Georgia Tech Lorraine

CentraleSupélec

Final Report

# OTA RESERVOIR'S NEURON DESIGN

June 7, 2019

Benjamin Criton, Hugo Schindler

# Acknowledgments

We would like to thank Professor Damien Rontani and Professor Alexandre Locquet for their regular supervision and valuable support throughout this project.

Professor Jennifer Hasler allowed us to make this project a reality and for took the time to answer our emails, for that, we want to thank her profusely.

Thanks to the doctoral student Aishwarya Natarajan from Georgia Tech for the long interview, which was very useful and brought us key elements to carry on with the project.

Thanks to the Georgia Tech Lorraine laboratory technician for lending us equipment and its time.

# Contents

# List of Figures

# Introduction

Thanks to Professor Hasler from Georgia tech, we have been able to implement an electrical circuits on the Field Programmable Analog Array (FPAA)[2]. This board (Figure 1) is composed of analog blocs, digital blocs, routing capabilities, etc. that once connected, will create our neural network. Thanks to the connectivity of the board, we can process signals through the integrated circuit.

The main advantages of using an integrated circuit is that it is power efficient, it can be extremely fast, and the reconfiguration of the board enables the user to test different neurons to find the best one.

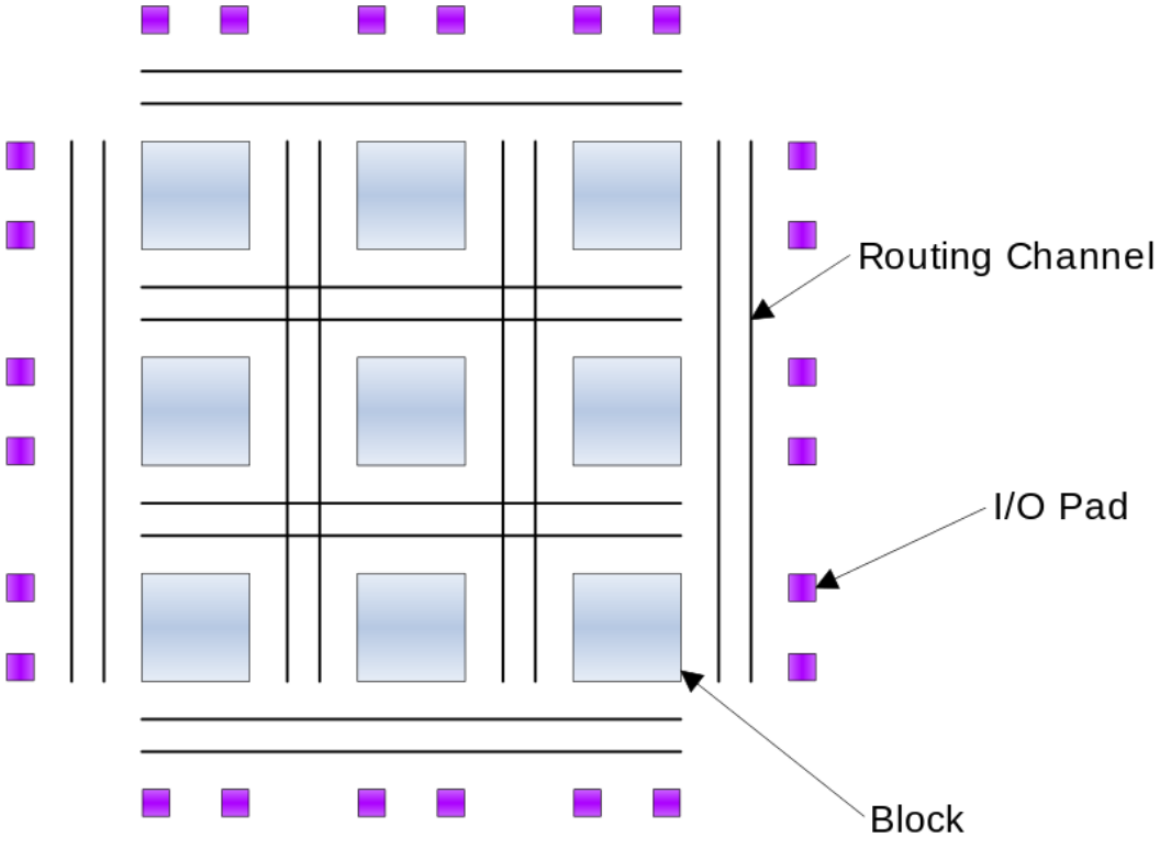


Figure 1: Schematic FPAA

Our goal is to implement a single node, delay-coupled virtual reservoir computing neuronal network with no feedback. Training will be supervised. The task is here to perform Wi-Fi channel equalization. This task is a first step towards the resolution of more complex tasks that neural networks can handle.

# Chapter 1

# Reservoir Computing

## 1.1 Neural network

A neural network is a method of computing based on the interaction of multiple interconnected nodes. After training them, they can solve the desired problem. This problem can be linear or not. Neural network applications include a large scale of domains such as pattern recognition or forecasting.

Because of the intrinsic differences in architecture between a microprocessor and a neural network, neural networks often need to be simulated. This is not an energy efficient solution.

This method, developed since 1943, might have a specific topology: one-layer or multi-layers of neurons. The connection between neurons can be feed-forward type or feedback type. Once your neurons are connected, the training can be supervised, unsupervised or reinforced.

## 1.2 Classification

This section will present a short overview of neural networks variety[6][7].

### 1.2.1 Topology

The simplest topology is a single layer of neurons as shown on the FIGURE 1.1.

By extension, FIGURE 1.2 is a multi-layer network.

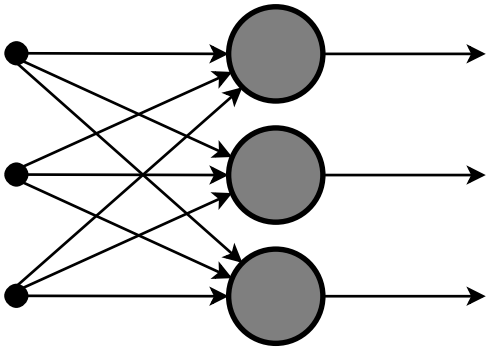

Figure 1.1: Single layer neural network

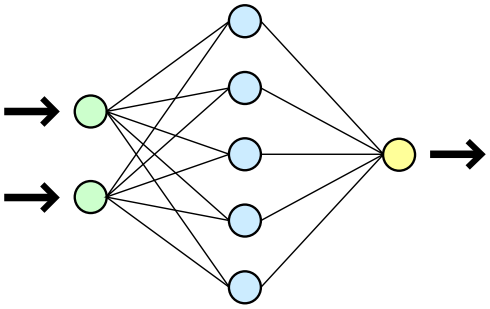

Figure 1.2: Multi-layer neural network

### 1.2.2 Connection type

The connection between neurons may only be able to go from the left-side to the right-side. This configuration is called a feed-forward network. If the connections can go in the opposite way, the configuration is called a dynamic feedback. FIGURE 1.3 is an example of a feed-forward network and FIGURE 1.4 an example of a feedback recurrent layer.

## 1.3 Learning method

### 1.3.1 Supervised learning

Supervised learning relies on training patterns. These patterns are made of a network input and a desired output. For each input sequence, the corresponding output sequence is recorded. Then adapted weights are computed to fit to the desired output. The aim is to set up a set of weights that minimize the error.

### 1.3.2 Unsupervised learning

On the opposite to the supervised learning, unsupervised does not require any training data or desired output. This kind of learning relies on learning by doing. Utilization of

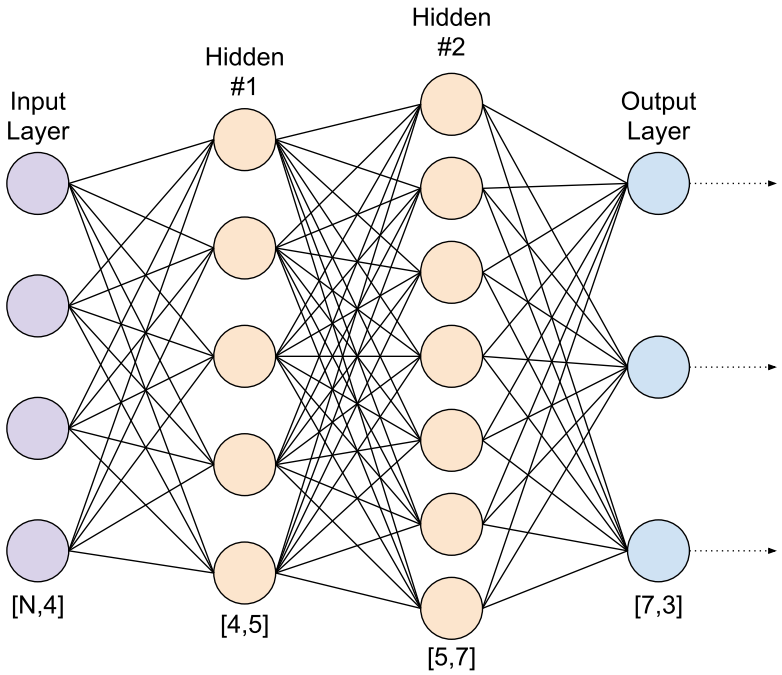


Figure 1.3: Feed-Forward multi-layer network

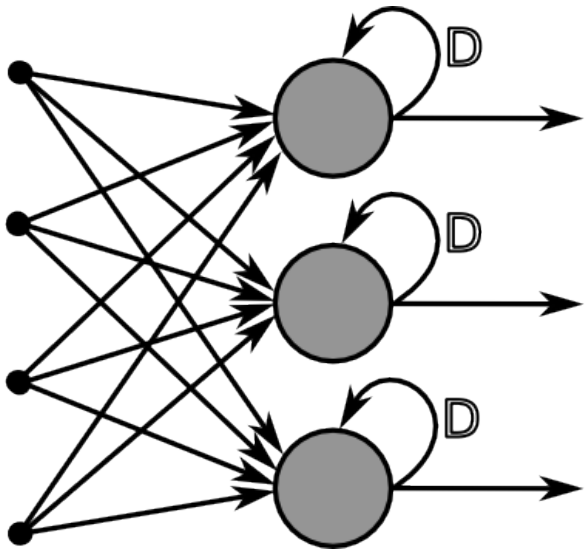


Figure 1.4: Feedback recurrent Layer

unsupervised learning are to solve clustering or compression problem.

### 1.3.3 Reinforcement learning

Reinforcement learning is not as extreme as unsupervised learning concerning data requirement. To apply this method, a teacher supplies training data and random generated weights. Then it scores the performance of the output. It is a slow learning method due to randomness, but it can be improved thanks to genetic selection.

## 1.4 Reservoir computing

Reservoir computing is a feedback, multi-layer network. FIGURE 1.5 is an example of a reservoir computing. The connectivity structure is usually random, and nodes are non-linear. Because of the feedback, the dynamics of the reservoir are affected by the past. It is easy to implement a supervised training on the readout layer. This kind of reservoir is widely used in photonics[9].

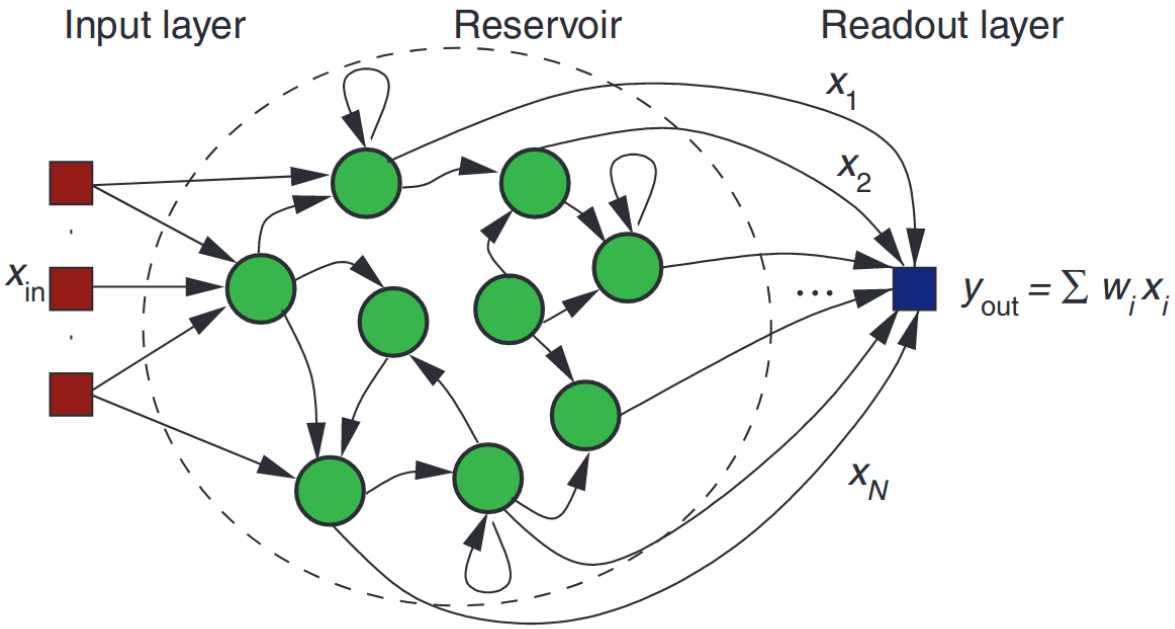


Figure 1.5: Reservoir neural network

# Chapter 2

# Selected Reservoir

Our goal is to implement a reservoir computing network. And to evaluate performance of this network, for this, we chose an easy task to solve.

## 2.1 Network description

Our simple network is a single node, delay-coupled reservoir computing neuronal network with no feedback. This network is a single neuron that answers to inputs with a step response time of $\tau$. The network is stimulated with a signal. With a higher sampling frequency, a scope collects the output of the neuron. If N samples are measured during a symbol, the single-node reservoir is equivalent to a virtual reservoir with N neurons. The FIGURE 2.1

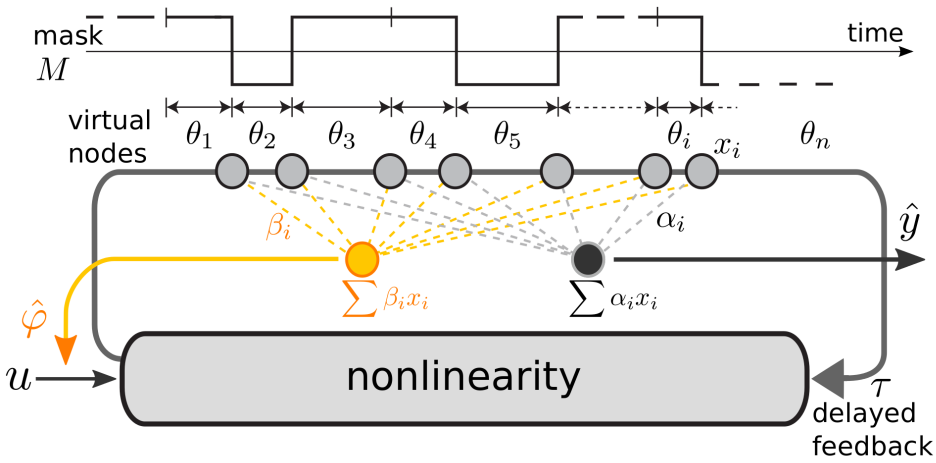


Figure 2.1: Single-node, delay-coupled reservoir

## 2.2 Task

The purpose of the Wi-Fi non-linear channel equalization task is to recognize noisy WI-FI symbols. Each symbol can be assigned a value from 4 levels. A simple threshold detector has 89% of success. The objective is to beat this method with the neural reservoir.

Our Wi-Fi non-linear channel equalization task is composed of 5000 symbols, about 4000 will be used for training, and 1000 for testing. The FIGURE 2.2 is the eye Diagram of the target signal. The FIGURE 2.3 is the eye Diagram of the noisy signal.

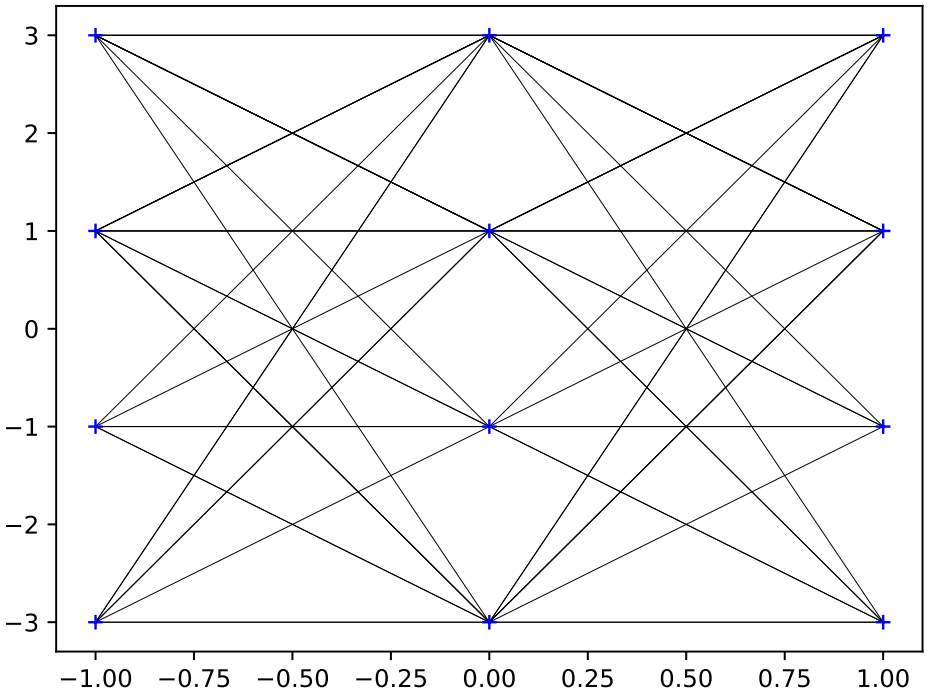


Figure 2.2: Eye diagram of the targeted signal

## 2.3 Input specification

To have richness in the output signal, the neuron must always be in a transition state and not in a stationary state. Since the response time of the neuron is $\tau$, if we choose a holding time of an input symbol greater than $\tau$, we risk putting the neuron in a stationary state.

The trick is to apply a mask to the input symbol. At each bit of the mask, we can measure the output of the neuron. Thus, we multiply the measured data, which is good for training. The duration of one bit of the mask is one fifth of the response time of the neuron. It must also be ensured that the rest of the mask is not periodic, and does not

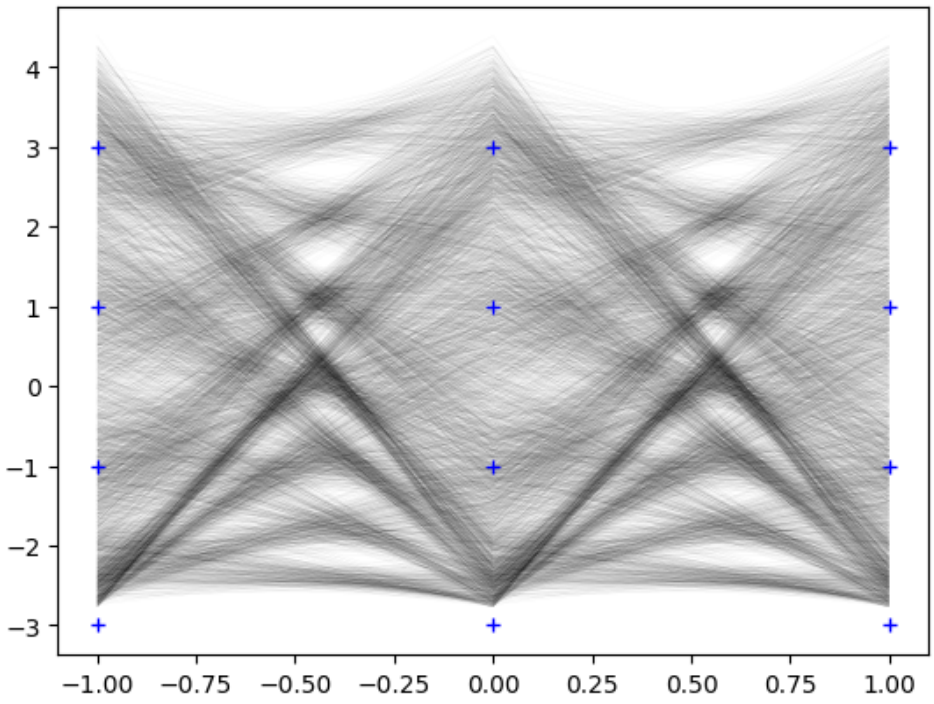


Figure 2.3: Eye diagram of the noisy signal

contain more than five consecutive circular bits. Otherwise the neuron's response to the mask will not benefit from the advantages of the mask.

The FIGURE 2.4 is an example of the beginning of an input signal. This signal starts with a blank symbol for an easy sync, then resized symbols are masked.

## 2.4 Training

For each bit of each symbol, several measurements are made. When generating symbols, the original symbol and the noisy symbol are saved. Thus, supervised training is applied once the signal has been processed by the board. A least squares method is used to determine the coefficients. Then these coefficients can be tested with another set of pre-processed data. We can do this because the treatment is at the end of the process.

We will design the neuron in the next chapters.

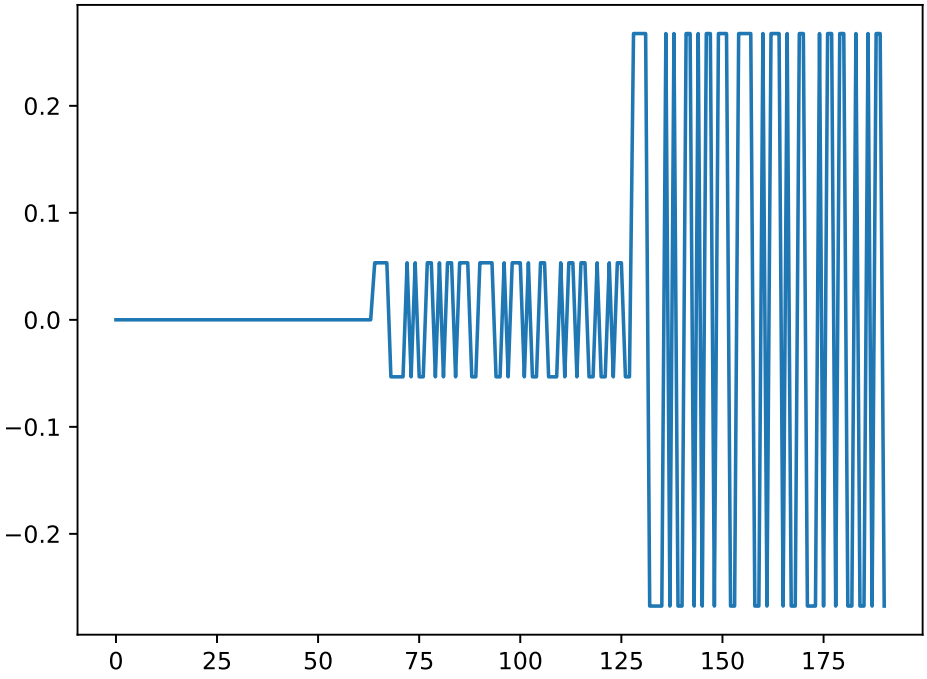


Figure 2.4: Masked input example

# Chapter 3

# Overview of a neuron node

Most artificial neural networks are composed of neurons interconnected with some properties, an input layer and an output layer. In our case, there will be only one neuron, since we want to design a Single-Node, Delay-Coupled Reservoir. However, the constitution of the neuron remains the same as in classical reservoirs. The FIGURE 3.1 details the constitution of an artificial analogue neuron.

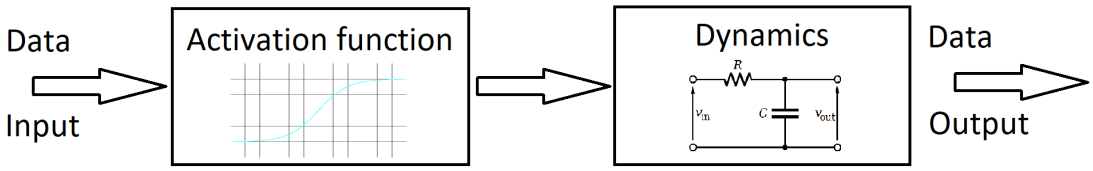


Figure 3.1: Schematic of a neuron

As we can see, there are two distinct parts in this neuron. An Activation Bloc allows distinctive inputs to have different outputs. A Dynamics Bloc filters the output of the Activation Bloc to add delay to the transient response.

## 3.1 Activation Bloc

The activation function is a non-linear function that transforms an input into an output. It is based on the action potential of brain neurons. In our neural network, we use a continuous function like a sigmoid/hyperbolic tangent. The benefit of using a continuous function opposite to an ON/OFF function is to have an infinity of output values depending of the input of the bloc. This means that each different input will create a different output, adding more information to the neuron. In a real neuron, action potential is the way a

neuron transmits information through electricity (sodium ion movement). Above a given threshold of the membrane potential, the neurotransmitter will be able to release sodium ion and so conduct information.

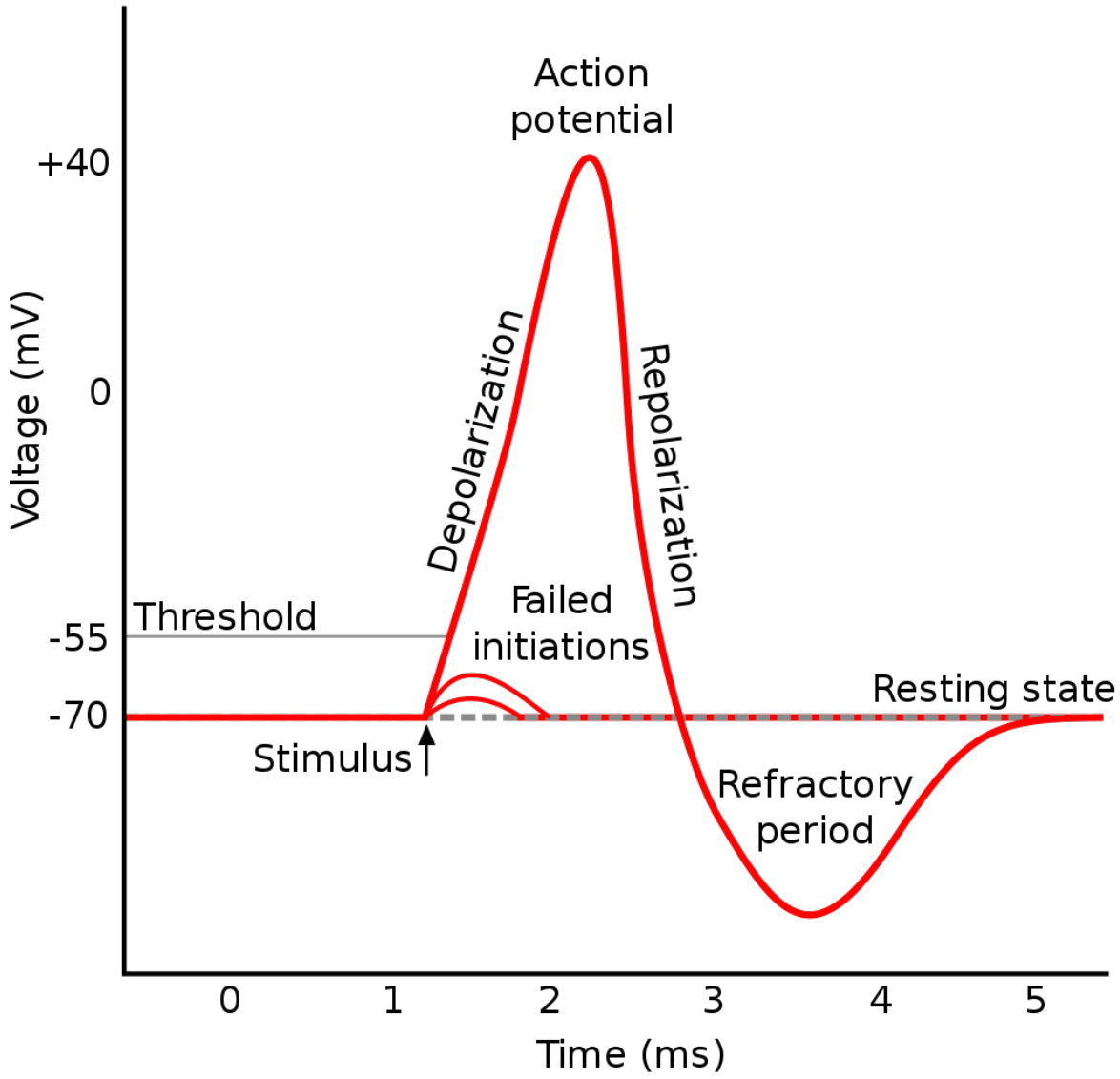


Figure 3.2: Action potential of a neuron

In reservoir computing, we want to reproduce the behavior of a real neuron. Thus, we use an activation function which is nonlinear, in order to approximate the action potential. Theoretically speaking, we can use every nonlinear function we want. But it has been proven that some activation function work better than the others, and from that, some properties surfaced :

- Nonlinear.

- Finite range.
- Continuously differentiable.
- Monotonic.
- Smooth functions with a monotonic derivative.
- Approximates identity near the origin.

Some of these properties are specific to given neural networks and thus may not be suitable for our study. But it still gives us a snapshot of which look an activation function must have.

The FIGURE 3.3 gives some usable nonlinear functions.

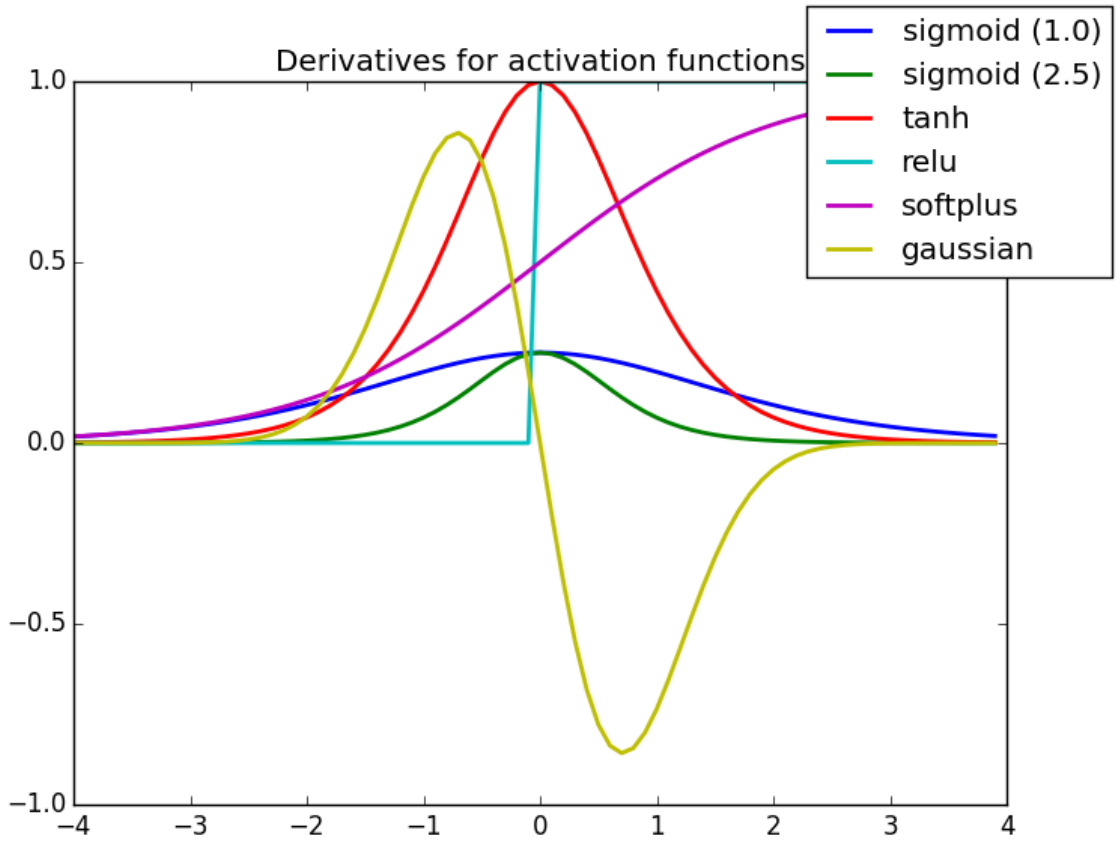


Figure 3.3: Action potential of a neuron

In our study, we will focus on the hyperbolic tangent/sigmoid activation function. Indeed, we will see that the available electronics components on the fpaa have this property.

## 3.2 Dynamics Bloc

The Dynamics bloc is used in order to add delay to the neural network. Thus, we can control the transient time of the neuron response. This filter will be useful when we will implement the Single-node, Delay-Coupled Reservoir because in this type of reservoir, we want the neuron to be stimulated with a signal which frequency is approximately the inverse of the time constant of the neuron. We will use a first order low pass filter for the Dynamics Bloc. This filter will give us the typical exponential response with square inputs.

# Chapter 4

# The 9 Transistors Transconductance Amplifier

The 9 Transistors Transconductance Amplifier (OTA) [5] is used in two distinct configurations for our neuron. A follower configuration is used to bufferize and filter the output signal of the Activation Bloc, and a simple comparator configuration is used to create the non-linearity in the Activation Bloc. Here, we will detail the operation of the OTA.

FIGURE 4.1 shows the internal structure of a nine transistor transconductance amplifier. This structure has been taken from [4].

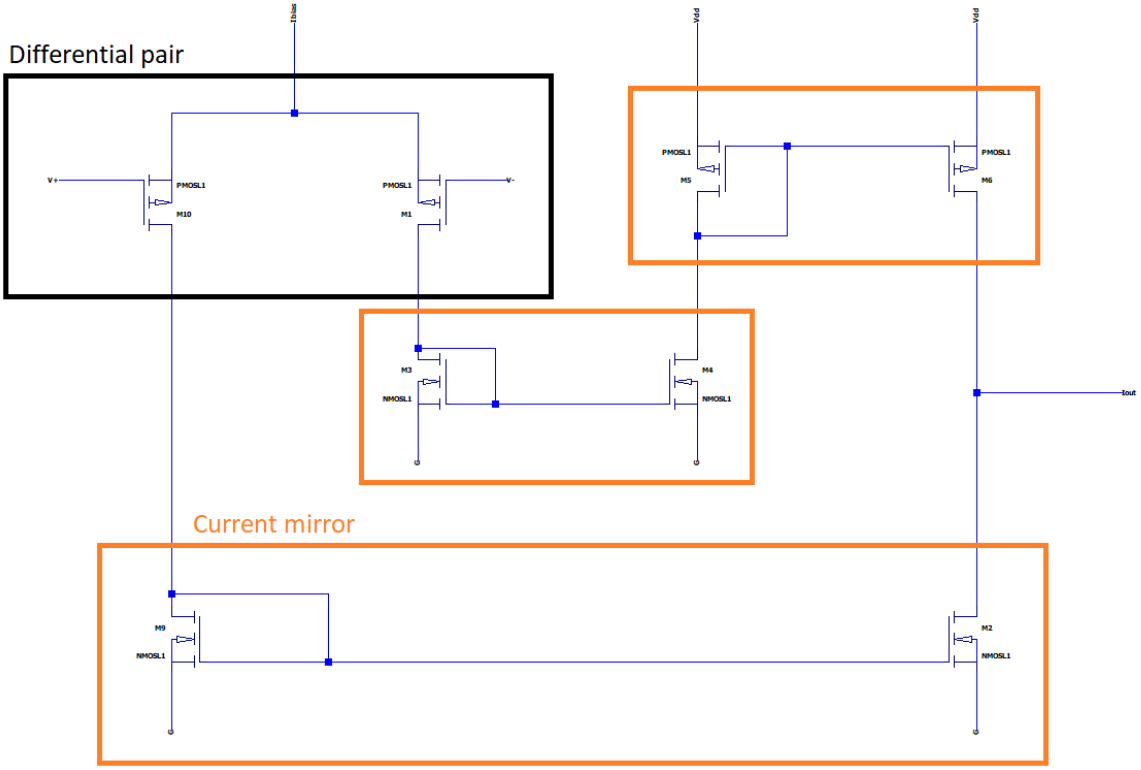


Figure 4.1: Structure of an OTA

We can see that there are three current mirrors that bring the differential current to the output of the OTA. The aim of these mirrors is to improve range characteristics of the device from Vdd to ground and the sensibility of the open-circuit output voltage as a function of input voltage. Thus, the output current is strictly the subtraction of Id(M10) and Id(M1).

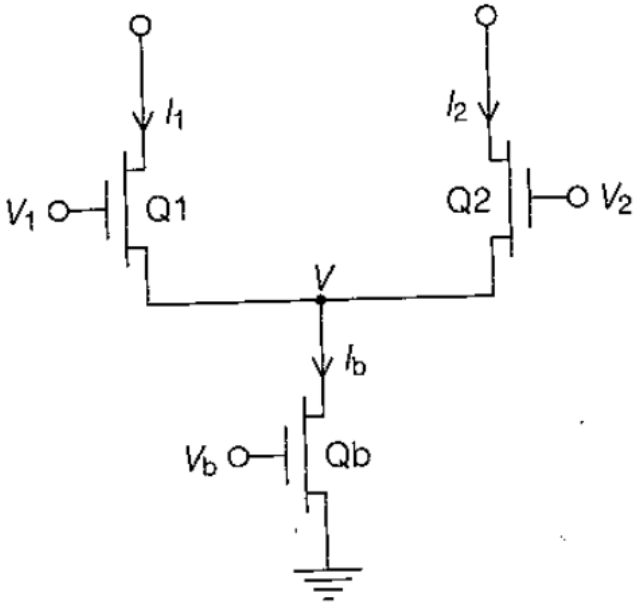


Figure 4.2: Structure of the differential pair

Let's call $I_1, V_1$ and $I_2, V_2$ the transistors drain current and grid voltage of the differential pair. Considering the circuit in the weak-inversion region, i.e. grid-to-source voltage below threshold voltage (Weak inversion: $V_{GS} - V_{TH} < -50mV$)[3]. The electric field is too weak to create the electron channel between source and drain. The behavior of the drain-to-source current as a function of grid to source voltage is exponential.

We have :

$$I_1 = I_0 e^{\frac{\kappa V_1 - V}{U_t}}, I_2 = I_0 e^{\frac{\kappa V_2 - V}{U_t}}$$

Thus,

$$I_1 + I_2 = I_0 e^{-\frac{V}{U_t}} (e^{\frac{\kappa V_1}{U_t}} + e^{\frac{\kappa V_2}{U_t}})$$

$$e^{-V} = \frac{I_1 + I_2}{I_0} \frac{1}{e^{\frac{\kappa V_1}{U_t}} + e^{\frac{\kappa V_2}{U_t}}}$$

So,

$$I_1 = (I_1 + I_2) \frac{e^{\frac{\kappa V_1}{U_t}}}{e^{\frac{\kappa V_1}{U_t}} + e^{\frac{\kappa V_2}{U_t}}} \quad \text{and} \quad I_2 = (I_1 + I_2) \frac{e^{\frac{\kappa V_2}{U_t}}}{e^{\frac{\kappa V_1}{U_t}} + e^{\frac{\kappa V_2}{U_t}}}$$

Yet, $I_1 + I_2$ is kept constant by the current sink made by transistor Qb. Thus, $I_1 + I_2 = I_b = cst$.

In our 9 transistors OTA, the configuration is a bit different, we have a current source instead of a sink. The current source is made by a floating gate MosFet : during programming, a constant charge is applied to the capacitance of the transistor's grid, commanding a constant voltage on the grid since no current moves from grid to source. This constant voltage insures a constant current $I_b$ to the source of the differential pair.

Thus, we have at the output of the last stage of the device :

$$I_{out} = I_1 - I_2 = I_b \frac{e^{\frac{\kappa V_1}{U_t}} - e^{\frac{\kappa V_2}{U_t}}}{e^{\frac{\kappa V_1}{U_t}} + e^{\frac{\kappa V_2}{U_t}}}$$

This result is very interesting for us because if we plot $I_{out}$ as a function of $V_1 - V_2$, we see a hyperbolic tangent. This will be the basis of our nonlinearity, and thus, we will use the OTA for the activation function. The configuration of the OTA allowing us to exploit this tanh characteristic is the simple comparator one.

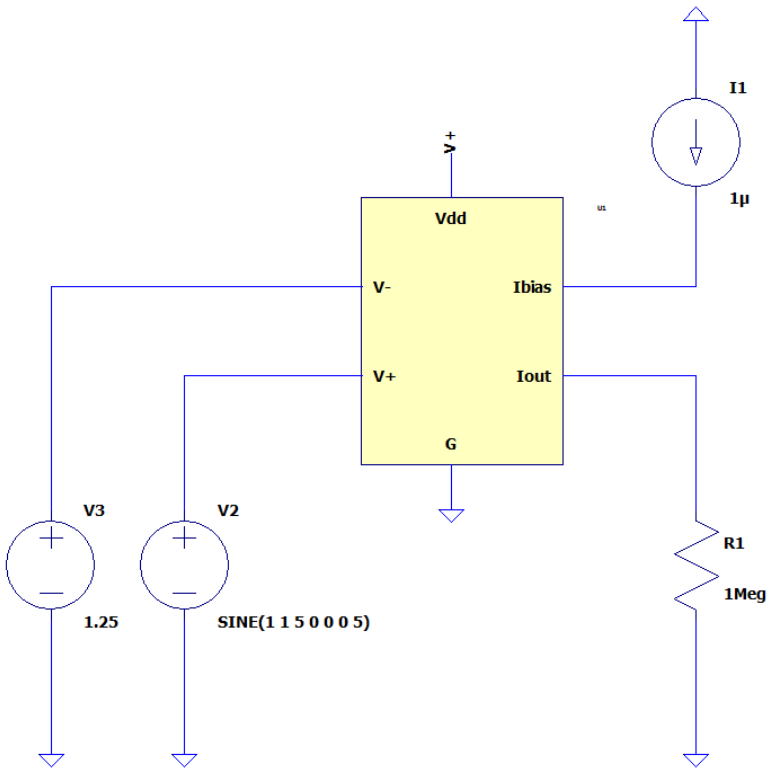


Figure 4.3: OTA in simple comparator configuration

Furthermore, we can consider the OTA as a regular amplifier and use it as a follower on condition that we keep our differential voltage in the linear part of the tanh relation. In this configuration, the output is fed back to the negative input. We thus have a buffer configuration.

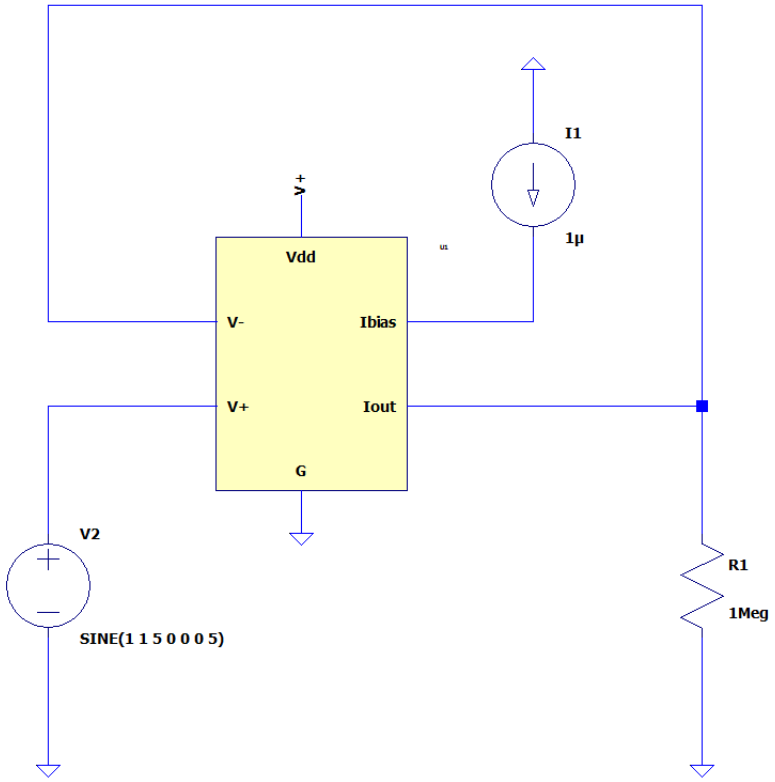


Figure 4.4: OTA in follower configuration

# Chapter 5

# Design of the Activation Bloc

## 5.1 Using of an OTA as the Activation Bloc

As we have said in the previous part, we want to create a nonlinear relation between the input signal and the output. This non-linearity will be a tanh-like function. We have also seen that the nine transistors OTA has a tanh relation between output current and differential input voltage. So, we will exploit this property to create our Activation Bloc.

$$I_{out} = I_1 - I_2 = I_b \frac{e^{\frac{\kappa V_1}{U_t}} - e^{\frac{\kappa V_2}{U_t}}}{e^{\frac{\kappa V_1}{U_t}} + e^{\frac{\kappa V_2}{U_t}}}$$

Thus, assuming $V_{in} = V_1 - V_2$, and factorizing by $e^{\kappa \frac{V_1+V_2}{2U_t}}$ both numerator and denominator, we finally have $I_{out} = I_b \tanh\left(\frac{\kappa V_{in}}{2U_t}\right)$

This relation gives the theoretical result of the FIGURE 5.1 given by the Scilab/Xcos simulation module. Moreover, the article [1] describes some of the expected results of the OTA on both simulation and experiments.

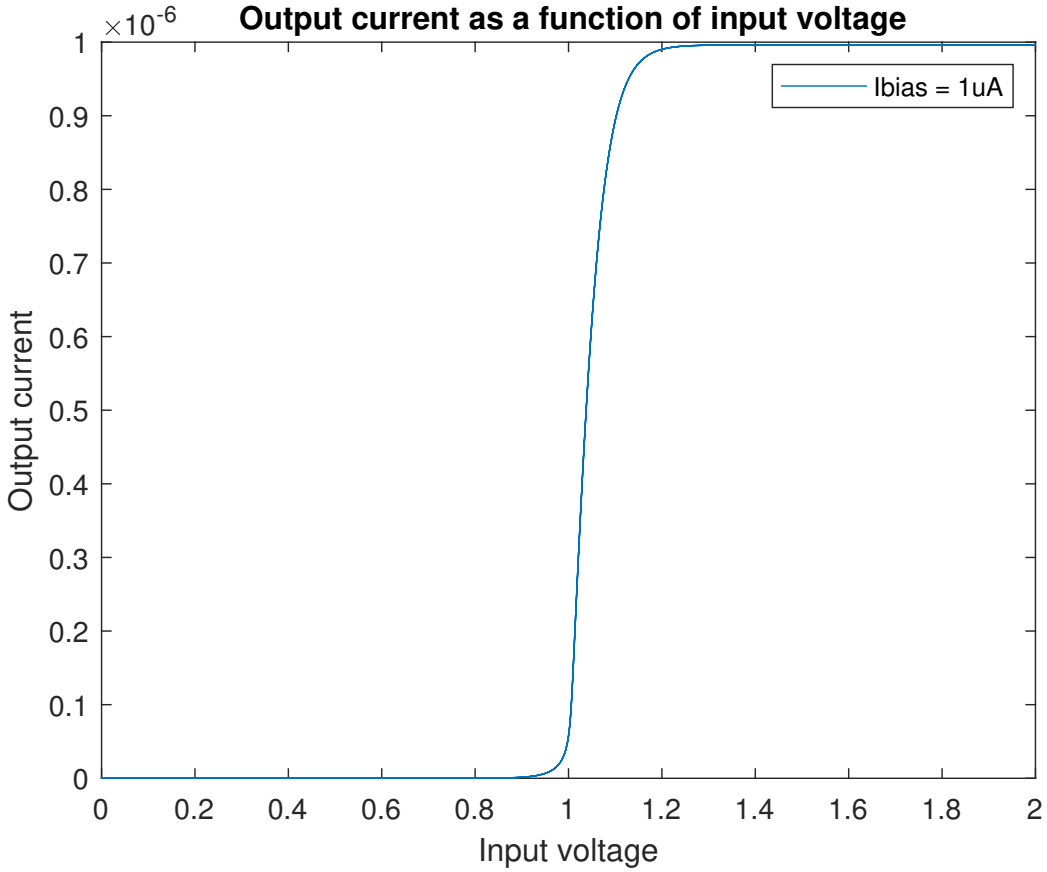


Figure 5.1: Theoretical tanh relation between input differential voltage and output current

We can see that in theory, we do have a tanh relation. The derivative function of $I_{out}$ gives us the value of the slope. This slope is important for our application since it determines how well different inputs will trigger different outputs.

Thus, we have $\frac{dIout}{d(V_{in})} = \frac{\kappa I_b}{2U_t}$, showing that we have control of the slope by adjusting the bias current of the OTA. Experiments have demonstrated that we do not have this versatility. There isn't this linear property of the slope as a function of $I_b$. We also experimented some issues with calibration and actual programming of bias current. Indeed, a 999pA bias current gives us a flat non-linearity, whereas a 1nA one gives us a steep step response.

In order to determine the characteristics of the OTA : $\kappa$, curve's look, etc., we conducted some experiments on the FPAA.

To have an idea of how the experiments were conducted, FIGURE 5.2 shows the temporal results of one data acquisition.

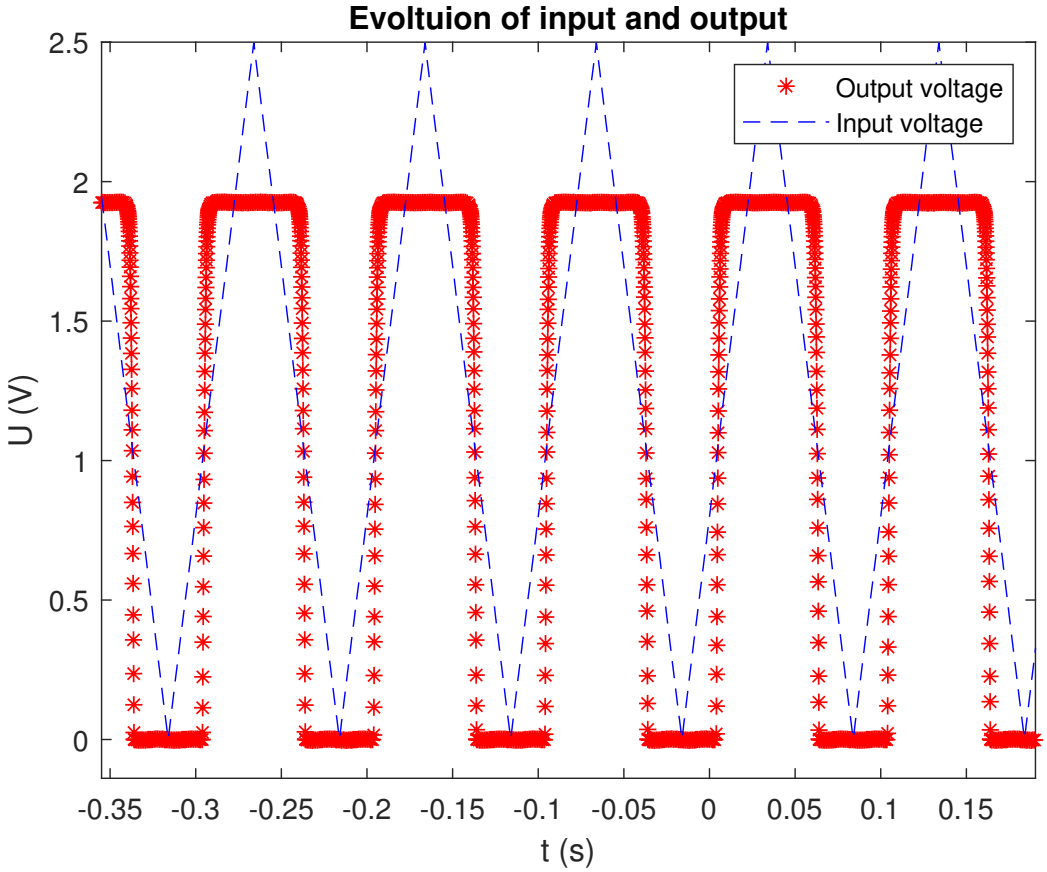


Figure 5.2: Temporal data of the simple comparator

In the FIGURE 5.3 we gather off chip data of the OTA simple comparator experiments and plot the mean curve of these experiments. To avoid the appearance of hysteresis, we only take the rising edges of the input and output signal. As a result of these acquisitions, we can then compute the derivative function of $I_{out}$ as a function of $V_{diff}$ (FIGURE 5.4).

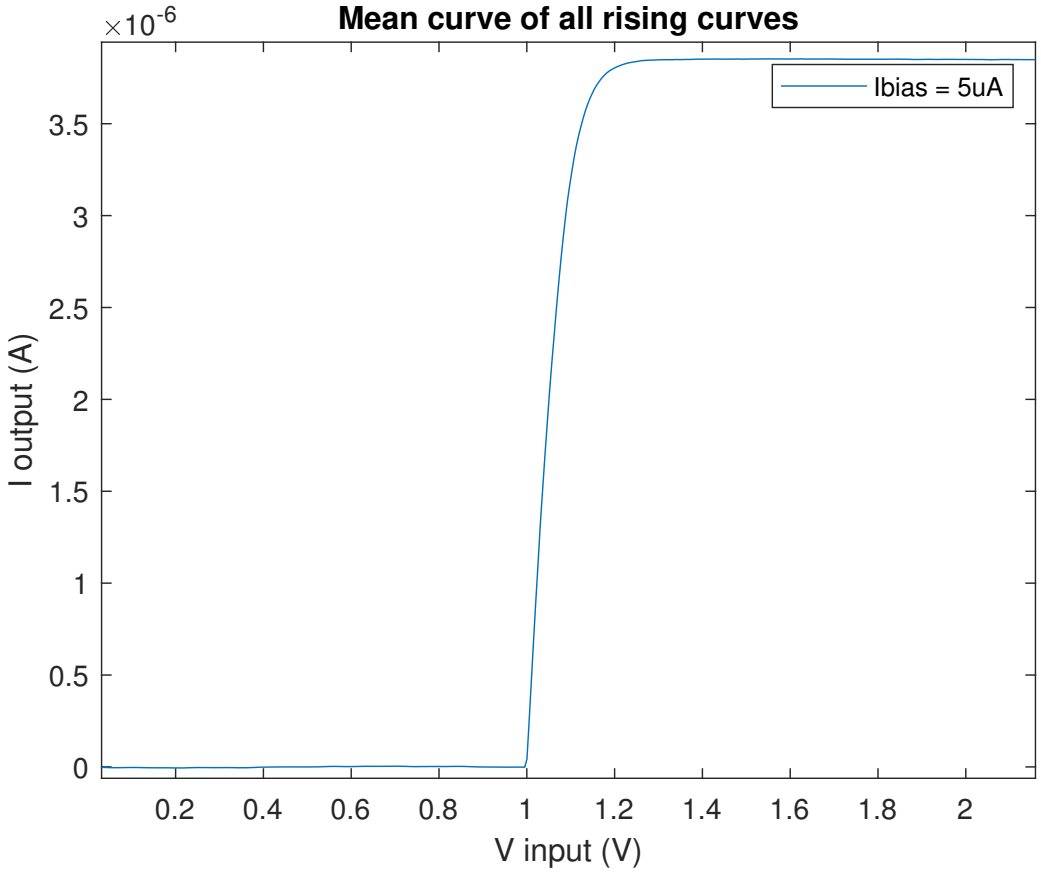


Figure 5.3: Experimental results of the OTA as a simple comparator

The derivative function gives us the transconductance of the OTA by the relation $G_m = \frac{\kappa I_b}{2U_t}$ The derivative has been calculated on the mean of multiple results in order to smooth its behavior with a simple Euler method. $G_m$ is the highest point of the derivative function, that is to say the inflection point of the tanh curve.

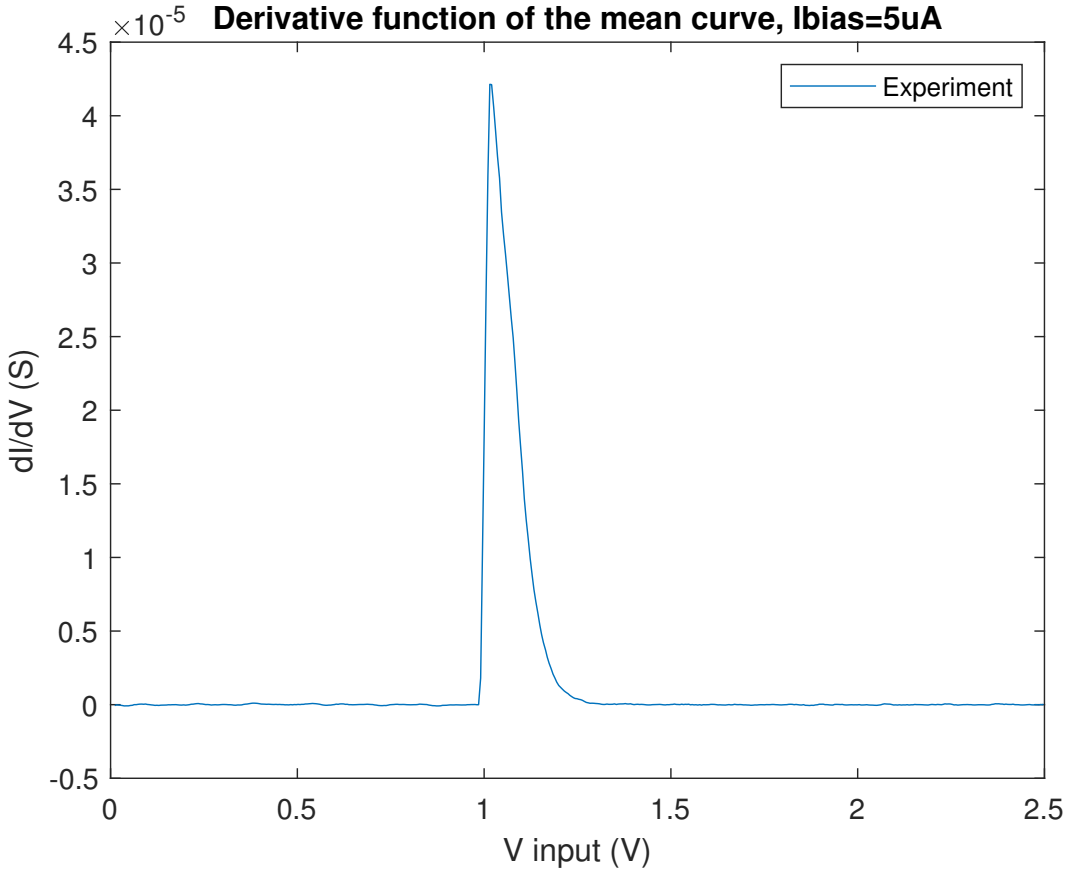


Figure 5.4: Derivative function of the previous curve

With these two figures, we can see that the output current doesn't reach the bias current (5uA) and saturates at 3.85uA. this can be due to transistor mismatch, FG MosFet current source not calibrated, output offset voltages, etc.. We didn't enter deep details about this fact, but it can be quite annoying when we want a precise result.

With FIGURE 5.5, we see the evolution of saturated output current parametrized by bias current. We can say that the evolution agrees with the theory since the saturated output current should be equal to the bias current of the OTA. Thus, rising the bias current must rise the saturated output current. This is what we can observe.

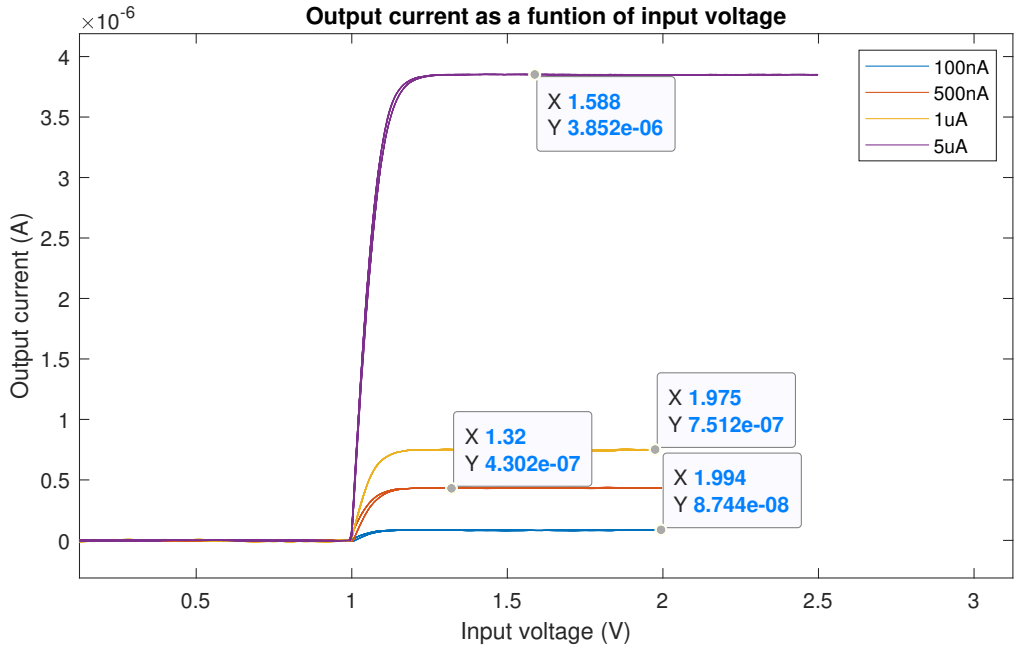


Figure 5.5: Results of five experiments with different bias currents

Saturated output current relative errors are as follow :

| $I_{bias}$ | experimental value | targeted value | error in % |
|---|---|---|---|
| 100nA | 87.44nA | 100nA | 12.56 |
| 500nA | 430.2nA | 500nA | 13.96 |
| 1uA | 751.2nA | 1uA | 24.88 |
| 5uA | 3.853uA | 5uA | 22.94 |

Table 5.1: statistics of the comparator experiments

These quite significant errors might be reduced by a proper calibration of the FPAA. We did not have the devices to perform calibration at GeorgiaTech Lorraine.

## 5.2 Considering the input signal's characteristics

The FIGURE 4.3 shows the functions connecting the input to the output for an OTA device. We can see that the non-linearity is steep i.e. the transconductance is high. This is expected since we are dealing with a comparator whose characteristic is to swing fast from OFF to ON. So, the range of linearity of the OTA is quite small : 0.2V for a bias current of 5uA. This can be limiting when we will input a signal. Indeed, if the signal is a square wave whose amplitude is above 0.2V, the output signal will be an identical square wave with a constant amplitude, independent of the input signal's range. This

is not wanted. We need a neuron that responds differently to different inputs. We can rightly think that reducing bias current will smooth the slop, of the transfer function. FIGURE 5.5 brings an answer to this : we can see that it indeed smooths the slop but it doesn't enlarge the linearity range. Then, we can imagine using signals having a range of about 0.2V. This can be achieved. But the smaller the linearity range is, the smaller the richness of input signals we will be able to have. Indeed, if the linearity range is small, it will be hard to have signals with distinct amplitudes at the input, and then, we won't be able to efficiently separates each input signal. To conclude, we don't have control of this parameter, and this is really limiting for our Activation Bloc.

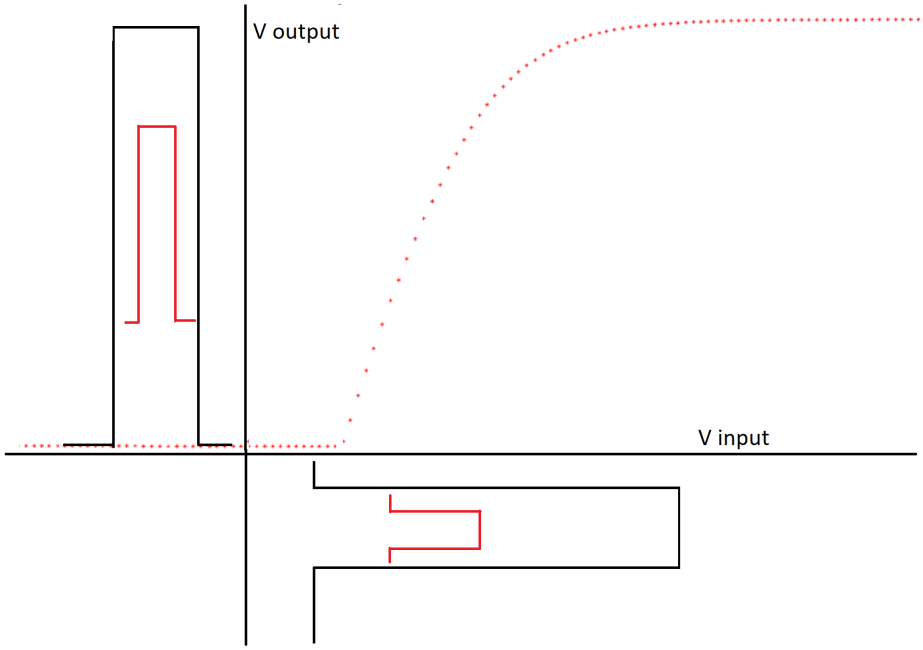


Figure 5.6: Output vs input signal amplification and saturation

This simplified figure (5.6) doesn't take into account the non-linearity of the transfer function between the input and the output.

## 5.3 Using an FGOTA as the Activation Bloc

On the FPAA board, we can access Floating Gate OTAs. FGOTAs are regular transconductance amplifiers with an added capacitor voltage divider on each of its two differential inputs (see FIGURE 5.7). The voltage divider constituted of the two capacitors allows us to increase the linearity of the signal but will decrease the gain of the bloc. Indeed, the voltage divider will divide the input voltage, and thus instead of having a 0.75mV linearity, we will get a 750mV one. This operation, even if it is made with capacitors, is theoretically independent of pulsation.

Indeed,

$$V_+ = \frac{j10fF\omega V_1}{j10fF\omega + j90fF\omega} \implies V_+ = \frac{1}{10}V_1$$

Which shows that the voltage applied on the differential inputs are independent of pulsation.

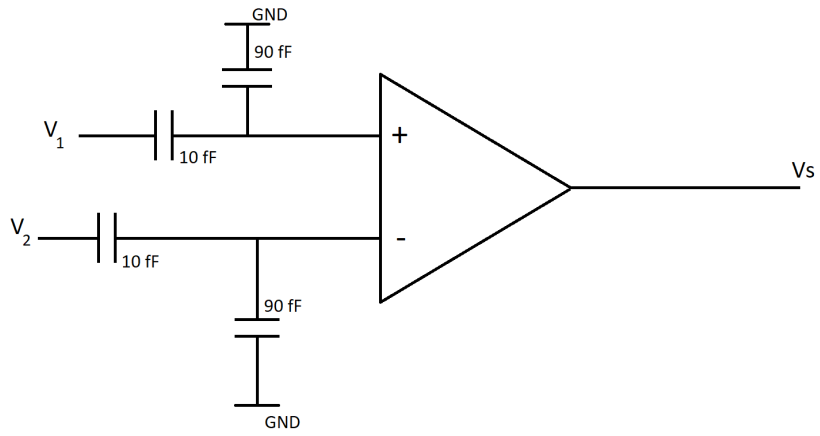


Figure 5.7: FGOTA circuit

Furthermore, during programming, an offset voltage is added to the ground-connected capacitor. This offset will modify the comparison level of the OTA. We chose to set it to 1.6V on both inputs for frequency efficiency reasons.

When we tested this bloc, we observed that the frequency response is very reduced in comparison with a regular OTA. This is unfavorable for our application. Indeed, we will have to process a signal at a certain rate. If the cutoff frequency of the FGOTA is limiting, we will have to reduce the signal rate, and thus the reservoir will be very slow.

The FIGURE 5.8 and 5.9 show the non-linearity we obtain when using a FGOTA. we can see that it has a more tanh looks than the simple OTA. It is also divided into two segments in the "linear" part.

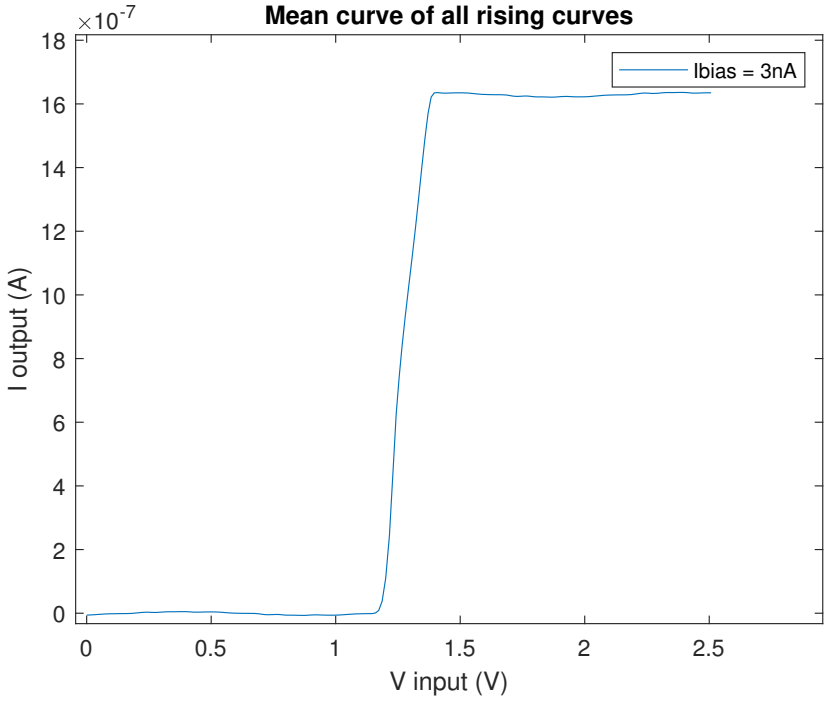


Figure 5.8: Non-linearity of the FGOTA, 3nA bias current, 1.6V offset on FG

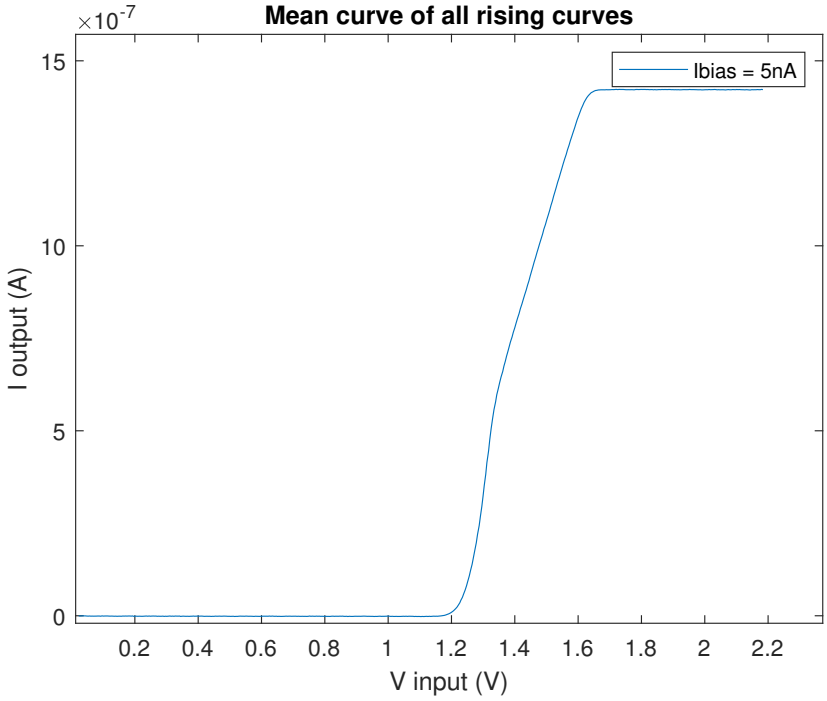


Figure 5.9: Non-linearity of the FGOTA, 5nA bias current, 1.6V offset on FG

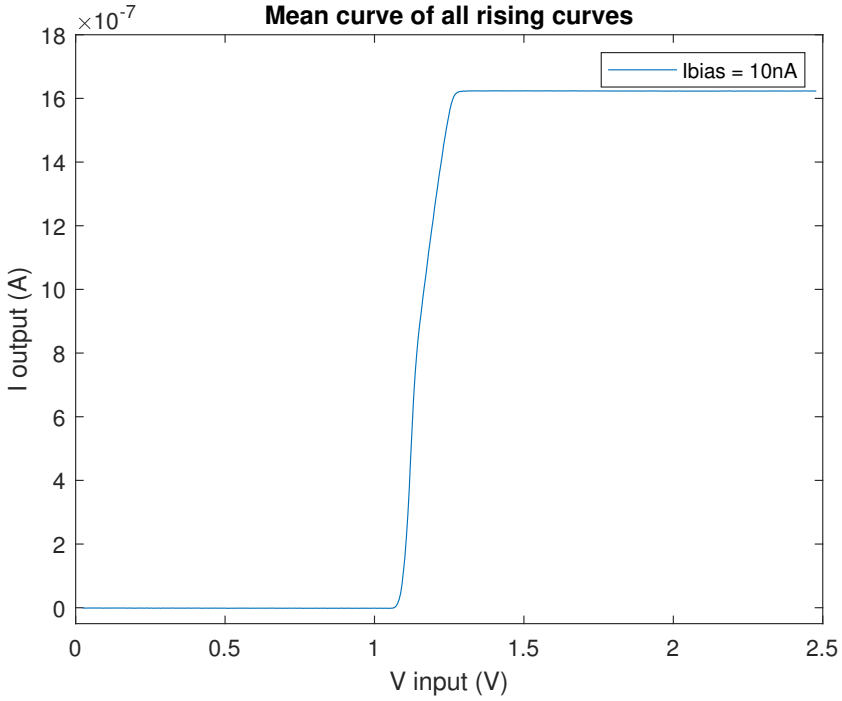


Figure 5.10: Non-linearity of the FGOTA, 10nA bias current, 1.6V offset on FG

These three figures show us the non-linearity of the FGOTA at different bias current (3nA, 5nA, and 10nA) at the same offset voltage. It is striking that the evolution of the slop with bias current is not as predicted. Indeed, bias current should only affect the OTA stage and not the capacitor bridge. We can see here that when increasing the bias current, we do not increase the slop (especially between 3nA and 5nA). This is another question we have about this board.

We will still use this component for our neural network and then compare it to the regular OTA.

# Chapter 6

# Design of the Dynamics Bloc

To design the Dynamics Bloc, we will also use an OTA, but this time in a follower configuration.

Using the parasitic capacitance of the integrated circuit and the follower as a current controlled resistance, we can create a first order low pass filter. The cutoff frequency can be modified using the OTA's bias current and thus its equivalent resistor.

Tuning of the Dynamics Bloc is very important because its cutoff frequency will depend on the frequency and amplitude of the input signal. Indeed, we want our neuron to constantly be in a transient state. In order to facilitate manipulations, we set the cutoff frequency and adjust the input characteristics to adequate the neuron's characteristics.

We conducted experiments in order to determine the relation between bias current of the OTA and cutoff frequency. We also want to estimate the parasitic capacitance and understand its behavior[1].

To do this, we first must measure the value of the $\kappa$ parameter of the OTA. This can be achieved with the calculation of the derivative function of the response of the OTA as a simple comparator.

Indeed, $G_m = \frac{\kappa I_b}{2U_t}$. Taking $I_b$ as the maximal current value when the output is saturated, and computing $G_m$, we can deduct the value of $\kappa$. Thereby, we calculated the value of $\kappa$ for different bias current, and for each bias current we did multiple data acquisitions. In the aim of determining the behavior of our OTA device, we want to know if $\kappa$ depends on bias current, is consistent through multiple acquisitions and if it is near the theoretical value.

Thus, we have statistically evaluated the consistency of the $\kappa$ parameter. Indeed, knowing the programmed bias current and measuring the transconductance, we can deduct the value of $\kappa$, which is normally a constant value. FIGURE 6.1 is a box plot of the $\kappa$ values as a function of bias current. To aggregate these results, we measured five responses of the comparator and calculated the standard error, the mean value... Each of the five

measurements are in fact the mean of a dozen other measurements. The data set is not big but in fact if we take more measurements, it won't improve the precision of the statistics because the OTA always have overall the same response to the input signal.

| $I_{bias}$ | $\bar{\kappa}$ | $\sigma$ | 95% interval |
|---|---|---|---|
| 100nA | 0.7819 | 0.0669 | [0.7221 ; 0.8417] |
| 500nA | 0.6991 | 0.0209 | [0.6804 ; 0.7178] |
| 1uA | 0.6923 | 0.0361 | [0.6600 ; 0.7246] |
| 5uA | 0.5782 | 0.0119 | [0.5676 ; 0.5889] |

Table 6.1: Statistics of the $\kappa$ parameter

From these experiments we can conclude that the values of $\kappa$ are not very consistent but are still in a good range around 0.65.

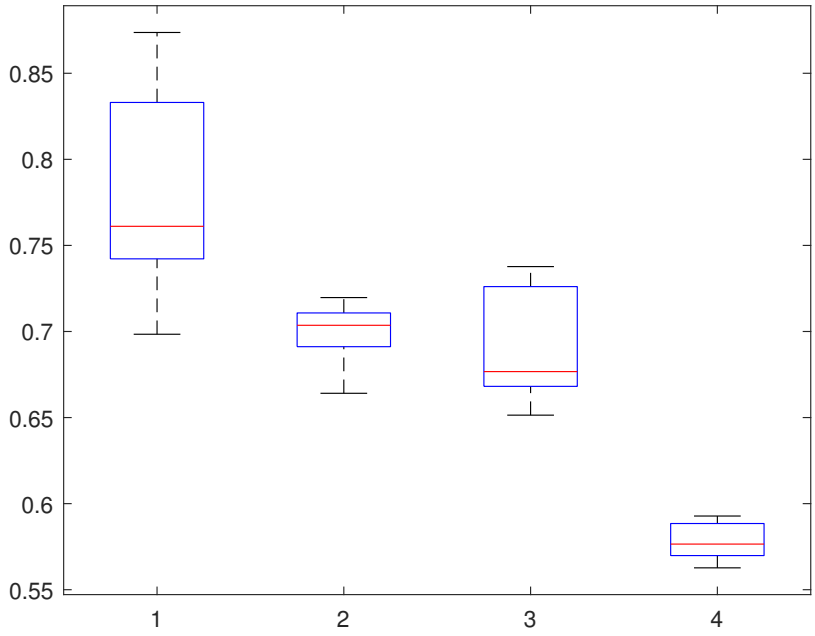


Figure 6.1: Box diagram of the $\kappa$ parameter for different bias currents

The box diagram shows the results of experiments aiming at determining the $\kappa$ parameter consistency as a function of bias current. We can see a downward trend in the $\kappa$ values with the rising of bias current {100nA 500nA 1uA 5uA}. This experiment will prove quite important when we will want to setup a low pass filter because the cutoff frequency formula involves $\kappa$

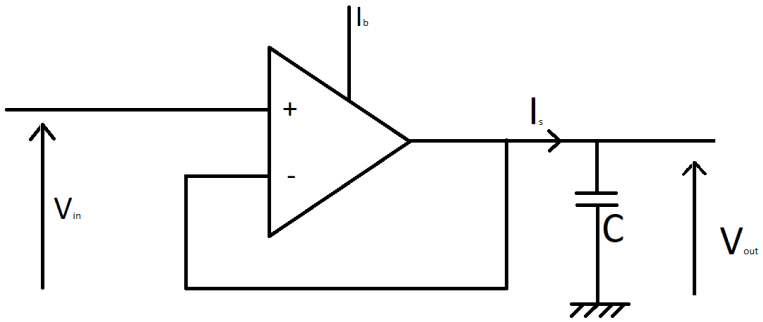


Figure 6.2: Schematic of the OTA LPF

We thus have the following equation:

$$C\frac{dV_{out}}{dt} = I_{bias}\tanh(\kappa\frac{V_{in}-V_{out}}{2U_t})$$

If we are in the linear part of the tanh, that is to say $V_{in} - V_{out} << \frac{\kappa}{2U_t}$, we can approximate the differential equation by:

$$C\frac{dV_{out}}{dt} = \frac{I_{bias}\kappa}{2U_t}(V_{in}-V_{out})$$

In harmonic regime:

$$V_{out} = \frac{1}{1+j\frac{2CU_t}{\kappa I_{bias}}\omega}V_{in}$$

and thus,

$$\omega_c = \frac{\kappa I_{bias}}{2CU_t}$$

$$f_c = \frac{\kappa I_{bias}}{4\pi CU_t} \approx 2.2\frac{I_{bias}}{C}$$

$$C = \frac{\kappa I_{bias}}{2\omega_c U_t}$$

The series of experiments consisted in using the OTA as a follower and the internal capacitance of the chip. We computed different values of bias current in order to modify the cutoff frequency as seen previously.

We saw that in the .blif compilation file of the on-chip LPF example, the bias current is explicitly written. For a 100Hz cutoff frequency, the bias current written in this file was $4.67 * 10^{-10}A$. So, in our experiments, we computed bias current around 440pA.

Matlab gives us the opportunity to transform any time response into frequency response with the **tfest** function. We used this function in order to output a first order low-pass

model of our time response.

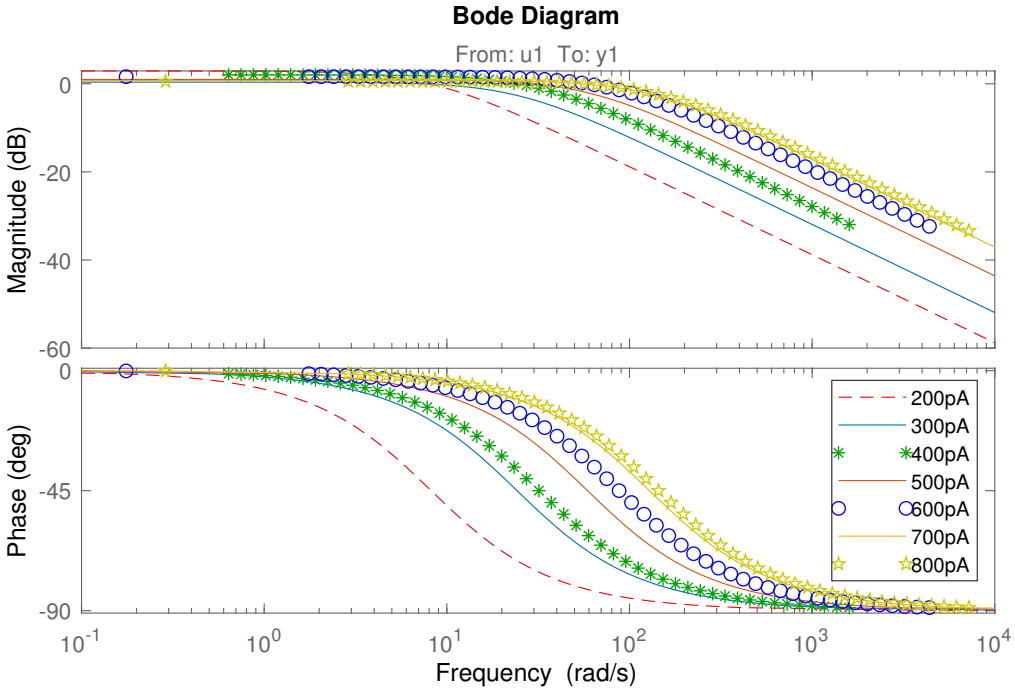


Figure 6.3: Bode response of the LPF

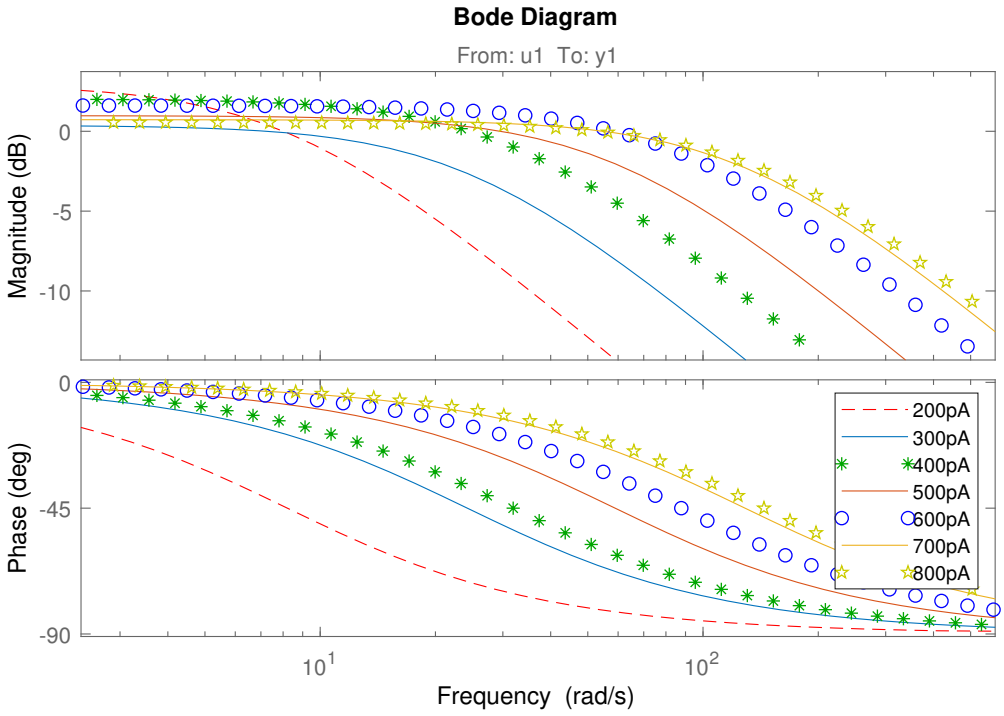


Figure 6.4: Bode response (zoom) of the LPF

We can see on the FIGURE 6.3 that a bias current of 467pA doesn't give us the 100Hz

cutoff frequency given by the compilation file. In fact, this result can only confirm the idea that the device is not calibrated. At 500pA, we have a 22.9Hz cutoff frequency instead of approximately 100Hz.

All the same, we have a first order bode diagram, and it is what we are looking for.

To confirm that we do have a first order filter, we have conducted an experiment with an external oscilloscope. FIGURE 6.5 shows the results of this off chip experiment. We can see that we have a first order behavior with a 0dB bandwidth gain. Furthermore, if we compare the cutoff frequency of this figure with the one of the FIGURE 6.3, we have a difference of 27Hz (32Hz for the on chip experiment, 5Hz for the off chip), this is a very bad result but maybe expected considering troubles with calibration and our experience with this board...

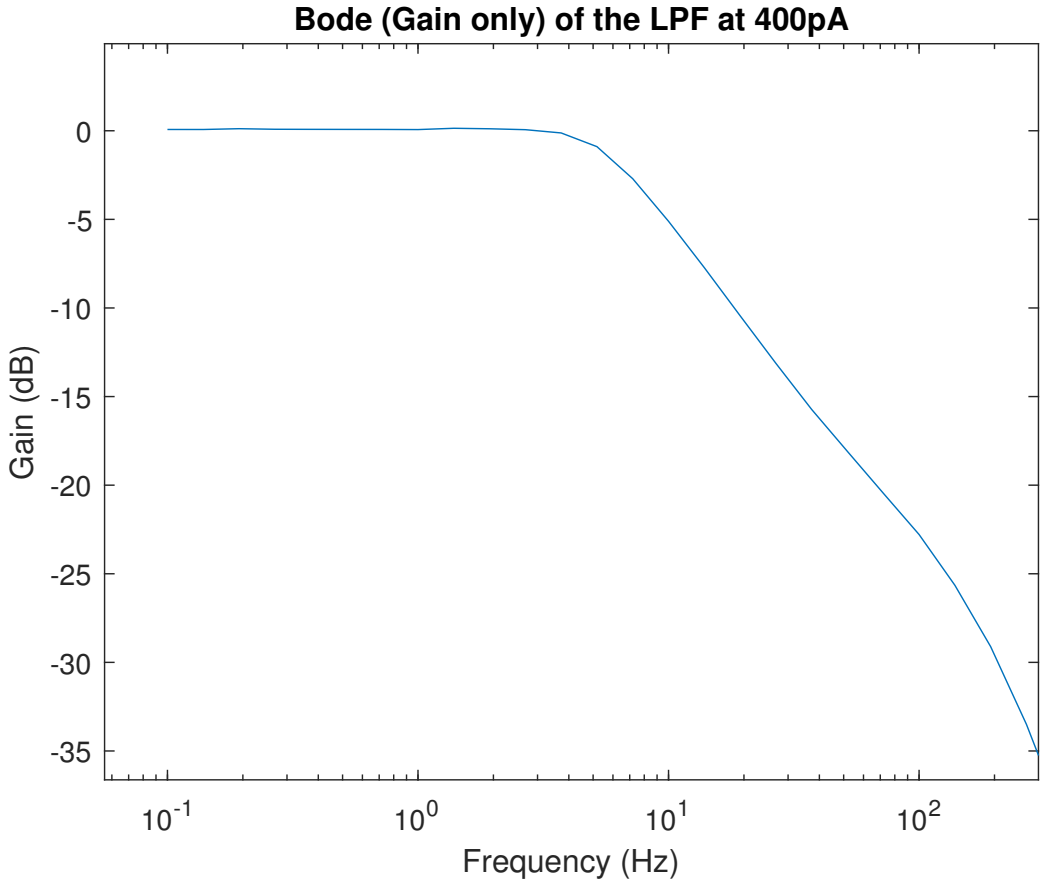


Figure 6.5: Bode response (off chip) of the LPF

Furthermore, it also seems that we have a linear relation between cutoff frequency and bias current

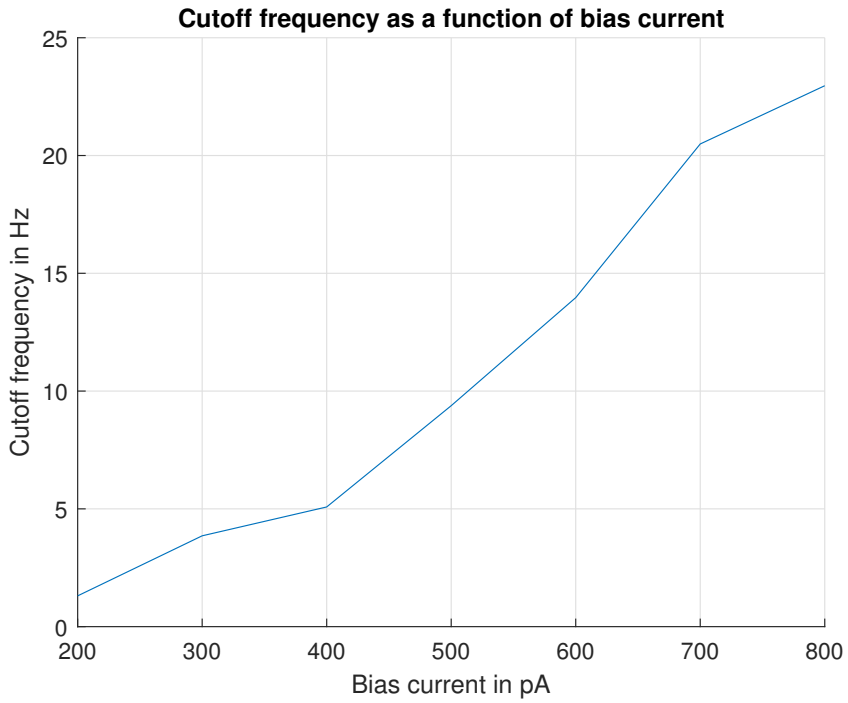


Figure 6.6: Evolution of cutoff frequency with bias current

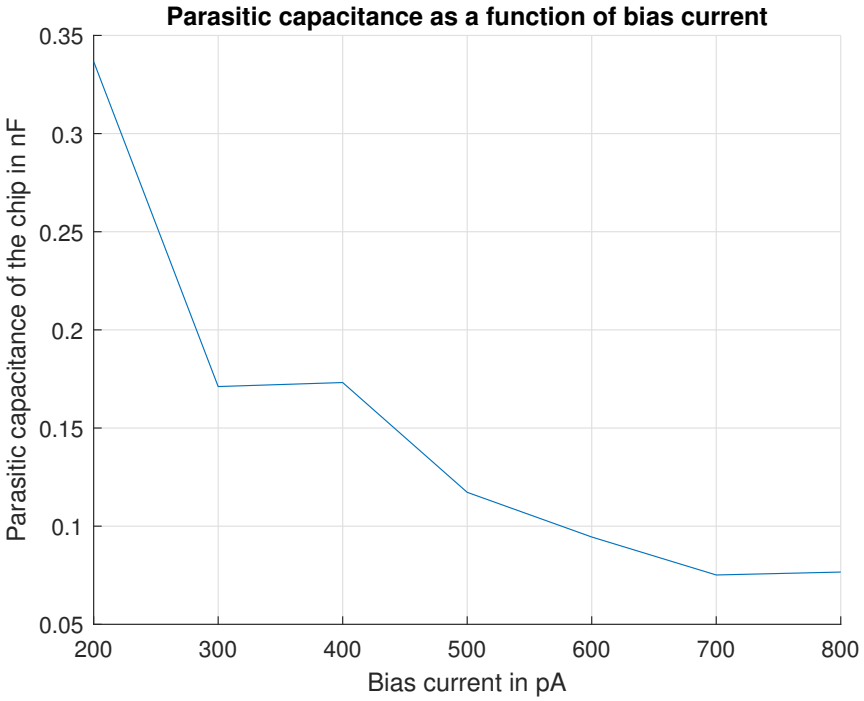


Figure 6.7: Evolution of parasitic capacitance with bias currents

In **Tuning of Multiple Parameters With a BIST System**[8] Sahil Shah and Jennifer Hasler highlight the importance of parasitic capacitance in the FPAA. They

mention some parasitic capacitances : Load capacitance $C_L$, feedback capacitance $C_2$ and input parasitic capacitance $C_W$. The FIGURE 6.8 taken from the same paper sums up these parasitic capacitances. Still in this paper, we learn that the load capacitance $C_L$ responsible for the good realizsation of our low pass filter varies "for different CABs and their route". That is understandable since each CAB is located on a part of the chip and the routing can be quite lengthy, increasing the load parasitic capacitance.

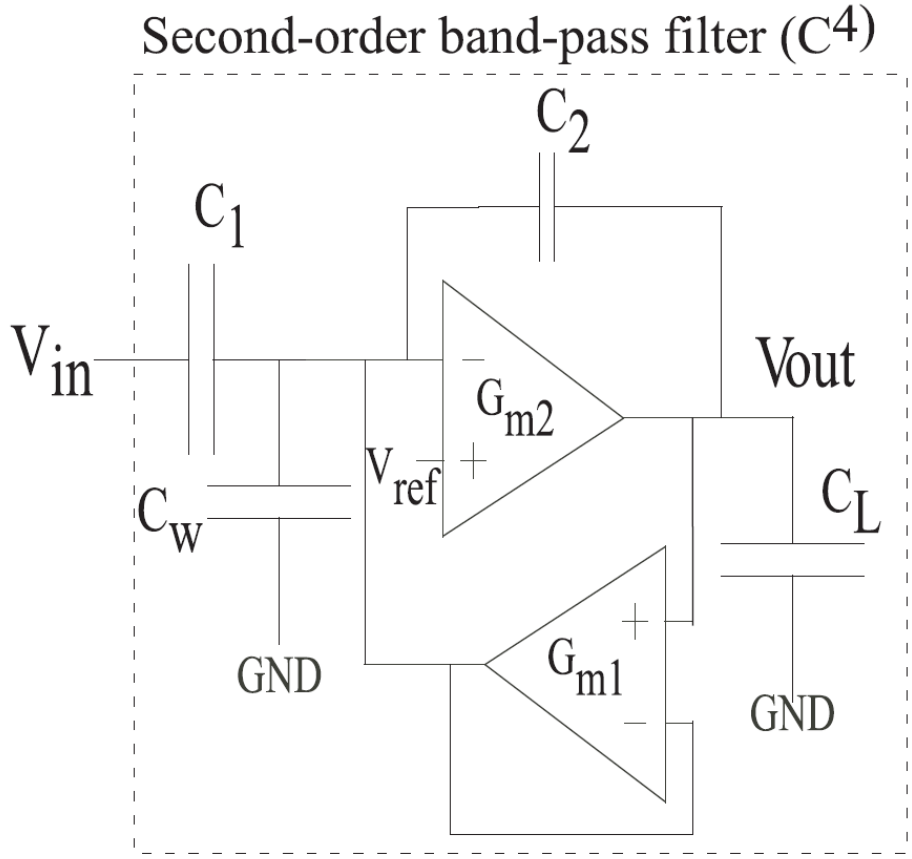


Figure 6.8: Parasitic capacitances of the OTA

Finally, we must be careful about impedance adaptation with the external oscilloscope. In order to avoid a major shift in results, we place an OTA in the follower configuration with 10uA bias current (Pr. Hasler in one of her course videos talks about a 60MHz cutoff frequency at this current level). So, this second OTA won't normally interfere with the low pass filter and guaranty a good impedance adaptation.

Yet, we can evaluate the input impedance of the oscilloscope stage which won't disturb our filter.

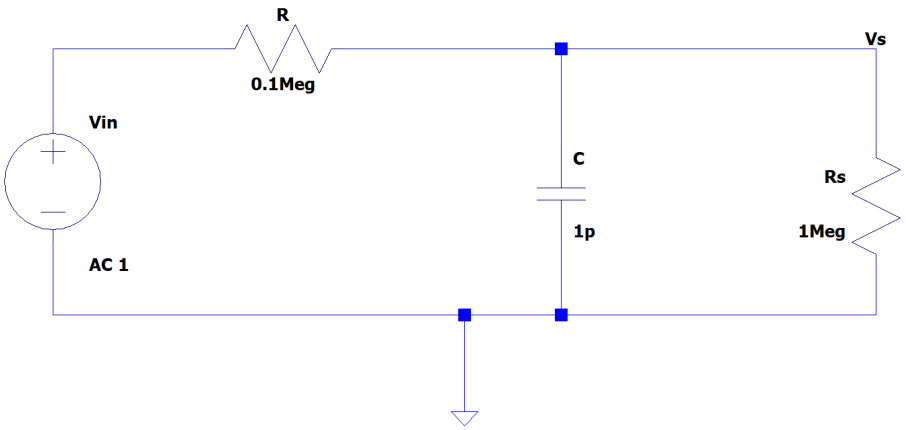


Figure 6.9: Equivalent schematic of the low pass filter

$$Z_{eq} = \frac{R_s}{1 + jR_sC\omega}$$

$$V_s = \frac{Z_{eq}}{R + Z_{eq}} = \frac{R_s}{R_s + R + jRR_sC\omega}V_{in} = \frac{R_s}{R_s + R}\frac{1}{1 + j\frac{RR_s}{R+Rs}C\omega}$$

Whereas with an infinite second stage input impedance, we would have $V_s = \frac{1}{1+jRC\omega}V_{in}$.

Thus, the added input resistance alters the filter bandwidth :

$$\tau = \frac{RR_s}{R + Rs}C, R = \frac{1}{G_m} \approx 0.1M\Omega, Rs = 1M\Omega$$

Thus, the output impedance of the oscilloscope can disturb the filter (error of about 9% on its time constant).

Therefore, $\tau \approx \frac{90.9}{100}RC$

To conclude, we have designed a low pass filter as proposed by Pr. Hasler in her papers using an OTA in the follower configuration and a parasitic load capacitance of the chip. We can set the cutoff frequency with bias current. The flaw of this filter is that the capacitance is a parasitic one and is not constant and depends on multiple parameters. This can be limiting since there is no reproducibility of the neuron through the integrated circuit.

# Chapter 7

# Experimentations on our virtual reservoir computing network

We did two experiments, one with a FGOTA (floating gate operational transconductance amplifier) neuron and one with an OTA neuron.

## 7.1 FG-OTA neuron

### 7.1.1 Circuit description

As explained before, we designed a neuron. This neuron is composed, as shown on the FIGURE 7.1, of a FGOTA used as a comparator, a first OTA used as a filter and a second OTA used as a buffer.

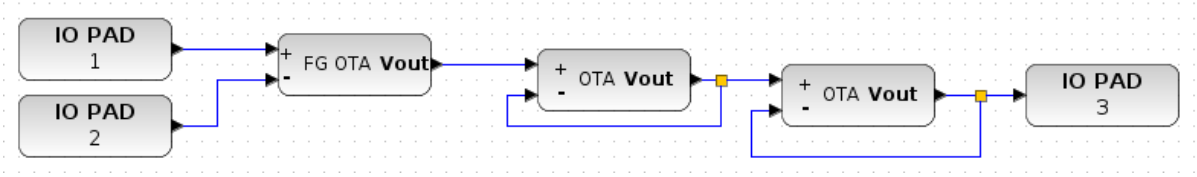


Figure 7.1: Schematic of an FG-OTA neuron

The comparator gets the input signal through IO pad 1 and a constant voltage value is applied on IO pad 2. The constant voltage value is 1.45V. The value must be adjusted at the last moment so that the output signal uses the full voltage range and that we have a desired non-linearity. The input signal is centered around 1.25V. The FGOTA is biased thanks to a current of 10nA and has a 1.6V of offset on each of its input. An order of magnitude of the response frequency is evaluated by programming the board with a neuron without low pass filter. We send a square signal to the circuit as shown on the FIGURE 7.3. We can see on the FIGURE 7.4 that if the neuron is not properly

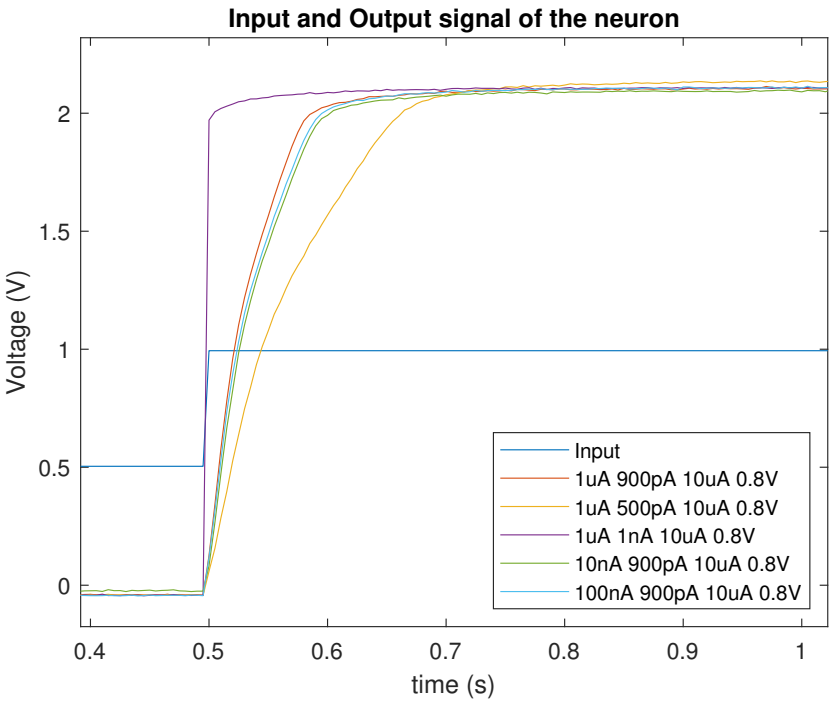


Figure 7.2: Response of the neuron to a step input

set up, it reaches a stationary state. We can see on the FIGURE 7.4 that the rise time is different from the fall time. Due to the capacity at the input of the FGOTA, we estimate experimentally the cutoff frequency at around a hundred Hertz.

The OTA filter is equivalent to a low pass filter, biased with a 900pA current. The value of cut-off frequency may vary due to our lack of board calibration. Her need a 100Hz cut-off frequency order of magnitude.

Then, we have a buffer. This buffer is an OTA in follower mode with a $10\mu$A bias current. This buffer allows the circuit not to be disrupted by the scope.

### 7.1.2 Experiment set up

Once our neuron programmed on the board, we tested it to set up the input signal characteristics. Thanks to a slot square signal, the neuron response time can be evaluated. The frequency of this signal is approximately 30Hz. As explained above, this frequency is multiplied by five to obtain the sampling frequency of the masked input signal.

The wave generator and the scope are included inside the Diligent Analog Discovery II. They can be commanded thanks to a Java script. The scope can acquire 8192 or 1 million samples at a time. Nevertheless, the generator has some troubles generating a high frequency arbitrary signal of long duration. As the frequency is set by the neuron, we choose to split the 5000 symbols into multiples smaller signals and thus have a smooth

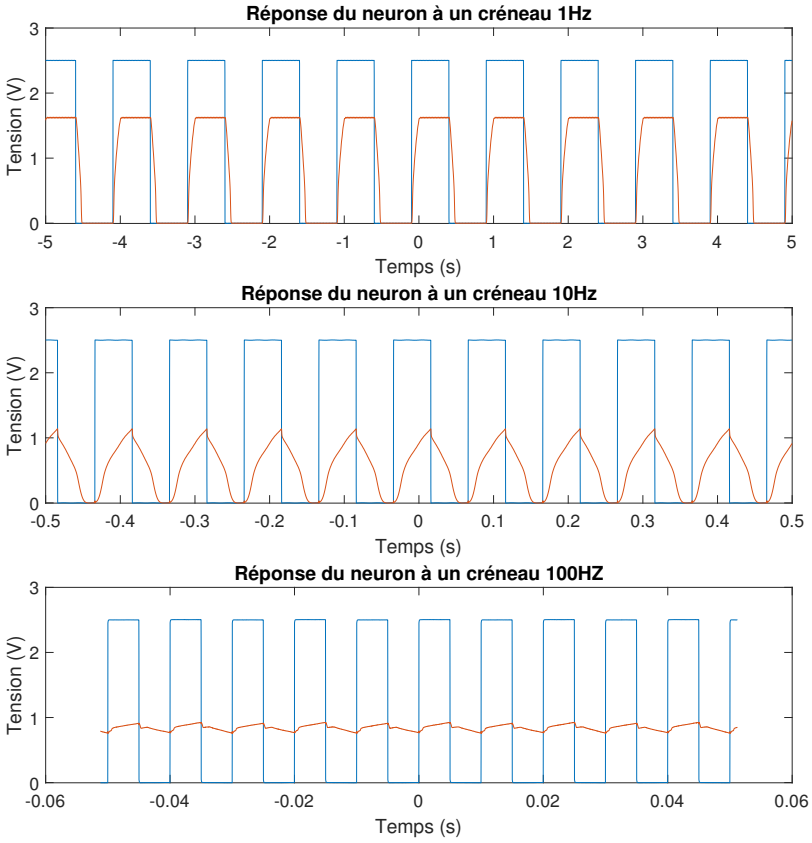


Figure 7.3: Signal input and output after an FG-OTA neuron at different sampling frequencies

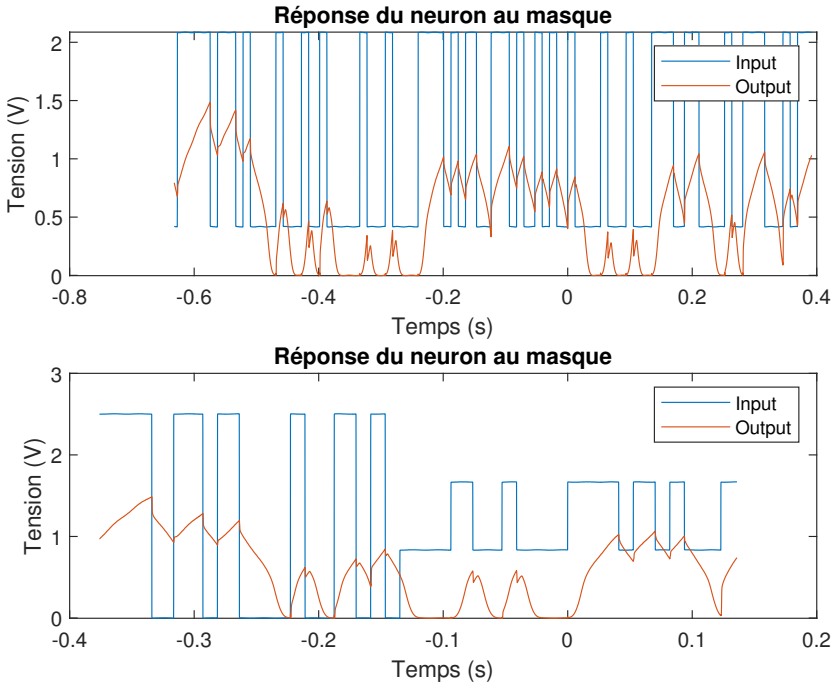


Figure 7.4: Signal input and output after a non-properly set-up FGOTA neuron

operation of the scope.

We have written a Java script that sets up the wave generator and the scope to perform the measures. The wave generator repeats a set of 15 Wi-Fi noisy masked symbols. A 16th synchronization blank symbol is added. This signal is at the sampling frequency of 600Hz. An offset and its amplitude are adjusted. We use small amplitude input shifted signals to ensure that this signal is inside the nonlinear part of the hyperbolic tangent FGOTA curve. On the input of the FGOTA there is a 0.1 gain in voltage due to a divider bridge made with capacitors. That allows to have a wider voltage input range than an OTA-based neuron. Thus, more state can be encoded in a symbol with a fix wave generator resolution of 1mV. With an amplitude modulation, we can see the linear (FIGURE 7.5) and non-linear ranges (FIGURE 7.6).

Then an oscilloscope measures the neuron output and input. The oscilloscope is set to 1200Hz sampling frequency. Thus, two periods of 16 symbols each are measured. We will necessarily have a complete period of 16 consecutive symbols that can be synchronized in post processing. We have made this choice because it often happens that we have a delay, when the scope is launched. This means that the first symbols displayed on the scope do not correspond to the first symbols of the sequence or are truncated. To have an automatic synchronization, we chose this solution.

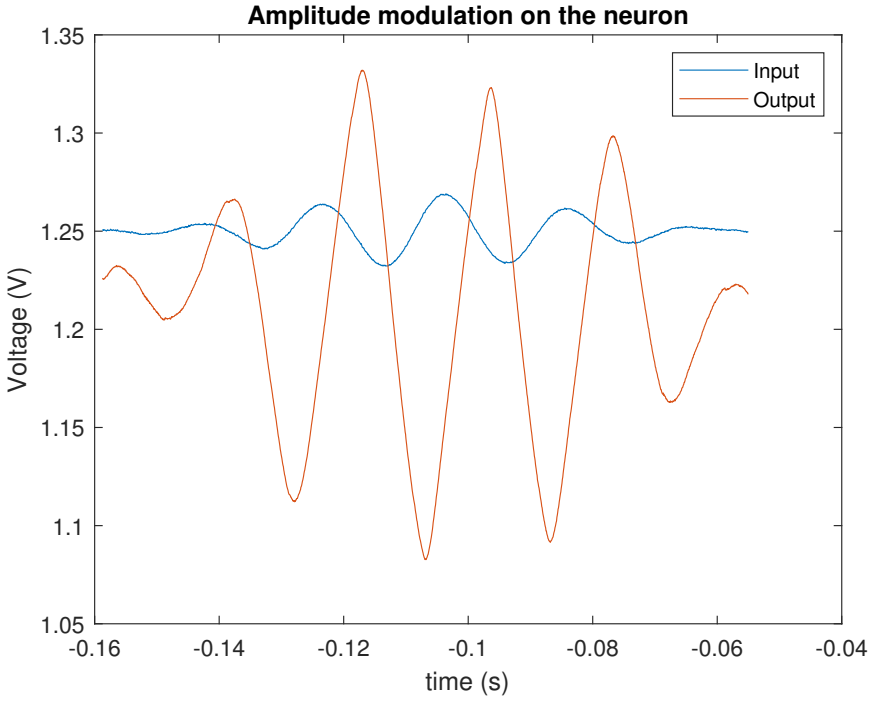


Figure 7.5: Amplitude modulation input signal and linear output signal

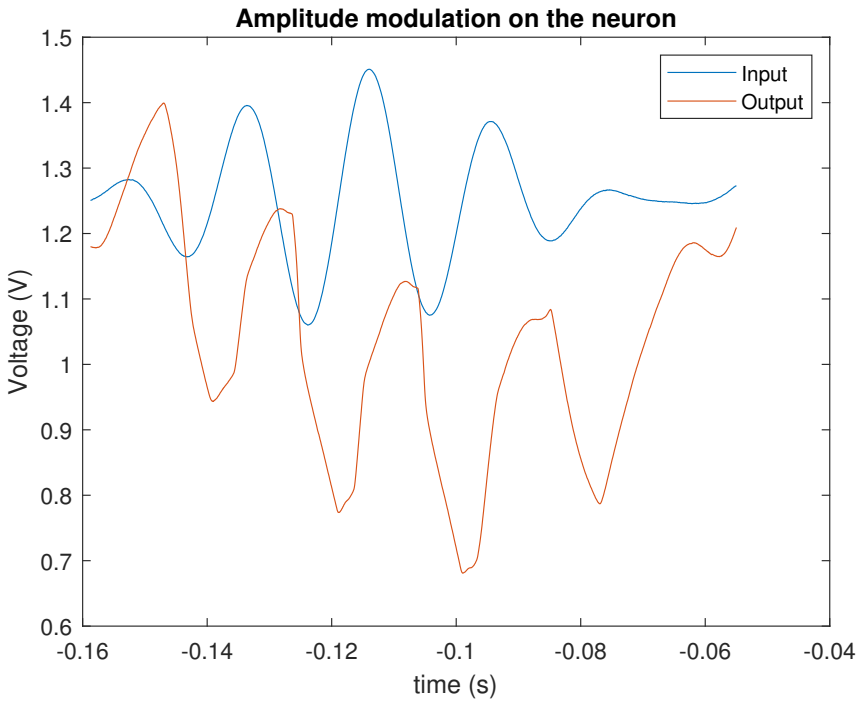


Figure 7.6: Amplitude modulation signal and non-linear output signal

### 7.1.3 Raw results

We launched the experiment. It took two hours and a half to compute the 5000 symbols (the neuron is working at 150Hz).

Once the split signals are gathered, we have a signal like shown on the FIGURE 7.7.

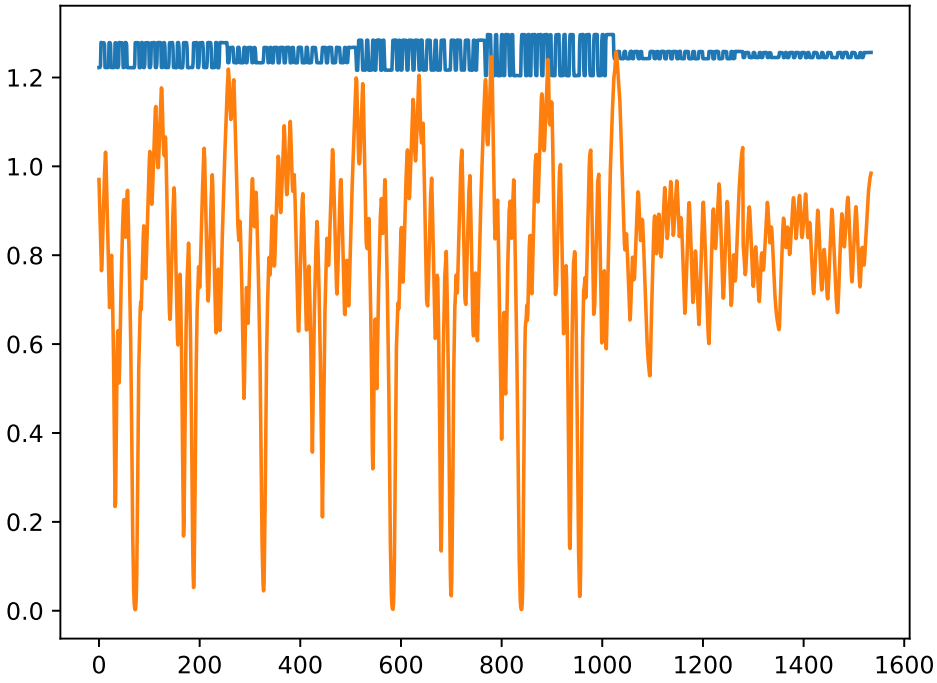


Figure 7.7: Signal input and output after an FG-OTA neuron

All the scripts are available in the appendix chapter at the end of this report.

## 7.2 OTA neuron

### 7.2.1 Circuit description

We did the same experiment on an OTA-based neuron. The non-linearity is not the same, but there is no capacity on the differential pair. The FIGURE 7.8 shows an OTA-based neuron.

The comparator is an OTA. This OTA has a 10nA bias current. The IOpad 2 voltage is 1.248V. The higher the working frequency, the more accurate the constant voltage value must be. Furthermore, the lower the bias current of the comparator is, the lower the working frequency will be. The low pass filter and the buffer have a 10$\mu$A bias current.

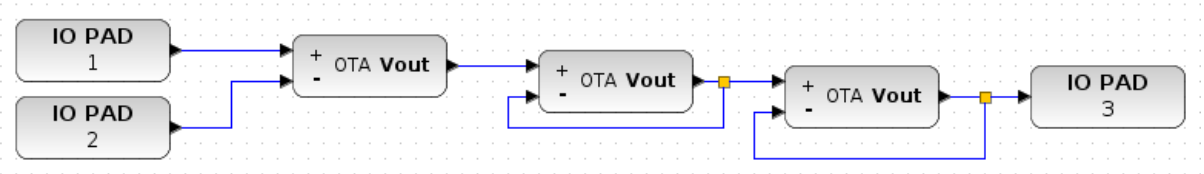


Figure 7.8: Schematic of an OTA neuron

### 7.2.2 Experiment set up

We measured a cutoff frequency without low pass filter of a thousand Hz. We chose a sampling frequency of the input signal of 5kHz (five times the cutoff frequency of the neuron to always be in transient regime). We did the same experiment as described in the previous section.

### 7.2.3 Raw results

The FIGURE 7.9 shows the raw experiments results.

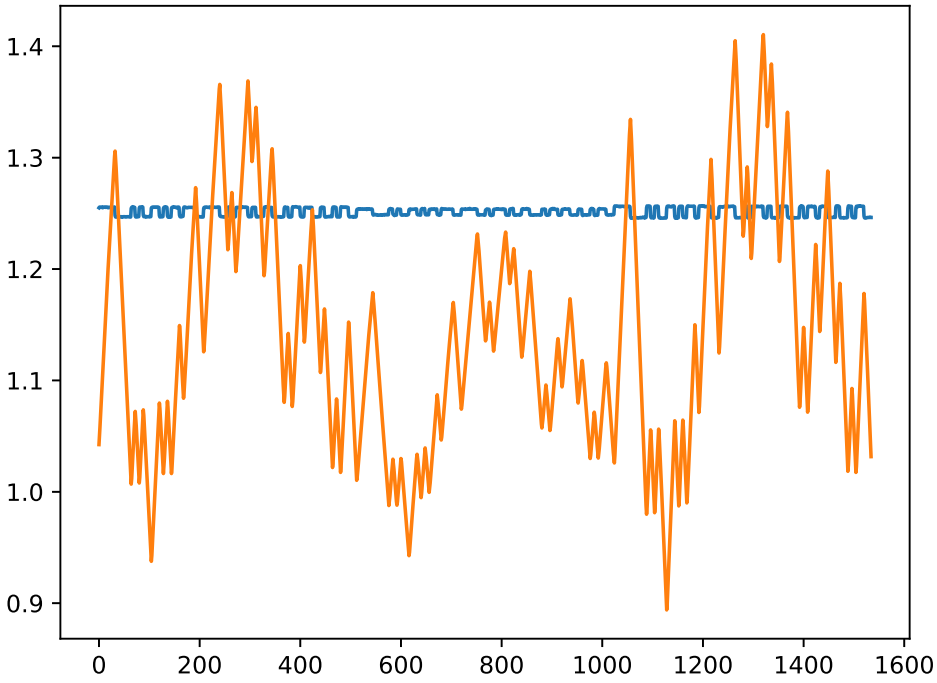


Figure 7.9: Signal input and output after an OTA neuron

# Chapter 8

# Training and testing our virtual reservoir computing network

The last stage of these experiments after passing the 5000 noisy Wi-Fi symbols through our virtual neural reservoir on the board will be process these symbols in order to train the network and test it after.

## 8.1 Simple threshold comparator

As the goal is to beat a simple threshold comparator, the FIGURE 8.1 illustrates the performances of such a comparator. Thresholds are set to the maximum available, i-e a range of one volt around each symbol value.

Simple threshold comparator results are shown on the FIGURE 8.1 and on the TABLE 8.1. This solution has a success rate of 89.3%.

| Minimum | Mean | Maximum | Standard deviation |
|---|---|---|---|
| 0.0011 | 0.5293 | 1.8624 | 0.3475 |

Table 8.1: Absolute error amplitude of the simple threshold comparator

## 8.2 FGOTA-based reservoir results

The training is based on a least square method. The residual error is 753.6981. For each bit of the mask, the scope has acquired 4 samples. Training has a success rate of 96.025%. Testing has a success rate of 93.3669%.

Results are shown on FIGURES 8.2 and 8.3 and detailed in the TABLE 8.2.

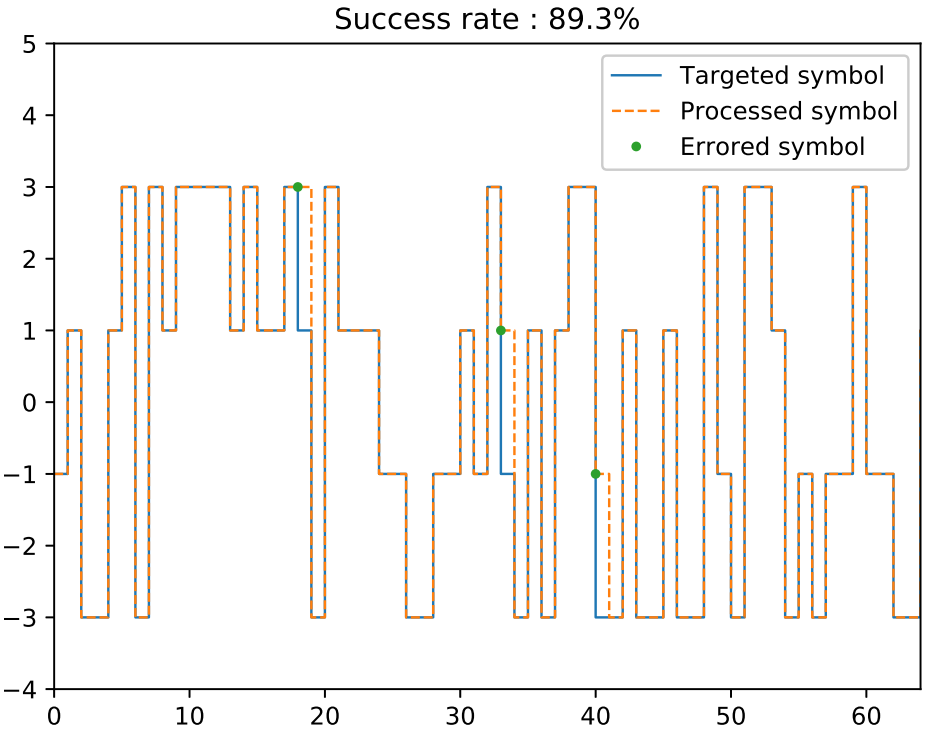


Figure 8.1: Simple threshold comparator results

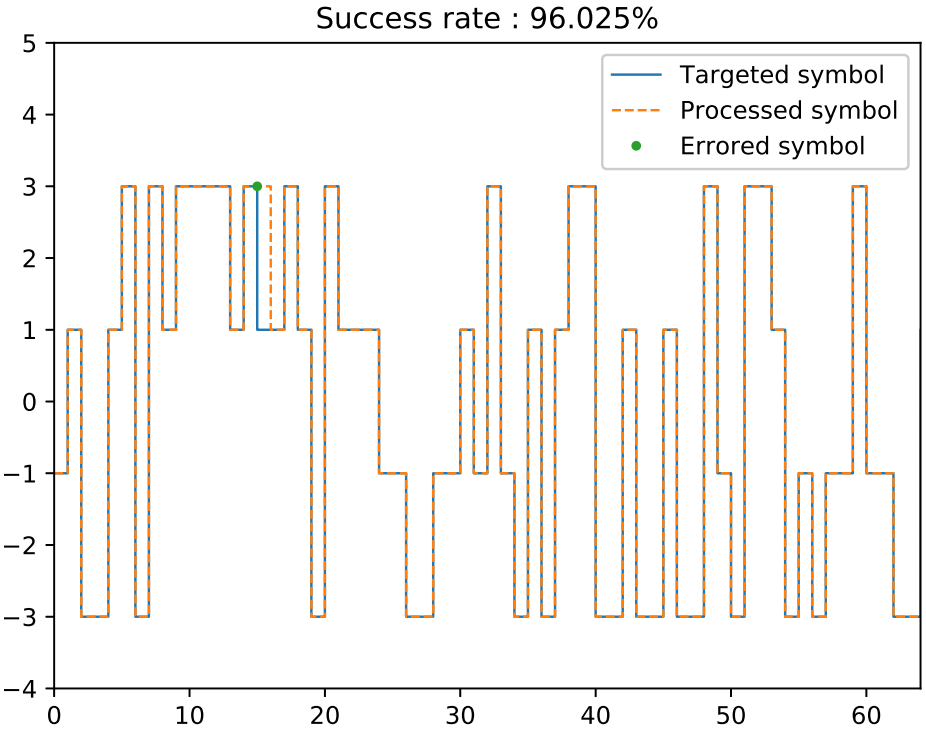


Figure 8.2: FG-OTA-based reservoir training results

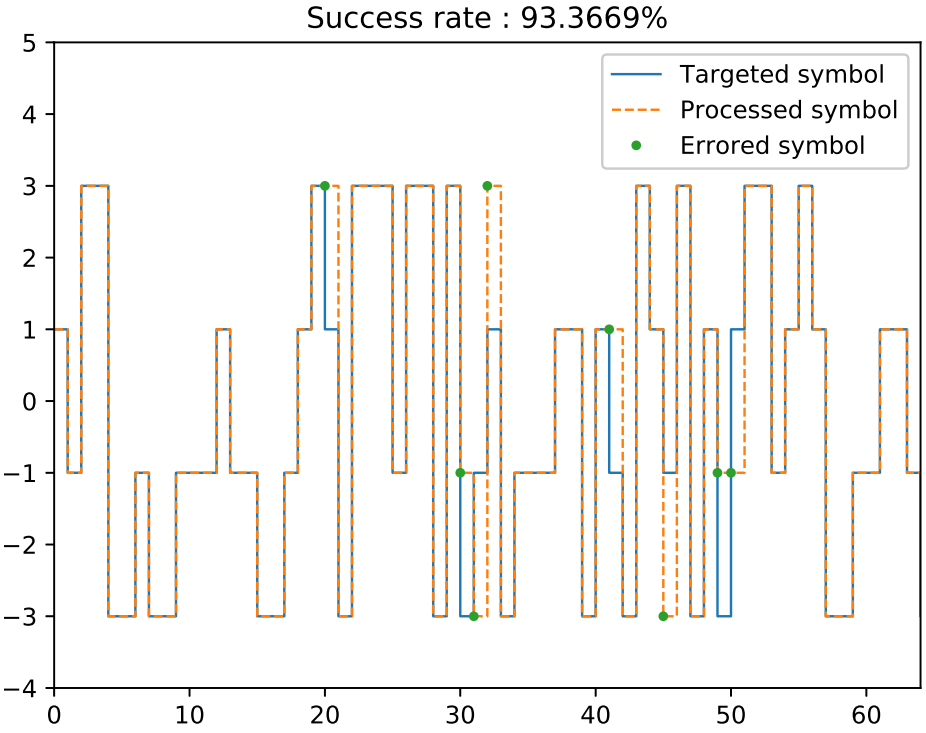


Figure 8.3: FG-OTA-based reservoir testing results

| | Minimum | Mean | Maximum | Standard deviation |
|---|---|---|---|---|
| Training | 0.0002 | 0.3193 | 2.1411 | 0.2941 |
| Testing | 0.0011 | 0.3675 | 7.4673 | 0.4979 |

Table 8.2: Absolute error amplitude of the FG-OTA reservoir computing network

In order to visualize the energy going through the neuron as a function of frequency for each symbol, we can plot the spectrogram of the time acquisition. With the spectrogram, we can observe that we gain energy at the output of the neuron in comparison with the input. Furthermore, we see that the energy of each output window equivalent to the length of a symbol are distinct. That leads us to assuming that there is a correlation between the energy of each symbol window and the ability to detect the expected result.

FIGURE 8.4 shows the spectrogram calculated for both the input signal and output signal. We can see that the output signal (upper subplot) is more energetic and thus maybe easier to process by the learning algorithm.

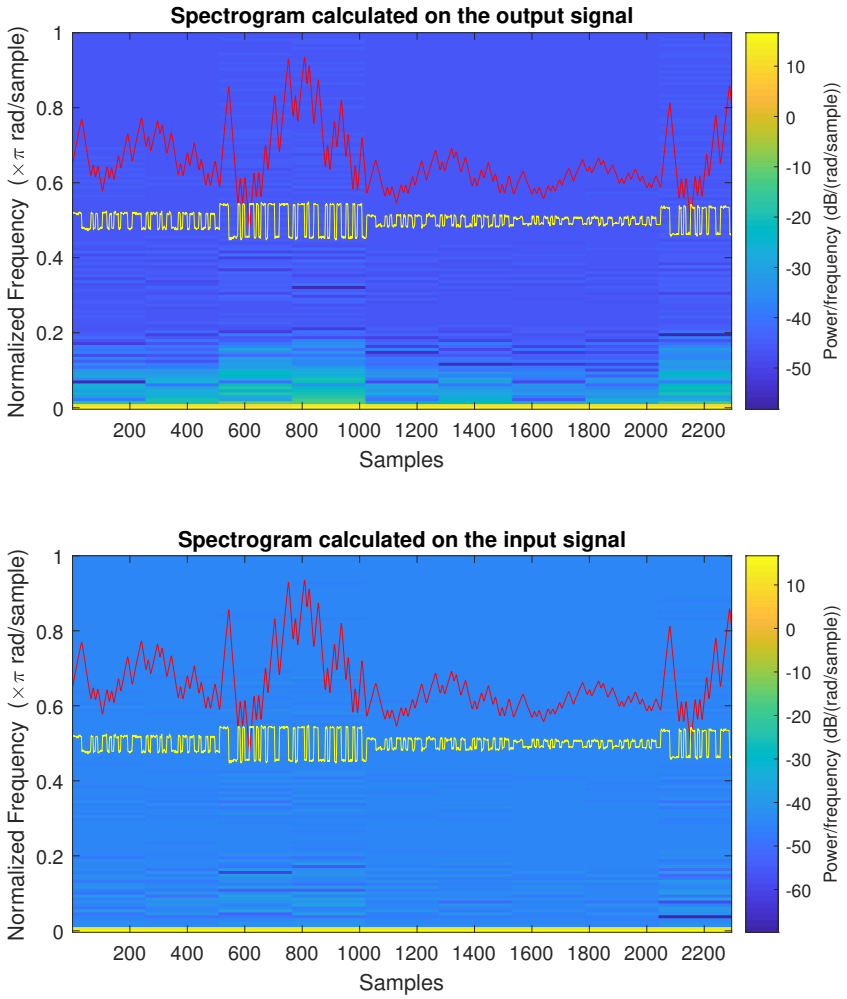


Figure 8.4: Spectrogram of input and output of the FGOTA neuron

Each window of the Fourier transform is of the length of a symbol modulated by the mask.

## 8.3 OTA-based reservoir results

The training is based on a least square method. For each bit of the mask, the scope has acquired 8 samples. The residual error is 1158.4298. Training has a success rate of 94.525%. Testing has a success rate of 91.3828%.

Results are shown on FIGURES 8.5 and 8.6 and detailed in the TABLE 8.3.

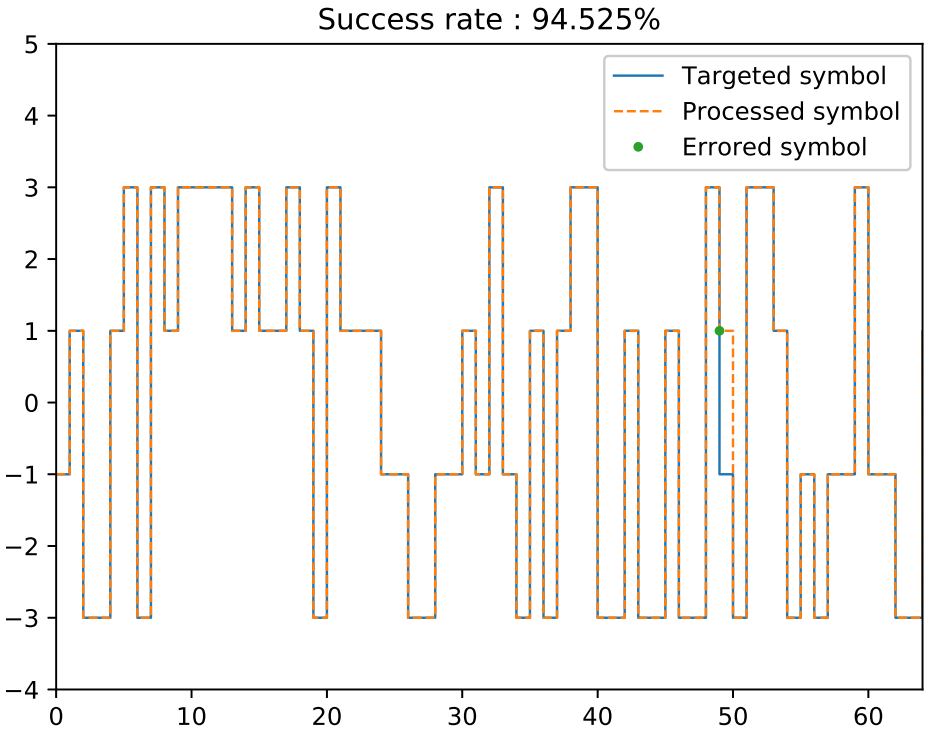


Figure 8.5: OTA-based reservoir training results

| | Minimum | Mean | Maximum | Standard deviation |
|---|---|---|---|---|
| Training | 0.0002 | 0.4120 | 5.9733 | 0.3463 |
| Testing | 7e-05 | 0.4717 | 2.3959 | 0.3696 |

Table 8.3: Absolute error amplitude of the OTA reservoir computing network

All these results may have an incertitude. But we were not able to calculate it.

Once again, we can plot the spectrogram of each symbol window and compare the energy of the input and output. We have the same analysis as before : the output of the neuron is more energetic, allowing an easier separation of symbols for the learning algorithm.

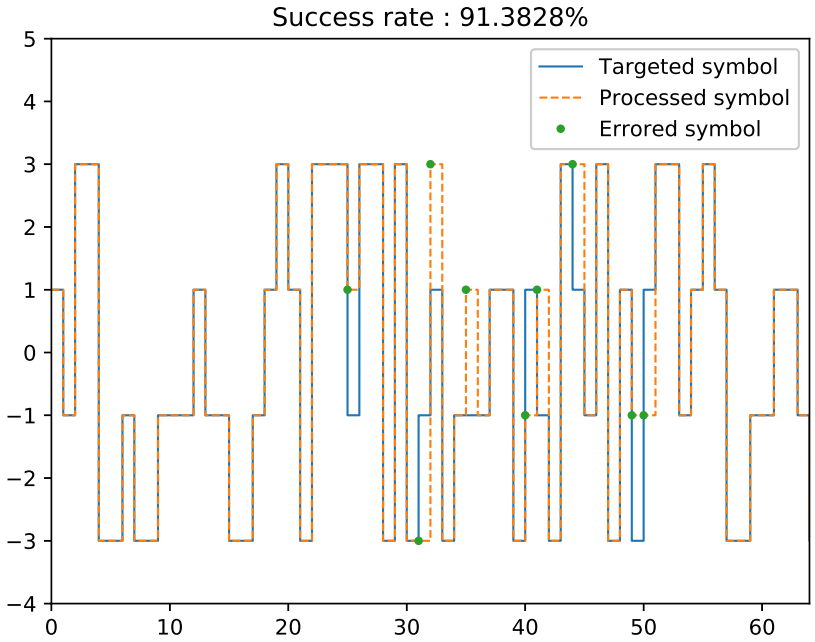


Figure 8.6: OTA-based reservoir testing results

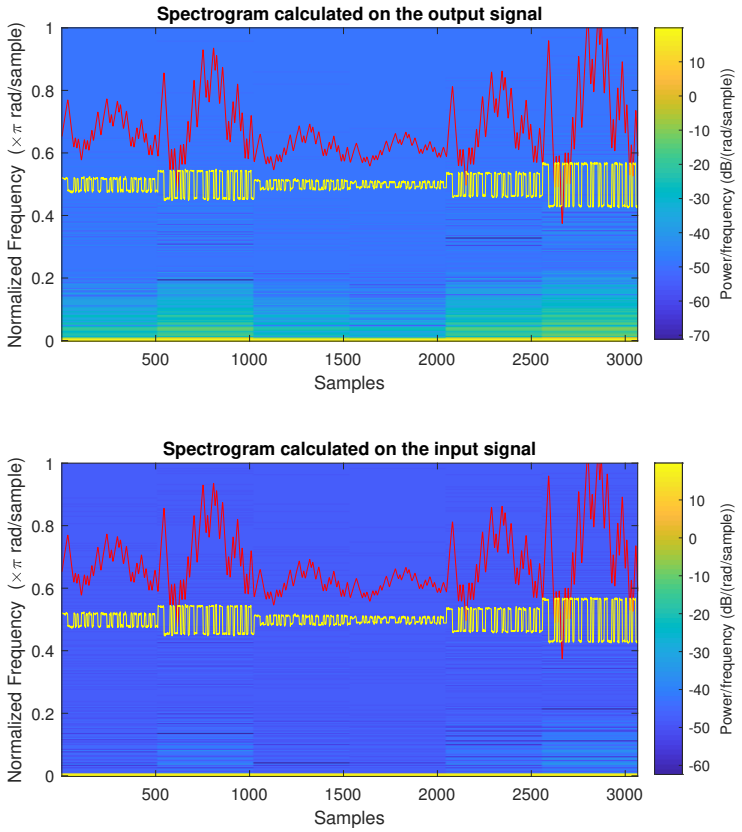


Figure 8.7: Spectrogram of input and output of the OTA neuron

# Conclusion

## Wi-Fi channel equalization task results sum up

To sum up, this single node, delay-coupled virtual reservoir computing neural network with no feedback gets better results than a simple threshold comparator. We thus have been able to create a working neural network with analogue components, power-efficient but not high frequency. The FPAA concept has proven very interesting since it allows us to test many different neurons and neural networks without having to print circuits or having to do any manipulations. Still, these results are a bit disappointing for us, in terms of success rate, rapidity and consistency. Maybe we could expect to have better success rates with a better training method. Some ideas to pursue this project are listed below.

## Project follow-up ideas

- Use a better machine learning algorithm than a least square method;
- Perform a real energy consumption measurement to compare on-board network and computer-emulated networks;
- Have access to the calibration devices in order to get more consistent results;
- Incorporate, in the integrated circuit a zero-holder, a multiplier and a summing node, to perform the calculation of the output layer on chip and thus lower the power consumption;
- Take advantage of the Field Programmable architecture to design more than a unique neural node;
- After calibration, it will be interesting to redo the experiments on low pass filters in order to rise the frequency rate of the neural network;
- Some resonating filters can be achieved by using OTAs in the feedback, this can be useful to add more non-linearity to the node;

- We want to study more thoroughly the correlation between the spectrogram of the output of the neuron and the efficiency of the neuron;
- It will be interesting to test plenty of different non-linearity (FGOTA and OTA) to see what makes a good electronic implemented neuron.

# Appendix A

# Codes

Inside the present appendix chapter, we will give you the main code files in order to reproduce our experiments.

## A.1 Arbitrary signal generator

```
#!/usr/bin/python3

import numpy as np
import itertools
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

dataCsv = open("WIFI_input.csv")



sampleRateWavegen = 5000#Hz
sampleRateScope = 80000#Hz
nNeurons = 64
syncMean = 0
syncDuration = 1#symbol for sync
nSymbolesPerFile=int(8192/sampleRateScope*sampleRateWavegen/nNeurons)-
    ↪ syncDuration
print(nSymbolesPerFile)
ampliMin=0.05
ampliMax=0.5
```

```
#generating a good mask
mask = np.random.choice([-1,1],size=nNeurons)
groupMask=[(x[0], len(list(x[1]))) for x in itertools.groupby(mask)]
while(max(groupMask, key=lambda x:x[1])[1]>4 or groupMask[0][0]==
    ↪ groupMask[-1][0]):
    mask = np.random.choice([-1,1],size=nNeurons)
    groupMask=[(x[0], len(list(x[1]))) for x in itertools.groupby(mask)]
print(hex(int("".join(str(int(0.5*(e+1))) for e in mask), 2)))

#data collecting
#data = [item for sublist in (list(itertools.product([1,2,3], repeat=3))
    ↪ ) for item in sublist]
data=[]
line=dataCsv.readline()
while line != "":
    data.append(float(line))
    line=dataCsv.readline()
dataN = [(ampliMax-ampliMin)*((float(i)-min(data))/(max(data)-min(data)))
    ↪ +ampliMin for i in data]#normalize
dataMasked = []
for value in dataN:
    dataMasked.append(list(value*mask))

#create csv
filenameBase = "export/Mask_"+hex(int(''.join(str(int(0.5*(e+1))) for e
    ↪ in mask), 2))+"_Symb_"
fileCount = 0
count=0
csv = open(filenameBase+str(fileCount)+".csv", "a")
for i in range(syncDuration*nNeurons):
    csv.write(str(syncMean)+"\n")
for listValue in dataMasked:
    if count == nSymbolesPerFile :
        fileCount += 1
        count = 0
        csv.close()
        csv = open(filenameBase+str(fileCount)+".csv", "a")
```

```
        for i in range(int(syncDuration*nNeurons)):
            csv.write(str(syncMean)+"\n")
    for value in listValue:
        csv.write(str(value)+'\n')
    count += 1

#zero order hold
timeVal=0
dataMaskedFlat = [item for sublist in dataMasked for item in sublist]#
    ↪ flatten
dataMaskedZOH=[[timeVal], [dataMaskedFlat[0]]]
for value in dataMaskedFlat[1:]:
    timeVal+=1/sampleRateScope
    if value != dataMaskedZOH[1][-1]:
        dataMaskedZOH[0].append(timeVal)
        dataMaskedZOH[1].append(dataMaskedZOH[1][-1])
    dataMaskedZOH[0].append(timeVal)
    dataMaskedZOH[1].append(value)

#plt.figure()
#plt.plot(list(dataMaskedZOH[0][0:nNeurons*nSymbolesPerFile]), list(
    ↪ dataMaskedZOH[1][0:nNeurons*nSymbolesPerFile]))
#plt.show()
```

## A.2 Digilent Analog Discovery II script

```
function doScope()
{
    print("Running␣Scope␣script");

    Wavegen1.Channel1.Mode.text = "Simple";
    Wavegen1.Channel1.Simple.Type.text = "DC";
    Wavegen1.Channel1.Simple.Offset.value = 1.25;
    Wavegen1.run();
```

```
print("!␣DEBRANCHER␣ALIMENTATION␣PC␣!\n␣\n!␣DEBRANCHER␣ALIMENTATION␣
    ↪ PC␣!");
wait(1);
Wavegen1.stop();

Wavegen1.Channel2.Mode.text = "Simple";
Wavegen1.Channel2.Simple.Type.text = "DC";
Wavegen1.Channel2.Simple.Offset.value = 1.248;

Scope1.Time.Samples.text = "Default";
Scope1.Time.Rate.value = 4000;

for (i=0; i<334; i++)
{

var WIFI = File("G:/Documents/2A/Projet␣Long/calculneurofpaa/captures
    ↪ /GenScriptBenjHugo/export/Mask_0x92152145373294f6_Symb_"+i+".
    ↪ csv").readArray();

Wavegen1.Channel1.Mode.text = "Custom"
Wavegen1.Custom.set("ok", WIFI);
Wavegen1.Channel1.Custom.SampleRate.value = 1000;
Wavegen1.Channel1.Custom.Amplitude.value = 0.05;
Wavegen1.Channel1.Custom.Offset.value = 1.25;

Scope1.single();
Wavegen1.run();
Scope1.wait();
Wavegen1.stop();


Scope1.Export("G:/Documents/2A/Releve/
    ↪ Mask_0x92152145373294f6__releve__"+i+".csv", "Oscilloscope",
    ↪ false, false, true);

print(i);
}
```

```
    print("Done␣!");

}
doScope();
```

## A.3 Results gathering

```
close all;
clear all;
clc;

Sequence = [];
p = 400;
epsilon = 1e-3;

M = csvread("Mask_0xf0a6b73cba62d92c__releve__"+1+".csv");
figure()
plot(M(:,2));


for k=0:1:713

    M = csvread("Mask_0xf0a6b73cba62d92c__releve__"+k+".csv");
    Intervalle = M(1:p+1,2);
    i = p+1;

    while ((max(Intervalle) - mean(Intervalle)) > epsilon) || ((min(
       ↪ Intervalle) - mean(Intervalle)) < - epsilon)
        i = i+1;
        Intervalle = M(i-p:i,2);
    end

    while ((max(Intervalle) - mean(Intervalle)) < epsilon) && ((min(
       ↪ Intervalle) - mean(Intervalle)) > - epsilon)
        i = i+1;
```

```
        Intervalle = M(i-p:i,2);
    end

    debut = i;
    fin = debut + 64*8*7-1;
    Sequence(:,:,k+1) = [M(debut:fin,2) M(debut:fin,3)];

end

disp("-----------------------------------------------");
disp("fin traitement, debut applatissement");

SequencePlat = zeros(length(Sequence(1,1,:))*length(Sequence(:,1,1)), 2);

i = 1;
for k=1:1:length(Sequence(1,1,:))
    SequencePlat(i:i+length(Sequence(:,1,1))-1,1) = Sequence(:,1,k);
    SequencePlat(i:i+length(Sequence(:,1,1))-1,2) = Sequence(:,2,k);
    i = i+length(Sequence(:,1,1));
end

disp("-----------------------------------------------");
disp("fin traitement applatissement");

SequencePlatTrainning = SequencePlat(1:4000*64*8,:);
SequencePlatTesting = SequencePlat(4000*64*8+1:end,:);

% csvwrite("Training.csv", SequencePlatTrainning);
% csvwrite("Testing.csv", SequencePlatTesting);

figure()
plot(Sequence(:,1,5));
hold on;
%plot(Sequence(:,1));
```

## A.4 Learning

```
#!/usr/bin/python3

import numpy as np
import matplotlib.pyplot as plt

def read_csv(filenameCsv, indexData):
    csv = open(filenameCsv)
    data = []
    line = csv.readline()
    while line != "":
        line=line.replace("\n", "").split(",")
        data.append(float(line[indexData]))
        line = csv.readline()
    return data

def form_dataSymbol(data):
    dataSymbol=[]
    global bitPerSymbol
    global samplePerBit
    for i in range(int(len(data)/(bitPerSymbol*samplePerBit))):
        dataOneSymbol=[]
        for j in range(bitPerSymbol):
            for k in range(samplePerBit):
                dataOneSymbol.append(data[bitPerSymbol*samplePerBit*i+j*
                    ↪ samplePerBit+k])
        dataOneSymbol.append(1)#last value for constant removing during
            ↪ training
        dataSymbol.append(dataOneSymbol)
    return np.array(dataSymbol)

def zero_order_hold_plot(data, style):
    timeVal=0
    dataZOH=[[timeVal], [data[0]]]
    for value in data[1:]:
        timeVal+=1
```

```
        if value != dataZOH[1][-1]:
            dataZOH[0].append(timeVal)
            dataZOH[1].append(dataZOH[1][-1])
        dataZOH[0].append(timeVal)
        dataZOH[1].append(value)
    return plt.plot(dataZOH[0], dataZOH[1], style, linewidth=1)


def print_results(dataTarget, dataSymbol, coeff, name):
    dataRes=[]
    numberErrors = 0
    threshold=1
    dataError=[]
    for i in range(len(dataSymbol)):
        if isinstance(dataSymbol[i], np.float64) or isinstance(dataSymbol
            ↪ [i], np.int64):
            value=dataSymbol[i]
        else:
            value=sum(dataSymbol[i]*coeff)
        dataRes.append([dataTarget[i], value])
        if dataTarget[i]+threshold<value or dataTarget[i]-threshold>value:
            numberErrors += 1
            dataError.append([i, value])
    dataRes=np.array(dataRes)
    dataError = np.array(dataError)
    graphT, = zero_order_hold_plot(dataRes[:, 0], "-")
    graphP, = zero_order_hold_plot(threshold_sig(dataRes[:, 1]), "--")
    graphE, = plt.plot(dataError[:, 0], threshold_sig(dataError[:, 1]), ".
        ↪ ")
    #graphE, = zero_order_hold_plot((dataRes[:, 0]-dataRes[:, 1]))
    plt.title("Success␣rate␣:␣" + str(100-0.0001*int(1000000*numberErrors
        ↪ /len(dataSymbol))) + "%")
    #plt.grid()
    plt.xlim(0, 64)
    plt.ylim(-4, 5)
    plt.legend([graphT, graphP, graphE], ["Targeted␣symbol", "Processed␣
        ↪ symbol", "Errored␣symbol"])
```

```
    plt.savefig(name, bbox_inches='tight')
    plt.show()
    #error rate
    print("Amplitude absolue max d'erreur : ", abs(max(dataRes[:, 0]-
        ↪ dataRes[:, 1], key=abs)), "Amplitude absolue mini d'erreur : ",
        ↪  abs(min(dataRes[:, 0]-dataRes[:, 1], key=abs)), "Amplitude
        ↪ moyenne absolue d'erreur : ", np.mean(abs(dataRes[:, 0]-dataRes
        ↪ [:, 1])), "Ecart type de l'erreur absolue", np.std(abs(dataRes
        ↪ [:, 0]-dataRes[:, 1])))


def threshold_sig(data):
    return np.array([2*int((x+4)/2)-3 for x in data])

#main
bitPerSymbol = 64
samplePerBit = 8

#data target
dataTarget = read_csv("WIFI_target.csv", 0)

#initial data
dataNoisy = np.array(read_csv("WIFI_input.csv", 0))
print_results(dataTarget, dataNoisy, 1, "initial.eps")

#training
dataSymbolTraining = form_dataSymbol(read_csv("Training.csv", 1))
dataTargetTraining=np.array(dataTarget[:4000])
resLstSq=np.linalg.lstsq(dataSymbolTraining, dataTargetTraining, rcond
    ↪ =-1)
coeff=resLstSq[0]
print("Erreur residuel", resLstSq[1])
print_results(dataTargetTraining, dataSymbolTraining, coeff, "
    ↪ training_OTA.eps")

#testing
dataSymbolTesting = form_dataSymbol(read_csv("Testing.csv", 1))
```

```
dataTargetTesting=np.array(dataTarget[len(dataSymbolTraining):len(
    ↪ dataSymbolTraining)+len(dataSymbolTesting)])
print_results(dataTargetTesting, dataSymbolTesting, coeff, "testing_OTA.
    ↪ eps")
```

# Bibliography


[1] Jennifer Hasler Aishwarya Natarajan. *Modeling, simulation and implementation of circuit elements in an open-source tool set on the FPAA*. Tech. rep. Analog Integrated Circuits and Signal Processing, Jan. 2017.

[2] S. George et al. *A Programmable and Configurable Mixed-Mode FPAA SoC*. Tech. rep. IEEE TRANSACTIONS ON VERY LARGE-SCALE INTEGRATION (VLSI) SYSTEMS.

[3] Eric A.Vittoz. *WEAK INVERSION INANALOG AND DIGITAL CIRCUITS*. 2003. URL: http://folk.uio.no/inf3410/docs/Vittoz_Weak_Inversion_In_Analog_And_Digital_Circuits.pdf.

[4] Shih-Chii Liu; Jorg Kramer; Giacomo Indiveri; Tobias Delbruck; Rodney Douglas. *Analog VLSI: Circuits and Principles*. The MIT Press, 2002. ISBN: 9780262122559.

[5] Pr. Jennifer Hasler. *Using CAB Components: Basic Transistor Circuits*. URL: http://hasler.ece.gatech.edu/Courses/ECE6435/Unit1/index.html.

[6] Xavier Hinaut. *Imperial College C395 Machine Learning*. URL: https://www.doc.ic.ac.uk/~nuric/teaching/imperial-college-c395-machine-learning-neural-networks.html.

[7] Xavier Hinaut. *Recurrent neural network for processing abstract sequences and grammatical structures, with an application to human-robot interactions, chapter 3*. URL: https://sites.google.com/site/xavierhinaut/phd-thesis.

[8] Jennifer Hasler Sahil Shah. *Tuning of Multiple Parameters With a BIST System*. Tech. rep. 7. The address of the publisher: IEEE TRANSACTIONS ON CIRCUITS and SYSTEMS–I, July 2017.

[9] Daniel Brunner Van der Sande Guy and Miguel C. Soriano. *Advances in photonic reservoir computing. Nanophotonics*. 2017. URL: https://www.degruyter.com/downloadpdf/j/nanoph.2017.6.issue-3/nanoph-2016-0132/nanoph-2016-0132.pdf.