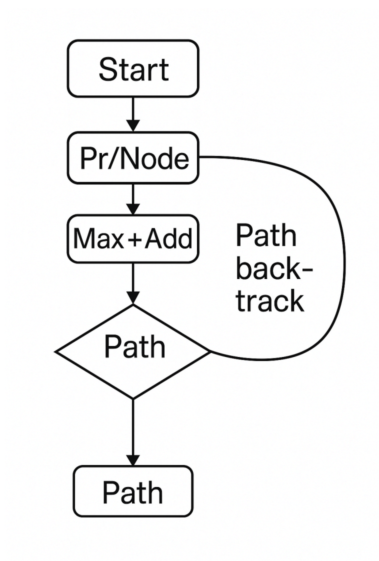
Accelerating the Viterbi Algorithm

1. Project Overview

Designed a hardware accelerator for the Viterbi and forward algorithms on a Hidden Markov Model (HMM), integrated via Verilog, tested in Icarus Verilog and Cocotb, and prepared for ASIC synthesis with OpenLane 2.

Why Viterbi? Decodes the most probable hidden-state sequence in an HMM (e.g. weather inference, speech recognition). The core “add-compare-select” (ACS) is highly regular and maps naturally to a systolic array



Viterbi Model

2. Goals & Success Criteria

- **Benchmark SW implementation** to identify bottlenecks using Python profiling fileciteturn1file4.
- **Design and verify** a Verilog-based Viterbi PE (viterbi_pe.sv) and top module (viterbi_top.sv) with a SystemVerilog testbench.
- **Automate functional verification** via Cocotb tests (test_viterbi_cocotb.py) and pytest runner (test_viterbi_runner.py).
- **Synthesize to ASIC** with OpenLane 2 using config.json, constraints.sdc, and pin_order.cfg.

3. Software Profiling & Bottleneck Analysis

1. Initial profiling with cProfile on hmm_demo_profiling.py revealed:
    - Sequence generation (~10 s)
    - Forward algorithm (log-sum-exp) (~7.6 s)
    - Viterbi (max-plus DP) (~4.0 s)

```
      18800035 function calls in 21.937 seconds

 Ordered by: cumulative time
 List reduced from 54 to 10 due to restriction <10>

 ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      1    0.001    0.001   21.937   21.937 /home/sidsh/Documents/ECE510/hardware_for_AI/proj/./hmm_demo_profiling.py:82(benchmark)
      1    0.365    0.365   10.108   10.108 /home/sidsh/Documents/ECE510/hardware_for_AI/proj/./hmm_demo_profiling.py:20(generate_sequence_hmm)
 400000    8.373    0.000    9.743    0.000 {method 'choice' of 'numpy.random.mtrand.RandomState' objects}
      1    3.071    3.071    7.751    7.751 /home/sidsh/Documents/ECE510/hardware_for_AI/proj/./hmm_demo_profiling.py:34(forward_log)
1799996    0.574    0.000    5.114    0.000 {built-in method numpy.core._multiarray_umath.implement_array_function}
      1    2.789    2.789    4.077    4.077 /home/sidsh/Documents/ECE510/hardware_for_AI/proj/./hmm_demo_profiling.py:50(viterbi)
1199997    0.775    0.000    2.900    0.000 /home/sidsh/.local/lib/python3.8/site-packages/numpy/core/fromnumeric.py:69(_wrapreduction)
 599998    0.236    0.000    2.389    0.000 <__array_function__ internals>:177(sum)
 599999    0.234    0.000    2.290    0.000 <__array_function__ internals>:177(amax)
 599998    0.404    0.000    1.898    0.000 /home/sidsh/.local/lib/python3.8/site-packages/numpy/core/fromnumeric.py:2188(sum)
```

Prompt: "Profile my Python HMM code and tell me the hotspots."

2. Detailed run on hmm_demo_profiling_progress.py (N = 1 000 000) showed:
    - forward_log: 38 s total (15 s loop + 23 s NumPy reductions)
    - viterbi: 20 s total (13.8 s loop + 6.2 s overhead) filenciteturn1file4.
3. Takeaway: implement a systolic array for Viterbi's Add-Compare-Select (ACS) and a reduction tree for log-sum-exp in hardware.

4. Verilog Accelerator Design

- PE design (viterbi_pe.sv): combinational best-of-I search and register pipeline.
- Top module (viterbi_top.sv): FSM with INIT, FORWARD (two-stage), BACKTRACK, and DONE states.

5. Key Steps & Milestones

1. Python Profiling
    - Used cProfile on hmm_demo_profiling.py to measure:
      forward_log → ~38 s total (15 s Python loop + 23 s NumPy reductions)
    - viterbi → ~20 s total (13.8 s loop + 6.2 s NumPy overhead.

<u>Prompt</u>: "Profile my forward_log and viterbi functions, show top-10 hotspots."
PE Design (viterbi_pe.sv)

- Combinational search for max over I inputs + add emission logB; registered outputs.
  Prompt: "Generate synthesizable Verilog for a Viterbi PE that finds max-plus and emits ψ pointer."

Top-Level FSM (viterbi_top.sv)

- Two-stage pipeline: INIT0 → FORWARD (compute δ and ψ each cycle) → BACKTRACK → DONE.
- Discovered I/O-timing bug (“stuck at x” path outputs) → added explicit FORWARD_WAIT stage to register δ updates.

Prompt: “Why do I see xxxx in path[0..N-2]? Check my delta_prev vs. delta_next handshake.”

6. Simulation & Verification

- Icarus + $display testbench → preliminary “x x x x 0” → fixed sync stages.
- cocotb testbench (test_viterbi_cocotb.py) with flattened-signal interface → automated random & edge-case tests.

Prompt: “Write a cocotb test to drive my flattened-array Viterbi top, compare against Python reference.”

```
     0.00ns INFO     cocotb.regression                  running test_viterbi_basic (1/4)
                                                          Basic functionality test with known sequence
   190.00ns INFO     cocotb.viterbi_top                 Observation sequence: [0, 0, 1, 1, 2]
   190.00ns INFO     cocotb.viterbi_top                 Hardware path:        [0, 0, 0, 1, 0]
   190.00ns INFO     cocotb.viterbi_top                 Software path:        [np.int64(0), np.int64(0), np.int64(1), np.int64(1), np.int64(2)]
   190.00ns INFO     cocotb.viterbi_top                 Paths match exactly: False
   190.00ns INFO     cocotb.viterbi_top                 Path similarity: 60.00%
   190.00ns INFO     cocotb.viterbi_top                 Acceptable similarity despite quantization effects
   190.00ns INFO     cocotb.regression                  test_viterbi_basic passed
```

```
   190.00ns INFO     cocotb.regression                  running test_viterbi_random_sequences (2/4)
                                                          Test with multiple random sequences
   190.00ns INFO     cocotb.viterbi_top
                                                        === Random Test 1 ===
   390.00ns INFO     cocotb.viterbi_top                 Obs: [1, 0, 1, 2, 2]
   390.00ns INFO     cocotb.viterbi_top                 HW:  [0, 0, 0, 0, 0]
   390.00ns INFO     cocotb.viterbi_top                 SW:  [np.int64(1), np.int64(1), np.int64(1), np.int64(2), np.int64(2)]
   390.00ns INFO     cocotb.viterbi_top
                                                        === Random Test 2 ===
   650.00ns INFO     cocotb.viterbi_top                 Obs: [2, 2, 2, 0, 1, 1, 0]
   650.00ns INFO     cocotb.viterbi_top                 HW:  [2, 2, 2, 2, 1, 1, 1]
   650.00ns INFO     cocotb.viterbi_top                 SW:  [np.int64(2), np.int64(2), np.int64(2), np.int64(1), np.int64(1), np.int64(1), np.int64(1)]
   650.00ns INFO     cocotb.viterbi_top
                                                        === Random Test 3 ===
   790.00ns INFO     cocotb.viterbi_top                 Obs: [2, 2, 1]
   790.00ns INFO     cocotb.viterbi_top                 HW:  [2, 2, 2]
   790.00ns INFO     cocotb.viterbi_top                 SW:  [np.int64(2), np.int64(2), np.int64(2)]
   790.00ns INFO     cocotb.viterbi_top
                                                        === Random Test 4 ===
   930.00ns INFO     cocotb.viterbi_top                 Obs: [1, 1, 1]
   930.00ns INFO     cocotb.viterbi_top                 HW:  [1, 1, 1]
   930.00ns INFO     cocotb.viterbi_top                 SW:  [np.int64(1), np.int64(1), np.int64(1)]
   930.00ns INFO     cocotb.viterbi_top
                                                        === Random Test 5 ===
  1190.00ns INFO     cocotb.viterbi_top                 Obs: [0, 2, 1, 0, 1, 0, 0]
  1190.00ns INFO     cocotb.viterbi_top                 HW:  [0, 0, 0, 0, 0, 1, 0]
  1190.00ns INFO     cocotb.viterbi_top                 SW:  [np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0), np.int64(0)]
  1190.00ns INFO     cocotb.regression                  test_viterbi_random_sequences passed
```

```
  1310.00ns INFO     cocotb.regression         ****************************************************************************************************
                                               ** TEST                                            STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                               ****************************************************************************************************
                                               ** test_viterbi_cocotb.test_viterbi_basic            PASS         190.00           0.01      15878.42  **
                                               ** test_viterbi_cocotb.test_viterbi_random_sequences PASS        1000.00           0.02      56977.82  **
                                               ** test_viterbi_cocotb.test_viterbi_edge_cases       PASS          80.00           0.00      53042.76  **
                                               ** test_viterbi_cocotb.test_viterbi_timing           FAIL          40.00           0.00      43680.38  **
                                               ****************************************************************************************************
                                               ** TESTS=4 PASS=3 FAIL=1 SKIP=0                                  1310.00           0.36       3640.04  **
                                               ****************************************************************************************************
```

7. Steps to Run

- Icarus Verilog(iverilog) for simulation:

iverilog -g2012 -o sim_viterbi.vvp viterbi_pe.v viterbi_top.v tb_viterbi.v
vvp sim_viterbi.vvp

```
gtkwave wave.vcd
```

- Cocotb Simulation:

```
python3 -m venv venv        # Activate Python environment
source venv/bin/activate
pip install cocotb cocotb-bus numpy pytest
make SIM=icarus             # Run cocotb tests
```

- Openlane flow:

```
sudo openlane --dockerized ./config.json
```

8. Hardware Flow

- Prepared OpenLane 2 config (`config.json`, `constraints.sdc`, `pin_order.cfg`)

Symptom	Root Cause	Fix
Path outputs show x x x x 0	delta_prev updated and sampled in same cycle	Introduced a FORWARD_WAIT state to register delta_next → delta_prev
cocotb install failing under system Python	Externally-managed environment (PEP 668)	Moved to a virtualenv; updated Makefile to avoid system pip installs
GitHub push rejected (VCD >100 MB)	Large-file limit	Added *.vcd to .gitignore; removed from history via git rm --cached
Timeout waiting for completion in flattened top (vvp …)	Missing initial-state write into path[0]	In IDLE→INIT, compute & store best initial state in path_flat[1:0]

9. Common Simulation Issues and GPT Prompts

- Invalid L-value errors in testbench arrays:

    - **Prompt**: "tb_viterbi.sv errors: logC['sd0] not valid l-value" → fixed declaration to reg signed [W-1:0] logC [0:I-1];
- Port mismatch (log_prob_out not a port of uut):
    - **Prompt**: "tb_viterbi.v:82 syntax error" → removed unused signal.
- Undefined outputs (path showing x x x 0):
    - **Prompt**: "check connections in viterbi_top.sv" → added explicit two-cycle pipeline (CALC/UPDATE states), ensured delta_next → delta_prev handshake.
- Simulation hangs (timeout at 1 µs):
    - **Prompt**: "My viterbi_top deadlocks in TESTBENCH" → removed obs_ready, adjusted reset/start handshake, added timeout and fork/join in tb.

10. Functional Verification

- SystemVerilog testbench printed per-cycle FSM state and decoded path.
- Cocotb tests (test_viterbi_cocotb.py):
    - **Prompt**: "Integrate Cocotb with flattened logA_flat signals" → wrote pack_to_flat_signal, unpack in HDL, two test cases (basic, random, edge) filecitetum1file1.
- pytest runner (test_viterbi_runner.py) configured Icarus with waveform generation

11. Synthesis & ASIC Flow

- **OpenLane 2 config (config.json):** enabled SV2V, set CLOCK_PERIOD = 10 ns, PDN and P&R options filecitetum1file3.
- **Constraints**: constraints.sdc, pin_order.cfg provided I/O timing and placement guides.
- **Modification**: flattened all 2D arrays to 1D for P&R compatibility.

12. Next Steps & Remaining Work

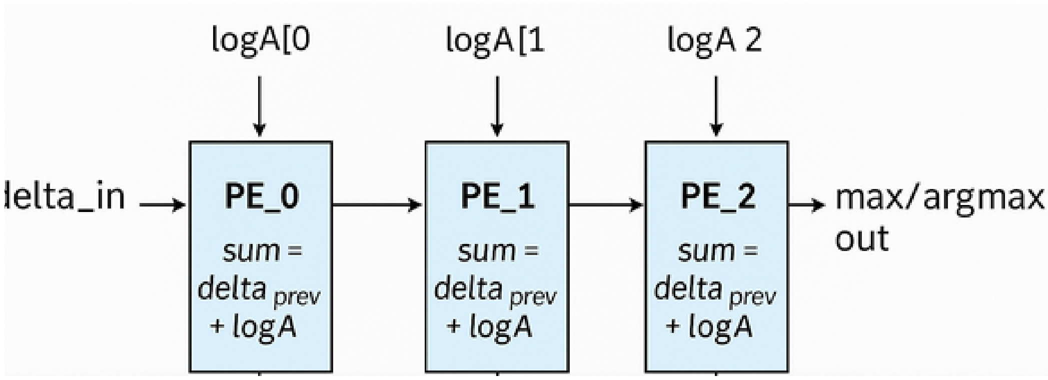
- **Complete OpenLane 2 placement** and evaluate timing, area, power. (Currently stuck due to congestion of pins)
- **Implement forward-log accelerator** (log-sum-exp tree) alongside Viterbi PE array.
- **Automate regression** in GitHub Actions: nightly SW benchmarks vs. HW performance.

12. Topics Learnt and implemented

During this project, I learned these main topics:

- **HW/SW Co-Design**: partitioning algorithms between software and hardware
- **SystemVerilog RTL**: multi-dimensional arrays, generate loops, FSMs, pipelining
- **Fixed-Point Arithmetic**: log-domain representation, two's-complement packing
- **Performance Profiling**: Python's cProfile, line_profiler, and py-spy for compute/memory hotspots

- **Systolic Arrays**: design of 1D arrays for ACS operations (max-plus DP)



Systolic Arrays

- **Verification**: Icarus Verilog testbenches and cocotb-based Python verification
- **ASIC Flow**: Yosys synthesis, OpenLane 2 place-&-route, PDN and I/O constraints
- **High-Level Synthesis (HLS)**: mapping Python-style loops to hardware pipelines
- **Physical Design**: power grid insertion, congestion analysis, constraints scripting
- **Documentation & Portfolio**: codefest reports, Heilmeier questions, GitHub portfolio curation