Lab 07

Part A – Return of the Two-High-Bit Challenge

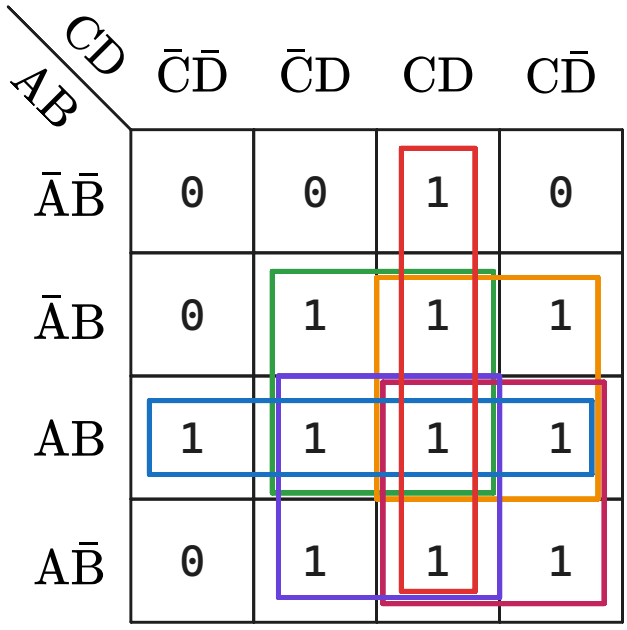
Description

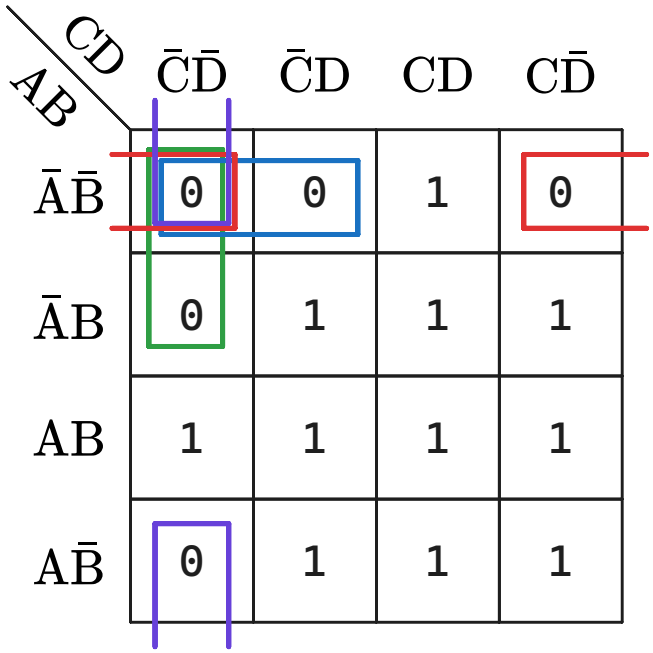
In this lab module, you will continue working on the two-high-bit detection challenge from Labs 05B and 06C. You will derive more efficient hardware implementations and consider factors of power, speed, and complexity.

Procedure

K-Map

1. Consider the solutions to the K-Map for sum-of-product (left) and product-of-sum (right) below.





2. The equations for these are as follows:
   - For sum-of-product:

$$\mathrm{O} = \mathrm{AB} + \mathrm{CD} + \mathrm{BD} + \mathrm{BC} + \mathrm{AD} + \mathrm{AC}$$

   - For product-of-sum:

$$\overline{\mathrm{O}} = \overline{\mathrm{A}}\,\overline{\mathrm{B}}\,\overline{\mathrm{C}} + \overline{\mathrm{A}}\,\overline{\mathrm{B}}\,\overline{\mathrm{D}} + \overline{\mathrm{A}}\,\overline{\mathrm{C}}\,\overline{\mathrm{D}} + \overline{\mathrm{B}}\,\overline{\mathrm{C}}\,\overline{\mathrm{D}}$$
$$\mathrm{O} = (\mathrm{A} + \mathrm{B} + \mathrm{C}) \cdot (\mathrm{A} + \mathrm{B} + \mathrm{D}) \cdot (\mathrm{A} + \mathrm{C} + \mathrm{D}) \cdot (\mathrm{B} + \mathrm{C} + \mathrm{D})$$

Gate Implementation

3. Below, draw the hardware for both SoP and PoS, consisting of 2-input `AND` and `OR` gates.

Efficient Gate Implementation

- As a brief review of what has been covered in the lecture:
  - Every gate consists of a small number of transistors.
  - The simplest gate, the inverter, or `NOT` gate, costs 2 transistors.
  - `NAND` and `NOR` both cost 4 transistors.
  - However, `AND` and `OR` are actually implemented as inverted `NAND` and `NOR`, respectively.
    - Thus, each `AND` or `OR` gate costs 6 transistors.
  - If you investigate the truth tables of `NAND` and `NOR`, you will notice the following:
    - A `NAND` gate can be represented as an `OR` gate with inverted inputs.
    - A `NOR` gate can be represented as an `AND` gate with inverted inputs.
  - You can assume that the inputs are available as both active high and active low (i.e., A and $\bar{\mathrm{A}}$ are both available).

4. Implement the hardware as 2-input `NAND` (for sum-of-product) and `NOR` (for product-of-sum) gates below by following these steps:
   - (0) Implement as `AND`/`OR` gates, and count transistors (6 per `AND`/`OR` gate).
     - You have this from Step 3.
   1. "Double Bubbles" on all internal connectors
      - Invert outputs and inputs for each line that does not lead outside the circuit
   2. Circle all `NAND` (for sum-of-product) or `NOR` (for product-of-sum) gates
   3. Count transistors and calculate total transistors saved
      - 4 transistors per `NAND`/`NOR` gate
      - 2 transistors per `NOT` gate (inverter)
   4. Determine the critical path
      - Count the number of the longest sequence of gates and inverters between input and output

- The next page is left intentionally blank to provide space for you to draw the implementations.

Gate Implementation Analysis

5. How many transistors do your `AND`/`OR` implementations from Step 3 cost? Do the SoP and PoS implementations differ in cost?
6. How many transistors does your `NAND`/`NOR` implementations from Step 4 cost? Are they different from their corresponding implementations from Step 3? Are the different from each other? If so, by how many transistors?
7. How long are the critical paths from your `NAND`/`NOR` implementations from Step 4? Are the critical paths from the `NAND` and `NOR` implementations different? If so, by how much?
8. Which method (sum-of-product or product-of-sum) resulted in a more efficient hardware implementation from the perspective of complexity/power (number of transistors) and speed (critical path)? Why?
9. If you converted the `NAND` or `NOR` implementations to VHDL, would the resulting hardware use more or less resources (lookup tables) than the hardware from Lab 06C? Why?
    - Hint: remember that the lookup tables on the FPGA have 6 inputs and 1 output.

Deliverables

- Include as **an appendix** to your **formal report**:
    - Pictures of drawn gate implementations (Step 3)
    - Answers to handout questions (Steps 5, 6, 7, 8, and 9)
- This part alone can be in the format of an informal report.

Outcomes

- Understand how to derive boolean equations from a truth table using Karnaugh Maps.
- Understand how to transform boolean equations to a "Sum of Product" implementation using NAND gates.
- Understand how to transform boolean equations to a "Product of Sum" implementation using NOR gates.