We are IntechOpen,
the world's leading publisher of
Open Access books
Built by scientists, for scientists

6,900
Open access books available

186,000
International authors and editors

200M
Downloads

154
Countries delivered to

Our authors are among the
TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS
BOOK CITATION INDEX
INDEXED

WEB OF SCIENCE™
Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Chapter

Solving Partial Differential Equation Using FPGA Technology

Vu Duc Thai and Bui Van Tung

Abstract

This chapter introduces the method of using CNN technology on FPGA chips to solve differential equation with large space, with lager computing space, while limitation of resource chip on FPGA is needed, we have to find solution to separate differential space into several subspaces. Our solution will do: firstly, division of the computing space into smaller areas and combination of sequential and parallel computing; secondly, division and combination of boundary areas that are required to be continuous to avoid losing temporary data while processing (using buffer memory to store); and thirdly, real-time data exchange. The control unit controls the activities of the whole system set by the algorithm. We have configured the CNN chip for solving Navier-Stokes equation for the hydraulic fluid flow successfully on the Virtex 6 chip XCVL240T-1FFG1156 by Xilinx and giving acceptance results as well.

Keywords: Navier-Stokes equation, cellular neural network, field programmable gate array, boundary processing, separating computing space

1. Introduction

Solving the partial differential equation (PDE) has been investigated by many researchers, implementing digital decoding on PCs successfully. However, with the problem of large computing space, the resolution on the PC is difficult to meet the requirements of speed and accuracy calculations; in some cases, the problem cannot be solved because of the calculation. Cellular Neural Network technology (CNN) researchers have applied cellular neural network (CNN) technology successfully to perform analysis of the problem, design CNN chip, and solve some PDEs.

Using CNN technology for solving PDE, we have to analyze and difference the original particular equations of problem, find templates, design CNN architecture, and then configure FPGA to make a CNN chip. It means that there is no CNN chip for every equation, but for each problem (consist of some equations), there is need to design appropriate CNN chip. When solving large problems, computing resources are needed to configure blocks of CNN chips. In order to save resources, we have proposed a solution for dividing computing space into smaller subspaces and composite parallel and sequential calculations, which ensures high computing rates but saves resources of FPGA chips used.

Because the architecture of CNN chips varies depending on each problem, making the CNN chip is very difficult and costly with traditional methods. Using

IntechOpen

the FPGA technology, users can use hardware programming languages, such as Verilog and VHDL, to configure the logic elements in the FPGA to produce the electronic circuit of a CNN chip. The recent FPGA architectures (Virtex 7; Stratix 10) have many tools support to test, optimize, and coordinate data exchange. The CNN designer should use FPGA for making a CNN chip.

2. CNN and FPGA technology

2.1 Cellular neural network technology

Cellular neural network (CNN) was introduced by Chua and Yang at Berkeley University, California (USA), in 1988, which combined both analog spatial temporal dynamic and logic [1–3]. The CNN paradigm is a natural framework to describe the behavior of locally interconnected dynamic systems, which has an arrayed structure, so it is very useful in solving the partial differential equations [3–7]. Today, visual microprocessors based on this processing type can perform at TeraOPs computing power and approximately 50,000 fps. The possibility of developing algorithms and programs based on CNN was quickly exploited worldwide. Up to now, there are several CNN models for processing images, solving PDE, recognizing pattern, gene analysis, etc. Depending on problems, the designer can make a CNN chip having size of millions cells. The common CNN architectures are 1D, 2D, and 3D.

The standard CNN 2D is the dynamic system of autonomous cells that are connected locally with its neighbor forming a two-dimensional array [2, 18]. Each cell in the array C(i,j) contains one independent voltage source, one independent current source, a linear capacitor, resistors, and linear voltage-controlled current sources which are coupled to its neighbor cells via the controlling input voltage, and the feedback from the output voltage of each neighbor cell C(k,l). The templates A(i,j;kl) and B(i,j;k,l) are the parameters linking cell C(i,j) to neighbor C(k,l). The effective range of Sr(I,j) on radius r of cell C(I,j) is identified by the set of neighbor cells which satisfies (**Figure 1**).

$$\mathrm{Sr}(\mathrm{i},\ \mathrm{j}) = \{\mathrm{C}(\mathrm{k},\mathrm{l}) \mid \max\{|\mathrm{k}–\mathrm{i}|, |\mathrm{l}–\mathrm{j}|\} \le \mathrm{r}\}$$
$$\text{with } 1 \le \mathrm{k} \le \mathrm{M}, 1 \le \mathrm{l} \le \mathrm{N}.$$

The state equation of cell C(i,j) is given by the following equation:

$$C\frac{\partial x_{ij}}{\partial t} = -\frac{1}{R}x_{ij} + \sum_{C(k,l)\in S_r(i,\ j)} A(i,\ j;k,l)\,y_{kl} + \sum_{C(k,l)\in S_r(i,\ j)} B(i,\ j;k,l)u_{kl} + z_{ij} \quad (1)$$

With R, C is the linear resistor and capacitor; A(i,j;kl) is the feedback operator parameter; B(i,j;kl) is the control parameter; and zij is the bias value of the cell C(i,j). On the CNN chip, (A, B, z) are the local connective weight values of each cell C(i,j) to its neighbors. The output of the cell C(i,j) is presented by Y_{ij} as:

$$Y_{ij} = f\left(x_{ij}\right) = \frac{1}{2}\ |x_{ij} + 1| + \frac{1}{2}\ |x_{ij}–1| \quad (2)$$

The characteristic of the CNN output function $Y_{i,j} = f(x_{ij})$ is presented in **Figure 2**.

On the CNN 3D, beside connection with neighbors, the cell has other connection to upper and lower layer in the three-dimensional space [18] as shown in **Figure 3**.

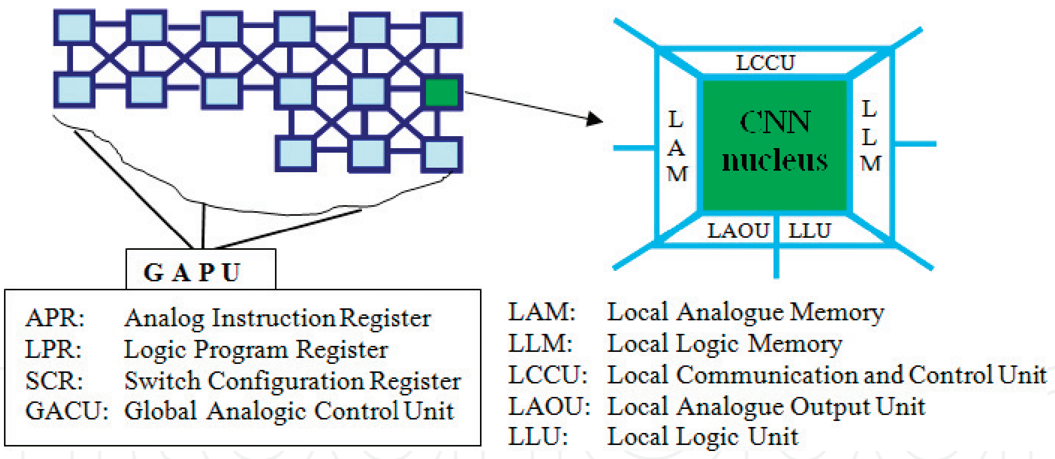

LCCU
L A M
CNN nucleus
L L M
LAOU
LLU
GAPU
APR: Analog Instruction Register
LPR: Logic Program Register
SCR: Switch Configuration Register
GACU: Global Analogic Control Unit
LAM: Local Analogue Memory
LLM: Local Logic Memory
LCCU: Local Communication and Control Unit
LAOU: Local Analogue Output Unit
LLU: Local Logic Unit

Figure 1.
The architecture of a CNN chip

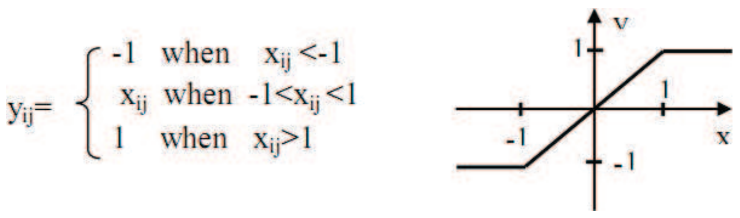


Figure 2.
CNN circuit output function.

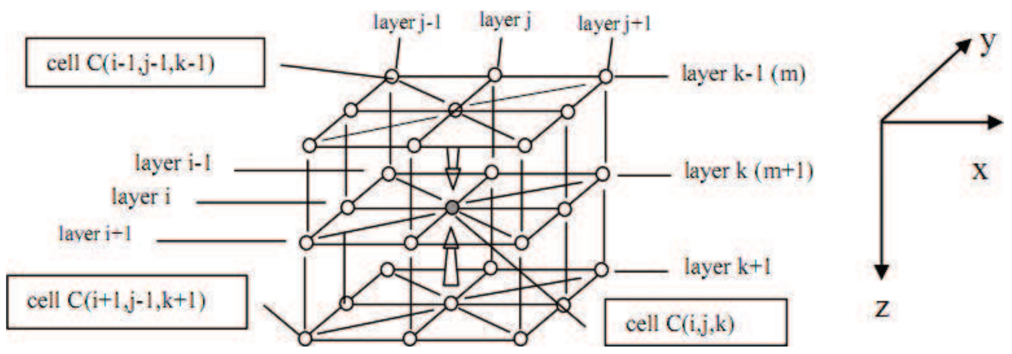


Figure 3.
The 3D CNN, with r = 1, (having 26 neighbors) in three dimensions coordinates x,y,z.

Thus, if radius r = 1, the cell C(i,j,k) has 26 neighbors; hence, the templates A and B have more three coefficients A(i,j,k) and B(i,j,k).

The state equation of CNN 3D takes the form:

$$C\frac{\partial x_{ijk}}{\partial t} = -\frac{1}{R}x_{ijk} + \sum_{C(l,m,n)\in S_r(i,\ j,k)} A(i,j,k;l,m,n)\,y_{lmn} + \sum_{C(l,m,n)\in S_r(i,\ j,k)} B(i,j,k;l,m,n)\,y_{lmn} + z_{ijk} \tag{3}$$

The output function is similar to CNN 2D:

$$y_{ijk} = f(\mathrm{x}_{ijk}) = \frac{1}{2}\left(|x_{ijk}+1| + |x_{ijk}-1|\right)$$

For the problem-solving of three-dimensional PDE, the CNN 3D must be used. The original PDE is differentiated and from that the appropriate templates (A,B,z) of the CNN 3D are generated.

IntechOpen

2.2 Field-programmable gate array technology

Field-programmable gate array (FPGA) is the technology in which the blank blocks have available resources of logic gates and RAM blocks are used to implement complex digital computations. FPGAs can be used to implement any logical function. The FPGA block is able to update the functionality after shipping, partial reconfiguration of a portion of the design, and the low nonrecurring engineering costs relative to an ASIC design [13–16].

A recent trend has been to take the coarse-grained architectural approach by combining the logic blocks and interconnects of traditional FPGAs with embedded chips and related peripherals to form a complete "system on a programmable chip" [17–19].

Users like teachers and students could use FGGA for making prototypes for testing application system, with VHDL or Verilog users easily design and test and then reconfigure the system until it has desired results.

2.3 Using FPGA to make CNN chip for solving PDE

Because the CNN architecture is not the same for every application, based on the standard model, the designer develops a particular chip for each problem. FPGA is the most useful for configuring a blank chip to make a CNN chip using programming language like Verilog or VHDL. For solving PDE, firstly, one needs to analyze (differencing) the original model of partial differential equations for finding appropriate template, then base on template found designing architecture CNN chip, finally, using VHDL to configure FPGA following designed hardware making CNN chip.

Some PDEs have been solved using the CNN technology:

Burger equation [3]:

$$\frac{\partial u(x,t)}{\partial t} = \frac{1}{R}\frac{\partial^2 u(x,t)}{\partial x^2} - u(x,t)\frac{\partial u(x,t)}{\partial x} + F(x,t)$$

Klein-Gordon equation [19]:

$$\frac{\partial^2 u(x,t)}{\partial t^2} = \nabla^2 u(x,t) - \sin u(x,t)$$

Heat diffusion equation [3]:

$$\frac{\partial u(x,\ y,t)}{\partial t} = c\nabla^2 u(x,\ y,t)$$

Black-Scholes equation [9]:

$$\frac{\partial V(x,t)}{\partial t} = rV(x,t) - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V(x,t)}{\partial S^2} - \mathrm{rS}\frac{\partial V(x,t)}{\partial S}$$

Air pollution equation [4]:

$$\frac{\partial \varphi}{\partial t} + div\boldsymbol{v}\varphi + \sigma\varphi - \gamma\frac{\partial^2 \varphi}{\partial z^2} - \mu\nabla^2\varphi = f(x,\ y,z)$$

Saint venant 2D equation [5]:

$$\frac{\partial H}{\partial t} + \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

$$\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + g\frac{\partial H}{\partial x} + \frac{\partial uv}{\partial y} = -gu\frac{\left(u^2+v^2\right)^{1/2}}{K_x^2 H^2}$$

$$\frac{\partial v}{\partial t} + \frac{\partial v^2}{\partial y} + g\frac{\partial H}{\partial y} + \frac{\partial uv}{\partial x} = -gv\frac{\left(u^2+v^2\right)^{1/2}}{K_y^2 H^2}$$

Saint venant 1D equation [6]:

$$b\frac{\partial h(x,t)}{\partial t} + \frac{\partial Q(x,t)}{\partial x} = q \tag{4}$$

$$\frac{\partial Q(x,t)}{\partial t} + \frac{\partial\left[\frac{Q(x,t)^2}{bh(x,t)}\right]}{\partial x} + gbh(x,t)\frac{\partial h(x,t)}{\partial x} - gIbh(x,t) + gJbh(x,t) = k_q q \tag{5}$$

Example of making a CNN chip for solving Saint venant 1D:

- Designing the templates

First, changing the original equation (4)

$$b\frac{\partial h(x,t)}{\partial t} + \frac{\partial Q(x,t)}{\partial x} = q$$

$$\Leftrightarrow \frac{\partial h(x,t)}{\partial t} = \frac{-\partial Q(x,t)}{b\partial x} + \frac{q}{b} \tag{6}$$

and then choosing the difference space of variables x with step Δx for right part of (6). After differencing only the right side of (6) for space variable x by Taylor expansion, one has equation for cell at position (i):

$$\frac{\partial h}{\partial t} = -\frac{1}{2b\Delta x}\left(Q_{i+1} - Q_{i-1}\right) + \frac{q}{b} \tag{7}$$

Note that, following the CNN algorithm, on the left, we do use symbol $(\partial h/\partial t)$. From (7), one has found templates:

$$A^{hQ} = \left[\frac{1}{2b\Delta x}\ \ \frac{1}{R^h}\ \ \frac{-1}{2b\Delta x}\right]; \mathrm{B^h} = [0\ 1\ 0]; \mathrm{z^h} = 0;$$

where $\mathrm{R^h}$ is the linear resistance on cell circuit of h.

For Eq. (5), changing slightly with assumptions above:

$$\frac{\partial Q(x,t)}{\partial t} + \frac{\partial\left[\frac{Q(x,t)^2}{bh(x,t)}\right]}{\partial x} + gbh(x,t)\frac{\partial h(x,t)}{\partial x} - gIbh(x,t) + gJbh(x,t) = k_q q \qquad (8)$$

Assume that q > 0, then $k_q = 0$. After differencing, applying the template design algorithm of CNN, one can has templates for (8):

$$A^Q = \frac{Q_{i+1}}{2b\Delta x h_{i+1}} \quad \frac{1}{R^Q} \quad -\frac{Q_{i-1}}{2b\Delta x h_{i-1}}];$$

$$A^{Qh} = \left[\frac{gbh_i}{2\Delta x} \quad gb(I-J) \quad -\frac{gbh_i}{2\Delta x}\right]; B^Q = 0; z^Q = 0;$$

From template found, we can design the CNN architecture for problem as (1) two layered-1D CNN chip (**Figure 4**) and (**2**) the h, Q processing block (**Figure** 5).

The cell is mixed both of h, Q in one block to make the physical architecture of a CNN cell.

In general, for each calculation, we need some basic computing block like ADDITION, SUBTRACT, MULTIPLE, DIVIDE. When designing a CNN cell using FPGA, one has to design many separate blocks of them to perform arithmetical

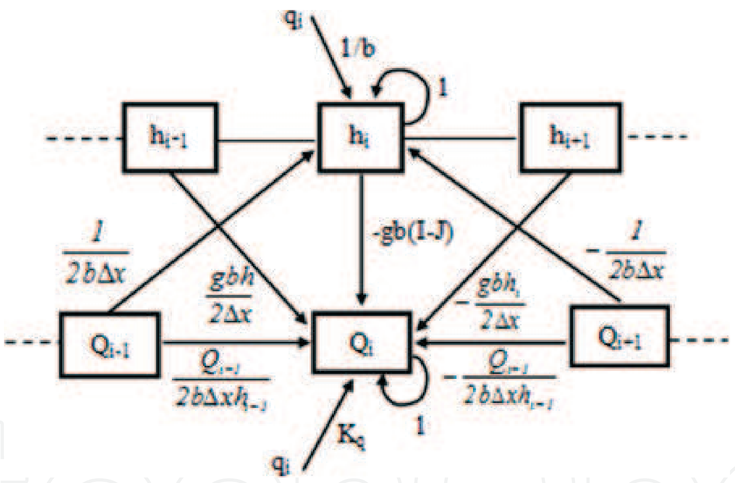


Figure 4.
Logical architecture of a CNN cell.

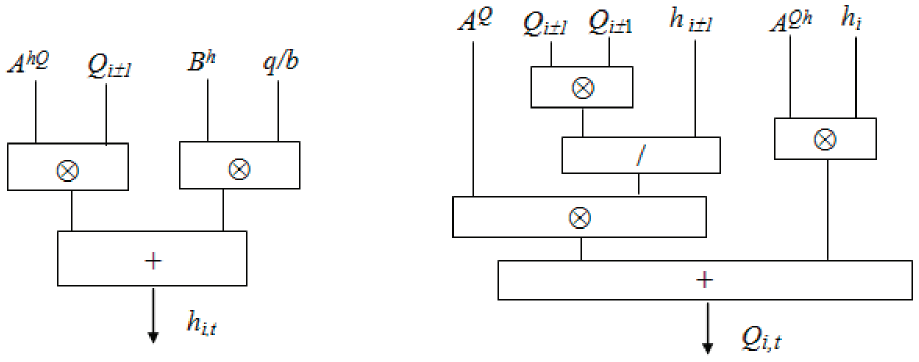


Figure 5.
Logical architecture of a h, Q cell.

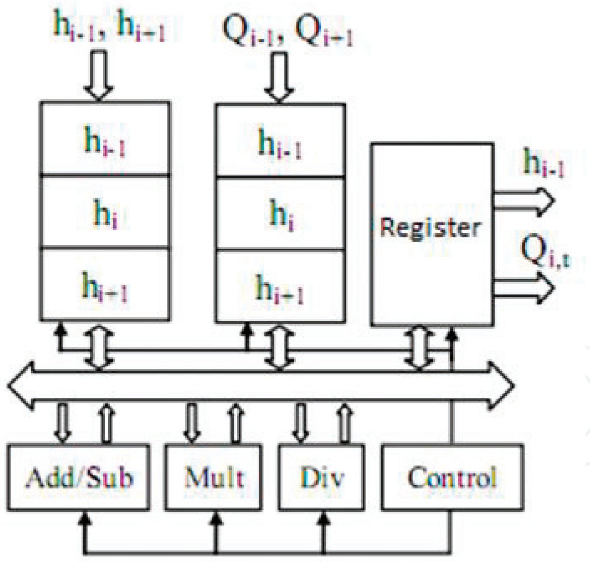


Figure 6.
Physical architecture of CNN cell.

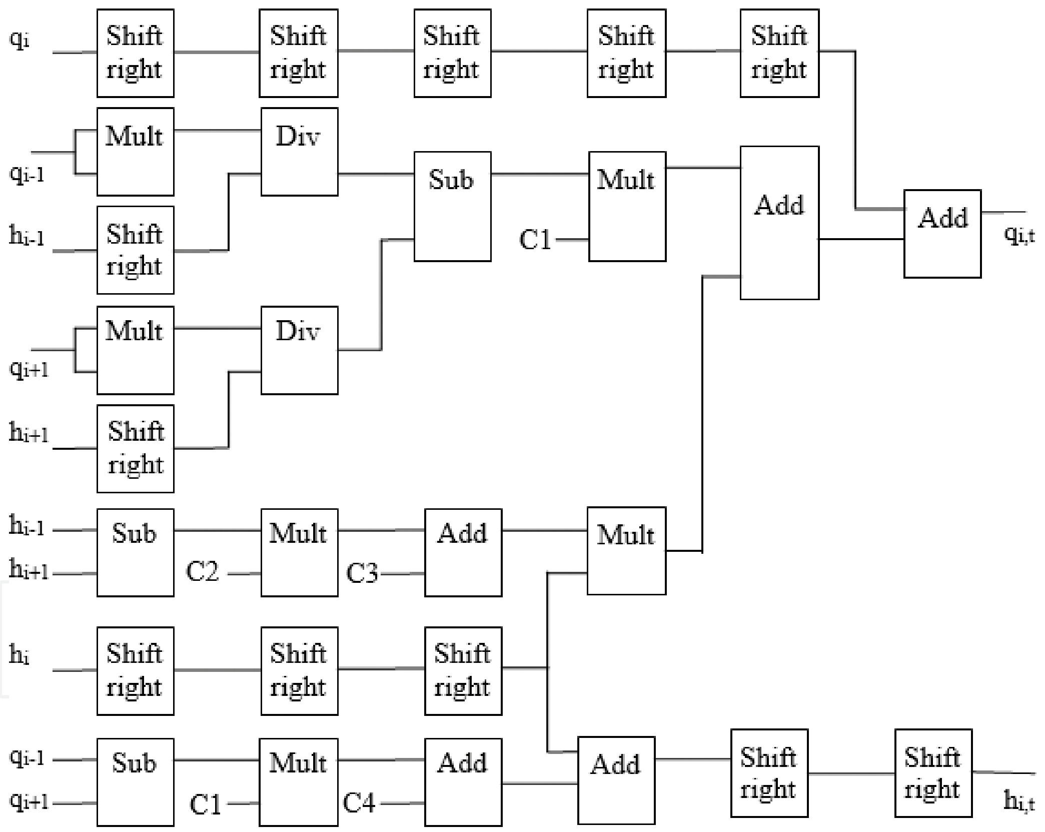


Figure 7.
Solution for physical architecture CNN chip.

processing for each input. In order to save computing resource in FPGA, the method that shares basic block in one cell leading to sequential calculating can be used (**Figure 6**). In this case, the processing time of each cell will be high. To reduce the processing time of each cell, we can use a pipeline mechanism shown in **Figure 7**, but it needs more computing resource for each cell. Finally, for cells in a CNN chip, we process parallel as in **Figure 8**.

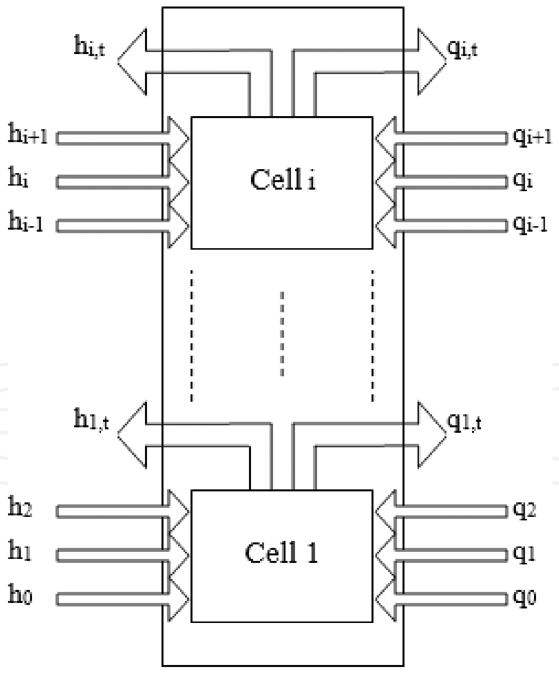


Figure 8.
A core architecture for CNN chip.

C1, … , C4 are the coefficients as shown in **Figure 7**, (C1= $\frac{1}{2b\Delta x}\Delta t$; C2= $\frac{gb}{2\Delta x}\Delta t$; C3= $gb(I-J)\Delta t$; C4= $\frac{q}{b}\Delta t$).

If each cell is uses a pipeline mechanism shown in **Figure 7**. With the length of a pipeline is 6, the first calculation pays 6 clock pulse (clk), and each calculation after that only needs 1 clk.

3. Solving Navier-Stokes equations

3.1 Physico-mathematical model of Navier-Stokes equations

In hydraulics, many flow models have been researched, such as flows in channels, streams, or rivers, for controlling the flow for preventing disasters, saving water, and exploiting energy of the flow as well. Most of mathematical models of those phenomena are partial differential equations like Saint venant equations and Navier-Stokes equations [8, 9]. Some types of Navier-Stokes equations have various parameters and constraints. Using CNN technology, we could solve some of them which have clear values of boundary conditions; it means we do not research boundary problems deeply. The effectiveness of the CNN technology is making a physical parallel computing chip to increase the computing speed for satisfying a real-time system.

Navier-Stokes equations here consist of three partial differential equations, with functional variables representing water height, and flow velocity in x- and y-directions. The empirical model is a flow through a small port, which diffuses in two directions Ox and Oy.

Solving Navier-Stokes equations by using CNN requires the discretion of continuity model by difference method, the smaller difference intervals the higher accuracy. However, if difference intervals are too small, then it leads to increasing the calculation complexity and time. The CNN chip with parallel physically processing abilities, the above difficulties will be overcome.

3.2 Description equations in Navier-Stokes equations

- Equations describing the water level

$$\frac{\partial \rho z_{\mathrm{w}}}{\partial \mathrm{t}} + \frac{\partial \rho q_{\mathrm{x}}}{\partial x} + \frac{\partial \rho q_{\mathrm{y}}}{\partial y} = \rho q_A \tag{9}$$

Assume that the height of water is taken from the bottom of the flow, which is regarded as the origin of the coordinate system, so z_{w} has no negative values.

- Momentum equations in x-direction:

$$\frac{\partial \rho \mathrm{q_x}}{\partial \mathrm{t}} + \frac{\partial}{\partial x}\left(\rho\beta\frac{q_x^2}{d}\right) + \frac{\partial}{\partial y}\left(\rho\beta\frac{q_x q_y}{d}\right) + \rho g d\frac{\partial z_{\mathrm{w}}}{\partial \mathrm{x}} + \rho g d S_{f\mathrm{x}} - \tau_{\mathrm{wx}} - \frac{\partial}{\partial x}\left(\rho K_L\frac{\partial q_x}{\partial x}\right) - \frac{\partial}{\partial y}\left(\rho K_T\frac{\partial q_x}{\partial y}\right) = 0 \tag{10}$$

- Momentum equations in y-direction:

$$\frac{\partial \rho \mathrm{q_y}}{\partial \mathrm{t}} + \frac{\partial}{\partial y}\left(\rho\beta\frac{q_y^2}{d}\right) + \frac{\partial}{\partial x}\left(\rho\beta\frac{q_y q_x}{d}\right) + \rho g d\frac{\partial z_{\mathrm{w}}}{\partial y} + \rho g d S_{fy} - \tau_{\mathrm{wy}} - \frac{\partial}{\partial y}\left(\rho K_L\frac{\partial q_y}{\partial y}\right) - \frac{\partial}{\partial x}\left(\rho K_T\frac{\partial q_y}{\partial x}\right) = 0 \tag{11}$$

Explain the meanings of quantities in the equations:

- $\frac{\partial \rho q_x}{\partial t}$ and $\frac{\partial \rho q_y}{\partial t}$: quantities characterizing the momentum variation over time in x-axis and y-axis, respectively.
- $\frac{\partial}{\partial x}\left(\rho\beta\frac{q_x^2}{d}\right)$ and $\frac{\partial}{\partial y}\left(\rho\beta\frac{q_y^2}{d}\right)$: kinetic energy variations of flow in x- and y-directions.
- $\rho g d\frac{\partial z_{\mathrm{w}}}{\partial x}$ and $\rho g d\frac{\partial z_{\mathrm{w}}}{\partial y}$: potential energy variations of flow in x- and y-directions.
- $\rho g d S_{f\mathrm{x}}$ and $\rho g d S_{fy}$: influence of friction by bottom and walls of channel on flow in x- and y-directions. Values of S_{fx} and S_{fy} are determined based on physical properties of bottom and walls of hydraulic channels according to the following formulas:

$$S_{f\mathrm{x}} = q_x\frac{n^2\left(q_x^2+q_y^2\right)^{1/2}}{d^{1/3}};S_{fy} = q_y\frac{n^2\left(q_y^2+q_x^2\right)^{1/2}}{d^{1/3}} \text{ (n is Manning coefficient)}$$

- τ_{wx} and τ_{wy}: wind pressure on free surface of hydraulic flow in x-and y-directions are calculated as follows:

$$\tau_{\mathrm{wx}} = c_s\rho_a\mathrm{W}^2\cos(\Psi);\tau_{\mathrm{wy}} = c_s\rho_a\mathrm{W}^2\sin(\Psi),$$

where:

$$c_x = \left\{\begin{array}{l} 10^{-3};khi\ \mathrm{W} \le \mathrm{W_{min}} \\ [\mathrm{c_{s1}} + c_{s2}(\mathrm{W\text{-}W_{min}})].10^{-3};khi\ \mathrm{W}>\mathrm{W_{min}} \end{array}\right\};$$

With c_{s1}; c_{s2}; W_{min} are values get from practical, for example: $W_{min} = 4$ m/s; wind speed is 10 m/s, then $c_{s1} = 1.0$; $c_{s2} = 0.067$;

- ρ_a is the air density at free surface (kgm^{-3}); W is wind speed at free surface; and Ψ is the angle between wind direction and x-axis.

- Expressions, $\frac{\partial}{\partial x}\left(\rho K_L \frac{\partial q_x}{\partial x}\right) - \frac{\partial}{\partial y}\left(\rho K_T \frac{\partial q_x}{\partial y}\right)$ and $\frac{\partial}{\partial y}\left(\rho K_L \frac{\partial q_y}{\partial y}\right) - \frac{\partial}{\partial x}\left(\rho K_T \frac{\partial q_y}{\partial x}\right)$, are the impact of turbulence in hydraulic flow caused between x- and y-directions, where: $K_L = \frac{q_x l}{p_e}$ with Pe as the Peclet coefficient with the value of 15–40; l as the length of flow; K_L as coefficient varying according to locations along flow; and K_T = 0.3–0.7 K_L.

3.3 Analyzing and designing CNN to solve the equations

To simplify, change parameters as: the water level $z_w = h$; and the velocity in x-axis $q_x = u$, in y-axis $q_y = v$. Assume that $q_A = 0$; the kinetic influence of turbulent values between velocity in the direction from 0y to 0x (or 0x to 0y) is trivial since horizontal velocity is small enough to be considered as zero; then (9)–(11) are rewritten:

$$\frac{\partial h}{\partial t} + \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \Leftrightarrow \frac{\partial h}{\partial t} = -\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} \tag{12}$$

$$\begin{aligned} &\frac{\partial v}{\partial t} + \frac{\partial}{\partial y}\left(\beta \frac{v^2}{d}\right) + \frac{\partial}{\partial x}\left(\beta \frac{vu}{d}\right) + gd\frac{\partial h}{\partial y} + gdS_{fy} - \frac{\tau_{wy}}{\rho} - \frac{\partial}{\partial y}\left(K_L \frac{\partial v}{\partial y}\right) = 0 \\ &\Leftrightarrow \frac{\partial v}{\partial t} = \frac{\partial}{\partial y}\left(K_L \frac{\partial v}{\partial y}\right) - \frac{\partial}{\partial y}\left(\beta \frac{v^2}{d}\right) - \frac{\partial}{\partial x}\left(\beta \frac{vu}{d}\right) - gd\frac{\partial h}{\partial y} + \left(\frac{\tau_{wy}}{\rho} - gdS_{fy}\right) \end{aligned} \tag{13}$$

$$\begin{aligned} &\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\beta \frac{u^2}{d}\right) + \frac{\partial}{\partial y}\left(\beta \frac{uv}{d}\right) + gd\frac{\partial h}{\partial \text{x}} + gdS_{f\text{x}} - \frac{\tau_{wx}}{\rho} - \frac{\partial}{\partial x}\left(K_L \frac{\partial u}{\partial x}\right) \\ &\Leftrightarrow \frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(K_L \frac{\partial u}{\partial x}\right) - \frac{\partial}{\partial x}\left(\beta \frac{u^2}{d}\right) - \frac{\partial}{\partial y}\left(\beta \frac{uv}{d}\right) - gd\frac{\partial h}{\partial \text{x}} + \left(\frac{\tau_{wx}}{\rho} - gdS_{f\text{x}}\right) \end{aligned} \tag{14}$$

Step 1: Differencing equations following Taylor formula

Using finite difference grid with difference interval in x-axis as Δx and in y-axis as Δy and apply Taylor difference formulas for Eqs. (12)–(14); we have difference equations corresponding to the equations:

$$\frac{\partial h_{ij}}{\partial t} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} - \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y} \tag{15}$$

$$\begin{aligned} \frac{\partial u_{i,j}}{\partial t} = -\frac{\beta}{d}\left[\frac{u_{i+1,j}}{2\Delta x}u_{i+1,j} - \frac{u_{i-1,j}}{2\Delta x}u_{i-1,j}\right] - \frac{\beta}{d}\left[\frac{v_{i,j+1}}{2\Delta y}u_{i+1,j} - \frac{v_{i,j-1}}{2\Delta y}u_{i-1,j}\right] \\ -gd\frac{h_{i+1,j} - h_{i-1,j}}{2\Delta x}g\text{dS}_{\text{fx}} + \frac{1}{\rho}\tau_{\text{wx}}K_L\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta \text{x}^2}] \end{aligned} \tag{16}$$

$$\frac{\partial v_{i,j}}{\partial t} = -\frac{\beta}{d}\left[\frac{v_{\mathrm{i,\,j+1}}}{2\Delta y}v_{i,j+1} - \frac{v_{\mathrm{i,\,j\text{-}1}}}{2\Delta y}v_{i,j-1}\right] - \frac{\beta}{d}\left[\frac{u_{\mathrm{i+1,\,j}}}{2\Delta x}v_{i,j+1} - \frac{u_{\mathrm{i\text{-}1,\,j}}}{2\Delta x}v_{i,j-1}\right] \\ -gd\frac{h_{i,j+1}-h_{i,j-1}}{2\Delta x} - g\mathrm{d}S_{\mathrm{fy}} + \frac{1}{\rho}\tau_{\mathrm{wy}}K_L\frac{v_{i,j+1}-2v_{i,j}+v_{i,j-1}}{\Delta y^2}] \tag{17}$$

Step 2: Designing a sample of CNN

Based on CNN state equations and difference equations (15)–(17), we can have CNN templates for layers h, u, v:

- Layer h:

$$A^{hu} = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{2\Delta x} & 0 & \frac{-1}{2\Delta x} \\ 0 & 0 & 0 \end{bmatrix} \quad A^{hv} = \begin{bmatrix} 0 & \frac{1}{2\Delta y} & 0 \\ 0 & 0 & 0 \\ 0 & \frac{-1}{2\Delta y} & 0 \end{bmatrix} \tag{18}$$

- Layer u:

$$A^{uv} = \begin{bmatrix} 0 & \frac{\beta u_{i,j-1}}{2d\Delta y} & 0 \\ 0 & 0 & 0 \\ 0 & \frac{-\beta u_{i,j+1}}{2d\Delta y} & 0 \end{bmatrix}; A^{uh} = \begin{bmatrix} 0 & 0 & 0 \\ \frac{gd}{2\Delta x} & 0 & \frac{-gd}{2\Delta x} \\ 0 & 0 & 0 \end{bmatrix}; B^u = \frac{1}{\rho}\tau_{wx}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$A^u = \begin{bmatrix} 0 & 0 & 0 \\ \frac{\beta u_{i-1,j}}{2d\Delta x} + \frac{K_L}{\Delta x^2} & gd\frac{n^2\left(u_{ij}^2+v_{ij}^2\right)^{1/2}}{d^{1/3}} + \frac{1}{R_u} + \frac{4K_L}{\Delta x^2} & \frac{-\beta u_{i+1,j}}{2d\Delta x} + \frac{-K_L}{\Delta x^2} \\ 0 & 0 & 0 \end{bmatrix}; \quad z^{\mathrm{u}} = 0 \tag{19}$$

- Layer v:

$$A^{vh} = \begin{bmatrix} 0 & \frac{g\mathrm{d}}{2\Delta y} & 0 \\ 0 & 0 & 0 \\ 0 & \frac{-g\mathrm{d}}{2\Delta y} & 0 \end{bmatrix}; A^{vu} = \begin{bmatrix} 0 & 0 & 0 \\ \frac{\beta u_{i-1,j}}{2d\Delta x} & 0 & \frac{-\beta u_{i-1,j}}{2d\Delta x} \\ 0 & 0 & 0 \end{bmatrix}; B^v = \frac{1}{\rho}\tau_{\mathrm{wy}}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}; z^v = 0$$

$$A^v = \begin{bmatrix} 0 & \frac{\beta v_{i,j+1}}{2d\Delta y} + \frac{K_L}{\Delta y^2} & 0 \\ \frac{K_L}{\Delta y^2} & g\mathrm{d}\frac{n^2\left(u_{i,j}^2+v_{i,j}^2\right)^2}{d^{1/3}} + \frac{1}{R^v} + \frac{K_L}{\Delta y^2} & \frac{-K_L}{\Delta y^2} \\ 0 & \frac{-\beta v_{i,j+1}}{2d\Delta y} - \frac{K_L}{\Delta y^2} & 0 \end{bmatrix} \tag{20}$$

Step 3: Designing hardware architecture of CNN to solve Navier-Stokes equations

Based on templates found in (18)–(20), we can design an architecture for circuit for CNN chip. It is a three-layered CNN 2D. Then, the arithmetic unit for each layer and links to perform parallel calculation on chip can be made. **Figure 9** shows the architecture of layer h and layer u (the layer v is similar to u).

3.4 Proposed system architecture for MxN CNN

The empirical problems that need a solution is that: firstly, identifying boundary points of whole difference grid (space); secondly, dividing the entire computing space into smaller subspaces. Division and combination of boundary areas need to perform appropriately avoiding incorrect results because of tep time computing time; thirdly, controlling real-time data exchange and combining sequential and parallel computing in a CNN chip. The CNN chip proposed in this chapter has solved similarity in the previous problems [4, 5]. The new issues here are dividing computing space processing dynamic sub-boundary and combining sequential and parallel.

3.4.1 General MxN CNN

Each CNN cell has its own data element and a core that performs the computing function. The CNN has MxN CNN cells in which only (M-2)x(N-2) CNN cells have computing functions, so that the CNN has MxN data elements and (M-2)x(N-2) cores (**Figure 10**).

The Buffer supplies MxN data elements for CNN. Each MxN data element is called as one block of data (**Figure 11**).

The white area is the data element for CNN boundary cells; and the gray part is the data area which requires to be processed by CNN. The CNN arithmetic unit has size of (M-2)x(N-2) cells processing data for the gray area which is inside the input buffer unit.

The Input memory has PxQ blocks of data. It is a true dual port memory.

The Temp memory also has PxQ blocks of data. It is a simple dual port memory. It is used to temporarily store data computed from CNN core and supply data for Boundary updating unit.

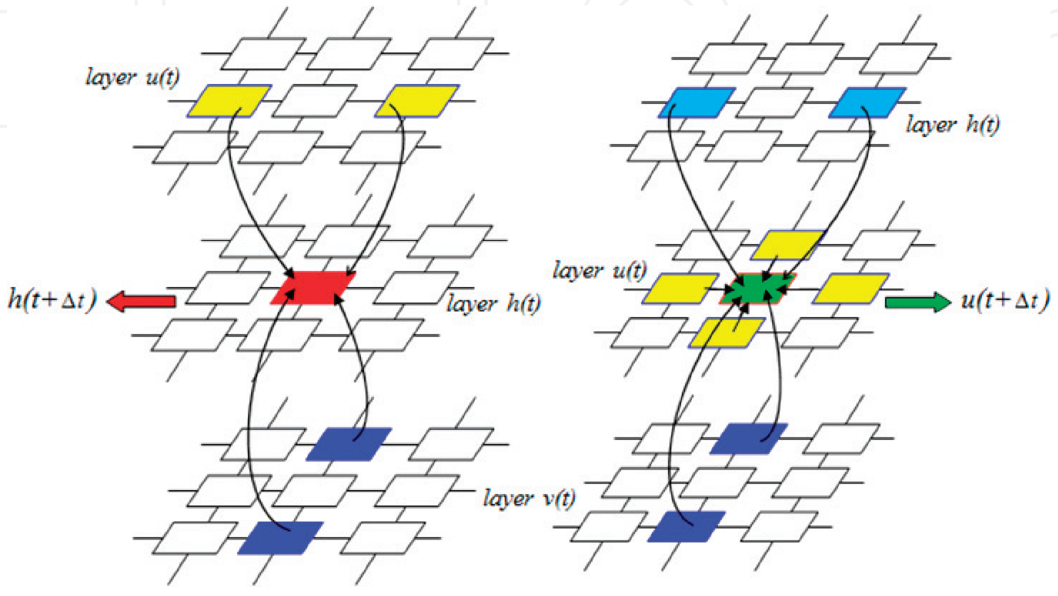


Figure 9.
Logic architecture of cell of h, u.

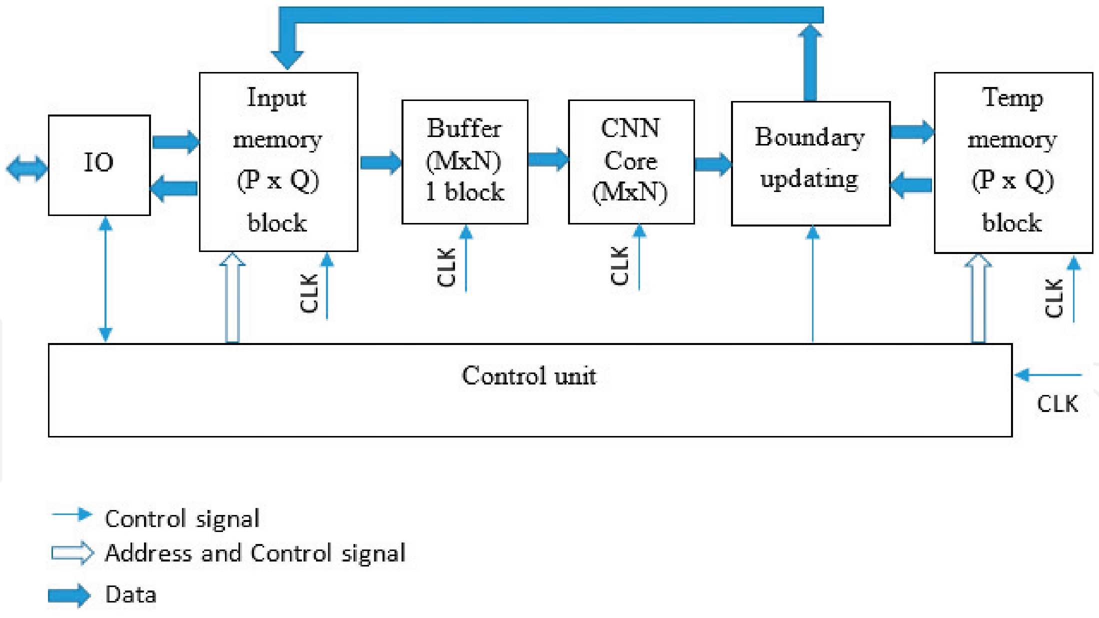
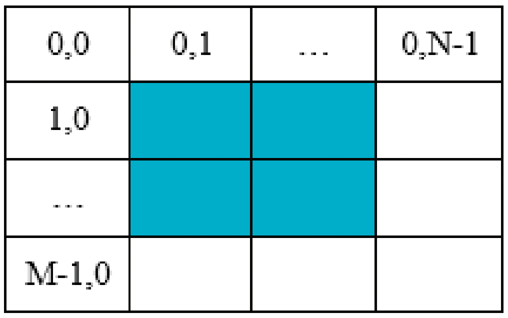


Figure 10.
General architecture of a CNN chip.



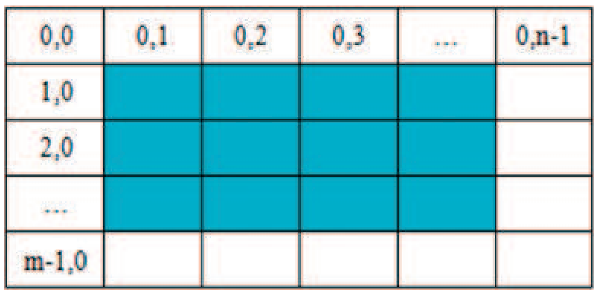
0,0	0,1	...	0,N-1
1,0	▨	▨	
...	▨	▨	
M-1,0			

Figure 11.
Buffer (MxN) for CNN core.

Data that need processing sent from PC have the size of mxn (**Figure 12**).

Assume that m = 5, n = 6, M = 3, and N = 4; the white part is boundary and the gray part is the area requiring to be processed. Before the processing data, temporary vertical and horizontal boundaries be need to be added, as in **Figure 13**, column (0,3) and row (3,0).

Temporary vertical and horizontal boundaries are added to the data structure similar to CNN buffer. The data after being added from temporary vertical and horizontal boundaries will be sent to Input memory. The blocks of data in the Input memory unit (in case that mxn = 5x6, MxN = 3x4) are detailed as follows (**Figure 14**).



0,0	0,1	0,2	0,3	...	0,n-1
1,0	▨	▨	▨	▨	
2,0	▨	▨	▨	▨	
...	▨	▨	▨	▨	
m-1,0					

Figure 12.
Computing space with main boundary.

0,0	0,1	0,2	0,3	0,2	0,3	0,4	0,5
1,0	🟦	🟦			🟦	🟦	
2,0							
1,0							
2,0	🟦	🟦			🟦	🟦	
3,0							
2,0							
3,0	🟦	🟦			🟦	🟦	
4,0							

Figure 13.
Divide computing space into subspace with subboundary.

0, 1, 2,.., 6 are the addresses of blocks. In case that mxn = 5x6 and MxN = 3x4, we have P = 3 and Q = 2.

$$PxQ = \frac{m-2}{M-2}x\frac{n-2}{N-2}$$

The Boundary updating unit is in detail structure as follows (in case MxN = 3x4) (**Figure 15**).

The control unit controls the activities of the whole system set by the algorithm which is as follows:

```
(1) At every posedge of clk do
(2) {
(3)         if (has IO event)
(4)                 do the IO task;
(5)         else
(6)                 buffer = read(Input memory)
(7)                 if (finish computing the first block)
(8)                         if (BoundaryUpdating())
(9)                                 write(Input memory)
(10) }
```

3.4.2 Proposed CNN architecture when M = 3 (3xN CNN)

The 3xN CNN architecture is similar to the general MxN CNN architecture (M = 3). In order to reduce the memory consumption and simplify the Boundary updating unit, there are some differences (**Figure 16**).

Each block of data in the memory (Input memory or Temp memory) is 1xN data elements. Assume that the data which need processing sent from PC has the size of mxn, m = 5, n = 6, and assume that N = 4. As mention above, the data will be processed after temporary vertical boundaries are added; so that, the Input Memory unit will has 5x2 blocks of data (m = 5, Q = 2) as follow (**Figure 17**).

Each block has size of 1x4 data elements.

The Buffer unit is a Shift up register that has size of 3xN. The input and output have sizes of 1xN and 3xN, respectively. The input is at the bottom.

The Input memory has m rows and Q columns of blocks of data. The control unit reads the blocks in the Input memory by vertical and puts the block of data to the

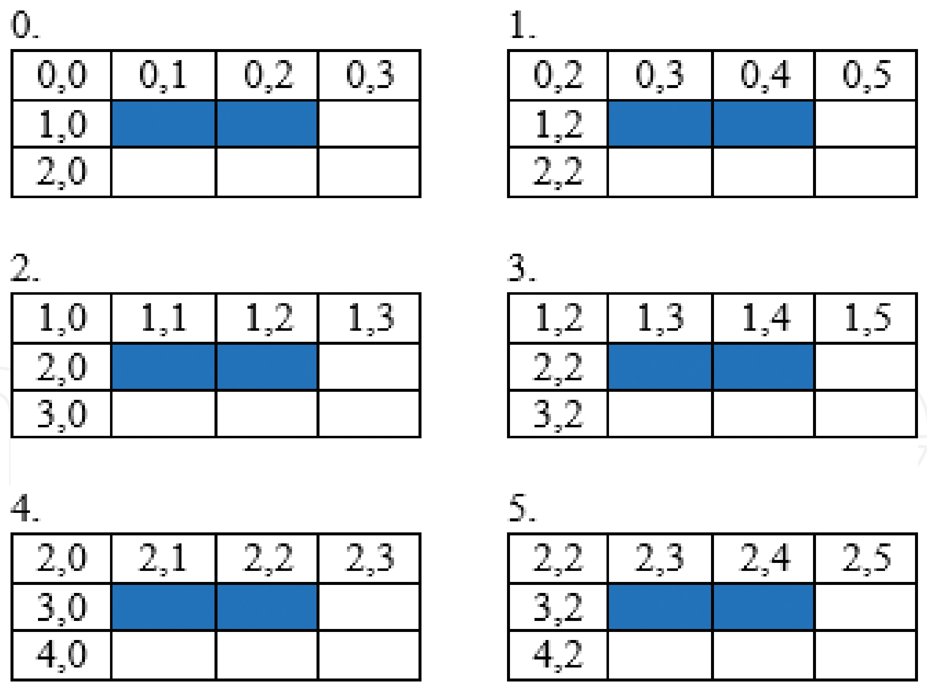


Figure 14.
The blocks of data in the Input memory in case that mxn = 5x6, MxN = 3x4.

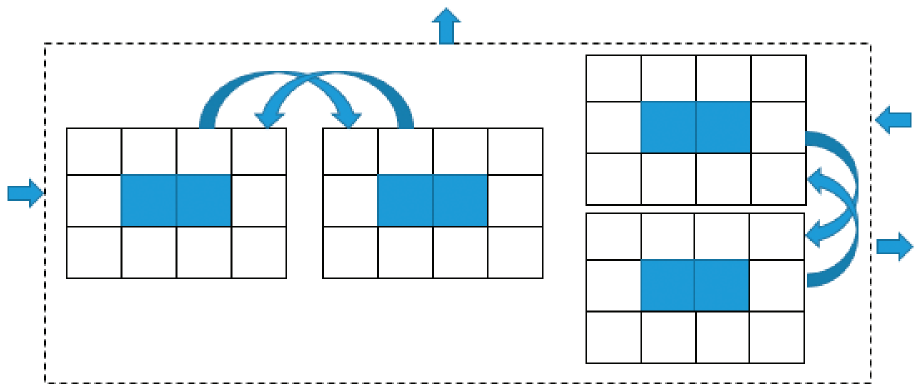


Figure 15.
The Boundary updating structure (MxN = 3x4).

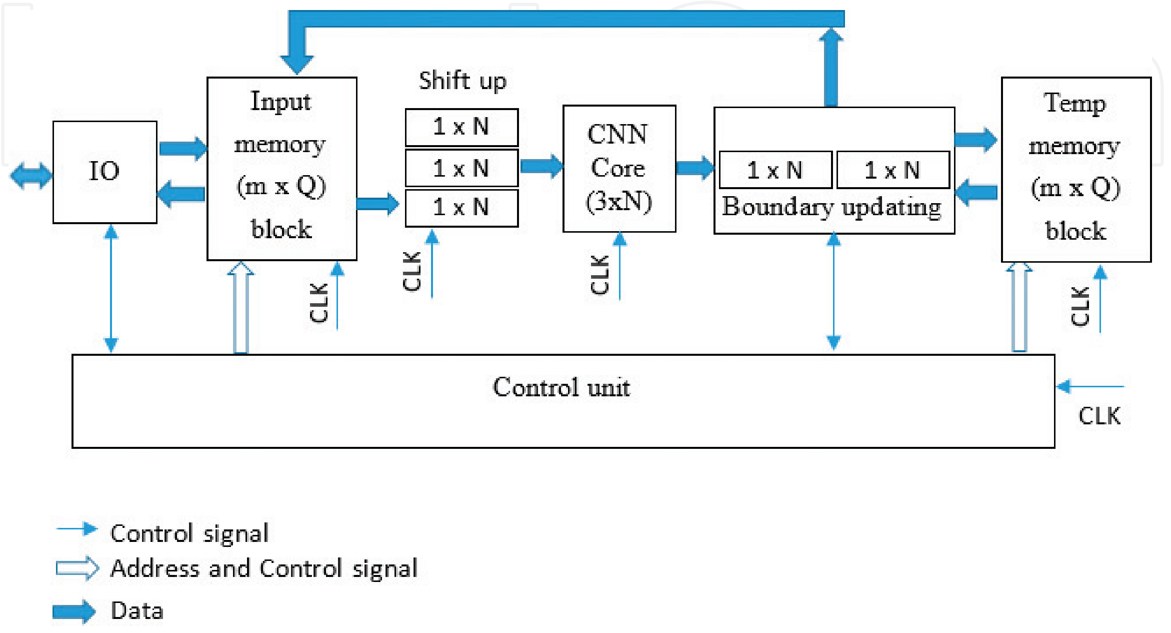


Figure 16.
The architecture of 3xN CNN chip.

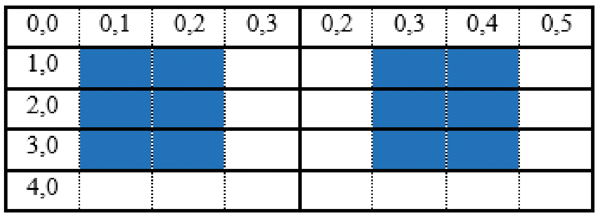


Figure 17.
The memory with 5x2 blocks (m==5, n = 6, N = 4).

input of buffer. The buffer shifts up 1 step. After step 3, the Buffer has 3xN blocks of data to supply to CNN core. After each step, the Buffer has 3xN blocks of data that need to supply to CNN core (**Figure 18**).

The output of CNN core has the size of 1xN.

The Boundary updating unit is shown in **Figure 19**.

The control algorithm for control unit (**Figure 20**).

```
(1) At every posedge of clk do
(2) {
(3)         if (has IO event)
(4)                 do the IO task;
(5)         else
(6)                 buffer = read(Input memory);//read by vertical
(7)                 if (finish computing the first block of column q)
(8)                         if (column_of_current_block==0)
                                    write(Temp memory);
                        else
                                    BoundaryUpdating(CNNoutput,read(Temp
                                    memory));
(9)                                 write(Input memory);
(10) }
```

3.5 Implementation

In this part, we implement the 3xN CNN. Q, m, and N are the parameters that we can configure before compiling and programming to the FPGA chip. For defaulting, we assigned Q = 2, m = 8, and N = 4.

3.5.1 Development environment

For experiencing, the ISE Design Suite software version 14.7 and ML605 evaluation board including chip XCVL240T-1FFG1156 (Virtex 6) are used to implement the schematic of CNN.

First, we use Verilog HDL language to describe the CNN architecture. Then, we use ISim simulator to verify our system. Finally, we program the system to the FPGA chip on ML605 board.

The image of experience system as in **Figure 20** is as follows.

3.5.2 Input data for h, u, v values

The input of CNN to solve the Navier-Stokes Equation has h, u, v values. We use three Input memory units, three Buffer units, and three Temporary memory units to store h, u, v values. The data element is represented in 32-bit floating point

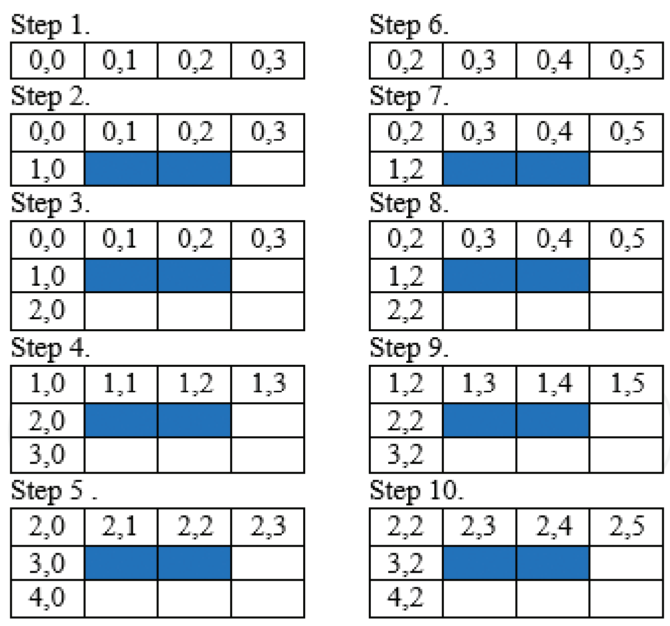


Figure 18.
The Buffer's state after each step (m==5, n = 6, N = 4).

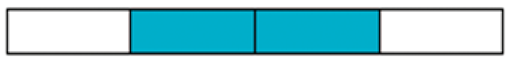


Figure 19.
The output size of CNN core (N = 4).

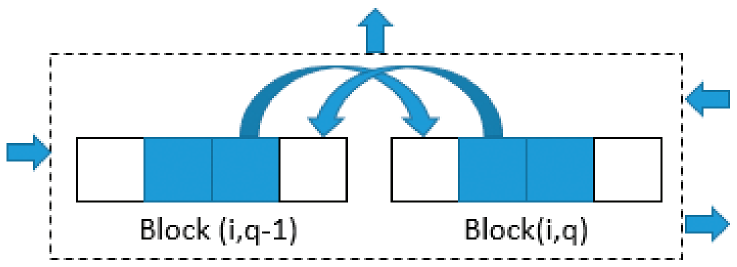


Figure 20.
The Boundary updating structure (N = 4).

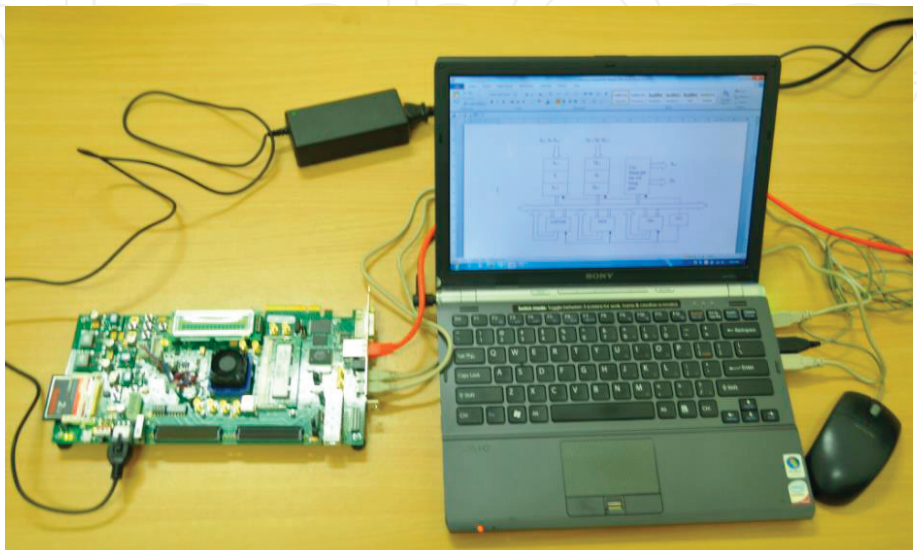
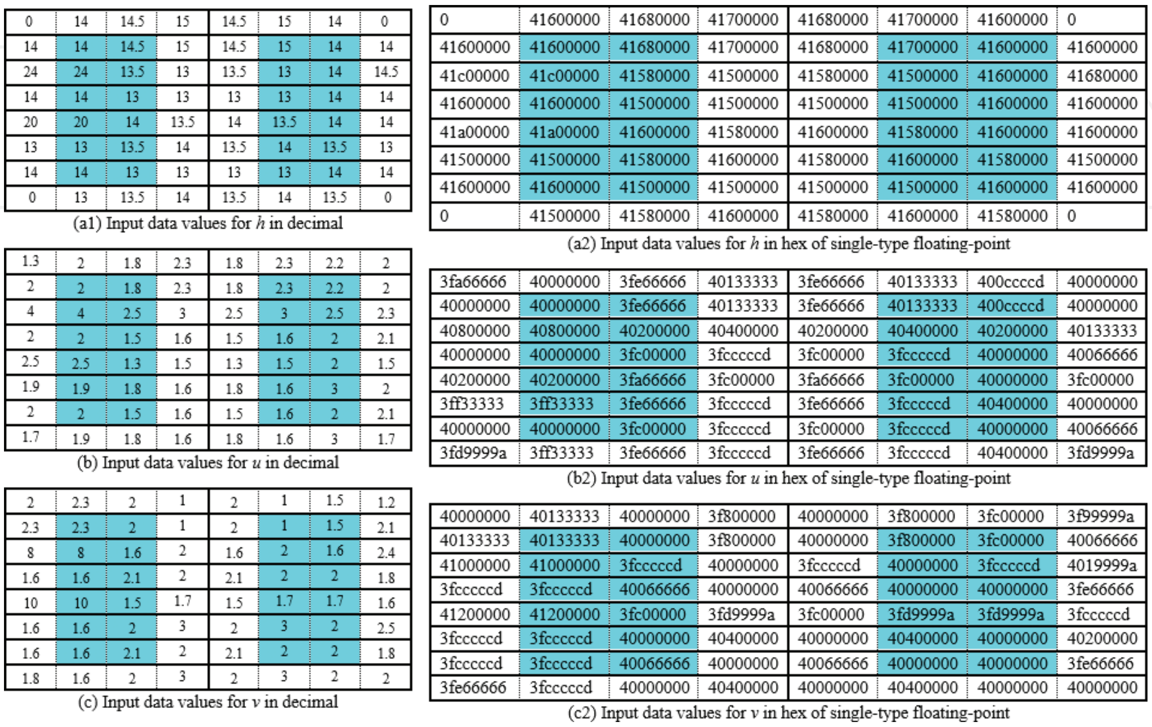


Figure 21.
The chip Virtex 6 (XCVL240T-1FFG1156) connected to PC for configuring to make CNN chip and performing calculation.

real numbers. Data into h, u, v are added with temporary boundaries, detailed as follow (presented in Decimal and Hex of Single-type Floating-point) (**Figure 22**).

The interface of each Input memory, Temporary memory for h, u, v is configurated as same in **Figure 23**. The initial data for the Input memory h, u, v is initialed by COE files. A COE file stores initial values for a memory (**Figure 24**).



0	14	14.5	15	14.5	15	14	0
14	14	14.5	15	14.5	15	14	14
24	24	13.5	13	13.5	13	14	14.5
14	14	13	13	13	13	14	14
20	20	14	13.5	14	13.5	14	14
13	13	13.5	14	13.5	14	13.5	13
14	14	13	13	13	13	14	14
0	13	13.5	14	13.5	14	13.5	0

(a1) Input data values for *h* in decimal

0	41600000	41680000	41700000	41680000	41700000	41600000	0
41600000	41600000	41680000	41700000	41680000	41700000	41600000	41600000
41c00000	41c00000	41580000	41500000	41580000	41500000	41600000	41680000
41600000	41600000	41500000	41500000	41500000	41500000	41600000	41600000
41a00000	41a00000	41600000	41580000	41600000	41580000	41600000	41600000
41500000	41500000	41580000	41600000	41580000	41600000	41580000	41500000
41600000	41600000	41500000	41500000	41500000	41500000	41600000	41600000
0	41500000	41580000	41600000	41580000	41600000	41580000	0

(a2) Input data values for *h* in hex of single-type floating-point

1.3	2	1.8	2.3	1.8	2.3	2.2	2
2	2	1.8	2.3	1.8	2.3	2.2	2
4	4	2.5	3	2.5	3	2.5	2.3
2	2	1.5	1.6	1.5	1.6	2	2.1
2.5	2.5	1.3	1.5	1.3	1.5	2	1.5
1.9	1.9	1.8	1.6	1.8	1.6	3	2
2	2	1.5	1.6	1.5	1.6	2	2.1
1.7	1.9	1.8	1.6	1.8	1.6	3	1.7

(b) Input data values for *u* in decimal

3fa66666	40000000	3fe66666	40133333	3fe66666	40133333	400ccccd	40000000
40000000	40000000	3fe66666	40133333	3fe66666	40133333	400ccccd	40000000
40800000	40800000	40200000	40400000	40200000	40400000	40200000	40133333
40000000	40000000	3fc00000	3fcccccd	3fc00000	3fcccccd	40000000	40066666
40200000	40200000	3fa66666	3fc00000	3fa66666	3fc00000	40000000	3fc00000
3ff33333	3ff33333	3fe66666	3fcccccd	3fe66666	3fcccccd	40400000	40000000
40000000	40000000	3fc00000	3fcccccd	3fc00000	3fcccccd	40000000	40066666
3fd9999a	3ff33333	3fe66666	3fcccccd	3fe66666	3fcccccd	40400000	3fd9999a

(b2) Input data values for *u* in hex of single-type floating-point

2	2.3	2	1	2	1	1.5	1.2
2.3	2.3	2	1	2	1	1.5	2.1
8	8	1.6	2	1.6	2	1.6	2.4
1.6	1.6	2.1	2	2.1	2	2	1.8
10	10	1.5	1.7	1.5	1.7	1.7	1.6
1.6	1.6	2	3	2	3	2	2.5
1.6	1.6	2.1	2	2.1	2	2	1.8
1.8	1.6	2	3	2	3	2	2

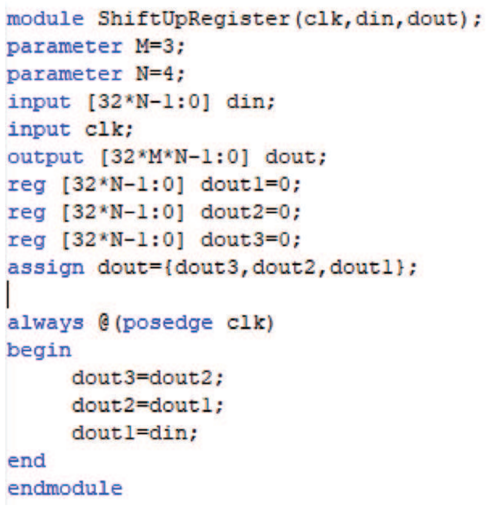
(c) Input data values for *v* in decimal

40000000	40133333	40000000	3f800000	40000000	3f800000	3fc00000	3f99999a
40133333	40133333	40000000	3f800000	40000000	3f800000	3fc00000	40066666
41000000	41000000	3fcccccd	40000000	3fcccccd	40000000	3fcccccd	4019999a
3fcccccd	3fcccccd	40066666	40000000	40066666	40000000	40000000	3fe66666
41200000	41200000	3fc00000	3fd9999a	3fc00000	3fd9999a	3fd9999a	3fcccccd
3fcccccd	3fcccccd	40000000	40400000	40000000	40400000	40000000	40200000
3fcccccd	3fcccccd	40066666	40000000	40066666	40000000	40000000	3fe66666
3fe66666	3fcccccd	40000000	40400000	40000000	40400000	40000000	40000000

(c2) Input data values for *v* in hex of single-type floating-point

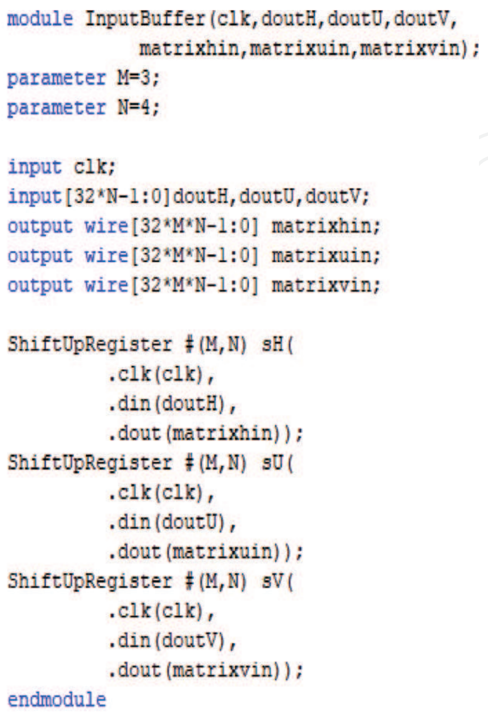
Figure 22.
Initial data for the Input memory h, u, v.

3.5.3 Shift up register



```
module ShiftUpRegister(clk,din,dout);
parameter M=3;
parameter N=4;
input [32*N-1:0] din;
input clk;
output [32*M*N-1:0] dout;
reg [32*N-1:0] dout1=0;
reg [32*N-1:0] dout2=0;
reg [32*N-1:0] dout3=0;
assign dout={dout3,dout2,dout1};

always @(posedge clk)
begin
    dout3=dout2;
    dout2=dout1;
    dout1=din;
end
endmodule
```



```
module InputBuffer(clk,doutH,doutU,doutV,
          matrixhin,matrixuin,matrixvin);
parameter M=3;
parameter N=4;

input clk;
input[32*N-1:0]doutH,doutU,doutV;
output wire[32*M*N-1:0] matrixhin;
output wire[32*M*N-1:0] matrixuin;
output wire[32*M*N-1:0] matrixvin;

ShiftUpRegister #(M,N) sH(
        .clk(clk),
        .din(doutH),
        .dout(matrixhin));
ShiftUpRegister #(M,N) sU(
        .clk(clk),
        .din(doutU),
        .dout(matrixuin));
ShiftUpRegister #(M,N) sV(
        .clk(clk),
        .din(doutV),
        .dout(matrixvin));
endmodule
```

3.5.4 CNN core

```
module CNNCore(clk,matrixhin,matrixuin,matrixvin,matrixhout,matrixuout,matrixvout);
parameter M=3;
parameter N=4;
parameter PipelineLength=6;

input clk;
input [32*M*N-1:0] matrixhin,matrixuin,matrixvin;
output[32*N-1:0] matrixhout,matrixuout,matrixvout;

ShiftRightN #(PipelineLength) HSR1(clk,matrixhin[32*N+31:32*N],matrixhout[31:0]);
ShiftRightN #(PipelineLength) HSR2(clk,matrixhin[32*(2*N)-1:32*(2*N)-32],matrixhout[32*N-1:32*N-32]);
ShiftRightN #(PipelineLength) HSR3(clk,matrixuin[32*N+31:32*N],matrixuout[31:0]);
ShiftRightN #(PipelineLength) HSR4(clk,matrixuin[32*(2*N)-1:32*(2*N)-32],matrixuout[32*N-1:32*N-32]);
ShiftRightN #(PipelineLength) HSR5(clk,matrixvin[32*N+31:32*N],matrixvout[31:0]);
ShiftRightN #(PipelineLength) HSR6(clk,matrixvin[32*(2*N)-1:32*(2*N)-32],matrixvout[32*N-1:32*N-32]);

generate
   genvar i,j;
   for(i=0;i<M-2;i=i+1)
      for(j=0;j<N-2;j=j+1)
         Cell TB(
            .clk(clk),
            .Hisubljsubl    (matrixhin[32*((i+2)*N+j+2)+31:32*((i+2)*N+j+2)]),
            .Hijsubl        (matrixhin[32*((i+2)*N+j+1)+31:32*((i+2)*N+j+1)]),
            .Hiaddljsubl    (matrixhin[32*((i+2)*N+j+0)+31:32*((i+2)*N+j+0)]),
            .Hisublj        (matrixhin[32*((i+1)*N+j+2)+31:32*((i+1)*N+j+2)]),
            .Hij            (matrixhin[32*((i+1)*N+j+1)+31:32*((i+1)*N+j+1)]),
            .Hiaddlj        (matrixhin[32*((i+1)*N+j+0)+31:32*((i+1)*N+j+0)]),
            .Hisubljaddl    (matrixhin[32*((i+0)*N+j+2)+31:32*((i+0)*N+j+2)]),
            .Hijaddl        (matrixhin[32*((i+0)*N+j+1)+31:32*((i+0)*N+j+1)]),
            .Hiaddljaddl    (matrixhin[32*((i+0)*N+j+0)+31:32*((i+0)*N+j+0)]),
            .Hijnew         (matrixhout[32*(i*N+j+1)+31:32*(i*N+j+1)]),
            .Uisubljsubl    (matrixuin[32*((i+2)*N+j+2)+31:32*((i+2)*N+j+2)]),
            .Uijsubl        (matrixuin[32*((i+2)*N+j+1)+31:32*((i+2)*N+j+1)]),
            .Uiaddljsubl    (matrixuin[32*((i+2)*N+j+0)+31:32*((i+2)*N+j+0)]),
            .Uisublj        (matrixuin[32*((i+1)*N+j+2)+31:32*((i+1)*N+j+2)]),
            .Uij            (matrixuin[32*((i+1)*N+j+1)+31:32*((i+1)*N+j+1)]),
            .Uiaddlj        (matrixuin[32*((i+1)*N+j+0)+31:32*((i+1)*N+j+0)]),
            .Uisubljaddl    (matrixuin[32*((i+0)*N+j+2)+31:32*((i+0)*N+j+2)]),
            .Uijaddl        (matrixuin[32*((i+0)*N+j+1)+31:32*((i+0)*N+j+1)]),
            .Uiaddljaddl    (matrixuin[32*((i+0)*N+j+0)+31:32*((i+0)*N+j+0)]),
            .Uijnew         (matrixuout[32*(i*N+j+1)+31:32*(i*N+j+1)]),
            .Visubljsubl    (matrixvin[32*((i+2)*N+j+2)+31:32*((i+2)*N+j+2)]),
            .Vijsubl        (matrixvin[32*((i+2)*N+j+1)+31:32*((i+2)*N+j+1)]),
            .Viaddljsubl    (matrixvin[32*((i+2)*N+j+0)+31:32*((i+2)*N+j+0)]),
            .Visublj        (matrixvin[32*((i+1)*N+j+2)+31:32*((i+1)*N+j+2)]),
            .Vij            (matrixvin[32*((i+1)*N+j+1)+31:32*((i+1)*N+j+1)]),
            .Viaddlj        (matrixvin[32*((i+1)*N+j+0)+31:32*((i+1)*N+j+0)]),
            .Visubljaddl    (matrixvin[32*((i+0)*N+j+2)+31:32*((i+0)*N+j+2)]),
            .Vijaddl        (matrixvin[32*((i+0)*N+j+1)+31:32*((i+0)*N+j+1)]),
            .Viaddljaddl    (matrixvin[32*((i+0)*N+j+0)+31:32*((i+0)*N+j+0)]),
            .Vijnew         (matrixvout[32*(i*N+j+1)+31:32*(i*N+j+1)])
            );
endgenerate
endmodule
```

3.5.5 Boundary updating

```
module BoundaryUpdating(iEnable,iLeftBoundary,iRightBoundary,oLeftBoundary,oRightBoundary);
parameter N = 4;

    input iEnable;
    input [32*N-1:0] iLeftBoundary;
    input [32*N-1:0] iRightBoundary;
    output reg[32*N-1:0] oLeftBoundary;
    output reg[32*N-1:0] oRightBoundary;
always @(iEnable or iLeftBoundary or iRightBoundary)
   begin
      oLeftBoundary=iLeftBoundary;
      oRightBoundary=iRightBoundary;
      if(iEnable)
         begin
            oRightBoundary[32*N-1:32*N-32]=iLeftBoundary[32*2-1:32*2-32];
            oLeftBoundary[32*1-1:32*1-32]=iRightBoundary[32*(N-1)-1:32*(N-1)-32];
         end
   end
endmodule
```

3.5.6 Control unit

The interface of Control unit is described as follows.

```
module Control(clk,CountCLK,rdaddressHUV,wraddressHUV,wren,
         rdaddressHUVnew,wraddressHUVnew,wrenNew,
         start,EnableBoundaryUpdating,finish,read);
parameter N=4
parameter m_row=8;
parameter Q_Column=2;

input clk;
input start;
input read;

output reg[8:0]rdaddressHUV=0;
output reg[8:0]wraddressHUV=0;
output reg wren=0;
output reg[8:0]rdaddressHUVnew=0;
output reg[8:0]wraddressHUVnew=0;
output reg wrenNew=0;
output reg finish=1;
output reg EnableBoundaryUpdating;
output reg[9:0]CountCLK=0;
```

3.5.7 System scheme

To verify the system, the interface of the top module of the system should include all the signals that we want to verify.

The top module is described as follows.

```
module CNNSystem(clk,start,rdaddr,doutH,doutU,doutV,
             matrixhin,matrixuin,matrixvin,matrixhout,matrixuout,matrixvout,
             wraddrTemp,wrenTemp,HNewTemp,UNewTemp,VNewTemp,
             rdaddrTemp,doutHNewTemp,doutUNewTemp,doutVNewTemp,
             EnableBoundaryUpdating,wraddr,wren,HNew,UNew,VNew,finish,CountCLK);
parameter M=3;
parameter N=4;
input clk;
input start;
//Read Input memory
output wire[5:0]rdaddr;
output wire [32*N-1:0]doutH,doutU,doutV;
//CNN core
output wire[32*M*N-1:0] matrixhin;
output wire[32*M*N-1:0] matrixuin;
output wire[32*M*N-1:0] matrixvin;
output wire[32*N-1:0] matrixhout;
output wire[32*N-1:0] matrixuout;
output wire[32*N-1:0] matrixvout;
//Write Temp memory
output wire[5:0]wraddrTemp;
output wire wrenTemp;
output wire[32*N-1:0]HNewTemp,UNewTemp,VNewTemp;
//Read Temp memory and Boundary updating
output wire[5:0]rdaddrTemp;
output wire [32*N-1:0]doutHNewTemp,doutUNewTemp,doutVNewTemp;
output wire EnableBoundaryUpdating;
//Write results Input memory
output wire[5:0]wraddr;
output wire wren;
output wire[32*N-1:0]HNew,UNew,VNew;
output wire finish;
output wire[9:0]CountCLK;
```

```
Control CU(
               .CountCLK(CountCLK),
               .wraddressHUVTemp(wraddrTemp),
               .rdaddressHUVTemp(rdaddrTemp),
               .wrenTemp(wrenTemp),
               .clk(clk),
               .wraddressHUV(wraddr),
               .rdaddressHUV(rdaddr),
               .wren(wren),
               .start(start),
               .EnableBoundaryUpdating(EnableBoundaryUpdating),
               .finish(finish));
InputMemoryHUV #(N) InputMemory(
                          clk,rdaddr,doutH,doutU,doutV,
                          wraddr,wren,HNew,UNew,VNew);
InputBuffer #(M,N) Buffer(
                          clk,doutH,doutU,doutV,
                          matrixhin,matrixuin,matrixvin);
CNNCore #(M,N) uut(
                          .clk(clk),
                          .matrixhin(matrixhin),
                          .matrixuin(matrixuin),
                          .matrixvin(matrixvin),
                          .matrixhout(matrixhout),
                          .matrixuout(matrixuout),
                          .matrixvout(matrixvout));
BoundaryUpdatingHUV #(N) Boundary(
                          matrixhout,matrixuout,matrixvout,
                          doutHNewTemp,doutUNewTemp,doutVNewTemp,
                          EnableBoundaryUpdating,
                          HNewTemp,UNewTemp,VNewTemp,
                          HNew,UNew,VNew);
TempMemoryHUV #(N) TempMemory(
               clk,wraddrTemp,wrenTemp,HNewTemp,UNewTemp,
               VNewTemp,
               rdaddrTemp,doutHNewTemp,doutUNewTemp,doutVNewTemp);
               endmodule
```

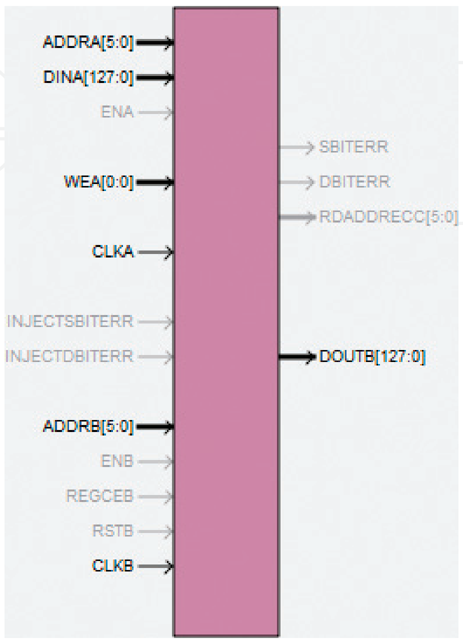
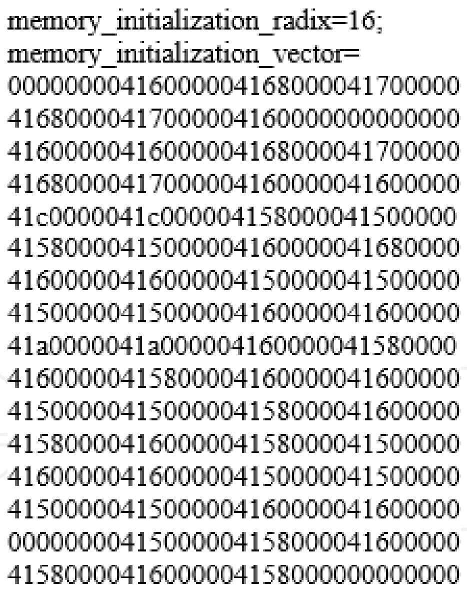


Figure 23.
Interface for Input and Temp memory h, u, v.



```
memory_initialization_radix=16;
memory_initialization_vector=
000000004160000041680000041700000
41680000417000004160000000000000
41600000416000004168000041700000
41680000417000004160000041600000
41c0000041c000004158000041500000
41580000415000004160000041680000
41600000416000004150000041500000
41500000415000004160000041600000
41a0000041a000004160000041580000
41600000415800004160000041600000
41500000415000004158000041600000
41580000416000004158000041500000
41600000416000004150000041500000
41500000415000004160000041600000
00000000415000004158000041600000
41580000416000004158000000000000
```

Figure 24.
An example of h.core file to initial data for the Input memory h.

3.6 Simulation results

The ISE design software shows the device utilization summary as in **Table 1**.

Figures 25–27 show the schematics synthesized by the ISE design software.

Comparing the new values of h in **Figure 28i, k** (doutH) with **Figure 29**, we can see that the 3x4 CNN system worked well.

The simulation results show the properness and effectiveness of installation methods. The cost for calculating the first three blocks of 1xN taken from memory units h, u, v is 10 clock pulses, of which 1 clock pulse is for initial reading Input memory, 3 clock pulse is for initial updating buffer to CNN, and 6 clock pulses for initial calculation. Each successive 1xN unit takes only 1 clock pulse to calculate, due to the use of the pipeline mechanism to update buffer to CNN and calculate at CNN arithmetic unit. After finishing reading each column of blocks of data in the Input memory, it needs 2 more clocks for initiating the buffer again. It also takes 1 clk for initial writing Temp memory, 1 clk for initial reading Temp memory, and 1 clk for initial writing result back to Input memory.

Devices used summary (estimated values)			
Logic utilization	**Used**	**Available**	**Utilization**
Number of slice registers	3952	301,440	1%
Number of slice LUTs	16,365	150,720	10%
Number of fully used LUT-FF pairs	1770	18,547	9%
Number of bonded IOBs	3112	600	518%
Number of Block RAM/FIFO	12	416	2%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48E1s	132	768	17%

Table 1.
Device utilization summary.

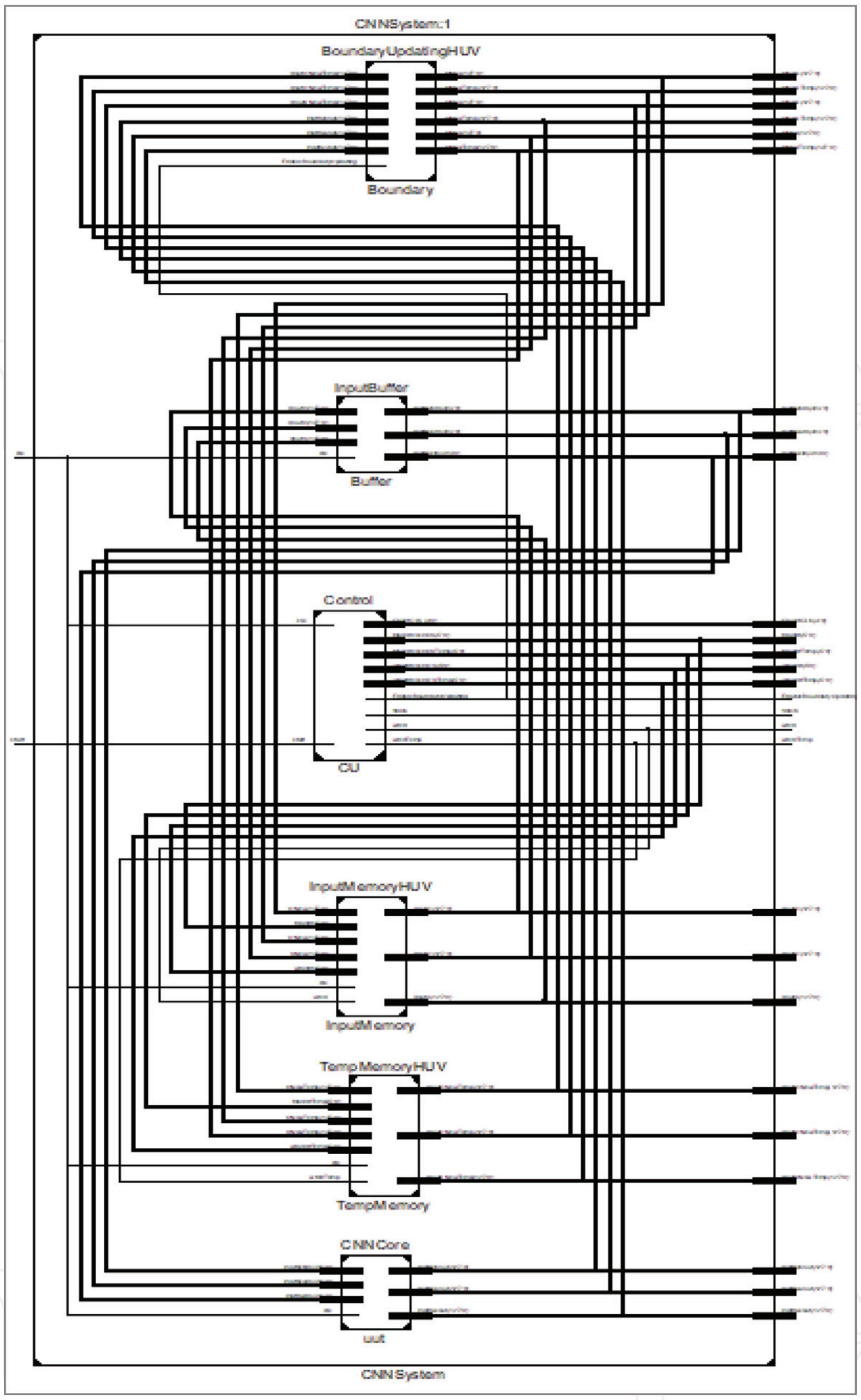


Figure 25.
The architecture of CNN chip.

As a result, the time for one computing cycle is:

$$T = 8 + m(Q + 1)\ (clk)$$

As the above implementation, m = 8, Q = 2, and T = 32 (clk).

4. Conclusion

This chapter gives the solution for configuring CNN chip to solve Navier-Stokes equations, especially concerning to solution in the temporary boundary problem

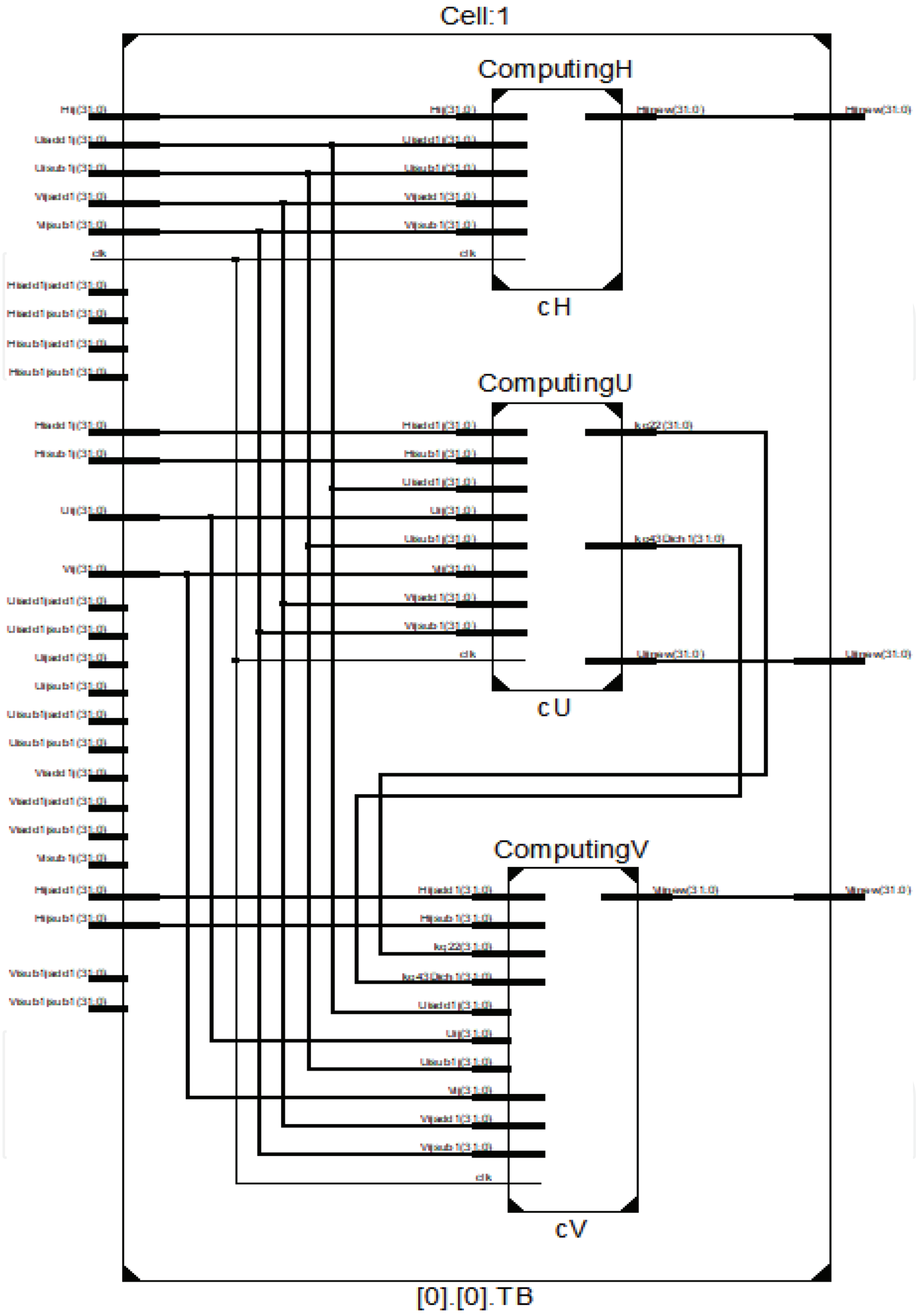


Figure 26.
The architecture of one CNN cell.

when it is required. The purpose is to divide the big data space into many subspaces. The processing of the big data space is based on the calculation of each subdata. With the input data of 32-bit floating point real number and FPGA chip Virtex 6 XCVL240T-1FFG1156, the CNN of 1x12 cells has successfully installed. The installation results show that the effectiveness of this solution mainly lies on the

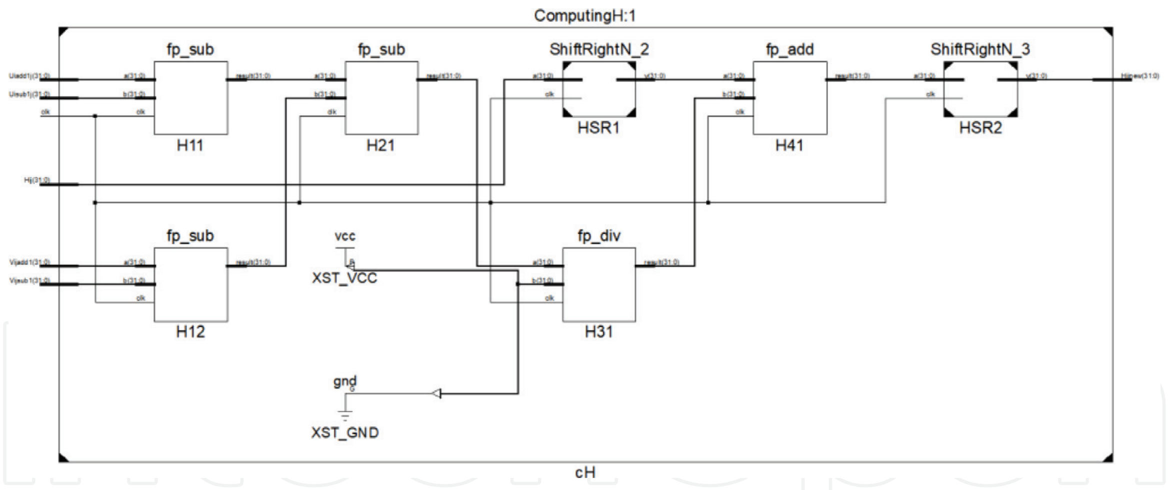
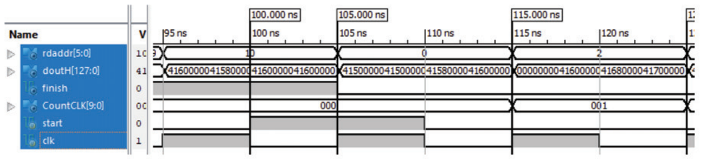
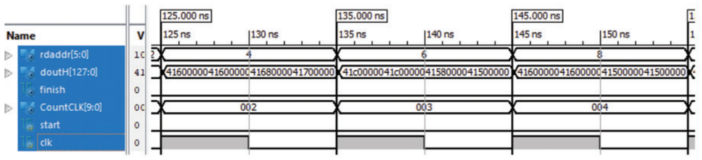


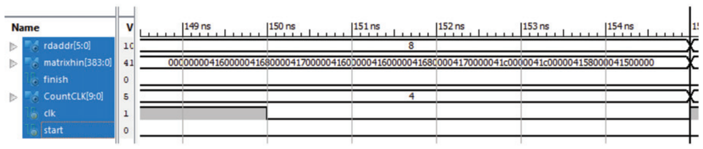
Figure 27.
Inside electronic circuit for h.



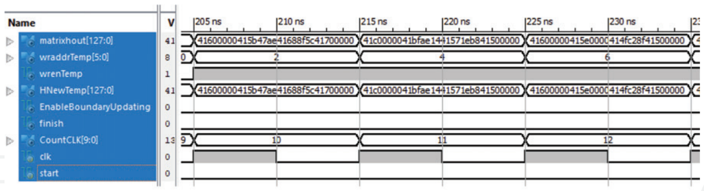
(a)



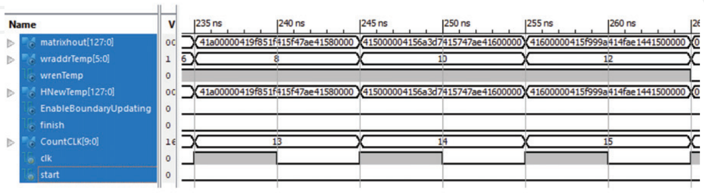
(b)



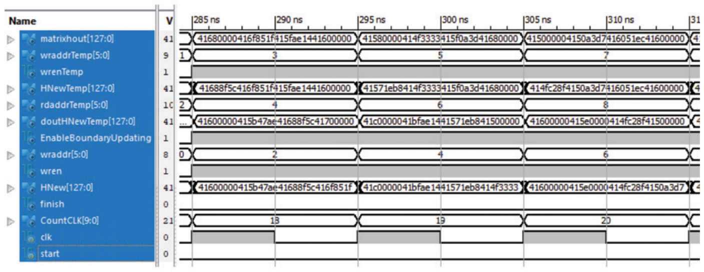
(c)



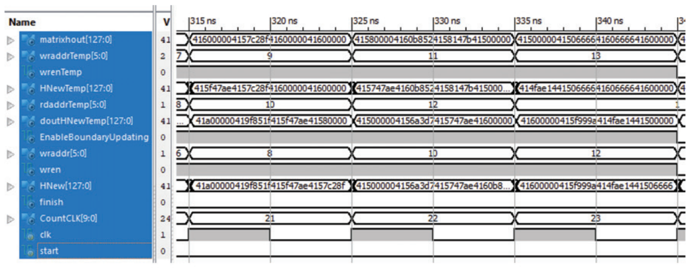
(d)



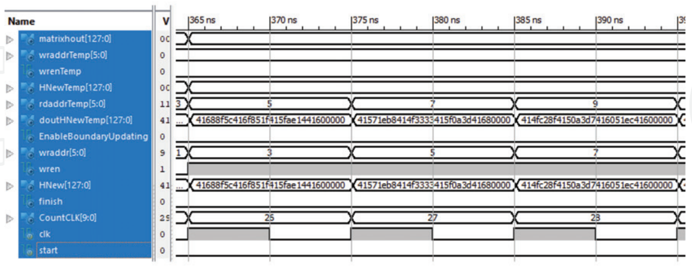
(e)



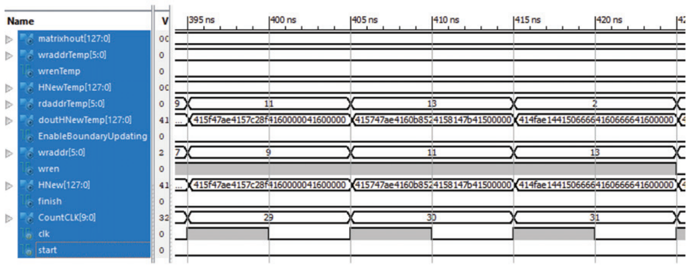
(f)



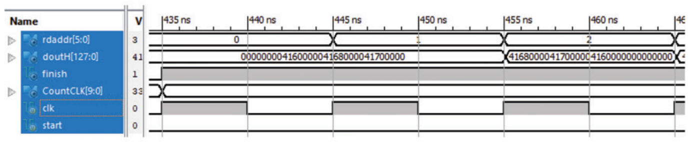
(g)



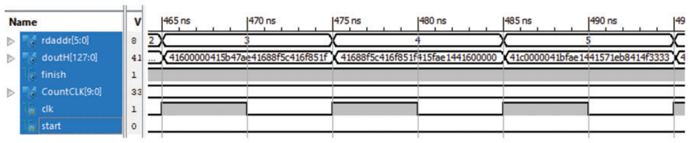
(h)



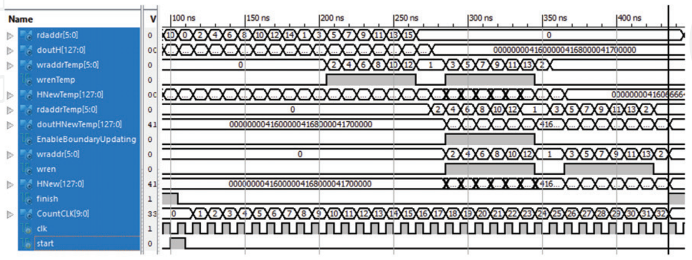
(i)



(j)

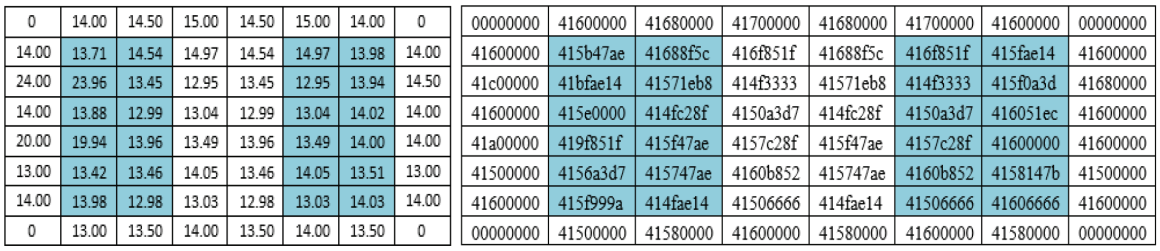


(k)



(l)

Figure 28.
Signals operating inside the 3x4 CNN system, m = 8, Q = 2. (a) Starting a computing cycle by setting start = 1. (b) The output of Input memory (doutH). (c) The data outputting from Buffer after 4 clks. (d) The results from CNN core after 10 clks; and start writing the results to Temp memory. (e) The CNN core finish computing the first column of blocks of data at 16 clks; and pause writing the results to Temp memory at 16 clks. (f) The results from CNN core after 18 clks; read Temp memory, start updating boundaries, and write the results to Input memory. (g) Pause updating boundaries from 24 clks. (h) The CNN core finishes computing; read the last column of blocks of data from Temp memory and write to Input memory. (h) Finish writing all results of the first computing cycle to Input memory. (i) The controller sets finish = 1 at 33 clks. (k) The output of Input memory shows the results computed at previous computing cycle. (l) The overview of signals.



0	14.00	14.50	15.00	14.50	15.00	14.00	0
14.00	13.71	14.54	14.97	14.54	14.97	13.98	14.00
24.00	23.96	13.45	12.95	13.45	12.95	13.94	14.50
14.00	13.88	12.99	13.04	12.99	13.04	14.02	14.00
20.00	19.94	13.96	13.49	13.96	13.49	14.00	14.00
13.00	13.42	13.46	14.05	13.46	14.05	13.51	13.00
14.00	13.98	12.98	13.03	12.98	13.03	14.03	14.00
0	13.00	13.50	14.00	13.50	14.00	13.50	0

00000000	41600000	41680000	41700000	41680000	41700000	41600000	00000000
41600000	415b47ae	41688f5c	416f851f	41688f5c	416f851f	415fae14	41600000
41c00000	41bfae14	41571eb8	414f3333	41571eb8	414f3333	415f0a3d	41680000
41600000	415e0000	414fc28f	4150a3d7	414fc28f	4150a3d7	416051ec	41600000
41a00000	419f851f	415f47ae	4157c28f	415f47ae	4157c28f	41600000	41600000
41500000	4156a3d7	415747ae	4160b852	415747ae	4160b852	4158147b	41500000
41600000	415f999a	414fae14	41506666	414fae14	41506666	41606666	41600000
00000000	41500000	41580000	41600000	41580000	41600000	41580000	00000000

Figure 29.
The new values of h computed by excel for the first computing cycle.

expansion of calculation space and resource saving and the accuracy of the calculation acceptable as well. This model can be further developed to feasibly solve similar problems in larger computing space and could be developed for some types of complicated (mixed) boundaries as well.

Acknowledgements

We would like to deeply acknowledge Professor Roska Tamas, the head of the Analogic and Neural Computing Research Laboratory and Chairman of the Scientific Council—Institute of the Hungarian Academy of Sciences; and Associate Professor Pham Thuong Cat, the Head of Automation Laboratory—Institute of Information Technology—Vietnam Academy of Science and Technology, for giving us many important instructions.

Author details

Vu Duc Thai* and Bui Van Tung
Thai Nguyen University, Thai Nguyen, Vietnam

*Address all correspondence to: vdthai@ictu.edu.vn

IntechOpen

© 2019 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

References

[1] Chua LO, Yang L. Cellular neural networks: Theory. IEEE Transactions on Circuits and Systems. 1988;**35**(10): 1257-1272

[2] Chua LO, Yang L. Cellular neural networks: Application. IEEE Transactions on Circuits and Systems. 1988;**35**:1273-1290

[3] Roska T, Chua LO, Wolf D, Kozek T, Tetzlaff R, Puffer F. Simulating nonlinear waves and partial differential equations via CNN—Part I: Basic techniques. IEEE Transactions on Circuits and Systems. 1995;**42**(10): 807-815

[4] Thai VD, Cat PT. Modeling air pollution problem by cellular neural network. In: Proceeding (ISI) of 10th International Conference on Control, Automation, Robotics and Vision; Hanoi, Vietnam; 2008. pp. 1115-1118

[5] Thai VD, Cat PT. Solving two-dimensional Saint venant equation by using cellular neural network. In: Proceeding of the 7th Asian Control Conference—ASCC2009; Hong Kong; 2009. pp. 1258-1263

[6] Thai VD, Cat PT. Equivalence and stability of two-layer cellular neural network solving Saint venant 1D equation. In: Proceeding (ISI) of 11th International Conference on Control, Automation, Robotics and Vision (ICARCV2010); Singapore; 2010. pp. 704-709

[7] Thai VD, Anh BT, Duong VT. Develop some application of Cyclone—DE2C35 chip. Journal of Science and Technology - Thai Nguyen University. 2015;**10**(140):103-108

[8] Thai VD, Linh LH, Linh NM. Solving Navier-Stokes equation using FPGA cellular neural network chip. In: Proceeding of International Conference on Advances in Information and Communication Technology (ICTA2016). Springer Publishing; 2016. pp. 562-571

[9] Rusin WM. On solution to Navier-Stokes equation in critical spaces [Thesis of Doctor Philosophy]. 2010. Available from: http://conservancy.umn.edu/.../Rusin_umn_0130E_11277.pdf

[10] Hruska J. Intel launches Stratix 10: Altera FPGA combined with ARM CPU, 14nm manufacturing. Extremetech. 2016. Available from: https://www.extremetech.com/computing/237338-intel-launches-stratix-10-altera-fpga-combined-with-arm-cpu-14nm-manufacturing

[11] la Pedus M. Intel-Altera deal to shake up foundry landscape. Chip Design Magazine. 2013. Available from: http://chipdesignmag.com/display.php?articleId=5215

[12] Clive M. The Design Warrior's Guide to FPGAs: Devices, Tools and Flows. Elsevier; 2004. Available from: https://www.eu.elsevierhealth.com/about-us

[13] Clive M. Programmable Logic DesignLine. Xilinx Unveil Revolutionary 65nm FPGA Architecture: The Virtex-5 Family. 2006. Available from: https://www.eetimes.com/document.asp?doc_id=13001899

[14] David WP. Google patent search. Dynamic Data Reprogrammable PLA. 2009

[15] David WP, Peterson LR. Google patent search. Dynamic Data Reprogrammable PLA. 2009

[16] Wisniewski R. Synthesis of Compositional Microprogram Control Units for Programmable Devices.

University of Zielona Góra Press; 2009, (ul. Podgórna 50, 65-246 Zielona Góra. Dane kontaktowe)

[17] Black F, Scholes MS. The pricing of options and corporate liabilities. Journal of Political Economy. 1973;**81**(3):59-637

[18] Chua LO, Roska T. Cellular Neural Networks and Visual Computing. Cambridge University Press; 2000. ISBN: 0-521-65247-2. Available from: https://en.wikipedia.org/wiki/Cambridge_University_Press

[19] FPGA Architecture for the Challenge. Available from: http://www.eecg.toronto.edu/∼vaughn/challenge/fpga_arch.html

[20] Intel® FPGAs offer a wide variety of configurable embedded SRAM, high-speed transceivers, high-speed I/Os, logic blocks, and routing. Built-in intellectual property (IP) combined with outstanding software tools lower FPGA development time, power, and cost. Available from: https://www.intel.com/content/www/us/en/products/programmable/fpga.html