ITI1100 Section D
Digital Systems I

Chapter 3:

Gate–Level minimization (1)

Prof. Mohammad Alja'Afreh

Summer 2019

Cost of Implementing a Logic Circuit

The cost of implementing a logic circuit is related to the number of gates used and with the number of inputs in each gate.

A *literal* is a boolean variable or its complement.

The cost of a boolean equation B_i represented in a sum-of-products form is given by:

$$C(B_i) = \sum_{j=0}^{k-1} P_j(B_i) + O(B_i)$$

where k is the number of terms in B_i,

$O(B_i)$ is related to the number of terms in B_i,

and $P_j(B_i)$ is related to the number of literals in the j^{th} term of B_i.

Cost of a Logic Circuit Examples

$$C(B_i) = \sum_{j=0}^{k-1} P_j(B_i) + O(B_i)$$

$$P_j(B_i) = \begin{cases} m & \text{if the } j^{\text{th}} \text{ term of } B_i \text{ has } m \text{ literals} \\ 0 & \text{if the } j^{\text{th}} \text{ term of } B_i \text{ has 1 literals} \end{cases}$$

$$O(B_i) = \begin{cases} m & \text{if } B_i \text{ has } m \text{ terms} \\ 0 & \text{if } B_i \text{ has 1 term.} \end{cases}$$

What is the cost of the following boolean equations?

f1(w,x,y,z) = wxy'z + wxyz'

C(f1) = 4+4+2=10

f2(w,x,y,z) = w' + x' + yz + y'z'

C(f2) = 0+0+2+2+4=8

g1(XYZ) = XY + X'Z + YZ

C(g1) = 2+2+2+3=9

g2(XYZ) = XY + X'Z

C(g2) = 2+2+2=6

h1(a,b) = ab

C(h1) = 2 + 0 = 2

h2(a,b) = b'

C(h2) = 0 + 0 = 0

The Karnaugh map

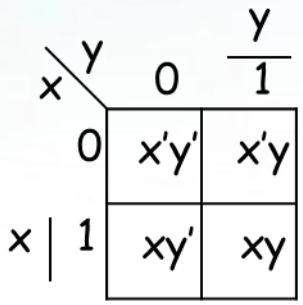
A Karnaugh Map is a graphical tool to assist on the minimization of logic equations.

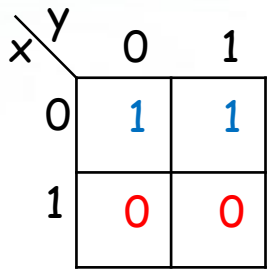
- **K-map** is an **alternate approach** to **representing Boolean functions**
- K-map representation can be used to **minimize Boolean functions**
- **Easy conversion** from **truth table** to **K-map**
- **Simple steps** that leads to **minimized SOP/POS representation** and **much faster** than using **Boolean algebra**
- K-map is **ideally suited** for **four or less variables**, becoming cumbersome for five or more variables.

The Karnaugh map (cont'd)

- **K-map** consists of **squares** and **each square** represents a **term**
- A K-map of **n variables** will have **2^n squares**
- map is arranged such that **two neighbors** differ in **only one variable** (e.g. xy and xy')
- For a Boolean expression, **product terms (minterms)** are denoted by **1's**, while **sum terms (Maxterms)** are denoted by **0's**.

$F = \Sigma(m_0, m_1) = x'y + x'y'$

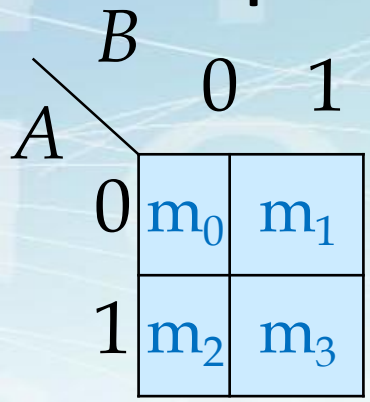




x	y	F
0	0	1
0	1	1
1	0	0
1	1	0

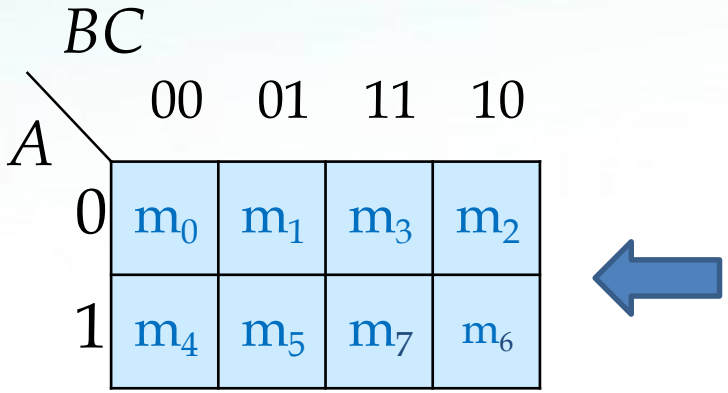
The Karnaugh map (cont'd)

- **Two-variable map:**



A	B	F
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3

- **Three-variable map:**

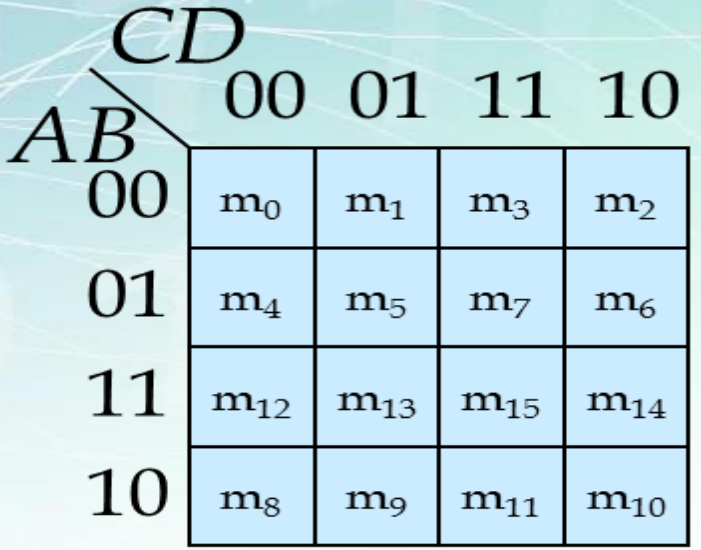


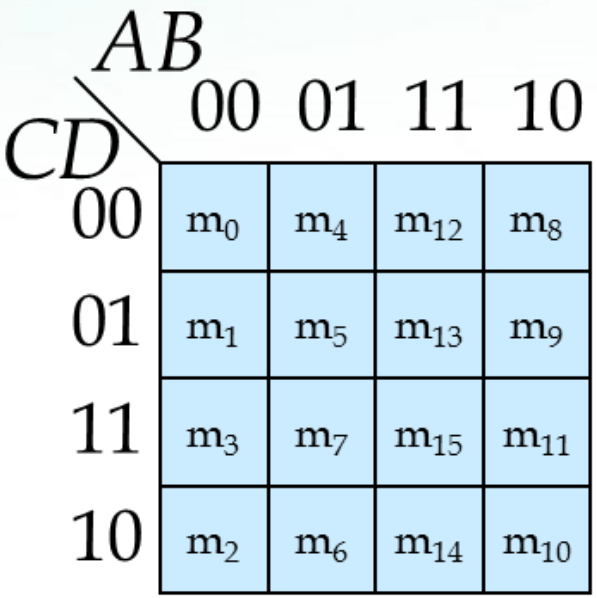
A	B	C	F
0	0	0	m_0
0	0	1	m_1
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7

The Karnaugh map (cont'd)

- **Four-variable map:**

A	B	C	D	f
0	0	0	0	m0
0	0	0	1	m1
0	0	1	0	m2
0	0	1	1	m3
0	1	0	0	m4
0	1	0	1	m5
0	1	1	0	m6
0	1	1	1	m7
1	0	0	0	m8
1	0	0	1	m9
1	0	1	0	m10
1	0	1	1	m11
1	1	0	0	m12
1	1	0	1	m13
1	1	1	0	m14
1	1	1	1	m15



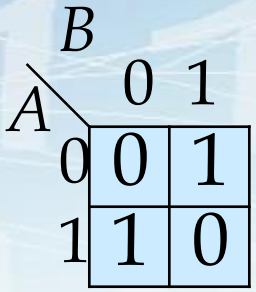


The Karnaugh map (cont'd)

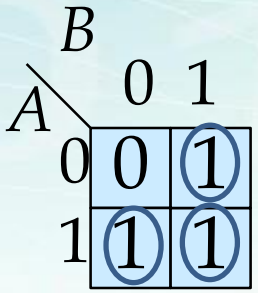
- A **K-map** is a **graphical tool** for assisting in the general **simplification procedure**
- In the **SOP form**, we can **reduce functions** by **grouping** (or **circling) 1s** in the **K-map**
- Each **circle** represents **reduction** in both **minterms** and **literals**
- Following circling, we can **deduce minimized AND-OR (or SOP) form**
- The **simplified logic expression** obtained from a **K-map** is **not always unique**; **groupings** can be made in **different ways**
- **Rules to consider:**
  - **A group (or circle) must contain only 1s and no 0s can be included in any group**
  - **Groups can overlap; i.e., same 1s can be included in more than one group**
  - **A group must be as large as possible; i.e., it must include large number of squares**
  - **The number of squares in a group must be equal to 2^n; i.e., 2, 4, 8, etc.**

The Karnaugh map (cont'd)

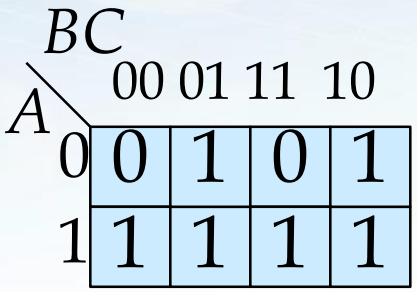
- **Examples:**



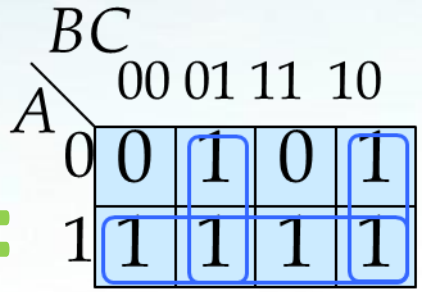
$F=AB'+A'B$



$F=AB+A'B+AB'$

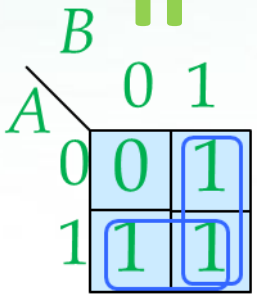


=



$F\_Simplified=A+B'C+BC'$

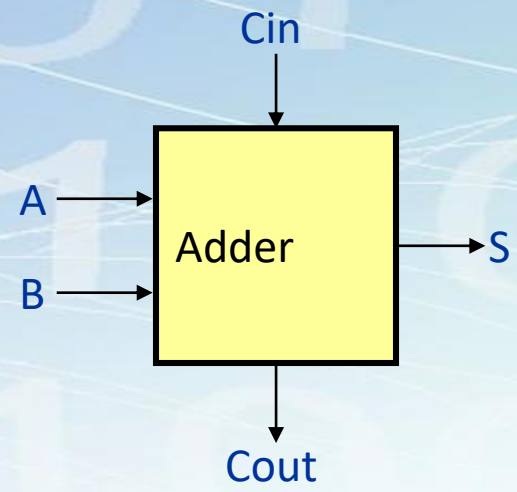
=



$F\_Simplified=A+B$

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

Application of Karnaugh Maps: The One-bit full Adder



A	B	Cin	S	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

How to use a Karnaugh Map instead of the Algebraic simplification?

S = A'B'Cin + A'BCin' + AB'Cin' + ABCin

Cout = A'BCin + A B'Cin + ABCin' + ABCin

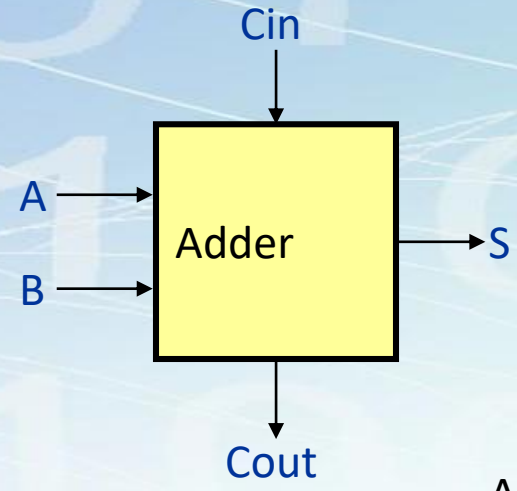
= A'BCin + ABCin + AB'Cin + ABCin + ABCin' + ABCin

= (A' + A)BCin + (B' + B)ACin + (Cin' + Cin)AB

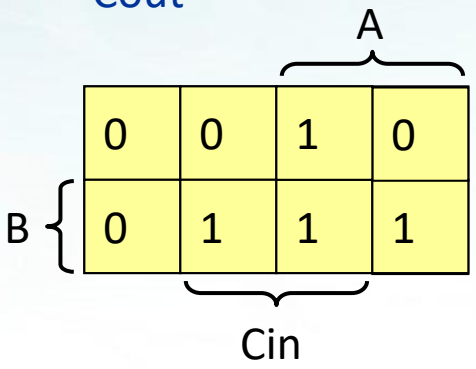
= 1·BCin + 1· ACin + 1· AB

= BCin + ACin + AB

Application of Karnaugh Maps: The One-bit Adder



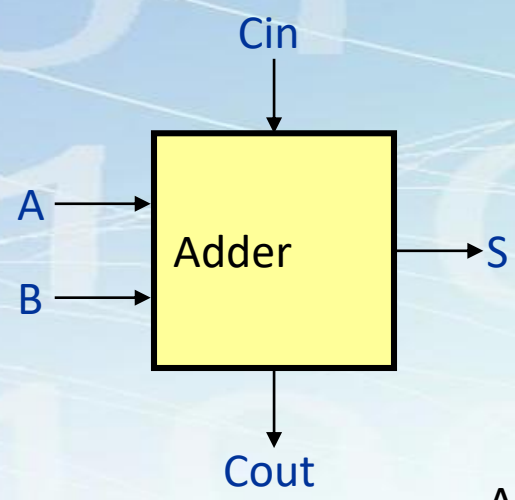
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



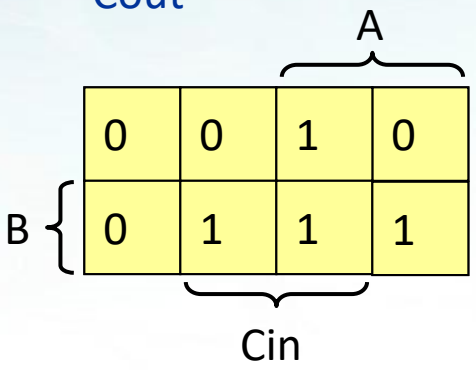
Karnaugh Map for Cout

Now we have to cover all the 1s in the Karnaugh Map using the largest rectangles and as few rectangles as we can.

Application of Karnaugh Maps: The One-bit Adder



A	B	Cin	S	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

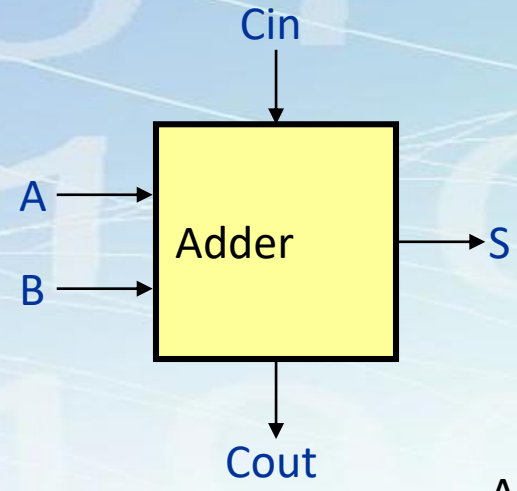


Karnaugh Map for Cout

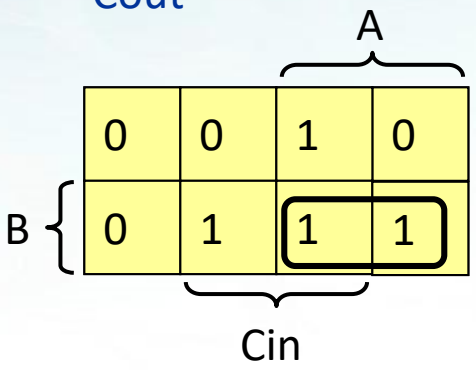
Now we have to cover all the 1s in the Karnaugh Map using the largest rectangles and as few rectangles as we can.

Cout =

Application of Karnaugh Maps: The One-bit Adder



A	B	Cin	S	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

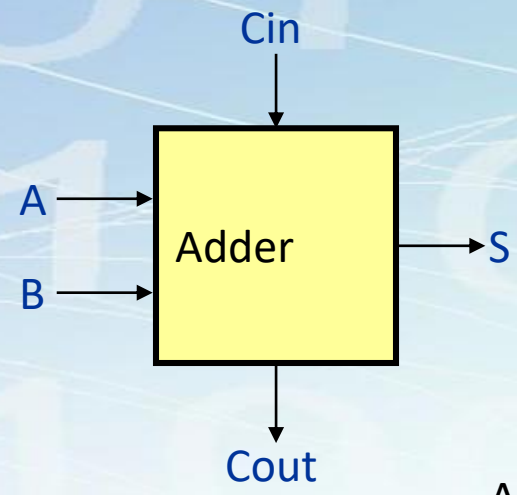


Karnaugh Map for Cout

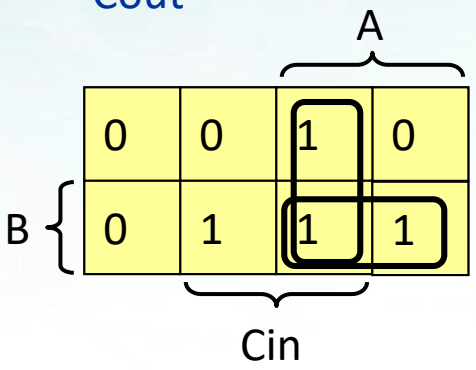
Now we have to cover all the 1s in the Karnaugh Map using the largest rectangles and as few rectangles as we can.

Cout = AB

Application of Karnaugh Maps: The One-bit Adder



A	B	Cin	S	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

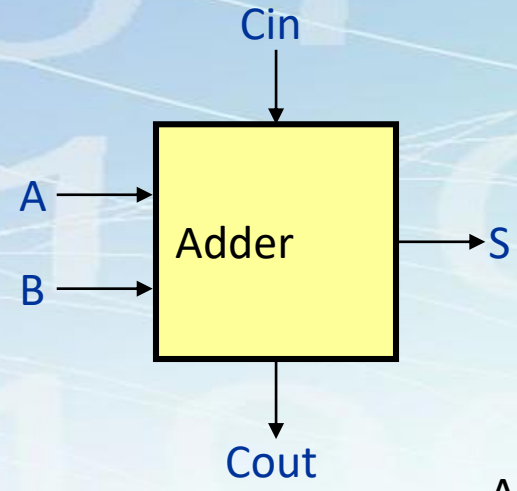


Karnaugh Map for Cout

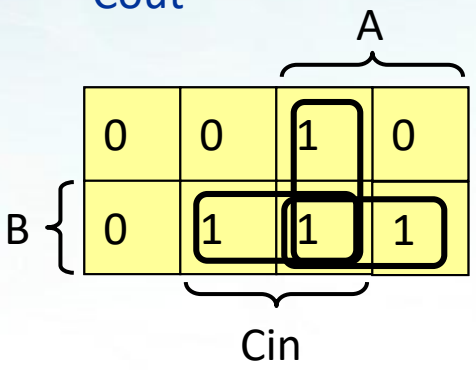
Now we have to cover all the 1s in the Karnaugh Map using the largest rectangles and as few rectangles as we can.

Cout = AB + ACin

Application of Karnaugh Maps: The One-bit Adder



A	B	Cin	S	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

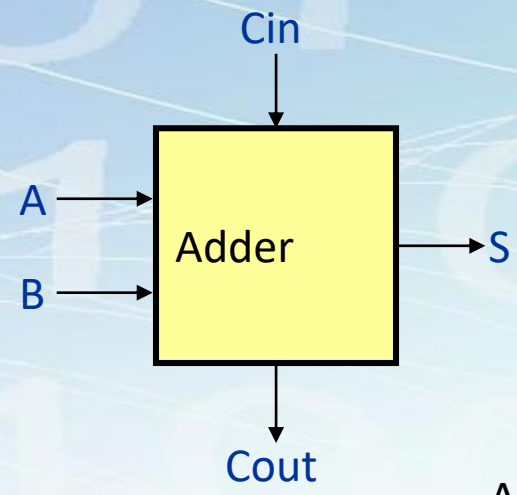


Karnaugh Map for Cout

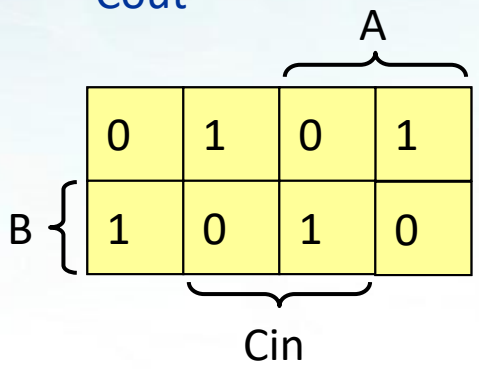
Now we have to cover all the 1s in the Karnaugh Map using the largest rectangles and as few rectangles as we can.

Cout = AB + ACin + BCin

Application of Karnaugh Maps: The One-bit Adder



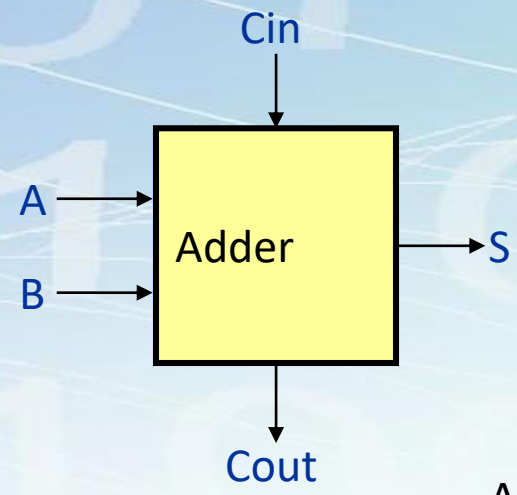
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



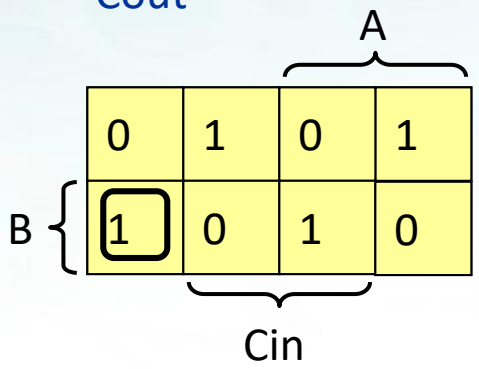
Karnaugh Map for S

S =

Application of Karnaugh Maps: The One-bit Adder



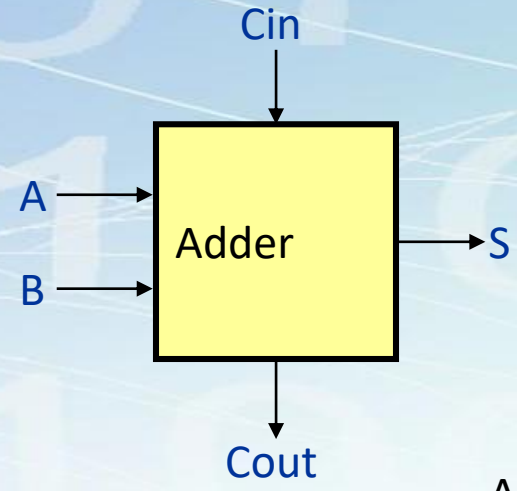
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



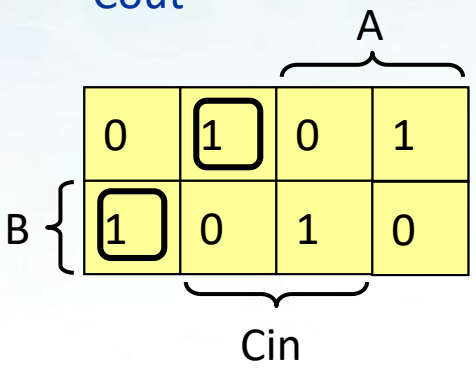
Karnaugh Map for S

S = A'BCin'

Application of Karnaugh Maps: The One-bit Adder



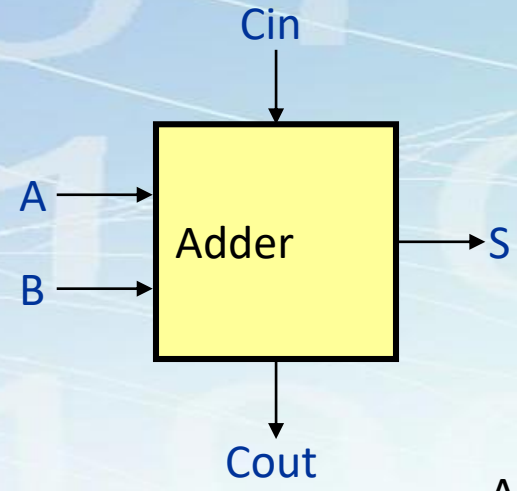
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



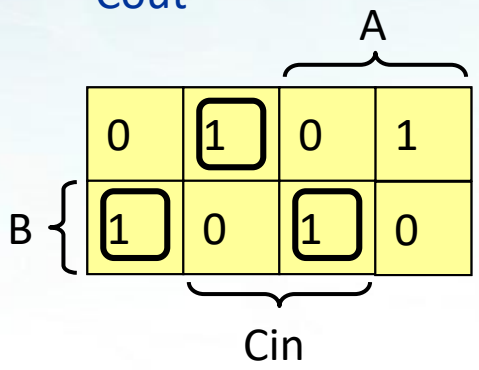
Karnaugh Map for S

S = A'BCin' + A'B'Cin

Application of Karnaugh Maps: The One-bit Adder



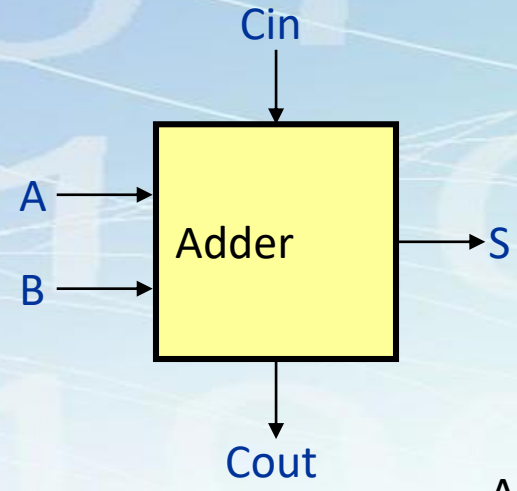
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



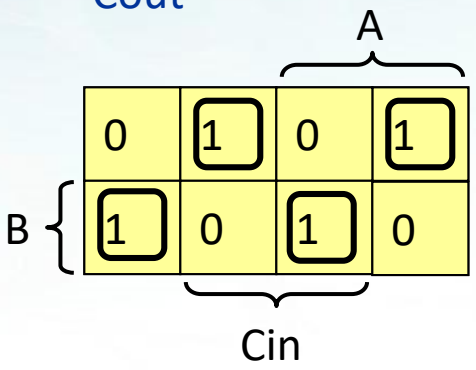
Karnaugh Map for S

S = A'BCin' + A'B'Cin + ABCin

Application of Karnaugh Maps: The One-bit Adder



A	B	Cin	S	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



Karnaugh Map for S

S = A'BCin' + A'B'Cin + ABCin + AB'Cin'