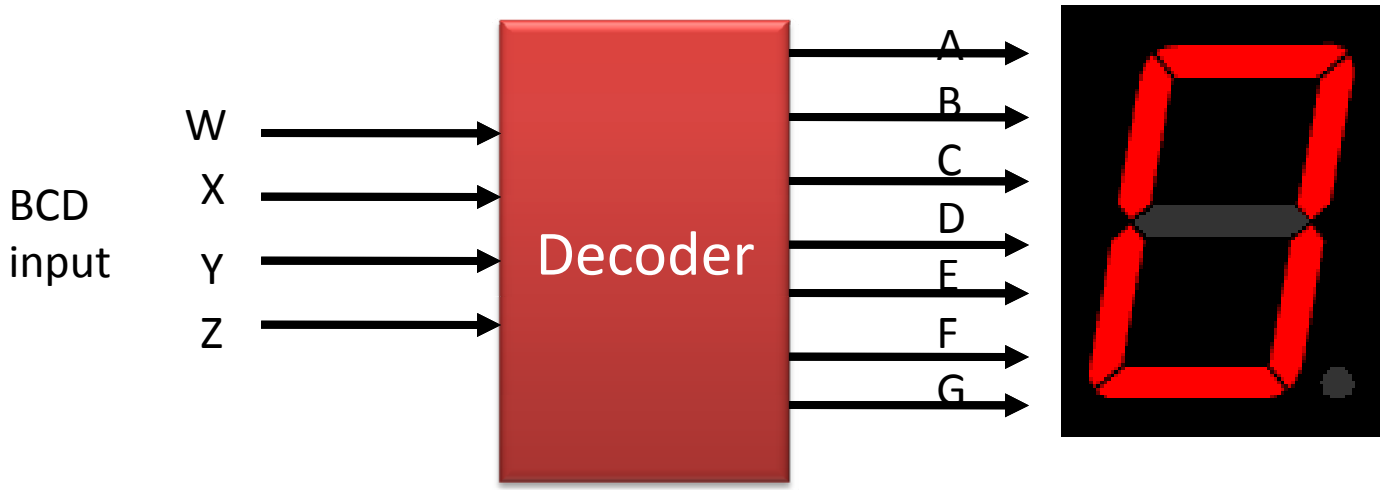


Activation of LEDs

- 0 : a,b,c,d,e,f
- 1: b,c
- 2:a,b,g,e,d
- 3:a,b,g,c,d
- 4:f,g,b,c
- 5:a,f,g,c,d
- 6:a,f,g,c,d,e
- 7:a,b,c
- 8:a,b,c,d,e,f,g
- 9:a,b,c,d,f,g

BCD to 7 Segment Display

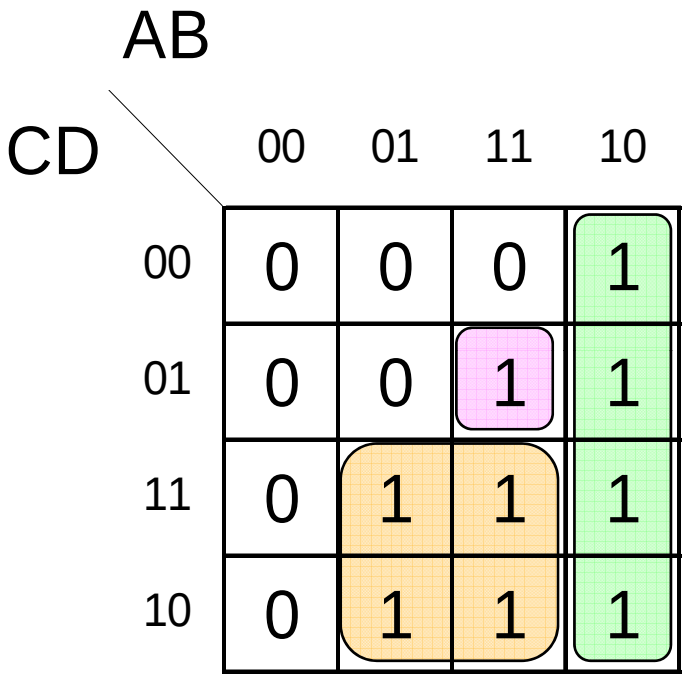
- BCD are 4 bit
- Design a decoder to drive 7 segment LED



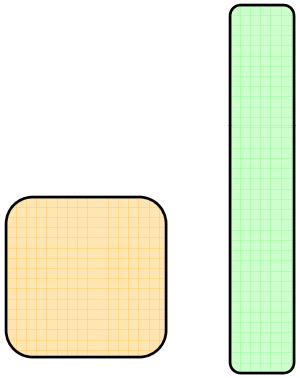
Prime Implicants

- A group of one or more 1's which are adjacent and can be combined on a Karnaugh Map is called an implicant.
- The *biggest* group of 1's which can be circled to cover a given 1 is called a prime implicant.
  - They are the only implicants we care about.

Prime Implicants



Prime Implicants



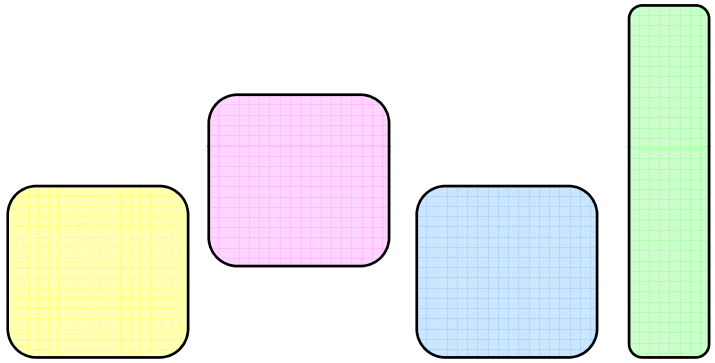
Non-prime Implicants

Are there any additional prime implicants in the map that are not shown above?

All The Prime Implicants

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

Prime Implicants



When looking for a minimal solution –
only circle prime implicants…
A minimal solution will ***never*** contain
non-prime implicants

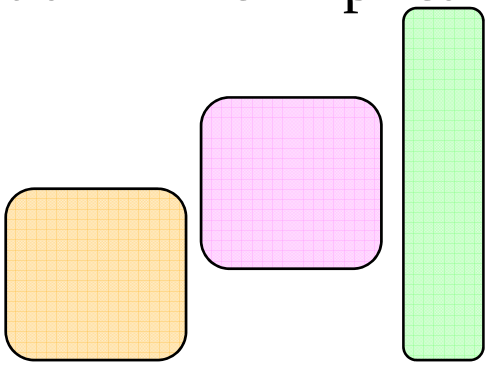
Essential Prime Implicants

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

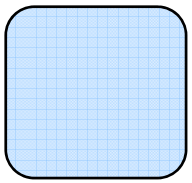
Not all prime implicants are required…

A prime implicant which is the only cover of some 1 is ***essential*** – a minimal solution requires it.

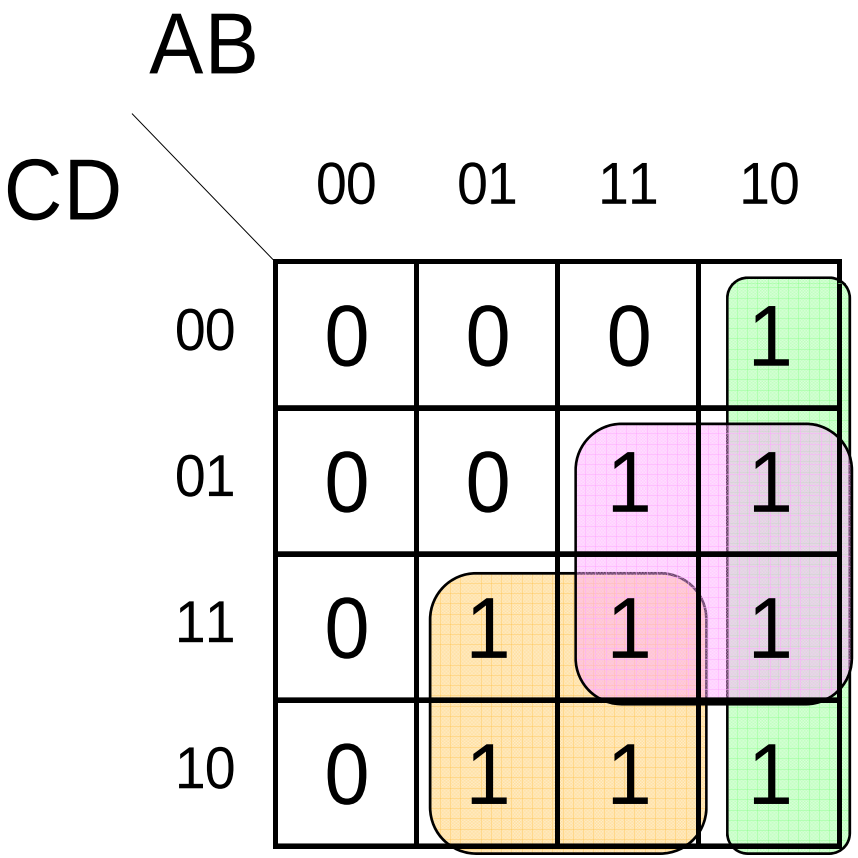
Essential Prime Implicants



Non-essential Prime Implicants



A Minimal Solution Example



$F = AB' + BC + AD$

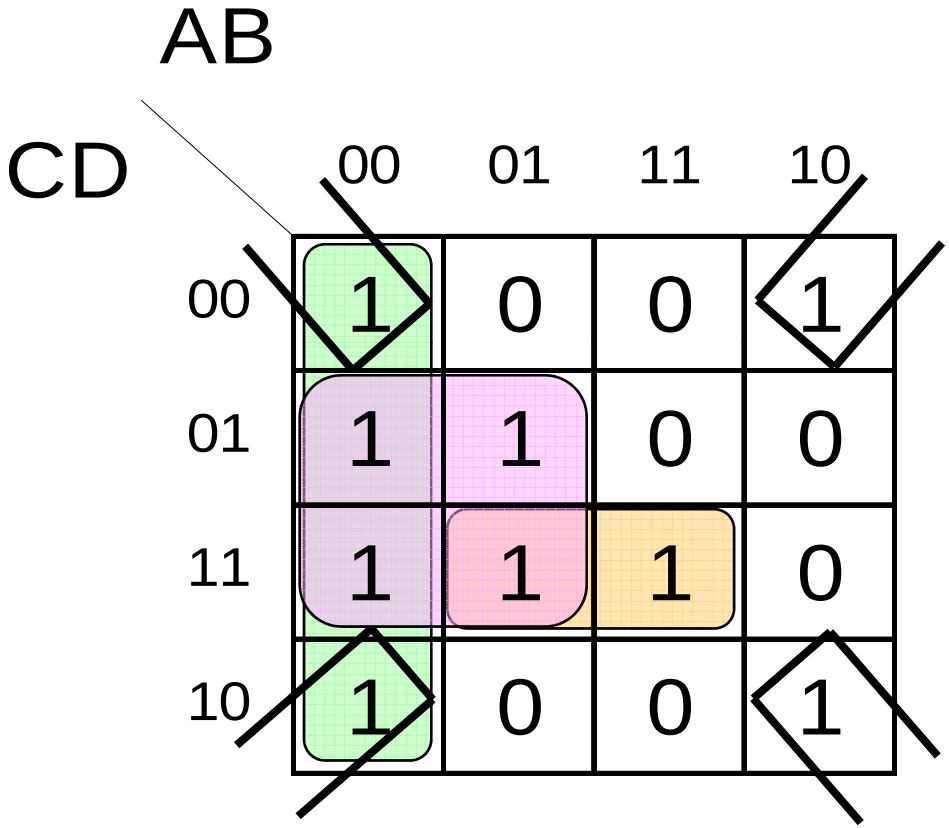
Minimum

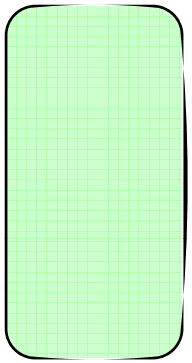
Not required…

Another Example

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

Another Example





A'B' is not required…
Every one one of its locations is covered by multiple implicants

After choosing essentials, everything is covered…

F = A'D + BCD + B'D'

Minimum

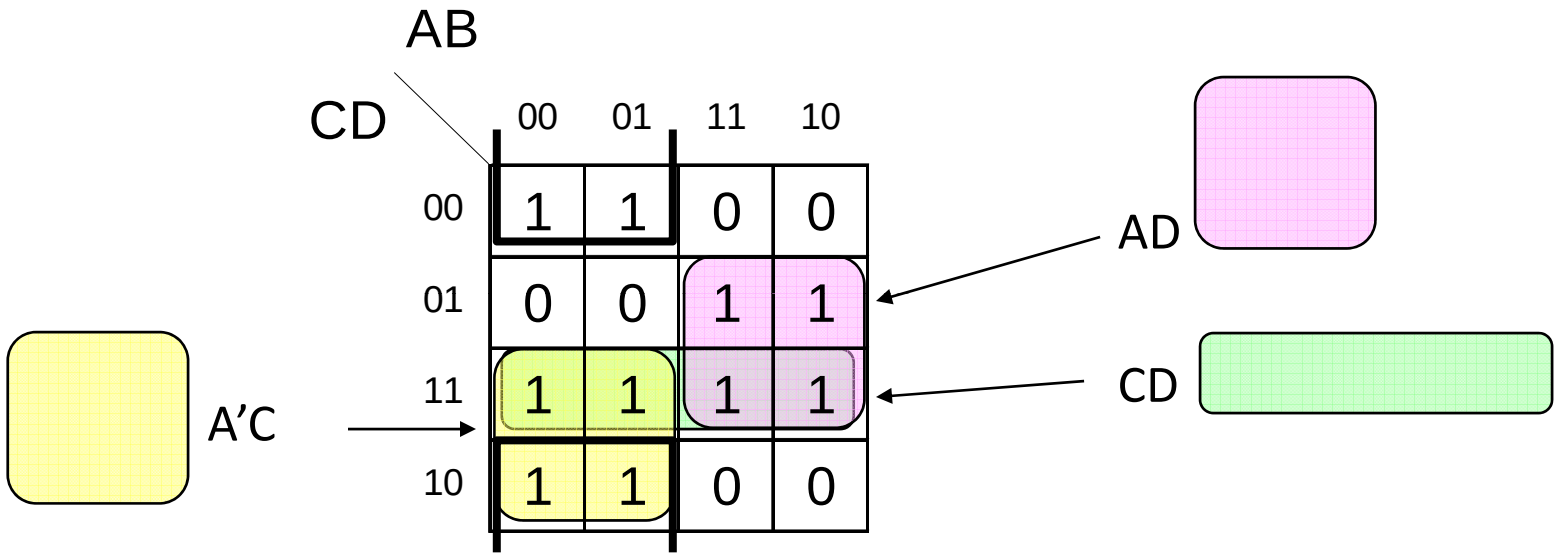
Finding the Minimum Sum of Products

1. Find each essential prime implicant and include it in the solution.
2. Determine if any minterms are not yet covered.
3. Find the minimal # of remaining prime implicants which finish the cover.

Yet Another Example
(Use of non-essential primes)

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

Yet Another Example
(Use of non-essential primes)



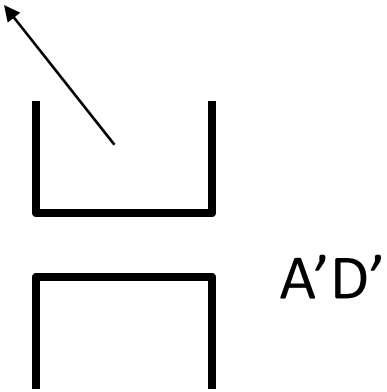
Essentials: A'D' and AD

Non-essentials: A'C and CD

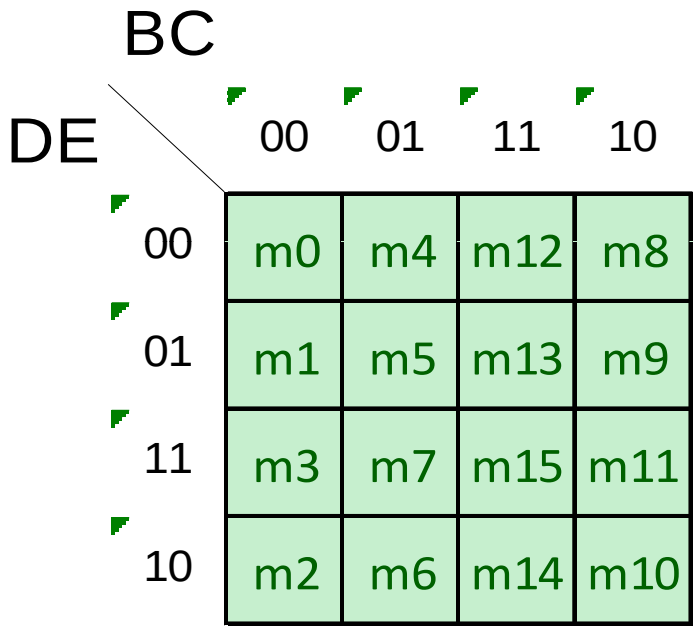
Solution: A'D' + AD + A'C

or

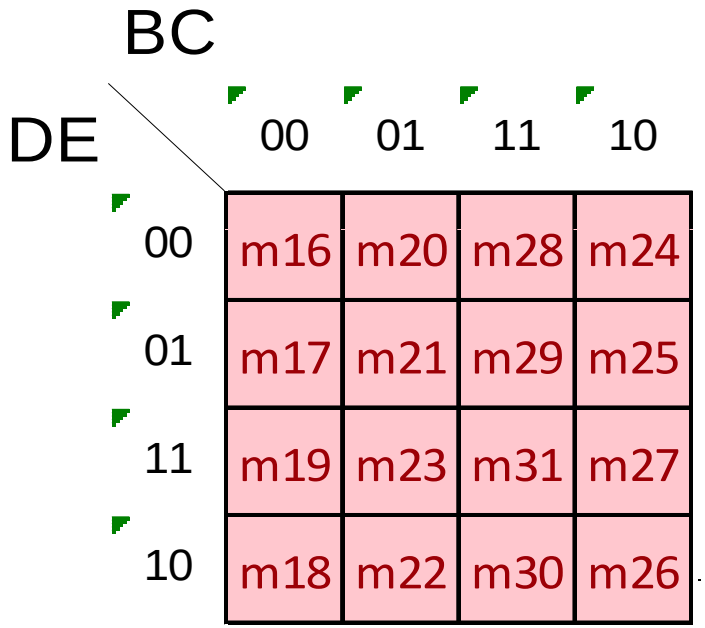
A'D' + AD + CD



5-Variable Karnaugh Map



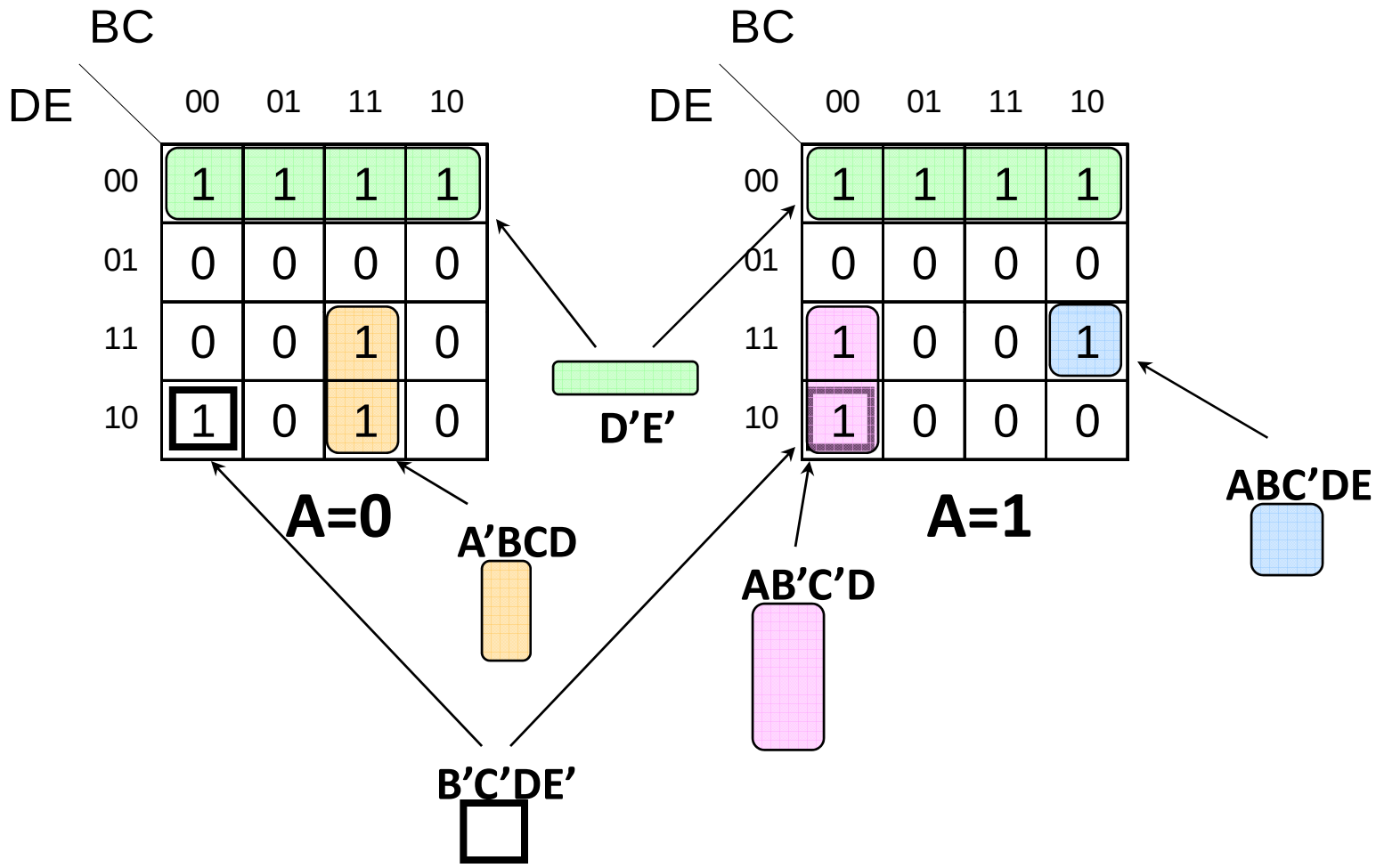
This is the A=0 plane



This is the A=1 plane

The planes are adjacent to one another (one is above the other in 3D)

Some Implicants in a 5-Variable KMap

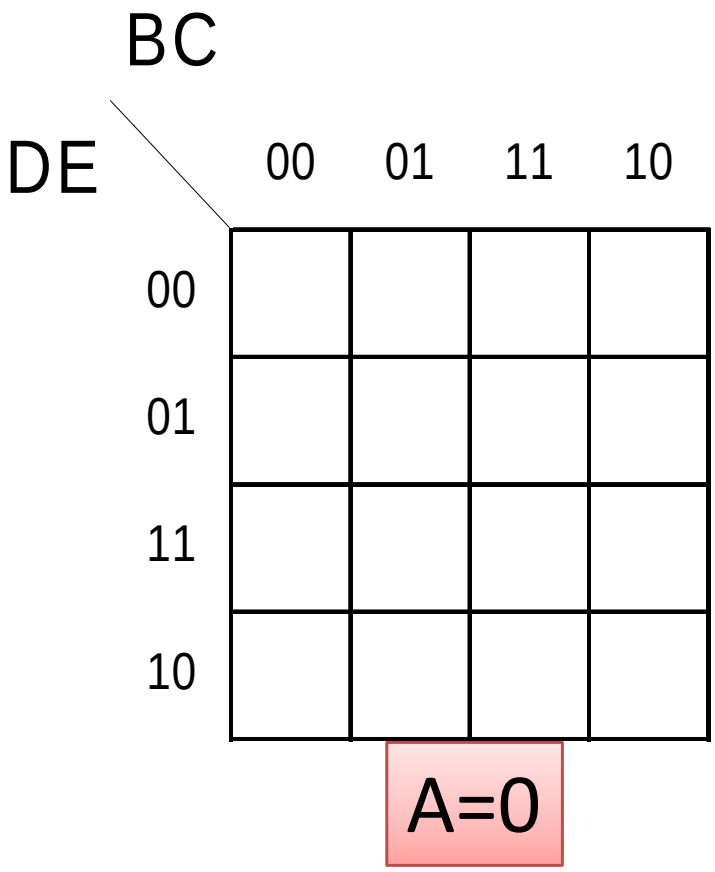


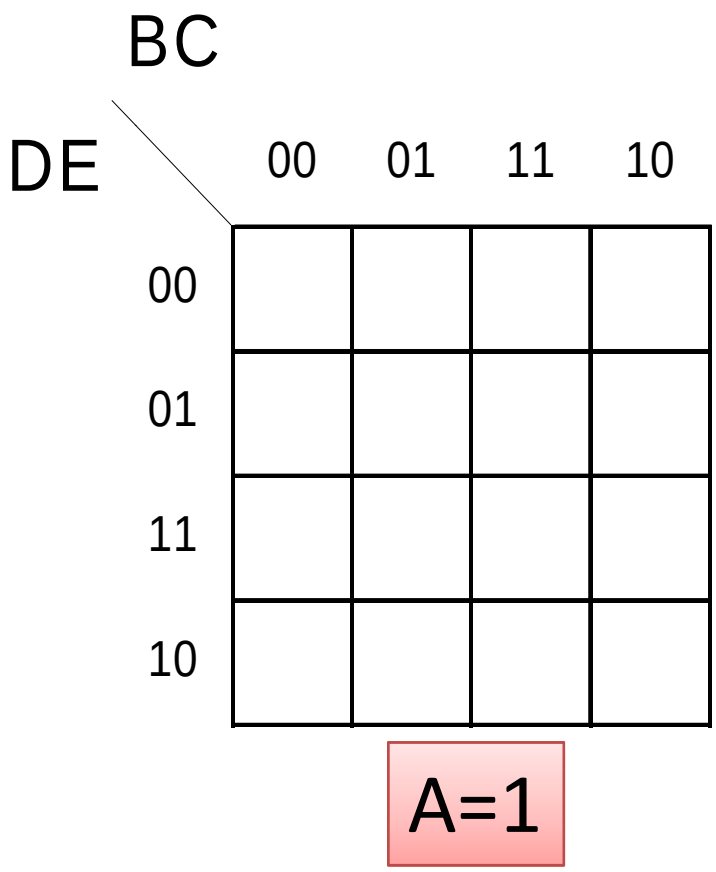
Some of these are not prime…

5-Variable KMap Example

Find the minimum sum-of-products for:

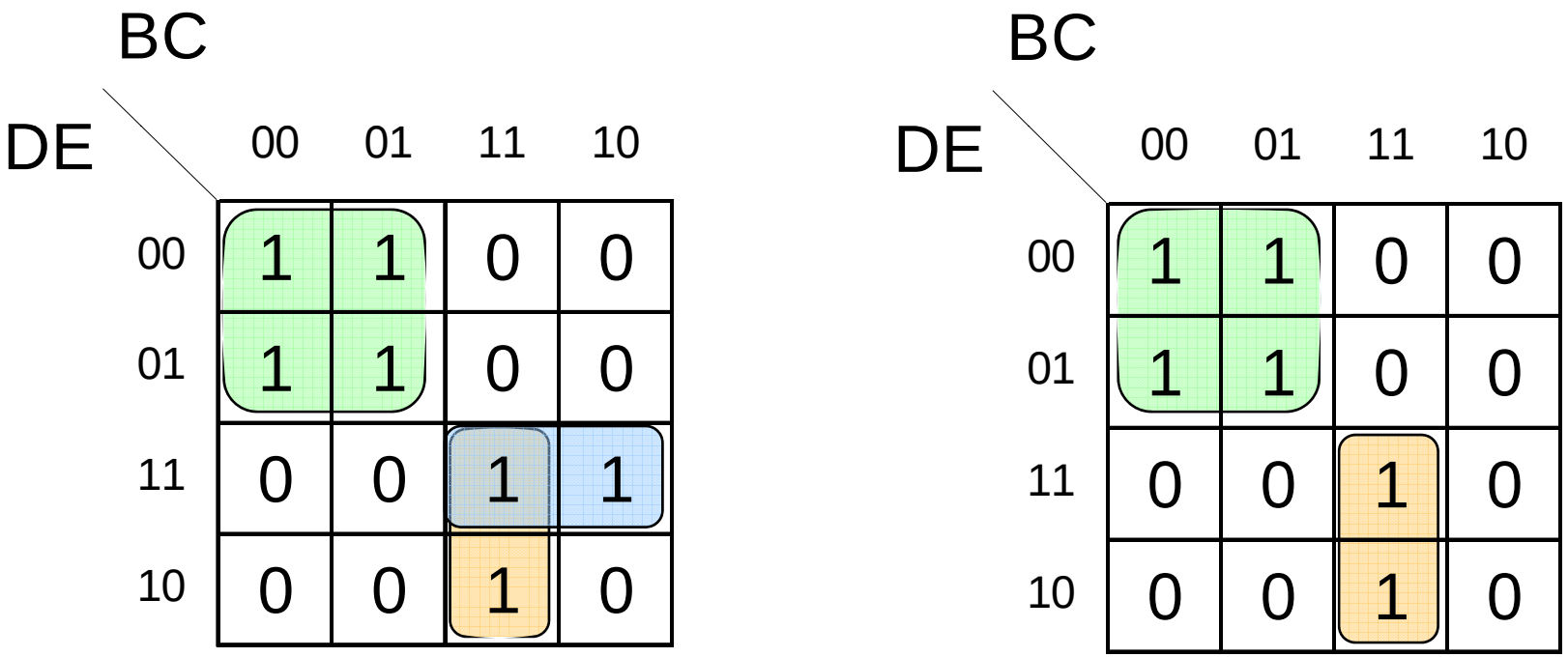
$F = \Sigma\, m\,(0,1,4,5,11,14,15,16,17,20,21,30,31)$





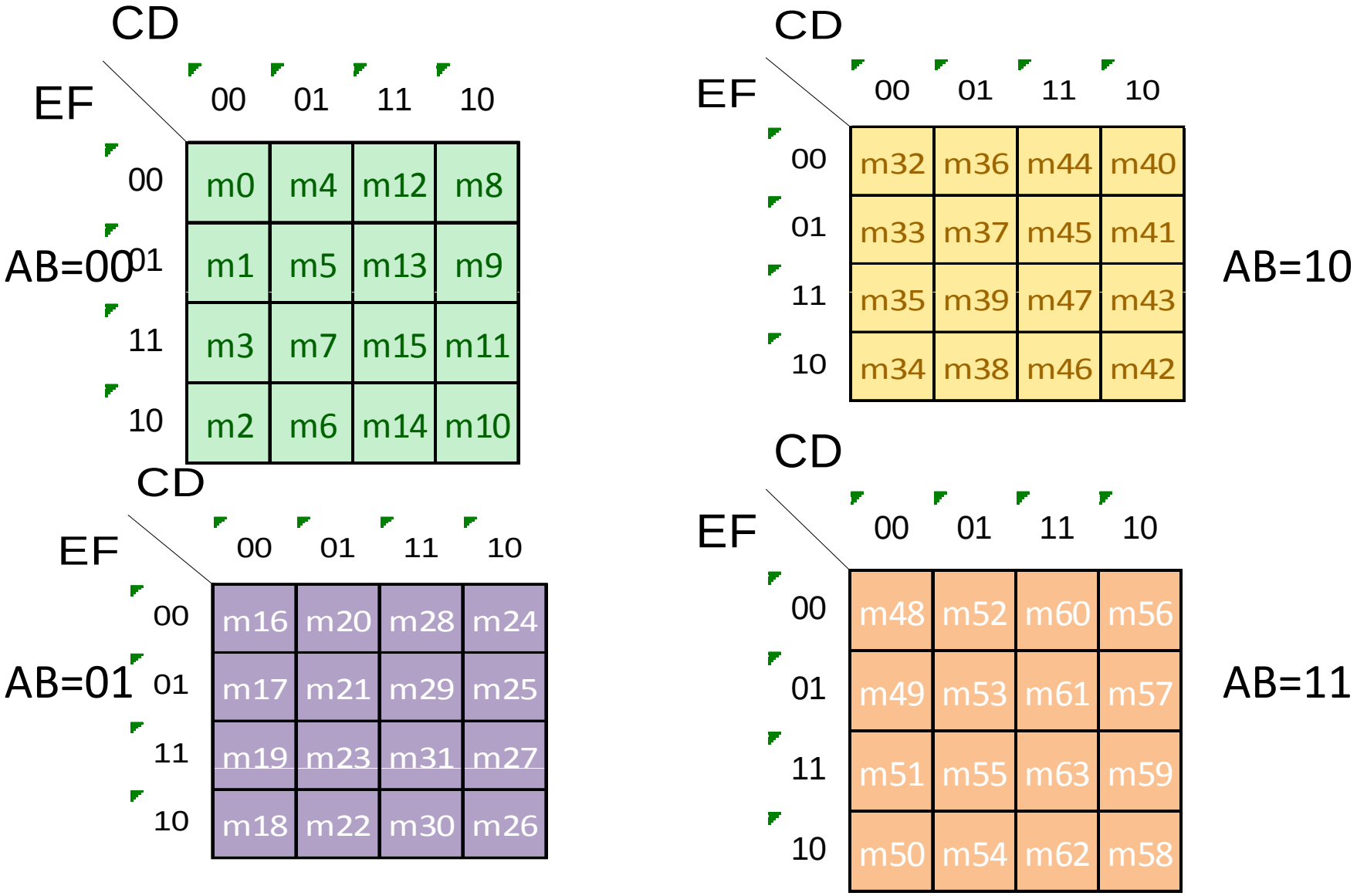
5-Variable KMap Example

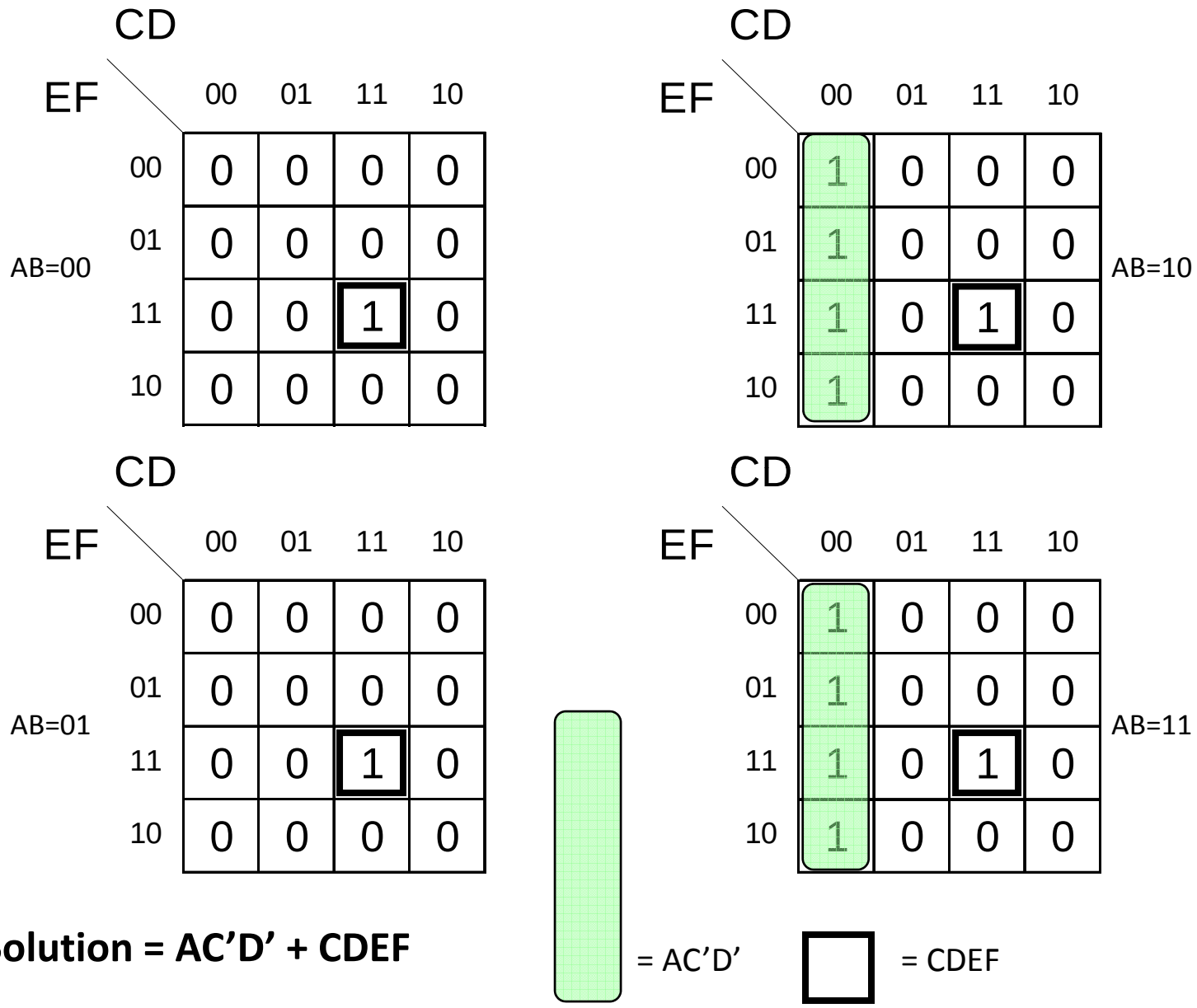
Find the minimum sum-of-products for:
$F = \sum m\,(0,1,4,5,11,14,15,16,17,20,21,30,31)$



$F = B'D' + BCD + A'BDE$

6-Variable Karnaugh Map





Solution = AC'D' + CDEF

KMap Summary

- A Kmap is simply a folded truth table
  - where physical adjacency implies logical adjacency
- KMaps are most commonly used hand method for logic minimization
- KMaps have other uses for visualizing Boolean equations
  - you may see some later.

Thanks

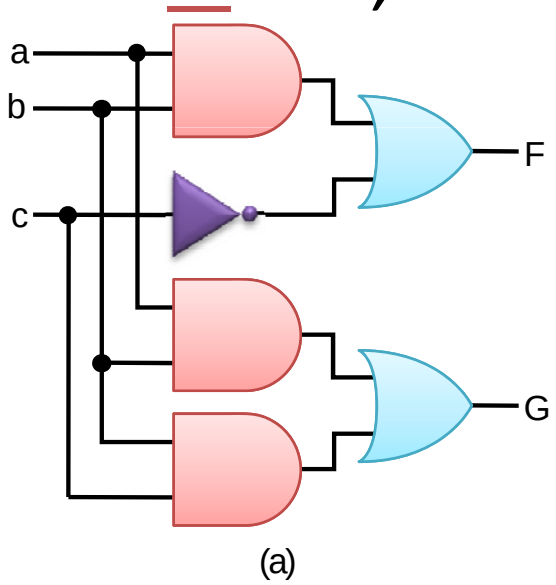
Logic Components & QM Logic Minimization

Outline

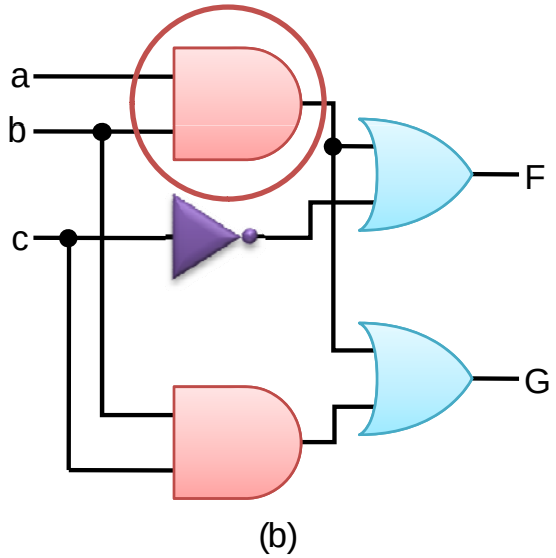
- Karnaugh map with >= 5 variable
- Study of Components
  - Multiplexor, Decoder
  - Logic Implementation Using MUX & Decoder
  - Mux: 7 Segment Display
  - 4 Bit Adder
- Quine-McCluskey (QM) Logic Minimization
- More Examples

Multiple-Output Circuits

- Many circuits have more than one output
- Can give each a separate circuit, or can share gates
- Ex: F = ab + c', G = ab + bc



Option 1: Separate circuits



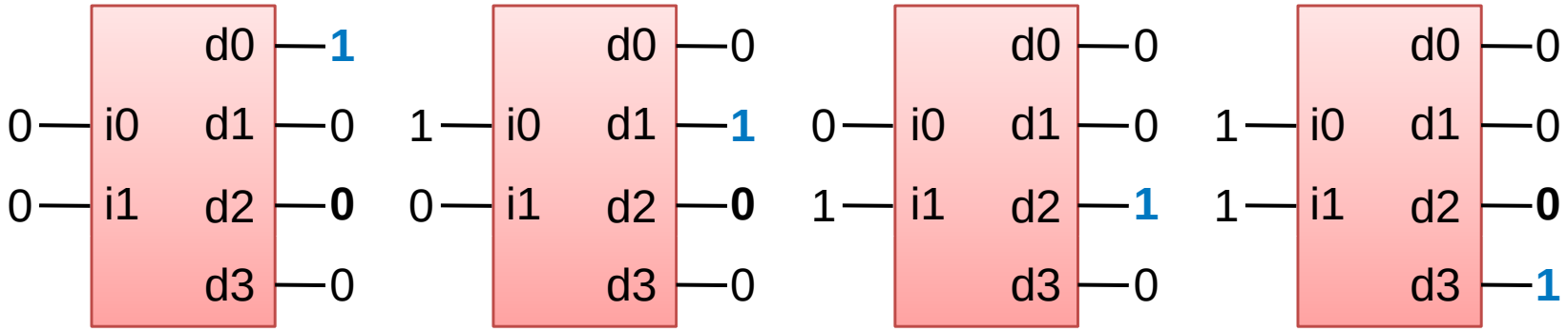
Option 2: Shared gates

Decoder

- Reception counter : When you reach a Academic Institute
  - Receptionist Ask: Which Dept to Go ?
  - Receptionist Redirect you to some building according to your Answer.
- Decoder : knows what to do with this: Decode
- N input: 2^N output
- Memory Addressing
  - Address to a particular location

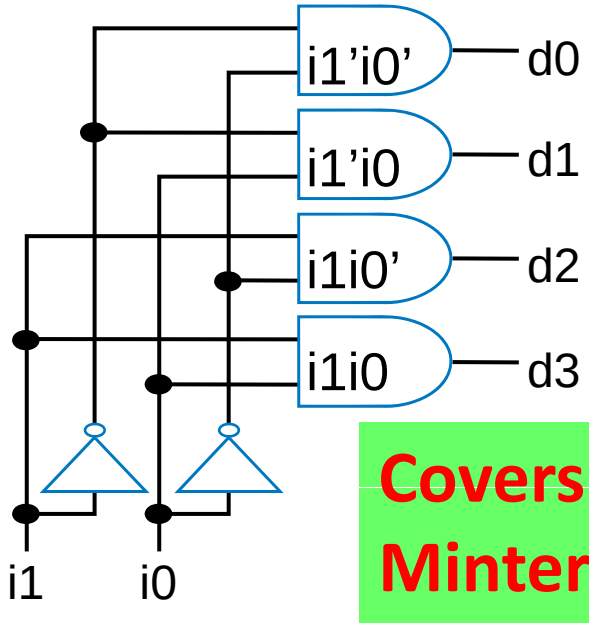
Decoders

- **Decoder**: Popular combinational logic building block, in addition to logic gates
  - Converts input binary number to one high output
- 2-input decoder: four possible input binary numbers
  - So has four outputs, one for each possible input binary number

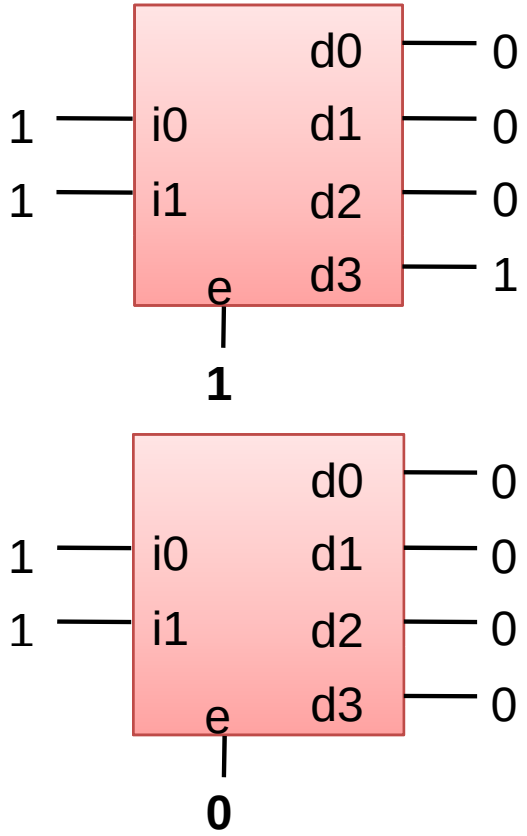


Decoders and Muxes

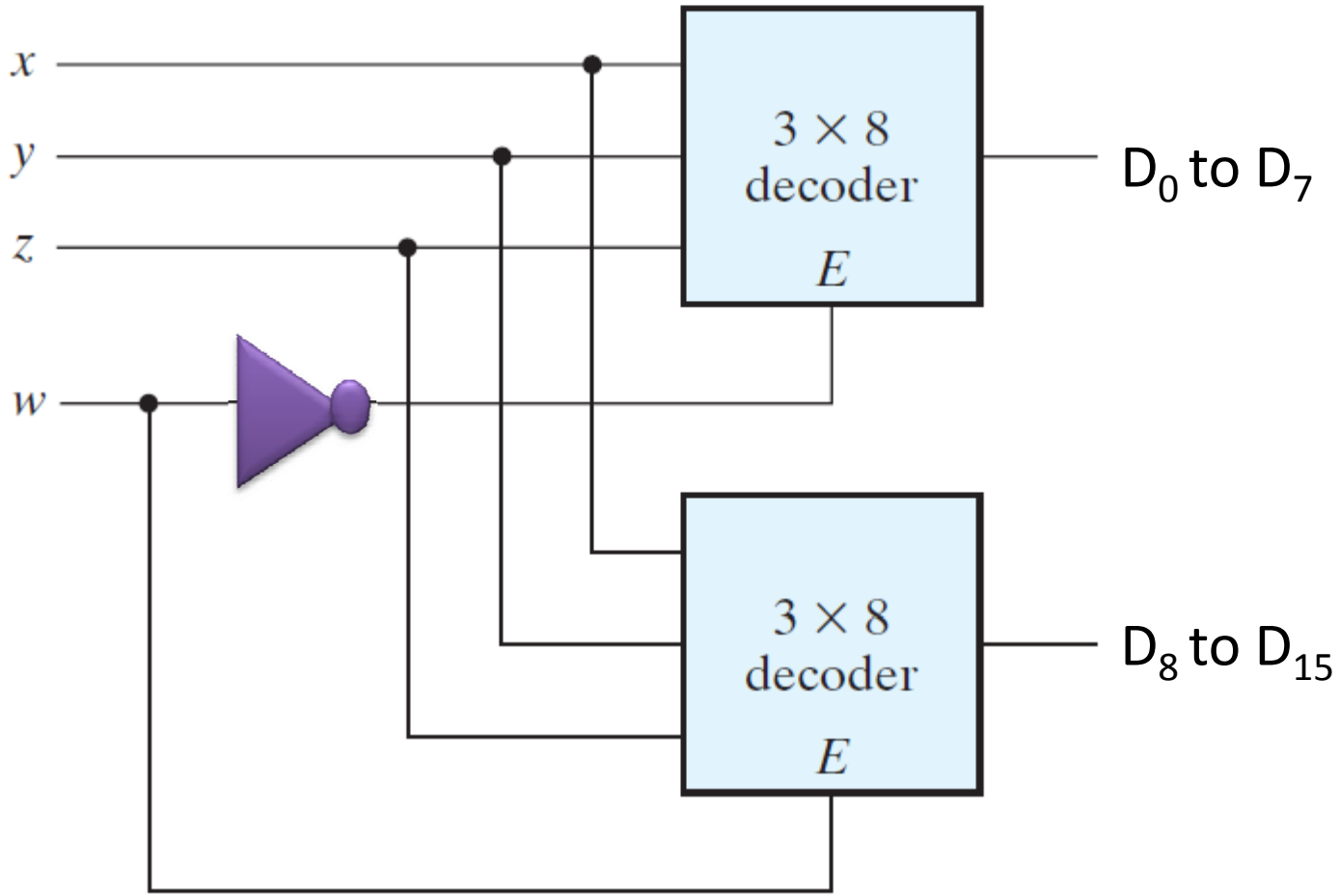
- Internal design
  - AND gate for each output to detect input combination
- Decoder with enable e
  - Outputs all 0 if e=0, Regular behavior if e=1
- n-input decoder: 2^n outputs



Covers All Minterms



4-to-16 Decoder using two 3-to-8 Decoders



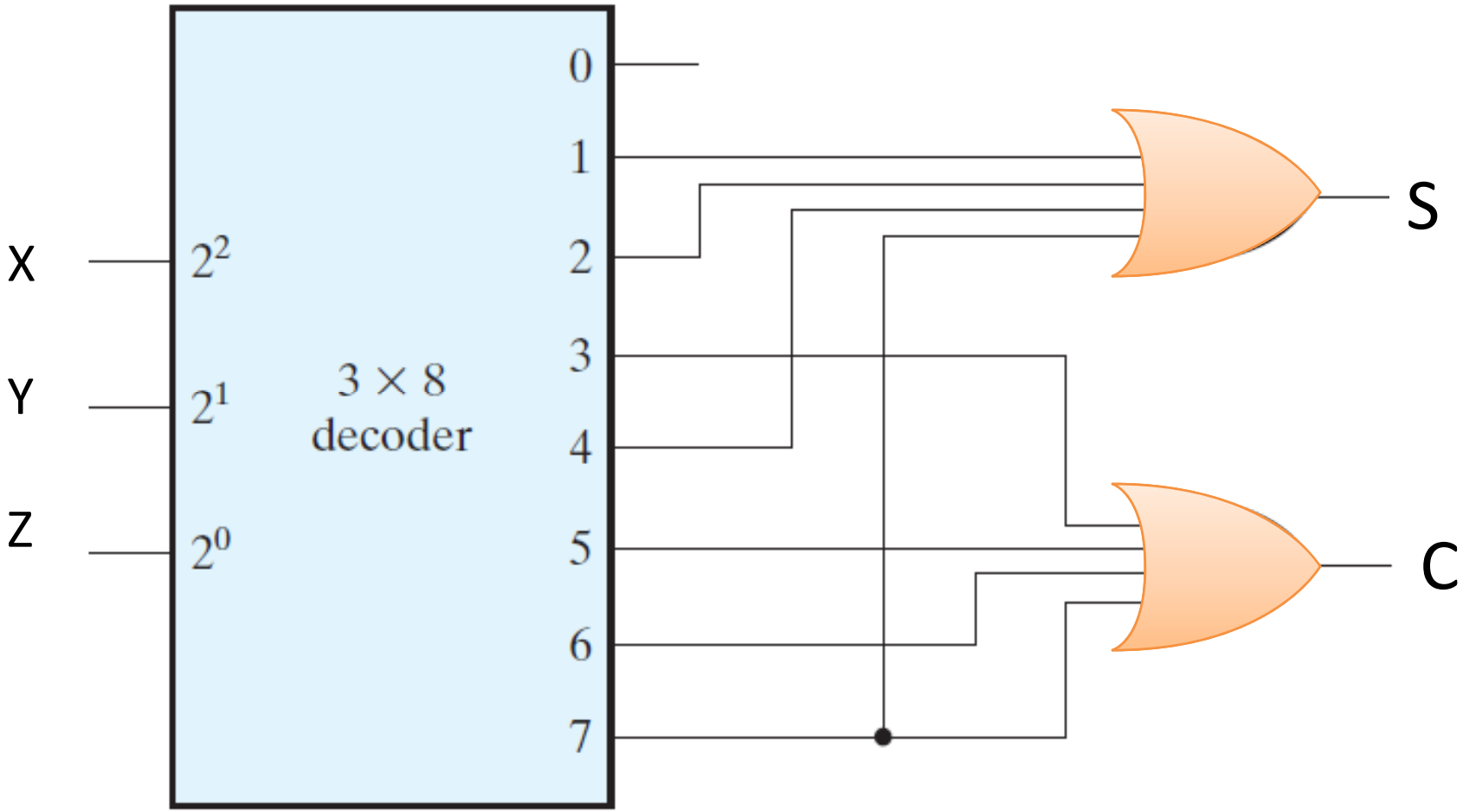
Boolean Function Implementation using Decoders

- Using a n-to-2n decoder and OR gates any functions of n variables can be implemented.
- Example:

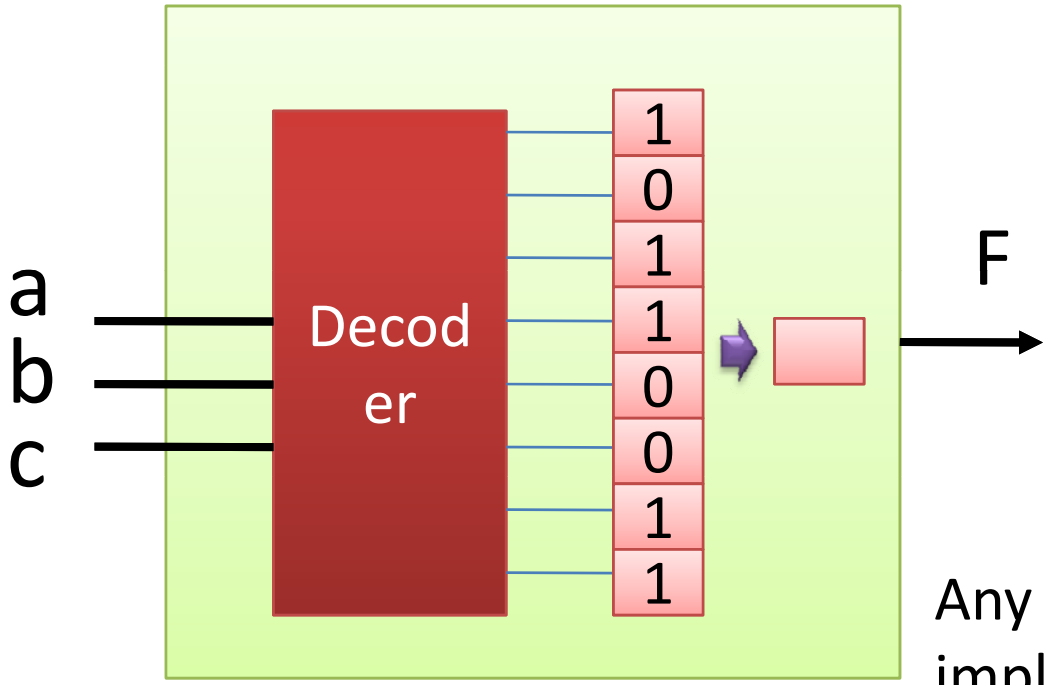
  $S(x,y,z)= \Sigma(1,2,4,7)$, $C(x,y,z)=\Sigma(3,5,6,7)$
- Functions S and C can be implemented using a 3-to-8 decoder and two 4-input OR gates

Decoder: Covers All Minterms

Implementation of S and C



Configurable Function using Decoder & Memory



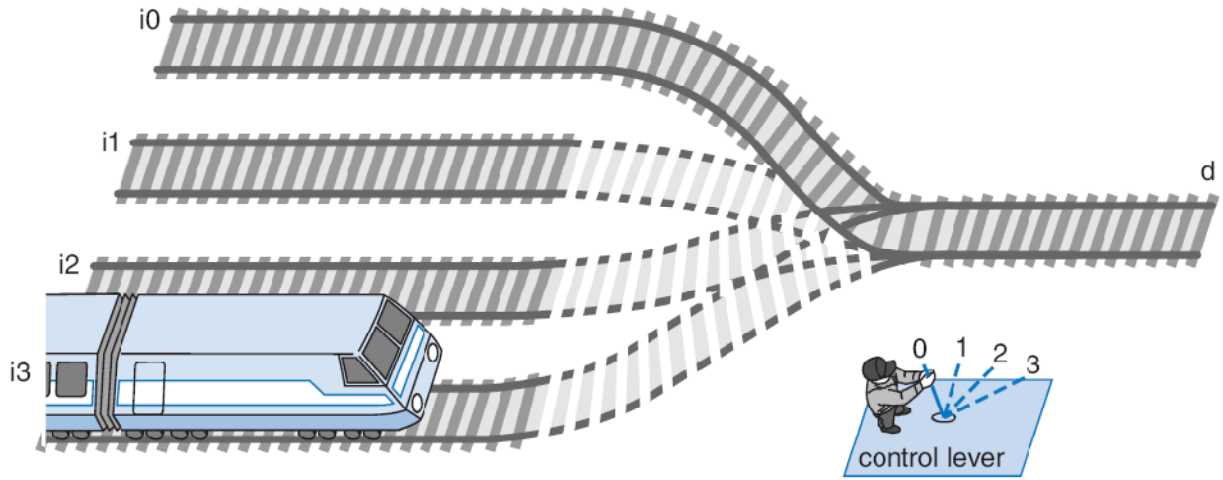
Any Function can be implemented

8 row x 1 col memory

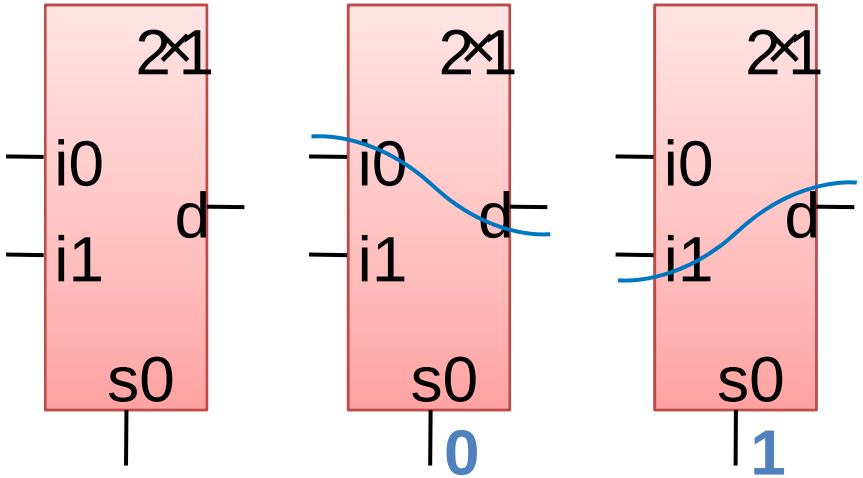
Function depend on Memory Elements, Direct correspondence to Truth Table

Multiplexor (Mux)

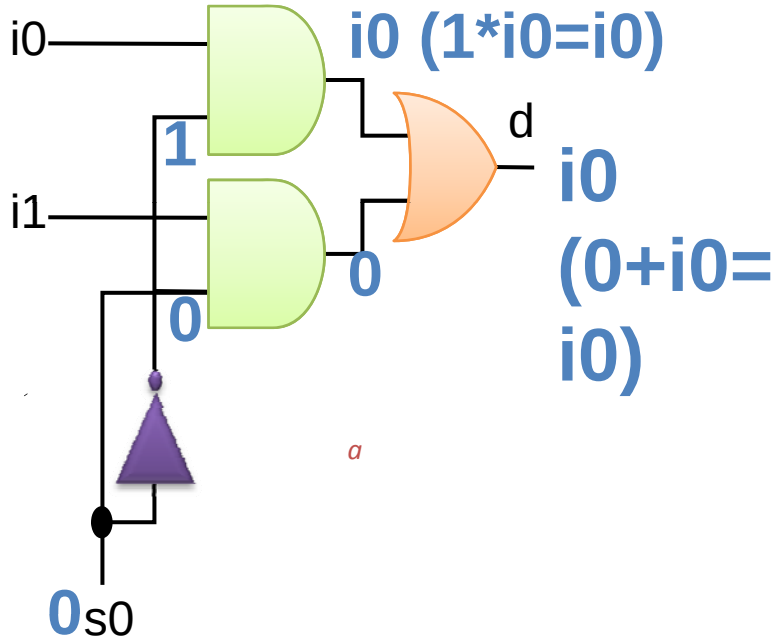
- Mux: Another popular combinational building block
  - Routes one of its N data inputs to its one output, based on binary value of select inputs
    - 4 input mux → needs 2 select inputs to indicate which input to route through
    - 8 input mux → 3 select inputs
    - N inputs → $\log_2(N)$ selects
  - Like a railyard switch



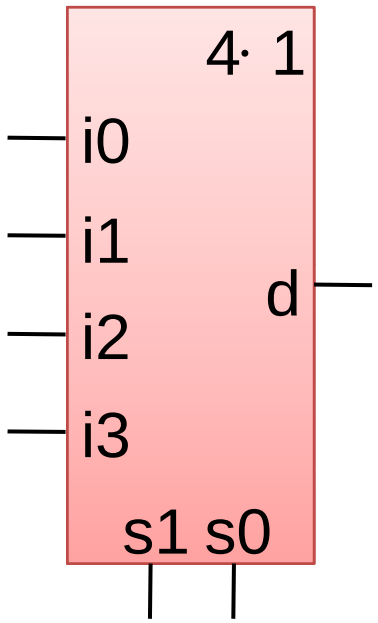
Mux Internal Design



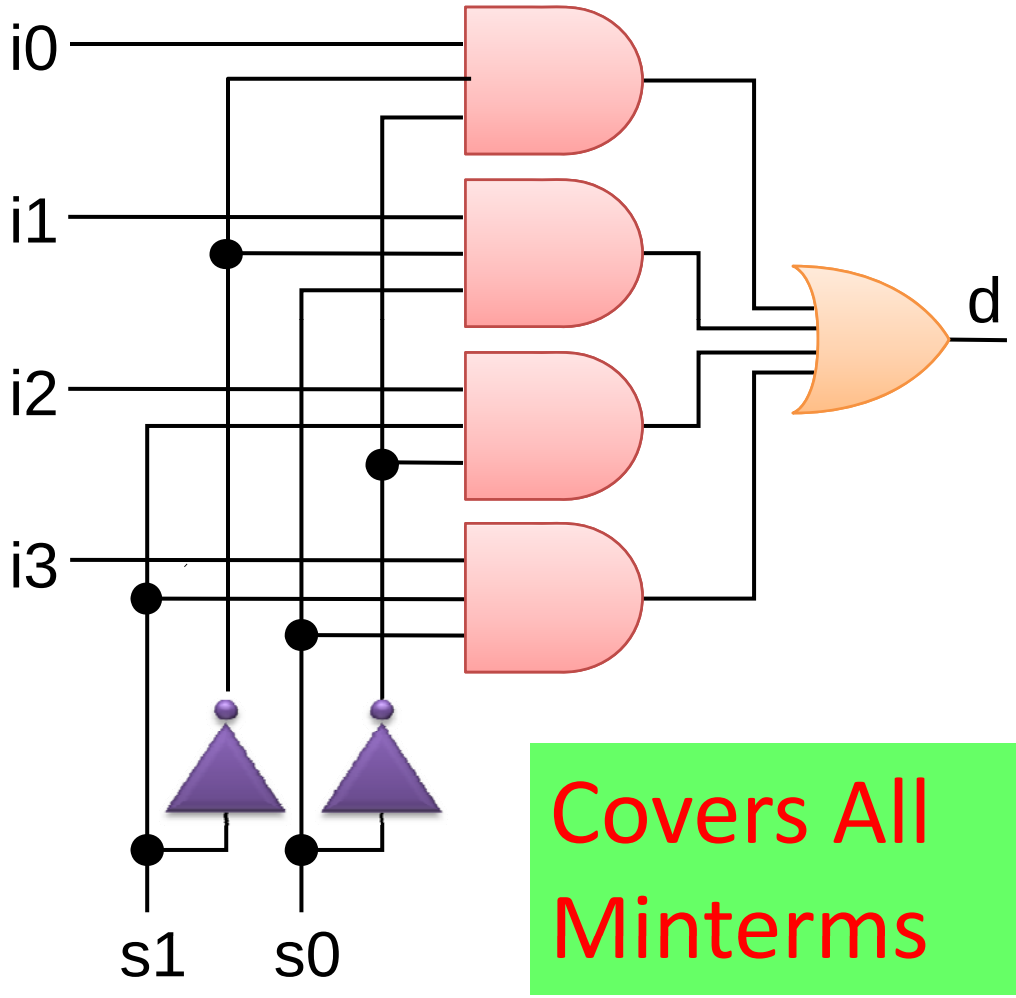
2x1 mux



Mux Internal Design

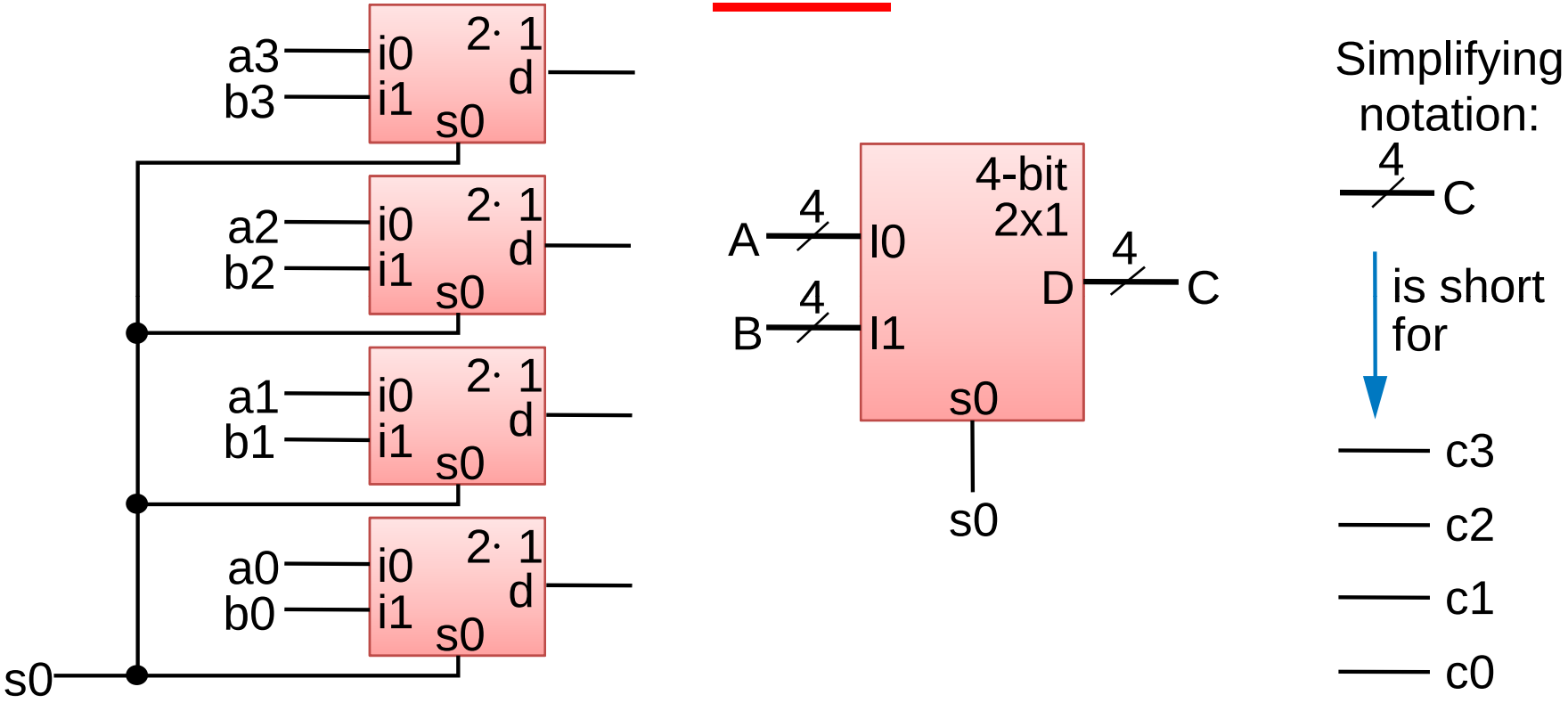


4x1 mux



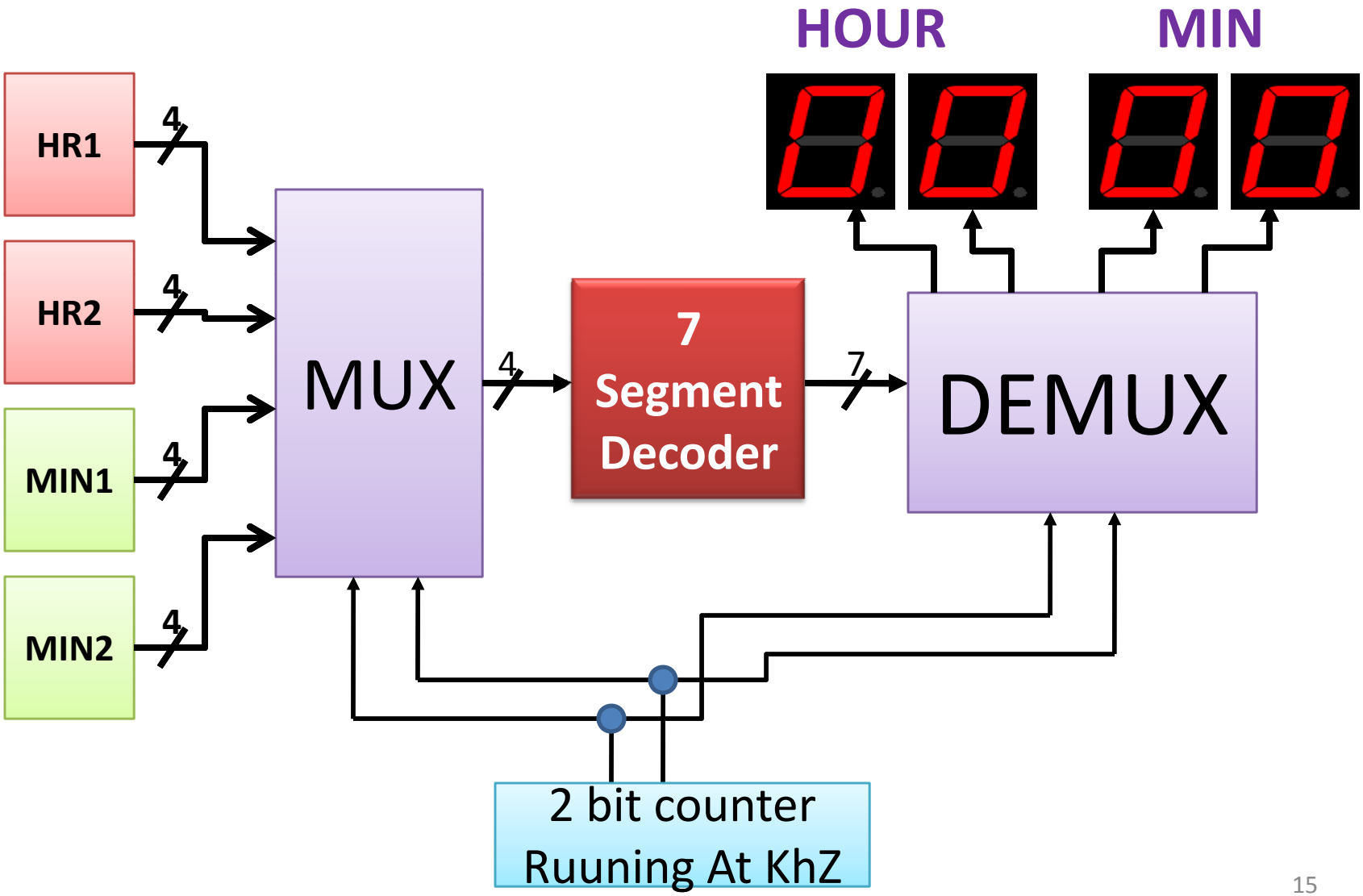
Covers All Minterms

Muxes Commonly Together -- N-bit Mux

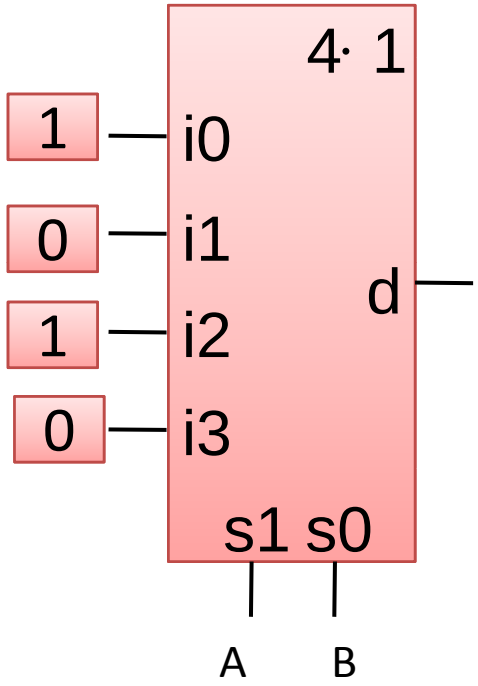


- Ex: Two 4-bit inputs, A (a3 a2 a1 a0), and B (b3 b2 b1 b0)
  - 4-bit 2x1 mux (just four 2x1 muxes sharing a select line) can select between A or B

Multiplexed 7 Segment Decoder



Implementing logic Function using MUX



F (A, B) = ∑m(0,2)

4x1 mux

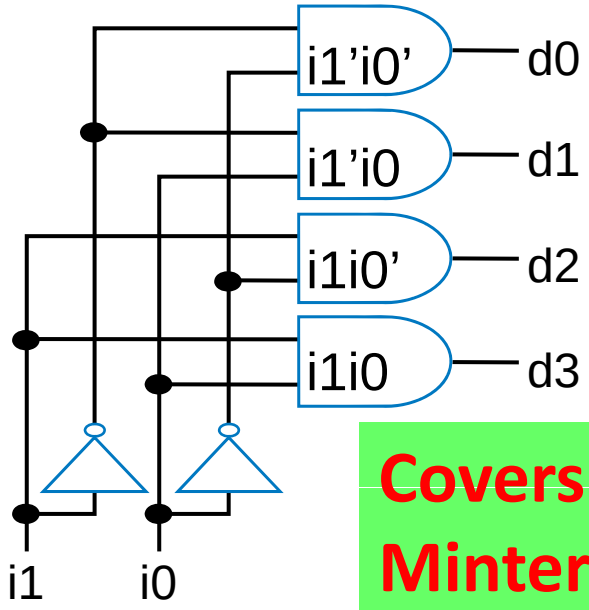
QM Logic Minimization

Outline

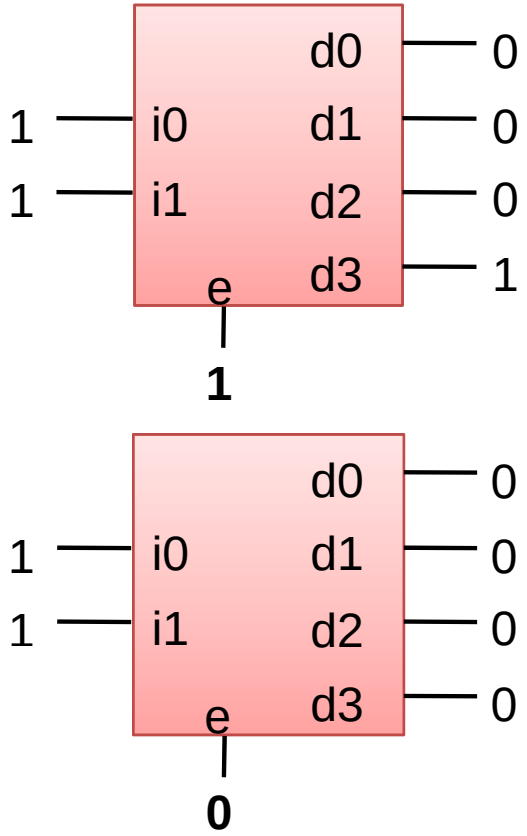
- Study of Components
  - Logic Implementation Using MUX & Decoder
  - 4 Bit Adder, BCD adder
- Quine-McCluskey (QM) Logic Minimization
- Examples
- Writing C/C++ program for QM Method

Decoders

- Internal design
  - AND gate for each output to detect input combination
- Decoder with enable e
  - Outputs all 0 if e=0, Regular behavior if e=1
- n-input decoder: 2^n outputs



Covers All Minterms



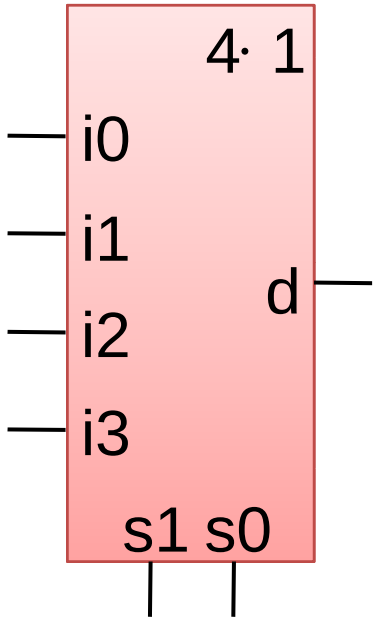
Boolean Function Implementation using Decoders

- Using a n-to-2n decoder and OR gates any functions of n variables can be implemented.
- Example:

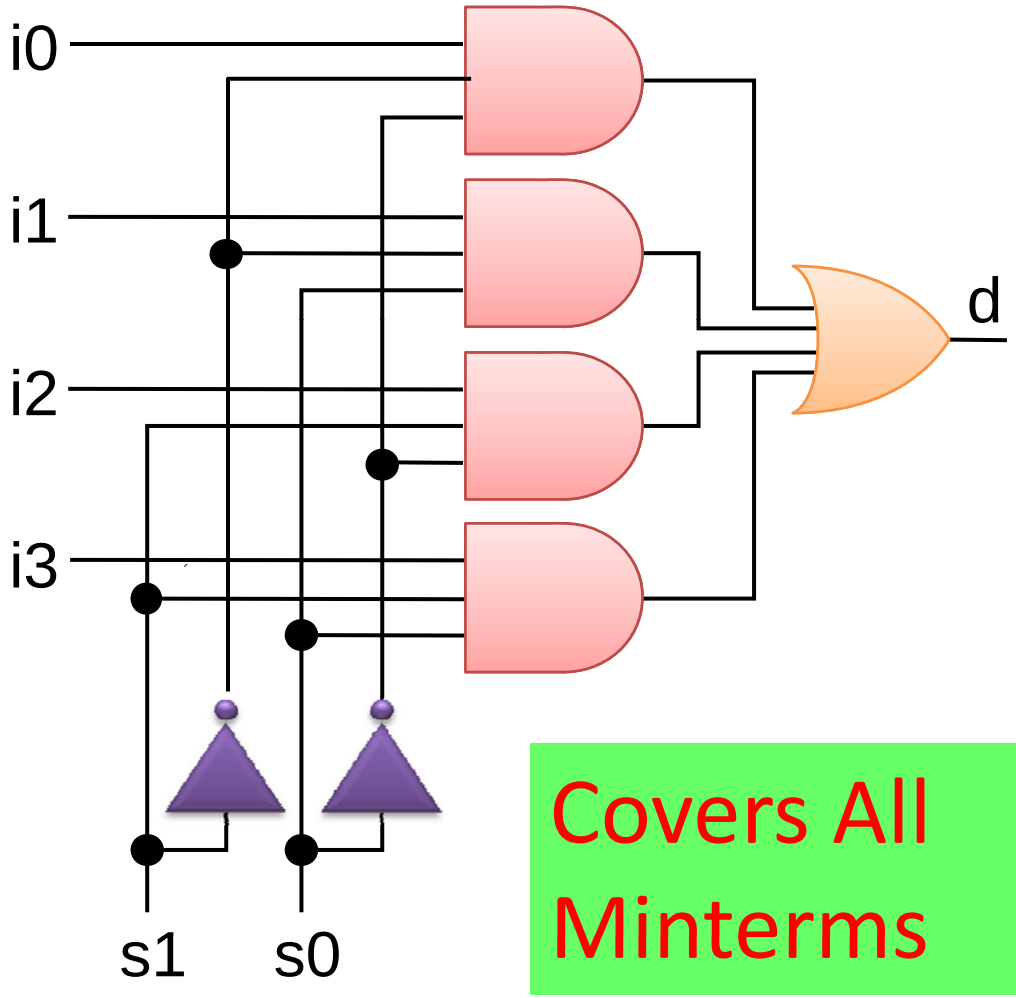
  $S(x,y,z)= \Sigma(1,2,4,7)$, $C(x,y,z)=\Sigma(3,5,6,7)$
- Functions S and C can be implemented using a 3-to-8 decoder and two 4-input OR gates

Decoder: Covers All Minterms

Mux

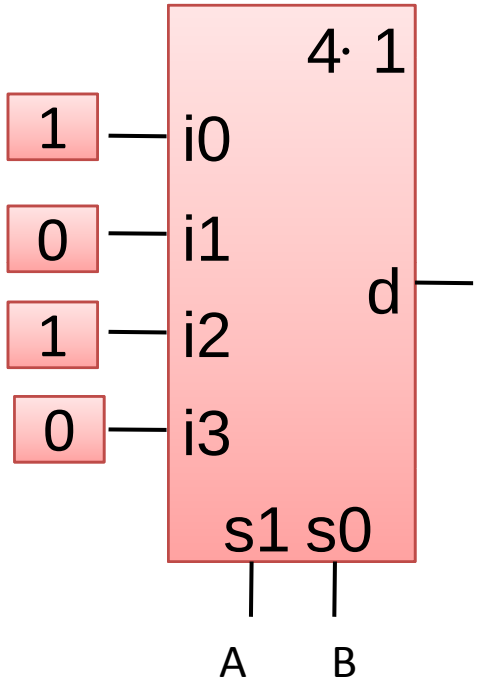


4x1 mux



Covers All Minterms

Implementing logic Function using MUX

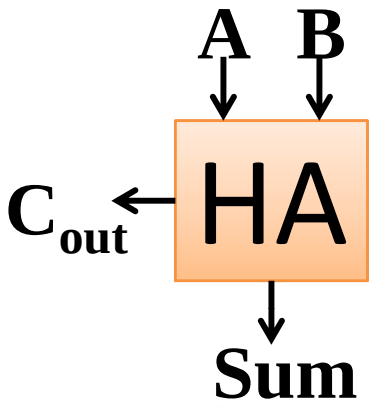


F (A, B) = ∑m(0,2)

4x1 mux

Adding Two One-bit Operands

- One-bit Half Adder:

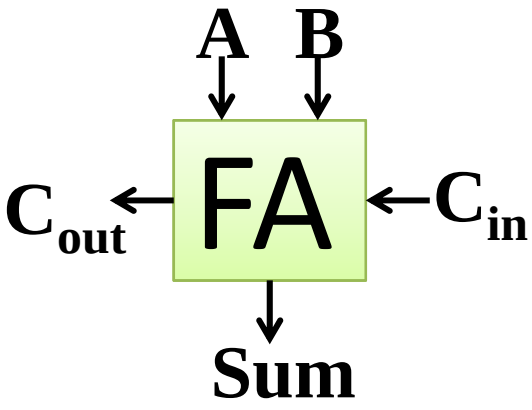


$$Sum = A \oplus B$$

$$Cout = A.B$$

A	B	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- One-bit Full Adder:



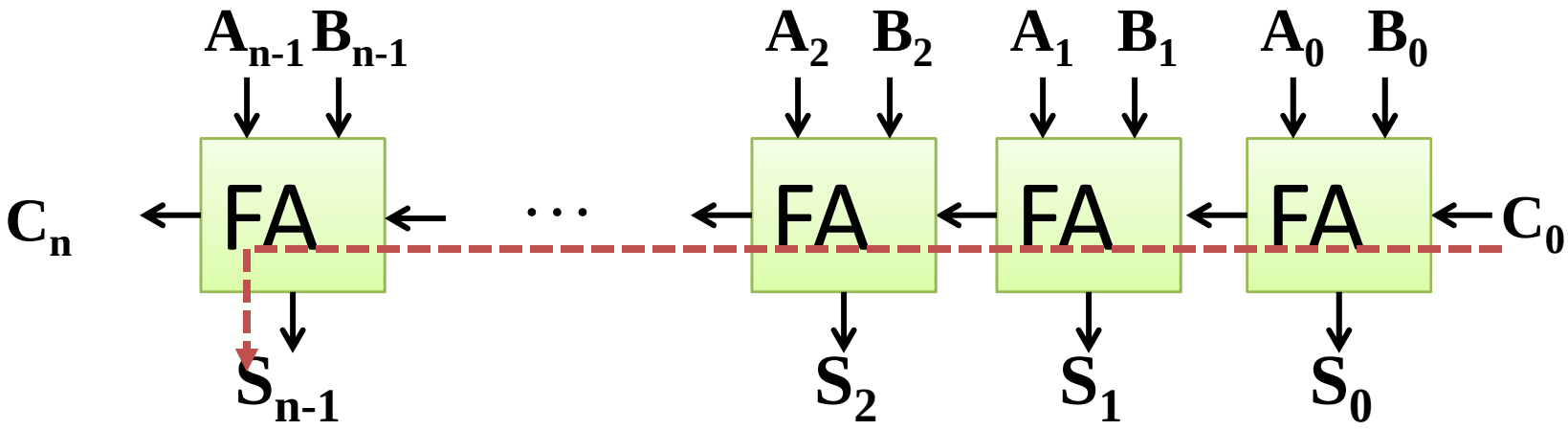
$$\mathbf{Sum = A \oplus B \oplus Cin}$$

$$\mathbf{Cout = A.B + B.Cin + A.Cin}$$

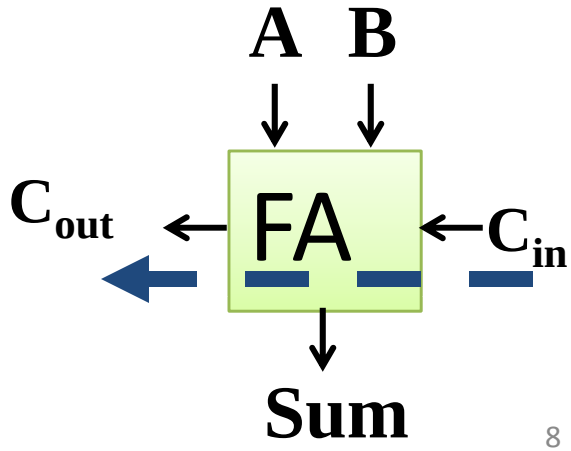
C_{in}	A	B	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

N-Bit Ripple-Carry Adder: Series of FA Cells

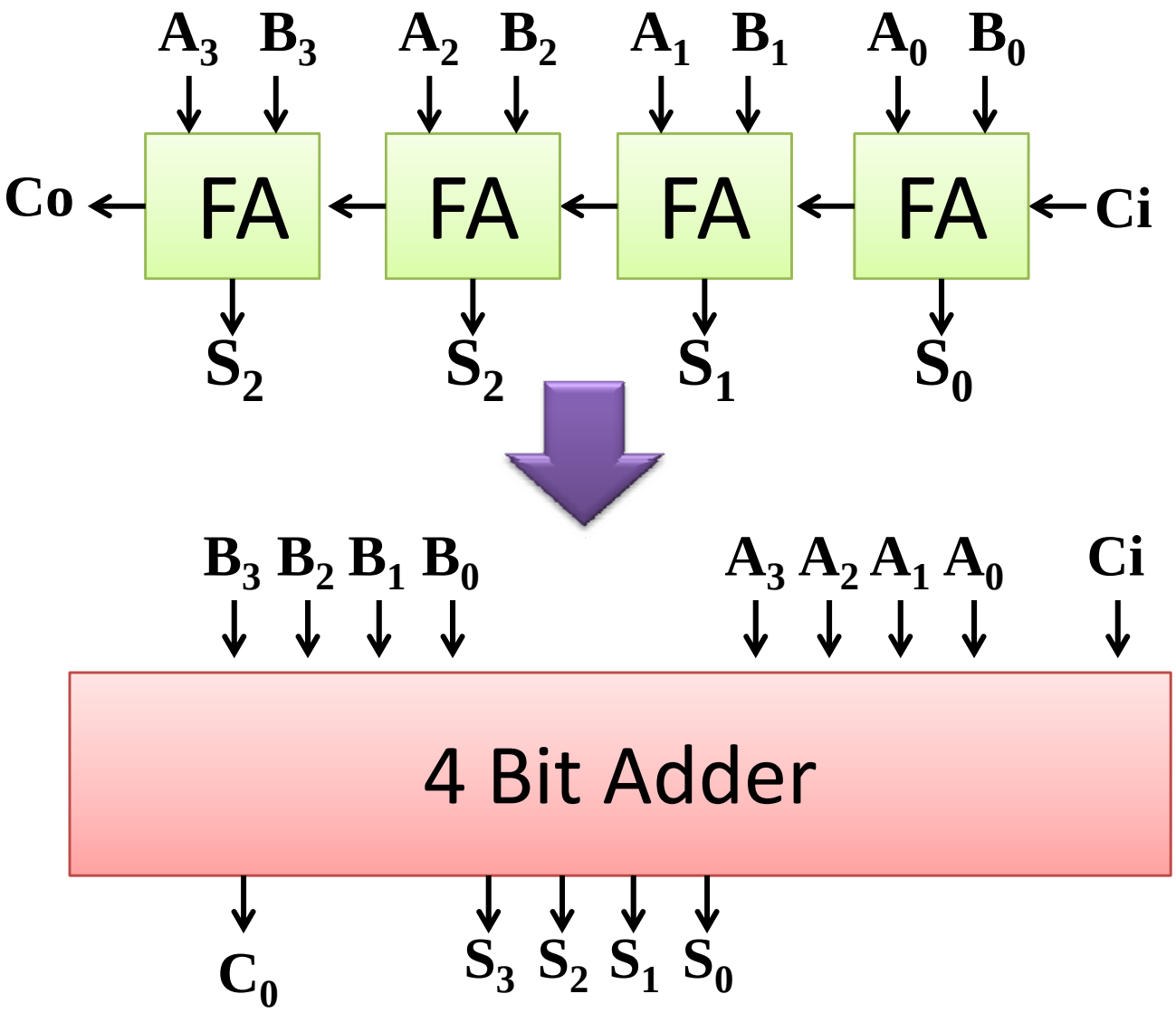
- To add two n-bit numbers



- Adder delay = Tc * n
- Tc = (C_{in} to C_{out} delay) of a FA

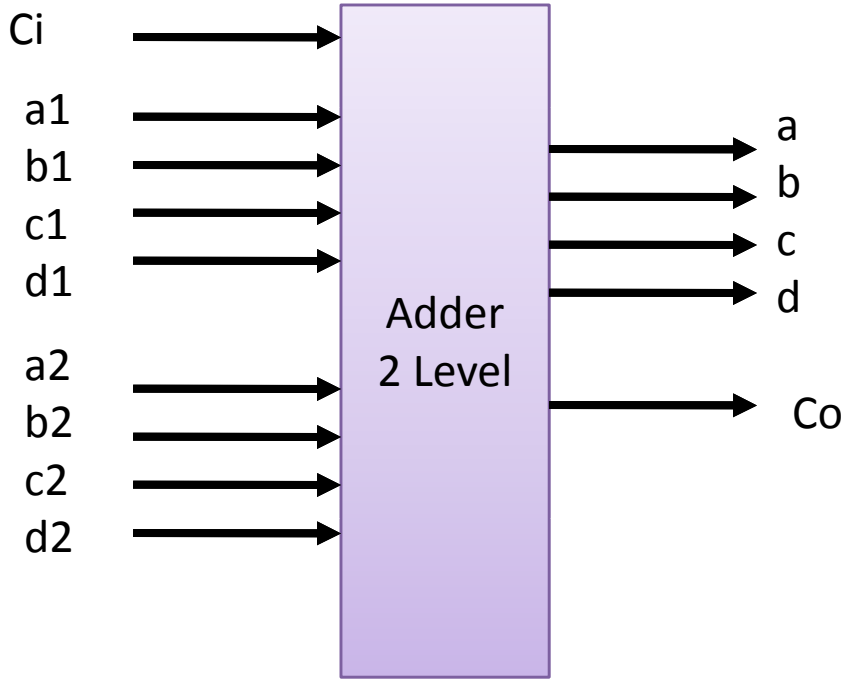


4 bit Binary Adder



Binary Adder (Two Level)

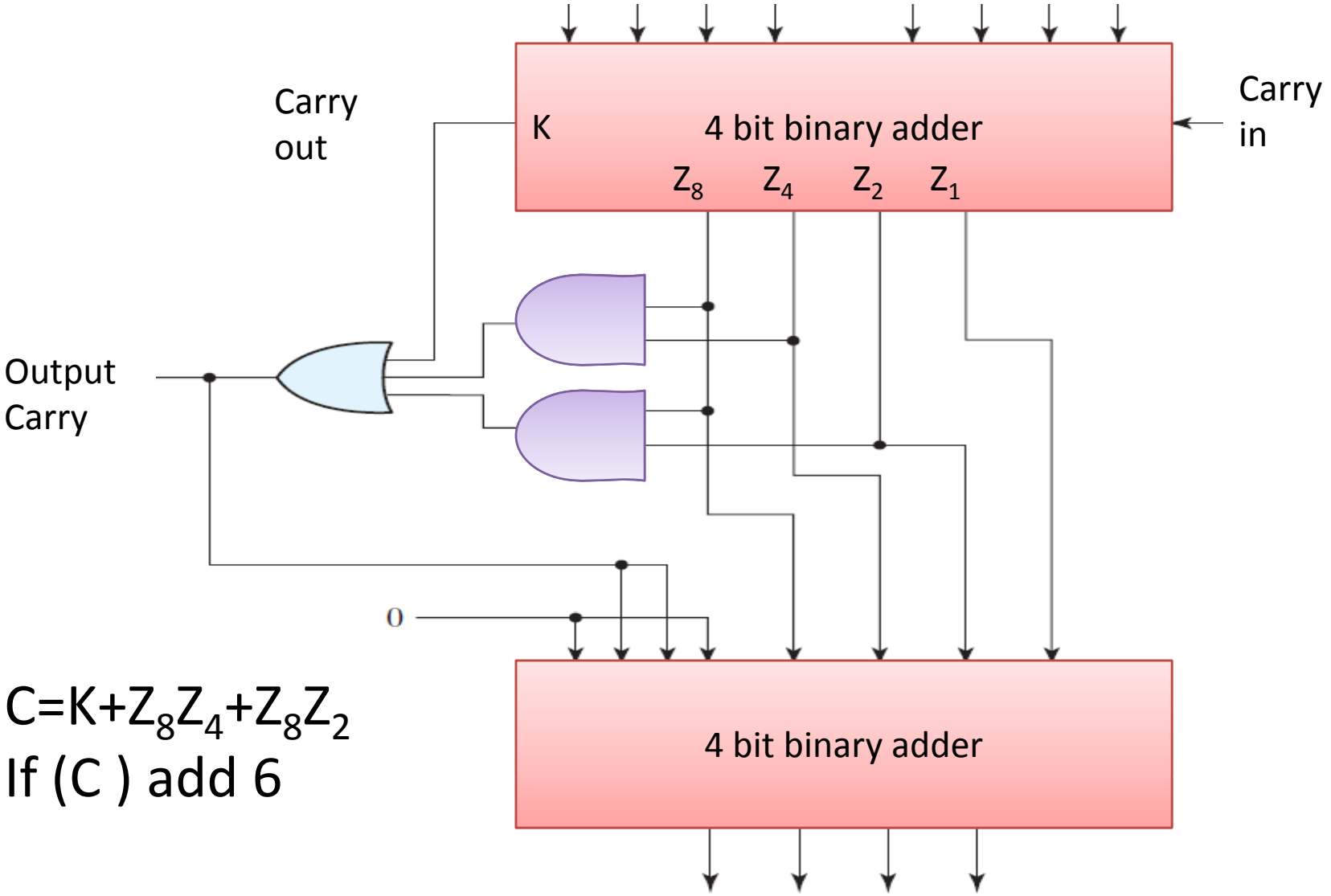
- Treat as 9 input & 5 output functions
- Generate Truth Table for each outputs
- Solve each function using KMAP/QM Method
- Only Two Level: No carry Propagation



Decimal Adder

- Decimal numbers are represented with BCD code.
- When two BCD digits A and B are added
  - if A+B<10 result is a valid BCD digit
  - if A+B>9 result will not be valid BCD digit. ˙It must be corrected by adding 6 to the result
- If A+B >9 add 6 to solve this issue

Decimal Adder

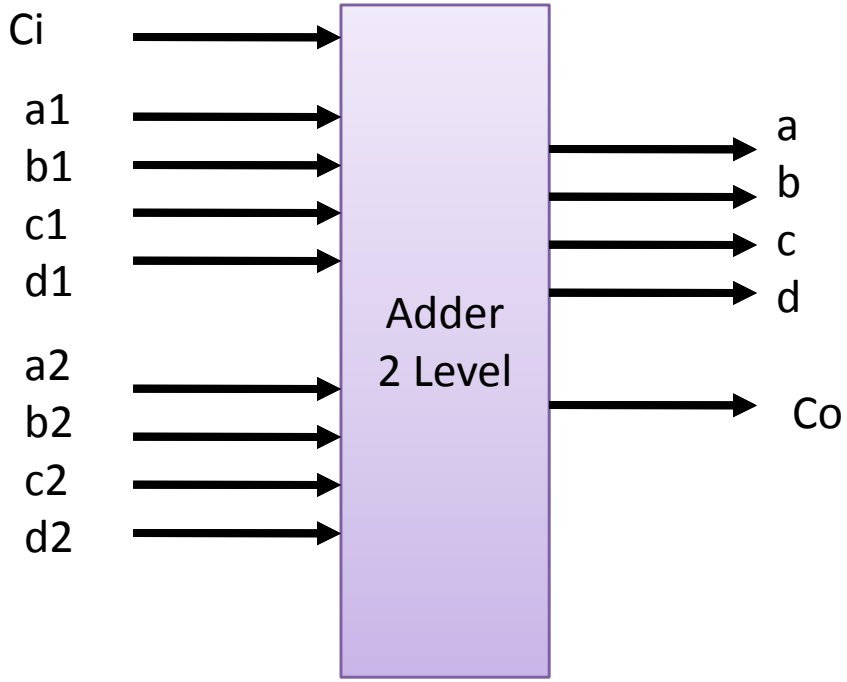


$C=K+Z_8Z_4+Z_8Z_2$

If (C) add 6

BCD Adder (Two Level)

- Treat as 9 input & 5 output functions
- Generate Truth Table for each outputs
- Solve each function using KMAP/QM Method
- Only Two Level: No carry Propagation



Quine-McCluskey Method for Minimization

- KMAP methods was practical for at most 6 variable functions
- Larger number of variables: need method that can be applied to computer based minimization
- **Quine-McCluskey** method

- For example:

$$\sum m(0,1,2,3,5,7,13,15)$$

QM Method

- Phase I : finding Pis
  - Tabular methods: Grouping and combining
- Phase II: Covers minimal PIs

QM Method

- Minterms that differ in one variable's value can be combined.
- Thus we list our minterms so that they are in groups with each group having the same number of 1s.
- So the first step is ordering the minterms according to their number of 1s (0-cube list)
- only minterms residing in adjacent groups have the chance to be combined.):

QM Method

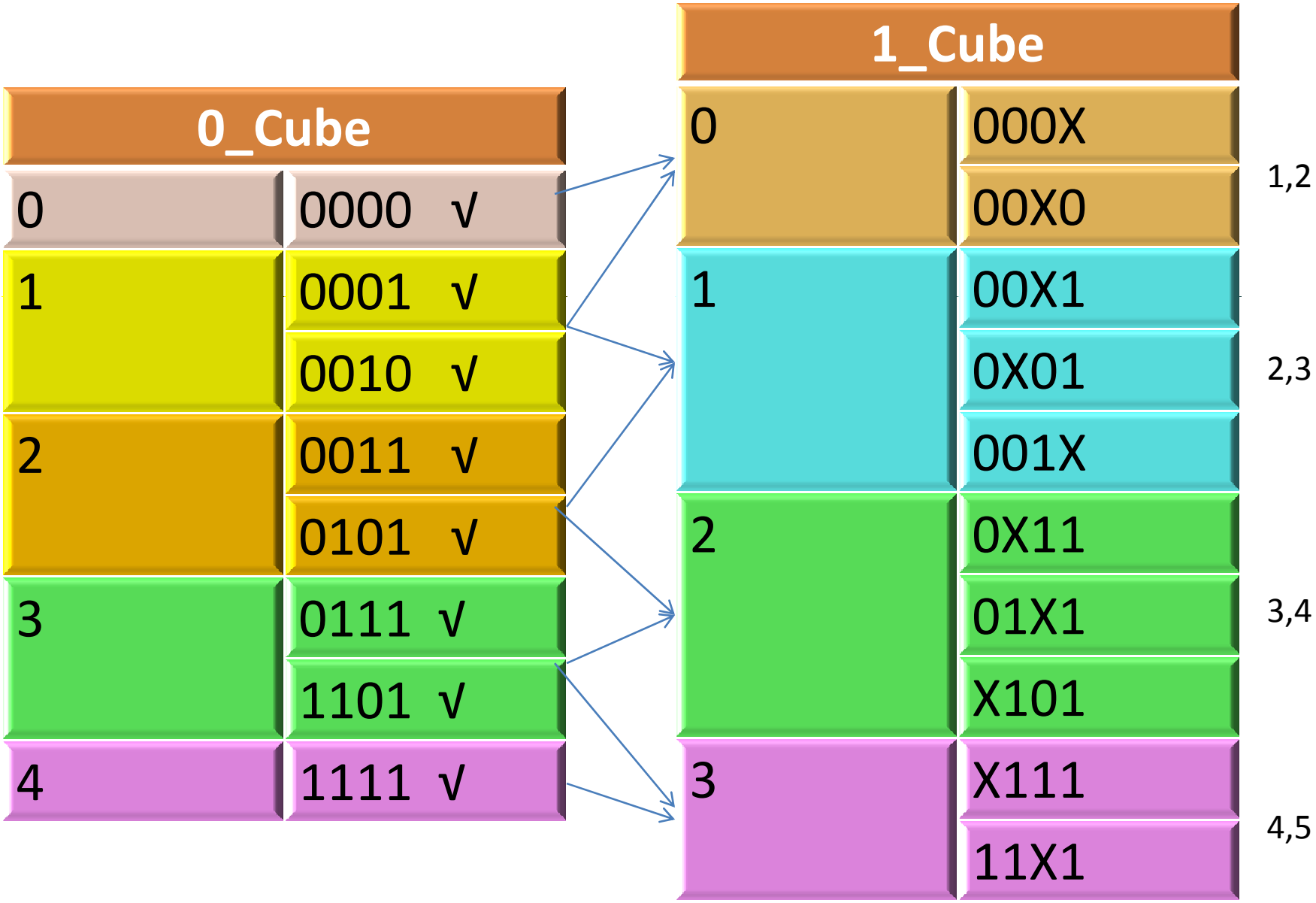
$$\sum m(0,1,2,3,5,7,13,15)$$

0_Cube	
0	0000
1	0001
	0010
2	0011
	0101
3	0111
	1101
4	1111

QM Method: Combining Adjacent

- Compare minterms of a group with those of an adjacent one to form 1-cube list.
- When doing the combining, we put checkmark alongside the minterms in the 0-cube list that have been combined.

QM Method

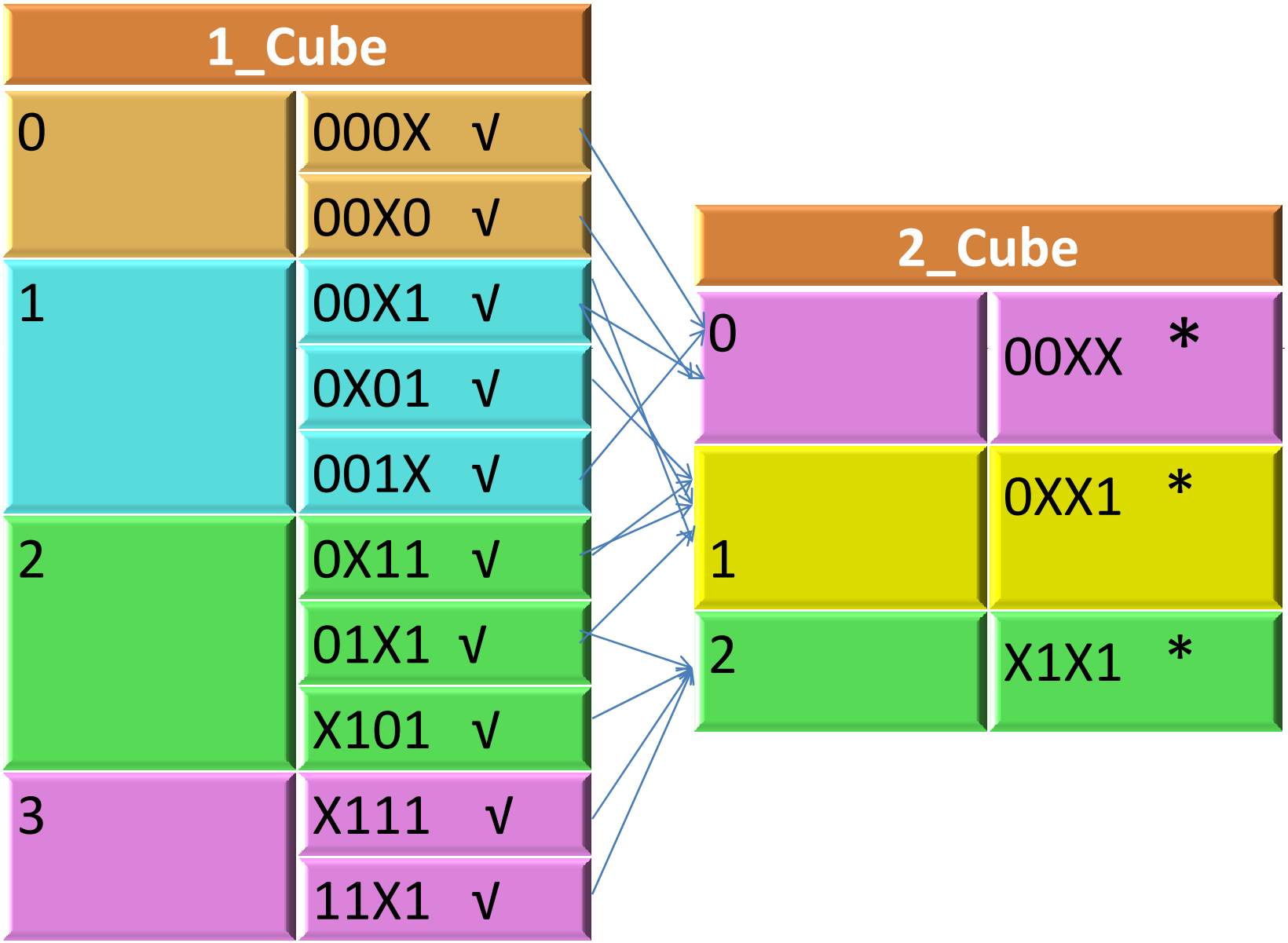

0_Cube
0
0000 √
1
0001 √
0010 √
2
0011 √
0101 √
3
0111 √
1101 √
4
1111 √
1_Cube
0
000X
00X0
1,2
1
00X1
0X01
001X
2,3
2
0X11
01X1
X101
3,4
3
X111
11X1
4,5

QM Method: Combining Adjacent

- Do same combination of comparing adjacent group minterms
  - To form 2-cubes, 3-cubes and so on.

- Only minterms of adjacent groups have the chance of being combined
  - **Which have an X in the same position.**

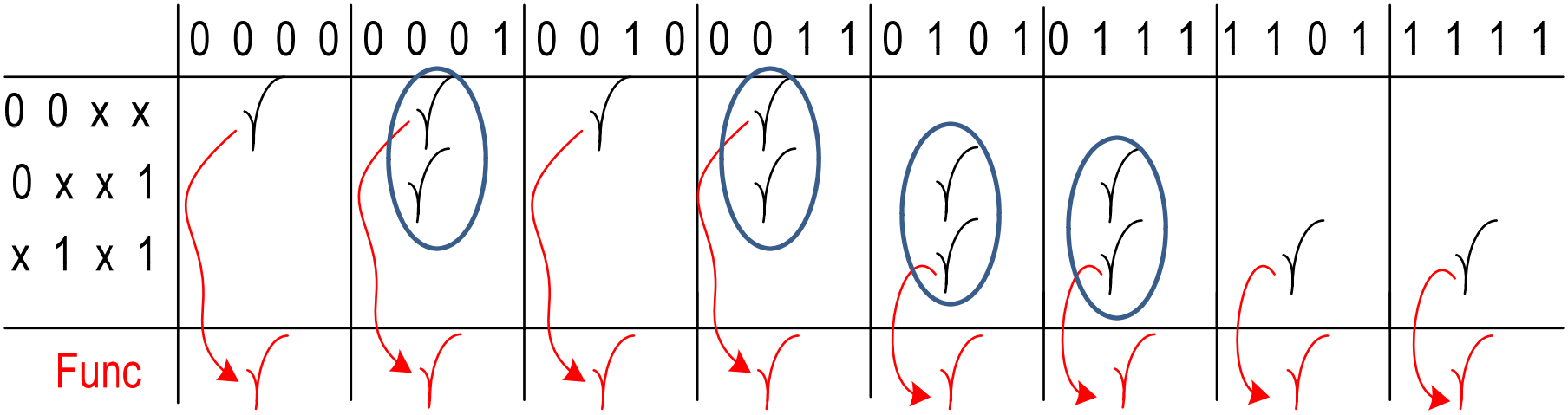
QM Method


1_Cube
0
000X √
00X0 √
1
00X1 √
0X01 √
001X √
2
0X11 √
01X1 √
X101 √
3
X111 √
11X1 √
2_Cube
0
00XX *
1
0XX1 *
2
X1X1 *

Q-M Method: Cover PIs

- PIs : terms left without checkmarks.
- After identifying our PIs, we list them against the minterms needed to be covered

$$\sum m(0,1,2,3,5,7,13,15)$$



QM Method : Covers

- To find a minimal cover, we first need to find essential PIs
- To do this we need to find columns that only have one checkmark in them, the according row will thus show the essential PI.
- After identifying essential PIs, that are necessarily part of the cover, we cover any remaining minterms using a minimal set of PIs.

In this example: $F(a,b,c,d) = \overline{a}\overline{b} + bd$

Thanks