ADDING TWO 2-BIT NUMBERS WITH LOGIC GATES

1 INTRODUCTION

Binary addition sits at the heart of digital electronics. Even complex CPUs reduce to networks of simple gates that compute sums and propagate carries. Here, I design and explain a combinational circuit that adds two 2-bit inputs, **$A = A_1A_0$** and **$B = B_1B_0$**, to produce a 2-bit result **$C = C_1C_0$**. Because two 2-bit numbers can sum to as high as 6 (binary `110`), I also expose a **carry-out** flag **K** to indicate overflow beyond two bits. This keeps the requested 2-bit result while remaining faithful to binary arithmetic (Mano & Ciletti, 2017; Harris & Harris, 2021).

2 DESIGN OVERVIEW AND ASSUMPTIONS

- **Inputs:** `A1, A0, B1, B0` (most to least significant).
- **Outputs:** `C1, C0` (two-bit sum modulo 4) and **K** (overflow/carry-out).
- **Assumption:** The primary result is the 2-bit sum `C1C0 = (A + B) mod 4`. The flag **K = 1** marks cases where `(A + B) ≥ 4`. If a strict 2-bit-only output is required, simply ignore **K**; if correctness across full arithmetic range matters, observe **K**.

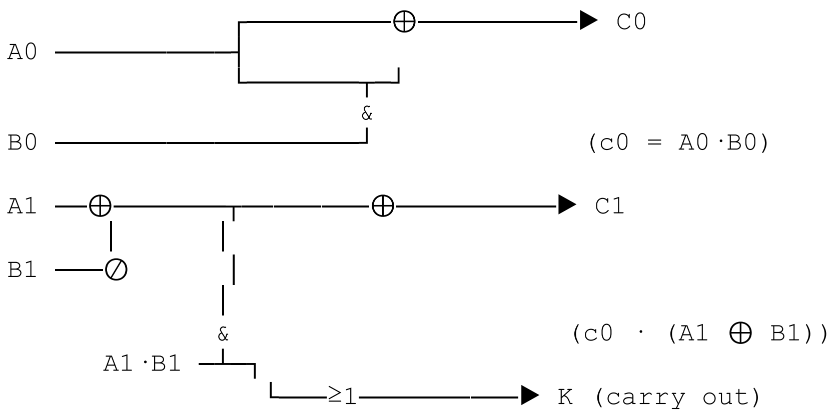
3 GATE-LEVEL DESIGN (USING AND, OR, XOR)

The circuit is a ripple of a **half-adder** for the LSB and a **full-adder** for the MSB:

- **LSB (bit 0) — Half-Adder**
  - `C0 = A0 ⊕ B0`
  - `c0 = A0 · B0` (carry from bit 0 to bit 1)
- **MSB (bit 1) — Full-Adder**
  - `C1 = (A1 ⊕ B1) ⊕ c0`
  - `K = (A1 · B1) + [c0 · (A1 ⊕ B1)]`

These equations use only **XOR**, **AND**, and **OR** gates and implement the standard adder pattern taught in digital design (Mano & Ciletti, 2017; Harris & Harris, 2021).

4 ASCII Circuit Diagram



5 Truth Table

All 16 input combinations, showing the 2-bit sum (`C1 C0`) and overflow (`K`). The last column shows the exact decimal sum for reference.

A1	A0	B1	B0	C1	C0	K	A+B (dec)
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	1
0	0	1	0	1	0	0	2
0	0	1	1	1	1	0	3
0	1	0	0	0	1	0	1
0	1	0	1	1	0	0	2
0	1	1	0	1	1	0	3
0	1	1	1	0	0	1	4
1	0	0	0	1	0	0	2
1	0	0	1	1	1	0	3
1	0	1	0	0	0	1	4
1	0	1	1	0	1	1	5
1	1	0	0	1	1	0	3
1	1	0	1	0	0	1	4
1	1	1	0	0	1	1	5
1	1	1	1	1	0	1	6

6 How Inputs Flow Through the Gates (Step-by-Step)

1. **Bit 0 addition:** The XOR gate computes the least-significant sum `C0 = A0 ⊕ B0`. In parallel, the AND gate generates the carry `c0 = A0 · B0`.
2. **Carry propagation:** The intermediate `c0` becomes an input to the MSB stage, exactly like a ripple adder.

3. **Bit 1 tentative sum:** The MSB XOR computes `A1 ⊕ B1`, which is then XORed with `c0` to produce `C1`. If `c0 = 1`, it flips that tentative sum, reflecting the addition of a "1" into the MSB.
4. **Final carry-out:** Two ways to get a carry from the MSB are OR-combined: (i) `A1 · B1 = 1` (both MSB inputs are 1), or (ii) a carry-in arrives (`c0 = 1`) **and** exactly one of `A1`, `B1` is 1 (`A1 ⊕ B1 = 1`). The resulting `K` marks overflow beyond two bits.

7 BEHAVIORAL ANALYSIS AND OUTCOME CATEGORIES

- **No-overflow sums (K = 0):** When `A + B ≤ 3`, the 2-bit output `C1C0` equals the true sum, and the circuit behaves like an exact 2-bit adder. Examples: `01 + 10 = 11` (1 + 2 = 3).
- **Overflow cases (K = 1):** When `A + B ≥ 4`, the 2-bit output wraps modulo 4 and **K** alerts the system. Examples: `01 + 11 = 00` with `K = 1` (1 + 3 = 4), `10 + 11 = 01` with `K = 1` (2 + 3 = 5).
- **Symmetry:** Swapping A and B does not change the outputs; the adder is commutative.
- **Input patterns:**
  - If **both LSBs are 1**, `c0 = 1`, guaranteeing the MSB stage sees a carry-in.
  - If **both MSBs are 1**, then `A1 · B1 = 1` and `K` will be 1 unless `c0 = 0` and `A1 ⊕ B1 = 0` (here it still produces `K = 1` because `A1 · B1 = 1`), consistent with the truth table.
    These behaviors align with standard half-adder/full-adder theory and give the circuit predictable, testable outcomes (Harris & Harris, 2021).

8 CONCLUSION

The circuit combines a half-adder at the LSB with a full-adder at the MSB to implement 2-bit addition using **XOR** for sum bits and **AND/OR** for carries. The 2-bit output `C` satisfies the assignment, while the carry flag **K** preserves arithmetic correctness by signaling overflow. This modular structure scales naturally to wider adders and mirrors the gate-level foundations used in real processors (Mano & Ciletti, 2017).

Discussion question: In a 4-bit ripple-carry adder, how would you modify the design to detect **signed overflow** (two's-complement) using only gate-level signals from the MSB stage?

References

Harris, D. M., & Harris, S. L. (2021). *Digital design and computer architecture: RISC-V edition*. Morgan Kaufmann. https://www.amazon.com/Digital-Design-Computer-Architecture-RISC-V/dp/0128200642

Mano, M. M., & Ciletti, M. D. (2017). *Digital design* (6th ed.). Pearson. https://www.pearson.com/en-us/subject-catalog/p/digital-design-with-an-introduction-to-the-verilog-hdl-vhdl-and-systemverilog/P200000003241/9780137501984

Word count: 977