Development and Evaluation of two Memory Architectures for the Hardware Version of CNN Face Recognizer

Basil Sh. Mahmood
College of Electronic Engineering,
University of Mosul,
basil_mahmood@yahoo.com

Shefa A. Dawwd
Computer Department, College of Engineering, University of Mosul,
shefadawwd@yahoo.com

Abstract

The design of the proposed Convolutional Neural Network (CNN) architecture for face image recognition takes the constraints on the bandwidth of the communications between memory and processor into the account. The coarse grained parallelism which performed in the bottom layer node's calculations is reduced in consequent manner until the calculation of one simple node in the upper layer is achieved sequentially. Two methods of segmentation are used to buffer the image data required for these parallel to sequential calculations from the image RAM to multi-port RAMs. A comparison between these two methods with respect to the whole number of RAM access required to generate the system recognition code is performed. A speedup of 44 is achieved when the hardware system is implemented with the using of the 1st method of segmentation as compared to a Pentium 4, 2.4 GHz sequential computer software implementation. While a speedup of 88 is achieved when the same hardware system is implemented but with the using of the 2nd segmentation method, compared to the same mentioned sequential computer.

Keywords: convolution neural networks, parallel processing, memory architecture.

تطوير وتقييم نموذجين من معماريات الذاكرة الخاصة بالنموذج المادي لمميز الوجوه المستخدم للشبكات العصبية اللافوفية

باسل شكر محمود
كلية هندسة الالكترونيات
جامعة الموصل

شفاء عبدالرحمن داؤد
قسم الحاسبات – كلية
الهندسة
جامعة الموصل

الخلاصة

ان تصميم معمارية الشبكة العصبية اللافوفية المخصصة لمهام تمييز صور الوجوه اخذ هذه مشكلة الى ظهور مشكلة عدم توائم نقل المعلومات بين المعالج والذاكرة بعين الاعتبار.لقد اعتمد التصميم الاساس للمنظومة بكل وحداته على المعيار الذي ينصح باستخدام المعالجة الرقمية المتوازية في طبقات الشبكة الهرمية القريبة من الادخال وتقليل التوازي تدريجيا بالاقتراب من طبقة الاخراج. ولغرض تسريع الحسابات فقد تم تصميم وحدة ادخال وتقطيع المعلومات الصورية اللازمة للتنفيذ المتوازي/ المتوالي المعتمد على المعيار اعلاه بشكل كفوء وذلك اعتمادا على مبدأ الخزن الكاشي لمتجهات الادخال في مخازن FPGA متعددة المرافئ المسماة RAMs واللازمة لبدء الحسابات في المعالجات المتوازية.تم تصميم وحدة ادخال وتقطيع المعلومات الصورية بطريقتين، ثم اجريت مقارنة بين الطريقتين نسبة الى العدد الكلي لمرات الوصول الى الذاكرة اللازمة لاكمال الحسابات اللازمة لانتاج شفرة كل صورة مطبقة على ادخال المنظومة. تم تقييم عمل منظومة الزمن الحقيقي المادية المنفذة من خلال مقارنة انجازها مع منظومة برمجية مصممة لنفس الغرض ومنفذة باستخدام حاسبة شخصية(Pentium 4, 2.4 GHz) . تم الحصول على تسارع بمقدار88 عند استخدام الطريقة الاولى مقارنة مع تسارع بمقدار 44 تم الحصول عليه باستخدام الطريقة الثانية .

Received 26 March 2007 **Accepted 10 Sep. 2007**

1 Introduction

Any CPU-based implementation of real-time image processing algorithm has two major bottlenecks: data transfer bandwidth and sequential data processing rate. A CPU adds large overhead to the actual computation. Its time splits among moving data between memory and registers, applying arithmetic and logic operators on the data, and keeping the algorithm flow, handling branches, and fetching code. To complete all these tasks, large number of clock cycles are needed.

Most Digital Image Processing (DIP) algorithms can be decomposed into simpler operations [1], which are commonly categorized as global, local, or point operations. Global image processing operations produce an output in which every value is a function of every pixel in the original image. An example is a histogram operation, in which the number of times each individual intensity value or range of intensity values occurs in the image is counted. This class of operations tends to be the most difficult to parallelize. A local, or windowed, operation produces an output that depends on the pixels in the local area about a pixel. An example of such an operation is an averaging filter, which can be used to smooth out grainy images. While some local operations are more easily parallelizable than others, it is generally possible to process multiple windows in parallel, speeding up the computation. Finally, point (or *pixel*) operations produce an output (generally a second image) in which each output value is a function of only one pixel of the image. These operations are the most easily parallelizable – in the extreme, every pixel could be processed simultaneously.

Although image-processing operations may access memory in different patterns, they share some very important features. First, they are all very memory intensive. Traditional computing applications perform a significant amount of processing on each piece of data before loading the next. However, in image processing at least one new pixel of information is typically needed for each step in the computation. According to one study using traditional DSP's and a variety of common algorithms (threshold detection, erode/dilate, median, convolve, and others) there are up to two memory accesses for each floating-point instruction [2].

The temporal locality of the image-processing operations is minimal. When a pixel is accessed for a local or global operation, it might never be used again until the operation is repeated. In windowed operations, the pixel is only needed for the duration of the time it is in the window; when the window moves past a pixel it will not be accessed again until the window reaches the end of the current row. Thus, traditional caching schemes cannot effectively speed up or reduce the number of memory accesses. However, the spatial locality of image processing operations tends to be quite high; if a certain pixel of a frame buffer is accessed, the most likely next accesses are to its immediate neighbors. Because of the two-dimensional nature of the image (plus the added dimension of time in video processing), pixels that are adjacent in the image may not be located at adjacent memory addresses. This reduces the locality of the data unless complicated memory addressing is used.

An efficient image processor is the one which can overcome on the bottlenecks that appear in the mentioned CPU implementation. The sequential processing rate major bottleneck can be overcome simply by using multiple processors to process the acquired data, whereas, it is difficult to overcome the data transfer bandwidth bottleneck, since the acquired image data from image sensors is assumed to be stored in one frame buffer. Therefore, that imposed that any accessing to the data should be done sequentially since it can not address more than one location in one clock cycle.

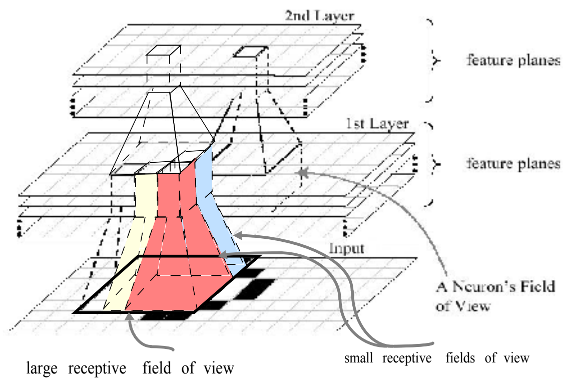
This paper presents a prototype of an embedded face-recognition system completely implemented in hardware. The system uses the Convolutional Neural Network (CNN) to recognize each class of face's images. A CNN [3] prototype was built using a field programmable gate array (FPGA). In CNN, every image window is convolved with a number of pre-trained weight vectors (features). Input image may contain a number of overlapped windows equal to the number of pixels in the image. That means, a huge number of memory accesses are required to complete the convolution processes every image. Each pixel should be accessed many times depend on the number of windows and the number of of pixels in each window in the image. With the data transfer bandwidth bottleneck mentioned above adds a negative effect to the benefit and speedup expected from the hardware implementation of the convolution processes. To minimize this effect, researchers deal with it in different ways. For example, by caching the last windows in internal registers to use its pixel’s data in the next windows calculations [4], or by using a high density DRAM integrated with computing logics on a single die to store the image data[1], where a window of 16 pixels could be loaded by a bus width of 128 bits in a single DRAM cycle. These strategies were dealing with the problem of minimizing the same pixel reread, while the approach proposed in this paper tries to solve the problem of pixel reread from the image frame buffer and to parallelize the pixel buffering to different storage elements. Therefore, this paper concentrates on the method used to reduce the data transfer bandwidth major bottleneck mentioned earlier in this section.

The rest of this paper is organized as flows: second section is a general overview to the CNN face recognition system with the proposed image segmentation methods required to transfer the image data for calculations, and their algorithms. Section 3 includes the results and section 4 represents the conclusions.

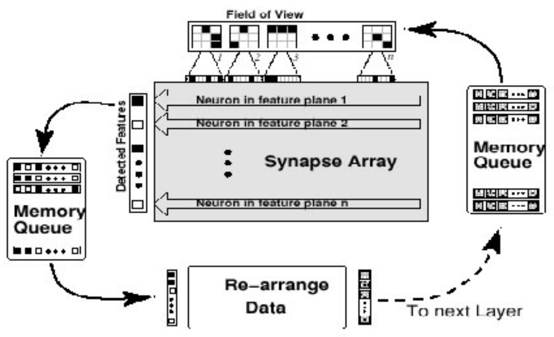
2 Hardware Implementation Of Cnn

One of the intentions in developing the parallel convolutional neural hardware is to provide a powerful platform for challenging biological inspired applications. Here, we propose a way to utilize this system to implement the Neocognitron, a massively parallel neural architecture for image understanding, introduced in the early eighties by Fukushima [5]. In this network model, neurons are organized in functionally equivalent layers (Fig. 1(a)). Each layer extracts certain shape-features, as for example edge orientation, from a localized region of the preceding layer and projects the extracted information to the next higher layer.

The complexity and abstractness of the detected features grow with the layer height, until complicated objects can be recognized. A layer consists of a number of feature planes, each of which is assigned to recognize one specific image feature. Neurons belonging to the same plane are identical in the sense that they share the same synaptic weights. This architecture, showing a high degree of self-similarity, seems particularly dedicated to be implemented on a parallel hardware platform.



(a)



(b)

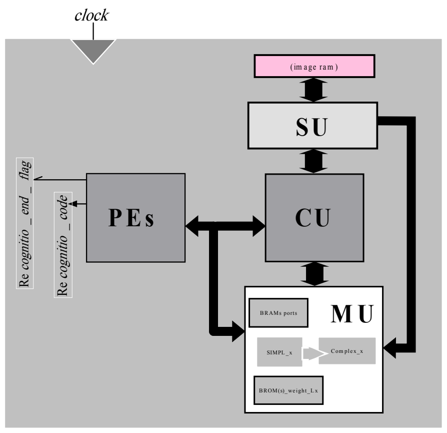


Figure1: (a) The Neocognitron's hierarchical network. For simplicity, only the first two layers are shown. Every pixel in each feature plane corresponds to one neuron, receiving its input from a localized region of all feature planes of the preceding layer. (b) Implementation on the hardware. All small receptive fields relate to one large receptive field are executed in parallel, each tuned to a specific feature. Between adjacent layers, two memory queues act as data buffers. A special stage is necessary to rearrange the layer output into field of views serving as input for the next layer. (c) CNN Layer implementation blocks.

(c)

Parallel, distributed processing, seen in convolutional neural network architectures, are an attractive model for computation. An FPGA VLSI architecture that implements the hierarchical convolutional networks is proposed [9][10]. Because the number of processing circuits integrated in an FPGA chip is restricted, it is difficult to realize all connections of the hierarchical network by real processing circuits. Therefore, in the designed architecture, neuron circuits are repetitively used by time-sharing operation. Time-sharing operation implementation in the convolutional network is shown in Fig.1(b).

In the other hand, and for the same reason mentioned above, all layers use the same resources to build a compact resources shared system, therefore, all layers are executed on the same processing elements unit. The CNN architecture is a processor array with SIMD control. It consists of four parts as shown in Fig.1(c). This is a fairly SIMD array configuration with one control unit (CU), one memory unit (MU), one segmentation unit (SU) and many processing elements unit (PEs).

In order to use the network modules in parallel, strategies are necessary to coordinate the distributed resources. Therefore , the CU controls the PEs and manages the data flows and signals over the PEs and other units. CU is responsible of beginning and ending of calculations in each element in the PEs, and their synchronizations. It consists of many status registers that manage the successive calculation operations performed in the PEs. It has also many counters and sequencers which provide the necessary addresses needed to change data between PEs and MU. MU is used to store the input data, weights, and other temporary data which generate during calculations in BRAMs, BROMs and internal registers respectively. PEs are used to manipulate the CNN layer's calculations in consequent manner as it will be discussed in section 2.1.1.

2.1 Segmentation Unit (SU)

The image is assumed to be captured and stored in a single memory RAM, in row order. In CNN the raw image is partitioned to overlapped field of views (receptive fields).

In this work, a large receptive field* is defined. This receptive field contains many overlapped small receptive fields. Note that, the image has numbers of overlapped large receptive fields.

A large receptive fields are processed sequentially, while their small receptive fields are processed in parallel to speedup the calculations(as mentioned in section 1: window operation). Parallel processing needs the access of the input data in parallel, which could not be met with the data stored in a single port memory. Therefore, it is urgently needed to partition the image RAM and transferring the data stored in, to multi-port RAMs. The total number of ports equal to the number of processing elements**.

The SU is responsible to buffer the data from image RAM to multi-port (dual port in the FPGA)RAMs. SU provide the necessary interleaving addresses and signals to perform the required action. SU is interconnected with CU and MU, while it has no connection with the PEs.

The bottleneck problem associated with the SU is its sequential implementation since in each access to the single image RAM, it can not address more than one location in one clock time. This problem adds a negative effect to the speedup. But in this paper, an efficient segmentation method to transfer the data from image RAM to multi-port RAMs that resides in the FPGA is used. To discuss this method in details, it is necessary to compare the method with the traditional interleaving method and it is necessary to depict the techniques which relate to parallel logical design of the segmentation unit.

2.1.1 Reading the receptive field vectors

Although using *serial* digital signal processing is advisable at the upper levels of the hierarchy, it might not be so adequate for early processing[6]. Operating with images at the bottom level of the processing hierarchy implies intensive memory accesses and poses important constraints on the bandwidth of the communications between memory and processor. Also, having a chip to sense the visual information (*imager*) and another one to process it (*processor*), requires high-speed data conversions and transferences to achieve large frame rates. As can be seen in Fig. 1, all small receptive field vectors related to one large receptive field area are convolved in parallel, therefore the data for all small receptive field vectors should be accessed in parallel.

***each large receptive field covers a part of input image which is required to calculate the output of one simple cell in the last layer(see Fig. 1(a)).**

**** In this work, and since each 4 PEs calculate 4 Manhattan distances among the same small receptive field vector and 4 weight vectors(features), therefore, it is required one storage element for each 4 PEs.**

The image processing algorithms of the CNN accesses the image data (as usual) from external image RAM. The portion of this data that is to be processed should be temporarily buffered into internal memories of the chip. These buffers are

mapped on the FPGA RAM blocks which operate at the same clock frequency that the FPGA elements operate.

Before discussing the proposed methods of transferring the windows data from the image RAM to multi-port BRAMs in FPGA, it is necessary first to give some details about the parameters of the designed system that is used as face recognizer. A resolution of 32x32 pixels is sufficient for the task of face recognition, since a face is a primarily characterized by the existence of eyes, nose and mouth together with their geometrical relationship all of which can be recognized at low spatial resolution[7]. Receptive fields sizes are chosen as 5x5, with 4 overlapped pixels(each field is overlapped over another by four pixels in both horizontal and vertical directions). The features in the hidden layers are organized as (4x4)x4, with 2 overlapped pixels,(4x4)x4, with 3 overlapped pixels,(4x4)x16, with 2 overlapped pixels, and (4x4)x16.To complete calculations of one simple node of the last layer, it should store 100 of 5x5 small receptive fields of a 14x14 large receptive field as vectors (see Fig. 2).

2.1.2 First method of segmentation

If the image data to be processed is stored in one single port RAM, then there is no chance to deal with more than one location in a single clock cycle. Also, accessing the consequent pixels from such RAM needs a complex addressing scheme.

Therefore, the data needs to be grouped and re-arranged. So, each acquired pixel should be written to one of the one_port_BRAMs(the block RAM memory has two completely independent access ports, labeled Port A and Port B). Therefore it can be splitted into two equal block RAMs, the first is with port A and called BRAM-A, and the second is with port B and called BRAM-B. Each part can be used as a single BRAM of one port and called one_port_BRAM. Thus, an interleaving addressing scheme in used to switch from one_port_BRAM to another (see Fig. 3(a)).

For Spartan-3 FPGA chip that is used for the implementation, each BRAM can be divided into two one_port_BRAMs of 8k bytes each. Further, this one_port_BRAM is partitioned to 32 partitions, each of length 32 bytes. Each partition is enough to store one small window(small receptive field vector), consequently, each one_port_BRAM can store 32 vectors. For example, the first 25 pixels_vector occupies the first 25 of 8-bits locations from the 1st partition of the first one_port_BRAM (C0=25, see the enlarged part of a one_port_BRAM in Fig.3(a)). The second 25 pixels_vector occupies the 2nd partition of the 1st one_port_BRAM, and so on. For one large receptive field of 14x14 pixels, 100 segments are required and should be equally distributed over all 3 BRAMs. The 1st one_port_BRAM holds 20 of small receptive field vectors (C1=20, refer to Fig.3(a)). The 2nd one_port_BRAM holds the second 20 segments, and so on. The algorithm calculates the physical address of the pixel image_RAM in order to be buffered from the image_RAM to the multi-ports BRAMs that reside in the FPGA (here, 5 one_port_BRAMs are used for 5 shared weights simple nodes to be processed in parallel each to detect one of the 4 features for their corresponding small receptive fields using 20 PEs) is summarized in Fig. 3(b).

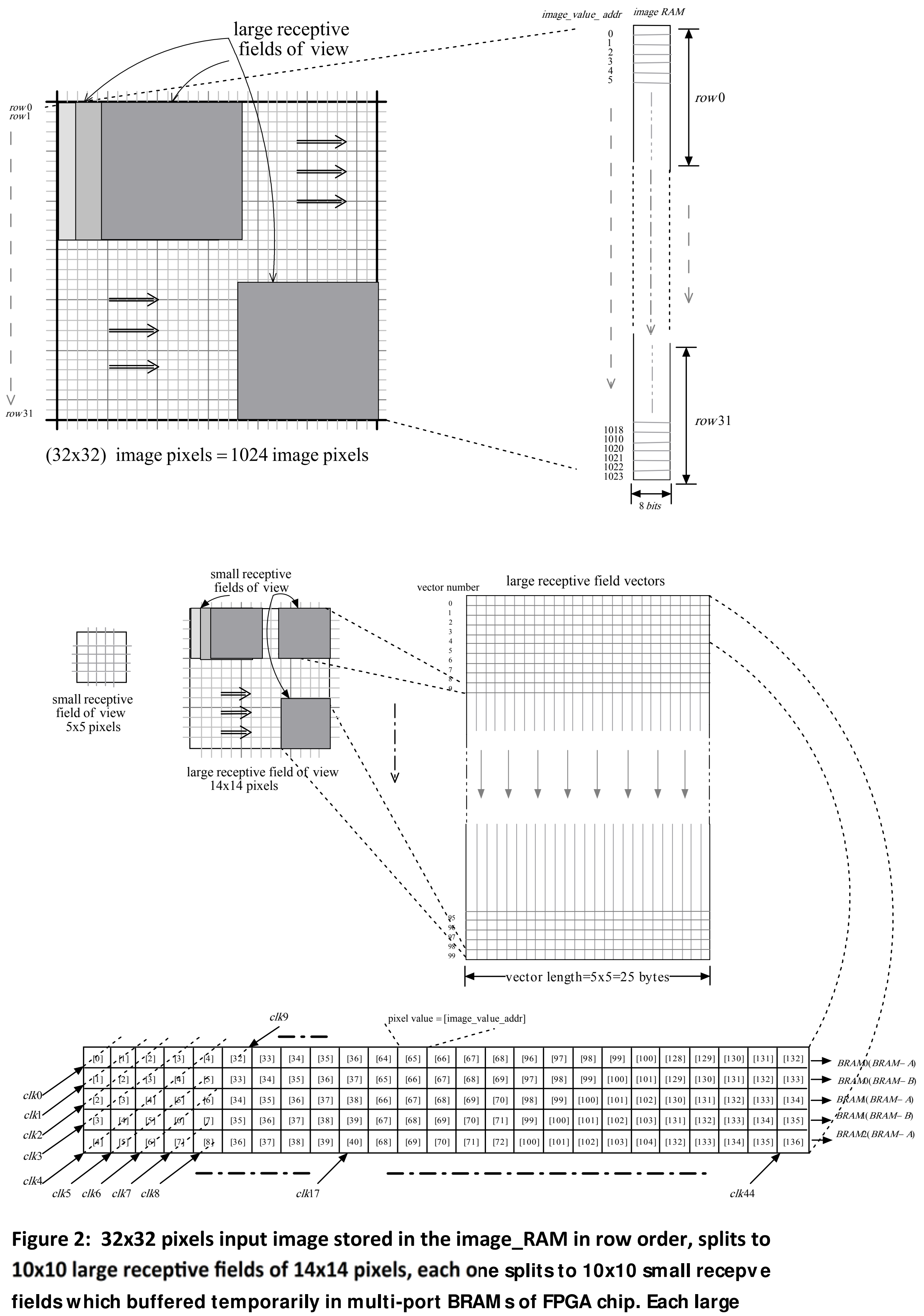
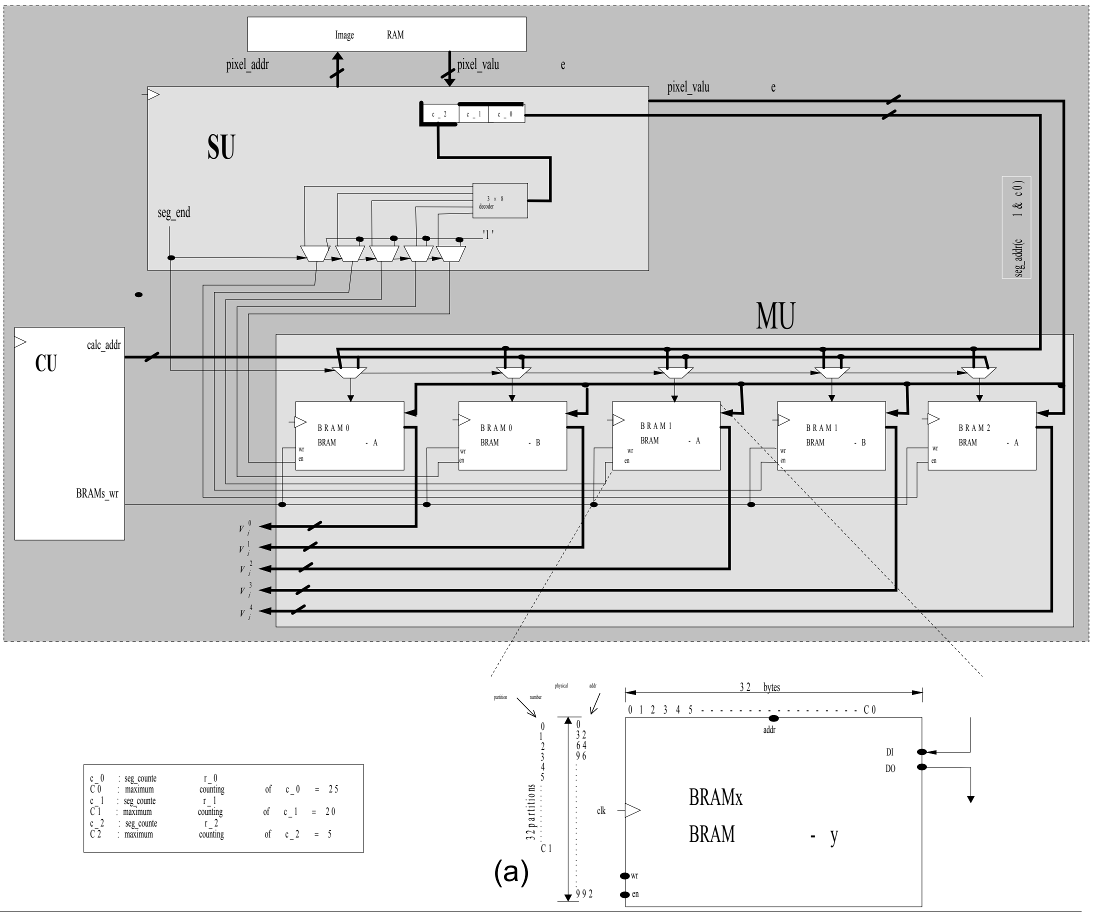

large receptive fields of view
image_value_ addr
image RAM
row0
row1
row31
(32x32) image pixels = 1024 image pixels
8 bits
small receptive fields of view
small receptive field of view 5x5 pixels
large receptive field of view 14x14 pixels
vector number
large receptive field vectors
vector length=5x5=25 bytes
pixel value = [image_value_addr]
clk0
clk1
clk2
clk3
clk4
clk5
clk6
clk7
clk8
clk9
clk17
clk44
BRAM0(BRAM– A)
BRAM0(BRAM– B)
BRAM1(BRAM– A)
BRAM1(BRAM– B)
BRAM2(BRAM– A)

Figure 2: 32x32 pixels input image stored in the image_RAM in row order, splits to 10x10 large receptive fields of 14x14 pixels, each one splits to 10x10 small recepve fields which buffered temporarily in multi-port BRAMs of FPGA chip. Each large receptive field covers the required area to complete one simple node calculations of



(a)

```
for v=0 to ($V_y$*2-2) "step=2'
    for u=0 to ($U_x$*2-2) "step=2"
        for n=0 to ($N_y$-1) "step=1"
            for m=0 to ($M_x$-1) "step=1"
                for j=0 to ($J_y$-1) "step=1"
                    for i=0 to ($I_x$-1) "step=1"
                            image_value_addr<=(i+m+u)+(j+n+v)*X
                        [(port_seg_addr)s] <=[image_value_addr]
                    end loop i
                end loop j
            end loop m
```

(b)

Figure 3(a) connecon scheme for first method of segmentaon (b) algorithm used to calculate the address of pixel to be read from image_RAM and to be written in one of the one port BRAMs

2.1.3 Second method of segmentation

It can be seen from the middle part of Fig.2, that four columns of two adjacent small windows have the same pixel values. The second adjacent window with respect to the first one has 3 common columns. The 3rd neighboring window with respect to the first one has 2 common columns, and the 4th neighborhood with respect to the first window has one common column. That means, there is no need to re-read all pixels for each window every time when its corresponding vector is to be accessed.

From the bottom part of Fig.2, it can be seen that, each pixel in the 3rd column of the first window acquired from the image_RAM can simultaneously be written to 5 one_port_BRAMs using the next clock cycle. It can also be seen that the relation among adjacent vectors can be used to form a parallel architecture capable for writing operations parallelism. This architecture and its algorithm are depicted in Fig.4(a,b). The SU is responsible for calculating the address of a pixel to be read from the image_RAM. At the same time it calculates where to write this acquired pixel in the one_port_BRAM. In this method of segmentation, the acquired pixel may be written to multiple storage elements (one_port_BRAMs) instead of one of the one_port_BRAMs that is seen in the first method of segmentation. This write operation depends on the signals of port_enable which changes according to the twisted ring counter (see Fig.4) and to the pipelined addressing scheme used to determine each one_port_BRAM address.

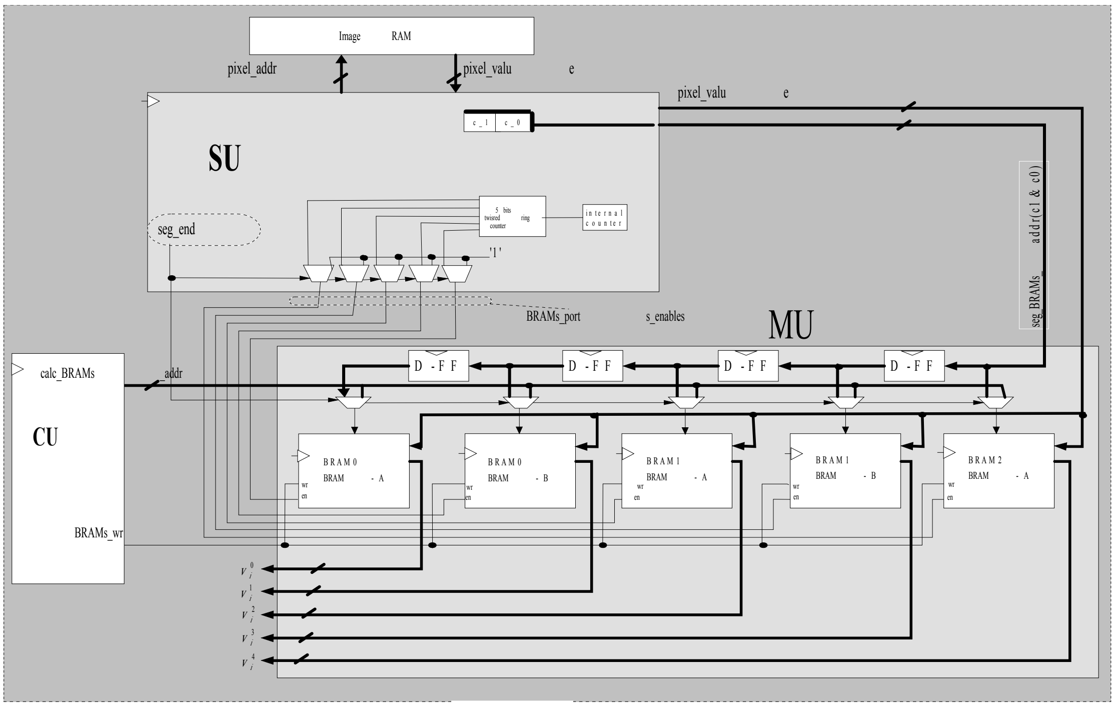
3 Experimental Results

The proposed system is used as a real time face recognizer. The hardware model of the CNN face recognizer has been designed using the Xilinx Foundation environment [8]. The early level of the processing hierarchy requires high-speed data transferences to meet speeding up calculations which performed afterward. Two methods of transferring the data from image RAM to multi-ports buffers represented by FPGA BRAM ports were used. Once the data required to calculate one simple cell in the last layer of the CNN transferred from image RAM to BRAM ports buffers, then, a sequence of parallel processing operations begin which project these data across the CNN simple and complex layers until the output value of its final simple cell is calculated. The parallelism reduces in subsequent manner. A comparison between the performances of these two methods are shown in Fig.5.

One can notice that the time required for segmentation in the first method of segmentation is approximately 2.77 times longer than that used time required in the second method of segmentation. In other word, every last layer simple cell output in the second segmentation method requires 1/2.77 of the time required by the first segmentation method (see the signal: one_simp2_end_flag in Fig. 5). In consequence, the time required to complete the calculations of 100 simple cells is highly reduced. This performance was achieved due to the efficient parallel processing scheme used in the second segmentation method to fill the BRAM ports in comparison with the interleaved method used to fill the BRAM ports in the 1st segmentation method (see BRAMx_py_out signals in Fig. 5).

In interleaved method of segmentation (Fig. 6(a)), one can see that all BRAM ports addresses are common due to the changes of seg_count_1 and seg_count_0 (c1 & c0).

The port_en signal is the output of the decoder whose input is connected to seg_count_2(c2) which counts up whenever c1 reaches to its final state.



(a)

```
for v=0 to (V_y*2-2) "step=2'
    for u=0 to (U_x*2-2) "step=2"
        for n=0 to (N_y-1) "step=1"
            for m=0 to (I_x) "step=I_x"
                for j=0 to (J_y-1) "step=1"
                    for i=0 to (t_x-1) "step=1"
                            image_value_addr<=(i+m+u)+(j+n+v)*X
                        [(port_seg_addr)s] <=[image_value_addr]
                    end loop i
                end loop j
            end loop m
```

(b)

Figure 4(a) connection scheme for second method of segmentation (b) algorithm used to calculate the address of pixel to be read from image_RAM and to be written to one_port_BRAMs.

Every image value is written to one of the BRAM ports (see signals (BRAMx_py_out)s in Fig. 6(a)). In contrast, in the second segmentation method, every acquired image value can be written to all BRAM ports in different locations simultaneously when the signal port_en (connected to twisted ring counter) reaches to its maximum count (see Fig. 6(b,c)). An internal counter is used to control the counting of the twisted ring counter. This counter ensures that each partition of the BRAM port is filled with the $I_x * J_y$ small receptive field vector's values. Reaching the maximum counting of this counter requires 45 clock cycles (see also the bottom part of Fig. 2).

The reduction of segmentation time not only achieved from the parallel writing operation to multiple ports in a single clock as was mentioned in the last paragraph, but the number of image RAM access (RA_2=90,000) which calculated by equation(2) when using the second segmentation method is highly reduced in comparison with that calculated by equation(1)(RA_1=250,000) when using the 1st method. Equations (1) and (2) are described as following:

$$RA_1=I_x.J_y.M_x.N_y.U_x.V_y \qquad \text{-------}(1)$$

$$RA_2=t_x.J_y.2.N_y.U_x.V_y \qquad \text{-------}(2)$$

Refer to Fig.4(b) and Fig.5(b) for description of equations parameters.

The implementation speed performances for CNN face recognizer system using the both methods of segmentation used to buffer the data from image RAM to FPGA BRAM ports are compared to the speed of software

implementation of the same CNN running on a 2.4GHz Pentium 4, GPP:General Purpose Processor, with 256 Mb RAM.

In Fig.7 and 8, one can see that for the 1st method of segmentation, the overall time required for processing one complete image on Xilinx Spartan-3 200,000-gates Platform FPGA (XC3S200) 50MHz is equal to (6.36)ms, while the same model needs (280) ms when implemented softwarely, resulting a speedup of (44). When using the architecture of the second method of segmentation and with the same other CNN components, the overall time required for processing one complete image is reduced to (3.17)ms resulting a speedup of (88).

The dashed columns shown in Fig. 8 indicate that the speedup can be further increased by a factor of 2.45 and 1.63 for first and second segmentation methods respectively. This is achieved when the designed systems are mapped onto an FPGA model that can operate on the maximum allowable operating frequencies.

4 Conclusions

Based on that criterion of using serial digital signal processing is advisable at the upper levels of the hierarchy and not be so adequate for early processing, the coarse grained parallelism that is executed in the bottom layer decreases gradually. In the last layer, only one receptive field is applied to the processing element units that detect its feature. Therefore, the time required to process an image is highly reduced. To speed up calculations, an efficient interfacing and segmentation unit is used to buffer the image data required for these parallel to sequential calculations from the image RAM to multi-port FPGA RAM Blocks.

The system is tested by using the Optical Recognition Library(ORL) face database for face recognition problem. A speed up of the calculations that achieved for the parallel architecture when using the second method of segmentation is duplicated as compared with the same architecture when using the fist method. Also, it can be seen that despite of that the maximum frequency achieved when using the second segmentation unit is lower than that in the first one, but the speedup is still higher.

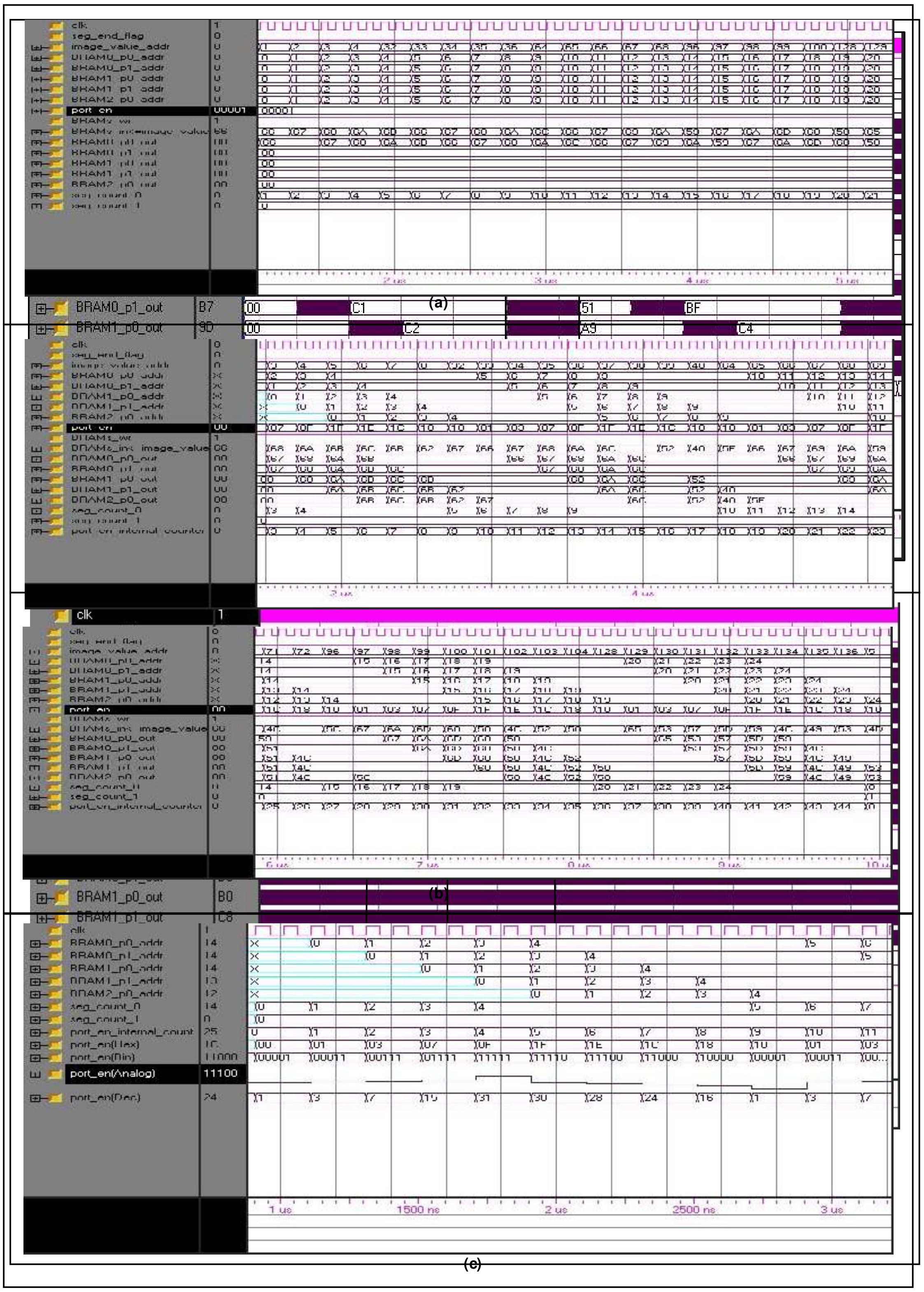


Figure 5: Compressed timing diagram of (a) first method of segmentation

Figure 6: Enlarged timing diagram of (a) first method of segmentation (b) second method of segmentation (c) port_en signals representations

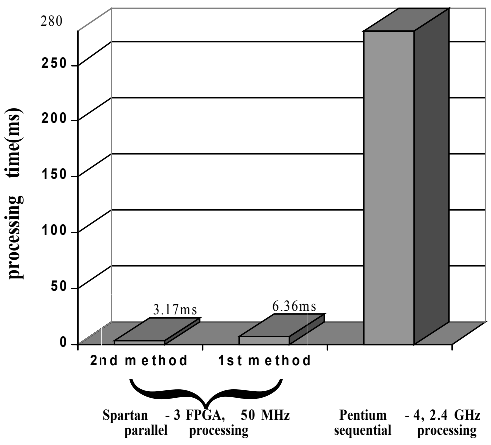


Figure 7: An image processing me d

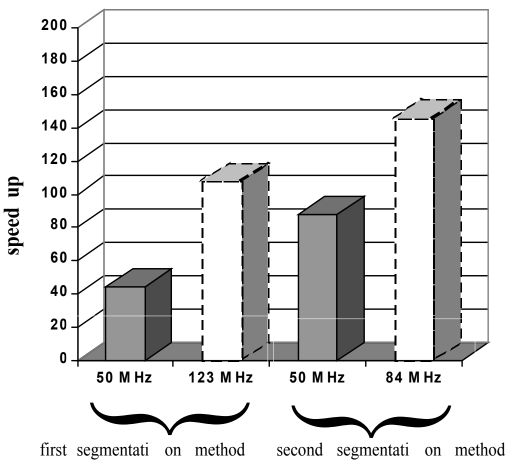


Figure 8: Obtained speed up for two

References

[1] David Landis, Paul Hulina, Scott Deno, Luke Roth, Lee Coraor, "Evaluation of Computing in Memory Architectures for Digital Image Processing Applications," *iccd*, p. 146, 1999 IEEE International Conference on Computer Design (ICCD'99), 1999.

[2] “Local Memory Enhances Performance of Machine Vision Application”, http://www.alacron.com/news/art04.html

[3] Browne, M. & Ghidary, S.," Convolutional Neural Networks for Image Processing: an Application to Robot Vision",http://www.gmd.gr.jp/html/matthew.browne/documents/AJCAI_draft.pdf.

[4]Bruce A.,J. Ross Bevwridge, "Accelerated Image Processing in FPGA". IEEE Transactions on Image Processing 12(12): 1543-1551 (2003)

[5] K. Fukushima and S. Miyake, “Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position”, pattern recognition Vol. 15, No. 6, pp.455-469, 1982.

[6] Angel Rodríguez-Vázquez,Gustavo Liñán-Cembrano, "ACE16k: The Third Generation of Mixed-Signal SIMD-CNN ACE Chips Toward VSoCs",IEEE Transactions on Circuits and Systems—i: regular papers, vol. 51, no. 5, may 2004.

[7] C. Neubauer, “Evaluation of convolutional neural networks for visual recognition”, IEEE transactions on neural networks. Vol. 9, No. 4, pp 685-696, July 1998.

[8]Xilinx, Inc., Datasheet: "*Spartan-3 FPGA Family:Complete Data Sheet*", DS099 March 4, 2004.

[9] Sh. A. Dawwd, B. Sh. Mahmood,” FPGA Design Implementation of a Reconfigurable Architecture for Convolutional Neural Network”,The 6th International Philadelphia Engineering Conference(IPEC 2006),19-21 Sep 2006.

[10] Sh. A. Dawwd,” Software and Hardware design of Image Neurorecognizer and its FPGA Implementation”, Ph.D thesis, Univ. of Mosul, Nov. 2006.

The work was carried out at the college of Engg. University of Mosul