Design and Simulation of a 2-Bit Arithmetic Logic Unit (ALU)

2023-2024

Objective

The aim of this practical is to design and simulate a 2-bit Arithmetic Logic Unit (ALU) using combinational logic circuits. The ALU will carry out fundamental arithmetic and logic operations, including addition, subtraction, AND, and OR, with two 2-bit input values.

1 Addition

1.1 Implementation of 2-bit Addition Using Basic Logic Gates

To add two 2-bit binary numbers, $A = A_1A_0$ and $B = B_1B_0$, we use basic logic gates to create a circuit that performs the addition. We design this using two **full adders** from AND, OR, and XOR gates.

Inputs and Outputs

- **Inputs:** A_1, A_0, B_1, B_0, C_{in} (Carry input, initialized to 0 for the first adder)
- **Outputs:** S_1, S_0, C_{out} (Carry out)

Full Adder Logic

- **Sum Output:** $S = A \oplus B \oplus C_{in}$
- **Carry Out:** $C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B))$

Step-by-Step Example

Add $A = 10$ (2) and $B = 11$ (3):

1. First Full Adder (A0, B0, Cin=0): $S_0 = 1$, $C_1 = 0$
2. Second Full Adder (A1, B1, C1=0): $S_1 = 0$, $C_{out} = 1$
3. Result: $S_1S_0 = 01$ (1) with $C_{out} = 1$ (overflow)

Conclusion

By implementing two full adders using basic logic gates, we can effectively perform the addition of two 2-bit binary numbers while managing the carry between the bits.

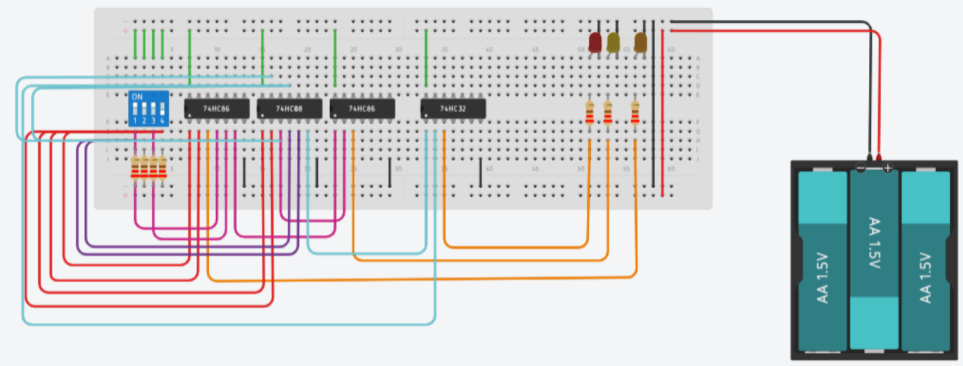

Figure 1: Circuit design for 2-bit addition

2 AND Operation

2.1 Implementation of 2-Bit AND Operation

To implement the **AND** operation for two 2-bit binary numbers $A = A_1A_0$ and $B = B_1B_0$, we perform bitwise AND on each corresponding pair of bits.

Inputs and Outputs

- **Inputs:** A_1, A_0, B_1, B_0
- **Outputs:** $C_1 = A_1 \cdot B_1$, $C_0 = A_0 \cdot B_0$

Circuit Implementation

Use two AND gates:

- One for MSB: A_1 and B_1
- One for LSB: A_0 and B_0

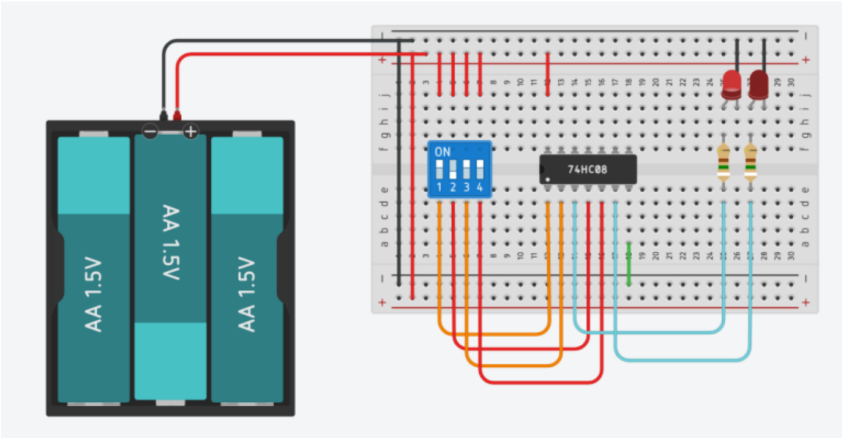

Figure 2: Circuit design for 2-bit AND operation

3 OR Operation

3.1 Implementation of 2-Bit OR Operation

To implement the **OR** operation for two 2-bit binary numbers $A = A_1A_0$ and $B = B_1B_0$, we perform bitwise OR on each corresponding pair of bits.

Inputs and Outputs

- **Inputs:** A_1, A_0, B_1, B_0
- **Outputs:** $C_1 = A_1 + B_1$, $C_0 = A_0 + B_0$

Circuit Implementation

Use two OR gates:

- One for MSB: A_1 and B_1
- One for LSB: A_0 and B_0

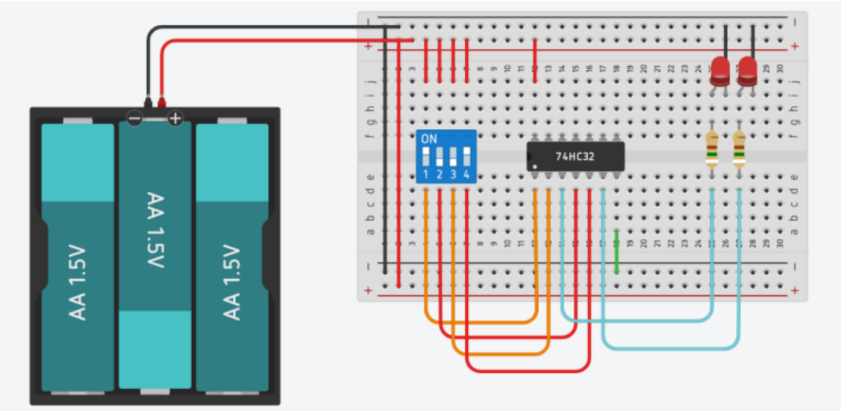

Figure 3: Circuit design for 2-bit OR operation

4 Subtraction

For subtraction, we handle only cases where the result is positive. If the result is negative, the output displays 0.

4.1 Negative Result Detection

We use the logical expression:

$$A_0' B_0 (B_1 + A_1') + A_1' B_1$$

to detect negative results.

4.2 Subtraction Using Two's Complement

We use the two's complement method to perform subtraction. The output is set to 0 when the result is negative.

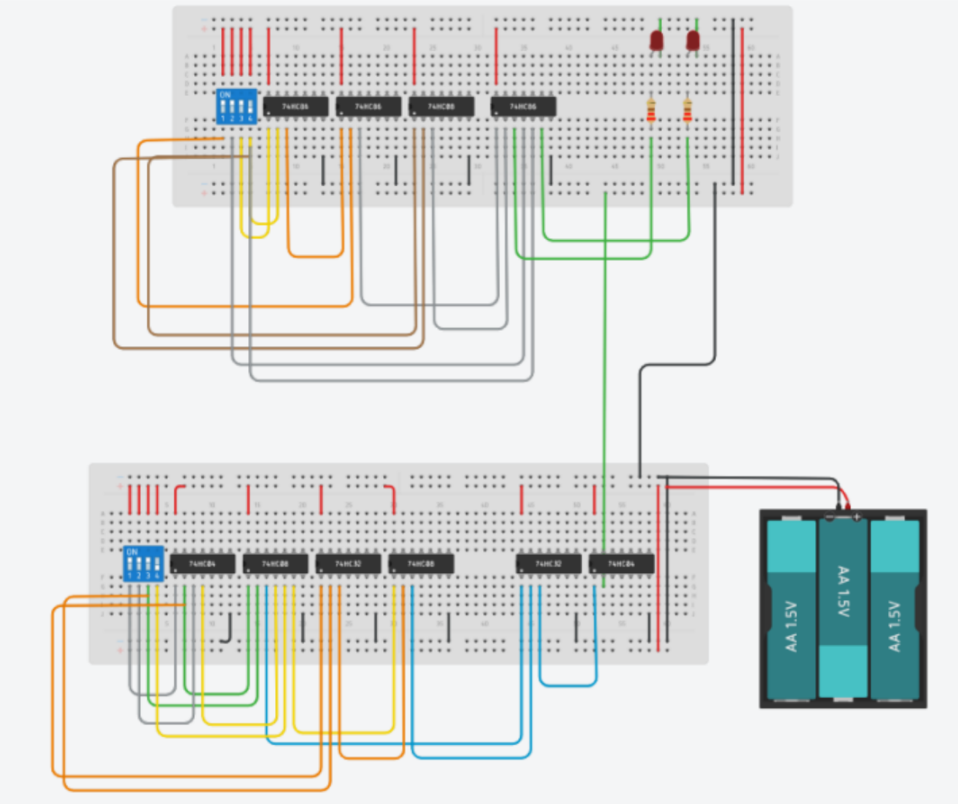

Figure 4: Circuit design for 2-bit subtraction

5 Multiplexer (MUX)

A 4-to-1 multiplexer selects one of four inputs based on two select lines S_0 and S_1.

Logical Expression

$$Y = (S_1' \cdot S_0' \cdot I_0) + (S_1' \cdot S_0 \cdot I_1) + (S_1 \cdot S_0' \cdot I_2) + (S_1 \cdot S_0 \cdot I_3)$$

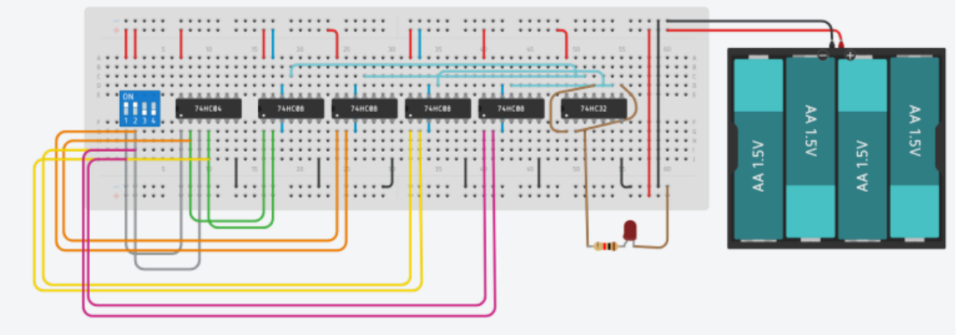

Figure 5: Circuit design for 4-to-1 multiplexer

6 ALU Implementation

We combine all individual circuits using three multiplexers to form the 2-bit ALU. Each multiplexer controls one bit of the final output, allowing the user to select the desired operation.

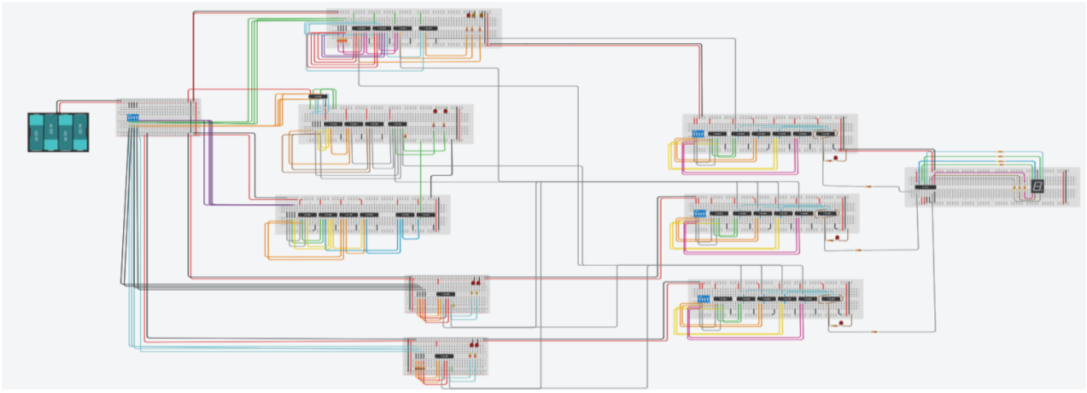

Figure 6: Complete 2-bit ALU circuit design

Implementation Links

- Addition: `https://www.tinkercad.com/things/cAnigQ77A7Y-addition?sharecode=LLxbkLNt17HHK-zVLd-owz3`
  `1CCBAxoVHcnCZETPckI`
- AND: `https://www.tinkercad.com/things/2fHm1r4Ugn4-and?sharecode=4pBMXhtPndCMvlynldH4wgxKYY3ZX-3`
- OR: `https://www.tinkercad.com/things/9Yp1nIrnEsg-or?sharecode=jjPvpOCLcjXVPdBBms1P5jbS4yezdEAkvS`
  `H5I`
- Subtraction: `https://www.tinkercad.com/things/kANkF8yFgjy-subtraction?sharecode=8mbGXt4bKrUSLhXD6`
- ALU: `https://www.tinkercad.com/things/7ysSKp0I9pY-alu?sharecode=_0XVVmkERtgFmN33kSqP8VGyaeBfv78`