Got it! I will redesign the half adder using only NAND gates, with an OR gate replacing the XOR function. The new design will:

* Use two NAND gates for the initial processing of A and B.
* Feed those outputs into another NAND gate to produce an intermediate signal.
* Use an OR gate instead of XOR to compute the Sum output.
* Ensure the Carry output is properly derived using NAND logic.

I'll provide a revised circuit diagram, truth table, and logic derivation shortly.

**Half Adder Redesign Using NAND and OR Gates**

**Overview**

A **half adder** is a basic combinational circuit that adds two single bits (A and B) and outputs a **Sum** and a **Carry** bit ([Adder (electronics) - Wikipedia](https://en.wikipedia.org/wiki/Adder_(electronics)#:~:text=sum%20is%20%5BImage%204%3A%20,2)). In the simplest form, the Sum is produced by an XOR gate and the Carry by an AND gate ([Adder (electronics) - Wikipedia](https://en.wikipedia.org/wiki/Adder_(electronics)#:~:text=sum%20is%20%5BImage%204%3A%20,2)). Here we redesign the half adder to **eliminate the XOR gate** by using an OR-based logic alternative, along with **NAND gates** as the primary building blocks. NAND gates are *universal*, meaning any logic function (NOT, AND, OR, etc.) can be constructed from them ([combinatorial logic - Can I simplify a half-adder when using only NAND gates? - Electrical Engineering Stack Exchange](https://electronics.stackexchange.com/questions/728987/can-i-simplify-a-half-adder-when-using-only-nand-gates#:~:text=The%20result%20is%20xy%27%20and,into%20the%20final%20NAND%20gate)). The goal is to implement the same truth table using only NAND and OR gates, which can be useful if XOR gates are unavailable or to demonstrate NAND’s universality. This redesigned half adder still has no carry-in and can serve as a building block for a full adder.

**Redesigned Logic Design (Using NAND & OR)**

**Replacing XOR with OR-based logic:** The XOR of two bits can be expressed in a form that uses an OR operation. One convenient form is the **“sum-of-products”** expansion:

In this expression, is AND, is OR, and is NOT. Equation (1) says the Sum output is 1 if *either* A is 1 and B is 0, **or** A is 0 and B is 1 – exactly the condition for XOR ([CS 211 Project 4: Gate Simulator](https://cs.gmu.edu/~kauffman/cs211/p4.html#:~:text=OR%20temp1%20temp2%20)). Using Boolean algebra, this formula can also be transformed into an equivalent **OR-based** form that factors out common terms:

This shows that XOR can be seen as “(A OR B) **and** (NOT (A AND B))” ([What are some good resources for completely UNDERSTANDING a half adder? : r/learnmath](https://www.reddit.com/r/learnmath/comments/1cwesx0/what_are_some_good_resources_for_completely/#:~:text=,%2F%2F%20distributive%20law)). We will base our design on these equivalent expressions for the Sum. The Carry output in a half adder remains the same:

*(In the above, we could also denote AND as ‘*’, OR as ‘+’, and NOT with an overline or prime: e.g. and .)\*

**Deriving Sum and Carry with NAND gates:** To implement the above logic using only NAND and OR gates, we use NAND gates to create the needed inverted and AND terms, then use an OR gate to combine results for Sum. Specifically:

* **Inversion (NOT)**: A NAND gate with both its inputs tied together acts as a NOT gate. Using this, derive ¬A and ¬B:
  + NAND1: Inputs (A, A) → Output = ¬A
  + NAND2: Inputs (B, B) → Output = ¬B
* **Partial product terms for Sum** (from Equation 1):
  + NAND3: Inputs (A, ¬B) → Output = ¬(A ∧ ¬B). This output is the negation of the first term in (1).
  + NAND4: Inputs (¬A, B) → Output = ¬(¬A ∧ B). This is the negation of the second term in (1).
  + We now have the complements of the two required terms. To get the actual terms and , we can invert each of the above outputs with another NAND used as NOT:
    - NAND5: Inputs (output of NAND3, output of NAND3) → Output = (double negation of NAND3’s result)
    - NAND6: Inputs (output of NAND4, output of NAND4) → Output =
* **OR for Sum**: Feed the two product terms into an OR gate to produce Sum:
  + OR1: Inputs (output of NAND5, output of NAND6) → Output = **Sum** = .
* *(Alternatively, one could implement the OR with NAND gates as well by applying De Morgan’s law: by NAND-ing the negated inputs (*[*combinatorial logic - Can I simplify a half-adder when using only NAND gates? - Electrical Engineering Stack Exchange*](https://electronics.stackexchange.com/questions/728987/can-i-simplify-a-half-adder-when-using-only-nand-gates#:~:text=The%20result%20is%20xy%27%20and,into%20the%20final%20NAND%20gate)*). However, here we use a literal OR gate as specified.)*
* **Carry generation**: Use NAND to get the carry:
  + NAND7: Inputs (A, B) → Output = ¬(A ∧ B) (this is the complement of Carry).
  + NAND8: Inputs (output of NAND7, output of NAND7) → Output = **Carry** = .

**Revised Circuit Structure:** In the new design, inputs A and B go through the network of NAND gates as described to derive the necessary signals, and the Sum is obtained via an OR gate instead of a direct XOR gate. The Carry is obtained via NAND gates functioning as an AND. The figure below conceptually illustrates the logic (each NAND is labeled by its function):

* A and B each feed into inverting NAND gates to produce ¬A and ¬B.
* Those outputs, together with original A and B, feed into two NAND-based AND sub-circuits to produce the **partial sums** and .
* An OR gate then combines the partial sums to yield the **Sum** output.
* In parallel, A and B also feed a NAND pair to produce the **Carry** output (equivalent to A AND B).

*(This implementation uses a total of 8 NAND gates plus 1 OR gate. Note that an alternative implementation based on Equation (2) could reduce the gate count by using the expression . In that case, one OR gate produces , one NAND produces , and a final NAND (with an inversion) can combine them to get the Sum. The core concept remains the same – using OR and NAND in place of XOR.)*

**Truth Table Verification**

The redesigned circuit yields the same truth table as a standard half adder. We verify that for all input combinations of A and B, the Sum and Carry match the expected results:

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **Sum** | **Carry** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

* *When A=0, B=1 (or A=1, B=0):* the Sum is 1 and Carry is 0. In our circuit, one of the partial NAND paths will produce a 1 (for the input that is 1 while the other is 0), and the OR gate outputs 1 ([CS 211 Project 4: Gate Simulator](https://cs.gmu.edu/~kauffman/cs211/p4.html#:~:text=OR%20temp1%20temp2%20)). The carry NAND pair outputs 0 (since A and B are not both 1).
* *When A=1, B=1:* both partial terms and evaluate to 0, so the OR gate outputs 0 (Sum=0). Meanwhile, the carry logic yields 1, since A and B being 1 makes .
* *When A=0, B=0:* both the Sum terms are 0, so Sum=0, and obviously Carry=0.

This matches the expected behavior of a half adder ([Adder (electronics) - Wikipedia](https://en.wikipedia.org/wiki/Adder_(electronics)#:~:text=Inputs%20Outputs%20A%20B%20C,1%201%201%201%200)). We have essentially implemented the XOR function through a combination of NAND and OR gates, which the truth table confirms.

**Boolean Expressions for Sum and Carry**

From the NAND-OR design, we can write the final simplified Boolean expressions for the outputs (confirming they equal the standard half adder outputs):

* **Sum** :

This is equivalent to (A XOR B) ([What are some good resources for completely UNDERSTANDING a half adder? : r/learnmath](https://www.reddit.com/r/learnmath/comments/1cwesx0/what_are_some_good_resources_for_completely/#:~:text=Via%20truth%20table%2C%20you%20can,A%20XOR%20B)), even though we did not use an XOR gate directly. We can also express this as ([What are some good resources for completely UNDERSTANDING a half adder? : r/learnmath](https://www.reddit.com/r/learnmath/comments/1cwesx0/what_are_some_good_resources_for_completely/#:~:text=,%2F%2F%20distributive%20law)), which is the OR-based form used in the design.

* **Carry** :

(This was obtained via NAND gates, since a NAND followed by a NAND-inverter yields an AND.) ([Adder (electronics) - Wikipedia](https://en.wikipedia.org/wiki/Adder_(electronics)#:~:text=and%20an%20AND%20gate%20for,of%20an%20%20155%20OR))

These formulas confirm the logic: *Sum is 1 if exactly one of A or B is 1*, and *Carry is 1 if both A and B are 1*. The use of NAND gates in the implementation does not change the algebraic form of these expressions; it only changes how we physically realize them.

**Rationale for OR-based Approach and Extendability**

**Why replace XOR with OR-based logic?** In many practical scenarios, XOR gates might not be readily available as basic components, whereas AND, OR, and NAND gates are common. By expressing in terms of OR, AND, and NOT operations, we can build the **Sum** using simpler gates. Here we chose an OR-based formulation of XOR because it fit naturally with using NAND gates to generate the needed sub-expressions (inverters and ANDs). Essentially, we decomposed the XOR into smaller gates: this showcases the **universality of NAND gates**, and it avoids a dedicated XOR component. Using the OR gate for combination also aligns with the typical sum-of-products design approach taught in boolean algebra (where a sum output is often the OR of minterms that produce a 1). This method is straightforward to derive from a truth table using Karnaugh maps or boolean simplification, and it’s logically equivalent to XOR ([What are some good resources for completely UNDERSTANDING a half adder? : r/learnmath](https://www.reddit.com/r/learnmath/comments/1cwesx0/what_are_some_good_resources_for_completely/#:~:text=Via%20truth%20table%2C%20you%20can,A%20XOR%20B)).

**Use of NAND gates strategically:** NAND gates were used because any required NOT or AND operation can be built with them. In our design, NAND gates handle all the *processing of A and B*:

* Inversions (NOT) for generating ¬A, ¬B, and any needed double-negatives.
* AND functionality (via NAND followed by inversion) for generating , , and .

By using NAND gates in this way, we ensure the circuit could be built even if only NAND ICs are available. (In fact, a half adder can be implemented with five NAND gates alone ([Adder (electronics) - Wikipedia](https://en.wikipedia.org/wiki/Adder_(electronics)#:~:text=,implemented%20with%20five%20NAND%20gates)), which is a known alternative design.) Here we included a single OR gate to explicitly perform the OR in the Sum logic, making the operation clear. If even the OR gate were not available, the final OR could be simulated with NAND gates as well by using De Morgan’s law ([combinatorial logic - Can I simplify a half-adder when using only NAND gates? - Electrical Engineering Stack Exchange](https://electronics.stackexchange.com/questions/728987/can-i-simplify-a-half-adder-when-using-only-nand-gates#:~:text=The%20result%20is%20xy%27%20and,into%20the%20final%20NAND%20gate)), but that would add a couple more NAND gates.

**Extending to a full adder:** The redesigned half adder is fully equivalent in function to the standard one, so it can be used in constructing a full adder. One common construction of a full adder is to use **two half adders and an OR gate** ([Adder (electronics) - Wikipedia](https://en.wikipedia.org/wiki/Adder_(electronics)#:~:text=,2)) ([Adder (electronics) - Wikipedia](https://en.wikipedia.org/wiki/Adder_(electronics)#:~:text=match%20at%20L372%20connected%20to,out)). In our case, because the half-adder already uses an OR gate for Sum, we can easily integrate a second half-adder (for adding the carry-in) and then use an OR gate to combine carry outputs. For example, a full adder can be made by connecting two of these NAND-OR half adders: one half adder computes the preliminary sum of A and B, the second half adder adds the carry-in to that sum, and an OR gate merges the two half-adders’ carry outputs ([Adder (electronics) - Wikipedia](https://en.wikipedia.org/wiki/Adder_(electronics)#:~:text=A%20full%20adder%20can%20also,in%7D%7D%5D%20as)). The result is a Sum bit equal to and a Carry-out bit equal to , which again can be realized with NAND and OR gates. Because our half-adder design already relies on NAND (universal gates) and OR, it fits seamlessly into this larger construction. In other words, replacing XOR with basic gates does not hinder extension – it actually mirrors how you’d derive a full adder logically. The approach demonstrates how complex functions like adders can be built from simpler gates, underlining the principle that any logic circuit (including adders) can be constructed from NAND gates ([Adder (electronics) - Wikipedia](https://en.wikipedia.org/wiki/Adder_(electronics)#:~:text=,implemented%20with%20five%20NAND%20gates)) and a handful of standard gates.

**Conclusion:** The half adder has been successfully redesigned using only NAND gates (for generating required signals) and an OR gate (to replace the XOR function). The revised circuit produces the correct Sum and Carry outputs for all input combinations, as verified by the truth table. By using an OR-based logic formulation for the Sum, we eliminated the need for a dedicated XOR gate while using NAND gates to implement the necessary sub-logic. This design meets the specifications for a half adder with no carry-in, and it serves as a valid foundation that can be extended to a multi-bit addition system (full adder and beyond) ([CS 211 Project 4: Gate Simulator](https://cs.gmu.edu/~kauffman/cs211/p4.html#:~:text=)). The exercise highlights how using universal gates like NAND, combined with fundamental Boolean transformations, allows flexibility in digital design without changing the functional behavior of the circuit.

**Sources:**

1. Kauffman, J. *CS 211 Project 4: Gate Simulator* – Example of constructing XOR from NOT, AND, OR gates ([CS 211 Project 4: Gate Simulator](https://cs.gmu.edu/~kauffman/cs211/p4.html#:~:text=OR%20temp1%20temp2%20)).
2. Reddit discussion on half-adder logic – shows and its equivalence to sum-of-products ([What are some good resources for completely UNDERSTANDING a half adder? : r/learnmath](https://www.reddit.com/r/learnmath/comments/1cwesx0/what_are_some_good_resources_for_completely/#:~:text=,%2F%2F%20distributive%20law)).
3. *Adder (electronics)* – Wikipedia article on half adders (using XOR & AND) and their NAND-gate implementation ([Adder (electronics) - Wikipedia](https://en.wikipedia.org/wiki/Adder_(electronics)#:~:text=sum%20is%20%5BImage%204%3A%20,2)) ([Adder (electronics) - Wikipedia](https://en.wikipedia.org/wiki/Adder_(electronics)#:~:text=,implemented%20with%20five%20NAND%20gates)).
4. StackExchange – explanation of simulating OR with NAND gates (universal gate principle) ([combinatorial logic - Can I simplify a half-adder when using only NAND gates? - Electrical Engineering Stack Exchange](https://electronics.stackexchange.com/questions/728987/can-i-simplify-a-half-adder-when-using-only-nand-gates#:~:text=The%20result%20is%20xy%27%20and,into%20the%20final%20NAND%20gate)).