Tutorial examples:

- A: Binary to BCD converter and
- B: Sequence detector

Extension:

- Sequence detector, spec change

Example 1: Binary to BCD Converter

- Model: GPT-4o-mini
- Parameters:
  (a) Max tokens: Default (as set by ChatGPT)
  (b) Temperature: Default

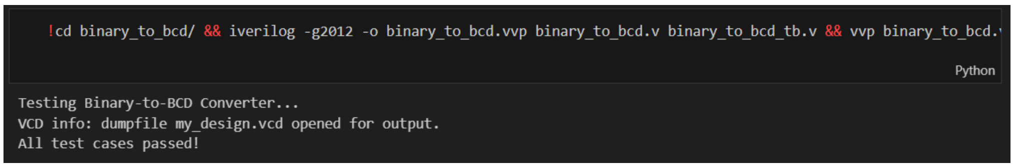
Final Prompt/Interface/Explanation

- I want to design a Verilog model of a binary to bcd converter.
  Model name: binary_to_bcd_converter
  specifications:
  Input: binary_input, must be 5 bits wide.
  Output: bcd_output, must be 8 bits wide

  Logic: The module should convert the 5-bit binary value (0-31) into two separate BCD digits. For example, if the input is 5'b11010 (26), the output should be 8'b0010_0110 (2 and 6)

- Explanation: This prompt was designed to clearly specify the functional requirements and interface of a binary-to-BCD converter in Verilog. The model name and exact input/output bit widths were explicitly defined to avoid ambiguity in module generation and ensure compatibility with the testbench. By constraining the input to a 5-bit binary value (0–31) and the output to an 8-bit BCD representation, the prompt enforces correct decimal digit separation into tens and ones. A concrete example (binary input 26 mapping to BCD output 2 and 6) was included to guide the model toward correct logic implementation and output formatting. Overall, the prompt emphasizes deterministic behavior, clear interface definition, and correctness over optimization.

- **Final iverilog Commands**



```
!cd binary_to_bcd/ && iverilog -g2012 -o binary_to_bcd.vvp binary_to_bcd.v binary_to_bcd_tb.v && vvp binary_to_bcd.
```

```
Testing Binary-to-BCD Converter...
VCD info: dumpfile my_design.vcd opened for output.
All test cases passed!
```

Example 2: Sequence Detector

- Model: GPT-4o-mini
- Parameters:
  (a) Max tokens: Default (as set by ChatGPT)
  (b) Temperature: Default

Final Prompt/Interface/Explanation

- Please design a Verilog module for a sequence detector that meets the following specifications:

  Interface:

  Inputs: clk (pos-edge), reset_n (active-low, synchronous), and data (3-bit wide: [2:0] data).

  Output: sequence_found (1-bit, asserted high for one clock cycle only when the full sequence is matched).

  Functional Logic: The module must detect a specific sequence of eight 3-bit patterns in this exact order:

  3'b001, 3'b101, 3'b110, 3'b000, 3'b110, 3'b110, 3'b011, 3'b101

  Requirements:

  Use a Finite State Machine (FSM) approach.

  Ensure the testbench feeds the data as 3-bit vectors per clock cycle to match the input width.

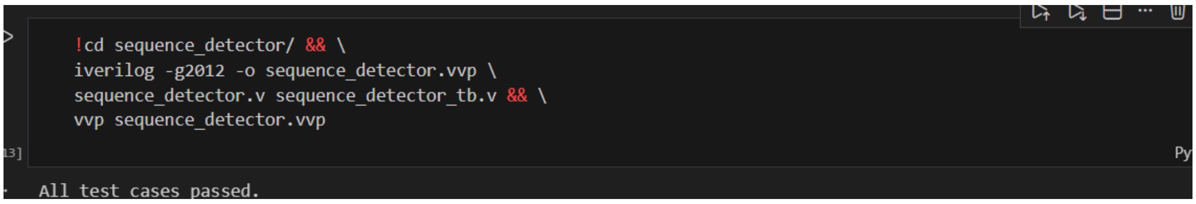
  Compatible with iverilog (SystemVerilog-2012)

- Explanation: This prompt was constructed to precisely define the behavior and interface of a sequence detector implemented using a finite state machine (FSM). The clocking scheme and reset behavior were explicitly specified, including a positive-edge-triggered clock and an active-low synchronous reset, to ensure deterministic state transitions compatible with the provided testbench. The input data width was constrained to 3 bits per clock cycle to match the expected stimulus format, preventing mismatches between the module and the testbench. By explicitly listing the exact eight 3-bit patterns and enforcing their order, the prompt guides the model toward a well-structured FSM with clearly defined states and transitions. Additionally, restricting the output signal to assert high for only one clock cycle ensures correct pulse-style detection behavior. Overall, the prompt emphasizes interface consistency, FSM-based design clarity, and simulation compatibility with Icarus Verilog (SystemVerilog-2012).

Iteration:

Numbers	1	2	3
Issue Observed	Compilation failed during elaboration due to port mismatches	Width mismatch： testbench drives a multi-bit `data` signal while DUT expects 1-bit input. Functional mismatch: at cycle 8, expected sequence_found = 1, but DUT output was 0.	Timing mismatch: cycle 8 should be 1 but the output is 0
Fixed applied	Refine the prompt to explicitly match the testbench port names and reset polarity	Refine the prompt to explicitly specify the width and semantics of the `data` input. #- Clarify the exact sequence to detect and when `sequence_found` should be asserted.	Adding a specific sequence of eight 3-bit patterns and finishing it in this exact order
Outcome	Port mismatching solved	Width mismatching solved	All test case pass

Final iverilog commands:



```
!cd sequence_detector/ && \
iverilog -g2012 -o sequence_detector.vvp \
sequence_detector.v sequence_detector_tb.v && \
vvp sequence_detector.vvp
```

```
All test cases passed.
```

Prompt-Engineering Extension

- Implementation: Add an enable signal to the prompt. The detector should only operate when en=1. If en=0, the FSM must remain in its current state and should not transition, even if the correct sequence is input.
- Prompt Decisions: I instructed the AI to "Add a 1-bit input named en. The FSM should only transition states when en is high."
- Impact: This modification changes the generated RTL logic by adding a conditional gate to every state transition. In the sequential logic block, the state update is now wrapped in an if (en) statement, ensuring the FSM "freezes" its current state whenever the enable signal is de-asserted.