The International Journal of Parallel, Emergent and Distributed Systems
Vol. 00, No. 00, Month 2011, 1–14

RESEARCH ARTICLE

An FPGA Implementation for Neural Networks with the FDFM Processor Core Approach

Yuki Ago, Yasuaki Ito and Koji Nakano*

Department of Information Engineering, Hiroshima University,
Higashi-Hiroshima, Japan

(*Received 00 Month 200x; in final form 00 Month 200x*)

This paper presents an FPGA implementation of a 3-layer perceptron using the FDFM (Few DSP blocks and Few block RAMs) approach implemented in the Xilinx Virtex-6 family FPGA. In the FDFM approach, multiple processor cores with few DSP slices and few block RAMs are used. We have implemented 150 processor cores for perceptrons in a Xilinx Virtex-6 family FPGA XC6VLX240T-FF1156. The implementation results show that the 150 processor cores for 32-32-32 input-hidden-output layer perceptrons can be implemented in the FPGA using 150 DSP48 slices, 185 block RAMs, and 9676 slices. It runs in 242.89MHz clock frequency and a single evaluation of 150 nodes perceptron can be performed 1.65×10^7 times per second.

Keywords: perceptron; neural networks; FPGA; DSP48 slice; block RAM; pipeline

1. Introduction

Artificial Neural Network (ANN) is a computational model based on biological neural networks. ANNs have been widely used in many fields, such as pattern recognition, signal processing, intelligent control and image processing, etc [1]. 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.

A Field Programmable Gate Array (FPGA) is a programmable logic device designed to be configured by the customer or designer by hardware description language after manufacturing. The most common FPGA architecture consists of an array of logic blocks, I/O pads, block RAMs and routing channels. Furthermore, recent FPGAs have embedded DSP slices that make a higher performance and a broader application.

The Xilinx Virtex-6 series FPGAs have DSP48E1 slices equipped with a multiplier, adders, logic operators, etc [2]. More specifically, as illustrated in Figure 1, the DSP48E1 slice has a two-input multiplier followed by multiplexers and a three-input adder/subtractor/accumulator. The DSP48E1 multiplier can perform multiplication of a 18-bit and a 25-bit two's complement numbers and produces one 48-bit two's complement production. Programmable pipelining of input operands, intermediate products, and accumulator outputs enhances throughput and improves the frequency. The DSP48E1 also has pipeline registers between operators to reduce the delay. The block RAM in the Virtex-6 FPGA is an embedded memory

*Corresponding author. Email: nakano@cs.hiroshima-u.ac.jp

ISSN: 1744-5760 print/ISSN 1744-5779 online
© 2011 Taylor & Francis
DOI: 10.1080/17445760.YYYY.CATSid
http://www.informaworld.com

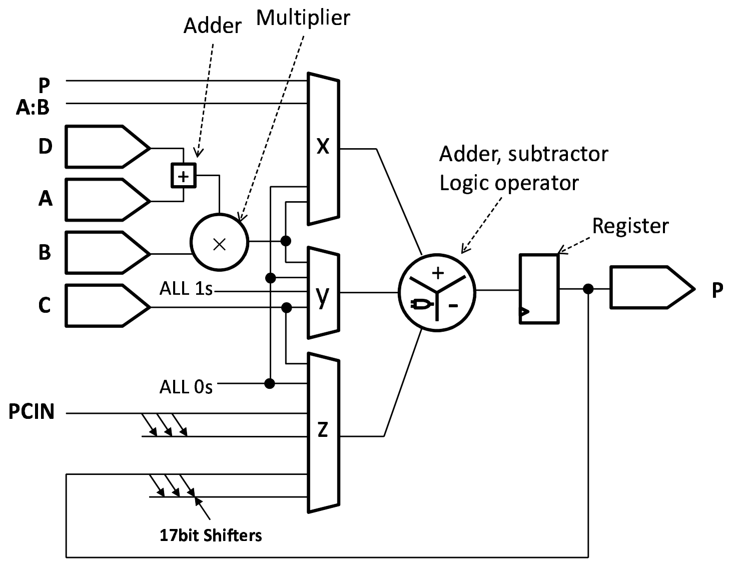

Figure 1. Architecture of DSP48E1

supporting synchronized read and write operations. In Virtex-6 FPGA, it can be configured as a 36k-bit dual-port block RAMs, FIFOs, or two 18k-bit dual-port RAMs. In our architecture, it is used as a 2k×18-bit dual-port RAM.

Since FPGA chips maintain 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. They are widely used in consumer and industrial products for accelerating processor intensive algorithms [2–7]. For the implementation of neural networks, An FPGA is a crucial hardware platform, which offers high performance and possibility to modify and change algorithms dynamically.

There is some research for accelerating the computation of neural networks. Recently, many applications have employed GPUs (Graphics Processing Units) as real platforms to achieve an efficient acceleration. To accelerate the computation of neural networks, several research using GPU support [8, 9]. The above implementations with GPUs achieved speed up factors of several ten times over the sequential implementation with a CPU at the most. However, applications that use neural networks, in general, need to process massive data and require higher throughput computing. Therefore, the ability of the implementations is not enough.

On the other hand, numerous works on FPGA implementation of Neural Networks have been proposed [1, 10, 11]. J. Beuchat *et al.* presented a prototype system of a neural network [12]. The system consists of four FPGAs and several off-chip RAMs. Cox *et al.* introduced implementations of two types of neural networks [13]. Each implementation uses an array of about 30 FPGAs, which are high performance FPGAs at the time, and off-chip RAMs. In [14], an implementation of a three layer perceptron using DSP slices is proposed. The idea of the implementation is similar to our approach for utilization of DSP slices. However, since they implemented a tiny prototype circuit with at most 13 neurons, compared to practical circuit, the size of the MLP is too small and it is not practical. E. M. Ortigosa *et al.* [15] presents several hardware implementations of an MLP for speech recognition using both serial and parallel hardware architecture. In paper [16], 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.

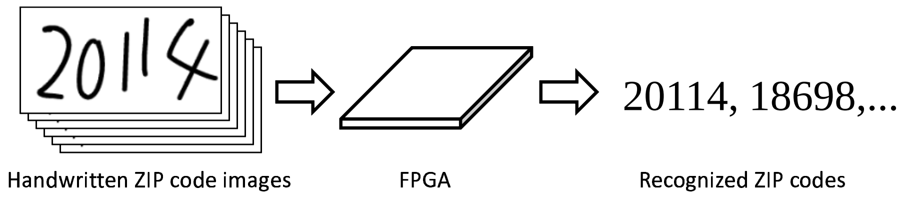

Figure 2. Handwritten ZIP code recognition system

In this paper, we propose an FPGA implementation of a three-layer perceptron using a parallel FDFM approach. In this approach, a perceptron is implemented with a processor core using few DSP slices and few block RAMs in the FPGA. The 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. The detail of the FDFM approach is shown in the following section. A single core for a three-layer perceptron with 32-32-32 nodes for input-hidden-output layers uses 1 DSP slice and eight 18k-bit block RAMs. The eight 18k-bit block RAMs are used as follows: (1) three 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 the perceptron with 32-32-32 nodes for input-hidden-output layers, our implementation by the FDFM approach runs in 242.89MHz using one DSP48 slice and eight 18k-bit block RAMs in a Xilinx Virtex-6 family FPGA 6VLX240T-FF1156. It computes all the outputs in 2124 clock cycles, that is, in 8.74μ 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 37 18k-bit block RAMs (30 O-RAMs, 4 S-RAMs, and 3 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 242.89MHz. Thus, a single evaluation of the perceptron can be performed 3.30×10^6 times per seconds. For further parallelization, we have implemented 5 clusters, that is, 150 processor cores. The 150 processor cores run in 242.89MHz, using 150 DSP48 slices, 185 18k-bit block RAMs (150 O-RAMs, 20 S-RAMs, and 15 W-RAMs), and 9676 slices.

Note that in this paper, we focus on accelerating the computation in feed-forward way. In other words, we deal with the neural network after learning. It is reasonable for the practical applications such as pattern recognition etc. One of the practical applications is a handwritten ZIP code recognition system. Given a large number of handwritten ZIP code images, the system recognizes the ZIP code digits (Figure 2). To deal with many letters, although the processing time is a little longer, the throughput of the system is more important. For such system, our FDFM approach that can provide high throughput process is suitable.

In this paper, we focus on the Xilinx Virtex-6 family FPGA. However, the FDFM approach can be applied to other types of FPGAs if they have embedded circuits whose functions similar to DSPs and block RAMs of the Virtex-6 FPGAs. Recent widely used FPGAs, such as Xilinx other families FPGAs and Altera Stratix series FPGAs, equip them [17–19]. Therefore, we can implement circuits based on the FDFM approach using the above FPGAs.

This paper is organized as follows. Section 2 introduces a three-layer perceptron. Section 3 shows the FDFM approach. We show the architecture of a processor core to evaluate the perceptron by the FDFM approach in Section 4. Section 5 presents a cluster architecture that involves multiple processor cores. Section 6 evaluates

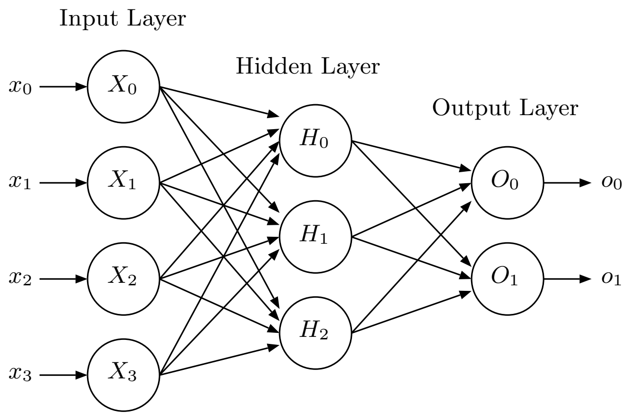

Figure 3. Three-layer MLP

the performance of the processor core and the cluster. We show the experimental results in Section 7. Finally, Section 8 concludes the paper.

2. 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 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. Also, 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:

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

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

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

$$o_k = f(o'_k) = f(d_k + \sum_{j=0}^{N_h-1} w_{j,k}h_j). \tag{2}$$

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 $N_xN_h + N_hN_o + N_h + N_o$ parameters $v_{i,j}$, $w_{j,k}$, c_j and d_k $(0 \leq i \leq N_x - 1,\ 0 \leq j \leq N_h - 1,\ 0 \leq k \leq N_o - 1)$. 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.

3. FDFM Approach

This section presents an 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 4 (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 4 (2)[20]. In this conventional approach the number of DSP 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 two 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 5 (1). Also, if enough hardware resources are available, we can implement multiple FDFM processor cores that work in parallel(Figure 5 (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. Actually, hardware algorithms for RSA encryption/decryption have been implemented

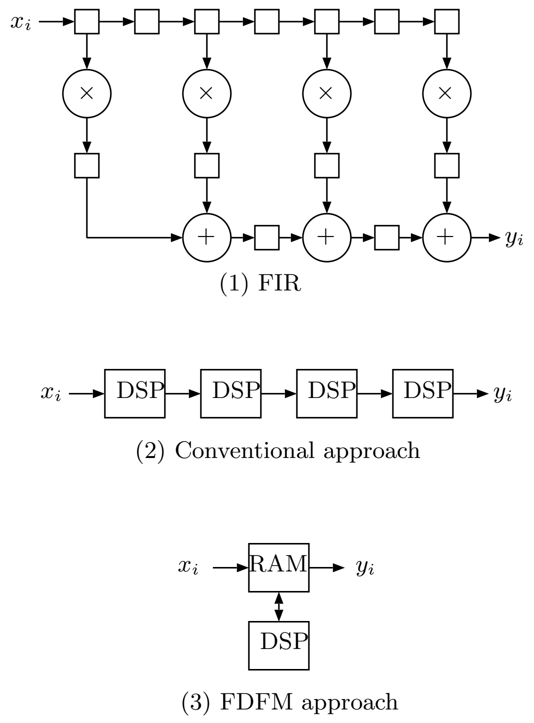

Figure 4. Our FDFM approach

in the FPGA using the FDFM approach [2, 21]. Their implementation results are better than the conventional approach [22].

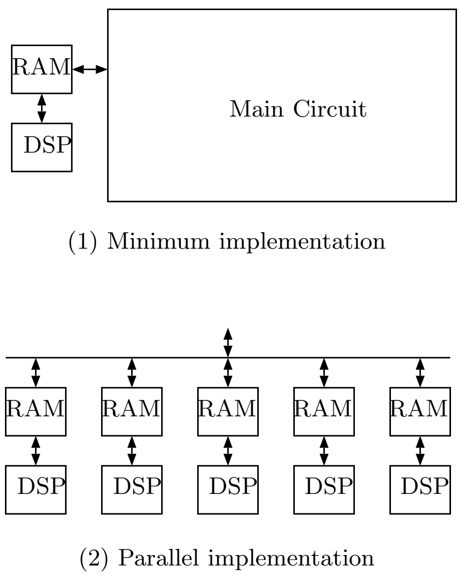

Figure 5. Two advantage of our FDFM approach

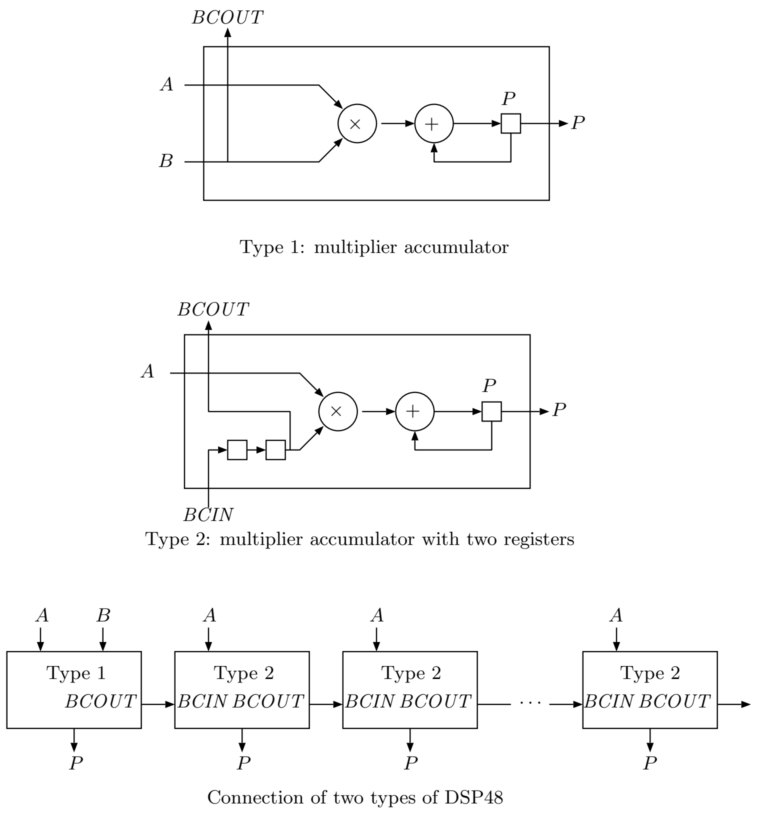
BCOUT
A
B
P
P
Type 1: multiplier accumulator
BCOUT
A
P
P
BCIN
Type 2: multiplier accumulator with two registers
A B
Type 1
BCOUT
A
Type 2
BCIN BCOUT
A
Type 2
BCIN BCOUT
A
Type 2
BCIN BCOUT
P
P
P
P
Connection of two types of DSP48

Figure 6. Two types of configurations of the DSP48 slice used in our FDFM approach

4. The Architecture of a Single Processor Core

This section describes the FDFM approach using a single DSP48 slice and several block RAMs in Xilinx Virtex-6 FPGA. We use Xilinx Virtex-6 family FPGA XC6VLX240T-FF1156 as the target device [23]. It consists of columns of slices each of which includes two Configurable Logic Blocks (CLBs), programmable Input/Output Blocks (IOBs), 36k-bit dual-port block RAMs, and DSP48 slices.

4.1 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 6. In Figure 6, Type 1 employs a 18×18 bit two's complement multiplier, a 48-bit adder, and a 48-bit register. It also has two 18-bit inputs 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 6.

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
An 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 7 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.

4.2 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 [15].

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 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 $h'_j \leftarrow -8$.

4.3 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 8 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.

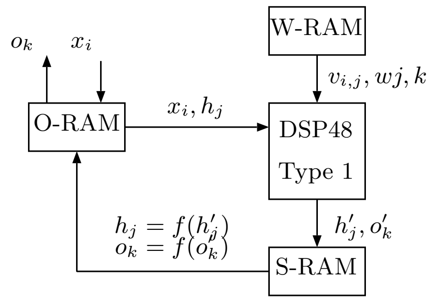

Figure 7. Basic structure of a single processor core

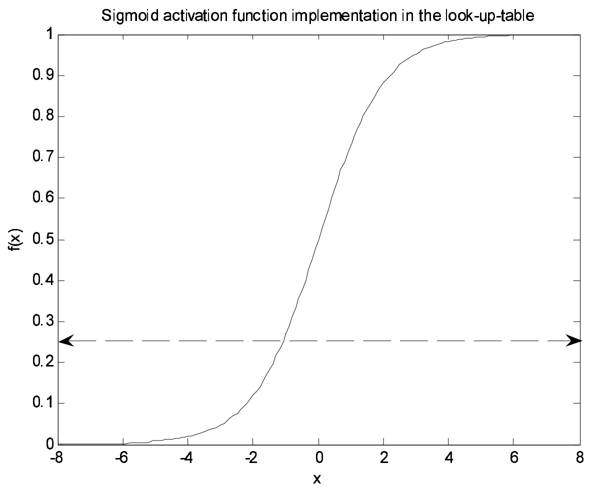

Figure 8. Sigmoid function

4.4 The behavior of a single processor core

Let us explain how a single processor core illustrated in Figure 7 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 the DSP48 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 computing h'_j, it starts to compute h'_{j+1}.

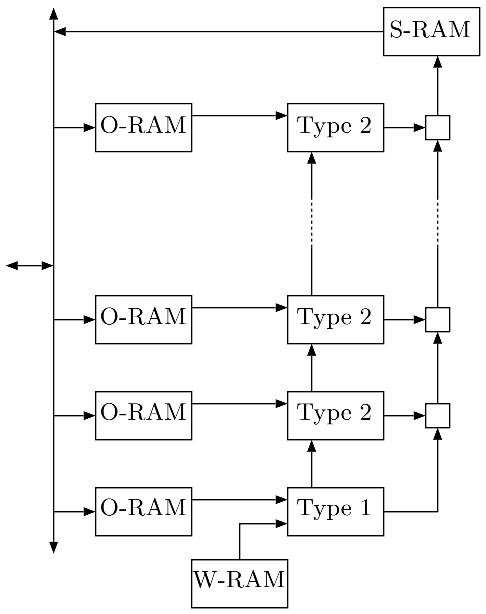

Figure 9. A cluster of 30 processor cores

5. A Cluster of Multiple Processor Cores

We have implemented many processor cores of the FDFM approach that works 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 evaluates the same perception, 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 9. 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.

We can further improve the throughput by using two or more clusters. Figure 10 illustrates the 5-cluster system, which has totally 150 processor cores.

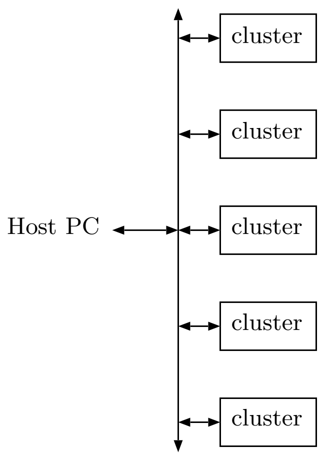

Figure 10. The 5-cluster system

6. Performance Evaluation

Let us evaluate the performance of the architecture by our FDFM approach. Again, let N_x, N_h, and N_o denote the numbers of input, hidden, and output nodes of the three-layer perceptron. It should be clear that the evaluation of the perceptron involves $(N_x+1)N_h+(N_h+1)N_o$ calculations. This is because that a node requires one addition for threshold value and N_x or N_o multiplications in every hidden or output node. This is also the lower bound of the number of clock cycles. In our implementation of a single processor core illustrated in Figure 7 the evaluation takes $(N_x+1)N_h+(N_h+1)N_o+3p+9$ clock cycles. Since at least $(N_x+1)N_h+(N_h+1)N_o$ clock cycles are necessary, this architecture is nearly optimal in terms of the 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.

Since the number of entries of the 18k-bit RAM is 1024, the numbers of 18k-bit RAMs for each of an O-RAM, a W-RAM, and an S-RAM are $\lceil \frac{N_x+N_h+N_o}{1024} \rceil$, $\lceil \frac{N_xN_h+N_hN_o+N_h+N_o}{1024} \rceil$, and 4. If we use a cluster with p processor cores, it uses p DSP48 slices, p O-RAMs, 1 W-RAM, and 1 S-RAM. Therefore, the cluster needs $p\lceil \frac{N_x+N_h+N_o}{1024} \rceil + \lceil \frac{N_xN_h+N_hN_o+N_h+N_o}{1024} \rceil + 4$ 18k-bit block RAMs. For example, if we have implemented 32-32-32 input-hidden-output-node perceptron in the cluster with 30 processor cores, $30\times\lceil \frac{32+32+32}{1024} \rceil + \lceil \frac{32\times 32+32\times 32+32+32}{1024} \rceil + 4 = 30\times 1+3+4 = 37$ 18k-bit RAMs are necessary.

Note that if a given application does not need higher data precision, the precision in our implementation may be too sufficient and can be reduced. For example, the 2-input logical-XOR problem that is a simple problem of neural networks requires 12-bit fixed point precision (1bit sign, 3bits integer and 8bits fraction) [24]. Therefore, some readers may think that the numbers of used DPS slices and block RAMs can be reduced. Reducing the data precision, however, the numbers of used DPS slices and block RAMs cannot be reduced because they are embedded circuits in the FPGA and their construction is fixed. The numbers of them depend only on the number of neurons.

Table 1. Performance evaluation of our FDFM approach for evaluating two types of neural networks

(a) 32-32-32 input-hidden-output layer perceptron

	1 processor core	1 cluster (30 cores)	5 clusters (150 cores)
DSP48 slices (out of 768)	1	30	150
18k-bit block RAMs (out of 832)	8	37	185
Slices (out of 301440)	83	1936	9676
Clock frequency [MHz]	242.89	242.89	242.89
Clock cycles	2124	2211	2211
Throughput [1/s]	1.14×10^5	3.30×10^6	1.65×10^7

(b) 400-300-10 input-hidden-output layer perceptron

	1 processor core	1 cluster (30 cores)	5 clusters (150 cores)
DSP48 slices (out of 768)	1	30	150
18k-bit block RAMs (out of 832)	126	155	775
Slices (out of 301440)	93	1946	9726
Clock frequency [MHz]	207.21	207.21	207.21
Clock cycles	123322	123409	123409
Throughput [1/s]	1.68×10^3	5.03×10^4	2.51×10^5

7. Experimental Results

We have evaluated the performance of our FDFM approach using Xilinx Virtex-6 family FPGA 6VLX240T-FF1156. Table 1(a) summarizes the experimental results for 32-32-32 input-hidden-output layer perceptron 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 36-kbit block RAMs, and the clock cycles are equal to the results of the evaluation presented in Section 6. In the table, 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 1.65×10^7 times per second. According to the table, the clock frequency does not depend on the number of cores.

We have also implemented the circuit for 400-300-10 input-hidden-output-node perceptrons. The experimental results are shown in the Table 1(b). The size of used RAMs is larger than that of the circuit for the 32-32-32 perceptron. However, we can find that the number of slices is little larger and the clock frequency does not decrease largely. Therefore, our method has scalability for the number of neurons. Also, a 400-300-10 input-hidden-output-node perceptrons is used in the recognition system for handwritten digits [25]. The system recognizes a handwritten digit for a binary image of 20×20. It reports that the false recognition rate toward the 60,000 patterns by 250 writers of this system is 1.6 percent. From the table, if the recognition system uses our 5 cluster system that has 150 cores, 251000 handwritten digits can be recognized in one second.

8. Conclusion

This paper presented an FPGA implementation for a 3-layer perceptron using the FDFM approach and implemented it in the Xilinx Virtex-6 family FPGA. In the FDFM approach, multiple processor cores with few DSP slices and few block RAMs are used. The performance evaluation shows that the performance is close to optimal. Experimental results show that our implementation can perform in extremely high speed and throughput.

References

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

[2] Y. Ito, K. Nakano, and S. Bo, *The parallel FDFM processor core approach for CRT-based RSA decryption*, International Journal of Networking and Computing 2 (2012), pp. 56–78.

[3] J.L. Bordim, Y. Ito, and K. Nakano, *Accelerating the CKY parsing using FPGAs*, IEICE Transactions on Information and Systems E86-D (2003), pp. 803–810.

[4] J.L. Bordim, Y. Ito, and K. Nakano, *Instance-specific solutions to accelerate the CKY parsing for large context-free grammars*, International Journal on Foundations of Computer Science (2004), pp. 403–416.

[5] Y. Ito and K. Nakano, *Efficient exhaustive verification of the Collatz conjecture using DSP blocks of Xilinx FPGAs*, International Journal of Networking and Computing 1 (2011), pp. 49–62.

[6] K. Nakano and E. Takamichi, *An image retrieval system using FPGAs*, IEICE Transactions on Information and Systems E86-D (2003), pp. 811–818.

[7] K. Nakano and Y. Yamagishi, *Hardware n choose k counters with applications to the partial exhaustive search*, IEICE Trans. on Information & Systems (2005).

[8] R.D. Prabhu, *GNeuron: Parallel Neural Networks with GPU*, in *Proc. of International Conference on High Performance Computing*, 2007.

[9] D. Strigl, K. Kofler, and S. Podlipnig, *Performance and scalability of GPU-based convolutional neural networks*, in *Proc. of 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, 2010, pp. 317–324.

[10] L.P. Maguire, T.M. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, *Challenges for large-scale implementations of spiking neural networks on FPGAs*, Neurocomputing 71 (2007), pp. 13–29.

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

[12] J. Beuchat, J. Haenni, and E. Sanchez, *Hardware reconfigurable neural networks*, in *Proc. of IPPS/SPDP Workshops*, 1998, pp. 91–98.

[13] C.E. Cox and W.E. Blanz, *Ganglion-a fast field-programmable gate array implementation of a connectionist classifier*, IEEE Journal of Solid-State Circuits 27 (1992), pp. 288–299.

[14] N. Nedjah, R.M. Silvada , L.M. Mourelle, and M.V.C. Silvada , *Dynamic MAC-based architecture of artificial neural networks suitable for hardware implementation on FPGAs*, Neurocomputing 72 (2009), pp. 2171–2179.

[15] 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 30 (2006), pp. 435–444.

[16] 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*, 2003, pp. 1062–1066.

[17] Xilinx Inc., *Virtex-4 FPGA USER GUIDE(V2.6)* (2008).

[18] Xilinx Inc., *Virtex-5 FPGA USER GUIDE(V5.2)* (2009).

[19] Altera Corp., *STRATIX V DEVICE HANDBOOK* (2012).

[20] Xilinx Inc., *VIRTEX-6 FPGA DSP48E1 SLICE USER GUIDE (V1.3)* (2011).

[21] 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.

[22] 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.

[23] Xilinx Inc., *VIRTEX-6 FAMILY OVERVIEW(V2.4)* (2012).

[24] J.L. Holt and J.N. Hwang, *Finite precision error analysis of neural network hardware implementations*, IEEE Transactions on Computers 42 (1993), pp. 281–290.

[25] Y. LeCun, L.D. Jackel, L. Bottou, A. Brunot, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, U.A. Muller, E. Sackinger, P. Simard, and V. Vapnik, *Comparison of learning algorithms for handwritten digit recognition*, International Conference on Artificial Neural Networks (1995).