2017 14th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), Mexico City, Mexico.
September 20-22, 2017

Time-Multiplexing Cellular Neural Network in FPGA for image processing

J. J. Morales-Romero[1], F. Gómez-Castañeda[1], J.A. Moreno-Cadenas[1]
M.A. Reyes-Barranca[1], L.M. Flores-Nava[1],
[1]Department of Electrical Engineering, CINVESTAV-IPN, Mexico City, Mexico
Phone (52) 55 5747 3800 Ext. 6261
E-mail: {jmoralesr, fgomez, jmoreno, mreyes, lmflores} @cinvestav.mx

***Abstract*— While efficient simulators for Time-Multiplexing Cellular Neural Networks have been reported, no reports on implementations in FPGA have been presented. A Time-Multiplexing Cellular Neural Network is implemented within a FPGA for image processing. The network has been used to perform tasks, such as edge detection and noise remover over several test templates. Implementation results are compared with a simulator using MATLAB and presented in this work showing that this procedure is very reliable for image processing.**

Keywords—Cellular Neural Network, Time-Multiplexing, FPGA, image processing.

I. INTRODUCTION

The Cellular Neural Network (CNN) was first introduced by Chua and Yang in the year 1998 [1]. A CNN is characterized for performing a parallel processing, having a continuous time dynamic and a global interaction among the elements of the network.

Due to the characteristics of the CNN, it has been presented as a good alternative to solve problems that otherwise are complicated to perform using conventional approaches. CNN can be used for processing both binary (black and white) and gray scale images. Examples of these processes are edge detection, noise remover, shadow extractor, global connectivity detector, etc.

To perform the processing of an image, the CNN must have the same size of the image, because the image is mapped one to one with the CNN, that is, one cell is require for each pixel of the image. However, the size of the current images, with thousands of pixels, makes it difficult to have a large CNN, with thousands of cells, because it requires a large consumption of hardware resources to manufacture such CNN with thousands of cells or can even have a large consumption of software resources for its simulation.

There are applications of digital CNN as in [2], however this proposal has the disadvantage that requires processor units (PU), leading to an increase in the consumption of hardware resources; in [3] they use a CNN universal machine (CNN-UM) with 128 cells, however this need a large consumption of hardware resources as well. Therefore, because of the above issues the Time-Multiplexing method has been proposed, which allows the processing of an image using a CNN of reduced dimensions [4-5]. In [6] a simulator of a time-multiplexing CNN has been implemented for edge detection in noisy images.

In this contribution, we present the approach Time-Multiplexing CNN on FPGA, using 16 cells, each cell uses one multiplier and the memories used are used to store the input and output images, and no memory for states storage is required, this CNN can process images up to 244 x 244 pixels.

II. ARCHITECTURE OF CELLULAR NEURAL NETWORK

The basic element of a CNN is called cell, which is interconnected in a similar way to those found in cellular automata, that is, each cell is connected to neighboring cells, as shown in Fig. 1.

Where $i = 1, 2, 3 \ldots, M$ and $j = 1, 2, 3, \ldots, N$ representing the rows and columns of the CNN, respectively.

The neighborhood $N_r(i, j)$ width radius r is the set of cells that are directly connected to cell $C(i, j)$, and is defined by

$$N_r(i,j) = \{C(k,l) | max\{|k-i|, |l-j|\} \leq r\}$$

where $1 \leq k \leq M$, $1 \leq l \leq N$ and r is a positive integer. Fig. 2 shows a cell with its neighborhood.

The dynamics of a cell is defined in (1), which is the normalized state equation [6].

$$\frac{dv_{xij}(t)}{dt} = -v_{xij}(t) + f_{ij}(t) + I_{ij} \qquad (1).$$

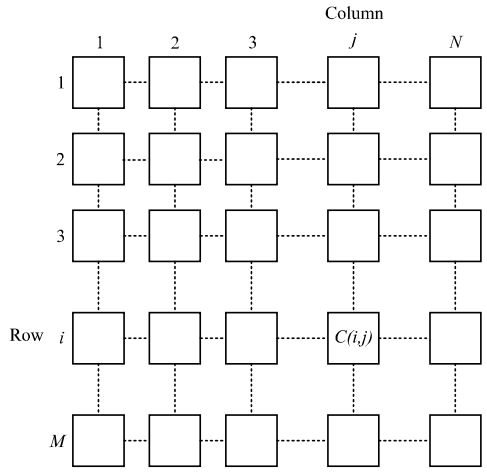


Fig. 1. Structure of a Cellular Neural Network (CNN).

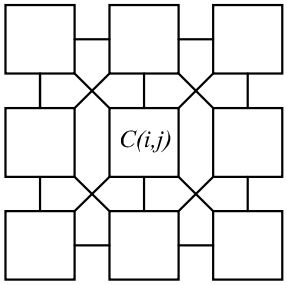


Fig. 2. Neighborhood of the cell *C(i, j)*.

where:

$$f_{ij}(t) = \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)\, v_{ykl}(t)$$

and

$$I_{ij} = \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l) v_{ukl} + I$$

where v_{xij}, v_{uij} and v_{yij} are called state, input and output of the cell $C(i,j)$, respectively; v_{ykl} and v_{ukl} are the output and input of the boundary cell, respectively; I is the bias; $A(i,j;k,l)$ and $B(i,j;k,l)$ are called the feedback and control templates, respectively; (i,j) indicate the index of the cell within the CNN and (k,l) are the indexes of the neighboring cell.

Regarding a CNN image processing, input values v_{uij} must be in the range of -1 to 1, where a value -1 corresponds to a white pixel, a value of 1 corresponds to a black pixel and intermediate values represent a gray scale.

The output of the cell is defined by the symmetric saturating linear transfer function (satlins), whose values are between -1 and 1; this function is defined in (2).

$$v_{yij} = \frac{1}{2}\left(\left|v_{xij}(t)+1\right| - \left|v_{xij}(t)-1\right|\right) \quad (2).$$

To be implemented in programmable logic devices such as FPGA, the differential equation (1) is approximated by a difference equation [1], which result is shown in (3).

$$v_{xij}(n+1) = v_{xij}(n) + h\left[-v_{xij}(n) + f_{ij}(n) + I_{ij}\right] \quad (3).$$

Where n represents the integration step and h is the time step constant.

Templates **A** and **B** together with bias I indicate the task to be performed by the CNN. Since most CNN's applications use only space-invariant CNN, together with a 3x3 neighborhood, we can define templates **A** and **B** by (4) and (5), respectively, as follows.

$$A = \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix} \quad (4).$$

$$B = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix} \quad (5).$$

III. Time-Multiplexing Cellular Neural Network

Time-Multiplexing in a CNN consists in taking blocks of the image to be processed having the same size that the CNN and processing each block in a lexicographic order, saving the results until the complete image has been scanned.

However, this approach leads to two errors in the calculation of border cells in the CNN, since they are calculated without the information of their neighbors. The first error is due to losing the feedback effect of neighboring cells and neglecting the feed-forward effect. The second error is due to losing the feed-forward effect and neglecting the feedback effect of neighboring cells. These errors are calculated in (6) and (7), respectively, using a neighborhood radius equal to one.

$$\varepsilon_{ij}^{A} = \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l) v_{ykl}(t) \quad (6).$$

$$\varepsilon_{ij}^{B} = \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l) sign(v_{ukl}) \quad (7).$$

where $sign(x)$ is called the sign function and is defined in (8).

$$sign(x) = \begin{cases} 1 \; if \; x > 0 \\ 0 \; if \; x = 0 \\ -1 \; if \; x < 0 \end{cases} \quad (8).$$

To reduce the first error, an overlap is performed between two adjacent blocks, as shown in Fig. 3.

For the second error, a belt of cells of the same size as the radius *r* of the original image is placed as shown in Fig. 4.

To save the values obtained by the CNN in the overlap is done as follows: for the block i the left part is saved and for the block i+1 the right side of the overlap is saved (taking as a reference the image shown in Fig. 3).

IV. Implementing Time-Multiplexing Cellular Neural Network in A FPGA

The design of CNN in analog integrated circuit has the advantage of processing an image faster than in a digital circuit. However, it is possible to design a CNN faster than the design of an analog integrated circuit; the design of a CNN on FPGA has the advantages that it can be updated, increased in size and it can change the parameters of the CNN, besides the design in analog integrated circuit is more expensive and it is complicated to change the values of the parameters like the templates **A** and **B** of the CNN. The algorithm implemented in a FPGA for Time-Multiplexing CNN is shown in Table I.

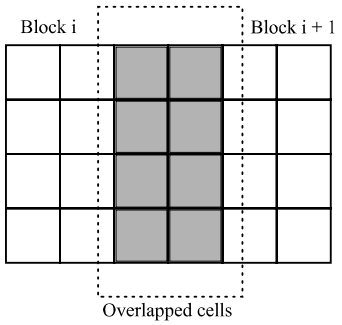


Fig. 3. Overlap between two neighboring blocks.

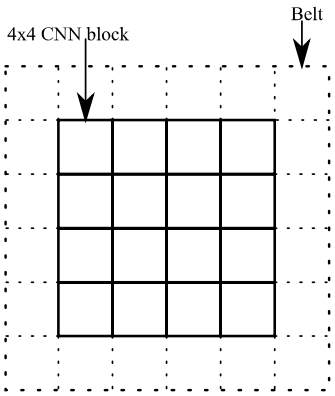


Fig. 4. Belt of cells in a block.

TABLE I. TIME-MULTIPLEXING ALGORITHM FOR FPGA.

Algorithm
1. Read settings. 2. Read input and output image and save them in RAM. 3. Read Block(i, j). 4. Convert input data from 8 bit format to 18-bit format data of Block(i, j). 5. Set data in the 4x4 CNN. 6. Process the 4x4 CNN. 7. Convert output data from 18-bit format to 8-bit format data of CNN. 8. Save the output data in RAM. 9. Was it the last block? Yes: Go to 10. No: Next block. Go to 3. 10. Send to pc the output image.

A grayscale image can have values between 0 and 255, then it is necessary to convert these values to something that can be recognized by the CNN (this is, values between -1 a 1). Equation (9) shows the formula used for this purpose:

$$y = 1 - \frac{2*x}{255} \qquad (9).$$

Where, y is the output and x is the input. To perform the operations in the FPGA, it has been proposed to use an 18-bit size for the operands, due to the use of the Digital Signal Processors (DSP) in the FPGA, which contains the multipliers which can handle this size of word. Specifically, the first bit is used as sign, the next six bits are used for the integer part and the last eleven bits are used for the fractional part. To perform the operation of (9) as mentioned in the step 4 of the algorithm into the FPGA and to reduce the time and resources consumption, the division $2/255$ was replaced by the value 0.007843017578125 and represented in 18-bit format, the first bit of this value represents the sign, the second bit represents the integer part and the last sixteen bits represents the fractional part.

The digital cell implemented into the FPGA is shown in Fig. 5. To process a reliable image, in this study the digital cell has been replicated 16 times to create a 4x4 CNN. These 16 cells work in parallel, which are controlled by finite state machine (FSM). In Fig. 6 the block diagram implemented is shown, where all the functional blocks that perform the Time-Multiplexing CNN are shown.

To check the operation of a digital cell and the complete 4x4 CNN, it has been implemented a simulator CNN in MATLAB, which algorithm is show in Table II. The CNN implemented in FPGA has been simulated using the templates of [1] and compared with the operation of a cell of the simulator and with the operation reported in [1], using the same input data. Fig. 7 shows the results obtained, in 7(a) it is shown the result obtained by the simulator and in 7(b) it is shown the result obtained by the implementation in FPGA, both results have the same behavior as shown in [1].

Next, each block of Fig. 6 is explained. The Transceiver block receives the input image and settings, and sends the output image using the RS232 protocol at a speed of 115200 bauds. The Setting block stores the settings for the CNN, such as the **A** and **B** templates, and I bias, as well as the number of processing blocks. The Random-Access Memory (RAM) block is responsible for storing the input image to be processed and the output of the image processed, internal blocks of RAM of the FPGA are used. The 8 to 18-bit converter and 18 to 8-bit converter blocks are responsible for doing the data type conversion, as mentioned before. The CNN Control block is responsible for controlling the 4x4 CNN. Finally, the Main Control block oversees the entire system.

The size of the image to be processed by the CNN in the FPGA is limited by the capacity of the internal RAM block. In this implementation, the largest image that can be processed is 244x244 pixels. However, for larger images it is possible to apply the approach Time-Multiplexing from Matlab, taking blocks of 244x244 pixels.

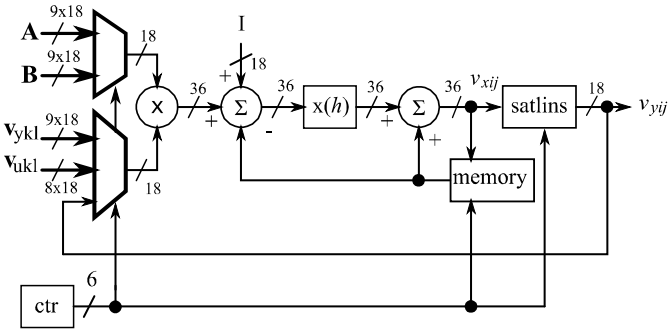


Fig. 5. Block diagram of a digital cell.

TABLE II. TIME-MULTIPLEXING ALGORITHM FOR MATLAB.

Algorithm
1. Read input image. 2. Convert input image to CNN format. 3. Set initial states equal to input image. 4. Read templates **A** and **B**, and bias I. 5. Read the maximum time to process and step time *h*. 6. For t = 0; t < maximum time; t = t + h do 7. For i = 1; i <= N; i++ do 8. For j = 1; j <= M; j++ do 9. Calculate state of C(i, j) and store. 10. Apply satlins to states of CNN and save in output image. 11. Show results.

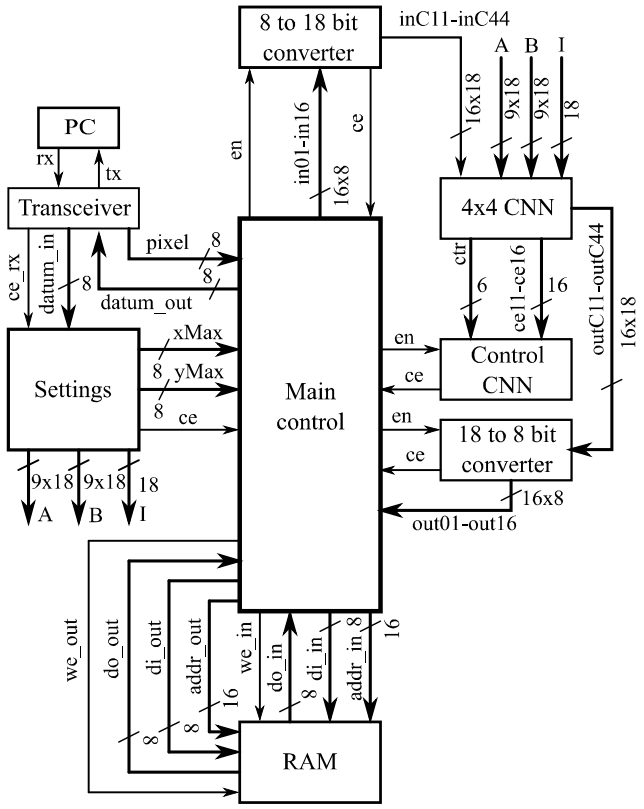
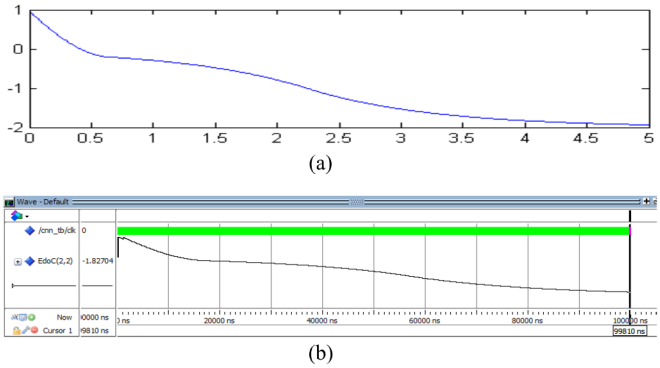


Fig. 6. Block diagram of the Time-Multiplexing CNN for image processing.



(a)

(b)

Fig. 7. Behavior of the digital neuron. (a) Simulation of cell in MATLAB (b) Simulation of cell in FPGA.

V. RESULTS

The development board used was Atlys™ Spartan 6 FPGA, in which the entire system for Time-Multiplexing CNN was synthesized, consuming the resources shown in Table III. To compare the results obtained from the FPGA, it was implemented a CNN simulator for image processing in MATLAB, however this simulator does not use the Time-Multiplexing approach.

Fig. 8 shows the processed image with template and the implemented task was for edge detection [8]. In 8(a) the original image is shown; in 8(b) the result after the simulation in MATLAB is shown and in 8(c) the result after the image was processed by the FPGA. The image has a size of 384x384 pixels, however to solve this, it was implemented the time-multiplexing from MATLAB, taking a block of size 244x244 pixels; the processing time obtained with the CNN implemented in the FPGA was 30.71 seconds, since the image was sent from the PC to the FPGA and back from the FPGA to the PC, however the real time taken to process the image inside the FPGA was 5.09 seconds, the remaining time corresponds to sending and receiving data; in contrast, the simulation time in MATLAB was 280.90 seconds.

Fig. 9 shows another processed image, with templates to which the task of noise remover was implemented [9]. The size of this image is 287x349 pixels; in this case, the processing time was 19.77 seconds, since the image was sent from the PC to FPGA and back from the FPGA to PC as well, however, the time taken to process the image inside the FPGA was 2.37 seconds, again the remaining time corresponds to sending and receiving data; for this test, the simulation time was of 182.49 seconds. Once again having a processing time well above from that of the CNN implemented with the FPGA. In 9(a) the original image is shown, in 9(b) the simulation made in MATLAB is shown and in 9(c) the result after this image was processed by the FPGA.

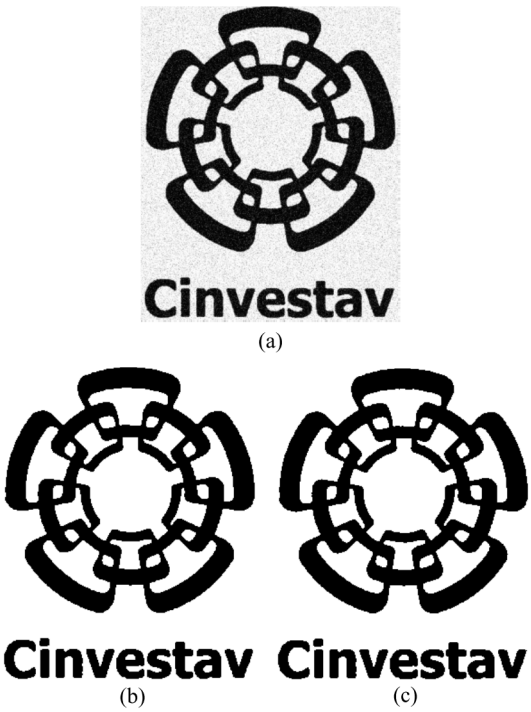
TABLE III. RESOURCES USED BY THE FPGA.

Device Utilization			
Logic Utilization	***Used***	***Available***	***Percent used***
Number of Slices Registers	3896	54576	7%
Number of Block RAM	52	116	44%
Number of DSP48A1s	19	58	32%

(a)

(b) (c)

Fig. 8. Edge detection using Time-Multiplexing CNN in FPGA (a) Input image (b) Result after simulation in MATLAB (c) Result after processed by FPGA.



(a) (b) (c)

Fig. 9. Noise remover using Time-Multiplexing CNN in FPGA. (a) Input image (b) Result after simulation in MATLAB (c) Result after processed by FPGA.

From these results, the hardware implementation of a CNN to make at least the task considered in this work, has its advantages over the simulated implementation, mainly because of the few resources used from the FPGA. Besides, regarding the hardware implementation results, one should have in mind that a great part of the time taken to finally process each image, is due to the transmission time between the PC and the evaluation board used for this implementation.

However, in these applications the images are bigger than the capacity that the proposed implementation can support Therefore, to solve this issue, the Time-Multiplexing approach has been implemented from the PC.

CONCLUSION

As can be seen from the results obtained in this work, using a Time-Multiplexing CNN on FPGA for image processing is attractive, since its reduced resources consumption is achieved compared when performing a CNN of the same size as the processed image. This method also has the advantage that it can process images of almost any size, limited only by the capacity of the available RAM, however, to solve this problem an external Time-Multiplexing can be performed, as shown above.

As mentioned before, the implementation of the CNN on FPGA has the advantage that it can be updated according to the size of the image that will be processed and the amount of resources needed for each task. This means that the size of the CNN can be increased; this reduces the number of blocks to be processed, decreasing the processing time. For the moment, a couple of task were tested but this can be extended to other ones, and a study of the time taken for the image processing depending on the algorithm and hardware can be done as well in the future.

REFERENCES

[1] Leon O. Chua and L. Yang, “Cellular Neural Networks: Theory & Applications”, *IEEE Trans. Circuits and Systems*, vol. CA-35, 1988, pp. 1257-1290.

[2] K. Kayaer and V. Tavsanoglu, “A new approach to emulate cnn on fpgas for real time video processing”, *The 11th International Workshop on Cellular Nanoscale Networks and their Applications*, 2008.

[3] Jens Müller, R. Becker, Jan Müller and R. Tetzlaff, “Cesar: Emulating cellular networks on fpga”, *The 13th International Workshop on Cellular Nanoscale Networks and their Applications*, 2012.

[4] Chi-Chien and J. Pineda de Gyves, “Time-Multiplexing CNN Simulator”, *IEEE Inter. Symp. On Circuits and Systems*, vol. 6, 1994, pp. 407-410.

[5] A. A. H. EL-Shafei and M. I. Sobhy, “A Time-Multiplexing simulator for Cellular Neural Network (CNN) using SIMULINK”, *IEEE Inter. Symp. On Circuits and Systems*, vol. 3, 1998, pp. 167-170.

[6] A. A. H. EL-Shafei and M. I. Sobhy, “A Time-Multiplexing simulator for Cellular Neural Network (CNN)”, *Cellular Neural Networks and Their Applications Proceedings*, 1998 Fitfth IEEE International Workshop on, pp. 14-18.

[7] V. Murugesh, V. Arthy and V. Agalya, "Edge detection of noisy images based on time-multiplexing CNN simulator," *Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT),* Hefei, 2014, pp. 1-5.

[8] Leon O. Chua and Tamás Roska, “Cellular neural networks and visual computing: Foundations and applications”, Cambridge U.K., Cambridge Iniversity Press 2004, Ch. 2, pp. 7-34.

[9] J. Ezequiel Molinar Solís, “Circuito integrado Análogico CMOS con arquitectura de Red Neuronnal Celular”, M.C. I.E. CINVESTAV, México D.F