Report

Topic : 4-bit ALU and 2-bit ALU

Made by

Teeranai Sangtaera Student NO.65125056

Pruettapon Paritsiraprapa Student NO.65115800

Thanaphat Tenghirun Student NO.65126955

Section. 2

2-bit ALU on the hardware level

A 2-bit ALU (Arithmetic Logic Unit) is a digital circuit that performs basic arithmetic and logic operations on 2-bit binary numbers at the hardware level. It is a fundamental component of most central processing units (CPUs) and microcontrollers. The 2-bit ALU takes two 2-bit binary inputs, performs a specified operation, and produces a 2-bit binary output along with some status flags.

Here's a basic description of the 2-bit ALU with common operations:

Inputs:

A[1:0]: The first 2-bit binary input.

B[1:0]: The second 2-bit binary input.

Control signals (OP[2:0]): These signals determine the operation to be performed by the ALU. The possible control signals and their corresponding operations are as follows:

OP[2:0] = 000: AND

OP[2:0] = 001: OR

OP[2:0] = 010: Addition (with carry-in)

OP[2:0] = 011: Subtraction (with borrow-in)

OP[2:0] = 100: Shift left (logical)

OP[2:0] = 101: Shift right (logical)

Outputs:

Result[1:0]: The 2-bit binary output that represents the result of the operation.

Carry-out (COUT): For addition, this indicates if there is a carry-out beyond the 2-bit result.

Borrow-out (BOUT): For subtraction, this indicates if there is a borrow-out beyond the 2-bit result.

Zero (ZERO): A status flag that is set to 1 when the result is zero and 0 otherwise.

Negative (NEG): A status flag that is set to 1 when the result is negative (MSB is 1) and 0 otherwise.

Operations:

a. AND:

Result[1] = A[1] AND B[1]

Result[0] = A[0] AND B[0]

COUT = 0

BOUT = 0

ZERO = 1 if (Result[1:0] == 00) else 0

NEG = Result[1]

b. OR:

Result[1] = A[1] OR B[1]

Result[0] = A[0] OR B[0]

COUT = 0

BOUT = 0

ZERO = 1 if (Result[1:0] == 00) else 0

NEG = Result[1]

c. Addition (with carry-in):

Result[1:0] = A[1:0] + B[1:0] + CIN (Carry-in)

COUT = 1 if (Result[2] == 1) else 0

BOUT = 0

ZERO = 1 if (Result[1:0] == 00) else 0

NEG = Result[1]

d. Subtraction (with borrow-in):

Result[1:0] = A[1:0] - B[1:0] - BIN (Borrow-in)

COUT = 0

BOUT = 1 if (Result[1] == 1) else 0

ZERO = 1 if (Result[1:0] == 00) else 0

NEG = Result[1]

e. Shift left (logical):

Result[1] = A[0]

Result[0] = 0

COUT = A[1]

BOUT = 0

ZERO = 1 if (Result[1:0] == 00) else 0

NEG = Result[1]

f. Shift right (logical):

Result[1] = 0

Result[0] = A[1]

COUT = A[0]

BOUT = 0

ZERO = 1 if (Result[1:0] == 00) else 0

NEG = Result[1]

Remember that this is a simplified 2-bit ALU with limited operations for illustrative purposes. In a real-world scenario, ALUs in modern CPUs are much more complex and handle larger word sizes with a wide range of operations. The ALU described here would be implemented using logic gates and flip-flops at the hardware level

Truth Table : 2-bit ALU (Addition)

Deci. Addi.	A_1	A_0	B_1	B_0	C_{in}	C_{out}	S_1	S_0	Deci. Sum
0+0	0	0	0	0	0	0	0	0	0
0+1	0	0	0	1	0	0	0	1	1
0+2	0	0	1	0	0	0	1	0	2
0+3	0	0	1	1	0	0	1	1	3
1+0	0	1	0	0	0	0	0	1	1
1+1	0	1	0	1	0	0	1	0	2
1+2	0	1	1	0	0	0	1	1	3
1+3	0	1	1	1	0	1	0	0	4
2+0	1	0	0	0	1	0	1	1	3
2+1	1	0	0	1	1	1	0	0	4
2+2	1	0	1	0	1	1	0	1	5

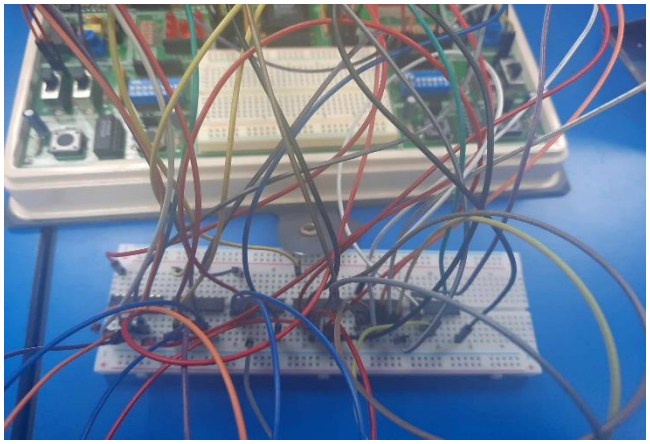
2+3	1	0	1	1	1	1	1	0	6
3+0	1	1	0	0	1	1	0	0	4
3+1	1	1	0	1	1	1	0	1	5
3+2	1	1	1	0	1	1	1	0	6
3+3	1	1	1	1	1	1	1	1	7

Truth Table : 2-bit ALU (Subtraction)

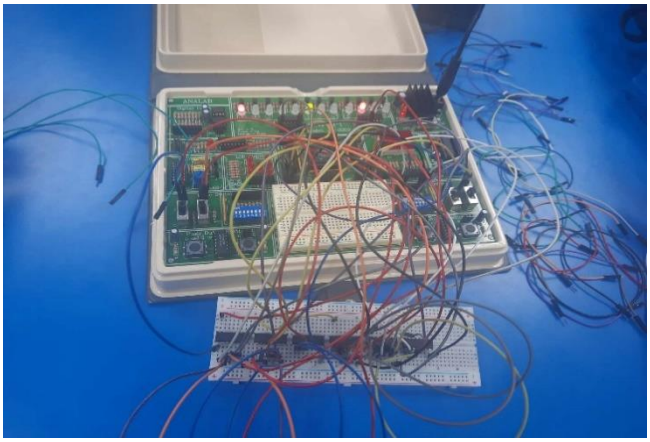
Deci. Sub.	A_1	A_0	B_1	B_0	C_{in}	C_{out}	S_1	S_0	Deci. Sum
0-0	0	0	0	0	0	0	0	0	0
0-1	0	0	0	1	0	1	1	1	-1
0-2	0	0	1	0	0	1	1	0	-2
0-3	0	0	1	1	0	1	0	1	-3
1-0	0	1	0	0	0	0	0	1	1
1-1	0	1	0	1	0	0	0	0	0
1-2	0	1	1	0	0	1	1	1	-1

1-3	0	1	1	1	0	1	1	0	-2
2-0	1	0	0	0	0	0	1	0	2
2-1	1	0	0	1	0	0	0	1	1
2-2	1	0	1	0	0	0	0	0	0
2-3	1	0	1	1	0	1	1	1	-1
3-0	1	1	0	0	0	0	1	1	3
3-1	1	1	0	1	0	0	1	0	2
3-2	1	1	1	0	0	0	0	1	1
3-3	1	1	1	1	0	0	0	0	0

Circuit connection results



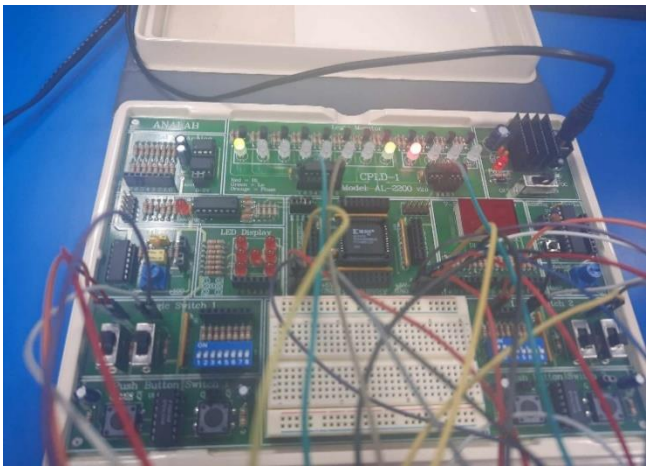
The procedure for connecting a 2-bit ALU circuit by connecting 3 logic gates AND OR XOR in connecting the ALU full adder The XOR is connected in front of the full adder circuit to change the input before it is calculated in the circuit so that our circuit can do both addition and subtraction. Next, Cin with 0 and 1 indicates whether to be addition or subtraction, Therefore 1 is subtraction, 0 is addition.



Enter binary : 11 + 10

Results : 101

$Carry_{Out}$ is 1 ,S_1 is 0 , S_0 is 1



Enter binary : 11 + 10

Results : 101

$Carry_{Out}$ is 1 ,S_1 is 0 , S_0 is 1

4 bit ALU

A 4-bit Arithmetic Logic Unit (ALU) is a digital circuit that performs various arithmetic and logical operations on 4-bit binary numbers. One of the fundamental operations an ALU can perform is addition. The work description of a 4-bit ALU addition involves designing and implementing the circuit to carry out this operation. Here's a step-by-step breakdown of the process:

Specification: Define the requirements and inputs/outputs of the 4-bit ALU addition. In this case, the ALU takes two 4-bit binary numbers as inputs (A and B) and produces a 4-bit binary sum (S) as output.

Binary Addition Algorithm: Review the binary addition algorithm. Start by adding the least significant bits (LSBs) of A and B along with a carry-in (Cin) of 0. If the sum is 2 (10 in binary), the output bit is 0, and a carry-out (Cout) of 1 is generated. If the sum is 3 (11 in binary), the output bit is 1, and a carry-out of 1 is generated. If the sum is less than 2, the output bit is the sum, and a carry-out of 0 is generated. Proceed similarly for the remaining bits, considering the carry-out generated from the previous stage as the carry-in for the next stage.

Logic Circuit Design: Design the logic circuit for each bit of the 4-bit ALU. You'll need basic logic gates like AND, OR, and XOR gates, along with some additional logic for generating the carry-out.

Implementation: Implement the designed circuit using hardware description languages (HDL) like VHDL or Verilog, or you can use a hardware design tool like Logisim to create a graphical representation of the circuit.

Simulation: Test the 4-bit ALU addition circuit using simulation software. Create test cases covering various scenarios, including cases with no carry, carry-in from LSB to MSB, and carry-out from LSB to MSB.

Verification and Debugging: Verify the correctness of the simulation results and debug any issues that may arise.

Synthesis and Implementation: If you're targeting a specific hardware platform, you'll need to synthesize the design to map it onto the target device. For FPGAs, you can use tools like Xilinx ISE or Vivado or Intel Quartus Prime for Intel FPGAs.

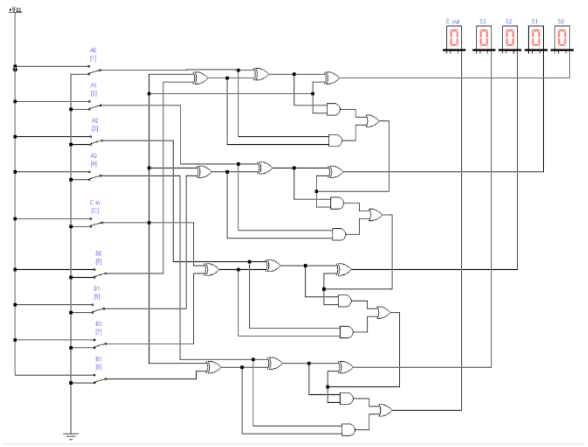
Testing on Hardware: Once synthesized, program the 4-bit ALU addition circuit onto the target hardware (such as an FPGA) and test it using real hardware with different input combinations to ensure its functionality.

Documentation: Properly document the design process, circuit diagrams, simulation results, and any other relevant information to ensure easy maintenance and understanding for future development or updates.

Keep in mind that the 4-bit ALU addition is just one of the many functions that an ALU can perform. A complete ALU may include additional functionalities like subtraction, logical operations (AND, OR, NOT), shift operations, and more, depending on the required specifications. The work description above focuses solely on the 4-bit binary addition part of the ALU.

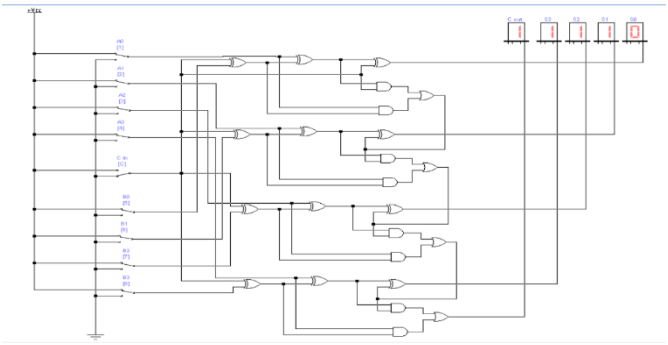
Truth table 4 bit ALU

A3	A2	A1	A0	B3	B2	B1	B0	C in	C out	S3	S2	S1	S0
0	1	0	1	0	1	0	1	0	0	1	0	1	0
1	0	1	0	0	0	1	1	0	0	1	1	0	1
1	0	0	1	0	0	1	0	0	0	1	0	1	1
0	1	0	1	0	0	1	1	0	0	1	0	0	0
1	0	1	1	1	0	0	1	0	1	0	1	0	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1	0	0	0	1	0	1
0	1	1	1	1	1	1	1	0	1	0	1	1	0
0	1	0	0	1	1	1	1	1	1	0	0	1	1
1	0	0	1	1	0	1	0	1	1	0	0	1	1
1	0	0	1	0	0	1	1	1	0	1	1	0	0
1	1	0	0	1	1	0	0	1	1	1	0	0	0
1	1	0	1	1	1	0	1	1	1	1	0	1	1
1	1	1	0	1	1	1	0	1	1	1	1	0	0
0	1	0	1	0	1	0	1	1	0	1	0	1	0
0	0	0	1	0	0	0	1	1	0	0	0	1	0



To connect a 4 bit ALU circuit, it is connected by using 4 full adder circuits connected to each other and then using 1 XOR to convert input B. Before the value is sent into the circuit to calculate so that the circuit can do both addition and subtraction. Cin with 0 and 1 indicates whether to be addition or subtraction, Therefore 1 is subtraction, 0 is addition.

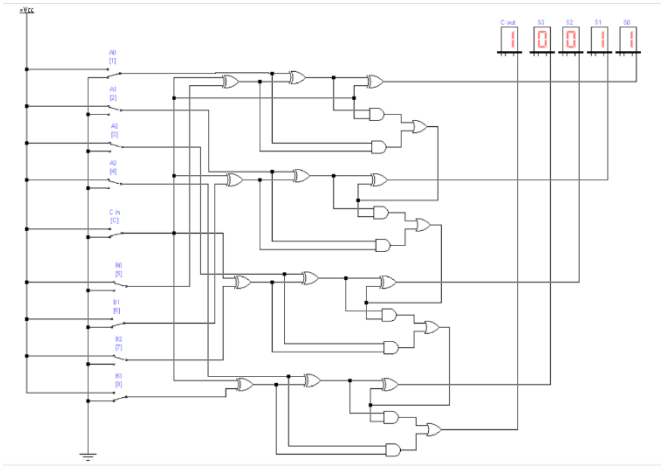
Example of addition procedure



Enter binary : 1111 + 1111

Results : 11110

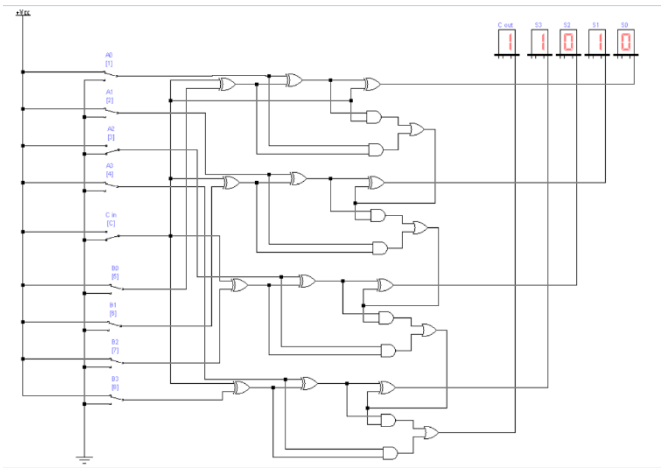
$Carry_{Out}$ is 1 ,S_3 is 1 , S_2 is 1 ,S_1 is 1 , S_0 is 0



Enter binary : 1110 + 0101

Results : 10011

$Carry_{Out}$ is 1 ,S3 is 0 , S2 is 0 ,S1 is 1 , S0 is 1

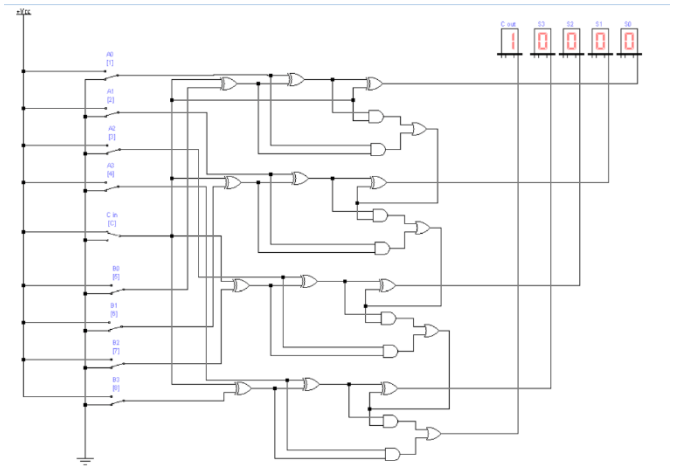


Enter binary : 1011 + 1111

Results : 11010

$Carry_{Out}$ is 1 ,S_3 is 1 , S_2 is 0 ,S_1 is 1 , S_0 is 0

Example of subtraction procedure



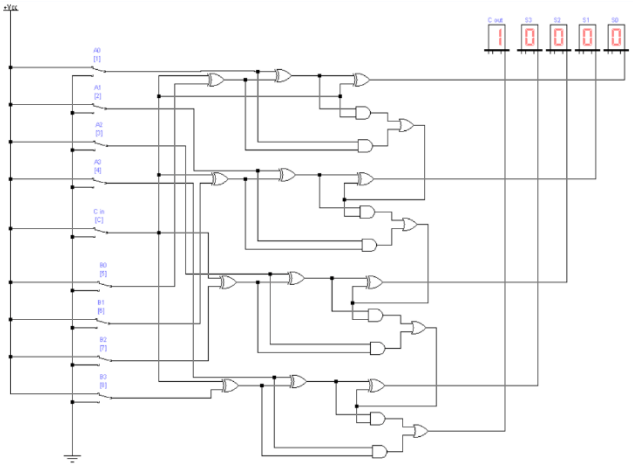
Therefore 1 is subtraction

Enter binary : 0000 + 0000

Results : 10000

$Carry_{Out}$ is 1 ,S_3 is 0 , S_2 is 0 ,S_1 is 0 , S_0 is 0

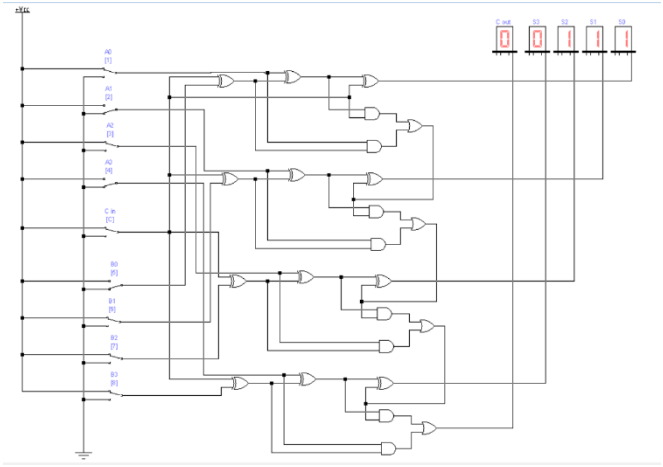
C_{out} is 1 because the circuit is now a subtraction circuit as it is connected, indicating that it is in subtraction mode.



Enter binary : 1111 + 1111

Results : 10000

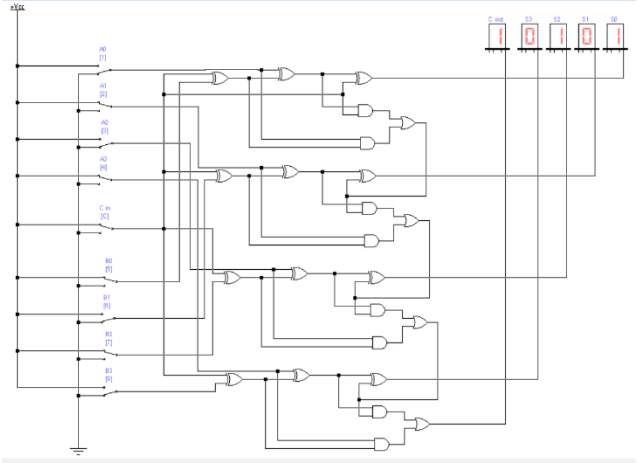
Carry$_{Out}$ is 1 ,S_3 is 0 , S_2 is 0 ,S_1 is 0 , S_0 is 0



Enter binary : 0101 + 1110

Results : 00111

Carry$_{Out}$ is 0 ,S_3 is 0 , S_2 is 1 ,S_1 is 1 , S_0 is 1



Enter binary : 1010 + 0101

Results : 10101

$Carry_{Out}$ is 1 ,S_3 is 0 , S_2 is 1 ,S_1 is 0 , S_0 is 1