HDL Design & Simulation of Hardware Accelerator

In previous challenges (#12, #13, #14), I analyzed my Viterbi Decoder Accelerator and explored HW/SW boundaries.
I had already identified the bottleneck portion (max + add operations) for hardware acceleration.
I also used LLMs earlier to help build initial RTL models (PyRTL, Python models).
Now the goal was to generate actual synthesizable Verilog/SystemVerilog code

1. HDL Description: Verilog Accelerator Model

- I decided to write Verilog RTL for my hardware accelerator block that performs:
  - Compute the maximum over states
  - Add emission log
  - Store argmax index (psi_out)
- I heavily leveraged LLMs to convert my PyRTL and high-level model to Verilog.

LLM prompt I used:

> Convert my PyRTL model for Viterbi delta update into synthesizable Verilog. The function computes max(delta_prev[i] + logA[i][j]) for i=0,1,2, and outputs best_val and best_idx.

The generated Verilog code became my viterbi_pe.v module.
I refined this code manually with LLM assistance to ensure synthesizability and correctness.

2. Verilog Testbench

I also generated a testbench with LLM help:

LLM prompt:

> Generate a Verilog testbench for my viterbi_pe module that drives sample inputs and checks the delta_out and psi_out outputs.

I reviewed multiple LLM versions and finally created a simplified functional testbench which applies sample inputs and observes the outputs.
This was my tb_viterbi_pe.v file.

3. Simulator Selection & Setup

I used Icarus Verilog (iverilog) for simulation:

- sudo apt install iverilog

I compiled my design using:

- iverilog -g2012 -o sim_viterbi.vvp viterbi_pe.v tb_viterbi_pe.v

I simulated using:

- vvp sim_viterbi.vvp

I debugged signal activity using $display() and observed correct output waveform behavior.
Later, I also generated waveform traces (.vcd) for deeper debugging with:

- $dumpfile("wave.vcd");
- $dumpvars(0, tb_viterbi_pe);

I viewed waveforms with:

- gtkwave wave.vcd

LLM worked very well for me with:

- Code translation from Python logic to Verilog
- Testbench generation
- Flattening multidimensional inputs for hardware
- Correcting synthesis errors (non-synthesizable constructs)
- Iterative prompting allowed me to refine HDL code gradually.

After Challenge #15, I now feel fully capable of building RTL designs from high-level algorithms. This challenge connected software analysis, hardware datapath design, HDL coding, full simulation.