Below is a consolidated set of notes and diagrams for **Lab Two: Building With Gates** (Carry-In = 0). This focuses on designing a **half adder** block (no carry-in) and documenting how the logic is derived, including the roles of NAND vs. AND gates, truth tables, and K-map derivations.

**1. Overview**

For this step, we build a simpler adder block where the carry-in is held at zero. In other words, this is a **half adder** that takes two inputs (A and B) and produces two outputs:

* **Sum** (the 1-bit result of adding A and B)
* **Carry** (the carry-out bit if both inputs are 1)

Initially, you may have seen extra signals or confusion around having multiple inputs for A and B. The fix:

* Ensure there are only two main inputs, A and B.
* Carry-in is tied to zero (no external carry-in).

Once that is clarified, we can derive and implement the half adder.

**2. Half Adder Circuit**

**2.1 Logic Description**

A half adder’s primary components are:

1. **XOR Gate** to generate the Sum bit.
2. **AND Gate** to generate the Carry bit.

When and , the XOR outputs 1, and AND outputs 0.  
When and , the XOR outputs 0, and AND outputs 1.

**2.2 Truth Table (Half Adder)**

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

**2.3 Example Inputs**

* **Case A=0, B=1**
  + XOR(0,1) = 1 → **Sum** = 1
  + AND(0,1) = 0 → **Carry** = 0
* **Case A=1, B=0**
  + XOR(1,0) = 1 → **Sum** = 1
  + AND(1,0) = 0 → **Carry** = 0
* **Case A=1, B=1**
  + XOR(1,1) = 0 → **Sum** = 0
  + AND(1,1) = 1 → **Carry** = 1

**2.4 Circuit Diagram (Half Adder)**

A─────┬────────►[XOR]───────► Sum  
 │ ▲  
 │ │  
 B─────┼──────────┘  
 │  
 └─────────►[AND]──────► Carry

* **XOR Gate** handles the bit-wise sum:
* **AND Gate** handles the carry out:

**3. NAND vs. AND Considerations**

You can realize the same half-adder function using **NAND** gates (often useful when minimizing a circuit to one gate family). Recall the NAND truth table:

|  |  |  |
| --- | --- | --- |
| A | B | A NAND B |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

A pure NAND-based half adder will typically involve inverters or strategic wiring of NAND gates to replicate the XOR and AND functionality. For instance:

* **XOR** can be built from a combination of NAND gates and inverters.
* **AND** can be built from a NAND gate plus an inverter on its output (because AND is just the inversion of NAND).

**3.1 K-Map for NAND Output**

A quick K-map for (i.e., NAND) looks like:

B=0 B=1  
A=0 1 1  
A=1 1 0

Which gives the SOP form:

If you wish to invert it back to get AND, you can feed the NAND output through an inverter or a second NAND gate with its inputs tied together.

**4. Putting It All Together**

1. **Set Inputs**
   * Two active inputs: A and B.
   * Carry-In is grounded or forced to 0.
2. **Use XOR for Sum**
   * The sum bit is high (1) exactly when A and B differ.
3. **Use AND (or NAND + Inverter) for Carry**
   * The carry bit is high (1) only if A and B are both 1.
4. **Test With Sample Inputs**
   * Verify each row of the truth table to confirm correct Sum and Carry.

**5. Additional Notes & Next Steps**

* **Full Adder Extension**: Once a half adder is verified, a **full adder** can be built by chaining two half adders plus an OR (or NAND-based equivalent) to handle carry-in and carry-out.
* **NAND-Only Implementation**: If you must minimize your design to a single gate type, replicate XOR and AND logic using combinations of NAND gates.
* **Fixed vs. Configurable Carry-In**: In many designs, you later extend the half adder to a full adder by adding a Cin input. For now, your half adder’s carry-in is permanently 0.

**6. Evaluation / Possible Refinements**

* **Clarity**: You may want to label your gate outputs clearly (“Sum” and “Carry”) in any schematic tool.
* **Completeness**: Show intermediate signals if using purely NAND gates to create XOR.
* **Consistency**: Confirm that your final circuit diagram matches the truth table exactly (especially if mixing NAND-based logic with inverters).

Overall, this half adder design provides the building block for more complex adders (full adders, ripple-carry adders, or carry-lookahead adders). By fixing carry-in to zero, you streamline the logic to handle just A and B in this initial step.

**Solution Endpoint**:  
A half adder circuit with no carry-in has been designed, verified with a truth table, and illustrated in schematic form using XOR and AND (or NAND-based equivalents). This block can now be integrated or extended toward a full adder.