CentraleSupélec

Georgia Tech Lorraine

BRAIN-INSPIRED COMPUTING BASED ON A FPAA PLATFORM

Spring 2019

NGUYEN Ngoc-Son-Dorian

Instructors:

LOQUET Alexandre, RONTANI Damien, PIOTR Antonik, BARRET Michel

Abstract

In this report, I present the result of my work on the implementation of a brain-inspired computing system on a FPAA platform. After introducing the concept of Reservoir Computing and the principal notions behind this type of implementation of machine learning I explain how to train these systems and present a software simulation that I developed to simulate and validate the implementation of Reservoir Computing on the DC RASP3.0A FPAA. After validating the ESN software implementation on known tasks and trying different settings to discuss the interest of the activation function and the connectivity of the reservoir, I design the electronic nodes that will be use on the FPAA.

After validating the obtained activation function on a simple task on my simulator, I try to simulate the whole board as a dynamic reservoir of NMOS nodes with low pass filters connected with capacitors. The results of the simulation cannot validate the implementation, but even if the designed reservoir is the cause of the failure, but different paths remain to implement a reservoir computing system on this board.

Table of content:

1. Context:

Nowadays, Machine learning became a powerful tool to solve complex problems and perform better than experts [1]. The algorithms can now drive a car, beat professional players at Go, Chess or even real time strategy video games like Dota 2 or Starcraft 2 (Alphastar versus the pro player Mana). But the implementation of these algorithm needs a lot of computational power hence consume a lot of energy. 20% of the world power consumption will be to aliment datacenters in 2025 [2] and these numbers should have attained 3.5% in 2019.

That's why implementing machine learning algorithm on using physical analogic calculation is interesting. The analogic calculation is instantaneous, consume a small amount of energy and can be thousands of time quicker [12-18][23]

1.1 Defining the project:

The global goal of the project is to **Implement type of a Reservoir Computing system** (a type of neural network) **on a *Field Programmable Analog Array*** (FPAA) using its analog components (MOS type transistor, operational transconductance amplifier (OTA)…)

This project was in common with 6 second year CentraleSupelec's students. More precisely, my goal was to design, simulate and test architectures with the least possible components, and validate it on a simple task.

2. The Reservoir Computing

2.1 Theory and definition[3-5]

In the field of machine learning , reservoir computing is a paradigm applied to signal that emphasized on two key point that are a randomly generated fixed dynamic reservoir, that will be in charge of the richness of the system by mapping the input to an higher dimension space using nonlinear transformations, and an output layer, that will be the only trained part of the Reservoir Computing system[2-7]. The reservoir is often a recurrent neural network (RNN)[6-8] and the output layer is commonly a linear readout of the states of the neurons.

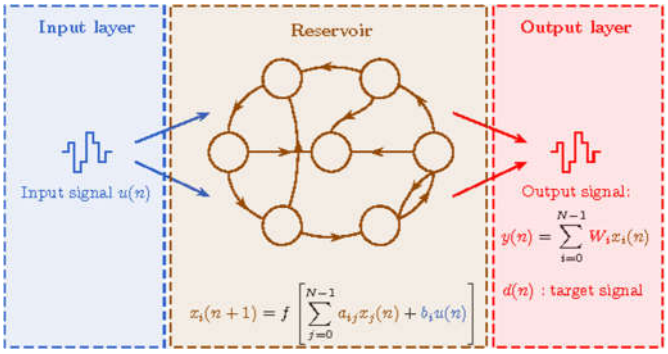


Figure 1:PIOTR Antonik, "FPGA Implementation of Reservoir Computing with Online Learning" (2015)

The idea behind the reservoir computing is to change the given problem to an easiest one by applying random non-linear transformations. A common example used to illustrate the idea is throwing a pebble in the water. If one can guess the weight of a pebble by observing without a scale, throwing the pebble in a know bucket of water and observing the wavelet can be a more precise method. For Reservoir computing, the input signal is the pebble, and the bucket is the reservoir.

The Reservoir should have few properties:

- Vanishing memory[3]: The reservoir's state return to a unique state if not driven by any input.
- Input separability[5]: The reservoir's excitation should differ between two different input

For the readout layer :

- Approximation property[5]: The ability of output layer to differentiate these responses and relation with the given output

2.2 Common models

The two main mathematical model of Reservoir are the Echo State Network, formalized by H. JAEGER in 2001 [3] and the Liquid State Machine (LSM), by W. MAAS in 2002 [5].

The LSM implementation use Spiking Neurons to build the reservoir while ESN use Formal Tanh Neuron [3]. The concept of ESN was generalized to any Formal Neuron and considering the feedback of the neuron on itself give the concept of Leaky ESN.

My implementation focusses on ESN description, that use formal non-spiking neural network.

For a non-Leaky ESN (Each node is a formal neuron) of size N, the update equation is:

$$X[n+1] = f\big(\, W.X[n] + W_i.U[n] + W_{fb}.Output[n]\big)$$

$$Output[n] = \; g\left(W_{out}.\begin{pmatrix} X[n] \\ U[n] \\ 1 \end{pmatrix}\right)$$

With:

- $X[n] = \; (x_0[n], x_1[n] \; \dots \; x_N[n])^T$ the state of the reservoir given by the state of the N neurons
- W the weight matrix of the RNN
- W_i the weight of the inputs
- W_{fb} the weight matrix of the feedback
- W_{out} the readout matrix
- B the weight of the input for the output
- f the activation function of the nodes
- g the activation function of the output

$g = id \; and \; W_{fb} = \mathbf{0}$ are commonly used in our tasks, giving the update function

- $X[n+1] = f(\, W.X[n] + W_i.U[n])$
- $Output[n] = W_{out}.\begin{pmatrix} X[n] \\ U[n] \\ 1 \end{pmatrix}$

(1) For a ESN without feedback ($W_{fb} = 0$) with f a λ-Lipchitz function, the Fading Memory propriety is guaranteed for any input by the condition $spectral\; radius\; (W) < \frac{1}{\lambda}$

a. DEMONSTRATION[3]:
For the two inputs $u_1[n]$ and $u_2[n]$ identic since the time k, the difference between the state of the reservoir is given by:

$$X_1[n] - X_2[n] = f(W.X_1[n-1] + W_i.U[n]) - f(W.X_2[n-1] + W_i.U[n])$$

f being λ-Lipschitz,

$$|X_1[n] - X_2[n]| < \lambda |W.X_1[n-1] + W_i.U[n] - \; W.X_2[n-1] - W_i.U[n]|$$

For n> t :

$$X_1[n] - X_2[n] \; < \lambda.|W.(X_1[n-1] - X_2[n-1])|$$

Let s the spectral radius of W :

$$|X_1[n] - X_2[n]| < \lambda.s.|X_1[n-1] - X_2[n-1]|$$

For $s < \frac{1}{\lambda}$,

$$|X_1[n+k] - X_2[n+k]| < (\lambda.s)^k |X_1[n-1] - X_2[n-1]|$$

$$\lim_{k \to \infty} |X_1[n+k] - X_2[n+k]| = 0$$

Hence, if the spectral radius of the adjacency matrix is smaller than $\frac{1}{\lambda}$, the initial conditions of the reservoir does not impact is state on an infinite horizon. This property guarantees the vanishing property, by considering $X_2[n] = X_0[n]$ the response from the reservoir to the null input and $U_2[n] = 0$.

(2) According to Jaeger W should be a Sparse ($Wi,j = 0$ with a probability 1-p and p small (1% for N>100). This condition should provide lowly connected oscillators in the reservoir, that will increase the richness of the non-linear transformation [3].

This second consideration was proven false [20] but the sparsity of the matrix does not decrease the performance of the reservoir and allow the computational complexity of an iteration of a step of the reservoir to remain low, the step being in O(p*N²).

2.2 Training the reservoir

2.2.1 Considering the overfitting

For the reservoir computing method, only the output layer is trained.

We use supervised learning to train it (The target of the reservoir for a given input is specified). To prevent overfitting, the system learning the training set by heart without being able to extrapolate, it is important to train the system on much more data than the number of parameters to train. Here, the readout matrix uses one weight per node, one for the biases and one for the input, so I need to train the reservoir on much more samples than the number of nodes (10 times the number of node seems to be a good starting point) [7]

2.2.2 For an ESN without feed-back and readout function:

To train these reservoirs for a given output $Y = (y[0], y[1] \dots y[k])$, you need to collect the states over time for a given inputs.

I add $U[k]$ and 1 as a state for the training.

Let's call $M = \begin{pmatrix} x_1[0] & \cdots & x_N[0] & U[0] & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_1[k] & \cdots & x_N[k] & U[k] & 1 \end{pmatrix}^T$ the collection of the state of the reservoir over time.

2.2.3 The optimization problem

Least square regression

- Let's tried to find the matrix W_{out} that minimized $\boxed{|| (W_{out}.M) - g^{-1}(Y) ||_{L_2}}$, the quadratic error to the target. We find $\boxed{W_{out} = g^{-1}(Y).M^{+} = g^{-1}(Y).(M^{T}M)^{-1}M^{T}}$, **if and only if M is full column rank.** The different features of the reservoir need to be non-colinear.
- We can add a low amplitude random noise to the state, guaranteeing that the column will not be colinear. (collinear with a probability of 0)

When the columns of M are nearly colinear, $|| W'_{out} ||_{L_2}$ be large, giving an high standard deviation to the prediction. To prevent that we can use the ridge regression:

Ridge Regression [21]

- Let's tried to find the matrix W_{out_λ} that minimize, for a given $\lambda > 0$, $\boxed{|| \left(M.W_{out_\lambda}\right) - g^{-1}(D) ||_{L_2} + \lambda \times || W_{out_\lambda} ||_{L_2}}$, the quadratic error to the **target plus a penalty on the L_2-Norm of the readout matrix**.
- We find $\boxed{W_{out_\lambda} = g^{-1}(Y).(M^{T}M + \lambda.I_d)^{-1}M^{T}}$ **for any matrix M even non-fully column ranked.** The limit case for $\lambda = 0$ correspond to the least square regression.

The metric used for comparing the performances is the Normalized Root Mean Square Error[7]

$$NRMSE = \sqrt{\frac{<|| y[k]-d[k] ||_{L_2}>}{{\sigma_d}^2}} = \sqrt{NMSE}$$

2.2.4 For an ESN with feed-back[7]:

For an ESN with feedback, the state of the ESN will depend on the values of W_{out}, because the evolution function $X[n+1] = f\left(W.X[n] + W_i.U[n] + W_{fb}.Output[n]\right)$ depend on the output of the reservoir.

Hence, I used the forced teaching method:

During the training, the state of the reservoir is not compute using the readout function, but using the teacher (forced teaching):

$$X_{training}[n+1] = f\left(W.X_{training}[n] + W_i.U[n] + W_{fb}.Y[n]\right)$$

This method consists of driving the reservoir with the teacher in place of the readout during the training phase. This method makes the hypothesis that:

- The matrix W_o that produce a reservoir with feedback that follow the dynamic of the teacher for these particular $W, W_{feedback}$ exist (Input separability and approximation property)
- This dynamic is stable, the reservoir will converge toward this dynamic from any random state.

I call $M = \begin{pmatrix} x_{training_1}[0] & \cdots & x_{training_N}[0] & U[0] & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_{training_1}[k] & \cdots & x_{training_n}[k] & U[k] & 1 \end{pmatrix}^T$ the collection of the state of the reservoir over time. To compute W_o we use the same methods as the previous case.

3. Software Implementation

3.1 The implementation

Link to the GitLab:

https://gitlab.centralesupelec.fr/nguyen_ngoc/reservoir-computing-simulator

Before trying to implement a reservoir computing on the Field Programmable Analog Array (FPAA), I need to understand how Reservoir Computing work and which association of component of the FPAA I can be used as Nodes for our reservoir. That's why I developed a reservoir computing simulator, to test the ideas of nodes in a simulation, before trying to implement them on chip.

The key features of this simulation is to choose freely the activation function of the neurons, the connectivity of the adjacency matrix and its spectral radius.

I made this software implementation with the Anaconda 3.5 environment – using Python 3.6, NumPy and matplotlib.

The class *Reservoir* can create any reservoir with $g = identity$ and U a scalar input and scalar readout.

The reservoir is trained using the *Ridge Regression*, a white noise of adjustable amplitude is added in the preactivation of each node, to prevent colinear columns, the *Least Square Estimator* can be obtained by fixing $\lambda = 0$ during the training. The forced teaching for the Feedback reservoir can be enable.

To validate this implementation and different hypothesis, I tried to recreate the literature's results on 3 common tasks:

- With a random noise as an input, generate NARMA10
- With the input $u[k] = \sin(\frac{k}{4})$ generate the output $y[k] = \sin^7(\frac{k}{4})$
- With no Input but with feedback, generate the output $y[k] = \sin^7(\frac{k}{4})$4.2 Experiments

3.2 The tests

3.2.1 NARMA10[6]

NARMA10 is a time-series using a random noise as an input. The state of the teacher d at the time k with the random input $U[k]$ is given by:

$$\mathrm{y[k]} = tanh(0.3.y[k-1] + 0.05.y[k-1].[\sum_{i=1}^{9} y[k-1-i]] + 1.5.U[k-1].U[k-10] + 0.1)$$

This task require memory over the 10 previous states (because of the terms $y[k-1].[\sum_{i=1}^{9} y[k-1-i]]$ and over the 10 previous inputs because of $.U[k-1].U[k-10]$.

Here, $U[k]$ follow an uniform distribution over $[0; 0.5]$

I used the same implementation as H. JAEGER [6] to solve this problem, hoping to validate the

Parameter	Description	Value
N	***The number of nodes***	100
P	*The connectivity of the connection matrix*	0.05
Sp	*The spectral radius of the connection matrix*	0.8
V	*The amplitude of the noise in the preactivation*	0.0001
$\boldsymbol{f}$	*The activation function*	$tanh$
Input Scaling	*The amplitude of the input matrix*	0.1
$\boldsymbol{g}$	*The readout function*	$identity$

simulation by finding the same results.

I used a Reservoir without feedback, using the parameters:

The reservoir is trained on 1000 points, after discarding the 400 first outputs to remove the influence of the random starting state.

The reservoir is then reset in the 0 state, feed for 402 points, then the NRMSE is computed on the 2 000 following samples. I tested 50 reservoirs, with each reservoir trained 10 times.

H. JAEGER publish the score of 0.179 = $\sqrt{0.032}$ [7] on this same task and a predictor with a NRMSE under 0.2 can be considered good.

Therefore, we can consider this software implementation of ESN valid

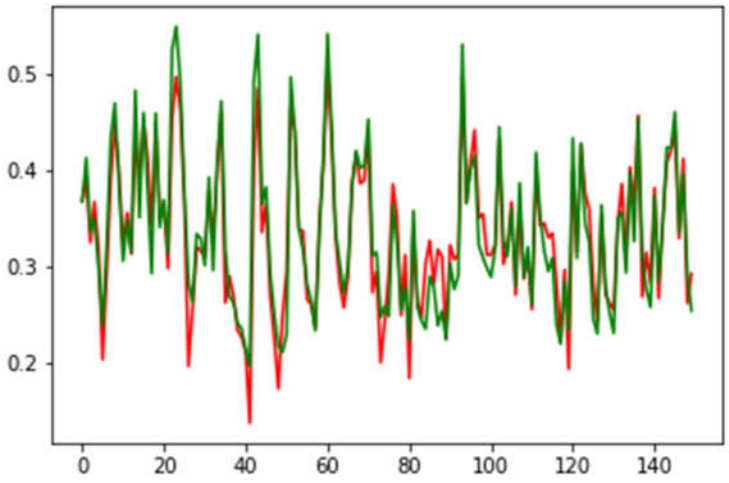


Figure 2: Narma10 and the output of the reservoir

NRMSE	Training	Testing
Mean	0.128	0.131
Max	0.183	0.186
Min	0.112	0.114
Standard deviation	0.0110	0.00923

3.2.2 Obtaining a Sinus to the power 7[6]:

For this experiment, I wanted to highlight two points:

- The (1) condition work for any Lipschitz function, not only for the $tanh$ function.
  - The activation function is here $x \to sign(x).\ln(1+|x|)$
- The (2) condition is not necessary for a practical reservoir.
  - The experiment is done with a fully randomly connected reservoir and with a sparsely connected reservoir.

For each of these experiments, $u[k] = \sin(\frac{k}{4})$ and $d[k] = \sin^7(\frac{k}{4})$

Number of nodes	**20**		**40**	
Connectivity	0.1	1	0.1	1
Spectral radius	0.8			
Noise's amplitude	0.001			
NRMSE on the training	0.096	0.055	0.034	0.034
NRMSE on the test	0.60	0.59	0.60	0.60

$p = 0.1$

The training occurred on 100 points, the testing on 200.

First, the "damping" show that the reservoir retains its echo/fading Memory property. From a random state, each node return to the 0 state if no input is given. The activation function $x \to sign(x).\ln(1+|x|)$ give to this reservoir the echo/fading memory property even if this function is not bounded. This seems to validate the idea that the function just needs to be a Lipschitz function to give the vanishing memory property.

The connectivity of the reservoir is low, in average there are 2 connection per nodes. This particularity reduces the time complexity of computing the actualization of the state in $O(2 * 20)$ in place of $O(20^2)$ for an optimized matrix multiplication. All the nodes are connected to another node. The nodes without connection from others node of the reservoir are connected to the input and their output is driven toward other nodes. The nodes which do not feed other nodes of the reservoir are feeding the readout matrix.

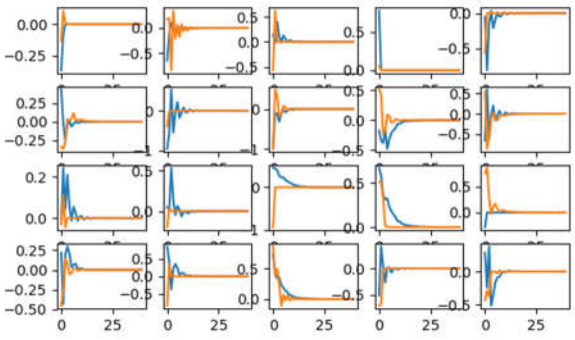


Figure 3: Two damping operation: the 20 nodes should return to the zero state.

The NRMSE calculation occurred after resetting the reservoir in a random state, driving it with the input for 100 steps, then compare the results to the teacher on 200 steps

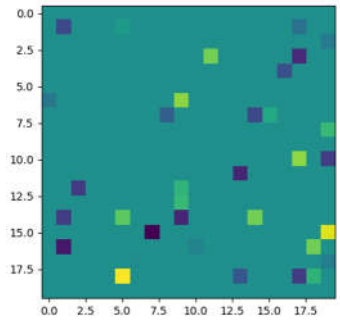


Figure 4: Connectivity of the reservoir.

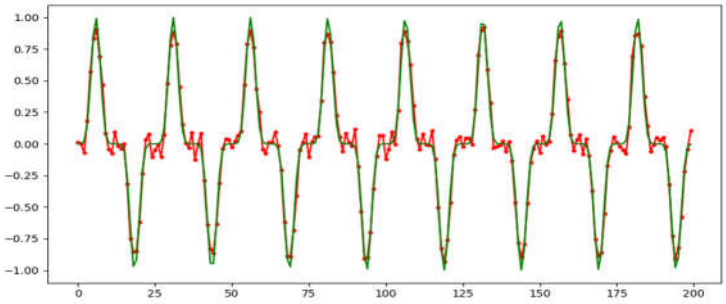


Figure 5: Output of the reservoir against the teacher on 200 points

$p = 1$

The training also occurred on 100 points, the testing on 200. The 'damping' takes more time than the reservoir of lower connectivity, but it occurred. I came to the same conclusion: The activation function $x \to sign(x).\ln(1 + |x|)$ give to this reservoir the echo/fading memory property even if it is not a bounded function.

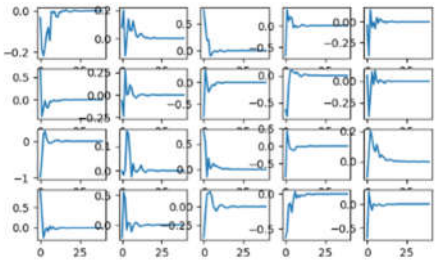


Figure 6: Damping operation: the 20 nodes should return to the zero state after some time.

The connectivity of the reservoir is 1, each node is randomly connected to each node. This does not impact the NRMSE on the testing set, but greatly decreased the NRMSE on the fitting error (-40%). I can fear that these reservoirs are more prone to overfitting than the sparse reservoir. The time complexity of computing the actualization $O(20^2)$, for an optimized matrix multiplication.

Again, the NRMSE calculation occurred after resetting the reservoir in a random state, driving it with the input for 100 steps, then compare the results to the teacher on 200 steps

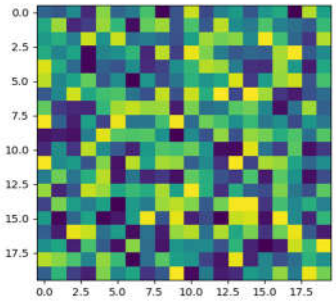


Figure 8:Connectivity of the reservoir.

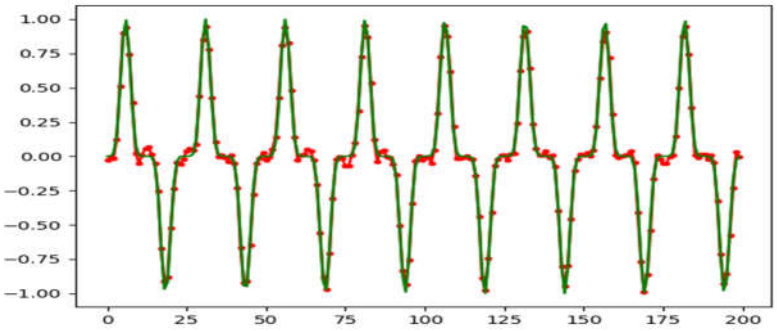


Figure 7:Output of the reservoir against the teacher on 200 points

The results are better for a fully connected reservoir, but this advantage disappear for a bigger number of nodes. Choosing a sparse reservoir does not r educe the performance on this task if we keep the number of connections around 5 per nodes.

3.2.3 Generating a sinus[7]

The goal of the experiment is to train a Reservoir with feedback. The feedback is needed to be able to obtain a highly dynamical reservoir with no input. W. JAEGER claim that there is no simple verification over W, W_{input}, W_{output} and $W_{feedback}$ that imply the Fading Memory (And in our particular case, I do not want this property, because I did to generate a signal from no input $W_{input} = 0$). I will set the spectral radius of W at 0.8, to guarantee the echo/fading memory for the reservoir without feedback, then I will train the reservoir using the forced teaching method.

The teacher is here $d[k] = \sin(\frac{k}{4})$

Number of nodes	**20**
Connectivity	0.1
Spectral radius	0.8
Noise's amplitude	0.001
NRMSE on the training	0.003
NRMSE on the test	0.066 /!\
Activation function	$f: x \to thanh(x)$

/!\After shifting the sinus, the absence of input mean that the initial phase of the output of the reservoir will be random.

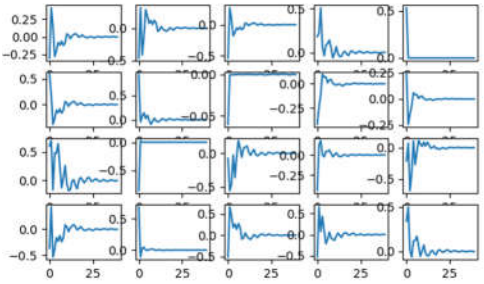


Figure 9:Damping 20 neurons

The damping operation went well, the form of the damping is the same than the damping on the previous experiment. For no input, from a random state, a 20 nodes reservoir with the $tanh$ activation function will tend toward the zero state.

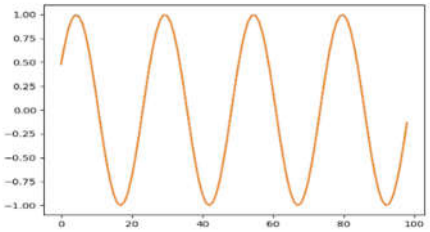


Figure 10: Fitting using the forced teaching method

The fitting using forced teaching on 100 points. The NRMSE on the fitting is 0.003, giving a nearly perfect sinus. We cannot clearly distinct the prediction from the teacher on this graph.

Again, the NRMSE calculation occurred after resetting the reservoir in a random state, driving it with the input for 100 steps, then compare the results to the teacher on 200 steps.

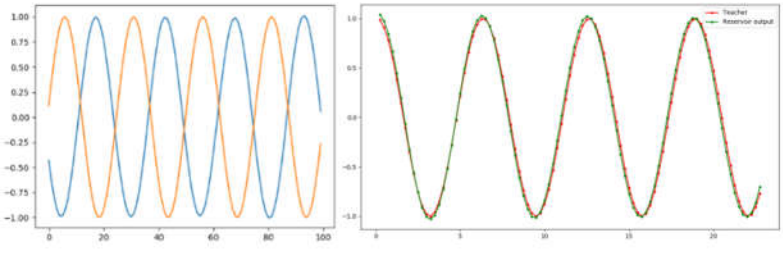


Figure 11: Output of the reservoir compared to the teacher, before/after shifting

But because the reservoir as no input, the sinus is shifted. The NRMSE make no sense as a metric in this situation, maybe the NRMSE on the Fourier transform would give better results. After reshifiting the output signal by synchronizing the first maximum of the signal, the NRMSE was 0.0662, which can be considered good.

I plotted the prediction and the teacher in a XY mode. If the generated sinus has the same frequency as the teacher, we should see a perfect ellipse. In our case, we can see that the ellipse radius

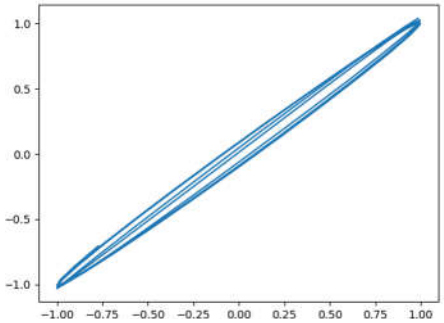


Figure 12: The output of the reservoir and the teacher in XY mode. The darker a point is, the younger it is

lightly changes during the time. Our prediction has the right frequency, but this amplitude varies over time.

4. Field Programmable Analog Array

4.1 Presentation

Using a FPAA to implement an analogic reservoir computing is not a new idea [10-17] but the most common way to implement them is to recreate spiking neural network on the chip, like the Hodgkin-Huxley node. Our main idea is to use the non-linearity of the less component as possible (directly a N-MOS, OTAs …).

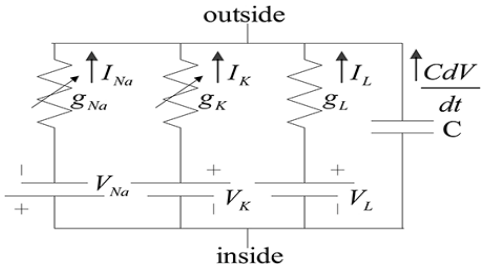


Figure 13: Hodgkin-Huxley node [12]

The core idea here is to implement an ESN by using electronic nodes. Because the ESN model use discrete time steps, the system should use a delay embedded in each node, in the form of a buffer or a low-pass filter. The main challenges are the design of the activation function, the realization of the synapses used for the connection and the implementation of the delay.

4.2 The possibilities of the board

On our board, the IC RASP-3.0A[24], the FPAA allow the use of N-MOSfet transistor, P-MOSfet transistors, Operational Transconductance Amplifiers (OTA) and floating gate MOS transistors. Theses component are highly non-linear and can be used in a lot of different ways:

- The transistors link a 3 voltage with a current. Using the EKV[37] model, for a given MOS ($\sigma, \kappa, I_{th}, U_t$ and V_t are fixed) we have:

$$Ids = f(V_g, V_d, V_s) = I_{th} * \left[ln\left(1 + e^{\frac{\kappa(V_g - Vt) - V_d + \sigma V_s}{U_t}}\right)^2 - ln\left(1 + e^{\frac{\kappa(V_g - Vt) - V_s + \sigma V_d}{U_t}}\right)^2 \right]$$

- the OTA are equivalent to OPA with an output in current. The input is the voltage difference on the differential pair, the output is a tanh of this voltage. This component is base on at least 9 MOSFET.
- The Floating Gate MOSFET[16,18] allow to set an offset on the gate of the MOSFET V_g by adding electrical charges on the gate, retained by a capacitor.

The goal is to design an activation Lipschitz activation function, that allow the creation of a reservoir with *Vanishing Memory* and *Input Separability* for which the Identity readout function present the *Approximation Property*.

We need to keep in mind that the IC RASP-3.0A embedded only capacitors as dipoles, a not any resistor or inductance.

4.2.1 Which physical value use

We could use for our reservoir both voltages and currents.

- The use of current allows
  - to use weighted current mirror to realize the synapses, with the only condition being positives weights [30].
  - Vector Matrix Multiplier in current is already implemented in the toolbox of the board [29,30].
  - the use of the Kirchhoff laws to create the sum at the preactivation
  - If an approximately linear way to convert tension in current is found, the OTA can directly be used as a tanh activation function, proven to show all the wanted properties.
  - Current to Current multiplication is possible by implementing a Gilbert Cell [34]
- The use of voltage allows:
  - To distribute the output of a node without relying on the use of a mirror
  - To use capacitors as synapses [26-27]
  - Voltage to Voltage multiplication is possible on the board

If the use of current seems easier, the main drawback of this method is the need of one FGMOSFET per synapse and 9 MOSFET per neuron. One of the constraints of the project being using the least MOS possible, we will try to implement a reservoir in voltage.

4.2.2 Which component use

The key point for this reservoir is to obtain a λ-Lipschitz funcion with a non-linearity. Our main components, the OTA and the NMOS, transform a voltage in current. We will need at least one more component to transform this voltage in current. A n-MOS in linear region seems to be a good way to replace a resistor and convert the current in voltage. To reduce the number of N-MOS needed for the reservoir, I decide to use principally MOSFET.

The Jaeger implementation of ESN emphasized on the use of Thanh function. Sigmoid function is also used for Neural Network implementation so obtaining exponential function is interesting in our case. The MOS transistor operating in Sub-threshold [35] region ($V_g > V_t$) follow the relation $Id = I_o e^{\frac{V_0 - \kappa.V_g}{U_t}}$ for $V_0 - \kappa.V_g << U_t$ and we say that the NMOS operate in Weak inversion. This current is always present but small and considered as losses in the other regions. Using the NMOS in weak inversion allow to use the property of the exponential

like in the VMM implementations [29-30] and to be efficient in energy (you only use the losses of the NMOS) [23].

5. Designing the node

5.1 Inspiration

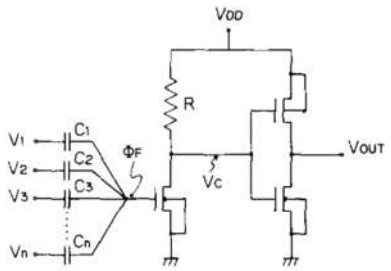
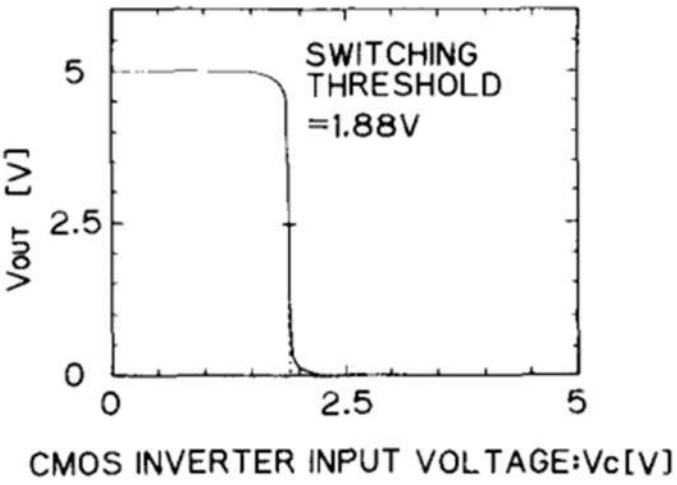
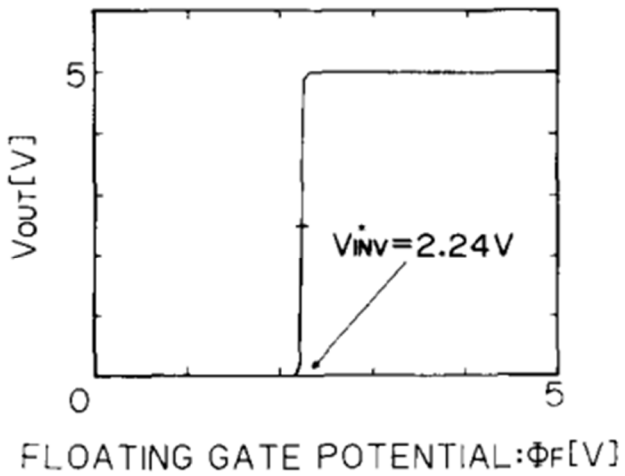


Fig. 9. An n-channel neuron circuit composed of discrete compc which has been utilized in the experiment. ϕ_F is the floating-gate pot and V_C the input to the CMOS inverter.



(a)



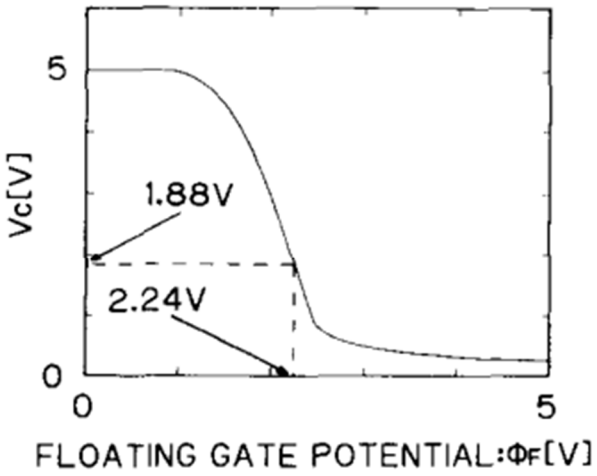


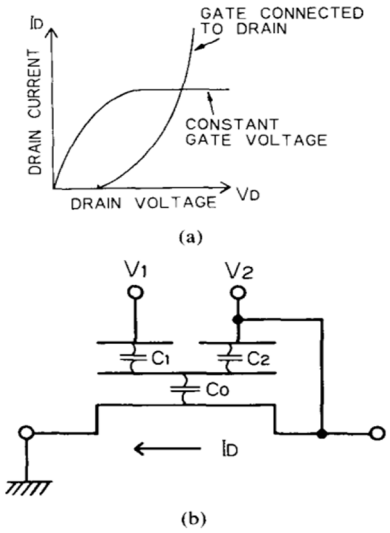
Figure 14 : Voltage of the functional MOS transistor featuring Gate level weitghed sum and threshold operation

In the paper « *A Functional MOS Transistor Featuring Gate-Level Weighted Sum and Threshold Operations »,* by Tadashi Shibata, Member and Tadahiro Ohmi (1992) [27], this model of gate is presented to produce a threshold function. The tension in this system are given by:

This implementation uses different interesting points:

- The sum of the voltage is obtained directly on the gate of the FGNMOSFET
- The V_c voltage, in input of the CMOS inverter, is non-linear and seems to be a Lipschitz function.
- The author being in a similar case than us, they used a FGNMOS to replace the resistor shown on the figure

The resistor implemented using a FGNMOS use a linear combinaison of a constant gate voltage and a feedback of the drain to the gate. By correctly choosing the ratio of the capacitor used to create the linear combinaison, we can nullify the non linear part of the relation between I_d and V_d



$$I_d = \mu_n C_{ox} \frac{W}{L} [(w_1 V_1 - V_{th}^*) V_d + \left(w_2 - \frac{1}{2}\right) V_d^2]$$

With $C_{tot} = C_0 + C_1 + C_2$,$w1 = \frac{C_1}{C_{tot}}$ and $w_2 = \frac{C_2}{Ctot}$.

By choosing $C_2 = \frac{C_{tot}}{2}$ we have $W_2 = \frac{1}{2}$ and

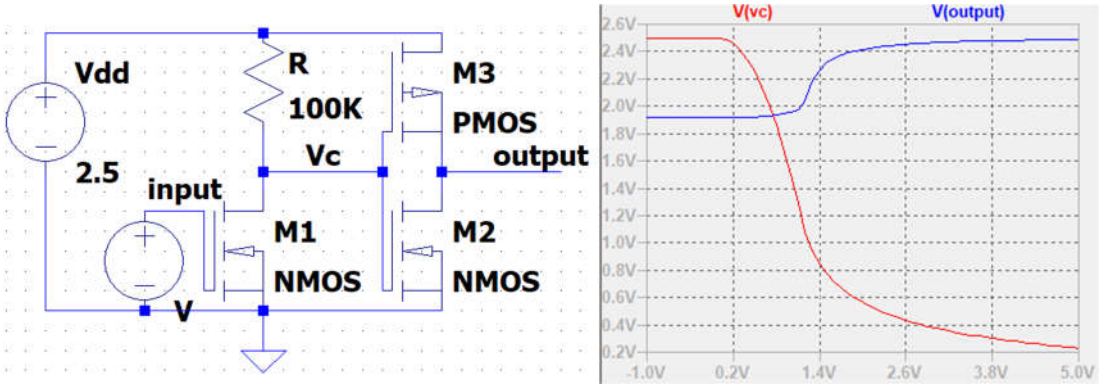
$I_d = \mu_n C_{ox} \frac{W}{L} (w_1 V_1 - V_{th}^*) V_d$, in the linear region.

This current can only be positive, but we obtain with this implementation a resistor with $R = \mu_n C_{ox} \frac{W}{L} (w_1 V_1 - V_{th}^*)$

Using the implementation of the gate with a CMOS inverter with a smaller slope should give us a usable activation function. Even with a linear CMOS inverter, the non-linearity of $V_c = f(\phi_f)$ should be enough.

5.2 Implementing this node with a resistor

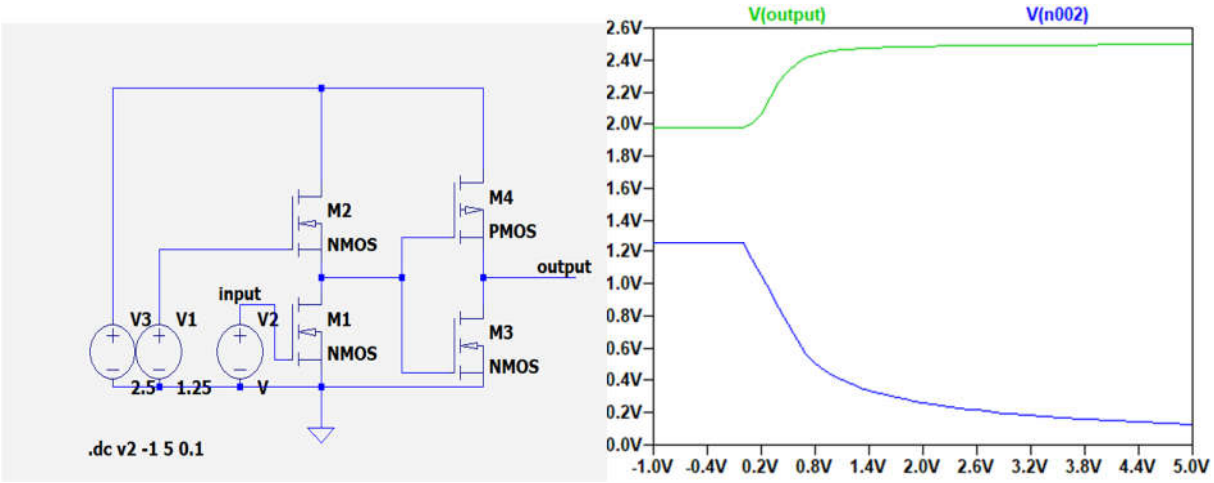
I simulate this gate on LT-Spice, using the limitations of the board.



Because the board's alimentation is not symmetrical, the CMOS can not be use as an inverter. The output remains between 1.9 V and 2.5V. The slope of the activation function can be change by modifying the value of R (V_1 for the FGMOSFET implementation of the resistor).

5.3 Implementing this node using a MOS in linear region as a resistor

Using a simple MOS in its linear region return similar results:



This implementation could be used if we were able to

- Inverse the voltage while using V_c
- Translate an amplify the output to obtain a function like a sigmoid

Unfortunately, using negative current is prohibited on the board, so the Voltage/Voltage multiplier could no inverse V_c and offsetting the tension use at least an OTA, which is in opposition to our goal that is to use the minimal amount of MOS.

Therefore, this implementation cannot be used for our reservoir.

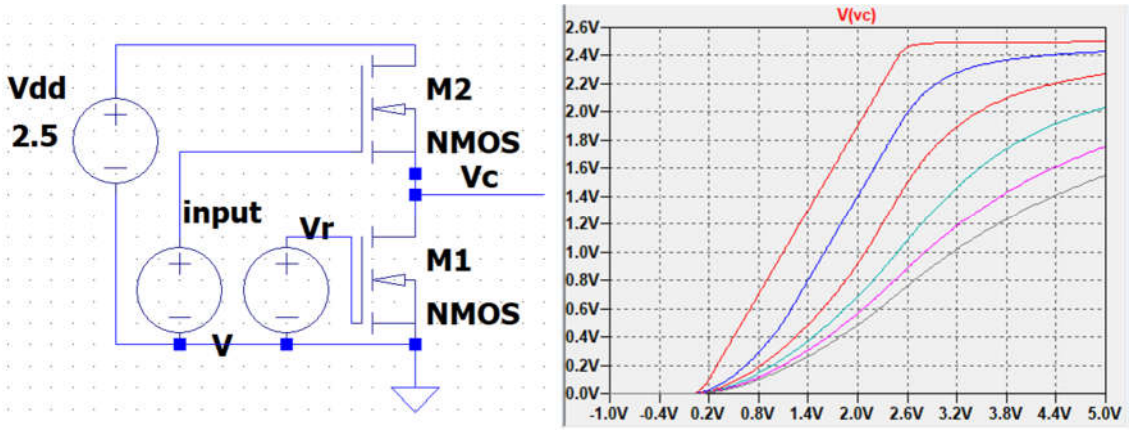
5.4 Improving the design

On the previous design, we have:

$$V_{dd} = V_{ds}^{nmos2} + V_{ds}^{nmos1} = V_{ds}^{nmos2} + V_c$$

$$\Rightarrow \quad Vc = V_{ds}^{nmos2} - V_{ds}^{nmos2}$$

Inverting the two MOSFET in our design should be enough to change the offset and the sign of the slope of the transfer function. I implemented this new design on LT-spice and obtained for $V_r \in [0.1V, 0.6V, 1.1V, 1.6V, 2.1V, 2.6V]$



The slope decreases when V_r increases, and for $V_r \leq 0.1 \approx V_{th}$ the activation function is similar to a Relu. For $V_r \geq 0.6$ the function starts to look like a sigmoid.

This node can be a good activation function to build our reservoir.

6 Testing the design

6.1 Computing the activation function

This activation function use the N-MOS in every region and we want the most accurate activation possible to validate this node. We will use the EKV model to compute the activation function.

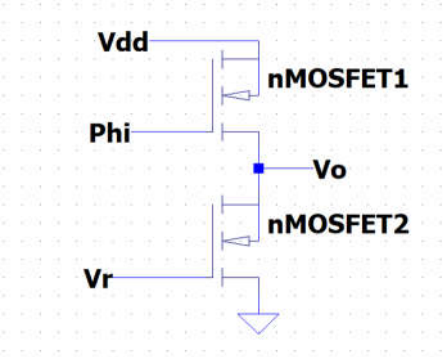
The non-linearity of the node is given by the response of the 2 n-MOSFET in series. The current in a MOS is given by the EKV as:

$$Ids = I_{th} * \left[ln\left(1 + e^{\frac{\kappa(V_g - Vt) - V_d + \sigma V_s}{U_t}}\right)^2 - ln\left(1 + e^{\frac{\kappa(V_g - Vt) - V_s + \sigma V_d}{U_t}}\right)^2 \right] [37]$$

For the n-MOSFET on the RASP3.0 FPAA, the parameters are given as follow [36]:

PARAMETER	VALUE
$\boldsymbol{\kappa}$	0.7
$\boldsymbol{\sigma}$	0.05
$\boldsymbol{I_{TH}}$	$5 * 10^{-8} A$
$\boldsymbol{V_t}$	$0.5V$
$\boldsymbol{U_t}$	$0.0026V$

For our node:



In our system, we have the same current Ids in both n-MOSFET.

For $V_{dd} = 2.5V$ we have the equation:

$$\begin{cases} Ids = 5 * 10^{-8} * \left[ln\left(1 + e^{\frac{0.7*(\phi - 0.5) - V_o + 0.05*2.5}{0.0026}}\right)^2 - ln\left(1 + e^{\frac{0.7*(\phi - 0.5) - 2.5 + 0.05*V_o}{0.0026}}\right)^2 \right] \\ Ids = 5 * 10^{-8} * \left[ln\left(1 + e^{\frac{0.7*(V_r - 0.5) + 0.05*V_o}{0.0026}}\right)^2 - ln\left(1 + e^{\frac{0.7*(V_r - 0.5) - V_o}{0.0026}}\right)^2 \right] \end{cases}$$

➔ $ln\left(1 + e^{\frac{0.7*(\phi - 0.5) - V_o + 0.05*2.5}{0.0026}}\right)^2 - ln\left(1 + e^{\frac{0.7*(\phi - 0.5) - 2.5 + 0.05*V_o}{0.0026}}\right)^2 - ln\left(1 + e^{\frac{0.7*(V_r - 0.5) + 0.05*V_o}{0.0026}}\right)^2 + ln\left(1 + e^{\frac{0.7*(V_r - 0.5) - V_o}{0.0026}}\right)^2 = 0$

I could not solve this equation, so I used a numerical method to compute the transfer function of the node for 100 value of V_r between -1V and 10V, and 25.000 value of ϕ between -5V and 20V.

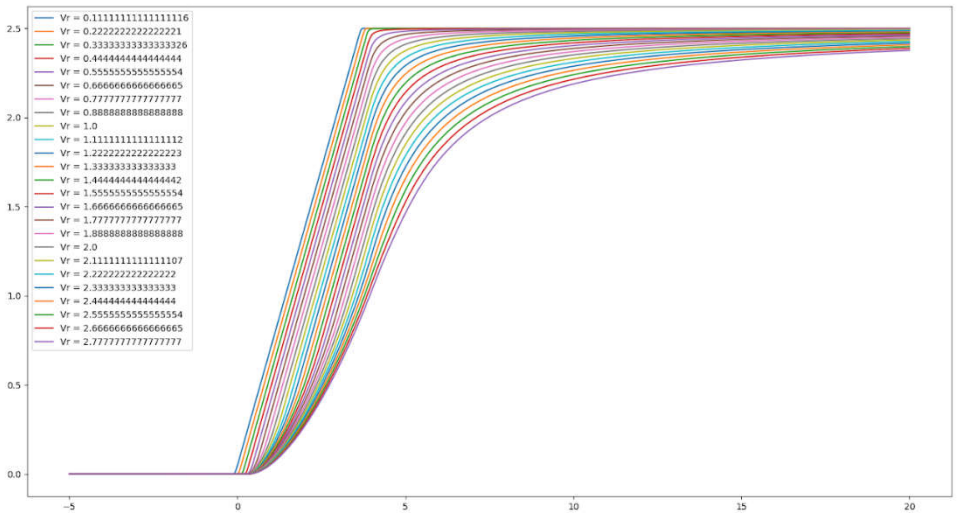
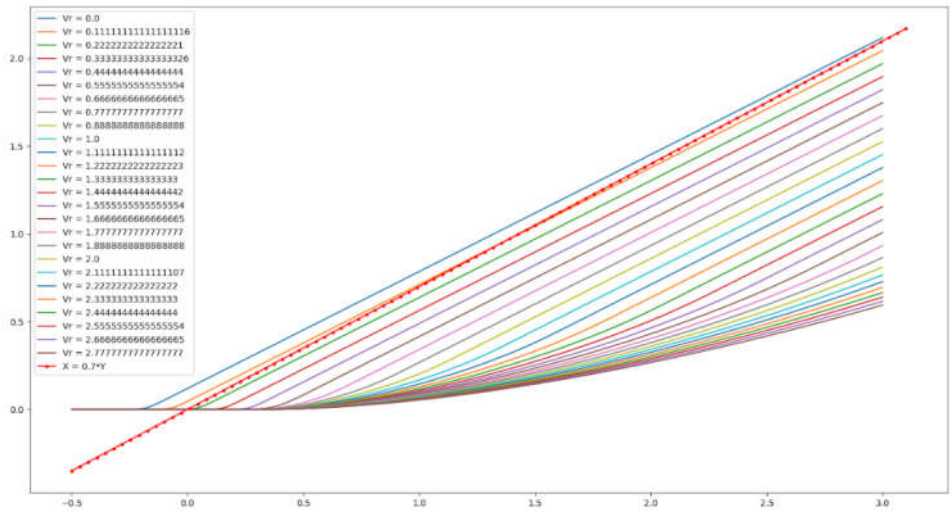


Figure 15: Activation function between -5V and 20V , for Vr between 0V and 2.8V

This function being a 0.7-Lipzicht function, the input will be between $0.2V$ and $2.5V$. The direct numerical method being heavy in computation time, we will use an approximation of this function in our simulation (Linear by interval)



6.2 Validating the activation function

This new activation function in the ESN simulator presented in the part 4, on the Santafe time series one step prediction problem. This time series was generated from measurement on a chaotic laser. This problem is considered as simple and its sole purpose is to validate the fact that this activation function can be used in an ESN. We will compare the results between

Number of nodes	**450**
Connectivity	0.1
Spectral radius	0.8
Noise's amplitude	0.0001
Activation function	$[f: x \rightarrow thanh(x)]$, $[f: x \rightarrow NodeActivation(x, V_r = 0.3)]$

A grid search on the number of nodes, the number of points on the training and the value of V_r show us that the best number of points on for the training to minimize the NRMSE on the testing is around 4000 points with $Vr = 0.3V$ and 450 nodes. The NRMSE oscillates between -48db and -63.4db during the grid search.

The training occurred on 4 000 points for both ESN and the 900 first points was discarded after resetting the ESN to forget the influence of the initial state. The results where:

Activation function	**Thanh**		**NodeActivation($V_r = 0.3$)**	
NRMSE	0.042	-63.4db	0.0095	-93.1db

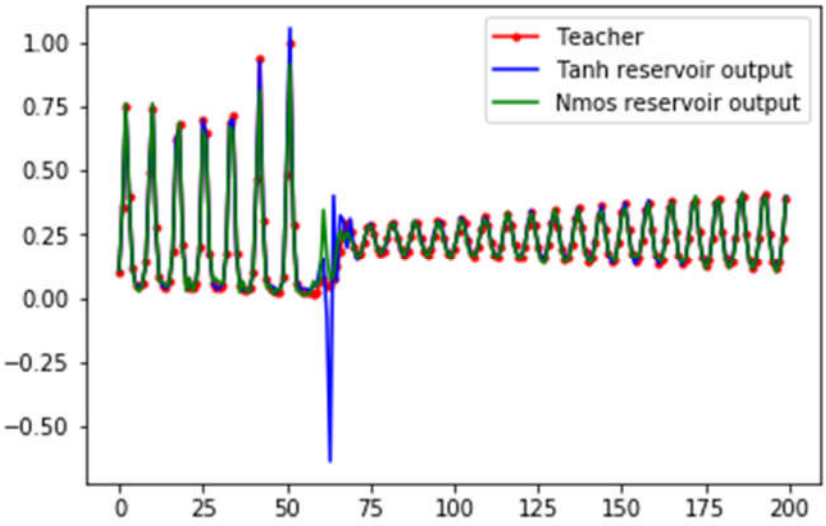


Figure 16: The last 200 points of the prediction for the Santafe time series, for both ESN

The results of our activation function where always behind the results of the tanh activation function, but -63.4 db is good enough to validate this node

7. Testing the dynamic implementation of the reservoir

7.1 Test on the software simulation

The implementation of this reservoir on the board will have several limitations:

- Because the synapses between the nodes are obtained using capacitors, the continuous component of the signal is not transmitted. Because the transistor will be FGMOSFET, it is possible to set a positive bias on the gate.
- The adjacency matrix is obtained on each row by setting $W[i,j] = \frac{C_{i,j}}{Ctot,j}$. In our case we will have only positive weights under 1, because C_0 will be small enough to be neglected, $\sum_{i=0} W[i,j] = 1$ making the adjacency a stochastic matrix, guarantying that the spectral radius will be one.
- The reservoir on the board will be a dynamic system using low-pass filter and not a discrete time implementation, so the state of the system is the state of the filters and their derivatives.
- The input is constant by interval, using a 0-order blocker

I modified the simulator to solve this problem. Each step of the reservoir corresponds to hundreds of time-steps using the 4 order Runge Kutta algorithm. The period of the steps and the period of calculation can be set independently. The low-pass filter used was a first order low pass filter with $\tau = 5 * T_s$ (The delay of the response of the filter should be slower than the sampling period). Firstly, I did not put any constraint on the weights of the output.

On the Santafe one-step prediction problem, we will use the same parameters as the previous test, except for $V_r = 0.8$, that give best results after a grid search:

Number of nodes	**450**
Connectivity	0.1
Sampling period	$T_s = 10^{-9}s$
Calculation period	$T_c = 10^{-11}s$
Tau	$\tau = 5.10^{-9}$
Input sampling	5
Lambda	$\lambda = 0.005$
Activation function	$[f: x \to thanh(x)]$, $[f: x \to NodeActivation(x, V_r = 0.8)]$

.

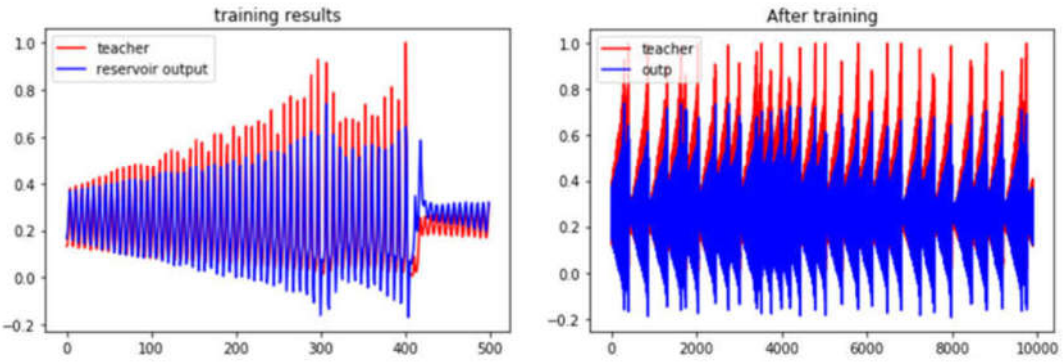


Figure 17: Dynamic simulation of the fpaa's reservoir

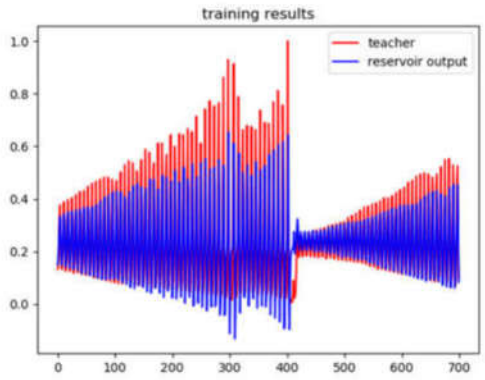


Figure 18: Results of the training for the Tanh activation on the Fpaa implementation

The condition of the training and test was the same as the previous test on the part 4's simulator.

The best results are:

Activation function	**Thanh**		**Activation($V_r = 0.8$)**	
NRMSE	0.96	-0.32db	0.73	-2.7db

These results are awful and cannot validate the implementation.

8. Discussion

These bad results can be caused by a bad software implementation of the problem or a reservoir that is truly inadequate for this problem. During my research, I did not find any similar simulation to create a reservoir and could not validate my implementation by testing a known problem with known setting and finding the known results**.**

Implementing and testing this reservoir on the DC RASP 3.0A should have been done but I did not have enough time to do so, the design of the node and the discussion with Aishwarya Natarajan on the use of RASP TOOL being held in April. It is necessary to at least try this reservoir on the board.

For this discussion, I will assume that the software implementation is working, and that the reservoir is not rich enough to solve the Santafe one step prediction problem.

The main differences between the ESN and our reservoir is the fact that the adjacency matrix is a stochastic matrix

- no negative weights

- No weights superior to 1

These limitations can hinder the richness of the reservoir. The prohibition of negative value on the board will cause most precaution using second order filter or more, that can attain negative values for $Q < 1$.

The approximation property is clearly not attained, so using a different readout function could be a solution to increase the performances of the reservoir. Using a non-linear input layer could also increase the richness of the reservoir.

To obtain a usable reservoir using a different implementation, one can try to:

- Use buffers in place of the filters, to obtain a classic ESN with condition on the adjacency matrix.
- Use the OTAs to implement a reservoir in current, using weighted current mirror does not cause a stochastic adjacency matrix.
- Try to implement a liquid state machine more than an echo state network, the literature contains several examples of successful liquid state machine's implementation.

9 Conclusion

In this report, I introduced the concept of Reservoir Computing and the principal notions behind this implementation of machine learning. I explained how to train these systems then I presented a software simulation that I developed to prepare the implementation of Reservoir Computing on the DC RASP3.0A FPAA. After validating the software implementation on known tasks and trying different settings to discuss the interest of the activation function and the connectivity of the reservoir, I started to design the electronic nodes that will be use on the FPAA. After validating the obtained activation function on a simple task on my simulator, I tried to simulate the whole board as a dynamic reservoir of NMOS nodes with low pass filters connected with capacitors. The results of the simulation could not validate the implementation, but different paths remain to implement a reservoir computing system on this board.

Bibliography:

[1]Yann Lecun,Pattern Recognition and Neural Networks (1995)

[2]JG Koomey," ESTIMATING TOTAL POWER CONSUMPTION BY SERVERS IN THE U.S. AND THE WORLD "(2007)

[3 Herbert Jaeger ''The "echo state" approach to analyzing and training recurrent neural networks'' (2001)

[4] Lennert Appeltant "Reservoir Computing based on Delay-dynamical Systems"

[5] Maass, Wolfgang; Natschläger, Thomas; Markram, Henry. "Real-time computing without stable states: a new framework for neural computation based on perturbations" (2002)

[6] H.Jaeger, Adaptive Nonlinear System Identification with Echo State Networks, 2004

[7]Mantas Lukosevicus, A practical guide to applying echo state network, 2012

[8]Francois Duport, Anteo Smerieri, Akram Akrout, Marc Haelterman & Serge Massar, Fully analogue photonic reservoir computer, 2016

[9] Y.Paquot, F.Duport, A.Smerieri, J.Dambre, B.Schrauwen, M.Haelterman & S.Massar, "Optoelectronic Reservoir Computing" (2012)

[10]Jialing Li, Chenyuan Zhao, Kian Hamedani, and Yang Yi, "Analog Hardware Implementation of Spike-Based Delayed Feedback Reservoir Computing System " (2017)

[11] Kian Hamedani_; Lingjia Liu_; Rachad Atat ; Jinsong Wu_; Yang Yi, Reservoir Computing Meets Smart Grids: Attack Detection Using Delayed Feedback Networks, 2018

[12]Hodgkin-Huxley Neuron and FPAA Dynamics Aishwarya Natarajan, Student Member, IEEE, Jennifer Hasler, Senior Member, IEEE 2018

[13]Puxuan Dong, Griff L. Bilbro, Mo-Yuen Chow,, Implementation of Artificial Neural Network for Real Time Applications Using Field Programmable Analog Arrays (2006)

[14]P. Dong, G. Bilbro, and M.-Y. Chow, "Controlling a Path-tracking Unmanned Ground Vehicle with a Field-Programmable Analog Array," IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Monterey, CA, 24-28 July, 2005.

[15]Helmy Widyantara, Muhammad Rivai, Djoko Purwanto, " Implementation Analog Neural Network for Electronic Nose using Field Programable Analog Arrays (FPAA)" (2012)

[16] Sihwan Kim, Jennifer Hasler, Integrated Floating-Gate Programming Environment for System-Level ICs (2015)

[17]Puxuan Dong ; G.L. Bilbro ; Chow, Implementation of Artificial Neural Network for Real Time Applications Using Field Programmable Analog Arrays (2006)

[18] Richard B. Wunderlich, FLOATING-GATE-PROGRAMMABLE AND RECONFIGURABLE, DIGITAL AND MIXED-SIGNAL SYSTEMS (2014)

[19]P. HASLER, Basics of Transconductance – Capacitance Filters

[20] Herbert Jaeger, Scholarpedia (2007)

[21] E. Hoerl et Kennard dans " Ridge regression : biased estimation for nonorthogonal problems"(1970)

[22] Harold Soh, Yiannis Demiris; Spatio-Temporal Learning With the Online Finite and Infinite Echo-State Gaussian Processes(2014)

[23]J. HASLER ; Finding a roadmap to achieve large neuromorphic hardware systems (2013)

[24]J. HASLER; Starting Framework for Analog Numerical Analysis for Energy-Efficient Computing (Journal of Low Power Electronics and Application - 2017)

[25] H. T. Siegelmann, E. D. Sontag; Analog computation via neural networks (Theorical Computer science 131 - 1994)

[26] P. HASLER, C. DIORIO, B. A. MINCH, C. MEAD; Single Transistor Learning Synapse with Long Term Storage, (Caltech – 1995)

[27]-T. SHIBATA, T. OHMI; Functional MOS Transistor Featuring Gate-Level Weighted Sum and Threshold Operation (IEEE Transactions on Electron Devices, Vol.39 – 1992)

[28] M. HOLLER, S. TAM, H. CASTRO, R. BENSON; An electrically Trainable Artificial Neural Network (ETANN) with 10240 "Floating Gate" Synapses (International 1989 Joint Conference on Neural Networks)

[29] C. R. SCHLOTTMANN, P. HASLER; A Highly Dense, Low Power, Programmable Analog VectorMatrix Multiplier: The FPAA implementation (IEEE journal on emerging and selected topics in circuits and systems, vol 1. No3. – 2011)

[30] C. R. SCHLOTTMANN, C. PETRE, P. HASLER; Vector matrix multiplier on field programmable Analog Array (2010 IEEE International Conference on Acoustics, Speech and Signal Processing- 2010)

[31] J. HASLER; Programmable and Configurable, Neurmorphically Inspired, Ultra-Low Power Computation, (ABQ Meeting – 2012, course)

[32] P. MASSON , J.L. AUTRAN; Transistor MOS à effet de champ, éléments de théorie et de pratique (Institut national des sciences appliqués de Lyon - 2001

[33] Michiel Hermans, Benjamin Schrauwen ; Memory in linear recurrent neural networks in continuous time (2010)

[34] J. HASLER; ECE 6414 - Multipliers and PLLs (Cours GeorgiaTech)

[35] The MOSFET Transistor in subthreshold (site :http://www.onmyphd.com/?p=mosfet.subthreshold.model)

[36] Aishwarya Natarajan and Jennifer Hasler, Modeling, simulation and implementation of circuit elements in an open-source tool seton the FPAA »

[37] Christian C. Enz, François Krummenacher, Eric A. Vittoz :"An analytical MOS transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications"