Chaos Computing: ideas and implementations

BY WILLIAM L. DITTO\textsuperscript{1,2}†, K. MURALI\textsuperscript{3‡}, SUDESHNA SINHA\textsuperscript{4¶}

\textsuperscript{1}Department of Biomedical Engineering, University of Florida, Gainesville FL 32611-6131, USA
\textsuperscript{2}ChaoLogix, Inc. 101 S.E. 2nd Place, Suite 201 - A, Gainesville, FL 32601, USA
\textsuperscript{3}Department of Physics, Anna University, Chennai 600 025, India
\textsuperscript{4}The Institute of Mathematical Sciences, Taramani, Chennai 600 113, India

We review the concept of the “chaos computing” paradigm, which exploits the controlled richness of nonlinear dynamics to obtain flexible reconfigurable hardware. We demonstrate the idea with specific schemes and verify the schemes through proof-of-principle experiments.

Keywords: Chaos, computation, logic gates

1. Introduction

Recently there has been a new theoretical direction in harnessing the richness of nonlinear dynamics, namely the exploitation of chaos to do flexible computations (Sinha & Ditto, 1998; 1999). The aim is to use a single chaotic element to emulate different logic gates and perform different arithmetic tasks, and further have the ability to switch easily between the different operational roles. Such a computing unit may then allow a more dynamic computer architecture and serve as ingredients of a general-purpose device more flexible than statically wired hardware.

A system is capable of universal general purpose computing if it can emulate all logic gates. The necessary and sufficient components of computer architecture today are the logical AND, OR, NOT and XOR (Exclusive OR) operations, from which we can directly obtain basic operations like bit-by-bit addition and memory (Bartree, 1991; Mano, 1993). We first recall the theoretical scheme for flexible implementation of all these fundamental logical operations utilizing low dimensional chaos (Munakata et al, 2002; Sinha et al 2002a, 2002b), and then we give the specific realisation of the theory in a discrete-time and a continuous-time chaotic circuit (Murali, et al, 2005).

2. Concept

We wish to use the rich temporal patterns embedded in a nonlinear time series in a controlled manner to obtain a computing medium that is flexible and reconfigurable.

Now we outline a theoretical scheme for obtaining all basic logic gates with a single chaotic system. Consider a chaotic element (our chaotic chip or chaotic
Table 1. Necessary and sufficient conditions to be satisfied simultaneously by the nonlinear dynamical element, in order to be capable of flexibly implementing the logical operations AND, OR, XOR and NOT with the same computing module.

<table>
<thead>
<tr>
<th>Input Set ((I_1, I_2))</th>
<th>Output for AND Logic</th>
<th>Necessary and Sufficient Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0)</td>
<td>0</td>
<td>(f(x_0) \leq x^*)</td>
</tr>
<tr>
<td>(0,1) / (1,0)</td>
<td>0</td>
<td>(f(x_0 + \delta) \leq x^*)</td>
</tr>
<tr>
<td>(1,1)</td>
<td>1</td>
<td>(f(x_0 + 2\delta) - x^* = \delta)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Set ((I_1, I_2))</th>
<th>Output for OR Logic</th>
<th>Necessary and Sufficient Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0)</td>
<td>0</td>
<td>(f(x_0) \leq x^*)</td>
</tr>
<tr>
<td>(0,1) / (1,0)</td>
<td>1</td>
<td>(f(x_0 + \delta) - x^* = \delta)</td>
</tr>
<tr>
<td>(1,1)</td>
<td>1</td>
<td>(f(x_0 + 2\delta) - x^* = \delta)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Set ((I_1, I_2))</th>
<th>Output for XOR Logic</th>
<th>Necessary and Sufficient Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0)</td>
<td>0</td>
<td>(f(x_0) \leq x^*)</td>
</tr>
<tr>
<td>(0,1) / (1,0)</td>
<td>1</td>
<td>(f(x_0 + \delta) - x^* = \delta)</td>
</tr>
<tr>
<td>(1,1)</td>
<td>0</td>
<td>(f(x_0 + 2\delta) \leq x^*)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input Set ((I))</th>
<th>Output for NOT Logic</th>
<th>Necessary and Sufficient Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>(f(x_0) - x^* = \delta)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>(f(x_0 + \delta) \leq x^*)</td>
</tr>
</tbody>
</table>

In order to obtain all the desired input-output responses of the different gates,
we need to satisfy the conditions enumerated in Table 1 simultaneously. So given a
dynamics \( f(x) \) corresponding to the physical device in actual implementation, one
must find values of threshold and initial state satisfying the conditions derived from
the Truth Table to be implemented. For instance, the exact solutions of the initial
\( x_0 \) and threshold \( x^* \) which satisfy the conditions in Table 1 when
\[
f(x) = 4ax(1-x)
\]
with parameter \( a = 1 \), and the constant \( \delta = \frac{1}{4} \) (common to both input and output
and to all logical gates) is as follows: \( x_0 = 0 \) and \( x^* = 3/4 \) for the AND operation; \( x_0 = 1/8 \) and \( x^* = 11/16 \) for the OR operation; \( x_0 = 1/4 \) and \( x^* = 3/4 \) for the
XOR operation; \( x_0 = 1/2 \) and \( x^* = 3/4 \) for the NOT operation.

Contrast our use of chaotic elements with the possible use of periodic elements
on one hand, and random elements on the other. It is not possible to extract all
the different logic responses from the same element in case of periodic components,
as the temporal patterns are inherently very limited. So periodic elements do not
offer much flexibility or versatility. Random elements on the other end have many
different temporal sequences. But they are not deterministic and so one cannot
use them to design components. Only chaotic dynamics enjoys both richness of
temporal behavior as well as determinism. Here we have showed how one can select
out temporal responses corresponding to different logic gate patterns from such
dynamics, and this ability allows us to construct flexible hardware.

However note, that while nonlinearity is absolutely necessary for implementing
all the logic gates, chaos may not be always be necessary. In the representative
element of the logistic map presented here, solutions for all the gates exist only at
the fully chaotic limit of the logistic map. But the degree of nonlinearity necessary
for obtaining all the desired logic responses will depend on the system at hand
and on the specific scheme employed to obtain the input-output mapping. It may
happen that certain nonlinear systems will allow a wide range of logic responses
without actually being chaotic.

3. Proof-of-Principle Experiments

Discrete time Nonlinear System:

Here we will present an implementation of the theory discussed in the Section
above, and we will verify the theoretical solutions are indeed realizable in electronic
circuits (Murali et al, 2005). We use a threshold controller with threshold level set
by voltage level \( x_0 \). The logic level input \( I = I_1 + I_2 \) is added to \( x_0 \) and used as the
new input to the logistic map iteration to generate \( x_{n+1} \). This implements step 1
of the scheme. By using a another threshold reference level voltage signal \( x^* \), the
signal difference between \( x_{n+1} \) and \( x^* \) is monitored as the logic level output. This
constitutes the third (and final) step of the scheme.

In Fig. 1a the circuit realisation of the chaotic logistic map is depicted. In the
circuit implementation \( x_{n-1}, x_n \) and \( x_{n+1} \) denote voltages normalized by 10 V as
the unit. An analog multiplier IC AD633 is used as a squarer and it produces
the output voltage of \( x_n^2/10V \) for the given \( x_n \) as the input. By using suitable
scale changer, summing amplifier and an inverter the voltage proportional to \( x_{n+1} \)
is available at the output of op-amp (OA3) circuit. A variable resistor VR1 is
Figure 1. Circuit Implementation of (a) logistic map module and (b) the timing pulses T1 and T2 generated from the clock generator providing a delay of feedback. The output voltage of OA3 becomes a new input voltage to the multiplier AD633 after passing through two sample-and-hold circuits provided the terminals A and B are connected together. The sample-and-hold circuits are constructed with LF398 or ADG412 ICs and they are triggered by T1 and T2. See text for more details.

employed to control the parameter $a$ from 0 to 1 in the logistic map. The output voltage of OA3 becomes a new input voltage to the multiplier AD633 after passing through two sample-and-hold circuits (SH1 and SH2) provided the terminals A and B are connected together. The sample-and-hold circuits are constructed with LF398 or ADG412 ICs and they are triggered by suitable delayed timing pulses T1 and T2 (shown in Fig. 1b). The timing pulses are usually generated from the clock generator providing a delay of feedback and the delay is essential for obtaining the solution $x_{n+1}$ of the logistic map. Usually the clock rate of either 5 kHz or 10 kHz is used.

If the terminals A and B in Fig. 1a are connected to the respective terminals of the circuit of Fig. 2, we have the general circuit configuration for the flexible logic gate implementation. In the circuit, all input and output variables are again normalized by 10 V. The control circuit (dotted line box) is the threshold control
unit which generates the signal $x_0$ at terminal C corresponding to the input signal $x_{n+1}$ at A under the threshold control voltage $V_0$. The input voltage $I = 0V, 0.25V$ or $0.5V$ corresponding to different logic gates. Here $x^*$ is another reference threshold voltage being used to produce the difference voltage $\delta$ from the $x_{n+1}$ signal. This $\delta$ and the input signal $I$ determines the logic condition of the different gates. Here op-amps OA4 to OA9 are implemented with $\mu$A741 or AD712. The resistor $R = 100 k\Omega$ and diode $D = \text{IN4148 or IN34A}$.

Fig. 3 shows the timing sequences of the implementation of 2 representative gates. The output waveforms are generated with both PSPICE circuit simulations and also through hardware implementations. Fig. 4 shows the schematic of the circuit implementing the half adder and Fig. 5 shows its corresponding timing sequences. These waveforms are again straightforwardly obtained in both simulation and experiments.

Continuous-time Nonlinear System:

We now present a somewhat different scheme for obtaining logic responses from a continuous-time nonlinear system. Our processor is now a continuous time system described by the evolution equation $\frac{dx}{dt} = F(x, t)$, where $x = (x_1, x_2, \ldots x_N)$ are the state variables and $F$ is a nonlinear function. In this system we choose a variable, say $x_1$, to be thresholded. Whenever the value of this variable exceeds a threshold $E$ it resets to $E$, i.e. when $x_1 > E$ then (and only then) $x_1 = E$. 
Figure 3. Timing sequences of the OR gate implementation (left), from top to bottom: (i) First Input $I_1$, (ii) Second Input $I_2$, (iii) State after chaotic update $f(x)$, and (iv) Output obtained by thresholding; and timing sequences of the NOT gate implementation (right), from top to bottom: (i) Input $I$, (ii) State after chaotic update $f(x)$, and (iii) Output obtained by thresholding. (Accuracy within 5 mV).

Now the basic logic operation on a pair of inputs $I_1, I_2$ in this scheme simply involves the setting of an inputs-dependent threshold, namely the threshold voltage $E = V_C + I_1 + I_2$, where $V_C$ is the dynamic control signal determining the functionality of the processor. By switching the value of $V_C$ one can switch the logic operation being performed. Again $I_{1,2}$ has value 0 when logic input is 0 and has value $V_in$ when logic input is 1. So the threshold $E$ is equal to $V_C$ when logic inputs are $(0,0)$, $V_C + V_{in}$ when logic inputs are $(0,1)$ or $(1,0)$, and $V_C + 2V_{in}$ when logic inputs are $(1,1)$. The output is interpreted as logic output 0 if $x_1 \leq E$, i.e. the excess above threshold $V_0 = 0$. The logic output is 1 if $x_1 > E$, and the excess above threshold $V_0 = (x_1 - E) \sim V_{in}$. The schematic diagram of this method is displayed in Fig. 6.

Now for a NOR gate implementation ($V_C = V_{NOR}$) the following must hold true:

(i) when input set is $(0,0)$, output is 1, which implies that for threshold $E = V_{NOR}$, output $V_0 = (x_1 - E) \sim V_{in}$

(ii) when input set is $(0,1)$ or $(1,0)$, output is 0, which implies that for threshold $E = V_{NOR} + V_{in}$, $x_1 < E$ so that output $V_0 = 0$.

(iii) when input set is $(1,1)$, output is 0, which implies that for threshold $E = V_{NOR} + 2V_{in}$, $x_1 < E$ so that output $V_0 = 0$.

For a NAND gate ($V_C = V_{NAND}$) the following must hold true:
Figure 4. Schematic representation of the half adder implementation on inputs $I_1$ and $I_2$, with $\text{SUM} \equiv \text{XOR} (I_1, I_2)$ and $\text{CARRY} \equiv \text{AND} (I_1, I_2)$

(i) when input set is $(0, 0)$, output is 1, which implies that for threshold $E = V_{\text{NAND}}$, output $V_0 = (x_1 - E) \sim V_{\text{in}}$

(ii) when input set is $(0, 1)$ or $(1, 0)$, output is 1, which implies that for threshold $E = V_{\text{in}} + V_{\text{NAND}}$, output $V_0 = (x_1 - E) \sim V_{\text{in}}$

(iii) when input set is $(1, 1)$, output is 0, which implies that for threshold $E = V_{\text{NAND}} + 2V_{\text{in}}$, $x_1 < E$ so that output $V_0 = 0$.

In order to design a dynamic NOR/NAND gate one has to find values of $V_c$ that will satisfy all the above input-output associations in a robust and consistent manner.

In our specific implementation, as the computing element, we consider a realisation of the double scroll chaotic Chua’s attractor given by the following set of (rescaled) 3 coupled ODEs

$$
\dot{x}_1 = \alpha(x_2 - x_1 - g(x_1)) \quad (3.1) \\
\dot{x}_2 = x_1 - x_2 + x_3 \quad (3.2) \\
\dot{x}_3 = -\beta x_2 \quad (3.3)
$$

where $\alpha = 10$, and $\beta = 14.87$ and the piecewise linear function $g(x) = bx + \frac{1}{2}(a - b)(|x + 1| - |x - 1|)$ with $a = -1.27$ and $b = -0.68$. The corresponding circuit component values are: $L = 18mH$, $R = 1710\Omega$, $C_1 = 10nF$, $C_2 = 100nF$, $R_1 = 220\Omega$, $R_2 = 220\Omega$, $R_3 = 2.2k\Omega$, $R_4 = 22k\Omega$, $R_5 = 22k\Omega$, $R_6 = 3.3k\Omega$, $D = \text{IN4148}$, $B_1, B_2 = \text{Buffers}$, $\mu$A741. Note that the circuit we use is the ring structure configuration of the classic Chua’s circuit (Dimitriev, et al, 2001).
We use the $x_1$ variable, corresponding to voltage $V_1$ across capacitor $C_1$ for thresholding. In the experiment we implement minimal thresholding (shown in the dotted box in Fig. 8). Instead of demanding that the $x_1$ variable be reset to $E$ if it exceeds $E$ we only demand this in Eqn. 2. This has very easy implementation, as it avoids modifying the value of $x_1$ in the nonlinear element $g(x_1)$, which is harder to do. So then all we need to do is to implement $\dot{x}_2 = E - x_2 + x_3$ instead of Eqn. 2, when $x > E$, and there is no controlling action if $x \leq E$. In the circuit the voltage $V_T$ corresponds to $E$ (Murali et al, 2003).

The schematic diagram for the dynamic NOR/NAND gate implementation is depicted in Fig. 7. The actual circuit implementation of the dynamic NOR/NAND gate module based on Chua’s circuit is shown in Fig. 8.
In the representative example shown in Fig. 9, $V_{in} = 2V$. The NOR gate is realized around $V_C = 0V$. At this value of control signal, we have the following: for input $(0,0)$ the threshold level is at $0$, which yields $V_0 \sim 2V$; for inputs $(1,0)$ or $(0,1)$ the threshold level is at $0$, which yields $V_0 \sim 0V$; and for input $(1,1)$ the threshold level is at $2V$, which yields $V_0 = 0$ as the threshold is beyond the bounds of the chaotic attractor (see Fig. 9 for timing sequences). The NAND gate is realized around $V_C = -2V$. The control signal yields the following: for input $(0,0)$ the threshold level is at $-2V$, which yields $V_0 \sim 2V$; for inputs $(1,0)$ or $(0,1)$ the threshold level is at $2V$, which yields $V_0 \sim 2V$; and for input $(1,1)$ the threshold level is at $4V$, which yields $V_0 = 0$ (see Fig. 9 for timing sequences).

4. On-going VLSI Implementation

We are currently developing a VLSI implementation of chaotic computing in a demonstration integrated circuit chip. The demonstration chip has a parallel read/write interface to communicate with a microcontroller, with standard logic gates. The read/write interface responds to a range of addresses to give access to internal registers, and the internal registers will interface to the demonstration chaotic computing circuits.

For the demonstration we selected circuits that were based upon known experimental discrete component implementations and as such, the circuits are larger than is necessary in this first generation of chip. Currently, the TSMC 0.18 μm
process is the IC technology chosen for the development. This process was chosen to demonstrate that the chaotic elements work in smaller geometries, and the extra metal layers in this process will provide a margin of safety for any routing issues that might develop.

For our proof of concept on the VLSI chip a small ALU (Arithmetic Logic Unit) with three switchable functions; two arithmetic functions (adder, multiplier, divider, barrel shifter, or others) and one function of scratchpad memory is being implemented. The ALU switches between at least two arithmetic functions and a completely different function like a small FIFO (First-In, First-Out memory buffer). This experiment takes a significant step toward showing the possibilities for future configurable computing. The three functions are combined into a single logic array controlled through a microcontroller interface. The microcontroller can switch functions, and then write data to the interface, and read the results back from the interface. Fig. 10 shows the simplified representation of this experiment (Ditto, et al., 2006).

5. Conclusions

In summary, we have demonstrated the direct and flexible implementation of all the basic logic gates utilizing nonlinear dynamics. The richness of the dynamics allows us to reverse engineer and select out all the different gate responses from the same processor by simply setting suitable threshold levels. These threshold levels are known exactly from theory and thus available as a look-up table. Arrays of such logic gates can conceivably be programmed on the run (for instance, with a stream of threshold values being sent in by an external program) to be optimized for the task at hand. For instance they may serve flexibly as an arithmetic processing
6. References


Figure 10. Simplified schematic of the proof of concept VLSI implementation of an ALU which can switch between at least two arithmetic functions and a completely different function such as a small FIFO (First-In, First-Out memory buffer).