SIRENA: A CAD Environment for Behavioral Modeling and Simulation of VLSI Cellular Neural Network Chips

R. Carmona, I. García-Vargas, G. Liñán, R. Domínguez-Castro, S. Espejo and A. Rodríguez-Vázquez

Instituto de Microelectrónica de Sevilla-CNM-CSIC / Universidad de Sevilla
Edificio CICA, C/Tarfia sn, 41012-Sevilla, SPAIN
Phone #34 5 4239923, FAX #34 5 4231832

Abstract

This paper presents SIRENA, a CAD environment for the simulation and modeling of mixed-signal VLSI parallel processing chips based on Cellular Neural Networks. SIRENA includes capabilities for: a) the description of nominal and non-ideal operation of CNN analog circuitry at the behavioral level; b) performing realistic simulations of the transient evolution of physical CNNs including deviations due to second-order effects of the hardware; and, c) evaluating sensitivity figures, and realize noise and Montecarlo simulations in the time domain. These capabilities portray SIRENA as better suited for CNN chip development than algorithmic simulation packages (such as OpenSimulator, Sesame) or conventional Neural Networks simulators (RCS, GENESIS, SFINX), which are not oriented to the evaluation of hardware non-idealities. As compared to conventional electrical simulators (such as HSPICE or ELDO-FAS), SIRENA provides easier modeling of the hardware parasitics, a significant reduction in computation time, and similar accuracy levels. Consequently, iteration during the design procedure becomes possible, supporting decision making regarding design strategies and dimensioning. SIRENA has been developed using object-oriented programming techniques in C, and currently runs under the UNIX operating system and X-Windows framework. It employs a dedicated high-level hardware description language: DECEL, fitted to the description of non-idealities arising in CNN hardware. This language has been developed aiming generality, in the sense of making no restrictions on the network models that can be implemented. SIRENA is highly *modular and* composed of independent tools. This simplifies future expansions and improvements.

Front-page Footnotes:

1

2

1. This work has been partially funded by spanish CICYT under contract TIC96-1392-C02-02 (SIVA).
2. Research of Ricardo Carmona has been partially supported by IBERDROLA, S. A. under contract INDES-94/377

SIRENA: A CAD Environment for Behavioral Modeling and Simulation of VLSI Cellular Neural Network Chips

R. Carmona, I. García-Vargas, G. Liñán, R. Domínguez-Castro, S. Espejo and A. Rodríguez-Vázquez

I. INTRODUCTION

Cellular **N**eural **N**etworks (CNNs) are arrays of identical *nonlinear* dynamic processing units (*cells*), arranged on a regular grid where *direct interactions* among cells are limited to a finite *local* neighborhood. CNNs were first proposed by L.O. Chua and L. Yang in 1988 [1] and have an architecture similar to cellular automata, although differ in that interactions among cells are *analog*, and in the dynamic nature of the processing performed by the cells. A primary reason for the interest of CNNs is the existence of many computational and signal processing problems that can be formulated as well-defined tasks on signal values placed on regular 2-D and 3-D grids, and also with direct interactions among signals limited to local receptive fields -- directly mappable onto CNNs for their solution. Another reason is their local connection feature, which reports advantages for IC implementation as compared to fully interconnected neural network models. For instance the silicon area occupied by the processing units in Hopfield [2] network increases in proportion to N, the neuron count, while the area needed to interconnect the neurons, the routing area, increases to N^3. On the contrary, the routing area of CNN chips is commonly a negligible fraction of the neuron area and thus, enables much larger density of processing cells than featured by fully interconnected models. The implementation advantage is especially pertinent for the important class of *translation-invariant* CNNs, where all inner cells are identical, layout is very regular, and, since all cells have identical interaction weights, *programmability* issues can be incorporated without significant extra routing cost, by adding one control line per weight [3].

Although CNNs are specially well suited for high speed image processing tasks, their reported applications cover a much wider range of activities, such as motion detection, classification and recognition of objects, associative memory, optimization, solution of partial differential equations, statistical and nonlinear filtering, etc. [4]. Also, the recent extension towards the definition of a *programmable analogic array computer*, the CNN Universal Machine, has opened many new application fields which can be handled through spatial and temporal task sequencing controlled by a stored program [5]. A key feature of CNNs is their potential for high operation speed in the processing of array signals. However, this does not make manifest if CNNs are realized in the form of software on conventional computers, but only if they are realized as *VLSI chips*. Typical CNN chips may contain up to about 200 transistors per pixel (including sensory and processing devices) [6] [7]. On the other hand, practical applications require large enough grid sizes; around 100×100. Thus, CNN designers must confront complexity levels larger than 10^6 transistors; most of them operating in analog mode.

The simulation of CNN chips, and in general of analog parallel processing chips, is a major obstacle for their design. Simulations are needed to assess the influence of many hardware non-idealities for which analytical descriptions are intractable. In addition, simulation provides the most reliable way to assess manufacturability previous to tape submission. Unfortunately, current algorithmic simulation packages [8] and neural networks simulators [9] are unable to consider the non-ideal effects of real electronic circuits. SPICE-type electrical simulators are barely capable to handle more than about 10^5 transistors and may take several days CPU time on Sparc-10 workstations for circuits of about 10^4 transistors [10]. One approach to solve these problems, and hence to allow the incorporation of simulation into the design cycle of CNN chips, is to use *macromodels* for the SPICE-type electrical simulators. However, mapping the hardware non-idealities into circuit descriptions is not simple, and the simulator must still handle the whole network interconnection topology -- not very efficient. The approach adopted in SIRENA [11] overcomes these limitations by focusing on the simulation of the circuit behavior, instead of the circuit topology. SIRENA allows the definition and simulation of large neural networks with different levels of description. It operates onto a user-defined CNN model which can enclose either the pure algorithm description or a detailed characterization of the anomalies resulting from the integrated circuit implementation. In addition to a high simulation efficiency, SIRENA is equipped with a powerful and friendly graphical interface under the X-windows framework.

This paper reports a progressive insight of this environment, beginning with a review of the system structure, in which different parts of SIRENA are presented. Afterwards, Sect. III analyzes the modelling capabilities of the environment and illustrates the inclusion of non-ideal effects in the network model as a refinement of an algorithmic description and a crude first level transistor model towards a more detailed picture of the final microelectronic circuit. Several models based on different implementations of the CNN paradigm are developed and simulated. A comparison between SIRENA and HSPICE involving CPU time consumption and accuracy is presented. Sect. IV describes advanced features of SIRENA's simulator core: sensitivity and Montecarlo analysis; and sketches the application of these capabilities to the design process of VLSI CNN chips. Finally, brief conclusions are given.

II. ARCHITECTURE AND OPERATION OF SIRENA

a) *Description of the Basic CNN Behavior*

According to [4] CNNs are:

- multidimensional arrays defined on a *grid* and composed of,
- mainly identical *nonlinear, dynamical* processing units (*cells*), one per grid vertex, which satisfy two properties:
- all significant variables are continuous-valued, and,
- physical interconnections among cells are mostly local, i.e. most cells are physically connected only to other located within finite radii of the grid.

The set of all the cells in the network define the *grid domain GD*. The interconnection region for the generic *c-th* cell constitutes its *neighborhood* $N_r(c)$, which includes the center cell itself. We use r_c (neighborhood *radius*) for the radius of this interconnection region. Although the more general model contemplates a spatial variation of the radius, here we will assume without loss of generality that it is constant, so that $r_c = r \quad \forall c \in GD$ [3]. This means that the topology of CNNs becomes univocally defined by the *grid shape* (rectangular, hexagonal, pentagonal, etc.), the value of *r*, and the spatial boundary conditions. These latter affect to a set of border cells which define the *grid surrounding, GS*.

Let us now focus on the operation of CNNs. It is described by using three variables per cell:

- Cell *state*: x_c, which conveys cell energy information as a function of time[4].
- Cell *output*: $y_c(t)$, obtained from the cell state through a nonlinear transformation,

$$y = f(x_c) \tag{1}$$

- Cell *input*: u_c, representing external excitations.

CNNs are multidimensional signal processing devices whose inputs are the input vector $\mathbf{u} = \{u_c, \forall c \in GD\}$ and the vector of initial states $\mathbf{x}(0) = \{x_c(0), \forall c \in GD\}$, and whose outcome is represented by the vector of output variables $\mathbf{y} = \{y_c, \forall c \in GD\}$. For given input and topology, the processing performed by CNNs is determined by the following:

- An *evolution law,* described by ordinary differential equations (ODEs), finite-difference equations (FDEs), or a mixture of both. For instance, in a case where ODEs are involved,

$$\tau_c \frac{dx^c}{dt} = g[x^c(t)] + d_c + \sum_{d \in N_r(c)} \{a_{cd}(y_d, t) + b_{cd}(u_d, t)\} \qquad \forall c \in GD \tag{2}$$

- The states, $\mathbf{x}_s$, and inputs, $\mathbf{u}_s$, of the cells in the grid surrounding.

Some significant design parameters which control the operation performed by CNNs are the shapes of dissipative and output functions (i.e. *f(.)* and *g(.)*), the *offset* values d_c, and the

3. *r* can be set to the value of the largest neighborhood, and, then, smallest neighborhoods handled by assuming zero-valued contributions of the outer cells.
4. The time variable may either be continuous, represented by *t*, or discrete, represented by *n*. Signals in this latter case are valid only at discrete time instances, $t = nT$, where $n = 0, 1, 2, 3, \ldots$

nature and values of the interactions within each cell neighborhood, which are represented through the *feedback* ($a_{cd}(.)$) and *control* ($b_{cd}(.)$) contributions.

It is not this paper´s purpose to discuss in detail the signal processing capabilities of CNNs, neither to provide coverage of the mathematical implications (stability, convergence, etc.) of their nonlinear dynamics. Broader views can be encountered in [4] and the many references quoted there.

b) SIRENA Environment Architecture

SIRENA is a simulation environment for Cellular Neural Networks oriented towards VLSI implementation. It has been developed using C language and object-oriented programming techniques and currently runs under the UNIX operating system and the X-Windows framework. The main objective in the development of this simulation package has been to achieve a significant degree of generality, efficiency and modularity, in an attempt to overcome the limitations in the field of CNN design of currently available non-specific CAD tools.

Generality means, for a specific tool like this, to avoid any restriction on the model of the basic processing unit of the network that can be implemented for simulation. Any network based on a cellular structure in which interconnection between the basic cells is, somehow, restricted to a specific neighborhood (which could be as large as the network itself), and whose operation relays on some nonlinear dynamics taking place within the individual processors of the array, can be described and simulated by SIRENA. An specially developed high-level language (DECEL) has been developed to accomplish this. A careful object-oriented programming of the environment allows feasible operation with widely different CNN models without further modification or recompilation of the main tools source code. Diverse models for the connectivity operators (see eq. (2)) can be defined. The extent and shape of the neighborhood, the strength of the contributions, and the nature of the functions that perform the connection, can all be arbitrarily set by appropriate DECEL descriptions. Within the cell core, different nonlinear operators responsible for the output generation, and subsequently determining the network dynamics due to the feedback performed by the cells output (see eq. (1)), can be defined. Higher order dynamics are also achievable, since an indefinite set of state variables and differential (or finite-differences) equations can be implemented and simulated. These low restrictions on the model definition makes SIRENA an important development tool for cellular and nonlinear dynamics research. Besides, the readiness for progressive model refinement converts this environment in a useful tool for CNN-based IC design and VLSI CNN-based circuits simulation.

Efficiency of SIRENA is related to the form in which network models are described, compiled and linked to other components of the system. Because it is a specific software package, committed to the simulation of cellular networks governed by local interactions and nonlinear dynamics, the network description comprises the local features of the basic processing unit while the global architecture and topology is embedded within the simulator core. Regularity

and uniformity of CNNs emphasizes the operation in the local range --the large scale performance emerges from the cooperative behavior of the individual processors. A hierarchical description of the network can be done, leading to smaller models and, consequently, requiring less resources and simulation time. Another key for efficiency, also available in SPICE-like simulators, is the arbitrary complexity of the model. From a pure algorithmic and theoretical sketch of the circuit to a highly detailed description, with a deep analysis of parasitics and higher order dynamics, DECEL offers the same simple and systematic approach that allows progressive insight on the network behavior based upon model refinement. For a particular study, some phenomena can be intentionally highlighted while some neglected.

Besides, *modularity* of the system resides on the independence of its principal components. Each of the three main tools (Fig. 1), namely: the Model Generator (GMS), the Simulator Core (NSS) and the Graphical User Interface (GUI), has been developed independently within the production of the whole project. Communication between them is performed with the help of an especially developed library of functions which establishes the necessary links and coordinates data interchange. These characteristics confer SIRENA workable expansion and improvement based upon this primal version of the system executable code. In other words, future versions or improved capabilities of any of the components can be realized and implemented without rethinking, reorganizing and recompiling the rest of the system. Integration of this environment within a higher structure, a comprehensive design framework, can be achieved without major difficulties. Finally, it will be an interesting study to address interaction between SIRENA and different simulation tools, concerning model and data exchange and compatibility.

Therefore, SIRENA makes use of a user-defined network description that is properly handled and compiled into a usable format by the model generation program, GMS. A network description file, written in DECEL, is fed into GMS and the proper linkage to the NSS is performed (Fig. 1). Specific auxiliary files describing network components and behavior, to be used during simulation processes, are generated. On the other hand, data files, written by the user as well, contain the information concerning the particular characteristics of the experiment, *i. e.* size of the network, input data, parameter values, etc. These files together are employed by the NSS to perform the simulation. The simulation process is initialized and configured from the GUI, via one of its tools (XINSS), or with the help of a simulation script in which integration method, convergence criterion, type and timing of the analysis are set by commands understandable by the Simulator Core. This command-like input to the NSS allows control of a simulation or a set of simulations by user-defined scripts. Outputs are appended to an output file that can be simultaneously parsed by the GUI (XEVMS) to visualize the evolution of the network simulation. The format of the output can be set in the simulation script before the execution time or modified after the simulation by any of the graphic format translation tools.

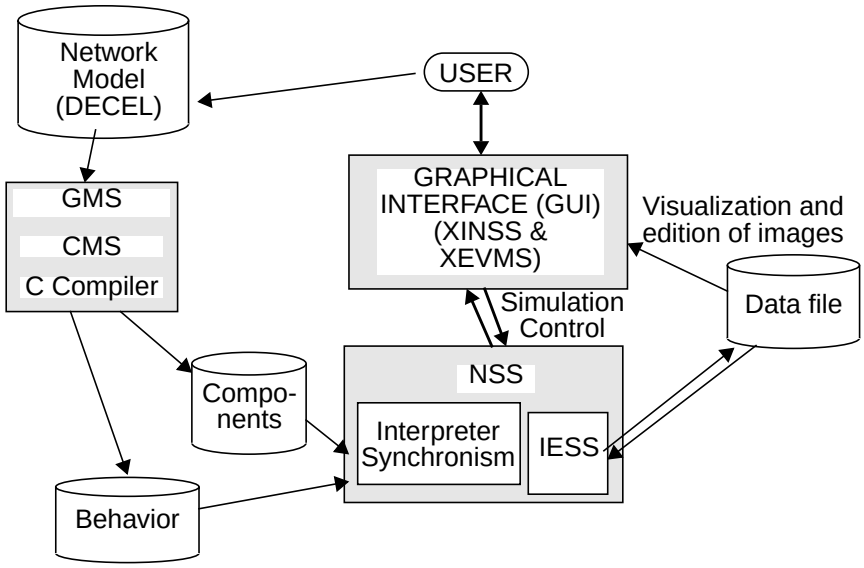


Fig.1: Conceptual block diagram of SIRENA.

c) SIRENA Model Generator (GMS)

The first step involved in the simulation of a CNN in SIRENA is the definition and creation of the network model. A high-level programming language, called DECEL, has been developed to realize this job. Let us use the example in Fig. 2 to illustrate the procedure for defining and compiling a CNN model to be handled by SIRENA tools. Any DECEL description file (**chua.cs** in the example) is composed of three parts: variables declaration, evolution section and network definition. In the first part every magnitude implicated in the CNN behavior is itemized indicating a type from a pre-defined set, namely: `matrix`, `template`, `bound`, `scalar`. This is necessary for an appropriate memory space management. After that, the `evolution` section states the relations between the declared variables. Equations stating these relations make use of the extensive C-language mathematical libraries. Besides, special operators have been conceived for nonlinearities implementation, for instance, some logical operators allow the introduction of piece-wise defined functions. Also connectivity and dynamic coupling with the neighborhood can be expressed in terms of DECEL statements. That includes formal clauses expressing connection operators, template elements, boundary conditions, etc. In this stage, different phenomena derived from a specific implementation of the algorithm can be addressed and included in the network model. Once the static part of the evolution section is fulfilled, listing explicit expressions for different variables as a function of some others, the dynamics must be detailed. Hence, the CNN model is defined as a system of first order differential equations (or finite differences equations, in the case of discrete-time networks). Higher order dynamics can be defined by a recursive method of sustitution. Therefore this 2nd-order differential equation:

$$\ddot{y} = a\dot{y} + by + cx + d \tag{3}$$

ought to be written as:

$$\left.\begin{aligned} \dot{z} &= az + by + cx + d \\ \dot{y} &= z \end{aligned}\right\} \tag{4}$$

and this same fashion applies for a higher order differential equation, supporting the simulation of *nth*-order dynamics. As depicted in the example (Fig. 2), several DECEL operators have been used to complete the network evolution section. Although some are self-explanatory, like `derivative` operator, some other may need further illustration. In the first place, connection evaluator (><) is found. This operator relates a template-type variable with a matrix-type one and performs multiplication of each template element with the neighborhood of each of the elements of the matrix. The boundary conditions evaluator (#) helps evaluating the connection of elements in the border of the matrix. Conditional operator (`:`) is employed in piece-wise defined functions, like the piece-wise linear function generating the cell output in the example. Finally, a one-layer network is defined in the `net` section. This section will resemble the same format in most of the cases, when working with single-layer CNNs. It will be more complex when concerning multilayered architectures. More details about the syntax of DECEL will be given in Sect. III, while exhibiting modeling capabilities of the environment.

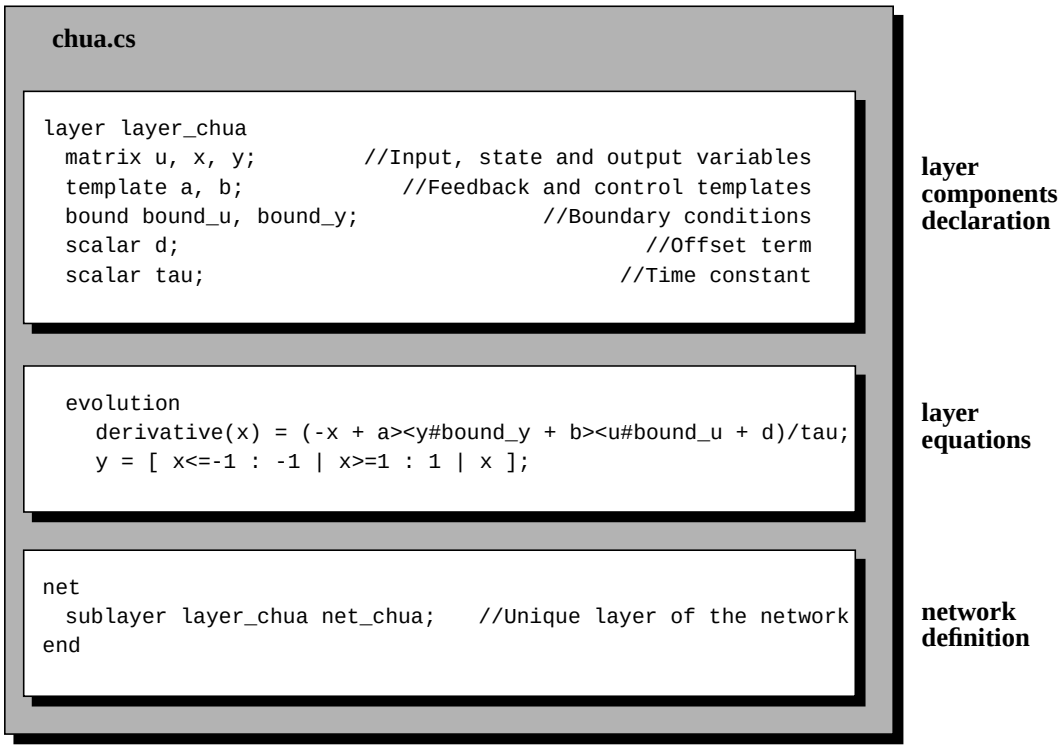


Fig.2: DECEL network description file sample.

Once the DECEL description of the network has been made, it is compiled and linked to

the simulator core for further operation. Two files are generated, **chua.des** and **chua.mro** in the example, which contain network components and behavior, respectively. It is important to mention that no specific values for the network variables are included in the model definition and compilation stages. Separation between functional description of the model and instance-values assignments permits the development of more general CNN models and the construction of a model library that can be extensively utilized by non-experienced in DECEL users. On the other hand, skilled user may program the CNN model directly in C language, avoiding any kind of limitations imposed by DECEL syntax.

d) The Simulator Core (NSS)

This tool is responsible for the numerical integration of the system of nonlinear differential equations describing the CNN behavior (eq. (2)). Before running the simulation process, SIRENA's simulator core (NSS) must be properly set up. It can be done interactively via the graphical user interface (GUI) or using a script file. Let us analyze the format of this file and the commands involved in the simulation environment initialization (Fig. 3). The same steps ought to be followed when using the GUI. First of all, the network model is specified loading the corresponding model description file, described in the previous section. After that, a few commands select the integration method, and give values to its associated parameters. A couple of integration algorithms are actually programmed, namely fourth-order Runge-Kutta with fixed time-step, and with an automatic time-step control. Other integration algorithms as Forward-Euler, Backward-Euler, Trapezoidal and Gear methods can be easily incorporated and included in future versions of SIRENA. The choice of Fourth-order Runge-Kutta (eq. (5) and (6) below), as the first integration method implemented in this tool, is motivated by the proven efficiency of this algorithm in the integration of this kind of systems of differential equations. A comparison between this method and Explicit Euler and Predictor-Corrector algorithms is made in [12].

$$f^c(\mathbf{x}(t)) = \tau\frac{dx^c}{dt} \qquad x^c(t_{n+1}) = x^c(t_n) + \int_{t_n}^{t_{n+1}} f^c(\mathbf{x}(t))dt \tag{5}$$

$$\int_{t_n}^{t_{n+1}} f^c(\mathbf{x}(t))dt = \frac{k_1^c}{6} + \frac{k_2^c}{3} + \frac{k_3^c}{3} + \frac{k_4^c}{6}$$

$$k_1^c = \Delta t \cdot f^c(\mathbf{x}(t_n))$$
$$k_2^c = \Delta t \cdot f^c\left(\mathbf{x}(t_n) + \frac{1}{2}\mathbf{k}_1\right)$$
$$k_3^c = \Delta t \cdot f^c\left(\mathbf{x}(t_n) + \frac{1}{2}\mathbf{k}_2\right) \tag{6}$$
$$k_4^c = \Delta t \cdot f^c\left(\mathbf{x}(t_n) + \frac{1}{2}\mathbf{k}_3\right)$$

The next step is the selection of the absolute or relative criterion for the convergence of

the CNN simulation. After the selection, parameter values are assigned and particularized for the simulation. In these conditions, a network has converged to a final state if the relative increment for each time-unit of its variables between two consecutive iterations is less than a certain predetermined value ε^k (7). This causes the simulation to stop and the final results are available.

$$\left|\frac{\Delta x^k}{max(x^k, min(\mathbf{x})) \cdot \Delta t}\right| < \varepsilon^k \qquad \Delta t \geq incrtmin \tag{7}$$

The rest of the script file is committed to the initialization of the network model variables. Each of the components of the CNN model is loaded from a specified file, in which values are given to the previously declared variables (Fig. 4). Therefore, a specific experiment, with specific network size and parameter values, is defined.

chua.com

```
loadnet chua.des
```

load network model

```
sel integ hf rkcp
coef pmax=1e-5 errmin=1e-3 pini=1e-6 eps=0.001
```

select integration method

```
sel conv hf conv_rel
coef eps=0.001 min=1e+6 incrtmin=1e-6
```

select convergence criterion

```
input hf.a<chua.a hf.in
input hf.b<chua.b hf.in
input hf.d<chua.d hf.in
input hf.u<chua.u hf.in
input hf.x<chua.x hf.in
input hf.boundy<chua.boundy hf.in
input hf.boundu<chua.boundu hf.in
input hf.tau<chua.tau hf.in

defoutput hf  out_hf std
output out_hf hf.y
result hf out_hf
```

I/O files and format

```
simulate hf tmax 5e-5
```

Simulation command

Fig.3: NSS script file example

The output files and format of these data are selected next and, finally, the simulation is

launched by the `simulate` command. In some special cases (those considering interaction between different networks) a special "synchronization" code must be specified. NSS can be run either in the foreground with an interactive command shell, in background from the UNIX shell, or activated from the X-Windows graphical user interface.

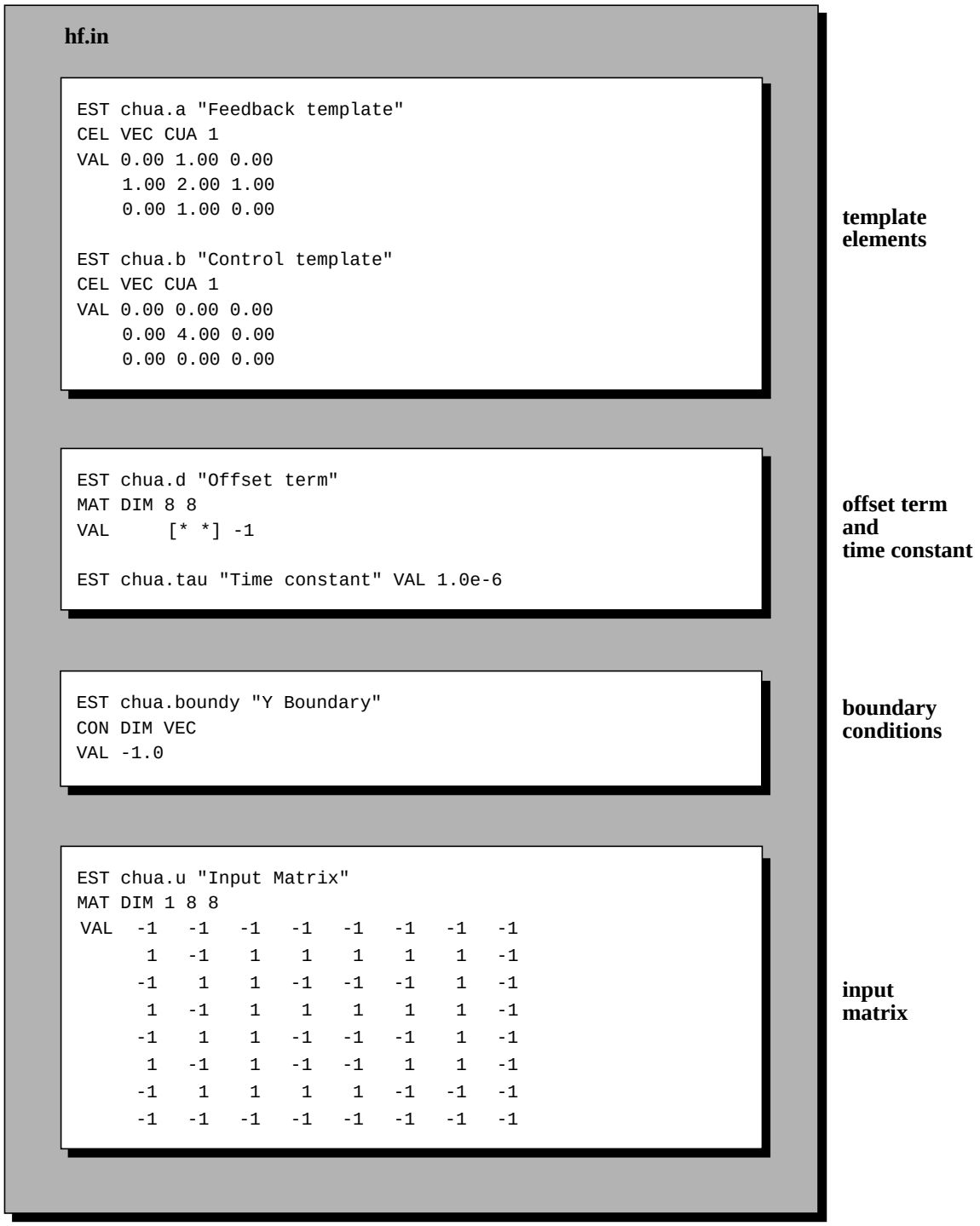


Fig.4: NSS input data file example

e) The Graphical User Interface

Communication between the user and SIRENA's simulator core can be realized in three different modes. Running in the foreground, using an interactive command shell for simulation configuration and control, or in background with the help of a script file. In either case, output visualization and input edition are restricted to the manipulation capabilities of other tools within the mainframe. In order to improve user interaction with the simulation control, inputs and outputs, a Graphical User Interface (GUI) for SIRENA was developed, in parallel to the NSS, under the X-Windows framework. The GUI is a collection of graphical tools and libraries which provides user-friendly communication with the simulator core. It allows the control and supervision of the NSS performance, facilitates input and output visualization and edition, and permits data exchange with other widespread graphical tools running under UNIX operating system. Fig. 5 gives a conceptual view of the GUI and its components interactions.

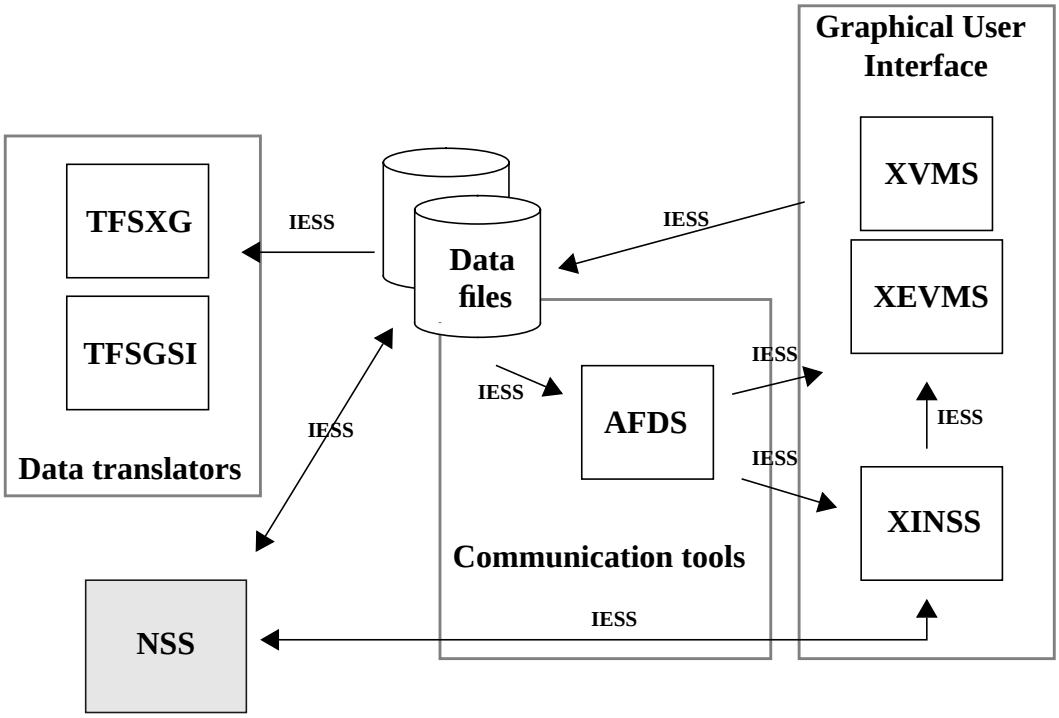


Fig.5: Conceptual block diagram of the Graphical User Interface.

These tools can be classified into three groups: communication tools, data translators and the properly called GUI. In the first class:

- IESS: an I/O functions library that manages data flow and exchange between the rest of the tools constituting the whole SIRENA package. Actually it is not accessed by a common user but it becomes a necessary reference for environment developers.
- AFDS: data files parser, prints error messages if syntax errors are found inside input data files avoiding NSS and visualization tools faults derived from mistaken input processing.

Data translators allow data formatting to make use of different graphic tools (Fig. 6). As SIRENA's output viewer is intended to visualize matrices in a color or gray scale, these routines make possible the representation of waveforms of any of the network variables.

- TFSXG: converts SIRENA output files, that can be ASCII or binary coded, to *xgraph* (*xvgr* compatible, used in Fig. 6.a) manageable files.
- TFSGSI: converts SIRENA output files to *gsi* [10] (*hsplot* compatible, Fig. 6. b) compatible format.

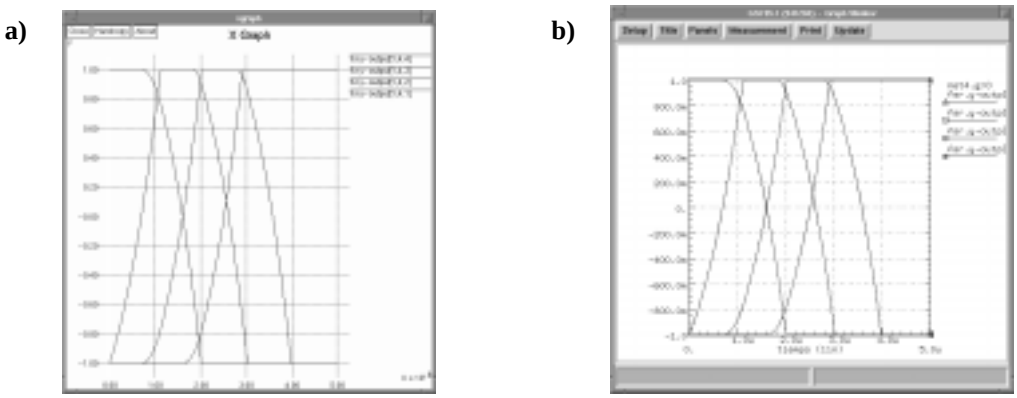


Fig.6: Waveform of network outputs visualization with a) *xgraph* and b) *gsi*.

Finally, the X-Windows based tools conform the graphic front end from which the user can control interactively the whole simulation process:

- XINSS: graphical interface with the simulator core of SIRENA under the X-Windows system. Provides interactive access to the simulation control and output, allowing visualization of results during the simulation and configuration-parameters edition.

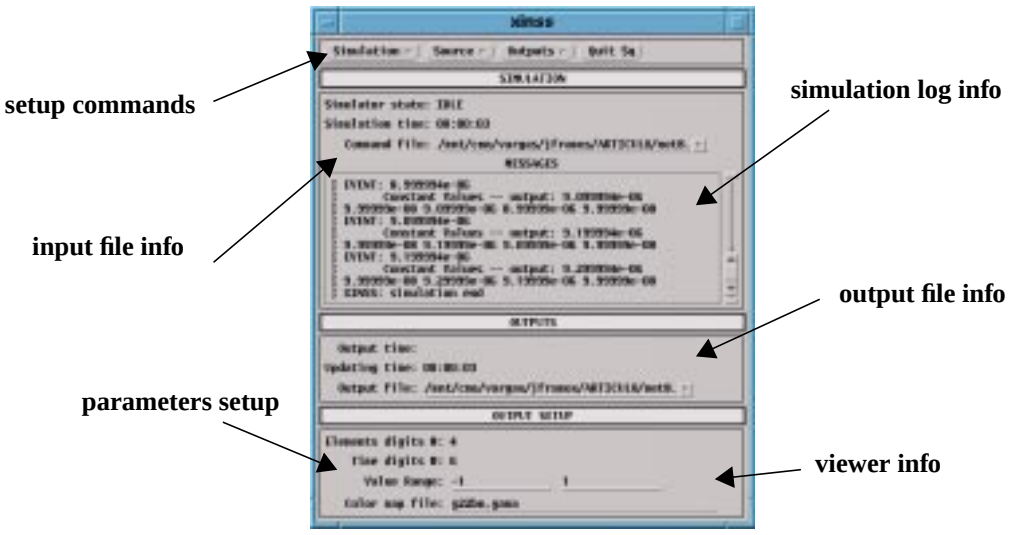


Fig.7: XINSS: graphical user interface with NSS

- XVMS: matrix viewer, allows image visualization and image-sequences animation. With the help of a command file (ASCII text format), XINSS starts the simulation (Fig. 7), and permits output monitoring using XVMS while it is running, as well as on-line simulation parameters modification. Fig. 8 shows the transient analysis of a 8×8 cells network for connected components detection as seen with XVMS.

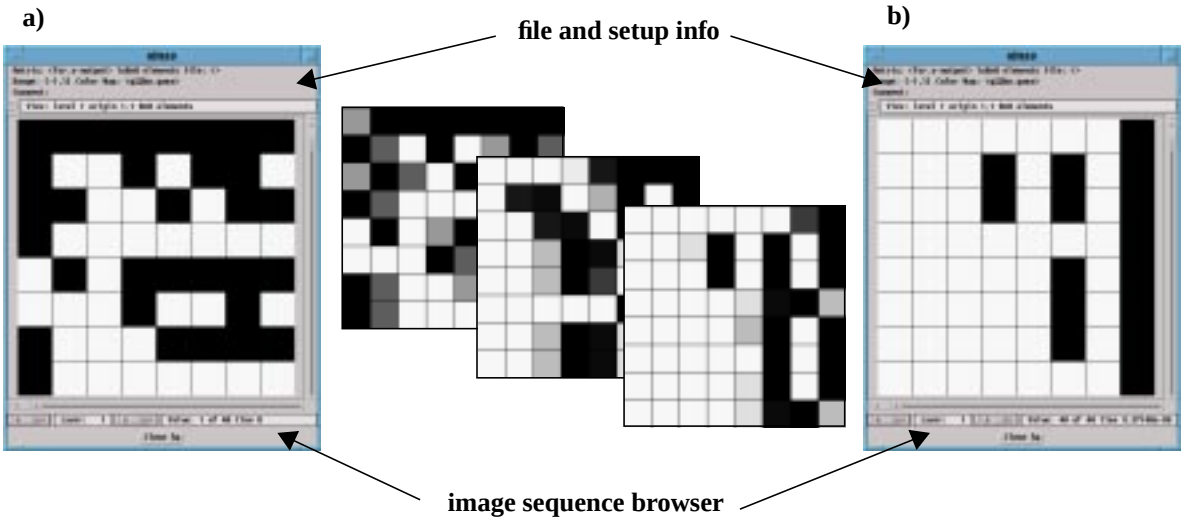


Fig.8: XVMS: Network input image (a) and output pattern (b) with a set of intermediate states in the simulation sequence.

- XEVMS: matrix editor, enhanced version of the visualizer with capabilities for the creation and modification of images, as well as import/export features (Fig. 9).

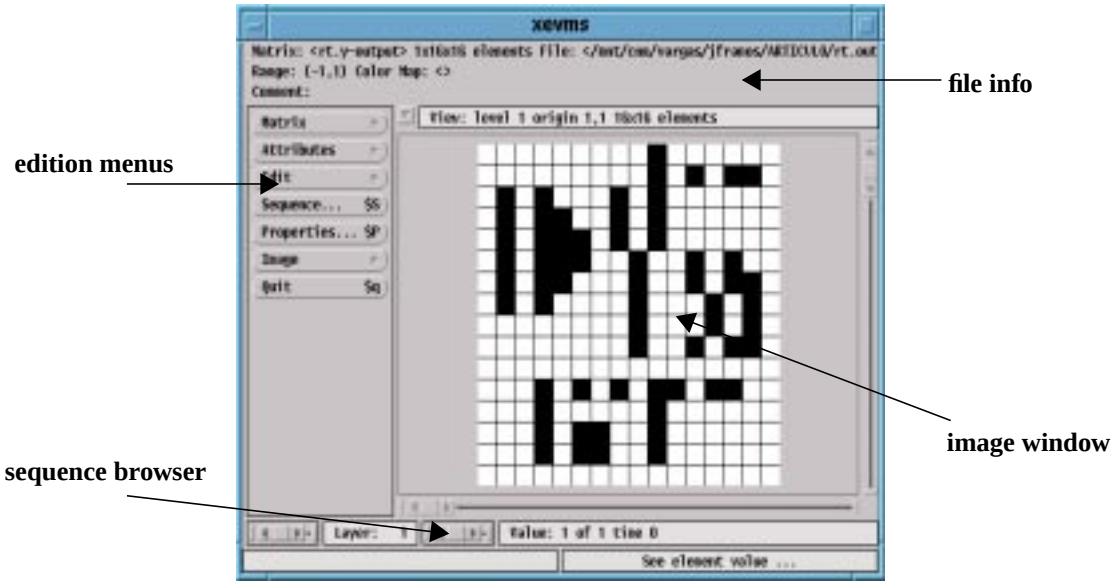


Fig.9: Input edition with the extended matrix viewer (XEVMS).

III. CNN MODELLING WITH SIRENA

Development of the CNN paradigm (2) concerning aspects of the electronic implementation requires efficient CAD tools based on arbitrarily detailed descriptions of the network. Basic algorithms and also low level physical effects must be representable. SIRENA makes use of an especially developed high-level programming language DECEL. It is used to specify the components --network constants and variables, and behavior of the cells --time evolution and components bindings. This results in a model that can range from the pure algorithmic level to a highly detailed description, with a vast number of non ideal characteristics accounting for predictable effects in electronic implementations (impedance coupling amongst input and output nodes of the cells, undesired nonlinearities, parasitic elements effects, non-uniformity caused by devices mismatch, etc.). SIRENA considers a CNN as a set of layers which evolve in an independent but coordinated manner and interact in determined time instants by transferring some variables value. Layer equations and variables are extracted from the DECEL description of the CNN model. The whole network definition is stated by the declaration of all the layers constituting the complete CNN and a synchronization code. This outlines simulation timing schedule and data interchange between layers in multi-layered networks and multi-network systems simulation. In other words, DECEL network definition begins at the bottom part of the hierarchy, stating components and behavior of each elementary processor. Then, layer variables and equations come out from the collection of the components and behavior of each individual cell, and finally, an assembly of synchronized (if required) layers conforms the top level --network definition. Let us describe some examples with different CNN implementations to illustrate the modeling capabilities of SIRENA.

a) Chua and Yang original CNN model

The original model proposed by Chua and Yang [1] describes time evolution and neighborhood coupling of each individual cell (c) within the grid domain (GD) in terms of a network time-constant (τ), a radius of vicinity (r), feedback and control templates (a_d^c and b_d^c) that weight the influence of the neighbors' input and output variables (u^d and y^d respectively) and an offset term (d^c) (Fig. 10). Cell state is the integral of a sum of weighted contributions from the coupled neighboring processors, an offset term and a losses term, and cell output is a sigmoidal non-linear function of the state, as expressed by the following equations:

$$\tau \frac{dx^c}{dt} = -x^c + \sum_{d \in N_r(c)} \{ a_d^c y^d + b_d^c u^d \} + d^c \qquad \forall c \in GD \tag{8}$$

$$y^c = f(x^c) = \begin{cases} -1 & \text{if} \quad x^c < -1 \\ x^c & \text{if} \quad |x^c| \le 1 \\ 1 & \text{if} \quad x^c > 1 \end{cases} \tag{9}$$

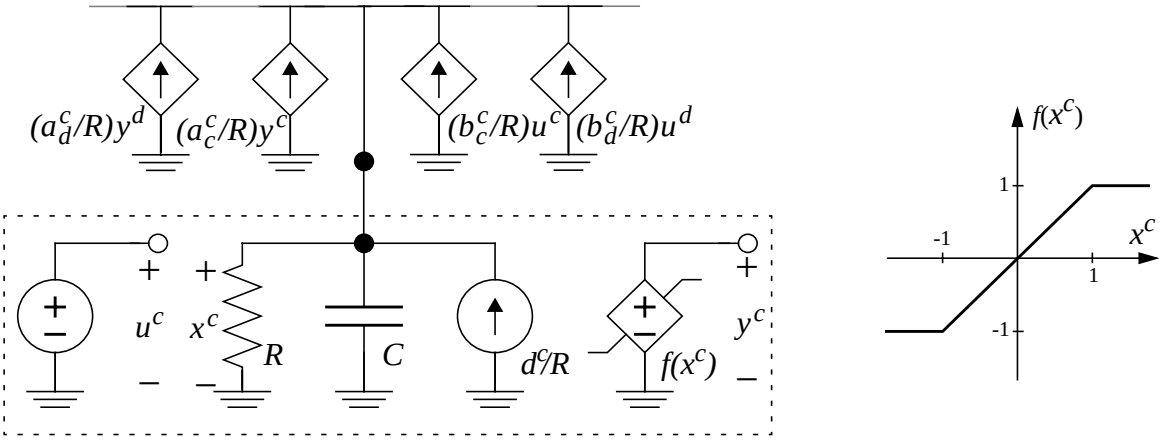


Fig.10: CNN model originally proposed by Chua and Yang.

Some modified versions of this initial model introduce new features and algorithm extension, like nonlinear or delay-type templates [13], or discrete-time emulations [14]. Other works report a model oriented towards VLSI implementation [15].

DECEL definition of this ideal model (Fig. 11) begins with variable declaration. After that, layer equations must be specified. This section is just a translation of the mathematical statements (8) and (9) into DECEL syntax. Finally a network with one layer is declared:

chua.cs

```
layer chua
  matrix u, x, y;         //Input, state and output variables
  template a, b;             //Feedback and control templates
  bound bound_u, bound_y;               //Boundary conditions
  scalar d;                                    //Offset term
  scalar tau;                                 //Time constant
```

layer components declaration

```
  evolution
    derivative(x) = (-x + a><y#bound_y + b><u#bound_u + d)/tau;
    y = [ x<=-1 : -1 | x>=1 : 1 | x ];
```

layer equations

```
net
  sublayer chua chua1;           //Unique layer of the network
end
```

network definition

Fig.11: DECEL description of the Chua-Yang CNN model.

This DECEL description of the CNN model is now compiled and the components and behavior files are linked to the NSS for further utilization. As referred before, no information about parameters value, nor even about network dimensions is contained inside the model description. Instance values assignments, as well as network sizing, is done in different input file, giving the model a large degree of generality. Any network of any size using any set of templates [16] for this CNN model can be simulated with this DECEL description, without recompiling the sources.

This ideal CNN model has been implemented in HSPICE (v. 95.1) using ideal elements and piece-wise-linear voltage-controlled current sources, and also in SIRENA. Several simulations have been run for different network sizes, ranging from 4 (arranged in a 2×2 matrix) to 1024 cells (32×32 matrix), in a Sun Microsystems SPARC Server 1000E with 4 CPUs and 512 Mb RAM. The employed templates performed Connected Component Detection of binary input patterns [17]. The information generated by the system when running these simulation processes under UNIX reports a certain advantage for SIRENA concerning the use of the system resources. The following graph (Fig. 12) illustrates CPU time consumptions by SIRENA and HSPICE. Fig. 13 shows the waveform plots of the cell state variable (x^c) and the cell output variable (y^c) generated by HSPICE and SIRENA for the simulation of a 4×4 CNN based on Chua-Yang model. *R* and *C* have been chosen to have a time constant of 1μs.

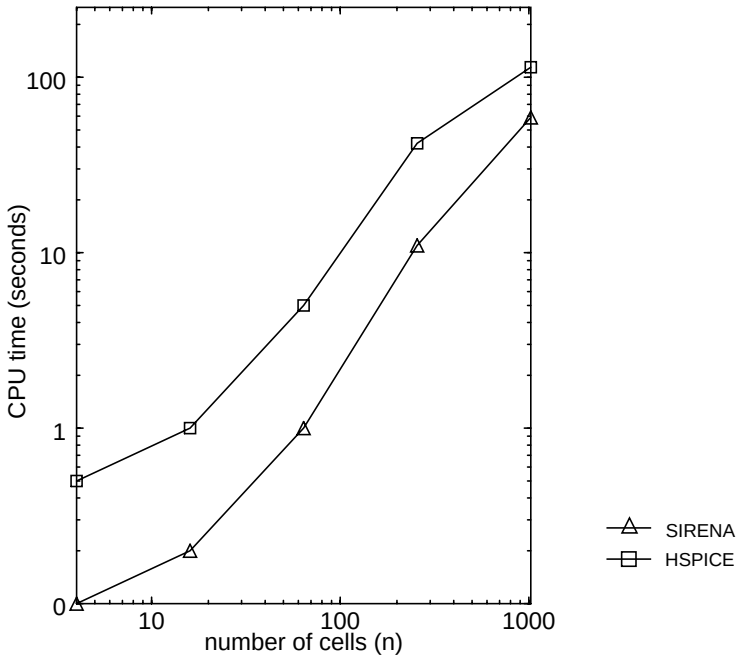


Fig.12: Log-log graphs of the CPU time consumed by HSPICE (95.1) and SIRENA simulations vs. the number of cells in the network.

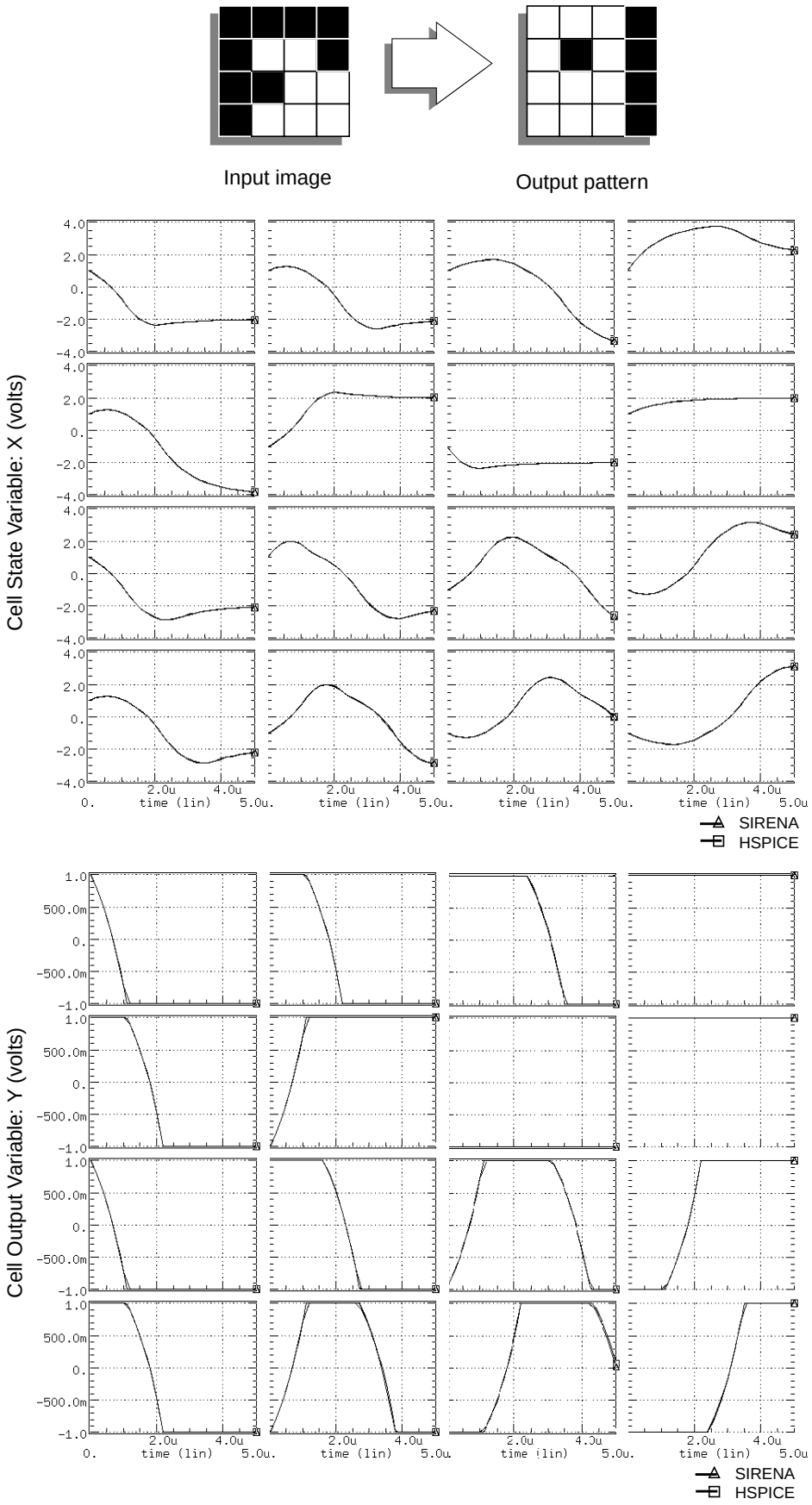


Fig.13: Simulation waveforms of a 4 × 4 cells CNN for Connected Component Detection using Chua-Yang original CNN model given by HSPICE (v.95.1) and SIRENA.

b) OTA-based CNN implementation

In this section, a straight forward realization of a programmable CNN is going to be modeled. This OTA based CNN [18] represents a direct mapping of the coupled differential equations defining the CNN dynamics onto some standard CMOS circuit primitives. In this approach, an OTA block performs the nonlinear operation on the state variable (Gm_A in Fig. 14). Multiplication of the input and output variables and the template elements is performed over the output current of the OTA. This current has been replicated a number of times to implement the whole set of template elements, a total of eighteen multipliers. As in [15], multiplication occurs inside each cell and the properly weighted contribution is passed to the neighborhood. Because of this, template elements should be reorganized to achieve the task for which they were designed [3]. Currents, representing these neighbor contributions and the self-feedback components, are added by wiring them together at the input node of each cell. Dynamic evolution of every cell within the grid domain (*GD*) is described by:

$$C_{x_{ij}} \frac{dV_{x_{ij}}}{dt} = -\frac{V_{x_{ij}}}{R_{x_{ij}}} + \sum_{k,\, l \in N} \{Gm_{A_{kl}}(V_{a_{kl}}, V_{x_{kl}}) + Gm_{B_{kl}}(V_{b_{kl}}, V_{u_{kl}})\} + I_{bias} \tag{10}$$

where Gm_A includes nonlinear operation and multiplication. To understand the way in which this is accomplished let us have a look at the OTA-multiplier block implementation (Fig. 15). To compute the currents through transistors Mn_1 and Mn_2, which operate as an input differential pair, we have:

$$I_B = I_{Mn_1} + I_{Mn_2} \qquad \text{and} \qquad \sqrt{\beta_n} V_{in} = \sqrt{I_{Mn_1}} - \sqrt{I_{Mn_2}} \tag{11}$$

from where it can be derived

$$I_{Mn_1} = \begin{cases} I_B & \text{if} \quad V_{in} > \sqrt{\dfrac{I_B}{\beta_n}} \\ \dfrac{I_B}{2} + \beta_n V_{in} \sqrt{\dfrac{2I_B}{\beta_n} - V_{in}^2} & \text{if} \quad |V_{in}| \le \sqrt{\dfrac{I_B}{\beta_n}} \\ 0 & \text{if} \quad V_{in} < -\sqrt{\dfrac{I_B}{\beta_n}} \end{cases} \tag{12}$$

and:

$$I_{Mn_2} = \begin{cases} 0 & \text{if} \quad V_{in} > \sqrt{\dfrac{I_B}{\beta_n}} \\ \dfrac{I_B}{2} - \beta_n V_{in} \sqrt{\dfrac{2I_B}{\beta_n} - V_{in}^2} & \text{if} \quad |V_{in}| \le \sqrt{\dfrac{I_B}{\beta_n}} \\ I_B & \text{if} \quad V_{in} < -\sqrt{\dfrac{I_B}{\beta_n}} \end{cases} \tag{13}$$

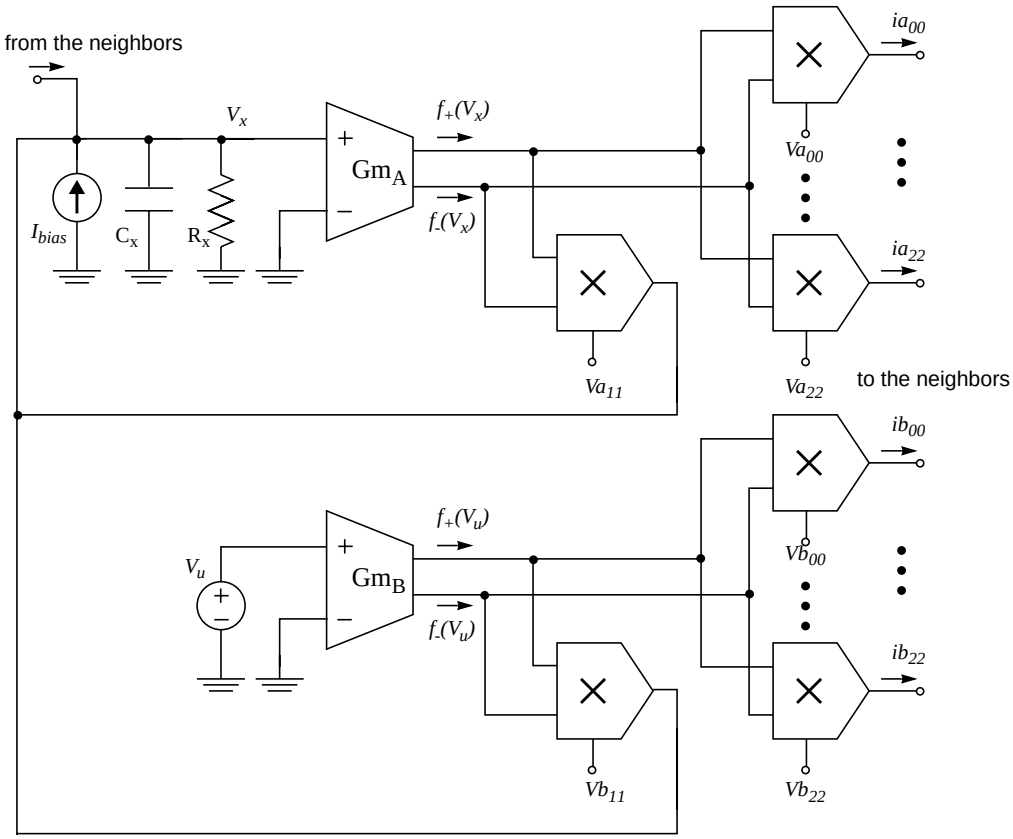


Fig.14: OTA-based CNN implementation conceptual schematics.

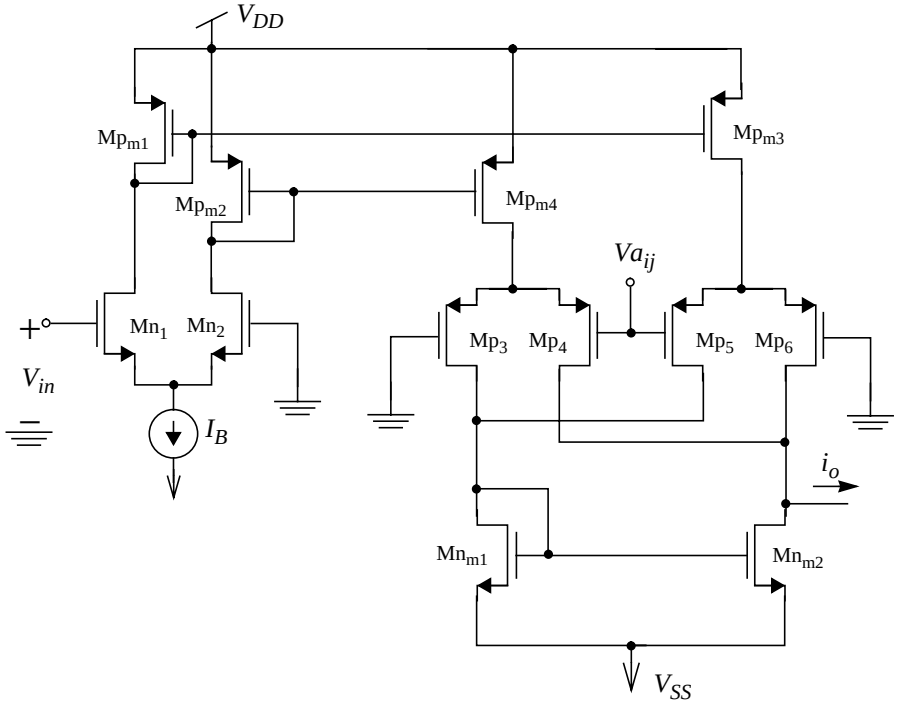


Fig.15: CMOS realization of the OTA-multiplier block.

For the p-channel differential pairs we have, approximating by the first order term of a Taylor

expansion, and within the linear range of operation $|Va_{ij}| \leq \sqrt{I_{Mn_{1,2}}/\beta_p}$:

$$I_{Mp_3} = \frac{I_{Mn_2}}{2} + \sqrt{2\beta_p I_{Mn_2}}\,Va_{ij} \qquad I_{Mp_5} = \frac{I_{Mn_1}}{2} - \sqrt{2\beta_p I_{Mn_1}}\,Va_{ij}$$
$$\text{and} \tag{14}$$
$$I_{Mp_4} = \frac{I_{Mn_2}}{2} - \sqrt{2\beta_p I_{Mn_2}}\,Va_{ij} \qquad I_{Mp_6} = \frac{I_{Mn_1}}{2} + \sqrt{2\beta_p I_{Mn_1}}\,Va_{ij}$$

and therefore, the output current is given by:

$$I_{out_{ij}} = (I_{Mp_6} - I_{Mp_5}) - (I_{Mp_3} - I_{Mp_4}) = \sqrt{2\beta_p}\,Va_{ij}(\sqrt{I_{Mn_1}} - \sqrt{I_{Mn_2}}) \tag{15}$$

Finally, using (11), (12) and (13), the output current of one of the multipliers can be approximated as a piece-wise-linear function of the state variable:

$$I_{out_{ij}} \cong \begin{cases} \sqrt{2\beta_p I_B}\,Va_{ij} & \text{if} \quad V_{in} > \sqrt{\dfrac{I_B}{\beta_n}} \\ \sqrt{2\beta_p\beta_n}\,Va_{ij}V_{in} & \text{if} \quad |V_{in}| \leq \sqrt{\dfrac{I_B}{\beta_n}} \\ -\sqrt{2\beta_p I_B}\,Va_{ij} & \text{if} \quad V_{in} < -\sqrt{\dfrac{I_B}{\beta_n}} \end{cases} \tag{16}$$

where the saturation limits are set to a value that becomes the normalizing factor for equation (10) in order to represent the evolution of a CNN. Template elements have to be redesigned because of this equation scaling [19]. Modelling of this specific implementation of a CNN within SIRENA environment (Fig. 16 shows file **ota.cs**) includes the formal declaration of the variables and CMOS parameters, relations between these variables, and the evolution section with the differential equation.The model description ends with the network definition.

A comparison between HSPICE (v. 96) output and SIRENA results can be made in order to validate this software implementation of the circuit. As before, templates for connected component detection are employed. Table 1 shows the values assigned to each variable and parameter, extracted from HSPICE models of the CMOS devices and the analysis of the circuits implementing the algorithm.

Level 3 models for the MOS devices have been used for HSPICE simulation. Using the graphical user interface [20] the waveforms of the state variables of the CNN array can be generated and observed. Also, with the help of one of its format conversion tools, a plot of the output of both simulators can be displayed with HSPICE graphic interface (GSI) for a comparison (Fig. 15). Once again, reliability of this tool is stated by the similarity of the results obtained by the two different simulators. CPU time and memory consumption figures give some advantage to SIRENA when compared with HSPICE. In this case, for a 4×4 cells CNN simulation in a SPARC Station 10/51 with 32 MB RAM, SIRENA needs 924kB RAM and 61.5 CPU seconds compared to the 7452kB and 279.8 CPU seconds required by HSPICE.

ota.cs

```
layer layer_otasc
  matrix vx;                       //State variable (voltage)
  matrix vu;                      // Input variable (voltage)
  matrix i1x,i2x;                //Differential pair currents
  matrix i1u,i2u;                //Differential pair currents
  matrix i3a00,i4a00,i5a00,i6a00;     //Multipliers currents
                  ⋮                                ⋮
  matrix i3b22,i4b22,i5b22,i6b22;     //Multipliers currents
  matrix iin;                       // Cell input node current
  matrix io00x,io01x,io02x;                //Output currents

  matrix io20u,io21u,io22u;                //Output currents
  matrix ibias;                          // Bias term current
  matrix iout;                             //Output current
  matrix b3a00,b4a00;               //Multipliers parameters
                  ⋮                                ⋮
  matrix b3b22,b4b22;               //Multipliers parameters
  matrix ibi1,ibi2;                    // Bias term currents
  template p00,p01,p02;     //Evaluation of the neighborhood
  template p10,p11,p12;     //Evaluation of the neighborhood
  template p20,p21,p22;     //Evaluation of the neighborhood
  boundary ic00x,ic01x,ic02x;       //Boundary cells current
                  ⋮                                ⋮
  boundary ic20u,ic21u,ic22u;       //Boundary cells current
  scalar va00,va01,va02;               // template voltages
                  ⋮                                ⋮
  scalar vb20,vb21,vb22;               // template voltages
  scalar vbias;                        // Bias term voltage
  scalar cx,rx;           // State capacitor & Loss Resistor
```

layer components declaration

```
evolution
//Input Diff pair currents
i1x = [ vx >= sqrt(ibx/b1x) : ibx |vx <= -sqrt(ibx/b1x) : 0 |
        ibx/2 + (1/2)*b1x*vx*sqrt(2*ibx/b1x-vx*vx)];
                  ⋮                                ⋮
i2u = [ vu >= sqrt(ibu/b1u) : 0 |vu <= -sqrt(ibu/b1u) : ibu |
        ibu/2 - (1/2)*b1u*vu*sqrt(2*ibu/b1u-vu*vu)];
// Multiplier a00
i3a00 = [ va00 >= sqrt(i2x/b3a00) : i2x |va00 <= -sqrt(i2x/b3a00) : 0 |
        i2x/2 + (1/2)*b3a00*va00*sqrt(2*i2x/b3a00-va00*va00)];
                  ⋮                                ⋮
// Multiplier b22
i3b22 = [ vb22 >= sqrt(i2u/b3b22) : i2u |vb22 <= -sqrt(i2u/b3b22) : 0 |
        i2u/2 + (1/2)*b3b22*vb22*sqrt(2*i2u/b3b22-vb22*vb22)];
i5b22 = [ vb22 >= sqrt(i2u/b3b22) : 0 |vb22 <= -sqrt(i2u/b3b22) : i2u |
        i2u/2 - (1/2)*b3b22*vb22*sqrt(2*i2u/b3b22-vb22*vb22)];
i6b22 = [ vb22 >= sqrt(i1u/b4b22) : i1u |vb22 <= -sqrt(i1u/b4b22) : 0 |
        i1u/2 + (1/2)*b4b22*vb22*sqrt(2*i1u/b4b22-vb22*vb22)];
i4b22 = [ vb22 >= sqrt(i1u/b4b22) : 0 |vb22 <= -sqrt(i1u/b4b22) : i1u |
        i1u/2 - (1/2)*b4b22*vb22*sqrt(2*i1u/b4b22-vb22*vb22)];

// Output currents
  iout=i1x-i2x;
  io00x=i6a00+i5a00-i3a00-i4a00;
            ⋮
  io22u=i6b22+i5b22-i3b22-i4b22;
  ibias=ibi1-ibi2;

// State variables time-evolution

    iin = p00><io00x#ic00x + p00><io00u#ic00u +
                  ⋮                    ⋮
          p22><io22x#ic22x + p22><io22u#ic22u +
          ibias;
   derivative(vx) =(iin -vx/rx)/cx;
```

layer equations

network definition

```
network
  sublayer layer_otasc otasc;
end
```

Fig.16: DECEL description of the OTA-based CNN model.

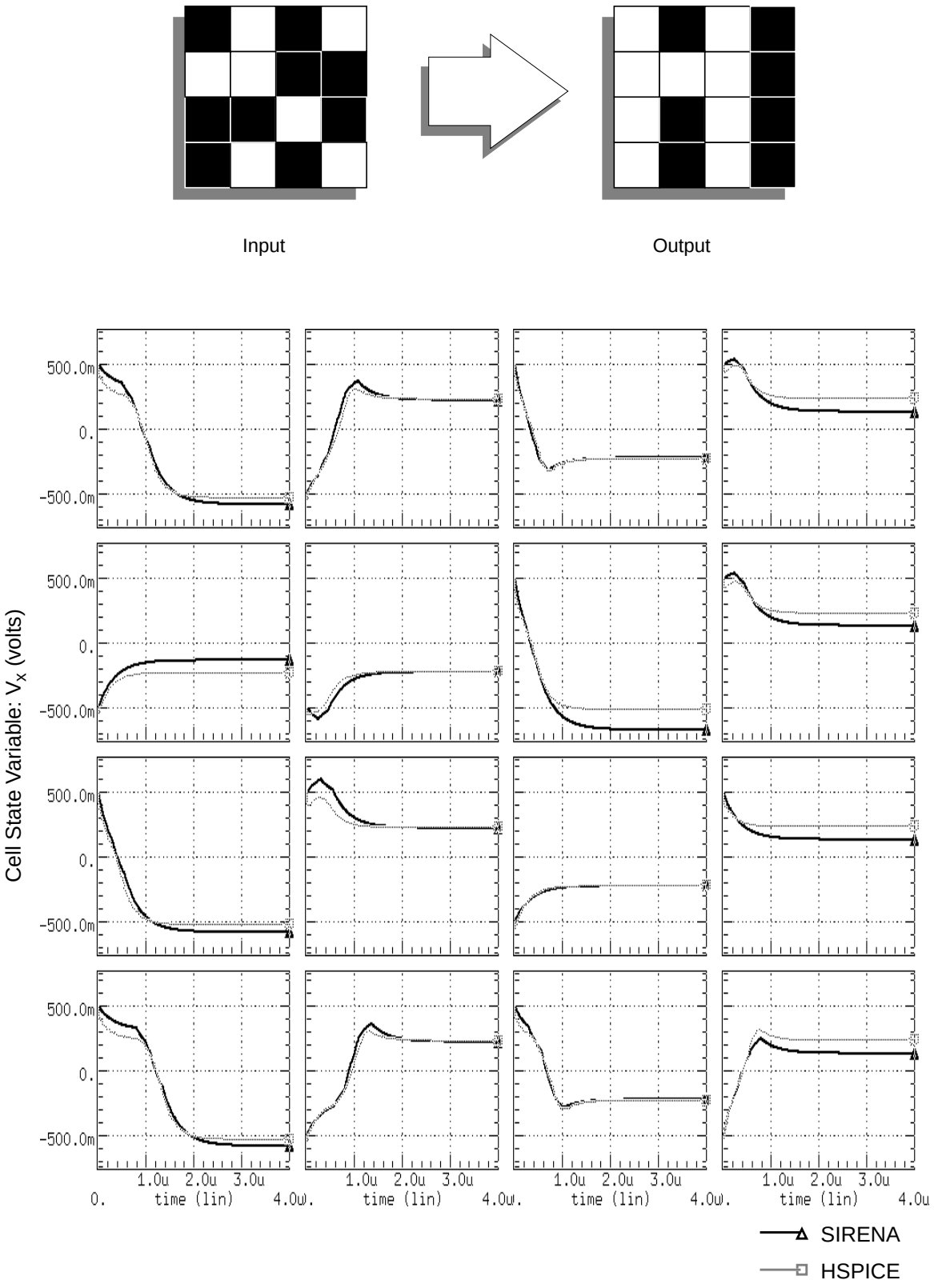


Fig.17: Waveforms of state variables of an OTA-based 4 × 4 CNN performing Connected Component Detection, simulated by HSPICE and SIRENA.

An extended version of this model could be employed for Montecarlo analysis of the cir-

Variable, Parameter	Name	Value	Units
Input and initial state (max)	`vx, vu`	0.50	volts
Input and initial state (min)	`vx, vu`	-0.50	volts
Diff. pairs bias currents	`ibx, ibu`	5.0×10^{-6}	amps
NMOS transconductance	`b1x,...,b2u`	240×10^{-6}	A/V^2
PMOS transconductance	`b3a00,...,b6b22`	60×10^{-6}	A/V^2
Template elements (unity)	`va00,...,vb22`	0.3	volts
Bias term voltage(unity)	`vbias`	0.3	volts
Boundary currents (unity)	`ic00x,...,ic22u`	3.0×10^{-6}	amps
State variable capacitor	`cx`	5.0×10^{-12}	farads
Losses resistor	`rx`	5.0×10^{4}	ohms

Table 1: OTA-based CNN model parameters.

cuit. This will be very useful to evaluate the implementation possibilities for a particular process. Actual tolerance and parameter deviation can be specified and their influence studied with the help of the statistical analysis capabilities of the environment.To take into account device mismatches, some extra variables must be included in the OTA-based model. In addition, the simplifications made to derive differential-pair currents (based on matched transistors) can not be assumed. As a consequence, the description of a simple differential pair becomes more complicated. As a starting point, let us consider that there are slight differences between electrical parameters (transconductance and threshold voltage) of the two ideally matched transistors of a differential pair:

$$\begin{aligned} \beta_{n_1} &= \beta_n + \frac{\Delta\beta_n}{2} \\ \beta_{n_2} &= \beta_n - \frac{\Delta\beta_n}{2} \end{aligned} \qquad \text{and} \qquad \begin{aligned} V_{Tn_1} &= V_{Tn} + \frac{\Delta V_{Tn}}{2} \\ V_{Tn_2} &= V_{Tn} - \frac{\Delta V_{Tn}}{2} \end{aligned} \tag{17}$$

The currents flowing through each of the two devices, given before by (12) and (13), are now expressed by:

$$I_{Mn1} = ((2\beta_n + \Delta\beta_n)(-2\beta_n\Delta\beta_n\Delta V_{Tn}^2 + \Delta\beta_n^2\Delta V_{Tn}^2 + 4\beta_n I_B + 4\beta_n\Delta\beta_n\Delta V_{Tn}V_{in} - 2\Delta\beta_n^2\Delta V_{Tn}V_{in} - 2\beta_n\Delta\beta_n V_{in}^2 + \Delta\beta_n^2 V_{in}^2 + (2\beta_n - \Delta\beta_n)(V_{in} - \Delta V_{Tn}) \text{sqrt}(-4\beta_n^2\Delta V_{Tn}^2 + \Delta\beta_n^2\Delta V_{Tn}^2 + 8\beta_n I_B + 8\beta_n^2\Delta V_{Tn}V_{in} - 2\Delta\beta_n^2\Delta V_{Tn}V_{in} - 4\beta_n^2 V_{in}^2 + \Delta\beta_n^2 V_{in}^2)))/16\beta_n^2 \quad (18)$$

$$I_{Mn2} = ((2\beta_n - \Delta\beta_n)(2\beta_n\Delta\beta_n\Delta V_{Tn}^2 + \Delta\beta_n^2\Delta V_{Tn}^2 + 4\beta_n I_B + 4\beta_n\Delta\beta_n\Delta V_{Tn}V_{in} - 2\Delta\beta_n^2\Delta V_{Tn}V_{in} - 2\beta_n\Delta\beta_n V_{in}^2 + \Delta\beta_n^2 V_{in}^2 + (2\beta_n + \Delta\beta_n)(V_{in} - \Delta V_{Tn}) \text{sqrt}(-4\beta_n^2\Delta V_{Tn}^2 + \Delta\beta_n^2\Delta V_{Tn}^2 + 8\beta_n I_B + 8\beta_n^2\Delta V_{Tn}V_{in} - 2\Delta\beta_n^2\Delta V_{Tn}V_{in} - 4\beta_n^2 V_{in}^2 + \Delta\beta_n^2 V_{in}^2)))/16\beta_n^2 \quad (19)$$

c) A current-mode CNN model based on a hardware realization

In order to deepen inside SIRENA macromodeling capabilities, let us consider now the current mode implementation [15] of a CNN based on the exhaustive use of current mirrors, and designed to perform Connected Component Detection [17] of binary input patterns. Templates for this operation have the following values:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad d = 0 \quad (20)$$

This specific-application CNN is rather simple, having null control template and offset term, and only two neighbors connected. Still, it will be useful to illustrate the capabilities of SIRENA. Besides, this circuit has been successfully designed, integrated, proven and reported [21].

Let us consider the schematic of the basic processor depicted in Fig. 18. First, state variables of basic cells must be appointed, and their evolution laws must be established. We have chosen the voltages across capacitors C_x, C_{yp} and C_{yn} to be our state variables, namely: x, y_p and y_n. These two latter are the output variables of the cell, limited versions of the cell state x, each with different sign. In these conditions, time evolution of the three state variables is described (with the help of some auxiliary currents that will be defined later) by the following set of equations:

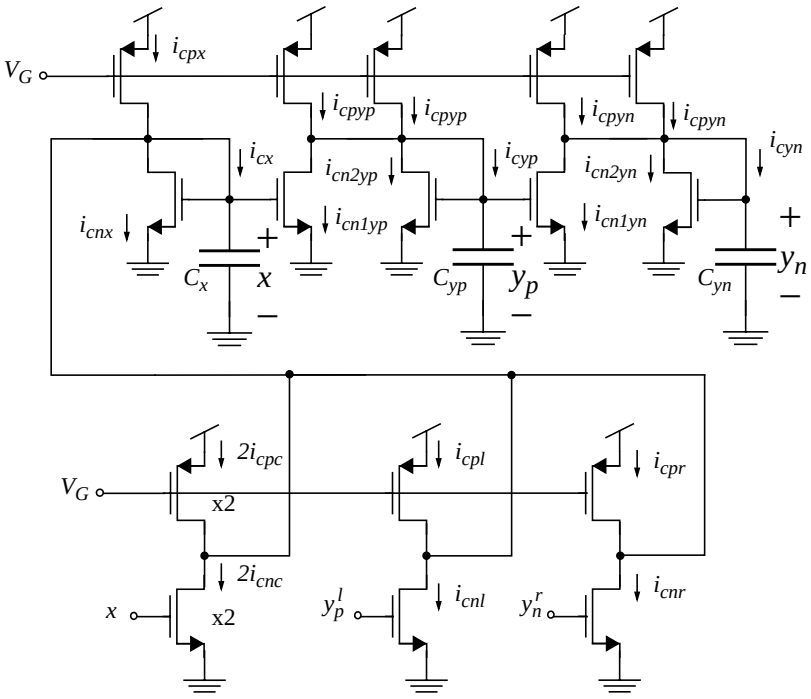


Fig.18: Schematic diagram of a current-mode CNN cell for Connected Component Detection based on bias-shifted current mirrors.

$$\begin{aligned} \frac{dx}{dt} &= \frac{i_{cx}}{C_x} \\ \frac{dy_p}{dt} &= \frac{i_{cyp}}{C_{yp}} \\ \frac{dy_n}{dt} &= \frac{i_{cyn}}{C_{yn}} \end{aligned} \tag{21}$$

Now, current i_{cx} is the sum of a feedback term, double of the current generated by cell state variable, and the neighbors contribution with the sign commanded by the feedback template, apart from the current injected by the current mirror supporting the state capacitor. Each current contribution towards this node has a positive component, due to the current sources implemented with PMOS transistors, and a negative term dragged by the n-channel devices,

$$i_{cx} = 2(i_{cpc} - i_{cnc}) + (i_{cpl} - i_{cnl}) + (i_{cpr} - i_{cnr}) + (i_{cpx} - i_{cnx}) \tag{22}$$

Currents flowing towards the other two state capacitors are computed in a similar manner:

$$i_{cyp} = 2i_{cpyp} - i_{cn1yp} - i_{cn2yp} \tag{23}$$

$$i_{cyp} = 2i_{cpyp} - i_{cn1yp} - i_{cn2yp} \tag{24}$$

Finally, each term in (22), (23) and (24) is obtained from level-one Schichman-Hodges MOS transistor equations for the drain-to-source current, detailed here for a clearer exposition of the DECEL description of this current-mode CNN model. For NMOS transistors:

$$I_{ds} = \begin{cases} 0 & \text{if} \quad V_{gs} < V_{Tn} \\ k_n s_n \left[(V_{gs} - V_{Tn}) V_{ds} - \dfrac{V_{ds}^2}{2} \right] & \text{if} \quad V_{ds} \le V_{gs} - V_{Tn} \\ \dfrac{k_n}{2} s_n (V_{gs} - V_{Tn})^2 & \text{if} \quad V_{ds} > V_{gs} - V_{Tn} \end{cases} \tag{25}$$

while for PMOS devices:

$$I_{sd} = \begin{cases} 0 & \text{if} \quad V_{sg} > V_{DD} - |V_{Tp}| \\ k_p s_p \left[(V_{sg} - |V_{Tp}|) V_{sd} - \dfrac{V_{sd}^2}{2} \right] & \text{if} \quad V_{sd} \le V_{sg} - |V_{Tp}| \\ \dfrac{k_p}{2} s_p (V_{sg} - |V_{Tp}|)^2 & \text{if} \quad V_{sd} > V_{sg} - |V_{Tp}| \end{cases} \tag{26}$$

Once the state variables and equations are stated, a DECEL description of the model can be made (Fig. 19). Let us go through **cm.es** file. Within layer components declaration, state variables are defined as matrices. Neighbors' output matrices are defined also for convenience. After that, auxiliary currents used to evaluate the contributions to the time derivatives (21) of the state variables are stated. It is necessary to have the information about neighbors output variable within each cell to compute the contributions i_{cpl}, i_{cpr}, i_{cnl} and i_{cnr} to the current flowing through node *x*. Two templates are declared for this purpose (weights of the connections can be included here or left to further description within layer equations statement). Finally, boundary conditions and several design and technological parameters are declared, namely the power supply and bias voltages, transistor aspect ratios, transconductance parameters and threshold voltages, and state capacitors.

Layer equations are depicted next. At the beginning of this section, output variables of the neighbors are assigned to cell variables *yl* and *yr.* After that, auxiliary currents are calculated based on the transistor model described by (25) and (26), concluding with DECEL statements equivalent to equations (22), (23) and (24). This *evolution* section ends with the time evolution laws of the state variables (21). The last part of the model description is the network definition, and once again this is a one-layer CNN.

Table 2 shows the values assigned to each variable and parameter, extracted from HSPICE models of the CMOS devices and circuit analysis.

cm.cs

```
layer layer_cm
  matrix x, yp, yn ;                        //State Variables
  matrix yl, yr;              //Neighborhood State Variables
  matrix icx, icyp, icyn;               //Auxiliary currents
               ⋮                                  ⋮
  matrix icn1yn,icn2yn,icn1yp2, icn1yn2;
  template ayl,ayr; //Template for neighbor state evaluation
  boundary bound_yp,bound_yn; //Boundary conditions (yp, yn)
  scalar vdd, vg;             //Power Supply and Bias Voltage
  scalar sn,sp;           //MOS Trans. Geometric aspect ratio
  scalar kp,kn,vtp,vtn;                   //Techn. Parameters
  scalar cx, cyp, cyn;                     //State Capacitors
```

layer components declaration

```
evolution
    yl= ayl><yp#bound_yp;
    yr= ayr><yn#bound_yn;

    icpl= [ x < vg+vtp : (kp*sp/2)*(vdd-vg-vtp)*(vdd-vg-vtp) |
        kp*sp*((vdd-vg-vtp)*(vdd-x)-(vdd-x)*(vdd-x)/2) ];
    icnl2= [ x > (yl-vtn) : (kn/2)*sn*(yl-vtn)*(yl-vtn) |
        kn*sn*((yl-vtn)*x -x*x/2) ];
    icnl= [ yl > vtn : icnl2 | 0 ];
    icpr=icpl ;
    icnr2= [ x > (yr-vtn) : (kn/2)*sn*(yr-vtn)*(yr-vtn) |
        kn*sn*((yr-vtn)*x -x*x/2) ];
    icnr= [ yr > vtn : icnr2 | 0 ];
    icpc=2*icpl ;
    icnc2= [ x > (yp-vtn) : kn*sn*(yp-vtn)*(yp-vtn) |
        2*kn*sn*((yp-vtn)*x -x*x/2) ];
    icnc= [ yp > vtn : icnc2 | 0 ];
    icpx= icpl;
    icnx= [ x < vtn : 0 | (kn/2)*sn*(x-vtn)*(x-vtn) ];
        icpyp= [ yp < vg+vtp : (kp*sp/2)*(vdd-vg-vtp)*(vdd-vg-vtp) |
        kp*sp*((vdd-vg-vtp)*(vdd-yp)-(vdd-yp)*(vdd-yp)/2) ];
    icn1yp2= [ yp > (x-vtn) : (kn/2)*sn*(x-vtn)*(x-vtn) |
        kn*sn*((x-vtn)*yp -yp*yp/2) ];
    icn1yp= [ x > vtn : icn1yp2 | 0 ];
    icn2yp= [ yp < vtn : 0 | (kn/2)*sn*(yp-vtn)*(yp-vtn) ];
    icyp= 2*icpyp-icn1yp-icn2yp;
    icpyn= [ yn < vg+vtp : (kp*sp/2)*(vdd-vg-vtp)*(vdd-vg-vtp) |
        kp*sp*((vdd-vg-vtp)*(vdd-yn)-(vdd-yn)*(vdd-yn)/2) ];
    icn1yn2= [ yn > (yp-vtn) : (kn/2)*sn*(yp-vtn)*(yp-vtn) |
        kn*sn*((yp-vtn)*yn -yn*yn/2) ];
    icn1yn= [ yp > vtn : icn1yn2 | 0 ];
    icn2yn= [ yn < vtn : 0 | (kn/2)*sn*(yn-vtn)*(yn-vtn) ];
    icyn= 2*icpyn-icn1yn-icn2yn;

    derivative(x)= icx/cx;
    derivative(yp)= icyp/cyp;
    derivative(yn)= icyn/cyn;
```

layer equations

```
network
  sublayer layer_cm cm;
end
```

network definition

Fig.19: DECEL description of the current-mode CNN model

Variable, Parameter	Name	Value	Units
Initial state (max.)	x, yp, yn	0.98	volts
Initial state (min.)	x, yp, yn	0.70	volts
Power Supply voltage	vdd	5.0	volts
Bias voltage	vg	3.6	volts
NMOS transconductance	kn	45×10^{-6}	A/V^2
PMOS transconductance	kp	15.4×10^{-6}	A/V^2
NMOS threshold voltage	vtn	0.60	volts
PMOS threshold voltage	vtp	0.98	volts
State capacitor (I)	cx	0.450×10^{-12}	farads
State capacitor (II)	cyp	0.112×10^{-12}	farads
State capacitor (III)	cyn	0.042×10^{-12}	farads
Positive boundary condition	bound_yp	0.98	volts
Positive boundary condition	bound_yn	0.70	volts
Losses resistor	rx	5.0×10^{4}	ohms

Table 2: Current-mode CNN model parameters.

Fig. 20 shows the waveform plots of the state variable *x* and the output variable y_p of every cell in a 4×4 current-mode CNN obtained from HSPICE (using level-two models for MOS) and from SIRENA using the macromodel described above. Notice that sign of y_p is reversed from that of the state variable due to the current inversion imposed by mirroring. Time scale of these graphs, in the range of tenths of microseconds, begin at 100ns because of network initialization. Similarity between each of the sixteen pairs of curves gives an idea of the accuracy of the macromodel employed in SIRENA. A higher degree of precision could be reached with the inclusion of second order effects like mobility degradation, non-linear gate capacitance, channel-length modulation, etc.

As with previous examples, several HSPICE and SIRENA simulations of different-size networks have been realized with the same machine (SPARC Server 1000E). CPU time plots versus network size (Fig. 21) shows a considerable advantage for SIRENA, reaching two orders of magnitude for large networks. The larger efficiency of this environment, within the scope for which it was developed, is therefore clear.

d) A Discrete-Time Hard-nonlinearity non-linear CNN model

Finally, in order to illustrate the broad variety of network types that can be described in

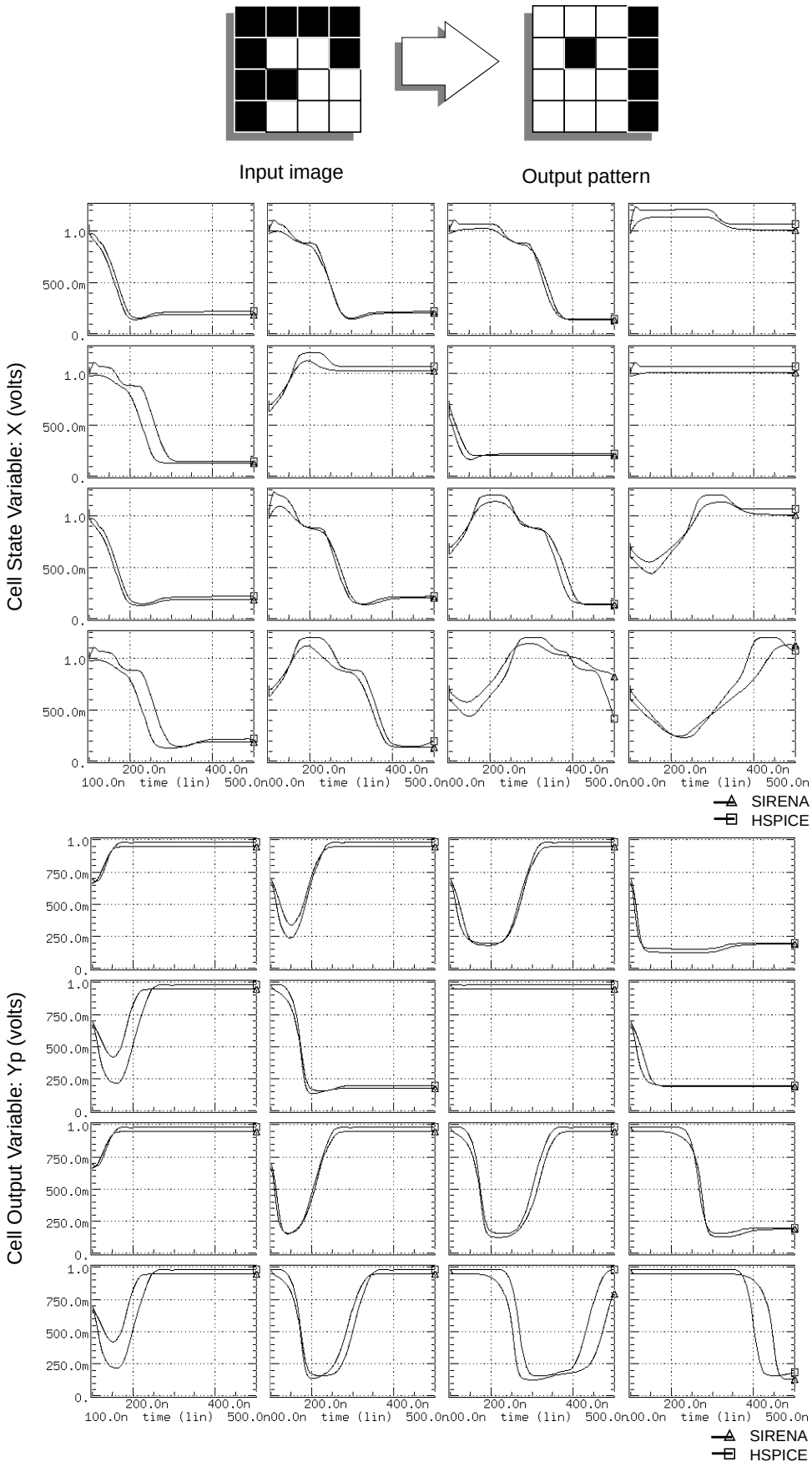


Fig.20: Simulation waveforms of a 4 × 4 cells Current-Mode CNN for Connected Component Detection obtained from HSPICE and SIRENA.

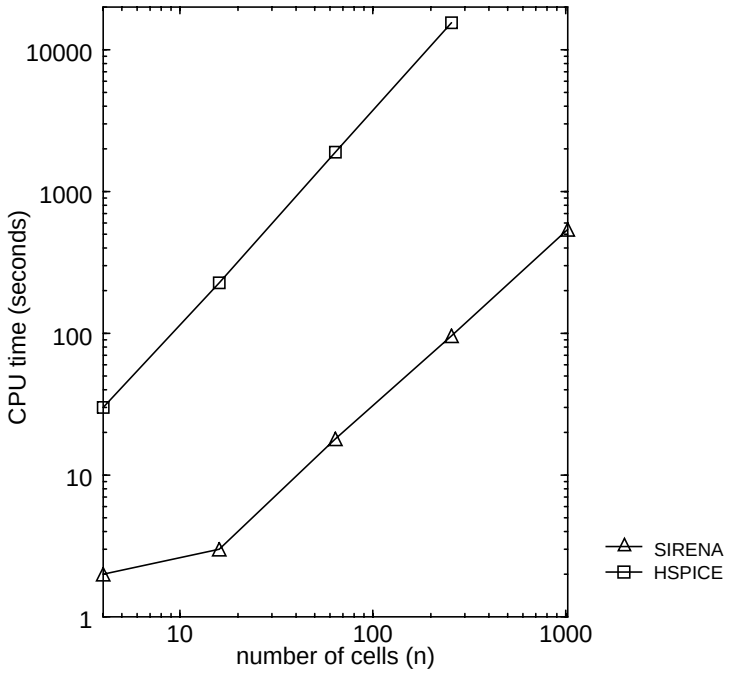


Fig.21: Log-log graphs of CPU times consumed by HSPICE (95.1) and SIRENA vs. number of cells in the network.

DECEL language, let us consider the case of a discrete-time CNN chip to perform the Radon Transform operation [22]. Some peculiar properties of this algorithm include the hard-limitation of the cell output variable and the use of non-linear template elements (which depend of the cell state). The operation consists on computing the integral of intensity values along each row of the network. For binary images, this results in a histogram of the input pattern. The discrete-time evolution of the network is now described by a system of finite-differences equations,

$$x^c(n+1) = \sum_{d \in N_r(c)} a_d^c[y^c(n)]y^d(n) \qquad \forall c \in GD \tag{27}$$

where the non-linear feedback template is described by

$$A(y^c) = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1-y^c}{2} & 0 & \frac{1+y^c}{2} \\ 0 & 0 & 0 \end{bmatrix} \qquad \forall c \in GD \tag{28}$$

and where the output variable is obtained through a hard-limiting non-linear function:

$$y^c(n) = f[x^c(n)] = \begin{cases} 1 & \text{if} \quad x^c > 0 \\ -1 & \text{if} \quad x^c \le 0 \end{cases} \qquad \forall c \in GD \tag{29}$$

Several differences can be observed between this network-model and those reported before. First of all, this is a discrete-time CNN. This forces the definition of a set of variables

within the DECEL description of the model (file **dt.cs** in Fig. 22) for evaluate convergence evaluation during the simulation process. Feedback-template dependence on the cell output must be included in the evolution law of the model, and therefore the connection operator must store neighbors output variables into separate matrices. The last part of the model description is the network definition.

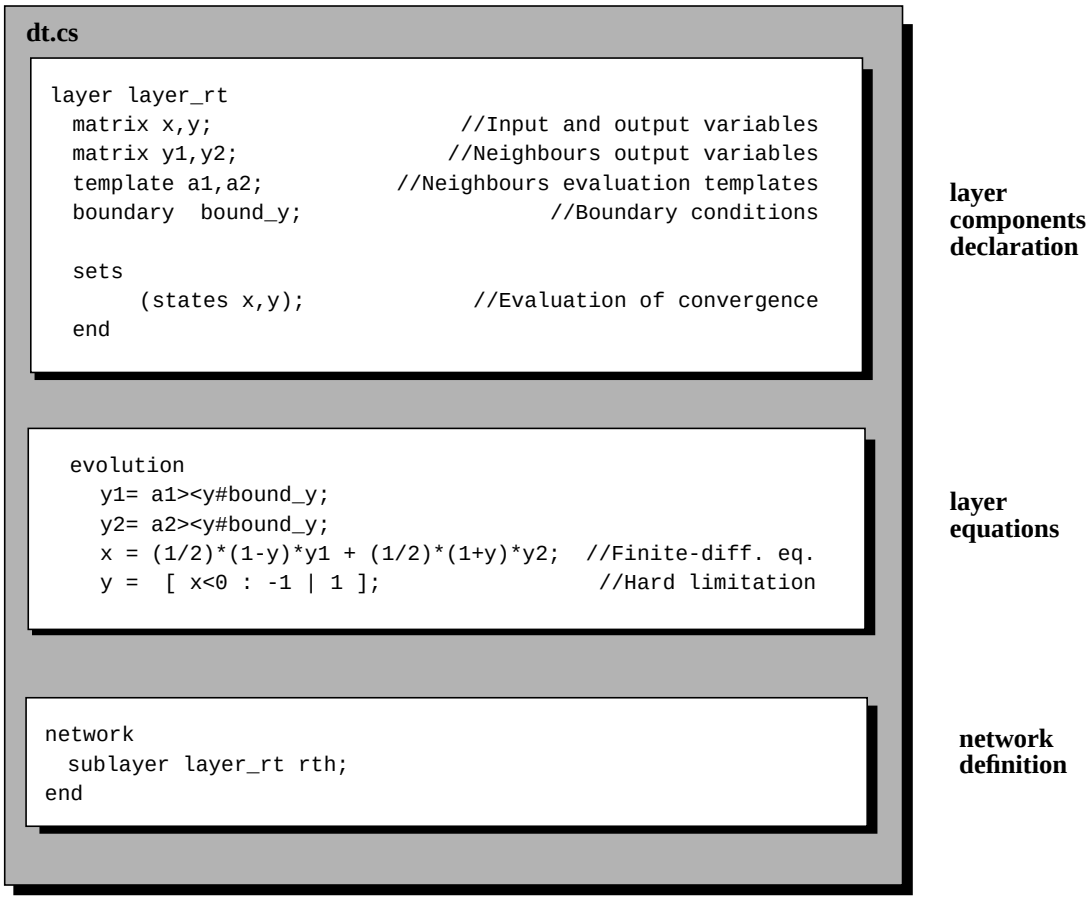


Fig.22: DECEL description of a DT-CNN model

Input and output patterns of SIRENA simulation are shown in Fig. 9 as depicted by the Graphical User Interface.

IV. OTHER SIMULATION CAPABILITIES

The basic capability of the simulator core (NSS) is the time-domain analysis of the network evolution. In addition, some multi-analysis features have been included, following the VLSI-orientation of this tool. Some of them emulate physical phenomena and situations present in real microelectronic circuits, like network-parameters deviations from their nominal values, signals noise, etc. Besides, multi-layer and multi-network analysis are included, broadening the class of algorithms which can be simulated in SIRENA.

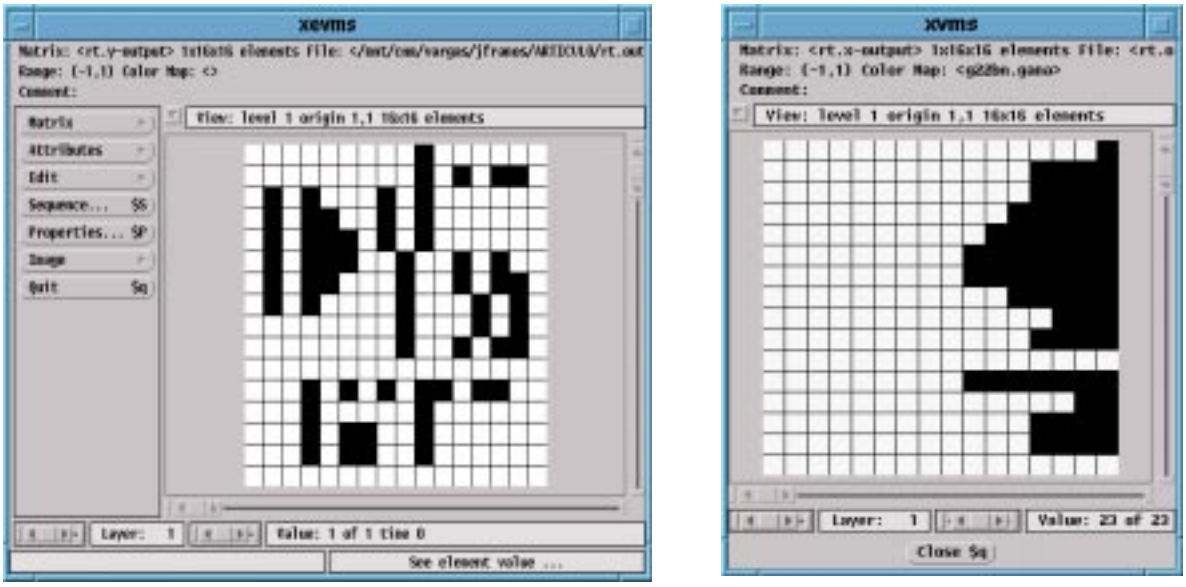


Fig.23: Input and output patterns of the Radon Transform CNN simulation as depicted by the Graphical User Interface.

a) Sensitivity and Montecarlo Analysis

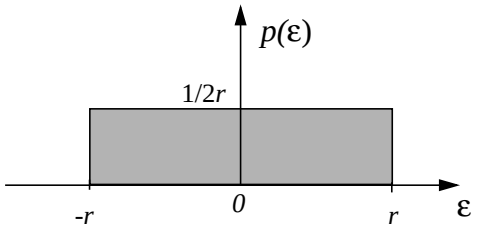
These functionalities consist in the automatic realization of multiple transient analysis. In sensitivity analysis, the value of a single parameter or variable is swept with a specified increment and range, while in Montecarlo analysis, one or more parameters or variables are randomly modified with specified probability distributions (Fig. 24) to obtain a number of modified networks.

Real random deviations are multifarious. Those affecting the whole silicon die, like process parameter deviation from run to run, act similarly over each cell of a cellular neural network (global deviations). On the other hand, spatial process-parameter variations within the die result in different parameter values for each cell in the network (local deviations). Local deviations can in turn be separated into a spatial-gradient component, and a random component (long- and short-distance mismatches, respectively). In general, gradient orientation and magnitude of the long distance variations are unpredictable as well, and therefore, of random nature from a design point of view.

The evaluation of these inaccuracies is a prerequisite for the fabrication of high-complexity analog VLSI circuits employing small devices. It is however extremely expensive in terms of CPU time, and hence, its analysis with traditional SPICE-like electrical simulators is, sometimes, virtually impossible. SIRENA models can include the three types of deviations.

Montecarlo and sensibility capabilities of SIRENA allow the evaluation of parameter deviation effects and the obtention of tolerance margins, something crucial for the characterization of CNN algorithms and specific hardware implementations. This permits the optimization

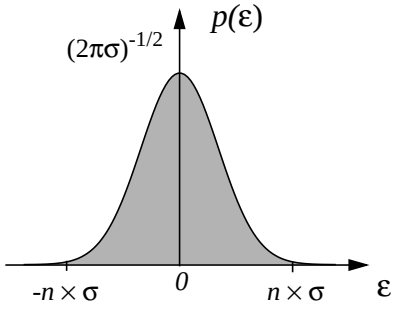
$$P = P_{nom} + \Delta P = P_{nom}\left(1 + \frac{\Delta P}{P_{nom}}\right) \qquad \text{where} \qquad \Delta P = \varepsilon_{abs} = P_{nom} \cdot \varepsilon_{rel}$$



Uniform error probability distribution:

$$\varepsilon \in (-r, r)$$

$$p(\varepsilon) = \frac{1}{2r}$$



Zero-mean gaussian distribution:

$$\varepsilon \in (-n \times \sigma, n \times \sigma)$$

$$p(\varepsilon) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{\varepsilon^2}{2\sigma^2}}$$

Fig.24: Error probability functions available in SIRENA for emulation of parameters deviation from their nominal values.

of the robustness[5] of the algorithm (or electronic implementation) against the expected anomalies, systematic or random (this is, design centering).

In order to illustrate the multianalysis capabilities of SIRENA, let us consider a template-set employed for edge-extraction of binary images [16]. Nominal values of the template elements are:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \qquad d = -1.5 \tag{30}$$

Montecarlo analysis of specific CNN implementations, based either on Chua's or Full-Signal-Range model [23], for this application, showed an unexpected, extremely low rate of networks with correct functionality. In particular, it was observed that either global or local devia-

5. A specific algorithm (or system) is said to be robust if, for each influent parameter, there is a wide enough interval around its aimed value within which the parameter can deviate without causing misbehavior.

tions of the offset term (*d*) had drastic yield effects, suggesting a low tolerance around the nominal offset term value. Repeated sensibility analysis of networks ranging from 8×8 to 32×32 cells, sweeping the offset term with 50 equidistant values around the nominal value, and with the help of a simple post-processor for output comparison against the expected output pattern, showed a tolerance margin below 0.01%. This results confirmed that the nominal value laid on the verge of the tolerance margin. The determination of this tolerance margin allowed the selection of a new nominal value of -1.195, for which subsequent sensitivity analysis showed a tolerance of 12%.

Fig. 25 shows interpolated yield curves obtained from Montecarlo analysis of the network with different mean values for the offset-term. The plots represent probability of correct opera-

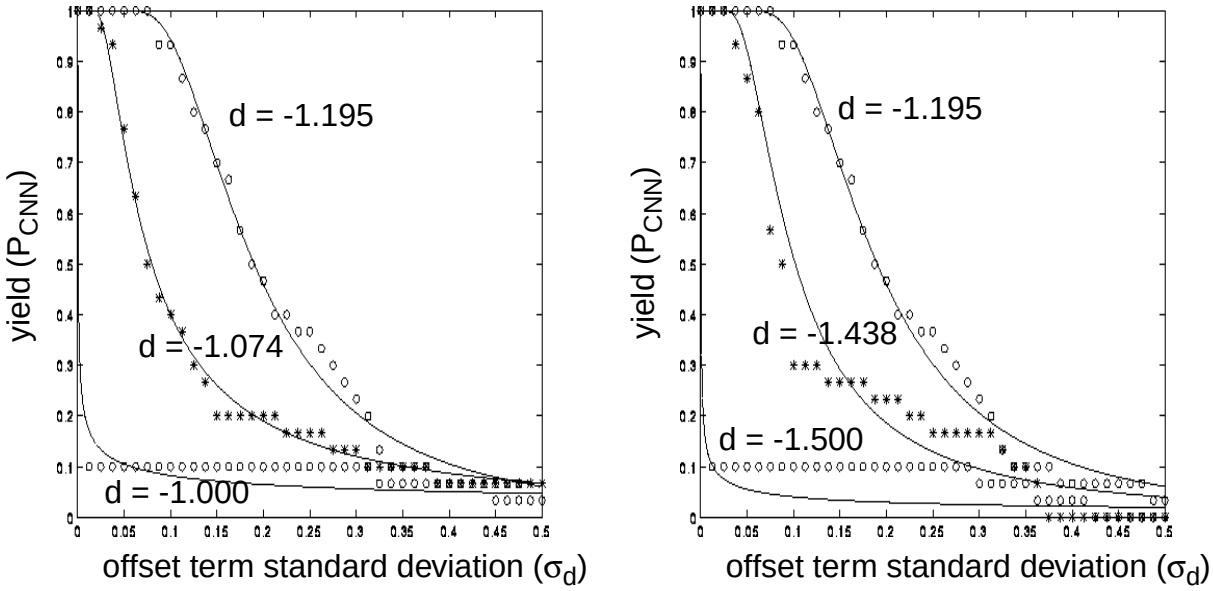


Fig.25: Yield optimization tuning the offset term for the Edge Detector set of templates.

tion of the network versus the standard deviation of the gaussian probability-density distribution assumed for the offset term. It is shown that the computed value (*d* = -1.195) displays a much more robust behavior compared to the initial value (-1.50).

b) Noise Analysis

This feature is intended for the evaluation of noise influence on the evolution of a given network. Because CNNs are strongly nonlinear systems, the conventional approach of traditional electrical simulators, in which noise analysis is associated to small signal equivalents, can not be applied. In our case, the simulator injects a discrete-time noise component on user-specified signals, with a user-specified discrete-time frequency. Noise probability distribution and spectral density can also be arbitrarily shaped.

A combined noise and sensitivity analysis allows the evaluation of the noise threshold that can be tolerated by the network. Next figures show time-evolutions of the state-signal of three cells from a 16×16 cells FSR CNN for edge detection, for three different levels of signal-to-noise ratio (SNR). Noise-free evolutions for the same cells and network are superimposed.

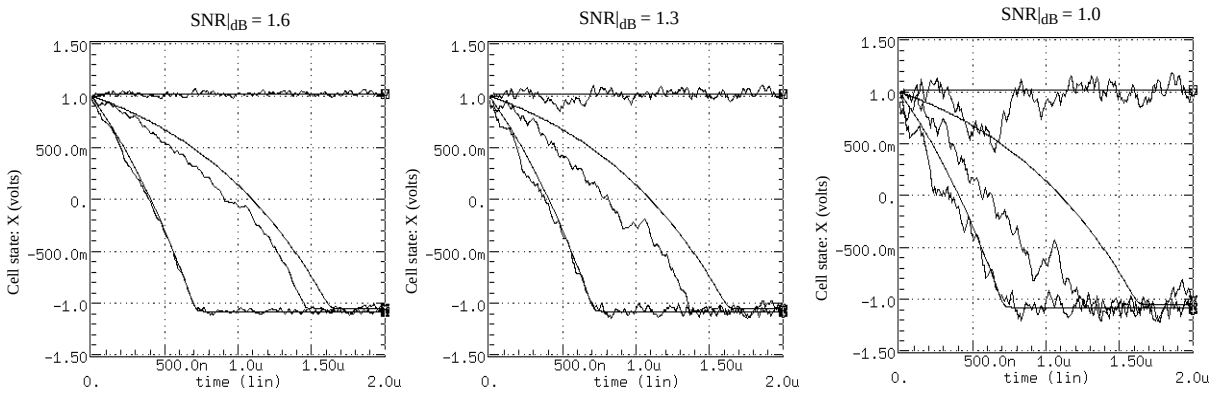


Fig.26: Waveform plots of three noisy cell-state variables given by SIRENA.

c) Multi-layer and Multi-network Analysis

These extended capabilities are directly related with the modelling versatility of the GMS tool. Multi-layered networks are described using several state-variables and interconnections among them. This capability has been tested with several examples, including the implementation of a continuous-time, two-layers CNN for Radon Transform operation based on Connected Component Detector templates [24]. Finally, NSS allows the simulation of complex systems composed of several CNNs interchanging information at specified time instants or following a prescribed protocol. This functionality is controlled by an additional "synchronization file" in which activation of different networks and data interchange among them is detailed. The syntax of this task-scheduler permits the evaluation of sequential and multipath algorithms, or in general, an arbitrary control-flow with conditional, jump and algebraic operators. In order to illustrate this capability, an object counting application [25] based on several CNNs will be described. This application requires the realization of several simple CNN tasks (Fig. 27). First, the binary input image must be hole-filled. Two copies of the resulting image are then processed in separate paths. In one of them, active pixels whose position is at the bottom of some object are marked. On the other, pixels being at the bottom of an object *and* belonging to a concave neighborhood are selected. Templates for these operations are reported in [16]. Afterwards, connected component detection is performed on the two resulting images. Finally, the last part of the algorithm (not accomplished by CNNs) counts the number of black pixels in both output patterns and obtains the total number of objects in the initial image from the difference of the two quantities.

Fig. 28 shows the synchronization file corresponding to the object counting example

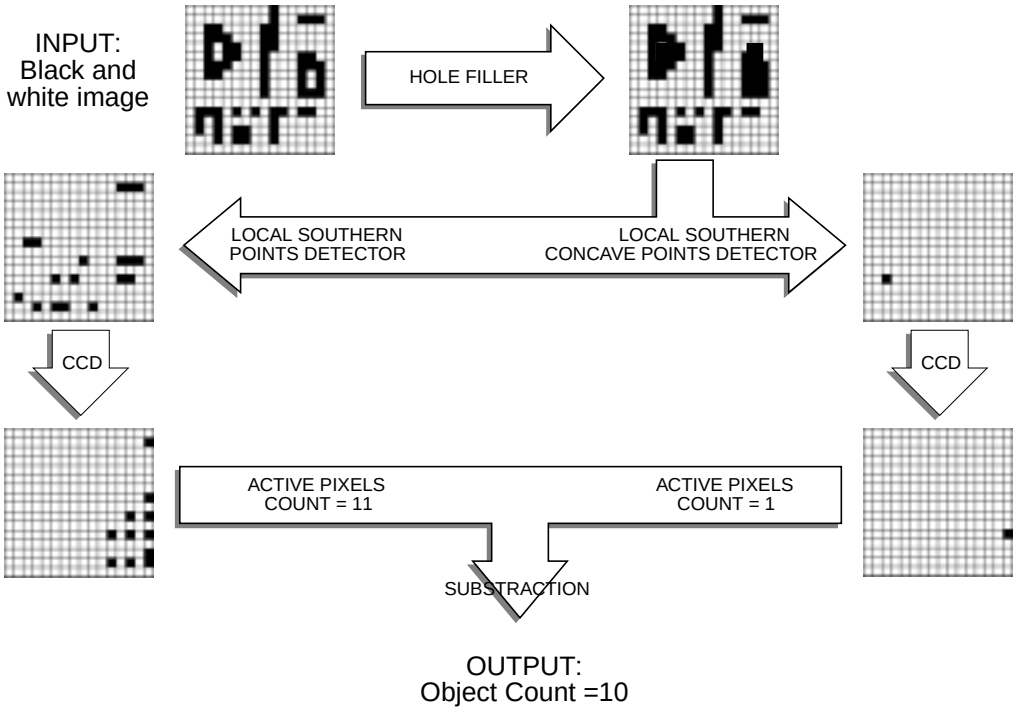


Fig.27: Flow diagram of the object counting application with CNNs [25].

(**multin.sync**). A prescribed order is established in which different networks are enabled (`active` command) following a specific sequence. Their outputs are defined as inputs to some other networks. Finally a `loadsync` statement (similar to `loadnet` command) in the simulation script (or in the command shell) is enough to set up the simulator core (NSS) for multi-network algorithm simulation.

V. CONCLUSIONS

This paper has presented SIRENA: a simulation environment for Cellular Neural Networks with emphasis towards VLSI-implementation needs. It has been developed to overcome the limitations of algorithmic simulators regarding non-ideal effects derived from physical implementations of microelectronic circuits, and those of SPICE-type electrical simulators which require excessively high CPU times. In SIRENA, CNN models are described with the help of a high-level programming language especially developed for this task: DECEL. Modeling guidelines have been sketched using several examples corresponding to specific algorithms and electronic-implementation alternatives. SIRENA modeling and simulation capabilities have been shown to be broad, including hardware-evaluation capabilities like sensitivity, Montecarlo and noise analysis. A graphical interface has been developed, with several user-friendly tools for simulation control and input/output images edition and visualization.

Efficiency comparisons between this dedicated environment and general electrical simu-

multin.sync

```
active hole_filler;            // enable hole filler network
active loc_southern          // enable bottom pixels detector
        ls.u=hf.y               // use hf output as ls input
        ls.x=hf.y;      // use hf output as ls initial state
active loc_so_concave //enable bottom concave pixels detector
        lc.u=hf.y      // use hole filler output as lc input
        lc.x=hf.y;      // use hf output as lc initial state
active ccd1                                //enable first ccd
        ccd1.u=ls.y           // use ls output as ccd1 input
        ccd1.x=ls.y;   // use ls output as ccd1 inital state
active ccd2                               // enable second ccd
        ccd2.u=lc.y           // use lc output as ccd2 input
        ccd2.x=ls.y;  // use lc output as ccd2 initial state
```

Fig.28: Synchronization file for a multi-network simulation.

lators show promising figures in terms of CPU-time consumption. It has been shown that SIRENA makes an efficient use of computing resources within the field for which it has been developed: CNN VLSI implementations. Its modularity allows feasible expansion and improvement, which combined with its present generality results in an adequate, model-independent CNN research tool.

REFERENCES

[1] L.O. Chua and L. Yang: "Cellular Neural Networks: Theory and Applications". *IEEE Trans. Circuits and Systems*, Vol. 35, pp 1257-1290, October 1988.

[2] Haykin, Simon S., *"Neural networks: a comprehensive foundation"*. New York, Macmillan Press 1994.

[3] S. Espejo, "*Cellular Neural Networks: VLSI Design and Modelling*", Ph. D. Thesis, Universidad de Sevilla, February 1994.

[4] L.O. Chua and T. Roska, "The CNN Paradigm". *IEEE Trans. Circuits and Systems I: Fundamental Theory and Applications*, vol. 40, pp. 147-156, March 1993.

[5] T. Roska, L.O. Chua: "The CNN Universal Machine: An Analogic Array Computer". *IEEE Trans. Circuits and Systems-II Analog and Digital Signal Processing*, Vol. 40, No. 3, pp 169-173, March 1993.

[6] S. Espejo, R. Carmona, R. Domínguez-Castro and A. Rodríguez-Vázquez, "A CNN Universal Chip in CMOS Technology". *International Journal of Circuit Theory and Applications.*Vol 24, No. 1, pp 93-110, January-February 1996.

[7] Kinget, P.; Steyaert, M. "An analog parallel array processor for real-time sensor signal processing". *Proc.IEEE International Solid-State Circuits Conference*. pp. 92-93. San Francisco, CA, USA, Feb. 1996

[8] M. Marinaro, and P. G. Morasso: "Modular Object-Oriented Neural Network Simulators and Topology Generalizations". Proc. of the Int. Conf. on Artificial Neural Networks, pp 747-750, Springer-Verlag, May 1994.

[9] M. Plonski and C. Joyce: "RCS, GENESIS and SFINX: Three Public-Domain Simulators for Neural Networks". *Neural Networks Review* 4, 1990.

[10] *"HSPICE User Manual"* Meta Software Inc. 1988.

[11] I.García-Vargas: *"SIRENA: Un Entorno para la Simulación de Redes Neuronales Artificiales. Generación de Modelos y Simulación"*. Proyecto Fin de Carrera, Facultad de Informática y Estadística. Universidad de Sevilla, 1994.

[12] H. Harrer: "Comparison of Different Numerical Integration Methods for Simulating Cellular Neural Networks". Institute for Network Theory and Circuit Design, Technical University of Munich. Report TUM-LNS-TR-90-9, September 1994.

[13] T. Roska, L. O. Chua: "Cellular Neural Networks with nonlinear and delay-type elements". *Proc. of the First IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pp 12-25, Budapest 1990.

[14] H. Harrer, J. A. Nossek and R. Steltz: "An Analog Implementation of Discrete-Time Cellular Neural Networks". *IEEE Trans. Neural Networks*, Vol. 3, pp. 466-476, May 1992.

[15] A. Rodríguez-Vázquez, S. Espejo, R. Domínguez-Castro, J.L. Huertas and E. Sánchez-Sinencio: "Current-Mode Techniques for the Implementation of Continuous-Time and Discrete-Time Cellular Neural Networks", *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, Vol. 40, pp. 132-146, March 1993.

[16] T. Roska and L. Kek, Eds.: "Analogic CNN program library". Analogical and Neural Computing Laboratory, Computer and Automation Institute of the Hungarian Academy of Sciences, Budapest. Report DNS-5-1994, June 1994.

[17] T. Matsumoto, L.O. Chua and H. Suzuki: "CNN Cloning Template: Connected Component Detector". *IEEE Trans. Circuits and Systems*, Vol. 37, pp 633-635, May 1990.

[18] J. M. Cruz and L.O. Chua: "Design of high-speed, high-density CNNS in CMOS technology". *International Journal of Circuit Theory and Applications*, vol.20, no.5, pp.555-572, Sept.-Oct. 1992

[19] P. Földesy, P. Szolgay and A. Zarándy: "Functional testing of the cP 300 14 x 14 CNN UNIVERSAL CHIP". Analogical and Neural Computing Laboratory, Computer and Automation Institute of the Hungarian Academy of Sciences, Budapest. Budapest, 1996.

[20] J. F. Ramos: *"SIRENA: Un Entorno para la Simulación de Redes Neuronales Artificiales. Interfase con el Usuario"*. Proyecto Fin de Carrera, Facultad de Informática y Estadística. Universidad de Sevilla, 1994.

[21] S. Espejo, A. Rodríguez-Vázquez, R. Domínguez-Castro, J.L. Huertas and E. Sánchez-Sinencio: "An Analog Design Technique for Smart-Pixel CMOS Chips". *Proc. of the 1993 European Solid-State Circuits Conference*, Sevilla, September 1993.

[22] S. Espejo, R. Carmona, R. Domínguez-Castro and A. Rodríguez-Vázquez. "Design of Sensory Processing CNN Chips". *Proc. of the International Symposium on Non-Linear Theory and its Applications, NOLTA'93.* Vol. 1, pp 5-10, Hawaii, December 1993.

[23] S. Espejo, A. Rodríguez-Vázquez, R. Domínguez-Castro and R. Carmona: "Convergence and Stability of the FSR CNN Model". *Proceedings of the 3rd International Workshop on Cellular Neural Networks and their Applications, CNNA'94*, pp 411-417, Rome, December 1994.

[24] Chai Wah Wu, L. O. Chua, T. Roska: "A Two-Layer Radon Transform Cellular Neural Network". *IEEE Trans. Circuits and Systems*, Vol. 39, pp 488-489, July 1992.

[25] L.O. Chua, T. Roska, P.L. Venetianer and A. Zarándy: "Some Novel Capabilities of CNN: Game of Life and Examples of Multipath Algorithms". *Proc. Second IEEE Int. Workshop on Cellular Neural Networks and their Applications*, pp 276-281, Munich, October 1992.