PH567: Non-Linear Dynamics

Chaogates: Nonlinear dynamical computation devices

Aswin Suresh 20B030011

Paritosh Hegde 200100108

Piyush Mourya 200260037

Reet Mhaske 20D170032

Tushnim Yuvaraj 20B030037

Guide: **Prof Amitabha Nandi**

Abstract:
Non-linear Maps can be used to implement logic gates. By harnessing their sensitivity of initial conditions and parameters, we can encode the logic inputs into different initial conditions and use the map to arrive at the output. In this report, we have reproduced two ways of doing the same, first by varying the parameters and another by changing the number of iterations (from Ref.[1]). We demonstrate this using the logistic map. The map being difficult to construct physically, we have come up with an alternative tent map, which has an easier analog circuit implementation. We then introduce the implementation of an efficient non-linear map using MOSFETS, and reproduce results from Ref. [2].We also demonstrate a basic logistic map based chaogate on a microcontroller.

Introduction:

Chaotic systems are known to show sensitive dependence on initial conditions. This dependence can be used to create a variety of reconfigurable logic gates. Unlike regular Field Programmable Gate Arrays (FPGAs), which require the physical reconnection of logic elements, a chaotic gate, henceforth referred to as a **chaogate**, can achieve the same functionality by variation of dynamic parameters. These chaogates are known to be fragile in their construction and analog implementation. In principle, any non-linear dynamical map can be used to achieve this purpose, provided we are able to find the optimal parameters for their implementation. In this report, we review the possibilities of such maps, along with their circuit implementation.

Contents:

I. How to implement a chaogate?

Given a 1D non-linear map $x_{n+1} = f(x_n)$, we need to correspond its output to a 2-input logic gate. We need to map a particular input to a unique initial condition, on which $f(x)$ would operate, then set up a threshold on the output to classify it as logic 0 or 1. We follow a general set of procedures. (Note: x_{gate}, δ, x_c are tunable parameters, that determine which gate the map would correspond to.)

1. *Mapping inputs to a unique initial condition:* If I_1 and I_2 are the logic inputs (which can only be 1 or 0), we set the initial condition to be: $x_0 = x_{gate} + \delta(I_1 + I_2)$.
2. *Action of the gate:* Evolve the initial condition x_0 under f, for n iterations, so that $x \to f^n(x_0)$.
3. *Determining the output:* Decide on a threshold x^*, such that if $f^n(x_0) \leq x^*$, set the logic output to 0, and if $f^n(x_0) > x^*$, the logic output is 1

Just by changing these parameters, it is possible to ***make the same map behave like a different logic gate.*** We need not use only chaotic maps; simple non-linear dynamical systems may also suffice, but non-linear maps have more sensitive dependence on initial conditions making them ideal for use.

There are two ways in which we can implement the above change of parameters. We show their implementation using different maps.

II. Parameter dependance

To implement a different logic gate, we fix the number of iterations of the map n, and change x^* and x_{gate}. By this we are using the sensitive dependence on the initial conditions of the non-linear map, since x_{gate} decides the initial conditions, we use the map on. Now, we need the set parameters such that the map can be used to implement a universal set of logic gates. Say for example, if we want it to act as a AND gate, then on input (0,0) i.e., initial condition x_{AND}, the output should be 0, i.e., $f(x_{AND}) < x^*$, if the input is (1,1) i.e., initial condition $x_{AND} + 2\delta$, output should be 1 i.e. $f(x_{AND}) > x^*$ and so on. Thus, we have constraints for each of the gate parameters x_{gate}, depending on the Truth Table of that gate. There can be many solutions for these parameters for a given map. We consider the logistic map given by

$$x_{n+1} = r x_n (1 - x_n)$$

(For $r = 4$), from the orbit diagram and the Lyapunov exponents for the logistic map, we see that this corresponds to chaos. We need values of x_{gate} and x^* for a given δ that satisfy the conditions that come from the Truth Tables as mentioned above.

One such solution set is as given [1] (in pg. 2), for $\delta = 0.25$ is

Operation	AND	OR	XOR	NAND
x_{gate}	0	1/8	1/4	3/8
x^*	3/4	11/16	3/4	11/16

We verify this solution, by simulating the nature of the map in the parameter space x_{gate} and x^* in the plot below.

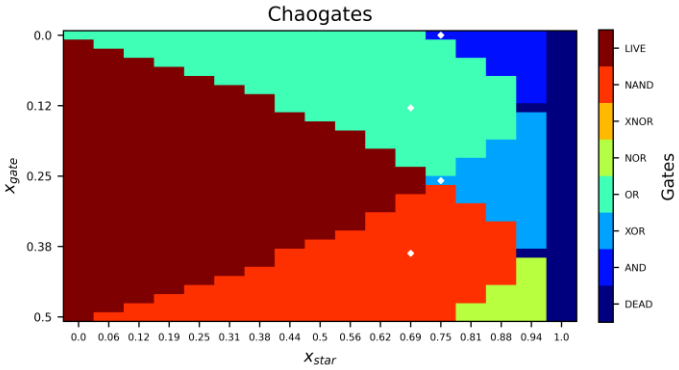


Fig. 2.1 Parametric implementation of Logic Gates: The white dots in the figure correspond to the points mentioned in the table above.

Thus, we see that the same map can be made to function like a different gate by changing its parameters. This is unlike conventional FPGAs, where we have to rewire connections between different components to achieve different functionality.

III. Iteration count dependance

Another way to use chaotic maps to achieve different is by keeping the parameters fixed but iterating the map for different number of times, which will depend on the logic gates one wants to achieve. Here, too we map the inputs to a particular initial condition, $x = x_{gate} + \delta(I_1 + I_2)$. We then evolve it different number of times. We compare the output against a reference x^*, to decide if the output is logic 0 or logic 1. Note, here x_{gate} , δ and x^* are independent of the gate we want to implement. For example, consider the logistic map again. We use $\delta = 0.25$, $x^* = 0.6$ and $x_{gate} = 0.325$. Let's start with input (0,0), i.e., initial condition $x = 0.325$, and evolve it for 5 iterations. In the first iteration, the output is above x^* , i.e., logic 1, after the second iteration it is below x^* (hence logic 0), and so on. Now, we repeat this for different logic inputs: find the initial condition, evolve up to 5 iterations, and see what logic output we obtain at each iteration. We plot the corresponding cobweb diagrams below. Now, the input-output relation of the first iteration always matches the Truth Table of a NAND gate! That of the second iteration matches that of a AND gate. We match these different iterations of different inputs to different Truth Tables and find what gate it corresponds to.

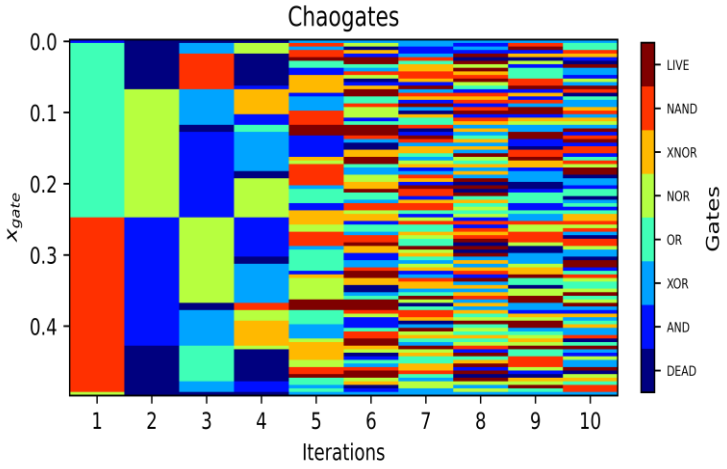


Fig.3.2 We identify which iteration corresponds to which gate by evaluating the outputs for all inputs at different values of x_{gate}

We can apply the above logic for different maps, to which iteration corresponds to which logic gate. Below, we reproduced a plot for the logistic map, with $x^* = 0.75$, for $n < 5$ and $x^* = 0.4$ for $n \geq 5$; $\delta = 0.25$ fixed. For example, for $x_{gate} = 0.4$, the first iteration is a NAND gate, second iteration is an AND gate, third one an XOR gate and so on.

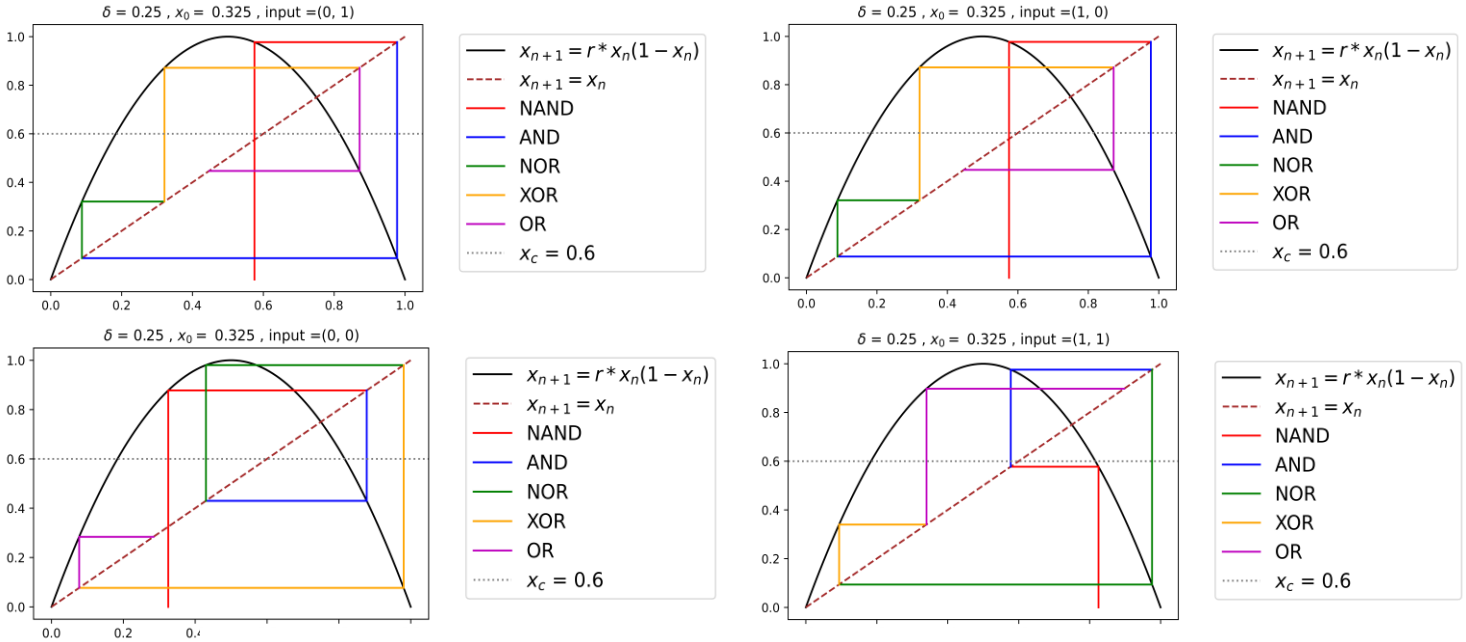


Fig. 3.1. Cobweb diagrams, showing the different iterations at different initial conditions

IV. Implementation using Tent Map

Consider the Tent Map given by:

$$x_{n+1} = r \times \min(x_n, 1 - x_n)$$

We plot the bifurcation diagram of this map, and find that for all $r > 1$, it exhibits chaos.

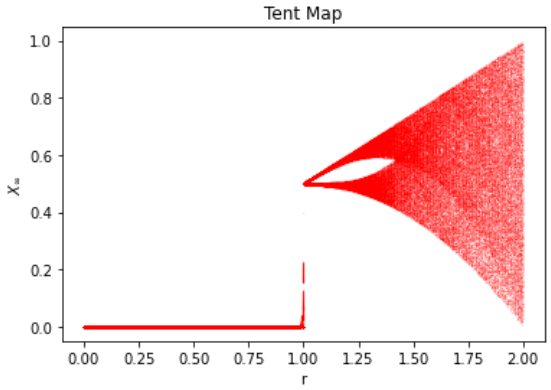


Fig 4.1 Orbit diagram of a tent map

We use $r = 1.75$, and plot the map, similar to Fig3.2, for $\delta = 0$. and identify which iteration corresponds to which gate. Tent Map shows similar diverse characteristics for the implementable gates, at the same time, having an easier analog circuit implementation than logistic map. We have designed one such circuit that implements the tent map.

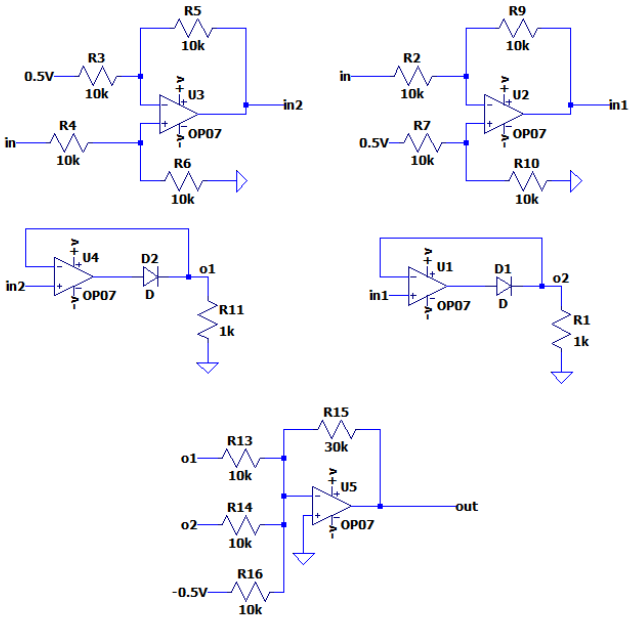


Fig 4.2: Analog circuit implementation of the tent map

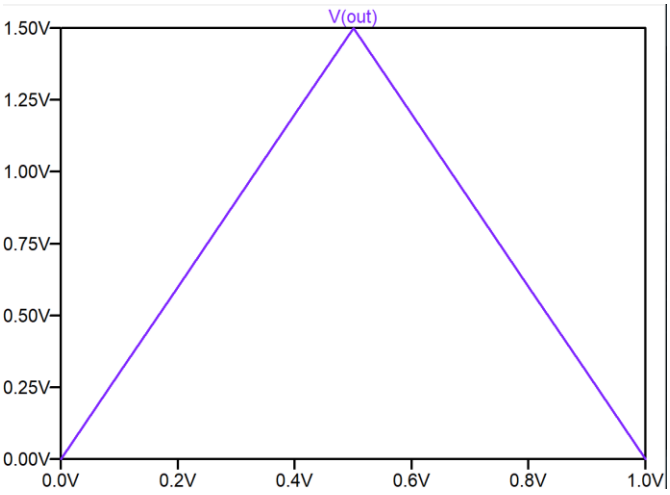


Fig 4.3: SPICE simulation of the above circuit, for $r = 1.5$.

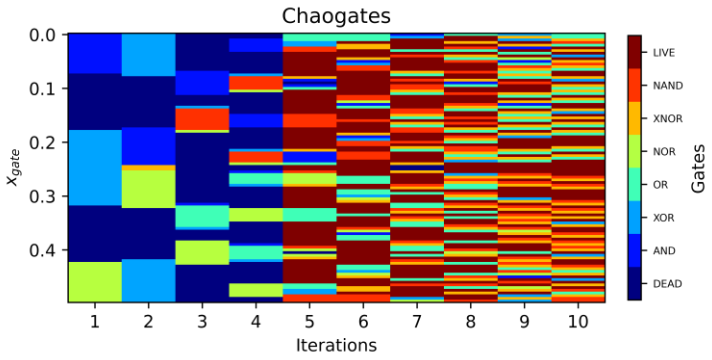


Fig 4.3: Gate identification for tent map

V. A CMOS Implementation

So far, we have discussed implementing chaogates using the logistic map and the tent map. The practical implementation of the tent map was rather complex, using several Operational Amplifiers, diodes and resistors. Modern foundry processes are, however, optimized for CMOS technologies. While it is possible to implement the aforementioned maps using CMOS, a more practical circuit is shown below:

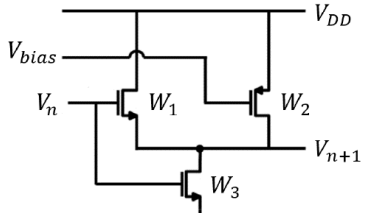


Fig 5.1: An efficient CMOS implementation of the non-linear map. Fig. taken Ref. [2]

The circuit parameters used are:

Parameters	Value
Bias Voltage	$V_{Bias} \in [0V, 1.2V]$
Power Supply	$V_{DD} \in [0.6V, 1.3V]$
MOSFET lengths	$L_1 = L_2 = L_3 = 65nm$
MOSFET width	$W_1 = W_2 = 120nm$ $W_3 = 2\mu m$
Temperature	$25°C$
Logic Threshold	$V_* = 0.6V$

Since the MOSFETs are in the sub-nanometer regime, the standard I-V (square law of MOSFETs) is not accurate. As a result, one has to turn to computer simulations to determine the transfer function $f(x)$ of the map. We shall be using NgSPICE. It is assumed that the various parameters remain constant during the operation of the circuit. The following transfer functions are obtained:

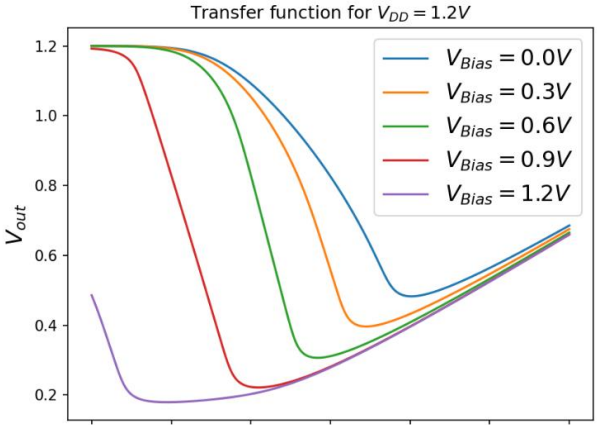


Fig 5.2: Transfer function for $V_{DD} = 1.2V$ and various bias points

VI. Choosing Bias Point

Now that we have the ability to determine the transfer function, we have to choose the bias point V_{Bias} and the supply voltage V_{DD}. Since we wish to use the map as a chaogate, it is necessary that the system is chaotic. Thus, we plot the Lyapunov exponent, given by:

$$\lambda = \lim_{n\to\infty} \frac{1}{N} \sum_{n=1}^{\infty} \ln \left| \frac{\partial f}{\partial V_n}(V_n) \right| \ldots (1)$$

It may also be calculated by:

$$\lambda = \lim_{n\to\infty} \ln \left| \frac{\delta(V_0, N)}{\delta(V_0, 0)} \right| \ldots (2)$$

in the parameter space. We obtain the following plot:

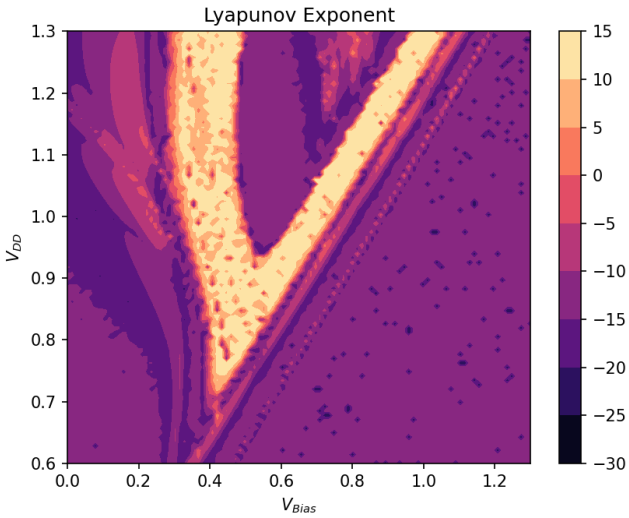


Fig 6.1: Lyapunov exponent over the parameter space, when calculated by definition (2)

The following plot is present in the paper:

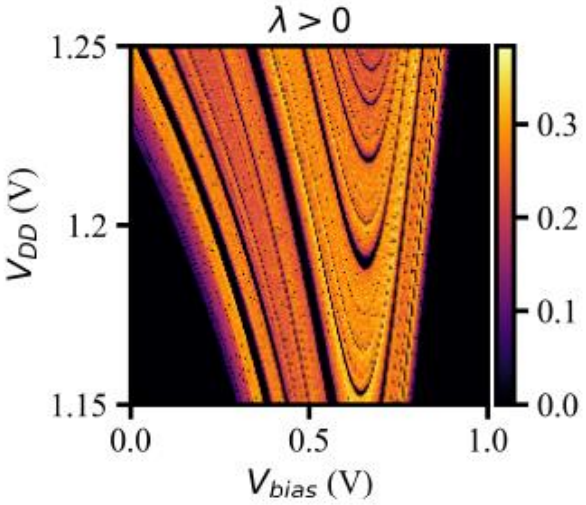


Fig 6.2: Lyapunov exponent, plotted over the parameter space. Figure taken from Ref. [2]

The plot obtained by our simulations and in the paper don't perfectly match. While we are not sure why this is the case, we believe that this might be because of the different methods for calculation of the Lyapunov constant. When we use definition (1) we get the Lyapunov constant to invariant w.r.t to

the system parameters, which is also not expected.

Notice from the above plots that the system is rather sensitive to the bias points. Since noise in real-world systems is of the order of tens of millivolts, care must be taken in biasing the circuit and choosing the initial state V_0.

Regardless, we can now choose bias points using the above points such that our system exhibits chaotic behavior. An example time-evolution is shown below:

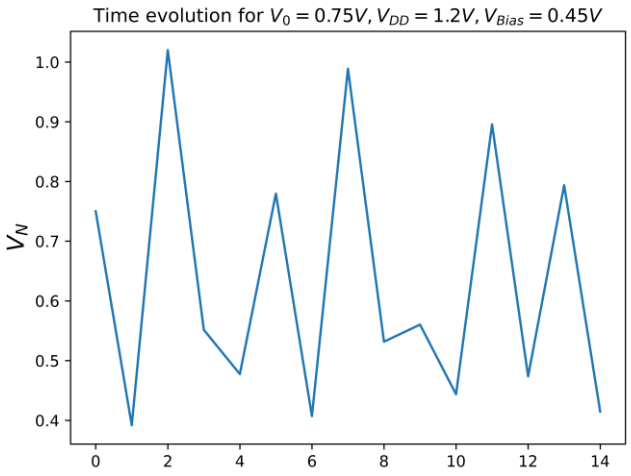
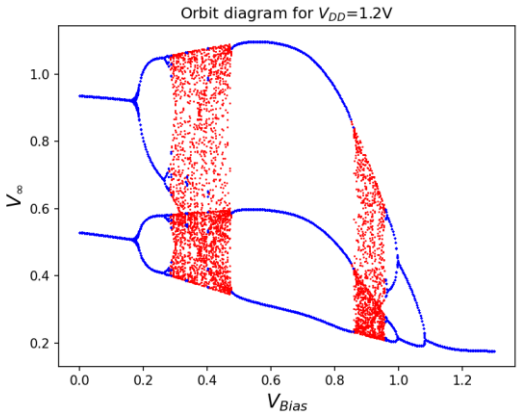


Fig 6.3: Time evolution of the state $V_0 = 0.75V$, for the bias conditions $V_{DD} = 1.2V, V_{Bias} = 0.45V$.

In addition, we also plot the orbit diagrams (bifurcation diagrams) for the transfer function. The regions where $\boldsymbol{\lambda > 0}$ are colored red while that with $\boldsymbol{\lambda < 0}$ are colored blue.



+

Fig 6.4: Bifurcation diagram of the system at $V_{DD} = 1.2V$.

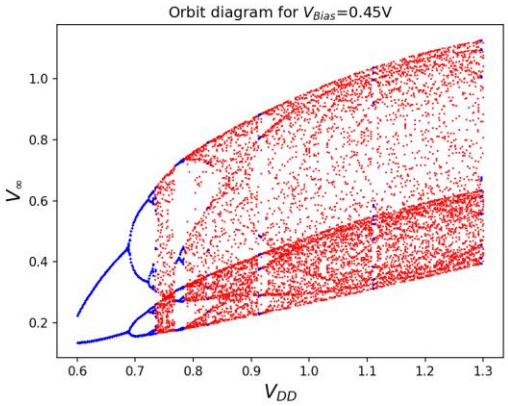


Fig 6.5: Bifurcation diagram of the system at $V_{Bias} = 0.45V$.

VII. Conclusion

We have seen how chaotic maps can be used to implement logical computing devices. We have demonstrated how the same system can act as a different gate for different set of parameters. In particular, we saw from fig 3.2 the type of logic gate that is implemented has sensitive dependence on the parameter x_{gate} for $n > 5$. We require the map to be dependent on these parameters to be able to control the type of gate implemented, but at the same time, not be too sensitive to fluctuations in parameter values. We then showed our idea for the physical implementation of the tent map, which can also be used to morph between different gates, just like the logistic map. We then discussed the implementation of another efficient non-linear map using CMOS. Here, we again found the sensitive dependence of the system on parameters (in this case voltage bias points). Chaotic computational devices, thus, are prone to environmental noise. Despite the ability to use the same system for different gates, chaogates suffer from the drawback of having to deal with additional noise. More work in the field of noise reduction could lead to harness the full potential of these devices.

VIII. Arduino Implementation:

We have simulated the logistic map on an Arduino Uno, by encoding logic gates using the iteration **dependence**.

Codes:
The jupyter notebooks used to arrive at these plots can be found here:
https://github.com/Jaguarrr-111/chaogate-NLD

Microcontroller Implementation:
Our microcontroller implementation of logistic map can be found here:
https://drive.google.com/file/d/1YrZvmx6EHtiI1kMkt4osG23rF8v4aJ89/view?usp=share_link

References:
Ref [1]: https://www.sciencedirect.com/science/article/pii/S096007790900071X?via%3Dihub
Ref [2]: https://ieeexplore.ieee.org/document/9762783
Ref [3]: https://www.mitchellspryn.com/2021/07/07/Computing-With-Chaos.html

Thank You and Have a Cookie!