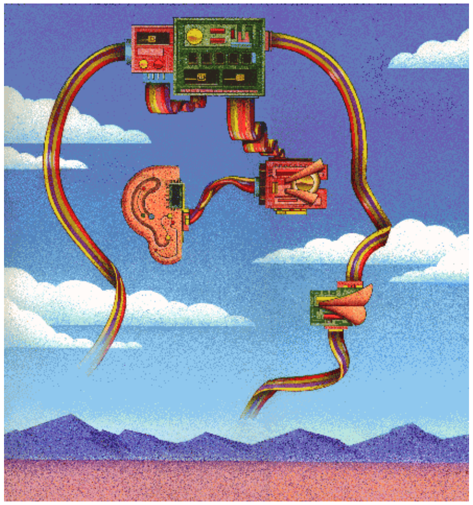
Generative AI for the Design of Digital Neuromorphic Spiking Neural Networks



Andreas G. Andreou

http:/andreoulab.net/

@andreoulab

Electrical and Computer Engineering,

Center for Language and Speech Processing

Johns Hopkins University

Contributions by:

Michael Tomlinson

Joe Li

Paola Vitolo.

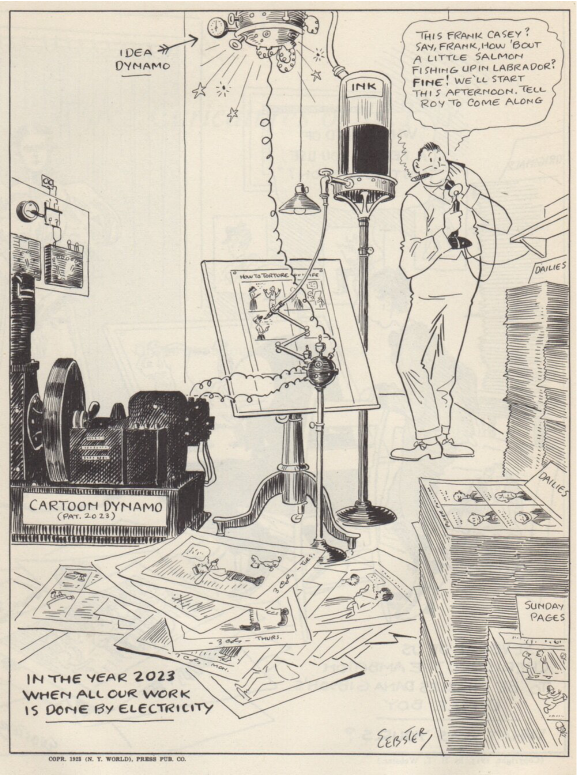
George Psaltakis

https://andreoulab.net

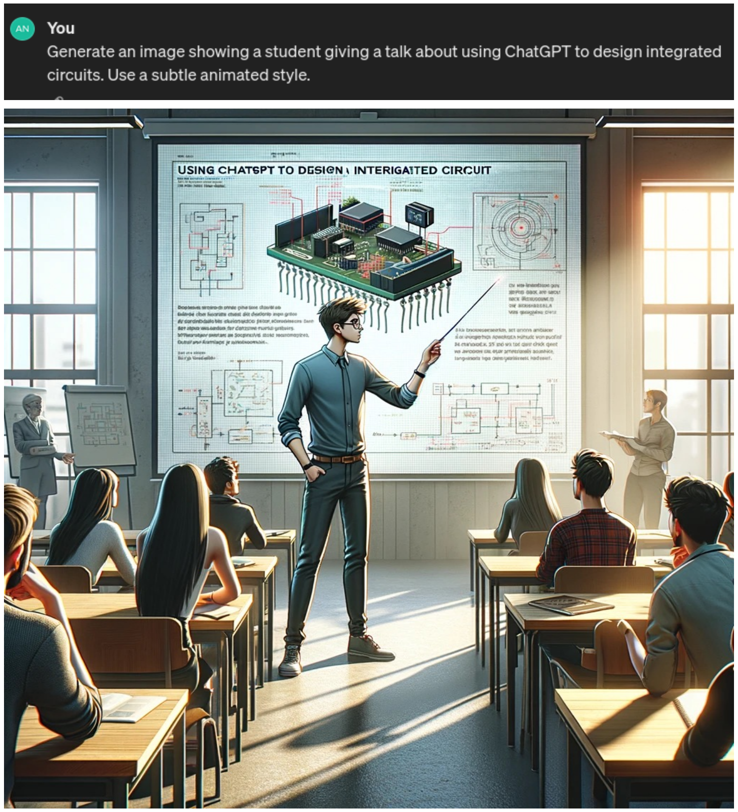
Outline

- Introduction
  - Historical perspective
  - Neuromorphic systems for embodied AI
- Natural Language based digital Spiking Neural Network chip design (a.k.a LLM assisted Verilog generation)
- Conclusions

2023: The Year of Generative-AI: Large Language Models and Stable Diffusion


IDEA DYNAMO
THIS FRANK CASEY? SAY, FRANK, HOW 'BOUT A LITTLE SALMON FISHING UP IN LABRADOR? FINE! WE'LL START THIS AFTERNOON. TELL ROY TO COME ALONG
INK
DAILIES
CARTOON DYNAMO (PAT. 2023)
DAILIES
SUNDAY PAGES
IN THE YEAR 2023 WHEN ALL OUR WORK IS DONE BY ELECTRICITY
WEBSTER
COPR. 1923 (N. Y. WORLD), PRESS PUB. CO.

A 1923 comic for *New York World* by cartoonist H. T. Webster (1885-1952)



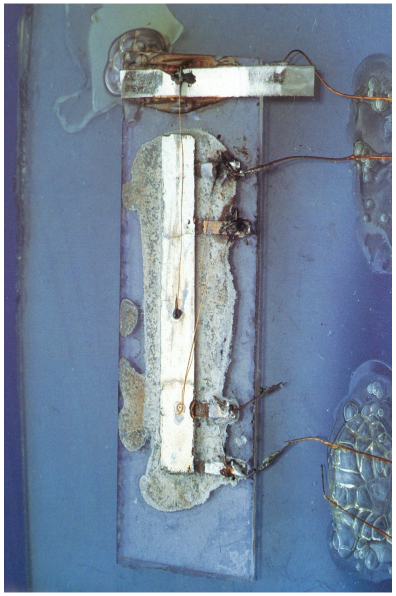
You

Generate an image showing a student giving a talk about using ChatGPT to design integrated circuits. Use a subtle animated style.

ChatGPT 4, March 2024

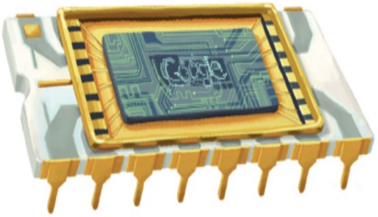
65 years of chips technology

Jack Kilby, Texas Instruments, Phase Shift Oscillator (1958)



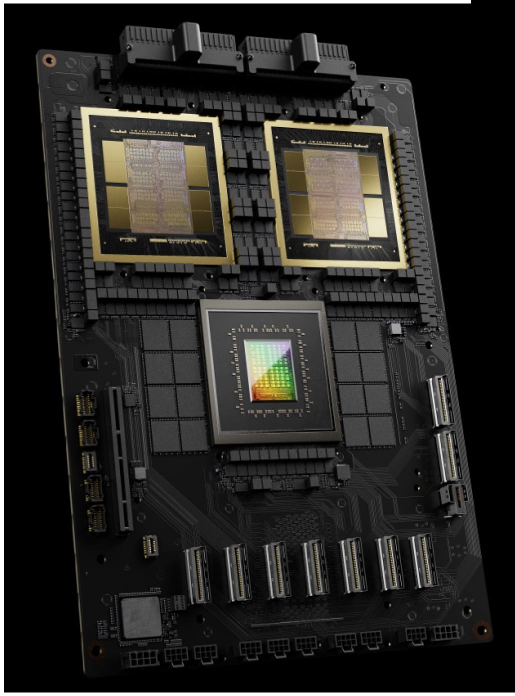
Feature size: mm

Robert Noyce, Fairchild/Intel Integrated Circuit (1959)



Google Doodle December 12, 2011

NVIDIA GB200 Grace Blackwell Superchip (2024)



Feature size: nm

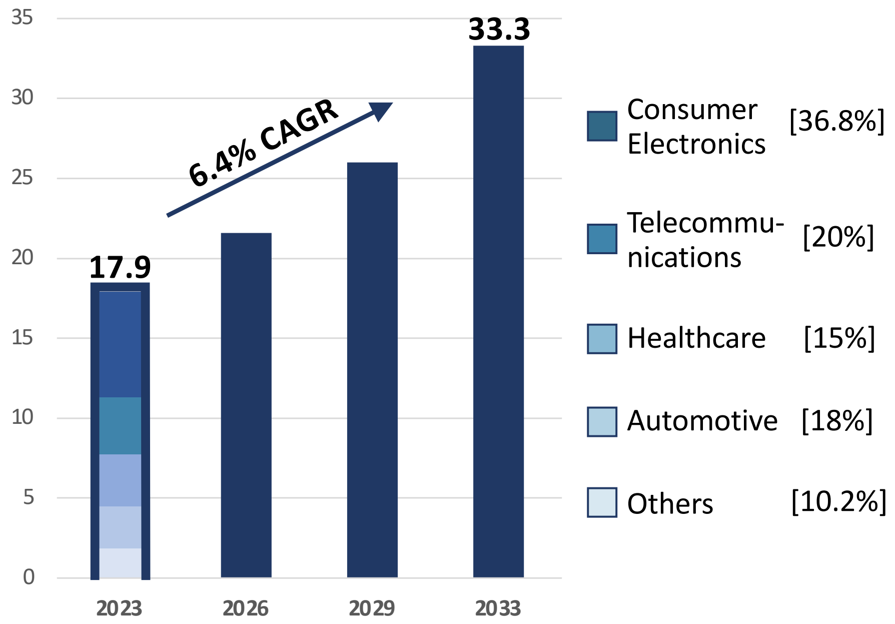
36 x GB200--- > GB200-NVL72

1 mm = 1000000 nm

ASICs – Application Specific Integrated Circuits

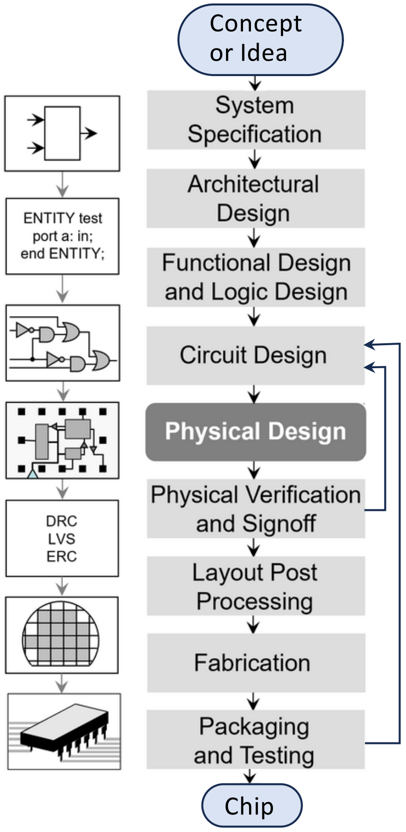
Image generated by ChatGPT 4.o

Global ASIC Market [USD Billion] [1]



[1] "Application specific integrated circuit market full report," market.us, accessed: 3 June 2024. [Online]. Available: https://market.us/report/application-specific-integrated-circuit-market.

Challenges in Hardware Design



Designing, **verifying**, and **fabricating** a chip is a **complex and expensive process**, often spanning several years and costing hundreds of millions of dollars.

Human Resources

Various expertise required:

- System Engineers
- Analog/Digital Engineers
- Layout Engineers
- Test & Verification Engineers
- Fabrication Teams

Time Resources

- Long design and verification times
- Fabrication times

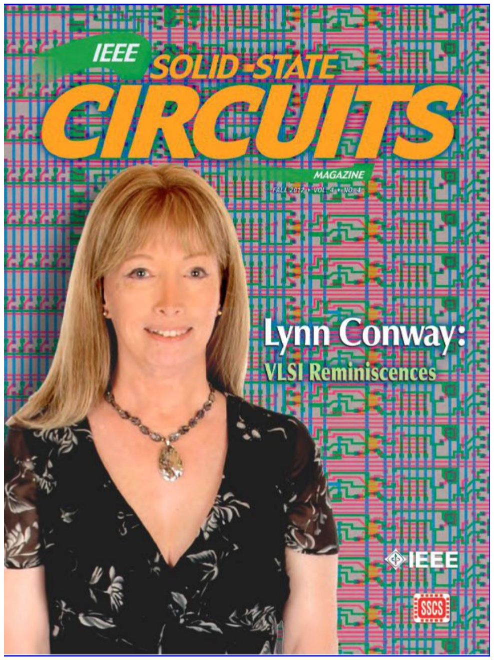
Physical Resources

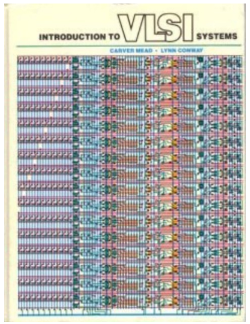
- Expensive Fabrication equipment
- Testing devices
- Software tools

Challenge: meeting the increasing demand for ASICs maintaining high levels of reliability and keeping costs low.

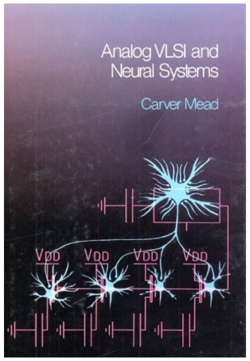
Natural Language for Architecture Exploration, Design and Verification

Design Methodology for Managing Complexity of Computational Systems





1980



1989

Lynn Ann Conway (January 2, 1938 – June 9, 2024)

2023 induction to the National Inventors Hall of Fame
https://www.invent.org/inductees/lynn-conway

Carver Mead (May 1, 1934 -)

2022 Kyoto Prize in Advanced Technology
(https://www.kyotoprize.org/en/2022/)

Moore's Law, CAD Tools and Foundry Services

Cramming more components onto integrated circuits

With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65,000 components on a single silicon chip

By Gordon E. Moore

Electronics, Volume 38, Number 8, April 19, 1965

cadence®

SYNOPSYS®
Predictable Success

Mentor Graphics®

Siemens EDA software

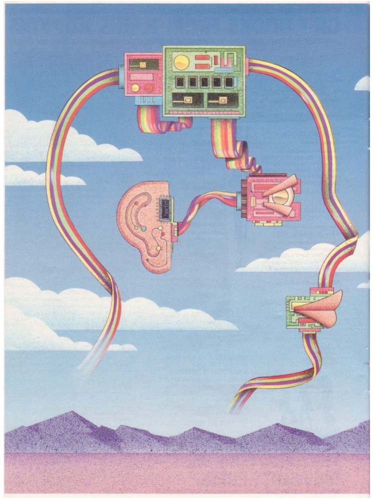
MOSIS Foundry

University EDA Tools
MAGIC, RSIM

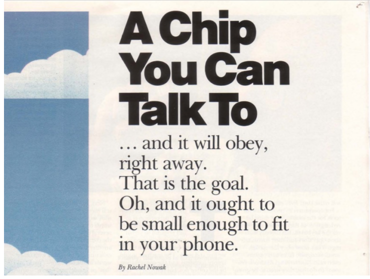
Advances in technology create advances in computers and CAD tools that in turn accelerate advances in technology

Neuromorphic Electronic Systems

Early 90s



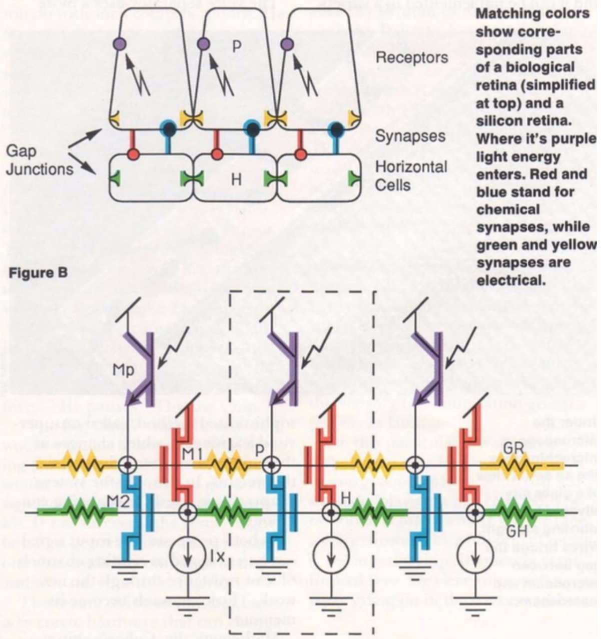
30 Johns Hopkins Magazine • December 1990



A Chip You Can Talk To

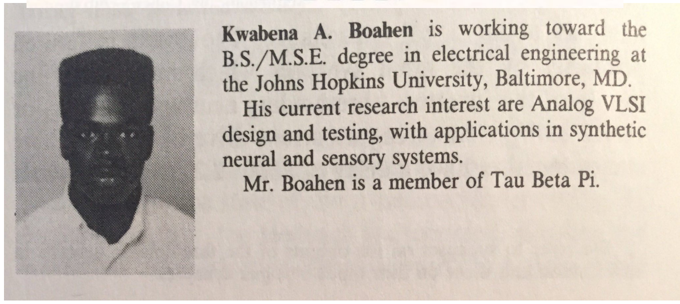
… and it will obey,
right away.
That is the goal.
Oh, and it ought to
be small enough to fit
in your phone.

By Rachel Nowak



Matching colors show corresponding parts of a biological retina (simplified at top) and a silicon retina. Where it's purple light energy enters. Red and blue stand for chemical synapses, while green and yellow synapses are electrical.

Figure B



Kwabena A. Boahen is working toward the B.S./M.S.E. degree in electrical engineering at the Johns Hopkins University, Baltimore, MD.

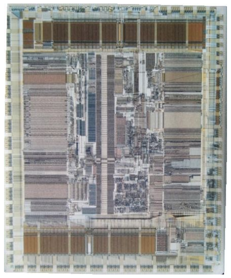
His current research interest are Analog VLSI design and testing, with applications in synthetic neural and sensory systems.

Mr. Boahen is a member of Tau Beta Pi.

<u>Compute environment:</u>

Pentium Pro, 0.5um,
5.5 M xtors
150 MHz, FSB:
66MHz, 3.3 Volts
on package 256KB L2
cache, 3.3V 35W,
64 MB RAM, 5GB HD

DEC Alpha 21064A,
0.5um, , 2.85 M xtors
200 MHz, 256KB
Bcache, 3.3 V, 30W,
64MB RAM, 5GB HD



<u>CAD environment</u>: Magic, IRSIM, SPICE

K. A. Boahen and A. G. Andreou, “A Contrast Sensitive Silicon Retina with Reciprocal Synapses,”Neural Information Processing Systems 3, 1990, vol. 4, pp. 764–772.

Nowak, R., & Andreou, A. G., A chip you can talk to, *Johns Hopkins Magazine*, 30–38, December 1990.

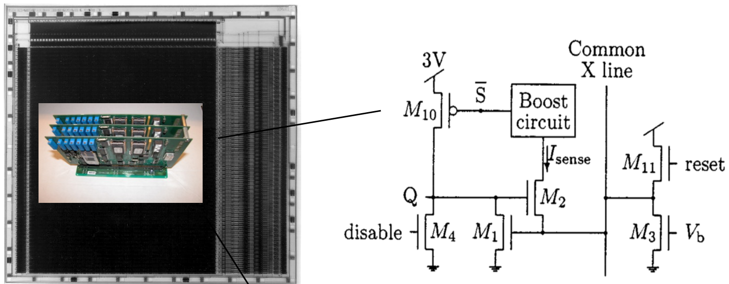
WAM Chip: Neuronmorphic processor In Memory (PIM), Compute In Memory (CIM), In Memory Computing (IMC)

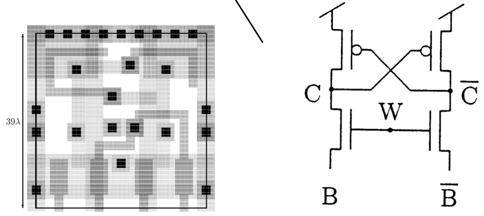
Winner-Takes-All Associative Memory:
A Hamming Distance Vector Quantizer

PHILIPPE O. POULIQUEN,[1] ANDREAS G. ANDREOU[1], AND KIM STROHBEHN[2]

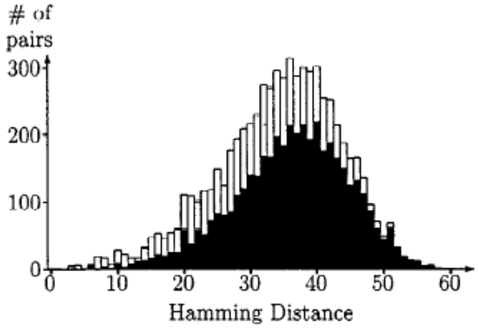
Analog Integrated Circuits and Signal Processing, 13, 211–222 (1997)

$$\max_{i=1,N}\left(\frac{|\vec{x}\wedge\vec{c}_i|}{a+|\vec{c}_i|}\right)$$

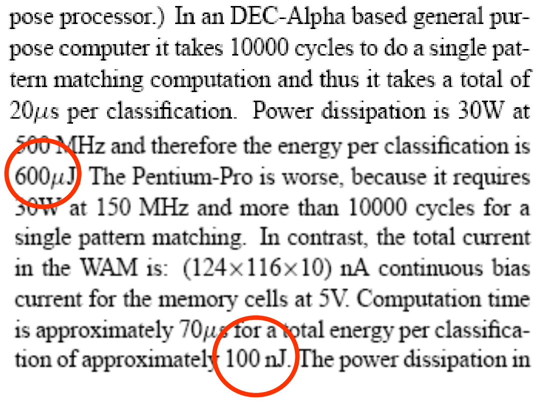




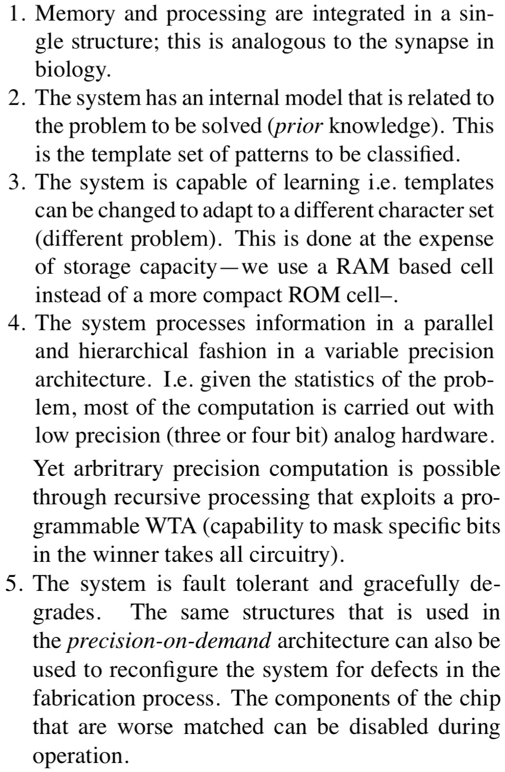
$$\vec{x}\wedge\vec{c}_i$$



exploiting problem statistics!



pose processor.) In an DEC-Alpha based general purpose computer it takes 10000 cycles to do a single pattern matching computation and thus it takes a total of 20μs per classification. Power dissipation is 30W at 500 MHz and therefore the energy per classification is 600μJ. The Pentium-Pro is worse, because it requires 30W at 150 MHz and more than 10000 cycles for a single pattern matching. In contrast, the total current in the WAM is: (124×116×10) nA continuous bias current for the memory cells at 5V. Computation time is approximately 70μs for a total energy per classification of approximately 100 nJ. The power dissipation in



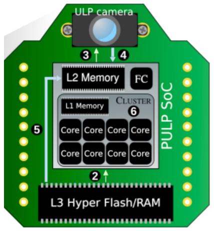
1. Memory and processing are integrated in a single structure; this is analogous to the synapse in biology.
2. The system has an internal model that is related to the problem to be solved (*prior* knowledge). This is the template set of patterns to be classified.
3. The system is capable of learning i.e. templates can be changed to adapt to a different character set (different problem). This is done at the expense of storage capacity—we use a RAM based cell instead of a more compact ROM cell–.
4. The system processes information in a parallel and hierarchical fashion in a variable precision architecture. I.e. given the statistics of the problem, most of the computation is carried out with low precision (three or four bit) analog hardware.

   Yet arbitrary precision computation is possible through recursive processing that exploits a programmable WTA (capability to mask specific bits in the winner takes all circuitry).
5. The system is fault tolerant and gracefully degrades. The same structures that is used in the *precision-on-demand* architecture can also be used to reconfigure the system for defects in the fabrication process. The components of the chip that are worse matched can be disabled during operation.

2um CMOS technology, 5Volts

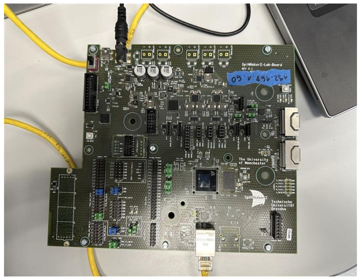
Neural AI accelerators for Edge

A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors.," Phil Trans A vol. 378, no. 2164, p. 20190155, Feb. 2020

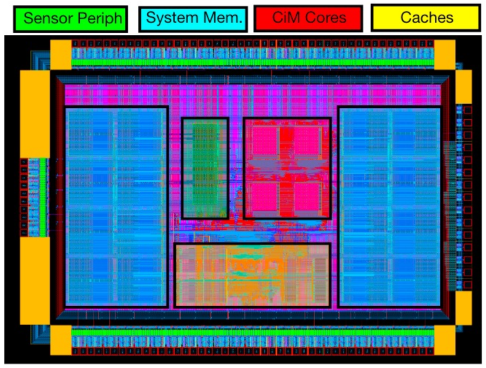


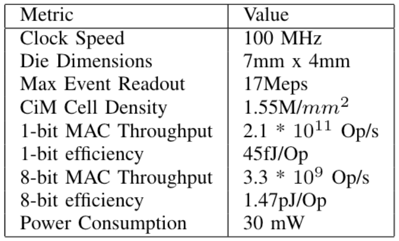
GAP-8
GAP-9

SpiNNaker2 172 M4 FP cores
DNN accelerators



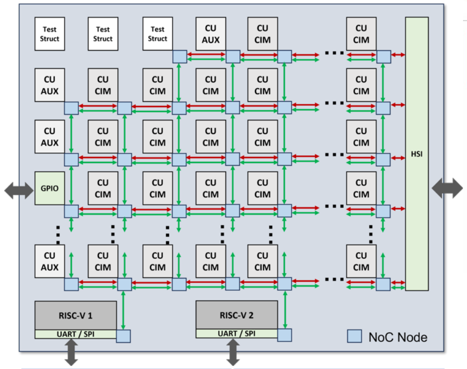
C. Mayr, S. Hoeppner, and S. Furber, "SpiNNaker 2: http://arxiv.org/abs/1911.02385

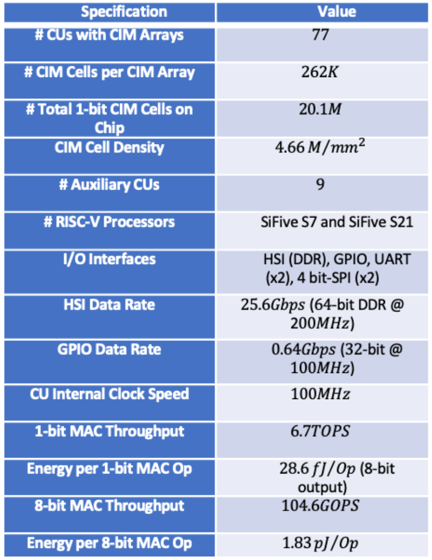




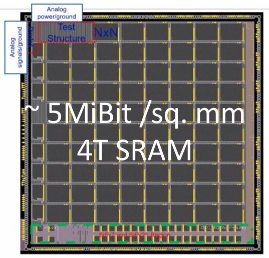
Metric	Value
Clock Speed	100 MHz
Die Dimensions	7mm x 4mm
Max Event Readout	17Meps
CiM Cell Density	1.55M/mm^2
1-bit MAC Throughput	2.1 $*$ 10^{11} Op/s
1-bit efficiency	45fJ/Op
8-bit MAC Throughput	3.3 $*$ 10^9 Op/s
8-bit efficiency	1.47pJ/Op
Power Consumption	30 mW

D. R. Mendat, A.G. Andreou *et al.*, "A RISC-V Neuromorphic Micro-Controller Unit (vMCU) with Event-Based Physical Interface and Computational Memory for Low-Latency Machine Perception and Intelligence at the Edge," in *ISCAS-2023.*





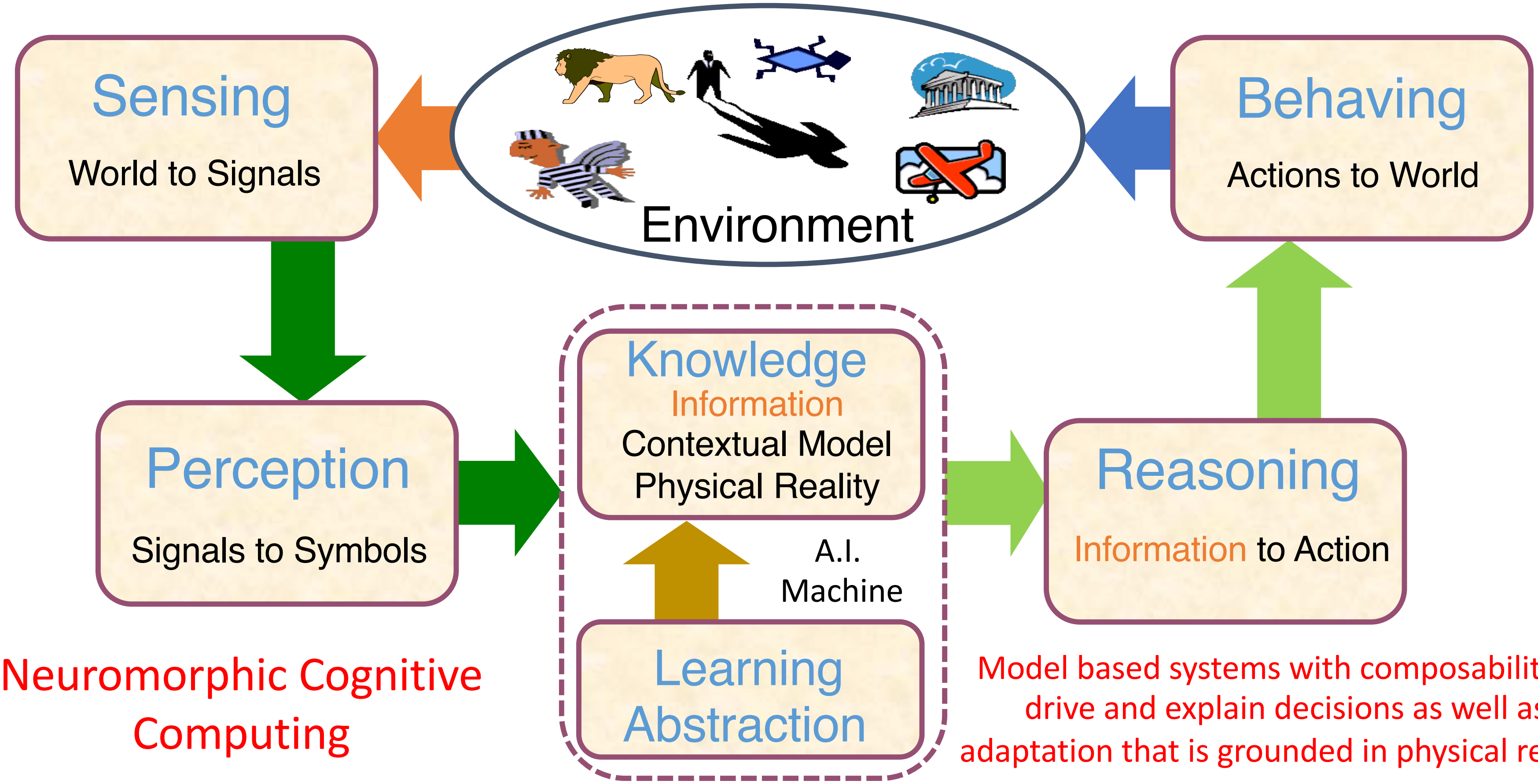
Specification	Value
# CUs with CIM Arrays	77
# CIM Cells per CIM Array	$262K$
# Total 1-bit CIM Cells on Chip	$20.1M$
CIM Cell Density	4.66 M/mm^2
# Auxiliary CUs	9
# RISC-V Processors	SiFive S7 and SiFive S21
I/O Interfaces	HSI (DDR), GPIO, UART (x2), 4 bit-SPI (x2)
HSI Data Rate	25.6$Gbps$ (64-bit DDR @ 200MHz)
GPIO Data Rate	0.64$Gbps$ (32-bit @ 100MHz)
CU Internal Clock Speed	100MHz
1-bit MAC Throughput	6.7$TOPS$
Energy per 1-bit MAC Op	28.6 fJ/Op (8-bit output)
8-bit MAC Throughput	104.6$GOPS$
Energy per 8-bit MAC Op	1.83 pJ/Op



~ 5MiBit /sq. mm
4T SRAM

J. L. Molin and A. G. Andreou et.al., "Edge Super Compute (ESC) 2.0 A Low Power System-on-Chip AI Accelerator for the Edge," in *Proceedings GOMACTech-2024*, 20 March 2024.

Today: AI/ML in embodied systems at the EDGE and 3rd Wave of AI



Sensing
World to Signals

Environment

Behaving
Actions to World

Perception
Signals to Symbols

Knowledge
Information
Contextual Model
Physical Reality

A.I.
Machine

Learning Abstraction

Reasoning
Information to Action

Neuromorphic Cognitive Computing

Model based systems with composability to drive and explain decisions as well as adaptation that is grounded in physical reality.

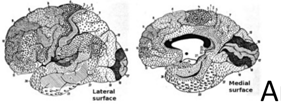
AI Makes Neuromorphic AI chips (2023–

Spiking Neural Networks (SNNs) designed with LLMs – ChatGPT

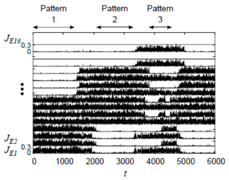
Computational Theory

Parallel Processing Under Physical Constraints (Delay, Energy, Variability)

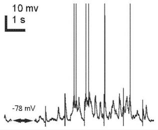
Algorithms Architecture and Representation



Brain Architectonics

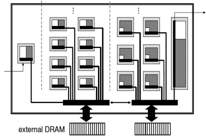


Attractor Dynamics and Network Computation

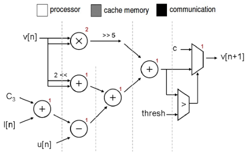


Spike based Processing EPSP/IPSPc

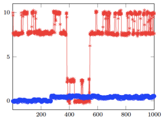
Multiprocessor Architecture



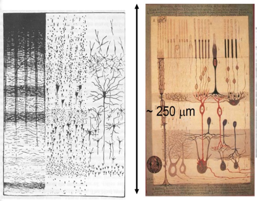
Fine-Grained Parallelism



Event Based Information Processing expoiting stochasticity



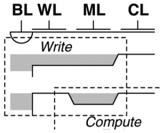
Physical Implementation



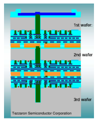
Santiago Ramon y Cajal

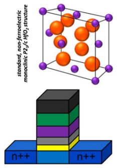
Canonical Microcircuits Laminar and Columnar 3D Organization

Mixed Mode Charge Based Circuits



Nano and 3D





BRAINS

BRAINWAY

Learning – Adaptation – Self-organization

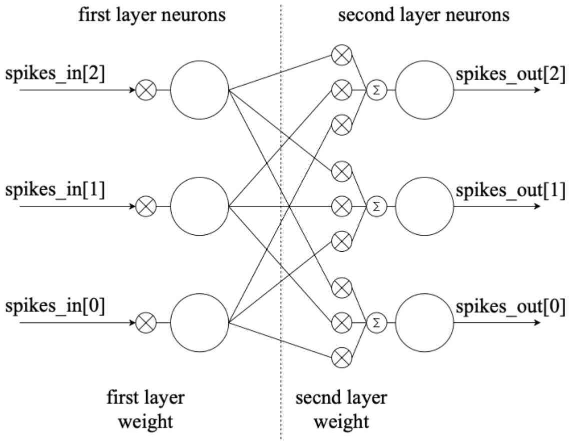
Cassidy, A. S., Georgiou, J., & Andreou, A. G. (2013). Design of silicon brains in the nano-CMOS era: spiking neurons, learning synapses and neural architecture optimization. *Neural Networks*, *45*, 4–26. http://doi.org/10.1016/j.neunet.2013.05.011

Example 1: Spiking Neural Network

Hello, could you please provide me a **Verilog module** describing Spiking Neeural Network

Is it even possible?

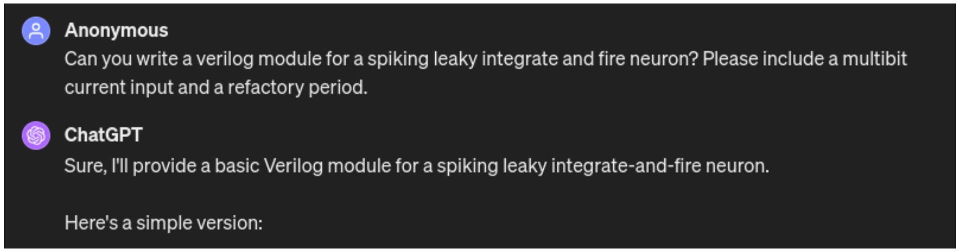
ChatGPT



Topic Search Results from Github

Search Term	Language	Number of Results
"integrate and fire"	Verilog	3
"integrate and fire"	Any	750
"snn"	Verilog	26
"snn"	Any	2.4k
"spiking neuron"	Verilog	6
"spiking neuron"	Any	478
"spiking"	Verilog	29
"spiking"	Any	8.7k
"alu"	Verilog	1.1k
"accelerator"	Verilog	339
"cnn"	Verilog	198
"spi"	Verilog	530

The Neuron: Prompt



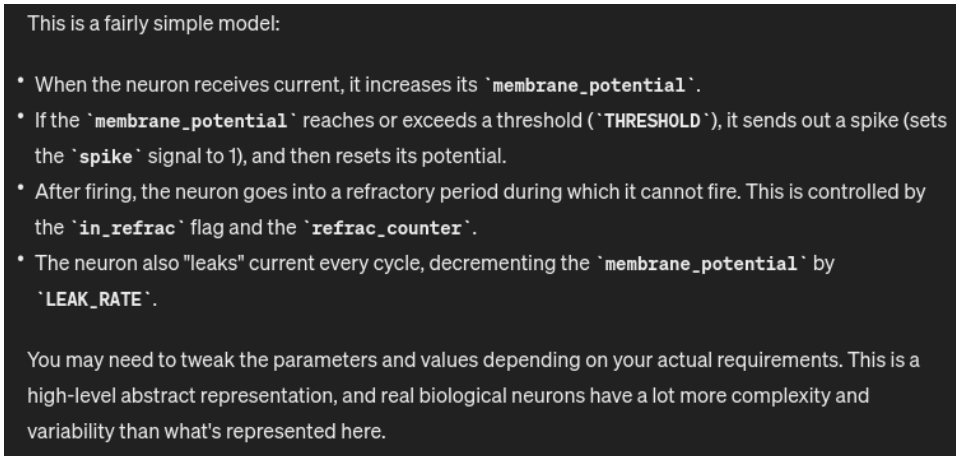
Anonymous

Can you write a verilog module for a spiking leaky integrate and fire neuron? Please include a multibit current input and a refactory period.

ChatGPT

Sure, I'll provide a basic Verilog module for a spiking leaky integrate-and-fire neuron.

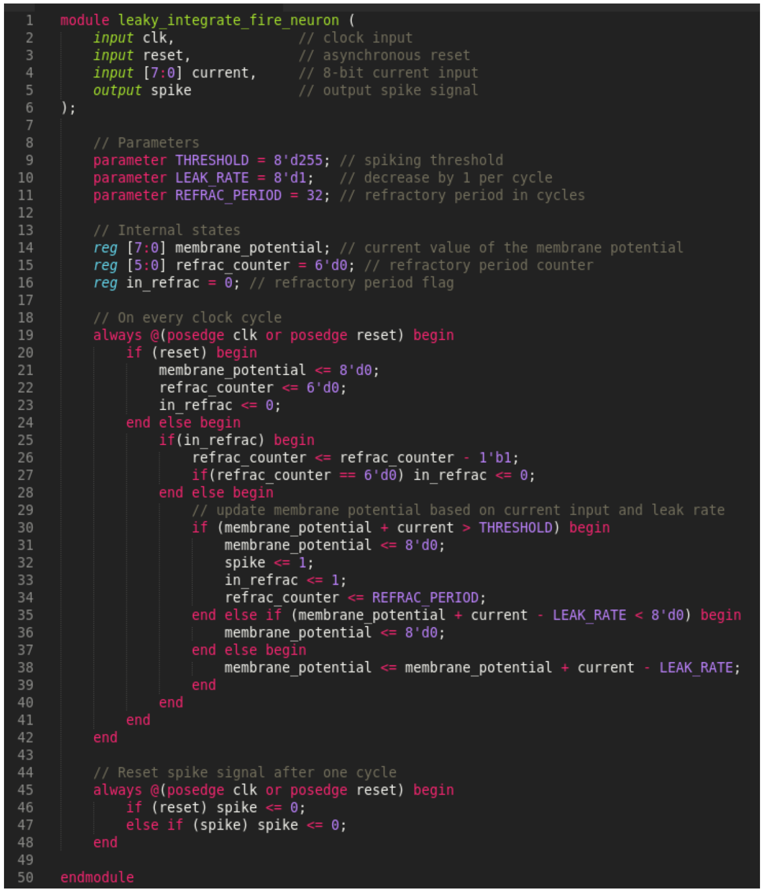
Here's a simple version:



This is a fairly simple model:

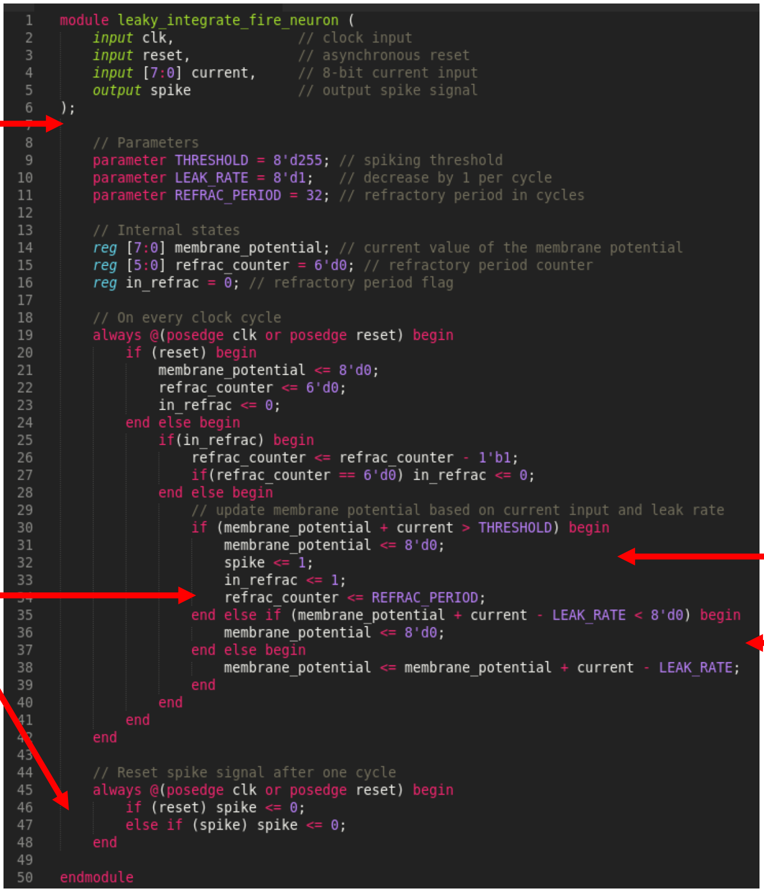
- When the neuron receives current, it increases its `membrane_potential`.
- If the `membrane_potential` reaches or exceeds a threshold (`THRESHOLD`), it sends out a spike (sets the `spike` signal to 1), and then resets its potential.
- After firing, the neuron goes into a refractory period during which it cannot fire. This is controlled by the `in_refrac` flag and the `refrac_counter`.
- The neuron also "leaks" current every cycle, decrementing the `membrane_potential` by `LEAK_RATE`.

You may need to tweak the parameters and values depending on your actual requirements. This is a high-level abstract representation, and real biological neurons have a lot more complexity and variability than what's represented here.



```verilog
module leaky_integrate_fire_neuron (
    input clk,               // clock input
    input reset,             // asynchronous reset
    input [7:0] current,     // 8-bit current input
    output spike             // output spike signal
);

    // Parameters
    parameter THRESHOLD = 8'd255; // spiking threshold
    parameter LEAK_RATE = 8'd1;   // decrease by 1 per cycle
    parameter REFRAC_PERIOD = 32; // refractory period in cycles

    // Internal states
    reg [7:0] membrane_potential; // current value of the membrane potential
    reg [5:0] refrac_counter = 6'd0; // refractory period counter
    reg in_refrac = 0; // refractory period flag

    // On every clock cycle
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            membrane_potential <= 8'd0;
            refrac_counter <= 6'd0;
            in_refrac <= 0;
        end else begin
            if(in_refrac) begin
                refrac_counter <= refrac_counter - 1'b1;
                if(refrac_counter == 6'd0) in_refrac <= 0;
            end else begin
                // update membrane potential based on current input and leak rate
                if (membrane_potential + current > THRESHOLD) begin
                    membrane_potential <= 8'd0;
                    spike <= 1;
                    in_refrac <= 1;
                    refrac_counter <= REFRAC_PERIOD;
                end else if (membrane_potential + current - LEAK_RATE < 8'd0) begin
                    membrane_potential <= 8'd0;
                end else begin
                    membrane_potential <= membrane_potential + current - LEAK_RATE;
                end
            end
        end
    end

    // Reset spike signal after one cycle
    always @(posedge clk or posedge reset) begin
        if (reset) spike <= 0;
        else if (spike) spike <= 0;
    end

endmodule
```

Neuron Verilog Code



```
 1  module leaky_integrate_fire_neuron (
 2      input clk,               // clock input
 3      input reset,             // asynchronous reset
 4      input [7:0] current,     // 8-bit current input
 5      output spike             // output spike signal
 6  );
 7
 8      // Parameters
 9      parameter THRESHOLD = 8'd255; // spiking threshold
10      parameter LEAK_RATE = 8'd1;   // decrease by 1 per cycle
11      parameter REFRAC_PERIOD = 32; // refractory period in cycles
12
13      // Internal states
14      reg [7:0] membrane_potential; // current value of the membrane potential
15      reg [5:0] refrac_counter = 6'd0; // refractory period counter
16      reg in_refrac = 0; // refractory period flag
17
18      // On every clock cycle
19      always @(posedge clk or posedge reset) begin
20          if (reset) begin
21              membrane_potential <= 8'd0;
22              refrac_counter <= 6'd0;
23              in_refrac <= 0;
24          end else begin
25              if(in_refrac) begin
26                  refrac_counter <= refrac_counter - 1'b1;
27                  if(refrac_counter == 6'd0) in_refrac <= 0;
28              end else begin
29                  // update membrane potential based on current input and leak rate
30                  if (membrane_potential + current > THRESHOLD) begin
31                      membrane_potential <= 8'd0;
32                      spike <= 1;
33                      in_refrac <= 1;
34                      refrac_counter <= REFRAC_PERIOD;
35                  end else if (membrane_potential + current - LEAK_RATE < 8'd0) begin
36                      membrane_potential <= 8'd0;
37                  end else begin
38                      membrane_potential <= membrane_potential + current - LEAK_RATE;
39                  end
40              end
41          end
42      end
43
44      // Reset spike signal after one cycle
45      always @(posedge clk or posedge reset) begin
46          if (reset) spike <= 0;
47          else if (spike) spike <= 0;
48      end
49
50  endmodule
```

Missing reg declaration for spike
(used as a reg on line 32)

Output reg spike is multiply driven

Overflow is not handled

Underflow is not handled

Syntax Errors

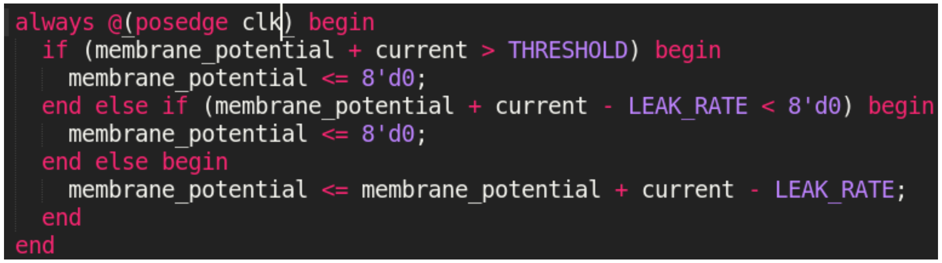
Invalid code

Logic Errors

Valid code, functionally incorrect

Neuron Verilog Code - Analysis

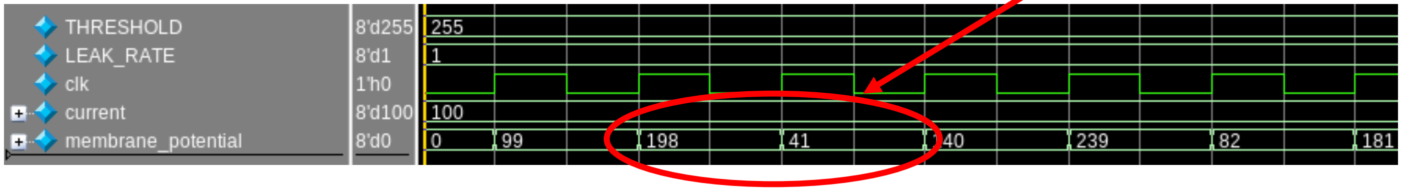
Membrane potential
code snippet



```verilog
always @(posedge clk) begin
  if (membrane_potential + current > THRESHOLD) begin
    membrane_potential <= 8'd0;
  end else if (membrane_potential + current - LEAK_RATE < 8'd0) begin
    membrane_potential <= 8'd0;
  end else begin
    membrane_potential <= membrane_potential + current - LEAK_RATE;
  end
end
```

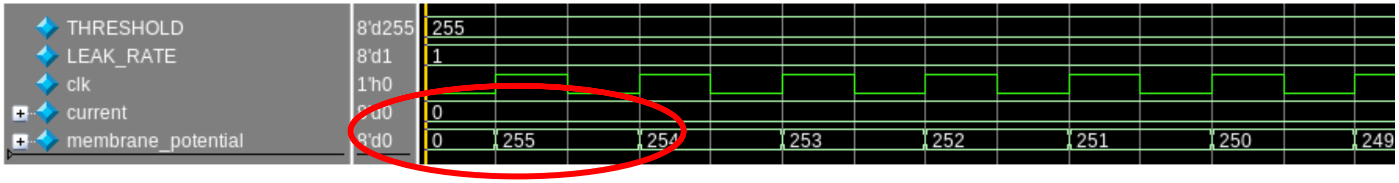
Should be set to 0
when overflow occurs

Overflow Error

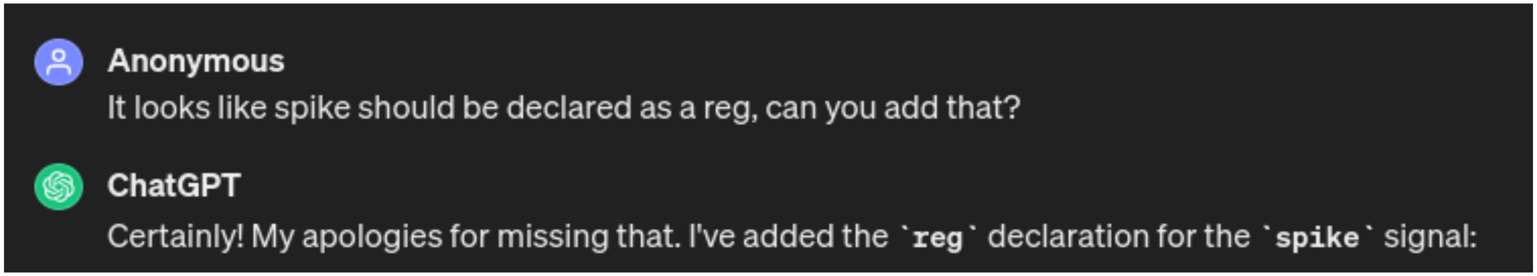


0 current results in the
membrane potential rolling over!

Underflow Error



Fixing the errors using Natural Language (I)

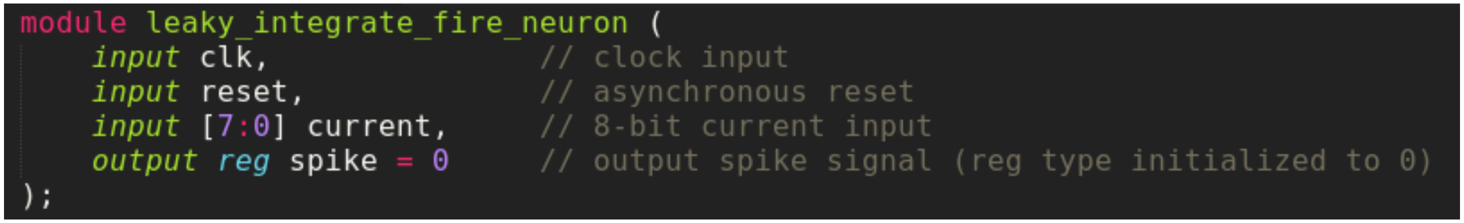


Anonymous

It looks like spike should be declared as a reg, can you add that?

ChatGPT

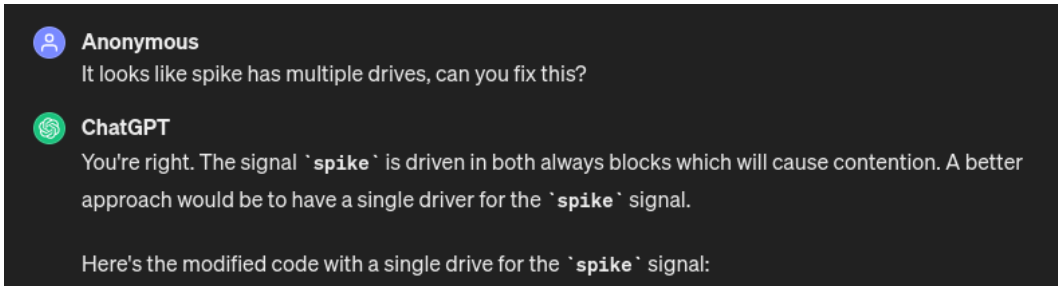
Certainly! My apologies for missing that. I've added the `reg` declaration for the `spike` signal:



```
module leaky_integrate_fire_neuron (
    input clk,                  // clock input
    input reset,                // asynchronous reset
    input [7:0] current,        // 8-bit current input
    output reg spike = 0        // output spike signal (reg type initialized to 0)
);
```

✓

Fixing the errors using Natural Language (II)



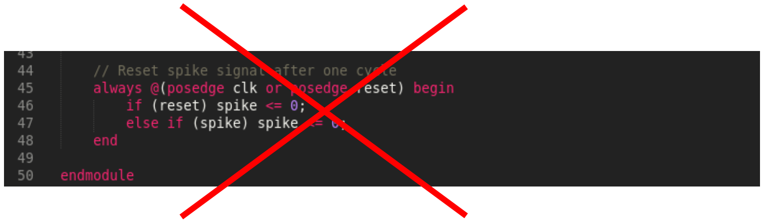
Anonymous

It looks like spike has multiple drives, can you fix this?

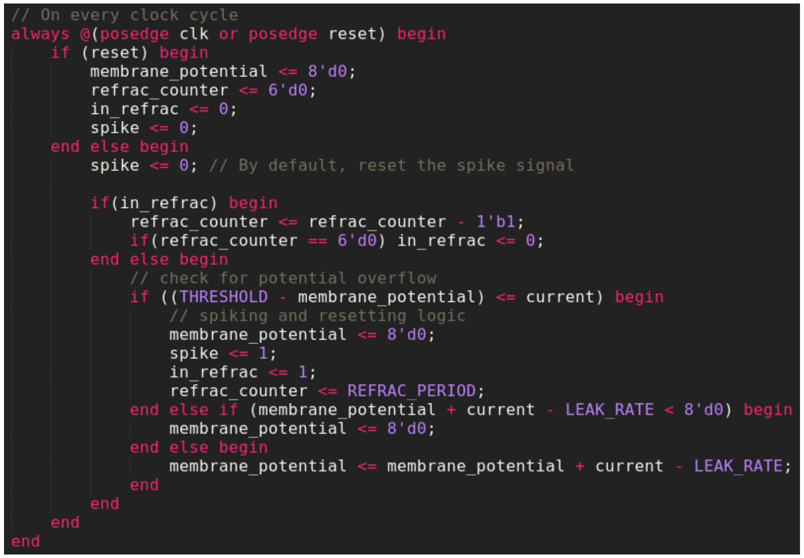
ChatGPT

You're right. The signal `spike` is driven in both always blocks which will cause contention. A better approach would be to have a single driver for the `spike` signal.

Here's the modified code with a single drive for the `spike` signal:

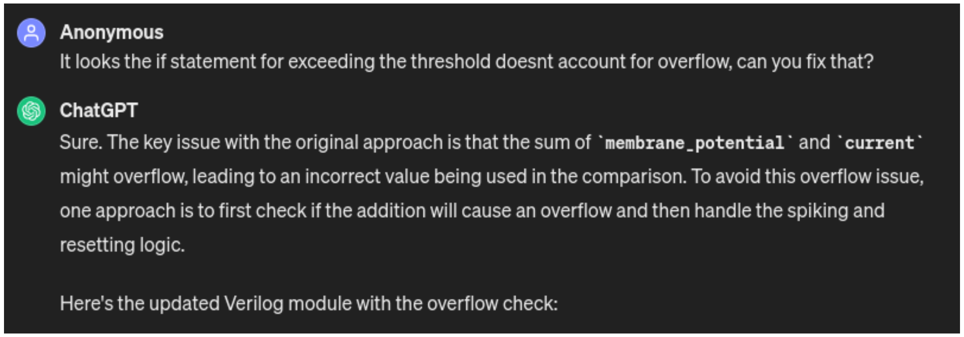


```
    // Reset spike signal after one cycle
    always @(posedge clk or posedge reset) begin
        if (reset) spike <= 0;
        else if (spike) spike <= 0;
    end

endmodule
```



```
// On every clock cycle
always @(posedge clk or posedge reset) begin
    if (reset) begin
        membrane_potential <= 8'd0;
        refrac_counter <= 6'd0;
        in_refrac <= 0;
        spike <= 0;
    end else begin
        spike <= 0; // By default, reset the spike signal

        if(in_refrac) begin
            refrac_counter <= refrac_counter - 1'b1;
            if(refrac_counter == 6'd0) in_refrac <= 0;
        end else begin
            // check for potential overflow
            if ((THRESHOLD - membrane_potential) <= current) begin
                // spiking and resetting logic
                membrane_potential <= 8'd0;
                spike <= 1;
                in_refrac <= 1;
                refrac_counter <= REFRAC_PERIOD;
            end else if (membrane_potential + current - LEAK_RATE < 8'd0) begin
                membrane_potential <= 8'd0;
            end else begin
                membrane_potential <= membrane_potential + current - LEAK_RATE;
            end
        end
    end
end
```

Fixing the errors using Natural Language (III)



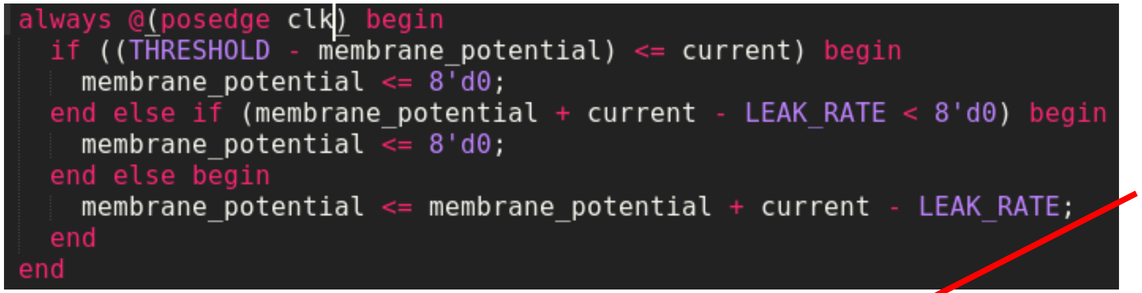
Anonymous

It looks the if statement for exceeding the threshold doesnt account for overflow, can you fix that?

ChatGPT

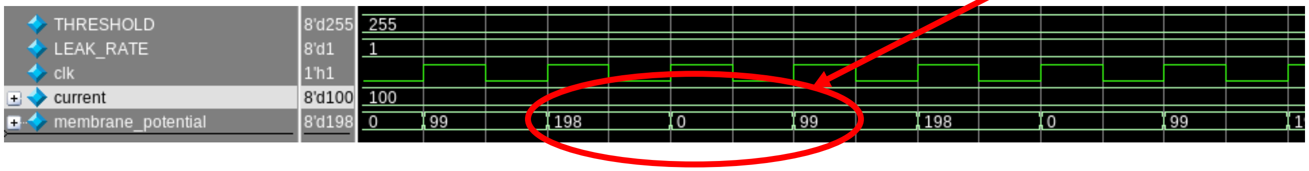
Sure. The key issue with the original approach is that the sum of `membrane_potential` and `current` might overflow, leading to an incorrect value being used in the comparison. To avoid this overflow issue, one approach is to first check if the addition will cause an overflow and then handle the spiking and resetting logic.

Here's the updated Verilog module with the overflow check:



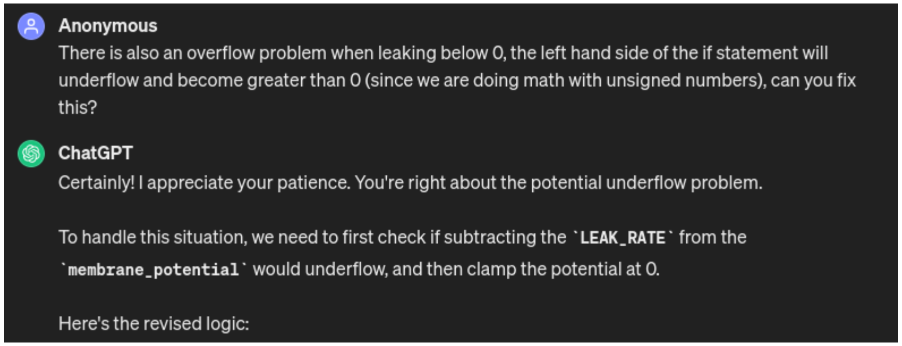
```
always @(posedge clk) begin
  if ((THRESHOLD - membrane_potential) <= current) begin
    membrane_potential <= 8'd0;
  end else if (membrane_potential + current - LEAK_RATE < 8'd0) begin
    membrane_potential <= 8'd0;
  end else begin
    membrane_potential <= membrane_potential + current - LEAK_RATE;
  end
end
```

Once the next value would exceed the threshold, it gets reset to 0



✓

Arithmetic Problem -Underflow Issue (I)



Anonymous

There is also an overflow problem when leaking below 0, the left hand side of the if statement will underflow and become greater than 0 (since we are doing math with unsigned numbers), can you fix this?

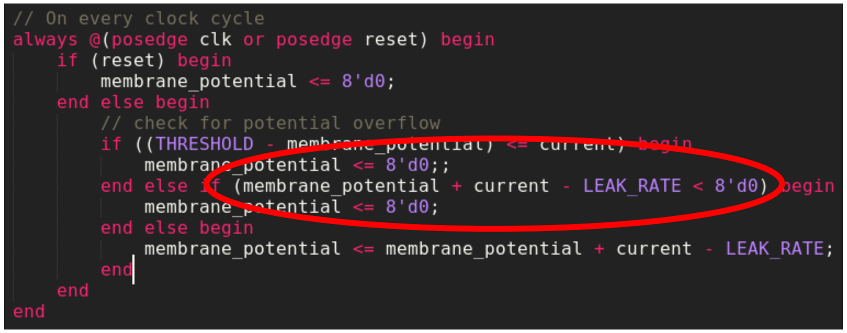
ChatGPT

Certainly! I appreciate your patience. You're right about the potential underflow problem.

To handle this situation, we need to first check if subtracting the `LEAK_RATE` from the `membrane_potential` would underflow, and then clamp the potential at 0.

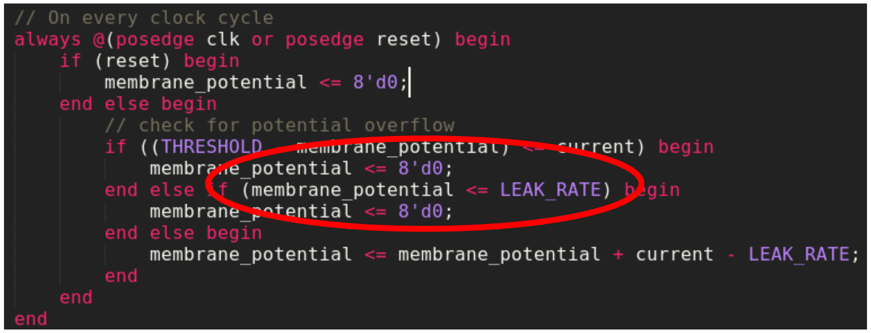
Here's the revised logic:

If membrane_potential = 0, the else clause will never execute!
membrane_potential gets stuck at 0



```
// On every clock cycle
always @(posedge clk or posedge reset) begin
    if (reset) begin
        membrane_potential <= 8'd0;
    end else begin
        // check for potential overflow
        if ((THRESHOLD - membrane_potential) <= current) begin
            membrane_potential <= 8'd0;;
        end else if (membrane_potential + current - LEAK_RATE < 8'd0) begin
            membrane_potential <= 8'd0;
        end else begin
            membrane_potential <= membrane_potential + current - LEAK_RATE;
        end
    end
end
```

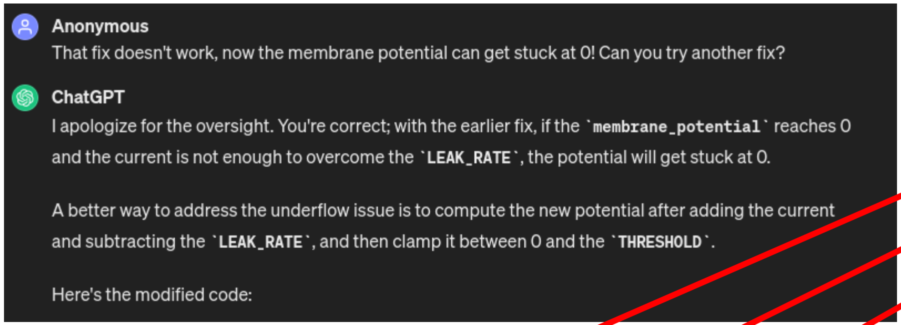
Previous Output



```
// On every clock cycle
always @(posedge clk or posedge reset) begin
    if (reset) begin
        membrane_potential <= 8'd0;
    end else begin
        // check for potential overflow
        if ((THRESHOLD - membrane_potential) <= current) begin
            membrane_potential <= 8'd0;
        end else if (membrane_potential <= LEAK_RATE) begin
            membrane_potential <= 8'd0;
        end else begin
            membrane_potential <= membrane_potential + current - LEAK_RATE;
        end
    end
end
```

New Output

Arithmetic Problem -Underflow Issue (II)



Anonymous

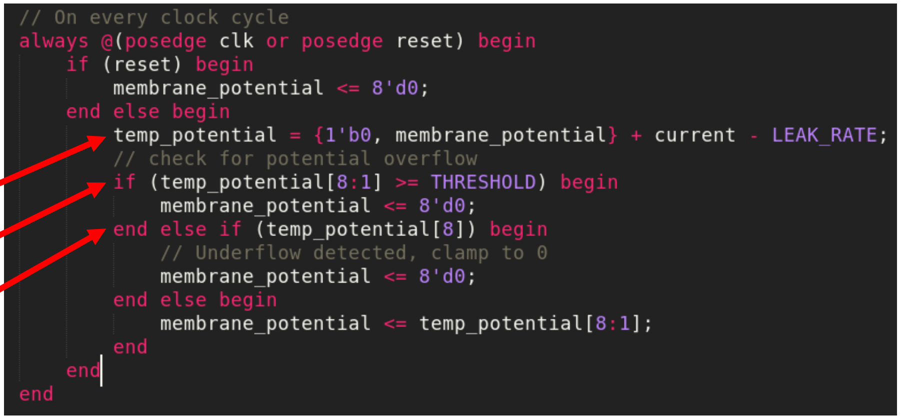
That fix doesn't work, now the membrane potential can get stuck at 0! Can you try another fix?

ChatGPT

I apologize for the oversight. You're correct; with the earlier fix, if the `membrane_potential` reaches 0 and the current is not enough to overcome the `LEAK_RATE`, the potential will get stuck at 0.

A better way to address the underflow issue is to compute the new potential after adding the current and subtracting the `LEAK_RATE`, and then clamp it between 0 and the `THRESHOLD`.

Here's the modified code:



```
// On every clock cycle
always @(posedge clk or posedge reset) begin
    if (reset) begin
        membrane_potential <= 8'd0;
    end else begin
        temp_potential = {1'b0, membrane_potential} + current - LEAK_RATE;
        // check for potential overflow
        if (temp_potential[8:1] >= THRESHOLD) begin
            membrane_potential <= 8'd0;
        end else if (temp_potential[8]) begin
            // Underflow detected, clamp to 0
            membrane_potential <= 8'd0;
        end else begin
            membrane_potential <= temp_potential[8:1];
        end
    end
end
```

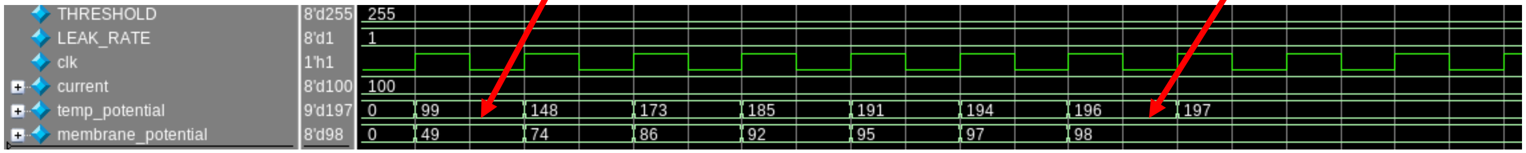
Tries to use a temp variable for overflow protection

Compares threshold with upper 8 bits of temp_potential??

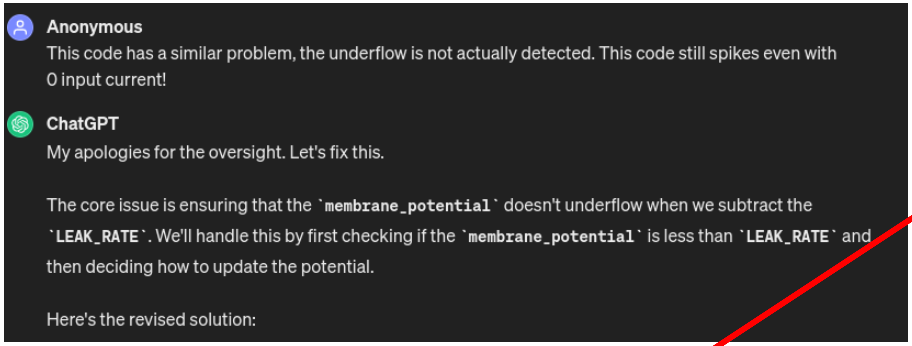
Checks msb to detect underflow, reg is **not** declared as signed

Increments are distorted due to bit select errors

Potential gets stuck at an intermediate value



Arithmetic Problem -Underflow Issue (III)



Anonymous

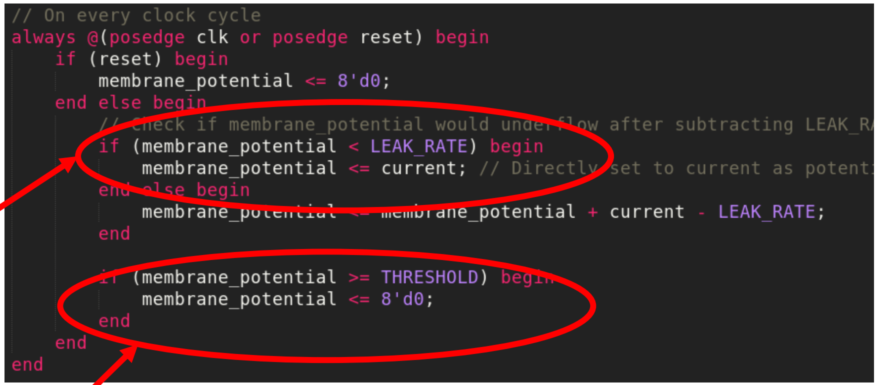
This code has a similar problem, the underflow is not actually detected. This code still spikes even with 0 input current!

ChatGPT

My apologies for the oversight. Let's fix this.

The core issue is ensuring that the `membrane_potential` doesn't underflow when we subtract the `LEAK_RATE`. We'll handle this by first checking if the `membrane_potential` is less than `LEAK_RATE` and then deciding how to update the potential.

Here's the revised solution:



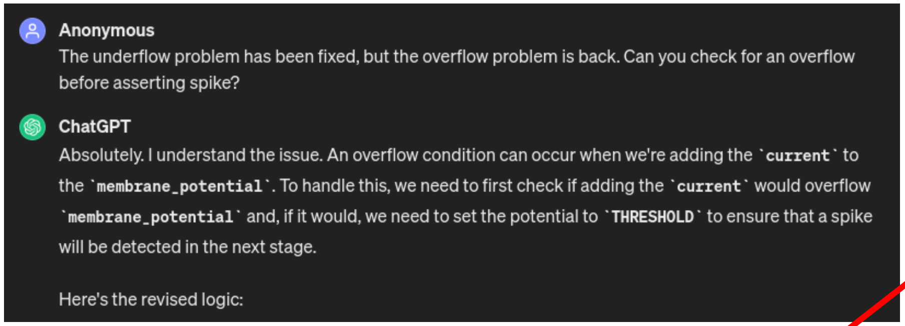
```
// On every clock cycle
always @(posedge clk or posedge reset) begin
    if (reset) begin
        membrane_potential <= 8'd0;
    end else begin
        // Check if membrane_potential would underflow after subtracting LEAK_RA
        if (membrane_potential < LEAK_RATE) begin
            membrane_potential <= current; // Directly set to current as potent
        end else begin
            membrane_potential <= membrane_potential + current - LEAK_RATE;
        end

        if (membrane_potential >= THRESHOLD) begin
            membrane_potential <= 8'd0;
        end
    end
end
```

Underflow gets handled correctly

Overflow problem reintroduced

✗

Arithmetic Problem -Underflow Issue Solved!



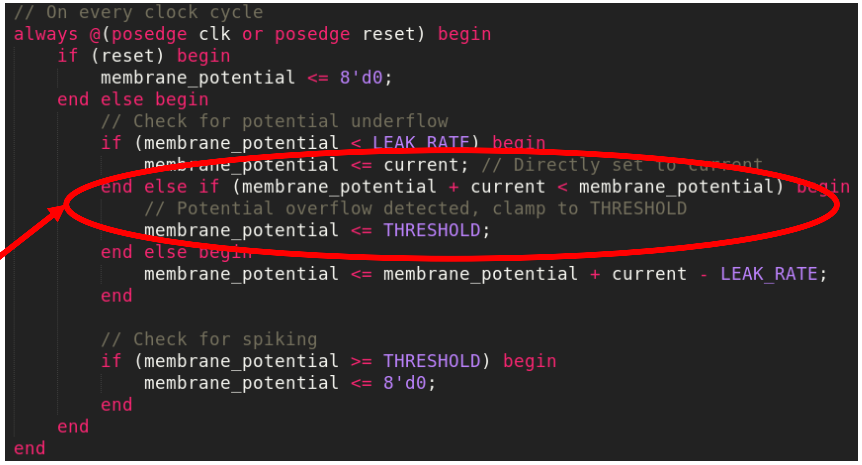
Anonymous

The underflow problem has been fixed, but the overflow problem is back. Can you check for an overflow before asserting spike?

ChatGPT

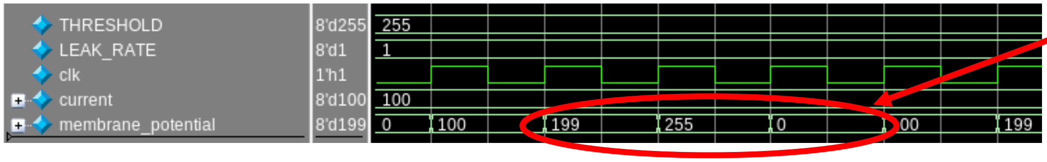
Absolutely. I understand the issue. An overflow condition can occur when we're adding the `current` to the `membrane_potential`. To handle this, we need to first check if adding the `current` would overflow `membrane_potential` and, if it would, we need to set the potential to `THRESHOLD` to ensure that a spike will be detected in the next stage.

Here's the revised logic:

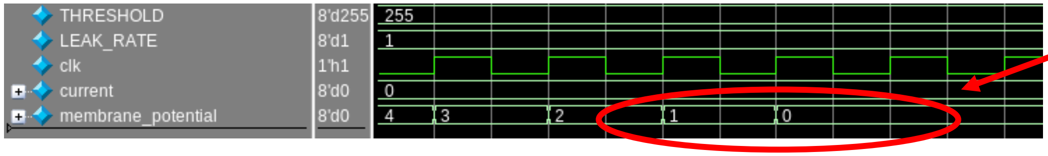


```
// On every clock cycle
always @(posedge clk or posedge reset) begin
    if (reset) begin
        membrane_potential <= 8'd0;
    end else begin
        // Check for potential underflow
        if (membrane_potential < LEAK_RATE) begin
            membrane_potential <= current; // Directly set to current
        end else if (membrane_potential + current < membrane_potential) begin
            // Potential overflow detected, clamp to THRESHOLD
            membrane_potential <= THRESHOLD;
        end else begin
            membrane_potential <= membrane_potential + current - LEAK_RATE;
        end

        // Check for spiking
        if (membrane_potential >= THRESHOLD) begin
            membrane_potential <= 8'd0;
        end
    end
end
```

Overflow detection

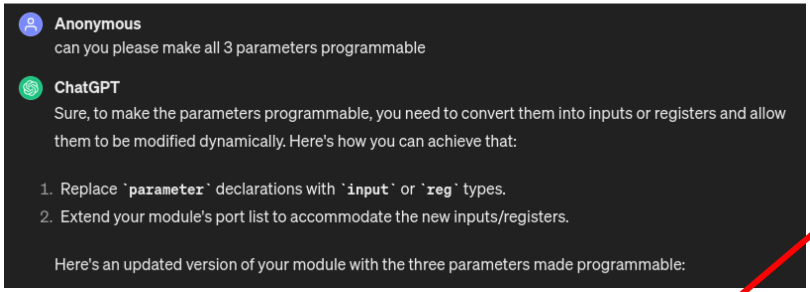


Takes an extra cycle due to clamping, but overflow is handled



Underflow is handled, no rollover occurs

Making the Neuron Programmable



Anonymous

can you please make all 3 parameters programmable

ChatGPT

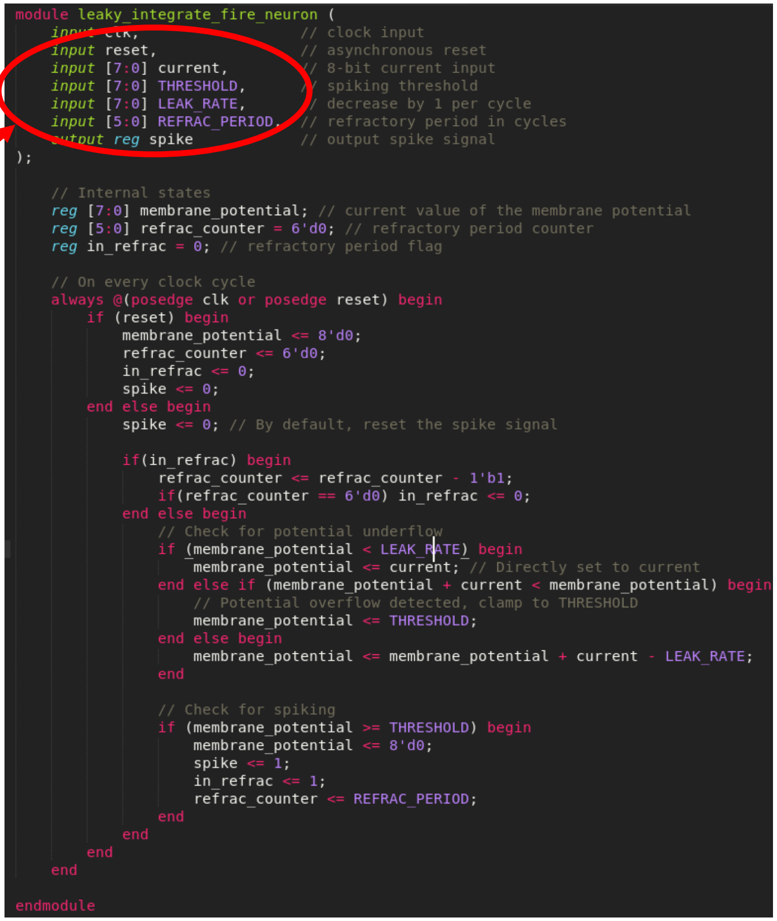
Sure, to make the parameters programmable, you need to convert them into inputs or registers and allow them to be modified dynamically. Here's how you can achieve that:

1. Replace `parameter` declarations with `input` or `reg` types.
2. Extend your module's port list to accommodate the new inputs/registers.

Here's an updated version of your module with the three parameters made programmable:

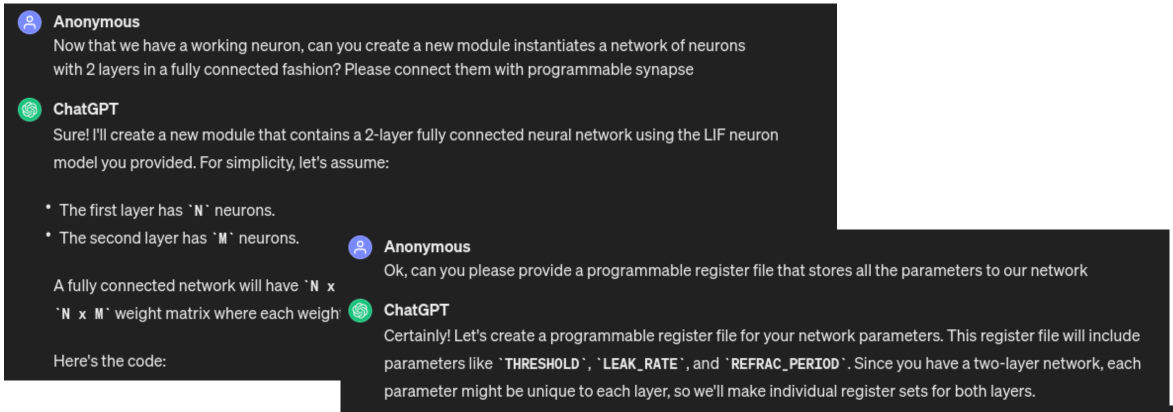
Internal parameters converted to ports for programmability

✓



```verilog
module leaky_integrate_fire_neuron (
    input clk,                  // clock input
    input reset,                // asynchronous reset
    input [7:0] current,        // 8-bit current input
    input [7:0] THRESHOLD,      // spiking threshold
    input [7:0] LEAK_RATE,      // decrease by 1 per cycle
    input [5:0] REFRAC_PERIOD,  // refractory period in cycles
    output reg spike            // output spike signal
);

    // Internal states
    reg [7:0] membrane_potential; // current value of the membrane potential
    reg [5:0] refrac_counter = 6'd0; // refractory period counter
    reg in_refrac = 0; // refractory period flag

    // On every clock cycle
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            membrane_potential <= 8'd0;
            refrac_counter <= 6'd0;
            in_refrac <= 0;
            spike <= 0;
        end else begin
            spike <= 0; // By default, reset the spike signal

            if(in_refrac) begin
                refrac_counter <= refrac_counter - 1'b1;
                if(refrac_counter == 6'd0) in_refrac <= 0;
            end else begin
                // Check for potential underflow
                if (membrane_potential < LEAK_RATE) begin
                    membrane_potential <= current; // Directly set to current
                end else if (membrane_potential + current < membrane_potential) begin
                    // Potential overflow detected, clamp to THRESHOLD
                    membrane_potential <= THRESHOLD;
                end else begin
                    membrane_potential <= membrane_potential + current - LEAK_RATE;
                end

                // Check for spiking
                if (membrane_potential >= THRESHOLD) begin
                    membrane_potential <= 8'd0;
                    spike <= 1;
                    in_refrac <= 1;
                    refrac_counter <= REFRAC_PERIOD;
                end
            end
        end
    end

endmodule
```

Building the rest of the system



Anonymous
Now that we have a working neuron, can you create a new module instantiates a network of neurons with 2 layers in a fully connected fashion? Please connect them with programmable synapse

ChatGPT
Sure! I'll create a new module that contains a 2-layer fully connected neural network using the LIF neuron model you provided. For simplicity, let's assume:

- The first layer has `N` neurons.
- The second layer has `M` neurons.

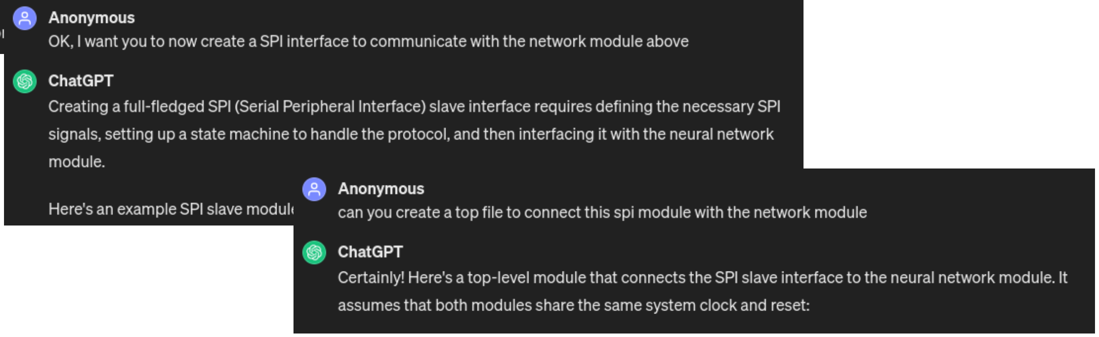
A fully connected network will have `N x
`N x M` weight matrix where each weigh

Here's the code:

Anonymous
Ok, can you please provide a programmable register file that stores all the parameters to our network

ChatGPT
Certainly! Let's create a programmable register file for your network parameters. This register file will include parameters like `THRESHOLD`, `LEAK_RATE`, and `REFRAC_PERIOD`. Since you have a two-layer network, each parameter might be unique to each layer, so we'll make individual register sets for both layers.

Here's a basic programmable register file fo



Anonymous
OK, I want you to now create a SPI interface to communicate with the network module above

ChatGPT
Creating a full-fledged SPI (Serial Peripheral Interface) slave interface requires defining the necessary SPI signals, setting up a state machine to handle the protocol, and then interfacing it with the neural network module.

Here's an example SPI slave modul

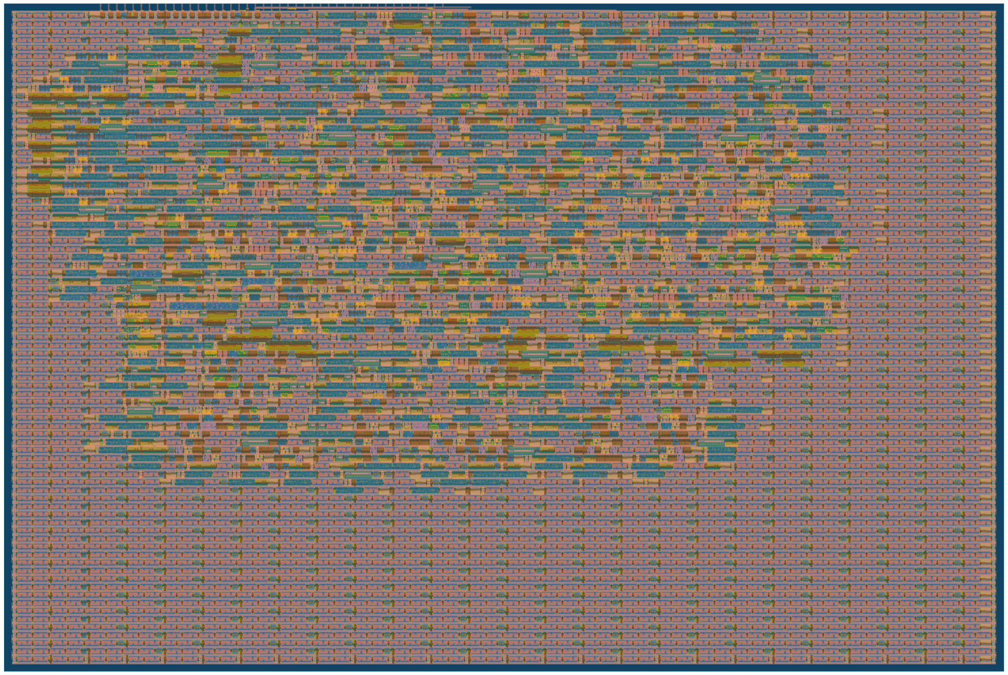
Anonymous
can you create a top file to connect this spi module with the network module

ChatGPT
Certainly! Here's a top-level module that connects the SPI slave interface to the neural network module. It assumes that both modules share the same system clock and reset:

Physical Layout and submission to fabrication

- Fully LLM written SPI programmable Spiking Neuron Array
- Generated verilog verified with functional simulation
- Design was submitted on Tiny Tapeout 5 using an open source RTL-> GDS flow
- 33% Density in a 320um x 200um tile
- All code, scripts, and ChatGPT transcripts are available on Github

- One of the first fully AI described ASICs
- Not an easy process, but a fully Natural Language -> Verilog flow is possible with extensive manual (human) verification



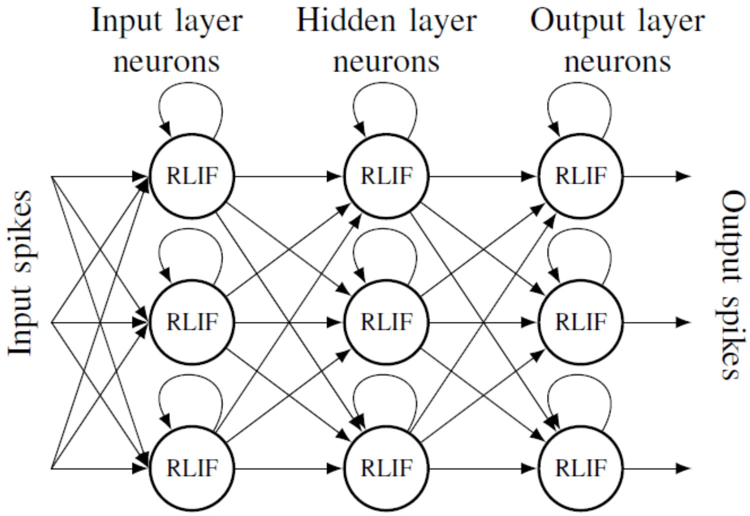
Layout for LLM Spiking Neuron Array

Spiking Neural Network chip: -Telluride Workshop on Neuromorphic Engineering, July, 2023, ChatGPT 4, Efabless Tiny Tapeout 5, 4 November 2023, Chips and boards back June 2024 –now-

Recurrent Spiking Neural Network

Hello, could you please provide me a Verilog module describing a Recurrent Spiking Neural Network **and a Test Bench** to test it?

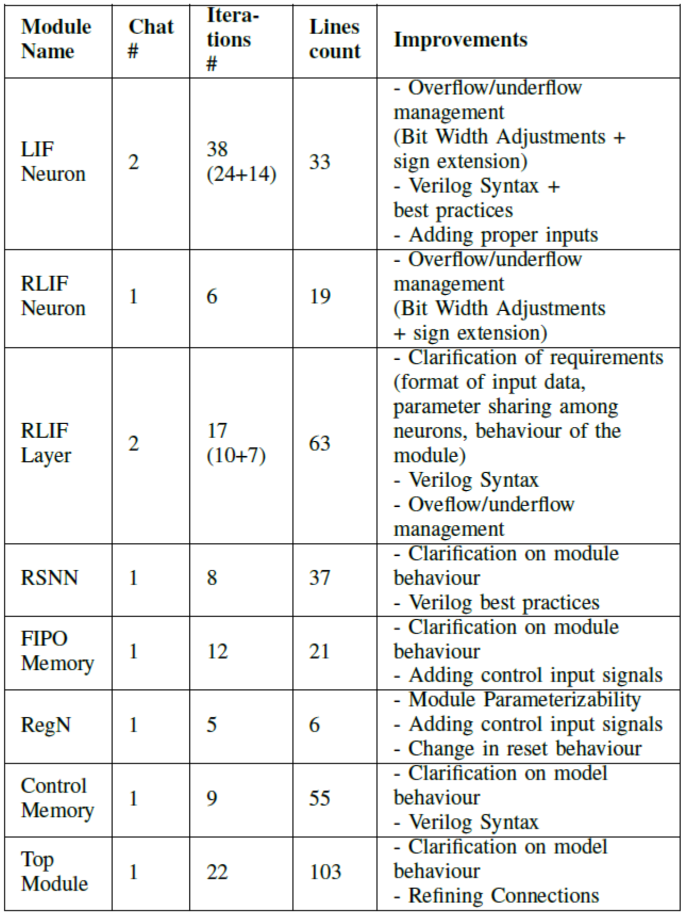
ChatGPT



Recurrent Spiking Neural Network chip: -2024 Andreou Lab, ChatGPT 4, April-May 2024, Efabless Tiny Tapeout 6, 15 May 2024

Recurrent Neural Network Results

- Fully LLM written SPI programmable Recurrent Spiking Neuron Array.
- Test-benches generated by ChatGPT
- Generated Verilog verified with functional simulation and mapping into an FPGA.
- Network parameters trained and tested for XOR and IRIS classification task
- Chip design submitted on Efabless Tiny Tapeout 6 using an open source RTL-> GDS flow
- All code, scripts, and ChatGPT transcripts are available on Github



Module Name	**Chat #**	**Itera-tions #**	**Lines count**	**Improvements**
LIF Neuron	2	38 (24+14)	33	- Overflow/underflow management (Bit Width Adjustments + sign extension) - Verilog Syntax + best practices - Adding proper inputs
RLIF Neuron	1	6	19	- Overflow/underflow management (Bit Width Adjustments + sign extension)
RLIF Layer	2	17 (10+7)	63	- Clarification of requirements (format of input data, parameter sharing among neurons, behaviour of the module) - Verilog Syntax - Oveflow/underflow management
RSNN	1	8	37	- Clarification on module behaviour - Verilog best practices
FIPO Memory	1	12	21	- Clarification on module behaviour - Adding control input signals
RegN	1	5	6	- Module Parameterizability - Adding control input signals - Change in reset behaviour
Control Memory	1	9	55	- Clarification on model behaviour - Verilog Syntax
Top Module	1	22	103	- Clarification on model behaviour - Refining Connections

Bibliography

2023: The year of LLM based EDA

Kaiyan Chang, Ying Wang, Haimeng Ren, Mengdi Wang, Shengwen Liang, Yinhe Han, Huawei Li, and Xiaowei Li. 2023. ChipGPT: How far are we from natural language hardware design. http://arxiv.org/abs/2305.14019 arXiv:2305.14019 [cs]

Blocklove, Jason, Siddharth Garg, Ramesh Karri, and Hammond Pearce. "Chip-Chat: Challenges and Opportunities in Conversational Hardware Design." In *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, 1–6, 2023. https://doi.org/10.1109/MLCAD58807.2023.10299874.

Meech, James T. "Leveraging High-Level Synthesis and Large Language Models to Generate, Simulate, and Deploy a Uniform Random Number Generator Hardware Design." *arXiv:2311.03489 [Cs]*, November 2023. http://arxiv.org/abs/2311.03489.

Thakur, Shailja, Jason Blocklove, Hammond Pearce, Benjamin Tan, Siddharth Garg, and Ramesh Karri. "AutoChip: Automating HDL Generation Using LLM Feedback." arXiv, November 8, 2023. http://arxiv.org/abs/2311.04887.

Tomlinson, Michael A, Joe Li, and Andreas G. Andreou. "Designing Silicon Brains Using LLMs: Leveraging ChatGPT for Automated Description of a Spiking Neuron Array." In Proceedings 2024 Argentine Conference on Electronics, 154–59. IEEE, March 2024.

Vitolo, Paola, George Psaltakis, Michael Tomlinson, Gian Domenico Licciardo, and Andreas G. Andreou. "Natural Language to Verilog: Design of a Recurrent Spiking Neural Network Using Large Language Models and ChatGPT." arXiv:2405.01419, May 2, 2024. http://arxiv.org/abs/2405.01419.

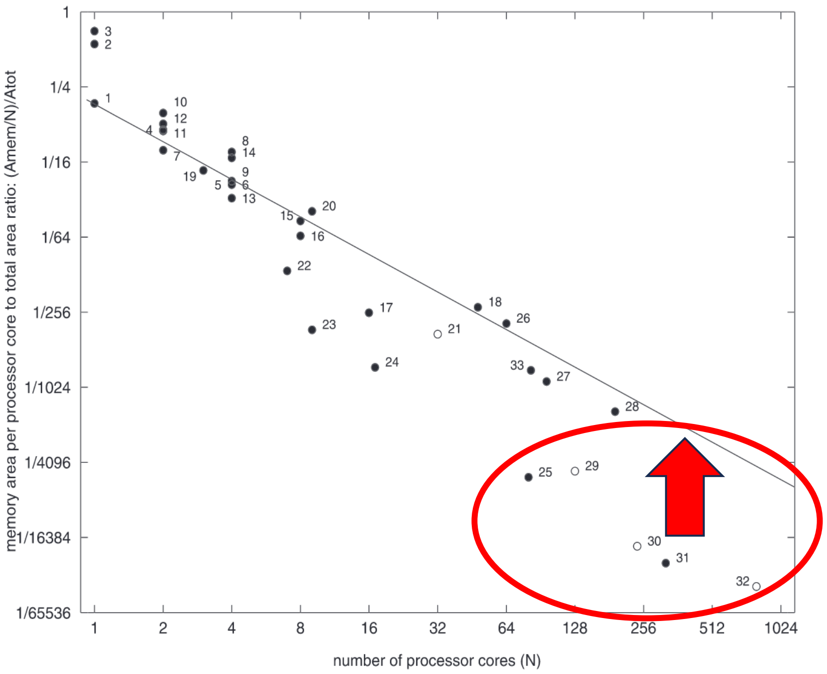
Generative AI acceleration: Neuromorphic Computational Memory Systems for Accelerating LLMs

The role of memory architecture in AI and the challenge of a 10 GiBits per square mm computational memory for Generative AI

2.5D and 3D Integration impact in AI Systems

IEEE TRANSACTIONS ON COMPUTERS, VOL. 61,

CASSIDY AND ANDREOU: BEYOND AMDAHL'S LAW: AN OBJECTIVE FUNCTIO



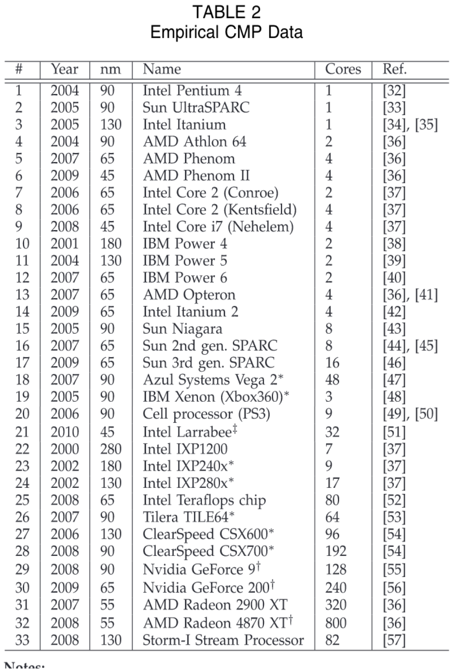
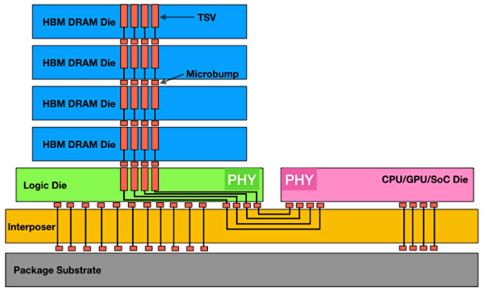


TABLE 2
Empirical CMP Data

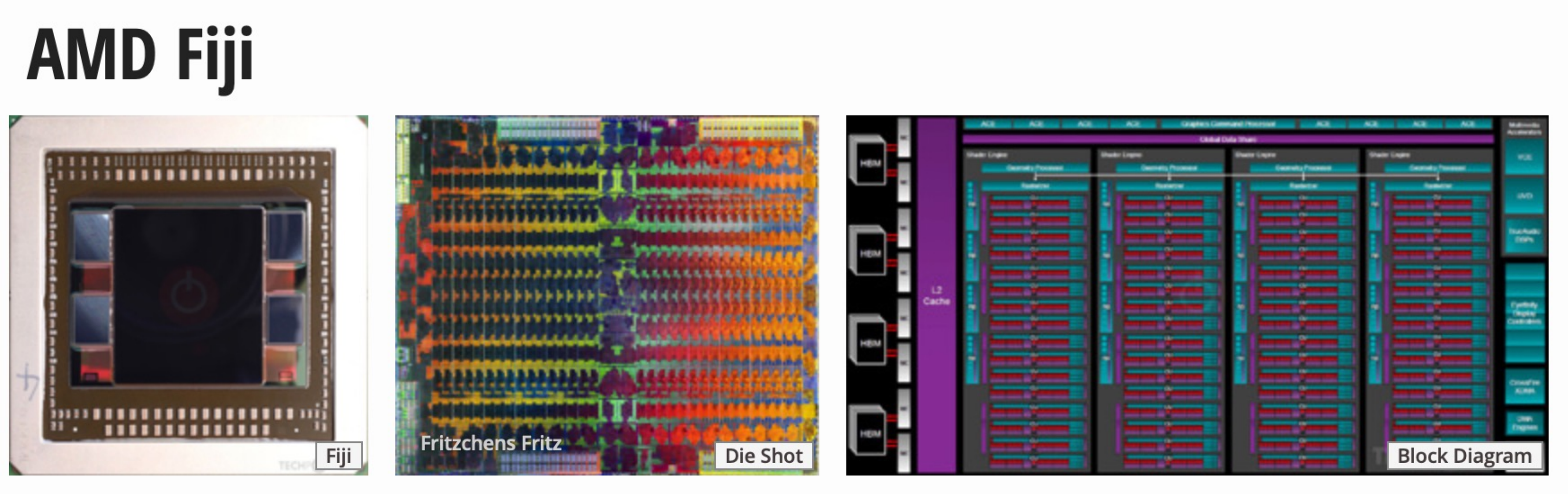
#	Year	nm	Name	Cores	Ref.
1	2004	90	Intel Pentium 4	1	[32]
2	2005	90	Sun UltraSPARC	1	[33]
3	2005	130	Intel Itanium	1	[34], [35]
4	2004	90	AMD Athlon 64	2	[36]
5	2007	65	AMD Phenom	4	[36]
6	2009	45	AMD Phenom II	4	[36]
7	2006	65	Intel Core 2 (Conroe)	2	[37]
8	2006	65	Intel Core 2 (Kentsfield)	4	[37]
9	2008	45	Intel Core i7 (Nehelem)	4	[37]
10	2001	180	IBM Power 4	2	[38]
11	2004	130	IBM Power 5	2	[39]
12	2007	65	IBM Power 6	2	[40]
13	2007	65	AMD Opteron	4	[36], [41]
14	2009	65	Intel Itanium 2	4	[42]
15	2005	90	Sun Niagara	8	[43]
16	2007	65	Sun 2nd gen. SPARC	8	[44], [45]
17	2009	65	Sun 3rd gen. SPARC	16	[46]
18	2007	90	Azul Systems Vega 2*	48	[47]
19	2005	90	IBM Xenon (Xbox360)*	3	[48]
20	2006	90	Cell processor (PS3)	9	[49], [50]
21	2010	45	Intel Larrabee‡	32	[51]
22	2000	280	Intel IXP1200	7	[37]
23	2002	180	Intel IXP240x*	9	[37]
24	2002	130	Intel IXP280x*	17	[37]
25	2008	65	Intel Teraflops chip	80	[52]
26	2007	90	Tilera TILE64*	64	[53]
27	2006	130	ClearSpeed CSX600*	96	[54]
28	2008	90	ClearSpeed CSX700*	192	[54]
29	2008	90	Nvidia GeForce 9†	128	[55]
30	2009	65	Nvidia GeForce 200†	240	[56]
31	2007	55	AMD Radeon 2900 XT	320	[36]
32	2008	55	AMD Radeon 4870 XT†	800	[36]
33	2008	130	Storm-I Stream Processor	82	[57]



https://semiengineering.com/knowledge_centers/memory/volatile-memory/dynamic-random-access-memory/high-bandwidth-memory/

Cassidy, Andrew S., and Andreas G. Andreou. "Beyond Amdah's Law: An Objective Function That Links Multiprocessor Performance Gains to Delay and Energy." *IEEE Transactions on Computers* 61, no. 8 (August 2012): 1110–26. https://doi.org/10.1109/TC.2011.169.

3D Memory in 2.5D GPU design



https://www.techpowerup.com/gpu-specs/amd-fiji.g774#

First Commercial Product: June 2015 AMD Radeon R9 Fury X

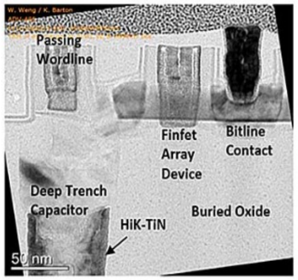
April 2016: Nvidia Tesla P100, June 2016 Intel Xeon-Phi

Jensen Huang: "HBM is a technology miracle …"

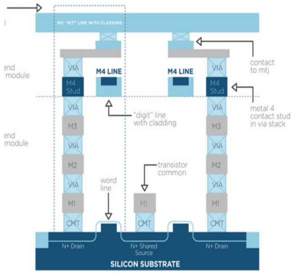
Sanni, Kayode A., and Andreas G. Andreou. "A Historical Perspective on Hardware AI Inference, Charge-Based Computational Circuits and an 8bit Charge-Based Multiply-Add Core in 16nm FinFET CMOS." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, no. 3 (September 2019): 532–43. https://doi.org/10.1109/JETCAS.2019.2933795.

Why 3D?

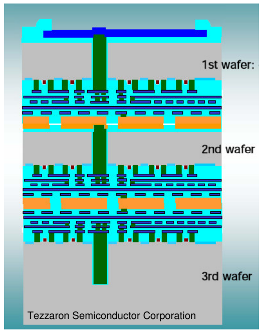
IBM 14nm FinFET
Embedded DRAM



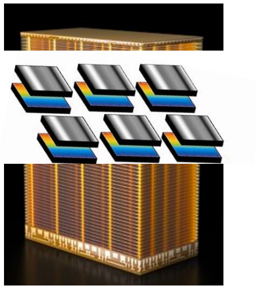
Everspin
Embedded MRAM



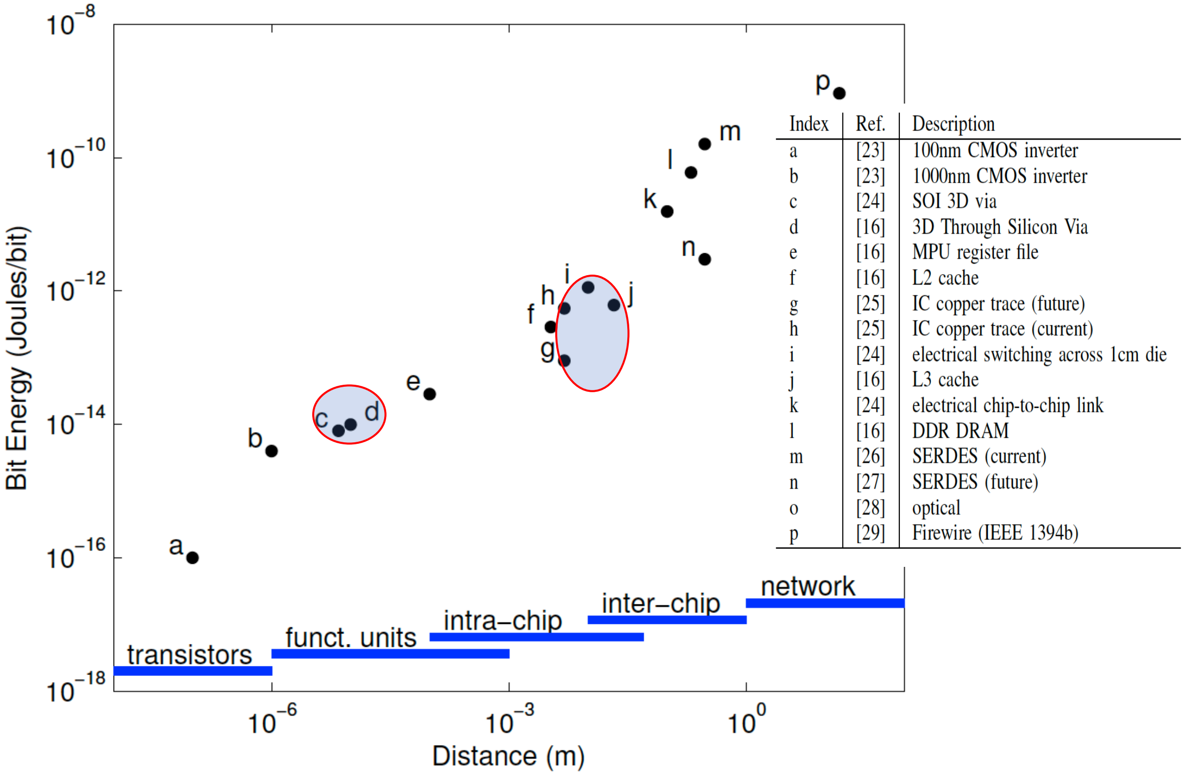
Tezzaron TSVs



6 Planes "columns"



Micron 232L Flash



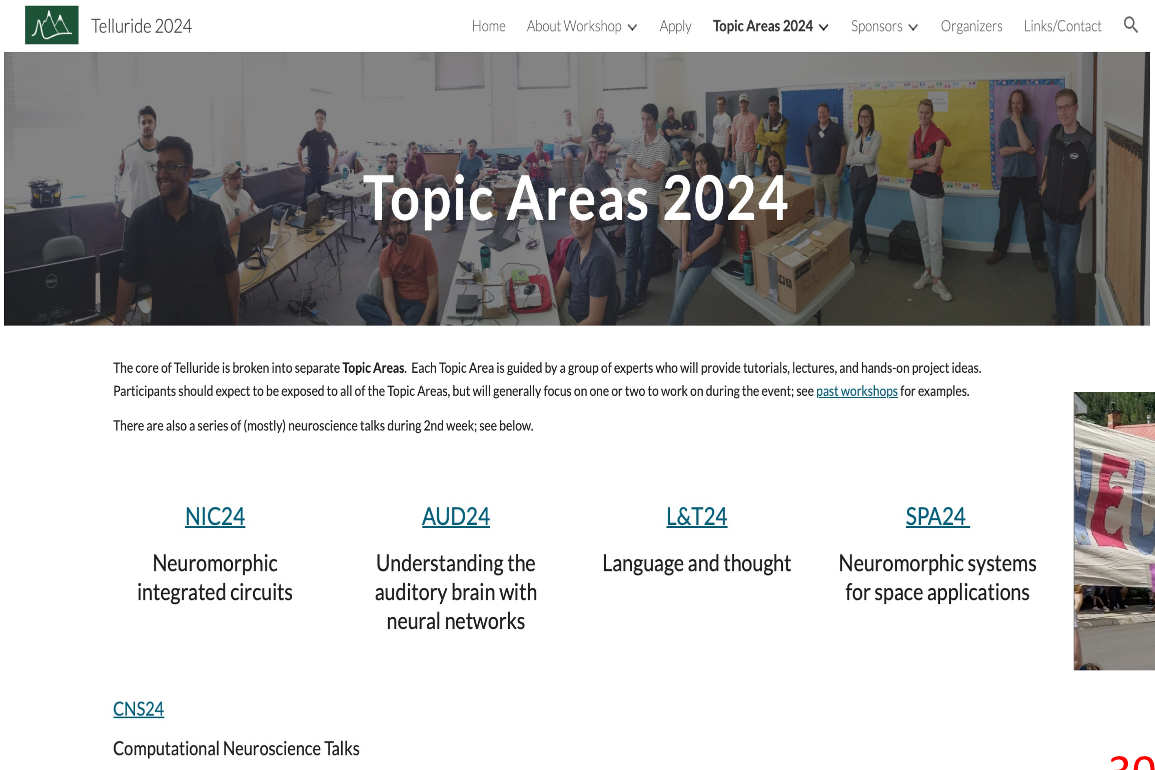
Index	Ref.	Description
a	[23]	100nm CMOS inverter
b	[23]	1000nm CMOS inverter
c	[24]	SOI 3D via
d	[16]	3D Through Silicon Via
e	[16]	MPU register file
f	[16]	L2 cache
g	[25]	IC copper trace (future)
h	[25]	IC copper trace (current)
i	[24]	electrical switching across 1cm die
j	[16]	L3 cache
k	[24]	electrical chip-to-chip link
l	[16]	DDR DRAM
m	[26]	SERDES (current)
n	[27]	SERDES (future)
o	[28]	optical
p	[29]	Firewire (IEEE 1394b)

M.A Marwick and A.G. Andreou, "Retinomorphic system design in three dimensional SOI-CMOS," *2006 IEEE International Symposium on Circuits and Systems.*
A. S. Cassidy and A. G. Andreou, "Beyond Amdahl's Law: an objective function that links multiprocessor performance gains to delay and energy," *IEEE Trans Comput*, vol. 61, no. 8, pp. 1110–1126, Aug. 2012.

Concluding Remarks

- Results from ”vanilla” LLMs for Natural Language to Verilog generation promising -OpenAI ChatGPT 4-
- Hierarchical and modular conception of the design by human is necessary
- Reduce design time
- Reduce time for documentation of design
- “Aligning” LLMs for HDL generation
- Opportunity: LLMs for end to end A2C- Application to Chip – for example Efabless KWS challenge.
- How about LLM assisted analog design?

Telluride Neuromorphic Engineering Workshop



Topic Areas 2024

The core of Telluride is broken into separate **Topic Areas**. Each Topic Area is guided by a group of experts who will provide tutorials, lectures, and hands-on project ideas. Participants should expect to be exposed to all of the Topic Areas, but will generally focus on one or two to work on during the event; see past workshops for examples.

There are also a series of (mostly) neuroscience talks during 2nd week; see below.

NIC24	AUD24	L&T24	SPA24
Neuromorphic integrated circuits	Understanding the auditory brain with neural networks	Language and thought	Neuromorphic systems for space applications

CNS24

Computational Neuroscience Talks

30th year anniversary:
Join for 3 weeks of fun @ 9000 feet

Variations of our team picture generated by Dall-E

Andreou Lab

https://andreoulab.net