ETH zürich

Marco Ferroni’s Portfolio

Last update: 12th of June 2025

Profile

I'm a 22 years old passionate student from a small town near Ravenna (IT).

Education

- **Bachelor:** Electronics Engineering at Politecnico di Milano (110L/110 with 29.48/30 final GPA).
- **Master Degree:** Electrical Engineering at ETH Zurich (current GPA 5.7/6).

Scholarships

- **Best freshmen award:** a 1000 euros scholarship awarded to the best freshmen of Politecnico di Milano (top 2.5%).
- **Fee exemption:** in my bachelor due to my high GPA (>29/30) I was granted a fee exemption.

Job experience:

- **Academic guest:** I'm currently an academic guest as part of the Transport at Nanoscale Interfaces laboratory at EMPA where I did quantum simulations of strained CNTs.
- **Specialized worker:** at 16 I've worked in a small-sized enterprise, where I managed alone the whole production line of industrial visual interfaces.

Me

Me and my cat!

Skills

My main field of interest is ASIC design; therefore, I'm focusing my learning efforts on it.

- **ASIC workflow**: Good general understanding of the many steps involved in making an ASIC, from microarchitecture to backend.
- **Cadence Virtuoso Suite**: Solid knowledge, as I've used it extensively for many courses and projects over the last 8 months.
- **RTL**: Good knowledge of both SystemVerilog and VHDL.
- **Functional verification**: Good understanding of how functional verification works and how to use QuestaSim.
- **Physical layout**: Good understanding of the required steps and how to use OpenRoad.
- **Standard cells**: Good understanding of the impact of design choices and logic styles on standard cell performance.
- **Microarchitecture**: Good understanding of how different microarchitectural choices can meet various needs.
- **Analog electronics:** Decent ability to understand, design and test analog circuits.
- **Electronic and optical devices:** Good understanding of the quantum principles behind electronic and optical devices.
- **Self-sufficiency**: Very good problem-solving skills and ability to adapt without requiring extensive supervision.
- **Communication**: Strong communication skills.

ETH zürich

Expanding the ez130 library

Expanding the ez130 library -Overview

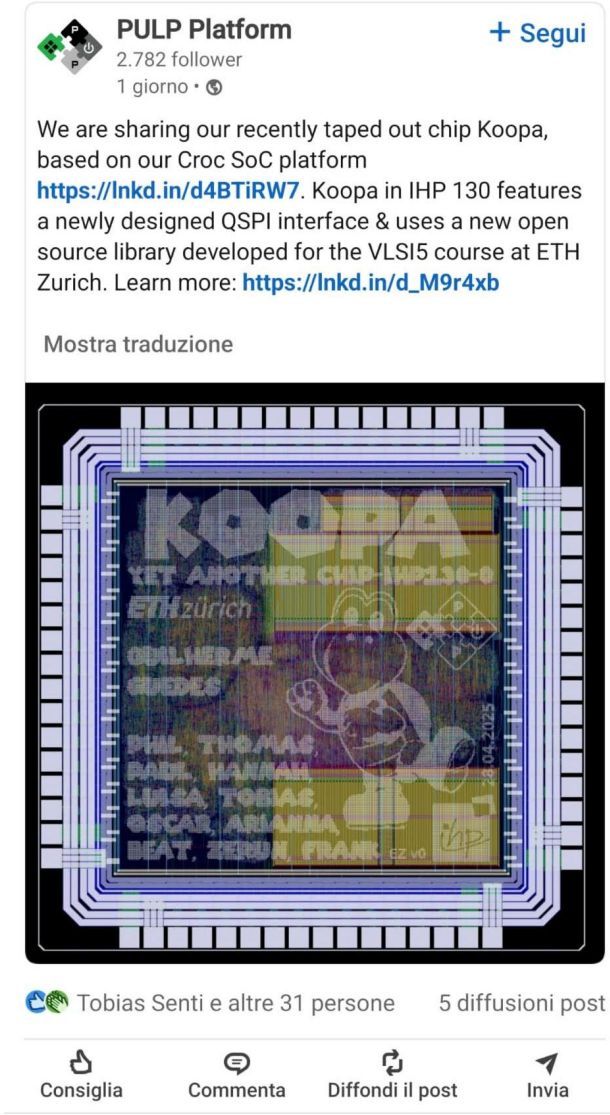
As part of the VLSI5 course offered in the spring semester of 2025 (for the first and maybe last time), I and 24 other selected students had the opportunity to expand the 130nm open-source library of the PULP group.

Each of us had 3 standard cells to implement, both in X1 and X2 strengths. Mine were:

- **NOR4:** a NOR gate with four inputs
- **AO31:** an AND gate with three inputs whose output is fed into a two-input NOR together with a fourth input
- **XNOR3:** a three-input XNOR gate

The workflow was the following:

1. **Schematic design:** exploiting logic restructuring and exploring different designs.
2. **Stick diagram:** thinking through the physical layout itself, trying to share diffusions and optimizing connections.
3. **Layout:** Implementing the layout on Cadence and making sure it's DRC and LVS clean.
4. **Characterization:** running PEX and characterizing the cell using Cadence Liberate, then creating the LEF and GDS files, and finally exporting the schematic as a CDL.
5. **Verification:** to make sure our cells were used we did the backend of an adder, and made sure it was DRC and LVS clean and performed functional verification.



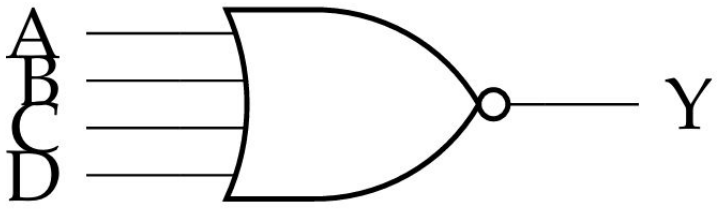
Our cells are already getting used

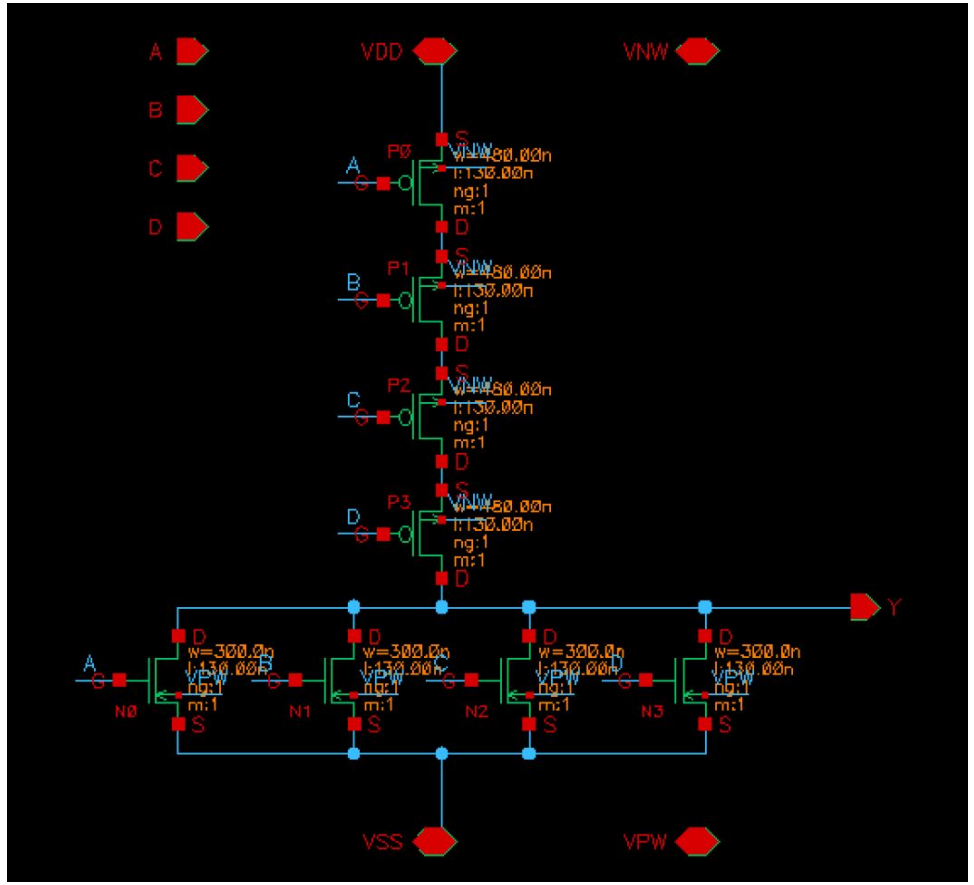
Expanding the ez130 library -NOR4X1

Logic function: $Y = \overline{A + B + C + D}$

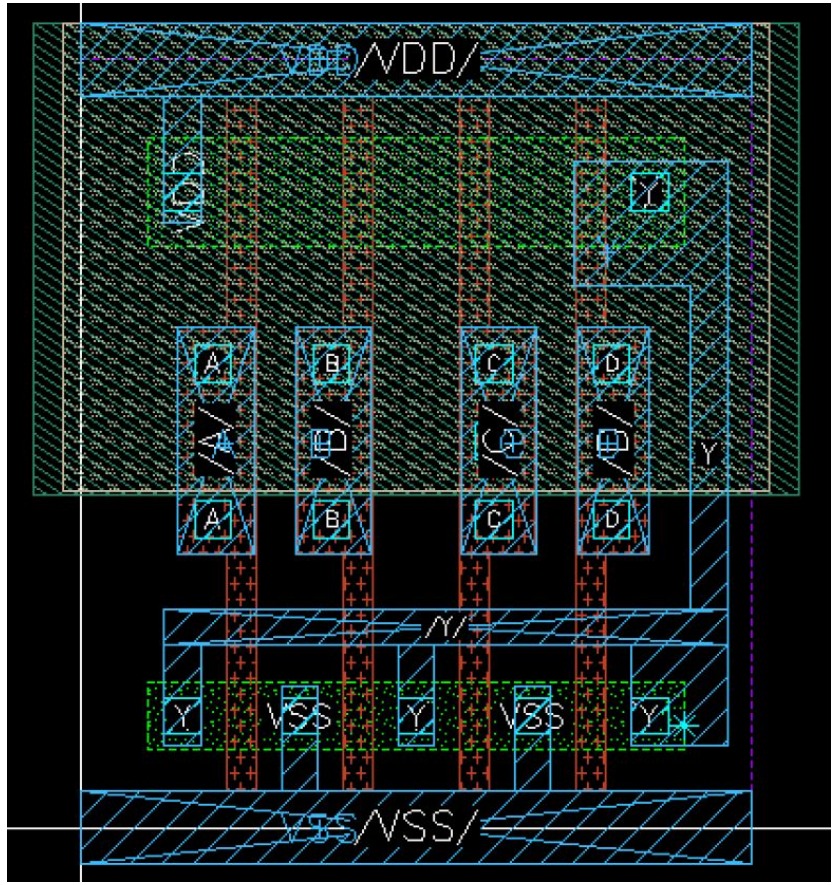
N° of transistors: 8

Area: 9.8784um^2





NOR4X1 schematic



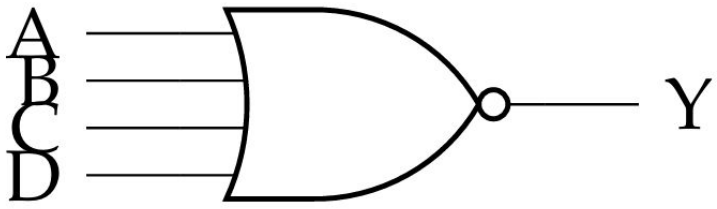
NOR4X1 layout

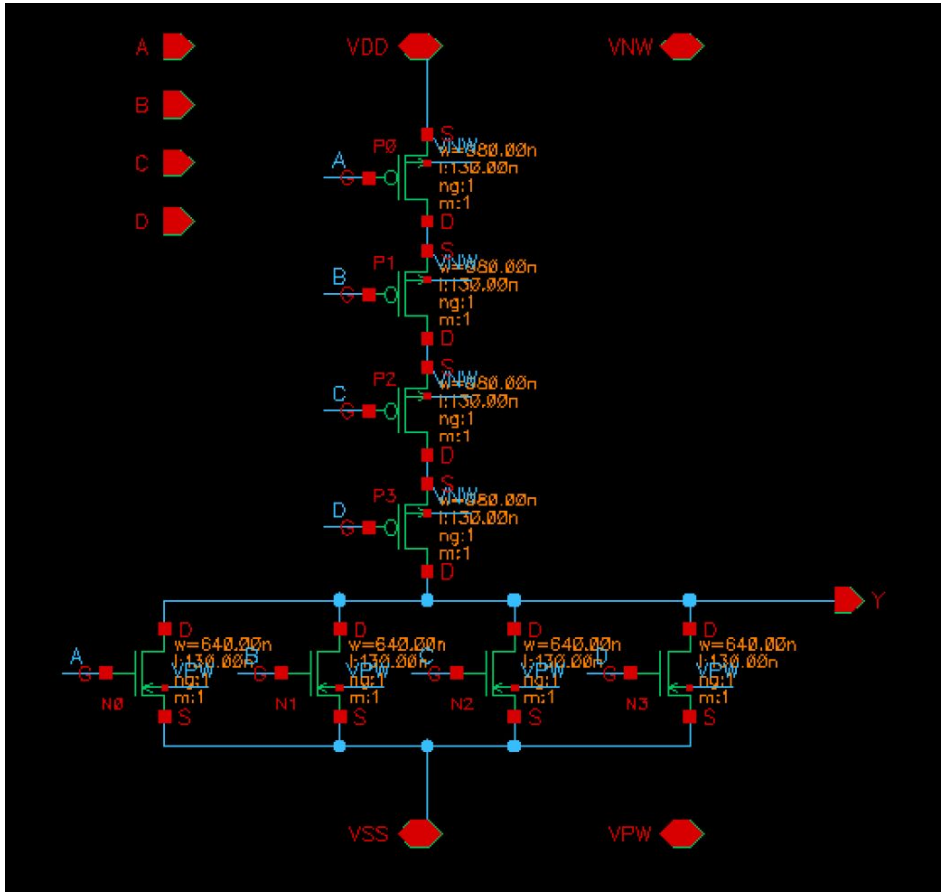
Expanding the ez130 library -NOR4X2

Logic function: $Y = \overline{A + B + C + D}$

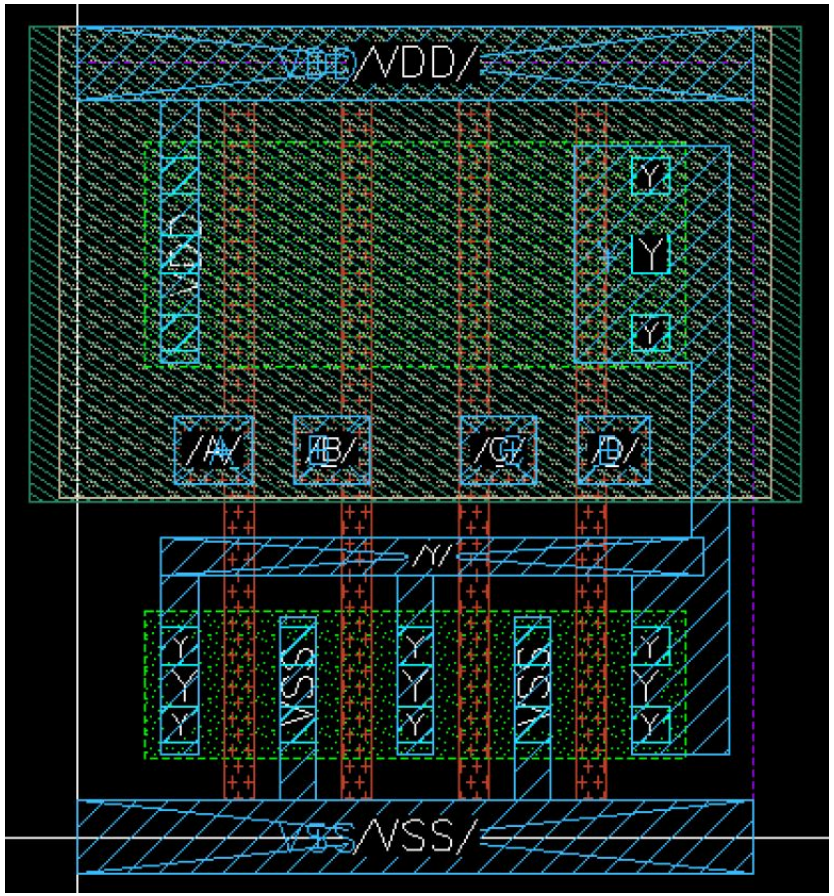
N° of transistors: 8

Area: 9.8784um^2





NOR4X2 schematic



NOR4X2 layout

Expanding the ez130 library -NOR4 performance

Input pin capacitance

Cell	Capacitance [pF]			
	A	B	C	D
NOR4X1	0.0016	0.0017	0.0017	0.0017
NOR4X2	0.0032	0.0032	0.0033	0.0033

Delay

Cell	Rising delay [ns]				Falling delay [ns]			
	A→Y	B→Y	C→Y	D→Y	A→Y	B→Y	C→Y	D→Y
NOR4X1	0.8152	0.8217	0.8253	0.8047	0.4155	0.4115	0.4054	0.3966
NOR4X2	0.7932	0.8030	0.8111	0.7965	0.4012	0.3971	0.3914	0.3840

Internal dynamic energy

Cell	Rising energy [pJ]				Falling energy [pJ]			
	A→Y	B→Y	C→Y	D→Y	A→Y	B→Y	C→Y	D→Y
NOR4X1	0.0068	0.0058	0.0048	0.0039	0.0003	0.0001	0.0000	-0.0001
NOR4X2	0.0126	0.0108	0.0090	0.0074	0.0005	0.0002	0.0000	-0.0002

Leakage power

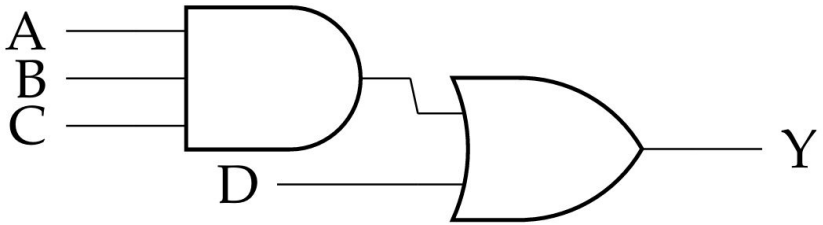
Cell	Leakage [nW]
NOR4X1	0.0325
NOR4X2	0.0812

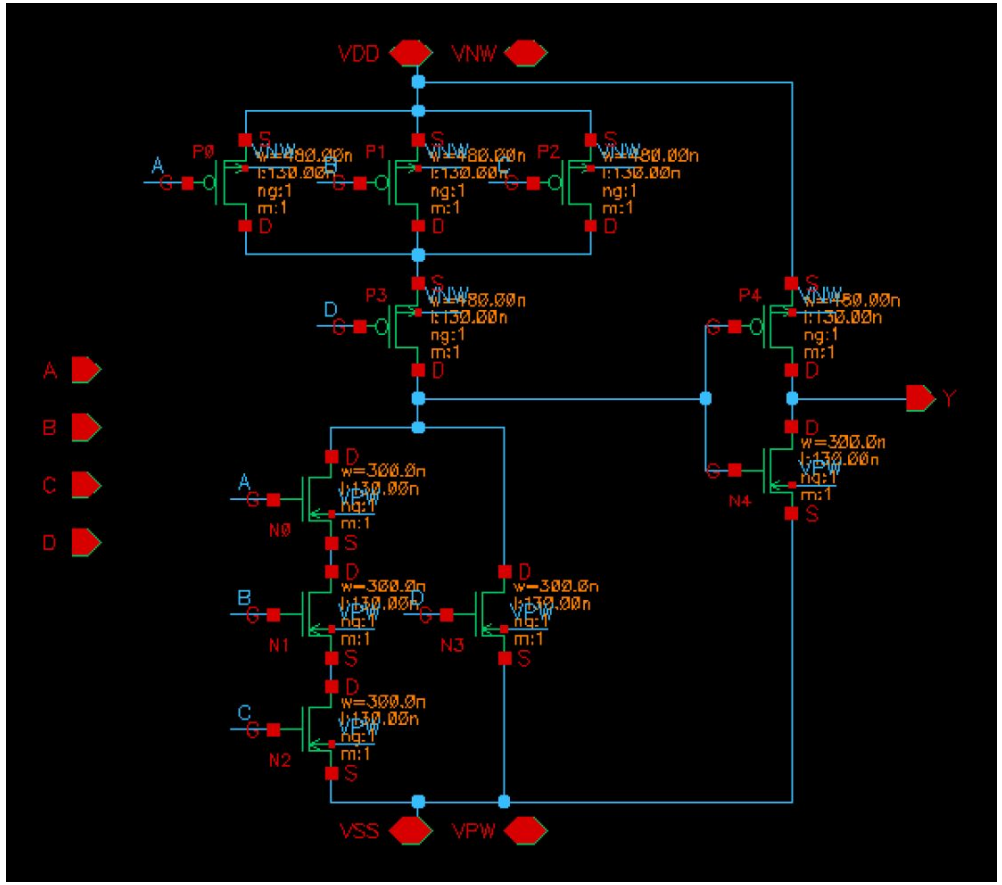
Expanding the ez130 library -AO31X1

Logic function: $Y = A \cdot B \cdot C + D$

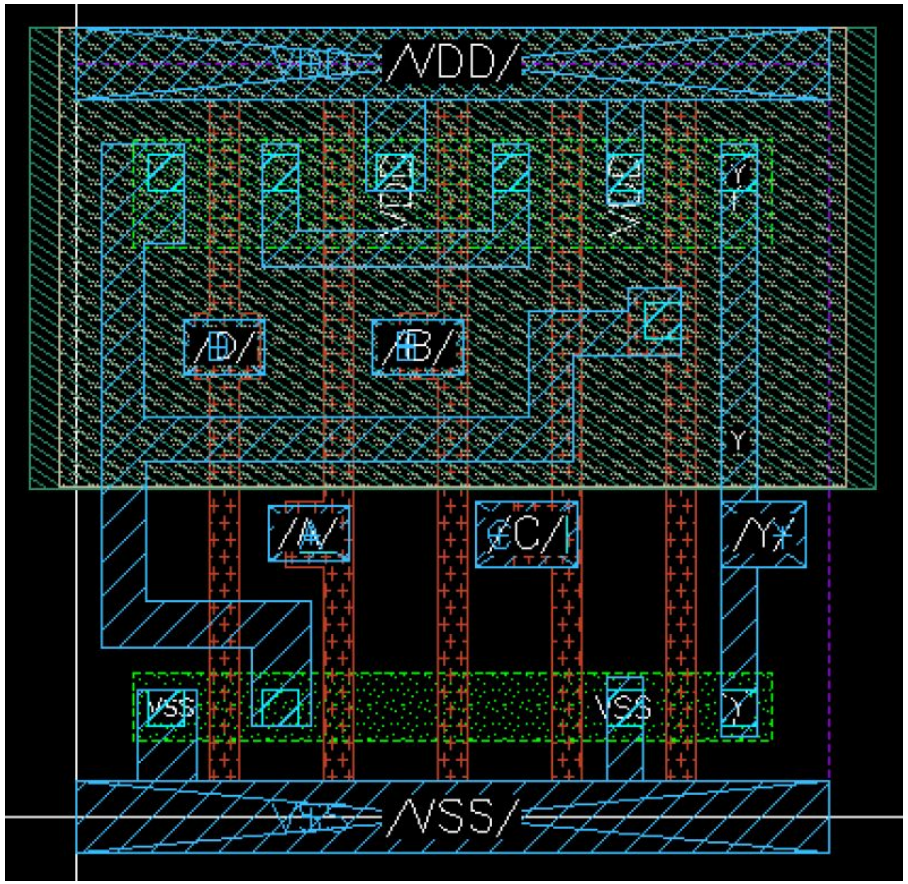
N° of transistors: 10

Area: 11.2896um^2





AO31X1 schematic



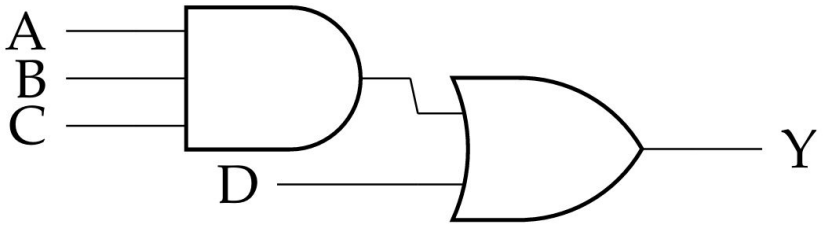
AO31X1 layout

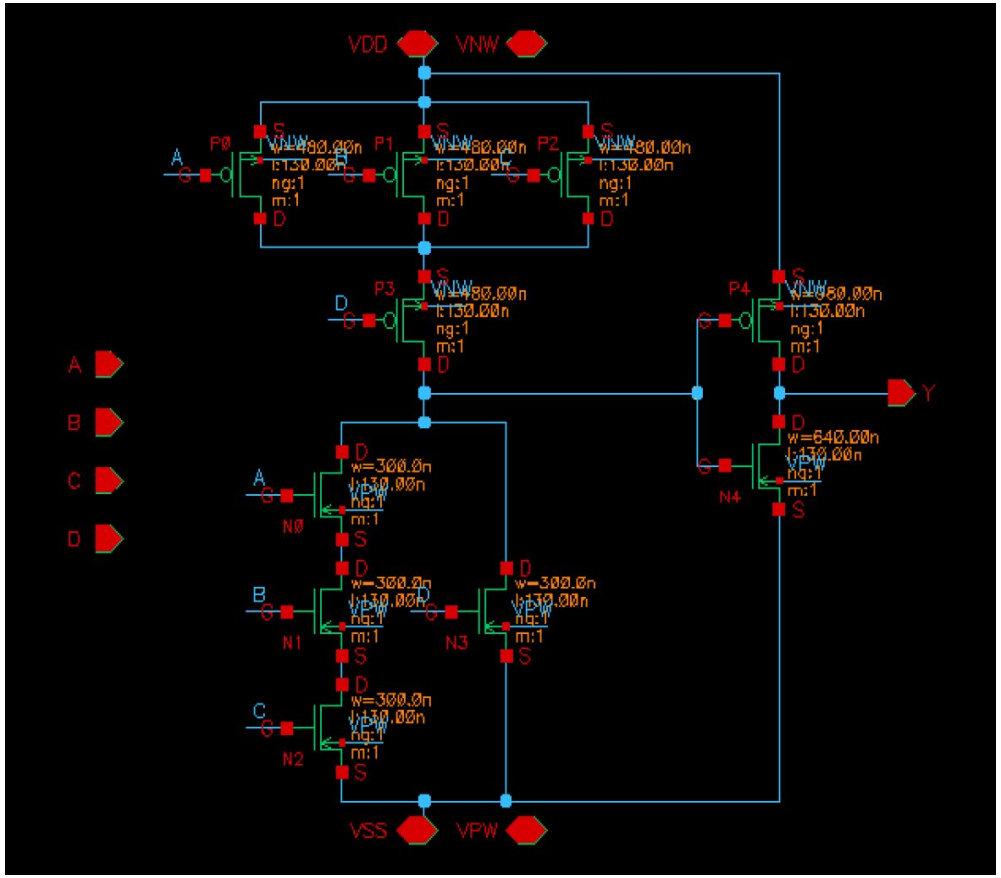
Expanding the ez130 library -AO31X2

Logic function: $Y = A \cdot B \cdot C + D$

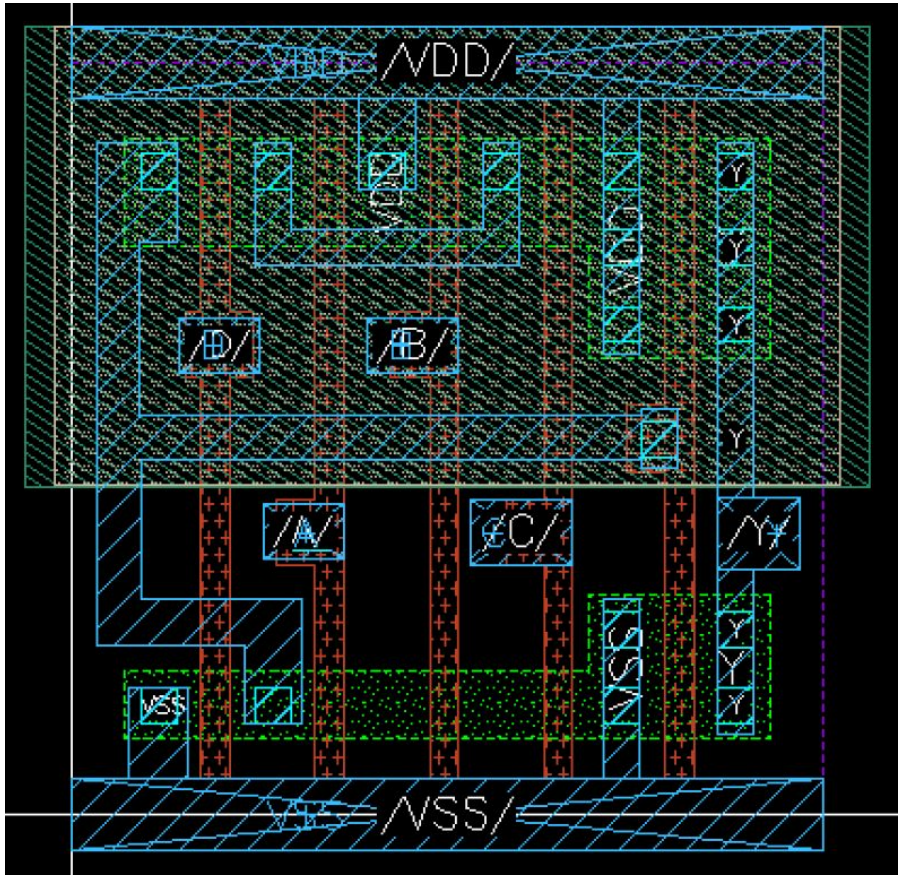
N° of transistors: 10

Area: 11.2896um^2





AO31X2 schematic



AO31X2 layout

Expanding the ez130 library -AO31 performance

Input pin capacitance

Cell	Capacitance [pF]			
	A	B	C	D
AO31X1	0.0016	0.0015	0.0015	0.0016
AO31X2	0.0016	0.0015	0.0015	0.0016

Delay

Cell	Rising delay [ns]				Falling delay [ns]			
	A→Y	B→Y	C→Y	D→Y	A→Y	B→Y	C→Y	D→Y
AO31X1	0.4408	0.4526	0.4465	0.3595	0.3782	0.4001	0.4182	0.4024
AO31X2	0.4905	0.4983	0.4880	0.3900	0.4205	0.4400	0.4562	0.4482

Internal dynamic energy

Cell	Rising energy [pJ]				Falling energy [pJ]			
	A→Y	B→Y	C→Y	D→Y	A→Y	B→Y	C→Y	D→Y
AO31X1	0.0025	0.0023	0.0023	0.0023	0.0064	0.0072	0.0081	0.0066
AO31X2	0.0047	0.0045	0.0045	0.0044	0.0084	0.0092	0.0101	0.0086

Leakage power

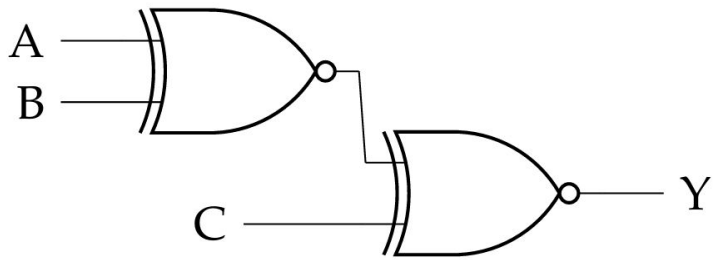
Cell	Leakage [nW]
AO31X1	0.0699
AO31X2	0.0971

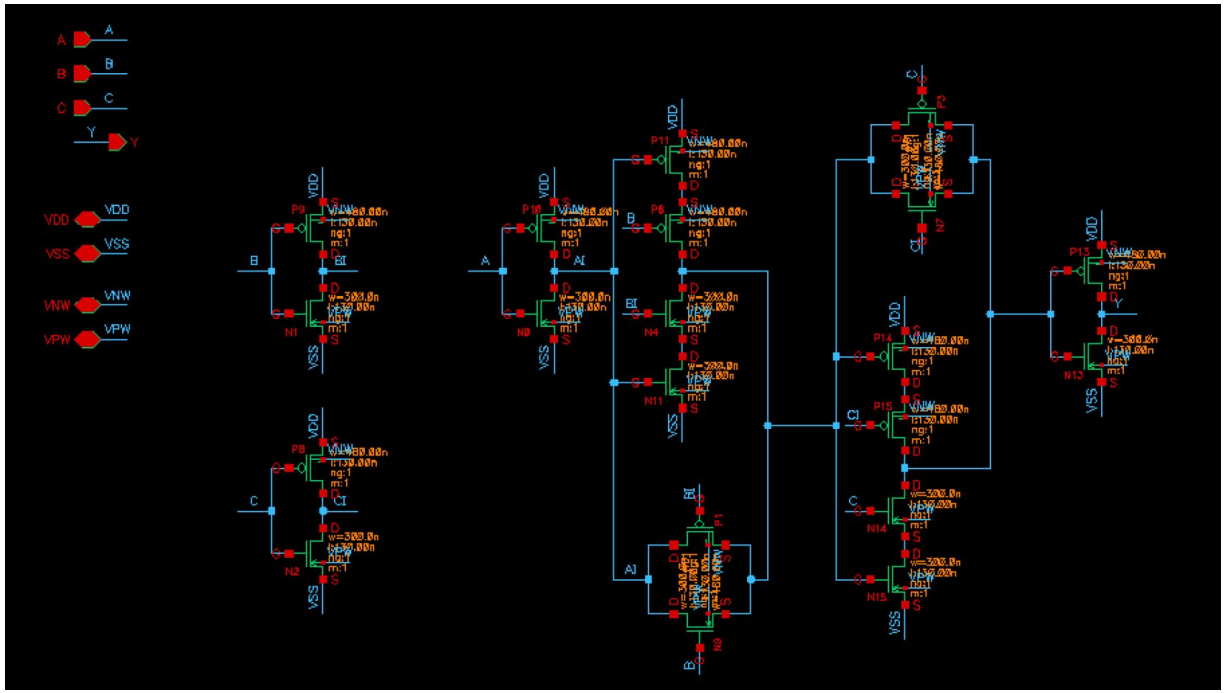
Expanding the ez130 library -XNOR3X1

Logic function: $Y = \overline{A \oplus B \oplus C}$

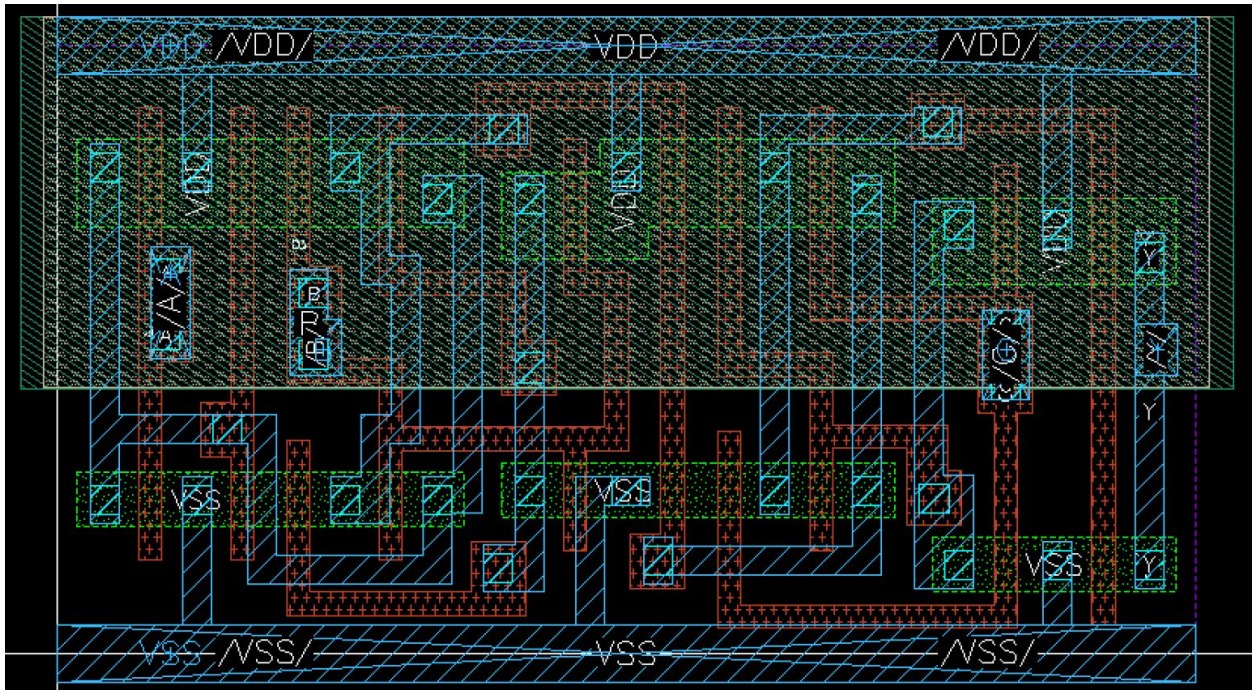
N° of transistors: 20

Area: 21.168um^2





XNOR3X1 schematic



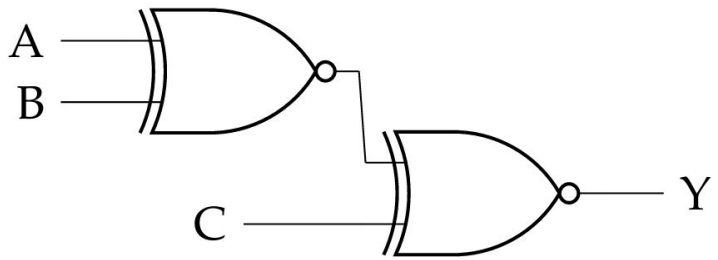
XNOR3X1 layout

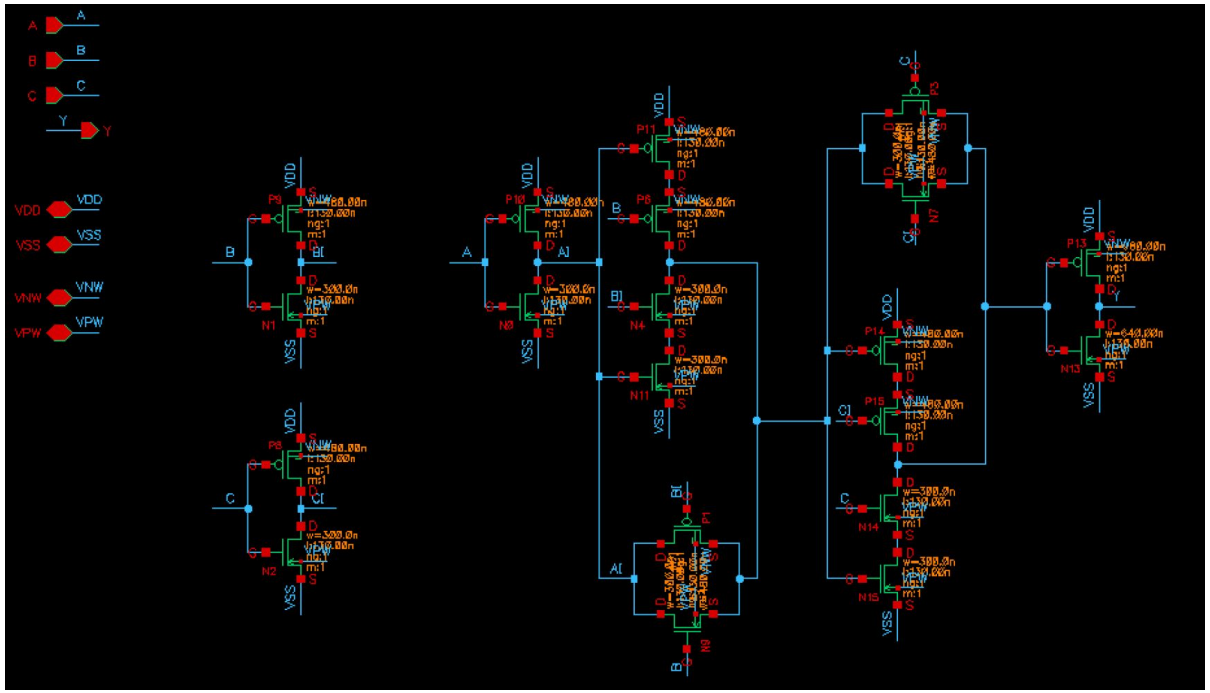
Expanding the ez130 library -XNOR3X2

Logic function: $Y = \overline{A \oplus B \oplus C}$

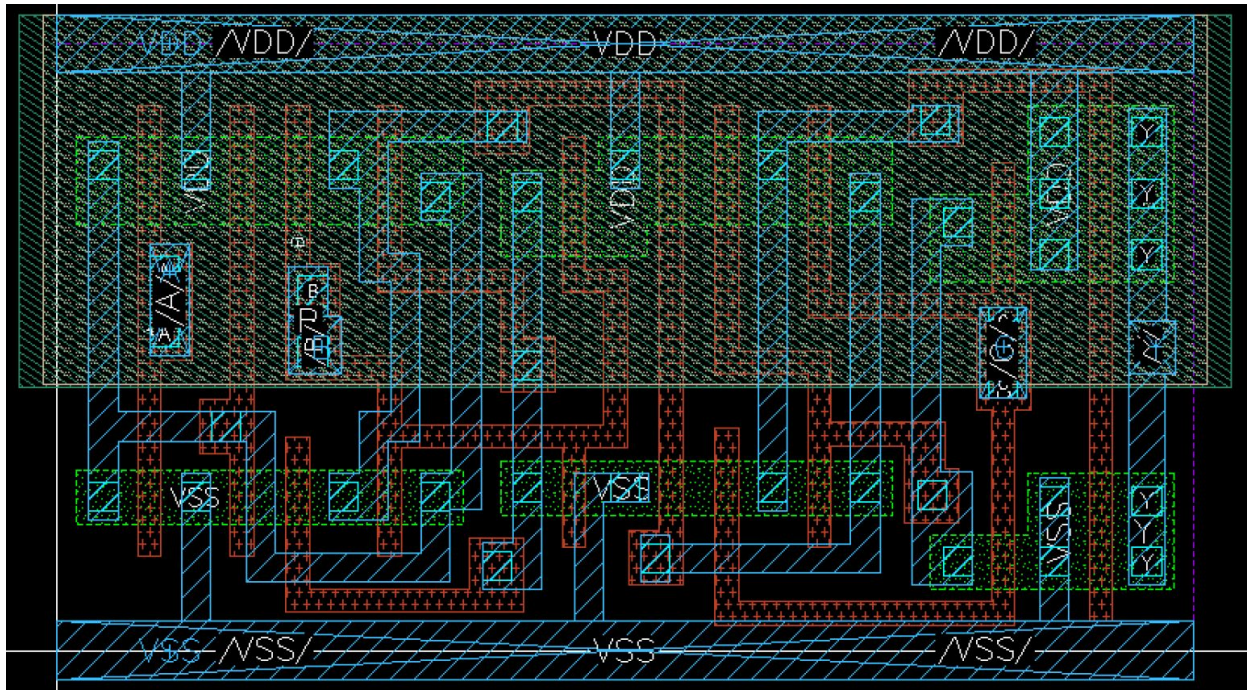
N° of transistors: 20

Area: 21.168um^2





XNOR3X2 schematic



XNOR3X2 layout

Expanding the ez130 library -XNOR3 performance

Input pin capacitance

Cell	Capacitance [pF]		
	A	B	C
XNOR3X1	0.0016	0.0032	0.0032
XNOR3X2	0.0016	0.0032	0.0032

Delay

Cell	Rising delay [ns]			Falling delay [ns]		
	A→Y	B→Y	C→Y	A→Y	B→Y	C→Y
XNOR3X1	0.5479	0.4918	0.3607	0.5625	0.5309	0.3804
XNOR3X2	0.5792	0.5270	0.4097	0.5978	0.5741	0.4427

Internal dynamic energy

Cell	Rising energy [pJ]			Falling energy [pJ]		
	A→Y	B→Y	C→Y	A→Y	B→Y	C→Y
XNOR3X1	0.0119	0.0116	0.0085	0.0134	0.0113	0.0078
XNOR3X2	0.0141	0.0138	0.0107	0.0155	0.0134	0.0097

Leakage power

Cell	Leakage [nW]
XNOR3X1	0.1888
XNOR3X2	0.2170

ETH zürich

Designing a SpMM accelerator

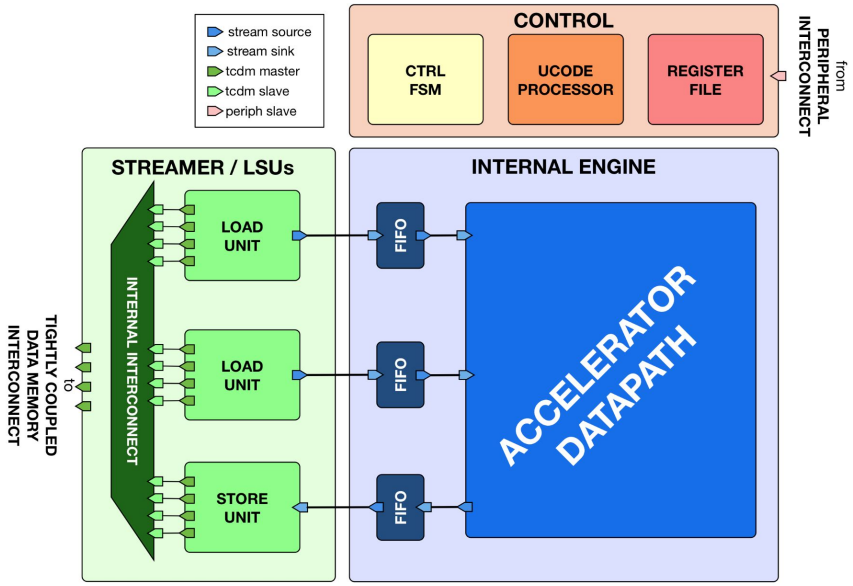
Designing a SpMM accelerator -Overview

I am currently developing a Sparse Matrix-Matrix Multiplication (SpMM) accelerator as part of my semester thesis under the supervision of Prof. Luca Benini. The main motivation behind this project is to exploit the sparsity of the X matrix in order to **reduce latency, power consumption,** and **data movement**.

1. The first step of the project involves exploring the most efficient dataflow strategy for matrix multiplication, aiming for low latency, minimal bandwidth usage, and high utilization of the computational engine. The dataflow options under consideration are the **Gustavson** and **Outer Product** in their three variants: **X, Y** and **Z**-stationary, where the matrix multiplication is defined as **X × Y = Z**.
2. Next, I will proceed to write the RTL implementation. I have been provided with a working example of a datamover, which I will modify and adapt to suit the needs of the project.
3. Then, I will design testbenches and carry out functional verification. Finally, I will synthesize the design to evaluate its area and power consumption.

The general architecture of the accelerator consists of a combinatorial matrix of MAC units, which process chunks of the input matrices.

My progress so far is available on my GitHub page: https://github.com/Ferrons02

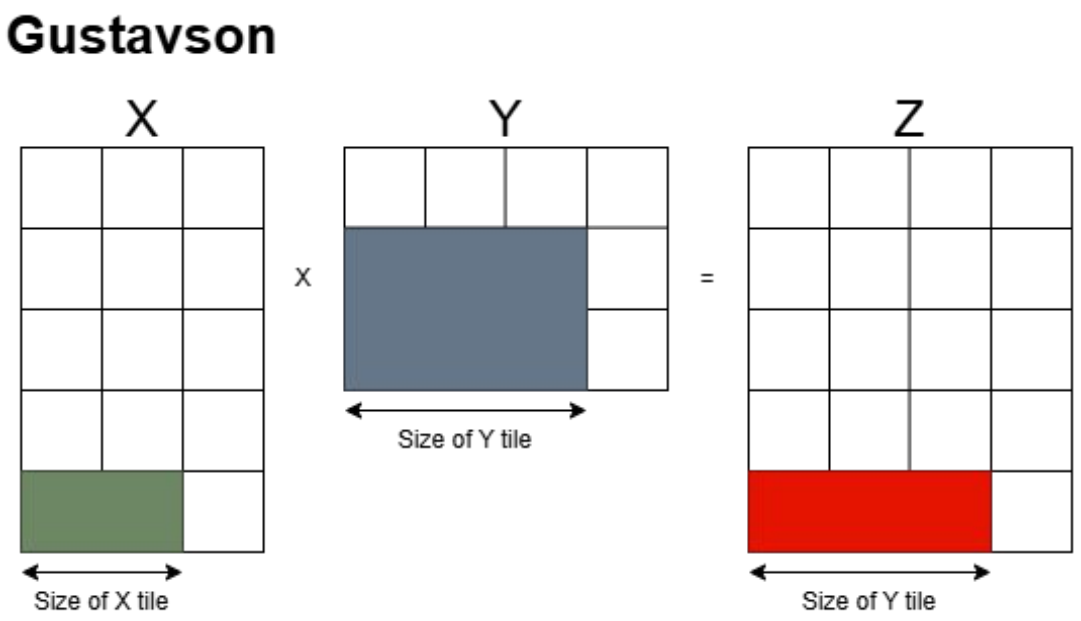


General structure of the accelerator

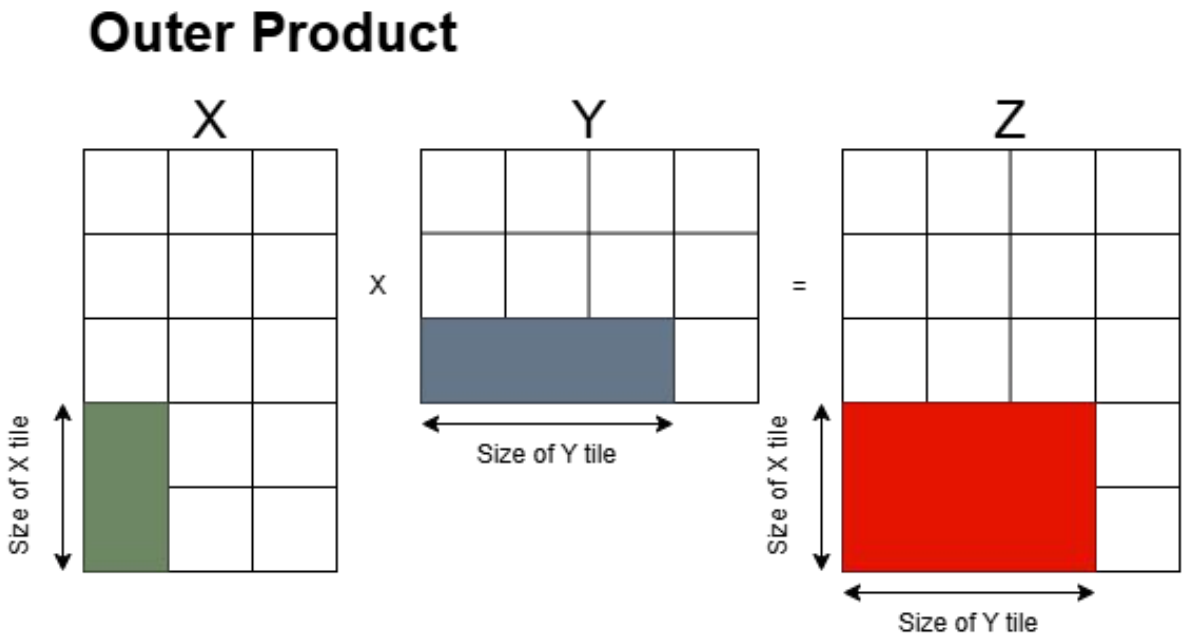
Designing a SpMM accelerator -Exploration introduction

The first step was to build a golden model for each dataflow, starting with the dense version and then extending it to the sparse version, where the sparsity of matrix X is exploited. These models are used to evaluate latency, bandwidth, and utilization. Additionally, I have implemented graphs showing memory accesses at each cycle to better understand the access patterns.

In my analysis, I will refer to X, Y, and Z "reuse" to indicate X-stationary, Y-stationary, and Z-stationary dataflows, respectively. The following conventions are also used to describe the size of the matrix chunks that the engine processes in a single cycle:



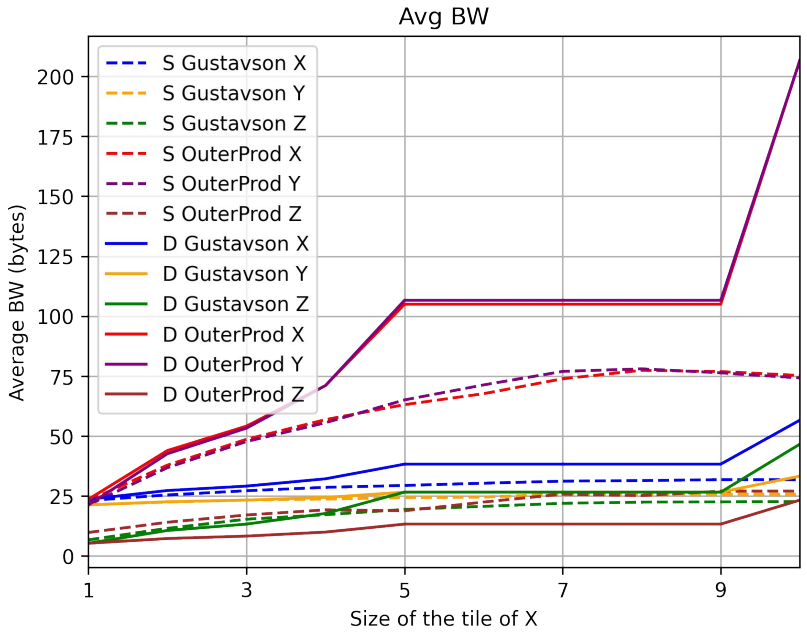
Conventions used for the Gustavson dataflow

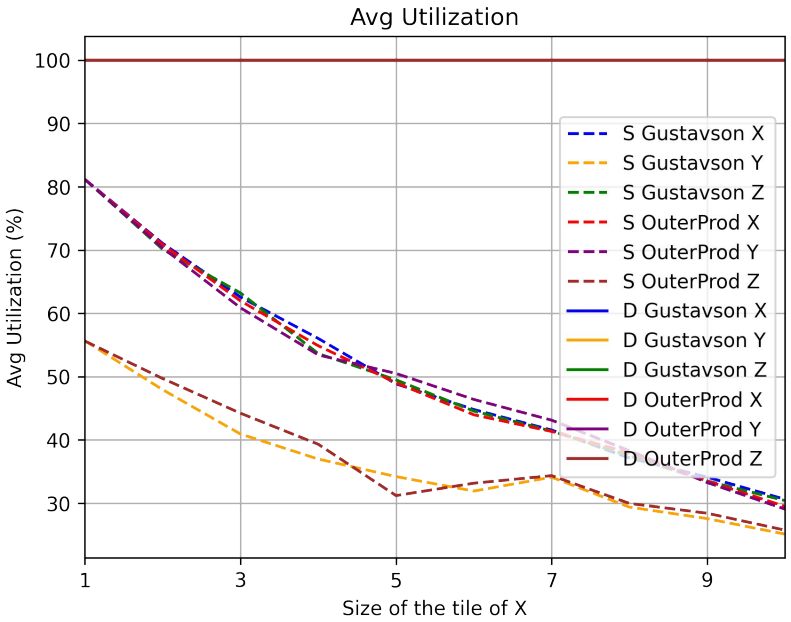


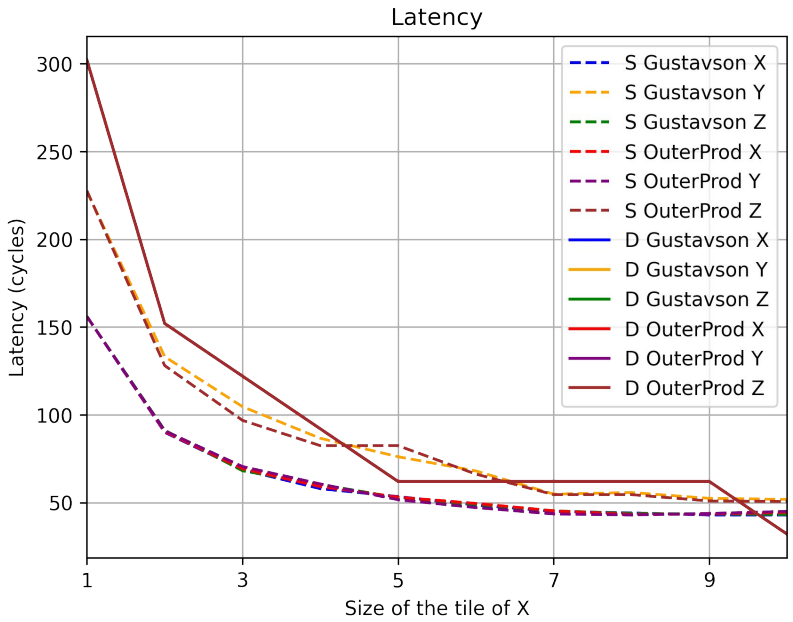
Conventions used for the Outer product dataflow

Designing a SpMM accelerator -Exploration (Size of X tile)

For the comparison between dataflows, I used the following default parameters: both X and Y matrices are 10×10, X_tile and Y_tile are set to 4, and the sparsity factor of X is 0.5. The Avg. BW refers to the combined bandwidth, including both input and output data from the accelerator, while Avg. Utilization indicates engine utilization across cycles, accounting for partial usage in some cycles due to sparsity. In the graphs, the prefixes “S” and “D” represent “sparse” and “dense”; in the dense version, X is stored in bitmap format, and its metadata is loaded in 32-bit chunks.

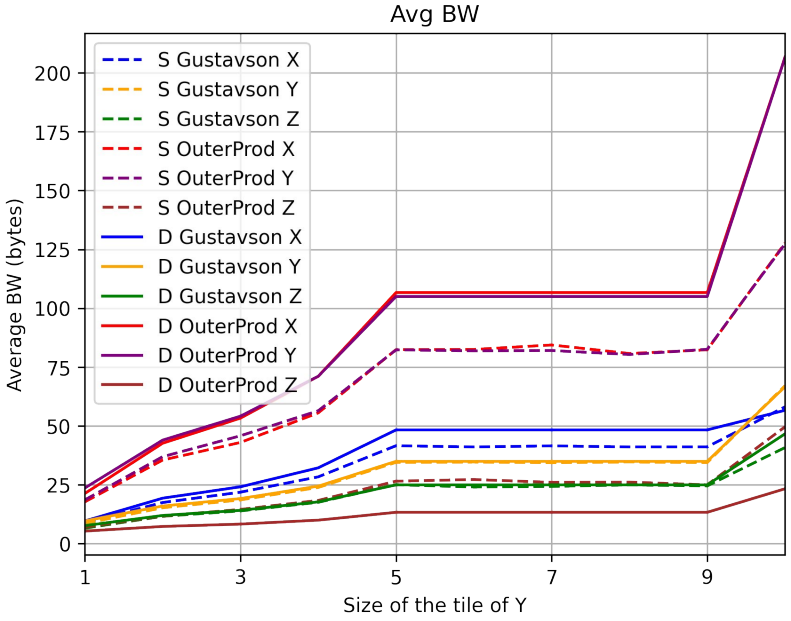


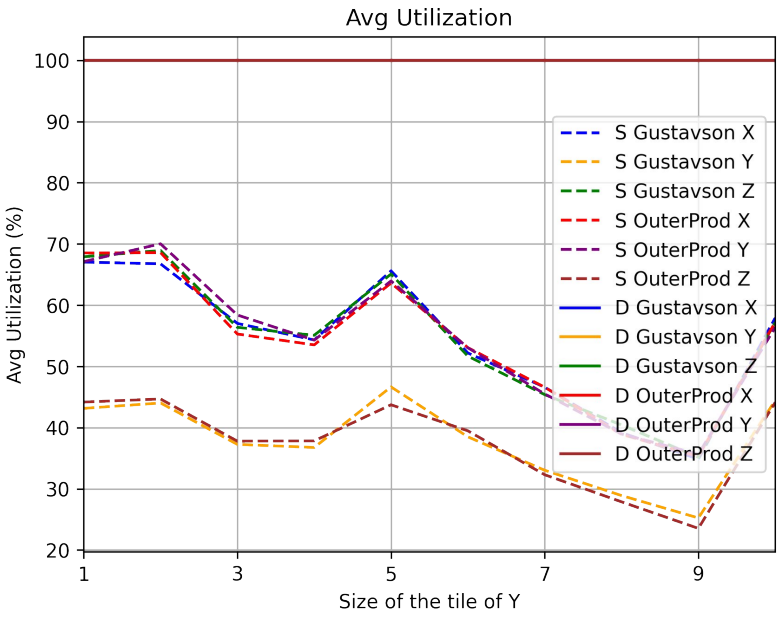


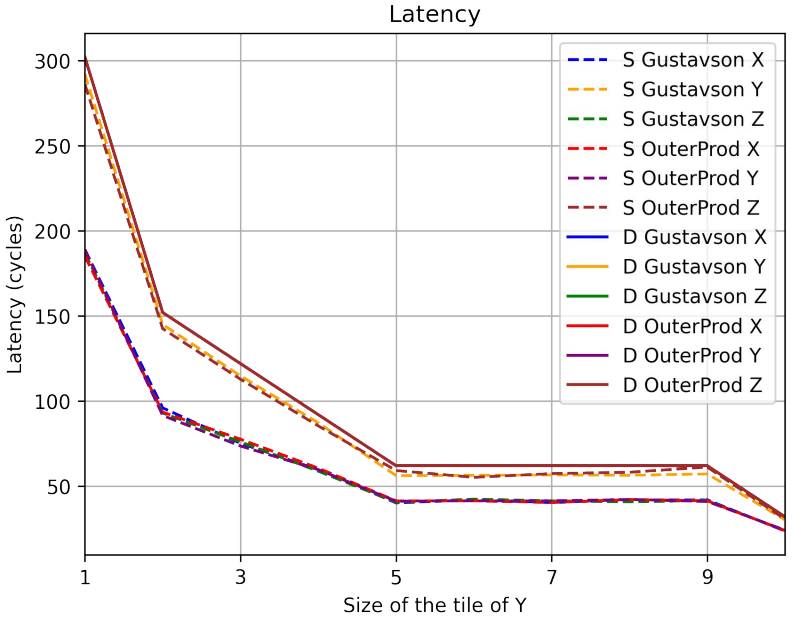


Designing a SpMM accelerator -Exploration (Size of Y tile)

For the comparison between dataflows, I used the following default parameters: both X and Y matrices are 10×10, X_tile and Y_tile are set to 4, and the sparsity factor of X is 0.5. The Avg. BW refers to the combined bandwidth, including both input and output data from the accelerator, while Avg. Utilization indicates engine utilization across cycles, accounting for partial usage in some cycles due to sparsity. In the graphs, the prefixes “S” and “D” represent “sparse” and “dense”; in the dense version, X is stored in bitmap format, and its metadata is loaded in 32-bit chunks.

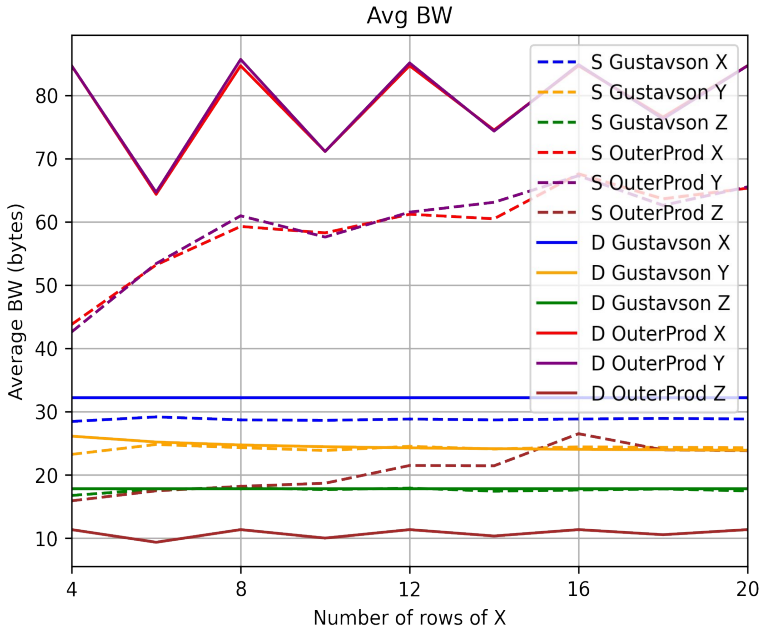


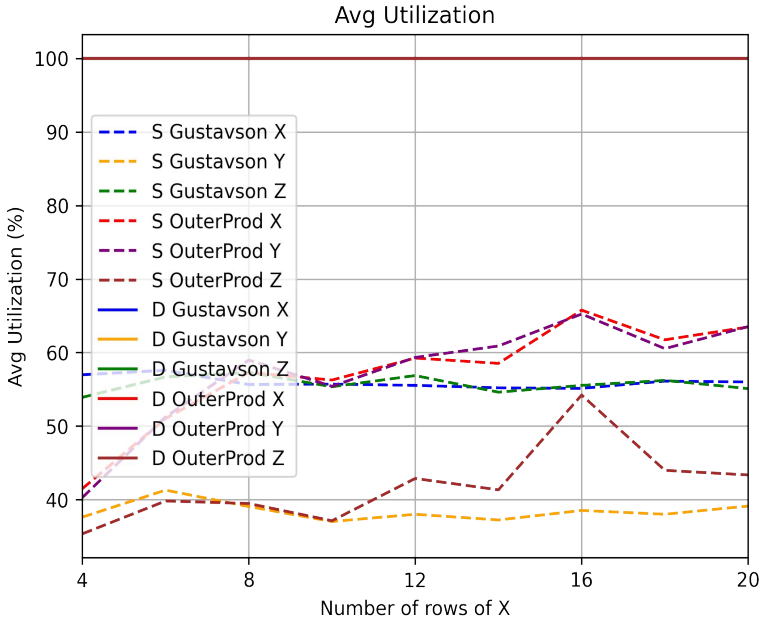


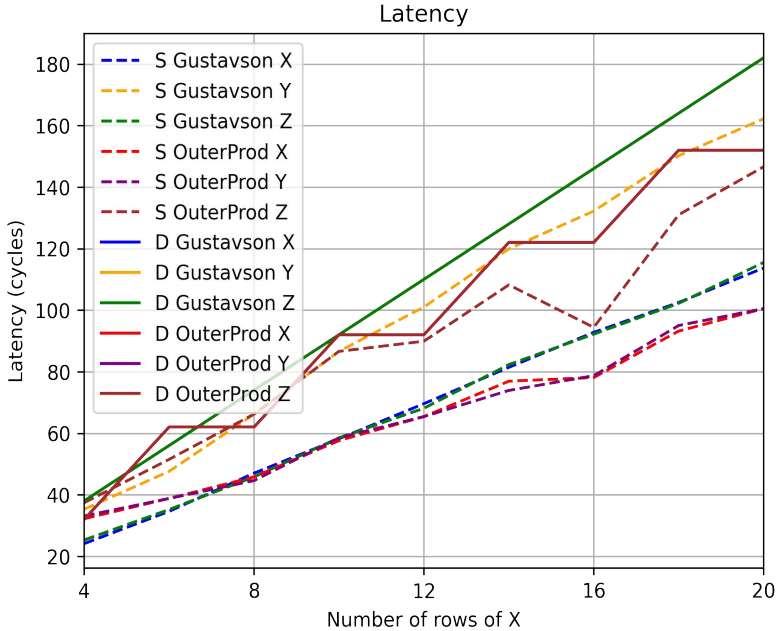


Designing a SpMM accelerator -Exploration (rows of X)

For the comparison between dataflows, I used the following default parameters: both X and Y matrices are 10×10, X_tile and Y_tile are set to 4, and the sparsity factor of X is 0.5. The Avg. BW refers to the combined bandwidth, including both input and output data from the accelerator, while Avg. Utilization indicates engine utilization across cycles, accounting for partial usage in some cycles due to sparsity. In the graphs, the prefixes “S” and “D” represent “sparse” and “dense”; in the dense version, X is stored in bitmap format, and its metadata is loaded in 32-bit chunks.

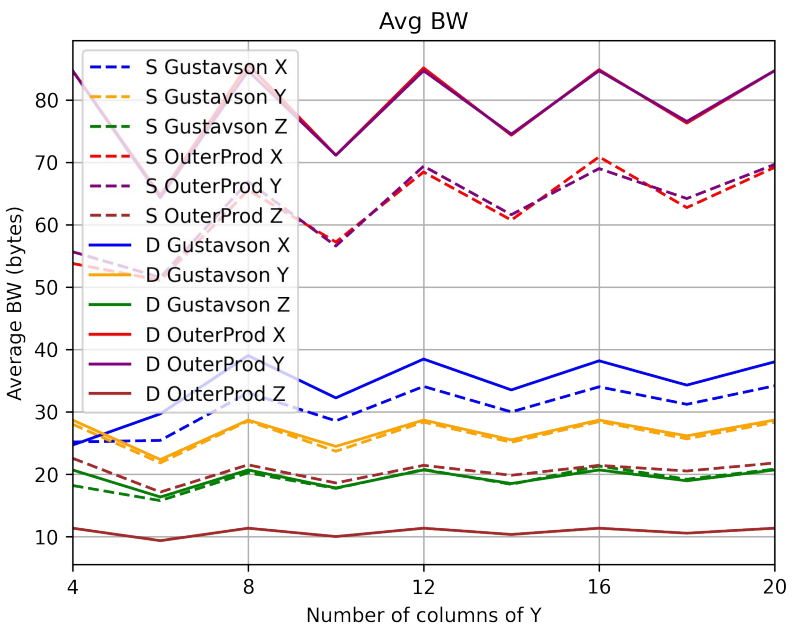


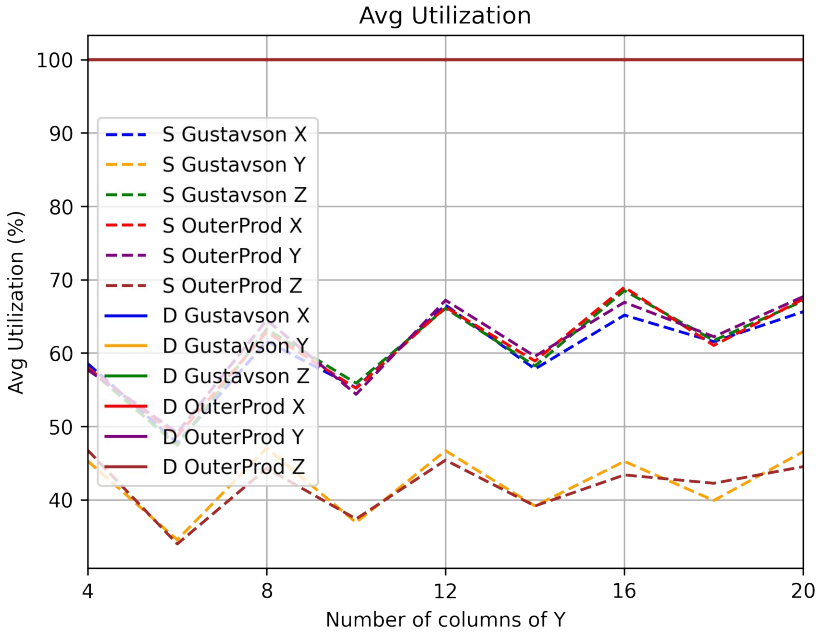


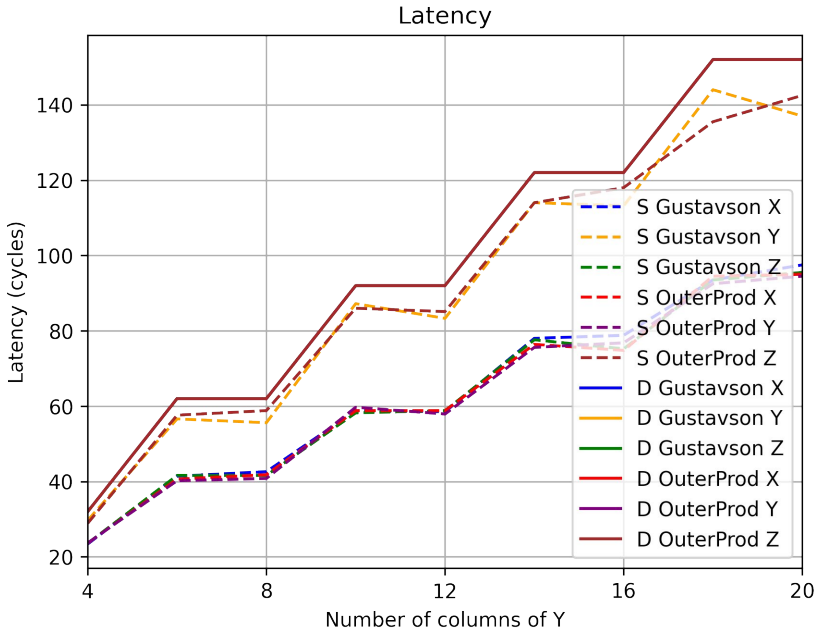


Designing a SpMM accelerator -Exploration (columns of Y)

For the comparison between dataflows, I used the following default parameters: both X and Y matrices are 10×10, X_tile and Y_tile are set to 4, and the sparsity factor of X is 0.5. The Avg. BW refers to the combined bandwidth, including both input and output data from the accelerator, while Avg. Utilization indicates engine utilization across cycles, accounting for partial usage in some cycles due to sparsity. In the graphs, the prefixes “S” and “D” represent “sparse” and “dense”; in the dense version, X is stored in bitmap format, and its metadata is loaded in 32-bit chunks.

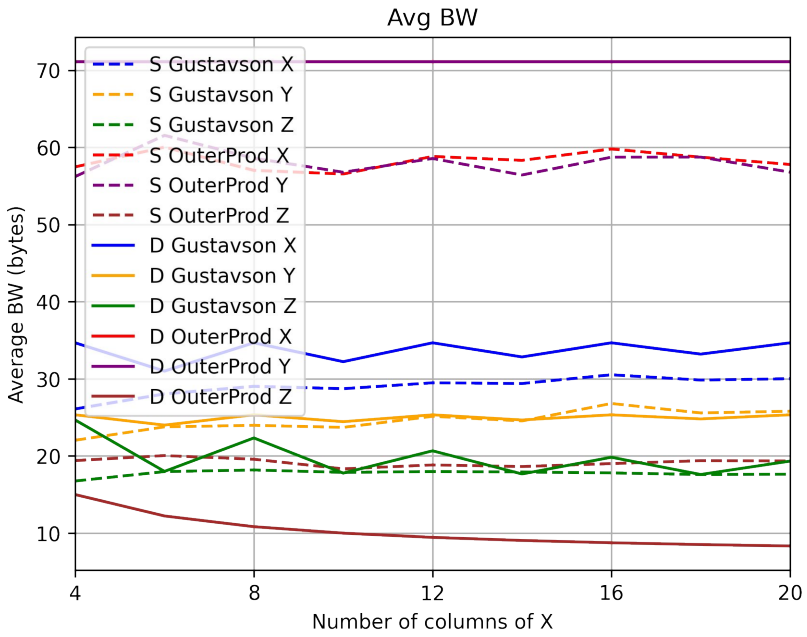


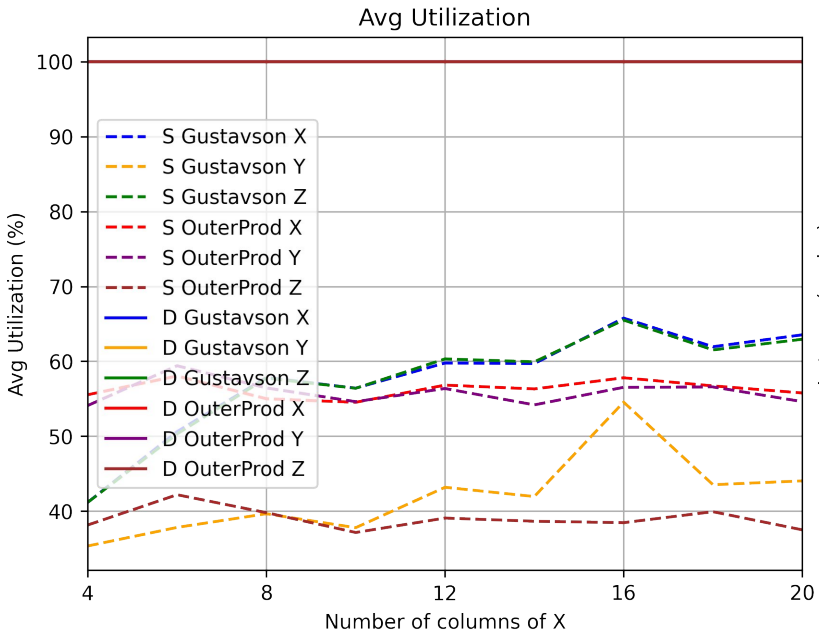


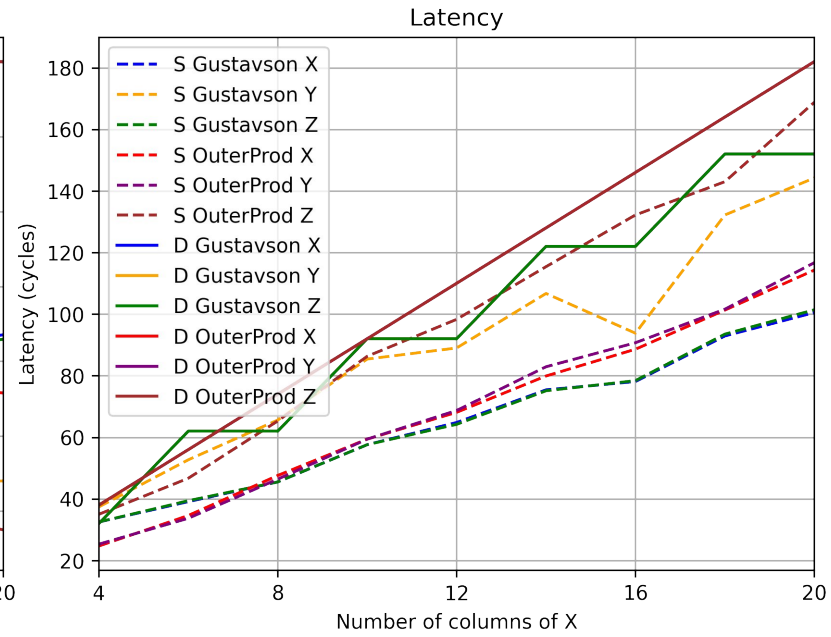


Designing a SpMM accelerator -Exploration (columns of X)

For the comparison between dataflows, I used the following default parameters: both X and Y matrices are 10×10, X_tile and Y_tile are set to 4, and the sparsity factor of X is 0.5. The Avg. BW refers to the combined bandwidth, including both input and output data from the accelerator, while Avg. Utilization indicates engine utilization across cycles, accounting for partial usage in some cycles due to sparsity. In the graphs, the prefixes “S” and “D” represent “sparse” and “dense”; in the dense version, X is stored in bitmap format, and its metadata is loaded in 32-bit chunks.

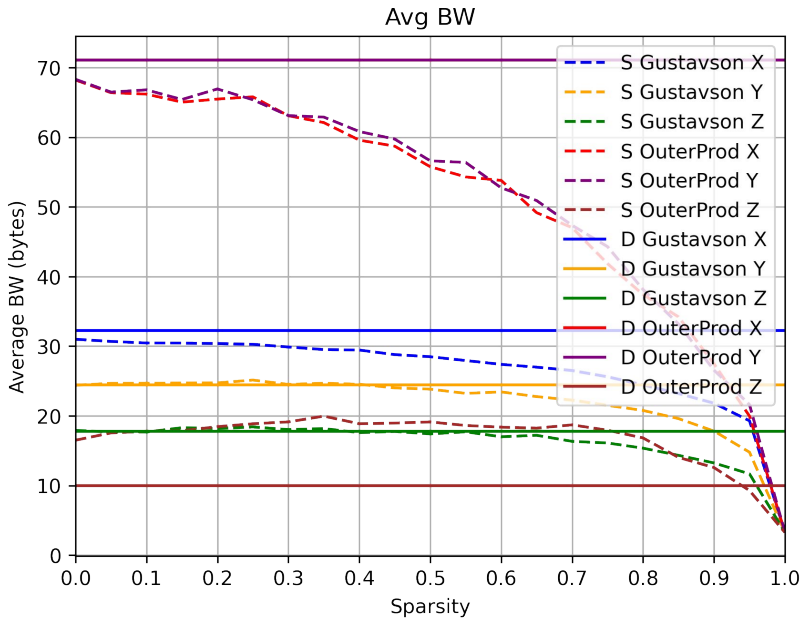


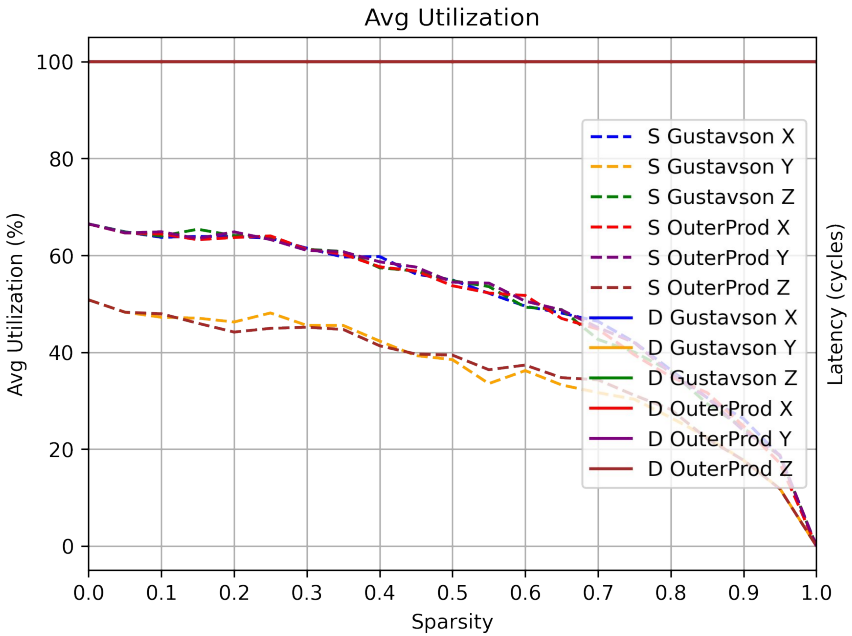


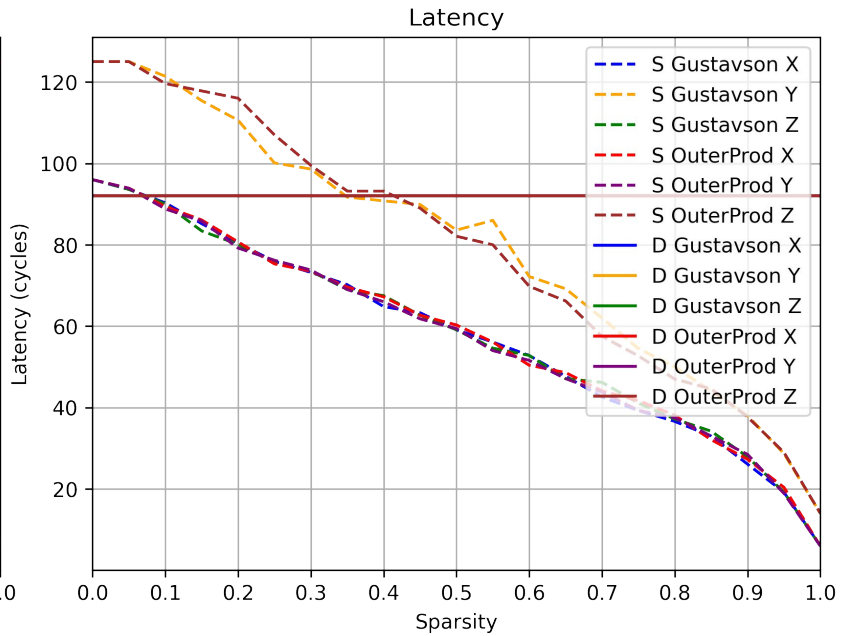


Designing a SpMM accelerator -Exploration (sparsity factor)

For the comparison between dataflows, I used the following default parameters: both X and Y matrices are 10×10, X_tile and Y_tile are set to 4, and the sparsity factor of X is 0.5. The Avg. BW refers to the combined bandwidth, including both input and output data from the accelerator, while Avg. Utilization indicates engine utilization across cycles, accounting for partial usage in some cycles due to sparsity. In the graphs, the prefixes “S” and “D” represent “sparse” and “dense”; in the dense version, X is stored in bitmap format, and its metadata is loaded in 32-bit chunks.





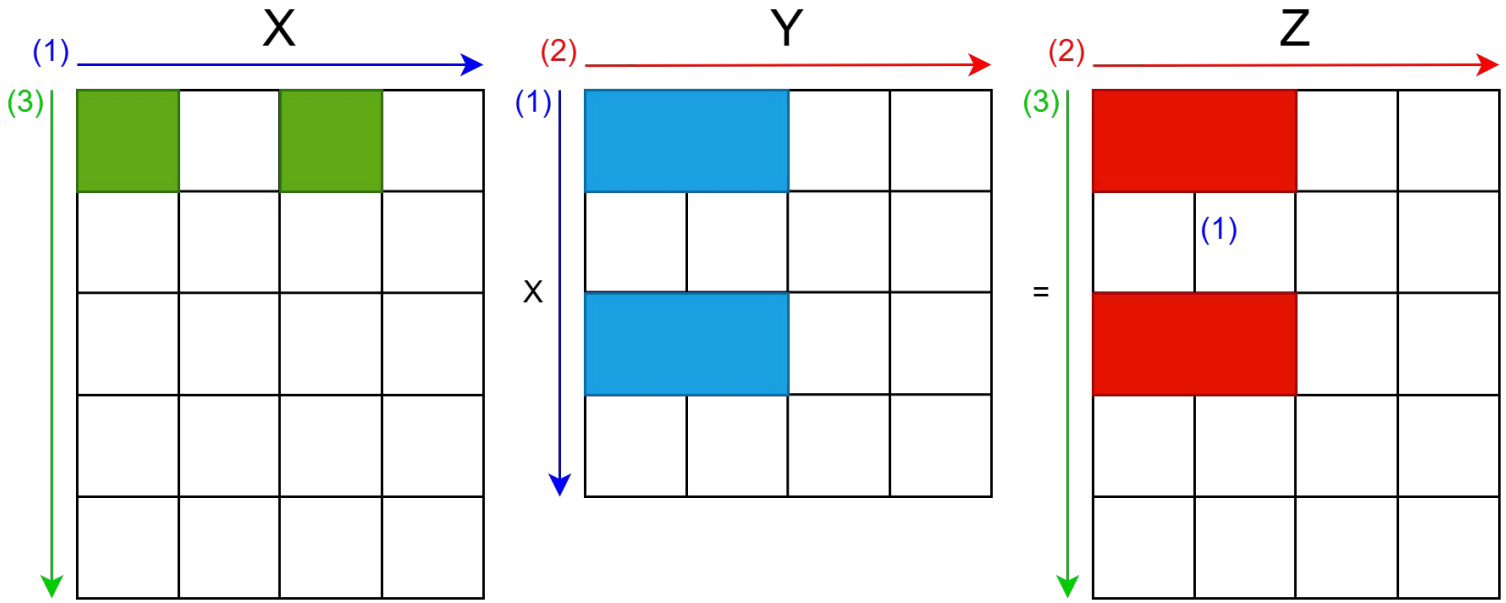


Designing a SpMM accelerator -Exploration (final decision)

Based on the data reported in the previous slides, the best overall dataflow appears to be Gustavson with Z stationary (Z reuse); it achieves the lowest Avg. BW and latency, along with the highest Avg. utilization.

This is intuitive, as the approach keeps the Z elements inside the accelerator until fully computed, avoiding storing and fetching partial results at each cycle. This reduces the Avg. BW, since each Z element is roughly three times larger than an X or Y element to accommodate all partial sums of the dot product between a row of X and a column of Y.

Gustavson sparse dataflow with Z stationary

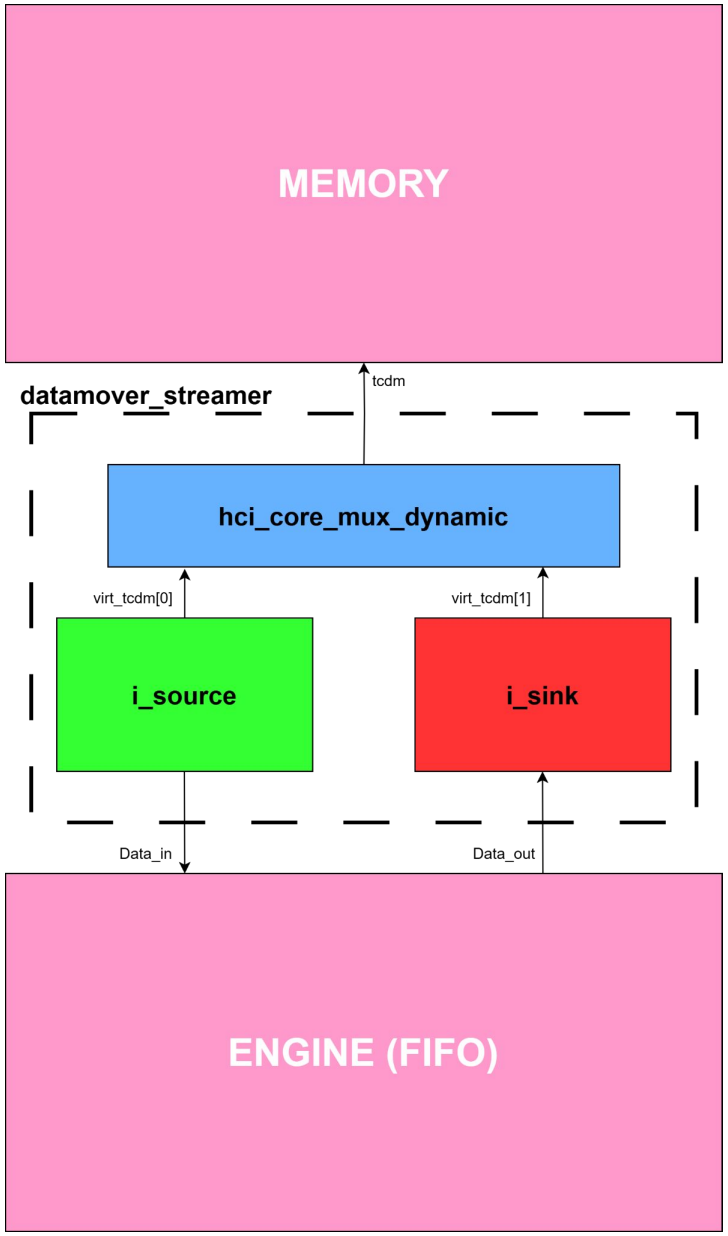


Gustavson sparse dataflow with Z stationary; the numbers and arrows refer to loop's order; 1 means "inner loop"

Designing a SpMM accelerator -RTL (overview)

The next step was to begin implementing the RTL. For this purpose, my supervisors provided me with a working datamover, which serves as a template accelerator designed solely to fetch data from memory and store it back.
The datamover is available at the following repository:
https://github.com/arpansur/hwpe-datamover-example/tree/prasadar/verif

My main objective is to modify the streamer that feeds data to the engine, ensuring that data is sent in the correct order as defined by the Gustavson Z dataflow, while also accounting for sparsity.
The engine itself will be simpler to implement, as it will consist of a block of MAC units.



Structure of the datamover

Designing a SpMM accelerator -RTL (the idea)

Loading data in the correct order while accounting for sparsity, maintaining low latency, and minimizing average bandwidth usage is a non-trivial task. My proposed solution is as follows:

Data Scheduler

There are three blocks, called Data Schedulers, responsible for generating requests to the source or sink so that the elements of X, Y, and Z are accessed in the correct order.

New Data

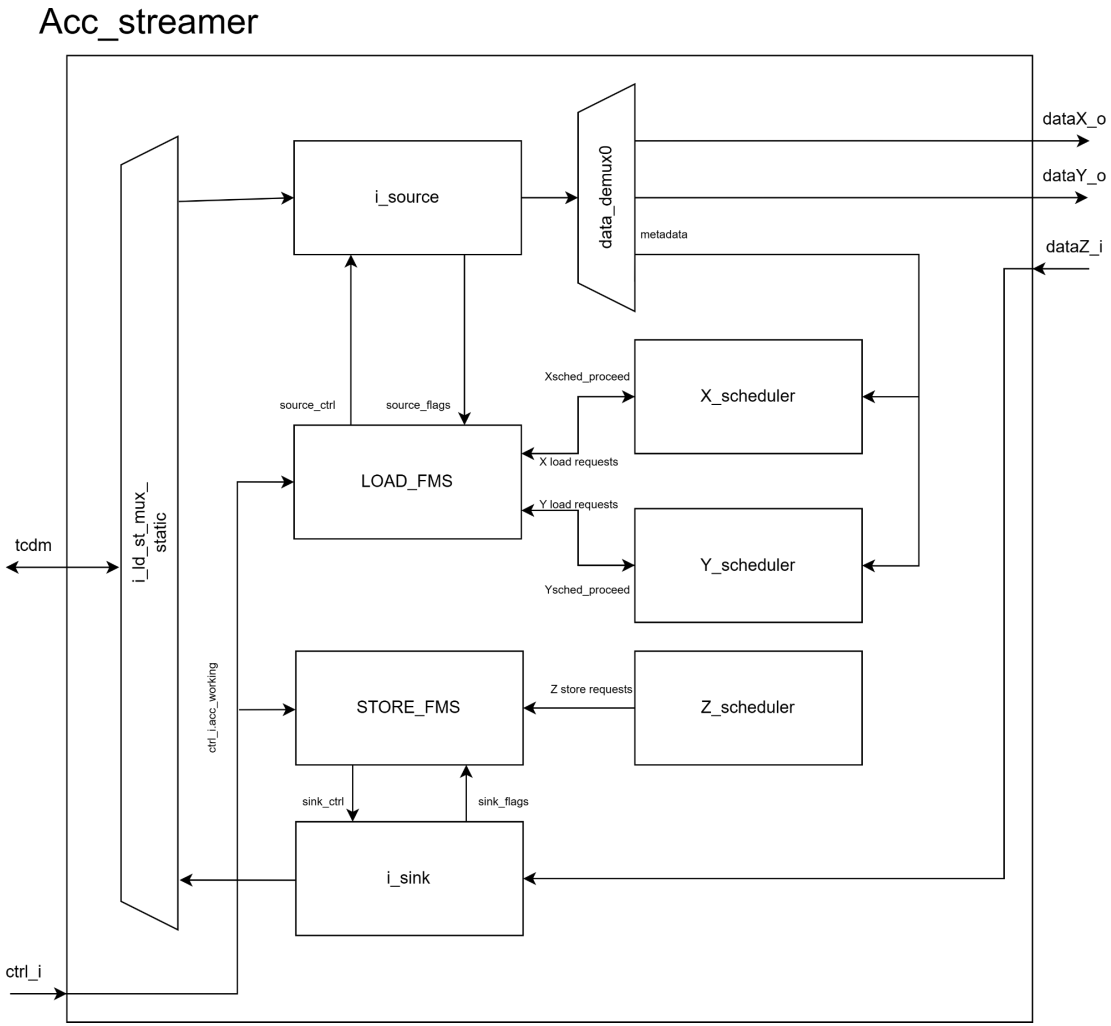
The load FSM continuously attempts to generate new data requests with the following priority: metadata, X, and Y. If the engine is not ready to receive a given type of data, the FSM moves on to the next one in the priority list. Since X is stored in bitmap format, all its dense elements are contiguous in memory. This allows a single memory request to fetch enough data to sustain multiple multiplication cycles of the engine.
In contrast, due to sparsity, the elements of Y are not contiguous. The row blocks must be fetched individually, each typically sufficient for only one multiplication cycle. Therefore, when the engine has enough X elements to operate for *n* cycles, only Y elements are loaded during that time.

New Metadata Request

When the current metadata chunk has been fully consumed, the schedulers pause and issue a request for new metadata before proceeding.

Designing a SpMM accelerator -RTL (streamer microarch.)

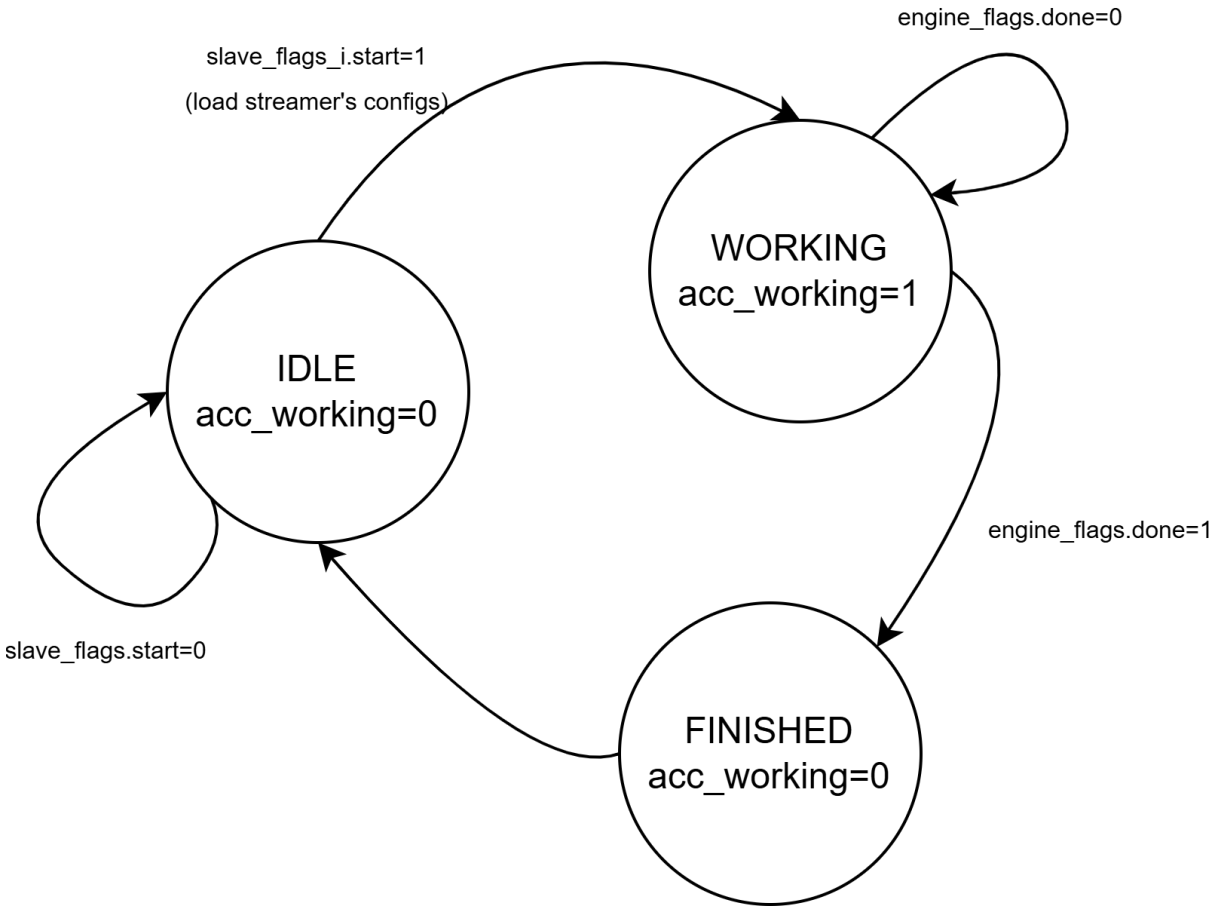
This is the streamer microarchitecture I came up with; X, Y and Z data on the right will be connected to the engine, while ctrl_i will be connected to a slave in the top module that communicates with the CPU.



Streamer's microarchitecture

Designing a SpMM accelerator -RTL (Status FSM)

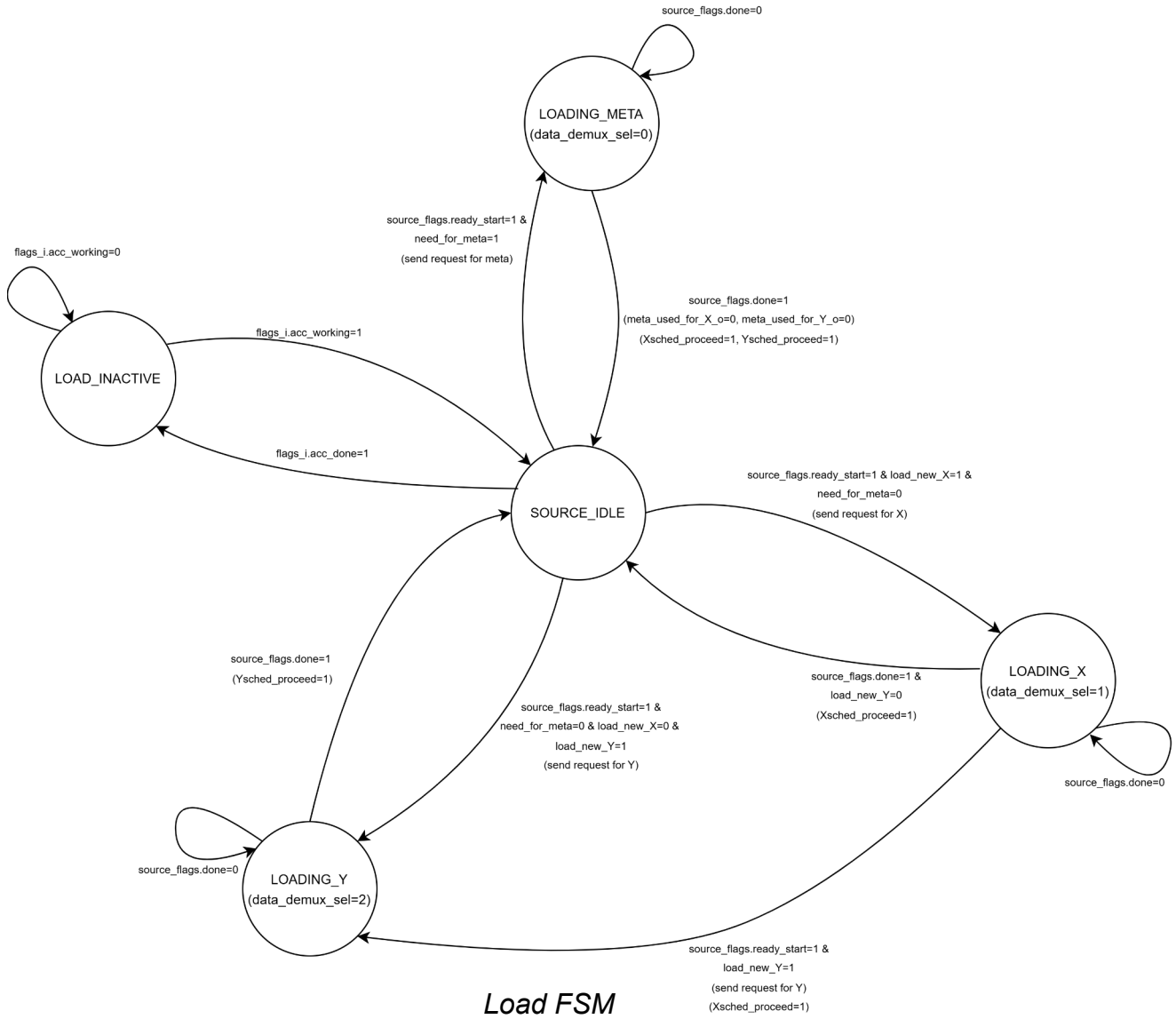
This is how the accelerator is controlled by the CPU; the communication between the two is mediated by a slave and this FSM is in the top module of the accelerator.



Accelerator's status FSM

Designing a SpMM accelerator -RTL (Load FSM)

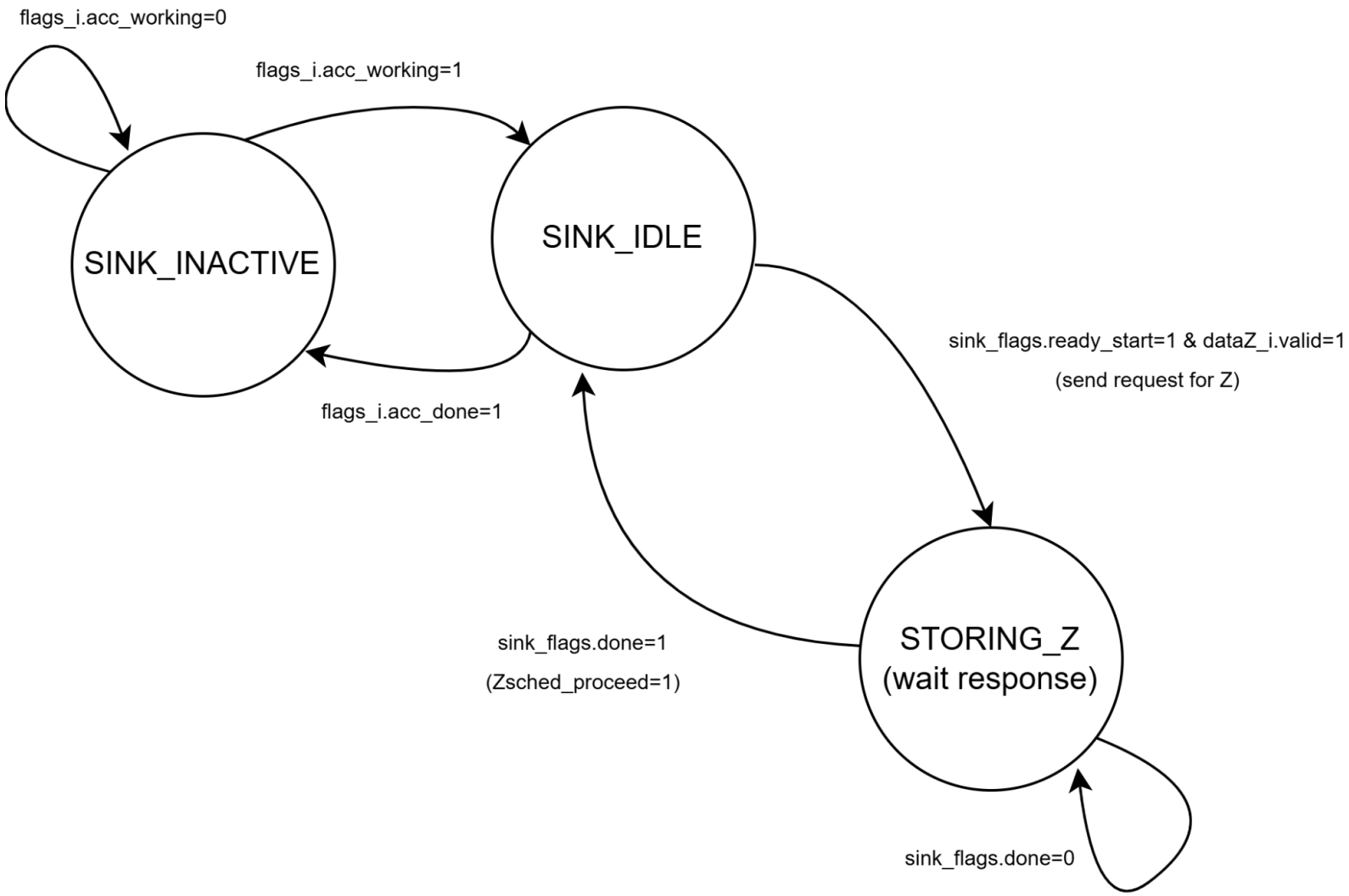
This is how a load request is made by the streamer; the flag “ready_start” from the source means that it can receive a new request



Load FSM

Designing a SpMM accelerator -RTL (Store FSM)

This is how a store request is made by the streamer; this also happens virtually at each cycle



Store FSM

Designing a SpMM accelerator -The future

This is the result of my first 5 weeks of work on the SpMM accelerator. I am currently implementing changes to the RTL. After that, I will proceed with the following steps:

- **Build a testbench**: To verify the streamer, I will develop a dedicated testbench.
- **Build golden models**: These will be used to generate the test stimuli.
- **Verification**: I will debug the module using the testbench and golden models.
- **Implement the engine**: I will design the engine and adapt the testbench accordingly.
- **Update golden models**: I will modify the models to validate the full accelerator functionality.
- **Final verification**: I will verify the complete system against the updated golden models.
- **Post-synthesis analysis**: I will synthesize the accelerator to estimate area and power consumption.

This project spans a total of 14 weeks and is scheduled to finish on July 29. I will continue to update the repository regularly on my GitHub at:
https://github.com/Ferrons02/Exploring-Outer-Product-and-Gustavson-Dataflows-for-SpMM-Acceleration

ETH zürich

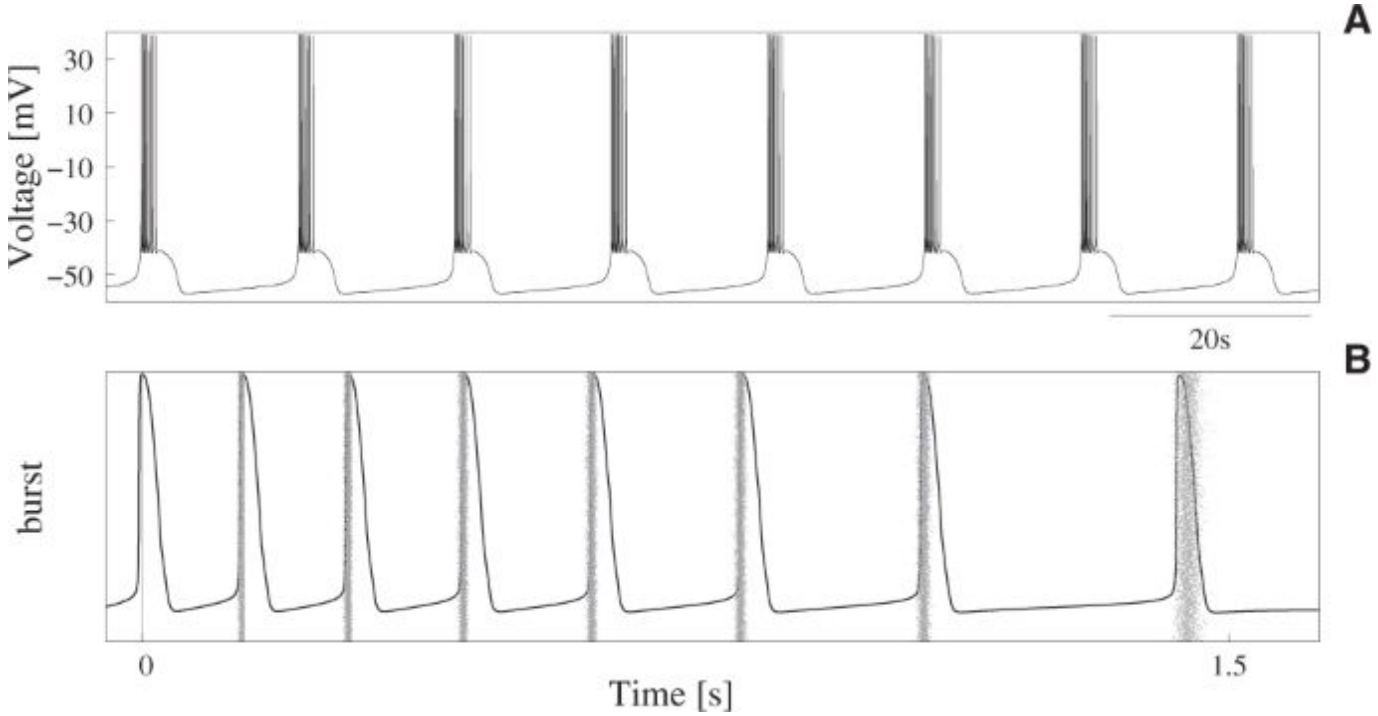
Fine-Tuning and Layout of a Biologically Plausible Bursting Neuron Circuit

Bursting Neuron’s circuit Layout -Overview

As part of the course “Neuromorphic Engineering 2”, I did the fine-tuning and layout of a biologically plausible circuit of a bursting neuron. The schematic was provided to all the students of the course, but we were responsible for sizing the transistors and tuning the parameters required for the bursting behavior to emerge.

The complete report of the project is available on my github page under: https://github.com/Ferrons02/Bursting_Neuron_Circuit

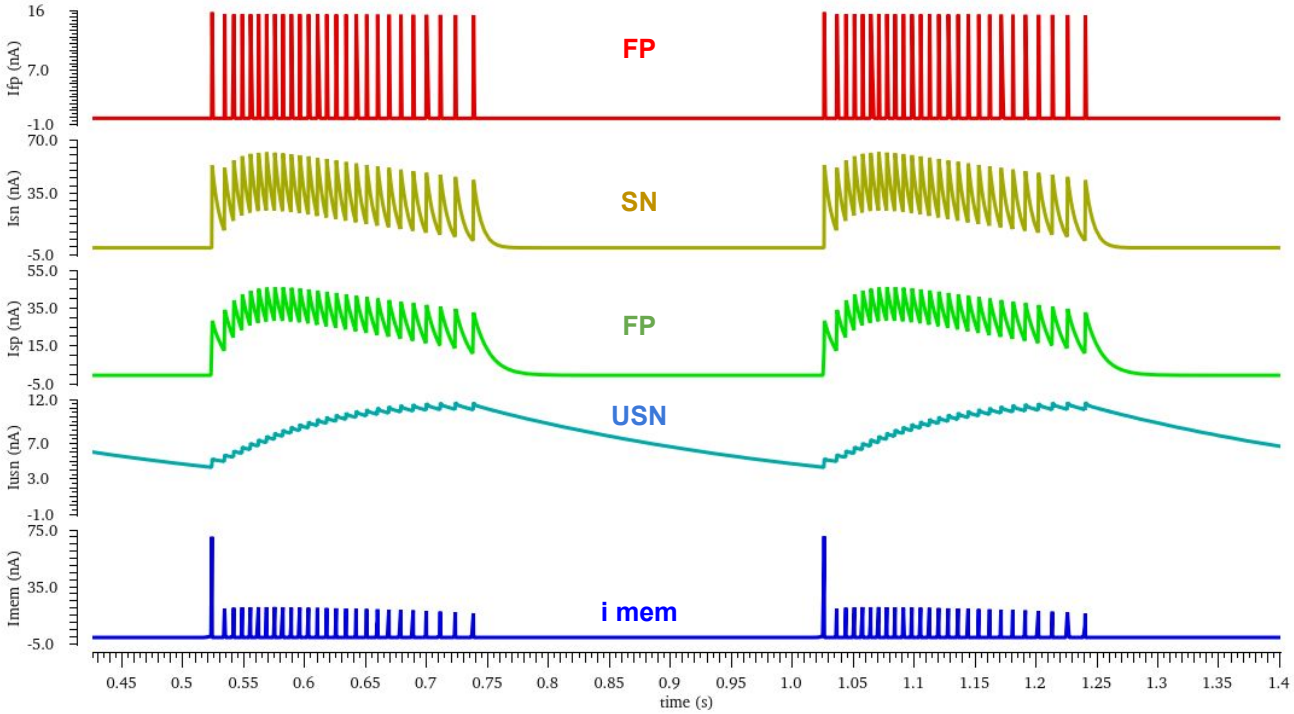
Bursting neurons play a key role in motor coordination and sensory perception. Their defining feature is that they communicate via trains of spikes, as shown in the following example:



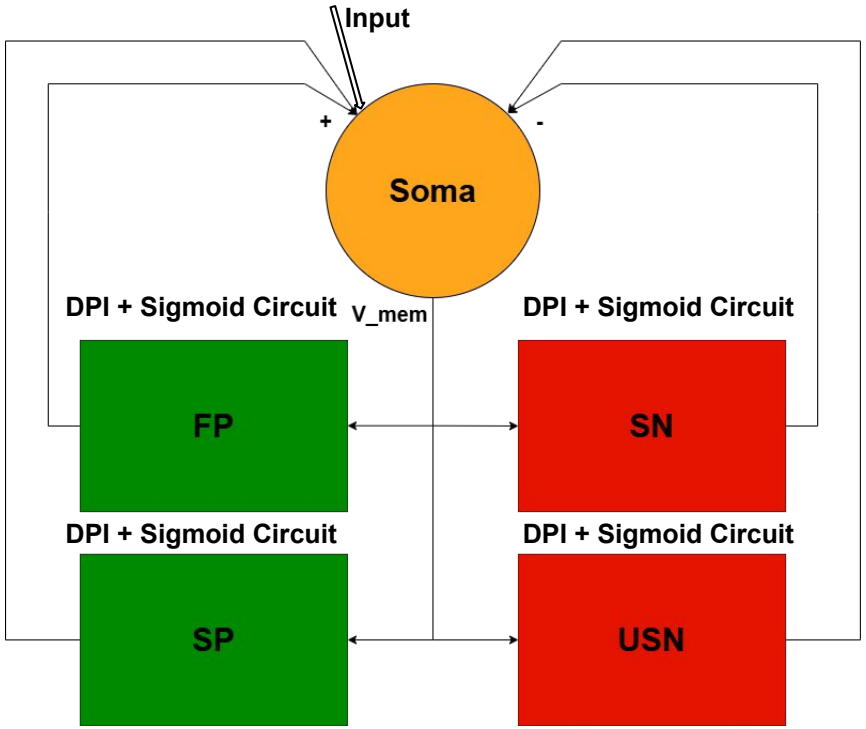
Membrane potential during bursts

Circuital implementation - Working principles

The model includes four feedback mechanisms that regulate membrane charge—and thus its potential: fast positive (FP), slow negative (SN), slow positive (SP), and ultraslow negative (USN). The positive mechanisms add charge to the membrane, while the negative ones remove it. When a DC input current is applied, the membrane starts to charge, activating the FP mechanism and triggering a spike in the membrane potential (V_mem). At the same time, the SN mechanism is activated, counteracting the spike by removing charge and thereby suppressing it. Once the spike is suppressed, V_mem returns to baseline, and both FP and SN rapidly decay. However, because SP decays more slowly than SN, it eventually dominates, initiating a new spike. Meanwhile, the USN mechanism gradually builds up, ultimately suppressing further spiking and ending the burst.



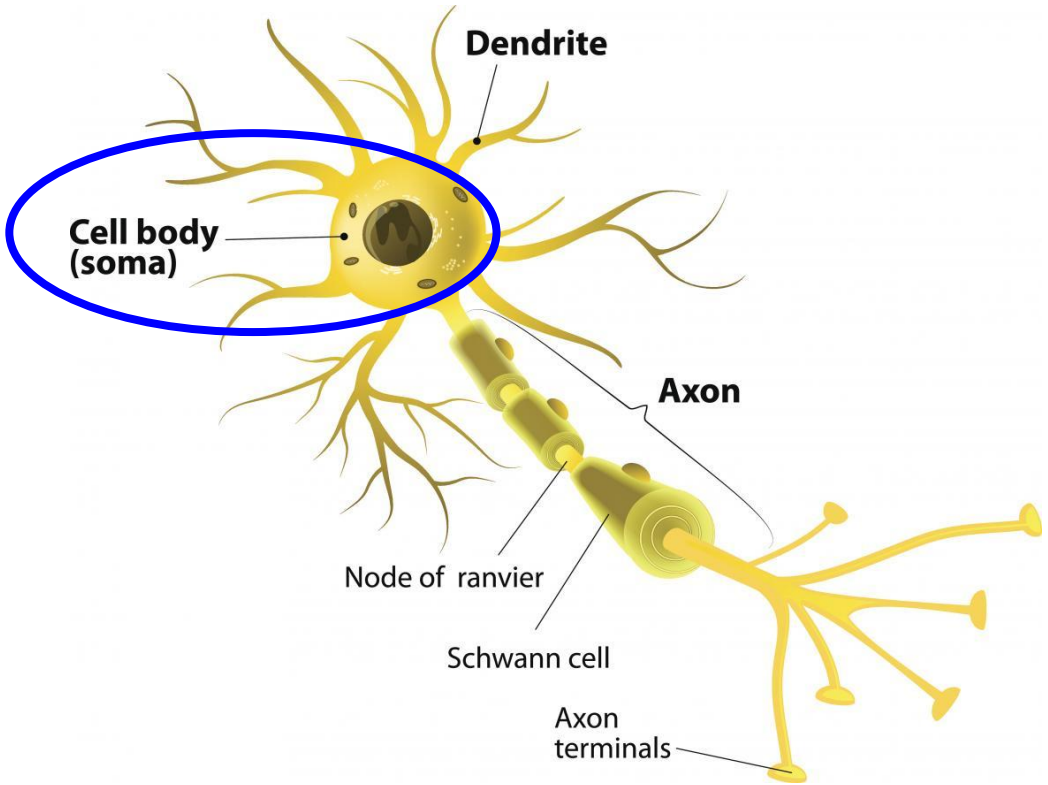
Bursting activity in our circuit



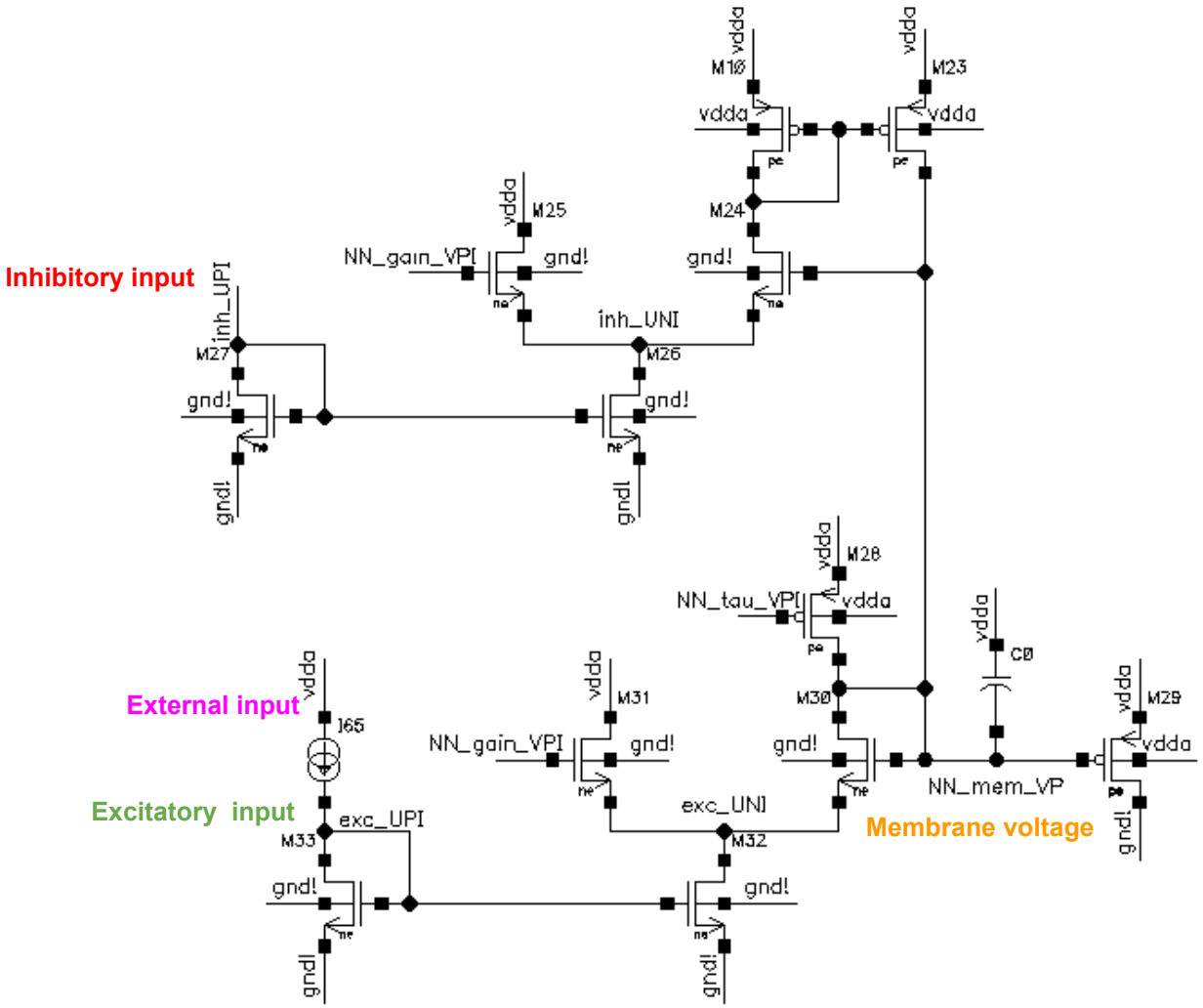
Model used

Building blocks - Soma

It merges the currents of the four feedback mechanisms; the top part is for the inhibitory contributions while the bottom part for the excitatory ones.



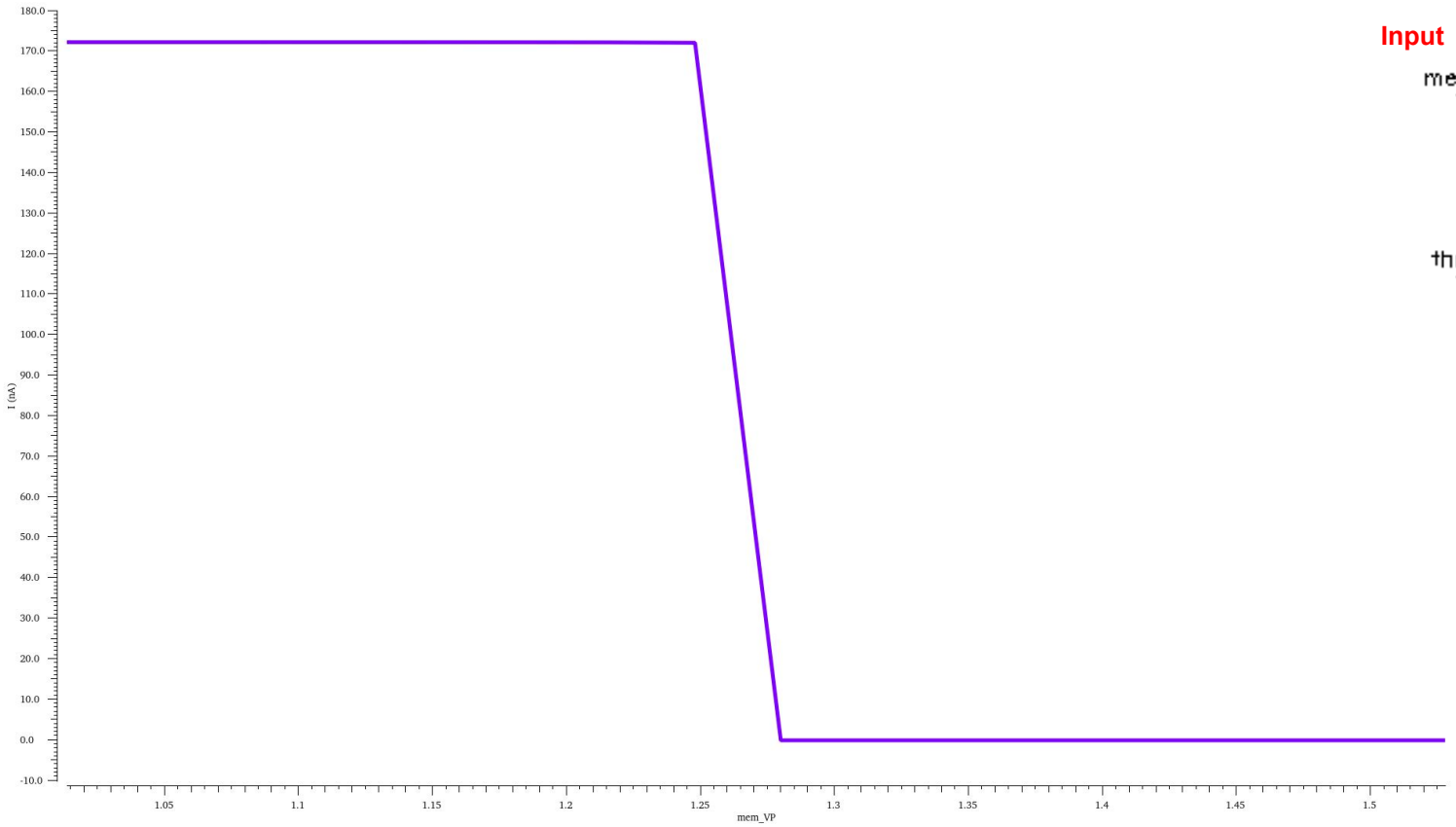
Scheme of a neuron



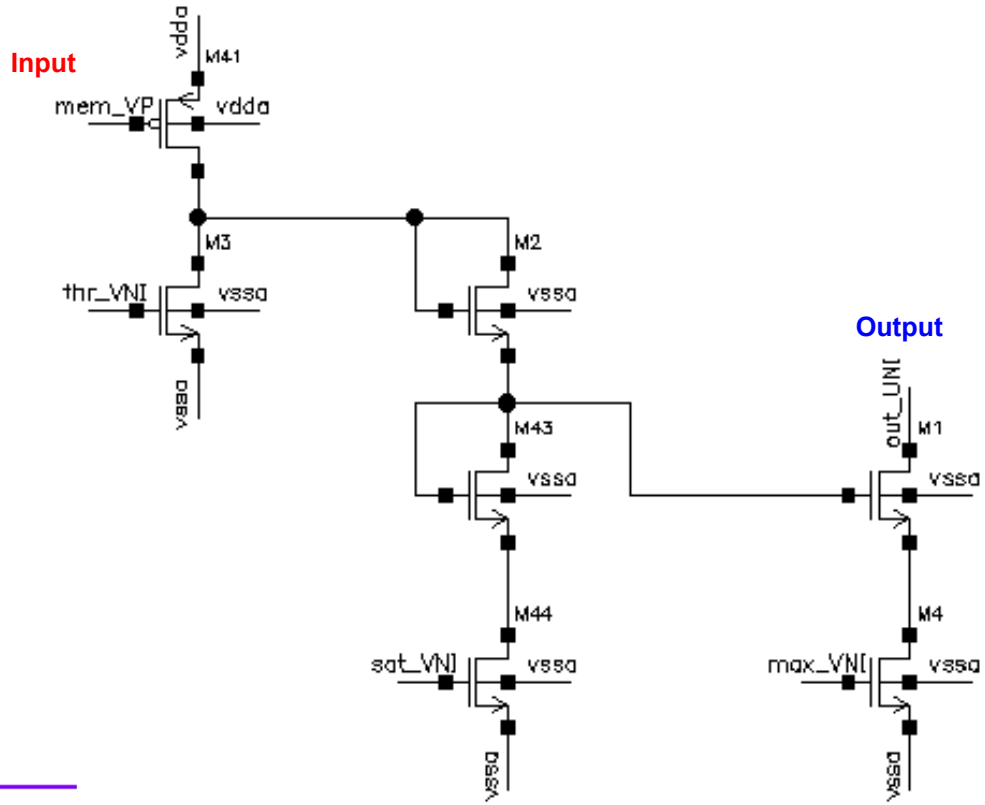
Schematic of the soma circuit

Building blocks - Sigmoid

It acts as a sigmoidal activation function with respect to the input voltage to simulate the nonlinear response of the neuron



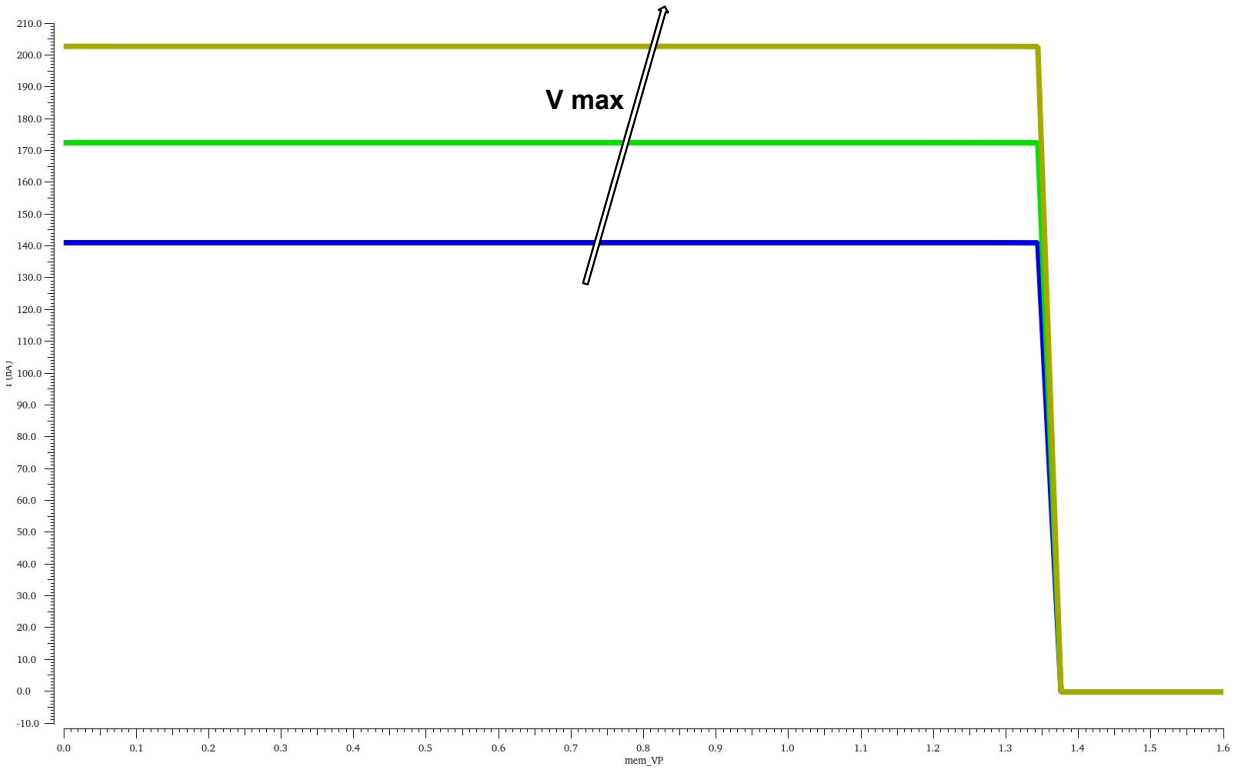
Output current vs the input voltage



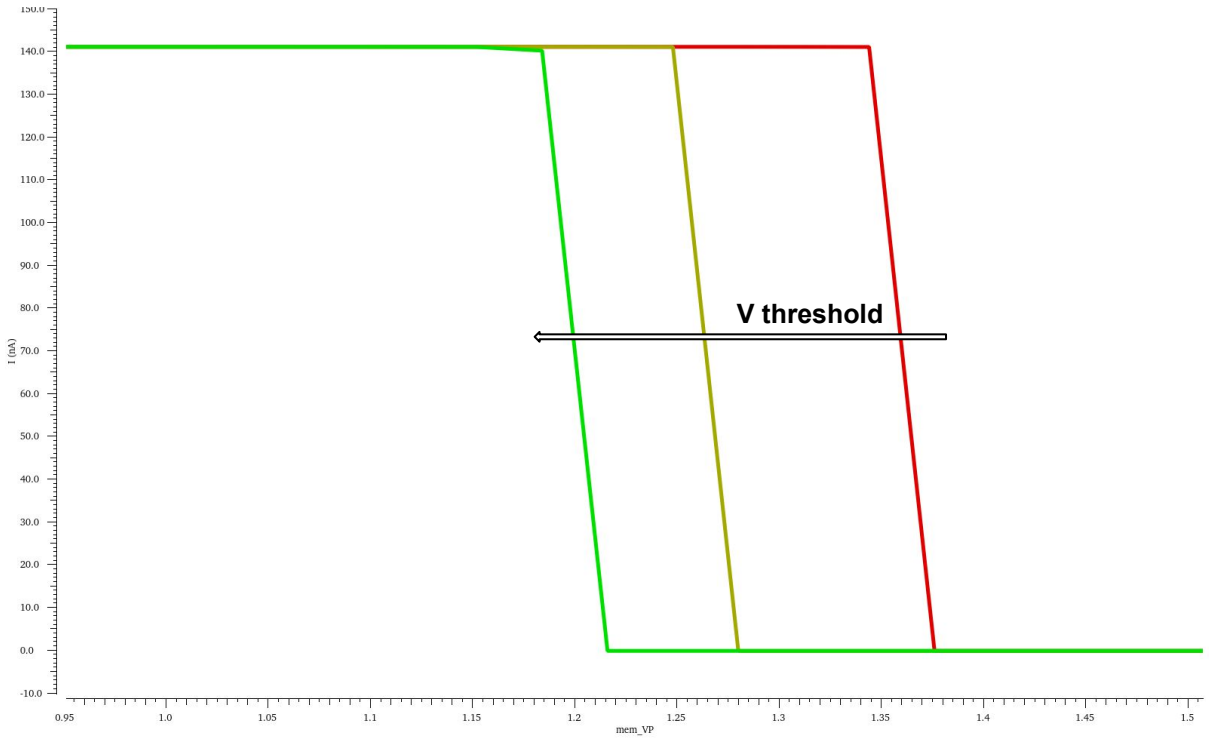
Schematic of the sigmoid circuit

Building blocks - Sigmoid simulations

I tuned the maximum value of the sigmoid through the **V_max** input and the threshold of the sigmoid through the **V_threshold** input. With the **V_saturation** input, one can also modify the slope of the transition, however since it has the same function as the gain of the DPI, I kept it fixed when tuning the circuit.



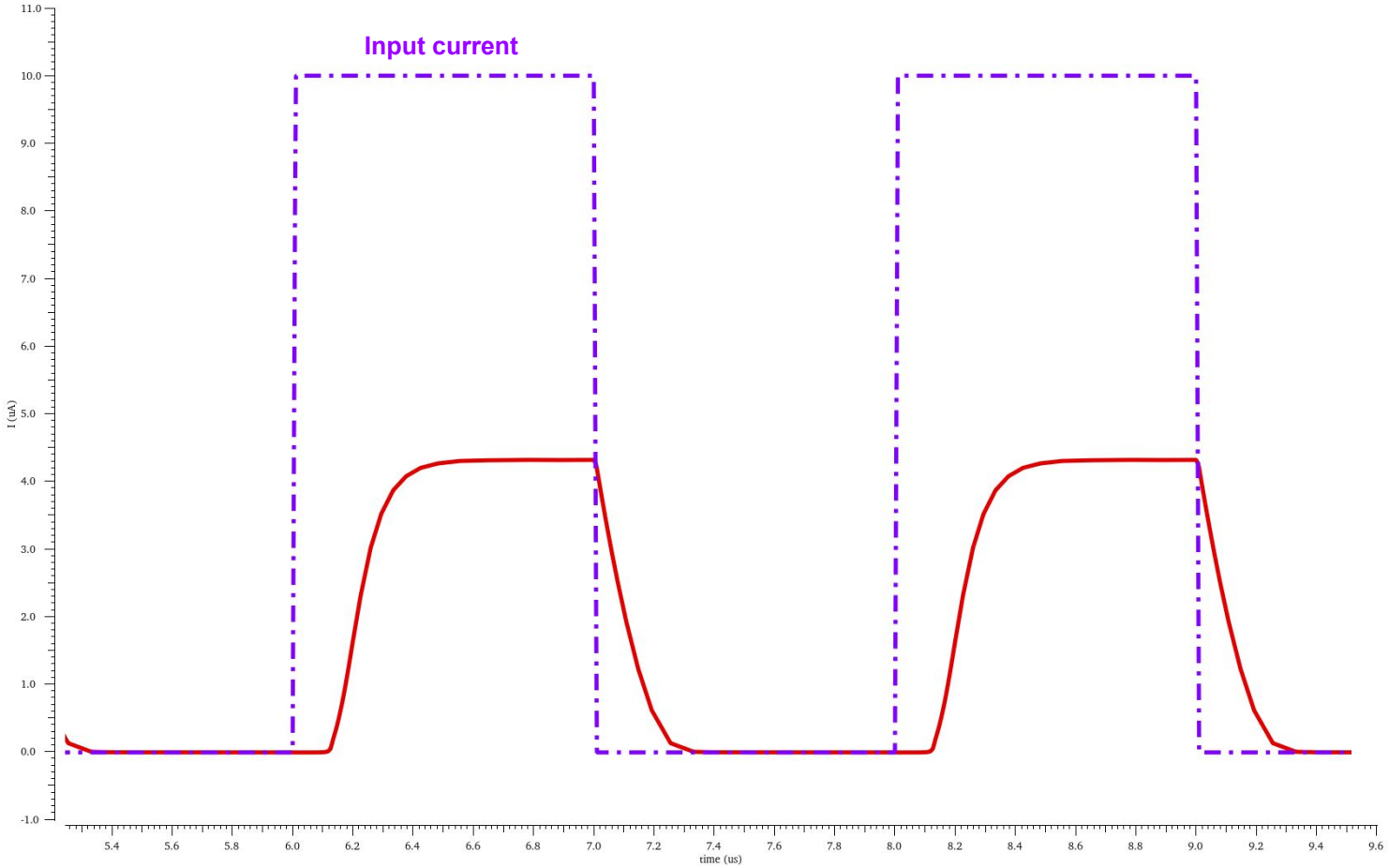
Effect of the max



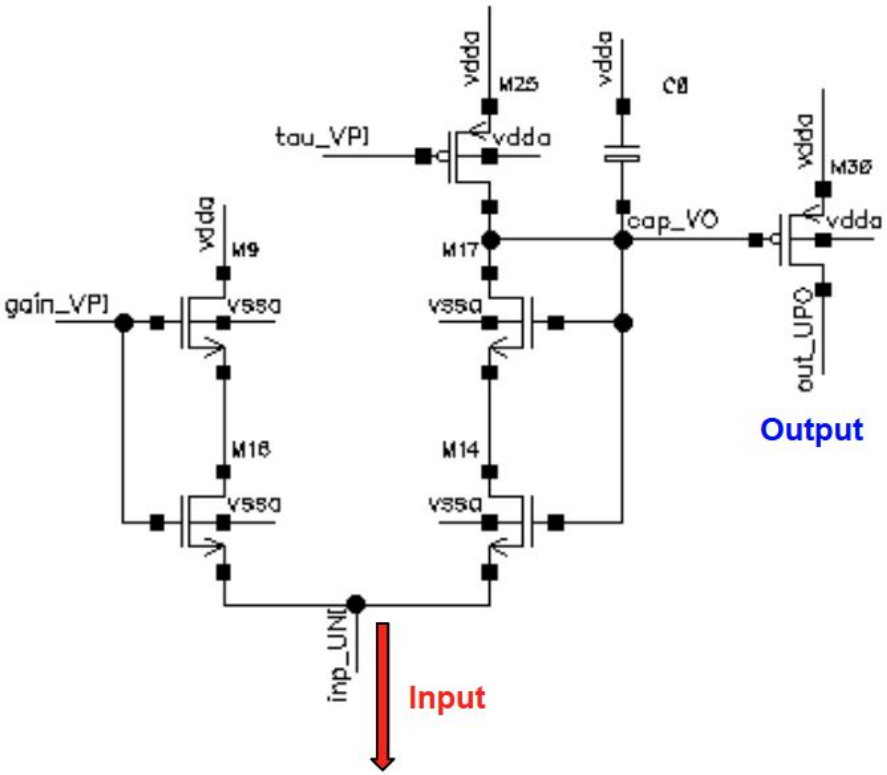
Effect of the threshold

Building blocks - DPI

It integrates the input signal, acting as a memory that represents the intensity of each mechanism



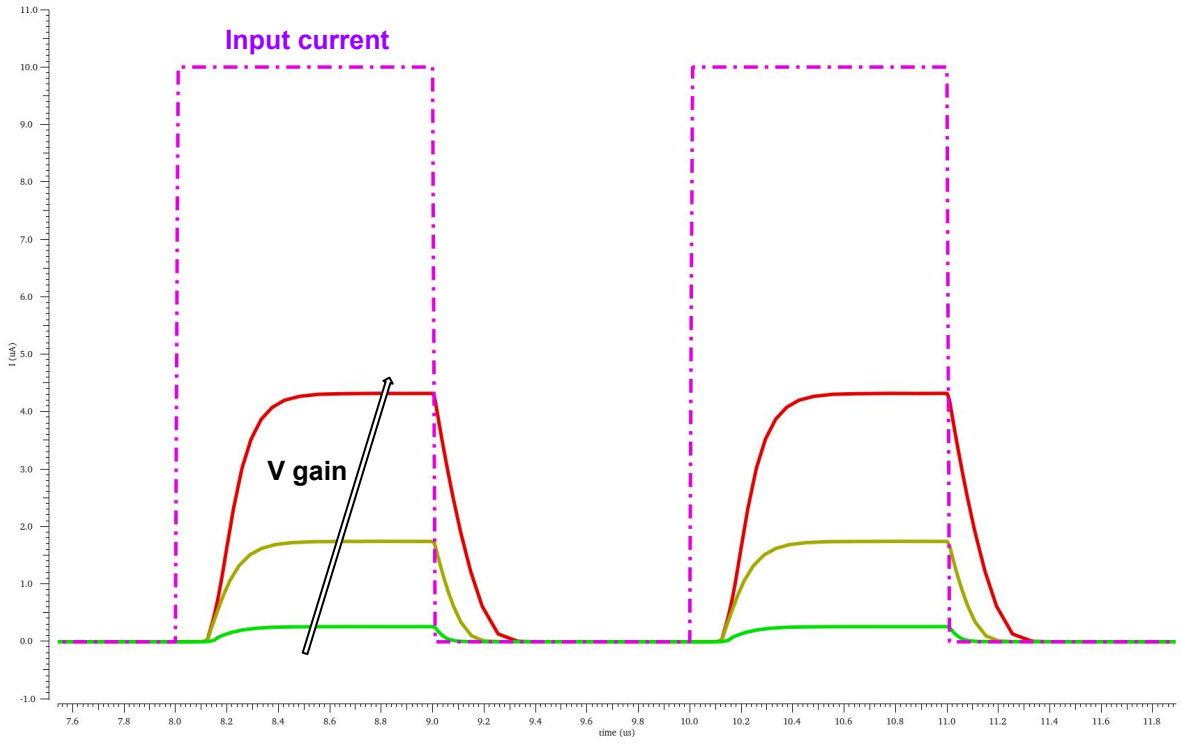
Output current with a pulsed input



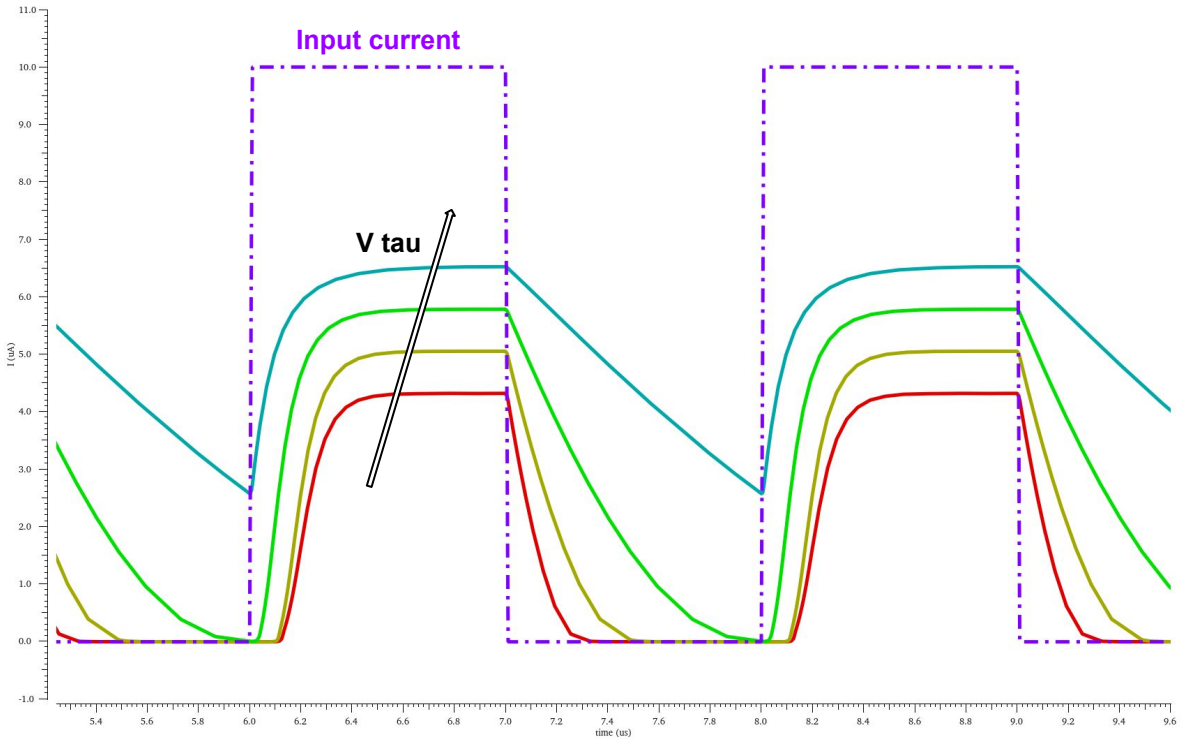
Schematic of the DPI

Building blocks - DPI simulations

I've tuned its response by changing two parameters; the **tau**, responsible for the decaying speed and the **gain**, which influences the maximum value reached



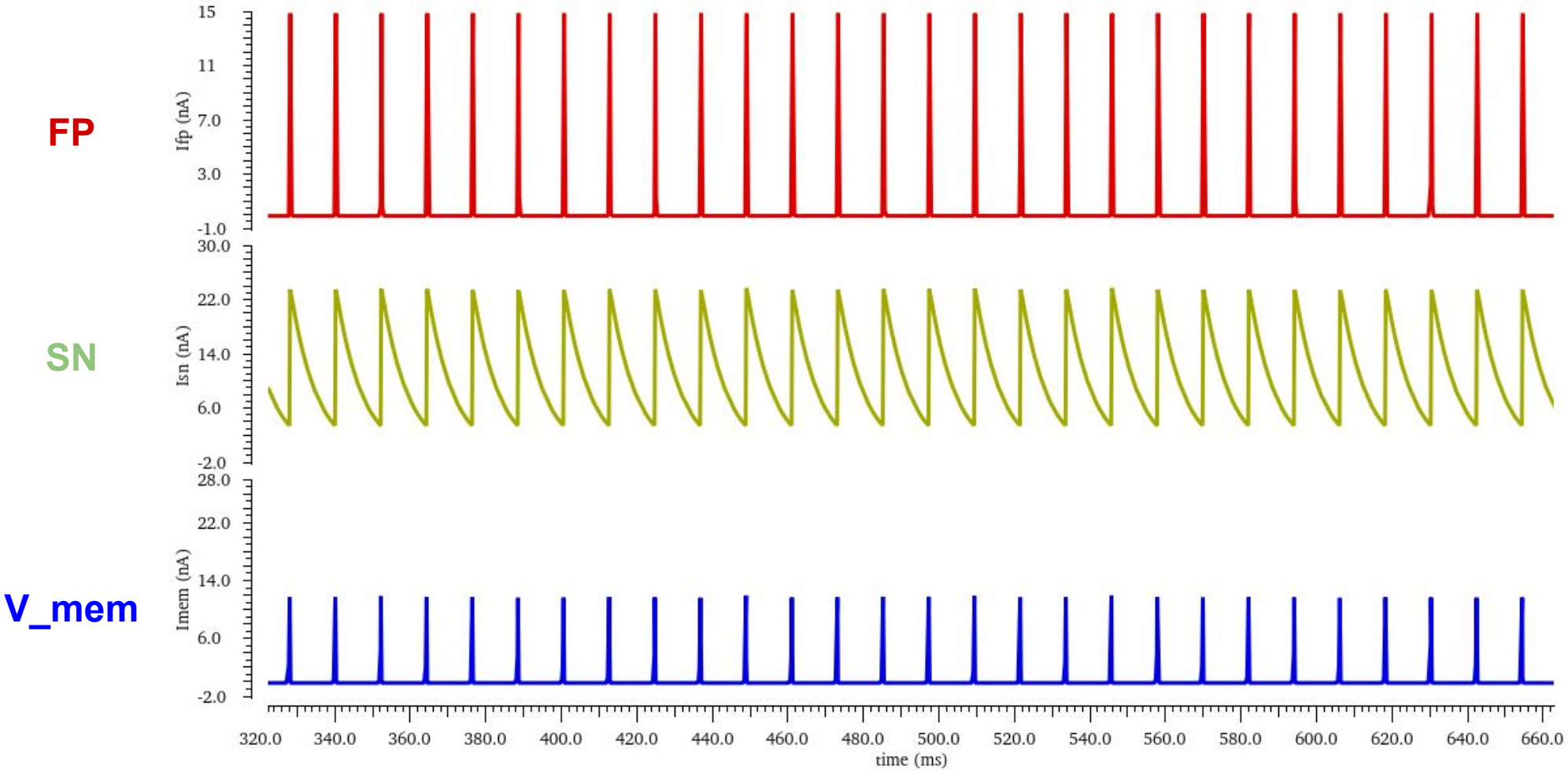
Effect of the gain



Effect of the tau

Pre-layout simulations - Monotone Spiking

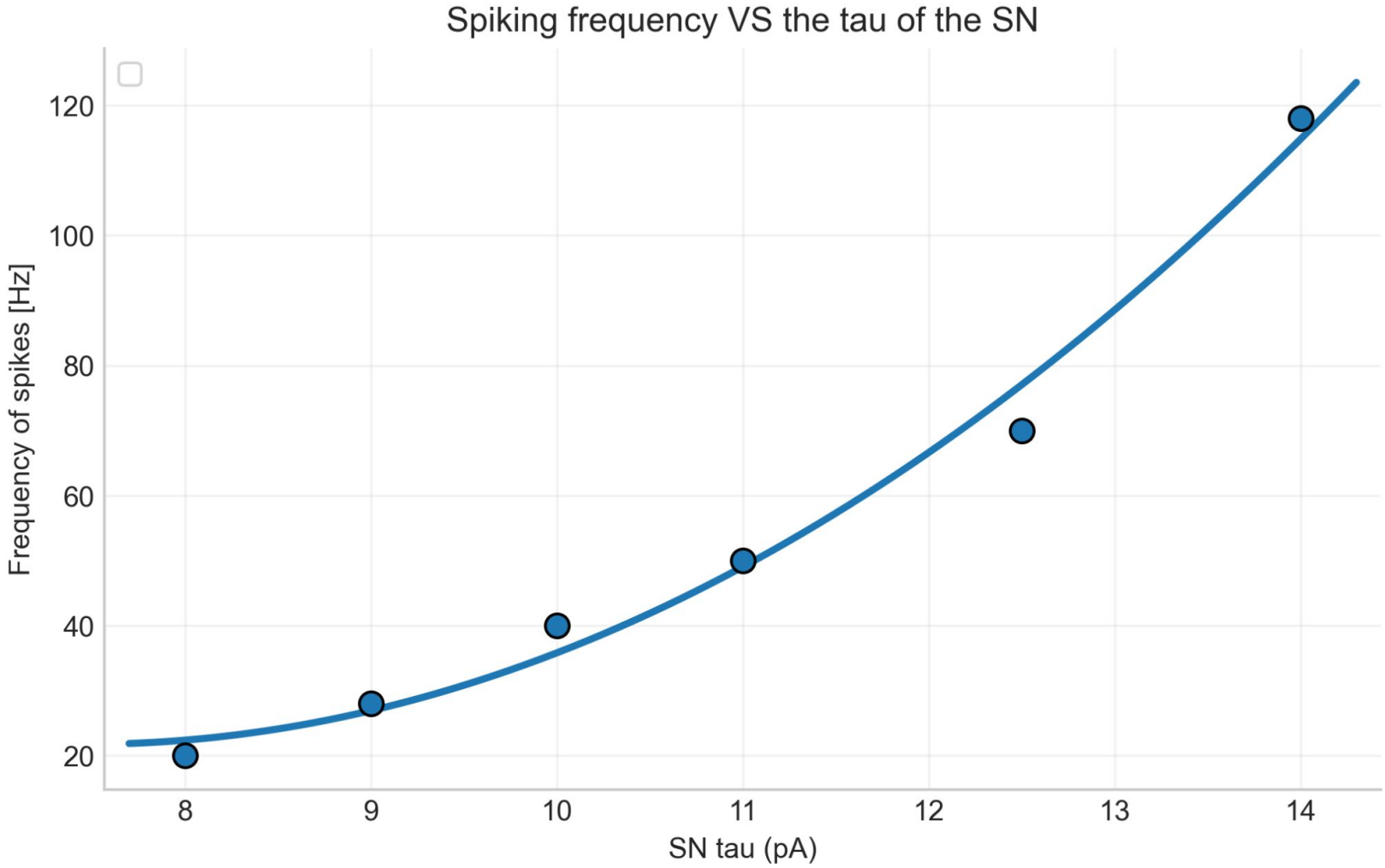
The first step was to achieve monotone bursting, which I was able to do by keeping only the FP and SN feedback mechanisms and finding the right parameters for their DPI and sigmoid circuits.



Monotone bursting activity in our circuit

Pre-layout simulations - Monotone Spiking (freq. mod.)

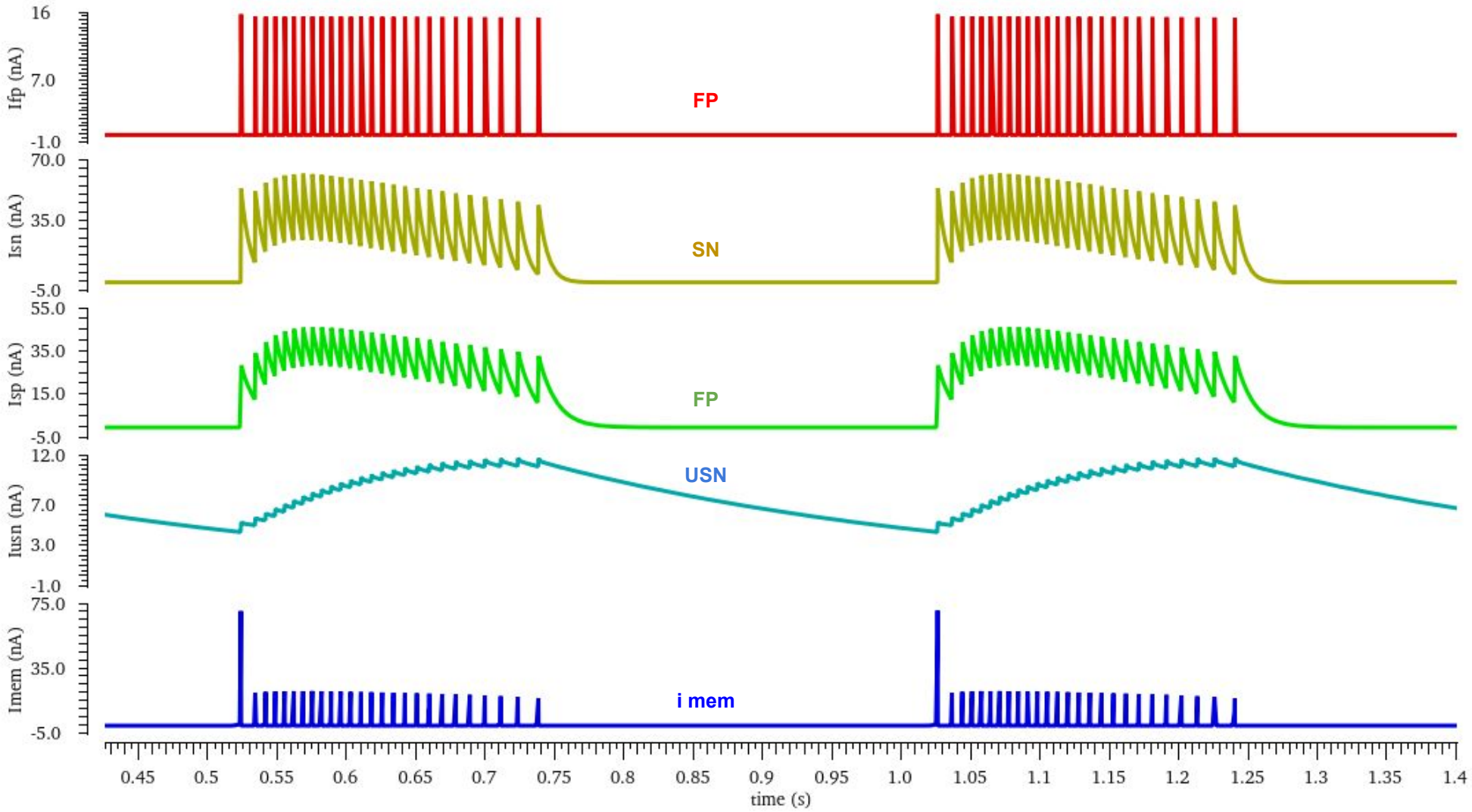
I've tried to tune the frequency of the monotone bursting through the tau of the SN feedback mechanism, finding an almost-linear relation between the two quantities



Monotone bursting frequency vs the tau of the SN mechanism

Pre-layout simulations - Bursting

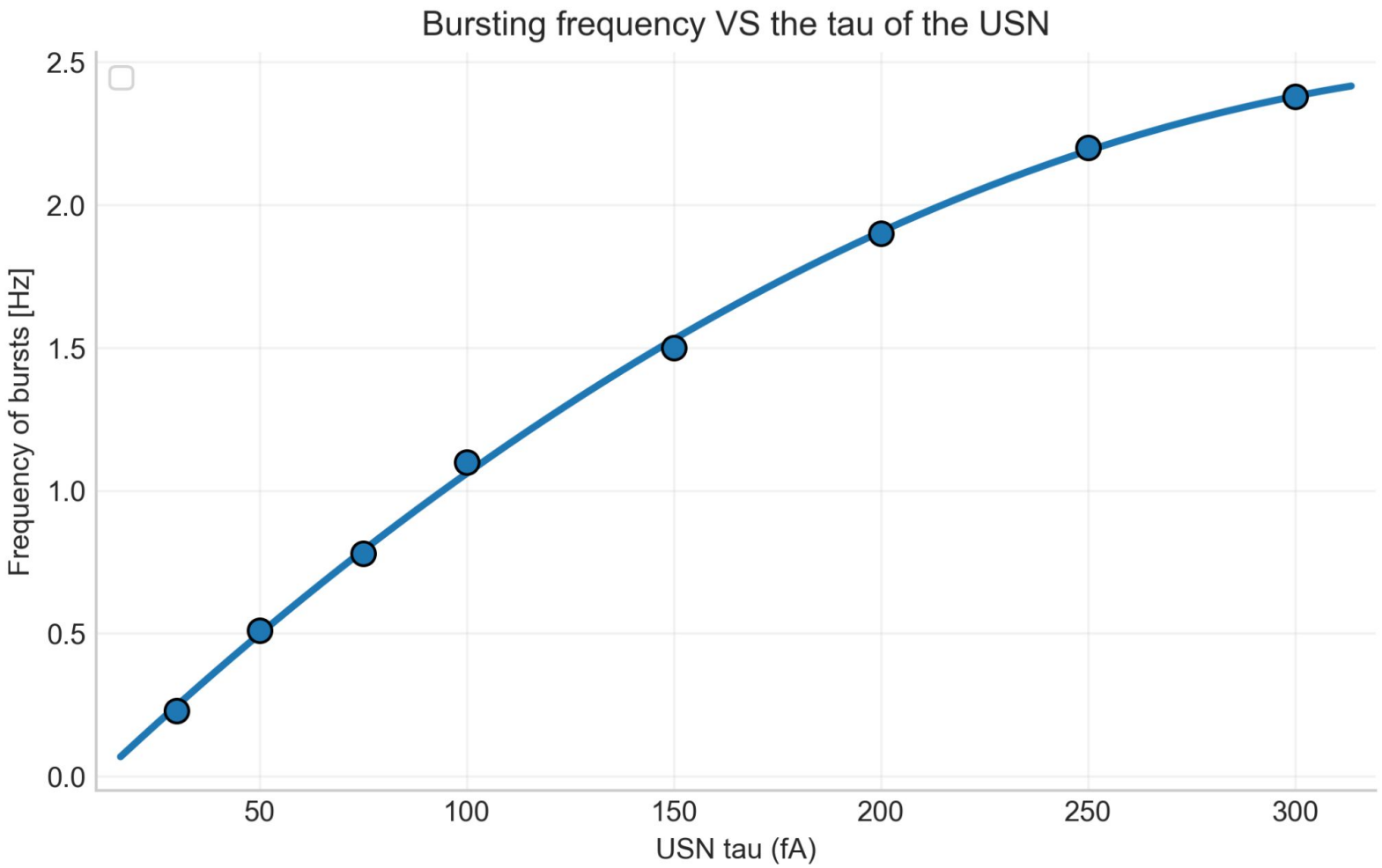
After adding the USN and SP feedback mechanisms I was able to see the bursting activity



Bursting activity in our circuit

Pre-layout simulations - Bursting frequency

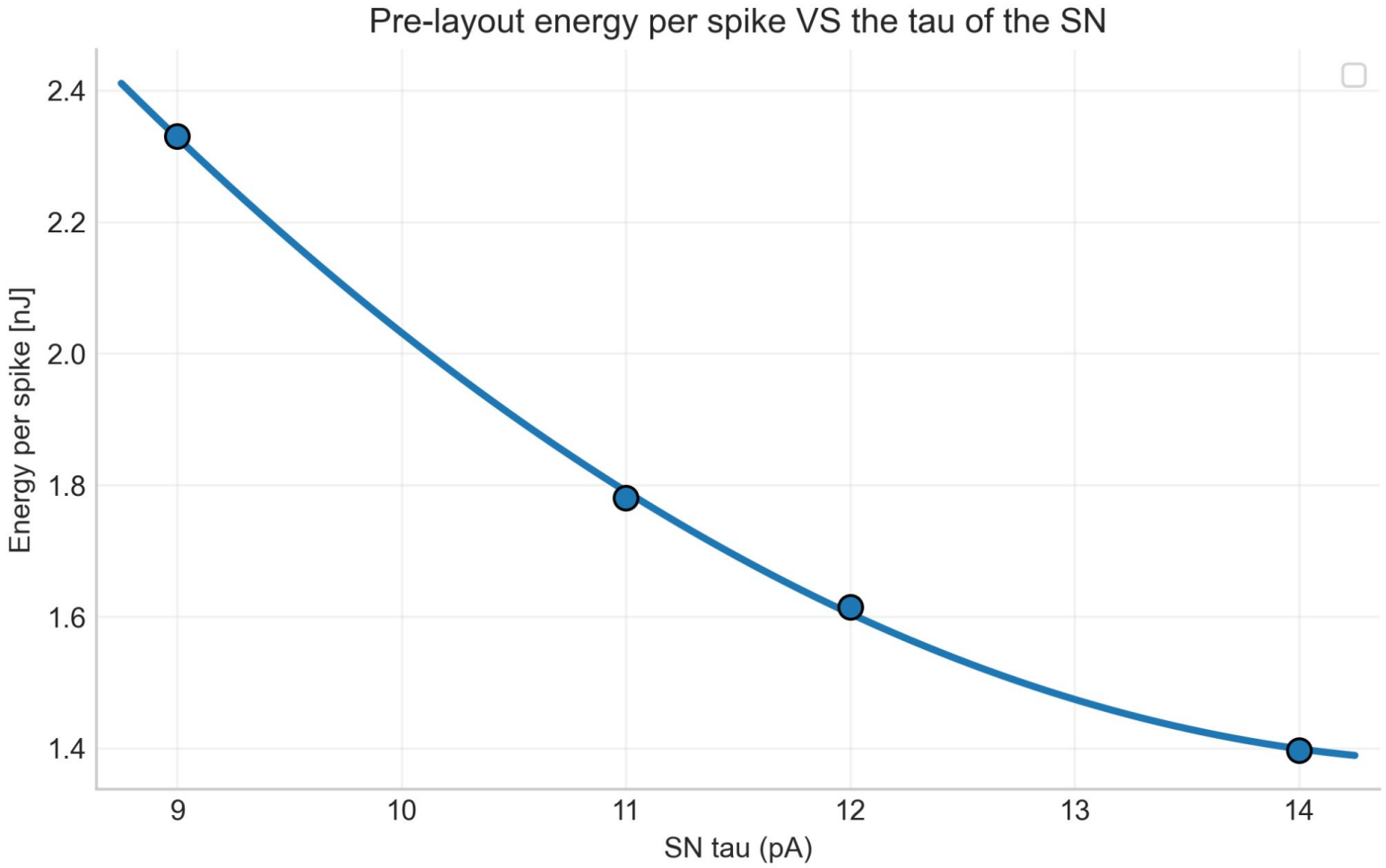
The bursting frequency can be tuned by changing the tau of the USN feedback which influences the length of the refractory period



Bursting frequency vs the tau of the USN feedback

Pre-layout simulations - Energy per spike

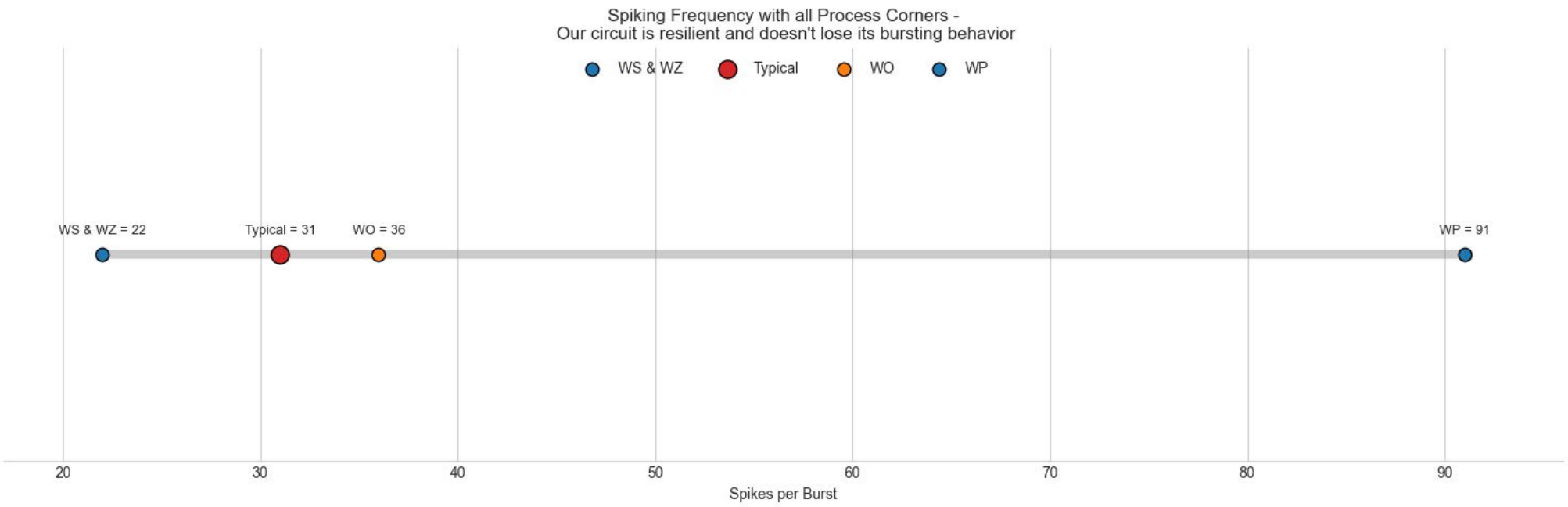
I found out that the energy consumption for each spike is in the order of few nJ and decreases with the tau of the SN since it makes the spikes more narrow.



Pre-layout energy per spike vs the tau of the SN feedback

Pre-layout simulations - Corner analysis

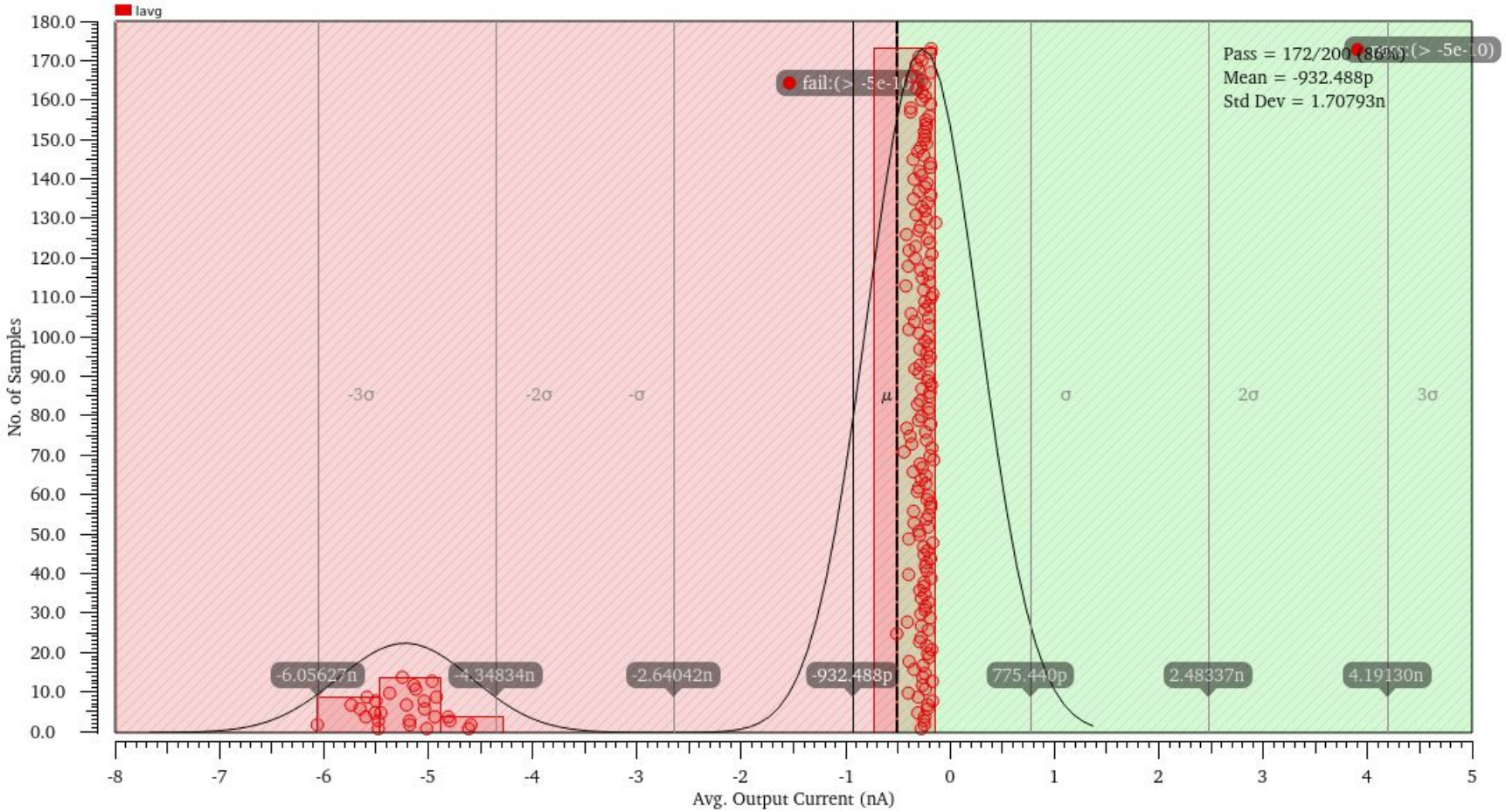
In the corner analysis the circuit turned out to still work at every corner but the number of spikes in each burst was affected



Avg number of spikes for each process corner

Pre-layout simulations - Montecarlo simulation

To ensure the robustness of my circuit to process variations and mismatch, I performed a Monte Carlo simulation plotting the average output current. I've observed that if the current exceeds (in absolute value) a certain threshold, the circuit is not bursting but either spiking monotonically or not functioning at all. Initial results showed a success rate of around 60%, but I improved it by reducing the sizes of some critical transistors, reaching an 86% success rate.



Montecarlo simulation results; each dot is a trial

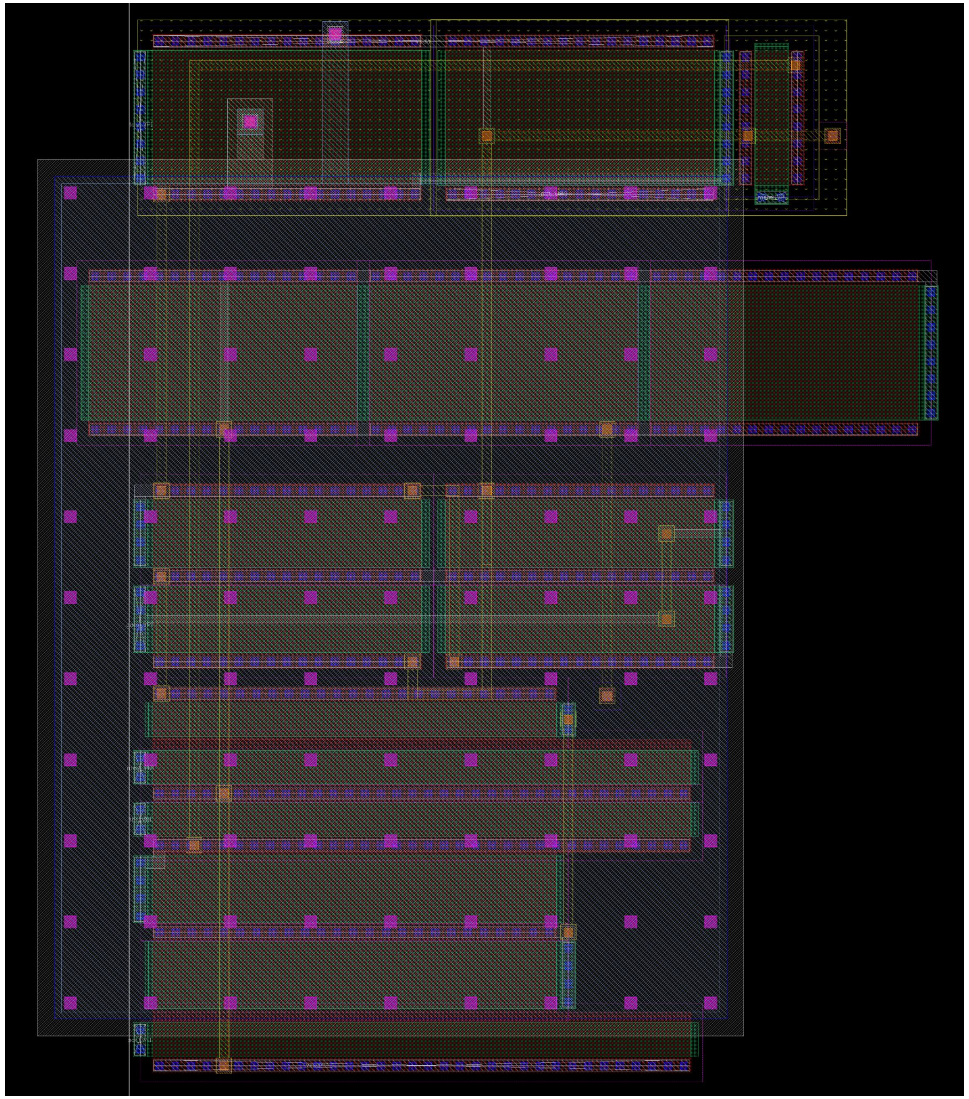
Circuital implementation - Feedback layout

To manage the circuit complexity, I decided to create the layout of each feedback mechanism separately and then reuse it four times in the final schematic.

I applied a multiplier of 2 to create a common-centroid arrangement for the transistors of the two branches of the DPI, since they had a big contribution in the mismatch and needed to be as closely matched as possible.

Additionally, the three NMOS transistors arranged horizontally are dummy devices that separate the PMOS and NMOS transistors and will form part of a larger ring once the layout is complete.

The unusual arrangement is designed to create space for the soma circuit in the center, as shown in the final layout on the next slide.



Layout of one of the feedback mechanisms

Circuital implementation - Final layout

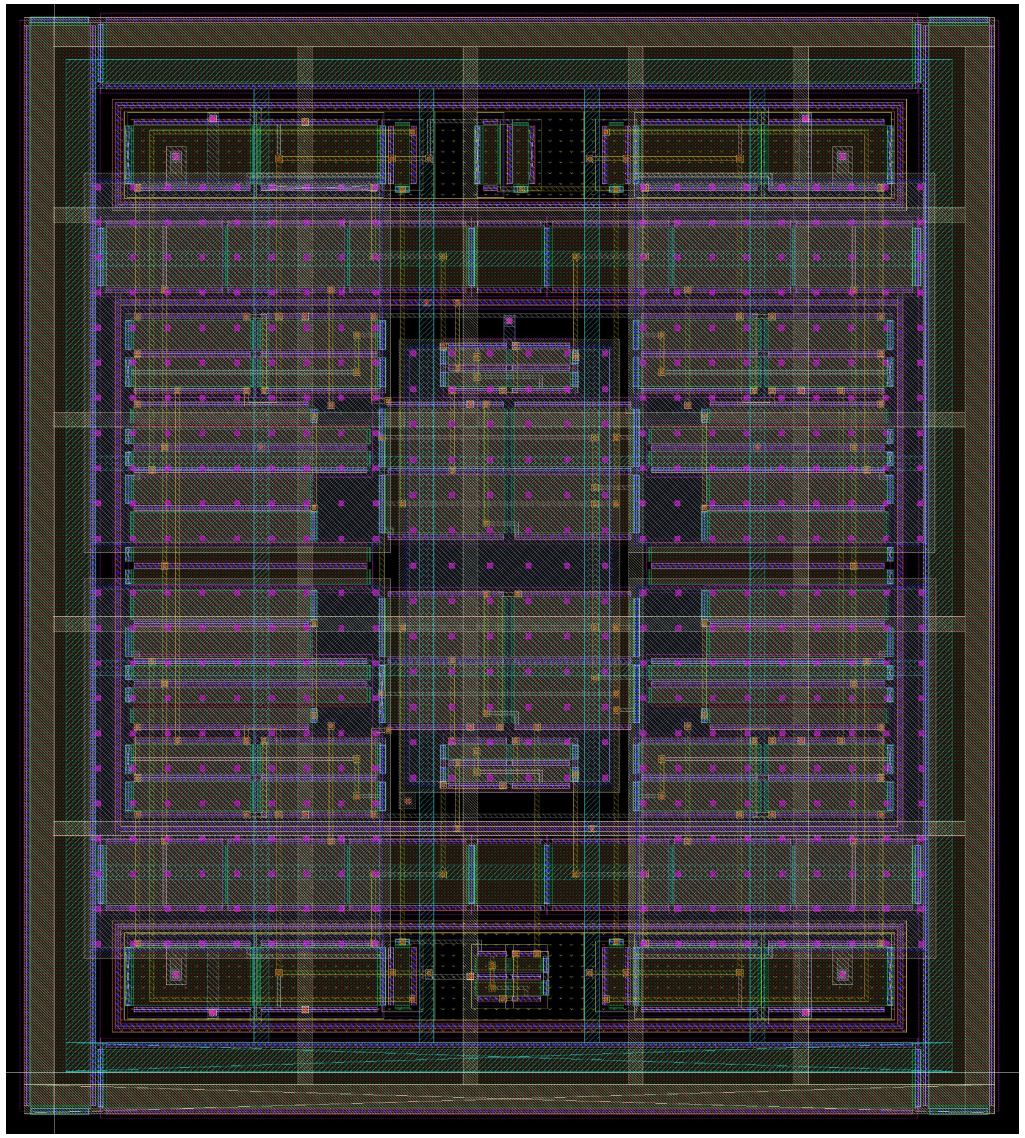
The final layout instantiates all four feedback blocks, mirrored to form a central cavity where the soma is placed.

The resulting structure features PMOS rows at the top and bottom, and a single NMOS block at the center.

Guard rings were added to prevent latch-up: two P-guard rings for the PMOS stripes and one large N-guard ring for the NMOS.

Additional dummy devices were placed around the entire circuit to ensure uniform boundary conditions in any instantiation of the block.

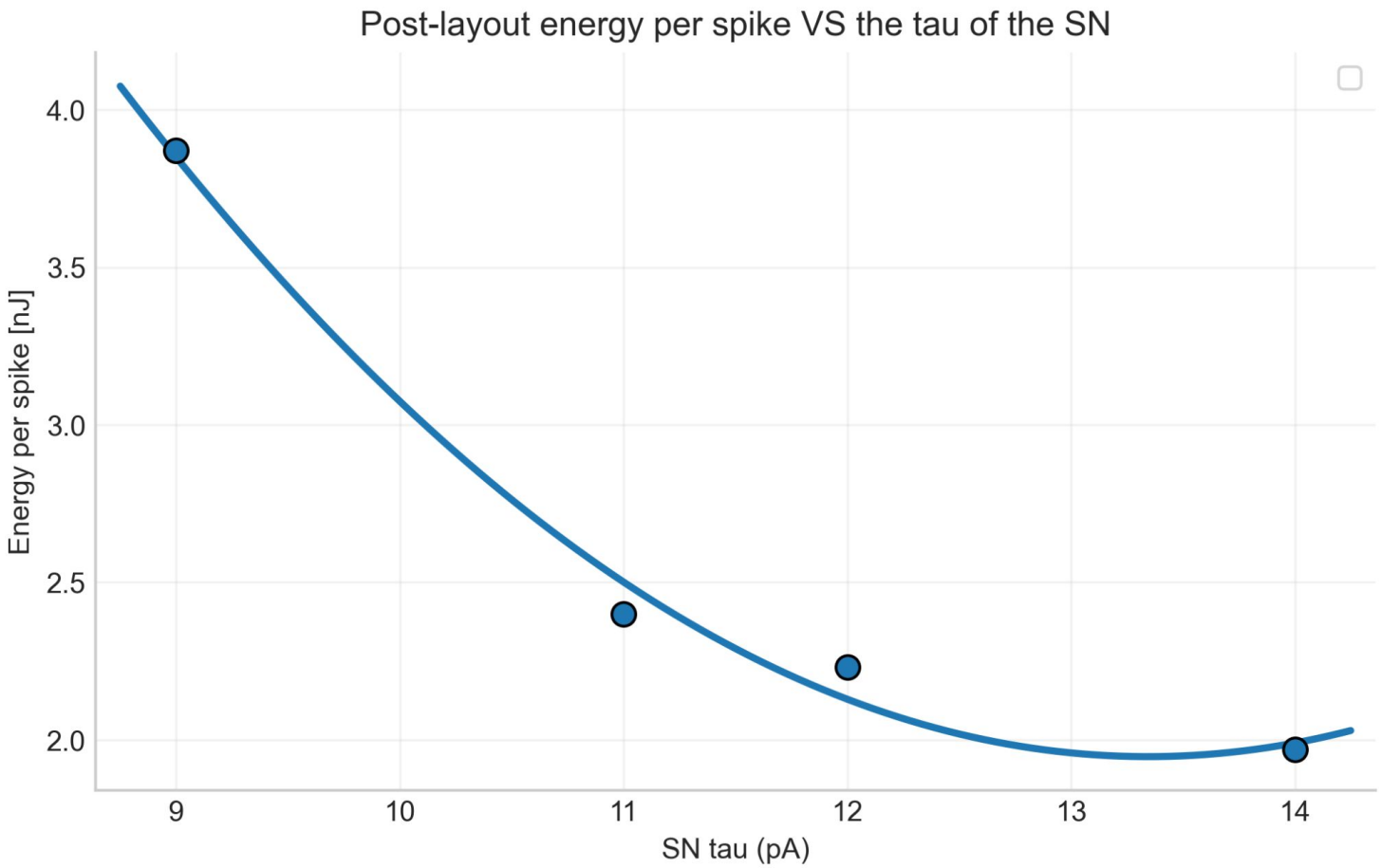
A power grid was then created to distribute power efficiently across the circuit with low impedance. The total layout area is 75.66×67.02μm2.



Layout of the final neuron

Post-layout simulations - Energy per spike

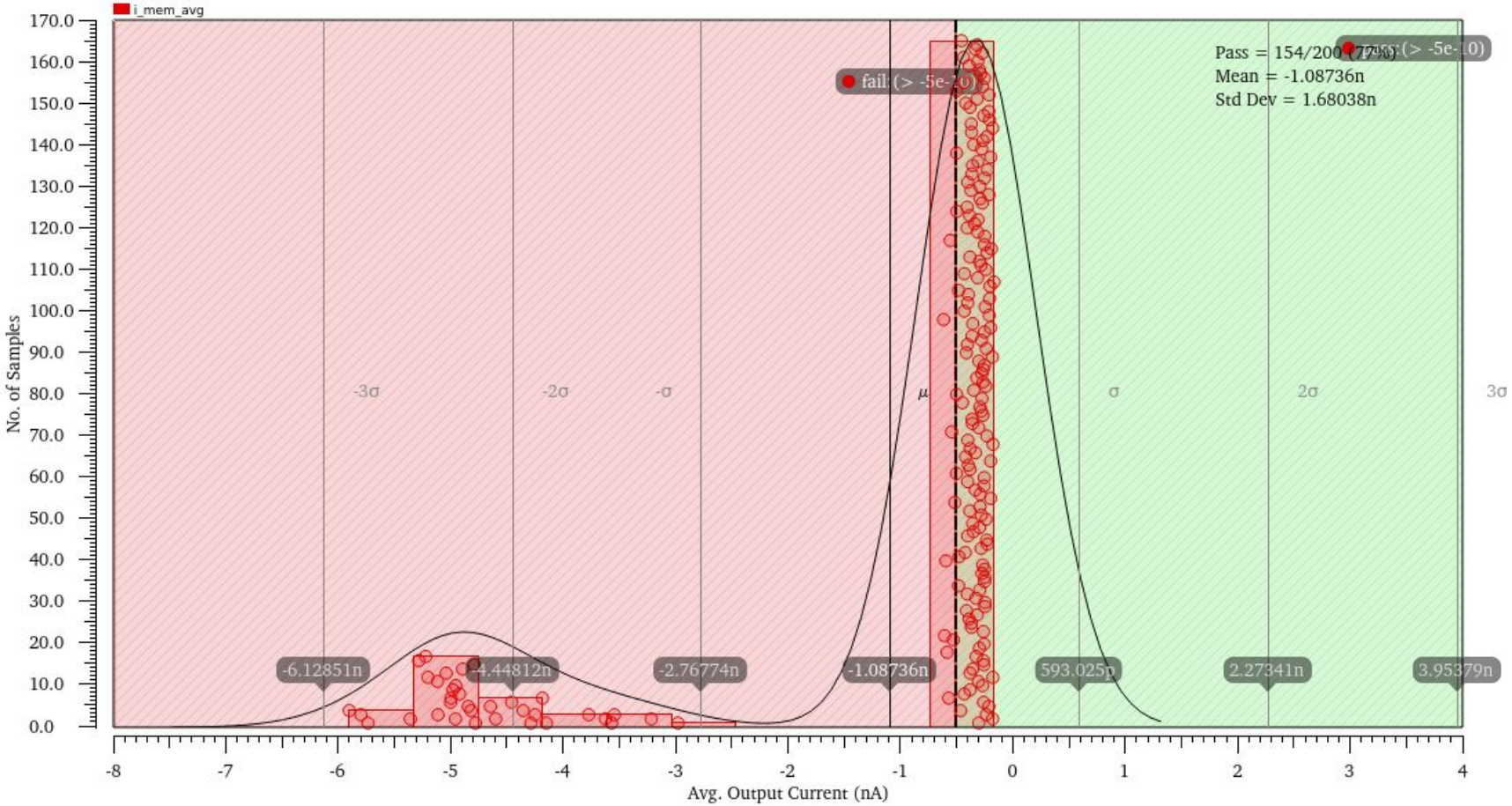
As with the pre-layout analysis, I measured the energy required to generate a spike. As expected, post-layout energy consumption was higher due to parasitic effects



Pre-layout energy per spike vs the tau of the SN feedback

Post-layout simulations - Montecarlo simulation

To evaluate post-layout robustness, I performed a new Monte Carlo simulation. This run was computationally intensive and took approximately 10 hours to complete. The final yield was 77%



Montecarlo simulation results; each dot is a trial

ETH zürich

The future

Future projects

In the future, I will complete my current projects and start new ones, specifically:

- **Croc expansion:** As part of the VLSI2 course during July, I will implement a new feature in the Croc SoC. I am currently planning to design a DFT accelerator, though I am still unsure due to tight area and time constraints. The goal is to justify the modification by demonstrating its advantages. We will handle everything from the golden model to the final backend so that the chip can actually be fabricated (up to 5 designs will be produced).
- **SuperPoint on Gap9:** As part of the System on Chip for ML and Data Analytics course, I will work on a mini-project aimed at reducing the size of a neural network called “SuperPoint,” which performs real-time feature recognition. The goal is to shrink it enough to fit inside the 1.5MB L2 cache of the Gap9 cluster.