Digital Fundamentals

Computer Engineering
3rd Semester

Combinational Logic Circuits

Boolean functions & representation

1. Sum-of-products (SOP) form (Disjunctive Normal Form)

$$f(A,B,C) = (A\bar{B} + \bar{B}C)$$

2. Products-of-sums (POS) form (Conjunctive Normal Form)

$$f(A,B,C) = (\bar{A} + \bar{B})(B + C)$$

Boolean functions & representation

3. Standard sum-of-products form (Minterm)
   - Contains all the variables of the function either in complemented or uncomplemented form.

$$\begin{aligned} f(A,B,C) &= \bar{A}B + \bar{B}C \\ &= \bar{A}B(C+\bar{C}) + \bar{B}C(A+\bar{A}) \\ &= \bar{A}BC + \bar{A}B\bar{C} + A\bar{B}C + \bar{A}\bar{B}C \end{aligned}$$

Value of Minterm = 1

   - Variable appears in uncomplemented form if it has a value of 1 in the combination and appears in complemented form if it has a value of 0 in the combination
   - Sum of minterms whose value is equal to 1 is the standard sum of products form of the function.
   - Minterms are denoted as m_0, m_1, m_2, …,
   - For 3 variable function, $m_0=\bar{A}\bar{B}\bar{C}$, $m_1=\bar{A}\bar{B}C$, $m_2=\bar{A}B\bar{C}$, $m_3=\bar{A}BC$, $m_4=A\bar{B}\bar{C}$, $m_5=A\bar{B}C$, $m_6=AB\bar{C}$, and $m_7=ABC$

Boolean functions & representation

- Canonical SOP form is shown by the sum of minterms for which the function equals 1.

$$f(A,B,C) = m_1 + m_2 + m_3 + m_5$$

Or

$$f(A,B,C) = \sum_m (1,2,3,5)$$

4. Standard product-of-sum form (Maxterm)
   - Derived by considering the combinations of which f = 0
   - Each term is a sum of all the variables
   - Variable appears in uncomplemented form if it has a value of 0 in the combination and appears in complemented form if it has a value of 1 in the combination

Boolean functions & representation

$$f(A,B,C) = (\bar{A}+\bar{B})(A+B)$$
$$= (\bar{A}+\bar{B}+C\bar{C})(A+B+C\bar{C})$$
$$= (\bar{A}+\bar{B}+C)(\bar{A}+\bar{B}+\bar{C})(A+B+C)(A+B+\bar{C})$$

Value of Maxterm = 0

- A sum term which contains each of the n variables in either complemented or uncomplemented form is called a maxterm.
- Maxterms are denoted as M_0, M_1, M_2, …,

$$f(A,B,C) = M_0M_1M_6M_7$$

Or

$$f(A,B,C) = \prod_M(0,1,6,7)$$

Simplification using K-Map

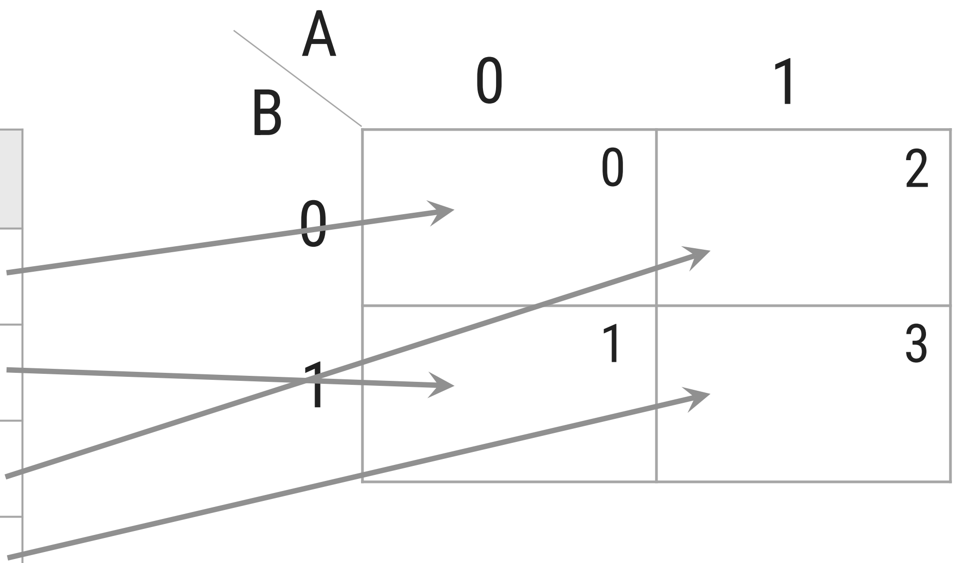
Introduction to K-Maps

- Simplification of Boolean functions leads to simpler (and usually faster) digital circuits.
- Simplifying Boolean functions using identities is time-consuming and error-prone.
- This special section presents an easy, systematic method for reducing Boolean expressions.
- A K-Map is a matrix consisting of rows and columns that represent the output values of a Boolean function.
- The output values placed in each cell are derived from the minterms of a Boolean function.
- A minterm is a product term that contains all of the function’s variables exactly once, either complemented or not complemented.

2 – Variable K-Map

- The two variables A and B have four possible combinations that can be represented by the map as follows

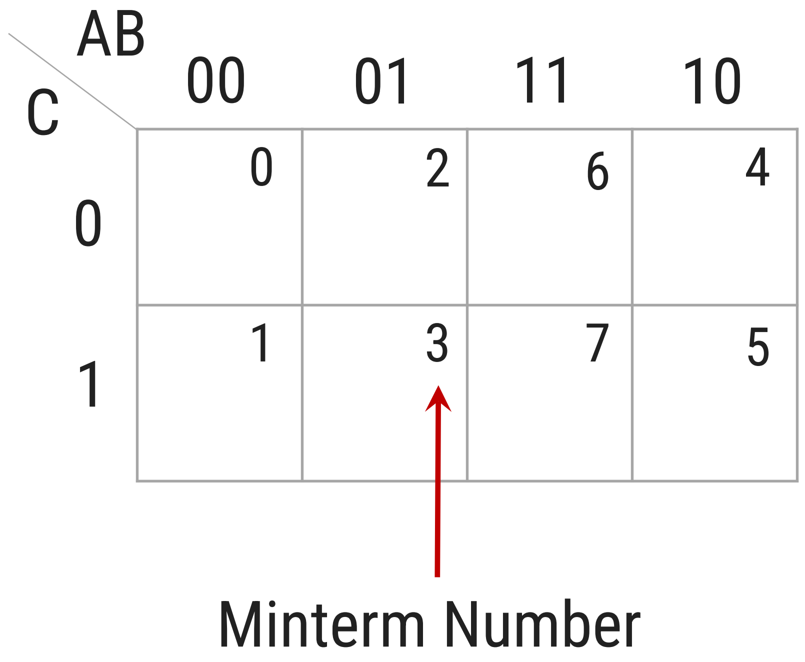
A	B	Minterm
0	0	$m_0 = A'B'$
0	1	$m_1 = A'B$
1	0	$m_2 = AB'$
1	1	$m_3 = AB$



3 – Variable K-Map

- The three variables A, B and C have eight possible combinations that can be represented by the map as follows

A	B	C	Minterm
0	0	0	m_0 = A'B'C'
0	0	1	m_1 = A'B'C
0	1	0	m_2 = A'BC'
0	1	1	m_3 = A'BC
1	0	0	m_4 = AB'C'
1	0	1	m_5 = AB'C
1	1	0	m_6 = ABC'
1	1	1	m_7 = ABC



4 – Variable K-Map

- The four variables A, B, C and D have sixteen possible combinations that can be represented by the map as follows

A	B	C	D	Minterm
0	0	0	0	$m_0 = A'B'C'D'$
0	0	0	1	$m_1 = A'B'C'D$
0	0	1	0	$m_2 = A'B'CD'$
0	0	1	1	$m_3 = A'B'CD$
0	1	0	0	$m_4 = A'BC'D'$
0	1	0	1	$m_5 = A'BC'D$
0	1	1	0	$m_6 = A'BCD'$
0	1	1	1	$m_7 = A'BCD$
1	0	0	0	$m_8 = AB'C'D'$
1	0	0	1	$m_9 = AB'C'D$
1	0	1	0	$m_{10} = AB'CD'$
1	0	1	1	$m_{11} = AB'CD$
1	1	0	0	$m_{12} = ABC'D'$
1	1	0	1	$m_{13} = ABC'D$
1	1	1	0	$m_{14} = ABCD'$
1	1	1	1	$m_{15} = ABCD$

CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

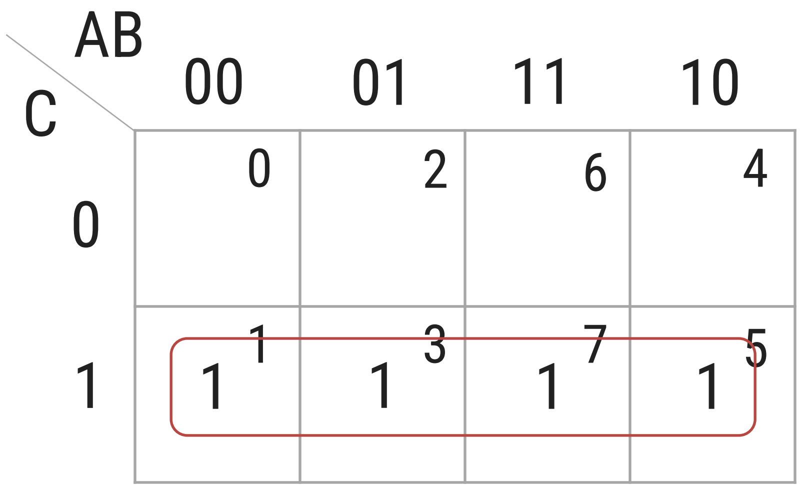
Reduce Boolean Expression

- Consider the function:

$$f(A,B,C) = A'B'C + A'BC + AB'C + ABC$$

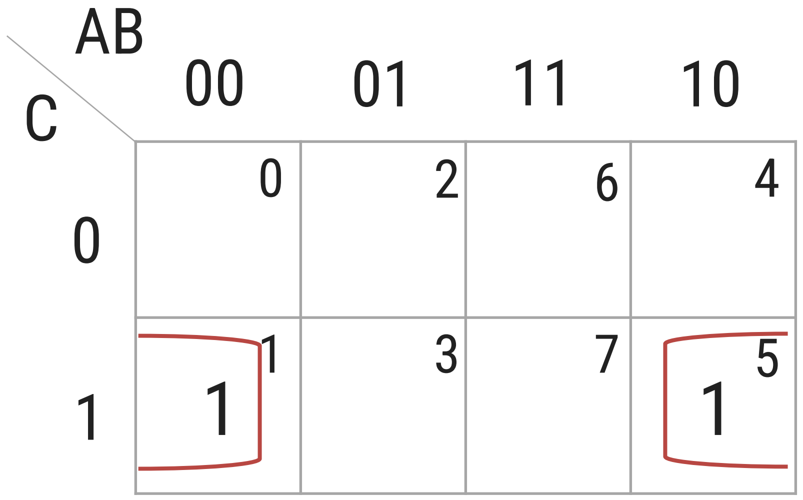
- Its K-Map is given below.
- What is the largest group of 1's that is a power of 2?
- This grouping tells us that changes in the variables *A* and *B* have no influence upon the value of the function: They are irrelevant.

$$f(A,B,C) = C$$



Examples (SOP)

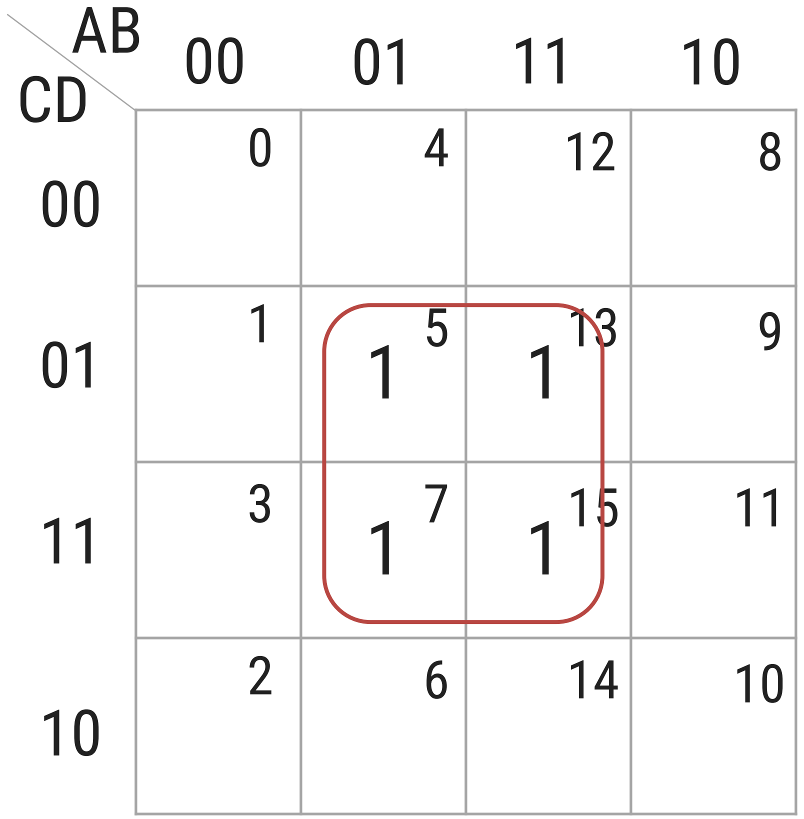
$f = AB'C + A'B'C$



C \ AB	00	01	11	10
0	^{0}	^{2}	^{6}	^{4}
1	^{1} 1	^{3}	^{7}	^{5} 1

$f = B'C$

$f = ABC'D + ABCD + A'BC'D + A'BCD$

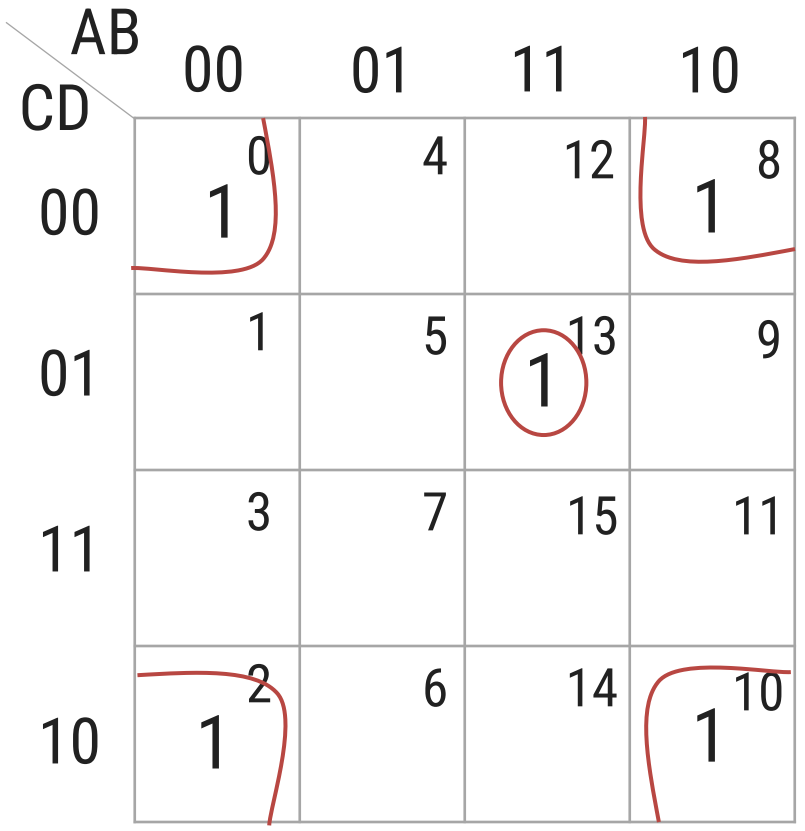


CD \ AB	00	01	11	10
00	^{0}	^{4}	^{12}	^{8}
01	^{1}	^{5} 1	^{13} 1	^{9}
11	^{3}	^{7} 1	^{15} 1	^{11}
10	^{2}	^{6}	^{14}	^{10}

$f = BD$

Examples (SOP)

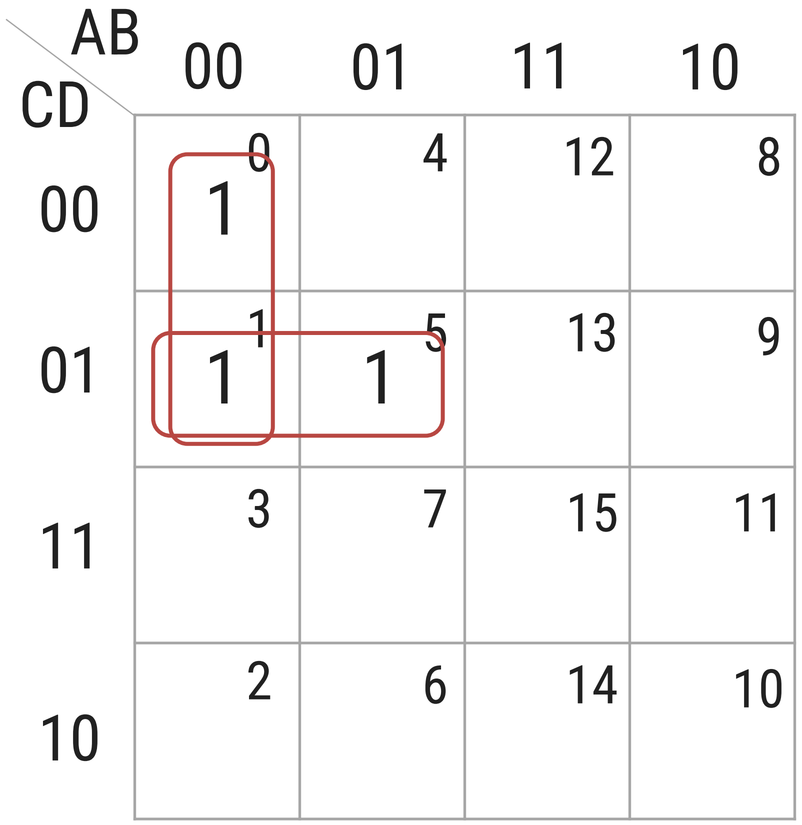
$f = ABC'D + AB'C'D' + AB'CD' + A'B'CD' + A'B'C'D'$



CD \ AB	00	01	11	10
00	0: 1	4	12	8: 1
01	1	5	13: 1	9
11	3	7	15	11
10	2: 1	6	14	10: 1

$f = ABC'D + B'D'$

$f = A'BC'D + A'B'C'D + A'B'C'D'$

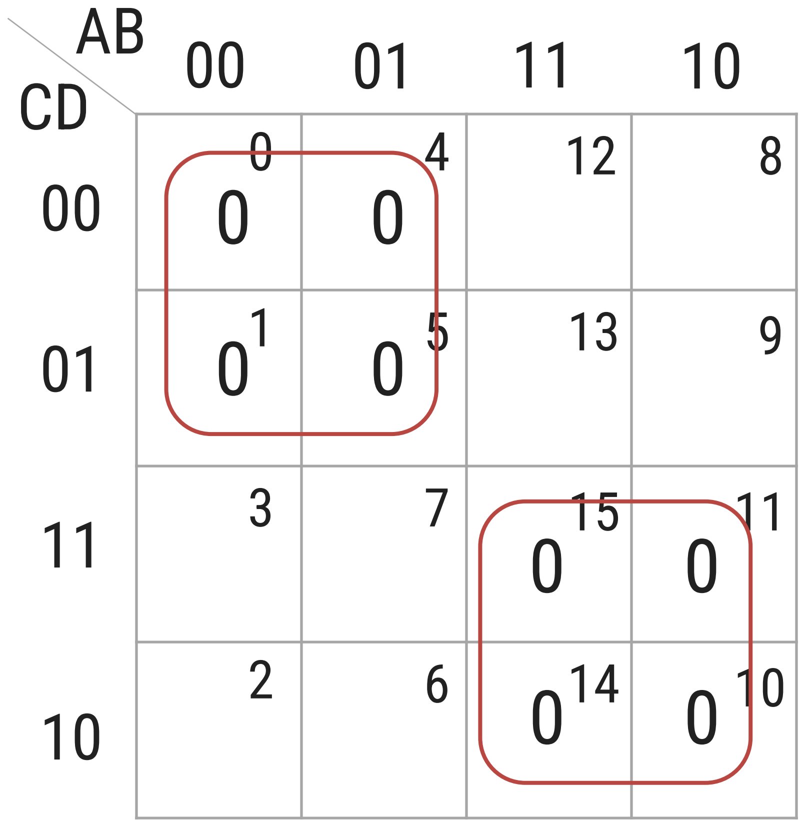


CD \ AB	00	01	11	10
00	0: 1	4	12	8
01	1: 1	5: 1	13	9
11	3	7	15	11
10	2	6	14	10

$f = A'B'C' + A'C'D$

Example (POS)

$f(A,B,C,D) = \prod_M (0,1,4,5,10,11,14,15)$

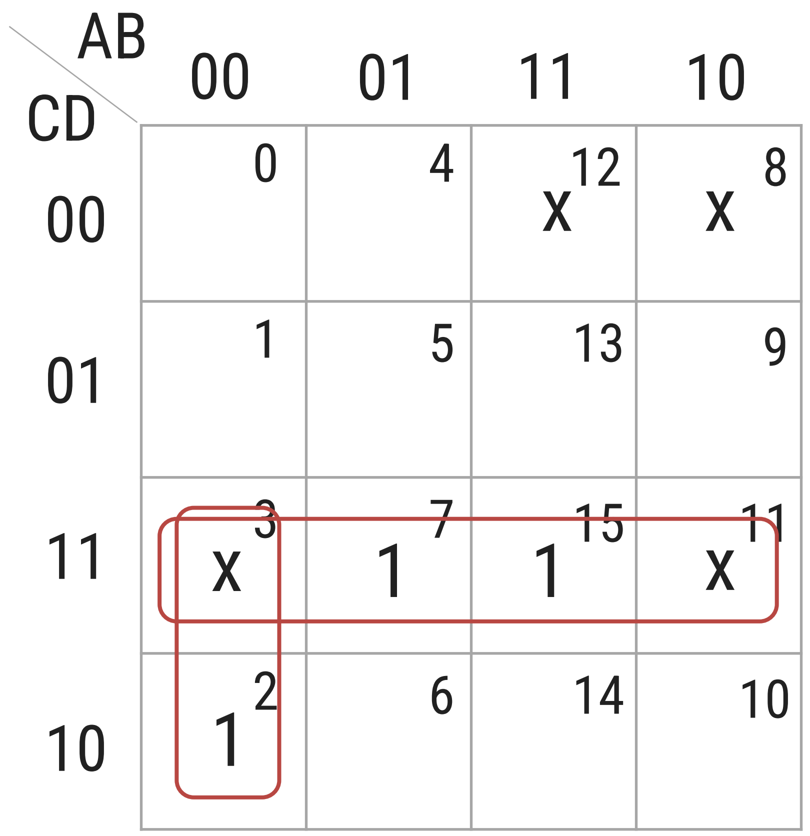


$f = (A + C)(A' + C')$

Don’t care conditions

- Suppose we are given a problem of implementing a circuit to generate a logical 1 when a 2, 7, or 15 appears on a four-variable input.
- A logical 0 should be generated when 0, 1, 4, 5, 6, 9, 10, 13 or 14 appears.
- The input conditions for the numbers 3, 8, 11 and 12 never occur in the system. This means we don’t care whether inputs generate logical 1 or logical 0.
- Don’t care combinations are denoted by ‘x’ in K-Map which can be used for the making groups.
- The above example can be represented as

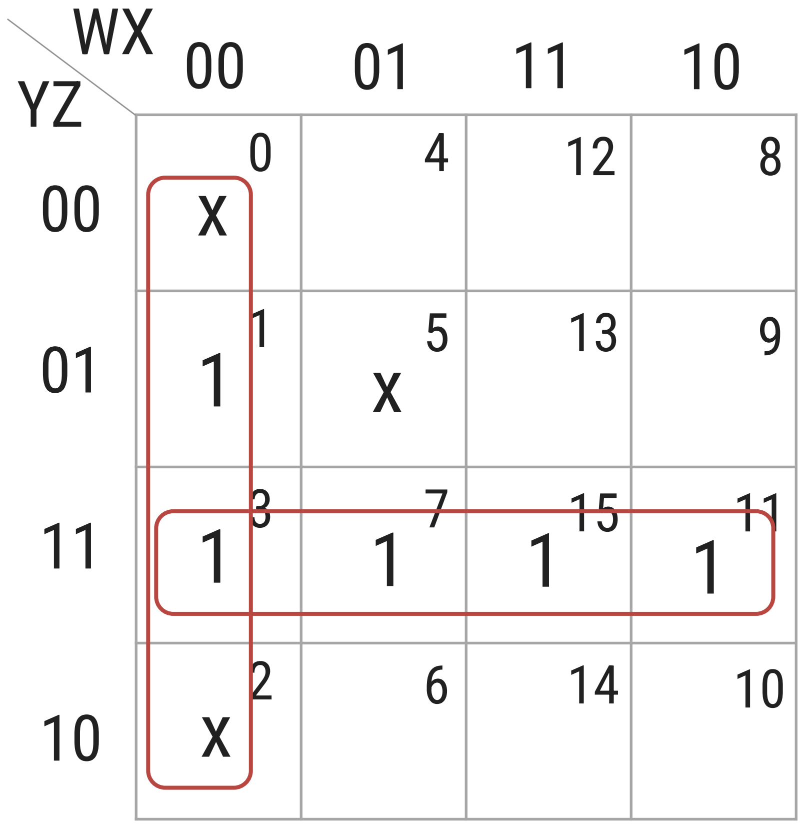
$f(A,B,C,D) = \sum_m (2,7,15) + d(3,8,11,12)$



$f = CD + A'B'C$

Example (Don't care)

$F(W,X,Y,Z) = \sum_m (1,3,7,11,15) + d(0,2,5)$



YZ \ WX	00	01	11	10
00	X (0)	(4)	(12)	(8)
01	1 (1)	X (5)	(13)	(9)
11	1 (3)	1 (7)	1 (15)	1 (11)
10	X (2)	(6)	(14)	(10)

$F = W'X' + YZ$

Variable-Entered Map (VEM)

- Variable-entered map can be used to plot an n-variable problem on n – 1 variable map
- Possible to reduce the map dimension by two or three in some cases
- Advantage of using VEM occurs in design problems involving multiplexers
- Map-entered variable for 3 variable function
- Consider the function

$$X = A'B'C' + ABC' + AB'C' + ABC$$

- Considering value of X to be function of map location with the variable C and plotting 2 variable map.

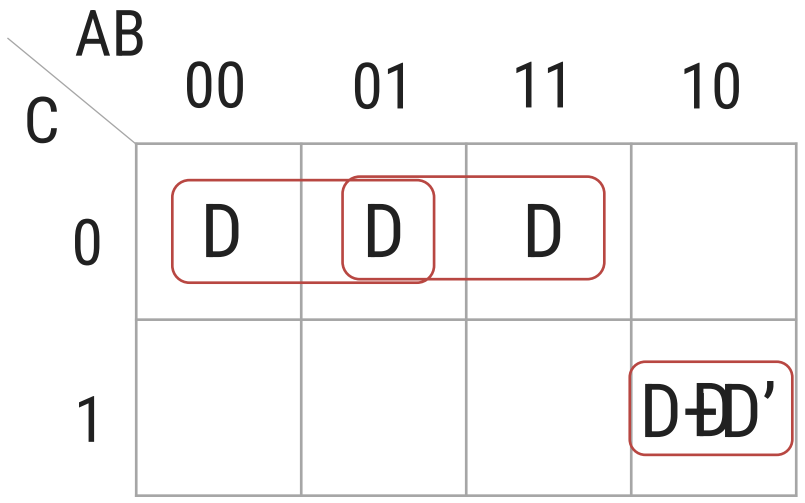
B \ A	0	1
0	X=1 if C’=1	X=1 if C’=1
1	X=0	X=1 if C’=1 or if C = 1

B \ A	0	1
0	C’	C’
1	0	C + C’

Reducing Expressions with VEM (Example)

- Choose groups of *similar terms* to cover all nonzero terms appearing in the map except "don't care" terms.
- Rest complete process is similar to K-Map

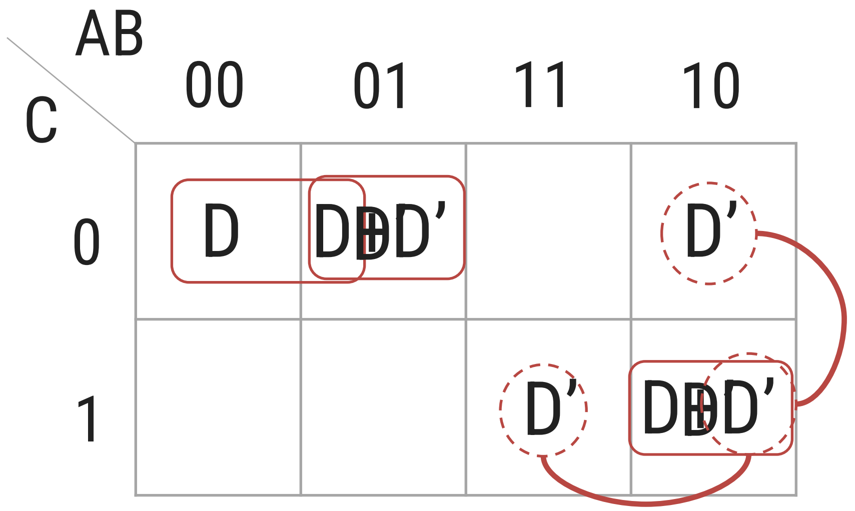
$$f = AB'CD + A'BC'D + AB'CD' + ABC'D + A'B'C'D$$



$$f = A'C'D + BC'D + AB'C$$

Reducing Expressions with VEM (Example)

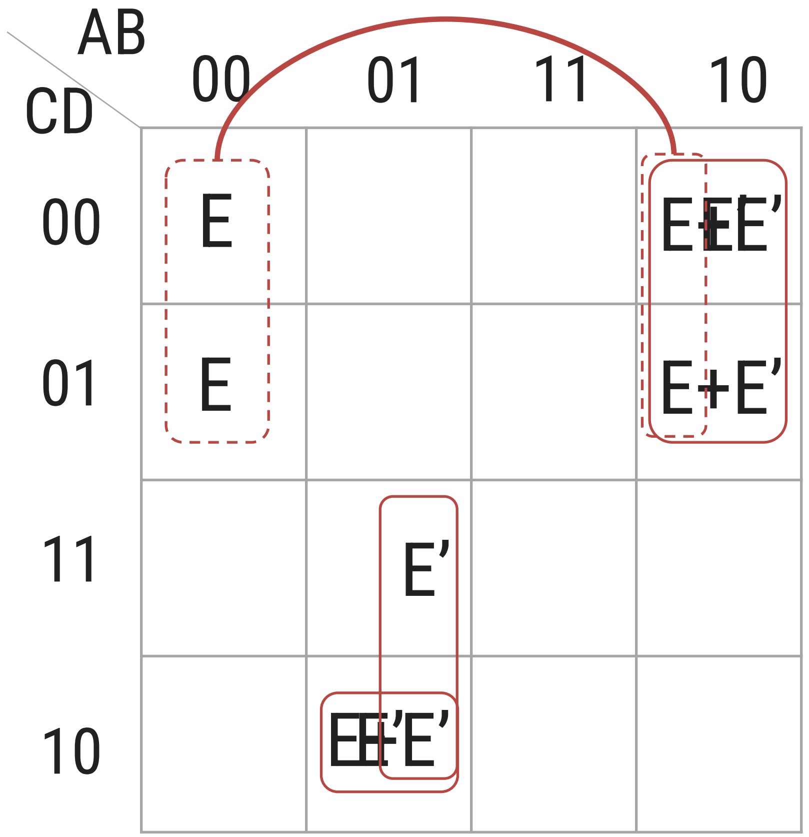
$$f = A'B'C'D + A'BC'D' + A'BC'D + AB'C'D' + AB'CD' + AB'CD + ABCD'$$



$$f = A'C'D + A'BC' + ACD' + AB'D' + AB'C$$

Reducing Expressions with VEM (Example)

$$f = A'B'C'D'E + A'B'C'DE + A'BCD'E' + A'BCD'E + AB'C'D'E' + AB'C'D'E + AB'C'D + A'BCDE'$$



$$f = B'C'E + AB'C' + A'BCD' + A'BCE'$$

Q-M Method (Tabulation Method)

Tabulation Method (Quine-McCluskey Method)

- Fundamental idea for tabulation method is combining theorem.

$$PA + PA' = P$$

- The above theorem is applied repeatedly to all adjacent pairs of terms which results in prime implicants.
- Consider the minimization of the expression

$$\begin{aligned}\textstyle\sum_m(0,1,4,5) &= A'B'C' + A'B'C + AB'C' + AB'C \\ &= A'B'(C + C') + AB'(C + C') \\ &= A'B' + AB' \\ &= B'(A + A') \\ &= B'\end{aligned}$$

Procedure for minimization using Tabulation Method

1. List all the minterms.
2. Arrange all minterms in groups of the same number of 1s in their binary representation in column 1. Start with the least number of 1s group and continue with groups of increasing number of 1s.
3. Compare each term of the lowest index group with every term in the succeeding group. Whenever possible, combine the two terms being compared by means of the combining theorem. Two terms from adjacent groups are combinable, if their binary representations differ by just a single digit in the same position; the combined terms consist of the original fixed representation with the differing one replaced by a dash (-). Place a check mark (√) next to every term, which has been combined with at least one term and write the combined terms in column 2. Repeat this by comparing each term in a group of index i with every term in the group of index i + 1, until all possible applications of the combining theorem have been exhausted.

Procedure for minimization using Tabulation Method

4. Compare the terms generated in step 3 in the same fashion; combine two terms which differ by only a single 1 and whose dashes are in the same position to generate a new term. Two terms with dashes in different positions cannot be combined. Write the new terms in column 3 and put a check mark next to each term which has been combined in column 2. Continue the process with terms in columns 3, 4 etc. until no further combinations are possible. The remaining *unchecked terms* constitute the *set of prime implicants* of the expression.
5. List all the prime implicants and draw the *prime implicant chart*. (The don't cares if any should not appear in the prime implicant chart).
6. Obtain the *essential prime implicants* and write the minimal expression.

Tabulation Method (Example)

$$f = \sum_m (0,1,6,7,8,9,13,14,15)$$

Step – 1: List all minterms

Minterms	Binary Designation
0	0 0 0 0
1	0 0 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

Step – 2: Arrange all minterms in groups of same number of 1s

Column 1		
Index	Minterms	Binary Designation
Index 0	0	0 0 0 0
Index 1	1 8	0 0 0 1 1 0 0 0
Index 2	6 9	0 1 1 0 1 0 0 1

Index 3	7
	13
	14
		0 1 1 1
		1 1 0 1
		1 1 1 0

Index 4	15	1 1 1 1

Tabulation Method (Example) Cont..

Step – 3: Compare each term of the lowest index group with every term in the succeeding group till no change

Column 1		
Index	Minterms	Binary Designation
Index 0	0	$\underline{0}\,0\,0\,\underline{0}$ √
Index 1	1 8	$\underline{0}\,0\,0\,\underline{1}$ √ $\underline{1}\,0\,0\,\underline{0}$ √
Index 2	6 9	$\underline{0}\,1\,1\,\underline{0}$ √ $\underline{1}\,\underline{0}\,0\,\underline{1}$ √
Index 3	7 13 14	$\underline{0}\,1\,1\,\underline{1}$ √ $1\,\underline{1\,0}\,1$ √ $\underline{1}\,1\,1\,\underline{0}$ √
Index 4	15	$\underline{1}\,1\,\underline{1\,1}$ √

Column 2	
Pairs	A B C D

Tabulation Method (Example) Cont..

Step – 4: Compare the terms generated in step 3 in the same fashion until no further combinations are possible

Pairs	A B C D	
0, 1	0 0 0 _	√
0, 8	_ 0 0 0	√
1, 9	_ 0 0 1	√
8, 9	1 0 0 _	√
6, 7	0 1 1 _	√
6, 14	_ 1 1 0	√
9,13	1 _ 0 1	S
7, 15	_ 1 1 1	√
13, 15	1 1 _ 1	R
14, 15	1 1 1 _	√

Quads	A B C D
	Q
	P

Tabulation Method (Example) Cont..

Step – 5: List all prime implicants and draw prime implicants chart

Prime Implicants : P(BC), Q(B’C’), R(ABD), S(AC’D)

PIs									

Step – 6: Obtain essential prime implicants and minimal expression

Essential Prime Implicants : P(BC), Q(B’C’)

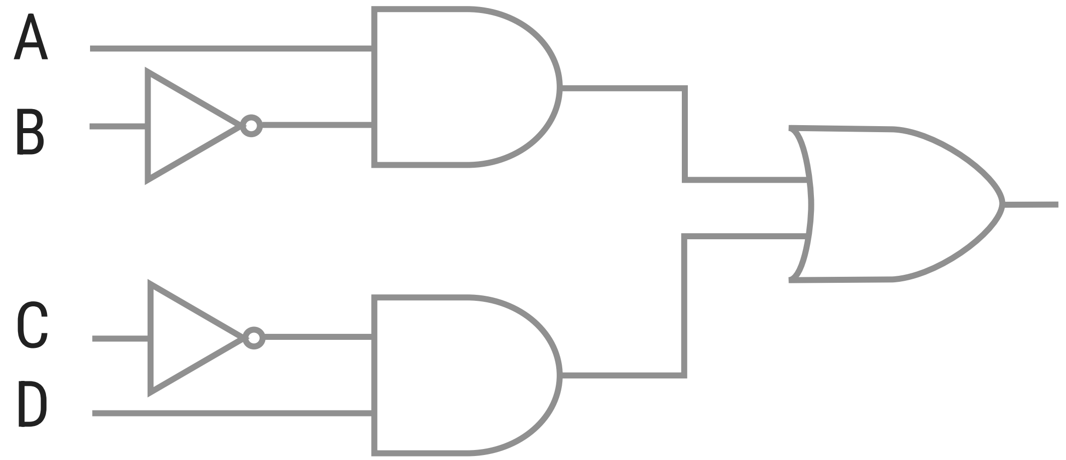
Minimal expression : P + Q + R = BC + B’C’ + ABD

OR P + Q + S = BC + B’C’ + AC’D

As minterm 13 is covered by R and S

Realizing Logic Function with Gates

- Implement AB’ + C’D using AND, OR & Invert Gates

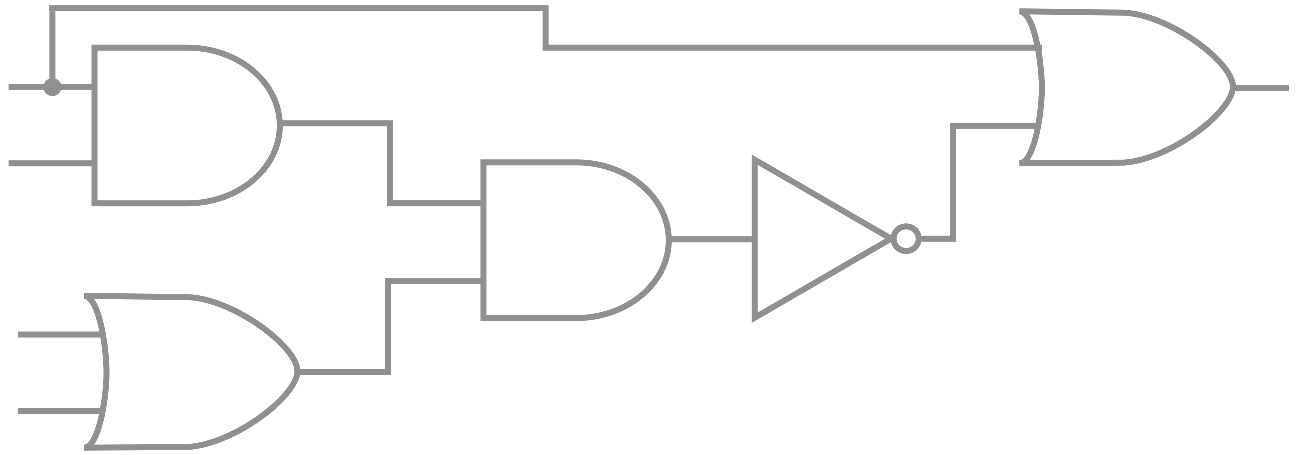


Converting AND/OR/Invert Logic to NAND/NOR Logic

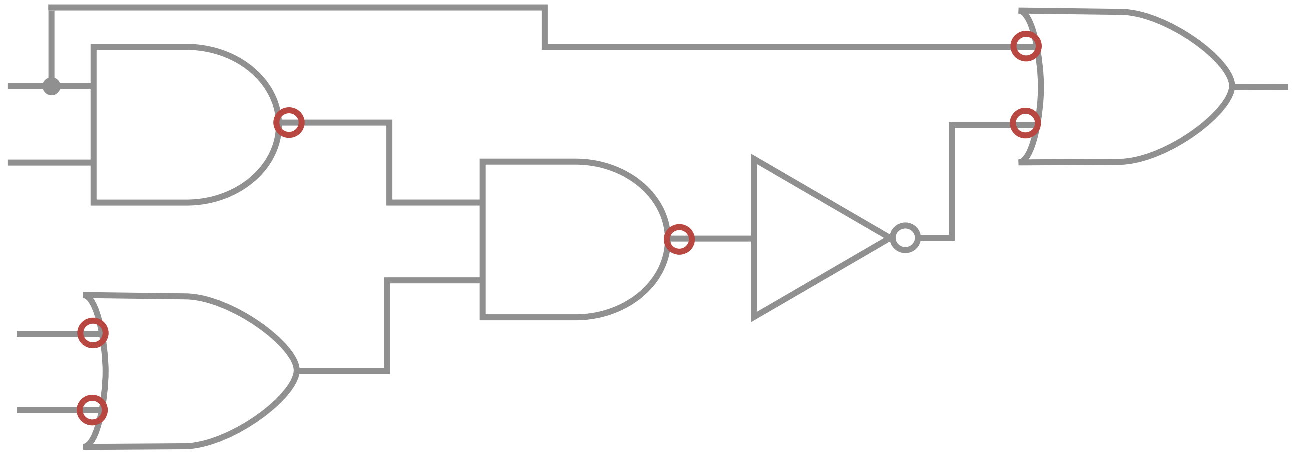
1. Draw the circuit in AOI logic.
2. If NAND hardware is chosen, add a circle at the output of each AND gate and at the inputs to all the OR gates.
3. If NOR hardware is chosen, add a circle at the output of each OR gate and at the inputs to all the AND gates.
4. Add or subtract an inverter on each line that received a circle in steps 2 or 3 so that the polarity of signals on those lines remains unchanged from that of the original diagram.
5. Replace bubbled OR by NAND and bubbled AND by NOR.
6. Eliminate double inversions.

Converting AND/OR/Invert Logic to NAND/NOR Logic (Example)

- Implement the following AOI logic using NAND logic

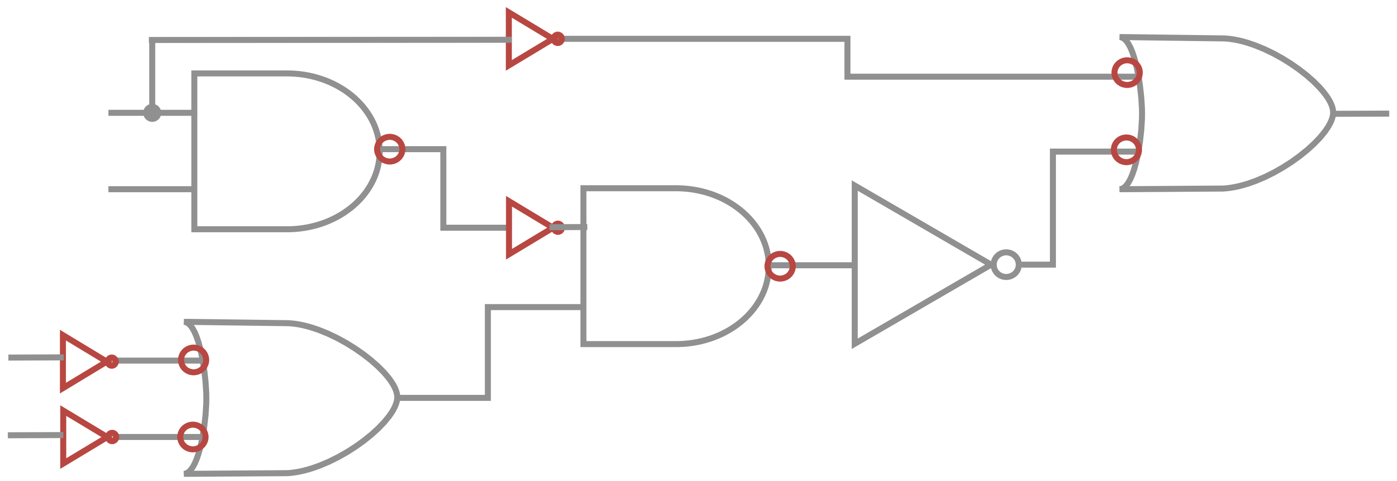


Put a circle at the output of each AND gate and at the inputs to all OR gates



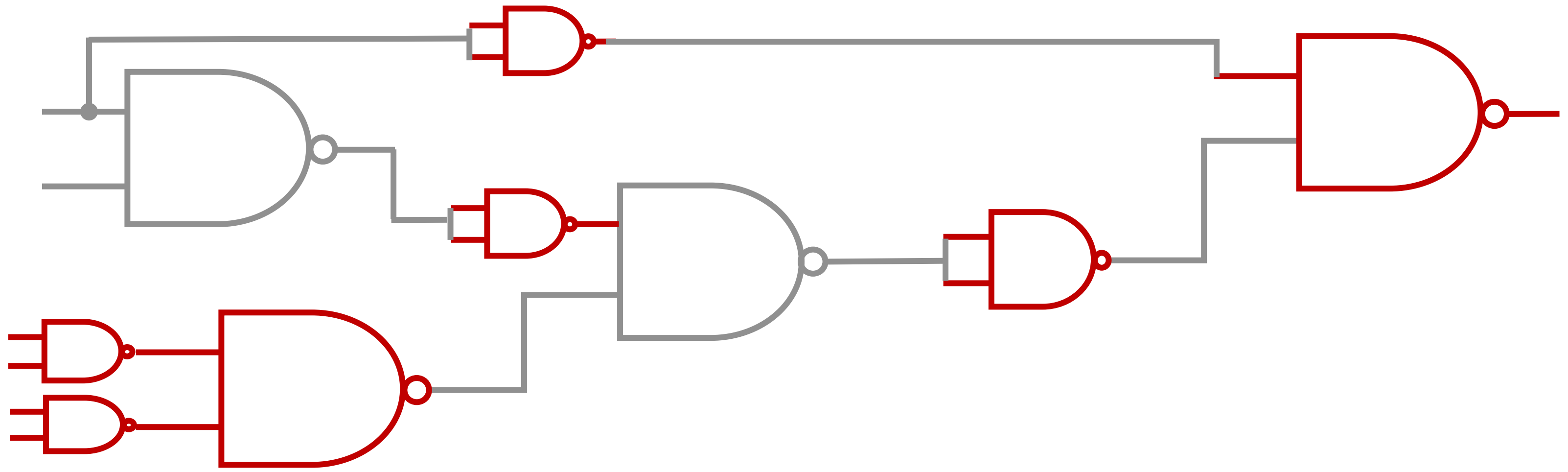
Converting AND/OR/Invert Logic to NAND/NOR Logic (Example)

Add an inverter to each of the lines that received only one circle at input so that polarity remains unchanged.



Converting AND/OR/Invert Logic to NAND/NOR Logic (Example)

Replace bubbled OR gates and NOT gates by NAND gates.

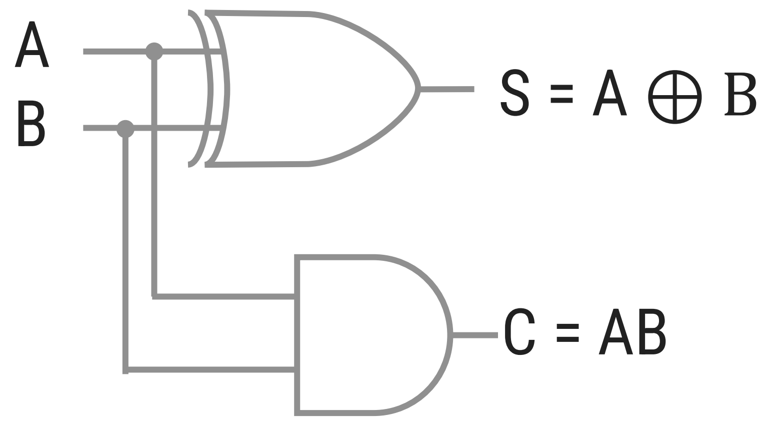


Adders, Subtractors

Half Adder

- A combinational circuit which adds two one-bit binary numbers is called a half-adder.

Inputs		**Outputs**	
A	**B**	**Sum**	**Carry**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



- The sum column resembles like an output of the XOR gate.
- The carry column resembles like an output of the AND gate.
- Limitations:
  - ↳ In multi-digit addition we have to add two bits along with the carry of previous digit addition. Such addition requires addition of 3 bits. This is not possible in half-adders.

Full Adder

- The full-adder adds the bits A and B and the carry from the previous column called carry-in C_{in} and outputs the sum bit S and the carry bit called carry-out C_{out}.

Inputs			Outputs	
A	**B**	**C_{in}**	**S**	**C_{out}**
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

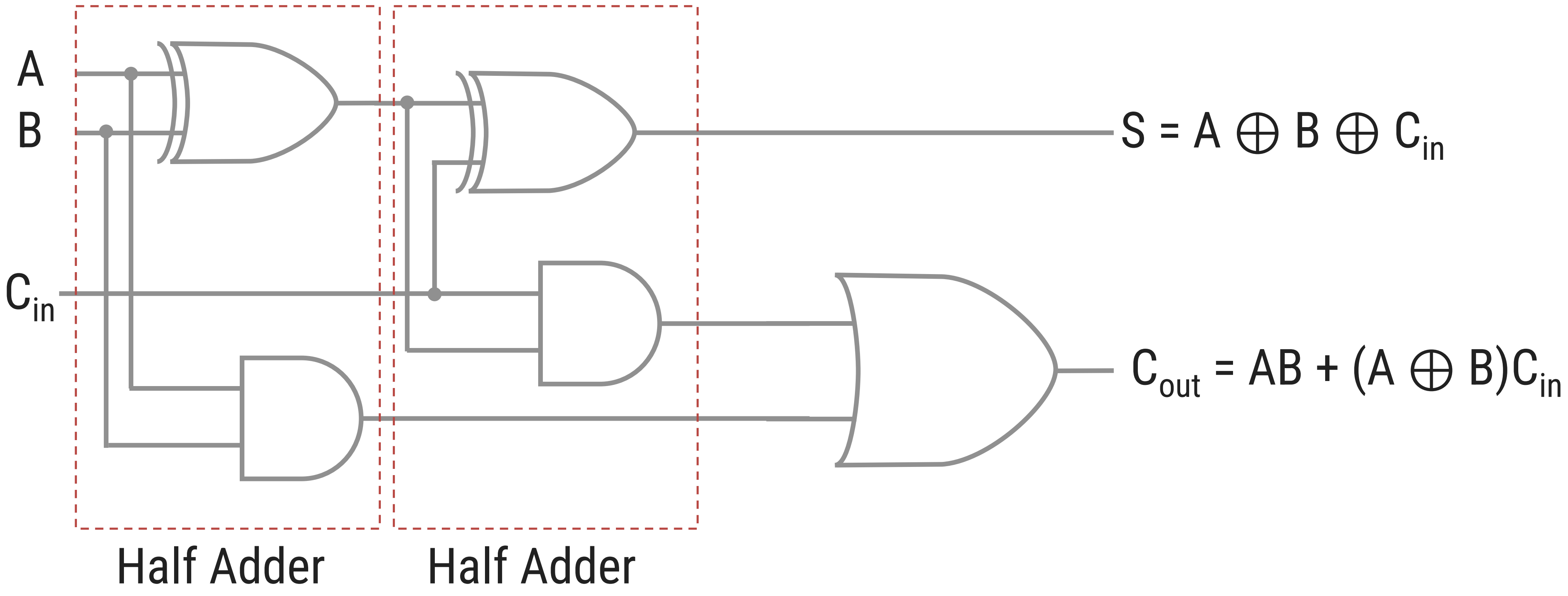
$$\begin{aligned} S &= A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in} \\ &= (AB' + A'B)C_{in}' + (AB + A'B')C_{in} \\ &= (A \oplus B)C_{in}' + (A \oplus B)'C_{in} \\ &= A \oplus B \oplus C_{in} \end{aligned}$$

$$\begin{aligned} C_{out} &= A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in} \\ &= AB + (A \oplus B)C_{in} \end{aligned}$$

Full Adder – Logical Diagram

$S = A \oplus B \oplus C_{in}$

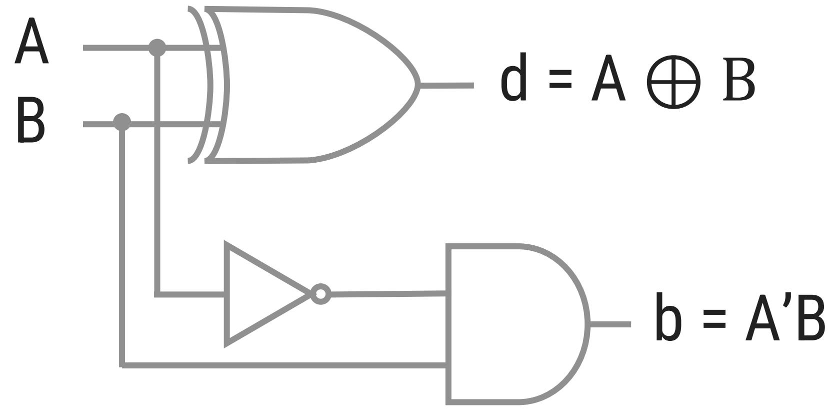
$C_{out} = AB + (A \oplus B)C_{in}$


A
B
C_{in}
$S = A \oplus B \oplus C_{in}$
$C_{out} = AB + (A \oplus B)C_{in}$
Half Adder
Half Adder

Half Subtractor

- Subtracts one bit from the other and produces the difference.
- Other output is to specify if 1 is borrowed.

Inputs		Outputs	
A	**B**	**d**	**b**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



- Limitation:
  - Half subtractor can be used only for LSB subtraction.

Full Subtractor

- Full subtractor is a combinational circuit with 3 inputs (A, B, b_i)
- Subtraction = A – B – b_i

Inputs			Outputs	
A	B	b_i	d	b
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

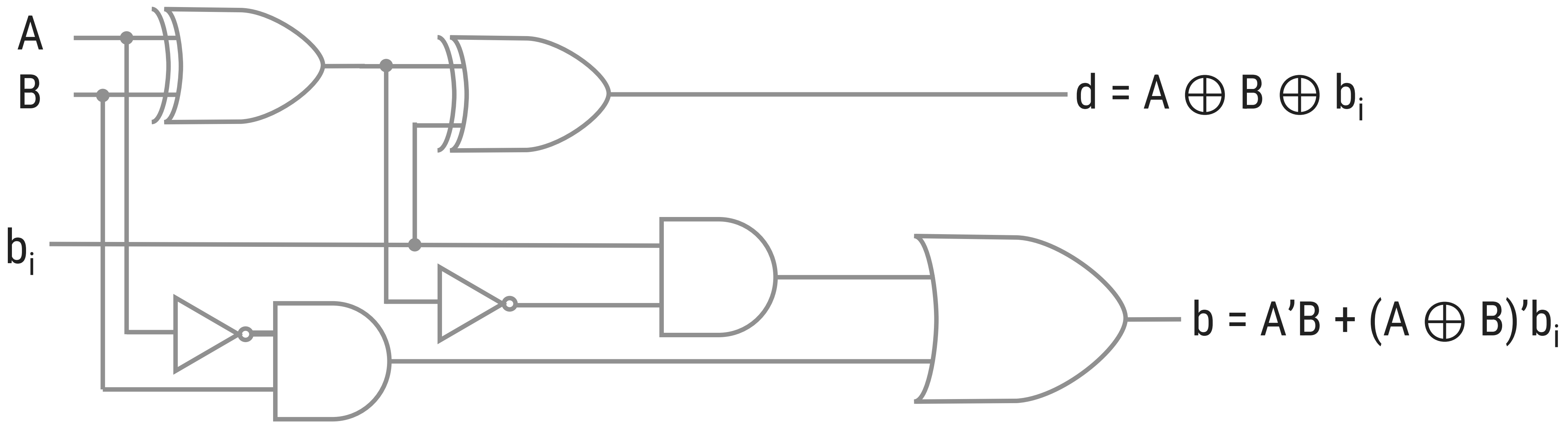
$$\begin{aligned} d &= A'B'b_i + A'Bb_i' + AB'b_i' + ABb_i \\ &= (AB' + A'B)b_i' + (AB + A'B')b_i \\ &= (A \oplus B)b_i' + (A \oplus B)'b_i \\ &= A \oplus B \oplus b_i \end{aligned}$$

$$\begin{aligned} b &= A'B'b_i + A'Bb_i' + A'Bb_i + ABb_i \\ &= A'B(b_i + b_i') + (AB + A'B')b_i \\ &= A'B + (A \oplus B)'b_i \end{aligned}$$

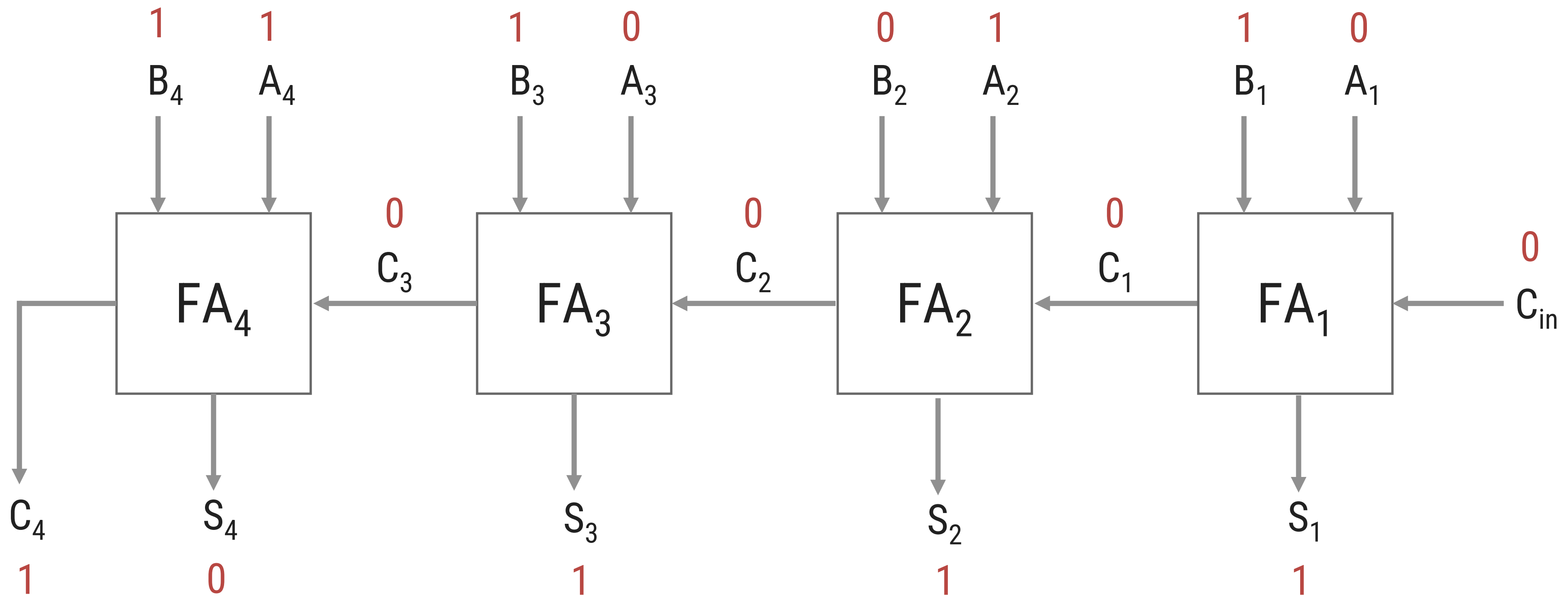
Full Subtractor – Logical Diagram

$d = A \oplus B \oplus b_i$

$b = A'B + (A \oplus B)'b_i$



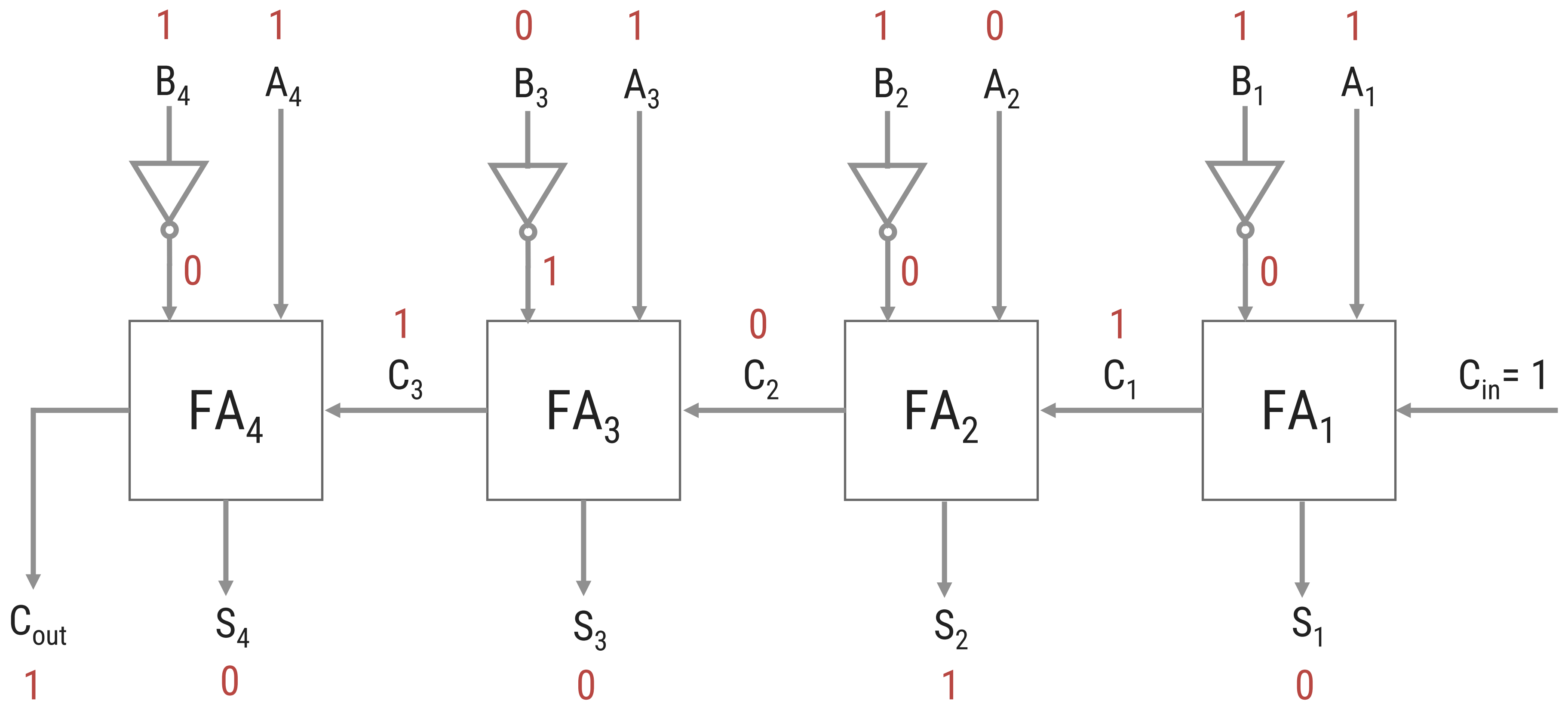
Binary Parallel Adder



Example: A = 1010, B = 1101

$A_4\ A_3\ A_2\ A_1 = 1010$
$B_4\ B_3\ B_2\ B_1 = 1101$

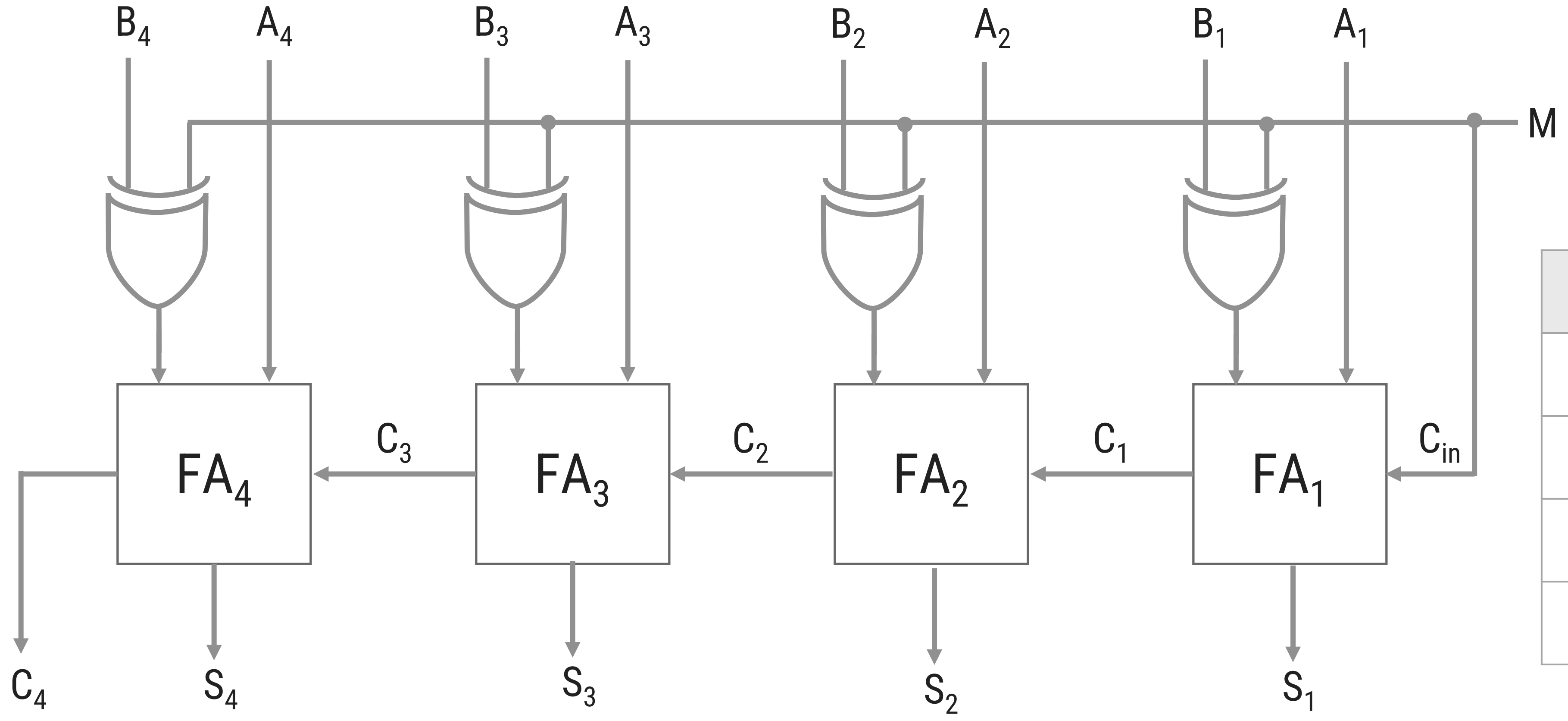
Binary Parallel Subtractor



Example: A = 1101, B = 1011

$A_4\ A_3\ A_2\ A_1 = 1101$
$B_4\ B_3\ B_2\ B_1 = 1011$

Binary Adder-Subtractor

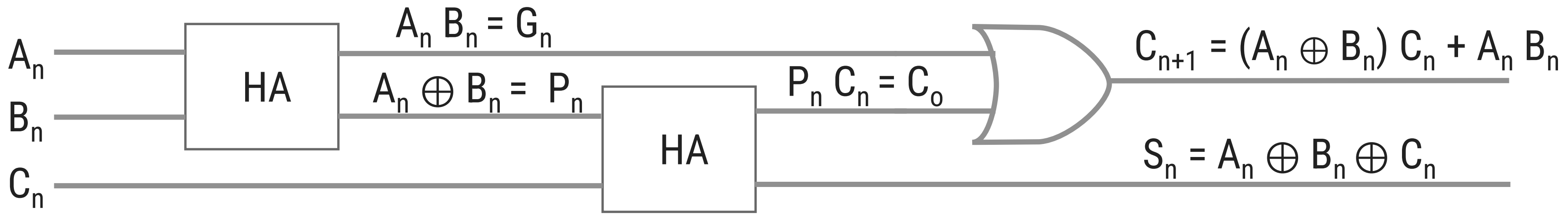


M	B_i	O/P
0	0	0
0	1	1
1	0	1
1	1	0

When M = 0, Circuit is an adder.
When M = 1, Circuit becomes a subtractor.

Look Ahead Carry Adder

- Speeds up the process by eliminating the ripple carry delay.
- The method of speeding up the addition process is based on: carry generate and carry propagate functions.
- Consider one full adder stage; say the nth stage of a parallel adder shown in below Figure.


A_n
B_n
C_n
HA
$A_n B_n = G_n$
$A_n \oplus B_n = P_n$
HA
$P_n C_n = C_o$
$C_{n+1} = (A_n \oplus B_n) C_n + A_n B_n$
$S_n = A_n \oplus B_n \oplus C_n$

Look Ahead Carry Adder

- $S_n = P_n \oplus C_n$ where $P_n = A_n \oplus B_n$
- $C_{n+1} = G_n + P_n C_n$ where $G_n = A_n B_n$
- Possible to express the output carry of a higher significant stage in terms of applied input variables A, B and carry-in to the LSB adder.
- Based on these, the expressions for the carry-outs of various full adders are as follows:

$$C_2 = G_1 + P_1 C_1$$

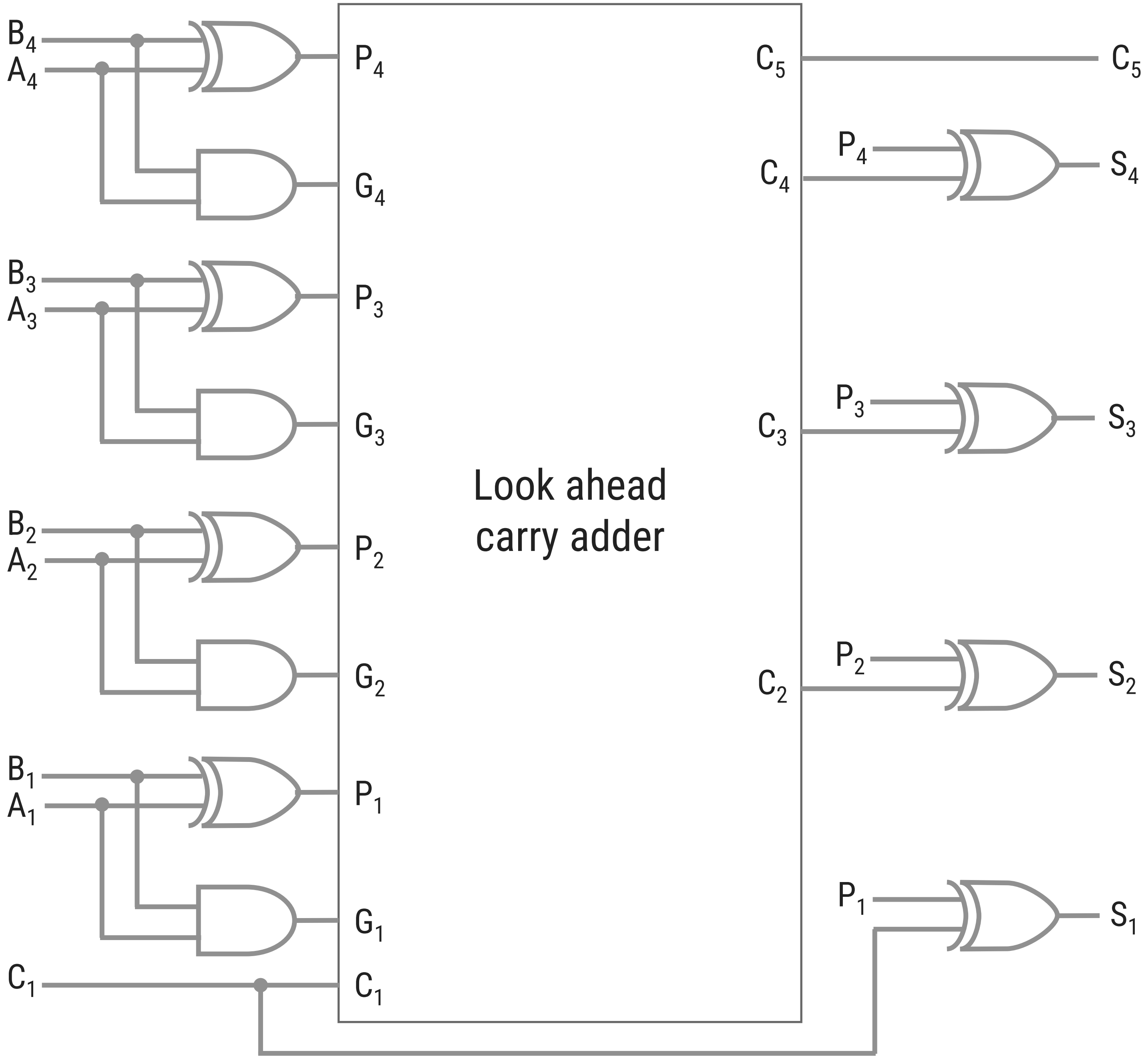
$$C_3 = G_2 + P_2 C_2$$

$$= G_2 + P_2 (G_1 + P_1 C_1)$$

$$= G_2 + P_2 G_1 + P_2 P_1 C_1$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$

$$C_5 = G_4 + P_4 C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_1$$



B4
A4
P4
G4
B3
A3
P3
G3
B2
A2
P2
G2
B1
A1
P1
G1
C1
C1
Look ahead carry adder
C5
C5
P4
C4
S4
P3
C3
S3
P2
C2
S2
P1
S1

Combinational Circuits

Binary to Gray Code Converter

$$G_4 = \sum m(8,9,10,11,12,13,14,15)$$

$$G_3 = \sum m(4,5,6,7,8,9,10,11)$$

$$G_2 = \sum m(2,3,4,5,10,11,12,13)$$

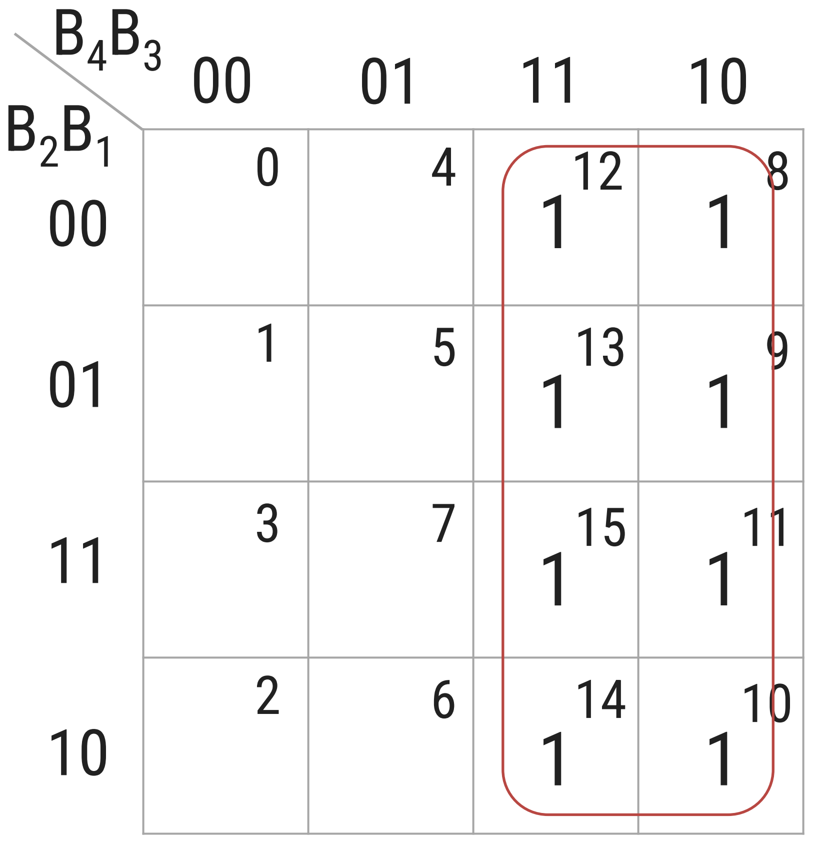
$$G_1 = \sum m(1,2,5,6,9,10,13,14)$$

4-bit Binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Binary to Gray Code Converter

- K-Map for G_4, G_3, G_2, G_1 function and their minimization are as follows:

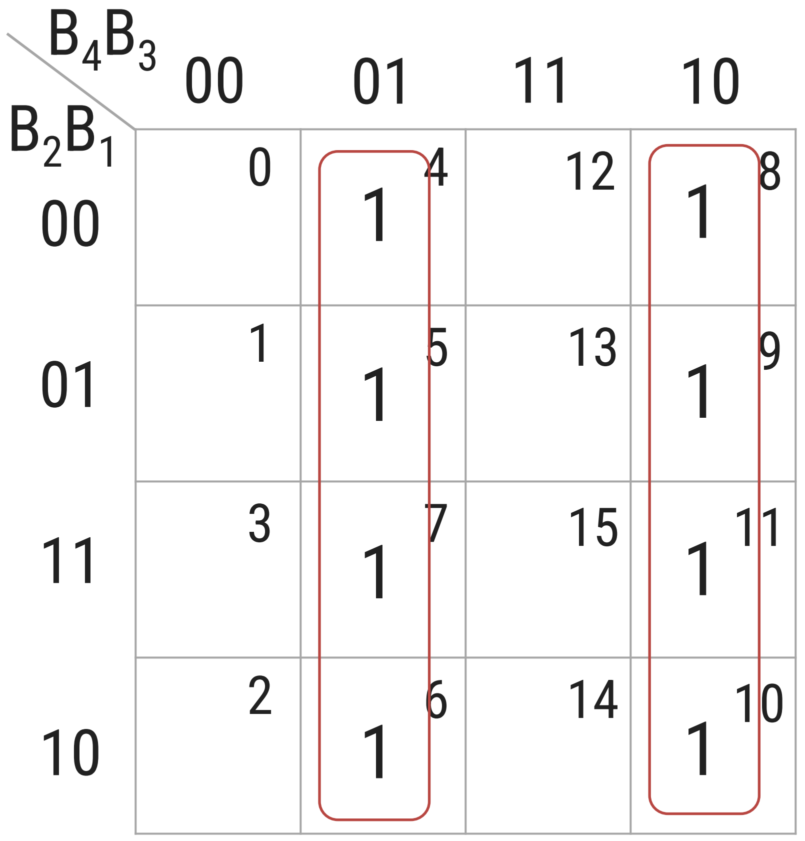
$$G_4 = \sum m(8,9,10,11,12,13,14,15)$$



$B_2B_1 \backslash B_4B_3$	00	01	11	10
00	0	4	12: 1	8: 1
01	1	5	13: 1	9: 1
11	3	7	15: 1	11: 1
10	2	6	14: 1	10: 1

$G_4 = B_4$

$$G_3 = \sum m(4,5,6,7,8,9,10,11)$$



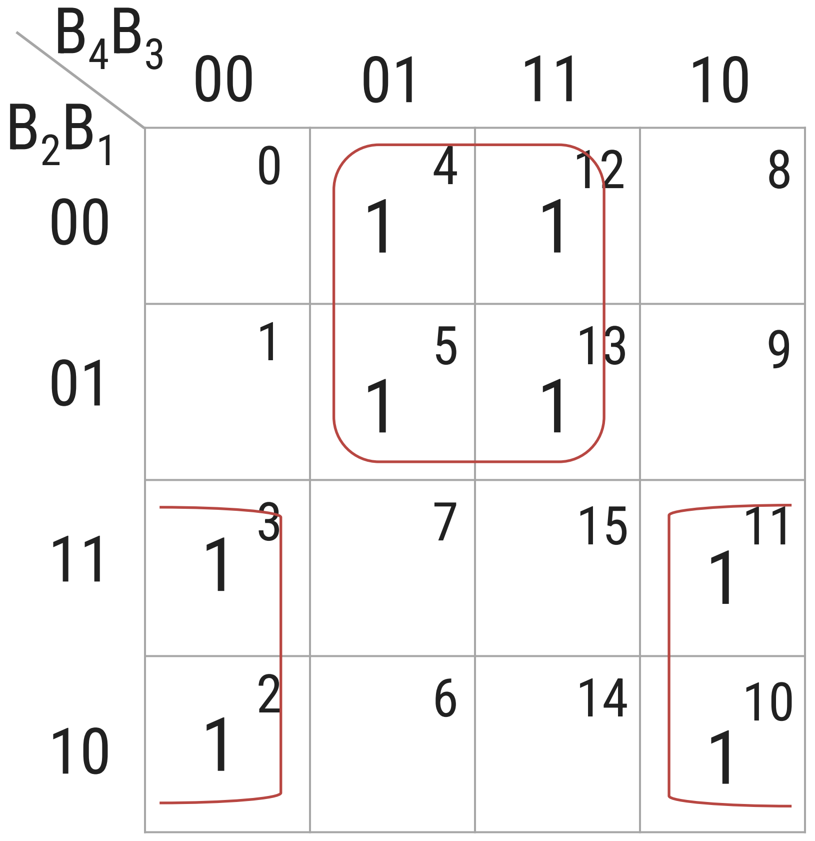
$B_2B_1 \backslash B_4B_3$	00	01	11	10
00	0	4: 1	12	8: 1
01	1	5: 1	13	9: 1
11	3	7: 1	15	11: 1
10	2	6: 1	14	10: 1

$G_3 = B_4'B_3 + B_4B_3' = B_4 \oplus B_3$

Binary to Gray Code Converter

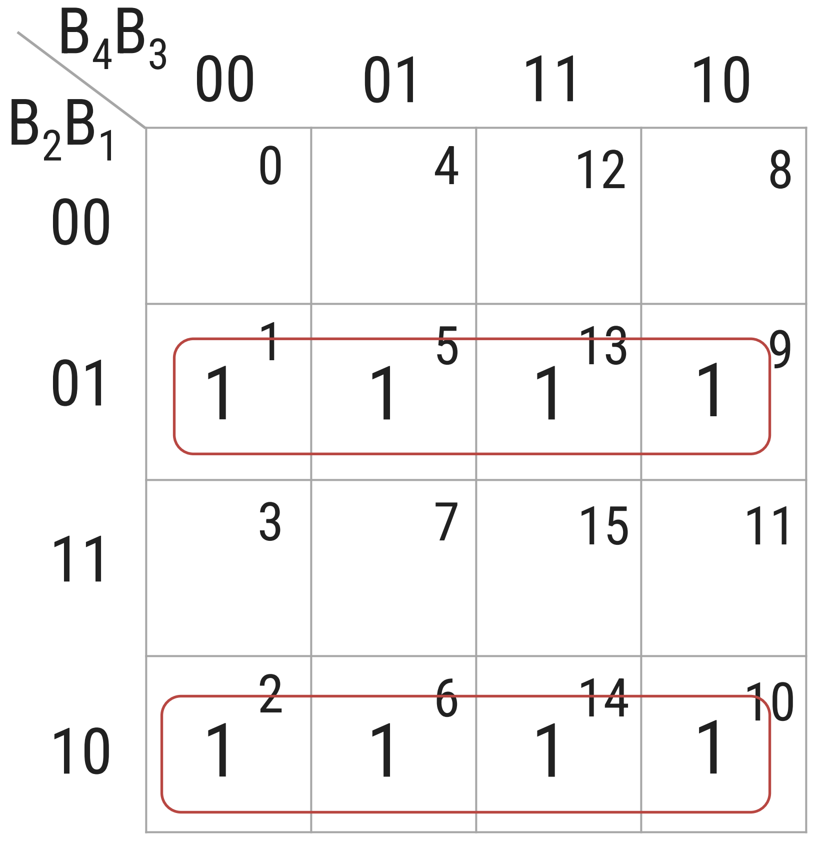
$$G_2 = \sum m(2,3,4,5,10,11,12,13)$$

$$G_1 = \sum m(1,2,5,6,9,10,13,14)$$



B_2B_1 \ B_4B_3	00	01	11	10
00	0	4 1	12 1	8
01	1	5 1	13 1	9
11	3 1	7	15	11 1
10	2 1	6	14	10 1

$G_2 = B_3'B_2 + B_3B_2' = B_3 \oplus B_2$



B_2B_1 \ B_4B_3	00	01	11	10
00	0	4	12	8
01	1 1	5 1	13 1	9 1
11	3	7	15	11
10	2 1	6 1	14 1	10 1

$G_1 = B_2'B_1 + B_2B_1' = B_2 \oplus B_1$

Binary to Gray Code Converter

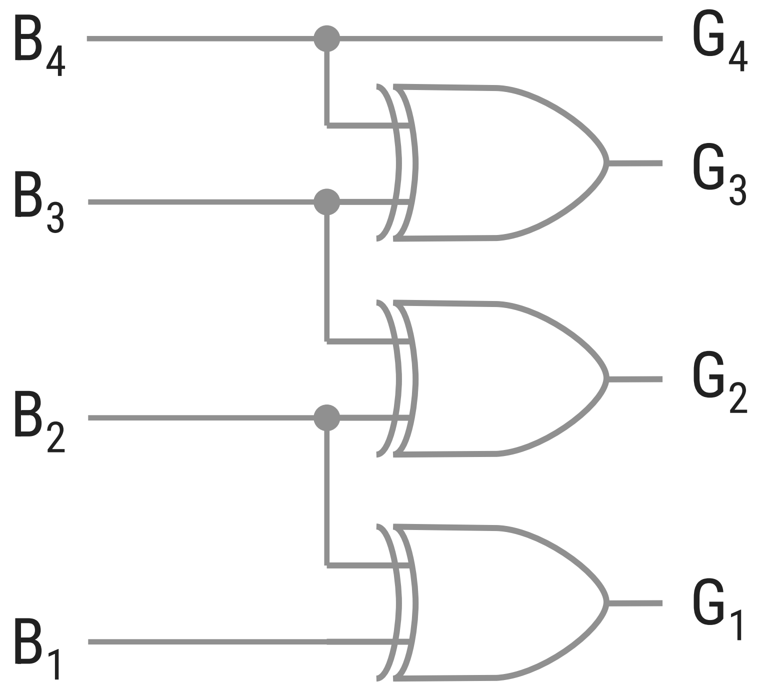
- Logic diagram for binary to Gray code converter is as follows:

$G_4 = B_4$

$G_3 = B_4'B_3 + B_4B_3' = B_4 \oplus B_3$

$G_2 = B_3'B_2 + B_3B_2' = B_3 \oplus B_2$

$G_1 = B_2'B_1 + B_2B_1' = B_2 \oplus B_1$



BCD to Excess-3 Code Converter

8421 BCD				XS - 3			
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

$$X_4 = \sum m(5,6,7,8,9) + d(10,11,12,13,14,15)$$

$$X_3 = \sum m(1,2,3,4,9) + d(10,11,12,13,14,15)$$

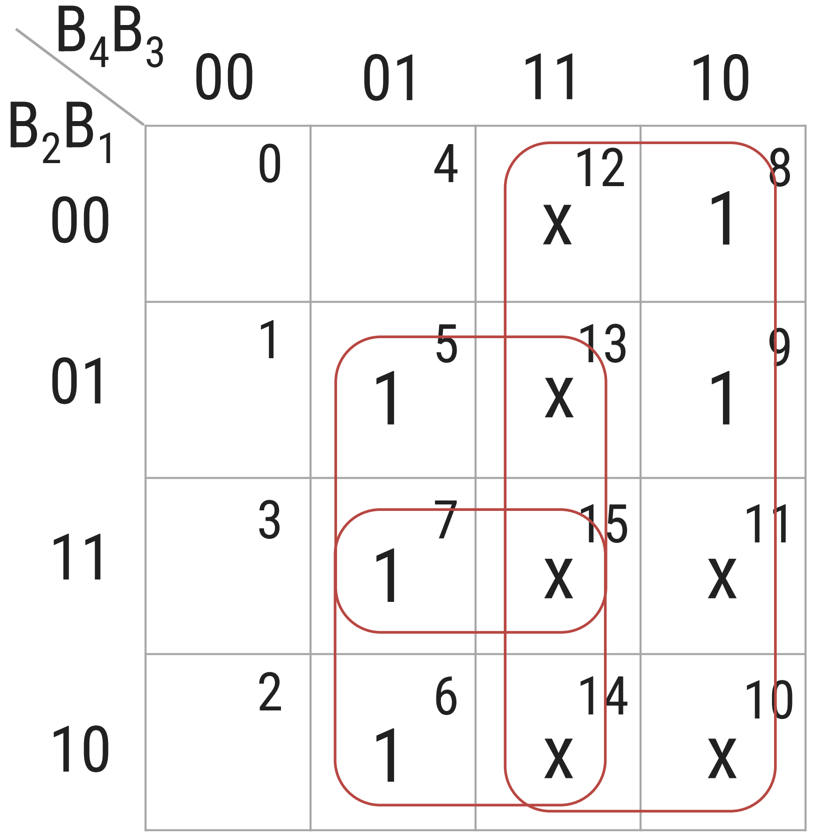
$$X_2 = \sum m(0,3,4,7,8) + d(10,11,12,13,14,15)$$

$$X_1 = \sum m(0,2,4,6,8) + d(10,11,12,13,14,15)$$

BCD to Excess-3 Code Converter

- K-Map for X_4, X_3, X_2, X_1 function and their minimization are as follows:

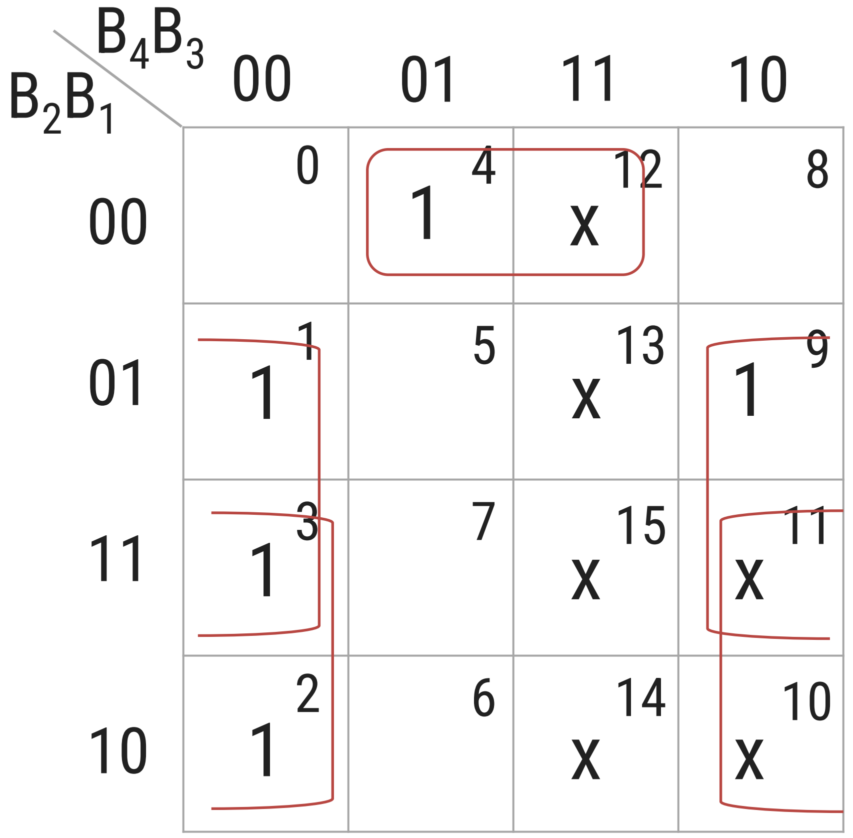
$$X_4 = \sum m(5,6,7,8,9) + d(10,11,12,13,14,15)$$



B_2B_1 \ B_4B_3	00	01	11	10
00	0	4	12 X	8 1
01	1	5 1	13 X	9 1
11	3	7 1	15 X	11 X
10	2	6 1	14 X	10 X

$X_4 = B_4 + B_3B_2 + B_3B_1$

$$X_3 = \sum m(1,2,3,4,9) + d(10,11,12,13,14,15)$$

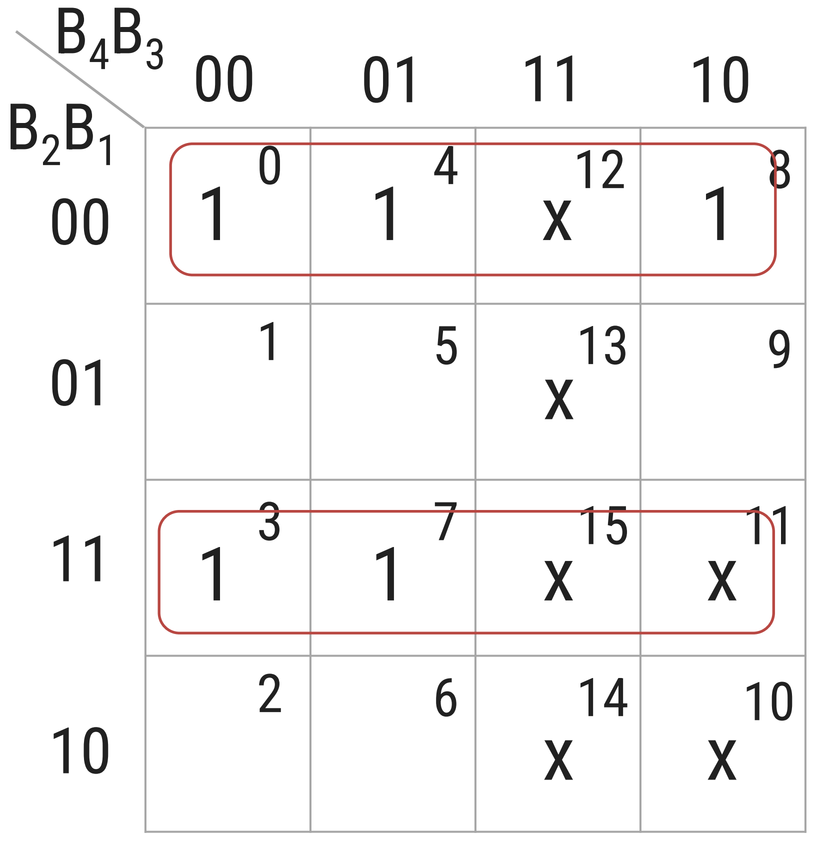


B_2B_1 \ B_4B_3	00	01	11	10
00	0	4 1	12 X	8
01	1 1	5	13 X	9 1
11	3 1	7	15 X	11 X
10	2 1	6	14 X	10 X

$X_3 = B_3B_2'B_1' + B_3'B_1 + B_3'B_2$

BCD to Excess-3 Code Converter

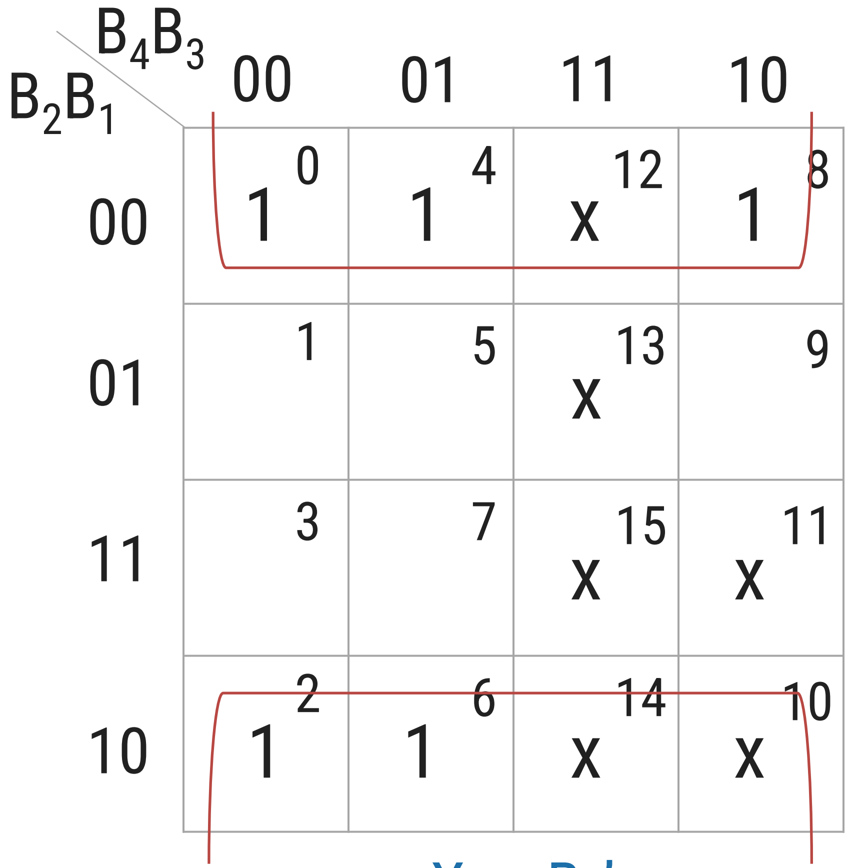
$$X_2 = \sum m(0,3,4,7,8) \; + d(10,11,12,13,14,15)$$



B_2B_1 \ B_4B_3	00	01	11	10
00	1^{0}	1^{4}	X^{12}	1^{8}
01	^{1}	^{5}	X^{13}	^{9}
11	1^{3}	1^{7}	X^{15}	X^{11}
10	^{2}	^{6}	X^{14}	X^{10}

$X_2 = B_2'B_1' + B_2B_1$

$$X_1 = \sum m(0,2,4,6,8) \; + d(10,11,12,13,14,15)$$

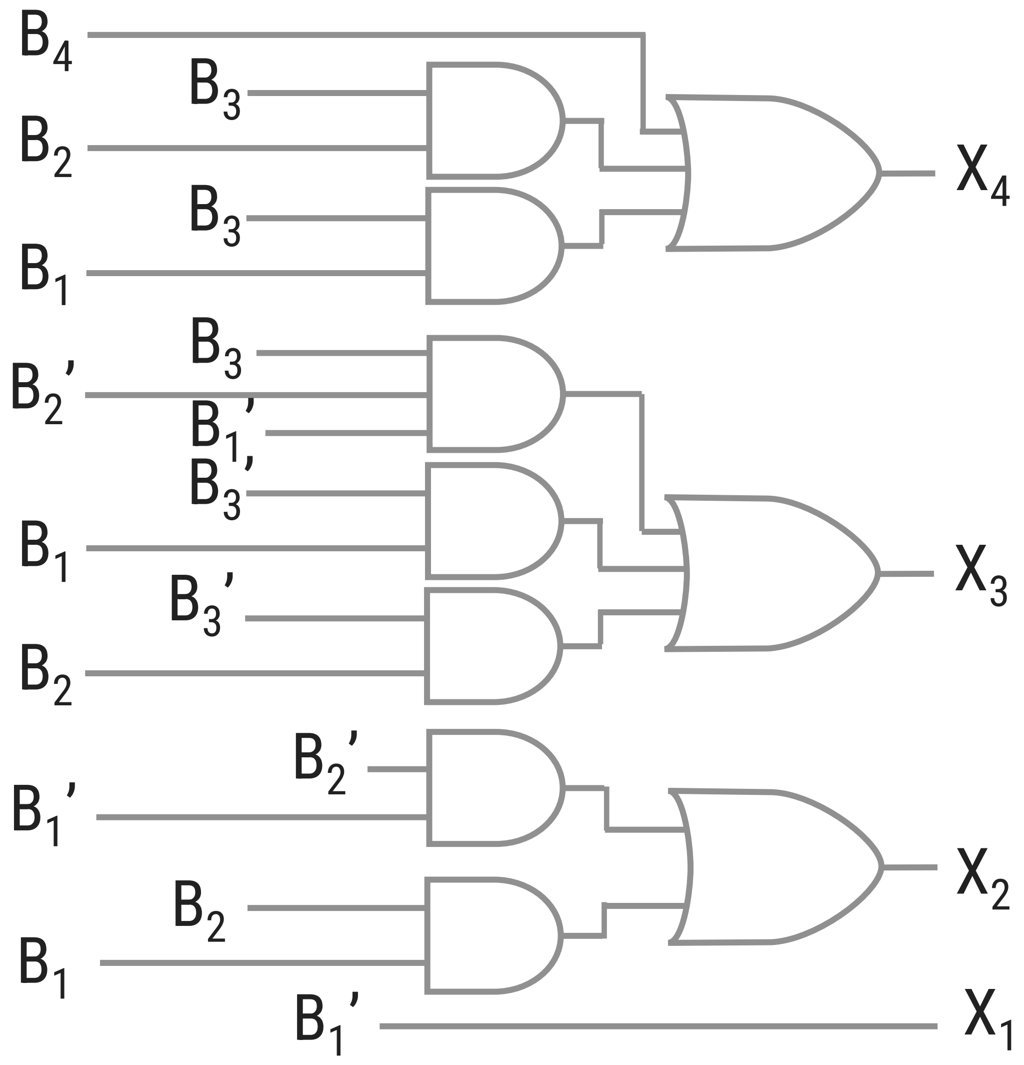


B_2B_1 \ B_4B_3	00	01	11	10
00	1^{0}	1^{4}	X^{12}	1^{8}
01	^{1}	^{5}	X^{13}	^{9}
11	^{3}	^{7}	X^{15}	X^{11}
10	1^{2}	1^{6}	X^{14}	X^{10}

$X_1 = B_1'$

BCD to Excess-3 Code Converter

- Logic diagram for a BCD to Excess-3 is as follows



Comparators

- A comparator is a logic circuit used to compare the magnitudes of two binary numbers.
- Comparator circuit provides 3 outputs
  1. A = B
  2. A > B
  3. A < B
- X-NOR gate is a basic comparator (the output is 1 if and only if the input bits coincide).
- 2 binary numbers are equal if and only if all their corresponding bits coincide.
- E.g. $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are equal if and only if $A_3=B_3$, $A_2=B_2$, $A_1=B_1$ and $A_0=B_0$

$$\text{Equality} = (A_3 \odot B_3)\,(A_2 \odot B_2)\,(A_1 \odot B_1)\,(A_0 \odot B_0)$$

1-bit Magnitude Comparator

- Let the 1-bit numbers be $A = A_0$ and $B = B_0$
- If $A_0 = 1$ and $B_0 = 0$ then $A > B$

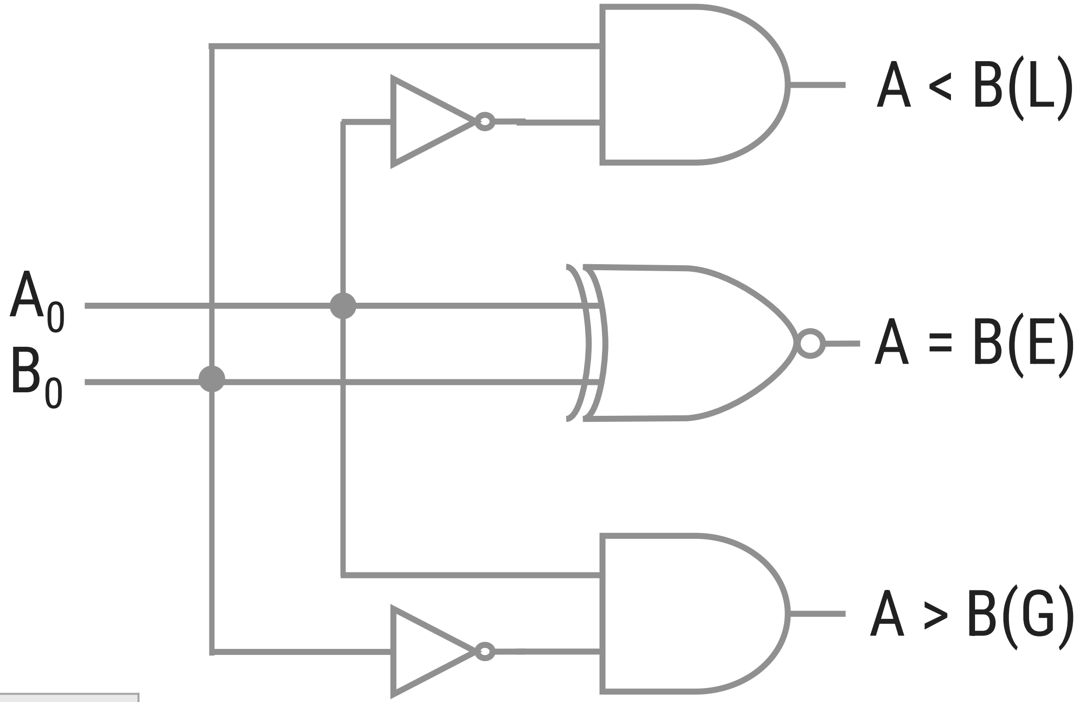
$$A > B : G = A_0 B_0'$$

- If $A_0 = 0$ and $B_0 = 1$ then $A < B$

$$A < B : L = A_0' B_0$$

- If $A_0 = 1$ and $B_0 = 1$ (coincides) then $A = B$

$$A = B : E = A_0 \odot B_0$$



A_0	B_0	L	E	G
0	0			
0	1			
1	0			
1	1			

2-bit Magnitude Comparator

- Let the two 2-bit numbers be $A = A_1A_0$ and $B = B_1B_0$

1. If $A_1 = 1$ and $B_1 = 0$, then $A > B$ or
2. If A_1 and B_1 coincide and $A_0 = 1$ and $B_0 = 0$, then $A > B$.

$$A > B : G = A_1B_1' + (A_1 \odot B_1)\, A_0B_0'$$

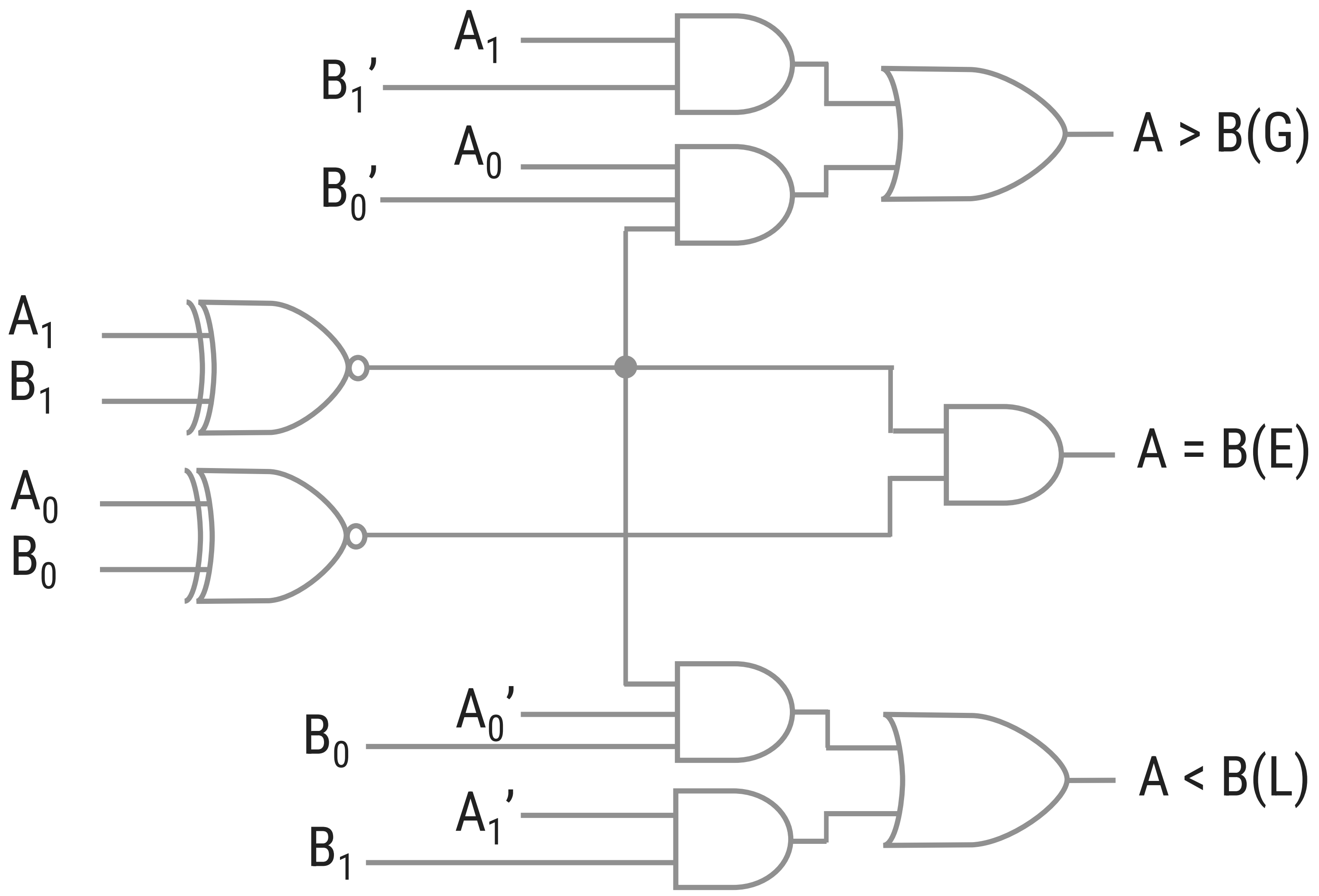
1. If $A_1 = 0$ and $B_1 = 1$, then $A < B$ or
2. If A_1 and B_1 coincide and $A_0 = 0$ and $B_0 = 1$, then $A < B$.

$$A < B : L = A_1'B_1 + (A_1 \odot B_1)\, A_0'B_0$$

1. If A_1 and B_1 coincide and if A_0 and B_0 coincide then $A = B$.

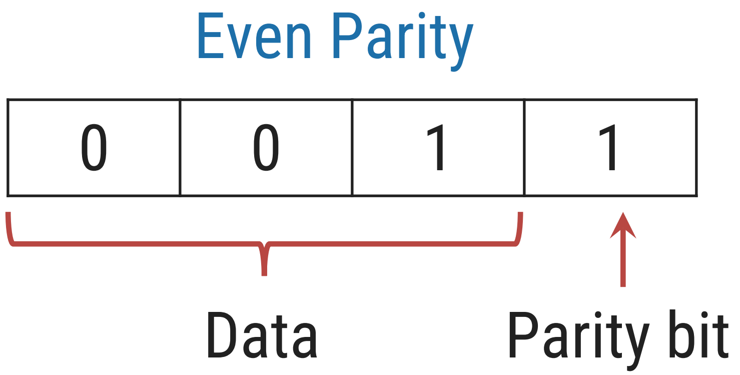
$$A = B : E = (A_1 \odot B_1)(A_0 \odot B_0)$$

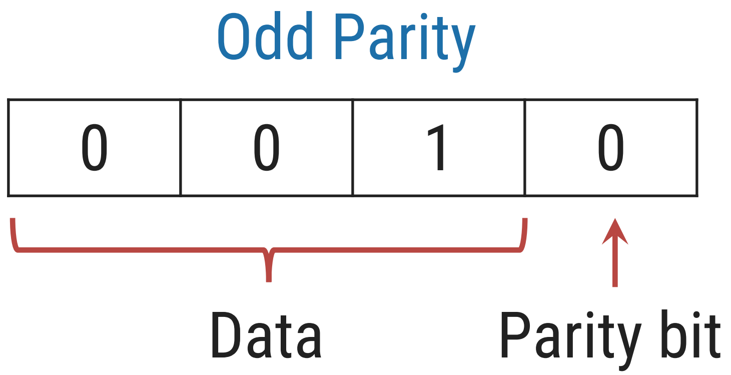
2-bit Magnitude Comparator



Parity Generator

- Binary data, when transmitted and processed is susceptible to noise that can alter its 1s to 0s and 0s to 1s.
- To detect such errors, an additional bit called parity bit is added to the data bits and the word containing the data bits and the parity bit is transmitted.
- At the receiving end the number of 1s in the word received are counted and the error, if any, is detected.

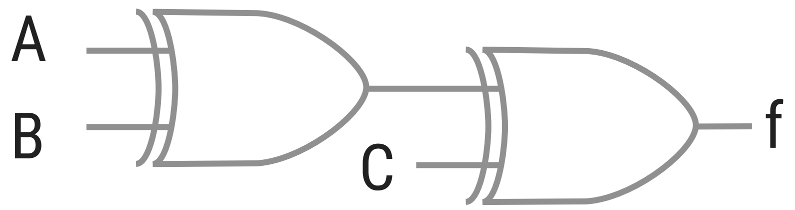




3-bit parity generator using even parity bit

Inputs			Outputs parity bit (f)
A	B	C	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

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



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

$$f = A'(B'C + BC') + A(BC + B'C')$$

$$f = A'(B \oplus C) + A(B \oplus C)'$$

$$f = A \oplus B \oplus C$$

Multiplexer/De-multiplexer, Encoder/Decoder

Multiplexer

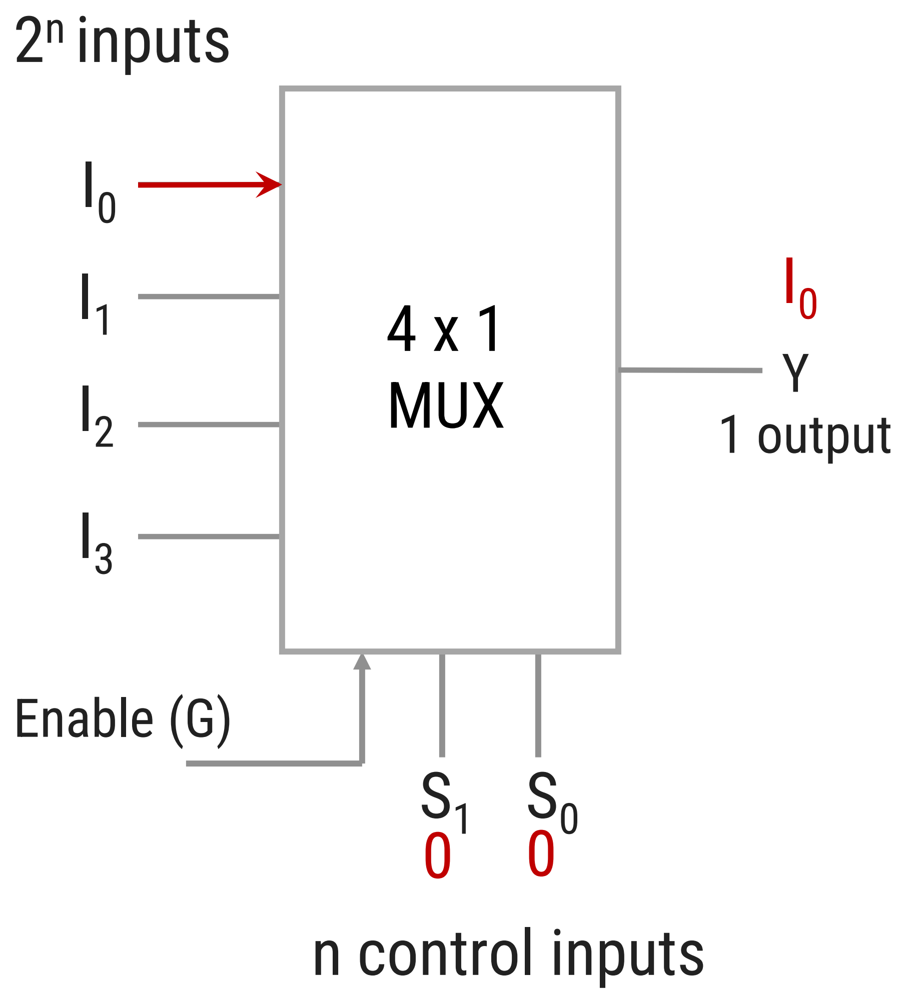
- A multiplexer(MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.
- Consider an integer ‘m’, which is constrained by the following relation:

  $m = 2^n$, where m and n are both integers.

- A m-to-1 Multiplexer has
  - m Inputs: $I_0, I_1, I_2, \ldots\ldots\ldots\ldots\ldots\ldots I_{(m-1)}$
  - One Output: Y
  - n Control inputs: $S_0, S_1, S_2, \ldots\ldots S_{(n-1)}$
  - One (or more) Enable input(s)

Such that Y may be equal to one of the inputs, depending upon the control inputs.

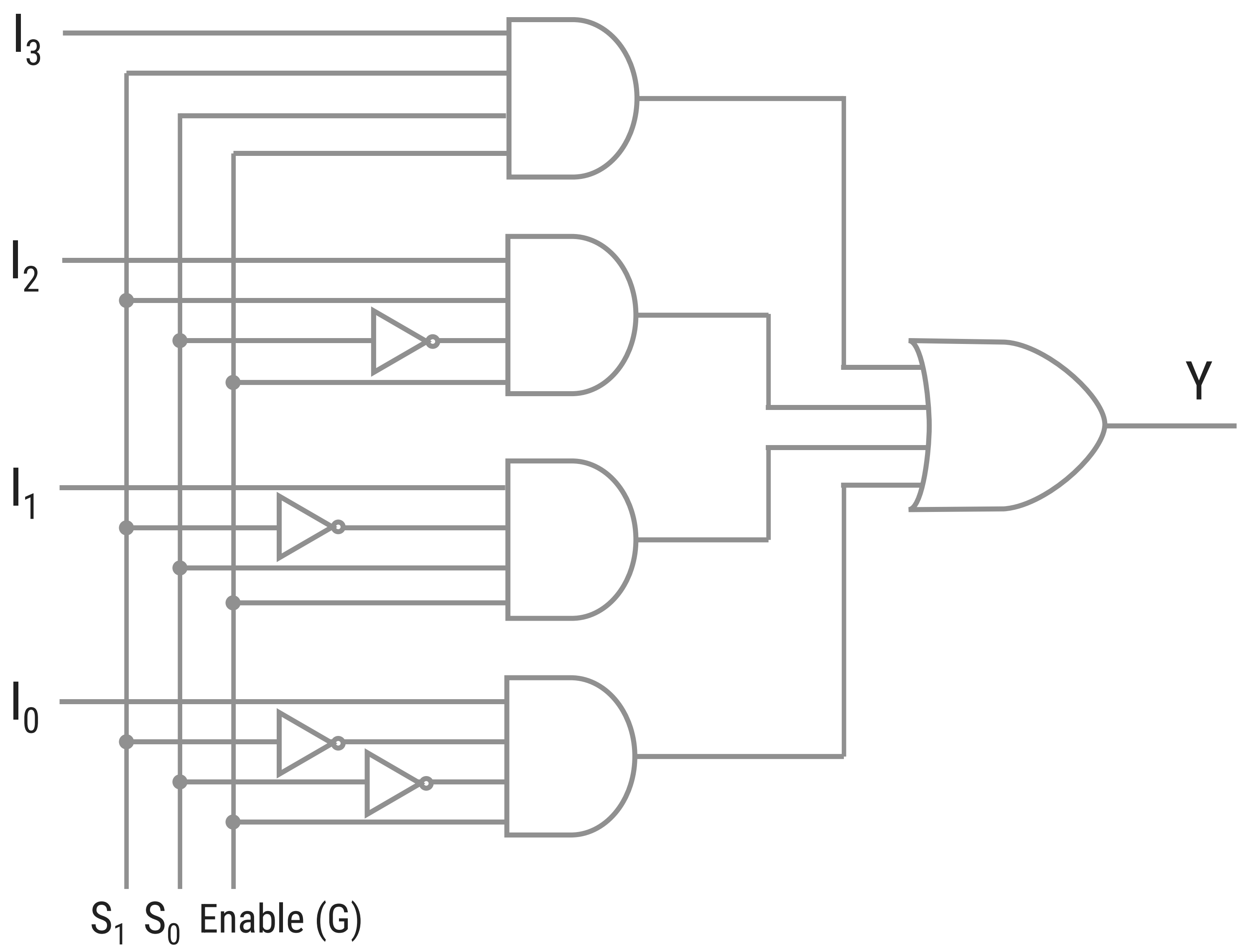
4-to-1 Multiplexer



Select Inputs		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

4 x 1 MUX Actual Circuit

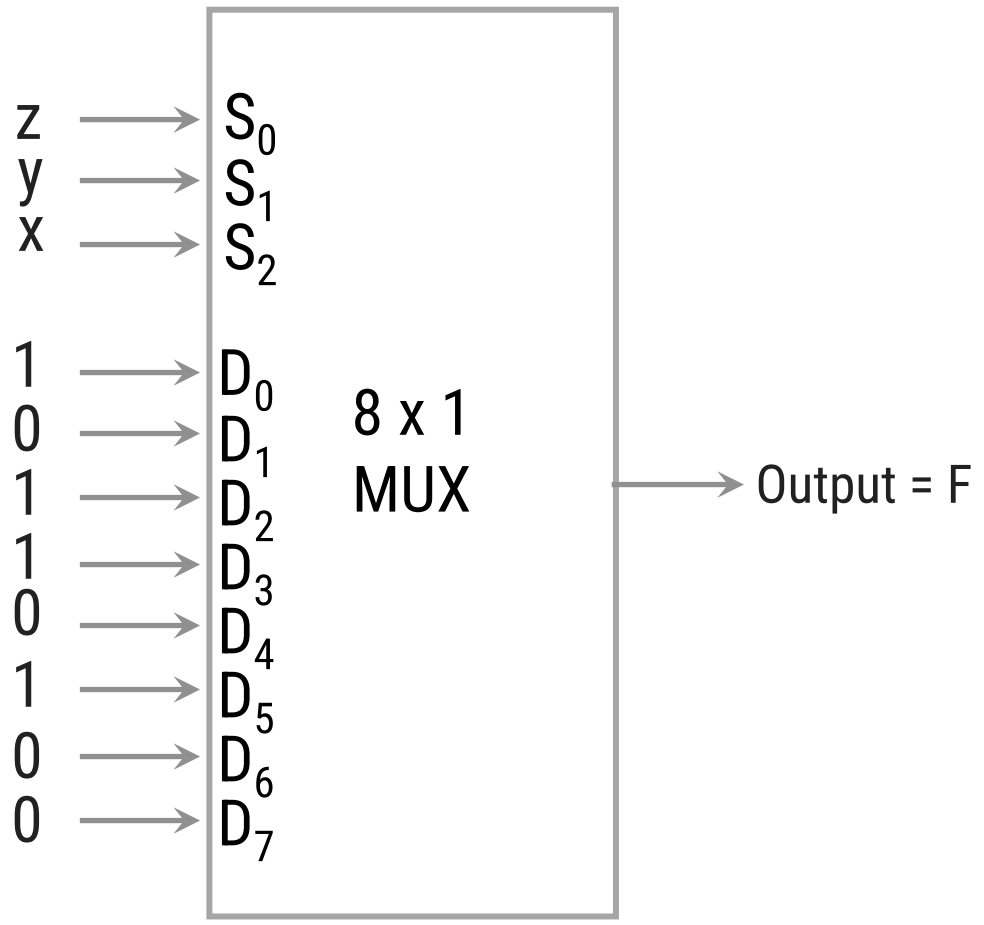


Logic function generator using Multiplexer

- Implement the following function using 8 to 1 MUX

$$F(x,y,z) = \sum_m (0,2,3,5)$$

S_2	S_1	S_0	F
x	y	z	
0	0	0	1
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	0



Logic function generator using Multiplexer (Method)

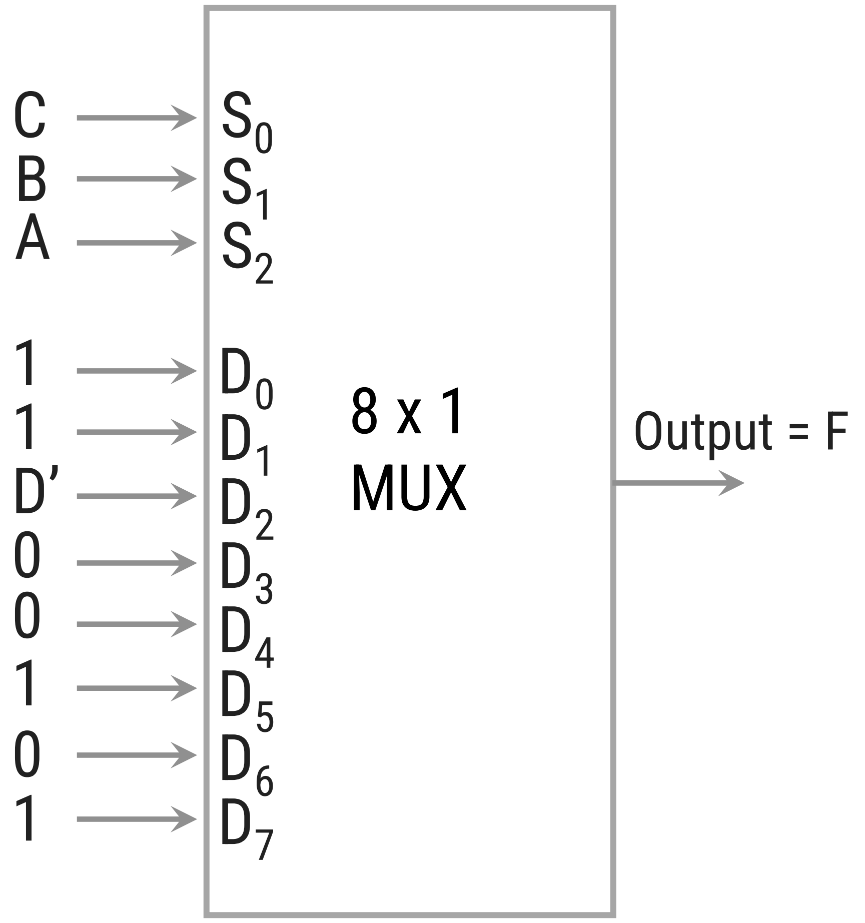
- Multiplexer with n-data select inputs can implement any function of n + 1 variables.
- The first n variables of the function as the select inputs and to use the least significant input variable and its complement to drive some of the data inputs.
- If the single variable is denoted by D, each data output of the multiplexer will be D, D’, 1, or 0.
- Suppose, we wish to implement a 4-variable logic function using a multiplexer with three data select inputs.
- Let the input variables be A, B, C, and D; D is the LSB.
- A truth table for the function F(A, B, C, D) is constructed with ABC has the same value twice once with D = 0 and again with D = 1.

Logic function generator using Multiplexer (Method)

- The following rules are used to determine the connections that should be made to the data inputs of the multiplexer.
  1. If F = 0 both times when the same combination of ABC occurs, connect logic 0 to the data input selected by that combination.
  2. If F = 1 both times when the same combination of ABC occurs, connect logic 1 to the data input selected by that combination.
  3. If F is different for the two occurrences of a combination of ABC, and if F = D in each case, connect D to the data input selected by that combination.
  4. If F is different for the two occurrences of a combination of ABC, and if F = D’ in each case, connect D’ to the data input selected by that combination.

Implement the following function using 8 to 1 MUX
F = $\sum_m(0,1,2,3,4,10,11,14,15)$

S_2	S_1	S_0	D	F	
A	B	C			
0	0	0	0	1	
0	0	0	1	1	
0	0	1	0	1	
0	0	1	1	1	
0	1	0	0	1	
0	1	0	1	0	
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	1	
1	0	1	1	1	
1	1	0	0	0	
1	1	0	1	0	
1	1	1	0	1	
1	1	1	1	1	



Demultiplexer

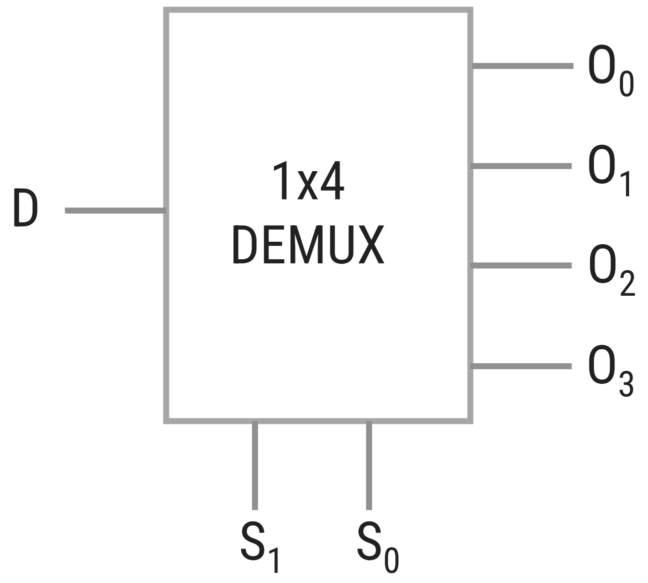
- A demultiplexer(DEMUX) is a device that allows digital information from one source to be routed onto a multiple lines for transmission over different destinations.
- Consider an integer ‘m’, which is constrained by the following relation:

  $m = 2^n$, where m and n are both integers.

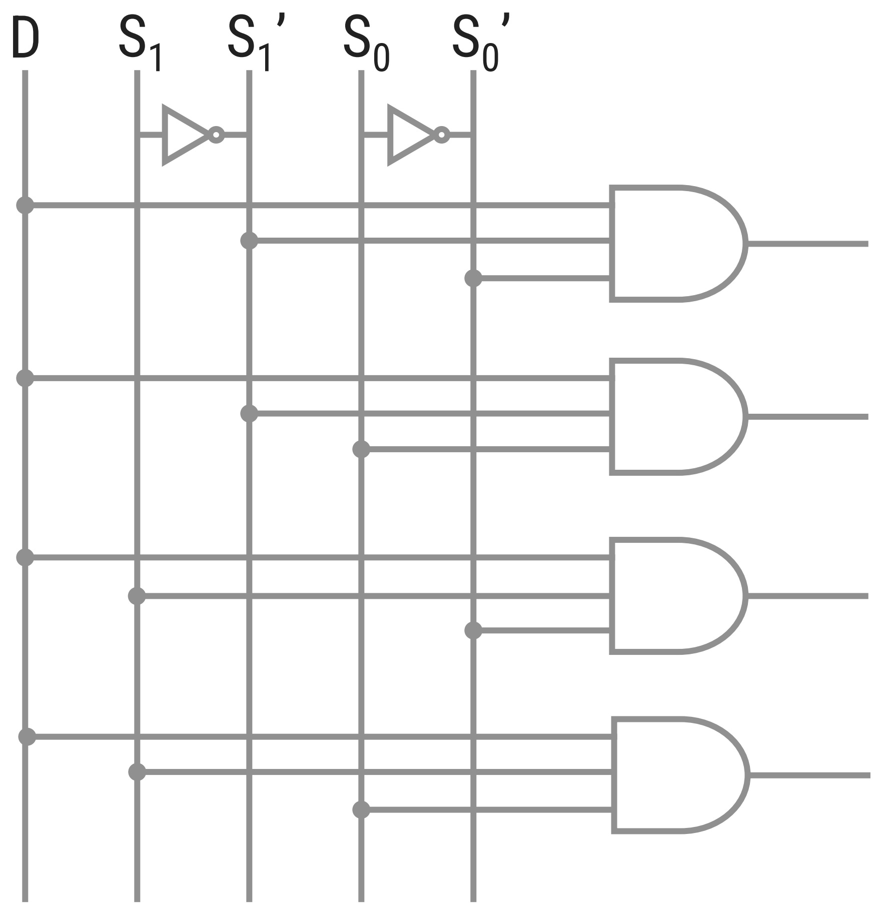
- A 1-to-m Demultiplexer has
  - One Input: D
  - m Outputs: O_0, O_1, O_2, ... $O_{(m-1)}$
  - n Control inputs: S_0, S_1, S_2, ... $S_{(n-1)}$
  - One (or more) Enable input(s)

  Such that D may be transfer to one of the outputs, depending upon the control inputs.

1-to-4 Demultiplexer



Select code		Outputs			
S_1	S_0	O_3	O_2	O_1	O_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0



$O_0 = D\,S_1'\,S_0'$

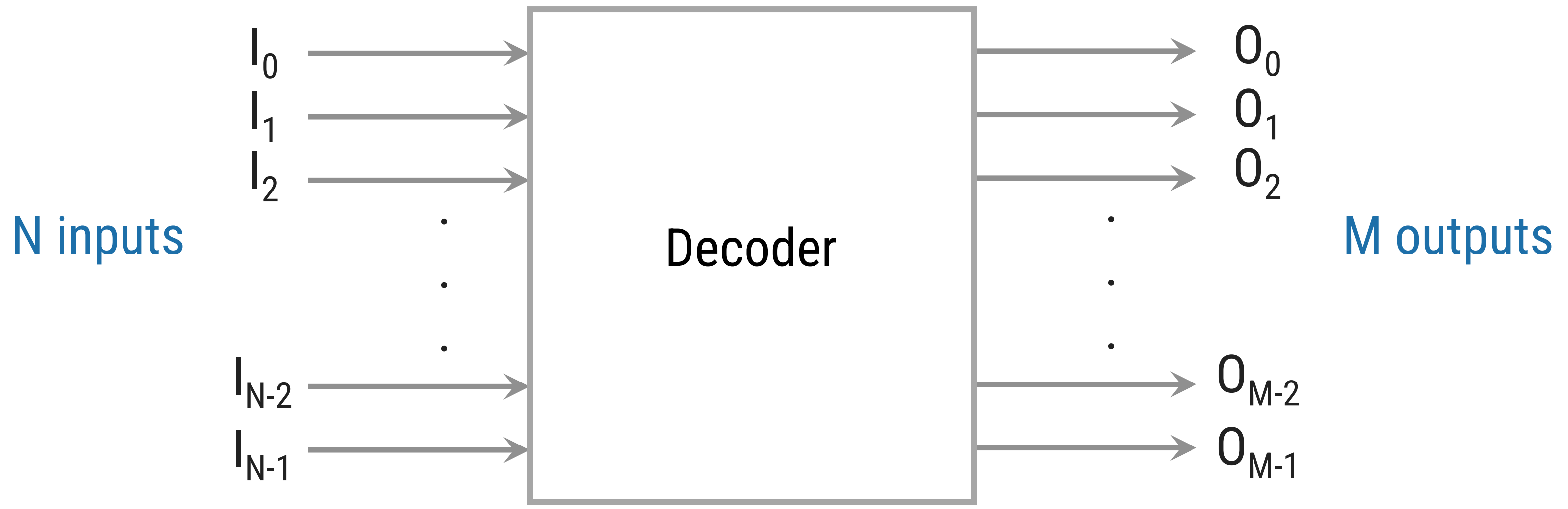
$O_1 = D\,S_1'\,S_0$

$O_2 = D\,S_1\,S_0'$

$O_3 = D\,S_1\,S_0$

Decoder

- A decoder is a logic circuit that accepts a set of inputs which represents a binary number and activates the only output that corresponds to the input number.
- In other words, a decoder circuit looks at its inputs, determines which binary number is present there, and activates the specific output which corresponds to that number; all other outputs remain inactive.
- In its general form, a decoder has N input lines to handle N bits and M output lines such that only one output line is activated for each one of the possible combinations of inputs.

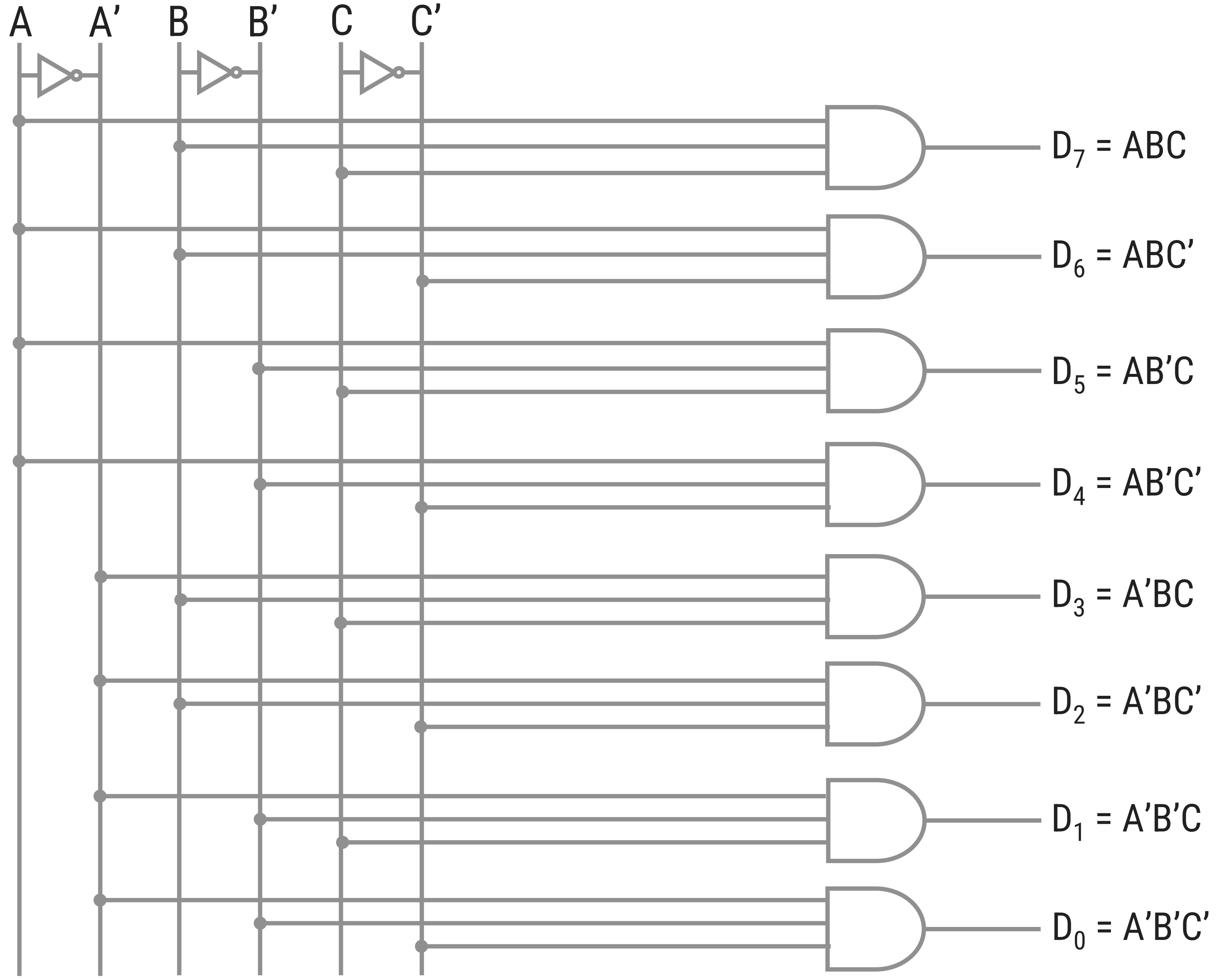


$M = 2^N$

3-Line to 8-Line Decoder

Inputs			Outputs							
A	B	C	D_0 A'B'C'	D_1 A'B'C	D_2 A'BC'	D_3 A'BC	D_4 AB'C'	D_5 AB'C	D_6 ABC'	D_7 ABC
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

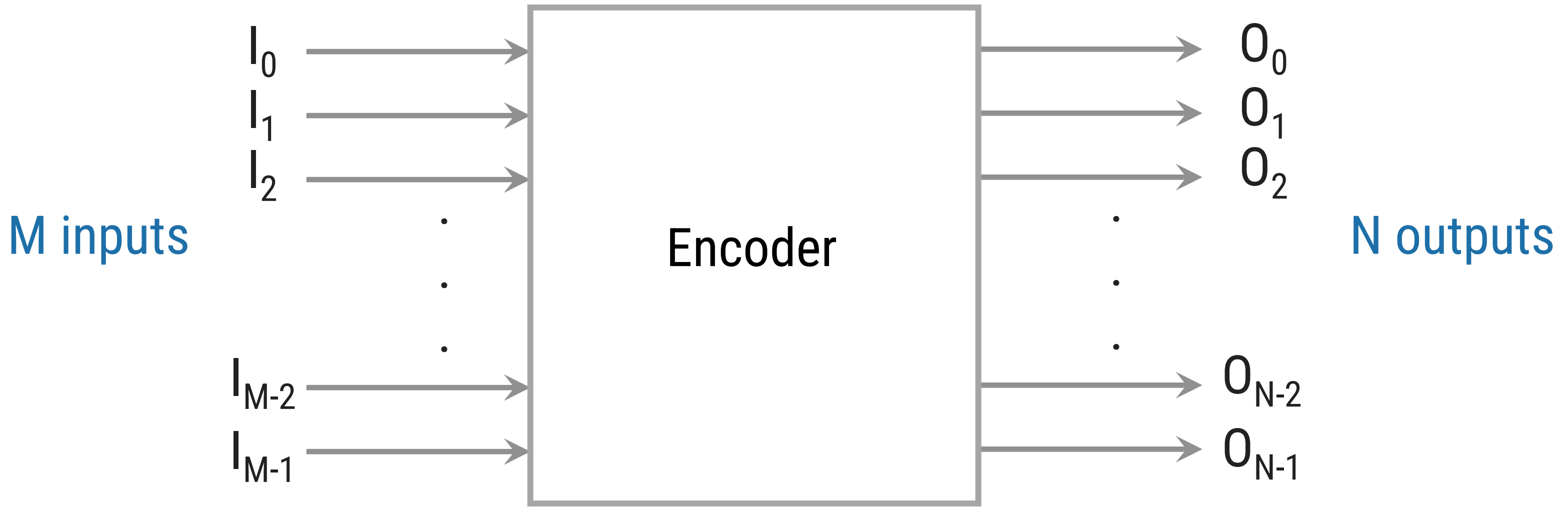
- 3 to 8 line decoder can be implemented using AND gates to achieve active-HIGH output.
- For active-LOW outputs, NAND gates are used.



A
A′
B
B′
C
C′
D_7 = ABC
D_6 = ABC′
D_5 = AB′C
D_4 = AB′C′
D_3 = A′BC
D_2 = A′BC′
D_1 = A′B′C
D_0 = A′B′C′

Encoder

- Device to convert familiar numbers or symbols into coded format.
- It has a number of input lines, only one of which is activated at a given time, and produces an N-bit output code depending on which input is activated.
- Figure shows the block diagram of an encoder with M inputs and N outputs.



$M = 2^N$

Priority Encoder

- A priority encoder is a logic circuit that responds to just one input in accordance with some priority system, among all those that may be simultaneously HIGH.
- The most common priority system is based on the relative magnitudes of the inputs; whichever decimal input is the largest, is the one that is encoded.
- For example, if both decimal 3 and decimal 4 are activated simultaneously, then a priority encoder would encode decimal 4.

Inputs				Outputs		
D_0	D_1	D_2	D_3	A	B	V
0	0	0	0	x	x	0
1	0	0	0	0	0	1
x	1	0	0	0	1	1
x	x	1	0	1	0	1
x	x	x	1	1	1	1

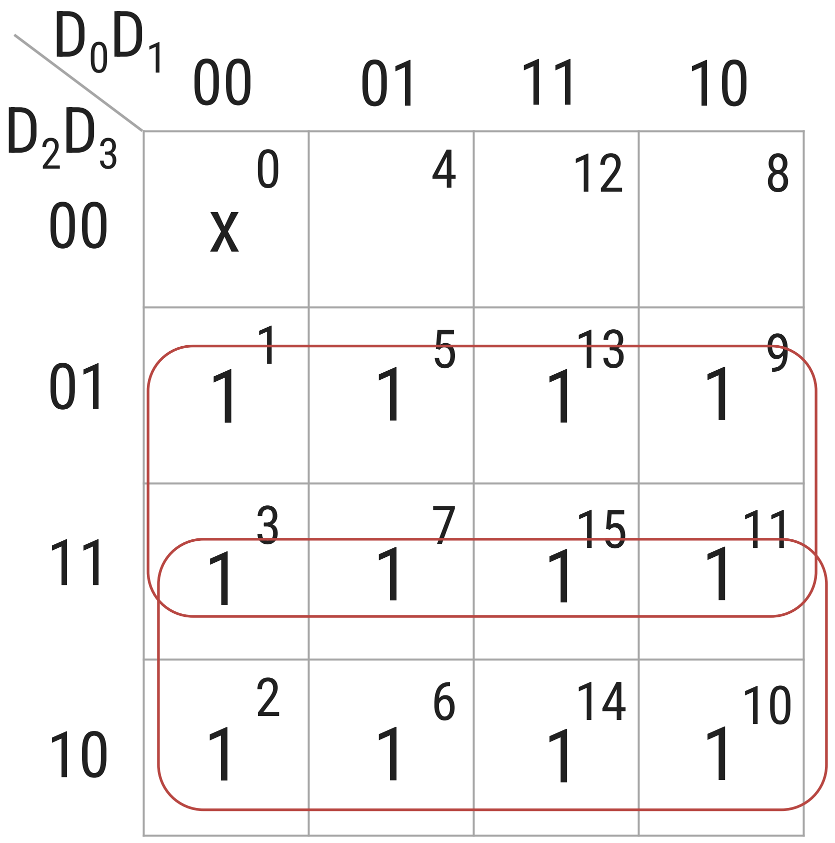
$$A = \sum_m (1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$$

$$B = \sum_m (1, 3, 4, 5, 7, 9, 11, 12, 13, 15)$$

$$V = \sum_m (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$$

Priority Encoder

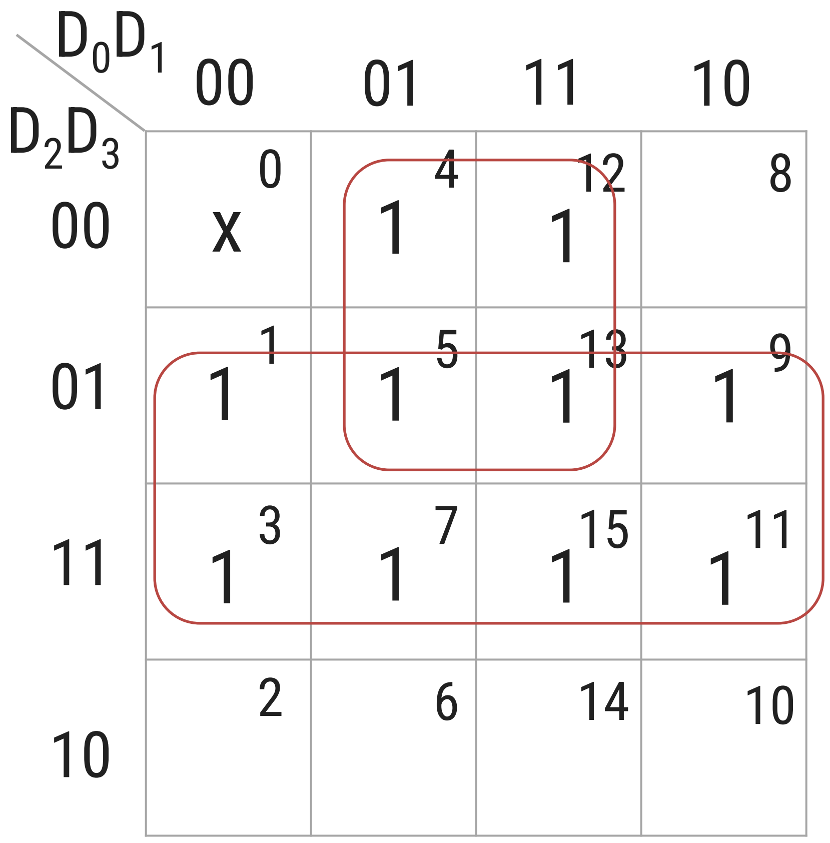
$$A = \sum_m (1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$$



$D_2D_3 \backslash D_0D_1$	00	01	11	10
00	X ^{0}	^{4}	^{12}	^{8}
01	1 ^{1}	1 ^{5}	1 ^{13}	1 ^{9}
11	1 ^{3}	1 ^{7}	1 ^{15}	1 ^{11}
10	1 ^{2}	1 ^{6}	1 ^{14}	1 ^{10}

$A = D_3 + D_2$

$$B = \sum_m (1, 3, 4, 5, 7, 9, 11, 12, 13, 15)$$

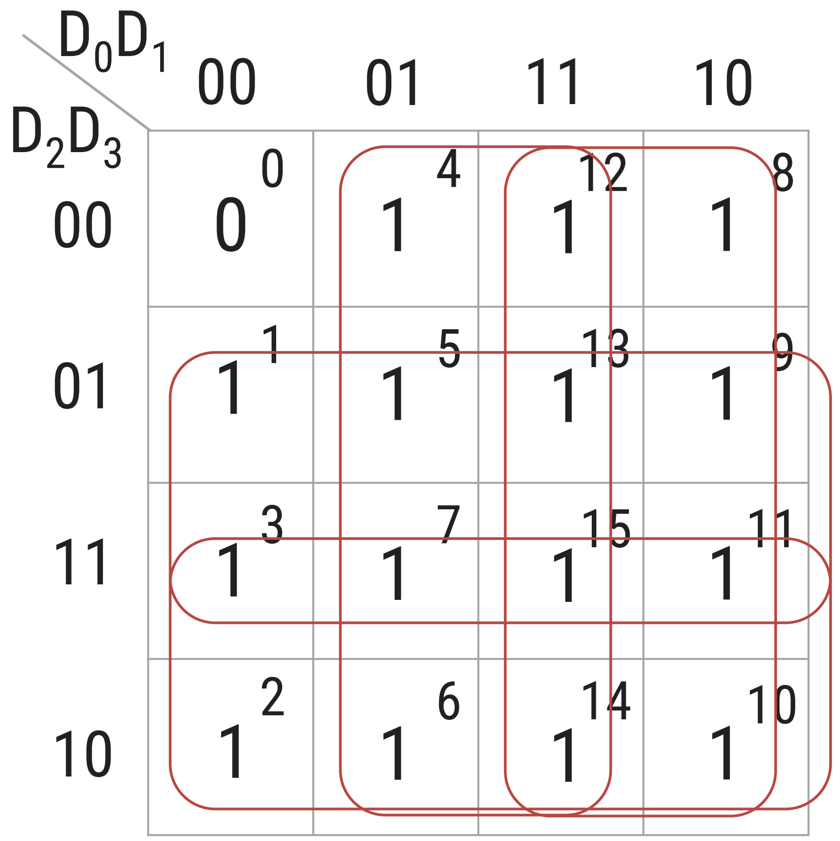


$D_2D_3 \backslash D_0D_1$	00	01	11	10
00	X ^{0}	1 ^{4}	1 ^{12}	^{8}
01	1 ^{1}	1 ^{5}	1 ^{13}	1 ^{9}
11	1 ^{3}	1 ^{7}	1 ^{15}	1 ^{11}
10	^{2}	^{6}	^{14}	^{10}

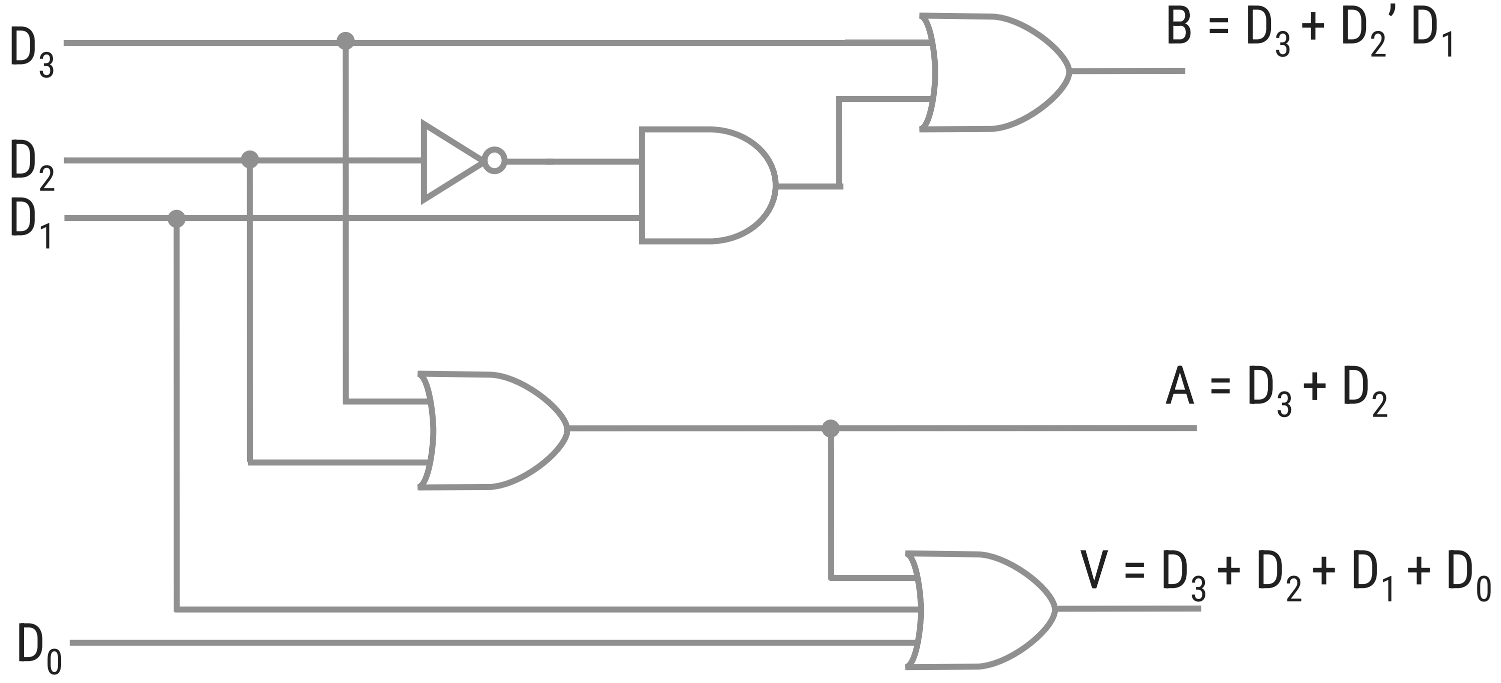
$B = D_3 + D_2' D_1$

Priority Encoder

$$V = \sum_m (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$$

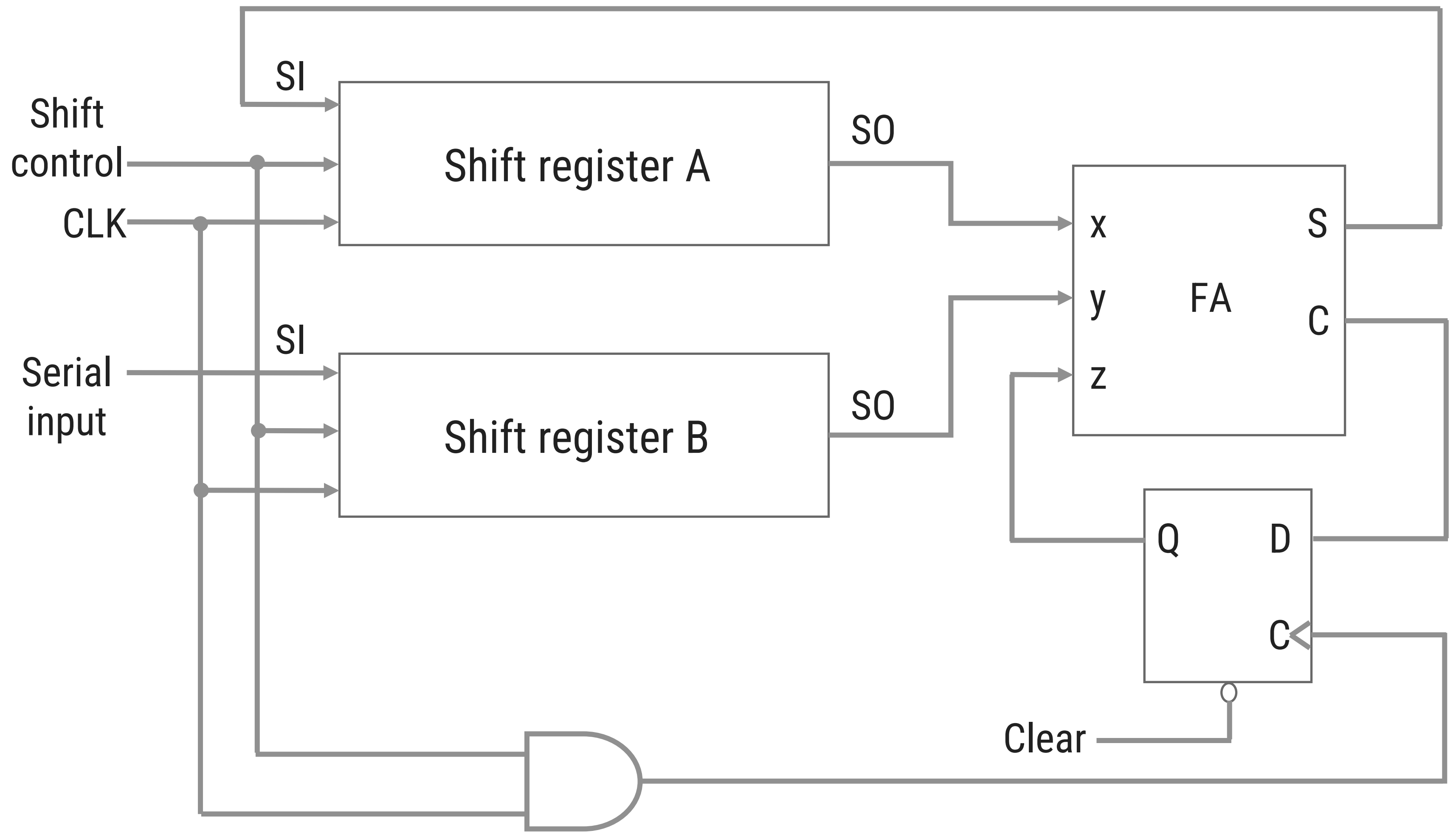


$D_2D_3 \backslash D_0D_1$	00	01	11	10
00	0 (0)	1 (4)	1 (12)	1 (8)
01	1 (1)	1 (5)	1 (13)	1 (9)
11	1 (3)	1 (7)	1 (15)	1 (11)
10	1 (2)	1 (6)	1 (14)	1 (10)

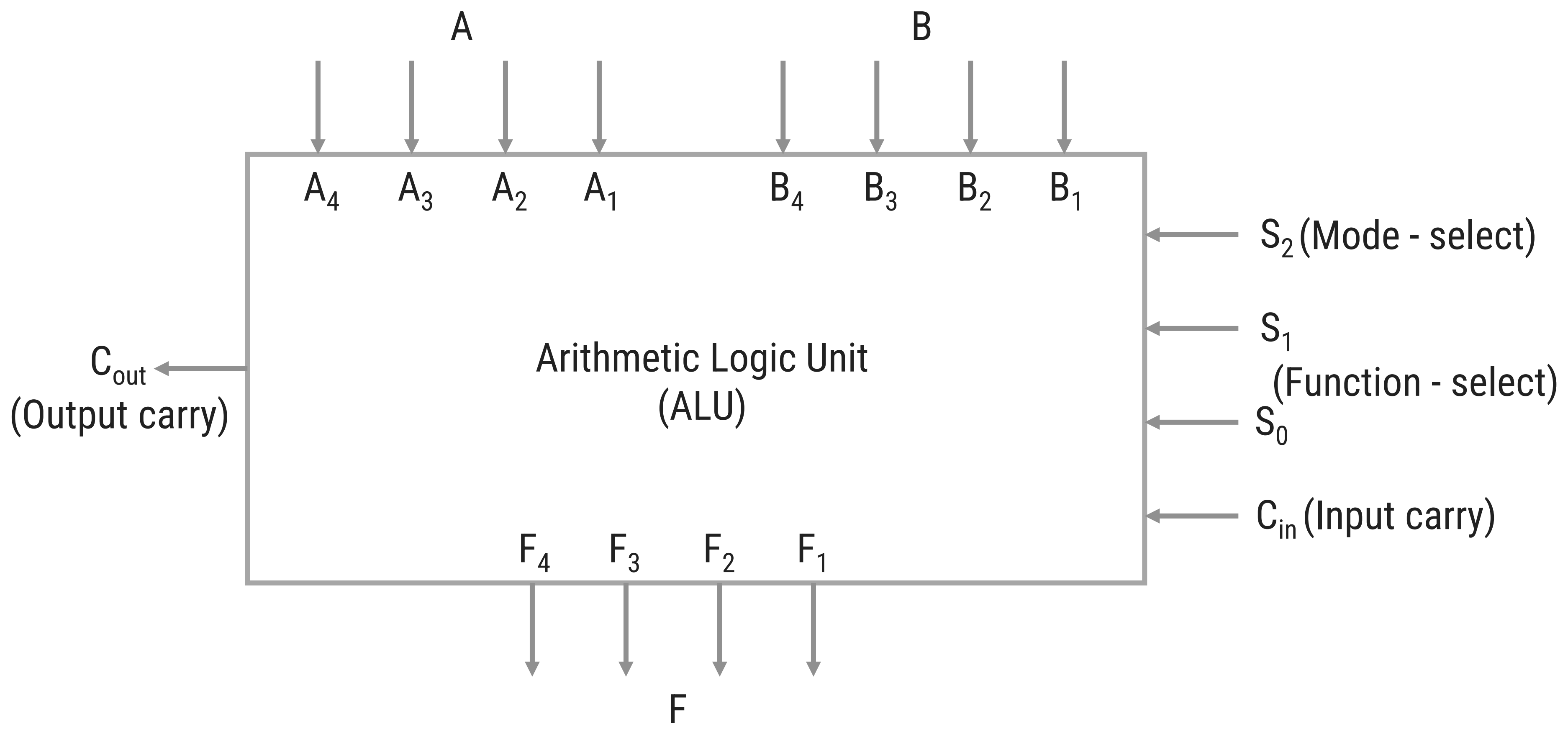


$V = D_3 + D_2 + D_1 + D_0$

Serial Adder



Arithmetic Logic Unit (ALU)



Arithmetic Logic Unit (ALU)

Selection				Output	Function
S_2	S_1	S_0	C_{in}		
0	0	0	0	F = A	Transfer A
0	0	0	1	F = A + 1	Increment A
0	0	1	0	F = A + B	Addition
0	0	1	1	F = A + B + 1	Add with carry
0	1	0	0	F = A − B − 1	Subtract with borrow
0	1	0	1	F = A − B	Subtraction
0	1	1	0	F = A − 1	Decrement A
0	1	1	1	F = A	Transfer A
1	0	0	X	F = A + B	OR
1	0	1	X	$F = A \oplus B$	XOR
1	1	0	X	F = A . B	AND
1	1	1	X	F = A’	Complement A