Implementation of One Dimensional CNN Array on FPGA - A Design Based on Verilog HDL

Alireza Fasih
Transportation Informatics Group
fasih@qazviniau.ac.ir

Jean C. Chedjou
Transportation Informatics Group
University of Klagenfurt
Klagenfurt-Austria
jean.chedjou@uni-klu.ac.at

Kyandoghere Kyamakya
Transportation Informatics Group
University of Klagenfurt
Klagenfurt-Austria
kyandoghere.kyamakya@uni-klu.ac.at

***Abstract*— In this paper an FPGA based Implementation of a 1D-CNN with a 3×1 template and 8×1 length will be described. The Cellular Neural Networks (CNN) is a parallel processing technology that has been generally used for image processing. This system is a reduced version of a Hopfield Neural Network. The local connections between a cell and the neighbors in this implementation of this technology is easier than in the case of Hopfield Neural Networks. There are various implementation options of CNN on chips, the best solution being using ASIC technology. The next best is an emulation on top of a digital reconfigurable chip such as FPGA. Designing and developing universal CNN based machines using these technologies is possible. Since FPGAs are COTS components and their growth is high, a simple and economical architecture is obtained by designing an CNN emulation on FPGA chips. This digital designing on FPGA does however have a tradeoff between speed and area. One key target is therefore to reach to a best performance for this emulation architecture under the mentioned constraints.**

Keywords— Cellular Neural Network; CNN Emulation on FPGA; Simulation; HDL.

I. INTRODUCTION

This paper briefly introduces a the design of digital emulation of CNN based on hardware description languages. Cellular Neural Network has been introduced by Chua and Yang from the University of California at Berkeley in 1988 [4]. This type of neural networks is a reduced version of Hopfield Neural Networks. One of the most important features of CNN is the local connectivity; in this technology each cell is connected only to its neighbor cells. Due to local connection between a cell and the neighbors a hardware implementation of this type of neural networks is easily realizable [5]. By digitizing the analog behavior of this system (i.e. emulation on a digital platform) one is able realize this system based on FPGA. Due to the local connectivity and processing around each cell the global system works like a parallel processor system [6].

The behavior of a CNN system is based on the settings of the template values. By changing these template values the CNN behavior is affected. This is a very feature for realizing a universal machine by using CNN [7]. A test-bed for CNN emulation on an FPGA evaluation board is a relatively cheap option and the required development time should be significantly low.

II. CNN DIGITAL EMULATION/MODELLING

CNN mathematical models for each cell are first-order equations like Eq-1 shown below.

Eq-1:

State Equation

$$C\frac{dv_{xij}(t)}{dt} = -\frac{1}{R_x}v_{xij}(t) + \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)v_{ykl}(t) + \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)v_{ukl} + I$$

$$1 \le i \le M\,; 1 \le j \le N$$

For modeling this equation in HDL, we must simplify nonlinear terms. The simplified equation takes the form of equation (Eq-2) as following.

Eq-2:

$$\dot{x} = -x + A * v_y + B * v_u + I$$

In this equation 'A' is a template for feedback operator and 'B' is a template for control.

Eq-3:

Output Equation :

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

$$1 \le i \le M\,; 1 \le j \le N$$

The output Equation is a linear sigmoid function for limiting the output state value. In some references the sigmoid function is noted by f (.).

Converting equation-2 to a discrete time model is possible. A Discrete Time CNN can easily be mapped to an FPGA by defining digital integrators, multipliers, adders and other digital operators. After defining fundamental operators in FPGA and wiring of/between these operators, a dynamic modeling of CNN is possible [7]. A single CNN cell model is like a first-order differential equation; therefore, solving this equation by this architecture is possible.

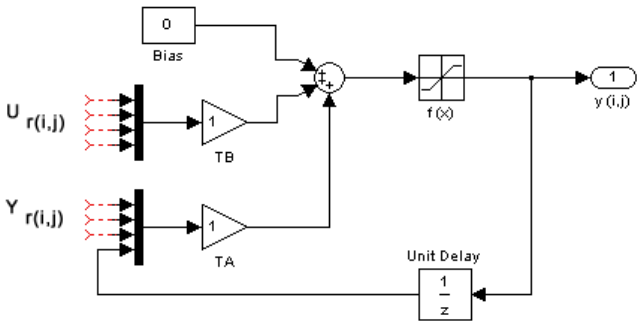

Figure 1. Simplify CNN Cell Dynamic Model

The architecture of this module consists of two main parts, hardwire and behavioral sequential units. In the hardwire part we must define the relationships of the CNN cells with their neighbors by the template values. These connections are based on convolution operators. With array cloning of convolution modules in HDL we are able to develop/extent the size of the CNN module. Using this approach we can realize a growth of the CNN structure up to a simple 8×1 width. In the CNN structure, there are 2 type templates: control templates and feedback templates. Due to the architecture of the feedback path we have to define a memory block for a appropriate handling of this path. Another main memory unit is defined for integrators components. In this system, all convolution unit should work together concurrently. Therefore the results of TA*Y and TB*U are immediately accessible (see Fig.1).

In the top-level CNN module Verilog code defines 16-Bit 2's complement variables for loading data and templates to this module. One bit for sign and 3 bit for round value and 12 bit float value. Therefore, we are able to load values to this module in the range of [-7, +7]. For the 12 bit fixed float register, the accuracy is 1/ (2^{12}). Input data range is limited to the range of [-1, +1]; -1 means black and +1 is white value. This procedure means that for image processing purposes we must rescale the image values to this range. On the other hand, value of each gray pixel must be in range of [-1, +1]. According to the Equation-4, we are able to normalize the input data.

Eq-4:

$$U = Pixel\_value * 2 - 1$$

The convolution module loads template values and inputs data and then return the product of these values. The following block diagram shows the convolution operator I/O (see Fig.2). We set zero for out of bound values in the CNN array. The module cloning based on this diagram is simple in HDL code.

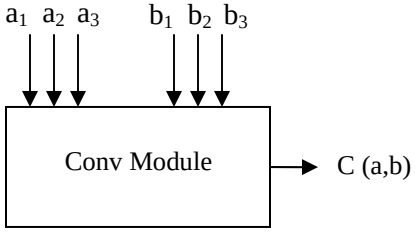

Figure 2. Convulotion I/O Block Diagram

This module of Fig.2 operates according to Equation-5.

Eq-5:

$$C(i,j) = \sum_{i=1}^{3} (a_i \times b_i)$$

This module is common for TA*Y and TB*U.

The code below shows the calling conv2 function for solving the convulotion between Control Template and Input Data on cell 7:

```
conv2 ccn7 (M0[7],VB1,VB2,VB3,16'd0,u7,u6)
```

Further, the convolution on feedback template and output state is similar to the code below:

```
conv2 fcn7 (M1[7],VA1,VA2,VA3,16'd0,Y[7],Y[6])
```

More details on the convolution module are presented in the paper appendix. Other main important units for developing this module are the integrator and linear sigmoid function. To implement an integrator in HDL we need a register. In previous steps we determined the convolution for feedback and control templates. In the integrator unit we must sum the result in each new cycle with previous values of the register. The convolution module's length defines on 18bits. According to the length of M0 and M1, the length of the integrator register should be 32 bit. The following code below is

obtained after synthesis and is like an 8 integrator that work concurrently.

```
always @(posedge clk)
begin
for (j=0;j<=7;j=j+1)
begin
  res[j] = S2[j];
end
```

In this code the term of S2 is the sum of C(TA,Y) and C(TB,U) from the previous cycle.

The best method to design a sigmoid function is to use an if-then rule. The following code below shows the way this unit operates. “Greater than values” will be limited by this procedure between +1 and -1.

```
for (j=0;j<=7;j=j+1)
begin
if (res[j]>32'sh00000_000)  // > 0
     begin
        Y[j]=16'h1_000;     // +1
        res[j]=32'h00001_000;
     end
     if (res[j]==32'sh00000_000) // = 0
     begin
        Y[j]=16'h0_000;      // 0
        res[j]=32'h00000_000;
     end
     if (res[j]<32'sh00000_000)  // < 0
     begin
        Y[j]=16'hf_000;      // -1
        res[j]=32'hfffff_000;
     end
End
```

In these units the “res” vector is a temporary register for simulating the integrator and “Y” variable is a memory for storing CNN output state. We used 2 level memories for designing totally a simple 8×1 CNN..

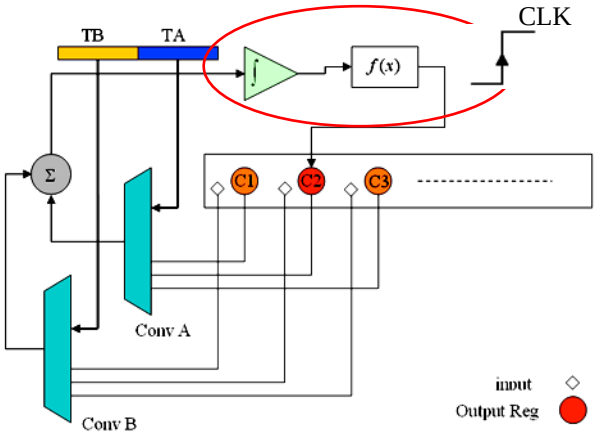

Figure 3. Digital Architecture of CNN

Figure 3 shows the digital architecture of a CNN cell introduced in this paper. According to this architecture the system is synchronous. Parts of integrator, sigmoid function and loading state variable are triggered by a rising clock.

III. Simulation Results

To simulate this system we use ModelSim 6 software. The result is very closed to the one obtained from a simulation in Matlab.

For example, by setting TA= [0.5, +1, -1] and TB=Bias=0, the following wave form appears. The converge time for this case in this architecture is 200ns (8 clock).

u= [+1,-1,-1, +1, +1,-1,-1, +1]

TA= [0.5, +1, -1]

TB=Bias=0

The output for this defined template is

xt=[+1,-1,+1,-1,+1,-1,+1,-1]

The converge time for the case below in this architecture is 150ns (6 clock).

u= [+1,-1,-1, +1, +1,-1,-1, +1]

TA= [-1, +2, +1]

TB=Bias=0

The output for this defined template is

xt=[+1,+1,+1,+1,-1,+1,-1,+1]

In an advanced mode, there is another way for simulating the HDL code. In Matlab 2006a we have been able to establish a connection between Modelsim simulator and Simulink.

By a TCP/IP connection between Simulink in Matlab and the Modelsim simulator we were able to test this CNN code on Images. We must operate this CNN module on each Image line separately.

TA= [-1, +2, +1]

TB=Bias=0

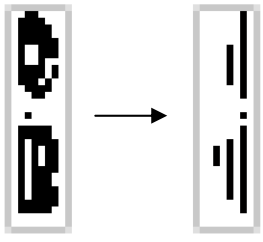

Figure 4. Hole Counting Sample, (Left) Input Image, (Right) output result.

TA= [+1, -1, +1]

TB= [0, 1, 0]

Bias = 0

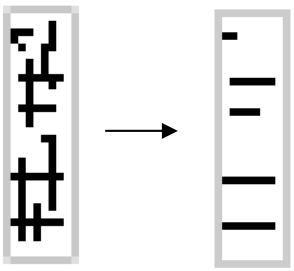

Figure 5. Filter Sample: (Left) Input Image, (Right) output result.

IV. CONCLUSION

In this paper we introduced a new model for simulation and digital implementation/emulation of digital CNN. Finally, the model could be simulated and validate by several templates. Future works are going to improve this module to realize a large-scale CNN based universal machine system.

REFERENCES

[1] Martinez, J. J., F. J. Toledo, et al. "New emulated discrete model of CNN architecture for FPGA and DSP applications." Lecture notes in computer science: 33-40.

[2] T. Roska, A. Rodriguez-Vazquez, “Review of CMOS implementations of the CNN universal machine-type visual microprocessors”, IEEE Int. Symp. on Circuits and Systems, ISCAS 2000, Geneva-Italia, vol. 2, pp. 120-123, 2000.

[3] Chua, L. O. and L. Yang (1988). "Cellular neural networks: applications." Circuits and Systems, IEEE Transactions on **35**(10): 1273-1290.

[4] Chua, L. O. and L. Yang (1988). "Cellular neural networks: theory." Circuits and Systems, IEEE Transactions on 35(10): 1257-1272.

[5] J.Zhao, Q. Ren, J. Wang, and H. Meng,"A New Approach for Image Restoration Based on CNN Processor",ISNN 2007, Part III, LNCS 4493, pp. 821–827, 2007.

[6] Toledo, F. J., J. J. Martínez, et al. (2005). "Image processing with CNN in a FPGA-based augmented reality system for visually impaired people." 8º Int. Work-Conference on Artificial and Natural Neural Networks, IWANN: 906-912.

[7] Martinez-Alvarez, J. J., F. J. Garrigos-Guerrero, et al. "High Performance Implementation of an FPGA-Based Sequential DT-CNN."

[8] Eric Y. Chou, Bing J. Sheu, Topzy H. Wu, Robert C. Chang ,"VLSI Design of Densely-Connected Array Processors", Proceedings of the International Conference on Computer Design: VLSI in computers & Processor (ICCD '95)

[9] Lai, K. and P. Leong "Implementation of Time-Multiplexed CNN Building Block Cell." Proc. MicroNeuro 96: 80-85.

[10] Sadeghi-Emamchaie, S., G. A. Jullien, et al. (1998). "Digital arithmetic using analog arrays." VLSI, 1998. Proceedings of the 8th Great Lakes Symposium on: 202-205.

[11] Espejo, S., A. Rodriguez-Vazquez, et al. (1994). "Smart-pixel cellular neural networks in analog current-mode CMOStechnology." Solid-State Circuits, IEEE Journal of 29(8): 895-905.

APPENDIX

// 1×3 Convolution Module

```
module conv2 (conv,VA1,VA2,VA3,Y1,Y2,Y3);
    output [17:0] conv; //17
    input [15:0]VA1;
    input [15:0]VA2;
    input [15:0]VA3;
    input [15:0]Y1;
    input [15:0]Y2;
    input [15:0]Y3;
    wire signed [17:0] conv;
    wire signed [15:0] out1;
    wire signed [15:0] out2;
    wire signed [15:0] out3;
    signe_mul MUL1(out1,VA1,Y1);
    signe_mul MUL2(out2,VA2,Y2);
    signe_mul MUL3(out3,VA3,Y3);
    assign conv = out1+out2+out3;
endmodule
```

// resule range [-7,+7] accuracy 12bit Fixed Float

```
module signe_mul (out,a,b);
    output [15:0] out;
    input [15:0] a;
    input [15:0] b;
    wire signed [15:0] out;
    wire signed [31:0] mul_out;
    assign mul_out = a*b;
    assign out = {mul_out[31],mul_out[26:12]};
endmodule
```