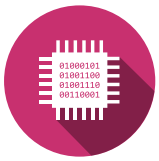



# Wired Multiplier

Labor Digital Design

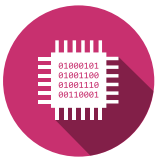
# Contents



# 1 Goal

In this lab, you will deal with the implementation of multipliers for natural and arithmetic numbers. You will learn how multiplication algorithms are realized in hardware and what special features must be considered when processing binary numbers.

By gradually developing a multiplier for natural numbers and extending it to arithmetic numbers in two's complement, you will gain a deeper understanding of digital circuits. You will use different logic blocks and analyze their behavior, taking into account calculation delays.



# 2 Multiplier for natural numbers

In this section, you will develop a multiplier for natural numbers, i.e., for non-negative integers stored in binary form.

## 2.1 Algorithm

Figure 1 shows the algorithm for multiplying two binary numbers. It is based on the calculation of partial products that are then summed. This method corresponds to multiplication written in the decimal system, except that binary digits are used here. Each partial product is obtained by multiplying a number by a single binary digit of the other number. The resulting partial products are then shifted one bit to the right and added to obtain the final result.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | $a_3$ | $a_2$ | $a_1$ | $a_0$ |
| | | | | $\times\ b_3$ | $b_2$ | $b_1$ | $b_0$ |
| | | | | $b_0{*}a_3$ | $b_0{*}a_2$ | $b_0{*}a_1$ | $b_0{*}a_0$ |
| | | | $b_1{*}a_3$ | $b_1{*}a_2$ | $b_1{*}a_1$ | $b_1{*}a_0$ | |
| | | $b_2{*}a_3$ | $b_2{*}a_2$ | $b_2{*}a_1$ | $b_2{*}a_0$ | | |
| | $b_3{*}a_3$ | $b_3{*}a_2$ | $b_3{*}a_1$ | $b_3{*}a_0$ | | | |
| $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |

Figure 1 - Multiplication algorithm

## 2.2 Analysis

First, we will analyze the multiplication of 2 natural numbers (unsigned) coded on 4 bits. The multiplication of 2 numbers with 4 bits results in a product with 8 bits. As shown in Figure 1.

Determine the largest and smallest number that can be reached with this multiplier.

Also, determine the number of bits required for the product of two natural binary numbers coded respectively on $n_1$ and $n_2$ bits.

## 2.3 Circuit

Figure 2 shows the circuit of a multiplier that works according to the algorithm given above.

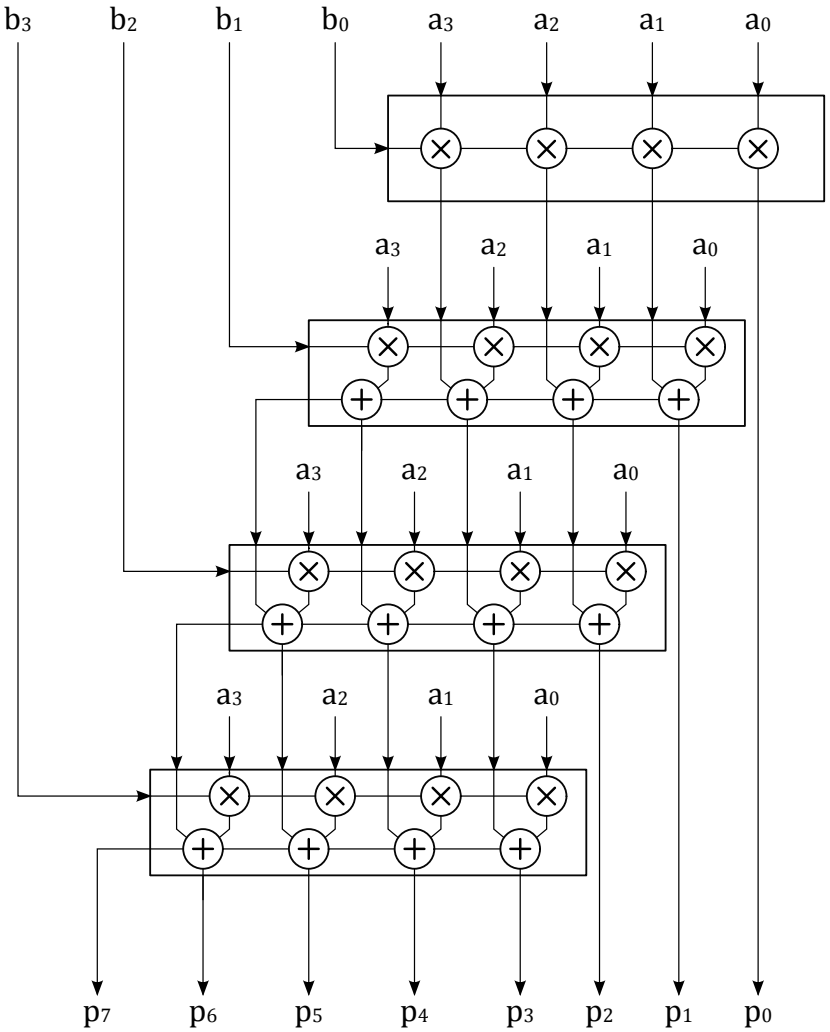
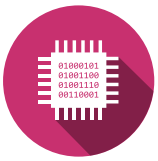



Figure 2 - Architecture of the multiplier

## 2.4 Implementation

Implement the circuit of the multiplier of Figure 2. Some circuit elements are already given in the block **`MUL/mul4Unsigned`**. Complete the missing elements to ensure the functionality of the multiplier.

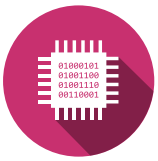
Using INV, AND, OR and XOR gates, complete the hierarchical diagram of the multiplier of Figure 2.




## 2.5 Simulation

Each implemented circuit must be correctly tested. The testbench **`MUL_test/mul4Unsigned_tb`** must check the **complete** operation of the circuit. How is this realized in the case of the tester **`MUL_test/mul4Unsigned_tester`**?

Simulate the testbench **`MUL_test/mul4Unsigned_tb`** with the simulation file **`$SIMULATION_DIR/MUL1.do`**.



# 3 Multiplier for arithmetic numbers

In this section, you will extend the multiplier previously implemented to arithmetic numbers, i.e., numbers represented in two's complement. This allows the multiplication of both positive and negative integers.

## 3.1 Algorithm

Figure 3 shows the Baugh-Wooley algorithm for the multiplication of two arithmetic numbers (signed) coded in two's complement with the same number of bits. Compared to Figure 1, several additional additions and inversions are performed here to obtain the correct result.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | $a_3$ | $a_2$ | $a_1$ | $a_0$ |
| | | | | $\times\ b_3$ | $b_2$ | $b_1$ | $b_0$ |
| | | | 1 | $\overline{b_0*a_3}$ | $b_0*a_2$ | $b_0*a_1$ | $b_0*a_0$ |
| | | | $b_1*a_3$ | $b_1*a_2$ | $b_1*a_1$ | $b_1*a_0$ | |
| | | $\overline{b_2*a_3}$ | $b_2*a_2$ | $b_2*a_1$ | $b_2*a_0$ | | |
| 1 | $b_3*a_3$ | $\overline{b_3*a_2}$ | $\overline{b_3*a_1}$ | $\overline{b_3*a_0}$ | | | |
| $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |

Figure 3 - Multiplication algorithm for signed numbers

## 3.2 Analysis

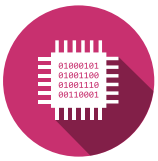
First, we will analyze the multiplication of two arithmetic numbers (signed) coded in two's complement with 4 bits. In contrast to the multiplication of natural numbers (unsigned), we must also consider negative numbers here.

Determine the largest and smallest number that can be reached with this multiplication.

## 3.3 Implementation

Create the circuit of the multiplier of Figure 3. Some circuit elements are already provided in the block **`MUL/mul4Signed`**. Complete the missing elements to ensure the functionality of the multiplier.

Using INV, AND, OR and XOR gates, complete the hierarchical diagram of the multiplier of Figure 3.

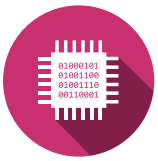



## 3.4 Simulation

Every implemented circuit must be correctly tested. The testbench **`MUL_test/mul4Signed_tb`** must verify the **complete** function of the circuit.

Simulate the testbench **`MUL_test/mul4Signed_tb`** with the simulation file **`$SIMULATION_DIR/MUL2.do`**.

How do you explain the many glitches that appear in the simulation?




# 4 Analysis

Assuming that all logic gates have the same delay of 1ns, analyze the maximum calculation delay of the multiplier for natural numbers (unsigned), shown in Figure 2.

The delay results from the number of cascaded gate paths that the signal must pass through before reaching the final result.

Determine the maximum calculation delay of the multiplier Figure 2

Propose an optimized architecture to reduce the calculation delay.