NeuroNoC: A Network on Chip Approach for Implementing Dynamic and Scalable Neural Networks on Reconfigurable Platforms

Authors

***Abstract*—The utilization of FPGA in implementation of Neural Networks (NN) presents better performance in contrast to other hardware realizations and software-based implementations of NN. For real-time applications, FPGAs are well suited and more efficient in terms of time and cost. Large number of neurons in NN has been always a challenge in hardware realization in terms of resource utilization. Additionally, on-chip learning of NN leads to consumption of more resources of FPGA as the learning is performed by many multiplication processes and it requires higher precision. In this work, we present the design and implementation of FPGA-based NN, called NeuroNoc, which utilizes off-chip learning. The proposed NeuroNoC allows the user the novel technique to adjust the number of neurons, interconnections, weighs and biases of each neuron by introducing the configuration packet format of NN that differs from the input signal packet format. The packets, which alters the NN parameters, are sent from the same node as the input signals to NN. The implementation is supported by both on-premise FPGAs as well as cloud-based instances. The NeuroNoC platform is available as an open-source, allowing other researchers and small research centers to transfer the software-based NN to hardware.**

***Index Terms*—FPGA, Artificial Neural Network, Reconfigurability, Configuration Packet**

I. Introduction

The growing demand for automated complex intelligent systems leads to dramatic changes in the development and deployment of Artificial Neural Networks (ANN). The capabilities of ANN to map, model and classify nonlinear systems have allowed to incorporate it in various applications in such fields as science, engineering and economics [n]. The development and implementation of ANN has been done mostly in software. However, despite the benefits of software-based ANN such as high level of abstraction (i.e. no need to know the inner workings of ANN for designers), it has severe problems in real-time applications in terms of execution time in contrast to its hardware-based counterparts [e]. In order to address that issue, there have been proposed several hardware adaptations of ANN [f]. Indeed, even though the software implementation offers flexibility, the high-speed operation in real-time applications is only achievable with the hardware-based networks [g].

There has been proposed numerous hardware architectures of ANN, which can be classified into two groups: analog and digital systems. The latter is more popular, since it provides higher accuracy, noise immunity, better scalability, higher flexibility and compatibility. There are three types of digital implementations of hardware-based ANN: field programmable gate array (FPGA) based, digital signal processor (DSP) based and application specific integrated circuit (ASIC) based implementations. The latter two types of architectures are less suitable for ANN implementation than the first type. DSP based architecture, for instance, is mainly sequential, as a consequence, it lacks to provide parallelism in ANN. ASIC based architectures, on the other hand, does not offer flexibility to reconfigure the ANN by the user [n]. It must be noted that ASIC and DSP can be designed to be highly parallel, however, it is expensive and complicated process to design [e]. FPGA is the best option to implement ANN, since it offers fully parallel and reconfigurable design capabilities.

Reconfigurability of FPGA allows the user to design application specific hardware architecture [g]. FPGA implementations of ANN with a large number of neurons is one of the challenges in hardware-based realizations, since ANN algorithms usually consists of vast amount of multiplication processes and requires higher precision [e]. Hence, on-chip learning is considered to be difficult and useless, as it causes a loss of efficiency in a hardware realization. As a result, off-chip learning is usually chosen when there is no necessity in dynamic learning. Also, the design changes can be made within a few hours, which significantly saves the time and cost of design production. Most of the hardware-based realizations are implemented in such a way that the structure of ANN can be altered through the reprogramming of FPGA [e].

In this paper, we propose the open-source reconfigurable hardware-based realization of ANN, which allows to alter the structure of ANN through the routing of packets in a system. ANN is implemented by the use of mesh topology, which consists of four-way switches with a programming element (PE) connected to it. Each switch with its PE represents a single neuron in ANN. Off-chip learning is chosen in our architecture due to the above-mentioned advantages over on-chip learning. The novelty of the proposed architecture is in the capability of changing the interconnections, weighs and biases on each neuron without explicitly reprogramming the FPGA, but by introducing the ANN configuration packet format, which can be given to the chip the same way as the usual input data packet.

The main contributions of this work are:

- Detailed architecture description of the FPGA-based implementation of ANN

- An open-source implementation of ANN targeting the Xillinx Virtex-7
- Characterization of the proposed platform in the terms of resource utilization and performance.

The rest of the paper is organized as following: Section 2 provides the background, Section 3 discusses the background, Section 4 presents the detailed description of the proposed architecture, Section 5 provides the results f the resource utilization and performance analysis, and Section 6 concludes the paper.

II. Background

ANN is an inspired and adapted model of biological brain. Network consists of many layers, where each layer is constructed form several neurons. The signal transition takes place between neurons through synapse. Each synapse has its own weight, which affects the signal passing though it: the signal is multiplied by the weight of the corresponding synapse. The inputs to the neuron from the synapses are summed up in the neuron and passed further to the activation function. Activation function transforms the input signals to output signals, and its main function is to make decision regarding how the output should behave to certain inputs. Fig. depicts the neuron with the basic operations it performs.

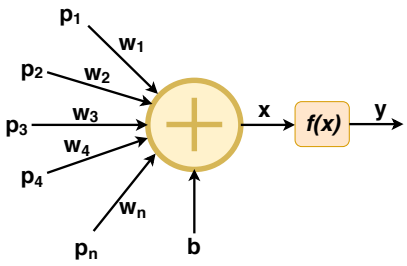

Fig. 1. Caption

ANN can consist of many layers, which are classified into three groups: input layer, hidden layer and output layer. One layer can be fully or partially connected to the next layer. Hidden layer can be built from many intermediate layers between input an output.

Number Representation

One critical factor in hardware implementation of ANN is the number representation of different data - synapse weights, biases and inputs/outputs of the neurons. Despite its widely usage in software ANNs, floating point arithmetic due to its prohibitively expensive cost is not well suited for hardware ANNs. As a result, the two's complement fixed binary point representation was chosen for data representation as it brings considerable optimization in terms of area usage and speed performance. However, this implies limited precision which is enough for many applications. Nevertheless, with this scheme learning process is accomplished off-chip in software using floating point representation. The conversions from decimal fractions to fixed point binary is done through equation 1.

$$b_x(d_x) = \lfloor d_x \cdot \frac{2^{n_x-1}-1}{2^{i_x-1}-2^{-f_x}} \rceil \quad (1)$$

$$d_x(b_x) = b_x \cdot \frac{2^{i_x-1}-2^{-f_x}}{2^{n_x-1}-1} \quad (2)$$

Here, n_x, x, f_x amount to total number of bits, integer bits and fractional bits of the given variable.

III. Proposed Architecture

A. Packet Format

:pktformat.

There are 2 formats of data packets: data packet, and weight, bias, routing table configuration packet. They are presented in figure 2

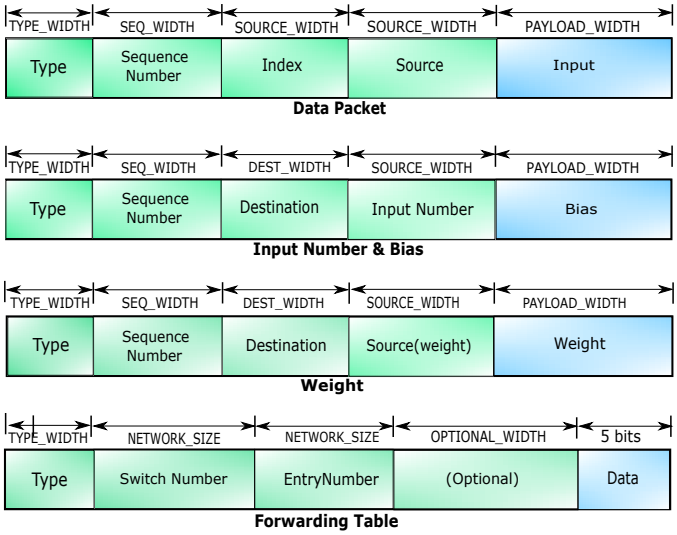

Fig. 2. PKTFORMAT

B. SNC

The SNC module is a constitutive element underlying the PE design. Since the NOC network is not designed to ensure the maintenance of correct packet ordering, an imperative design consideration is to evade out-of-order packet delivery by storing and reassembling the incoming packets. Upon sending input layer data, sending environment assigns each packet a sequence number which is entered into the seqNum field. However, it is worth to note that the number being assigned corresponds to the session number rather than to a single packet i.e a vector comprising n input layer data will be sent in n packets with same sequence number. After processing the input data, the output packet from PE retains sequence number of the input packets. The individual units of input layer data are differentiated using index field, while individual hidden and output layer packets are differentiated using sourceAddress field.

The figure **??** depicts SNC hardware block diagram. The packets arriving from the network interface are first stored in the hash table (HT) memory. The memory consists of

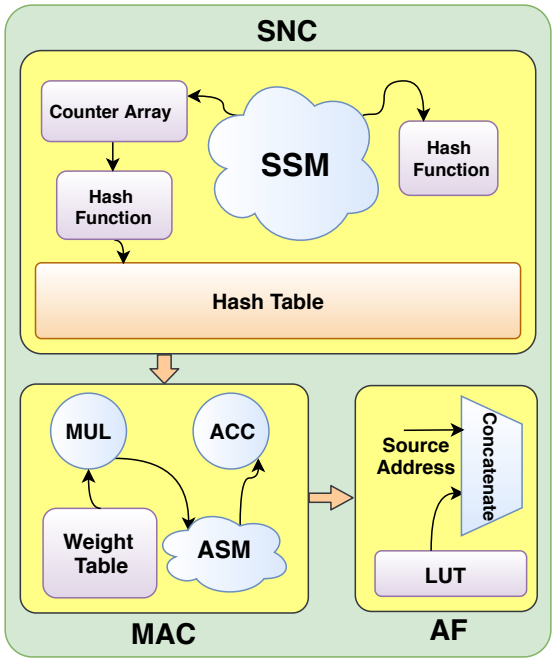

Fig. 3. Caption

2^{SEQ_WIDTH} blocks, where SEQ_WIDTH amounts to the number of bits in the seqNum field. For the sake of full configurability, for the given NOC of size S_{NOC}, each PE can be configured to receive inputs from up to S_{NOC} terminals. Thus, every block is further divided to contain place for S_{NOC} packets. The packets are mapped into memory through hash function, where

$$hf(seqNum, counter[seqNum]) = concatenate(seqNum, counter[seqNum]) \quad (3)$$

Here, the counter[seqNum] embodies the number of packets with the given sequence number that have arrived to PE and is stored in counter array (CR). After the packet is mapped into appropriate position within HT, the counter [seqNum] is incremented, so that the next packets with the same sequence number is stored in the next memory location. The SNC State Machine (SSM) (figure **??**) tracks current sequence number and counter to retrieve packets from the HT. The number of inputs linked to PE is configured using corresponding control packet.

C. MAC

The computation of the total synaptic input to the neuron constitutes the principal arithmetic operation to be implemented in a hardware design of a neural network. This is done by successive multiplication and addition operators i.e. a series of Multiply-Accumulate (MAC) operations. The reassembled packets from SNC are ushered towards Multiplication (MUL) stage. The weights are associated with the individual input packets through their source address, and these weights are stored in a Weight Table (WT) memory block which is shown in figure **??**. The weights can be configured on the run by sending corresponding control packets. The resultant product is accumulated at the Accumulation (ACC) stage. Assuming same format for neuron input and output data, the number of bits to represent weighted sum is

$$N_s = \lceil \log_2(S_{NOC} \cdot (2^{n_w-1})(2^{n_z-1}) + 2^{n_b-1} \cdot 2^{f_z+f_w+f_b}) \rceil + 1 \quad (4)$$

The number of inputs and bias are configured with the same control packets which were used during SNC.

D. AF

Implementation of a highly complicated and non-linear activation function, such as a sigmoid function, exploits look-up table (LUT), which may require large amount of memory area depending upon the desirable precision. If we define n_s as the most significant bits of N_s, increasing the value of n_s contributes to the accuracy of the LUT at the expense of memory size. The minimum value at which all the possible output values are present in the LUT is given by

$$n_s = i_s - \lceil \log_2(\frac{d_z(1)}{f'(0)}) \rceil \quad (5)$$

Furthermore, if we consider sigmoid function, most of the entries located too far away from 0 are replicated. With this in mind, it is possible to reduce the size of LUT to store the central interval $[x_{high}, x_{low}]$, in where expressions for x_{min} x_{max} are given by

$$x_{high} = d_s(\ln 2^{f_z-1}), x_{low} = -x_{high} \quad (6)$$

and the number of bits to address the LUT is $n_{LUT} = \lceil \log_2(x_{high} - x_{low} + 1) \rceil$. If the computed weighted sum falls within the central interval, then the output is taken from the LUT otherwise the output is assigned either 1 or 0.

E. NoC

The figure 4 illustrates the architecture of neural NoC which consists two parts: switches and PE for each of them. The size of NoC can be customized varying X_SIZE and Y_SIZE, subsequently number of switches, PE and size of neural network can be configured. For NoC the mesh topology was used, since this topology is convenient for multicast data transmission. Switches connected to each other with four directions: North,South, East, West. Each switch connected to its PE and both of them has unique number depending on their location in the NoC. Each switch and corresponding PE have a unique number depending on their location in the NoC. The sequence starts from left bottom coordinate and increases from left to right and from bottom to top.

F. Switch

The inner architecture of one switch is depicted in figure 5. Each switch able to send and accept packets from five direction: North, South, East, West and PE. In other words,

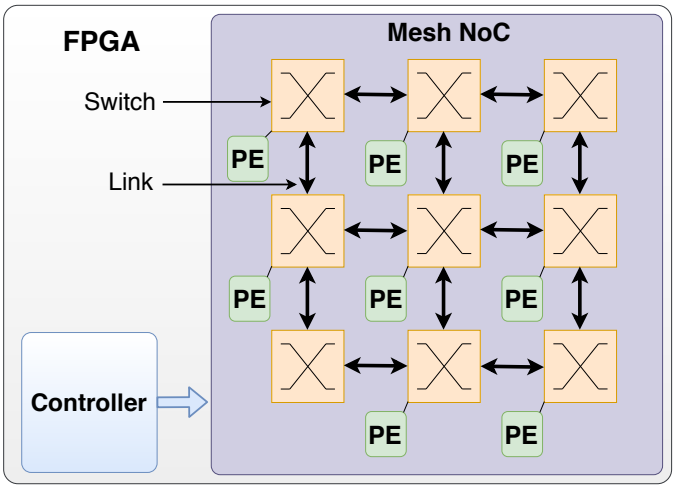

Fig. 4. MESH

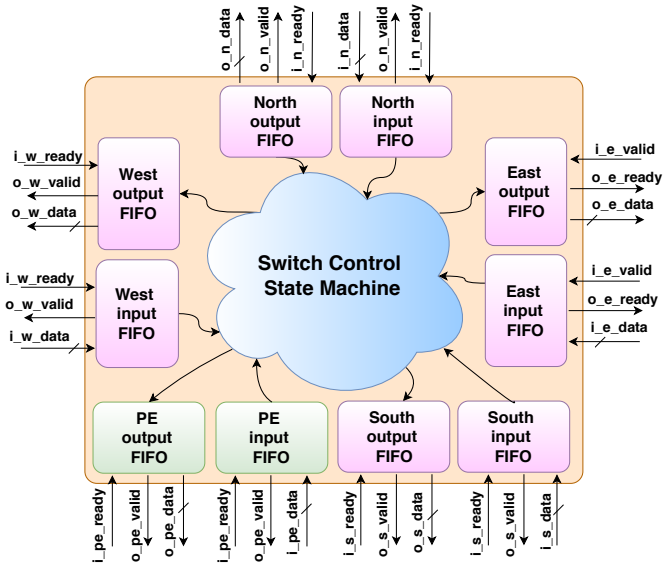

Fig. 5. Switch Architecture

switch can communicate with upper, lower, right, left neighboring switches and with own PE. Each direction has two FIFO for receiving and transmitting the data. The FIFO was implemented by using IP FIFO generator, therefore, the depth of them can be customized. Switches serve the FIFOs in a queue in a clockwise direction. Flow control is implemented through AXI-stream control signal as shown if figure. The o_valid wires are asserted whenever switch transmits the data for certain direction including PE. Similarly, whenever the data comes from other switches or PE the i_valid wire of incoming side of the switch is asserted. The switch asserts the o_ready signal whenever the FIFO has empty slot to accept the data. All FIFOs should hold the data on the bus until data is transmitted to all necessary FIFO. Each switch has registers for routing table, size of total size of NoC rows with 5 bits length. Switch has finite state machine controller, which routes the packet for certain directions depending on the packet type and refreshes the routing table. In one process time one packet can be transmitted to several FIFOs. Switches communicate with each other through FIFOs.

G. Routing Packets

Switch receives the packet from one FIFO and routes the data for required FIFOs. Finite state machine controller has three main approaches for routing the packets depending on the type of incoming packet. For weight and bias configuration packet switch sends the packet using the number of switch in the NoC. Depending on the number of destination switch, the present switch can route the packet for four directions. If the destination switch is located to the above, below, to the left or the right, it sends to North,South, East, West FIFOs respectively. Once the packet came to the predefined switch, instantly it will be sent to the PE FIFO corresponding to that switch for configuring the weight and bias. For routing table configuration packets switches behave the same as for weight and bias configuration oriented switches. The one difference is, when the packet came to the destination switch, it will refresh one slot of routing table and switch will serve next FIFO.

While previous packets can be sent only for one direction(unicast) from one switch, the data packets can be replicated and sent for all direction including PE in one process time (multicast). It was achieved by the logic of routing table. Each switch has routing table, which must be configured before usage. The rows in routing table is equal to the total amount of switches in NoC. The actual meaning of rows in routing table is address of source switch. Whenever data packet comes, switch checks the source provider address of this packet and takes the row, which number is equal to source address. Each row has 5 bits for the decision to sending the packet for certain direction. Each bit corresponds to 5 directions: North, South, East, West and PE. If the bit is equal to 1 then the incoming packet must be sent to corresponding direction,conversely, if bit is equal to 0 then the packet must not be sent to that direction. The routing table configuration directly depends on the neural network model and configuration packets must be created manually or using other software. For reliability of data in multicasted transmission, switch waits until the data is sent for all necessary direction, only after that it checks next FIFO. For implementation of packet routing the finite state machine principle was used and it is shown in figure **??**.

IV. Results and Discussion

V. Conclusion and Future Works

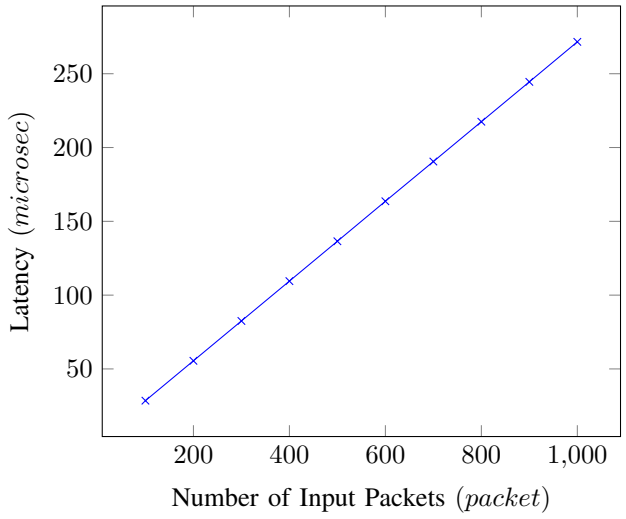

Fig. 6. Relationship between latency and the number of input packets

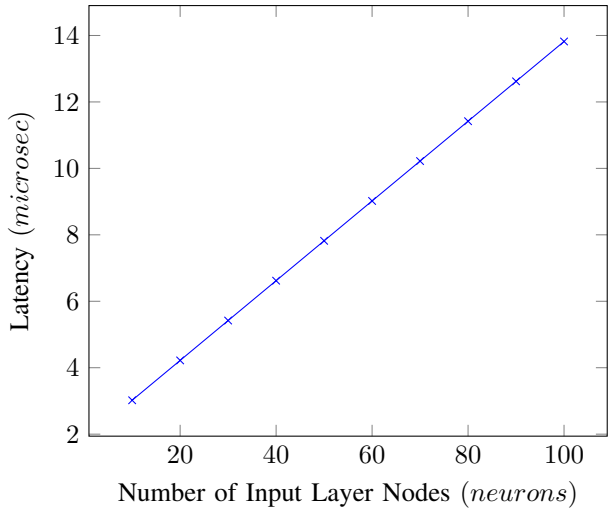

Fig. 7. Relationship between latency and the number of nodes in the input layer