Name: Naveen Kumar Senthil Kumar

NetID: ns6503

Lab 4 - Hierarchical Prompting for RTL Generation

MUX Hierarchy Example

This report applies the Lab 4 **Hierarchical Prompting** template to a custom Python/Colab notebook that generates a **multiplexer (MUX) hierarchy** in Verilog using an automated feedback loop. The notebook defines a staged build (mux2to1, mux4to1, mux8to1) and invokes a language model (model_choice =deepseek-ai/deepseek-r1`) to synthesize RTL, compile with Icarus Verilog, and iterate on errors until testbenches pass. We document the workflow, submodule interfaces, and practical debugging tactics and compare hierarchical prompting to flat prompting for this specific design.

2. Background & Goals

- **Flat prompting (baseline):** Ask for a complete top design in one shot and compile/simulate.
- **Hierarchical prompting (this lab):** Generate self-contained leaf modules first, then integrate into a top-level. Use compiler/simulator feedback to drive fixes.

Goals: 1) Build a parameterizable MUX hierarchy with staged generation and testing.
2) Exercise one or more modern LLMs for code generation.
3) Compare iteration patterns with flat prompting and record practical lessons.

3. Experimental Setup

3.1 Toolchain

- **Simulation:** Icarus Verilog (iverilog + vvp), invoked from Python.
- **Environment:** Google Colab-style notebook.
- **Pass criterion:** Testbench prints a success token (e.g., passed!).

3.2 Models

- **Primary selection in notebook:** deepseek-ai/deepseek-r1.
- **APIs imported:** OpenAI, Anthropic, and Google GenAI (not all need be used at once).
- **Note:** Actual model used depends on configured keys in the runtime.

3.3 Prompt Style

Each stage starts with a strict "code-only" instruction followed by a module-specific spec. Errors and warnings from iverilog/vvp are appended in subsequent turns to steer fixes. Non-code prose is discouraged to keep iterations clean and diff-friendly.

4. MUX Design Specification

We implement a three-level MUX hierarchy as independent modules validated by dedicated testbenches.

#@title Submodules

Each step is structured as [“filename”,“natural language description”]

submodules = [[“mux2to1”,“2-to-1 multiplexer”,“input wire in1, input wire in2, input wire select, output wire out”], [“mux4to1”,“4-to-1 multiplexer”,“input [1:0] sel, input [3:0] in, output reg out”], [“mux8to1”,“8-to-1 multiplexer”,“input [2:0] sel, input [7:0] in, output reg out”],]

5. Hierarchical Prompting Plan

1) Generate and validate each leaf (e.g., 2:1, then 4:1) with a focused testbench.

2) Integrate into the higher-level (e.g., 8:1) using proven leaves.

3) Keep interfaces consistent across iterations to avoid port drift.

6. Workflow

Step 1 Write testbenches. FILENAMEtb.v files per module that exhaustively (or near-exhaustively) cover inputs and select lines.

Step 2 Define submodules. Provide a structured list submodules = [[name, description, io_ports], ...] for the generation loop.

Step 3 Run the hierarchical loop. The script iterates: generate → compile (iverilog) → simulate (vvp) → parse errors → refine until the success token appears or the iteration cap is reached.

Step 4 Hardening. Constrain outputs to pure Verilog-2001 as needed; surface tool logs verbatim; keep a single source of truth for interfaces.

7. Results

- Leaf MUXes typically converge quickly (small interface surfaces, simple truth tables).
- The top-level MUX converges once lower leaves are discoverable on the compiler path and port names are aligned.
- Common first-pass issues: port name mismatches, reg/wire mix-ups in combinational blocks, and unused signals.

8. Comparison: Hierarchical vs. Flat Prompting

- **Flat prompting:** Often fastest for simple blocks (e.g., a single 8:1 MUX with a tight spec).
- **Hierarchical prompting:** Better for pedagogy, reuse, and isolating failures (each stage has its own TB and logs).

For compound designs (ALUs, protocol stacks), hierarchical prompting tends to scale with less thrash in late stages.

9. Lessons Learned & Debugging Playbook

1) **Compiler modes:** If you see SV constructs flagged as errors, either enable -g2012 or remove SystemVerilog from the output.
2) **Module discovery:** Ensure previously generated leaf RTL is on the include path when compiling the next stage.
3) **Port mismatches:** Keep a canonical interface string and auto-check TB vs. RTL ports.
4) **Prose trimming:** Enforce "first module … endmodule" extraction to strip commentary.
5) **Surface logs:** Print iverilog/vvp stderr/stdout directly to help the prompting loop target the right fix.

10. Inference

Using hierarchical prompting, the notebook reliably synthesizes and validates a MUX hierarchy in Verilog. While a flat prompt can be quicker for a single module, the staged approach provides cleaner boundaries, reusable leaves, and more systematic debugging, useful properties as designs grow beyond toy examples.