HW/SW co-design with design iterations

As part of Challenge #21 of ECE 510, I performed multiple hardware-software co-design iterations while working on the Viterbi Decoder Accelerator project. This design iteration helped me refine both my hardware design and my understanding of the algorithm.

Steps:

1. Profiling and Bottleneck Analysis
    a. I revisited my algorithm and performed further workload analysis and profiling.
    b. From earlier challenges (Challenge #9, #10), I learned that the Viterbi Forward Path (recursive dynamic programming step) is the computational bottleneck.
    c. The most intensive operation was repeatedly performing max + add across multiple states, which motivated me to design hardware that could accelerate this key operation.

2. Revising Data Structures for Hardware
    a. A key step was converting multi-dimensional (2D) arrays to flattened 1D arrays to simplify hardware design.
    b. This made synthesis easier since nested arrays create complexity for synthesis tools.
    c. For example, matrices like logA[3][3] and logB[3][3] were converted into flat 9-element arrays.

3. Synthesis-Driven Code Changes
    a. I made several modifications to ensure the design was fully synthesizable:
        i. Removed nested loops from combinational always blocks.
        ii. Avoided dynamic indexing and multi-dimensional memories in favor of static, flattened registers and wires.
        iii. Replaced behavioral-style indexing with case-statements or direct bit slicing.
    b. These changes allowed smooth synthesis and simulation using tools like Icarus Verilog and OpenLane2.
    c. Simplified FSM control logic for forward pass, backward path tracing, and initialization steps.

4. Microarchitecture Refinement
    a. I implemented pipelined versions of the Viterbi Processing Element (viterbi_pe) that computes max + add efficiently for each state.
    b. The control FSM was refined to handle both forward and backward passes sequentially.
    c. Deliberately separated state register updates to avoid combinational loops and improve timing closure.

5. Co-Design Principles Applied
    a. I applied multiple co-design ideas:

   i. Hardware-aware algorithm changes: Simplified state transitions for easier hardware mapping.
   ii. Progressive testbenching: First tested isolated modules (viterbi_pe), then moved to full system simulation (viterbi_top).
   iii. Golden model comparisons: Compared hardware outputs to software reference models for functional correctness.
   iv. Hardware-Software boundary: The datapath acceleration (forward + backward logic) was moved to hardware while the input pre-processing and control remain in software.

6. Verification and Iteration
   a. Iterative testing helped catch subtle bugs in:
      i. Path flattening
      ii. Psi backtracking logic
      iii. Edge conditions when reaching the final state
   b. I used cProfile, waveform viewers, and full functional simulations to validate design correctness.
   c. Multiple synthesis and simulation runs helped verify synthesizability and correct signal connections.

7. Physical Design Preparations
   a. My design is now fully synthesizable, suitable for tools like:
      i. OpenLane2 for ASIC synthesis
      ii. Yosys & NextPNR for FPGA flow (optional next steps)
      iii. Signal flattening and memory simplifications made place-and-route easier for physical design.

Through this design iteration:

- I translated my algorithm into hardware-optimized logic.
- Learned how to make my design synthesizable step-by-step.
- Resolved issues around array flattening, state machine design, and pipelining.
- Verified my design through progressive simulation and debugging.

This iterative process helped me solidify my understanding of both hardware design and hardware-software co-design principles, fully aligned with the learning goals of this course.