Binary Adders

Labor Digital Design

Contents

1 Goal

This lab aims to deepen the understanding of the binary representation of numbers and their processing in digital circuits. Different adder architectures are examined to analyze their operation, advantages and disadvantages, as well as their impact on calculation speed.

Two variants of adders are in focus:
- Carry-ripple adder: A basic method where the carry propagates step by step through the circuit.
- Carry-lookahead adder: An optimized version that calculates the carry in advance to reduce the delay.

2 Carry Ripple Adder

In digital circuits, adders are used to add binary numbers. A basic implementation is the Carry Ripple Adder, which consists of several cascaded adder blocks in which the carry propagates from one position to the next.

2.1 Circuit

A Carry Ripple Adder consists of several iterative blocks, each of which adds two equivalent bits (a_i, b_i) together with an incoming carry (c_i). Each block generates:
- A sum bit s_i, representing the result of the addition.
- An output carry c_{i+1}, which is passed to the next block.

The Figure 1 shows the circuit principle of such a Carry Ripple Adder:

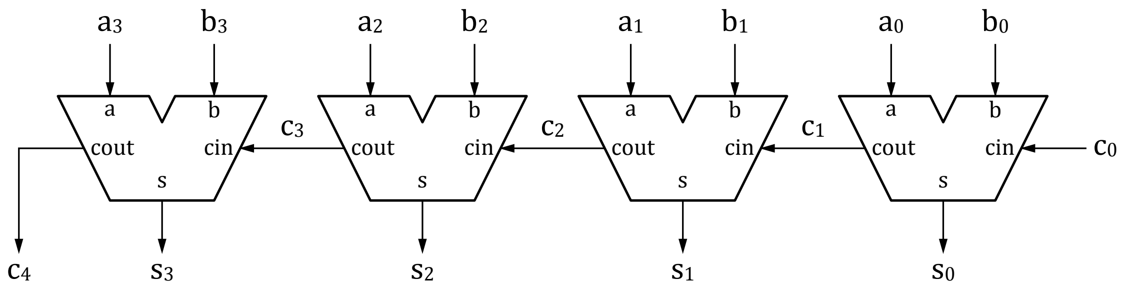

Figure 1 - Carry Ripple Adder

2.2 Implementation

Start by creating the truth table, derive the Boolean equation for the sum bit s_i and the carry c_{i+1}. Finally, develop the combinatorial circuit using INV, AND, OR, XOR gates.

Then implement the iterative scheme of a 4-bit Carry Ripple Adder in the project **`ADD/add4`**.

2.3 Simulation

Every implemented circuit must be correctly tested. The testbench **`ADD_test/add4_tester`** must verify the **full functionality** of the circuit. Answer the following questions:

1. How many test cases are theoretically necessary to verify the correct operation of the adder?
2. Which practical test cases are necessary to verify the correct operation of the adder?
3. How long does the simulation take with the practical test cases?

In the **`add4_tester`**, some *example* test cases are already implemented, you can use them for inspiration and you must adapt them. In Listing 1 you will find such a test case, which checks if the addition $s = a + b + c_{\text{in}} = 0 + 0 + 0 = 0$ is correct. Otherwise, the message **`test 1 wrong`** is displayed in the Modelsim terminal.

```vhdl
-------------------------------------------------------------------------------
-- Test 1:    0 + 0 + 0 = 0
--
a   <="0000";
b   <="0000";
cin <='0';
WAIT FOR clockPeriod;
assert (cout = '0') AND (s="0000")
  report "test 1 wrong"
  severity note;
```

Listing 1 - Example test case 1

Complete the **`ADD_test/add4_tester`** with the necessary tests to verify the full operation of the adder.

Simulate the Testbench **`ADD_test/add4_tb`** with the simulation file **`$SIMULATION_DIR/ADD1.do`**.

Investigate and find the longest propagation path in the adder.

3 Carry Lookahead Adder

This adder is optimized to reduce the delay by calculating the carry in advance.

3.1 Circuit

Figure 2 shows the circuit of a carry lookahead adder. It is similar to the carry ripple adder. However, the carry chain has been interrupted by 4 new blocks. These blocks must calculate the respective carries c_1, c_2, c_3 and c_4 with fewer sequential gates. This reduces the cascade/chain of logical gates to limit the propagation delay.

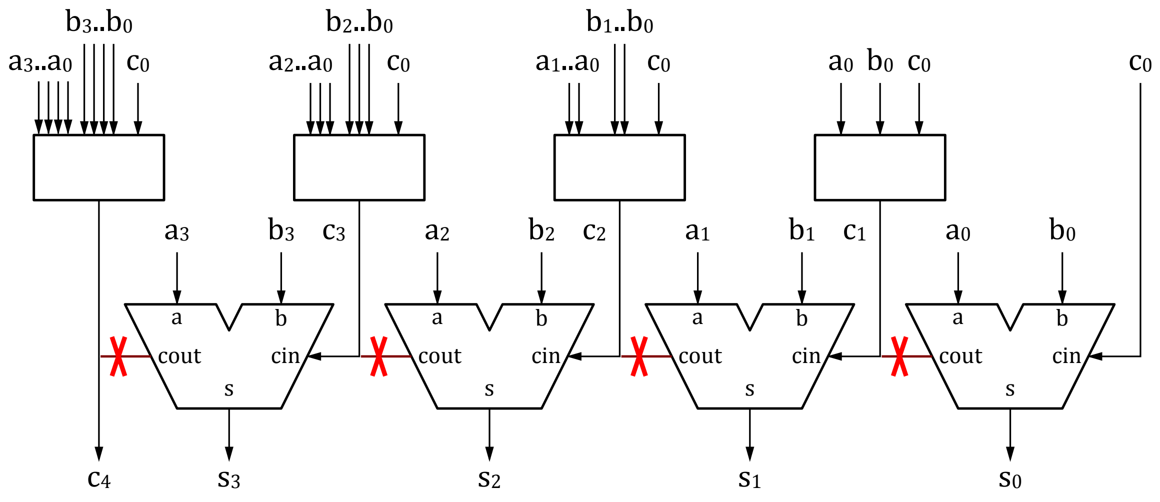

Figure 2 - Carry Lookahead Adder

3.2 Execution

First, let's analyze the adder of Figure 2.

Write the polynomial equation of c_1 and c_2. How many terms do they have?

Estimate the number of terms of the polynomial equation of c_3 and c_4.

To simplify the creation of the input carries, two transformations are introduced, g_i (generate) and p_i (propagate):

$$\begin{aligned} g_i &= a_i * b_i \\ p_i &= a_i + b_i \end{aligned} \quad (1)$$

Show that the output carry and the sum bit can be written as follows:

$$\begin{aligned} c_{\text{out}} &= g + p * c_{\text{in}} \\ s &= (\overline{g} * p) \oplus c_{\text{in}} \end{aligned} \quad (2) \qquad \begin{aligned} c_{i+1} &= g_i + p_i * c_i \\ s_i &= (\overline{g_i} * p_i) \oplus c_i \end{aligned} \quad (3)$$

Using this transformation Equation 3, write the polynomial form of c_1 to c_4 as a function of g_i, p_i and c_0.

How many terms are there in this case?

3.3 Creation

Using INV, AND, OR and XOR gates, draw the schema of the block that calculates the carries (**`ADD/lookAheadCarry`**).

Within provided 4-bit carry lookahead adder **`ADD/addLookAhead4`** complete the missing blocks **`ADD/lookAheadGp`** and **`ADD/lookAheadCarry`**.

Implement the circuit **`ADD/lookAheadGp`** which generated the signals p_i and g_i respectively.

Implement the circuit **`ADD/lookAheadCarry`** which generates the carries c_i.

3.4 Simulation

Also consider in this case how many test cases are necessary to verify the correct operation of the adder.

Simulate the Testbench **`ADD_test/addLookAHead4_tb`** with the simulation file **`$SIMULATION_DIR/ADD2.do`**.

4 Comparison

Calculate and compare the maximum propagation delay of the two created adders, one with carry ripple Figure 1 and the other with carry lookahead Figure 2.

What advantages does the carry lookahead adder **`ADD/addLookAhead4`** offer over the carry ripple adder **`ADD/add4`**? How do they behave in terms of speed and circuit size?

5 Checkout

Before leaving the lab, make sure you have completed the following tasks:

- ☐ Circuit Design
  - ☐ The carry-propagation adder **`ADD/add4`** has been designed and tested.
  - ☐ The carry-lookahead adder **`ADD/addLookAhead4`** has been designed and tested.
- ☐ Simulations
  - ☐ The specific tests of the respective testbenches (**`ADD_test/add4_tb`** and **`ADD_test/LooAhead4_tb`**) have been adapted to the circuit.
  - ☐ The two adders have been compared in terms of performance and resource usage.
  - ☐ The simulations contain no errors and all calculations are correct.
- ☐ Documentation and Projectfiles
  - ☐ Ensure all steps (design, circuit, simulations) are well-documented in your lab report.
  - ☐ Save the project to a USB stick or the shared network drive (**`\\filer01.hevs.ch`**).
  - ☐ Share files with your lab partner to ensure work continuity.