Comprehensive, **ready-to-adapt** template for your Lab Two report

Below is a comprehensive, **ready-to-adapt** template for your Lab Two report. It weaves together all the required steps—**from creating truth tables to verifying them in LTSpice**—and shows how to **optimize the LSB block** when carry-in is fixed at 0. Feel free to tailor the headers, numbering, and depth of explanation to your course requirements or instructor preferences.

Week 3 (23, 25) Week 3 Finish ALU, registers, memory and computer block diagram. CIS 240 Micro Arch & Prog (31563)

Lab Two: Building with Gates

Due Date: Monday, 2/3

1. Introduction

1.1 Objective

Briefly state the purpose of this lab. For example:

> *"The objective of this lab is to design, simulate, and test a single-bit adder block with carry-in = 0 using NAND gates and other basic logic in LTSpice. Then, we optimize the Least Significant Bit (LSB) block by removing the carry-in input entirely, creating a simplified half-adder."*

1.2 Overview

Give a high-level overview of Part 1 and Part 2:

> *"In **Part 1**, we build a single-bit adder block that assumes Carry_In = 0. In **Part 2**, we optimize by creating a simplified LSB block without any carry-in signals. We verify both designs via truth tables, Boolean derivations (SOP), and LTSpice simulations. Finally, we integrate the new LSB block into the main design."*

---

2. Part 1: Main Block Design (Carry-In = 0)

2.1 Understanding the Problem

- **Block Function**: This single-bit adder (LSB) adds two bits **A[0]** and **B[0]**, with a fixed carry-in = 0. Therefore, it behaves like a half-adder: [\text{Sum}[0] \leftarrow A[0] \oplus B[0], \quad \text{Carry_Out} \leftarrow A[0] ,\wedge, B[0].]
- **Why Carry_In = 0?** Because we are focusing on the least significant bit (initial stage of addition), there is no incoming carry from a previous bit.

2.2 Create Truth Tables

2.2.1 SUM Truth Table

A[0]	B[0]	Sum[0]	Explanation
0	0	0	0+0=0 (no carry)
0	1	1	0+1=1 (no carry)
1	0	1	1+0=1 (no carry)
1	1	0	1+1=2 → 0 + carry=1

2.2.2 CARRY-OUT Truth Table

A[0]	B[0]	Carry_Out	Explanation
0	0	0	No carry generated
0	1	0	0+1=1 → no carry
1	0	0	1+0=1 → no carry
1	1	1	1+1=2 → carry generated (2 in base 2)

2.3 Derive the Logic (SOP Method)

- **Minterm Identification**: We look at which rows produce output = 1.

2.3.1 Sum[0] = A'B + AB'

- SOP (unsimplified) = ((\overline{A} B) + (A ,\overline{B}))
- Recognizable as (A \oplus B) (XOR).

2.3.2 Carry_Out = A·B

- SOP (unsimplified) = ((A B))
- Recognizable as the AND function: (A \wedge B).

2.4 Enter the Circuits into LTSpice

1. **Schematic Layout**
   - **Inputs**: Label them `A[0]` and `B[0]`.
   - **NAND / Inverter Pairs** (for XOR-like behavior):
     - One NAND gate handles ((A, \overline{B})).
     - Another NAND gate handles ((\overline{A}, B)).
   - **Final OR (or "orx") Gate**: Combines outputs of the two NAND gates to produce `Sum[0]`.
   - **Carry Logic**:
     - You can use a separate AND gate or a NAND-based approach for `Carry_Out = A·B`.
2. **Why Use NAND + Inverters?**
   - NAND gates are universal. Using inverters on the inputs allows each NAND output to replicate part of an XOR function.

2.5 Create a Symbol for the Block

- **Symbol Creation Steps** in LTSpice:
  1. In your schematic, select **Hierarchy** → **Open Symbol** or **Edit** → **Create Symbol**.
  2. Draw the shape (rectangle or custom).
  3. Add pins labeled `A[0]`, `B[0]`, `Sum[0]`, `Carry_Out` (and ground or VDD if needed).
  4. Save as `.asy` file.

2.6 Develop a Test Schematic

- **Top-Level Test Schematic**
  - Place your new symbol (the block) as a single subcircuit.
  - Drive `A[0]` and `B[0]` with **pulse** or **voltage sources** that systematically test 00, 01, 10, 11.
  - Connect outputs to waveform probes for `Sum[0]` and `Carry_Out`.

2.7 Simulation and Results

- **Transient Simulation**
  - Configure a transient run (e.g., 0 to 100ns).
  - Step your inputs through all combinations of 0 and 1.
- **Waveforms**
  - Verify that `Sum[0]` and `Carry_Out` match the truth tables:
    - `Sum[0]` high only when inputs differ.
    - `Carry_Out` high only when both inputs are 1.

2.8 Backup and Submission of Design Files

- **Files to Backup**
  1. **.asc** – The LTSpice schematic(s).
  2. **.asy** – The block symbol(s).
  3. **Screenshots** – For documentation.

---

3. Part 2: LSB Block Optimization (No Carry-In Input)

3.1 Understanding the Problem

- Since the LSB (Least Significant Bit) has no prior bit feeding a carry, it's effectively the same half-adder scenario: [\text{Sum}[0] = A[0] \oplus B[0], \quad \text{Carry_Out} = A[0] ,\wedge, B[0].]
- **Difference from Part 1**: You eliminate any wire or pin referencing "Carry_In." This block strictly depends on `A[0]` and `B[0]`.

3.2 Create a Truth Table

It is identical to the half-adder table in Part 1 (carry_in not present here). For completeness:

A[0]	B[0]	Sum[0]	Carry_Out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

3.3 Derive the Logic

- **Sum**: (A \oplus B)
- **Carry_Out**: (A \cdot B)

(Exactly the same as the standard half-adder equations.)

3.4 Implementation in LTSpice

- **Schematic**: Even simpler because you no longer have to handle `Carry_In`.
- **Gate-Level**:
  - 2 NAND gates + 2 inverters to produce XOR for Sum.
  - 1 AND gate (or NAND with an inverter) for Carry_Out.

3.5 Create Symbol and Test Schematic

- **Symbol**: Only pins `A[0]`, `B[0]`, `Sum[0]`, `Carry_Out`.
- **Test**:
  - Drive inputs (0→1 transitions) over time.
  - Probe outputs to confirm correct logic.

3.6 Simulation Results

- Capture waveforms or a summary table.
- Confirm that `Sum[0]` toggles when (A \neq B), and `Carry_Out` is high when (A = B = 1).

3.7 Integration: Replace Original LSB

1. **Remove** the old LSB symbol from Part 1.
2. **Insert** your new simplified LSB block.
3. **Simulate** the entire design again. Confirm that the integrated circuit still produces valid outputs for all tested conditions.

---

4. Discussion

- **Key Insights**
  - Compare how both designs (Part 1 vs. simplified LSB) behave identically when Carry_In=0.
  - Note any design challenges (e.g., confusion about using NAND with inverters for XOR, or dealing with partial gates).
  - Mention how half-adders feed into larger adders if needed.
- **Carry Propagation**
  - Even though this block has no carry input, in a multi-bit adder, the carry-out of this block feeds the next stage's carry-in.

---

5. Conclusion

- **Summary**

  > *"Through SOP derivations and LTSpice simulations, we confirmed that both the main block (with Carry_In=0) and the simplified LSB design correctly implement half-adder behavior. The new block (no carry input) was integrated successfully into the overall design."*

- **Next Steps**
  - Possibly mention how this LSB block could expand into a full 4-bit or n-bit ripple-carry adder.
  - Or note if you plan to refine any timing/power considerations.

---

6. Appendix (Optional)

- **Extra Screenshots**
  - Additional waveforms, timing diagrams, or alternative schematic views.
- **References**
  - Textbooks, lecture slides, or websites that guided your Boolean logic or SOP derivations.

Final Checklist

4. **Truth Tables**
   - Part 1 (with carry-in=0)
   - Part 2 (optimized LSB, no carry-in)
5. **SOP Derivations**
   - Show minterm expansions (if required by instructor).
   - Final expressions for Sum and Carry.
6. **LTSpice Schematics**
   - Screenshots or `.asc` references (for both blocks).
   - Symbol files (`.asy`) for each block.
7. **Simulation Results**
   - Waveforms or tabular results verifying all input combinations.
8. **Integration**
   - Replacing the old LSB with the simplified version.
9. **Discussion & Conclusion**
   - Observations, pitfalls, timing analysis, etc.

Tips & Reminders

10. **Label everything clearly** in your schematics (inputs, outputs, intermediate nets).
11. **Keep your backup files** (`.asc` and `.asy`) in one folder to avoid losing references in LTSpice.
12. **Include screenshots** of your waveforms with input transitions (00 → 01 → 10 → 11) to demonstrate correctness.
13. **If you used pulse sources** (in the `Voltage` component settings), document how you set up `V1`, `V2`, `TD`, `TR`, `TF`, and the stepping intervals.

Good luck with your report submission!

By following this structured outline, you'll **(1)** clearly show the half-adder logic for Carry_In=0, **(2)** demonstrate how to simplify the LSB block by removing the carry-in, and **(3)** back up your designs with LTSpice simulations and waveforms. This complete set of documentation meets the lab requirements and provides a solid foundation for any future expansion to multi-bit adders.