COEN 312 DIGITAL SYSTEMS DESIGN - LECTURE NOTES

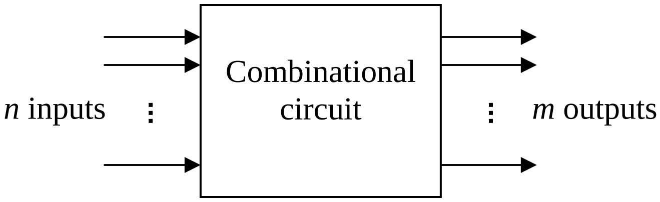
Concordia University

Chapter 4: Combinational Logic

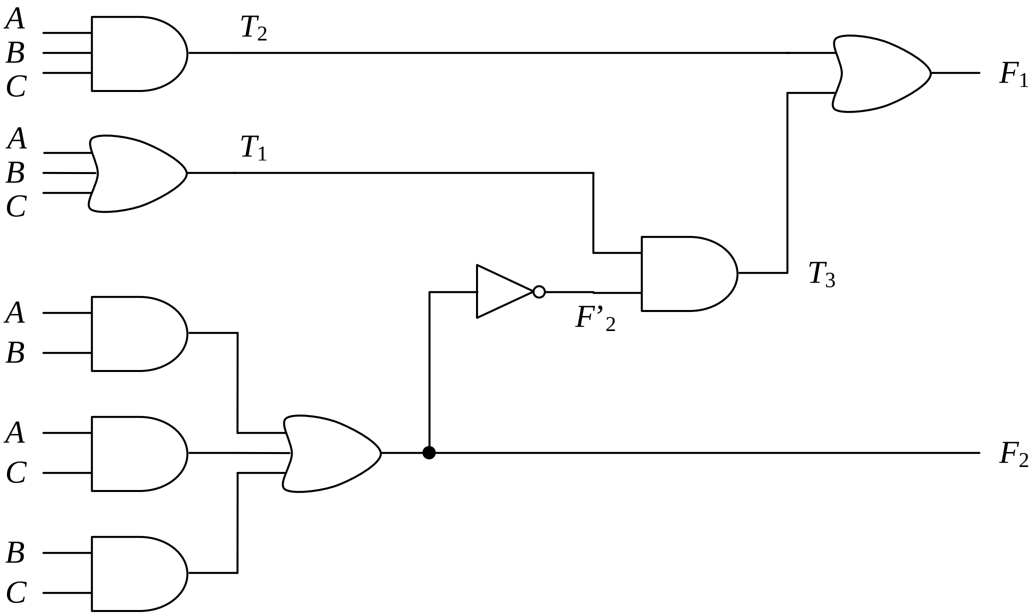
NOTE: For more examples and detailed description of the material in the lecture notes, please refer to the main textbook:
Digital Design 3rd Edition, By Morris Mano, Publisher Prentice Hall, 4th Edition
All examples used in the lecture notes are from the above reference.

Combinational Circuits

- A combinational circuit consists of input variables, output variables, and logic gates that transform binary information from the input to the output.



- A combinational circuit cannot have any storage elements (registers) or any feedback paths (connections from the output of one gate to the input of a second gate that directly or indirectly affects the input to the first gate).
- As an example, consider the following combinational circuit with 3 inputs and 2 outputs ($n = 3$, $m = 2$):



- If there are more than two levels of gates to generate a function, label the output of the corresponding gates and determine the Boolean expression for each of them (here, we have used the labels T_1, T_2, T_3, and F'_2 to find the Boolean expression for F_1).

$$F_2 = A.B + A.C + B.C$$
$$T_1 = A + B + C$$
$$T_2 = A.B.C$$
$$T_3 = F'_2.T_1$$
$$F_1 = T_3 + T_2$$

- By using the properties of Boolean algebra, the expression for F_1 will be equal to:

$$F_1 = A'.B.C' + A'.B'.C + A.B'.C' + A.B.C$$

- The truth table for the outputs of this combinational circuit can be obtained by using the above expressions for F_1 and F_2, or by using the labeled gate outputs in the truth table and obtaining $F_1 = T_3 + T_2$ and $F_2 = A.B + A.C + B.C$ for different combinations of the input variables as follows:

A	***B***	***C***	***F_2***	***F'_2***	***T_1***	***T_2***	***T_3***	***F_1***
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

- This is in fact the logic circuit for a full adder ($A+B+C=[F_2][F_1]$).

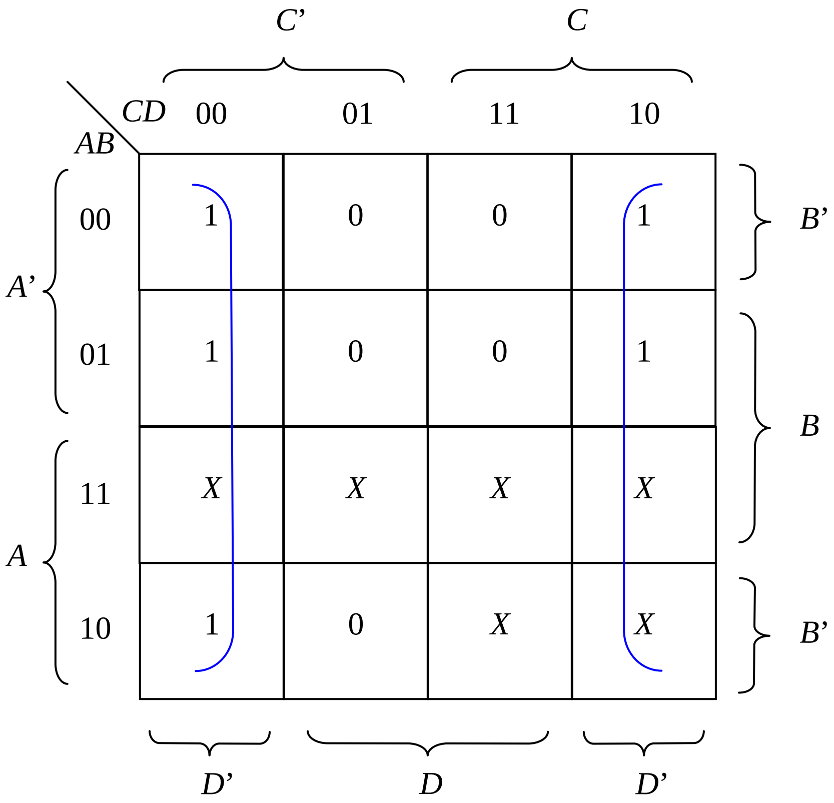
Design Procedure

- Important design constraints include the number of gates, number of inputs to a gate, propagation time of the signal through the gates, number of interconnections, etc.

Example

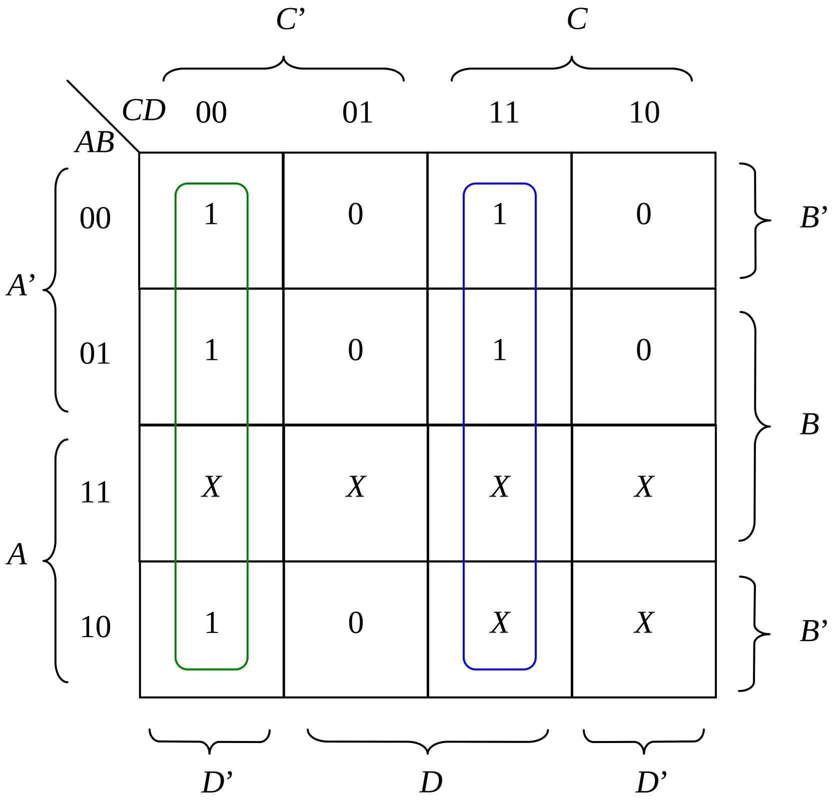
- Find a combinational circuit, which converts the binary coded decimal (BCD) to the excess-3 code for the decimal digits.

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



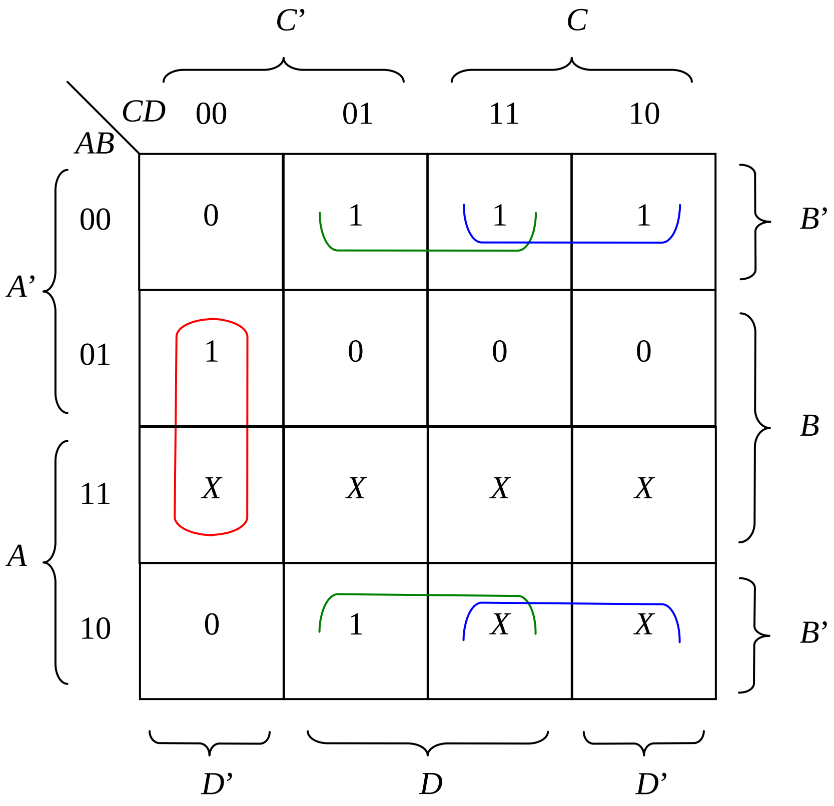
AB \ CD	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	*X*	*X*	*X*	*X*
10	1	0	*X*	*X*

- Simplified expression: $z = D'$



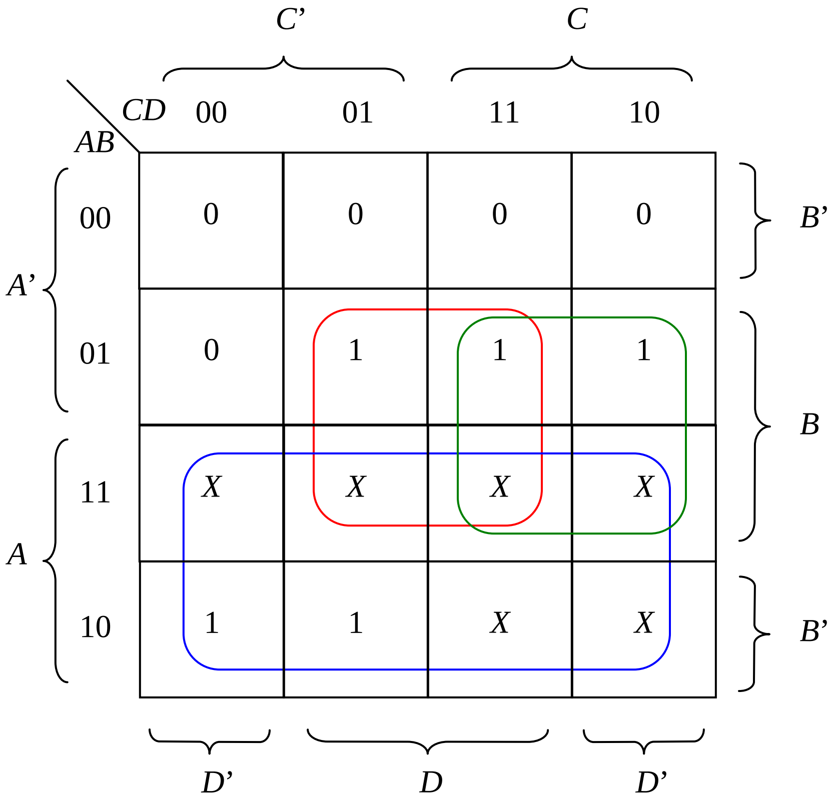
AB \ CD	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	*X*	*X*	*X*	*X*
10	1	0	*X*	*X*

- Simplified expression: $y = C.D + C'.D'$



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

- Simplified expression: $x = B'.C + B'.D + B.C'.D'$



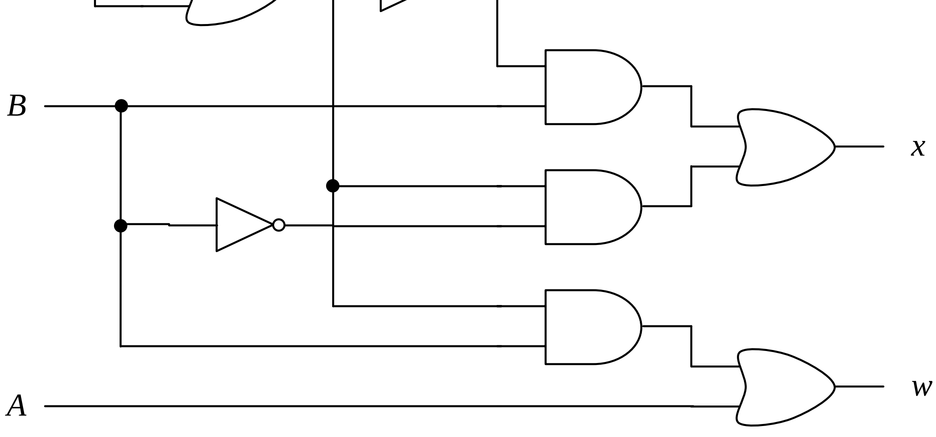
AB \ CD	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

- Simplified expression: $w = A + B.C + B.D$

- Using the above simplified expressions in sum of products, one can implement this combinational circuit using seven AND gates and three OR gates.
- One can use the properties of Boolean algebra to change the expressions in a way that some of the terms are shared between the output functions to make the implementation even simpler (fewer gates) as follows:

$$\begin{aligned}
z &= D' \\
y &= C.D + C'.D' \\
&= C.D + (C + D)' \\
x &= B'.C + B'.D + B.C'.D' \\
&= B'.(C + D) + B.C'.D' \\
&= B'.(C + D) + B.(C + D)' \\
w &= A + B.C + B.D \\
&= A + B.(C + D)
\end{aligned}$$

- These expressions can be implemented using four AND gates and four OR gates and one inverter (we assume that the inputs are available in both normal and complement forms).

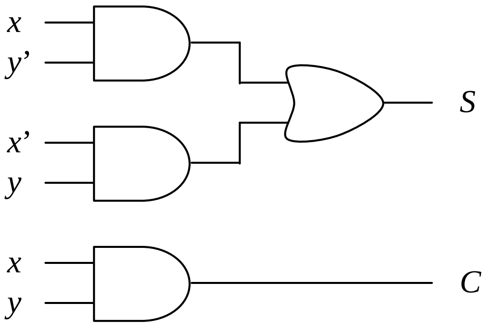


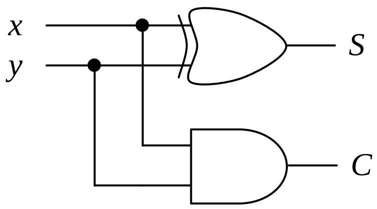
Binary Adder-Subtractor

- A combinational circuit that implements the addition of two bits is referred to as a half adder.
- A combinational circuit that implements the addition of three bits (two significant bits and a previous carry) is referred to as a full adder.
- A full adder can be implemented by using two half adders.
- Assuming that *S* and *C* denote the sum (the lower significant bit in the binary sum) and the carry (the higher significant bit in the binary sum) of the two input variables of a half adder, we will have:

x	***y***	***C***	***S***
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

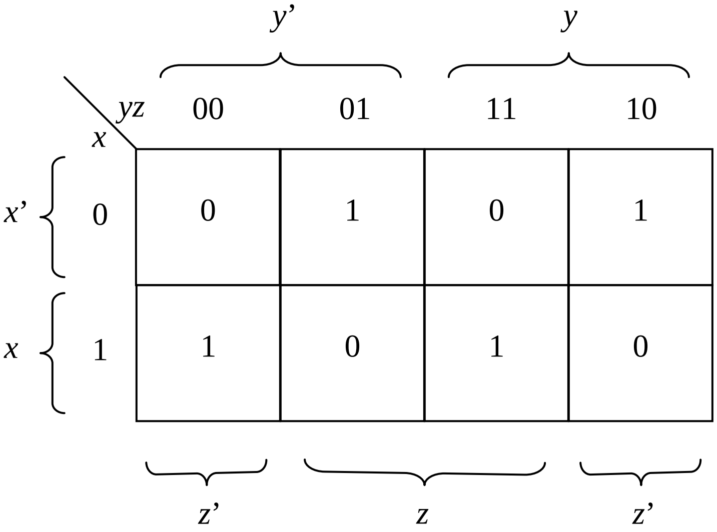
$$
\begin{aligned}
S &= x'.y + x.y' \\
&= x \oplus y \\
C &= x.y
\end{aligned}
$$





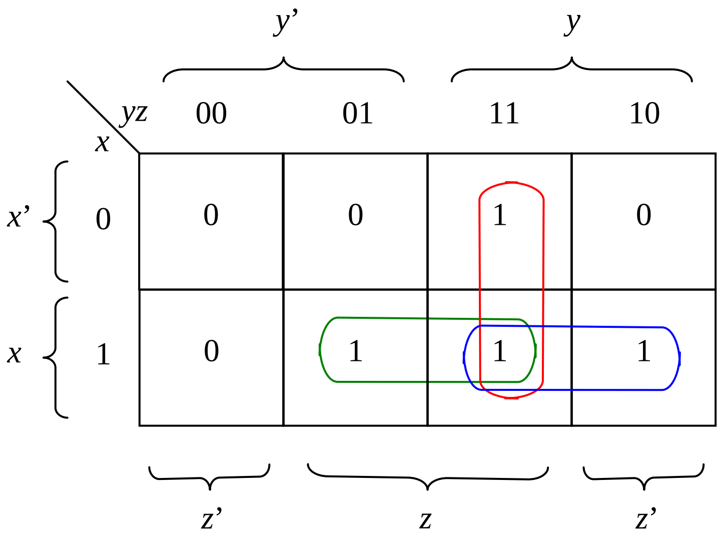
- For a full adder, where the third input *z* denotes the carry corresponding to the addition of the previous lower significant bits, we have:

x	***y***	***z***	***C***	***S***
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



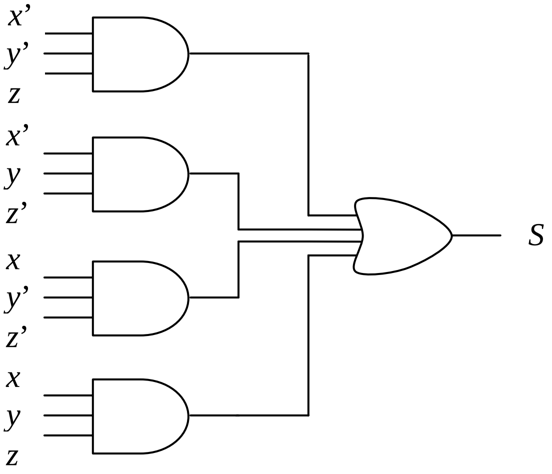
x \ *yz*	00	01	11	10
0	0	1	0	1
1	1	0	1	0

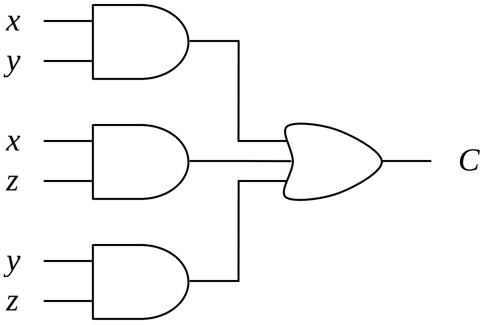
- Simplified expression: $S = x'.y'.z + x'.y.z'+x.y'.z'+x.y.z$ or: $S = x \oplus y \oplus z$



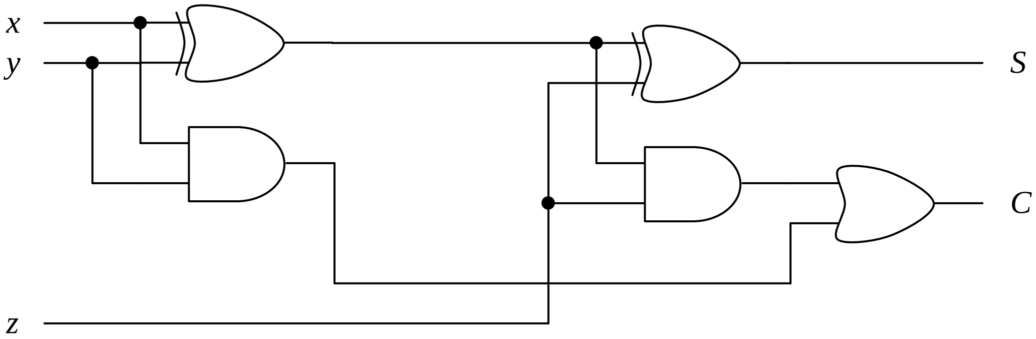
yz x	00	01	11	10
0	0	0	1	0
1	0	1	1	1

- Simplified expression: $C = x.y + x.z + y.z$ or $C = x.y + x.y'.z + x'.y.z = x.y + (x \oplus y).z$
- We will have the following logic circuits:

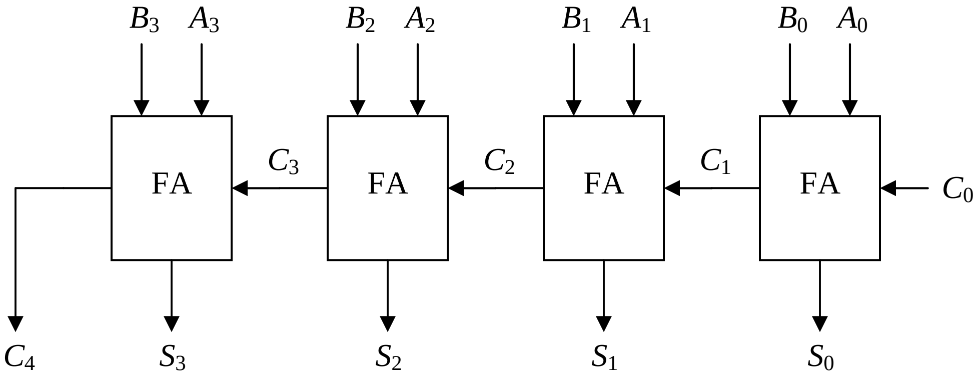




- Using the second expression given for *S* and *C*, we can implement the full adder with two half adders and an OR gate as shown below:



- One can construct an n-bit binary adder with n full adders in series, by connecting each output carry to the input carry of the next higher-order full adder.
- For example, a four-bit binary adder is given below:

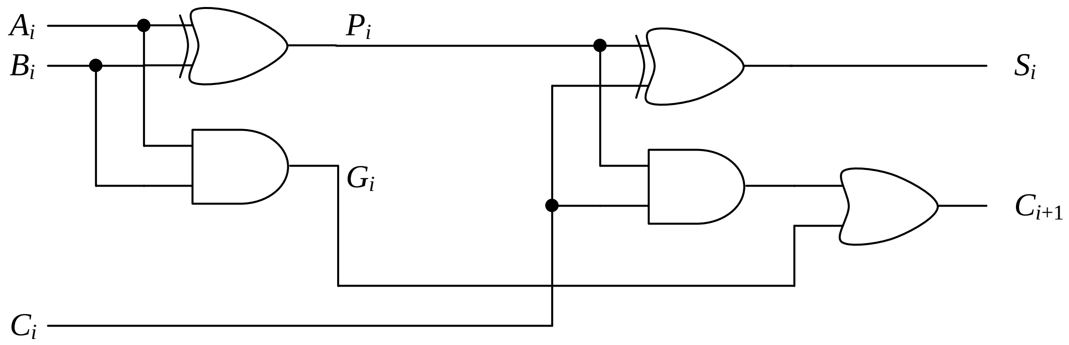


- For $A = A_3A_2A_1A_0 = 1011$ and $B = B_3B_2B_1B_0 = 0011$, we will have:

Subscript *i*:	**3**	**2**	**1**	**0**	
Input Carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output Carry	0	0	1	1	C_{i+1}

- Note that in this circuit we have nine inputs $A_0, A_1, A_2, A_3, B_0, B_1, B_2, B_3,$ and C_0, and so the implementation of the corresponding circuit using the method of Chapter 3 would require a truth table with $2^9 = 512$ rows (if we ignore the input carry C_0 since it must be zero, we will have 256 rows).
- The signal must propagate through the gates before the correct output sum is available and the longest propagation delay time in an adder is the time it takes for the carry to propagate through the full adders.

- The output of each full adder will not settle to its final value until the input carry is available from the previous stage.



- For a full adder, the carry propagates through an AND gate and an OR gate which constitutes two gate levels, which implies that in an *n*-bit adder, the carry propagates from input to output through 2*n* gate levels.
- Use faster gates with reduced delays.
- Reduce the carry delay time by increasing the equipment complexity properly.
- One of the popular methods for this purpose is to use carry lookahead as follows:

$$P_i = A_i \oplus B_i$$
$$G_i = A_i . B_i$$

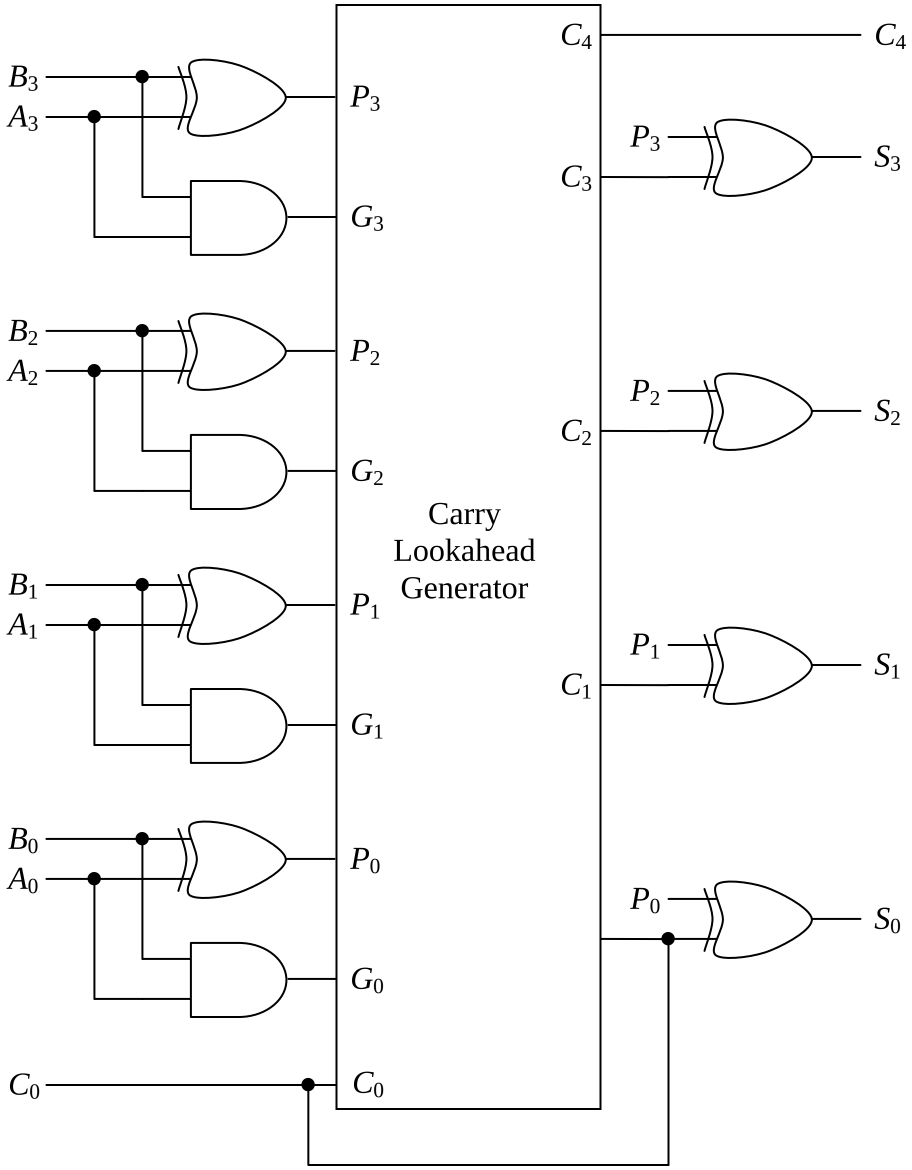
- P_i and G_i are referred to as carry propagate and carry generate respectively.
- The output sum and carry can then be written as:

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i . C_i$$

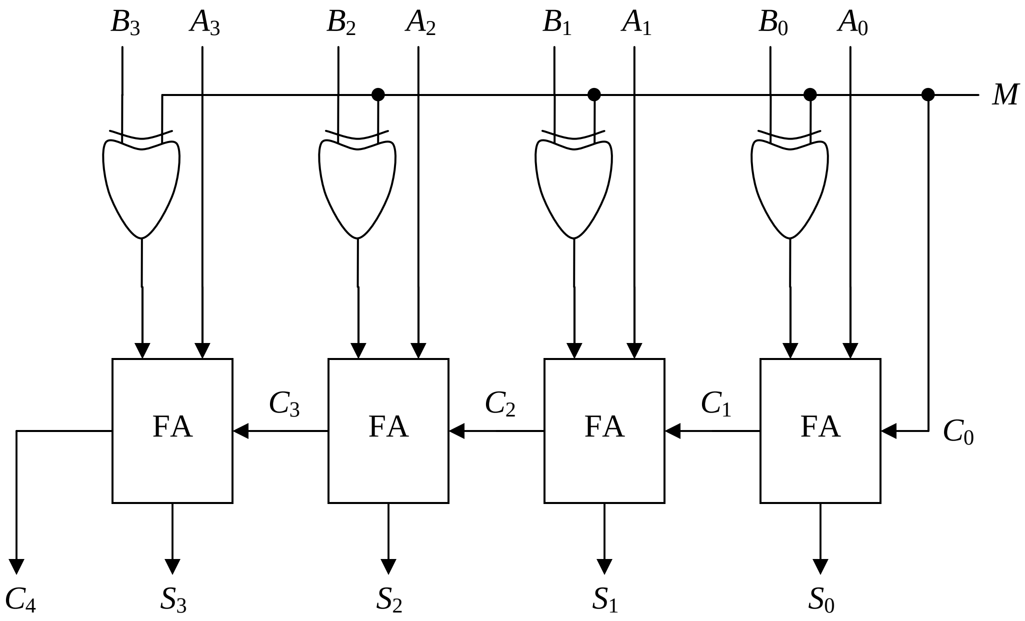
- We now use iterations to obtain each output carry C_{i+1} directly from the input carry C_0 as follows:

$$
\begin{aligned}
C_1 &= G_0 + P_0.C_0 \\
C_2 &= G_1 + P_1.C_1 = G_1 + P_1.(G_0 + P_0.C_0) = G_1 + P_1.G_0 + P_1.P_0.C_0 \\
C_3 &= G_2 + P_2.C_2 = G_2 + P_2.(G_1 + P_1.G_0 + P_1.P_0.C_0) = G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.C_0 \\
C_4 &= G_3 + P_3.C_3 = G_3 + P_3.(G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.C_0) \\
&= G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.C_0
\end{aligned}
$$

- Since all these expressions are in standard form, one can use a two-level circuit to implement this combinational system.
- A 4-bit adder with a carry lookahead scheme is given below:



- Binary subtraction $A-B$ can be done by adding A to the 2’s complement of B (which can be obtained by adding one to the 1’s complement).
- The logic circuit for a 4-bit adder/subtractor is given as follows:

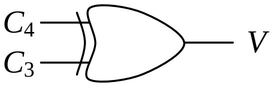


- The mode input M controls the output.
- When M is 0, then $B\oplus 0=B$ and the input carry is 0 which means that the output of the circuit is equal to A plus B.
- When M is 1, we have $B\oplus 1=B'$ and the input carry is 1, which implies that the output is equal to A plus the 1’s complement of B plus 1. This means that the output is A minus B (provided there is no overflow).
- Addition of two *n*-bit numbers which are both positive or both negative, may result in an overflow, i.e., the result may exceed the capacity of an *n*-bit register. This is shown for 8-bit registers below:

Carries:	0	1	
+70		0	1000110
+80		0	1010000
+150		1	0010110

Carries:	1	0	
-70		1	0111010
-80		1	0110000
-150		0	1101010

- In general, an overflow can be detected by applying exclusive-OR operation on the input carry and output carry of the sign bit position (the leftmost digit).
- Note that if the last two carries were equal, there would be no overflow.
- For example, for the 4-bit adder subtractor, we will have:



- If the numbers are unsigned (the numbers are considered positive and the leftmost bit does not represent the sign), the circuit performs A minus B if $A \geq B$ or the 2's complement of B minus A if $A \leq B$.
- For two unsigned numbers, the output carry of the leftmost bits detects a carry after addition or a borrow after subtraction.

Carry:	1	
70		1000110
80		1010000
150		0010110

Decimal Adder

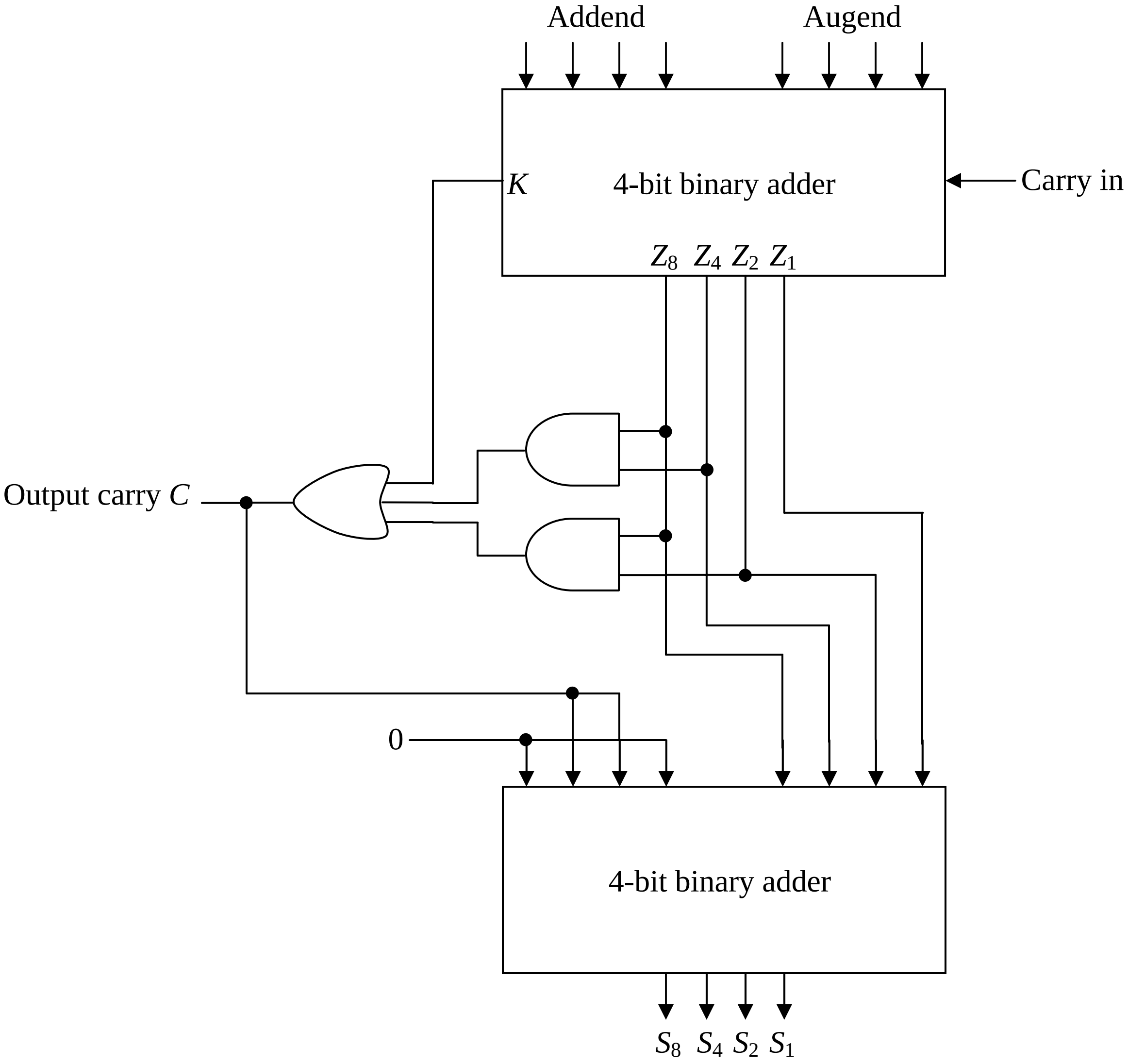
- The most common way to code a decimal number is binary coded decimal (BCD), where the decimal number is represented by encoding each digit in binary form.
- Applying two BCD digits to a 4-bit binary adder will result in a number ranging from 0 to 9+9+1=19 as shown in the following table:

Binary Sum					**BCD Sum**					**Decimal**
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

- The left side of the table gives the output of a 4-bit binary adder but we want the output to be in a BCD form.
- From the entries of the table, it can be concluded that a conversion from binary sum to BCD representation is needed if K=1 or Z_8 and Z_4 are both equal to 1 or Z_8 and Z_2 are both equal to 1.
- The Boolean expression to show the condition for a correction is:

$C = K + Z_8.Z_4 + Z_8.Z_2$

- It can be easily seen that a correction can be made by adding 0110 to the binary sum and providing an output carry for the next stage.
- The following figure shows a BCD adder (the input and the output are BCD digits).



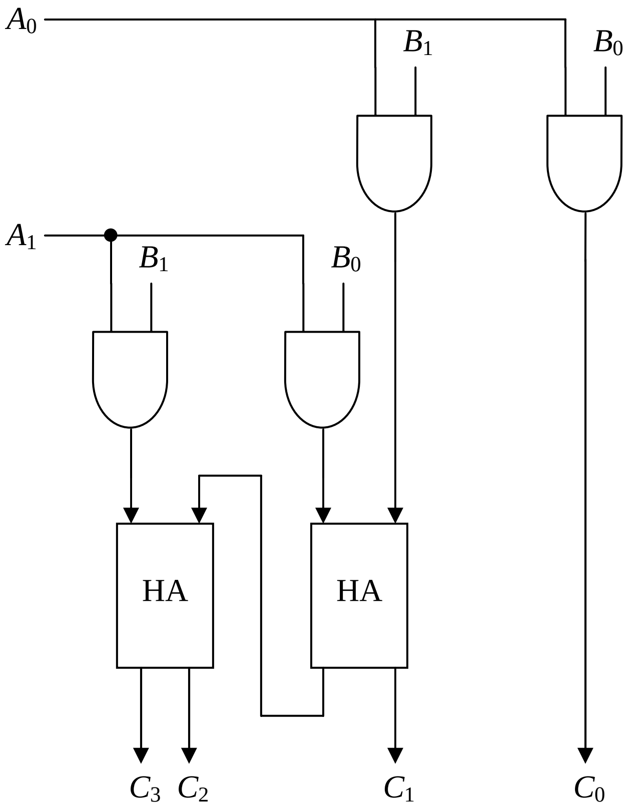
- From this circuit it can be seen that if the output carry (C) is equal to 1, binary number 0110 is added to the binary sum.
- The output carry (C) is, in fact, equal to the output carry generated from the bottom adder.

Binary Multiplier

- Multiplication of binary numbers is similar to the multiplication of decimal numbers as shown below for a 2-bit binary number:

$$\begin{array}{cccc} & & B_1 & B_0 \\ & & A_1 & A_0 \\ \cline{3-4} & & A_0 . B_1 & A_0 . B_0 \\ & A_1 . B_1 & A_1 . B_0 & \\ \hline C_3 & C_2 & C_1 & C_0 \end{array}$$

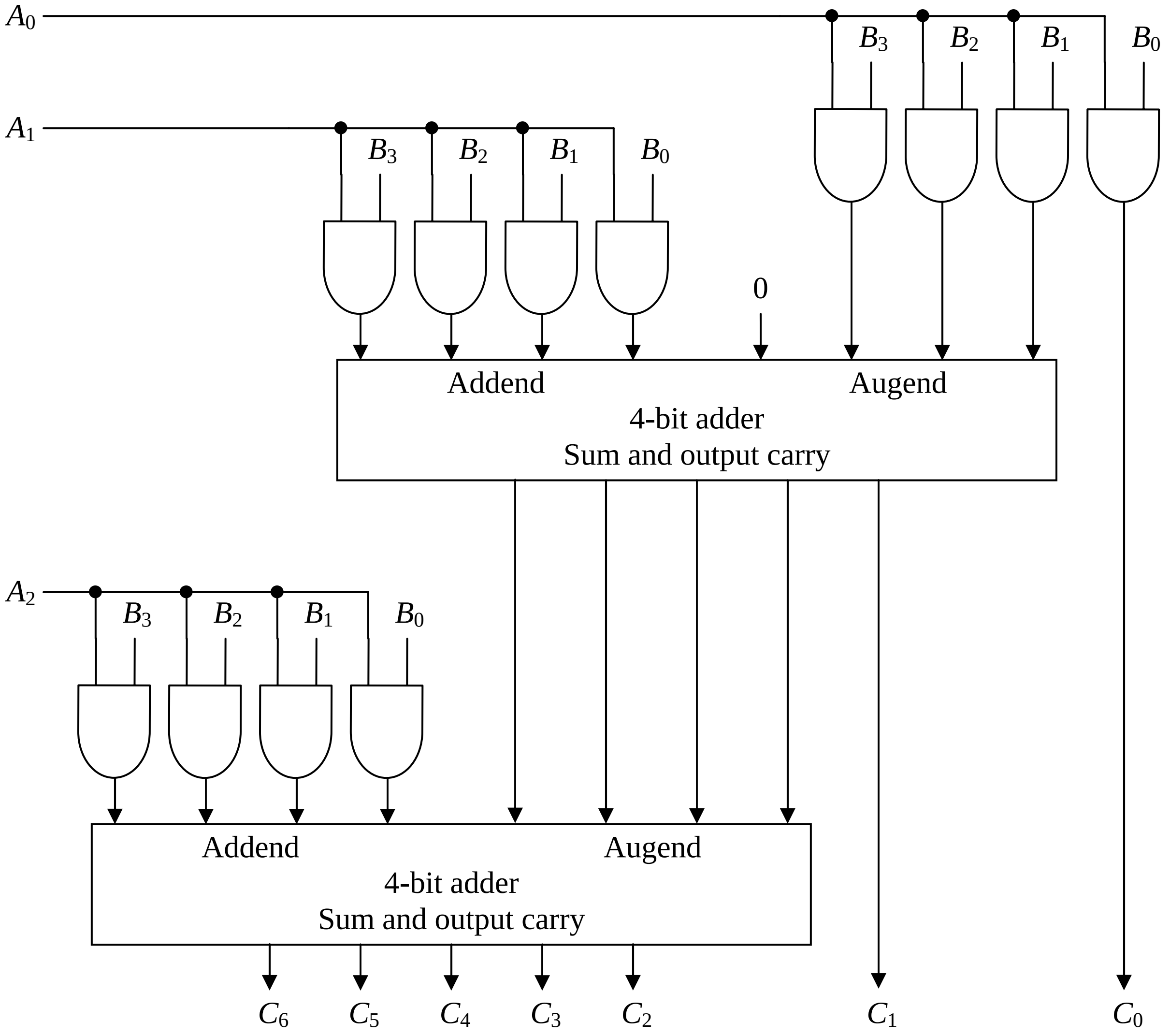
- So, multiplication of two 2-bit numbers can be implemented by a combinational circuit with two half adders as follows:



- Assume now that the multiplier has 3 bits and the multiplicand has 4 bits.

$$\begin{array}{ccccccc}
 & & & B_3 & B_2 & B_1 & B_0 \\
 & & & & A_2 & A_1 & A_0 \\
\cline{4-7}
 & & & A_0.\,B_3 & A_0.\,B_2 & A_0.\,B_1 & A_0.\,B_0 \\
 & & A_1.\,B_3 & A_1.\,B_2 & A_1.\,B_1 & A_1.\,B_0 & \\
 & A_2.\,B_3 & A_2.\,B_2 & A_2.\,B_1 & A_2.\,B_0 & & \\
\hline
C_6 & C_5 & C_4 & C_3 & C_2 & C_1 & C_0
\end{array}$$

- In this case, we will need two 4-bit adders and 12 AND gates as follows:



- In general, if the multiplier has J bits and the multiplicand has K bits, we will need $J-1$ K-bit adders and $J \times K$ AND gates and the product will have $J+K$ bits.

Magnitude Comparator

- A magnitude comparator is a combinational circuit that compares two numbers *A* and *B* and determines whether *A* > *B*, *A* = *B*, or *A* < *B*.
- Comparison operation should start by comparing the most significant bits of *A* and *B*.
- For example, consider two numbers *A* and *B* as follows:

$$A = A_3A_2A_1A_0$$
$$B = B_3B_2B_1B_0$$

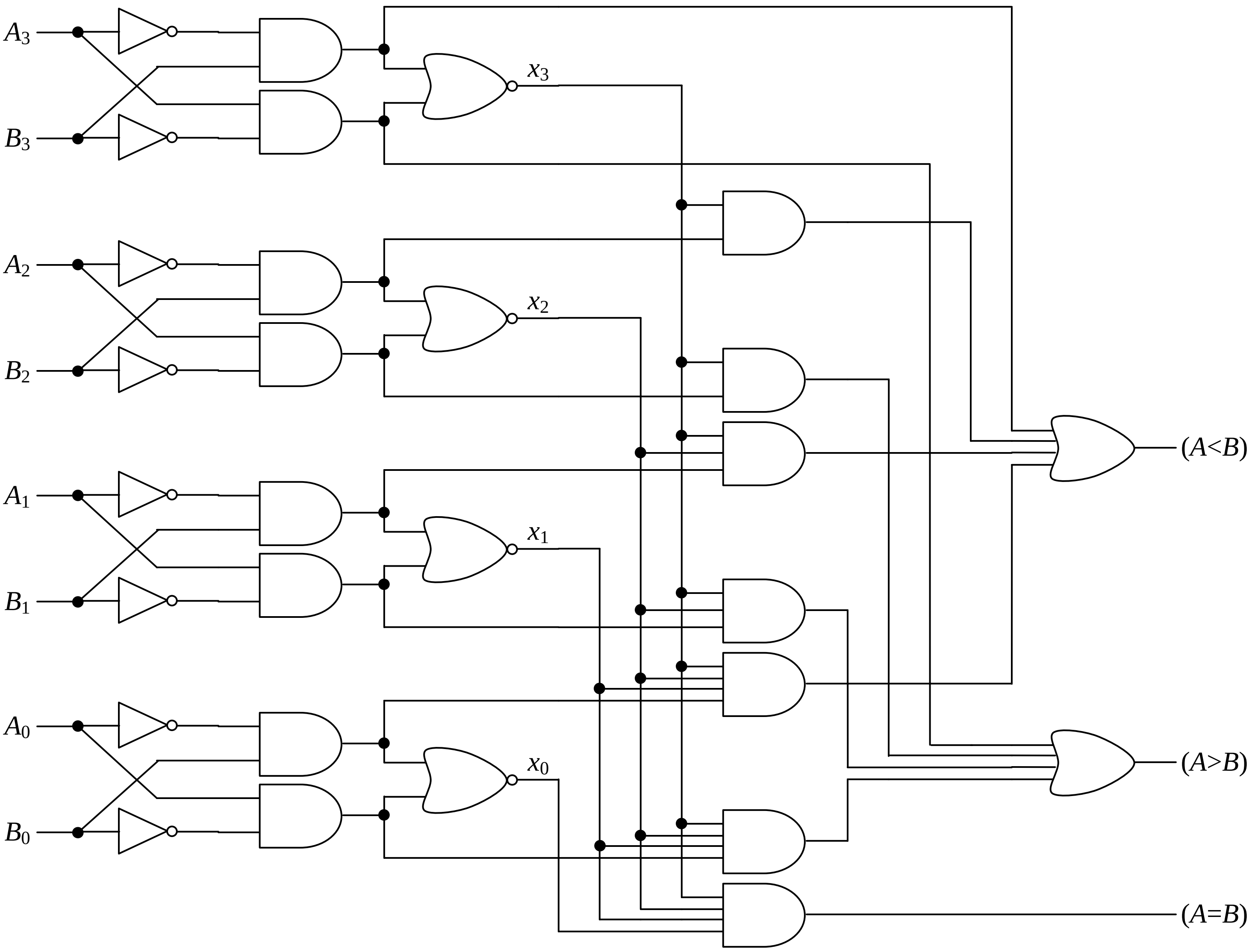
- Define the following functions (which are the bib-by-bit exclusive-NOR operations on *A* and *B*):

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

- It can easily be verified that:

$$A = B \Rightarrow x_3.x_2.x_1.x_0 = 1$$
$$A > B \Rightarrow A_3.B_3' + x_3.A_2.B_2' + x_3.x_2.A_1.B_1' + x_3.x_2.x_1.A_0.B_0' = 1$$
$$A < B \Rightarrow A_3'.B_3 + x_3.A_2'.B_2 + x_3.x_2.A_1'.B_1 + x_3.x_2.x_1.A_0'.B_0 = 1$$

- In the equations for inequalities, the most significant bits (with the same significance) in the two numbers are compared and if one of them is 1 and the other one is 0, the comparison is finished and only if they are equal, the next significant bits in the two numbers will be compared (this is represented by ANDing the x_i's with corresponding terms in the inequalities).
- A 4-bit magnitude comparator logic circuit is given below:

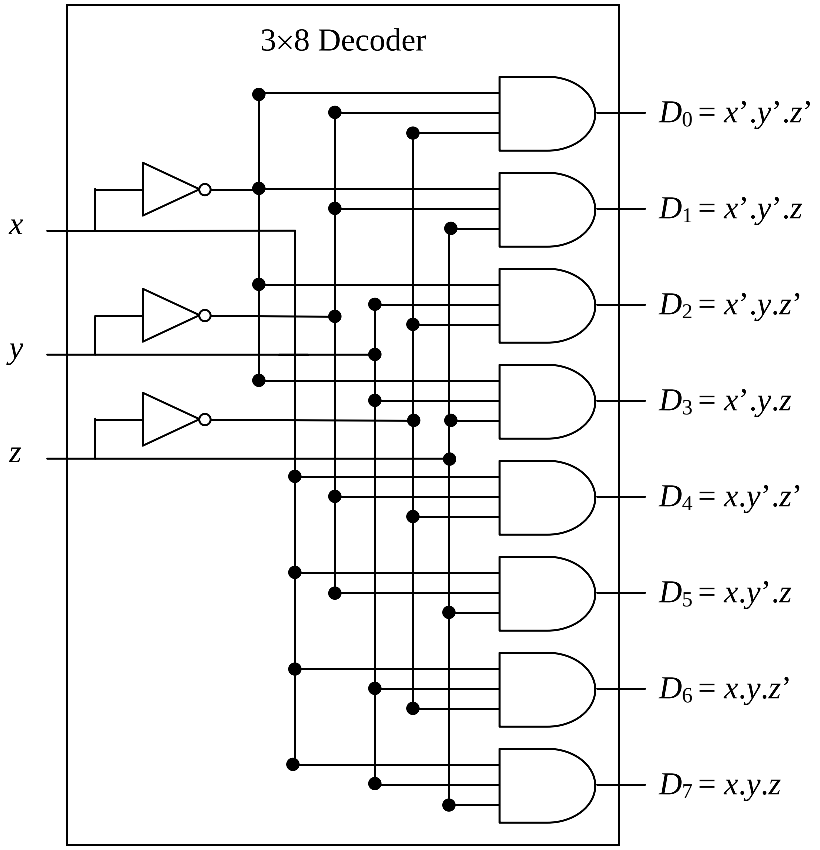


Decoders

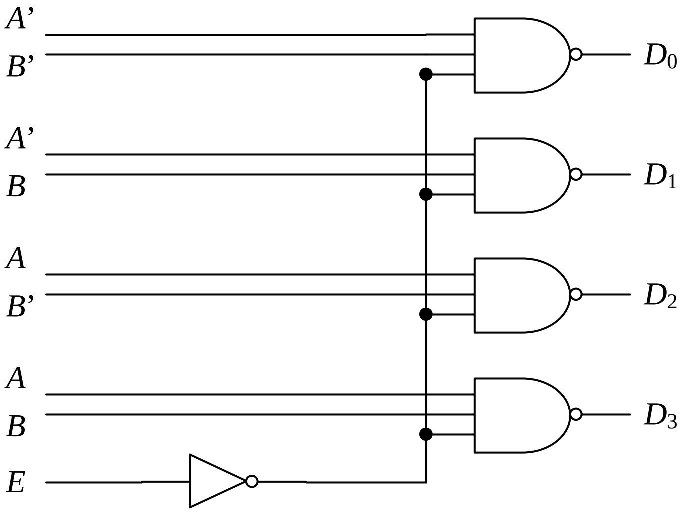
- Decoder circuits are used to decode encoded information.
- A decoder is a combinational circuit that has n inputs and a maximum of 2^n outputs.
- The truth table for a 3-to-8-line decoder is as follows:

Inputs			**Outputs**							
x	***y***	***z***	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
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

- The logic circuit for this decoder is shown in the following figure, where each output represents one of the minterms.



- Sometimes NAND gates are used instead of AND gates to generate complemented outputs.
- One or more enable inputs are often used in the decoders to control the operation of the circuit.
- A 2-to-4-line decoder consisting of NAND gates with an enable input is given below:

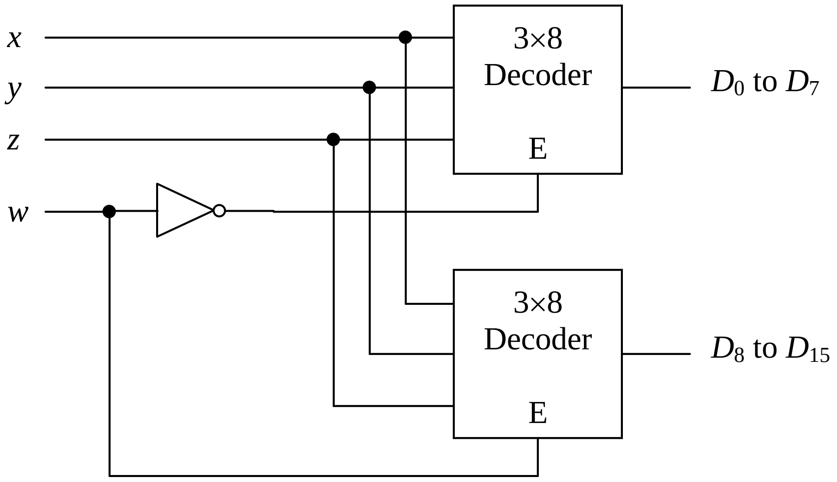


- The corresponding truth table is as follows:

E	***A***	***B***	D_0	D_1	D_2	D_3
1	*X*	*X*	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

- In general, the enable input signal *E* may be activated with a 0 or with a 1.
- Some decoders have more than one enable input, which must be equal to a specific binary number in order to enable the circuit.

- A demultiplexer is a circuit that receives data from a single input line and directs it to one of the output lines, which is determined by selection lines.
- One can build a demultiplexer by using a decoder with an enable line, as can be seen in the above example.
- In the above example, *E* is the data input line and *A* and *B* are selection inputs which determine which output will be the same as the input value *E* (this can easily be seen from the circuit).
- One can connect multiple decoders with enable lines to build a decoder with a greater number of outputs.
- For example, two 3-to-8-line decoders with enable lines can be connected to construct a 4-to-16-line decoder as follows:



- The minterms 0000 to 0111 are generated by the top decoder, while the bottom one generates minterms 1000 to 1111 as shown below:

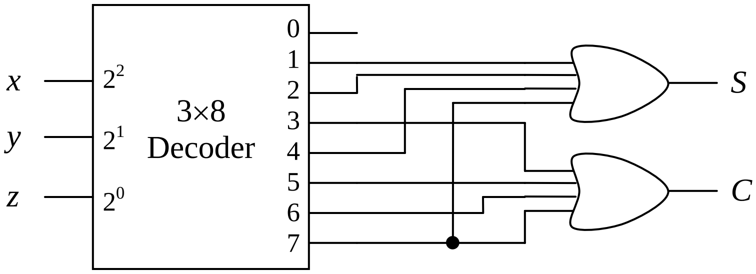
Inputs				**Outputs**							
w	x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	0	0	0	1	0
0	1	1	1	0	0	0	0	0	0	0	1

Inputs				**Outputs**							
w	x	y	z	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

- In the first table D_8 to D_{15} are all zero and D_0 to D_7 in the second one are all zero (assuming that the decoder is constructed by AND gates not NAND gates).
- In general, in logic components, enable inputs are used for interconnecting two or more components.

- Any *n*-input *m*-output combinational logic circuit can be implemented by using an *n*-to-2^n-line decoder (to generate different minterms) and *m* OR gates (to generate sum of products).
- For example, a full adder can be implemented as follows:

$$S(x,y,z)=\sum(1,2,4,7)$$
$$C(x,y,z)=\sum(3,5,6,7)$$



- If the number of minterms in the Boolean expression of a function is more than half of the number of the outputs of the decoder, it is easier to implement the complement function and then use a NOR gate instead of an OR gate, which will result in the same function with smaller number of inputs for the gate.
- If the decoder is implemented by NAND gates, one can use bubble technique and implement the function with a NAND gate instead of an OR gate in the output of the decoder.

Encoders

- An encoder’s operation is the inverse operation of a decoder.
- For example, an octal-to-binary encoder is given below:

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

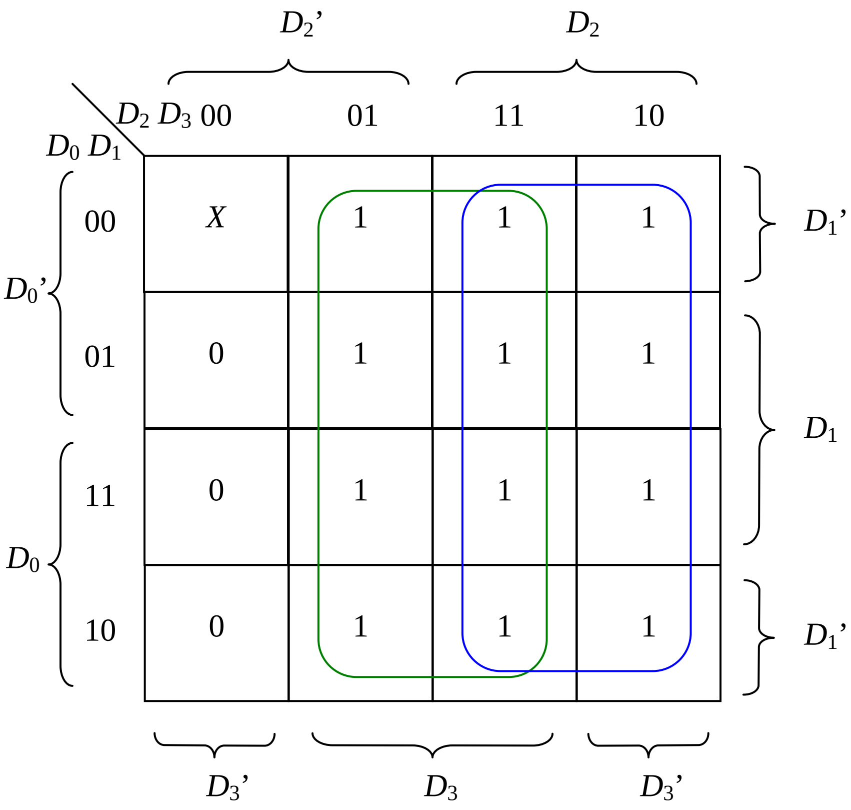
- The corresponding Boolean expressions are:

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

- The output *xyz*=000 can be generate from D_0 being equal to 1 or all inputs being equal to zero; a valid bit can be used to identify which one is the case.
- Also, an input priority must be established to guarantee that only one input is encoded (in case more than one input is turned on).
- For example, consider a 4-input priority encoder given below:

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	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

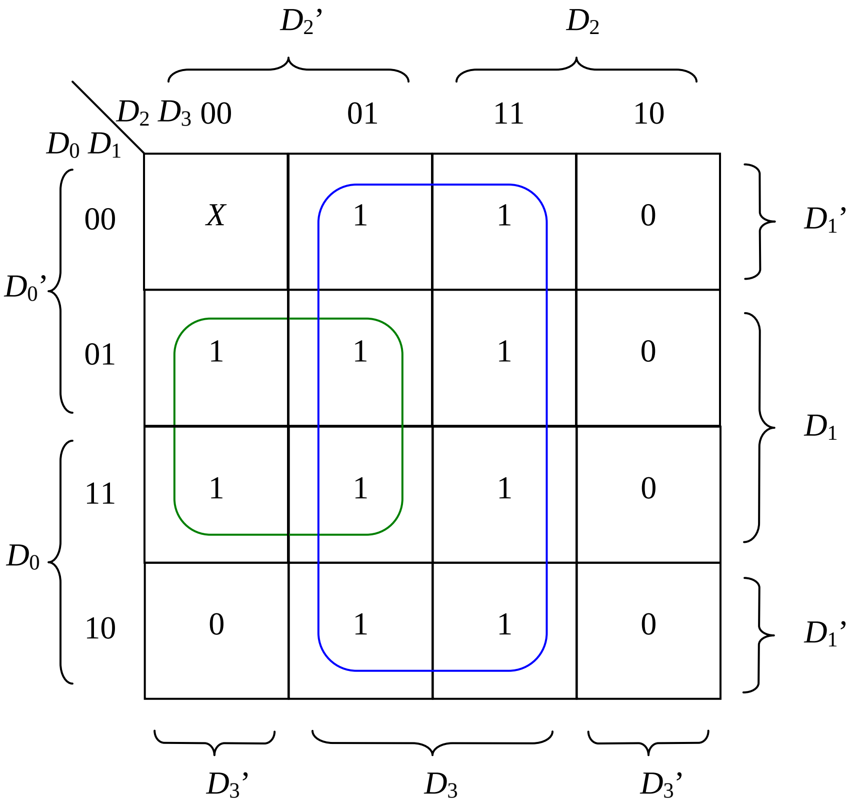
- *V* is a valid bit and is set to 1 when at least one of the inputs is equal to 1.
- Each *X* in the output lines represents a don't-care condition for that output, with the corresponding inputs.
- *X*'s in the input lines indicate that no matter what those variables are, the output is going to be the same.



D_0D_1 \ D_2D_3	00	01	11	10
00	*X*	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

- Boolean expression:

$$x = D_2 + D_3$$



$D_0 D_1$ \ $D_2 D_3$	00	01	11	10
00	*X*	1	1	0
01	1	1	1	0
11	1	1	1	0
10	0	1	1	0

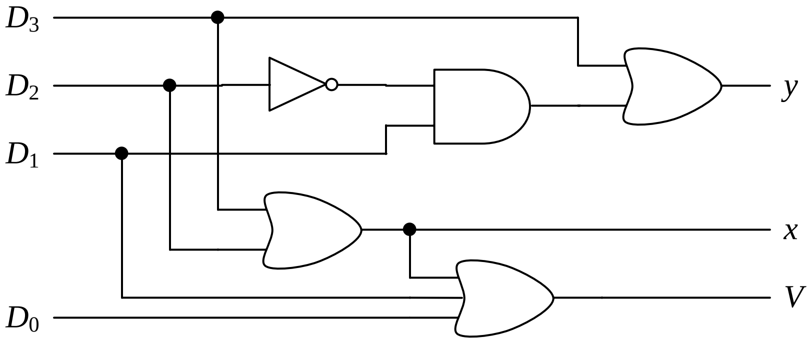
- Boolean expression:

$$y = D_3 + D_1.D_2'$$

- The valid bit *V* is simply equal to:

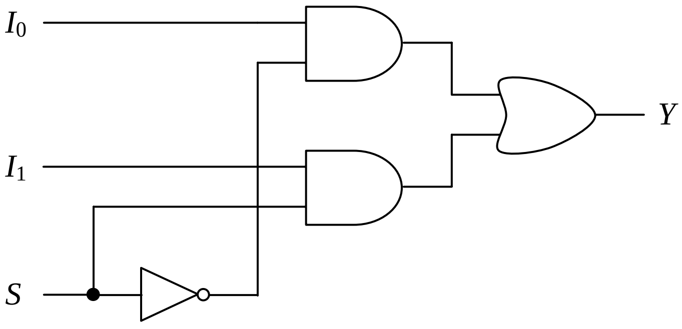
$$V = D_0 + D_1 + D_2 + D_3$$

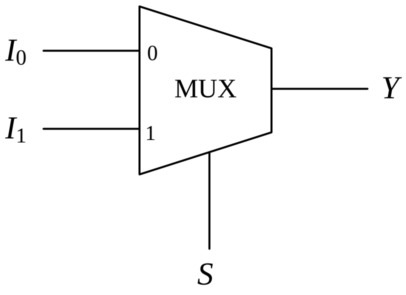
- The logic circuit of the above 4-input priority encoder is given below:



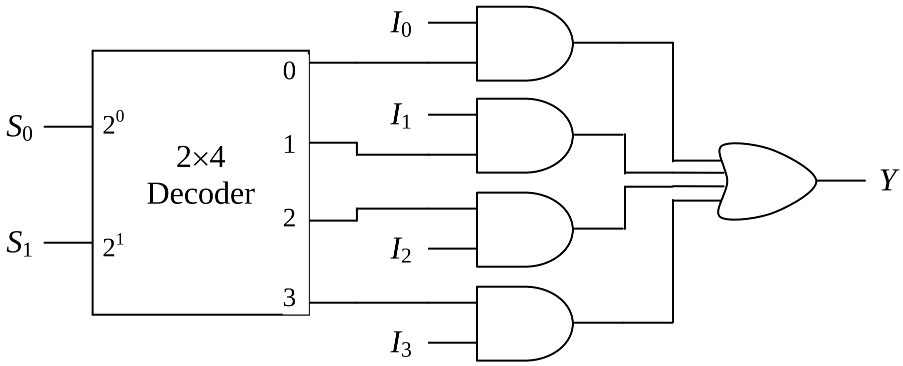
Multiplexers

- A multiplexer or data selector is a combinational circuit that chooses one of the inputs and reproduces the signal on this input at the output terminal.
- A 2-to-1-line multiplexer circuit and its block diagram are shown below:

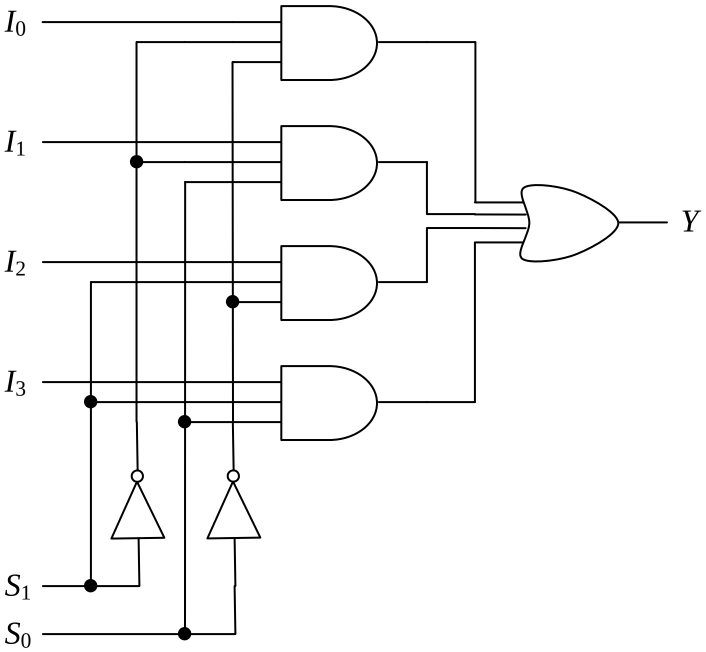




- I_0 and I_1 are the input lines, S is the selection line, and Y is the output.
- A 2^n-to-1-line Multiplexer is in fact a combination of a decoder, 2^n AND gates, and one OR gate.
- For example, a 4-to-1-line multiplexer can be built as follows:

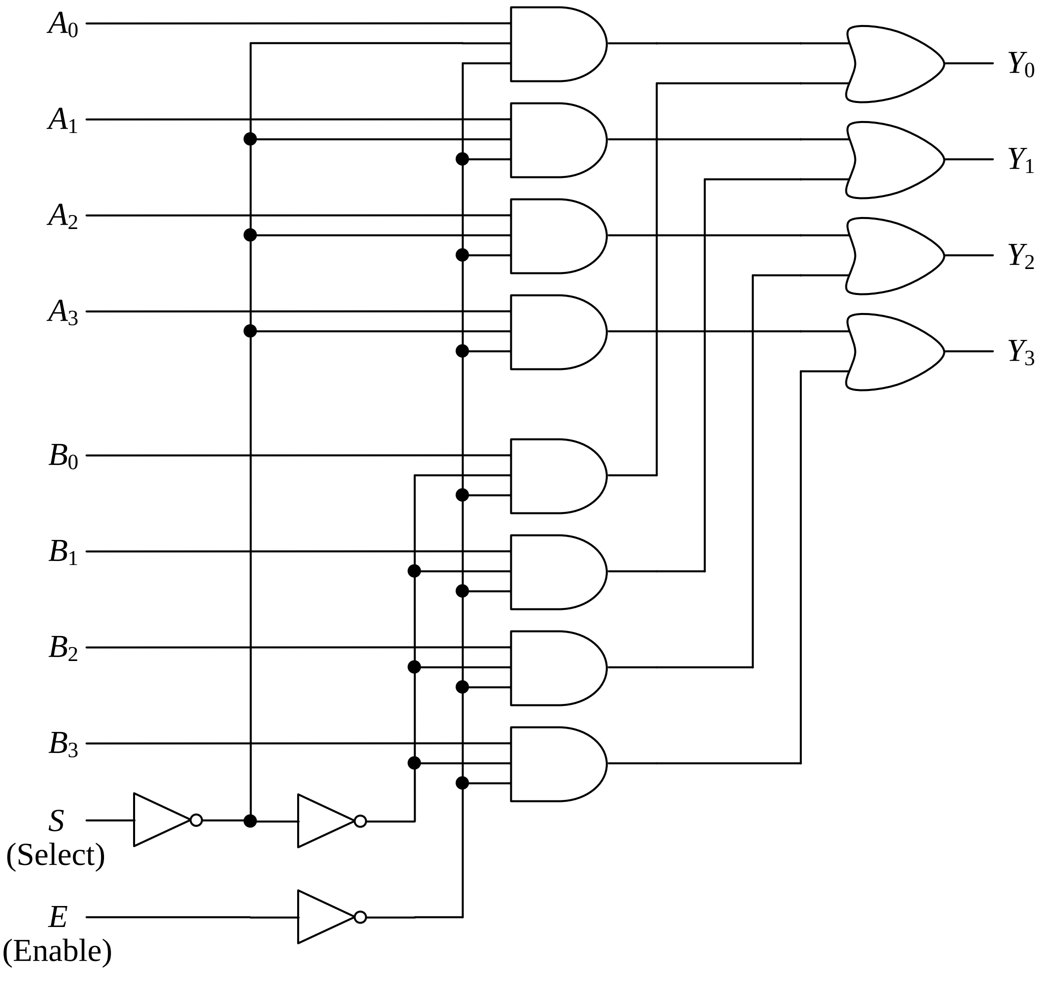


- One can combine the two-input AND gates with the two-input AND gates in the decoder and use three-input AND gates instead, as follows:



$\boldsymbol{S_1}$	$\boldsymbol{S_0}$	$\boldsymbol{Y}$
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- The AND gates in fact decode the selection lines S_0 to S_{n-1} and the multiplexer is a decoder with 2^n additional lines (I_0 to I_{2^n-1}), each of which goes to one of the AND gates.
- A multiplexer can have an enable line too.
- Multiplexer circuits can be combined to provide multiple bit selection logic.
- A circuit that selects one of two 4-bit sets of data lines can be provided by using 4 2-to-1-line multiplexers as shown below:



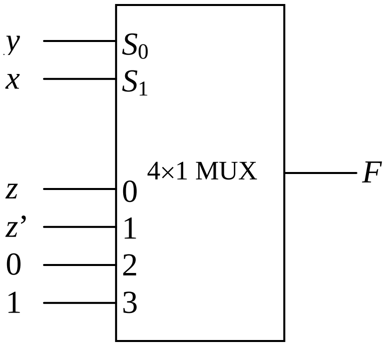
E	S	$Y_3Y_2Y_1Y_0$
1	X	All Zeros
0	0	$A_3A_2A_1A_0$
0	1	$B_3B_2B_1B_0$

- It can easily be verified that if S=0, the output lines $Y_0Y_1Y_2Y_3$ will be equal to $A_0A_1A_2A_3$, provided the enable line is active (here E=0).
- If S=1, then the output lines $Y_0Y_1Y_2Y_3$ will be equal to $B_0B_1B_2B_3$, provided the enable line is active (here E=0).
- The output lines are all 0's when E=1.
- Since a multiplexer is in fact a decoder with an internal OR gate, one can use a multiplexer to implement any Boolean function, in a way similar to the method that was used to implement Boolean functions using a decoder and OR gates.
- For this purpose, the first n-1 variables will be used as selection inputs and the remaining variable together with its complement, 1, and 0 will be used to build data inputs.
- Look at each pair of rows (2^{n-1} pair of rows) in the truth table and try to see if the corresponding function values are equal to 0, 1, the last variable in the input columns, or its complement.
- For example, consider the following Boolean function:

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

- This function can be implemented as follows:

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	



- As another example, consider the following Boolean function:

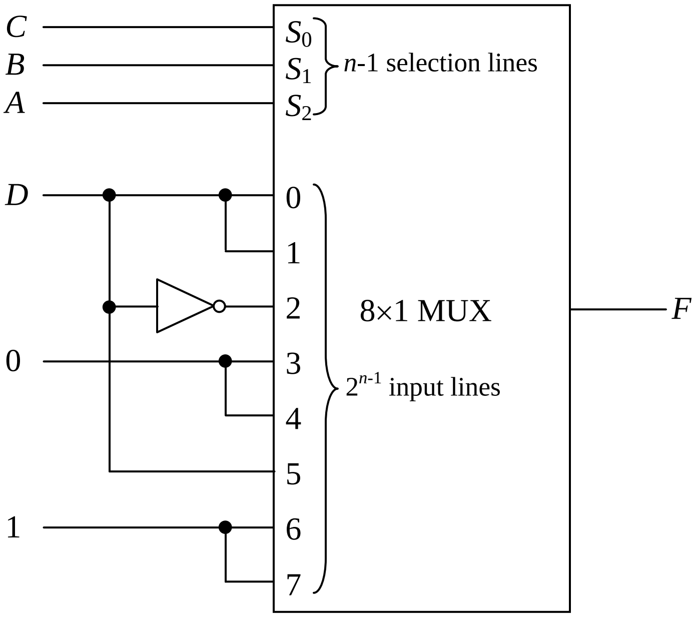
$$F(A, B, C, D) = \sum(1,3,4,11,12,13,14,15)$$

- We will have:

n-1 variables			1 variable		
A	***B***	***C***	***D***	***F***	
0	0	0	0	0	
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	$F = D'$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	1	
1	1	0	0	1	
1	1	0	1	1	
1	1	1	0	1	
1	1	1	1	1	

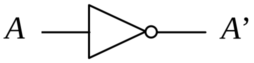
2^{n-1} pair of rows

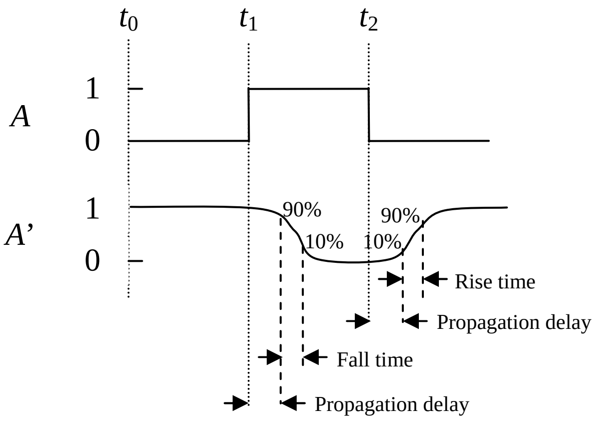
$F = 0$



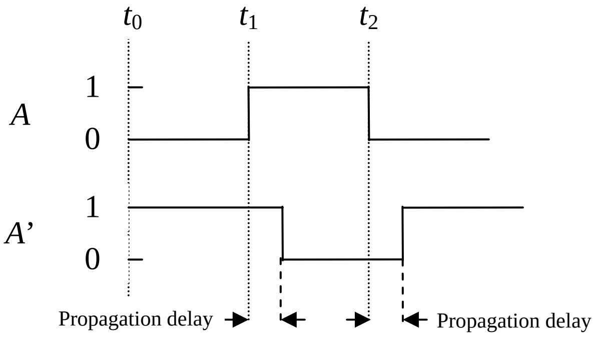
Timing Analysis [2]

- A timing analysis is usually necessary for any complex combinational circuit.
- Consider a NOT gate.



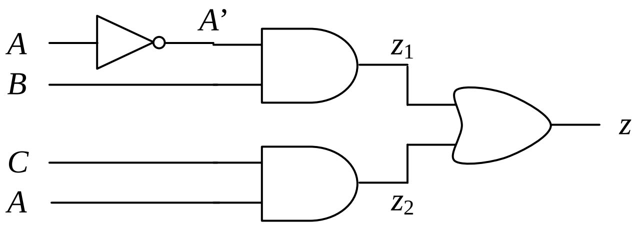


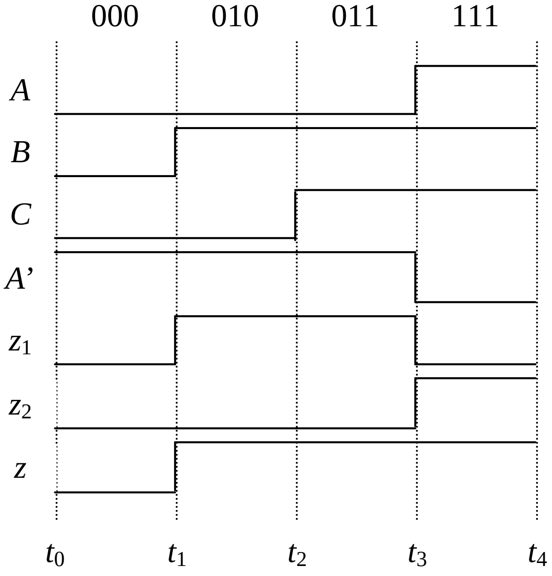
- We will simply consider a gate delay only (also referred to as propagation delay).



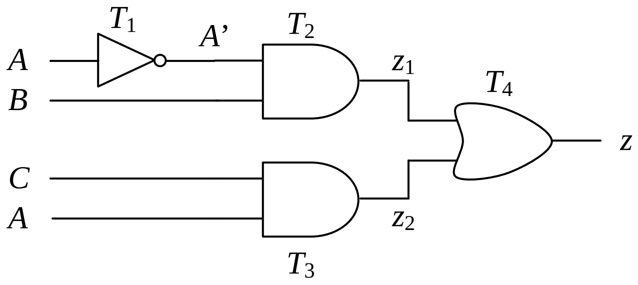
Example [2]

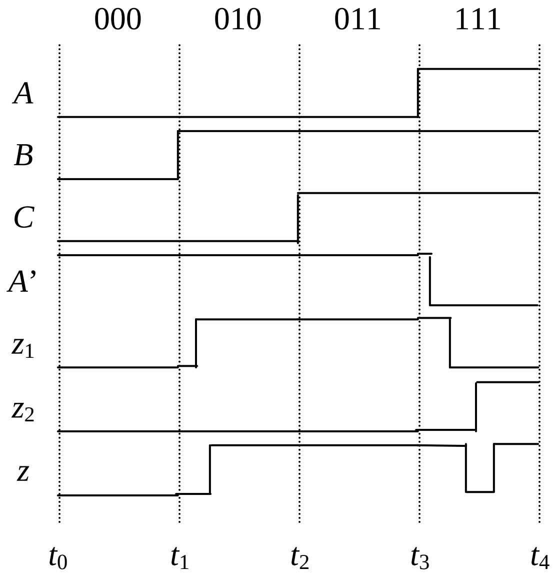
- Consider the following circuit and find the timing diagram for ideal (no delay) gates.



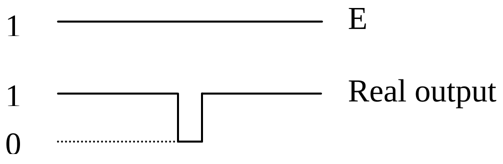


- Assume now that each gate has a delay shown on the following figure, and that $T_3>T_1+T_2$ (when two lines are going through several similar gates, the overall delay on each line can be quite different from the other one as in reality even similar gates have different delays).

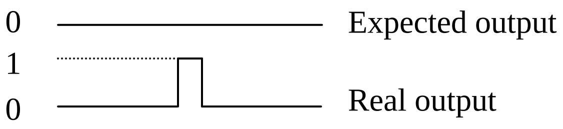




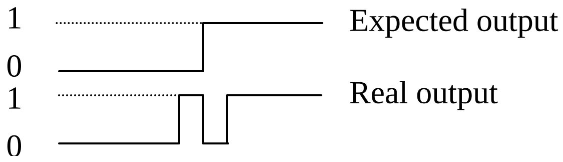
- Hazards are caused by unequal delay times in the signal paths from the input of the circuit to the input of the gate causing the hazard.
- The glitch shown above is called a static hazard.
- A static hazard is defined to be a momentary output change due to a single variable change, when no such output change is normally expected.
- Static-1 hazard:

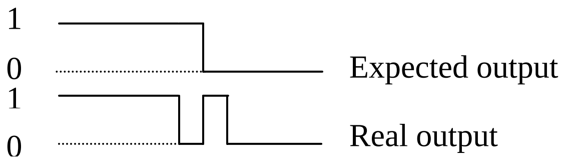


- Static -0 hazard:

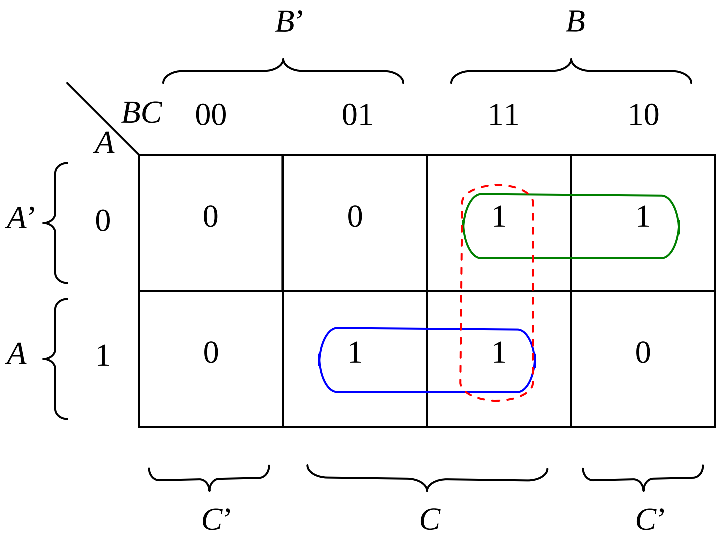


- This would not happen in the above circuit if $T_3<T_1+T_2$.
- Dynamic hazard is a change from 1 to 0 to 1 to 0 or 0 to 1 to 0 to 1 in the output of a gate, when it is expected to change from 1 to 0 or from 0 to 1 due to a change in the input.





- A circuit with no static hazard has no dynamic hazard either.
- Hazards can be prevented by adding additional gates.
- In the above example, the hazard occurs during the transition of *A* from 0 to 1, as shown in the K-map.



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

- This can be prevented by adding the redundant minterm *B.C* to the Boolean expression as shown in the K-map with dashed line, and using a redundant gate to implement that.

References:

[1] Digital Design 3rd Edition, By Morris Mano, Publisher Prentice Hall, 4th Edition.

[2] Introduction to Logic Design, By S. G. Shiva.