Lab Report-ChipChat Colab (GenAI Workflow & RTL Examples)

Course/Lab: Lab 1-ChipChat Colab
Student: D. Akhash Krishna
NetID: AD7252
Date: February 7th, 2026
Tools: Icarus Verilog (iverilog), Google Colab, ChipChat (GenAI), OpenRouter API

1) Tools and Settings:

LLM model(s) & version(s): `arcee-ai/trinity-large-preview:free`
Interface: Google Colab (Python SDK via `openai` client with
`base_url="https://openrouter.ai/api/v1"`)
Key parameters controlled:
- `max_tokens=129000
- `stream = False`
- `messages = [{"role":"user","content": verilog_generation_prompt}]`
- Temperature / top_p / system prompt: defaults (not explicitly set)

Simulation: Icarus Verilog for compile/sim
Versioning & Files: prompts and iterations logged inline per example; VCDs saved per testbench when applicable.

2) Reproducible GenAI Workflow

1. Initial prompt: Describe the RTL spec (I/O, resets, timing/behavior, parameters).
2. Generate code (Iteration N): Ask LLM for Verilog/SystemVerilog module that matches the
testbench's port names.

3. Simulate: `iverilog -g2012 -o sim.out design.v tb.v && vvp sim.out`
4. Collect feedback: Copy exact compile/run errors.
5. Refine prompt: Paste Tool Feedback (TF) and Simple Human Feedback (SHF).
6. Repeat until pass.
7. Record results.

PART 1

1) Example 1-Binary to BCD Converter

Specification:

Inputs: 5-bit binary

Outputs: 8-bit BCD

Behavior: Convert 0-31 into packed BCD.

Final Prompt:

'''
I am trying to create a Verilog model binary_to_bcd_converter for a binary to binary-coded-decimal converter. It must meet the following specifications:

- Inputs:
  - Binary input (5-bits)
- Outputs:
  - BCD (8-bits: 4-bits for the 10's place and 4-bits for the 1's place)

How would I write a design that meets these specifications?

```
binary_to_bcd_tb.v:8: error: Unknown module type: binary_to_bcd_converter
2 error(s) during elaboration.
*** These modules were missing:
      binary_to_bcd_converter referenced 1 times.
***

binary_to_bcd_tb.v:8: error: port ``binary_input'' is not a port of uut.
binary_to_bcd_tb.v:8: error: port ``bcd_output'' is not a port of uut.
2 error(s) during elaboration.
```

Testing Binary-to-BCD Converter...
VCD info: dumpfile my_design.vcd opened for output.
Error: Test case 10 failed. Expected BCD: 8'b10000, Got: 8'b100000
""

Final Module Interface:

```
module binary_to_bcd_converter (
    input [4:0] binary_input,
    output [7:0] bcd_output
);
```

Design Approach:

I approached the binary-to-BCD converter by first making sure the module interface exactly matched what the testbench expected, since port-name and width mismatches are the most common cause of early failures. I defined the module and signals with the exact names and sizes used by the testbench before implementing any logic. Once the interface was aligned, I implemented the conversion as simple combinational logic that translates the 5-bit binary input (0–31) into packed BCD by separating the value into tens and ones digits and concatenating them into an 8-bit output. By prioritizing interface correctness, I ensured the design passed all test cases once the functionality was added.

2) **Example 5: Dice Roller**

Specification

Inputs: clk, reset_n, die_select[1:0], roll

Output: rolled_number (width up to 8 bits)

Behavior: On rising roll, output a value uniform over:

- 00→ d4 (1–4)
- 01→ d6 (1–6)
- 10→ d8 (1–8)

o 11→ d20 (1–20)

Final Prompt:

‘’’ I am trying to create a Verilog model for a simulated dice roller. It must meet the following specifications:

- Inputs:
  - Clock
  - Active-low reset
  - Die select (2-bits)
  - Roll
- Outputs:
  - Rolled number (up to 8-bits)

The design should simulate rolling either a 4-sided, 6-sided, 8-sided, or 20-sided die, based on the input die select. It should roll when the roll input goes high and output the random number based on the number of sides of the selected die.

How would I write a design that meets these specifications?
Do not give me the Testbench give me the verilog code only.
Correct the following error : error: port ``rst_n'' is not a port of dut.
1 error(s) during elaboration.
Error: Invalid roll result for 4-sided die: 0
Results for die_select 00:
Rolled 1: 298 times
Rolled 2: 234 times
Rolled 3: 234 times
Rolled 4: 233 times
Results for die_select 01:
Rolled 1: 150 times
Rolled 2: 156 times
Rolled 3: 178 times
Rolled 4: 174 times
Rolled 5: 202 times
Rolled 6: 140 times
Results for die_select 10:
Rolled 1: 132 times
Rolled 2: 112 times
Rolled 3: 120 times
Rolled 4: 74 times
Rolled 5: 166 times
Rolled 6: 114 times
Rolled 7: 116 times
Rolled 8: 166 times

Results for die_select 11:
Rolled 1: 116 times
Rolled 2: 48 times
Rolled 3: 60 times
Rolled 4: 24 times
Rolled 5: 36 times
Rolled 6: 60 times
Rolled 7: 46 times
Rolled 8: 60 times
Rolled 9: 12 times
Rolled 10: 34 times
Rolled 11: 24 times
Rolled 12: 70 times
Rolled 13: 58 times
Rolled 14: 34 times
Rolled 15: 22 times
Rolled 16: 48 times
Rolled 17: 70 times
Rolled 18: 60 times
Rolled 19: 84 times
Rolled 20: 34 times
Testbench completed with 1 errors.””

Final Module Interface:

```
module dice_roller (
    input clk,
    input rst_n,
    input [1:0] die_select,
    input roll,
    output reg [7:0] rolled_number
);
```

Design Approach:

I approached the dice roller by first matching the module interface exactly to the testbench, especially fixing the reset port name (rst_n) to resolve the elaboration error. Once the interface was correct, I focused on functional behavior by detecting the rising

edge of roll so a value is generated only once per roll. I then ensured the generated number is always constrained to a valid range (1 to 4, 6, 8, or 20 based on die_select) to prevent invalid results like rolling a 0, while maintaining a roughly uniform distribution.

PART 2

Iteration 1

(a) What failed: Elaboration failed because the testbench could not find the DUT module.
Command used:

iverilog -g2012 -o binary_to_bcd.vvp binary_to_bcd.v binary_to_bcd_tb.v && vvp binary_to_bcd.vvp

Relevant output snippet:

```
binary_to_bcd_tb.v:8: error: Unknown module type: binary_to_bcd_converter
2 error(s) during elaboration.
*** These modules were missing:
        binary_to_bcd_converter referenced 1 times.
***
```

(b) What I changed: I ensured my Verilog file defined a module named exactly binary_to_bcd_converter.
(c) Why I expected it to help: The testbench instantiates binary_to_bcd_converter; if the module name doesn't match exactly, iverilog can't elaborate the design.

Iteration 2

(a) What failed: Elaboration still failed due to port name mismatches between DUT and testbench.
Command used:

iverilog -g2012 -o binary_to_bcd.vvp binary_to_bcd.v binary_to_bcd_tb.v && vvp binary_to_bcd.vvp

Relevant output snippet:

binary_to_bcd_tb.v:8: error: port ``binary_input'' is not a port of uut.
binary_to_bcd_tb.v:8: error: port ``bcd_output'' is not a port of uut.
2 error(s) during elaboration.

(b) What I changed: I renamed the DUT ports to match the testbench exactly: binary_input (5-bit) and bcd_output (8-bit).
(c) Why I expected it to help: Even if the logic is correct, the testbench connects by port names; mismatched names prevent compilation/elaboration.

Iteration 3

(a) What failed: The design compiled and ran, but one or more functional test cases failed due to incorrect BCD formatting (bit placement/packing).
Command used:

iverilog -g2012 -o binary_to_bcd.vvp binary_to_bcd.v binary_to_bcd_tb.v && vvp binary_to_bcd.vvp

Relevant output snippet:

Testing Binary-to-BCD Converter...
VCD info: dumpfile my_design.vcd opened for output.
Error: Test case 10 failed. Expected BCD: 8'b10000, Got: 8'b100000

(b) What I changed: I corrected the conversion logic to produce packed BCD as {tens[3:0], ones[3:0]} and verified that for input 10 the output is 8'b0001_0000 (tens=1, ones=0).
(c) Why I expected it to help: The failure indicated a digit/bit-shift packing problem—i.e., the tens/ones digits weren’t landing in the expected nibble positions—so explicitly forming {tens, ones} fixes the formatting.

Final passing run

Final command:

iverilog -g2012 -o binary_to_bcd.vvp binary_to_bcd.v binary_to_bcd_tb.v && vvp binary_to_bcd.vvp

Final output snippet (pass):

Testing Binary-to-BCD Converter...
All test cases passed.

PART 3

Extension Option - Binary to BCD:

I turned the converter into a register with a write-enable in front of the BCD output. Concretely, I added two new inputs: a clock signal (clk) and a write-enable signal (enable) that controls when the output is allowed to update. I kept the conversion math fully combinational internally, still computing ones = binary_input % 10 and tens = binary_input / 10, and forming {tens, ones} as the next BCD value. I then moved bcd_output into a sequential block so that, instead of changing whenever binary_input changes, it only updates on a rising clock edge when enable == 1. When enable == 0, the stored BCD value holds its previous state even if the input toggles.