2010 International Symposium on Nonlinear Theory and its Applications
NOLTA2010, Krakow, Poland, September 5-8, 2010

Architecture of The Next Generation Real Time CNN Processor: RTCNNP-v2

Evren Cesur†, Nerhun Yildiz† and Vedat Tavsanoglu†

†Dept. of Electronics and Communications Engineering, Yildiz Technical University
Barbaros Bulvari, 34349, Besiktas, Istanbul, Turkey
Email: nerhuny@yildiz.edu.tr, ecesur@yildiz.edu.tr, tavsanav@yildiz.edu.tr

Abstract—This paper is the continuation of our previous work reported in [1], where the local control structure of the *Second–Generation Real–Time Cellular Neural Network Processor* (RTCNNP-v2) was covered. The system is primarily designed for high resolution, high speed real–time video image processing. In this paper the block diagram of the new processor and an improvement over the local control structure are presented. The proposed structures are coded in VHDL and realized on an FPGA.

1. Introduction

Standard Cellular Neural Network (CNN) architecture is a two dimensional array of parallel processors [2]. In analog implementations the processing elements are neural cells which are organized as two dimensional spatial grids with temporal analog memory nodes. Spatial interconnections of the neighboring neurons define the spatio-temporal dynamics of the grid. The output is taken from a non-linear activation function which vary for different applications [3].

While the analog architecture has many benefits like high speed and low power dissipation, digital emulation methods are becoming more feasible as the digital technology tends to be faster and cheaper. Different CNN emulation infrastructures are proposed in the literature such as FPGAs, ASICs, DSPs, GPUs, etc.. Among the alternatives the FPGAs are preferred for our implementation, as their speed is comparable with ASICs, regularity is similar to the GPUs, and they are highly parallel, reconfigurable and reusable [4], [5].

This paper is the continuation of [1] where the first design specifications of the *Second–Generation Real–Time Cellular Neural Network Processor* RTCNNP-v2 are proposed. The general structure of the design is based on the first–generation processor, RTCNNP-v1 [6]. The system is designed as a real–time, high speed, high resolution video processing array.

First a brief mathematical overview of the CNN model used in both RTCNNP versions is given. Then the architecture and the control structure of the processor array is presented. Finally the FPGA realization results are discussed.

2. Mathematical Overview

Mathematical model of the CNN depends on the number of spatial dimensions, activation function, number of layers, etc.. Although the proposed architecture is capable of processing many variations of the CNN with some modifications and adjustments, the simpler two dimensional, single layer, space invariant CNN model using saturated linear activation function is used for simplicity [3].

2.1. Continous–Time CNN Model

The mathematical model of the continuous–time CNN is defined with the state equation

$$\begin{aligned} \frac{dx_{ij}(t)}{dt} &= -x_{ij}(t) + \boldsymbol{A} \circledast \boldsymbol{Y}_{ij}(t) + \boldsymbol{B} \circledast \boldsymbol{U}_{ij} + z_{ij}, \quad (1) \\ y_{ij}(t) &= f(x_{ij}(t)) = 0.5\left(\left|x_{ij}(t)+1\right| - \left|x_{ij}(t)-1\right|\right) \end{aligned}$$

where x_{ij} is the state, y_{ij} is the output, z_{ij} is the threshold, $\boldsymbol{Y}_{ij}$ is the translated masked output, $\boldsymbol{U}_{ij}$ is the translated masked input of the $_{ij}$'th cell, $\boldsymbol{A}$ is the feedback and $\boldsymbol{B}$ is the feed–forward cloning templates, $\circledast$ is the template–dot–product operator, i and j are the index variables of the neural cell.

2.2. Discrete–Time and Full Signal Range CNN Models

The discretization of (1) with forward–Euler approximation

$$\frac{dx_{ij}(t)}{dt} \cong \frac{x_{ij}^{n+1} - x_{ij}^{n}}{T_s} \quad (2)$$

yields

$$x_{ij}^{n+1} = x_{ij}^{n} + T_s\left(-x_{ij}^{n} + \boldsymbol{A} \circledast \boldsymbol{Y}_{ij}^{n} + \boldsymbol{B} \circledast \boldsymbol{U}_{ij} + z_{ij}\right). \quad (3)$$

Although direct implementation of (3) is possible, Full Signal Range (FSR) CNN model is easier to implement [7]. The FSR model is obtained by taking $x_{ij}^{n} = y_{ij}^{n}$ in (3) as

$$x_{ij}^{n+1} = (1 - T_s)y_{ij}^{n} + T_s\boldsymbol{A} \circledast \boldsymbol{Y}_{ij}^{n} + T_s(\boldsymbol{B} \circledast \boldsymbol{U}_{ij} + z_{ij}). \quad (4)$$

This equation can be written as

$$x_{ij}^{n+1} = \bar{\boldsymbol{A}} \circledast \boldsymbol{Y}_{ij}^{n} + \bar{\boldsymbol{B}} \circledast \boldsymbol{U}_{ij} + \bar{z}_{ij} \quad (5)$$

where

$$\bar{\boldsymbol{A}} \circledast \boldsymbol{Y}_{ij}^{n} = (1 - T_s)y_{ij}^{n} + T_s\boldsymbol{A} \circledast \boldsymbol{Y}_{ij}^{n},$$
$$\bar{\boldsymbol{B}} = T_s\boldsymbol{B}, \qquad \bar{z}_{ij} = T_s z_{ij}.$$

2.3. Decomposition of the CNN Model

In equation (5) the term

$$g_{ij} = \bar{\boldsymbol{B}} \circledast \boldsymbol{U}_{ij} + \bar{z}_{ij} \tag{6}$$

is calculated only once for each frame and used as a constant in every Euler iteration

$$x_{ij}^{n+1} = \bar{\boldsymbol{A}} \circledast \boldsymbol{Y}_{ij}^{n} + g_{ij}. \tag{7}$$

The equations (6) and (7) can be mapped to *B–Processing Unit* (BPU) and *A–Processing Unit* (APU) respectively. The computation flow in time can be given as

$$\begin{aligned} \text{BPU} &= g_{ij} = \bar{\boldsymbol{B}} \circledast \boldsymbol{U}_{ij} + \bar{z}_{ij} \\ \text{APU(1)} &= x_{ij}^{1} = \bar{\boldsymbol{A}} \circledast \boldsymbol{Y}_{ij}^{0} + \text{BPU} \\ \text{APU(2)} &= x_{ij}^{2} = \bar{\boldsymbol{A}} \circledast \text{APU(1)} + \text{BPU} \\ &\ldots \\ \text{APU(N)} &= x_{ij}^{N} = \bar{\boldsymbol{A}} \circledast \text{APU(N-1)} + \text{BPU} \end{aligned} \tag{8}$$

where N is the total number of Euler iterations. In short each Euler iteration is carried out by a different processor. The computed state is passed through the activation function at the output of each processor before entering to the next one. The initial condition $\boldsymbol{Y}_{ij}^{0}$ is either a constant or the input image for the most CNN algorithms.

3. Structure of The RTCNNPv2

The system consists of a video source, video decoder, FPGA, video encoder and a video sink (Figure 1). The video source can be any analog (VGA) or digital (DVI, HDMI, etc.) progressive video stream. The video sink may also be any analog or digital monitor, TV, etc. that accepts progressive video stream.

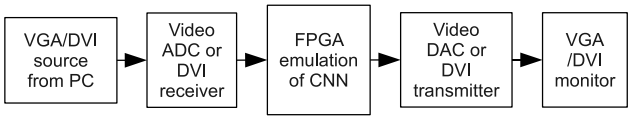


Figure 1: Block diagram of the system.

3.1. General Structure

A feed–forward video processing array is embedded in an FPGA that emulates the CNN (Figure 2). *Video input* block accepts control signals from the video decoder and translates them for the video processing array. Similarly, the *video output* block translates the control signals according to the video sink specifications.

Video processing array is an array of BPU and APU processors. The flow is organized according to (8). The BPU result is propagated through APUs. Each APU takes the results of the previous APU and the BPU as inputs and gives the new APU and the old BPU results to the next APU.

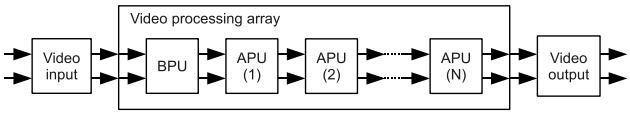


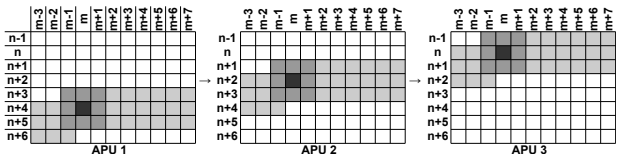
Figure 2: Block diagram of the FPGA implementation.

BPU and APU processors are designed as a single programmable Processing Unit (xPU) (Figure 4). *APU/$\overline{BPU}$* input configures the processor as BPU or APU. *clk*, *clka* and *clkp* clocks are the processing, programming and pixel clock respectively. *data_in*, *const_in*, *data_out* and *const_out* are the data inputs and outputs. Their functionality depends on the xPU type, BPU or APU. *sdata_in* and *sdata_out* are the serial programming input and output respectively. *hframe_in*, *hframe_out*, *vframe_in* and *vframe_out* signals are the control signals that control the xPUs and propagate through APUs.

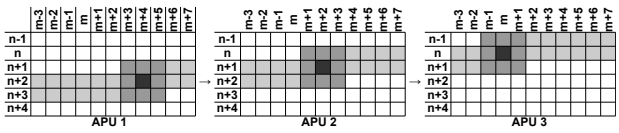
3.2. Local Control Structure

The main problem of controlling a processor array is the complexity of the central control unit. Every address input and every multiplexer select signal of each BPU and APU should be controlled independently and synchronously.

For 1–neighborhood CNN, consider the optimized and non–optimized data RAM structures in three consecutive APUs (Figure 3). The non–optimized memory structure is easier to control as the pixels at the same column are being processed by the processors. The optimized memory structure consumes less RAM in exchange for the control complexity as every control signal of each processor is different from one another. Furthermore, the control signals will be affected if the internal structure of the processors or number of iterations change.



(a) Non–optimized memory structure



(b) Optimized memory structure

Figure 3: Memory usage of consequent APUs (light gray), and pixels that are being processed (dark gray).

As seen from our previous processor RTCNNP-v1, the Block RAMs (BRAMs) and multipliers are the bottlenecks of such FPGA implementations and only about 30% of the logic blocks are used beside them [6]. This fact suggests

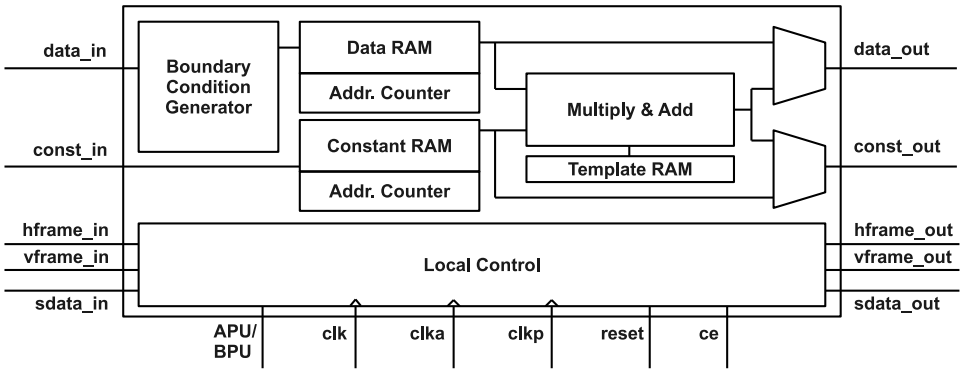


Figure 4: Simplified block diagram of the xPU.

that there are plenty of logic elements to use in the processors. Consequently, a local control structure is proposed instead of a huge, complex and non–flexible central control unit [1].

There are two control signals that control each xPU: horizontal frame signal *hframe* and vertical frame signal *vframe*. There is a blank area around the real video signal defined by the video standards. In this case there are nine different areas on the frame (Figure 5). *hframe* and *vframe* control signals define not only the visible area, but also define starting and ending points of the frame and lines, required for control. Video input block generates these control signals and controls the BPU, BPU propagates these signals according to its latency and control the first APU, and so on. Finally video output block receives these signals and translates them to standard video control signals of DVI, VGA, etc..

hframe = 0 vframe = 0	hframe = 1 vframe = 0	hframe = 0 vframe = 0
hframe = 0 vframe = 1	hframe = 1 vframe = 1 (visible area)	hframe = 0 vframe = 1
hframe = 0 vframe = 0	hframe = 1 vframe = 0	hframe = 0 vframe = 0

Figure 5: Visible and blank areas of standard video signals, and the definition of the *hframe* and *vframe* control signals.

Data and constant RAMs are optimized as seen in Figure 3b. They generate their own addresses, but reset and enabled by the local control. The RAM structure looks like circular buffers or shift registers from the functional perspective. There is also a boundary condition generation block that passes the input data or the boundary condition to the data RAM.

Frequency of the processing clock *clk* is obtained by multiplication of the pixel clocks *clkp* with a positive integer. By making the processing clock faster means less multiplier resources may be used, e.g. if clock frequency multiplier is 3, 3 multipliers are enough to calculate 9 multiplications in 3 clocks instead of 9 multiplications in 1 clock. *clka* is the auxiliary clock used for serial programming of the template RAM, boundary condition, etc.. The serial programming signal is also propagated through the processors in order to avoid long global interconnections.

The data and constant outputs are defined as

$$\begin{aligned} data\_out &= data\_in, \\ const\_out &= data\_in \circledast \bar{\boldsymbol{B}} + const\_in \end{aligned}$$

for the BPU. The data output is the initial condition of the first APU, which is the input pixel in many cases, and the processed data is the constant input of the first APU. On the other hand, for APUs

$$\begin{aligned} data\_out &= data\_in \circledast \bar{\boldsymbol{A}} + const\_in, \\ const\_out &= const\_in \end{aligned}$$

as the processed data is the next APU's data input, and the constant output is the BPU result that propagates through the APUs.

4. FPGA Implementation Results

The proposed system is codded in VHDL and partially implemented on a high–end *Altera® Stratix© IV GX 230* FPGA. The unimplemented part is the serial programming feature. Template and boundary conditions are entered directly into the VHDL code and fixed during the synthesize operation.

Templates of the *Global Connectivity Detection* is chosen for the implementation as all synaptic weights are non–zero and the algorithm is suitable for prototyping [9]. The templates and bias are

$$\bar{\boldsymbol{A}} = \begin{bmatrix} 6 & 6 & 6 \\ 6 & 9 & 6 \\ 6 & 6 & 6 \end{bmatrix}, \quad \bar{\boldsymbol{B}} = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 9 & -3 \\ -3 & -3 & -3 \end{bmatrix}, \quad \bar{z} = -4.5 \tag{9}$$

which are represented with signed 16-bits. The input and the intermediate states are chosen to be 8-bits, and the BPU output is propagated as 16-bits.

Table 1: Resource usage of the prototype.

Blocks	Combinational ALUTs	Dedicated Logic Registers	Block Memory (Kb)	DSP 18-bit Elements
One xPU	221-380 (0.1-0.2%)	370-491 (0.2-0.3%)	minimum 8 (0.6%)	6 (0.5%)
100 xPUs	23635 (12.9%)	35651 (19.6%)	871 (67.6%)	600 (46.6%)
Glue logic	1072 (0.6%)	668 (0.4%)	0 (0%)	0 (0%)
Total	24707 (13.5%)	36319 (19.9%)	871 (67.6%)	600 (46.6%)

The properties of the test video signal are Full HD 1080p@54Hz (1920x1080) with reduced blanking 134 MHz pixel clock frequency. The clock frequency multiplier is 2 that gives 268 MHz processing clock that is easier to synthesize with little or no further optimization. 99 Euler iterations are embedded in the FPGA, hence 99 APUs and 1 BPU resides in the prototype (100 xPUs).

The system is fully pipelined. The throughput of the system is the same as its input. The latency is about 200 lines of a video frame for 100 iterations, which corresponds to 3.5 ms for the video properties given above.

Resource usage statistics are given in Table 1. The number of multipliers used in each xPU should be even due to the architectural limitations of the FPGA. Hence even if 5 multipliers are sufficient to do 9 multiplications in 2 clock cycles in theory, the architecture of the FPGA forces the usage of 6 multipliers.

5. Conclusion

Structure of the next generation RTCNNP-v2 is presented with more details. The structure is also codded in VHDL and a prototype is realized on an FPGA. Only serial configuration feature is not implemented. Central address and control generation is not necessary as each block controls the next block in a synchronous pipelined manner. The local control design has proved to make the processor array fast, reconfigurable and reusable.

It is also worth to state that this is the fastest CNN implementation for the time being that even outperforms the analog CNN chips in many algorithms. Although analog CNN makes the processing in almost zero time, video input/output is their primary bottleneck. This design targets specifically the input/output data bandwidth problem. 100 CNN iterations of the 54 Frames Full HD 1080p video signal means that 100 billion Multiply–ACcumulate (MAC) operations or 200 billion total additions and multiplications are done in one second. It means that the system has 1/5'th the computational power of a supercomputer.

Finally this design is implemented on a high-end FPGA for benchmark purpose. It is possible to do downscaling in order to fit the design in a smaller FPGA with few changes.

Acknowledgment

This research was supported by The Scientific and Technological Research Council of Turkey — TÜBİTAK under project number 108E023.

References

[1] N. Yildiz, E. Cesur and V. Tavsanoglu, "A New Control Structure For The Pipelined CNN Processor Arrays," *12th IEEE CNNA — International Workshop on Cellular Nanoscale Networks and their Applications*, Berkeley, US, February 2010.

[2] L.O. Chua and L. Yang, "Cellular Neural Networks: Theory", *IEEE Transactions On Circuits and Systems*, Vol. 35, No. 10, October 1988.

[3] L.O. Chua and T. Roska, "Cellular Neural networks and Visual Computing," *Cambridge University Press*, UK, 2002.

[4] Z. Nagy and P. Szolgay, "Configurable multilayer CNN-UM emulator on FPGA," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, June 2003.

[5] J. Javier Martínez, F. Javier Toledo, E. Fernández and J. M. Ferrández, "A retinomorphic architecture based on discrete–time cellular neural networks using reconfigurable computing," *Neurocomputing*, 2008.

[6] K. Kayaer and V. Tavsanoglu, "A New Approach to Emulate CNN on FPGAs for Real Time Video Processing," *11th International Workshop on Cellular Neural Networks and their Applications*, Santiago de Compostela, Spain, July 2008.

[7] S. Espejo, A. Rodriguez-Vázquez, R. Domínguez-Castro and R. Carmona, "Convergence and Stability of the FSR CNN Model," *3rd IEEE International Workshop on Cellular Neural Networks and their Applications*, Italy, December 1994.

[8] N. Lawal, M. O'Nils, "Embedded FPGA memory requirements for real-time video processing applications", *NORCHIP Conference*, 2005.

[9] R. Matei, "New Image Processing Tasks On Binary Images Using Standard CNNs," *Proceedings of the International Symposium on Signals, Circuits and Systems, SCS'2001*, pp.305-308, July 2001, Romania