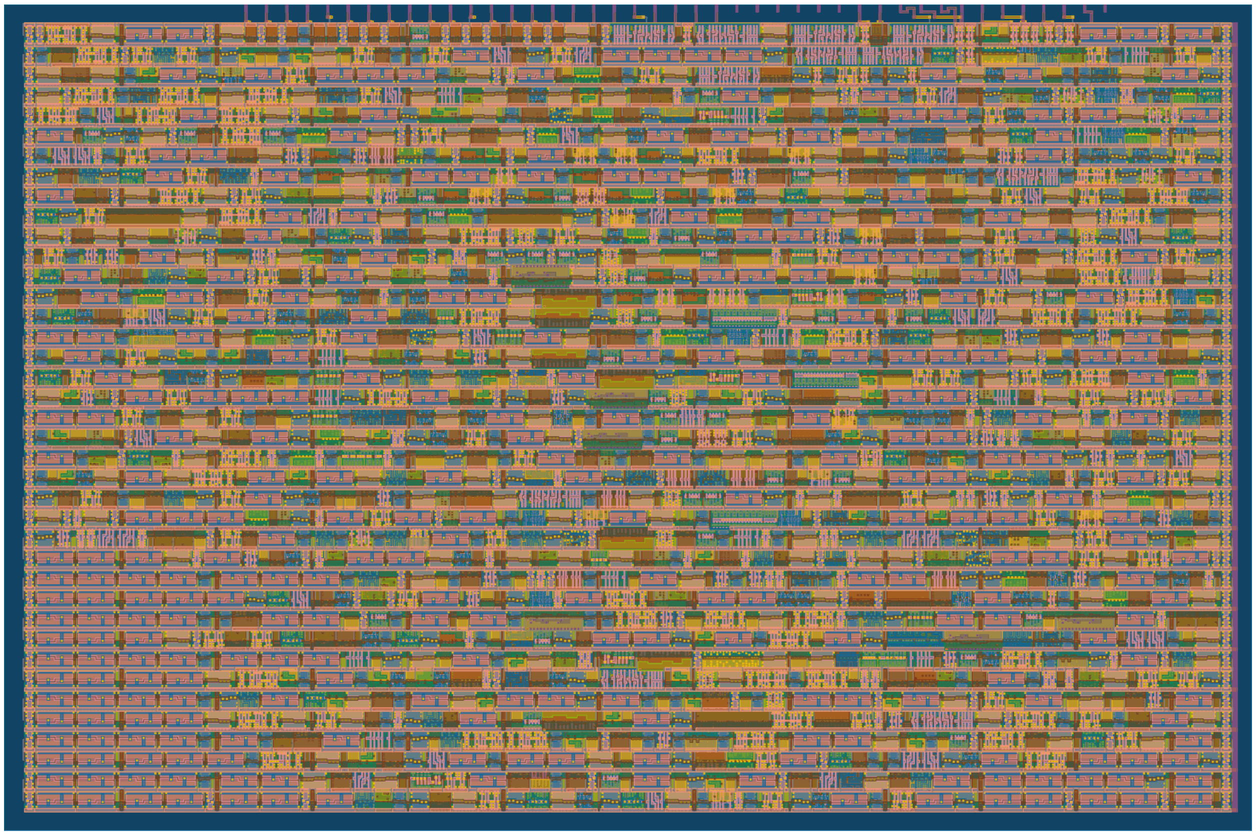
ECE183 CHIP DESIGN REPORT

Adaptive Leaky-Integrated Fire Neurons



Ruhai Lin

11.15.2023

Brain-Inspired Machine Learning

GitHub Link: https://github.com/ruhai-lin/tt05-lif-demo

ABSTRACT

The primary goal of this project was to delve into the world of Verilog, a hardware description language, and to embark on the design of a Leaky Integrate-and-Fire (LIF) neuron model. Instead of relying on pre-existing SNN libraries, I created my own LIF neuron with Verilog. Building upon this foundation, the project extended the adaptive threshold and adaptive decay rate. These additions enhanced spike sparsity while preserving the intrinsic characteristics of the original data as I expected.

CHIP DESIGN

In the inaugural stage of this chip design project, a foundational Leaky Integrate-and-Fire (LIF) neuron was meticulously crafted. The LIF neuron, a fundamental element within neural network architectures, was mathematically modeled to simulate the dynamic interplay of synaptic inputs, membrane potential, and the subsequent firing mechanism. The LIF model, expressed through the differential equation[1]:

$$U[t+1] = \beta U[t] + WX[t] - S_{out}(\beta U[t] + WX[t]) \qquad (1)$$

where U[t + 1] represents the membrane potential, β is the decay rate, W denotes the weight of the current injection, X[t] signifies the current injection, and S means whether there is a spike or not, building up the reset mechanism. Simply convert it to Verilog codes:

```verilog
assign next_state = (spike ? 0 : (current)) +
                    (spike ? 0 : (state * beta >> 8));

assign spike = (state >= threshold);
```

After that, I updated the LIF neuron by adding the adaptive threshold and adaptive beta. These two variables allowed the neuron to automatically adapt to the change of the current injection. When a spike is generated, the threshold and the decay rate will increase, thus the neuron will generate spikes less frequently. When there is no spike, the threshold and the decay rate will decrease, thus the neuron will spike more frequently.

By setting it so, the neuron will still generate spikes frequently when the synaptic input increases, but after a while the spike rate will go back to the normal level. In the real world, every spike consumes energy, so a lower spike rate leads to lower energy consumption. The figure in the test & simulation section will explain the spike sparsity improvement in a more straightforward way.

A problem I met when I was implementing LIF neuron and adaptive parameters is that Verilog does not know what fraction is. For example, multiplying the threshold by 0.9 is not allowed. The division operation is even a disaster in Verilog projects. In more severe cases, even the direct application of the multiplication sign is not allowed. Fortunately, this chip design project was not so desperate that I had to conserve the space needed for multiplication, and I was able to use bitwise operations. So, taking the adaptive threshold as an example, the way I obtained the new threshold was to first multiply the threshold by an integer, and then do a rightward bit shift operation, as shown in the code block below:

```
parameter ADAPTIVE_INCREMENT = 295;  // adaptive increment factor
parameter ADAPTIVE_DECREMENT = 250;  // adaptive decrement factor
if (spike) begin
    state <= 0;
    // see if adaptive threshold on onand avoiding overflow 255
    if (adaptive_threshold && (threshold < 220))
        // increase threshold
        threshold <= threshold * ADAPTIVE_INCREMENT >> 8;
    end else begin
        state <= next_state;
    // avoiding the threshold to be too low
    if (adaptive_threshold && (threshold > 32))
        // decrease threshold
        threshold <= threshold * ADAPTIVE_DECREMENT >> 8;
    end
end
```

My final design is an adaptive LIF neuron. Its spike threshold increases as the input current stimulus increases and decreases as it decreases. For its decay rate β the case is opposite. This will put the neuron in a state where it normally fires spikes at a steady rate: when the input increases, the neuron fires a large number of spikes, but when the threshold and decay rate return to stability, the neuron returns to its normal state. When the input decreases, the neuron will fire fewer spikes than usual, which will also return to a normal spike rate after a sustained period of time. This form of neuron attempts to maintain its own spike rate at a normal level while retaining the characteristics of input changes, which is consistent with the strategy adopted by neurons in living organisms. It will be verified in the next section. Finally, the design passed the Action test on Github, and Fig. 1 shows the final logic diagram of the circuit.

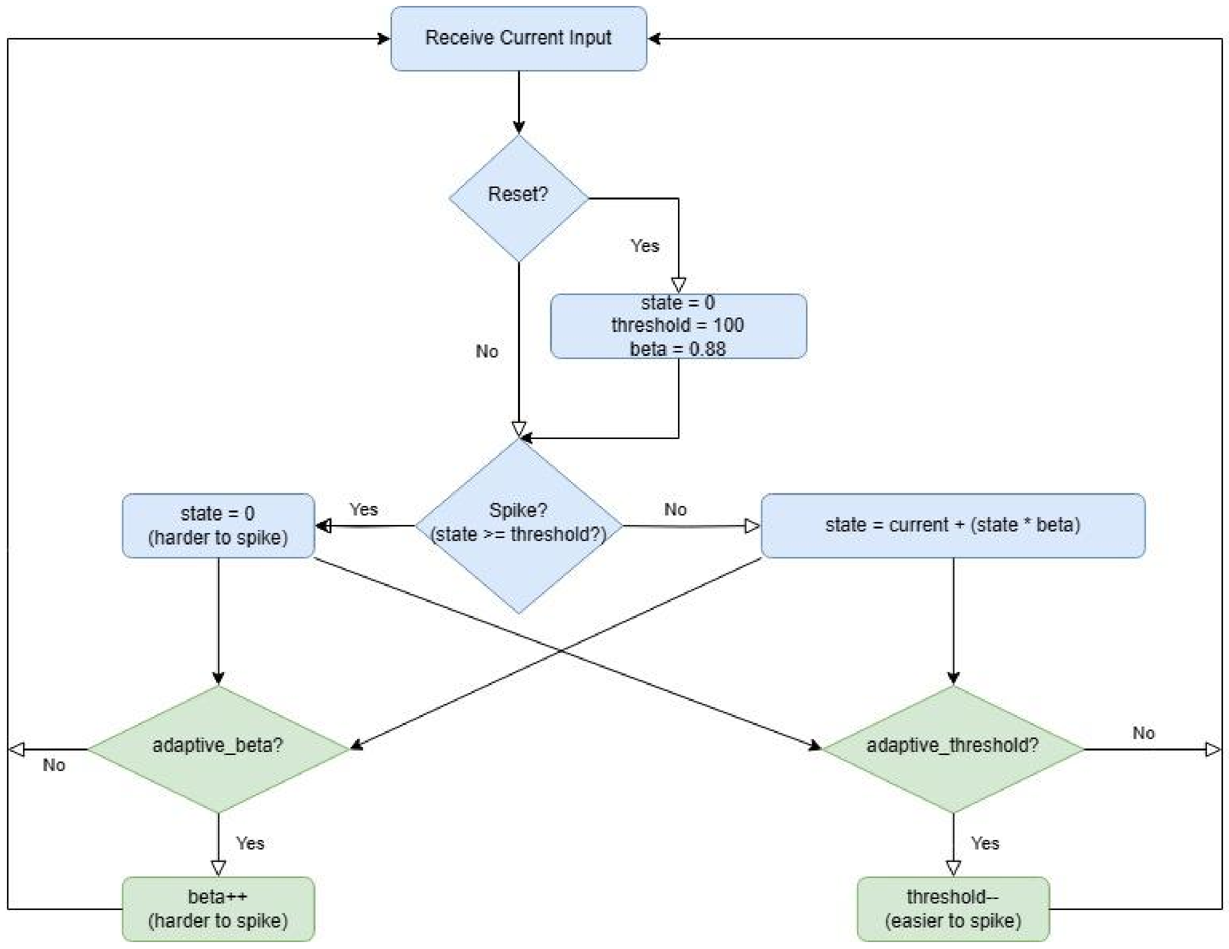


Fig. 1 | The Circuit Diagram of an Adaptive LIF Neuron. It takes a clock signal, a current, a reset signal, and two switching signals for adaptive function as the inputs, and outputs the state (membrane potential) of the neuron and a spike if generated.\

TEST & SIMULATION

To actually see the outputs, Docker is employed to overcome limitations in GitHub's testing capabilities for Verilog code, providing a comprehensive testing environment that includes Cocotb, Icarus Verilog, and pytest. This encapsulated container simplifies deployment, ensuring a uniform setup and facilitating easy code execution, with simulation results visualized using GTKWave software. (Note: Ridger gave me an insight about these, he should be rewarded)

Since I created adaptive thresholds and adaptive decay rates that can be turned on or off, there will be a total of four scenarios for the outputs, and it was very interesting to compare them together and evaluate the role of these two variables. Unfortunately, due to space constraints, I can only put an overview figure here, which is not very clear, as shown in Fig. 2.

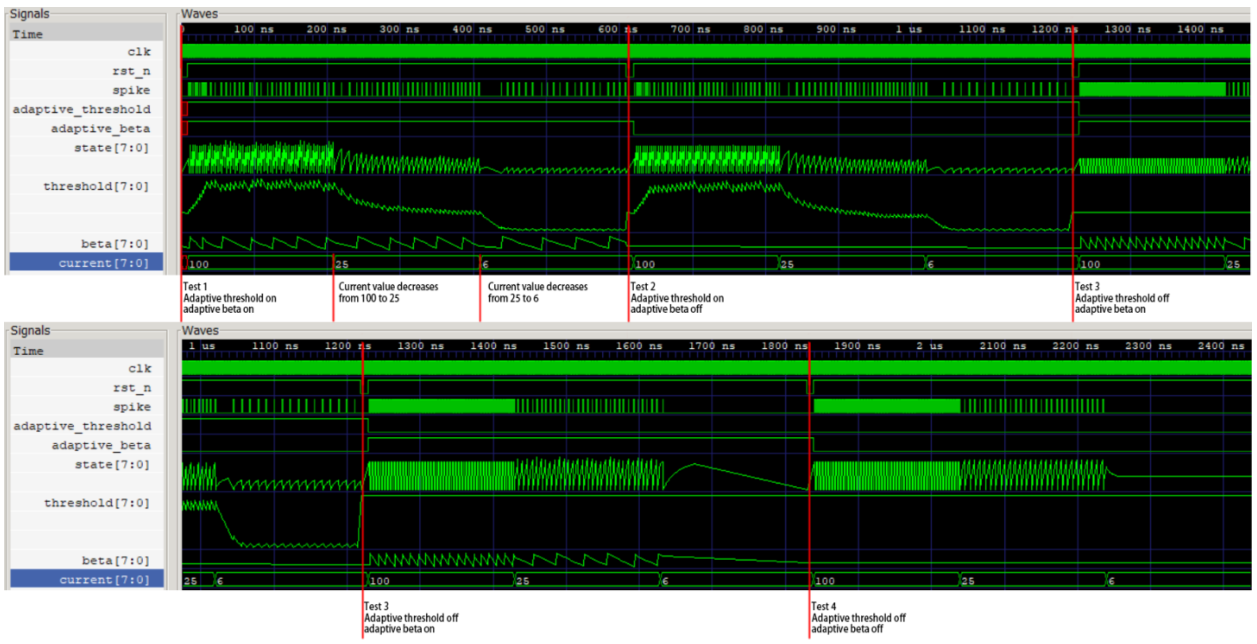


Fig. 2 | The Output Waveforms of a Single Adaptive Neuron. I used the red lines to divide the waveform into four sections, each of which was tested with three different strengths of current input. You can focus on the signals named "spike", which represents the main output of the neuron and directly visualizes the role of the adaptive parameters.

The observed outcomes align with the anticipated behavior during the chip design phase. In the initial test, where adaptive parameters were activated, the spike rate at 100 input current exhibited marginal deviation from that at 6 input current. In contrast, the fourth test, conducted with adaptive parameters deactivated, displayed marked extremes, with neurons exhibiting wild spiking activity at 100 input current and complete inactivity at 6 input current. As I mentioned in the previous section, this is the feature that I am expecting. This means the test verified the correctness of the project.

The characteristics exhibited by neurons with activated adaptive parameters resonate with the stability and self-regulation mechanisms observed in biological cells. This inherent quality in biology ensures that cellular signaling occurs within an optimal range, contributing to the organism's overall stability. In the context of neural networks, the sparsity of spikes resulting from adaptive thresholding introduces enhanced energy efficiency to the network. This finding is consistent with conclusions drawn in preceding studies, supported by authoritative researchers who posit that adaptive neurons store and transmit information through non-firing: A neuron that holds information in its increased firing threshold tends to fire less often[2].

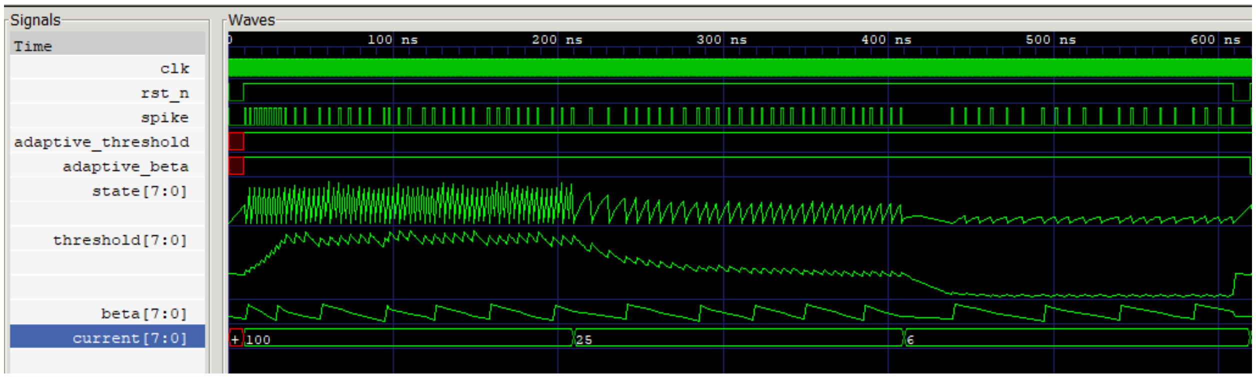
CONCLUSION

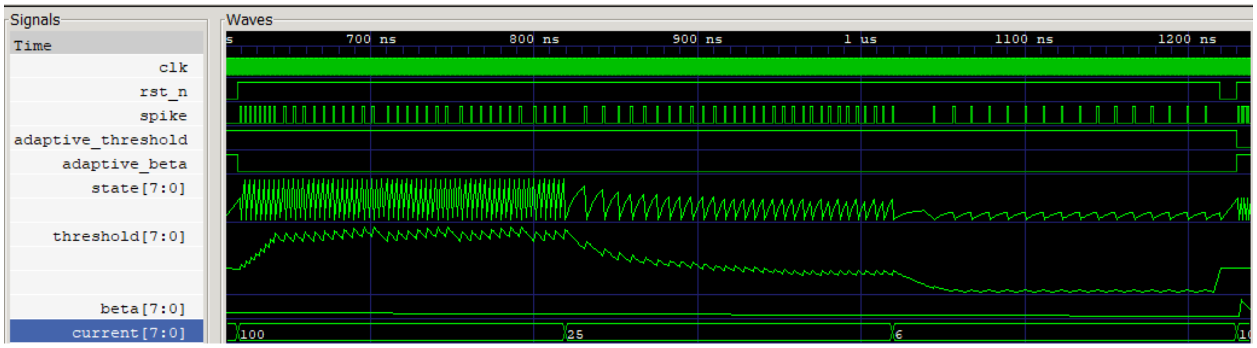
In summary, this project successfully implemented an adaptive Leaky Integrate-and-Fire (LIF) neuron in Verilog. The project achieved controlled spike rates akin to biological self-regulation. The test environment verifies the spike sparsity of the chip's output, emphasizes the energy efficiency gains it brings, and also corroborates with previous researchers' conclusions from a hardware perspective. I gained from this project a generalized solution for chip design and fabrication, an idea and environment to test the feasibility of the chip, a deeper and broader understanding of LIF neurons, and an insight into the relationship between hardware and software.

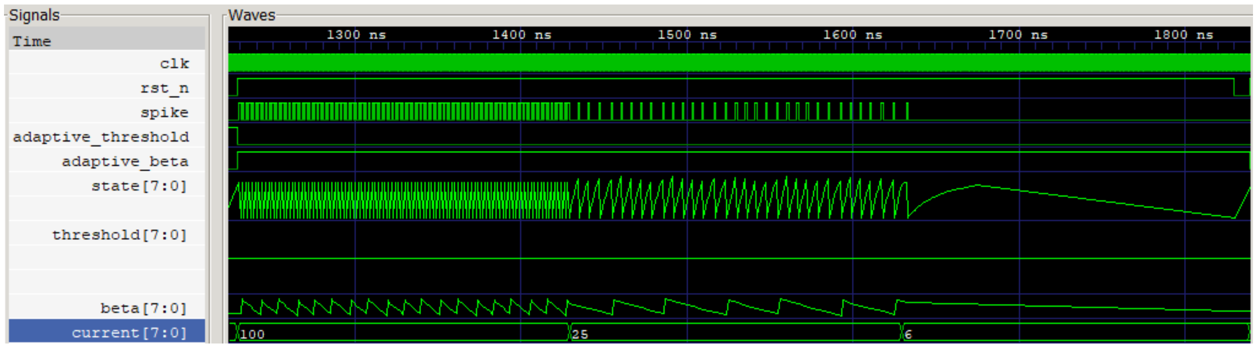
REFERENCES

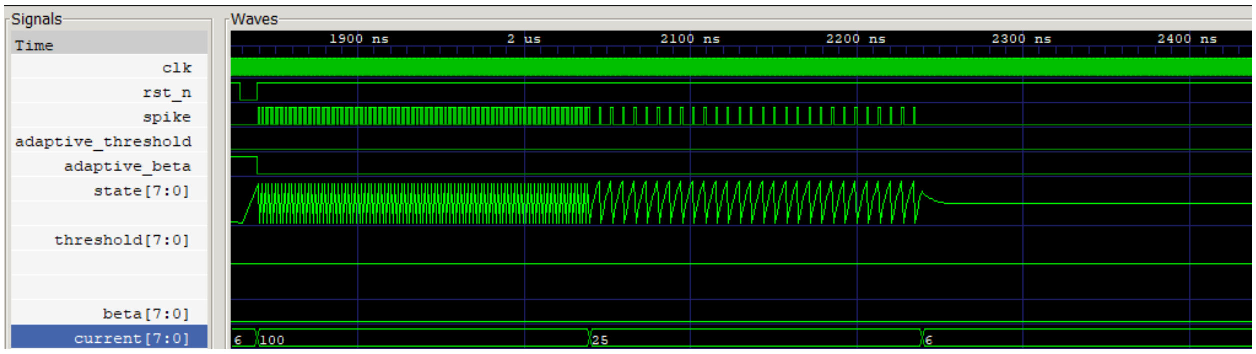
1. J. K. Eshraghian et al., "Training Spiking Neural Networks Using Lessons From Deep Learning," in Proceedings of the IEEE, vol. 111, no. 9, pp. 1016-1054, Sept. 2023, doi: 10.1109/JPROC.2023.3308088.
2. Bellec, Guillaume, et al. "Long short-term memory and learning-to-learn in networks of spiking neurons." Advances in neural information processing systems 31 (2018).

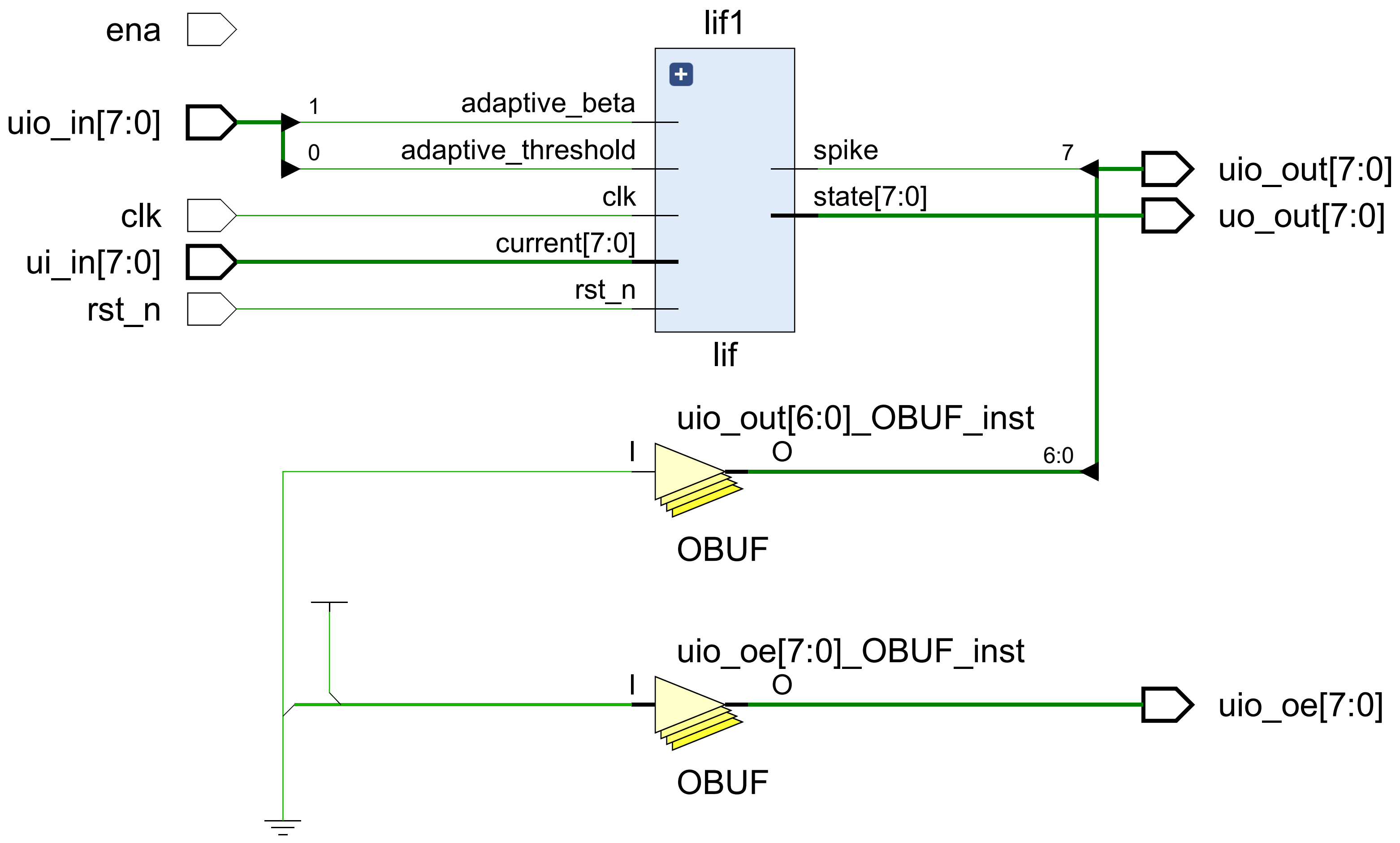
APPENDIX

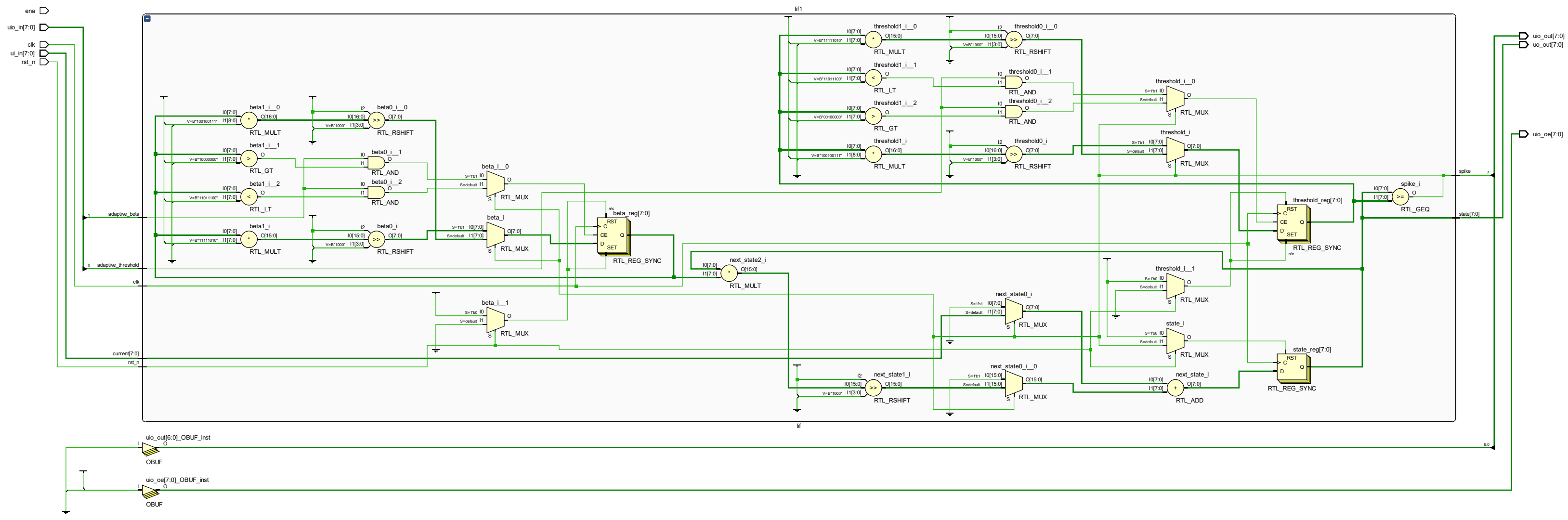

Signals
Waves
Time
clk
rst_n
spike
adaptive_threshold
adaptive_beta
state[7:0]
threshold[7:0]
beta[7:0]
current[7:0]
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
100
25
6


Signals
Waves
Time
clk
rst_n
spike
adaptive_threshold
adaptive_beta
state[7:0]
threshold[7:0]
beta[7:0]
current[7:0]
700 ns
800 ns
900 ns
1 us
1100 ns
1200 ns
100
25
6


Signals
Waves
Time
clk
rst_n
spike
adaptive_threshold
adaptive_beta
state[7:0]
threshold[7:0]
beta[7:0]
current[7:0]
1300 ns
1400 ns
1500 ns
1600 ns
1700 ns
1800 ns
100
25
6


Signals
Waves
Time
clk
rst_n
spike
adaptive_threshold
adaptive_beta
state[7:0]
threshold[7:0]
beta[7:0]
current[7:0]
1900 ns
2 us
2100 ns
2200 ns
2300 ns
2400 ns
6
100
25
6


ena
uio_in[7:0]
clk
ui_in[7:0]
rst_n
lif1
1
0
adaptive_beta
adaptive_threshold
clk
current[7:0]
rst_n
lif
spike
state[7:0]
7
uio_out[7:0]
uo_out[7:0]
uio_out[6:0]_OBUF_inst
I
O
6:0
OBUF
uio_oe[7:0]_OBUF_inst
I
O
uio_oe[7:0]
OBUF


ena
uio_in[7:0]
clk
ui_in[7:0]
rst_n
lif1
lif
adaptive_beta
adaptive_threshold
current[7:0]
beta1_i__0
RTL_MULT
V=B"100100111"
beta0_i__0
RTL_RSHIFT
V=B"1000"
beta1_i__1
RTL_GT
V=B"10000000"
beta0_i__1
RTL_AND
beta1_i__2
RTL_LT
V=B"11011100"
beta0_i__2
RTL_AND
beta1_i
RTL_MULT
V=B"11111010"
beta0_i
RTL_RSHIFT
beta_i__0
RTL_MUX
beta_i
RTL_MUX
beta_i__1
RTL_MUX
beta_reg[7:0]
RTL_REG_SYNC
next_state2_i
RTL_MULT
threshold1_i__0
RTL_MULT
V=B"11111010"
threshold0_i__0
RTL_RSHIFT
threshold1_i__1
RTL_LT
V=B"11011100"
threshold0_i__1
RTL_AND
threshold1_i__2
RTL_GT
V=B"00100000"
threshold0_i__2
RTL_AND
threshold1_i
RTL_MULT
V=B"100100111"
threshold0_i
RTL_RSHIFT
threshold_i__0
RTL_MUX
threshold_i
RTL_MUX
threshold_reg[7:0]
RTL_REG_SYNC
spike_i
RTL_GEQ
spike
state[7:0]
threshold_i__1
RTL_MUX
next_state0_i
RTL_MUX
state_i
RTL_MUX
next_state1_i
RTL_RSHIFT
next_state0_i__0
RTL_MUX
next_state_i
RTL_ADD
state_reg[7:0]
RTL_REG_SYNC
uio_out[7:0]
uo_out[7:0]
uio_oe[7:0]
uio_out[6:0]_OBUF_inst
OBUF
uio_oe[7:0]_OBUF_inst
OBUF