**Challenge #19: Implementing a Binary LIF Neuron — Findings & Reflections**

In this challenge, I explored and implemented a simplified **Binary Leaky Integrate-and-Fire (LIF) neuron model** in Verilog. The primary goal was to model a neuron's spiking behavior over time based on its internal membrane potential (P(t)), a leak factor (λ), and a threshold (θ). The neuron should accumulate input, decay over time, and emit a binary spike output (S(t)) when the threshold is reached. This simulation captures core aspects of biological spiking neurons while maintaining a minimal digital abstraction.

The initial task was to translate the mathematical formulation of the LIF neuron into a hardware implementation using Verilog. I used a fixed-point representation (e.g., 8.8 format) to model real-valued decay and accumulation efficiently in digital logic. I then implemented the potential update rule: P(t) = λP(t-1) + I(t) with support for a reset when the neuron spikes. A key insight was managing non-blocking assignments in SystemVerilog to ensure that potential values were updated correctly across clock cycles and to prevent logic from updating prematurely or getting reset before being evaluated.

Once the basic neuron design was complete, I developed a comprehensive testbench that demonstrated the neuron’s response under four scenarios: (1) constant low input, (2) accumulating input leading to threshold crossing, (3) input absence demonstrating leakage, and (4) strong input causing immediate spiking. During testing, I noticed unexpected behaviors—most notably, the neuron spiked too quickly or failed to spike at all in some conditions. This was due to overly aggressive input increments or incorrect threshold calibration. To address this, I adjusted the input magnitude, leak factor, and threshold until the neuron demonstrated realistic gradual buildup behavior and decayed as expected when inactive.

Throughout the challenge, I encountered multiple implementation and debug hurdles. For example, it wasn’t initially obvious that the spike condition wasn’t triggering correctly due to the timing of non-blocking assignments. I also had difficulty observing the internal potential correctly because I was printing values after they had already been reset. By consulting a large language model (LLM), I was able to quickly diagnose these timing issues, correct how internal variables were printed and compared, and gain insights on how to tune fixed-point arithmetic for smooth neuron behavior. The LLM also helped me explore various design trade-offs—like changing input granularity or threshold values—and provided ready-to-use Verilog templates that sped up the development process.

In summary, this challenge helped me solidify my understanding of how biological neuron dynamics can be abstracted into hardware logic. It also improved my practical skills in designing and debugging fixed-point systems, managing stateful sequential logic, and writing targeted testbenches. Leveraging an LLM during development gave me clarity on implementation nuances and accelerated my problem-solving process. This experience reinforced the importance of iterative refinement, parameter tuning, and simulation-driven validation in digital neuromorphic designs.