Throughout previous challenges, I was working on the Viterbi Decoder Acceleration project, and also explored earlier FrozenLake RL Q-learning bottleneck.

Now, in this challenge, I applied full co-design thinking to my Viterbi project and practiced HW prototyping on both Viterbi and FrozenLake workloads.

Profiling

Using cProfile, py-spy, line_profiler, I identified major computational bottlenecks in my Viterbi and RL workloads.

For Viterbi:

- The main bottleneck was repeatedly searching for max score (argmax across states).
- Secondary was multiply-accumulate in the score update formula.

For FrozenLake:

- The core bottleneck was in Q-value update:

```
Q_new = (1 - α) * Q_old + α * (R + γ * max Q_next)
```

Which Part To Accelerate?

Viterbi:

- The max-finding and path update across the trellis was chosen.
- Forward path was especially heavy for long sequences.

FrozenLake:

- The Q-value update formula (multiply-add + max()) was ideal for hardware.

HW acceleration is likely worth it because T5 + T6 + T7 << T3. Especially for workloads with many iterations.

Tools Used:

Tool Selection

I explored multiple HW design frameworks suggested in the challenge:

Tool	Decision
PyMTL3	Evaluated, but not picked (complex setup)
PyRTL	✅ **Selected** (very intuitive for my Python-based flow)
MyHDL	Explored, but limited adoption
cocotb	✅ Used earlier for testbench verification

yosys	Not needed at this stage

PyRTL Design Flow:

1. Build multiplier and adder blocks in PyRTL
2. Simulate the datapath with sample inputs
3. Exported PyRTL to Verilog for potential FPGA mapping
4. Validated correctness with Python reference outputs

LLM Prompt which helped me get the output, which I then cleaned up.:

"Write me a PyRTL code to model a Q-learning update datapath: Q_new = (1 – α) * Q_old + α * (R + γ * Q_max)"

HW/SW Co-Design & Bottleneck Acceleration