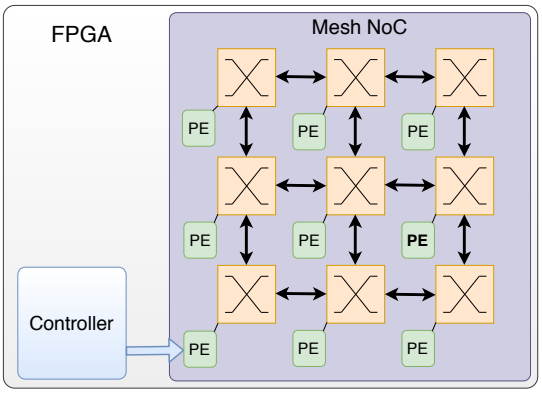

Fig. 1. Architecture of the proposed neuroNoC

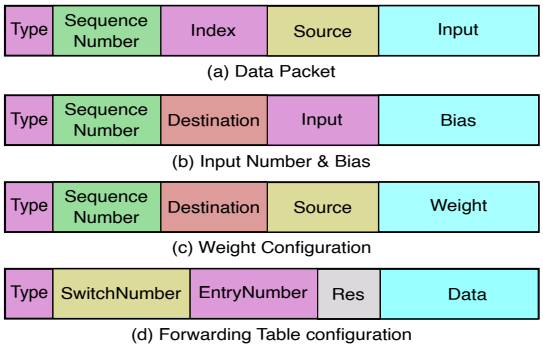

Fig. 2. NeuroNoC packet formats for supporting neuron and network configurations and data communication

I. Proposed Architecture

The overall architecture of NeuroNoC is depicted in Fig. 1. The NoC is arranged in mech topology with each PE representing a neuron, except the one at the bottom-left corner. Mesh topology was selected based on its higher bisection bandwidth and suitability in implementing multicast systems. The size of the NoC is configurable through an HDL parameter and can be set by users based on the type of the application and the resource capacity of the target FPGA device. The bottom-left PE (with zero X and Y coordinates) is used to inject data to the network from external world. In other words this PE represents the entire input layer of the NN. The output layer of the NN sends back the network output to this PE for transmitting it back to the host machine. Due to the packet-switched nature of NoC architecture, the packets may reach the output PE in out-of-order. Packets are reordered and sent back to the host computer based on the sequence number embedded in the data packets.

A. Packet Formats

NeuroNoC implements a variety of packet formats for supporting network and neuron configurations as well as for inter-neuron communication. Fig. 2 depicts the different packet types and their corresponding fields. For a given NeuroNoC implementation, all packet types have same size but depending on the size of the NoC, packet size varies. This is attributed to the variation in the size of the address fields which are NoC size dependent.

Data Packet: Data packets are used for inter-neuron communication. Once a neuron receives data packets from all its predecessors, it calculates the output based on inputs values, weights, bias and the activation function. It sends out another data packet with the calculated output in the *Data* field and the neuron address in the *source* field. The index field has significance only when the packet source is zero (bottom-left PE). Since the entire input layer is modeled by this single PE, the index number differentiates the different neurons of this layer. Otherwise it will be impossible to determine the weight values corresponding each input neuron in the first hidden layer.

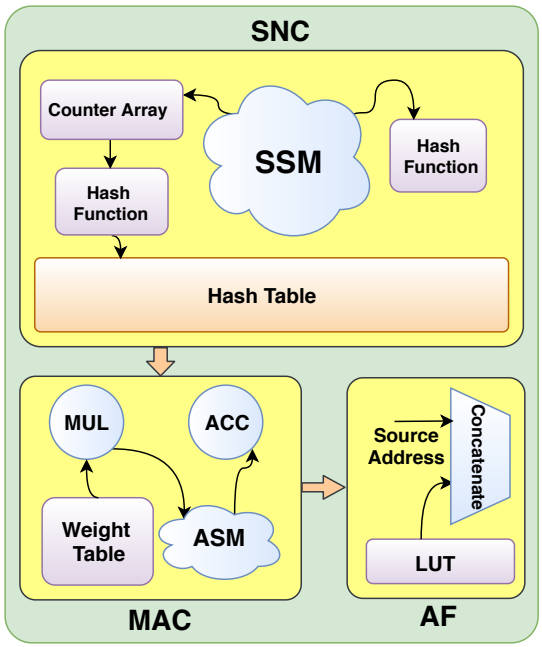

Fig. 3. Architecture of an artificial neuron

Input number and Bias Configuration Packet: This configuration packet is used for setting two parameters in each neuron. The *Input Number* field configures the number of predecessors from the previous layer. The *Bias* field sets the neuron bias for calculating its output. Each neuron stores these two parameters in appropriate registers during network configuration. The *Destination* field is used by the NoC to route the packet to appropriate neurons.

Weight Configuration Packet: The weight configuration packet sets the input weights for each neuron by storing the *Weight* field in its internal Weight Table. The *Source* field enables mapping a a particular weight with a particular input. Infact the Source field is used as the write address when storing weight in the weight table.

Routing Table Configuration Packet: These packets are used for configuring the NoC switch routing tables, enabling multicast routing algorithm described in subsection **??**.

B. Network Initialization

The following steps have to be performed by an external host machine before sending training or test data to the network.

- Initialize the routing tables of the NoC by sending *forwarding table configuration* packets
- Initialize each neuron with its number of predecessors and the bias value through *input number and bias configuration* packets
- Configure the weight tables of the neurons with either pre-trained weight values or random initialization values through the *weight configuration* packets.

Presently NeuroNoC does not support on-chip training of the network. This is mainly because the training algorithms such as gradient descent is not hardware friendly and even the approximate implementation consumes too much of hardware resources leaving very little for the actual network []. Hence either the network has to be pre-trained in a software environment to determine the weight values or a hardware-software co-design approach has to be adopted. In the hardware-software approach, the weight tables are initialized with random weights and training samples are injected to the network by the host machine. The network sends back the output to the host and it runs the back-propagation algorithm to determine the new weight values and updates it through configuration packets. The same procedure is followed in determining the bias values also.

C. Artificial Neuron

Artificial neurons in the NN are implementing by the processing elements (PEs) of the NoC. In the present implementation, we support traditional artificial neurons with linear, sigmoid, binary step and hyperbolic activation functions. In the future versions, more advanced neurons such as LSTM, spiking cells and recurrent cells will be supported. The main advantage of the the proposed architecture is that, the NoC portion of the network can remain intact for inter-neuron communication and the processing portion of the network can be modified based on requirements. Each neuron receives inputs from neurons in the previous layer, multiplies them with corresponding weights, sums up and generates the output based on an activation function. During configuration stage, each neuron is configured with the numbers, weights corresponding each input and bias. Fig. 3 shows the different submodules of an artificial neuron and are described in the subsequest sections.

1) **Sequence Number Checker (SNC):** Since packet switched NoCs do not preserve packet delivery order, neurons support out-of-order packet delivery by storing and reassembling the incoming data packets. The input layer (bottom-left PE) assigns each data packet a sequence number in the *sequence number* field. It is to be noted that the number assigned corresponds to an input number rather than to a single packet. In other words, an input with *n-feature vector space* will be sent as n packets with the same sequence number. For example when the NN is used for detecting hand written characters, the input to the network are grayscale images. Here each input is a grayscale image, which is composed of 784 pixels. Thus each input to the network is a 784-feature vector, where each feature is an intensity value of a pixel. A single input will be injected to the network as 784 packets with the same sequence number. It should be also noted that for this network the number of neurons in the input layer are 784, but in the NoC implementation they are represented as 784 packets with same sequence number. To distinguish each input neuron, the packets also carry a sequence number field as discussed in the previous section. In the absence of this field, the first hidden layer will not be able to determine the corresponding weight values for each input neuron. The index has no significance except for the first hidden layer since every neuron is represented by a physical PE except for the input layer. After processing the input data, the output packet from a neuron retains the same sequence number as that of the input packets.

The neuron initially stores the received data packets in a hash table (HT). The table consists of 2^{SEQ_WIDTH} blocks, where SEQ_WIDTH is the number of bits in the seqNum field. For a given NoC of size S_{NOC}, in the worst case each neuron may receive inputs from S_{NOC} neighboring neurons. Thus, every block in HT is further divided to S_{NOC} memory units. The data packets are mapped to HT memory through the following hash function,

$$hf(seqNum,count[seqNum]) = concat(seqNum,count[seqNum]) \quad (1)$$

Where *count[seqNum]* is the number of packets with the same sequence number already received by the neuron. The sequence number count is tracked by the counter array (CR), which has a dedicated counter for each sequence number. After a packet is stored in the HT, the corresponding sequence number counter is incremented, so that the next packets with the same sequence number is stored in the next memory location. The SNC State Machine (SSM) keeps track of the current sequence number and the corresponding counter to retrieve packets from the HT once the neuron receives packets from all its predecessors.

2) **Multipy-Accumulate Unit (MAC):** The total synaptic input to the neuron is implemented by successive multiplication and addition operations (MAC operations). Once the SNC module detects that the neuron has received input packets from all its predecessors, it sends a control signal to the MAC module. The MAC module then reads the inputs one-by-one from the hash table (HT) and multiplies and accumulates using the corresponding weight values read from the weight table (WT). Weights are stored in WT through the configuration packets described in Section I-A. Assuming the same format for neuron input and output data, the number of bits to represent weighted sum is

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

3) **Activation Function (AF):** The ANN implementation supports a variety of activation functions with the help of look-up-tables (LUTs). By changing the contents of the LUT, the activation function can be modified. LUT-based implementation of a complex and non-linear activation function, such as sigmoid function, may require a large memory depending on the desired precision. If we define n_s as the most significant bits of N_s, increasing the value of n_s improves the accuracy at the expense of memory size. The minimum value at which all the possible output values are present in the LUT is given by

$$n_s = i_s - \lceil \log_2(\frac{d_z(1)}{f'(0)}) \rceil \tag{3}$$

Moreover, if we consider sigmoid function, most of the entries located far from 0 are duplicated. Considering this, it is possible to reduce the size of LUT to store the values in the interval $[x_{high}, x_{low}]$, in where the expressions for x_{min} x_{max} are given by

$$x_{high} = d_s(\ln 2^{f_z - 1}), x_{low} = -x_{high} \tag{4}$$

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 this interval, the output is taken from the LUT otherwise the output is assigned 1 or 0 based on whether the value is above x_{high} or below x_{low}.

The output from the MAC module is used as the address for the activation function LUT. The output from the LUT is then concatenated with the address of the PE as well as the sequence number received from the input packets to generate the neuron output packet. This packet is then injected to the NoC for broadcasting to the successor neurons. Since the NoC switches support multi-cast routing, each PE has to inject a single packet irrespective of the number of destinations. This helps in reducing the latency and improves the overall throughput of the neurons.