**LLM-Assisted HDL Design for Neuromorphic Circuits**

**1. Introduction**

The intersection of artificial intelligence and hardware design has opened new frontiers for automation and innovation. This project explores how Large Language Models (LLMs), like ChatGPT, can be leveraged to automatically generate synthesizable Verilog code for biologically inspired neuron models. The primary inspiration is drawn from the Johns Hopkins paper titled "Designing Silicon Brains using LLM" (<https://arxiv.org/abs/2402.10920>), which demonstrates the feasibility of generating spiking neuron arrays using LLMs.

The focus here is on designing two types of neurons: the **Leaky Integrate-and-Fire (LIF)** neuron and the **Rectified Linear Unit (RLU)** neuron. The project aims to generate HDL code, organize reusable module structures, and optionally synthesize the designs using tools like OpenLane.

**2. Learning Goals**

* Experiment with natural language-driven design for digital circuits.
* Generate HDL (Verilog) code for spiking and analog neuron models.
* Understand key hardware modeling patterns (state retention, thresholding, reset logic).
* Document prompt design, iteration loops, and challenges in code synthesis.
* Explore alternative neuron models and assess their feasibility in silicon.

**3. Design and Implementation**

* **A. LIF Neuron** (File: lif\_neuron.v)

The Leaky Integrate-and-Fire (LIF) model is inspired by biological neurons. It integrates incoming spikes into a membrane potential, which leaks over time. Once the potential crosses a certain threshold, the neuron fires and resets.

**Key features:**

* Stateful logic using `membrane\_potential`
* Threshold comparison and reset
* Leak parameter for decay control

* **B. LIF Neuron Array** (File: lif\_neuron\_array.v)

To enable multiple neurons to work in parallel, a parameterized neuron array was built using Verilog’s `generate` block.

**Key features:**

* Scalable using `N` parameter
* Each neuron functions independently
* Input/output is vectorized
* **C. RLU Neuron** (File: relu\_neuron.v)

The \*\*Rectified Linear Unit (RLU)\*\* is a widely used activation function in machine learning. It’s a simple model where outputs are passed through directly if positive or clamped to zero if not.

**Key features:**

- Stateless logic

- Signed input/output for analog computation

- Ideal for hardware-based inference engines

**4. LLM Prompting Strategy**

To generate the HDL code, a series of carefully crafted prompts were used with ChatGPT:

**Example Prompts:**

1. “Write Verilog code for a Leaky Integrate-and-Fire (LIF) neuron with spike input, threshold, membrane decay, and spike output.”

2. “Extend the LIF neuron module to support an array of neurons using generate blocks.”

**Challenges Encountered:**

* Some versions omitted `parameter` definitions.
* Port directions were sometimes incorrect or missing.
* Needed to refactor state machines and memory into registers.
* LLM responses needed several iterations for optimized synthesis.

Despite these issues, LLM was incredibly effective in quickly generating working HDL blueprints.

**5. Simulation (Optional)**

While simulation was not required in the challenge, testbenches were created for both LIF and RLU neurons using Icarus Verilog. The simulations aimed to test:

* Spike accumulation and reset in LIF neurons
* Clamping logic in RLU neuron for various signed inputs

**Simulations can be run using:** `iverilog` + `vvp` + `gtkwave` (optional)

If skipping simulation, it’s recommended to explain that focus was placed on LLM-HDL generation and structural correctness.

**6. OpenLane Synthesis (Optional / Future Work)**

To fully replicate the Johns Hopkins paper, the HDL can be passed through OpenLane, a digital synthesis flow, to generate physical layout (GDSII).

**Planned workflow:**

* Use OpenLane Docker setup
* Run synthesis, floorplanning, placement, and routing
* Compare area, timing, and power metrics

This step was deferred due to time constraints, but code was written with synthesis readiness in mind.

**7. Comparative Table: LIF vs RLU**

|  |  |  |
| --- | --- | --- |
| **Feature** | **LIF Neuron** | **RLU Neuron** |
| Memory | Yes (membrane potential) | No |
| Threshold-based | Yes | Yes (at 0) |
| Stateful | Yes | No |
| Biologically-inspired | Yes | No |
| HDL Complexity | Medium | Low |
| Use Case | Spiking Neural Networks | Deep Learning Accelerators |

**8. Suggested Improvements**

* Add AXI or bus-based wrappers to support memory-mapped integration.
* Include runtime threshold/leak adjustment using input ports.
* Auto-generate testbenches using LLM (future automation).
* Explore neuron model switching based on application needs (e.g., RLU for CNN, LIF for SNN).

**9. Conclusion**

This project successfully demonstrated the ability of LLMs to assist in HDL design for neuromorphic computing. Both the LIF and RLU neuron models were implemented with minimal iteration, showing that prompt engineering can accelerate circuit design. While LLMs require guidance to ensure correctness and synthesis-compatibility, they represent a promising path for rapid hardware prototyping.

Future work includes completing OpenLane synthesis, adding synapse modeling, and benchmarking real-world inference workloads on these custom architectures.

**10. Enhanced LIF Neuron (with Parameters)**

To align more closely with the paper and support tunability, the LIF neuron was modified to accept runtime parameters:

```verilog

module lif\_neuron (

input wire clk,

input wire reset,

input wire [7:0] current\_in,

input wire [7:0] threshold,

input wire [7:0] leak\_rate,

output reg spike

);

reg [7:0] membrane\_potential;

always @(posedge clk or posedge reset) begin

if (reset) begin

membrane\_potential <= 0;

spike <= 0;

end else begin

if (membrane\_potential < leak\_rate)

membrane\_potential <= current\_in;

else

membrane\_potential <= membrane\_potential + current\_in - leak\_rate;

if (membrane\_potential >= threshold) begin

membrane\_potential <= 0;

spike <= 1;

end else begin

spike <= 0;

end

end

end

endmodule

```

This change allows each neuron to behave differently during simulation or when instantiated in a larger system.

**Reflection Q&A**

|  |  |
| --- | --- |
| **Question** | **Answer** |
| Can ChatGPT generate bug-free HDL? | Almost — small syntax and logic issues are common and must be debugged. |
| What issues appear often? | Missing register declarations, incorrect sensitivity lists, misuse of `wire` vs `reg`. |
| Can LLMs fix their own mistakes? | Yes, especially when given clear feedback or prompt corrections. |
| Do LLMs understand HDL architecture? | Partially — they mimic structure well but may struggle with deep hierarchy or reuse. |
| Is this useful for beginners? | Very helpful for guidance but requires supervision. Beginners may struggle to debug without context. |

**Future Work**

* Implement SPI or AXI interfaces to connect neurons with a CPU or controller.
* Explore top-level integration in a system-on-chip (SoC) testbench.
* Attempt GDS layout generation using OpenLane with PDKs.
* Add neuron-specific initialization and programmable leakage control.

**Final Reflection**

This project helped me build confidence in using LLMs to assist with real-world digital design. By iterating through multiple architectural styles and refining HDL through prompts, I not only improved my understanding of neuromorphic models but also realized the practical benefits of AI-assisted engineering workflows. It has motivated me to explore LLM-driven simulation, synthesis, and verification further in upcoming projects.