Chaotic computing

A Project Report

Submitted by

ADARSA.S	**AM107EC001**
ARATHY.R.KUMAR	**AM107EC010**
NAVYA.M.K	**AM107EC040**

under the guidance

Of

Dr. NITHIN NAGARAJ

Br. KARTHI BALASUBRAMANIAN

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

At

Amrita School of Engineering

AMRITA VISHWA VIDYAPEETHAM

Amritapuri - 690 525

MAY 2011

Declaration

I hereby declare that the work presented in this thesis entitled, **"Chaotic computing"** submitted for the B.Tech Degree is my original work and the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles.

Signature of the Students:
Adarsa S
Arathy R Kumar
Navya M K

Place: Amritapuri,
Date:_____

In the name of God, dedicated to our parents and all Teachers who inspire us to gain true knowledge and walk the path of righteousness.

CONTENTS

LIST OF FIGURES iii

LIST OF TABLES v

ACKNOWLEDGMENT vi

ABSTRACT vii

1. **INTRODUCTION TO CHAOS** 1

1.1 SENSITIVE DEPENDENCE ON INITIAL CONDITIONS 1

1.2 CHAOS-BUTTERFLY EFFECT-WEATHER PREDICTIONS 1

1.3 LOGISTIC MAP 2

1.3.1 PERIODIC POINTS 2

1.3.2 BIFURCATION DIAGRAM FOR LOGISTIC MAP 3

1.3.3 PERIOD DOUBLING PHENOMENON 4

1.3.4 INTERMITTENCY 4

PROJECT OVERVIEW 5

2. **DYNAMIC EVOLUTION OF LOGIC FROM CHAOTIC ELEMENTS** 6

2.1 INTRODUCTION 6

2.2 BASIC LOGIC OPERATIONS WITH A CHAOTIC MAP 7

2.3 GENERATION OF A SEQUENCE OF LOGIC OPERATIONS USING ITERATES OF A CHAOTIC MAP 8

2.3.1 CASE STUDY: LOGISTIC MAP $f(x) = 4x(1-x)$ 8

2.3.1.1 MULTIPLE ITERATIONS 8

3. **LITERATURE SURVEY VERIFICATION** 12

4. **MODULO ADDITION** 21

4.1 DETERMINING THE NECESSARY CONDITIONS 21

4.2 REDUCTION OF TABLE AND CHOOSING THE PARAMETERS . . . 22

4.3 CODING TO CHECK FOR PARAMETERS SATISFYING THE CONDITION 23

4.4 REDUCTION OF PARAMETERS FOR COMPUTATIONAL EFFICIENCY 24

4.5 ANALYSIS 24

5. **USING SYNCHRONIZATION TO OBTAIN DYNAMIC LOGIC GATES** 34

CONCLUSION 43

FUTURE ENHANCEMENT 45

LITERATURE CITED 46

LIST OF FIGURES

1.1 Lorentz attractor [1] 3

2.1 Three types of input/output configurations: (a) logical AND, OR, and XOR; (b) logical NOT; (c) bit-by-bit arithmetic addition [2] 10

2.2 Schematic diagram of the nonlinear evolution based logic operations [3] 10

2.3 Graphical iteration representation of the logistic map 10

2.4 Necessary and sufficient conditions to be satisfied by a chaotic element [4] 11

2.5 Template showing different logic patterns [4] 11

3.1 Bifurcation 12

3.2 Lorenz attractor 13

3.3 Multigate without tolerance for logistic map 14

3.4 Multigate with tolerance for logistic map 15

3.5 3D for logistic map 16

3.6 Tent map 17

3.7 Multigate without tolerance for tent map 17

3.8 Multigate with tolerance for tent map 18

3.9 Multiple iteration for logistic map 19

3.10 Multiple iteration for tent map 20

4.1 Analysis figure 1 25

4.2 Analysis figure 2 25

4.3 Modified piecewise linear map 26

4.4 Result figure 26

4.5 Result figure 27

4.6 Graphically obtained solutions 27

4.7 Implementation of AND gate using modified piecewise linear map . . . 30

4.8 Implementation of OR gate using modified piecewise linear map 30
4.9 Implementation of NAND gate using modified piecewise linear map . . 31
4.10 Implementation of NOR gate using modified piecewise linear map . . . 31
4.11 Implementation of XOR gate using modified piecewise linear map . . . 32
4.12 Implementation of XNOR gate using modified piecewise linear map . . 32
4.13 Implementation of NOT gate using modified piecewise linear map . . . 33
5.1 Synchronization using logic gates [5] 35
5.2 Chua circuit [5] 35
5.3 Parameter setting of drive system yielding output of operation for six logic gates [5] 36
5.4 Result figure 37
5.5 Result figure 38
5.6 Result figure 39
5.7 Result figure 40
5.8 Result figure 41
5.9 Result figure 42

LIST OF TABLES

2.1 Necessary and sufficient conditions to be satisfied by the chaotic element to implement each of the logical operations [2] 7

2.2 Single iteration of f(x)[2] 8

4.1 Truth table for 3-inputs 22

4.2 Truth table for 2-inputs 23

4.3 Necessary and sufficient conditions for modulo addition 23

4.4 Possible values of T1 28

4.5 Necessary and sufficient conditions to be satisfied by the chaotic element to implement each of the logical operations [6] 29

4.6 The values of initial condition, threshold and delta for the implementation of logic gates using modified piecewise linear map 29

ACKNOWLEDGMENT

Our humble pranams at the lotus feet of Amma, Satguru Mata Amritanandamayi Devi, our guiding force and inspiration.

We would like to express our deepest gratitude to Dr. K Sankaran, Principal, Amrita School of Engineering, Amritapuri Campus, for providing necessary facilities and an ideal environment to carry out this work.

We are highly grateful to Dr. Sundararaman Gopalan, Chairperson, Department of Electronics and Communication for his valuable support.

We are thankful to Mrs. Poorna.S.S, Ms. Parvathy Nair and Mrs. Sreedevi.K.Menon, Project Coordinators, for their suggestions and willingness to help.

We avail this opportunity to express our sincere gratitude to our teachers Br. Karthi Balasubramanian, Assistant Professor and Dr. Nithin Nagaraj, Assistant Professor for their guidance, advice and encouragement at every step of this endeavor.

We extend our profound thanks to all the faculty members of our department for the technical know-how provided to us which greatly helped us in accomplishing the work successfully.

We are greatly indebted to our parents and friends for their wholehearted support and prayers which made it possible for us to successfully complete this project in time. Above all we are always thankful to God Almighty for giving us good health and mental strength for completing the project.

ABSTRACT

In our project, our endeavour is to implement fundamental computing functions by using chaotic elements. This would provide a theoretical foundation of computer architecture based on a totally new principle other than silicon chips. The fundamental functions discussed are: the logical AND, OR, NOT, XOR, and NAND operations (gates) and modulo arithmetic operation. The chaotic elements employed in the implementation were logistic map (nonlinear), tent map (symmetric piecewise linear) and a newly introduced modified piecewise linear map.

Dynamical evolution of logic can be obtained either by thresholding or by synchronization between two maps. Initially we studied the implementation of basic logic gates, half adder and subtractor using the concept of threshold. Our research then delves into design of modulo arithmetic logic. Having understood the inability to implement modulo arithmetic using logistic and tent map, it has been successfully implemented using modified piecewise linear map. Later we studied the concept of synchronization between two maps and implemented logic gates using it. Hence the research would provide a firm ground to build other modulo arithmetic and computer operations.

CHAPTER 1
INTRODUCTION TO CHAOS

Chaos is the phenomenon of occurrence of bounded non periodic evolution in completely deterministic nonlinear dynamical systems with high sensitive dependence on initial conditions. This is called deterministic chaos since governing equations are deterministic.

For a system to be chaotic, it should have sensitive dependence on initial conditions. It has aperiodic orbit. Though it seems to be random, system is actually deterministic. The rate of separation of nearby trajectories is positive.

1.1 SENSITIVE DEPENDENCE ON INITIAL CONDITIONS

Let f be a map on R. A pt x_0 has sensitive dependence on initial conditions if there is a non-zero distance 'd' such that some points arbitrarily near x_0 are eventually mapped at least d units from the corresponding image of x_0. More precisely, there exists $d > 0$ such that any neighbourhood ϵ of x_0 contains a point x such that $|f'_k - f'_k(x0)| \geq d$ for some non-negative integer k. Sometimes we will call such a point x_0 a sensitive point [1].

1.2 CHAOS-BUTTERFLY EFFECT-WEATHER PREDICTIONS

The sensitive dependence of chaotic solution on initial conditions was first observed by E.N.Lorentz in a system of three coupled first order ordinary differential equations describing hydrodynamic flow now popularly known as Lorentz system. When the notion of sensitive dependence of chaotic system was applied to atmosphere, which is expectedly non periodic, it indicates that prediction of a sufficiently distant future is impossible by any method, unless the present conditions as well as subsequent evolutions are known exactly. In view of the inevitable inaccuracy and incompleteness of weather observations, precise and very long range forecasting would seem to be nonexistent-that is, even a minute perturbation can cause realizable effects in

a finite time under chaotic evolution. To describe the dramatic extreme sensitive dependence of chaotic solution, Lorentz coined the term butterfly effect. "As small a perturbation as a butterfly fluttering its wings somewhere in the Amazons can in a few days time grow into a tornado in Texas". That is even a minute perturbation can cause realizable effects in a finite time under chaotic evolution.

Equilibrium conditions: Equilibrium(0,0,0) exists for all r, and for $r < 1$ its a stable attractor. The origin corresponds to fluid at rest with a linear temperature profile - hot at bottom and cool at top. Two new equilibria exist for $r \geq 1$, $C_+ = (\sqrt{b(r-1)}, \sqrt{b(r-1)}, r-1)$ $C_- = (\sqrt{b(r-1)}, -\sqrt{b(r-1)}, r-1)$ representing steady convective circulation(clockwise or anticlockwise flow). This pair of equilibria branch off from origin at $r = 1$ and move away as r is increased. For $r \geq 1$, origin is unstable with two equilibria representing convective rolls, C_+ and C_- are stable at their birth $r = 1$ and remain stable.

1.3 LOGISTIC MAP

A function whose domain(input) space and range(output) space are the same will be called a map. Let x be a point and let f be a map. The orbit of x under f is a set of points $x, f(x), f^2(x), ...$ The starting point x for the orbit is called the initial value of the orbit. A point p is a fixed point of the map f if $f(p) = p$. Let f be a map on R and let p be a real number such that $f(p) = p$. If all the points sufficiently close to p are attracted to p, then p is called a sink or an attracting fixed point. If all the points sufficiently close to p are repelled from p, then p is called a source or repelling fixed point [1].

1.3.1 PERIODIC POINTS

Let f be a map on R. We call 'p' a periodic point of period k if $f'_k(p) = p$, and if k is the smallest such positive integer. The orbit with initial point p (which consist of k points) is called periodic orbit of period k. Let f be a map and assume that p is a periodic k point. The periodic k orbit of p is a periodic sink if p is a sink for the map f'_k. The orbit of p is a periodic source if p is a source for the map f'_k. The periodic orbit is a sink if $|f'(p_k), ..., f'(p_1)| < 1$ and is a source if $|f'(p_k), ..., f'(p_1)| > 1$ [1].

$$\dot{x} = -\sigma x + \sigma y$$

$$\dot{y} = -xz + rx - y$$

$$\dot{z} = xy - bz.$$

$$\sigma,\ r,\ b > 0$$

- σ – Prandtl number
- r– reynold's number
- b –system parameter (width of flow rolls)
- x --->circulatory fluid flow velocity
- y--->temperature diference
- z--->distortion of vertical temperature profile frm its eqbm

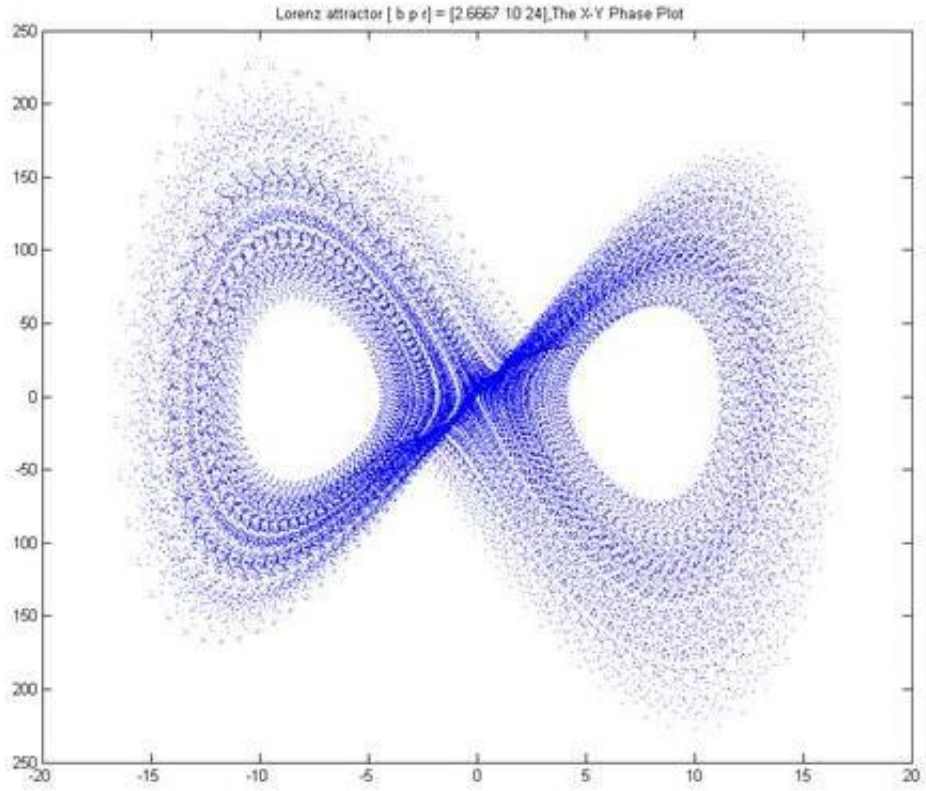


Figure 1.1: Lorentz attractor [1]

1.3.2 BIFURCATION DIAGRAM FOR LOGISTIC MAP

In bifurcation diagram, at $a = 3.4$, a vertical line intersects the diagram in two points of a periodic sink. For 'a' slightly larger, there appears to a periodic 4 sinks. There is an entire sequence of periodic sinks, one for each period 2^n, n=1,2,3,....Such a

sequence is called period doubling cascade. A bifurcation diagram: shows the birth, evolution, and death of attracting sets bifurcation diagram for $f(x) = ax(1-x)$.

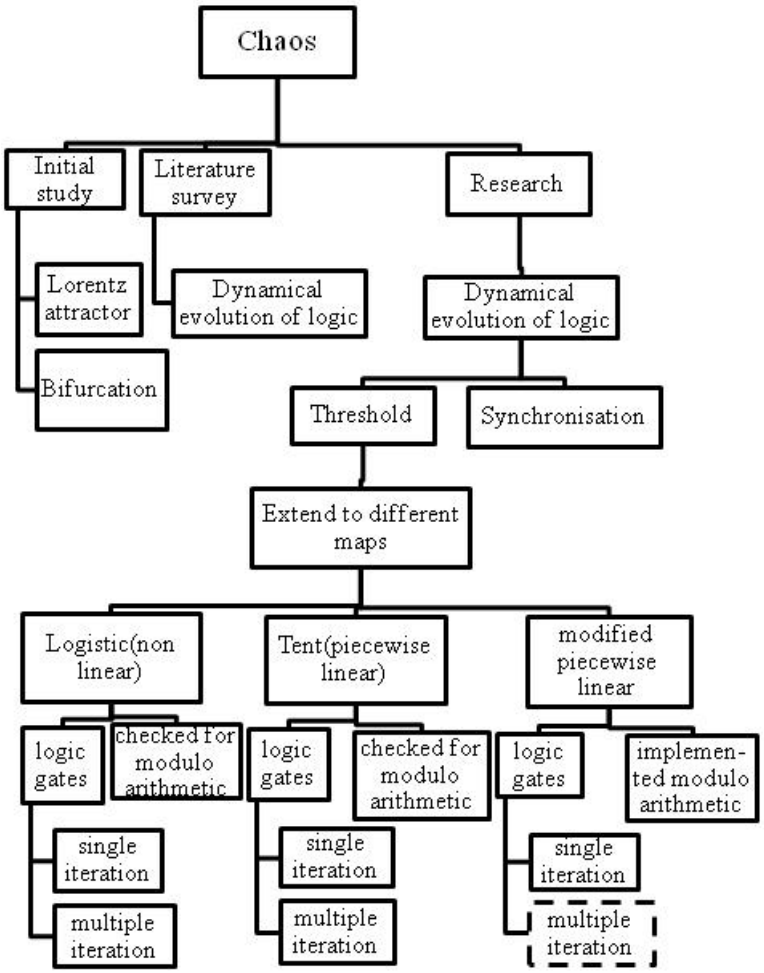
1.3.3 PERIOD DOUBLING PHENOMENON

Period-1 solution $x^* = (a-1)/a$ is unstable for $a > 3$. Since slope crosses $f' = -1$ at $a = 3$(period-2 born). Period-2 solution becomes unstable at $a = 1+\sqrt{6}$. Therefore, $f'(x_1^*)f'(x_2^*) = -1$ at $a = 1+\sqrt{6}$ Similarly, period 4 cycle is unstable at condition where $f'(x_1^*)f'(x_2^*)f'(x_3^*)f'(x_4^*)$ becomes =-1 When the stability determining quantity of a period k solution becomes -1, a bifurcation occurs giving birth to a stable period 2k solution. Such bifurcations associated with stability determining quantity crossing value = -1 is called flip bifurcation(period doubling or subharmonic bifurcation). Period 4 cycle exist for $1+\sqrt{6} < a < 3.544112$ and this solution bifurcates into period 8 cycle solution at $a = a_c = 3.544112$ and process proceeds further to infinity. The successive bifurcations occurs faster and faster as a is increased. Ultimately an converges to a limiting value $a_c = 3.57$. This convergence is essentially geometric in the limit of large n, the distance between successive transitions shrinks by a constant factor or universal constant d, the so called Feigenbaum's universal number or Feigenbaum's constant. An important characteristic properties of the solutions of logistic map for $a < a_c$(where only periodic solution occur) is that they are insensitive to initial values of x provided $0 < x < 1$.

1.3.4 INTERMITTENCY

For a period-3 window, system exhibits an interesting kind of chaos. The orbit returns to the ghostly 3 cycle repeatedly, with intermittent bouts of chaos between visits near saddle node bifurcation. This phenomenon is called intermittency. Intermittency repeats as nearby motion interrupted by occasional irregular bursts. The time between bursts is like a random variable. As the parameter is moved farther away from periodic window, the burst become more frequent until the system becomes fully chaotic. This progression is called intermittency route to chaos.

PROJECT OVERVIEW



CHAPTER 2
DYNAMIC EVOLUTION OF LOGIC FROM CHAOTIC ELEMENTS

2.1 INTRODUCTION

Chaos is all around us and found in many disciplines, such as physics, chemistry, biology, medicine, and engineering. For example,chaotic phenomena are found in lasers, electronic circuits, chemical systems, brains and hearts. Many researchers have worked in the field because of both theoretical and practical importance of the subject, and challenging and fascinating features of extreme nonlinearity. Chaos represents a deterministic dynamical system that is nonlinear, sensitive to initial conditions (the so-called butterfly effect), and exhibits sustained irregularity.

A small change in the initial condition can yield a significantly different sequence of random numbers. A recurring theme of research into chaotic systems over the last decade has been that chaos provides flexibility in the performance of natural systems and provides such systems with a rich variety of behaviors that can be utilized for improved performance. Successful implementations of this concept have included the exploitation of chaotic behavior for control, synchronization, encoding information and communications.

Here we present an analysis for implementing a complete set of logical gates with two inputs and one output, realized by 1-D (one-dimensional) chaotic (or very nonlinear system) element. We present the basic concepts underlying chaos computing and describe methods to design nonlinear systems to flexibly yield all fundamental logic functions by "programming" different dynamical systems. The patterns produced by varying the initial conditions and the parameters of a chaotic system, as well as the patterns produced by its time evolution, are amenable to performing computation. By combining both techniques, one can produce a computational device of immense power and elegance.

2.2 BASIC LOGIC OPERATIONS WITH A CHAOTIC MAP

Consider a single chaotic element whose state is represented by a value x, as our chaotic chip or chaotic processor. The state of the element evolves according to some dynamical rule exhibiting chaos. For instance, the updates of the state of the element from time n to $n+1$ may be well described by a map, i.e., $x_{n+1} = f(x_n)$ where f is a nonlinear function chosen to obtain chaotic dynamics. Now this element receives two inputs (for AND, OR, and XOR) or one input (in case of NOT) and outputs a signal.

Refer Fig 2.1. Analysis steps include:

1. **Initialisation** of state x of system and addition of external inputs. $x = xo + x1 + x2$.

2.**Chaotic update**, $x = f(x)$,where $f(x)$ is a chaotic function.

3.**Threshold mechanism** to obtain output z.

$z = 0$, if $f(x) \leq x^*$;

$z = \delta = f(x) - x^*$ if $f(x) > x^*$, where x* is called the threshold value of the system.

Refer Fig 2.2.

Operation	AND	OR	XOR
Conditions	$f(x_0) \leq x^*$ $f(x_0 + \delta) \leq x^*$ $f(x_0 + 2\delta) - x^* = \delta$	$f(x_0) \leq x^*$ $f(x_0 + \delta) - x^* = \delta$ $f(x_0 + 2\delta) - x^* = \delta$	$f(x_0) \leq x^*$ $f(x_0 + \delta) - x^* = \delta$ $f(x_0 + 2\delta) \leq x^*$

Operation	NAND	NOT
Conditions	$f(x_0) = \delta$ $f(x_0 + \delta) - x^* = \delta$ $f(x_0 + 2\delta) \leq x^*$	$f(x) - x^* = \delta$ $f(x_0 + \delta) \leq x^*$ $f(x_0 + \delta) - x^* = \delta$

Table 2.1: Necessary and sufficient conditions to be satisfied by the chaotic element to implement each of the logical operations

We demand δ to be a common positive constant for all the operations so that an output from one gate can directly be fed into another gate as input. On the other hand, x_0 and x^* differ among the operations, although the same symbols are used for simplicity.

Typically, we may start selecting function f(x). This may be determined by characteristics of a physical device for actual implementation. Given a specific function, we may search for solutions in terms of δ, since this must be a constant throughout all the operations. We determine a pair of (x_0, x^*) for each of the operations consistent with δ, where δ may have to be chosen appropriately so that solutions for all operations exist. We note that our computing scheme under many chaotic systems will be robust for background noise.

2.3 GENERATION OF A SEQUENCE OF LOGIC OPERATIONS USING ITERATES OF A CHAOTIC MAP

2.3.1 CASE STUDY: LOGISTIC MAP $f(x) = 4x(1-x)$

We select the constant δ, common to both input and output and all logic gates, to be 1/4.

operation	AND	OR	XOR	NAND	NOT
x_0	0	1/8	1/4	3/8	1/2
x^*	3/4	11/16	3/4	11/16	3/4

Table 2.2: Single iteration of f(x)[2]

For AND operation, for example, selecting $x_0 = 0$ and $x^* = 3/4$ suffices the three conditions given in Table I as follows:

$f(x_0) = f(0) = 0 < 3/4 = x^*$

$f(x_0 + \delta) = f(1/4) = 3/4 \leq x^*$

$f(x_0 + 2\delta) - x^* = f(1/2) - 3/4 = 1 - 3/4 = 1/4 = \delta$

2.3.1.1 MULTIPLE ITERATIONS

The constant δ, common to all logical gates, is fixed as 0.25. The inequalities(necessary and sufficient conditions) have many possible solutions based on the size of δ. For example, by setting δ= 0.25, we can simulate the equation for the different time shifts that each gate requires. Thus the inputs setup the initial state $x_0 + I1 + I2$. Then the system evolves over n iterative time steps to an updated state x_n. The evolved state is compared to a monitoring threshold x^* at every n.

If the state at iteration n, is greater than x^* a logical 1 is the output and if the state is less than or equal to x^* a logical 0 is the output. This process is repeated for subsequent iterations. Relating inputs with the obtained outputs provides us the operation that is performed at a specific iteration. For illustrative purposes, Graphical iteration representation of the logistic map with three logic initial Inputs (a)$x = x_0$, (b)$x = x_0 + \delta$ and (c)$x = x_0 + 2\delta$ corresponding to Table is shown. Here $x^* = 0.75$ is used to recover logic operations NAND, AND, NOR and XOR. For OR logic operation $x^* = 0.4$ is utilized. Various initial values corresponding to different logic inputs is depicted in Fig. 2.2. The initial values are denoted by labels a, b and c. For clarity, the state of x_n For first 5 iterations ($0 < n < 5$) can be identified from this diagram. It is interesting to note that first 5 iterations satisfy the realization of basic logic gates as indicated in Table 1. In addition, subsequent iterations beyond $n > 5$ continue to yield different logic gate operations including XNOR operation. A more exclusive template of various logic responses being admitted by this system (Eq. (1)) for different iterations n versus range of x_0 values is depicted in Fig. 2.3. To generate this template, the representative value of δ is fixed as 0.25. The value $x^* = 0.75$ is used for $1 \leq n \leq 4$ and $x^* = 0.4$ is used for $n > 4$. Fig 2.3 shows the logic behavior arising from a system with initial state x_0 evolving over n iterative steps, with n = 1, 2, . . . , 10. It is clear from this figure, that while the system will always yield some logic behavior, the robustness of the response, with respect to initial state specification is lost after n around 5 or so. This is expected from the chaotic nature of the dynamics, and so for large n the response is extremely sensitive to the precision with which x_0 is set. However note that one need not go to iterates beyond 5 or so, as all basic logic outputs can be obtained within the first few iterates, in large robust ranges of initial state x_0. After n around 5 or so, the system can be re-set, for instance by the threshold controller mentioned earlier, and the nonlinear system can be 're-used' after this re-initialization.

Refer Fig 2.3,Fig 2.4 and Fig 2.5.

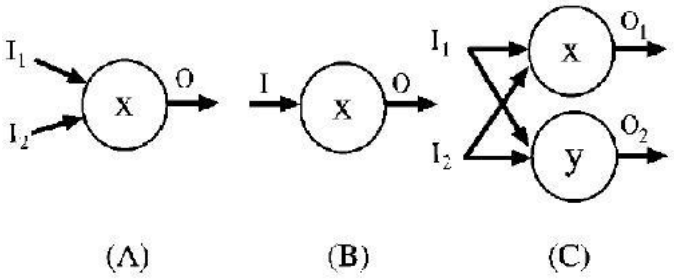


Figure 2.1: Three types of input/output configurations: (a) logical AND, OR, and XOR; (b) logical NOT; (c) bit-by-bit arithmetic addition [2]

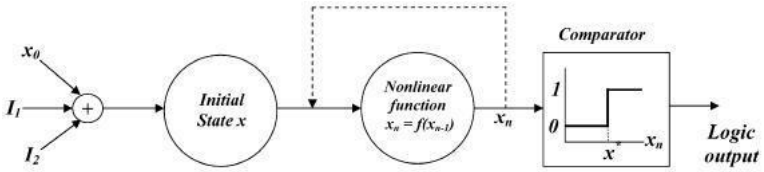


Figure 2.2: Schematic diagram of the nonlinear evolution based logic operations [3]

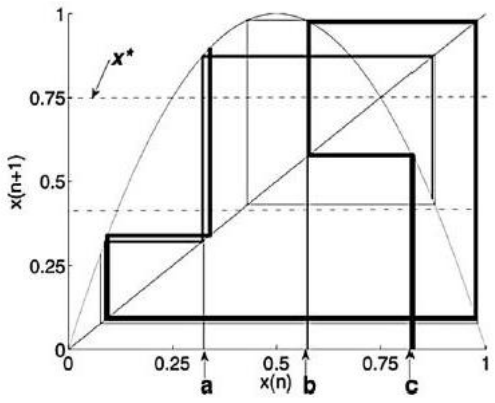
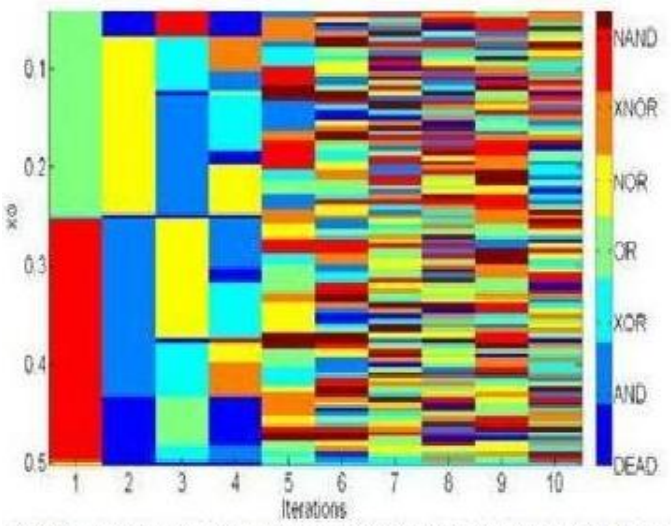


Figure 2.3: Graphical iteration representation of the logistic map

LOGIC	NAND	AND	NOR	XOR	OR
Iteration 'n'	1	2	3	4	5
Condition 1: Logic input (0, 0) $x_0 = 0.325$	$x_1 = f(x_0) > x*$ $x_1 = 0.88$	$f(x_1) < x*$ $x_2 = 0.43$	$f(x_2) > x*$ $x_3 = 0.98$	$f(x_3) < x*$ $x_4 = 0.08$	$f(x_4) < x*$ $x_5 = 0.28$
Condition 2: Logic input (0, 1) or (1, 0) $x_0 = 0.575$	$x_1 = f(x_0 + \delta) > x*$ $x_1 = 0.9775$	$f(x_1) < x*$ $x_2 = 0.088$	$f(x_2) < x*$ $x_3 = 0.33$	$f(x_3) > x*$ $x_4 = 0.872$	$f(x_4) > x*$ $x_5 = 0.45$
Condition 3: Logic input (1, 1) $x_0 = 0.825$	$x_1 = f(x_0 + 2\delta) < x*$ $x_1 = 0.58$	$f(x_1) > x*$ $x_2 = 0.98$	$f(x_2) < x*$ $x_3 = 0.1$	$f(x_3) < x*$ $x_4 = 0.34$	$f(x_4) > x*$ $x_5 = 0.9$

Figure 2.4: Necessary and sufficient conditions to be satisfied by a chaotic element [4]



TEMPLATE SHOWING DIFFERENT LOGIC PATTERNS FOR RANGE OF x0(0 TO 0.5) VERSUS ITERATION n(0 TO 10).HERE x*=0.75 FOR 1<=n<=4 AND x*=0.4 FOR n>4,delta IS FIXED AS 0.25.

Figure 2.5: Template showing different logic patterns [4]

CHAPTER 3
LITERATURE SURVEY VERIFICATION

The first step in analysing logistic map was to obtain the bifurcation diagram by varying 'a' from 0 to 4. This was done using Matlab and the following figure was obtained. We can see that fixed points that exist for small values of a gives way to a period 2 orbit at the bifurcation point, 'a'=3 which in turn leads to more and more complicated orbits for larger values of 'a'. Notice that fixed point is plotted only while it is a sink.

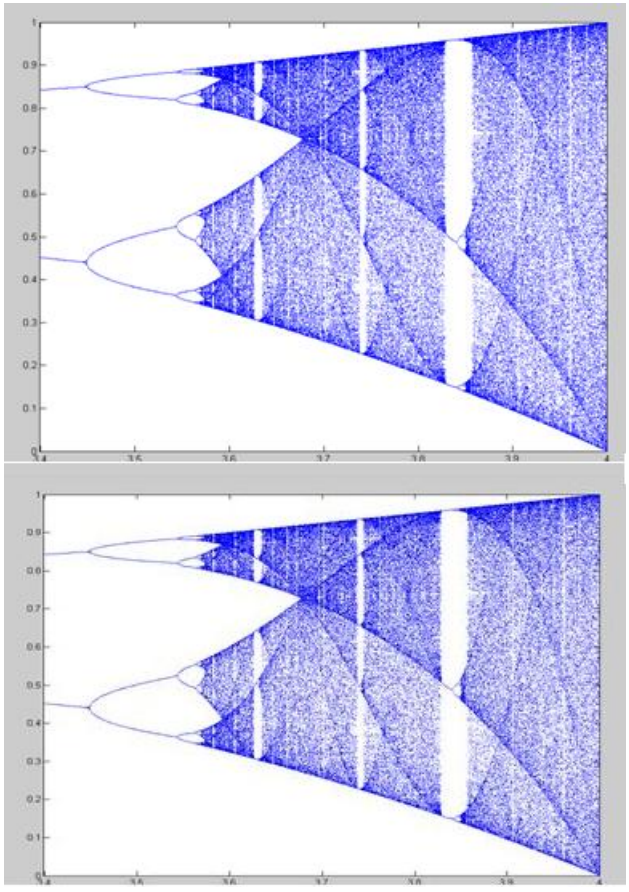


Figure 3.1: Bifurcation

We can see onset of period 3 for value of 'a' greater than 3.8 leading to chaos. Also

the self similarity nature of the map may be observed. The other system studied for understanding chaos, the Lorentz attractor was implemented next and the following figure was obtained. The parameters were fixed as $\sigma = 10$, b=8/3 and r was varied around 24 which is considered as the traditional value for chaos. During this the transient chaos was observed. One can see the two stable points around which the symmetric figure is obtained.

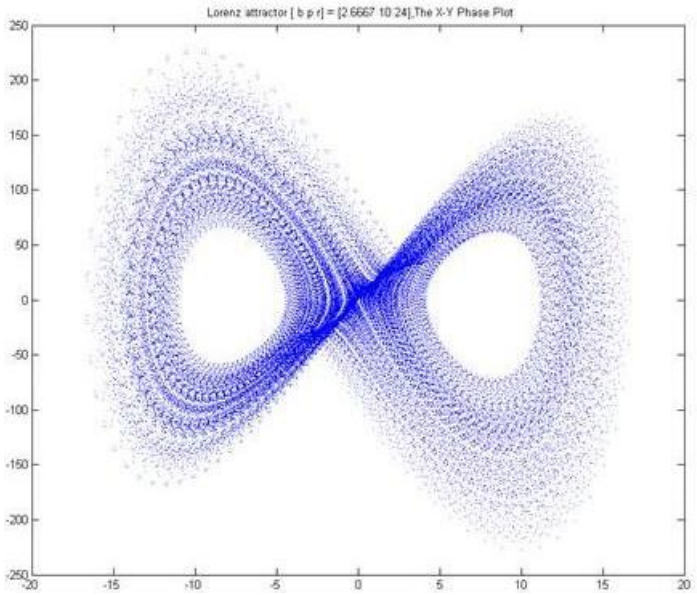


Figure 3.2: Lorenz attractor

The non-periodic nature and sensitive dependence on initial condition, the inherent property of chaos, can be observed evidently in this case. Hence it provided a good idea about chaos and its nature.

The method of chaotic computing proposes that we can directly implement the most fundamental computer functions(basic logic operations AND, OR, NOT, XOR and NAND, bit by bit arithmetic operations such as addition) using chaotic element. Once the systems were obtained, our motive was to search for logic gates amidst chaos. We have seen that at a=4, the logistic map exhibits chaos. Hence we chose this region to implement possible logic gates. There were three variables to be fixed namely initial condition, delta and threshold(refer chapter 2). Initially we tried

fixing each parameter one by one. From background study [7], it was noted that by fixing delta as 0.25 and varying the other two parameters, it was possible to implement five logic gates. Thus at delta=0.25 the following gates were obtained. Each gate is represented here using different colours namely red for AND, cyan for OR, blue for NAND, magenta for NOR, green for XOR, yellow for XNOR. It must be noted that since the gates, OR and NAND are obtained at threshold 0.6875, it is not possible to obtain them during a sweep from zero to one with a precision of 0.001. It was separately obtained by taking interval length of 0.6875.

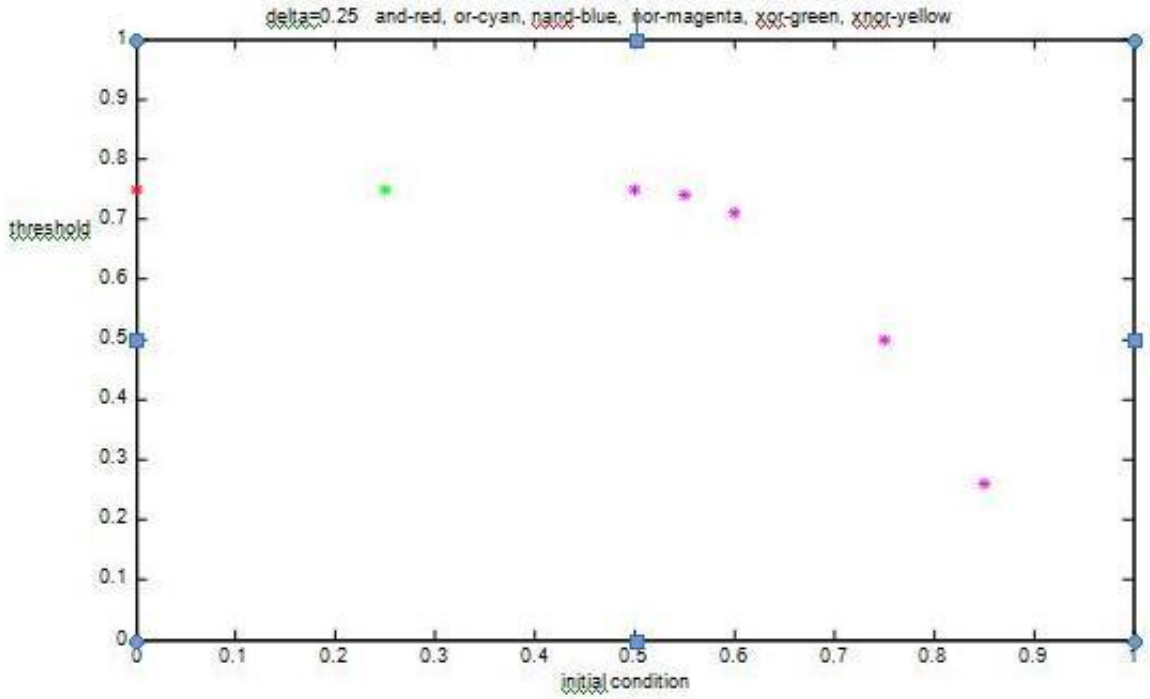


Figure 3.3: Multigate without tolerance for logistic map

Similarly, we tried plotting different logic gates at different values of threshold, delta and initial conditions. The unpredictable nature of chaos surely presented some stunning results!

Considering the implementation of the above mentioned logic gates in hardware it becomes important to introduce a limit of tolerance. It is highlighted by the property of sensitive dependence on initial condition of the system. We used a range of

tolerance on the value of delta i.e, instead of the condition $|f(x) - threshold| = delta$, a small deviation of $|f(x) - threshold - delta| <= tolerance$ was introduced. As expected, it could be observed that as tolerance increases, the number of feasible points also increased.

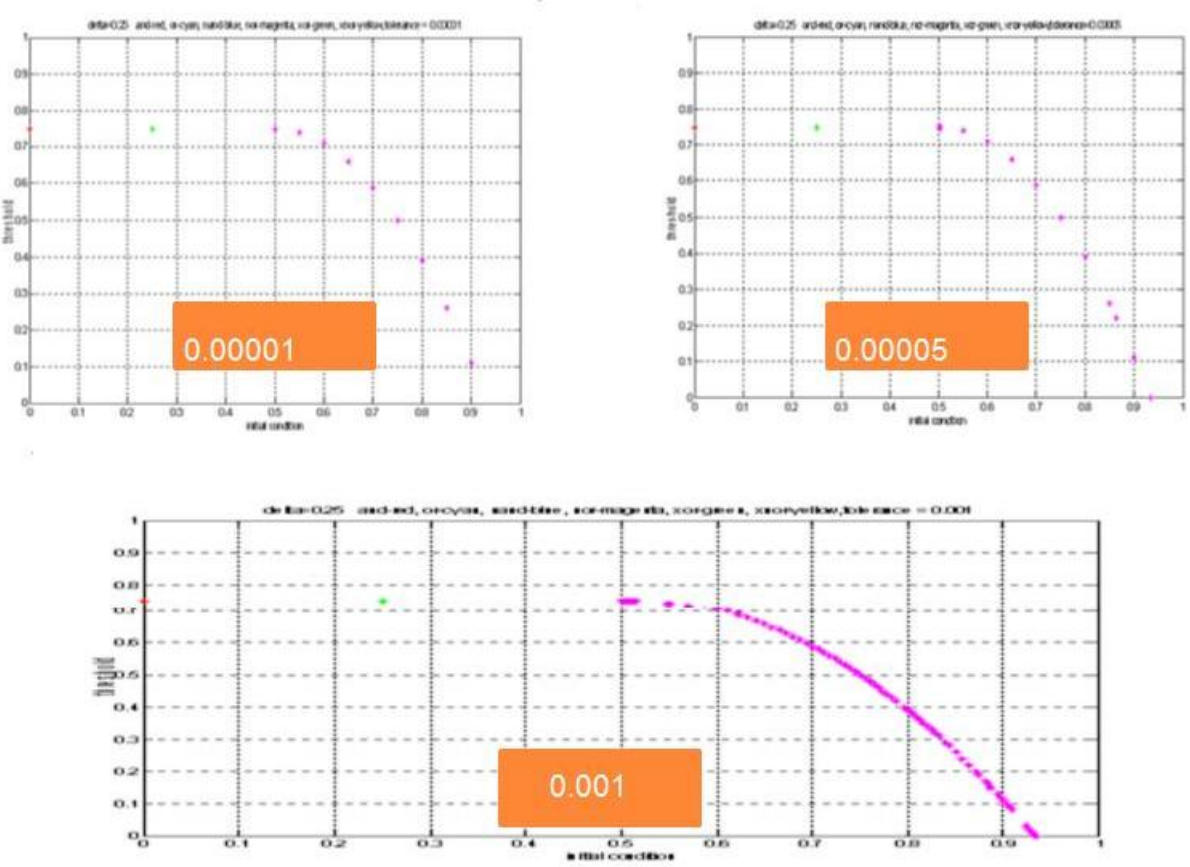


Figure 3.4: Multigate with tolerance for logistic map

Having obtained possible values by fixing one of the parameters, we moved on to varying all the three parameters over a range of zero to one each in the 3-D plane. Thus the figure shown in the next page was obtained.

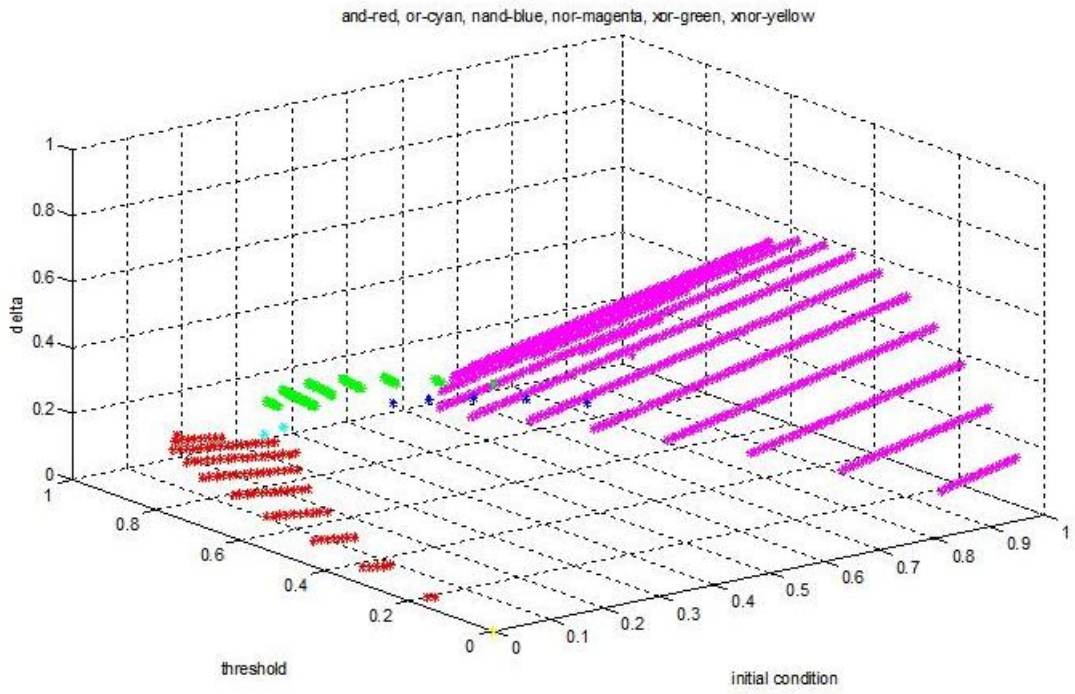


Figure 3.5: 3D for logistic map

It may be noted at this point that it is impossible to find points satisfying conditions for XNOR gate within our limits of computation. But we were delighted to find points corresponding to NAND and OR, though few, within our ranges. Since our real motive is the implementation of a two bit adder, we needed a choice of more than one chaotic system so that we could choose the most efficient one for our implementation. Hence our choice was the one very similiar to logistic map defined by the equation:

$f(x) = 2x$, $0 <= x <= 1/2$

$f(x) = 2(1 - x)$, $1/2 <= x <= 1$

For the tent map for fixed values of delta, the initial conditions and thresholds were varied to search for possible gates. At delta=0.5, the following gates were obtained. Each gate is represented here using different colours namely red for AND, cyan for OR, blue for NAND, magenta for NOR, green for XOR, yellow for XNOR.

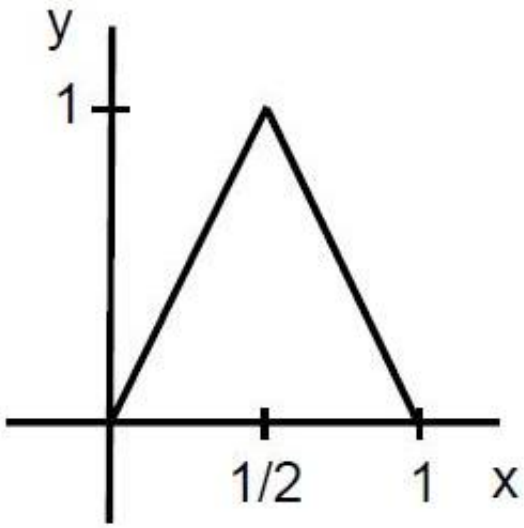


Figure 3.6: Tent map

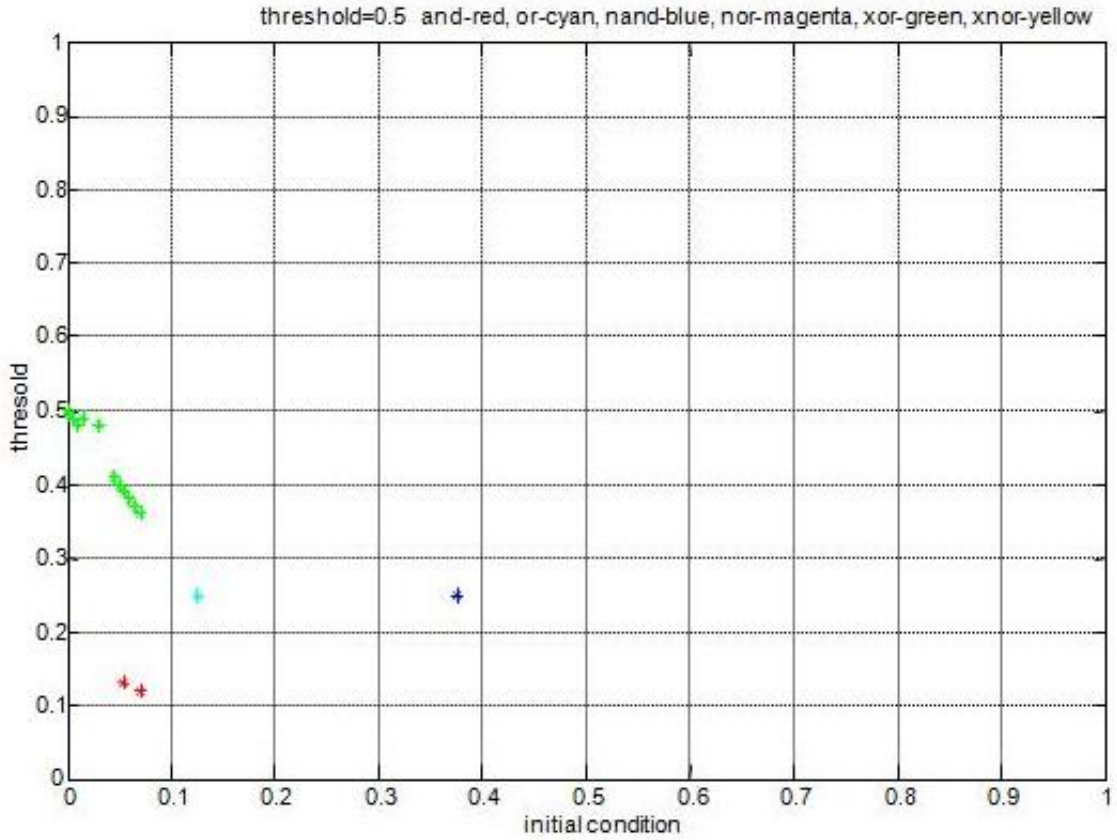


Figure 3.7: Multigate without tolerance for tent map

Interestingly, it may be noted that for the tent map, we are able to obtain OR and NAND gates directly while NOR gate seems to be missing. Even though XNOR is missing in all the cases, these results have made the analysis worthwhile. As done in the case of logistic map, further on we introduced different values of tolerance. The similarities in the results may be accredited to the similarities between the map namely (1)both have a fixed point to the right of critical point.(2)each has a single period 2 orbit. It could be said that for each point x in the tent map domain [0,1], there is a specified companion point C(x) in the logistic map domain [0,1] that imitates its dynamics exactly [1].

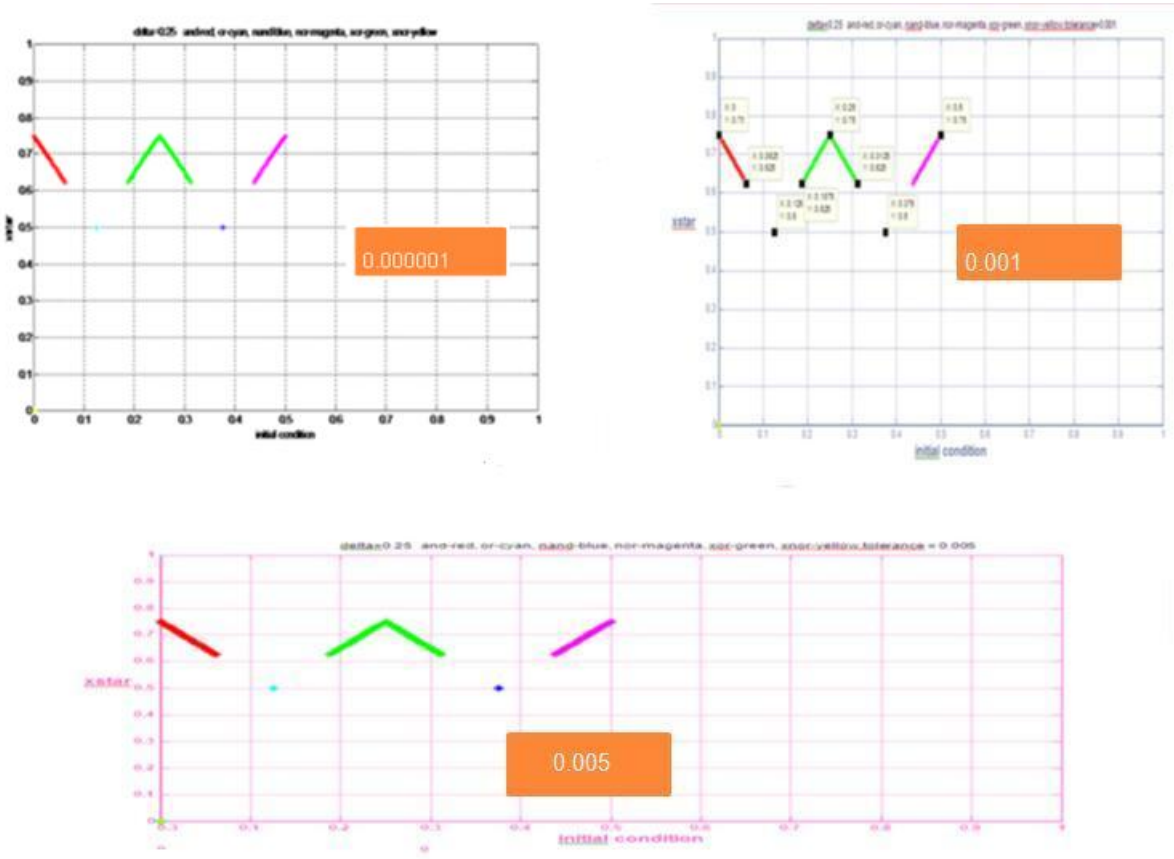


Figure 3.8: Multigate with tolerance for tent map

Until now, the comparison was done using single iteration i.e, each input was iterated in the function once. In the Fig 2.3, one can note that this corresponds to a single interpolation to the line y=x. In the multiple iteration technique, we carry on this interpolation further. From Fig 2.5 [3], one can observe different gates that can be implemented in different iterations of the logistic map. This method has the advantage that one can simply fix all the three parameters namely initial condition, delta and threshold and still obtain different gates by simply iterating the map. For instance, at around $x_0 = 0.34$ in five iterations, one obtains five different gates. Hence, this acts as multiple sequentially connected nonlinear maps with single iteration thus being a prospective candidate for implementation of bit-by-bit arithmetic addition. Using Matlab, we varied the initial condition over a range 0 to 0.5 fixing delta as 0.25 and threshold at 0.75 for iterations from one to four and 0.4 for iterations above four in the logistic map and we obtained a result as proposed in the reference [4]. The same was worked out for the tent map(introduced earlier). Fixing the threshold as 0.5 and delta as 0.25, we obtained logic gates till the second iteration.

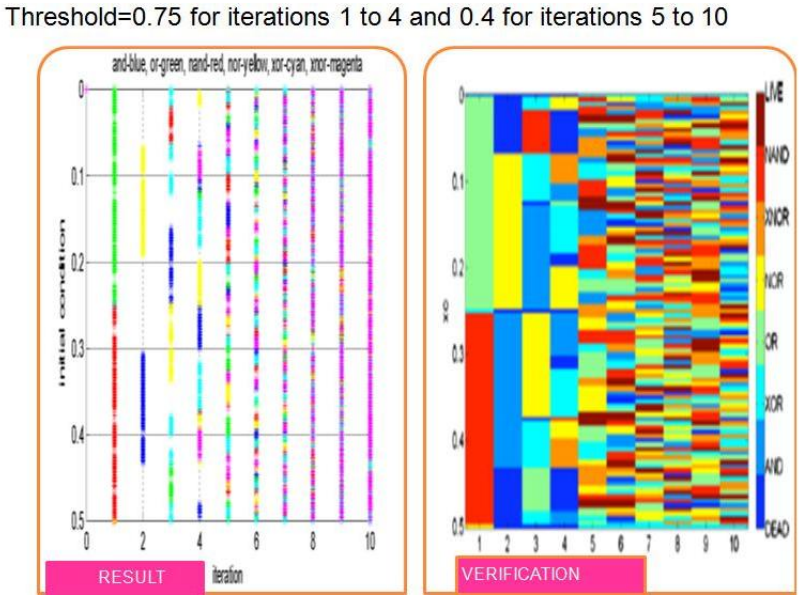


Figure 3.9: Multiple iteration for logistic map

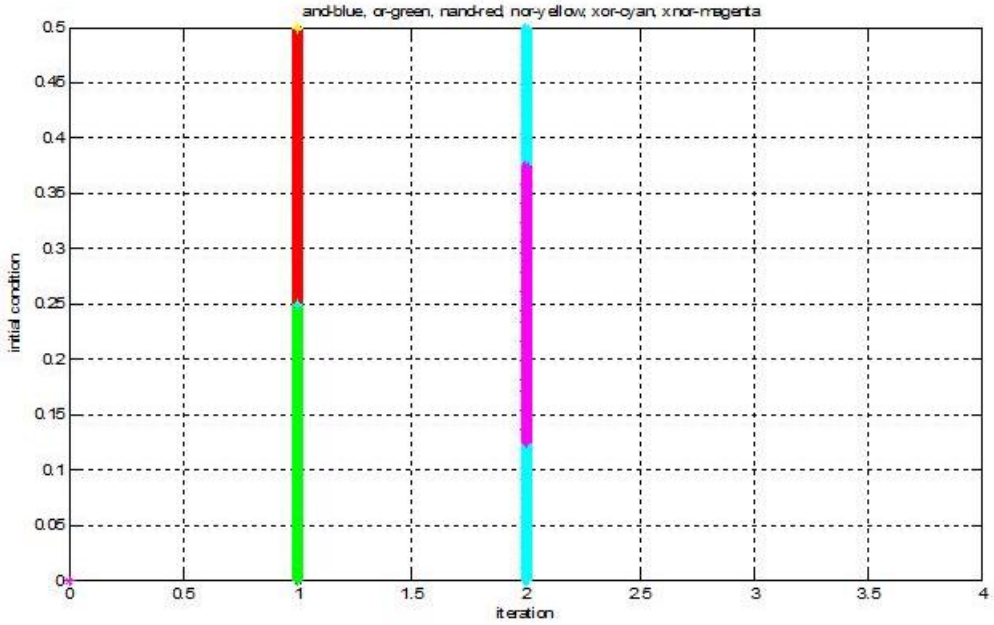


Figure 3.10: Multiple iteration for tent map

It may be noted here that for the first iteration, the pattern of gates is exactly similiar to that of logistic map. Having studied both the maps, with this idea, we moved on to studying and implementing modulo arithmetic which is discussed in the following chapter.

CHAPTER 4
MODULO ADDITION

The aim of research so far was to be able to implement a logic for modulo arithmetic. In our project, we implemented mod 3 addition. This is extendable directly to subtraction. It was observed that logics of higher mod becomes more complex and presents a wide area of study.

A mod n adder accepts inputs from 0 to n-1. The addition is performed and mod n operation is done on it. In the coming sections, we moved to implementing bit-by-bit addition which in terms of hardware is analogous to a half adder. This would in effect give two outputs, a sum and a carry. Hence, a chaotic system would need to satisfy all the conditions imposed simultaneously by sum and carry on the inputs. The implementation was carried out through the following steps:

1. Determining the necessary conditions
2. Reduction of table and choosing the parameters
3. Coding to check for parameters satisfying the condition
4. Reduction of parameters for computational efficiency
5. Analysis

4.1 DETERMINING THE NECESSARY CONDITIONS

The first system studied was a mod 3 full adder. It accepts 3 inputs which may be 0, 1 or 2 and outputs, a sum and carry which again may be 0, 1 or 2. This would involve 27 set of necessary conditions each for sum and carry. A simpler adder can have only 2 inputs, thus 9 sets of necessary conditions.

INPUT1	INPUT2	INPUT3	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	0	2	2	0
0	1	0	1	0
0	1	1	2	0
0	1	2	0	1
0	2	0	2	0
0	2	1	0	1
0	2	2	1	1
1	0	0	1	0
1	0	1	2	0
1	0	2	0	1
1	1	0	2	0
1	1	1	0	1
1	1	2	1	1
1	2	0	0	1
1	2	1	1	1
1	2	2	2	1
2	0	0	2	0
1	1	1	0	1
1	1	2	1	1
1	2	0	0	1
1	2	1	1	1
1	2	2	2	1
2	0	0	2	0
2	0	1	0	1
2	0	2	1	1
2	1	0	0	1
2	1	1	1	1
2	1	2	2	1
2	2	0	1	1
2	2	1	2	1
2	2	2	0	2

Table 4.1: Truth table for 3-inputs

4.2 REDUCTION OF TABLE AND CHOOSING THE PARAMETERS

For simplicity and ease of computation, we consider the condition for sum alone. Since inputs 0, 1 and 1,0 are inherently the same, we can reduce the conditions further to six conditions. Let x0 be the initial condition and 0 correspond to input

INPUT1	INPUT2	SUM	CARRY
0	0	0	0
0	1	1	0
0	2	2	0
1	0	1	0
1	1	2	0
1	2	0	1
2	0	2	0
2	1	0	1
2	2	1	1

Table 4.2: Truth table for 2-inputs

0, delta1 to 1 and delta2 correspond to 2. Let TS1 and TS2 be two thresholds to distinguish between three possible values of outputs such that values from 0-TS1 correspond to 0, TS1-TS2 to 1 and TS2-1 to2. Hence we obtain table 4.3.

CONDITION	INPUT1	INPUT2	f()	OUTPUT
1	0	0	x0	0
2	0	1	$x0 + \delta 1$	1
3	0	2	$x0 + \delta 2$	2
4	1	1	$x0 + 2\delta 1$	2
5	1	2	$x0 + \delta 1 + \delta 2$	0
6	2	2	$x0 + 2\delta 2$	1

Table 4.3: Necessary and sufficient conditions for modulo addition

4.3 CODING TO CHECK FOR PARAMETERS SATISFYING THE CONDITION

Having obtained the reduced table,we need to find five parameters which would satisfy the six conditions. An algorithm for this would be a process of order n^5 (where n =1/smallest interval in which we check for feasible value). Hence this involves a lot of memory and computation time. So we were unable to work with the code for smaller intervals.

4.4 REDUCTION OF PARAMETERS FOR COMPUTATIONAL EFFICIENCY

So as to make the code more efficient, we fixed some of the parameters.This not only reduced the order, but we could also scale down the region to be checked in for other parameters since they are related. But even after days of trial and error in regions that seemed to be prospective from the graph, we were unable to find a solution .

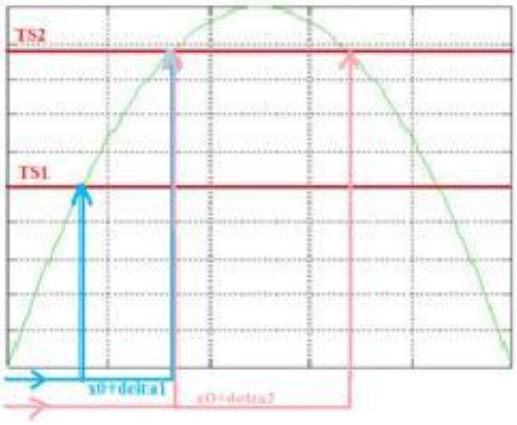
4.5 ANALYSIS

As to understand the inability to find possible points satisfying the required conditions, we tried reducing the required conditions. It was observed that even when we check for in large intervals(0.1) for lesser number of condition(4), we have quite a few values. This is also inferable from the large number of points we got for logic gates (having only 3 conditions).

So what makes these conditions complex? Analysing with the help of Fig 4.1, conditions two and three implies the function value of both is below the first peak and so delta1 has to be less than delta2. Condition 5 says function value x0+delta1+delta2 must be below TS1 while condition 6 says x0+2delta2 must give function value between TS1 and TS2. This means $x0+2delta2 < x0+delta1+delta2$ or $delta2 < delta1$ which contradicts condition drawn from 2 and 3. Hence a solution does not exist.

In this situation, the first way forward seemed to be changing the definition of each region. A cyclic shift of 0,1 and 2 over the regions 0-TS1, TS1-TS2 and TS2-1 does not solve this problem. So regions where randomly defined. Neverthless, a choice without similar contradictions where not obtained.

Hence the need was for a map that would solve the contradiction for the condition 5 and 6. This is not possible with a map having a constant sign(so far negative) of second derivative. In other words, we needed a map which would shoot up again after the second minima. Since obtaining the equation for such a map, without



INPUT1	INPUT2	$f()$	OUTPUT
0	0	x0	0
0	1	x0+δ1	1
0	2	x0+δ2	2
1	1	x0+2δ1	2
1	2	x0+δ1+δ2	0
2	2	x0+2δ2	1

Figure 4.1: Analysis figure 1

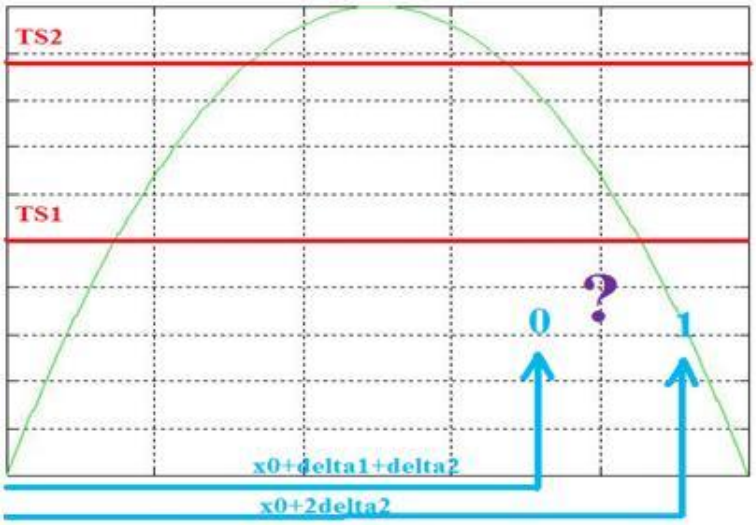


Figure 4.2: Analysis figure 2

discontinuities would be complex, we chose a piecewise linear map as shown in Fig 4.3. Having removed the constraints, it was now possible to implement the adder. The corresponding figure obtained for implementation is shown in Fig 4.4. It was obtained at a random choice derived from graphical analysis by choosing initial condition as 0, delta1=0.1, delta2=0.34, lower threshold=0.3 and upper threshold=0.6.

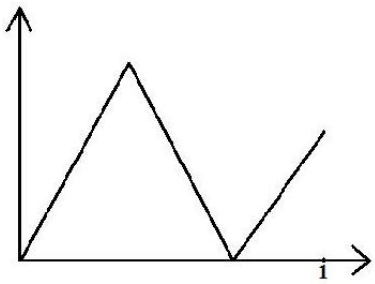


Figure 4.3: Modified piecewise linear map

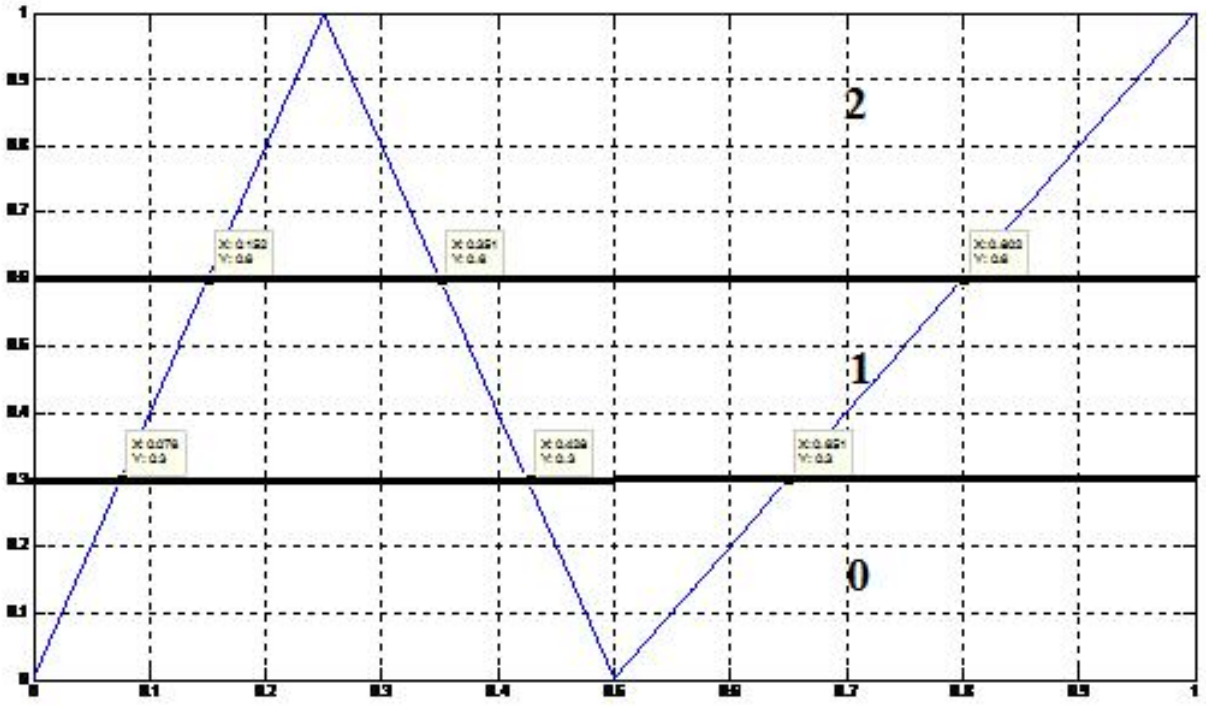


Figure 4.4: Result figure

Similarly, by choosing delta1 as 0.15, delta2 as 0.35, the other parameters remaining the same as above.

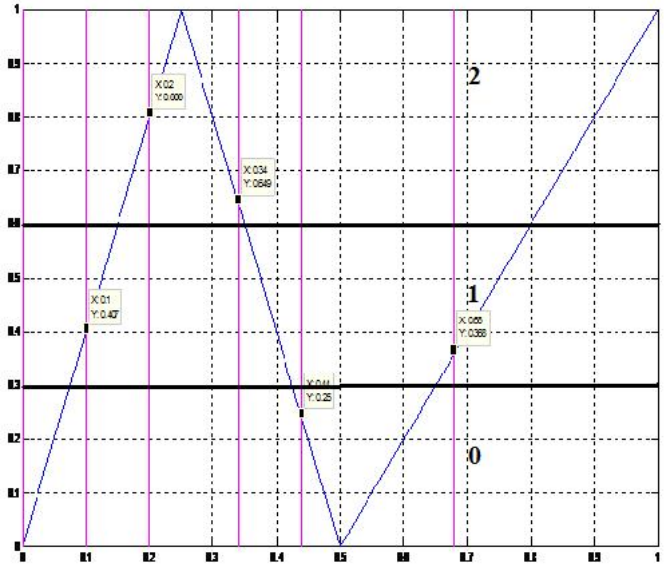


Figure 4.5: Result figure

Having obtained particular values for the adder, we now needed a generic solution or more precisely, a larger number of feasible regions. Hence, using the obtained values, we tried to accumulate more of these. This was done using the conventional method of sweeping used so far by fixing four parameters. The results obtained are tabulated below.

Trial no.	Initial condition (x0)	Delta1	Delta2	Lower threshold (T1)	Upper threshold (T2)
1	0	0.1	0.34	0.3	0.6
2	0	0.15	0.35	0.3	0.6

Figure 4.6: Graphically obtained solutions

1)Sweeping T1(40 values)

x0=0, delta1=0.15, delta2=0.35, T2=0.6

0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.1	0.11	0.12	0.13	0.14	0.15	0.16	0.17	0.18	0.19
0.2	0.21	0.22	0.23	0.24	0.25	0.26	0.27	0.28	0.29
0.3	0.31	0.32	0.33	0.34	0.35	0.36	0.37	0.38	0.39

Table 4.4: Possible values of T1

2)Sweeping T2

The only possible value of T2 for the same set of x0,delta1,delta2 and T1 is T2 = 0.6.

3)sweeping x0

Only x0 = 0

4)sweeping delta1

possible values of delta1 are:

0.0800 0.0900 0.1000 0.1100 0.1200 0.1300 0.1400

5)sweeping delta2

possible values of delta2 are:

0.3300 0.3400 0.3500

For the generality of the proposed piecewise linear map, it is not only enough to implement the adder, we need to show that different logic gates implemented in logistic and tent map can also be realized using this model.

The necessary and sufficient conditions to be satisfied by the modified piecewise linear map for the implementation of various logic operations are given below:

Operation	AND	OR	XOR
Conditions	$f(x_0) \leq T$ $f(x_0+\delta) \leq T$ $f(x_0+2\delta) > T$	$f(x_0) \leq T$ $f(x_0+\delta) > T$ $f(x_0+2\delta) > T$	$f(x_0) \leq T$ $f(x_0+\delta) > T$ $f(x_0+2\delta) \leq T$

Operation	NAND	XNOR	NOT
Conditions	$f(x_0) = \delta$ $f(x_0+\delta) > T$ $f(x_0+2\delta) \leq x^*$	$f(x_0) > T$ $f(x_0+(\delta)) <= T$ $f(x_0+(2\delta)) > T$	$f(x) > T$ $f(x_0+\delta) \leq T$

Table 4.5: Necessary and sufficient conditions to be satisfied by the chaotic element to implement each of the logical operations

A few examples were tried out and their corresponding initial condition, delta and threshold were tabulated as follows:

GATE	INITIAL CONDITION(x0)	DELTA	THRESHOLD(T)
AND	0	0.5	0.5
OR	0	0.15	0.5
NAND	0.2	0.1	0.5
NOR	0.3	0.1	0.5
XOR	0	0.25	0.5
XNOR	0.3	0.3	0.5
NOT	0.2	0.2	0.5

Table 4.6: The values of initial condition, threshold and delta for the implementation of logic gates using modified piecewise linear map

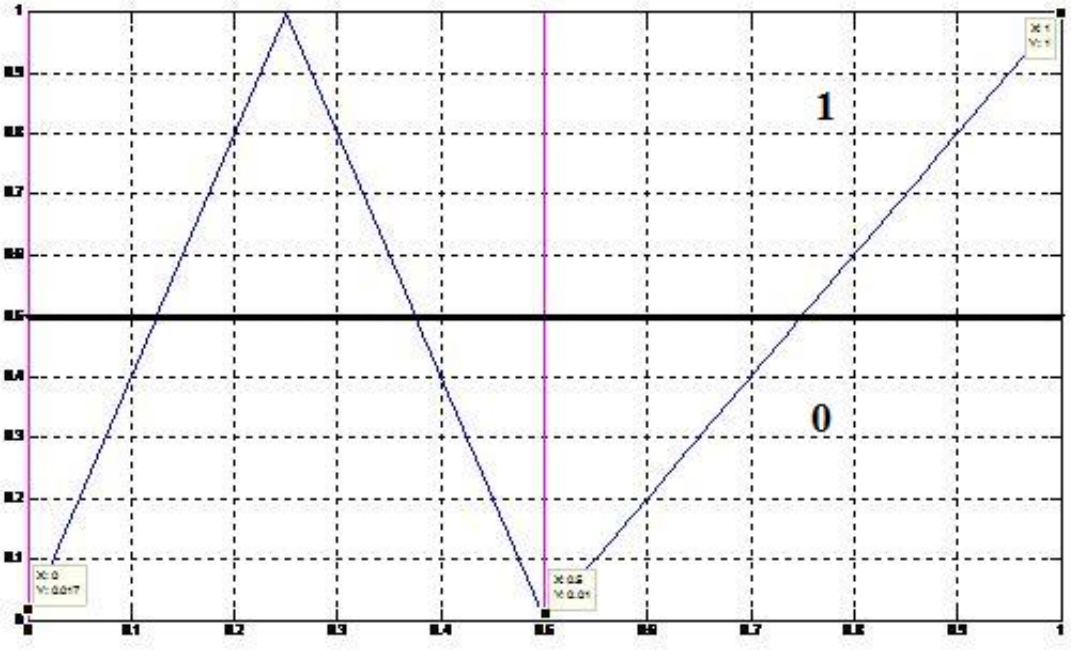


Figure 4.7: Implementation of AND gate using modified piecewise linear map

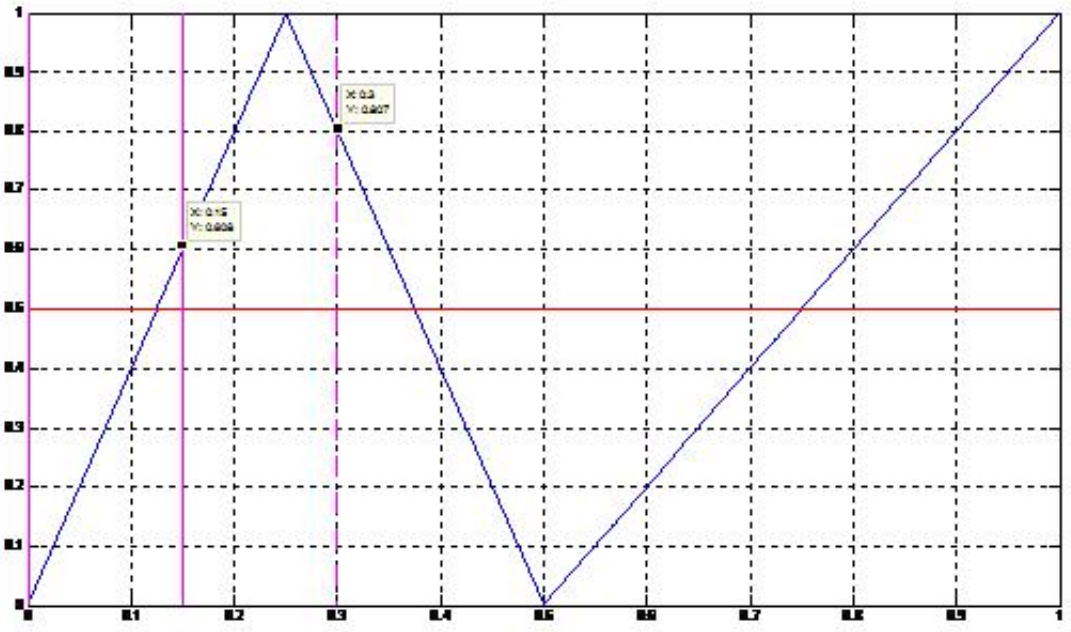


Figure 4.8: Implementation of OR gate using modified piecewise linear map

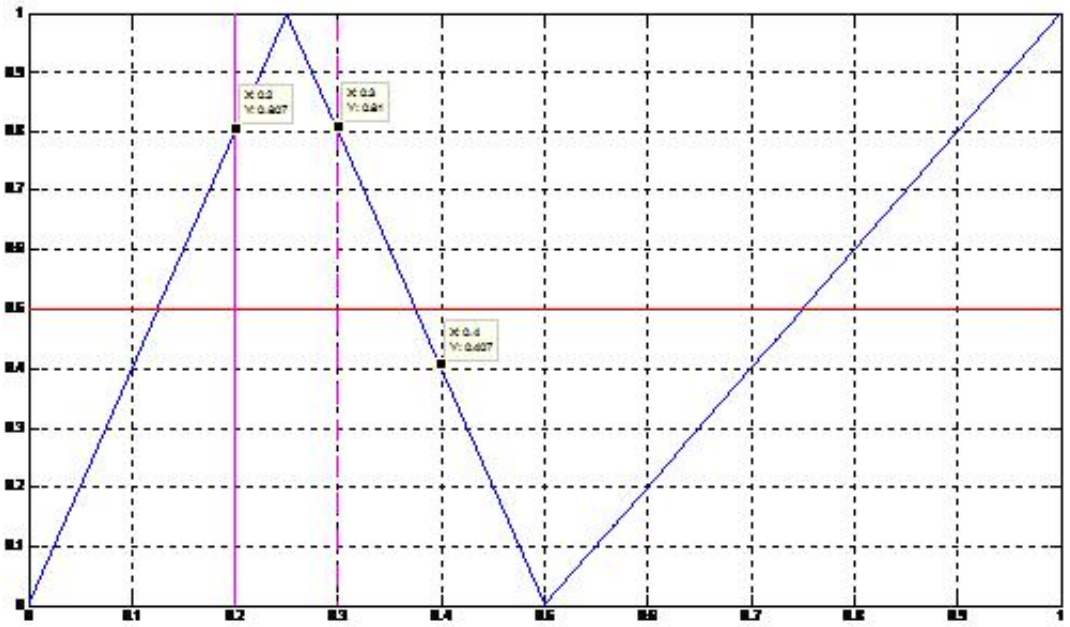


Figure 4.9: Implementation of NAND gate using modified piecewise linear map

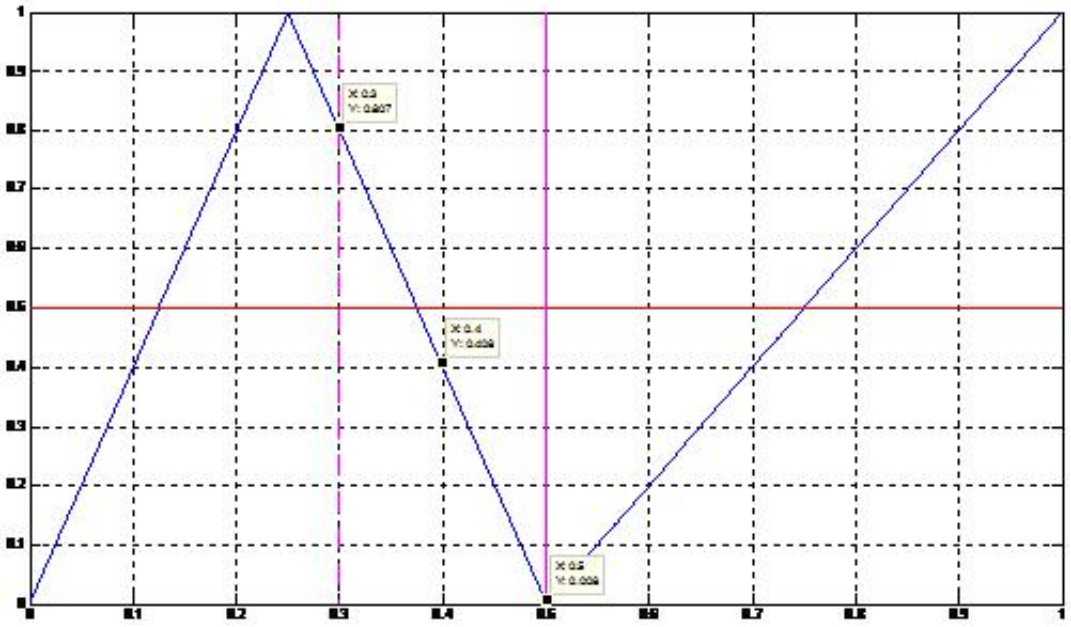


Figure 4.10: Implementation of NOR gate using modified piecewise linear map

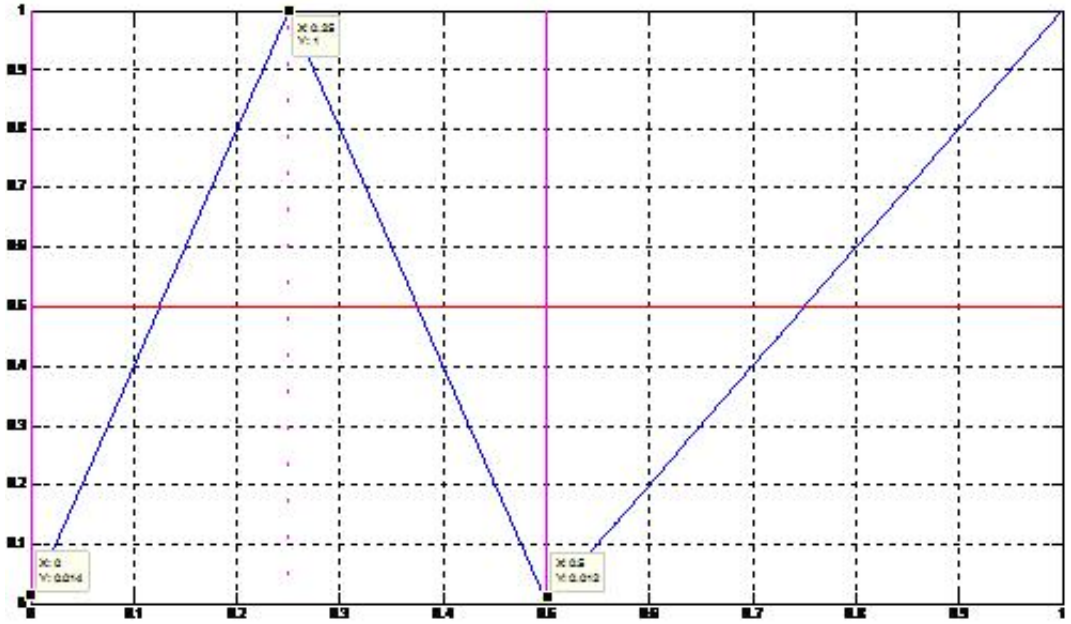


Figure 4.11: Implementation of XOR gate using modified piecewise linear map

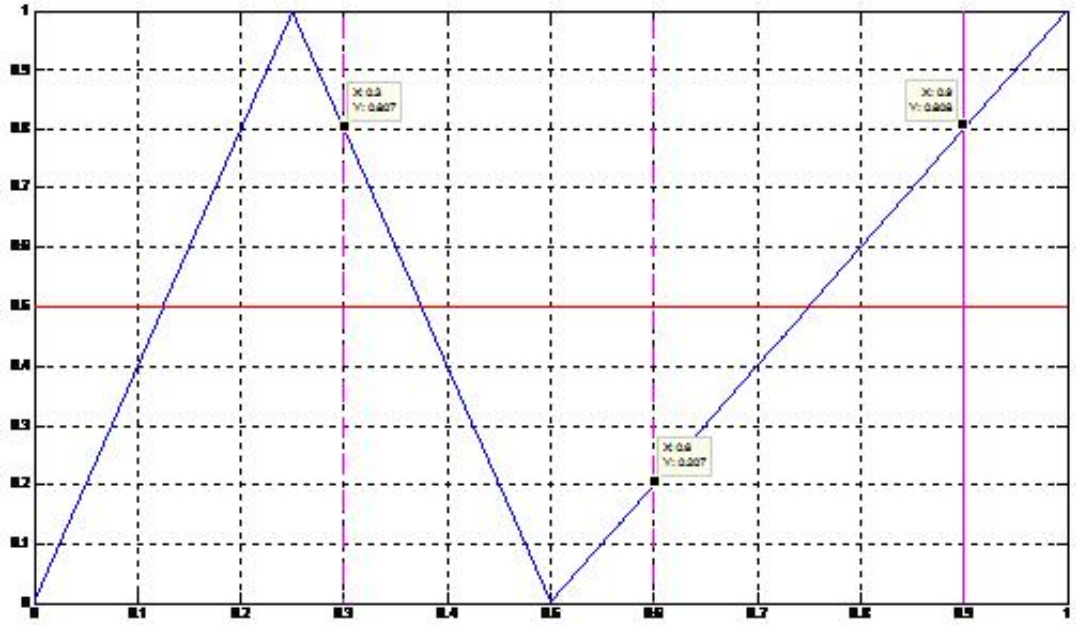


Figure 4.12: Implementation of XNOR gate using modified piecewise linear map

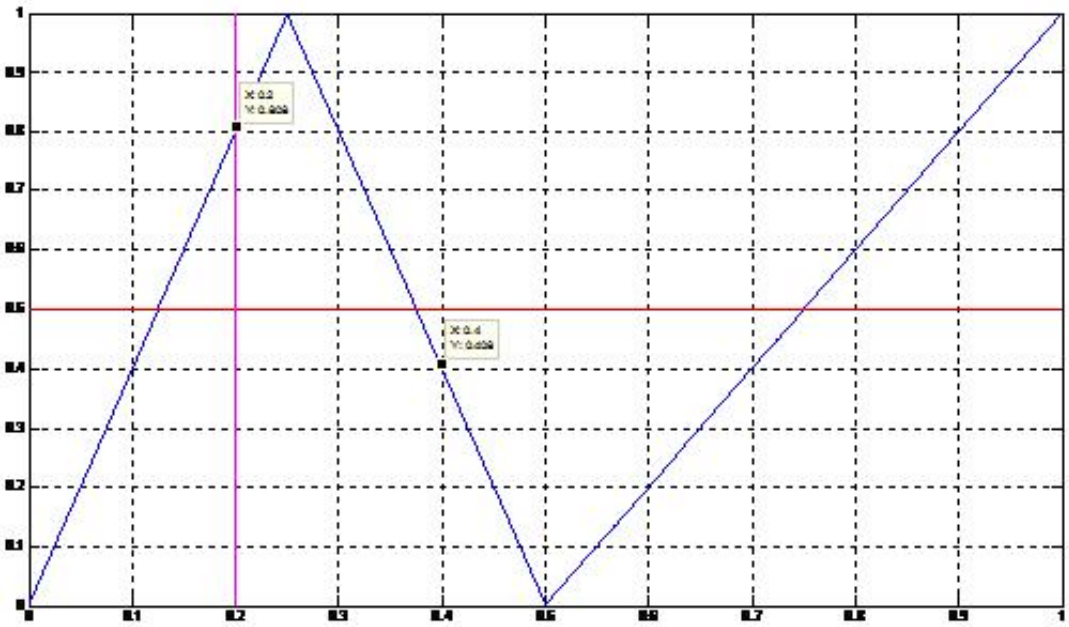


Figure 4.13: Implementation of NOT gate using modified piecewise linear map

Thus, now we finally have a model which can completely implement the logics proposed until now. Though more intricate maps would be necessary in expanding the arithmetic, the conclusions drawn above and the mode of analysis are sure to pave way for future enhancements.

CHAPTER 5
USING SYNCHRONIZATION TO OBTAIN DYNAMIC LOGIC GATES

The aim of chaotic computing is to use a single chaotic unit to emulate different logic gates and perform different arithmetic tasks, and further have the ability to switch easily between the different operational roles. The explicit chaos-computing schemes that have been designed and implemented so far was all been based on the thresholding method to achieve controlled responses from a chaotic system. We now move on to a very different scheme to implement dynamic logic gates: We will use the synchronization of a driver-and-response nonlinear system, to achieve logic operations.

Generally, synchronization can be considered as the appearance of some relations between functionals of two processes due to interactions. Complete synchronization, namely, the exact coincidence of the states of systems, is the strongest form of synchronization, and here we will base our scheme on this phenomenon. It may be noted that synchronization has been widely used as a basis for communication schemes. Now, in this Rapid Communication, we will demonstrate the use of synchronization to obtain dynamic logic gates.
The logic cell here is comprised of one-way coupled nonlinear systems. The logic output will be given by the synchronization error between the two systems comprising the cell(see the schematic in Fig 5.1)

In our circuit implementation we consider one-way coupled Chua circuit, discussed in introduction as the computing element. The corresponding circuit component values are $L = 18mH$, $R = 1710ohms$, $C1 = 10nF$, $C2 = 100nF$. Chua's diode parameters are $R1 = 220ohm$, $R2 = 220ohm$, $R3 = 2.2kohm$, $R4 = 22kohm$, $R5 = 22kohm$,$R3 = 3.3kohm$, and buffer=opamp AD712. Note that the circuit is the one-way coupling configuration of the classic Chua's circuit shown in fig 5.1 [5]

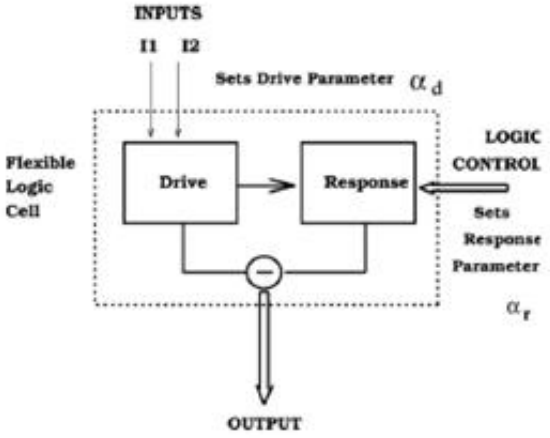


Figure 5.1: Synchronization using logic gates [5]

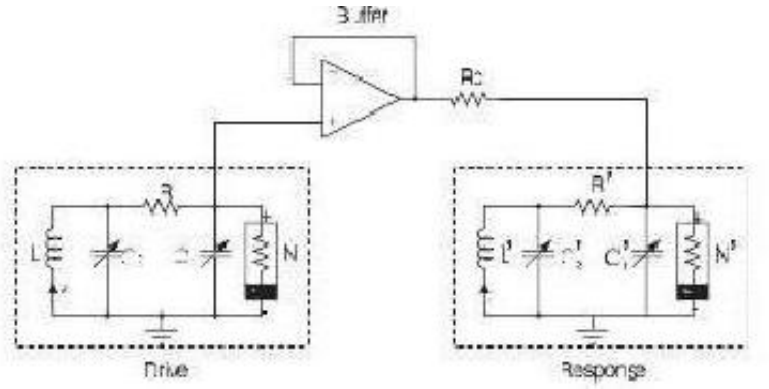


Figure 5.2: Chua circuit [5]

We varied the parameters α_r through 7 to 10 and α_d was assigned values 7, 8.5 and 10 corresponding to inputs 00, 01/10 and 11 respectively. Using Range-kutta method the differential equations were solved and synchronization error between x3 and x3' was found. On fixing threshold for synchronization error for different initial conditions, different logic gates could be implemented the condition for which is mentioned in the table.

ODE's for drive parameter:

$$\dot{x_1} = \alpha_d[x_2 - x_1 - g(x_1)] \tag{5.1}$$

$$\dot{x_2} = x_1 - x_2 + x_3 \tag{5.2}$$

$$\dot{x_3} = -\beta x_2 \tag{5.3}$$

ODE's for response parameter:

$$\dot{x_1}' = \alpha_r[x_2' - x_1' - g(x_1')] \tag{5.4}$$

$$\dot{x_2}' = x_1' - x_2' + x_3' \tag{5.5}$$

$$\dot{x_3}' = -\beta x_2' \tag{5.6}$$

Logic gate	Input set (I_1, I_2)	Drive parameter (Input set)	Response parameter (Logic control)	Synchronization error	Output
AND	00	α_1	σ_{NOR}~α_1	large	0
	01/10	α_2		large	0
	11	α_3		small	1
OR	00	α_1	σ_{NOR}~α_1	large	0
	01/10	α_2		small	1
	11	α_3		small	1
NAND	00	α_1	σ_{NOR}~α_1	small	1
	01/10	α_2		small	1
	11	α_3		large	0
NOR	00	α_1	σ_{NOR}~α_1	small	1
	01/10	α_2		large	0
	11	α_3		large	0
XOR	00	α_1	σ_{NOR}~α_1	large	0
	01/10	α_2		small	1
	11	α_3		large	0
XNOR	00	α_1	σ_{NOR}~α_1	small	1
	01/10	α_2		large	0
	11	α_3		small	1

Figure 5.3: Parameter setting of drive system yielding output of operation for six logic gates [5]

Synchronization error envelope for drive parameters: α_1=7 corresponding to input set (00) - square; α_2=8.5 corresponding to input set (01/10) -circle; α_3=10 corresponding to input set (11)-triangle.

Results:

Initial condition [x1 x2 x3]=[x1' x2' x3']=[-14 3 0]

Threshold for synchronization error: 0.25

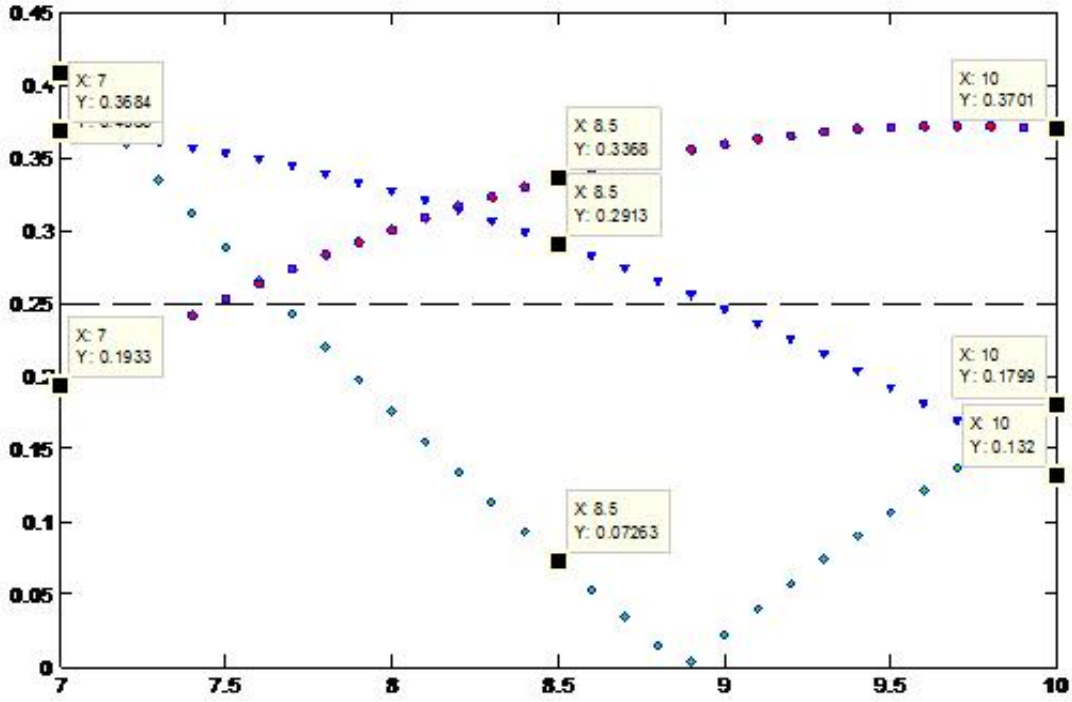


Figure 5.4: Result figure

At α_r= 7: XOR

For α_d=7(INPUT 00), $|X3 - X3'| > 0.25$ implies 0

For α_d=8.5(INPUT 10/01), $|X3 - X3'| < 0.25$ implies 1

For α_d=10(INPUT 11), $|X3 - X3'| > 0.25$ implies 0. Hence XOR.

At α_r =8.5: NOR

For α_d=7(INPUT 00), $|X3 - X3'| < 0.25$ implies 1

For α_d=8.5(INPUT 10/01), $|X3 - X3'| > 0.25$ implies 0

For α_d=10(INPUT 11), $|X3 - X3'| > 0.25$ implies 0. Hence NOR.

At α_r =10: XNOR

For α_d=7(INPUT 00), $|X3 - X3'| < 0.25$ implies 1

For α_d=8.5(INPUT 10/01), $|X3 - X3'| > 0.25$ implies 0

For α_d=10(INPUT 11), $|X3 - X3'| < 0.25$ implies 1. Hence XNOR.
Initial condition: [x1 x2 x3]=[x1' x2' x3']= [-14 0 0]
Threshold for synchronization error: 0.2

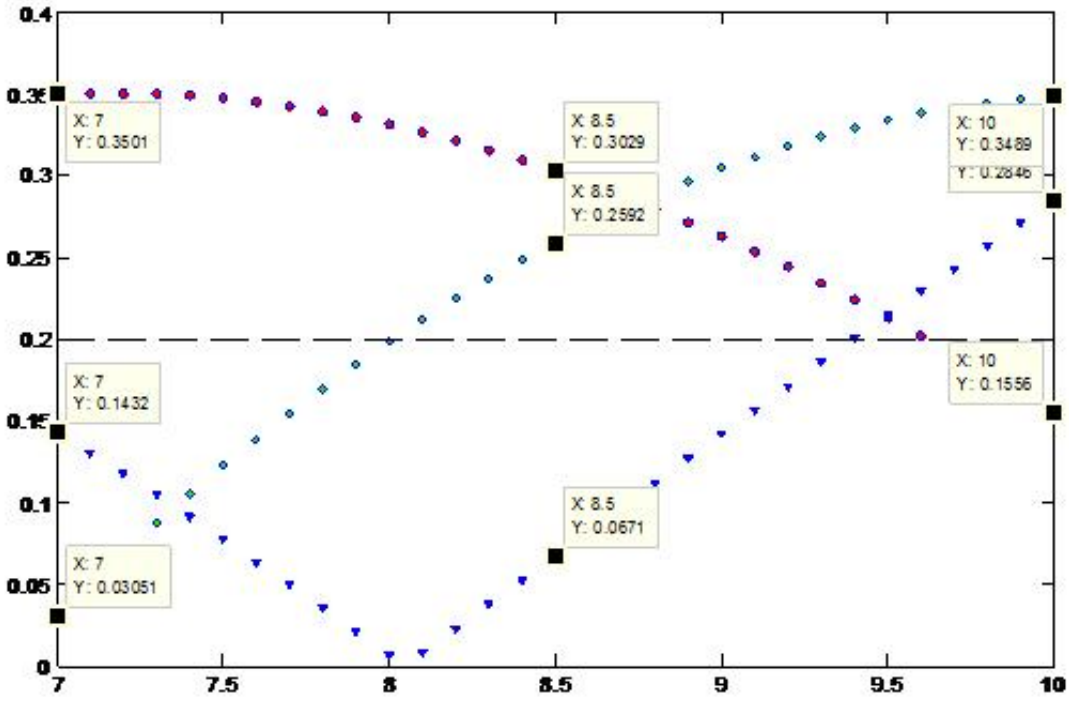


Figure 5.5: Result figure

At α_r= 7: XNOR
For α_d=7(INPUT 00), $|X3 - X3'| < 0.2$ implies 1
For α_d=8.5(INPUT 10/01), $|X3 - X3'| > 0.2$ implies 0
For α_d=10(INPUT 11), $|X3 - X3'| < 0.2$ implies 1. Hence XNOR.
At α_r =8.5: OR
For α_d=7(INPUT 00), $|X3 - X3'| > 0.2$ implies 0
For α_d=8.5(INPUT 10/01), $|X3 - X3'| > 0.2$ implies 0
For α_d=10(INPUT 11), $|X3 - X3'| < 0.2$ implies 1. Hence OR.
At α_r =10: XOR
For α_d=7(INPUT 00), $|X3 - X3'| > 0.2$ implies 0
For α_d=8.5(INPUT 10/01), $|X3 - X3'| < 0.2$ implies 1

For α_d=10(INPUT 11), $|X3 - X3'| > 0.2$ implies 0. Hence XOR.

Initial condition: [x1 x2 x3]=[x1' x2' x3']= [-14 0 3]

Threshold for synchronisation error: 0.15

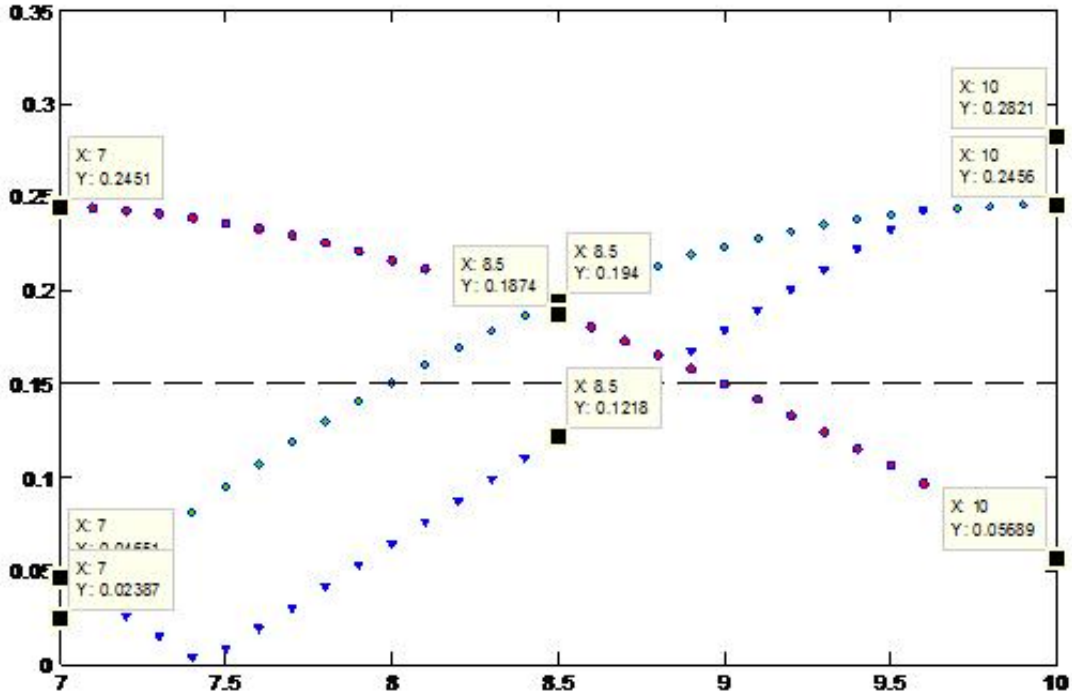


Figure 5.6: Result figure

At α_r= 7: XNOR

For α_d=7(INPUT 00), $|X3 - X3'| < 0.15$ implies 1

For α_d=8.5(INPUT 10/01), $|X3 - X3'| > 0.15$ implies 0

For α_d=10(INPUT 11), $|X3 - X3'| < 0.15$ implies 1. Hence XNOR.

At α_r =8.5: AND

For α_d=7(INPUT 00), $|X3 - X3'| > 0.15$ implies 0

For α_d=8.5(INPUT 10/01), $|X3 - X3'| > 0.15$ implies 0

For α_d=10(INPUT 11), $|X3 - X3'| < 0.15$ implies 1. Hence AND.

At α_r =10: (0 1 0) XOR

For α_d=7(INPUT 00), $|X3 - X3'| > 0.15$ implies 0

For α_d=8.5(INPUT 10/01), $|X3 - X3'| < 0.15$ implies 1

For α_d=10(INPUT 11), $|X3 - X3'| > 0.15$ implies 0. Hence XOR.

Note in all the case that for different values of α_r we are able to implement different logics. In the above cases, instead of choosing α_r in intervals of 1.5 we could choose any value. In this particular circuit, we could desirably vary it around 9. For instance consider the initial condition [-14 0 3] discussed above.

At $\alpha_r = 9.7$, fixing error threshold at 0.25, we have,

For $\alpha_d = 7$(INPUT 00), $|X3 - X3'| < 0.25$ implies 1

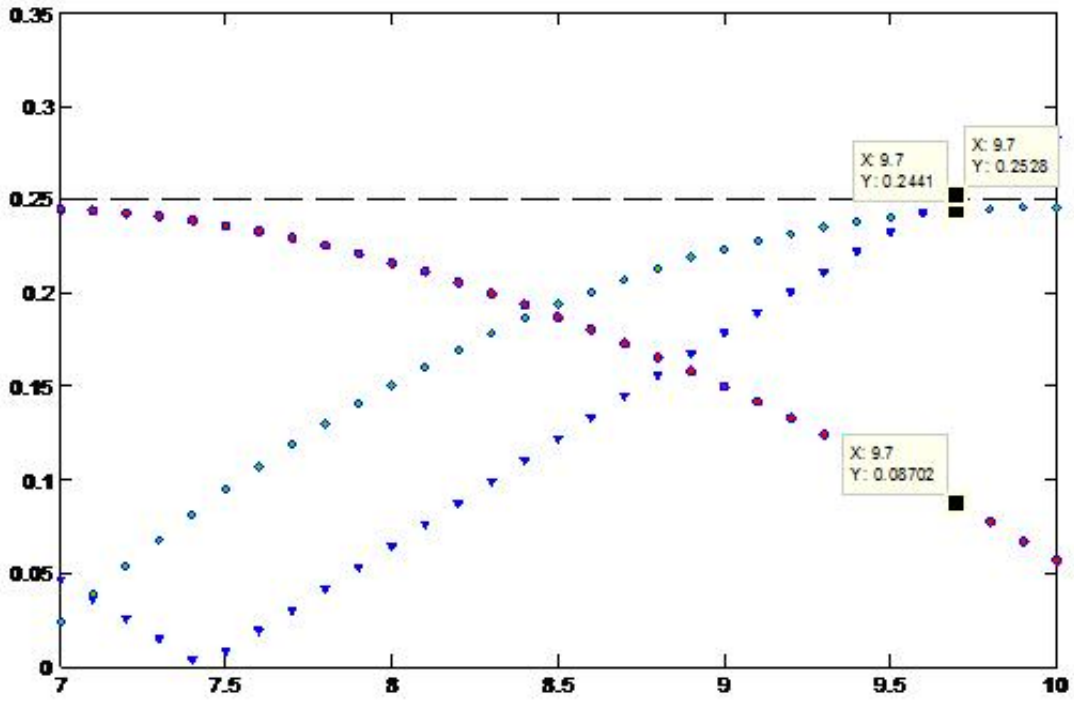


Figure 5.7: Result figure

For $\alpha_d = 8.5$(INPUT 10/01), $|X3 - X3'| < 0.25$ implies 1

For $\alpha_d = 10$(INPUT 11), $|X3 - X3'| > 0.25$ implies 0. Hence NAND logic is implemented.

If suppose for the same initial condition, error threshold is fixed at 0.2 and α_r=9,
For α_d=7(INPUT 00), $|X3 - X3'| > 0.2$ implies 0

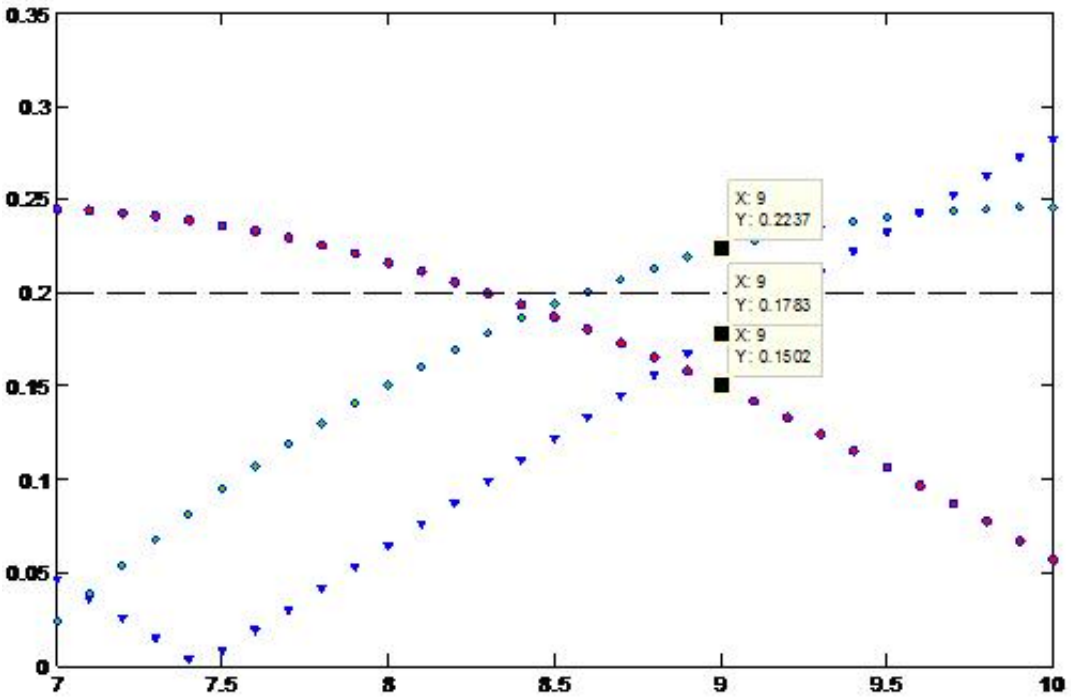


Figure 5.8: Result figure

For α_d=8.5(INPUT 10/01), $|X3 - X3'| < 0.25$ implies 1
For α_d=10(INPUT 11), $|X3 - X3'| < 0.25$ implies 1. Hence OR logic is implemented.

Yet again, at α_r=7, if error threshold is fixed at 0.04,

For α_d=7(INPUT 00), $|X3 - X3'| < 0.04$ implies 1

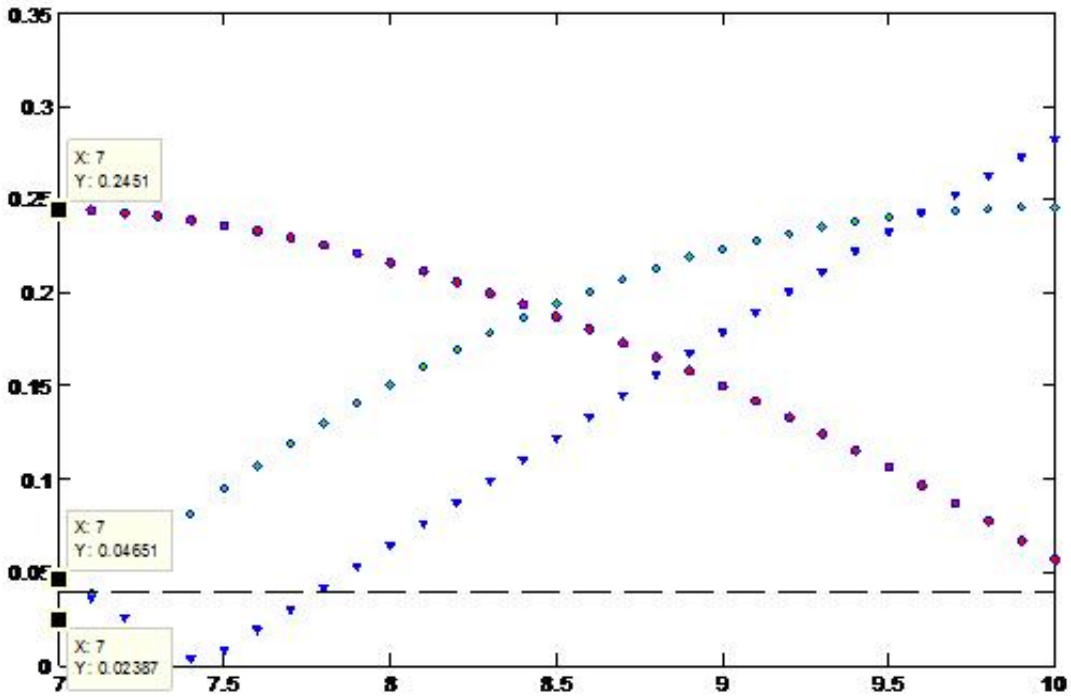


Figure 5.9: Result figure

For α_d=8.5(INPUT 10/01), $|X3 - X3'| > 0.04$ implies 0

For α_d=10(INPUT 11), $|X3 - X3'| > 0.04$ implies 0. Hence OR logic is implemented.

CONCLUSION

A recurring theme of research in chaotic systems over the last decade has been that chaos provides "flexibility" in the performance of natural systems and provides such systems with a rich variety of behaviours that can be utilized for more versatile performance. Our aim was to construct general multipurpose programmable hardware out of chaotic elements. Literature survey suggested that this would also mean increase in computational speeds through the exploitation of many dynamical states available to chaotic systems.

The results we obtained in chaos computing have shown that a single nonlinear dynamical system can (with proper tuning of parameters and control inputs) become any logic gate. We have been able to filter quite a number of parameter sets for AND, OR, NAND, NOR and XOR gates in three different maps and using techniques of single as well as multiple iteration.The gates where implemented using logistic map and topological tent map. Since modulo arithmetic could not be implemented using both, we introduced a piecewise linear map to implement it. This further emphasis the flexibility of the technique over hardwired control. In multiple-iteration one notes that varied temporal patterns embedded in the dynamical evolution of nonlinear systems are capable of performing sequences of logic operations in time. Thus minimal control is needed thereby we only invoke control mechanism on initialization, from there on we just monitor the state and the morphing between gates takes place in time evolution, instead of varying the control parameters. So one can set a global parameter and let time evolve the logic, rather than micromanage each morphing step through a separate parameter change.

Further, trying to implement the modulo arithmetic, the importance of precision and dependence on initial condition was very evident. Even though the constrains of modulo three arithmetic was apparent on analysis, considering the complex pattern of non linear systems, one cannot expect it to be so for higher modulo opera-

tions. This stresses on the requisite of a more general map for the operations. This presents wide latitude of further research area in the field.

Similarly, in the research we have been introduced to the potential of the method of synchronization. In particular, we have shown the direct and flexible implementation of the basic logic gates, using a single drive-response unit. Arrays of such logic cells may be reconfigured easily by a stream of logic-control parameter values to the response system. So such systems are readily programmable and architectures based on such logic implementations may serve as ingredients of a general purpose computing device, that who knows may in the next electronics revolution replace the existing statically wired hardware!

FUTURE ENHANCEMENT

Having found out a map to implement logic gates as well as mod three addition, it could be extended for higher modulo and a generic map to implement up to mod-n can be derived. Further, the condition for carry has to be added and thus expanding our model would give a complete full adder. This would be a great leap in chaotic computing since one can implement subtractor, modulo multiplication and division using such a model.

Also we have seen the enormous potential of the method of synchronization. It could be used to implement logics without the complexities of multiple iterations. In hardware implementation, it would be possible to switch between a wide range of logic by varying a lesser parameters than in case of thresholding technique.

With the help of faster computing techniques, there lies ahead a huge scope for chaotic computing. There is a vast amount of power of chaos to be harnessed.

LITERATURE CITED

[1] J. A. Y. K.T.Alligood, Tim D. Sauer, *Chaos: An Introduction to dynamical systems.*

[2] W. S.Sinha and T.Munakata, "Flexible parallel implementation of logic gates using chaotic elements."

[3] W. L. S. S. K.Murali, Abraham Miliotis, "Logic from nonlinear dynamical evolution."

[4] K. S. W.L.Ditto, A.Milliotis and M.L.Spano, "Chaogates:morphing logic gates that exploit dynamical patterns."

[5] S. S. K.Murali, "Using synchronization to obtain dynamic logic gates."

[6] T.Munakata and S.Sinha, "Implementation of fundamental logic gates by 1-d chaotic elements."

[7] W. T.Munakata, "Chaos computing:implementation of fundamental logic gates by chaotic elements."