MSI ADDERS and SUBTRACTORS

MSI ADDERS

There are commercially available 1-bit, 2-bit, and 4-bit full adders, each in one package. In the figure 1, it is indicated the logic topology for 2-bit addition.

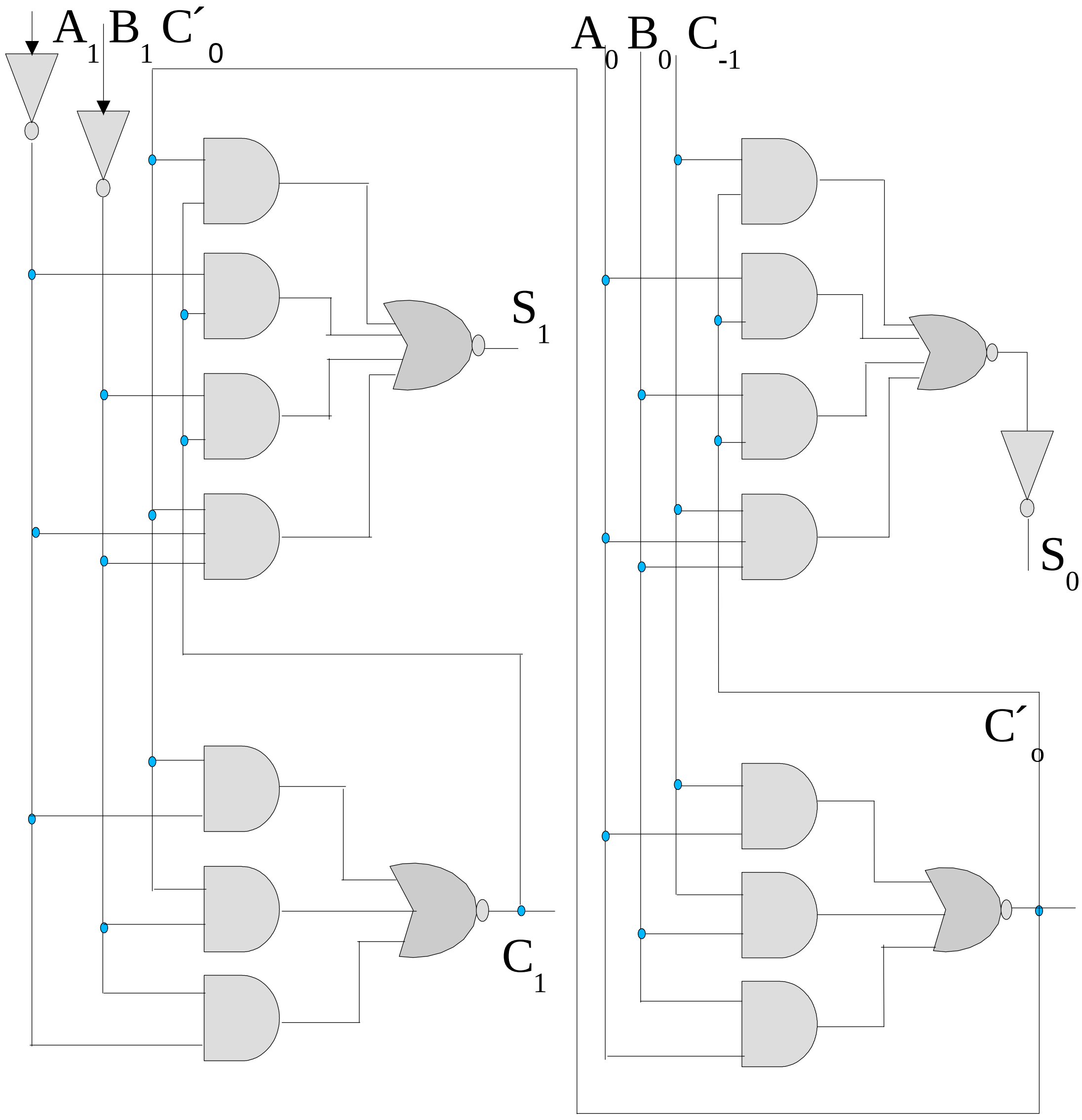


Fig. 1: Logic diagram of an integrated 2-bit full adder

The inputs to the first stage are A_0 and B_0; the input marked C_{-1} is grounded. The output is the sum S_0. The carry C_0 is connected internally and is not brought to an output pin. This 2^0 stage (LSB) is identical with that in the figure 2, with *n* = 0. The abbreviation LSB means *least significant bit.*

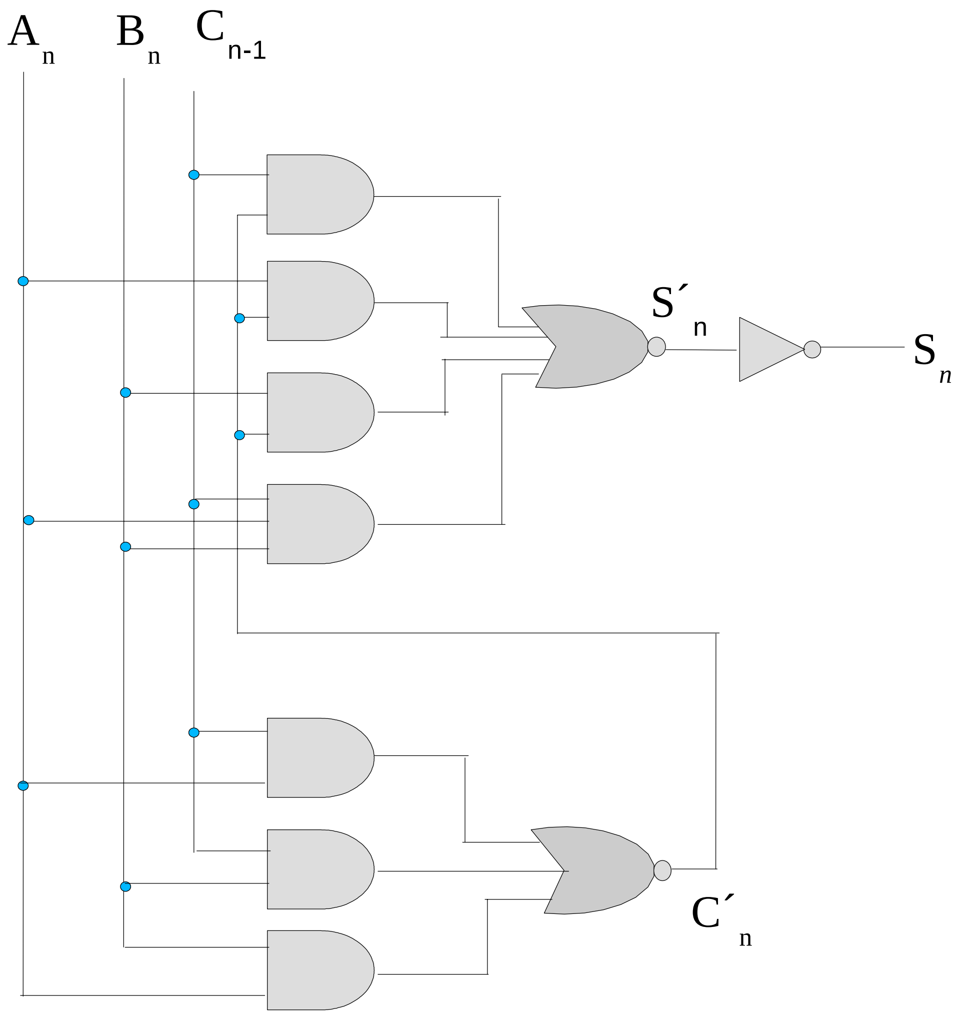


Fig. 2: Block diagram implementation of the *n*-th stage of a full adder

Since the carry from the first stage is C'_0, it should be negated before it is fed to the 2^1 stage. However, the delay introduced by this inversion is undesirable, because the limitation upon the maximum speed of operation is the propagation delay of the carry through all the bits in the adder.

The NOT-gate delay is eliminated completely in the carry by connecting C'_0 directly to the following stage and by complementing the inputs A_1 and B_1 before feeding these to this stage. This latter method is used in the figure 1.

Note that now the outputs S_1 and C_1 are obtained directly without requiring inverters. The logic followed by this second stage for the carry is given by an equation from the full adder

$$C_n = B'_n C'_{n-1} + A'_n C'_{n-1} + A'_n B'_n ,$$

and for the sum by the modified form of the equation

$$S_n = A_n C'_n + B_n C'_n + C_{n-1} C'_n + A_n B_n C_{n-1},$$

where each symbol is replaced by its complement.

In a 4-bit adder C_1 is not brought out but is internally connected to the third stage, which is identical with the first stage. Similarly, the fourth and second stages have identical logic topologies.

A 4-bit adder requires a 16-pin package: 8 inputs, 4 sum outputs, a carry output, a carry input, the power-supply input, and ground. The carry input is needed only if two arithmetic units are cascaded; for example, cascading a 2-bit with a 4-bit adder gives the sum of two 6-bit numbers.

If the 2-bit unit is used for the 2^4 and 2^5 digits, then 4 must be added to all the subscripts in the figure 1. For example, C_{-1} is now called C_3 and is obtained from the output carry of the 4-bit adder.

The MSI chip (TI 74LS83, a specific IC from Texas Instruments) for a 4-bit binary full adder contains somewhat over 200 components (resistors, diodes, or transistors). For high-speed, low-power operation, Schottley transistors and diodes are used in each AOI block, and each gate output contains a Darlington pair.

The propagation delay time of the carry is typically 50 *ns*, and the power dissipation is 75 mW.

SERIAL OPERATION

In a serial adder the inputs A and B are synchronous pulse trains on two lines in the computer. We have four different figures in the following to show these operations. If you minutely look at those figures, you can find some similarities between the binary operations and the binary bit strings.

Figure 3(*a*) and 3(*b*) show typical pulse trains representing, respectively, the decimal numbers 13 and 11. Pulse trains representing the sum (24) and difference (2) are shown in the figure 3(*c*) and 3(*d*), respectively.

Let us take two arbitrary numbers: 13 and 11 as the inputs B and A, respectively. The binary form of these are 01101_2 and 01011_2. If we add these two binary numbers, we shall find the following result:

```
0 1 1 0 1     ⇒13
0 1 0 1 1     ⇒11
-----------
1 1 0 0 0     ⇒24
```

The following figure shows the binary number representation for 13 in the figure 3(*a*) and then 11 in the figure 3(*b*). The figure 3(*c*) shows the sum of these two numbers.

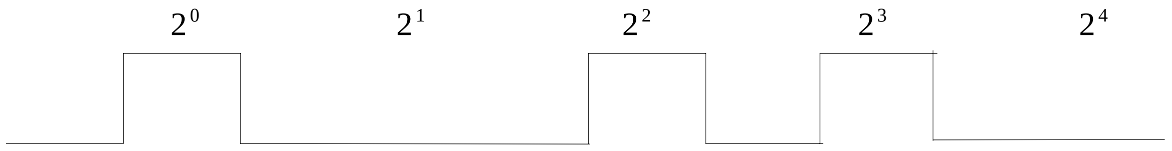


Fig. 3(*a*): Pulse waveforms representing number 13 ⇒ 01101, with 2^0 as LSB

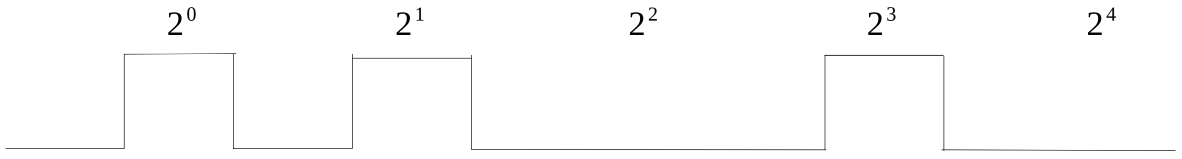


Fig. 3(*b*): Pulse waveforms representing number 11 ⇒ 01011, with 2^0 as LSB

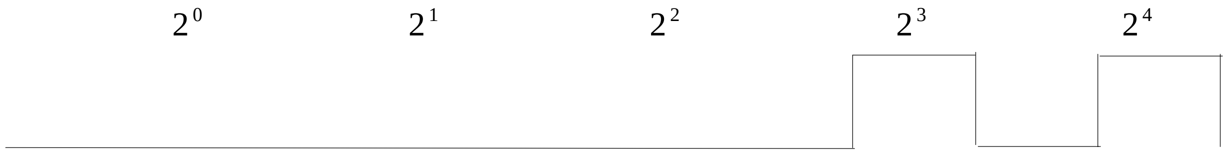


Fig. 3(*c*): Pulse waveforms representing sum of 13 and 11; 24 ⇒ 11000

Now if we take difference of these two numbers, we may have:

```
0 1 1 0 1      ⇒13
0 1 0 1 1      ⇒11
-----------
0 0 0 1 0      ⇒2
```

The following figure represents the difference or the subtraction result as derived above.

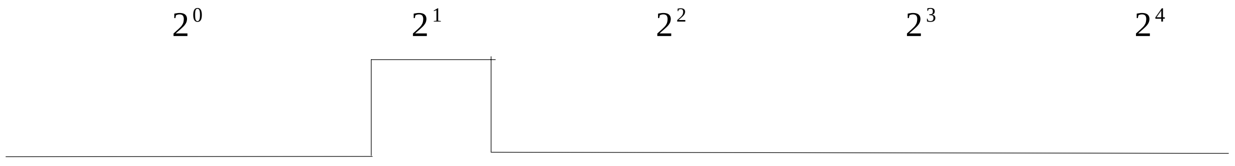


Fig. 3(*d*): Pulse waveforms representing subtraction of 13 and 11; 2 ⇒ 00010

A serial *adder* is a device which will take as inputs the two wave forms of the figure 3(*a*) and 3(*b*); and deliver the output wave form in the figure 3(*c*). Similarly, a subtraction will yield the output shown in the figure 3(*d*).

We have already emphasised that the sum of two multidigit numbers may be formed by adding to the sum of the digits of like significance the carry (if any) which may have resulted from the next lower place. With respect to the pulse trains of the figure 3, the above statement is equivalent to saying that at any instant of time, we must add (in binary form) to the pulses A and B the carry pulse (if any) which comes from the resultant formed one period *T* earlier.

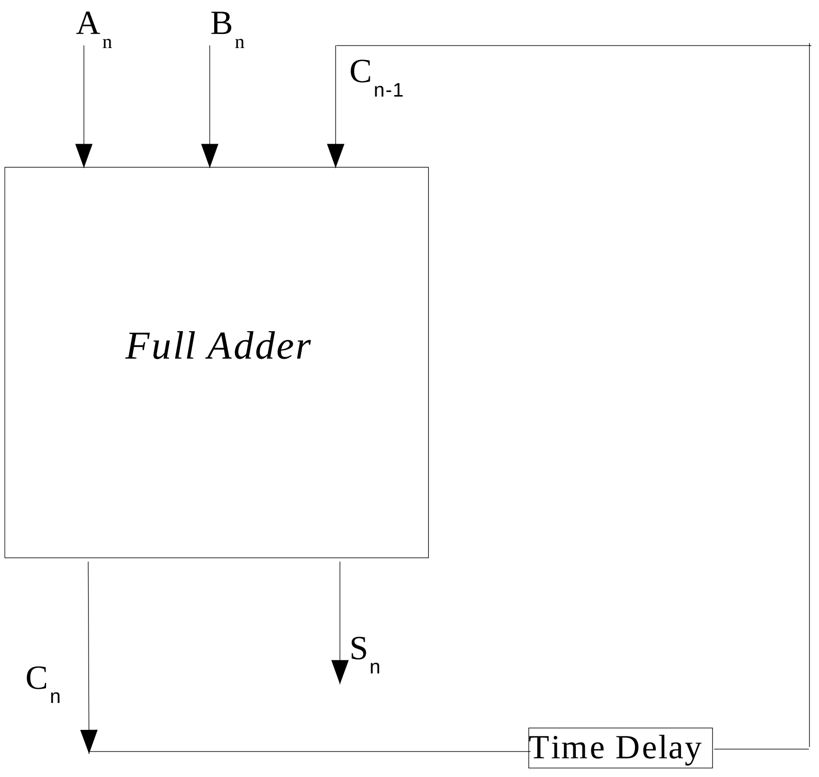


Fig. 4: A serial binary full adder

The logic outlined above is performed, by the full-adder circuit of the figure 4. This circuit differs from the configuration in the parallel adder as shown in the following figure, [that we had seen in previous chapter] by the inclusion of a time delay TD which is equal to the time T between pulses. Hence the carry pulse is delayed a time T and added to the digit pulses in A and B, exactly as it should be.

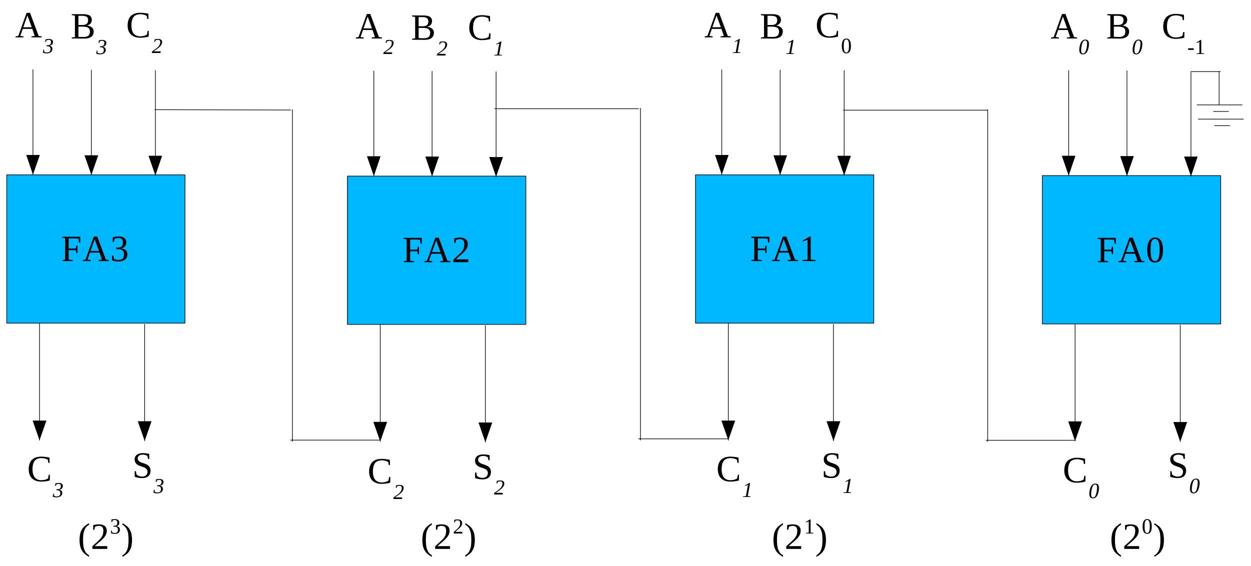


Fig. 5: A 4-bit parallel binary adder constructed from cascaded full adder

A comparison of the figures 5 and 4 indicates that parallel addition is faster than serial because all digits are added simultaneously in the former, but in sequence in the latter. However, whereas only one full adder is needed for serial arithmetic, we must use a full adder for each bit in parallel addition. Hence parallel addition is much more expensive than serial operation.

The time delay unit *time delay* [TD] is a type D FLIP-FLOP, and the serial numbers A_n, B_n, and S_n are stored in *shift registers*.

SUBTRACTORS

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this method, the subtraction operation becomes an addition operation requiring full adders for its machine implementation.

It is possible to implement subtraction with logic circuits in a direct manner, as done with paper and pencil. By this method, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position.

The fact that a 1 has been borrowed must be conveyed to the next higher pair of bits by means of a binary signal coming out (*output*) of a given stage and going into (*input*) the next higher stage. Just as there are half- and full-adders, there are half- and full-subtractors.

HALF SUBTRACTOR

A half-subtractor is a combinational circuit that subtracts two bits and produces their difference. It also has an output to specify if a 1 has been borrowed. Designate the minuend bit by x and the subtrahend bit by y.

To perform $x - y$, we have to check the relative magnitudes of x and y. If $x > y$, we have three possibilities: $0 - 0 = 0$, $1 - 0 = 1$, and $1 - 1 = 0$. The result is called the *difference bit*.

If $x < y$, we have $0 - 1$, and it is necessary to borrow a 1 from the next higher stage. The 1 borrowed from the next higher stage adds 2 to the minuend bit, just as in the decimal system a borrow adds 10 to a minuend digit. With the minuend equal to 2, the difference becomes $2 - 1 = 1$.

The half-subtractor needs two outputs. One output generates the difference and will be designated by the symbol **D**. The second output, designated **B** for borrow, generates the binary signal that informs the next stage that a 1 has been borrowed. The truth table for the input-output relationship of a half-subtractor can now be derived as follows:

x	***y***	***B***	***D***
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

The output borrow **B** is a 0 as long as $x > y$. It is a 1 for $x = 0$ and $y = 1$. The **D** output is the result of the arithmetic operation 2**B** + $x - y$.

The Boolean functions for the two outputs of the half-subtractor are derived directly from the truth table:

$$D = x'y + xy'$$

$$B = x'y$$

It is interesting to note that the logic for **D** is exactly the same as the logic for output **S** in the half-adder.

FULL SUBTRACTOR

A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account that a 1 may have been borrowed by a lower significant stage. This circuit has three inputs and two outputs. The three inputs x, y, and z, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, **D** and **B**, represent the difference and output borrow, respectively. The truth table for the circuit is as follows:

x	***y***	***z***	***B***	***D***
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

The eight rows under the input variables designate all possible combinations of 1's and 0's that the binary variables may take. The 1's and 0's for the output variables to determined from the subtraction of $x - y - z$.

The combinations having input borrow z = 0 reduce to the same four conditions of the half-adder. For $x = 0$, borrow a 1 from the next stage, which makes **B** = 1 and adds 2 to x. Since 2 – 0 – 1 = 1, **D** = 1.

For $x = 0$ and $yz = 11$, we need to borrow again, making **B** = 1 and $x = 2$. Since 2 – 1 – 1 = 0, **D** = 0. For $x = 1$ and $yz = 01$, we have $x - y - z = 0$, which makes **B** = 0 and **D** = 0. Finally, for $x = 1$, $y = 1$, $z = 1$, we have to borrow 1, making **B** = 1 and $x = 3$, and 3 – 1 – 1 = 1, making **D** = 1.

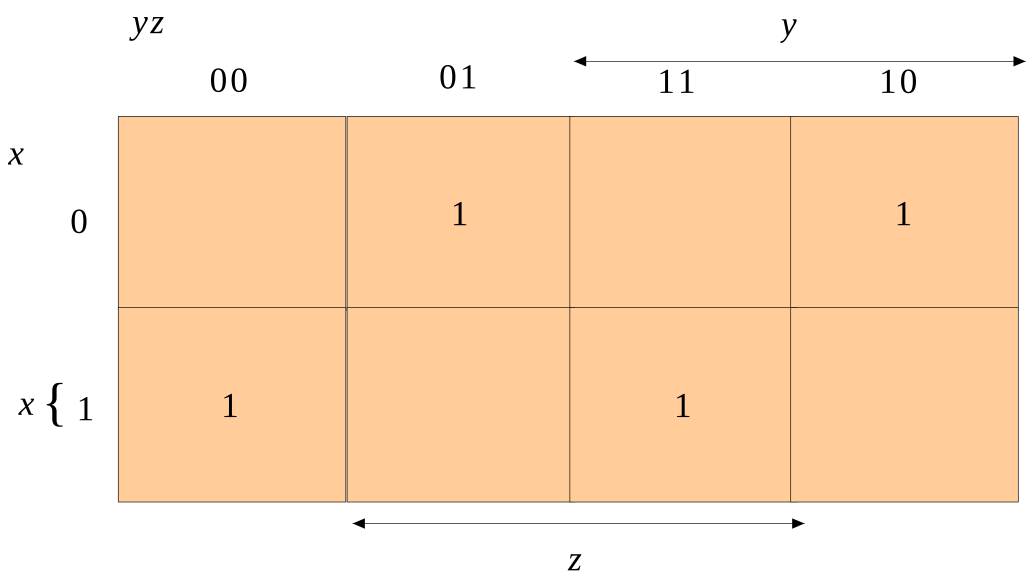
The simplified Boolean functions for the two outputs of the full-subtractor derived in the maps of the figure 6. The simplified sum of products output

$$D = x'y'z + xyz' + xy'z' + xyz$$

$$B = x'y + x'z + yz$$

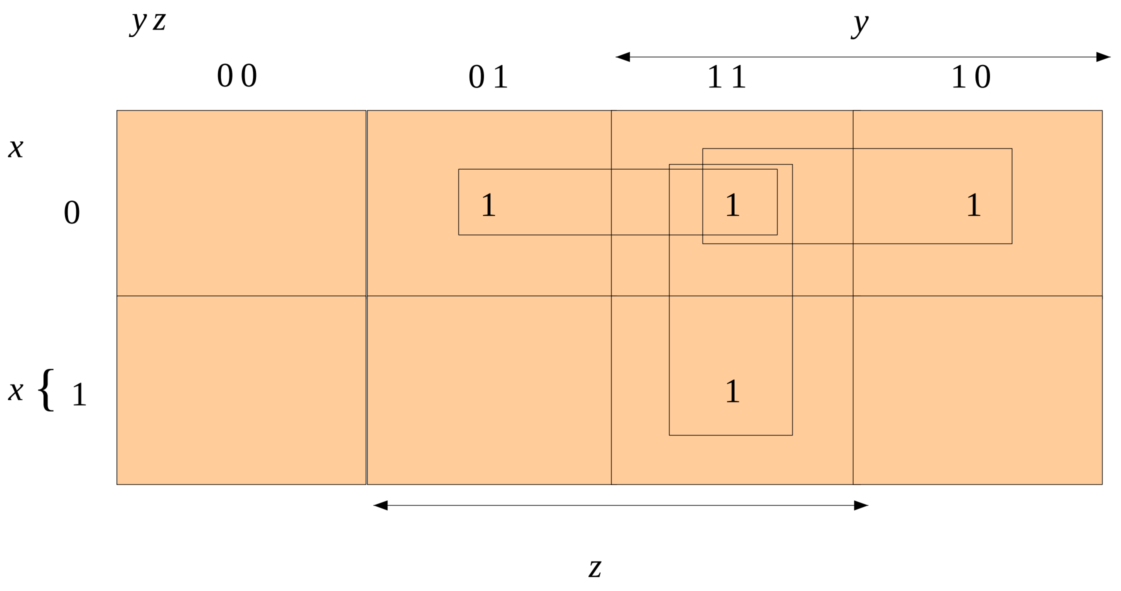
Again we note that the logic function for output **D** in the full-subtractor is exactly same as output **S** in the full-adder. Moreover, the output **B** resembles the on for **C** in the full-adder, except that the input variable x is complemented.

Because of these similarities, it is possible to convert a full-adder into a full-subtractor merely complementing input x prior to its application to the gates that form the carry output.



D* = *x'y'z* + *xyz'* + *xy'z'* + *xyz

Fig. 6(*a*): Karnaugh Map for full-subtractor



$$B = x'y + x'z + yz$$

Fig. 6(*b*): Karnaugh Map for full-subtractor