A Highly Integrable FPGA-Based Runtime-Configurable Multilayer Perceptron

Jan Skodzik, Vlado Altmann, Benjamin Wagner, Peter Danielis, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany, Tel./Fax: +49 381 498-7284 / -1187251
Email: jan.skodzik@uni-rostock.de

***Abstract*—In this paper, a highly integrable Field Programmable Gate Array-based hardware design of multilayer perceptron as a realization of an artificial neural network is presented. Such a hardware solution ensures a deterministic behavior required for any hard real-time compositions. The integration into existing systems is achieved by the application of UPD/IP. Additionally, the presented design is highly flexible due to a parameterizable multilayer perceptron approach. However, most reconfigurations usually require a hard coded reimplementation, resynthesis, and the download of a new bitfile to the target platform, which also requires an additional host PC. Contrary with the presented solution, it is possible to configure the multilayer perceptron's parameters during runtime via a software interface. This approach allows the multilayer perceptron to be adapted to nearly any application. The developed design combines the flexibility of a software solution to generate and comfortably reconfigure the multilayer perceptron as well as the high performance of a hardware solution. As proof of concept, a running prototype has been realized, which shows the design to be highly flexible and with good performance while the hardware resource consumption is kept minimal.**

***Index Terms*—Artificial Neural Networks; Multilayer perceptrons; Reconfigurable logic; Runtime;**

I. Introduction

An artificial neural network (ANN) is a mathematical construct, which emulates natural adaptive learning. A software solution of an ANN is useful on a van Neumann machine as it offers a possibility to investigate new or modified neural network models but falters in performance and does not guarantee real-time behavior per se. By using an Field Programmable Gate Array (FPGA)-based hardware realization, a fast parallel computing of high amount of data within a predefined time is enabled. In addition, a hardware implementation enables hard real-time computations due to its predictable runtime behavior [1]. The high performance and real-time behavior of a hardware solution qualifies ANN to be applied in many application areas, e.g., in the field of automation.

In this work, a multilayer perceptron (MLP) is introduced, which fulfills the following requirements. The hardware solution should be application independent, which makes it usable for several problems during runtime. This can be achieved by high flexibility. High flexibility can be guaranteed by a highly paramterizable design. The neurons (also called perceptrons in this case) transfer function can be freely chosen, which makes it adaptable to different scenarios. Additionally, an easy integration of the MLP into existing infrastructures is important. Most solutions use a dedicated host PC, which configures and interacts with the FPGA containing the MLP. The authors of this paper renounce the use of connections like RS232 or PCI due to their limitations for integration. Therefore, the presented solution bases on UDP/IP. Despite the high flexibility the design must scale in terms of hardware resources due to the realization of an MLP with a high number of neurons required for many applications. One of the application could be the monitoring of automation equipment through image processing by an MLP in real-time. There is a need for a high number of inputs neurons due to a high number of input pixels. Moreover, the MLP must possibly adapt the weights and transfer function to new light conditions (day and night). It is indispensable that this adaptation must be done during runtime. These presented requirements will be solved by the presented solution. The training of the MLP is done directly in software using Matlab. A concept is defined and realized to comfortably interact with the MLP. Additionally, the resulting design is investigated in terms of the size and performance as well as compared with the trend of available hardware resources of different FPGAs over time. Below, the following main contributions are briefly described:

- Brief description of the basic concept is given and required parameters of an ANN are investigated.
- Design of a parameterizable MLP implementation is described.
- Concept for configuration and interaction with the MLP is presented.
- The hardware utilization and performance of the developed MLP are presented.

The remainder of this paper is organized as follows: Section II contains a comparison of the proposed approach with related work. Section III presents the basic concept of the mechanism and system architecture and the corresponding hardware realization. In Section IV, an approach is presented for configuration and interaction with the presented hardware solution. Section V presents the evaluation of the final place and route results for different neural network sizes. The resource utilization and performance of the design is discussed before the paper concludes in Section VI.

II. Related Work

In [2], a scalable system consisting of self-developed Labomat 3 boards containing an FPGA is presented. The system uses multiple board instances, which results in a higher communication and hardware complexity. Contrary, the authors' approach only uses one single common Xilinx Evaluation Board with a single FPGA. Furthermore, [2] bases on 8-bit computations at a frequency of 10 MHz. The authors were able to realize 16 neurons using eight Labomat 3 boards. An additional processor is used to process a software stack required by the design on the FPGA, which introduces important software overhead. A threshold function serves as transfer function.

In [3], the author presents an ANN implemented on an FPGA, which can be configured during runtime. The communication bases on a serial RS232 connection, which lacks performance and bandwidth. An ANN with 10 neurons at a frequency of 40 MHz is realized in this case. It is possible to change the weight factors and the connections by means of a multiplexer. The influence on size due to greater multiplexers is not described. The presented implementation supports 16-bit operations. Most parts of the genetic algorithm training are done in software outside the FPGA. Only the sigmoid function servers as transfer function.

The authors of [4] describe a system called NeuroFPGA, which supports an MLP with several layers not limited to a given number. However, the presented realization in this paper supports a three-layer MLP, which is usually sufficient for the most use cases. Introducing more layers has been shown to be disproportional to the gain in functionality [1]. Several parameters can be adjusted to make the system flexible but only at design time. The NeuroFPGA is realized as PCI card, which requires a host PC. The maximum performance is a synthesis result for an Altera APEX II device, which achieves 360 million products per second for a big network using 48 multipliers of 16 bits each and a 35 MHz clock. The number of equivalent neurons is not given by the authors. The sigmoid function is supported only as transfer function.

Gomberts et. al. [5] developed an MLP realization on an FPGA, which is parameterizable. The backpropagation training has been realized in hardware. The performance is given with 530 million multiply-add operations with 12-bit value width that are necessary in the forward pass per second in the offline mode without training. The supported transfer function is the sigmoid function realized in LUTs. Parameters like the number of neurons in the layer cannot be changed during runtime. The authors of this paper renounce to integrate the training of the MLP, like in [3] and [5], into hardware in order to save hardware resources and to increase the performance. Furthermore, a Matlab-to-FPGA-interface using a C++ software interface is presented, which allows Matlab to train the MLP in software and transmit the weights to the MLP realized in hardware. As software solution, Matlab is much more flexible than a solution implemented in hardware. The training itself is assumed as non-timing critical part. Due to state-of-the-art

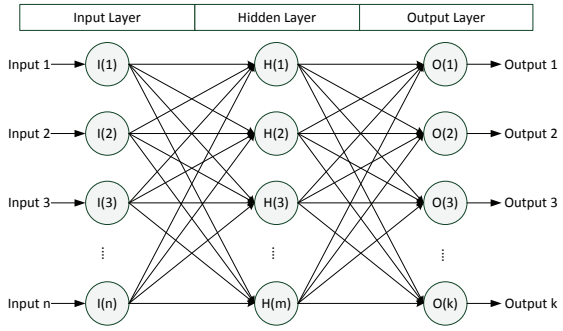

Fig. 1. MLP with three layers

hardware, the presented design works with a three layer MLP with a maximum number of 111 neurons at a frequency of about 33 MHz. The authors of this paper renounce the use of a processor and software on the FPGA, like in [2], and instead directly implemented the design in VHDL and directly support the computations with 32 bit signed integer values, which results i a higher number range and precision. This is also done for the processing of communication frames from a PC over the network, which uses UDP packets to keep the overhead at a minimum. The work in this paper achieves about 483 million multiply-add operations per second using 32 bit signed integer values with a maximum sized 100-9-2 MLP. Furthermore, the presented solution does not require a PCI connection to a host PC like in [4] or serial RS232 like in [3].

It can be simply integrated via Ethernet and UDP into existing networks. Additionally, the presented work enables the user to choose between different transfer functions during runtime to adapt to different circumstances or applications, while others only support one transfer function. The transfer function can be chosen layer independent. However, sometimes an MLP with different transfer functions at different layers offers much better accuracy in contrast to a network with one fixed transfer function [6].

A main focus is on the dynamic configuration via UDP packets in runtime without redesigning the MLP.

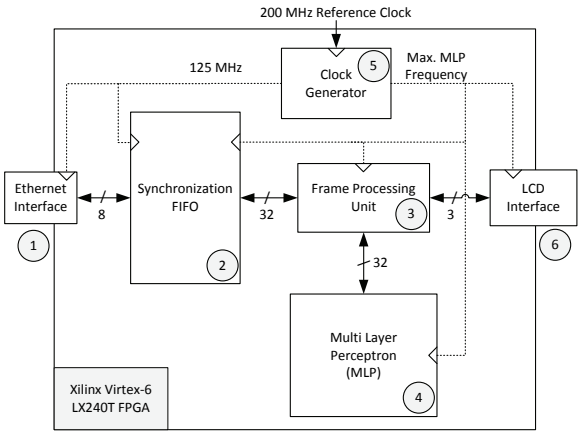

Fig. 2. FPGA system architecture

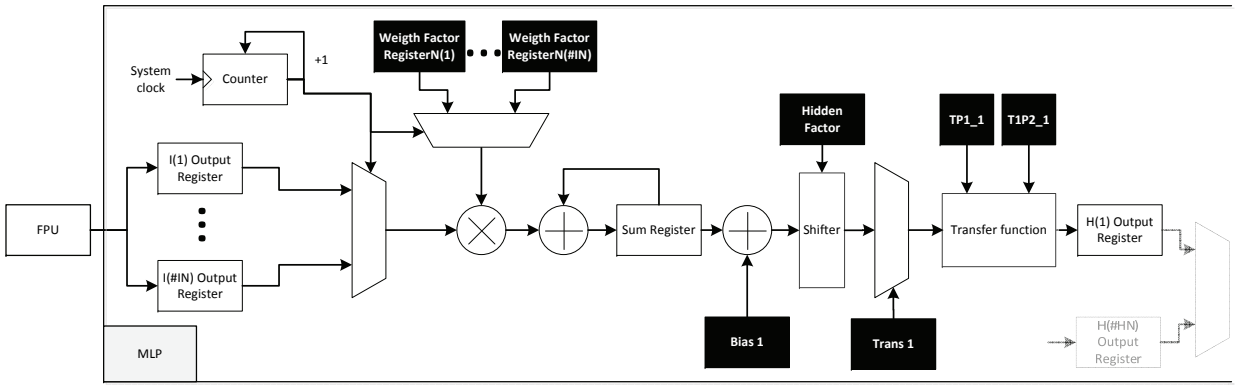

Fig. 3. Calculation of the output of neuron N in the hidden layer

III. Basics and design concept

There are many concepts for designing MLPs, e.g., fully parallelized, serialized, or even a mixed architecture. However, MLPs are well known, established, and suited for a realization in hardware because of their regular structure. A three layer MLP is depicted in Figure 1. An MLP consists of several artificial neurons. These neurons are represented by perceptrons [7]. A three layers MLP is realized and each layer contains a set of neurons. The first layer is the input layer and represents the input values. The hidden layer is the second layer and is fully meshed to the previous input layer. The output layer generates the outputs and receives the values from the hidden layer.

To train the MLP, weight factors of the connections have to be set depending on a given use case. The determination of the weight factors is called training. Each neuron of a layer is connected to each neuron of the following layer.

The MLP is a "feed-forward", fully connected ANN. Therefore, the input value of a neuron is the sum of the n neuron outputs of the previous layer (see Formula 1).

$$Neuron_{Input} = \sum_{i=1}^{n} Input_i * WeigthFactors_i \qquad (1)$$

There are different training algorithms available. For the presented realization, the backpropagation learning algorithm is used to generate weight factors [8]. The training algorithm is not realized in hardware as it is a non-timing critical process and can be implemented in software, which results in less hardware resource consumption and a higher performance of the hardware solution. To realize the mentioned aspects, a system architecture design has been developed as depicted in Figure 2.

A. System Overview

An Ethernet interface (1) ensures the integration of the system into an Ethernet network (1 Gbit/s connection). All UDP packets needed to configure or transmit input data are received by the Ethernet interface. It is necessary to synchronize data received from the Ethernet interface with the rest of the logic because of two different clock domains. In this case, the synchronization FIFO (2) enables a secure clock domain crossing and forwards the data from the 1 GBit/s interface with a data width of 32 bits to the frame processing unit (FPU) (3). The FPU captures all incoming UDP packets and solely processes frames with UDP port 55555. This port may be used as it avoids any conflicts with other standardized protocols [9]. The MLP (4) module contains a three layer MLP as a realization of an ANN. Parameters to configure the MLP at runtime, input values to the MLP from the host, and results from the MLP to the host PC are handled by the FPU. A clock generator (5) generates the two required clocks. One clock is needed by the Ethernet interface for the communication with PC that configures the MLP. The clock is fixed to 125 MHz for 1 GBit/s. The FPU, LCD Interface (6), and MLP are limited to the maximum reachable frequency of the MLP as this module contains the critical path and represents the second clock domain. The LCD interface is connected to an LCD display and provides the user with information about the transmitted frames, e.g., the type of the last transmitted packets.

B. Hardware Realization of the MLP in VHDL

The hardware realization of the calculation from the input layer to the output of the hidden layer is depicted in Figure 3. The calculation of the output layer result is similar to the calculation of the hidden layer result. The summation in the hidden and output layer of each neuron is realized serially. However, the summation consists of a multiplication and addition with the previous calculated value (see Formula 1). Contrary, the summation must be done for all neurons of a layer and is realized as a parallel process. Therefore, all neurons of a layer are executing the summation at the same time. The values in the black boxes in Figure 3 are parameters, which can be set by the user. Several parameters are available to configure the MLP according to a given application. These parameters are listed in Table I. The "#IN" parameter gives the number of neurons, which are used in the input layer. The "#HN" and "#ON" parameter are the corresponding values for the number of the neurons in the hidden and output layer. The output values of the hidden and output layers can be very

large and could produce overflows of the 32 bit signed integer values depending on the use case. Therefore, it is possible to downscale the values with the "Hidden factor" and "Output factor" parameter. This allows the user to scale the values in the network without touching the weights.

PARAMETER	DESCRIPTION
#IN	Number of neurons in the input layer
#HN	Number of neurons in the hidden layer
#ON	Number of neurons in the output layer
Hidden factor	Downscale of hidden layer output
Output factor	Downscale of output layer output
Trans1	Declare hidden layer transfer function
Trans2	Declare output layer transfer function
TP1-1	First parameter for the hidden layer transfer function
TP2-1	Second parameter for the hidden layer transfer function
TP1-2	First parameter for the output layer transfer function
TP2-2	Second parameter for the output layer transfer function
Bias1	Setting the Bias value of input layer
Bias2	Setting the Bias value of hidden layer

TABLE I
MLP CONFIGURATION PARAMETERS

Additionally, the "Trans1" and "Trans2" indicators determines the transfer function of the hidden and output layer. There are three transfer functions, which can be selected by the user. The user can chose between a sigmoid, linear, or Heaviside step transfer function. The transfer function and the corresponding parameters can be set independently for each layer. The parameters "TP1-1", "TP2-1", "TP1-2", and "TP2-2" are given to configure the transfer functions and adapt them to the users constraints.

1) The sigmoid transfer function: Therefore, the sigmoid function itself can be scaled in x and y dimension using the transfer function parameters. The sigmoid function usually is a logarithmic function and hard to realize in hardware without complex arithmetic operations. However, a piecewise linear approximation (PLA) is done to realize the sigmoid transfer function of a neuron with high effectiveness in hardware. The PLA is defined by the Formula 2. All multiplications and divisions inside the sigmoid function are done to power of two so that these operations can be replaced by simple shift operations. Only the multiplication with the input value x have to be realized with a multiplier.

$$\mathrm{T}(x) = \begin{cases} 16 * 2^{TP1} & \text{if } x \geq 64 * 2^{TP2} \\ \frac{x}{16} * \frac{2^{TP2}}{2^{TP1}} + 12 * 2^{TP1} & \text{if } 0 < x < 64 * 2^{TP2} \\ 8 * 2^{TP1} & \text{if } x = 0 \\ \frac{x}{16} * \frac{2^{TP2}}{2^{TP1}} + 4 * 2^{TP1} & \text{if } -64 * 2^{TP2} < x < 0 \\ 0 & \text{if } x \leq -64 * 2^{n} \end{cases} \tag{2}$$

In Figure 4, n is set to 1 and no further scaling of the value ranges is applied for the sigmoid function.

2) The linear transfer function: The second transfer function is a linear function. In Figure 4 TP1 and TP2 are set to the same value, which results in a slope of 1 for the linear function. However, using TP1 and TP2 it is possible to change the slope during runtime. TP1 and TP2 are simple integers values, which indicates the number of shift operations. Therefore, the slope equals a power of two. Formula 3 describes the function. This is necessary because otherwise a real division would have to be realized to determine the final slope, which would decrease performance heavily.

$$T(x) = x * \frac{2^{TP2}}{2^{TP1}} \tag{3}$$

3) The Heaviside step transfer function: In Figure 4, the Heaviside step transfer function is depicted where TP1 is set to 1 and TP2 is set to 0. The TP1 parameter defines the output value of the neuron if it fires. Additionally, the TP2 parameter is used to set the threshold value for the neuron output to send the value TP1. The function is defined by the Formula 4.

$$\mathrm{T}(x) = \begin{cases} 0 & \text{if } x < TP2 \\ TP1 & \text{if } x \geq TP2 \end{cases} \tag{4}$$

The "Hidden factor", and "Output factor" parameter are values to the power of two, which allows the realization as hardware friendly shift operations. Additionally, the MLP comprises an extra bias neuron in the input and hidden layer. The host is able to set the values directly by setting the "Bias1" and "Bias2" parameter. If the Bias value for every node is different, the user only has to modify the weight of the Bias to neuron connection.

IV. CONFIGURATION AND INTERACTION WITH THE MLP

For convenient configuration and interaction, an approach is presented, which meets the requirements for an efficient configuration of the MLP. The weight factors are required to configure the MLP and to define the architecture. An maximum sized MLP is the basic MLP. If less neurons are needed the weight factors of the connections to the unnecessary neuron are set to zero. Consequently, the neuron will always output a zero. Therefore, a previously loaded maximum sized MLP is flexible and the architecture is simple to change. The main flow of the configuration and communication is depicted in Figure 5.

The hardware part of the MLP and additional modules were described in the previous Section III. Matlab, a software interface, and an own developed protocol are used. Two UDP sockets are created handling the outgoing and incoming packets. Port 55555 of the first UDP socket is used to send the packets to the FPGA. The second sockets for receiving packets uses port 55554. If there is a need for a higher PC performance the pcap library can be used to speed up the processing of the incoming packets. Additionally, as the training process is given as a non timing critical process Matlab can perform an online training incidentally. If it necessary to change the MLP parameters Matlab can reconfigure the MLP during runtime.

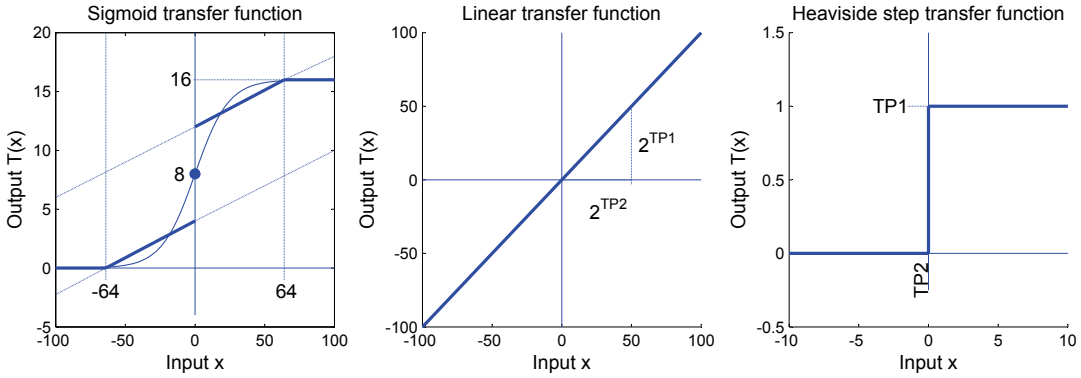

Fig. 4. Supported parameterizable transfer functions

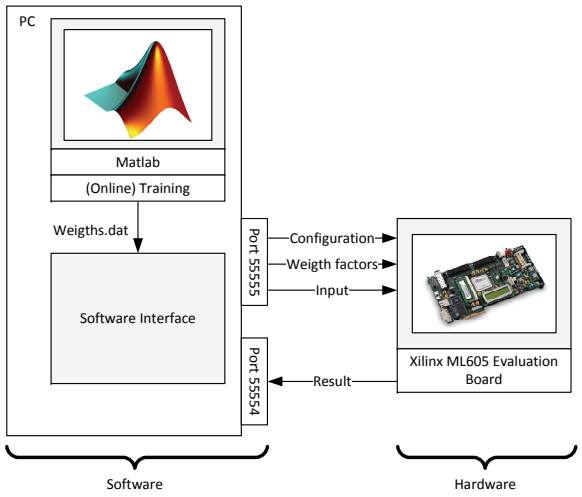

Fig. 5. Developed software hardware approach

A. Creation of the Weight Factors

The MLP needs the correct weight factors to operate properly. These weights factors are generated during a training process like the supervised backpropagation learning. Therefore, Matlab is used to handle the training generating the needed weight factors for the paths between the neurons. The Matlab internal neural network toolbox can be used to generate a trained neural network. Furthermore, using Matlab to train the neural network saves additional hardware resources on the FPGA. The generated weight factors are written into a ”Weights.dat” file. A software interface has been developed using C++. Thus with the software interface, it is possible to set several parameters and to select the ”Weights.dat” file containing the weight factors, which have to be transmitted to the FPGA. All steps during configuration and interaction with the MLP are realized using the software interface.

B. Configuration of the FPGA

The configuration of the FPGA is done directly via UDP packets. A UDP socket on port 55555 sends the configuration packets to the FPGA containing the MLP. This can be performed by any PC in the network. There are different packet types, which are listed in Table II.

PACKET TYPE	ID	DESCRIPTION
”IP_Config”	1	Addressing information
”NN_Config”	2	Neural network parameters
”Weigth”	3	Weights for the inter neuron connections
”Input”	4	Input values for the neural network
”Result”	5	Results from the neural network

TABLE II
DIFFERENT PACKET TYPES OF DEVELOPED PROTOCOL

The ”IP_Config”, ”NN_Config”, ”Weight”, and ”Input” packets are transmitted from the host PC to the FPGA. The packets are processed by the FPU and all relevant information are forwarded to the MLP module.

”IP_Config” packet: For an easy integration of the system into an existing network based on Ethernet and IP it is necessary to be able to configure the addresses. However, this includes the Mac address, IP address and port address, which are sent with the ”IP_Config” packet.

”NN_Config” packet: Inside the ”NN_Config” all parameters are transmitted to configure the MLP. These parameters are defined in Table I. A ”NN_Config” can be send during runtime.

”Weights” packet: The ”Weights” packets contain the weight factors for the MLP. The weight factors are directly extracted from the ”Weights.dat” file generated by the Matlab neural network toolbox. The weights factors are stored in registers for fast access by the the MLP logic.

C. Interaction with the MLP

The interaction between the MLP and the user is done directly using UPD packets. The user can create the ”Input” packets directly using the software interface. The software interface also processes the incoming ”Result” packets.

”Input” and ”Result” packet: The ”Input” packets contain 32 bit signed values in the payload. These values will be directly forwarded to the input layer neurons. The number depends on the given #IN parameter. The ”Result” packet contains the the

results from the MLP, which will be sent back to the requesting host for processing.

V. Performance Evaluation

As there are more and more resources available in each new FPGA generation, it offers more storing and logic capacity. The multipliers and adders consume most of the available hardware resources [10]. However compared to other FPGAs today, it is possible to generate greater scaled ANNs/MLPs containing more multipliers. All divisions needed for a parameterizable design are realized as shift operations by using only factors to the power of two. If integer divisions are supported the design needs to realize full division modules, which results in very low performance. The maximum working frequency will be decreased by a factor of 5. Additionally, the implementation with a full division module needs 45,1% extra LUT hardware resources on average. The number of used registers does not change and only depends on the number of connections in the MLP. Furthermore, the influence of the architecture has been investigated.

Hardware utilization: The number of used LUTs for the exemplary architectures 5-x-2, 25-x-2, and a 100-x-2 MLP are depicted in Figure 6. The x-axis represents the number of neurons in the hidden layer. All three MLPs need more LUTs ressources if the number of neurons in the hidden layer increase. This dependence is linear. The same behavior is when varying the number of neurons in the input and output layer.

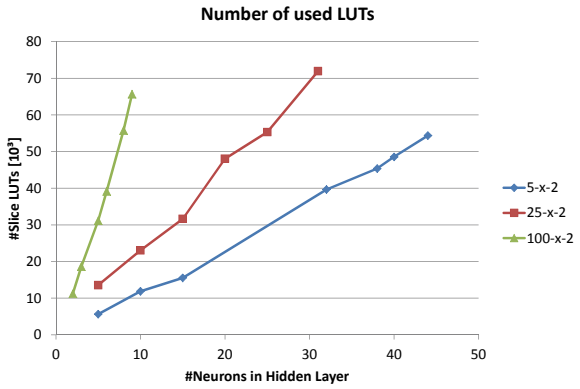

Fig. 6. LUTs hardware utilization

The number of used registers is depicted in Figure 7. For all three architectures, it is also a linear behavior as the number of registers only depends on the number of weight factors for the connections.

DSP Slices were used to speed up the mathematical operations and directly depend on the number of neurons in the hidden and output layer. The number of output neurons is fixed to two for this paper. Therefore, the DSP slice consumption linearly depends on the number of hidden layer neurons. The DSP slice consumption is depicted in Figure 8.

All results are the result of the place and route of the MLP module for a Xilinx ML605 Evaluation board containing a XC6VLX240T-1FFG1156 FPGA. ISE 14.1 was used to

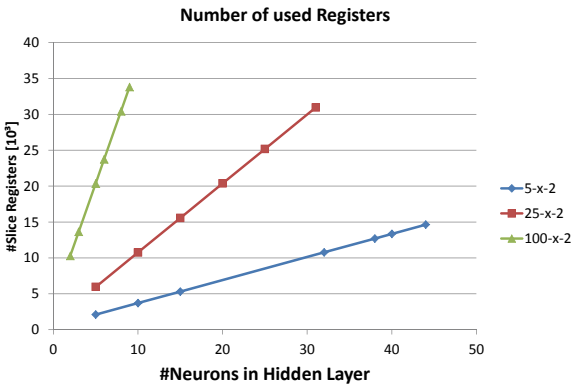

Fig. 7. Register hardware utilization

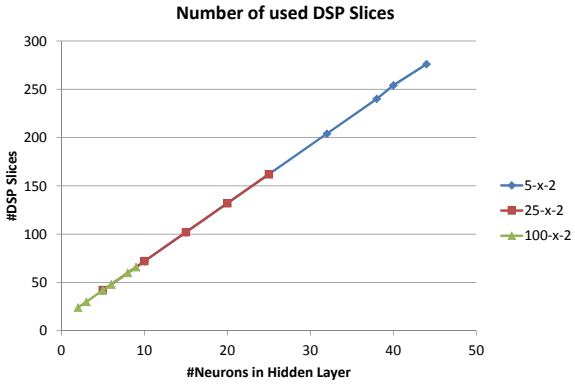

Fig. 8. DSP utilization

generate the design. All other modules were constant in size and additionally consume 1192 Registers and 1094 LUTs. In summary, the whole design is linearly proportional to the number of the neurons and the resulting weight factors and adders. However, it is assumed that the FPGA size is raising according to Moore's law or even more [11]. Furthermore, in Figure 9 the available LUTs and Flip Flops of the model with the highest available resources of each generation is depicted. The behavior is exponential. This positive trend allows to realize greater scaled MLPs without putting further effort in keeping the resources at a minimum.

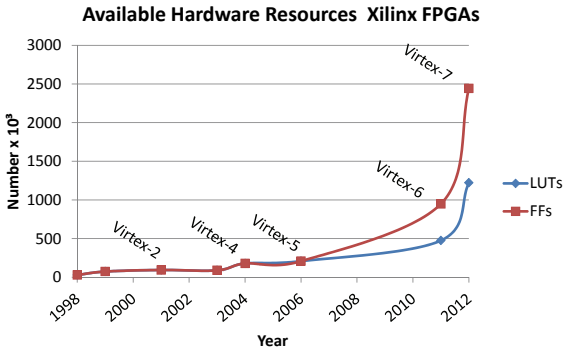

Fig. 9. Xilinx FPGA hardware resource trend [12]

However, that is why it is not necessary to regenerate the MLP every time and reprogram the FPGA if there will be enough hardware resources available in FPGAs.

Latency: The performance itself cannot be driven by the maximum reached frequency for any MLP because the clock generator has to be instantiated once again and a new bitfile must be created to program the FPGA. Therefore, the maximum guaranteed frequency f_{maxMLP} for all MLPs is 33 MHz, which each design is able to reach after place and route. For hard real-time environments, it could be necessary to affect the response time of the MLP (latency). It is possible to have an impact on the number of required clock cycles (CCs) by changing the number of neurons in the layer. The resulting latency is defined by Formula 5 and is directly depending on the maximum reached MLP frequency and the number of CCs needed to read in the input values and compute a result for the output.

$$Latency = f_{maxMLP} * CCs \tag{5}$$

The calculation of the CCs is described in Formula 6.

$$CCs = 2 * \#IN + \#HN + CCs_{Transfer} + \#Bias \tag{6}$$

The #IN counts twice. The input values in an input packet need to be excluded and given to the MLP, which takes one clock cycle per input value. Therefore, each input value needs one clock cycle. Furthermore, the clock cycles for summation in the hidden layer equal #IN because the summation of a neuron (consisting of add and multiplication operations) is done in serial but for all neurons in parallel in the hidden layer. The needed CCs for the computation in the output layer is equal #HN. As the input for an output layer neuron has to be calculated in serial similar to the input calculation of an hidden layer neuron. Additionally, the number of CCs consumed by the sigmoid function has to be added. The transfer function is used twice in the hidden and output layer and needs only one CC. In this case, $CCs_{Transfer}$ is equal two and independent from #IN, #HN and #ON. For each Bias, one extra CC during the summation in the hidden and output layer is needed. These additional CC are described by #Bias.

The latency of the different systems depending on the number of used hidden layer neurons is depicted in Figure 10. Due to Formula 5 the behavior is linear depending on the number of activated neurons.

Another important aspect is the time needed to reconfigure the MLP. The values have been determined for the three maximum sized MLPs achieved and realized by a working bitfile for a Xilinx ML605 Evaluation Board. The time needed to configure the MLP during runtime for a 100-9-2 configuration is only 31.04 us (see Table III). The needed time is rising due to a higher number weights, which have to be transmitted.

All requirements defined in the introduction are met using the presented realization. Furthermore, despite a high level of functionality and flexibility, a low latency and time for configuration are achieved. The complete system is easy to integrate into existing network structures as the configuration is done by UDP. In summary, this work combines the flexibility of a software approach with low hardware costs.

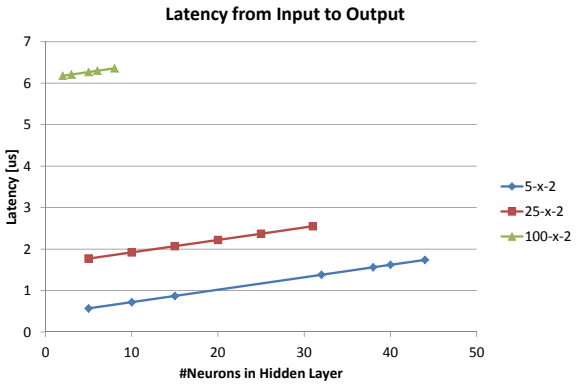

Fig. 10. Latency of different MLP structures

MLP	TIME [US]
5-44-2	11.52
25-31-2	28.45
100-9-2	31.04

TABLE III
TIME NEEDED TO CONFIGURE MAXIMUM SIZED MLP DURING RUNTIME

VI. CONCLUSION

A working prototype of an MLP is presented. It is highly configurable by a developed protocol. Using Matlab and a developed software interface, a user can easily change the architecture and parameters of the MLP during runtime. Due to the Ethernet connectivity and IP accessibility, it is easy integrable into existing systems. An approach is presented showing that the development of FPGA resources follows an exponential trend. Contrary, the hardware utilization of the MLP is linear proportional to the number of neurons. Therefore, it is a suggested trade-off between high usability and resources constraints.

REFERENCES

[1] A. R. Omondi and J. C. Rajapakse, *FPGA Implementations of Neural Networks*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[2] H. Restrepo, R. Hoffmann, A. Perez-Uribe, C. Teuscher, and E. Sanchez, "A networked fpga-based hardware implementation of a neural network application," in *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*, 2000, pp. 337 –338.

[3] D. D. Earl, "Development of an fpga-based hardware evaluation system for use with ga-designed artificial neural networks," Ph.D. dissertation, May 2004.

[4] D. Ferrer, R. Gonzalez, R. Fleitas, J. Acle, and R. Canetti, "Neurofpga-implementing artificial neural networks on programmable logic devices," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 3, feb. 2004, pp. 218 – 223 Vol.3.

[5] A. Gomperts, A. Ukil, and F. Zurfluh, "Development and implementation of parameterized fpga-based general purpose neural networks for online applications," *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 1, pp. 78 –89, feb. 2011.

[6] B. Wagner, G.Ruscher, D. Timmermann, and T.Kirste, "Device-free user localization utilizing artificial neural networks and passive rfid," in *Proceedings of the International Conference on Ubiquitous Positioning, Indoor Navigation and Location-Based Service*, 2012.

[7] F. Rosenblatt, “The Perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, pp. 386–408, 1958.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, October 1986.

[9] I. S. Association, “Ethertype field,” 2012. [Online]. Available: http://standards.ieee.org/develop/regauth/ethertype/eth.txt

[10] J. Liu and D. Liang, “A survey of fpga-based hardware implementation of anns,” in *Neural Networks and Brain, 2005. ICNN B ’05. International Conference on*, vol. 2, oct. 2005, pp. 915 –918.

[11] L. Latif, “Fpga manufacturer claims to beat moore’s law,” *the Inquirer*, 2010. [Online]. Available: http://www.theinquirer.net/inquirer/news/1811460/fpga-manufacturer-claims-beat-moores-law

[12] Xilinx, “Fpgas,” 2012. [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/index.htm