Combinational Logic Design

Dr. Chandan Karfa
CSE IIT Guwahati

- Code conversion
- Parity bit generator
- Comparator
- Multiplexer

Source

- Chapter 4: M. M. Mano and M. D. Ciletti, Digital Design, 5th Ed., Pearson Education.

- Chapter 5: Z. Kohavi and N. Jha, Switching and Finite Automata Theory, 3rd Ed., Cambridge University Press, 2010.

Basic

- The number of gate inputs that can be driven by the output of a single gate is limited. The maximum such number is called the **fanout** of the gate.

- The bound on the number of inputs that a single gate may have is referred to as the **fanin** of the gate.

- A finite amount of time is required to propagate a signal through a gate, or to switch a gate output from one value to another is known as the **propagation delay**.

Logic design with integrated circuits

Integrated circuits are produced in packages, or chips, and are historically classified into four categories:

1. **Small-scale integration (SSI)** usually refers to packages containing single gates, e.g., AND, OR, NOT, NAND, NOR, XOR, or small packages containing two or four gates of the same type.

2. Medium-scale integration (MSI) refers to intermediate packages containing up to about 100 gates. They usually realize standard circuits that are used often in logic design, e.g., code converters, adders, etc.

3. Large-scale integration (LSI), may contain many hundreds or thousands of gates in a single package. Some LSI circuits are standard, e.g., subsystems for computer control or for a computer arithmetic unit, while other LSI circuits are manufactured to the specification of the logic designer.

4. Very-large-scale integration (VLSI) is what we currently observe, in chips in which there may be millions of gates.

ANALYSIS OF COMBINATIONAL CIRCUITS

A combinational circuit is analyzed by tracing the output of each gate, starting from the circuit inputs and continuing toward each circuit output.

The output, designated C0, is given by

C0 = AB + (A + B)C = AB + AC + BC

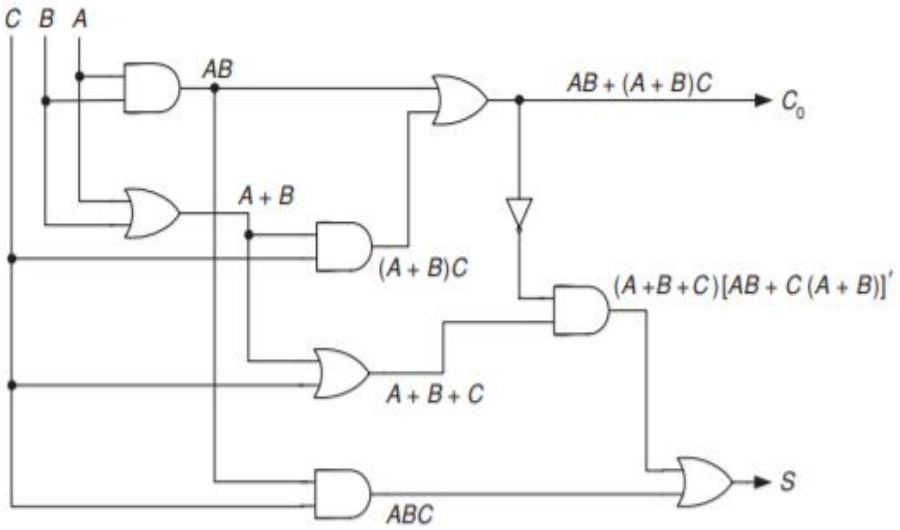
The second output, designated S, is found to be

S = (A + B + C)[AB + (A + B)C]' + ABC

= (A + B + C)(A' + B')(A' + C')(B' + C') + ABC

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

= A ⊕ B ⊕ C



Design Procedure

- From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
- Derive the truth table that defines the required relationship between inputs and outputs.
- Obtain the simplified Boolean functions for each output as a function of the input variables.
- Draw the logic diagram and verify the correctness of the design (manually or by simulation).

Code Conversion Example : BCD to Excess-3

Truth Table for Code Conversion Example

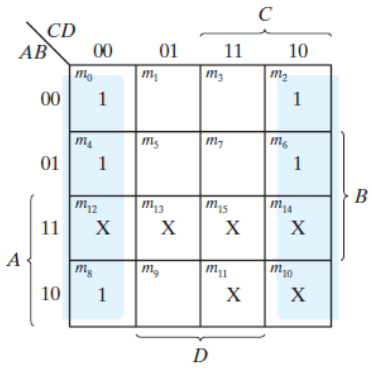
Input BCD				Output Excess-3 Code			
A	**B**	**C**	**D**	**w**	**x**	**y**	**z**
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

$$z = D'$$

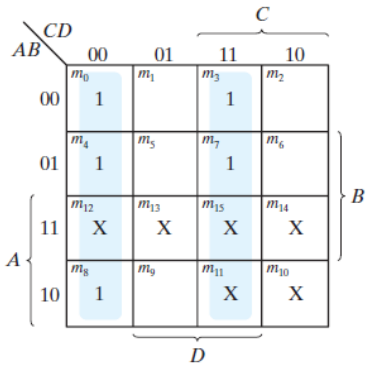
$$y = CD + C'D' = CD + (C + D)'$$

$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$

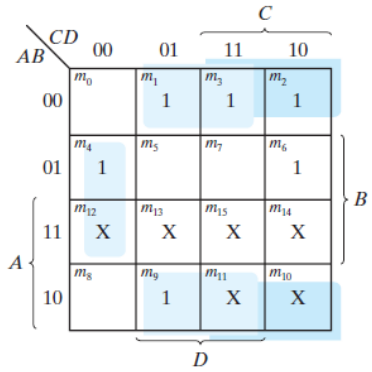
$$w = A + BC + BD = A + B(C + D)$$



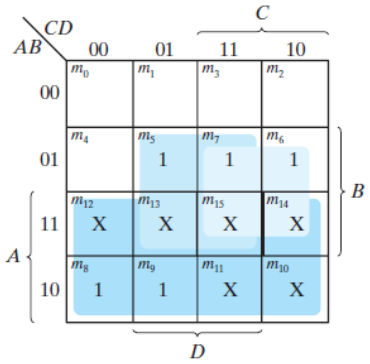
$z = D'$



$y = CD + C'D'$

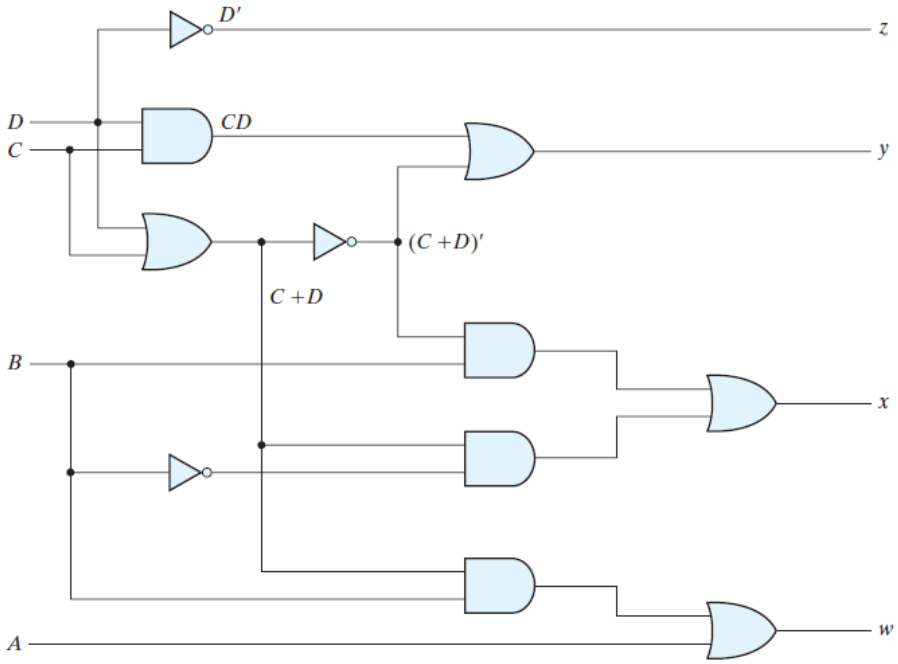


$x = B'C + B'D + BC'D'$



$w = A + BC + BD$

Code Conversion Example



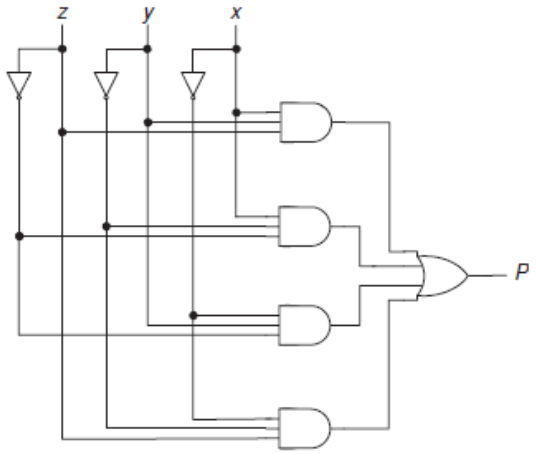
$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$

Parity-bit generator

- Produce an output value 1 if and only if **an odd number of its inputs have the value 1**.
- p = x'y'z + x'yz' + xy'z' + xyz.

z \ xy	00	01	11	10
0	0	1	0	1
1	1	0	1	0

(*a*) Map.

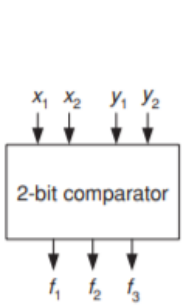


(*b*) Implementation.

COMPARATORS

An **n-bit comparator** is a circuit that compares the magnitude of two numbers X and Y . It has three outputs f1, f2, and f3, such that:
f1 = 1 iff (if and only if) X>Y;
f2 = 1 iff X = Y;
f3 = 1 iff X < Y.

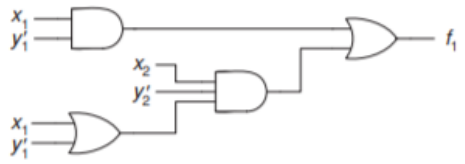
$$
\begin{aligned}
f_1 &= x_1x_2y_2' + x_2y_1'y_2' + x_1y_1' \\
&= (x_1 + y_1')x_2y_2' + x_1y_1', \\
f_2 &= x_1'x_2'y_1'y_2' + x_1'x_2y_1'y_2 + x_1x_2'y_1y_2' + x_1x_2y_1y_2 \\
&= x_1'y_1'(x_2'y_2' + x_2y_2) + x_1y_1(x_2'y_2' + x_2y_2) \\
&= (x_1'y_1' + x_1y_1)(x_2'y_2' + x_2y_2), \\
f_3 &= x_2'y_1y_2 + x_1'x_2'y_2 + x_1'y_1 \\
&= x_2'y_2(y_1 + x_1') + x_1'y_1.
\end{aligned}
$$



(a) Block diagram.

y_1y_2 \ x_1x_2	00	01	11	10
00	2	1	1	1
01	3	2	1	1
11	3	3	2	3
10	3	3	1	2

(b) Map for f_1, f_2, and f_3.



(c) Circuit for f_1.

2 bits comparator

4 bits Comparator

$$A > B, A = B, \text{or } A < B.$$

$$A = A_3\, A_2\, A_1\, A_0$$
$$B = B_3 B_2 B_1 B_0$$

The two numbers are equal if all pairs of significant digits are equal: $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, and $A_0 = B_0$.

$$x_i = A_i B_i + A_i' B_i' \quad \text{for } i = 0, 1, 2, 3$$

$$(A = B) = x_3 x_2 x_1 x_0$$

A is greater or less than B , we inspect the relative magnitudes of pairs of significant digits, starting from the most significant position. If the two digits of a pair are equal, we compare the next lower significant pair of digits. The comparison continues until a pair of unequal digits is reached. If the corresponding digit of A is 1 and that of B is 0, we conclude that A > B.
If the corresponding digit of A is 0 and that of B is 1, we have A < B

$$(A > B) = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$$
$$(A < B) = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1' + x_3 x_2 x_1 A' n_0 B_0'$$

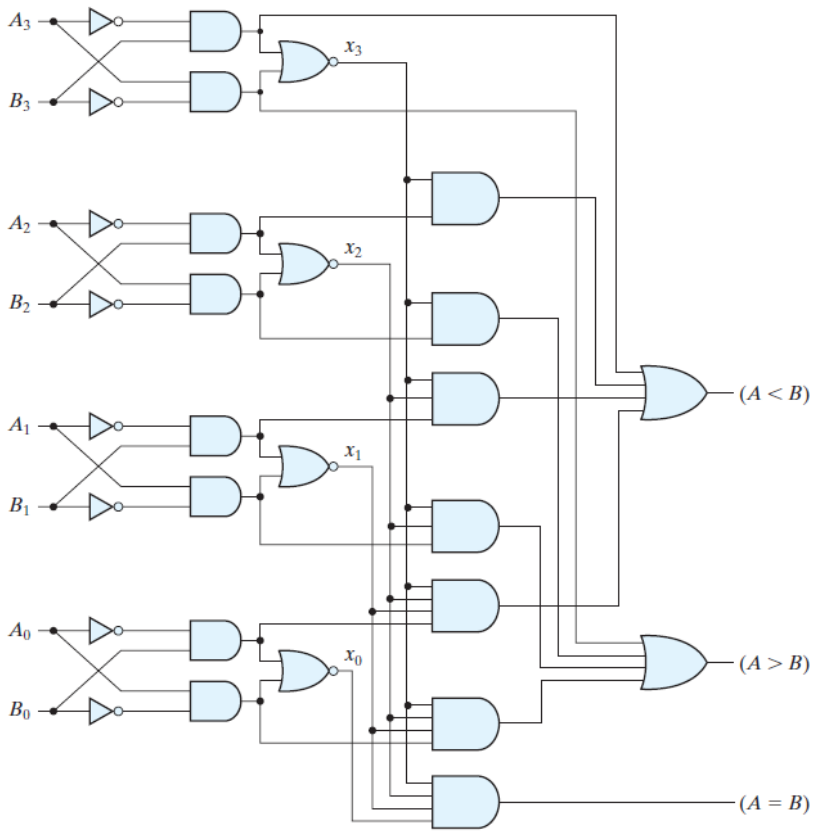
4 bits Comparator

$$x_i = A_iB_i + A_i'B_i' \quad \text{for } i = 0, 1, 2, 3$$

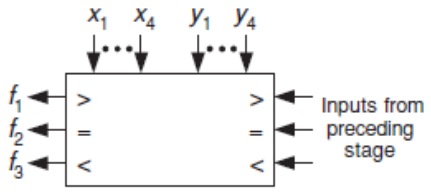
$$(A = B) = x_3x_2x_1x_0$$

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

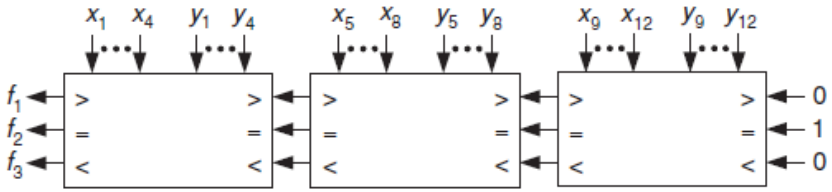
$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1' + x_3x_2x_1A'n_0B_0'$$



12-bits Comparator



(*a*) A 4-bit comparator.



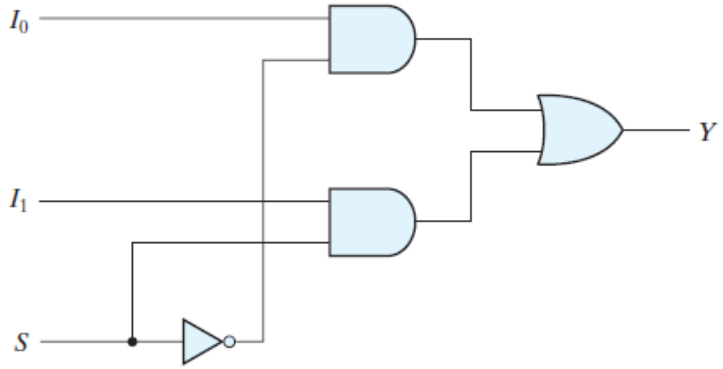
(*b*) A 12-bit comparator.

Initial conditions are inserted at the inputs of the comparator corresponding to the least significant bits in such a way that the outputs of this comparator will depend only on the values of its own x's and y's

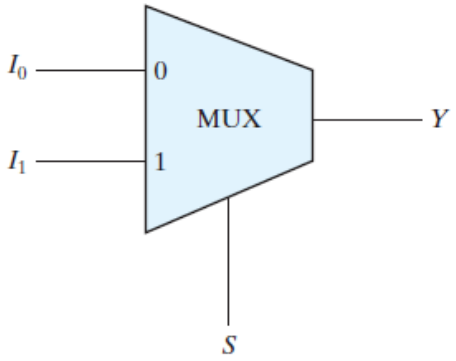
Data selectors/Multiplexers

- Multiplexer is an electronic switch that can **connect one out of n inputs to the output**.
- A data selector has n data input lines D_0, D_1,...,D_{n-1}, m select digit inputs s_0, s_1,...,s_{m-1}, and one output.
- The m select digits form a binary select number ranging from 0 to 2^{m-1}, and when this number has the value k then Dk is connected to the output.
- The number of select digits must equal m = log2 n, so that it can identify all the data inputs.

2-to-1 Multiplexer

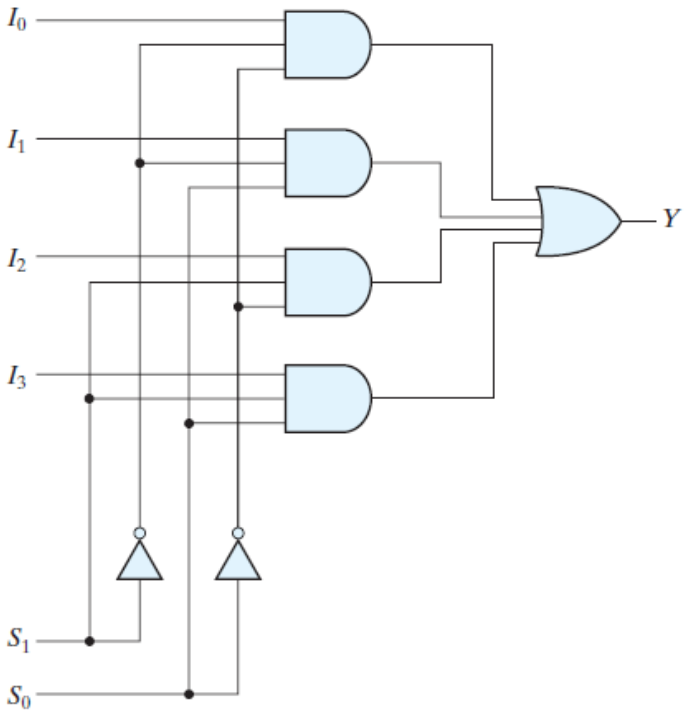


(a) Logic diagram



(b) Block diagram

4-to-1 Multiplexer



(a) Logic diagram

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

Implementing switching functions with data selectors

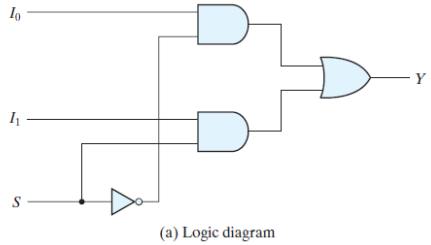
- Implementing of the arbitrary switching functions is an important application of data selectors.

- For example: we show how functions of two variables can be implemented by data selector

  if s=0, then $z=D_0$

  if s=1, then $z=D_1$

  $z = sD_1 + s'D_0$.



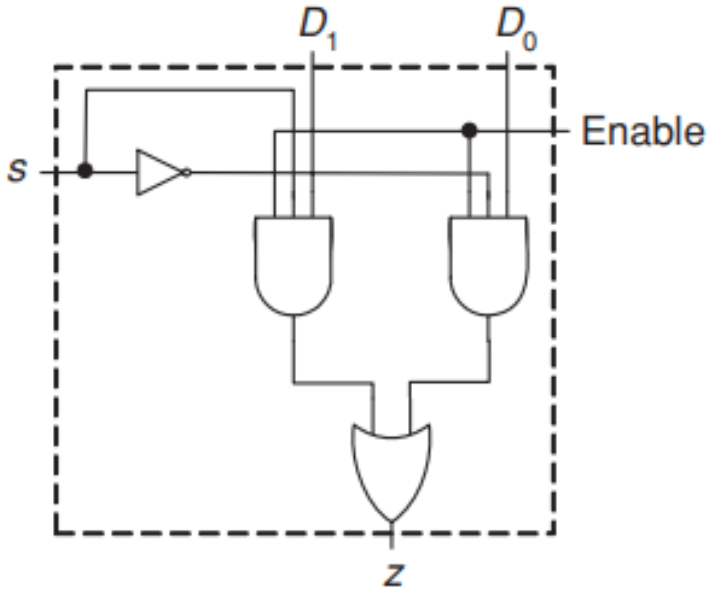
(a) Logic diagram

- Implementation of EXCLUSIVE-OR operation $A \oplus B$, s=A, $D_1=B'$, $D_0=B$

  $z = AB' + A'B = A \oplus B$

- Similarly, NAND operation $z = A' + B'$. S=A, $D_1=B'$, $D_0=1$
- Any logic gate can be realized using a 2-to-1 MUX

Implementing two variable function with data selector



$z = sD_1 + s'D_0$.

If $s = A$, $B = D_0$, and $B' = D_1$ then $z = A \oplus B$.

If $s = A$, $D_0 = 1$, and $D_1 = B'$ then $z = A' + B'$.

Boolean Function Implementation

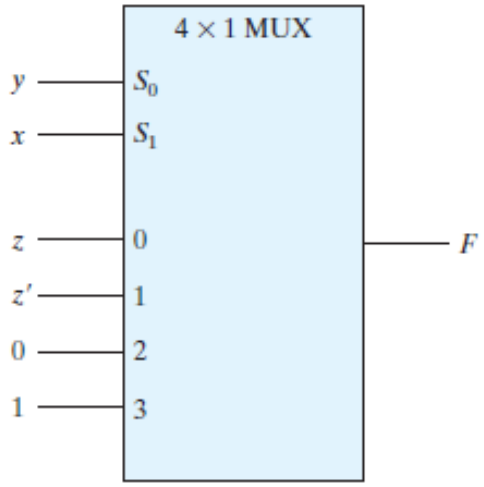
- Implementing a Boolean function of *n* variables with a multiplexer that has *n* - 1 selection inputs.
- The first *n* - 1 variables of the function are connected to the selection inputs of the multiplexer.
- The remaining single variable of the function is used for the $2^{(n-1)}$data inputs.
- If the single variable is denoted by *z* , each data input of the multiplexer will be *z* , *z'*, 1, or 0

Boolean Function Implementation

$$F(x, y, z) = \Sigma(1, 2, 6, 7)$$

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

Boolean Function Implementation

$$F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$$

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1