Digital Circuit
&
Logic Design
(CSDC-0101)

Combinational Logic

Logic circuits for digital systems may be combinational or sequential.

A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.

A combinational circuit performs an operation that can be specified logically by a set of Boolean functions. In contrast, sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements.

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Combinational Circuits

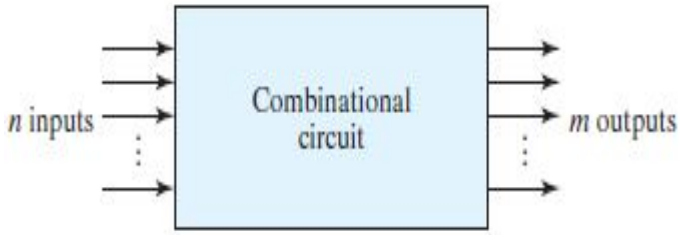
A combinational circuit consists of an interconnection of logic gates.

Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data.

For n input variables, there are 2^n possible combinations of the binary inputs. For each possible input combination, there is one possible value for each output variable.

Thus, a combinational circuit can be specified with a truth table that lists the output values for each combination of input variables.

A combinational circuit also can be described by m Boolean functions, one for each output variable. Each output function is expressed in terms of the n input variables.



There are several combinational circuits that are employed extensively in the design of digital systems.

These circuits are available in integrated circuits and are classified as standard components.

They perform specific digital functions commonly needed in the design of digital systems.

Analysis Procedure

The analysis of a combinational circuit requires that we determine the function that the circuit implements.

This task starts with a given logic diagram and culminates with a set of Boolean functions, a truth table, or, possibly, an explanation of the circuit operation.

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

The first step in the analysis is *to make sure that the given circuit is combinational* and not sequential.

The diagram of a combinational circuit has logic gates with no feedback paths or memory elements.

A feedback path is a connection from the output of one gate to the input of a second gate whose output forms part of the input to the first gate.

Once the logic diagram is verified to be that of a combinational circuit, one can proceed to *obtain the output Boolean functions or the truth table*.

If the function of the circuit is under investigation, then it is necessary to interpret the operation of the circuit from the derived Boolean functions or truth table.

The success of such an investigation is enhanced if one has previous experience and familiarity with a wide variety of digital circuits.

To obtain the output Boolean functions from a logic diagram, we proceed as follows:

1. Label all gate outputs that are a function of input variables with arbitrary symbols— but with meaningful names. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

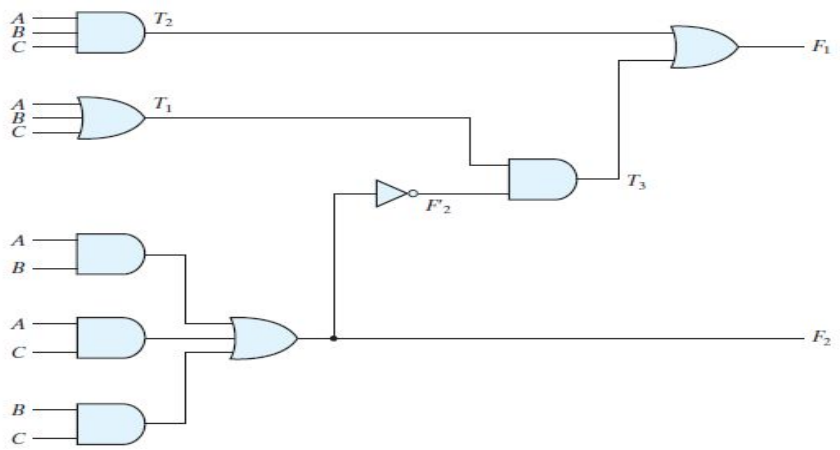
SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

To obtain the truth table directly from the logic diagram without going through the derivations of the Boolean functions, we proceed as follows:

1. Determine the number of input variables in the circuit. For n inputs, form the 2n possible input combinations and list the binary numbers from 0 to (2n - 1) in a table.
2. Label the outputs of selected gates with arbitrary symbols.
3. Obtain the truth table for the outputs of those gates which are a function of the input variables only.
4. Proceed to obtain the truth table for the outputs of those gates which are a function of previously defined values until the columns for all outputs are determined.

Another way of analyzing a combinational circuit is by means of logic simulation. This is not practical, however, because the number of input patterns that might be needed to generate meaningful outputs could be very large. But simulation has a very practical application in verifying that the functionality of a circuit actually matches its specification.

Logic diagram for analysis example



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

The circuit has three binary inputs— A , B , and C —and two binary outputs— F1 and F2. The outputs of various gates are labeled with intermediate symbols.
The outputs of gates that are a function only of input variables are T1 and T2. Output F2 can easily be derived from the input variables. The Boolean functions for these three outputs are:

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

Next, we consider outputs of gates that are a function of already defined symbols:

$$T_3 = F'_2 T_1$$
$$F_1 = T_3 + T_2$$

To obtain F1 as a function of A , B , and C , we form a series of substitutions as follows:

$$\begin{aligned} F_1 = T_3 + T_2 &= F'_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Design Procedure

The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained. The procedure involves the following steps:

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

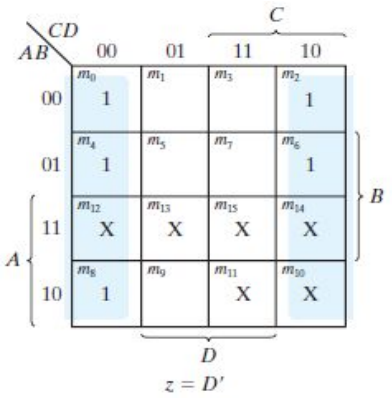
SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Imp.

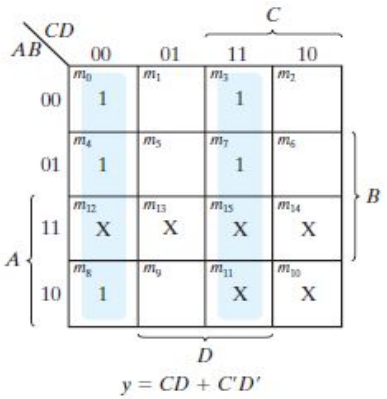
Example, binary coded decimal (BCD) to the excess-3 code for the decimal digits.

Truth Table for Code Conversion Example

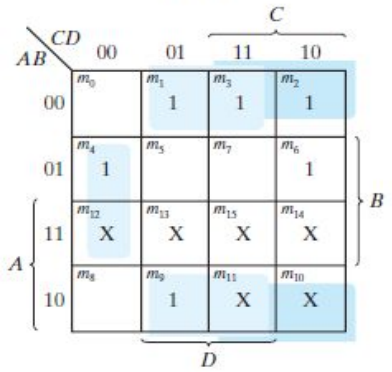
Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



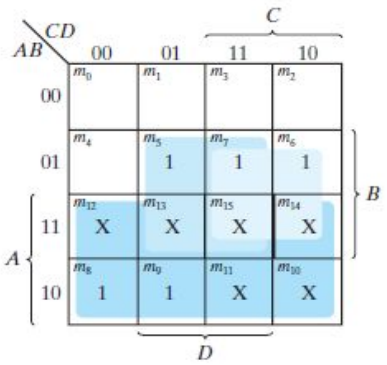
$z = D'$



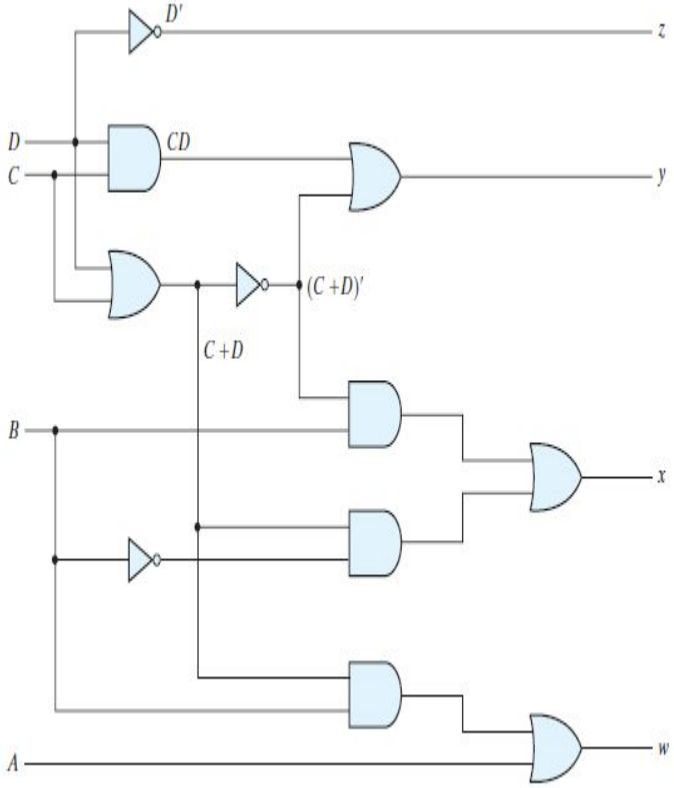
$y = CD + C'D'$



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



$w = A + BC + BD$



Binary Adder–Subtractor

Digital computers perform a variety of information-processing tasks utilizing various arithmetic operations.

The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of four possible elementary operations:

0 + 0 = 0,

0 + 1 = 1,

1 + 0 = 1, and

1 + 1 = 10.

The first three operations produce a sum of one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a carry. When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits.

A combinational circuit that performs the addition of two bits is called a **half adder**. One that performs the addition of three bits (two significant bits and a previous carry) is a **full adder**. The names of the circuits stem from the fact that two half adders can be employed to implement a full adder.

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Binary Adder–Subtractor

A **binary adder–subtractor** is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers.

The half adder design is carried out first, from which we develop the full adder. Connecting n full adders in cascade produces a binary adder for two n-bit numbers. The subtraction circuit is included in a complementing circuit.

Half Adder

This circuit needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits; the output variables produce the sum and carry. We assign symbols x and y to the two inputs and S (for sum) and C (for carry) to the outputs. The C output is 1 only when both inputs are 1. The S output represents the least significant bit of the sum.

The simplified sum-of-products expressions are:

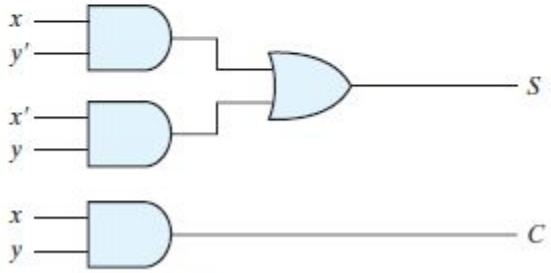
$$S = x'y + xy'$$

$$C = xy$$

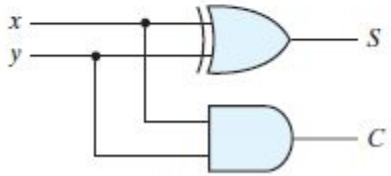
SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Half Adder

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



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

Full Adder

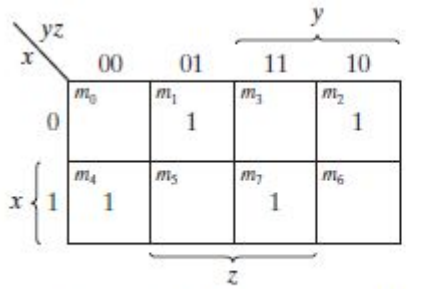
Addition of n-bit binary numbers requires the use of a full adder, and the process of addition proceeds on a bit-by-bit basis, right to left, beginning with the least significant bit. A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs.

The simplified expressions are: $S = x'y'z + x'yz' + xy'z' + xyz$

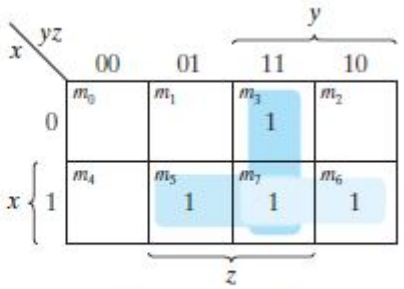
$C = xy + xz + yz$

Full Adder

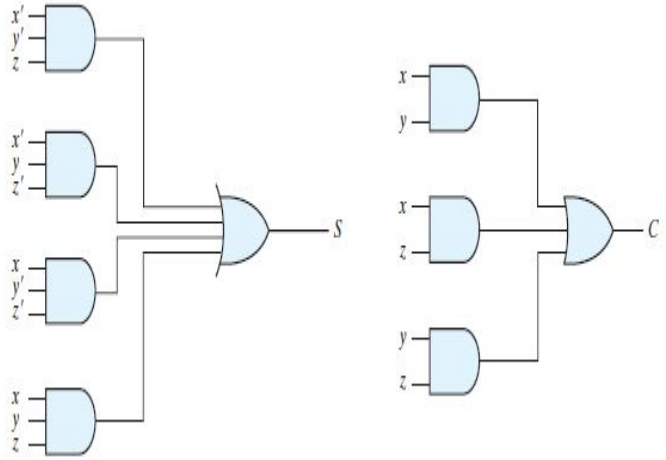
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



(a) $S = x'y'z + x'yz' + xy'z' + xyz$



(b) $C = xy + xz + yz$



Binary Adder

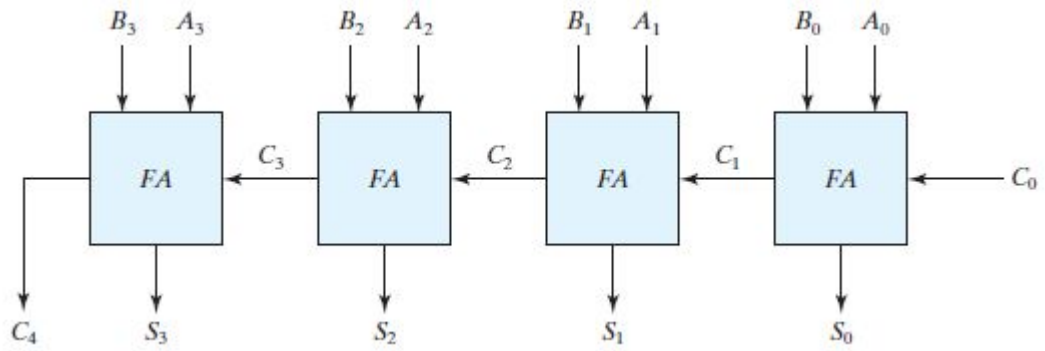
A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.

It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.

Addition of n-bit numbers requires a chain of n full adders or a chain of one-half adder and n-1 full adders.

Example, 4-bit Ripple Carry Adder

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}



SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Carry Propagation

As in any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals. The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit. The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders.

The number of gate levels for the carry propagation can be found from the circuit of the full adder.

The carry propagation time is an important attribute of the adder because it limits the speed with which two numbers are added.

Since all other arithmetic operations are implemented by successive additions, the time consumed during the addition process is critical. An obvious solution for reducing the carry propagation delay time is to employ faster gates with reduced delays. However, physical circuits have a limit to their capability.

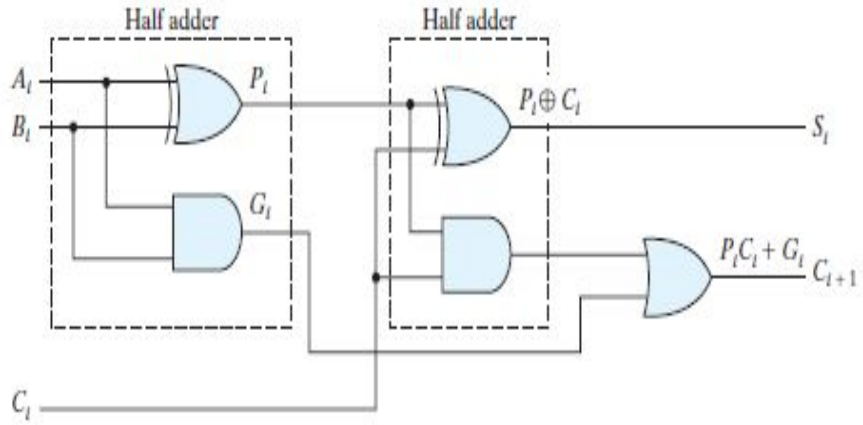
Another solution is to increase the complexity of the equipment in such a way that the carry delay time is reduced. There are several techniques for reducing the carry propagation time in a parallel adder. The most widely used technique employs the principle of ***carry lookahead logic*** .

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Consider the circuit of the full adder with new binary variables:

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$



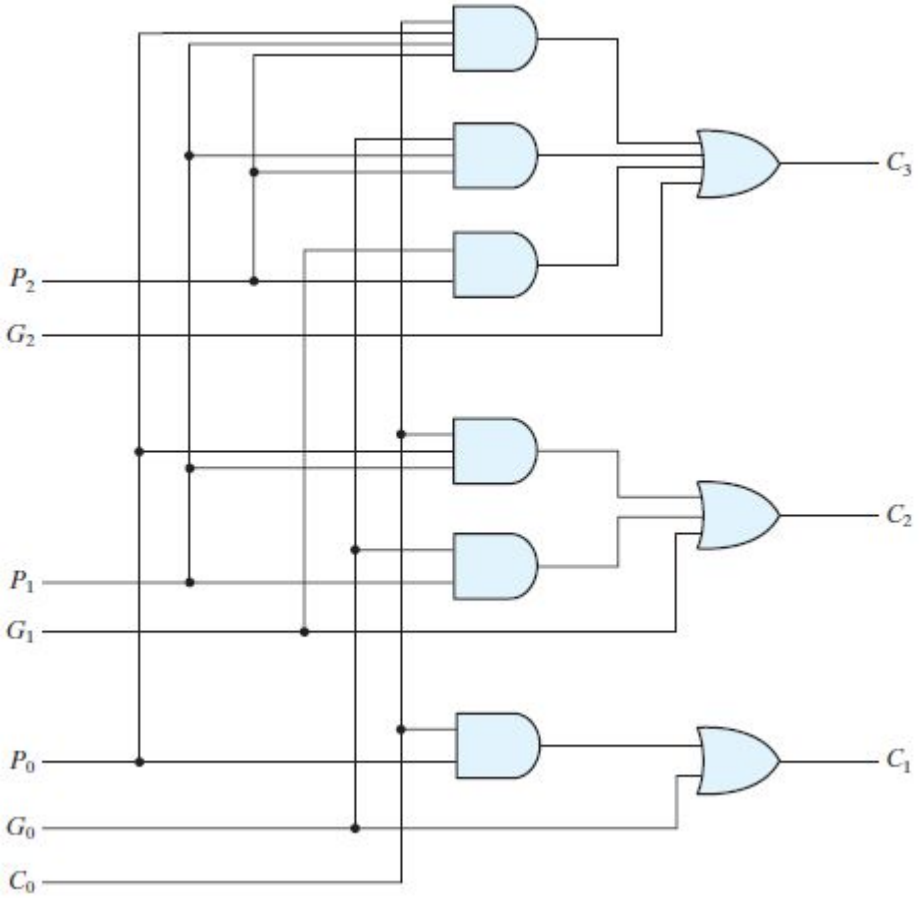
the output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$

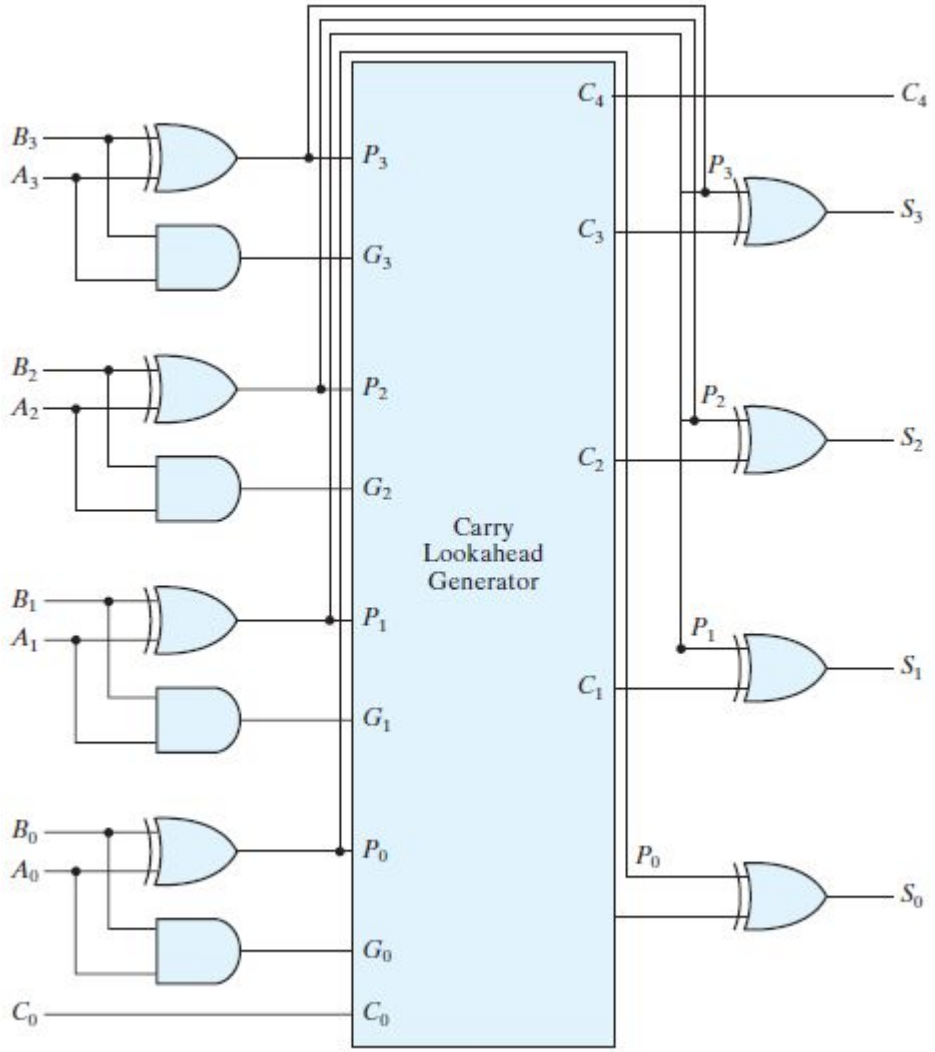
$$C_{i+1} = G_i + P_i C_i$$

G_i is called a *carry generate*, and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i. P_i is called a *carry propagate*, because it determines whether a carry into stage i will propagate into stage i + 1 (i.e., whether an assertion of C_i will propagate to an assertion of C_{i+1}).

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1



Logic diagram of carry lookahead generator



Four-bit adder with carry lookahead

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Binary Subtractor

The subtraction of unsigned binary numbers can be done most conveniently by means of complements.

Remember that the subtraction A - B can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.

The circuit for subtracting A - B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C_0 must be equal to 1 when subtraction is performed. The operation thus performed becomes A, plus the 1's complement of B, plus 1. This is equal to A plus the 2's complement of B.

For unsigned numbers, that gives A - B if A ≥ B or the 2's complement of (B – A) if A < B. For signed numbers, the result is A - B, provided that there is no overflow. (*Overflow is a problem in computers where the number of bits that hold a number is finite, and a result that exceeds the finite value by 1 which cannot be accommodated*)

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder.

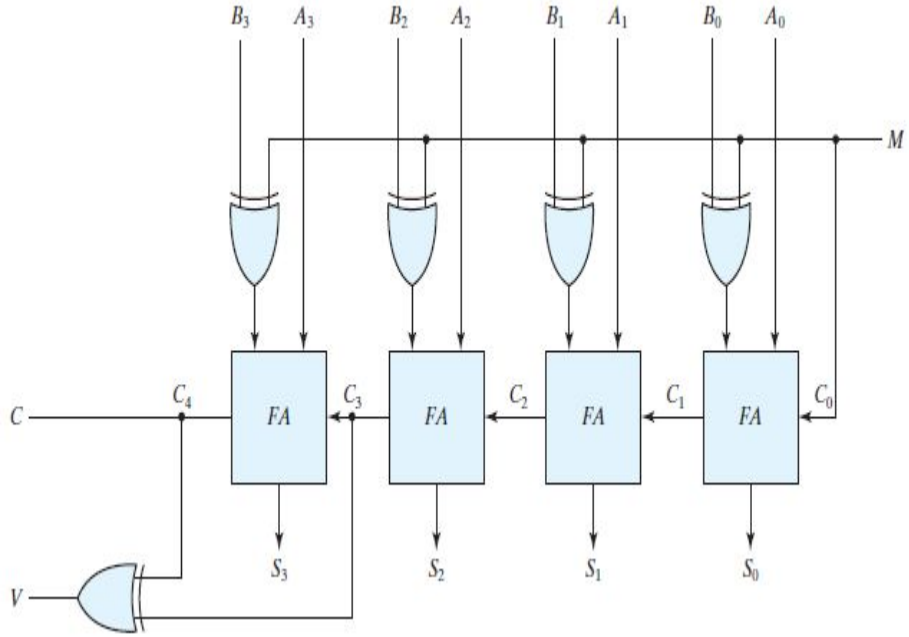
It is worth noting that binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as are unsigned numbers.

Therefore, computers need only one common hardware circuit to handle both types of arithmetic.

For the given circuit,

- when M = 0, the circuit is an adder with B ⊕ 0 = B. The full adders receive the value of B , the input carry is 0, and the circuit performs A plus B .
- when M = 1, the circuit becomes a subtractor with B ⊕ 1 = B' and C_0 = 1. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B .

The exclusive-OR with output V is for detecting an overflow.



Four-bit adder–subtractor (with overflow detection)

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Overflow:

When two numbers with n digits each are added and the sum is a number occupying n+1 digits, we say that an overflow occurred.

Overflow is a problem in digital computers because the number of bits that hold the number is finite and a result that contains n+1 bits cannot be accommodated by an n-bit word. For this reason, many computers detect the occurrence of an overflow, and when it occurs, a corresponding flip-flop is set that can then be checked by the user.

The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned. When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position. In the case of signed numbers, two details are important: the leftmost bit always represents the sign, and negative numbers are in 2's-complement form. When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.

An overflow cannot occur after an addition if one number is positive and the other is negative, since adding a positive number to a negative number produces a result whose magnitude is smaller than the larger of the two original numbers. An overflow may occur if the two numbers added are both positive or both negative.

An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. If these two carries are not equal, an overflow has occurred.

Example: Two signed binary numbers, +70 and +80, are stored in two eight-bit registers. The range of numbers that each register can accommodate is from binary +127 to binary -128. Since the sum of the two numbers is +150, it exceeds the capacity of an eight-bit register.

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Decimal Adder

Computers or calculators that perform arithmetic operations directly in the decimal number system represent decimal numbers in binary coded form. An adder for such a computer must employ arithmetic circuits that accept coded decimal numbers and present results in the same code.

A decimal adder requires a minimum of nine inputs and five outputs, since four bits are required to code each decimal digit and the circuit must have an input and output carry.

A decimal parallel adder that adds n decimal digits needs n BCD adder stages. The output carry from one stage must be connected to the input carry of the next higher order stage.

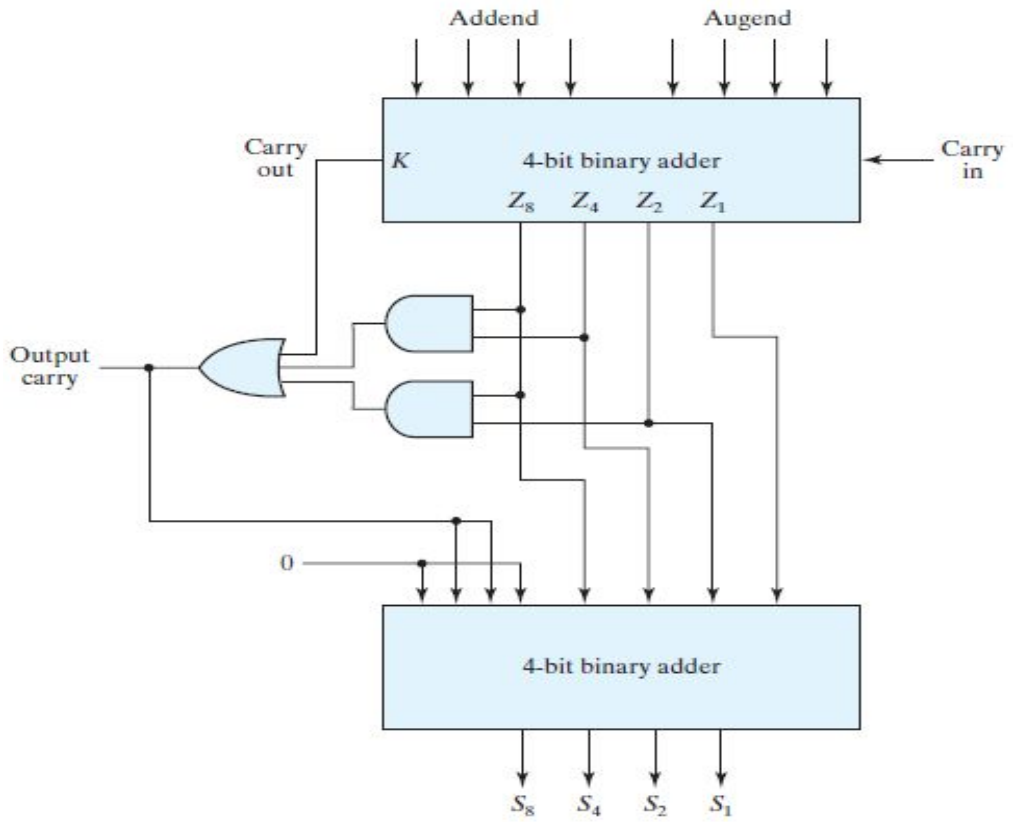
BCD Adder

Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 9 + 9 + 1 = 19, the 1 in the sum being an input carry. Suppose, we apply two BCD digits to a four-bit binary adder. The adder will form the sum in binary and produce a result that ranges from 0 through 19.

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Derivation of BCD Adder

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



K is the carry, and the subscripts under the letter Z represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code.

The condition for a correction and an output carry can be expressed by the Boolean function:

$$C = K + Z_8Z_4 + Z_8Z_2$$

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Binary Multiplier

Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers.

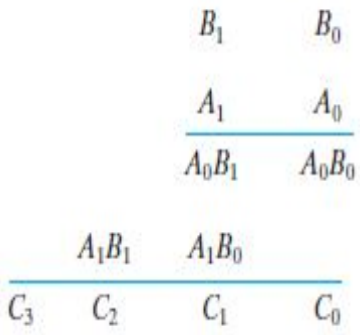
The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit. Each such multiplication forms a partial product.

Successive partial products are shifted one position to the left. The final product is obtained from the sum of the partial products.

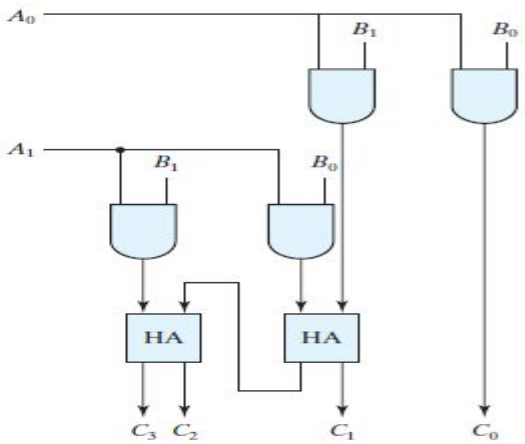
For instance, the multiplicand bits are B_1 and B_0, the multiplier bits are A_1 and A_0, and the product is $C_3C_2C_1C_0$. The first partial product is formed by multiplying B_1B_0 by A_0.

The multiplication of two bits such as A_0 and B_0 produces a 1 if both bits are 1; otherwise, it produces a 0. This is identical to an AND operation.

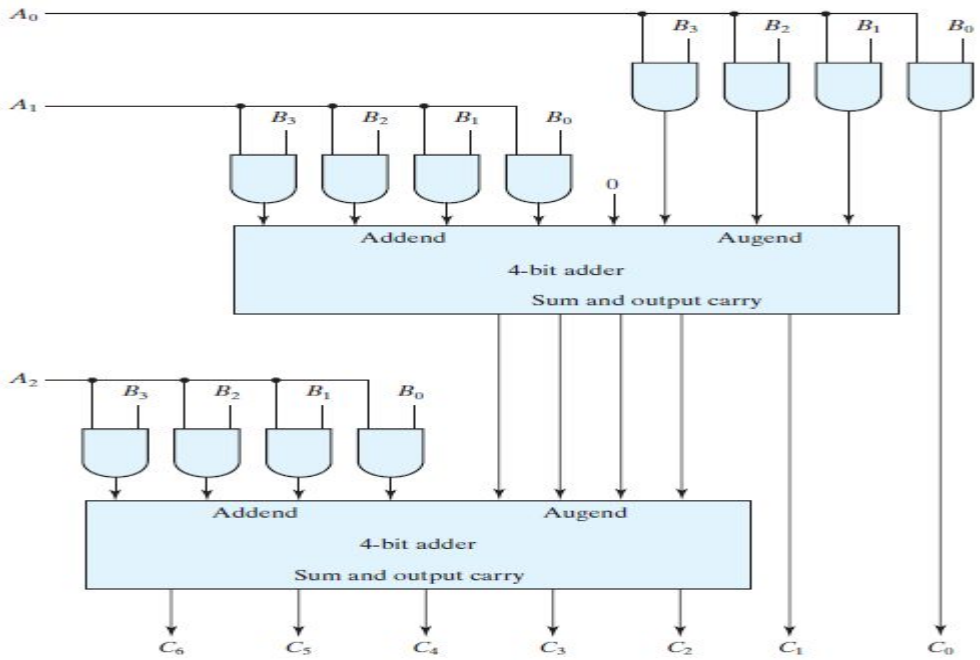
Therefore, the partial product can be implemented with AND gates. The second partial product is formed by multiplying B_1B_0 by A_1 and shifting one position to the left.



SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1



Two-bit by two-bit binary multiplier



Four-bit by three-bit binary multiplier

Since K = 4 and J = 3, we need 12 AND gates and two 4-bit adders to produce a product of seven bits.

The two partial products are added with two half-adder (HA) circuits.

Note that the least significant bit of the product does not have to go through an adder, since it is formed by the output of the first AND gate.

For J multiplier bits and K multiplicand bits, we need (J * K) AND gates and (J – 1) K -bit adders to produce a product of (J + K) bits.

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

Magnitude Comparator

A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicate whether A > B, A = B, or A < B.

On the one hand, the circuit for comparing two n-bit numbers has 2^{2n} entries in the truth table and becomes too cumbersome, even with n = 3. On the other hand, as one may suspect, a comparator circuit possesses a certain amount of regularity.

An algorithm is defined for the design of a four-bit magnitude comparator which is a direct application of the procedure a person uses to compare the relative magnitudes of two numbers. Consider two numbers, A and B, with four digits each. Write the coefficients of the numbers in descending order of significance:

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

The two numbers are equal if all pairs of significant digits are equal.

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1

When the numbers are binary, the digits are either 1 or 0, and the equality of each pair of bits can be expressed logically with an exclusive-NOR function as:

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

where $x_i = 1$ only if the pair of bits in position i are equal (i.e., if both are 1 or both are 0).

Note that, for equality to exist, all x_i variables must be equal to 1, a condition that dictates an AND operation of all variables.

To determine whether A is greater or less than B, we inspect the relative magnitudes of pairs of significant digits, starting from the most significant position.

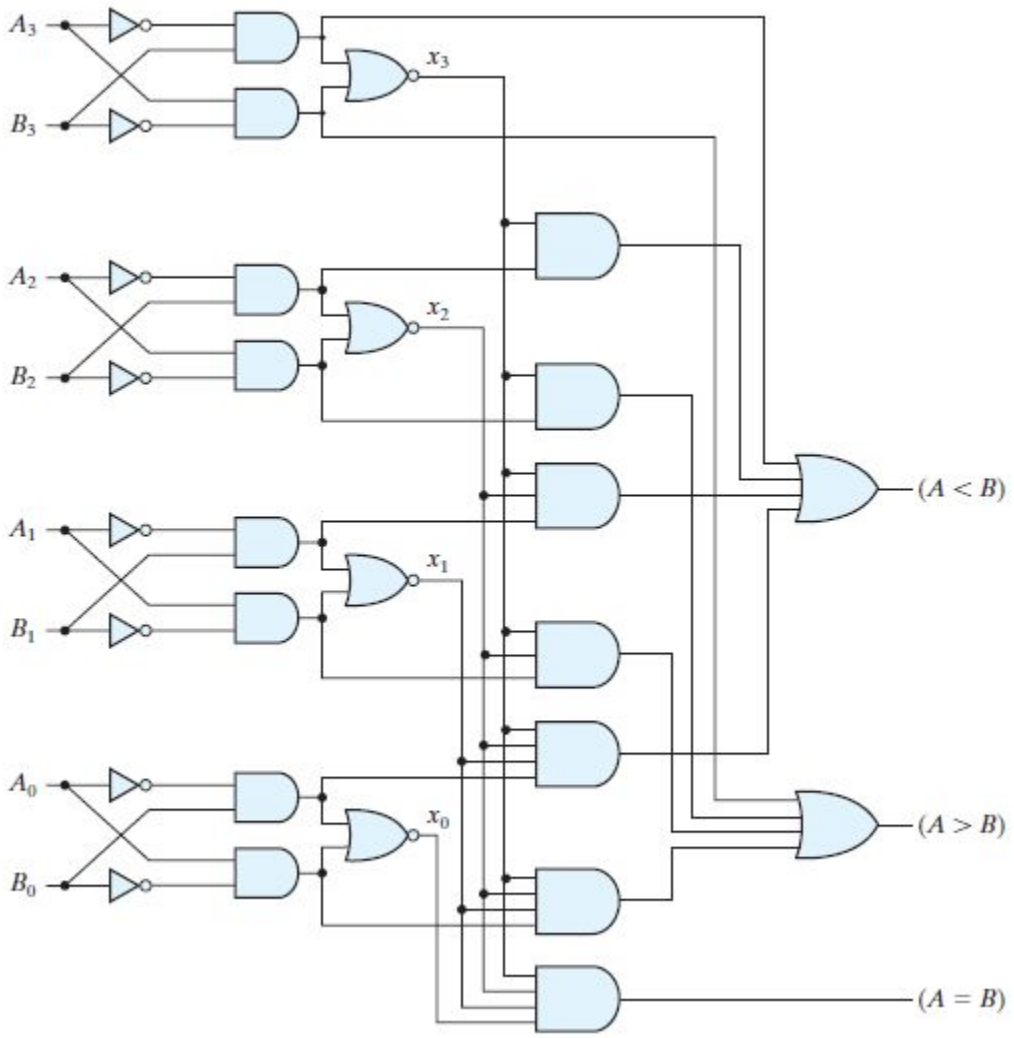
If the two digits of a pair are equal, we compare the next lower significant pair of digits. The comparison continues until a pair of unequal digits is reached.

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. The sequential comparison can be expressed logically by the two Boolean functions:

$$(A > B) = A_3 B'_3 + x_3 A_2 B'_2 + x_3 x_2 A_1 B'_1 + x_3 x_2 x_1 A_0 B'_0$$

$$(A < B) = A'_3 B_3 + x_3 A'_2 B_2 + x_3 x_2 A'_1 B'_1 + x_3 x_2 x_1 A' n_0 B'_0$$

SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1



SOURCE: HTTPS://DOI.ORG/10.1007/978-3-319-66775-1_1