The Parallel FDFM Processor Core Approach for Neural Networks

Yuki Ago, Atsuo Inoue, Koji Nakano, Yasuaki Ito
Department of Information Engineering
School of Engineering, Hiroshima University
Kagamiyama, Higashi-Hiroshima, 739-8527, JAPAN
$\{$*agou, inoue, nakano, yasuaki*$\}$*@cs.hiroshima-u.ac.jp*

***Abstract*—This paper presents a parallel FDFM (Few DSP blocks and Few block RAMs) processor core approach for implementing a perceptron. In our new approach, a perceptron is implemented a processor core using few DSPs and few block RAMs in the FPGA. This approach is promising because we can obtain high throughput using multiple FDFM cores that work in parallel. Also, even if the FPGA does not have enough remaining space for a perceptron, we can implement it using only few DSP slices and few block RAMs. We have implemented 150 processor cores for perceptrons in a Xilinx Virtex-4 family FPGA XC4SX35-10FF668. The implementation results show that 150 processor cores can be implemented in the FPGA using 150 DPS48 slices, 190 block RAMs, and 11679 slices. It runs in 161.546MHz clock frequency and a single evaluation of 96 nodes perceptron can be performed 10.959×10^6 times per second. We have also implemented in the FPGA board of the Virtex-4 Xtreme DSP development kit and confirmed that our 150 processor cores work correctly.**

***Keywords*-Perceptron; Neural Networks; FPGA; DSP48 slice; Block RAM; Pipeline; Parallelize;**

I. Introduction

Artificial Neural Networks (ANN) is a computational model based on biological neural networks. ANNs have been widely used in many fields, such as pattern recognition, signal processing, intelligence control and image processing, etc. Multilayer perceptron (MLP) is a type of ANNs. It is a multilayer feed forward network with supervised learning typically using a so called Back Propagation (BP). MLP has been applied successfully to many complex real-world applications.

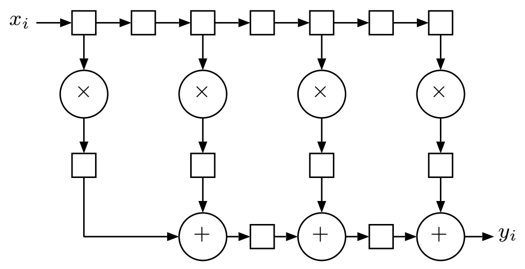
An FPGA is a programmable logic device designed to be configured by the customer or designer by hardware describe language after manufacturing. Since FPGA chip maintains relative lower price and its programmable features, it is widely used in those fields which need to update architecture or functions frequently such as communication and education areas. The most common FPGA architecture consists of an array of logic blocks, I/O pads, block RAMs and routing channels. Furthermore, resent FPGAs have embedded DSP slices that makes a higher performance and a broader applications.

They are widely used in consumer and industrial products for accelerating processor intensive algorithms. For the implementation of Neural Network, FPGA is a crucial hardware platform, which offers high performance and possibility to modify and change algorithms dynamically. Numerous works on FPGA implementation of Neural Networks have been proposed in [1], [2].

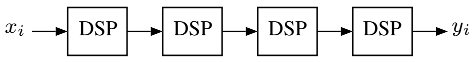
E. M. Ortigosa *et al.* [3] presents several hardware implementations of an MLP for speech recognition using both serial and parallel hardware architecture. In paper [4], the forms of parallelism that can be exploited for neural network implementations on FPGA-based reconfigurable computing environments are described. However, their implementation mainly use logic blocks in the FPGAs.

The main contribution of this paper is to present a new approach that we call *the FDFM (Few DSP slices and Few block RAMs) approach*. The key idea of the FDFM approach is to use few DSP slices and Few block RAMs to perform routine computation. Let us explain the FDFM approach using a simple example. Figure 1 (1) illustrates a hardware algorithm to compute the output of FIR (Finite Impulse Response) $y_i = a_0 \cdot x_i + a_1 \cdot x_{i-1} + a_2 \cdot x_{i-2} + a_3 \cdot x_{i-3}$. A conventional approach implementing the FIR is to use four DSP slices as illustrated in Figure 1 (2)[5]. In this conventional approach the number of DPS blocks must be the same as that of multiplier in the hardware algorithm. On the other hand, our FDFM approach uses one or few DSP slices and one or few block RAMs to implement the FIR. The coefficients $a_0, a_1, \ldots$ are stored in the block RAM.

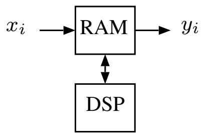
Our FDFM approach has several advantages. First, even if large main circuit occupies the most of hardware resources in the FPGA, we can implement a necessary hardware algorithm in the FPGA using remaining few hardware resources as illustrated in Figure 2 (1). Also, if enough hardware resources are available, we can implement multiple FDFM processor cores that work in parallel(Figure 2 (2)). The resulting hardware implementation has maximum throughput by parallel computation. We can use the FPGA effectively by implementing FDFM cores in all the remaining hardware resources in the FPGA to obtain maximum performance. Further, a conventional approach illustrated in Figure 1 (1) needs more circuit elements. Hence the working clock frequency may decrease due to the propagation delay and the clock skew. On the other hand, since the FDFM approach uses less hardware resources, the circuit may work in higher clock frequency. Actually, hardware algorithms for RSA



(1) FIR



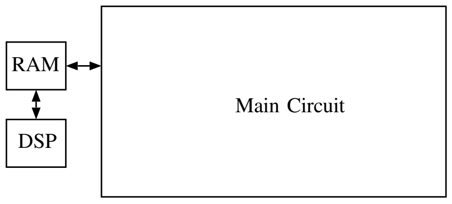
(2) Conventional approach



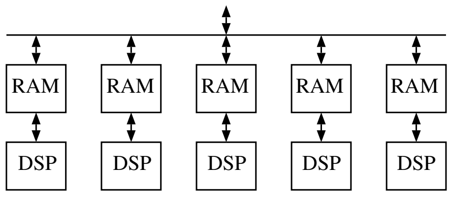
(3) FDFM approach

Figure 1. Our FDFM approach

encryption/decryption have been implemented in the FPGA using the FDFM approach [6], [7]. Their implementation results is better than the conventional approach [8].



(1) Minimum implementation



(2) Parallel implementation

Figure 2. Two advantage of our FDFM approach

In this paper, we propose an implementation of a three-layer perceptron using the FDFM approach. A single core for a three-layer perceptron uses 1 DSP slice and nine 18k-bit block RAMs. The nine 18k-bit block RAMs are used as follows: (1) four for storing the weights (W-RAM), (2) four for a table to compute the sigmoid function (S-RAM), and (3) one for storing the output value of each perceptron node (O-RAM). For a perceptron with 32-32-32 nodes for input-hidden-output layers, our implementation by the FDFM approach runs in 161.546MHz using one DSP48 slice and nine 18k-bit block RAMs in a Xilinx Virtex-4 family FPGA XC4SX35-10FF668. It compute all the outputs in 2124 clock cycles, that is, in 13.148μ seconds. For parallel implementation, we use 30 cores as a cluster of parallel computation. Since 30 cores can share a block RAM for storing the weights (W-RAM) and a table to compute the sigmoid function (S-RAM). So, we use 30 DSP48 slices and 38 18k-bit block RAMs (30 O-RAMs, 4 S-RAMs, and 4 W-RAMs) to implement 30 processor cores. For a perceptron with 32-32-32 nodes for input-hidden-output layers, 30 FDFM cores run in 161.456 MHz. Thus, a single evaluation of the perceptron can be performed 2.192×10^6 times per seconds. For further parallelization, we have implemented 5 clusters, that is, 150 processor cores. The 150 processor cores run in 161.456 MHz, using 150 DPS48 slices, 190 18k-bit block RAMs (150 W-RAMs, 20 S-RAMs, and 20 O-RAMs), and 11679 slices.

This paper is organized as follows. Section II introduces a three-layer perceptron. We show the architecture of a processor core to evaluate the perceptron by the FDFM approach in Section III. Section IV presents a cluster architecture that involves multiple processor cores. Section V evaluate the performance of the processor core and the cluster. We show the experimental results in Section VI Finally, Section VII concludes the paper.

II. Three-layer perceptron

The main purpose of this section is to review a three-layer MLP.

As illustrated in Figure 3, it has three layers: input layer, hidden layer, and output layer. Each layer has a set of nodes. Let N_x, N_h, and N_o denote the numbers of nodes in the input layer, the hidden layer and the output layer, respectively. There are N_x nodes $X_0, X_1, \ldots, X_{N_x-1}$ in the input layer, N_h nodes $H_0, H_1, \ldots, H_{N_h-1}$ in the hidden layer and N_o nodes $O_0, O_1, \ldots, O_{N_o-1}$ in the output layer as illustrated in Figure 3.

A real number x_i in the range of $[0, 1]$ is given to each node X_i in the input layer as an input, they are transferred to all nodes in hidden layers. Some computation is performed in every node H_j of the hidden layer, and it outputs real numbers h_j in the range of $[0, 1]$. These values are transferred to all nodes in the output layers. Similar computation is performed in every node O_k of the output layer, and it outputs real numbers o_k in the range of $[0, 1]$. These real numbers are output of the three-layer MLP. For each pair of nodes X_i and H_j ($0 \leq i \leq N_i - 1$, $0 \leq j \leq N_h - 1$), a fixed real number $v_{i,j}$ is given as a weight. Also, for each pair of nodes H_j and O_k ($0 \leq j \leq N_h - 1$, $0 \leq k \leq N_o - 1$), a

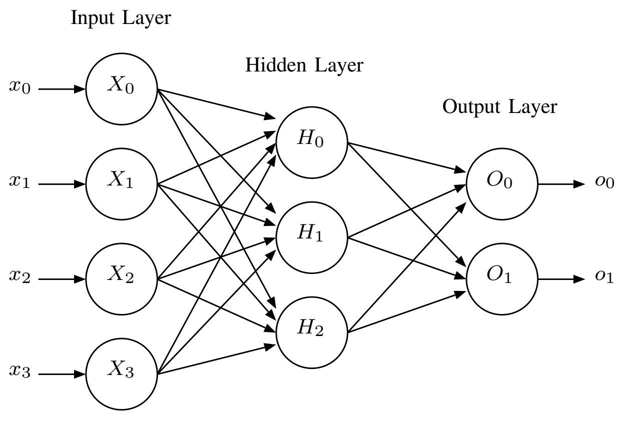


Figure 3. Three-layer MLP

fixed real number $w_{j,k}$ is given as a weight. In addition, for each hidden nodes H_j a fixed real number c_j is given as a threshold value. Similarly, for each output nodes O_k, a fixed real number d_k is given as a threshold value. In a hidden node H_j, the following weighted sum h'_j is computed by:

$$h'_j = c_j + \sum_{i=0}^{N_x-1} v_{i,j} x_i. \tag{1}$$

After that, the sigmoid function $f(x) = 1/(1+e^{-x})$ is applied to obtain the output of each hidden node. More specifically, output h_j of node H_j is computed as follows:

$$h_j = f(h'_j) = f(c_j + \sum_{i=0}^{N_x-1} v_{i,j} x_i). \tag{2}$$

Similar computation is performed for each output node. Let o_k $(0 \le k \le N_o - 1)$ denote the output of node O_k. The value of o_k is computed by the following formulas:

$$o'_k = d_k + \sum_{j=0}^{N_h-1} w_{j,k} h_j, \tag{3}$$

$$o_k = f(o'_k) = f(d_k + \sum_{j=0}^{N_h-1} w_{j,k} h_j). \tag{4}$$

Thus, for input $x_0, x_1, \ldots, x_{N_x-1}$ in $[0,1]$ given to nodes in the input layer, the three-layer MLP outputs $o_0, o_1, \ldots, o_{N_o-1}$ in $[0,1]$ from nodes in the output layer. The resulting output values are controlled by $N_x N_h + N_h N_o + Nh + No$ parameters $v_{i,j}$, $w_{j,k}$, c_j and d_k $(0 \le i \le N_x - 1, 0 \le j \le N_h - 1, 0 \le k \le N_o - 1)$. These parameters are determined in the training phase using back propagation. Intuitively, a pair of inputs and the corresponding correct outputs is given. These parameters are adjusted such that the MLP outputs the correct outputs.

In the training phase, parameters are repeatedly adjusted for a number of pairs of inputs and the corresponding correct outputs. In our work, the training phase is performed on a host PC to determine appropriate parameters. These parameters are stored in block RAMs of the FPGA connected to a host PC.

III. The architecture of a single processor core

This section describes our FDFM approach using a single DSP48 slice and several block RAMs in Xilinx Virtex-4 FPGA [9]. We use Xilinx Virtex-4 Family FPGA XC4SX35-10FF668 as the target device. It consists of columns of slices each of which includes two Configurable Logic Blocks (CLBs), programmable Input/Output Blocks (IOBs), 18k-bit dual-port block RAMs, and DSP48 slices.

A. A single processor core architecture using a DSP48 slice and block RAMs

The DSP48 slice is a configurable block with a multiplier, an adder, and registers. In our work, we use two types of configurations illustrated in Figure 4. In Figure 4, Type 1 employs a 18×18 bit two's complement multiplier multiplier, a 48-bit adder, and a 48-bit register. It also has two 18-bit input A and B, and one 48-bit output P. Basically, it repeatedly computes $A \times B + P \leftarrow P$. Type 2 also employs two additional 18-bit registers. Since ports *BIN* and *BOUT* of adjacent DSP48 are directly connected, one Type 1 DSP48 block and several Type 2 DSP48 blocks can be connected as illustrated in Figure 4.

We have used three types of block memories for the FDFM approach.

W-RAM four 18-kbit block RAMs are used to store the weights $v_{i,j}$ and $w_{j,k}$.

S-RAM four 18-kbit block RAMs are used to store a table to compute the sigmoid function (S-RAM).

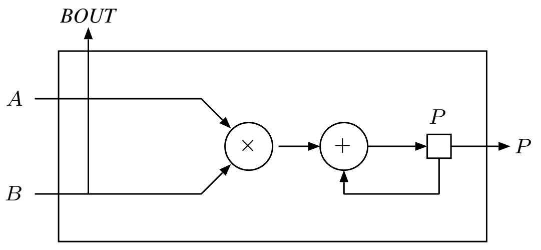
O-RAM one 18-kbit block RAM is used to store the output values, x_i, h_j and o_k of all nodes.

The reader should refer to Figure 5 for illustrating the basic structure of a single processor core for a perceptron. We use Type 1 DSP48, W-RAM, S-RAM, and O-RAM as illustrated in the figure.

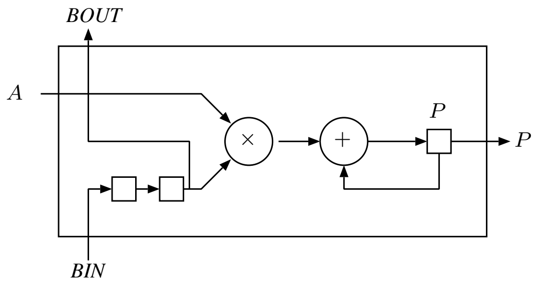
B. Data Representation

The choice of data precision is guided by the implementation cost in terms of area, simplicity of design, speed and power consumption. On the one hand, higher precision will lead to less quantization error in the final implementation. On the other hand, lower precision will produce more compaction and faster designs with less power consumption. A trade-off choice needs to be made depending on the given application and available FPGA resources [3].

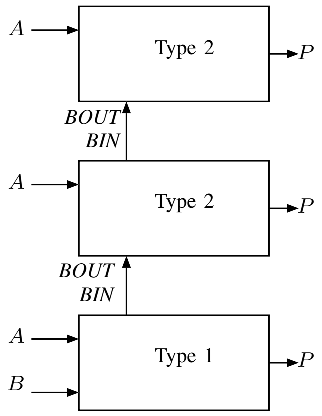
In our work, in order to minimize chip space and computation time, short fixed-point representations of numbers are used. According to paper [1], the minimum required



Type 1: multiplier accumulator



Type 2: multiplier accumulator with two registers



connection of two types of DSP48

Figure 4. Two types of configurations of the DSP48 slice used in our FDFM approach

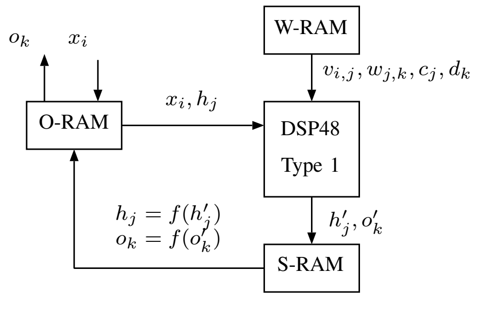


Figure 5. Basic structure of a single processor core

fixed-point precision for weights is 16bits (1bit sign, 3 bits integer and 12 bits fraction). Hence the data format (input, weight, intermediate result and output) is 18-bit fixed point number in our system, which consists of 1-bit sign, 3-bit integer, and 14-bit fraction based on two's complement. The data format is just like *SIII.FFFFFFFFFFFFFF*, where *S* is sign bit, *I* is integer bit and *F* is fraction bit. Thus, the discrete error is at most $\epsilon = 2^{-14} \approx 6.1 \times 10^{-5}$, and the maximum is $0111.11111111111111 = 8 - \epsilon$ and the minimum is $1000.00000000000000 = -8$. Consequently, real numbers in our system are in the rage $[-8, 8 - \epsilon]$ with precision 6.1×10^{-5}. Also, if interim value h'_j or o'_k is out of this range, it is rounded either to the maximum or the minimum. For example, if $h'_j > 8 - \epsilon$ then $h'_j \leftarrow 8 - \epsilon$, and if $h'_j < -8$ then $h'_j \leftarrow -8$.

C. Sigmoid Function Implementation

One of the design challenges with perceptrons based on the FPGA is the activation function. Activation function is a typically nonlinear, monotonically increasing function. The sigmoid function is used widely in the activation function for our perceptron. We use the sigmoid function is $f(x) = 1/(1 + e^{-x})$ as an activation function. Figure 6 shows a graph of the sigmoid function. In our implementation, the values of $f(h'_j)$ and $f(o'_k)$ are computed. We take a look-up-table implementation using a block RAM to compute the sigmoid function. Recall that, h'_j and o'_k take 18-bit fixed point representation. In the look-up-table, the value of $f(x)$ is stored in the address of x. If we implement full 18-bit precision for computing $f(x)$ we need look-up-table of size 18×2^{18}. However, the size of a block RAM is 18×2^{10}. Thus, we use four 18kbit block RAMs and the most significant 12 bits of interim values of h'_j and o'_k as the address of look-up-table. More specifically, 12 bits of form *SIII.FFFFFFFF* is used in h'_j and o'_k. Address x of the block RAM is storing the value of $f(x) = 1/(1 + e^{-x})$ in the 18-bit fixed point format.

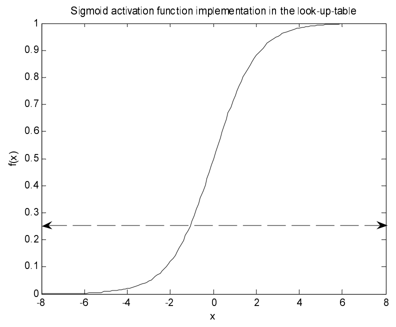


Figure 6. Sigmoid function

D. The behavior of a single processor core

Let us explain how a single processor core illustrated in Figure 5 works.

Step 1 The value $x_0, x_1, \ldots, x_{N_x-1}$ of the input nodes are written into the O-RAM.

Step 2 Each h'_j $(0 \leq j \leq N_h)$ is computed, in turn, using theDSP48 slice. Necessary values x_i, $v_{i,j}$ and c_j $(0 \leq i \leq N_x - 1, 0 \leq j \leq N_h)$ are provided from the O-RAM and the W-RAM, respectively. As soon as h'_j is obtained, $h_j = f(h'_j)$ is computed by the S-RAM and the resulting value h_j is written in the O-RAM.

Step 3 Each o'_k $(0 \leq k \leq N_o)$ is computed, in turn, using the multiplier accumulator in the DSP48 slice. Necessary values h_j, $w_{j,k}$ and d_k $(0 \leq j \leq N_h - 1, 0 \leq k \leq N_o)$ are provided from the O-RAM and the W-RAM, respectively. As soon as o'_k is obtained, $o_k = f(o'_k)$ is computed by the S-RAM and the resulting value o_k is written in the O-RAM.

Note that all of the computation is performed by the pipelining technique. For example, as soon as the DSP48 finish to compute h'_j, it starts to compute h'_{j+1}.

IV. A CLUSTER OF MULTIPLE PROCESSOR CORES

We have implemented many processor cores of the FDFM approach that work in parallel. More specifically, we have designed *a cluster* that consists of 30 processor cores.

Before showing the architecture of the cluster of 30 cores, we will observe the behavior of a single processor core. The Type 1 DSP48, the W-RAM, and the O-RAM operate in almost all clock cycles. However, the S-RAM is used only few clock cycles. For example, the computation of h'_j takes N_x clock cycles and then, $h_j = f(h'_j)$ is computed in one clock cycle. the S-RAM is used only the one clock cycle and it is idle for the other clock cycles. So, the multiple processor cores can share the S-RAM to compute the sigmoid function. Also, since each of the multiple processor cores evaluate the same perseptron, we can share the W-RAM that stores the weights of the perceptron.

From this observation, we design a cluster of 30 cores as illustrated in Figure 7. The cluster is designed as follows:

- 30 DSP48 slices (1 Type 1 and 29 Type 2 DSP48 slices) and 30 O-RAMs are used.
- One W-RAM and one S-RAM are shared by the processor cores.
- The weights and threshold values are provided from the W-RAM, to the port B of Type 1 DSP48 slices.
- The product sums h'_j and o'_k are sent to the registers outside of the DSP slices. They are transferred to the S-RAM in the pipeline technique.
- The resulting values h_j and o_k are stored in the O-RAM.

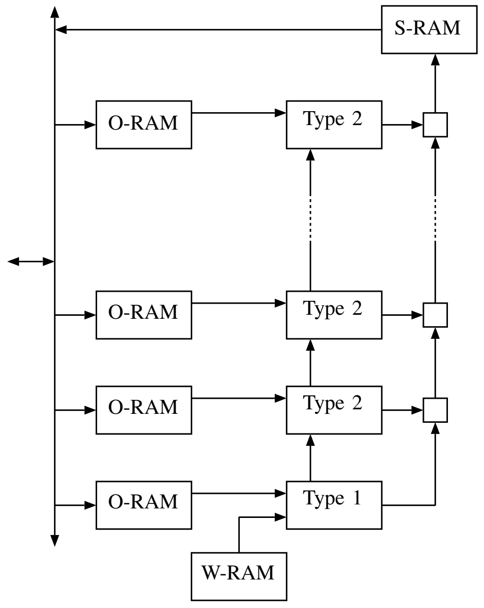


Figure 7. A cluster of 30 processor cores

We can further improve the throughput by using two or more clusters. Figure 8 illustrates the 5-cluster system, which has totally 150 processor cores.

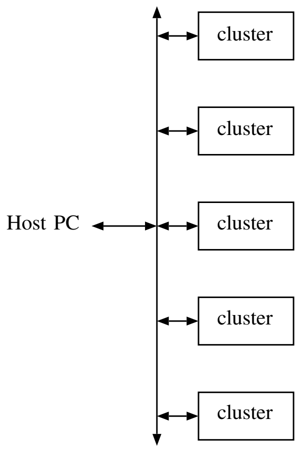


Figure 8. The 5-cluster system

V. PERFORMANCE EVALUATION

Let us evaluate the performance of the architecture by our FDFM approach.

Recall thatN_x, N_h, and N_o denote the numbers of input, hidden, and output nodes of the three-layer perceptron. It takes N_x+1 clock cycles to evaluate the value h'_j by (1); One clock cycle is used to load c_j in the DSP slice and N_x clock cycles to compute the product sum. Similarly, N_h+1 clock cycles are used to evaluate the value of o'_k by (3). Hence it takes $(N_x+1)N_h+(N_h+1)N_o$ clock cycles to evaluate the values of all h'_j's and o'_k's. Including miscellaneous overhead such as evaluation of sigmoid function and data routing, our implementation of a single processor core illustrated in Figure 5 runs in $(N_x+1)N_h+(N_h+1)N_o+12$ clock cycles to obtain the values of all o'_k's. Since the computation of the perceptron involves $N_xN_h+N_hN_o$ multiplications, our implementation using a multiplier in one DSP is very close to optimal.

If we use a cluster with p processor cores, our implementation uses p DSP48 slices, p O-RAM, 1 W-RAM, and 1 S-RAM. Since a O-RAM, a W-RAM, and a S-RAM use 1, 4, and 4 18k-bit block RAMs, the cluster needs $p+8$ block RAMs. It also runs in $(N_x+1)N_h+(N_h+1)N_o+3p+9$ clock cycles. For example, if we have implemented 32-32-32 input-hidden-output-node perceptron in the cluster with 30 processor cores, it runs $(32+1)\times 32+(32+1)\times 32+3\times 30+9=2211$ clock cycles.

VI. Experimental results

We have evaluated the actual performance of our FDFM approach using Xilinx Virtex-4 FPGA XC4SX35-10FF668. Table I summarize the experimental results using ISE Foundation 13.1. The performance is evaluated for 1 processor core, 1 cluster (30 cores), and 5 clusters (150 cores). The numbers of used DSP48 slices and 18-kbit block RAMs, and the clock cycles are equivalent to the results of the evaluation presented in Section V. The throughput means that the number of the evaluation of the 32-32-32 perceptron can be performed per second. For example the 5 clusters can perform the computation of the perceptron 10.959×10^6 times per second.

We have also implemented the cluster in the FPGA board of the Virtex-4 Xtreme DSP kit (Figure 9). The input data $x_0, x_1, \ldots, x_{N_x-1}$ are given to the processor cores though the PCI bus. We have confirmed that the clusters of processor cores work correctly.

VII. Conclusion

This paper introduces the FDFM approach for the FPGA design. We have presented an architecture for a 3-layer perceptron using the FDFM approach and implemented it in the Xilinx Virtex-4 family FPGA. The experimental results show that the performance is close to optimal.

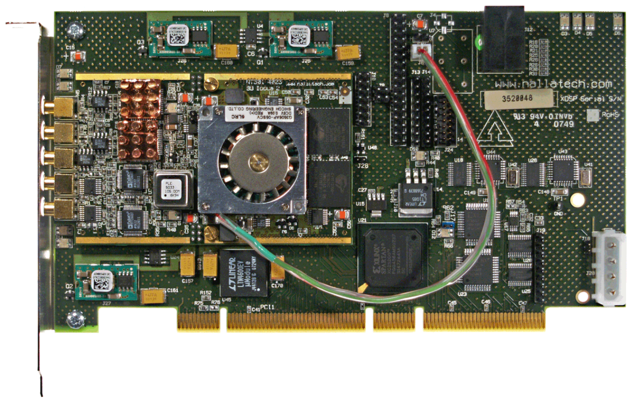


Figure 9. Virtex-4 Xtreme DSP kit

References

[1] A. R. Omondi and J. C. Rajapakse, *FPGA Implementations of Neural Networks*. Springer, 2006.

[2] J. Zhu, G.Milne, and B.Gunther, "Towards an FPGA based reconfigurable computing environment for neural network implementations," in *Proceedings of 9th International Conference on Artificial Neural Networks*, vol. 2, 1999, pp. 661–667.

[3] E. M. Ortigosa, A. C. nas, E. Ros, P. M. Ortigosa, S. Mota, and J. Díaz, "Hardware description of multi-layer perceptrons with different abstraction levels," *Microprocessors and Microsystems*, vol. 30, no. 7, pp. 435–444, 2006.

[4] J. Zhu and P. Sutton, "FPGA implementations of neural networks-a survey of a decade of progress," in *Proceedings of the 13th International Conference on Field-Programmable Logic and Applications (FPL'03)*, September 2003.

[5] *XtremeDSP for Virtex-4 FPGA UG073 (v2.7)*, 2008.

[6] B. Song, K. Kawakami, K. Nakano, and Y. Ito, "An RSA encryption hardware algorithm using a single DSP block and a single block RAM on the FPGA," in *Proc. of International Conference on Networking and Computing*, Nov. 2010, pp. 140–147.

[7] B. Song, K. Nakano, and Y. Ito, "CRT-based decryption using DSP blocks on the Xilinx Virtex-6 FPGA," in *Proc. IEEE International Parallel and Distributed Processing Symposium*, May 2011, pp. 1–8.

[8] K. Nakano, K. Kawakami, and K. Shigemoto, "RSA encryption and decryption using the redundant number system on the FPGA," in *Proc. IEEE International Symposium on Parallel and Distributed Processing*, May 2009, pp. 1–8.

[9] *Virtex-4 User Guide UG070 (v1.5)*, 2006.

Table I
PERFORMANCE EVALUATION OF OUR FDFM APPROACH FOR EVALUATING A 32-32-32 INPUT-HIDDEN-OUTPUT LAYER PERCEPTRON IN XILINX VIRTEX-4 FPGA XC4SX35-10FF668

	1 processor core	1 cluster (30 cores)	5 clusters (150 cores)
# DSP48 slices (out of 192)	1	30	150
# 18k-bit block RAMs (out of 192)	9	38	190
# Slices (out of 15360)	132	2370	11679
Clock frequency (MHz)	161.546	161.546	161.546
Clock cycles	2124	2211	2211
Throughput (10^6/s)	0.0760	2.192	10.959