University of Washington – Computer Science & Engineering
Spring 2026 Instructor: Naomi Alterman 2026-04-28

CSE 369 QUIZ 1

Name: _____

Student ID Number: _____

Please do not turn the page until 2:30

Instructions

- This quiz contains 6 pages, including this cover page. You may use the blank page at the back for scratch work, but clearly label which problems your work corresponds to.
- Please clearly indicate (box or circle) your final answer.
- The quiz is closed book and closed notes.
- Please silence and put away all cell phones and other mobile or noise-making devices.
- Remove all hats, headphones, smart glasses, watches, and other digital wearables.
- You have 20 (+5) minutes to complete this quiz.

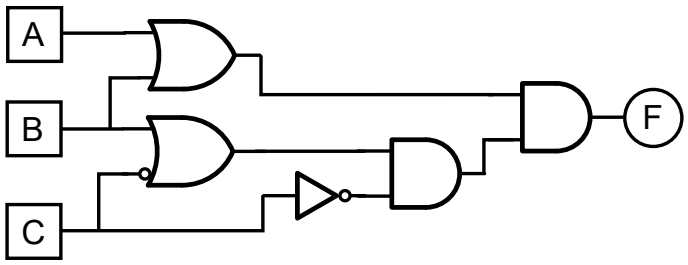
Advice

- Read questions carefully before starting. Read *all* questions first and start where you feel the most confident to maximize the use of your time.
- There may be partial credit for incomplete answers; please show your work.
- Relax. If you've been practicing, you got this. If you haven't, you'll learn something now.

Question	Points	Score
(1) CL Gates	12	
(2) K-map	6	
(3) Block Diagrams & Verilog	14	
Total:	32	

Question 1: Combinational Logic Gates [12 pts]

(A) Write out a Boolean expression for the circuit diagram below. **Do not simplify.** Please use the following notation: + (OR), . (AND), ¯ (NOT), and any parentheses to make your answer unambiguous. [6 pts]

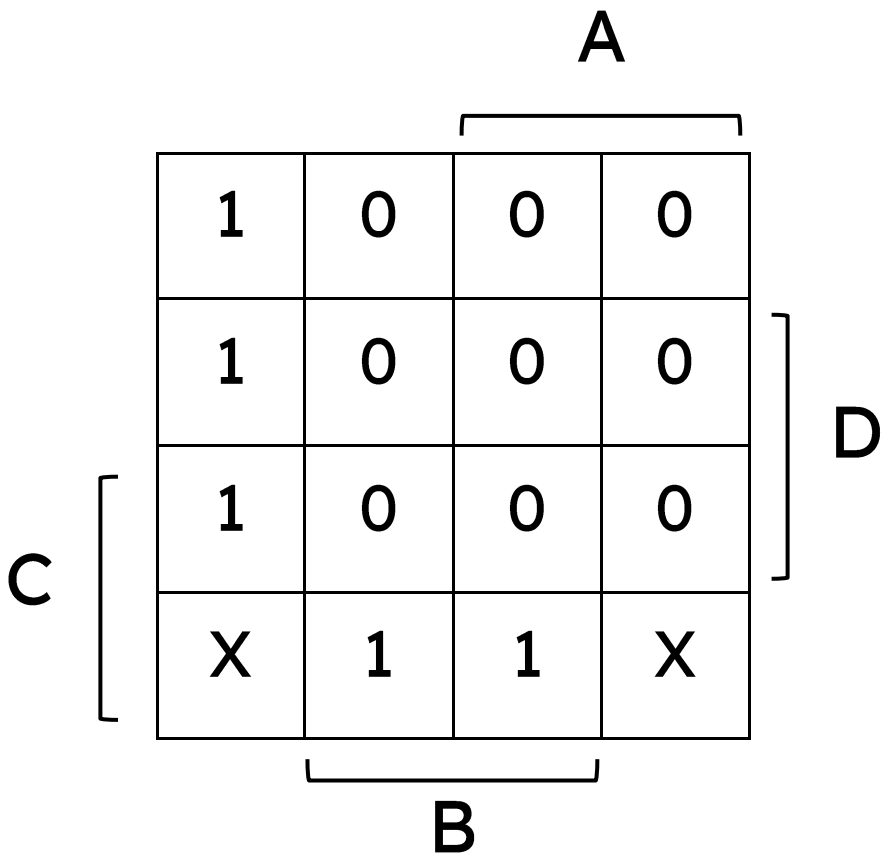


(B) Find a minimal implementation of the function below using only **2-input NAND gates**. *We will only accept circuit diagrams.* [6 pts]

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

Question 2: Karnaugh Maps [6 pts]

Find a ***minimum two-level sum-of-products solution*** for the K-map shown below. Show your work by circling groups on the map and writing out the resulting Boolean expression.



Question 3: Block Diagrams & Verilog [14 pts]

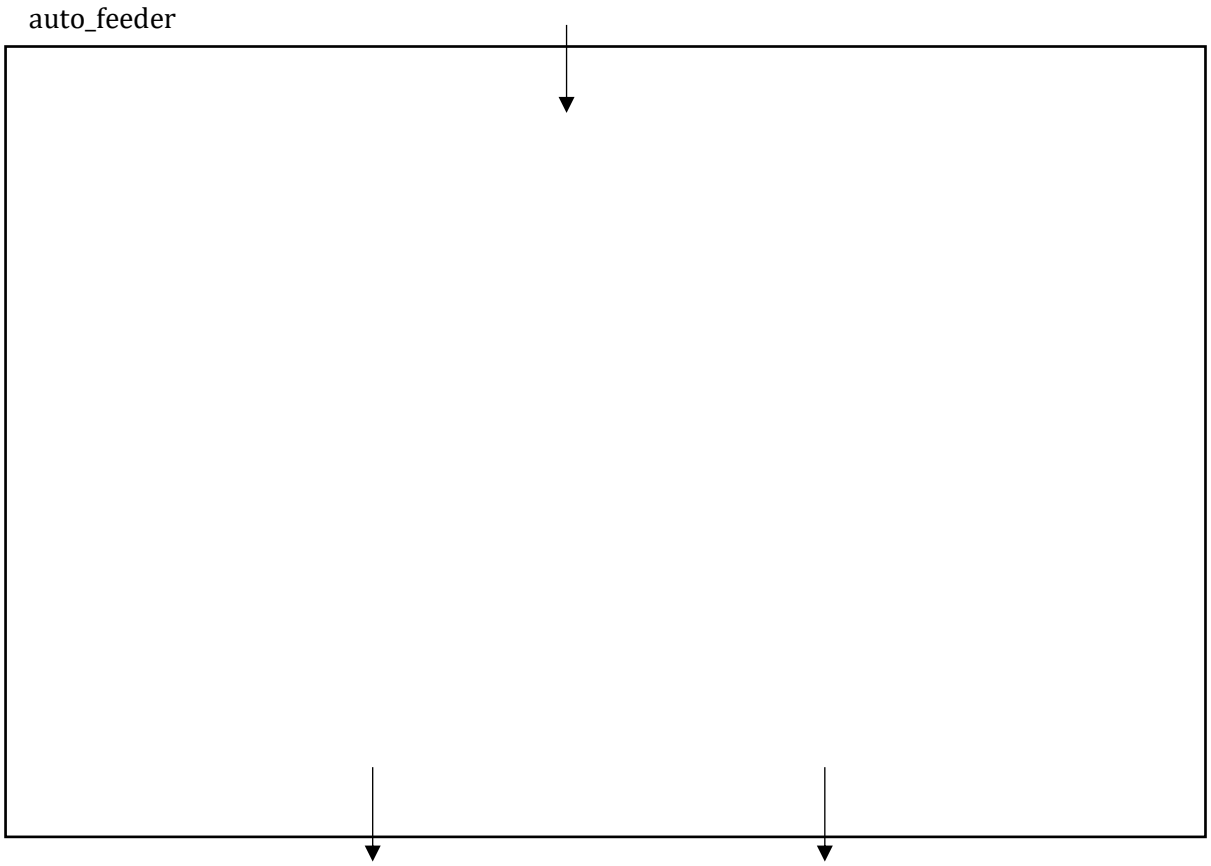
(A) We're building an **auto_feeder** for Danni the Cat. It's made of purely combinational logic and has these ports:

- **time**: Input, 8 bits, the current time of day
- **bell**: Output, 1 bit, rings a bell when high to announce a meal
- **amount**: Output, 4 bits, how much food to dispense

Internally, it's built from two sub-modules:

- **meal_time**: Detects whether the time input represents a meal time
  Ports:
  - **time**: Input, 8 bits, represents current time of day
  - **is_meal:** Output, 1 bit, high if meal time
  - **type:** Output, 2 bits, reports type of meal to be breakfast (2'b00), lunch (2'b01) or dinner (2'b10). Type is "don't care" if not is_meal is 0.
- **portioner**: Takes a type of meal and outputs how much kibble to dispense
  Ports:
  - **type**: Input, 2 bits, meal type to give portion for
  - **amount:** Output, 4 bits, how much food to dispense in grams

Draw a block diagram for the auto feeder below. Make sure to clearly specify all bus widths, port names, and signal directionality (eg, input/output) [8 pts]:



(B) The design above uses two bits to represent the meal types “breakfast”, “lunch” and “dinner”. How many bits would we need for the signal if we could *also* specify a meal type called “special snack” ? Why? [2 pts]

(C) Fill in the blanks to complete the Verilog code that implements the **portioner** module [4 pts]:

```
module portioner (type, amount);

   _____________ logic [_______] type;

   _____________ logic [_______] amount;

   always_comb begin
      case (type)
         2’d0: amount = 4’d12;
         2’d1: amount = 4’d6;
         2’d2: amount = 4’d10;
         default: amount = 4’d0;
      endcase
   end

endmodule
```

This page reserved for scratch work