Neurocomputing xxx (xxxx) xxx

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Data multiplexed and hardware reused architecture for deep neural network accelerator

Gopal Raut [a], Anton Biasizzo [b], Narendra Dhakad [a], Neha Gupta [a], Gregor Papa [b], Santosh Kumar Vishvakarma [a,*]

[a] Department of Electrical Engineering, Indian Institute of Technology Indore, India
[b] Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia

ARTICLE INFO

Article history:
Received 8 March 2021
Revised 26 August 2021
Accepted 7 November 2021
Available online xxxx
Communicated by Zidong Wang

Keywords:
Activation function
Embedded system design
Hardware reused architecture
Deep neural network
Data multiplexing
Programmable logic
Processing system

ABSTRACT

Despite many decades of research on high-performance Deep Neural Network (DNN) accelerators, their massive computational demand still requires resource-efficient, optimized and parallel architecture for computational acceleration. Contemporary hardware implementations of DNNs face the burden of excess area requirement due to resource-intensive elements such as multipliers and non-linear Activation Functions (AFs). This paper proposes DNN with reused hardware-costly AF by multiplexing data using shift-register. The on-chip quantized log_2 based memory addressing with an optimized technique is used to access input features, weights, and biases. This way the external memory bandwidth requirement is reduced and dynamically adjusted for DNNs. Further, high-throughput and resource-efficient memory elements for sigmoid activation function are extracted using the Taylor series and its order expansion have been tuned for better test accuracy. The performance is validated and compared with previous works for the MNIST dataset. Besides, the digital design of AF is synthesized at *45 nm* technology node and physical parameters are compared with previous works. The proposed hardware reused architecture is verified for neural network `16:16:10:4` using 8-bit dynamic fixed-point arithmetic and implemented on Xilinx Zynq xc7z010clg400 SoC using 100 MHz clock. The implemented architecture uses *25%* less hardware resources and consumes *12%* less power without performance loss, compared to other state-of-the-art implementations, as lower hardware resources and power consumption are especially important for increasingly important edge computing solutions.

1. Introduction

The Deep Neural Networks (DNNs) have become a popular algorithm in pattern recognition since the recent evolution of computer hardware, which fulfilled the high computational power requirements of learning algorithms [1,2]. The main advantage of the DNN over other prediction techniques is its capability to learn hidden relationships in data with unequal variability [3]. Further, DNN is a popular choice due to its diverse applications and is applied to various non-linear detection problems, some of which are lane detection, pattern recognition, fault detection, and monitoring in the industry [4,5]. However, these applications often require a DNN and real-time processing, which need high computational power.

In this article, the authors refer to a fully connected feed-forward Artificial Neural Network (ANN) with multiple layers between the input and output layers as a DNN. Deep learning opens up a need for different platforms like GPU, CPU, ASIC, or FPGA to accelerate the computation of the DNN algorithm. The CPU and GPU-based DNN implementations are general-purpose platforms with specialized hardware supporting various operations, including Multiply-Accumulate (MAC) operation. However, the drawback of CPU and GPU is the low utilization of their resources that reflects in high power consumption [6]. In comparison, ASIC and FPGA devices provide fast multiplication operations with minimal resources utilization and lower power consumption than streaming pixels and learning features [7]. Further, FPGAs also have a specialized hardware structure for MAC operation, but their design is customized to DNN implementation; thus, they achieve high resource utilization and lower power consumption. At the same time, ASIC-based hardware accelerators are the fastest and most energy-efficient. However, they are constrained by their

* Corresponding author.
E-mail address: skvishvakarma@iiti.ac.in (S.K. Vishvakarma).

https://doi.org/10.1016/j.neucom.2021.11.018
0925-2312/

inability to reconfigure neural networks with different features and the inferred network's limited size [8]. Attractive FPGA implementation schemes, focusing on the usage of Xilinx families, are described in the book edited by Ormandi and Rajapakse [9]. Besides, FPGA has hardware efficiency, resources utilization flexibility, and reconfigurable logic design architecture to optimize performance for any specific type of application [10].

In DNNs, each neuron performs two basic functions: sums weighted input features using MAC and evaluates Activation Function (AF) from calculated sum as shown in Fig. 1. DNNs are computationally expensive for data-intensive applications that may contain many neurons and several parameters. In a fully parallel implementation of deep neural networks, each layer needs a distinct memory space for its weight and bias in order to maximize performance efficiency. Previous works have rarely investigated flexible hardware architecture that scales well with the size of a neural network without compromising accuracy. The area-efficient architecture is made using different design techniques for MAC unit, AF, and layer architecture [8,10–13]. Moreover, the implementation of the AF is challenging due to its non-linear nature. Therefore, its accurate design and implementation requires a large number of hardware resources. An area-efficient design can be made by configuring many arithmetic operations implementation such as MAC and AF on the same hardware at the cost of lower throughput [12].

Despite the configurability, the design must scale in terms of hardware due to a profound DNN realization. Our goal is to develop an area-efficient architecture of a DNN that can reuse the AF within the DNN to make it suitable for implementation on small FPGAs. Considering the demands on trade-off in area, power, and speed of the computational block, we propose a data-multiplexed and hardware-reused DNN architecture with an efficient way of memory mapping. Further, compute efficient AF is optimized using Taylor series expansion, and we reuse the AF within each layer by multiplexing and serializing the data-flow using the Parallel-in Serial-out (PISO) mechanism.

1.1. Motivation

The main bottlenecks of the FPGA implementations are limited hardware resources and limited Programmable System (PS)-Programmable Logic (PL) data transmission bandwidth. In order to reduce the memory bandwidth of the DNN implementations, the weights and bias constants should be stored as close to the processing element as possible. Further, neural network MAC featuring a single multiplier in each neuron will access the data serially and compute iteratively. The final weighted sum of neuron's j-inputs is calculated by MAC unit after j^{th} clocks considering

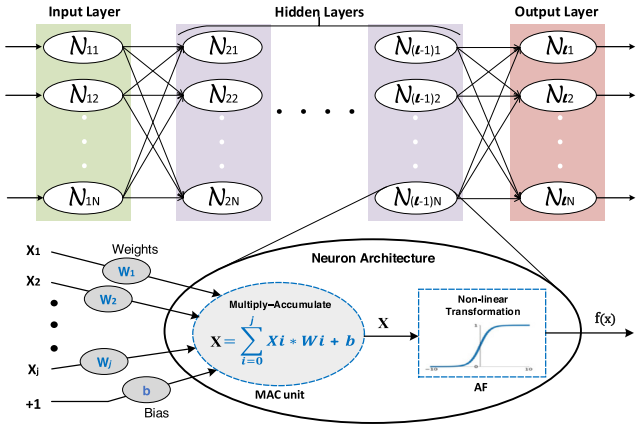

Fig. 1. Fully parallel feed-forward Deep Neural Network architecture.

that j accumulative iterations are required for the final desired output. At the same time, implemented sigmoid AF at each neuron is unused for j clocks and gets active at $(j+1)^{th}$ clock. Furthermore, the parallel AF implementation in each layer takes more hardware resources, and inherently unused hardware of the AF during MAC accumulation for j clocks comes with static power dissipation. We address some of these difficulties by developing and implementing an FPGA-based DNN implementation that reuses the AF in particular and is suited for systems where hardware resources are limited.

Lesser hardware resources and lower power consumption are significant for increasingly important edge computing solutions. In this respect, we have designed an efficient DNN architecture with better resource utilization and other performance parameters. The proposed hardware implementation technique uses multiplexed and serialized data path that allows reusing the AF. We have serialized output of the MAC array within the layer using shift register in each layer and efficiently reused single AF for excitation. The embedded design approach is divided into two parts: firstly, a DNN core is designed for proposed hardware-reused architecture with a control unit for weight/bias access that includes optimized AF. An efficient log_2 quantized scheme is used for serial data access using FIFO. Secondly, an embedded block design is implemented on the Zybo FPGA board to verify the design and compare physical performance parameters. Additionally, to address the ASIC DNN implementation, our AF is synthesized at 45nm technology, and physical performance parameters are compared with the state of the art.

1.2. Contribution

The proposed work focuses on efficient architecture in terms of area and power overhead without performance loss. The contribution of the FPGA based DNN architecture is summarized in the following three points:

- In order to compute a more popular sigmoid AF, we implement its computation through a memory element that uses a positive exponential function with Taylor series expansion and uses BRAM utilization to store the memory element that benefits at higher precision. The AF model physical parameters are also evaluated at the 45 nm technology node.
- In order to find the Pareto point in the Taylor series order expansion for sigmoid AF implementation, accuracy is analyzed. Also, accuracy versus bit precision is analyzed for different AF design architectures.
- The performance-centric, resources-efficient fully connected DNN is presented that reuses AF by multiplexing the data-flow. The throughput and other physical parameters' impact are analyzed. Further, we present efficient memory addressing scheme to read/write of weights and biases.

This paper confirms that DNN architecture with data multiplexing that reuses hardware resources improves area utilization and decreases energy consumption. Furthermore, the authors have verified the proposed architecture by implementing the `16:16:10:4` Deep Neural Network on Zybo xc7z010clg400 and evaluated its performance parameters. The performance validation has been done using the 16-level thermometer to four binary-level classification applications.

1.3. Organization

The rest of this paper is organized as follows. Section 2 gives related works and motivation. Section 3 describes the embedded

system architecture. The proposed design of a hardware-reuse technique for the neural network is presented in detail in Section 4. Section 5 shows the experimental setup, followed by simulation results and discussion in Section 6. Finally, the concluding remarks are outlined in Section 7.

2. Background and related works

In this section, we discuss related works which target hardware implementation for DNNs. DNNs contain one input layer, multiple hidden layers, and one output layer. Conventionally a DNN can either be efficient in terms of area/ power or inference accuracy/ throughput. Therefore, research has been done primarily on computational arithmetic blocks such as MAC, AF, and neuron layer interconnection within the network to optimize neural network acceleration performance.

2.1. Related work for DNNs implementation and MAC computation optimization

The mobile devices with resource-constrained hardware for DNN deployment are attracting much attention. Further, the success of AI drives Application-Specific Integrated Circuit (ASIC) design for the area and performance-efficient DNN implementation [14]. DNNs perform computations for many layers, which come with higher area and power demands [15], and hence the area and power constraint in DNNs represent an increasingly major challenge, especially in mobile devices and AI-enabled IoT applications. Significant research has scrutinized the efficient design architectures at different abstraction layers to enhance the performance of DNNs [16]. An ASIC design as Tensor Processing Unit (TPU) [17] has been implemented for on-chip acceleration. An ASIC/FPGA-based DNN framework such as Eyeriss [18], Gemmini [19], Simba [20] have been proposed in the state-of-the-art. However, the reused hardware resources with no throughput loss have not been addressed. Therefore, explorations of hardware efficiency, such as hardware reuse, have prompted more attention to reduce area utilization [21].

The fully connected neural network with reconfigurable nodes within the layer has been designed in [22]. Furthermore, the configurable layer reused DNN is designed for minimalist hardware architecture and simplifies data movement between memory and compute elements [8]. In [23,24], authors have implemented a binarized neural network which drastically cut down the hardware consumption at the cost of insignificant accuracy loss. Consequently, a significant amount of research is focused on the lower area utilization and power-efficient hardware-based neural networks while compromised with the throughput and accuracy performance.

DNNs with a higher number of layers have been studied in the last decade to improve DNN accuracy. Many architectures have been proposed in the state-of-the-art FPGA-based neural network implementations [25–28]. Such architectures substantially increase the number of neurons that increase the MAC utilization whichdemands more hardware resources, processing time, and power consumption. Further, previous works have investigated the efficient architecture for a MAC unit that can use minimum resources without performance loss [8,11]. In [29], the authors overcome this limitation by using dynamic partial reconfiguration to reuse the FPGA resources. While this technique does improve the resource utilization of the FPGA device, it significantly increases the delay of the circuit. Another way to cope with this increased resource utilization is to use approximate computing techniques to implement MAC [30,31], if some degree of error is tolerable.

The application-specific reconfigurable DNNs have been implemented in [32]. The number of neurons in each hidden layer can be configured within the DNN through which on-chip power has been significantly saved. However, design can further explore hardware reused architecture. An area-efficient method with layer multiplexing has been addressed in which physical implementation of the number of hidden layers is halved [13]. Although, the network implementation has lower area utilization, the design suffers from lower throughput, halved compared to the fully parallel layer architecture. The resource-hungry non-linear AF has been reused by implementing the multiplexers between the parallel MAC units and the AF within the layer [33]. This kind of architecture is efficient for very low precision, and tiny neural networks as multiplexers are hardware costly for higher precision and increase the complexity in the data-flow, which comes with a higher critical delay. However, we have presented an efficient hardware design that reuses the AF with an insignificant throughput performance loss in DNN, and the design technique can be efficient for all fixed-point arithmetic precision implementations. Furthermore, our propsoed architecture is adequate in the hardware design of DNN with runtime configurable for many AFs [34].

The high-throughput MAC unit with a fully parallel array of multipliers is shown in Fig. 2. Usually, the bit-width of MAC output is the sum of input bits, trained weight bits, and extra overhead bits to save the overflow bits generated during the accumulation. The performance-efficient state-of-the-art neuron computational unit having a MAC unit with a parallel multiplier and adder tree followed by AF is shown in Fig. 2. However, it is impossible to instantiate parallel multipliers in MACs for every neuron due to the limited hardware resources. The hardware-costly multiply and accumulate unit can be shared by the multiple consecutive hidden layers at the cost of throughput loss [35,36]. Furthermore, dedicated AF in each neuron is accountable for area overhead due to its underlying non-linear nature. Primarily, MAC unit consists of multiplier and accumulator block, whereas arithmetic relation between input and output of a n^{th} neuron in the l^{th} layer is given by Eq. 1,

$$\mathbf{a}_n^l = f\left(\sum_{j=1}^{J} \mathbf{W}_{n,j}^l \cdot \mathbf{a}_j^{l-1} + \mathbf{b}_\mathbf{n}^\mathbf{l}\right) \tag{1}$$

where f is the Activation Function (e.g. *simoid*, *tanh*, *ReLU*),

$w_{n,j}^l$ is the weight of the j^{th} input a_j^{l-1}, and b_n^l is a bias of the n^{th} neuron. Note that the output of the layer $(l-1)^{\text{th}}$ is the input to the layer l. Computation of a whole layer formulated in matrix form is given in Eq. 2.

$$\mathbf{a}^l = f(\mathbf{W}^l \cdot \mathbf{a}^{l-1} + \mathbf{b}^l) \tag{2}$$

The Deep Neural Networks have numerous layers, and more MACs are a consequence that demand more hardware resources for implementation. This problem will be more dominant when the network processes the high-resolution images that need higher precision computational elements. Therefore, DNN implementation on resource-constrained hardware mobile or edge devices is attracting much attention. In previous works, multi-bit precision (8, 16, 32, and 64-bit) data representation is used for MAC unit arithmetic computation, considering the trade-off between accuracy and physical performance parameters. Furthermore, compared to the floating-point, the fixed-point is preferred which has maximized utilization density and throughput. However, it is also prefered when the resources are limited, and some degree of error is tolerable [37]. Therefore, a neural network design with 8-bit precision operands is preferable [12]. In this proposed efficient architecture of the DNN accelerator, we have used signed 8-bit fixed-point arithmetic precision.

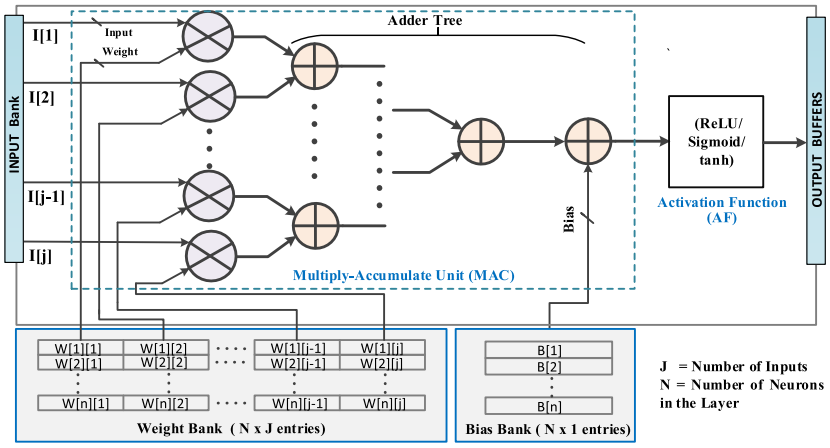

Fig. 2. The memory mapped neuron computational element design having parallel multipliers for *J* inputs, adder tree followed by AF. Considering *N will* be the number of neurons in corresponding l^{th} Layers.

2.2. Related work for techniques used in hardware design and implementation of AF

The state of the arts use different mathematical approaches to implement sigmoid/tanh calculations, as summarized in Table-1. One can notice that all these functions are different forms of the same function but vary in terms of the requirement of arithmetic operations. Further, it can use different on-chip memory resources such as Lookup Tables (LUTs), Block RAM (BRAM), Distributed FPGA memory, or even external DRAM for local storage of AF parameters. Some of the implementation methods are summarized below as

1. LUTs Based Implementation by Storing Function [38]
2. LUTs Based Implementation by Storing Parameters [39]
3. Approximation in Calculation into *base*-2 exploration
4. Coordinate Rotation Digital Computer (CORDIC) Algorithm [12,27,40]
5. Digital/Combinational logic implementation

An exact calculation of the AF using hardware implementation is complex due to its continuous and non-linear nature. Therefore, Piece-Wise Linear (PWL) technique is used for these functions implementation [38,42]. Moving from lower precision to higher precision, such as 8-bit, 12-bit, and 16-bit, the number of quantization states increases exponentially, leading to an exponential rise in required memory elements. The error contribution due to PWL-AF at different precision is expressed in [43]. Moreover, the non-linear AFs such as sigmoid/tanh cannot be approximated efficiently using only combinational logic [12]. However, using purely combinational logic has the benefits of providing low latency with small area overhead compared to conventional ROM-based approaches. In [44], an approximation scheme for tanh AF implementation has been proposed using combinational logic design to explore sigmoid function evaluation further. However, the complexity of circuit design will increase for higher precision AF implementation. The reuse of hardware resources with an improved architecture for configurable AF implementation is investigated in [45]. This technique achieves high utilization of the FPGA and remarkably improves the physical parameters of FPGA but suffers from low throughput.

In [11], the authors used a single multiplier and adder with iterative accumulation for MAC computation, and AF has been used in every neuron. In DNNs, each layer has many neurons, and it requires more hardware resources due to parallel architecture and consumes more on-chip power. Based on the concise review, we address the hardware-efficient and performance-centric solution. We have designed a signed 8-bit dynamic fixed-point hardware-reused DNN accelerator with insignificant loss in throughput. In addition, we have resized the MAC output that allows efficient use of AF with lower precision implementation. Further, area-efficient Taylor series expansion is used for Piece-Wise Linear AF implementation. Finally, we used Block RAM to store pre-calculated values of sigmoid AF, which is beneficial at higher precision.

Table 1
Sigmoid and tanh activation function computational equations with different representation for its design exploration and evaluation.

Preliminary Work	Activation Function Implementation	
	Sigmoid	Tanh
[41,46,26]	$\frac{1}{(1+e^{-z})}$	$1-2Sigmoid(-2z)$
[47,48]	$\frac{1}{(1+e^{-z})}$	$\frac{(e^{2z}-1)}{(e^{2z}+1)}$
[27,45]	$\frac{1+tanh(z/2)}{2}$	$\frac{(e^{z}-e^{-z})}{(e^{z}+e^{-z})}$
[6,12]	$\frac{e^{z}}{(1+e^{z})}$	$\frac{sinh(z)}{cosh(z)}$

3. Embedded system architecture

This section explains the methodology for the efficient design and implementation of the DNN accelerator. The FPGA-based embedded approach is the best option because of its good performance, fast implementation, energy efficiency, and reconfigurability [42,49]. In this context, we have developed an efficient hardware architecture of the DNN that employs the AF reuse technique that reduces the hardware resource requirements. The proposed system architecture is validated on a 16-level thermometer to 4-bit binary code converter using four layers *16:16:10:4* DNN with a configuration of 1-input, 2-hidden, and 1-output layers. In each layer, output values of an array of MACs are serialized using a shift register (PISO) and iteratively passed through a single AF.

We developed an efficient architecture that gives the required performance within the overall system requirement (size, weight, power, cost, etc.). The complete embedded architecture with the DNN hardware accelerator is implemented on the Xilinx Zybo board. The FPGA implementation have been performed using Xilinx *Vivado* HLx and Xilinx SDK for hardware design and data initialization, respectively. The Advanced eXtensible Interface (AXI) bus

is used to connect the custom DNN core to the controlling CPU. The AXI bus supports DMA data transfer and achieves high data throughput. The hardware implementation have done using VHDL hardware description language, and the external communication through the Processing System (PS) is developed in 'C' for an embedded architecture.

3.1. System overview

This work demonstrates the four-layer neural network on an affordable Xilinx FPGA SoC. Since BRAM memory has low latency, our architecture uses BRAM memory for the FIFO buffer implementation, for storing weights, and for storing pre-calculated values of the AF. The software control of the embedded processing system has been implemented using Xilinx SDK. The software control module is responsible for loading input features and weights into FIFO buffers. The design of the embedded architecture DNN co-processor is shown in Fig. 3.

FIFO buffers are efficiently implemented using internal BRAM since it is dual-port memory and has low latency [50]. The weights that are stored in the BRAM are loaded through FIFO buffers. The data is serialized before AF, allowing the reuse of AF, and the internal hardware architecture is customized for the data-flow. The weights are loaded serially into the FIFO buffer and out from it into the MAC local registers. After processing all layers of the neural network, the output of the last layer is stored in output FIFO buffers. Thus, the runtime access of weights and biases, i.e.. loading and accessing them, is efficiently designed. We devised an efficient technique for loading all necessary weights into FIFO buffers in sequence for all MAC computational elements. It allows the MACs not to wait to load the weights and access the BRAM, which has already been loaded in the local registers of the computational element.

The $\langle 8,7\rangle$ fixed-point format shown in Fig. 4 demonstrates the arithmetic computations used in developed architecture. This 9-bit field representation uses 1-bit as a sign bit, 1-bit for the integer part, and 7-bits for the fractional part. Symbolically, this is written as a signed 'Fixed $\langle 8,7\rangle$'. Thus, it represents a signed 8-bit fixed-point number of which seven rightmost bits are fractional. The HDL hardware coded DNN architecture *16:16:10:4* is validated for thermometer level to binary value conversion using 'fixed $\langle 8,7\rangle$' representation, and it achieves 98.9% accuracy with insignificant throughput loss compared to fully parallel architecture.

3.2. Memory addressing scheme for reading/ writing of weights and biases

In a fully connected neural network, DNN hyperparameters vary with different applications. Hence, architecture should be upgradable regarding the number of layers in the neural network and the number of neurons in each layer. The weights/bias addressing should have a viable scheme for memory allocation.

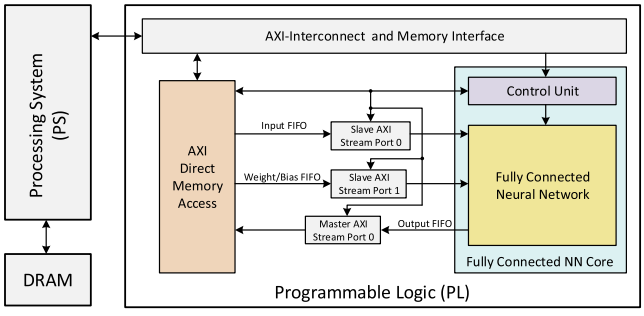

Fig. 3. The DNN co-processor architecture in which DNN architecture and on-chip BRAM memory banks, used to cache weights/biases, are implemented using programmable logic.

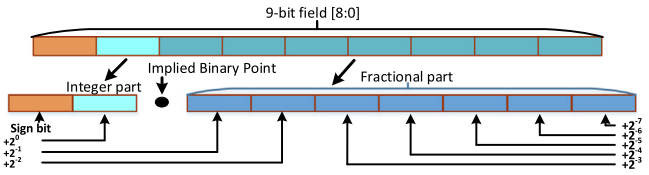

Fig. 4. Signed Fixed $\langle 8,7\rangle$ precision representation used for the data representation with binary point implication and arithmetic calculation.

In fixed dedicated memory allocation in adaptable Deep Neural Networks such as [51], several memory locations do not correspond to any physical implementation. Hence, it should avoid read or write operation on these unimplemented addresses. Thus, an efficient scheme of memory addressing is required for the weight/bias access. Here, we use FIFO for the temporary data storage, and BRAM is used for the on-chip storage. An efficient address mapping scheme for weight/bias memory for each neuron is shown in Fig. 5. The scheme is derived using the following labels:

- L is the total number of layers of the DNN model,
- $N(l)$ is the number of neurons or biases in l^{th} layer,
- $J(l)$ is the number of inputs to the l^{th} layer.

Note that the number of neurons in current layer is a number of inputs to the following neuron as given in Eq. 3.

$$J(l+1) = N(l) \tag{3}$$

Fig. 5(a) gives an addressing scheme to access the DNN parameters (weights and biases). In this scheme, a parameter is identified by its layer ID, by its type (weight/bias), by its neuron ID, and in the case of weights by the corresponding input ID. First $\lceil log_2(L)\rceil$ MSB bits represent the layer ID of the addressed parameter. The next bit is called a *select bit*, and it determines the type of the parameter given in subsequent bits. *Select bit* = '1' denotes that the following bits represents bias address, whereas '0' denotes that the following bits represents the weight address. Bits following the *select bit* represent weight or bias RAM address as shown in Fig. 5(b). The length of the weight or bias address $R\_addr(l)$ is the maximum of length of the weight address and the bias address and is given by Eq. 4a. Whereas the bias address consist only of neuron identifier with length $\lceil log_2 N(l)\rceil$, the weight address includes also input identifier with the length $W\_addr(l) = \lceil log_2 J(l)\rceil$. The required address length including layer ID and parameter type $Addr(l)$ for layer l is given by Eq. 4b.

$$R\_addr(l) = \lceil log_2 N(l)\rceil + \lceil log_2 J(l)\rceil \tag{4a}$$

$$Addr(l) = \lceil log_2 L\rceil + 1 + R\_addr(l) \tag{4b}$$

Since the required address length varies depending on the layer and since fixed address length is used in the addressing scheme, the address length $Addr$ is the maximum required address length over all layers $l = 1,2,\ldots,L$, and is given by Eq. 5b.

$$R\_addr = \max_{l=1,2,\ldots,L}\{\lceil log_2 N(l)\rceil + \lceil log_2 J(l)\rceil\} \tag{5a}$$

$$Addr = \lceil log_2 L\rceil + 1 + R\_addr \tag{5b}$$

4. Hardware design and implementation of deep neural network

This section presents the proposed performance-centric hardware-reused DNN implementation. The proposed architecture

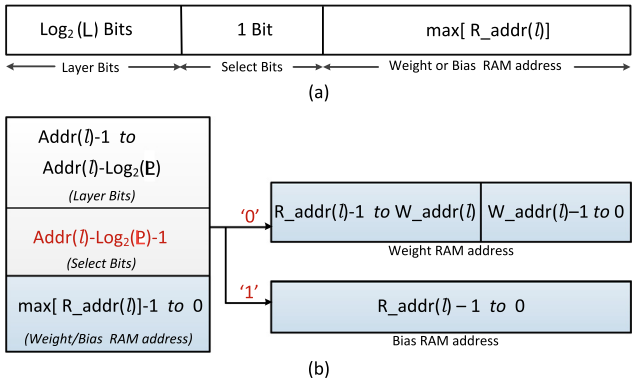

Fig. 5. Mapping scheme for address bits that requires to address weights and bias for the individual neurons.

reuses the resource-hungry non-linear AF using data multiplexing and is area and power-efficient, and with minimal performance loss compared to fully parallel implementation. It is implemented and tested on FPGA; however, the proposed DNN core can also be efficiently implemented in ASIC. The complete DNN core is written in *VHDL*-hardware description language, and it is integrated using block design of Xilinx *Vivado Design Suite*.

The dynamic $\langle 8,7 \rangle$ fixed-point arithmetic used in the developed DNN architecture was selected through experimental evaluations as a compromise between resource usage, performance, and accuracy. Compared to the floating-point format, the $\langle 8,7 \rangle$ fixed-point format comes with two advantages. Firstly, fixed-point computational units are usually fast and consume minimal hardware resources and power, i.e.. the logic design with fixed-point arithmetic will allow more instantiating for the given area. Secondly, memory footprint will be reduced, thus allowing larger models in given memory capacity. In addition, it dramatically increases data parallelism. The following subsections describe key features of the accelerator design and implementation.

4.1. Integrated block design of DNN accelerator

Hardware-based implementations of Deep Neural Networks face the challenges of memory bandwidth limits. Therefore, the efficient addressing scheme for dynamic loading of the weights and the bias constants in the local registers of the MAC unit is required. All FIFO registers used to store trained weight/bias constants are loaded using the scheme as discussed in Section 3.2. The block design of the embedded architecture is shown in Fig. 6. Annip_0 block, which is marked in orange, is the proposed DNN core and is connected to PS by three slave and one master AXI interconnects. The two slave AXI streams are used to access the input features and weight/bias constants. The third slave interface is an AXI lite interface used by the DNN core control module. The control module manages data exchange, handshake signals, and other configurable signals.

`Ready`, `DNNInit`, and `DNNDone` are three control signals at the top-level module that control data exchange, initiate computing, and acknowledge completed calculations to the top-level module. The `Ready` signal initiates the loading of the weight/bias into the local registers of the DNN core. The `DNNInit` signal triggers the computation once the weight/bias loading is done. The `DNNDone` is a control signal which indicates that the computation has finished and the DNN output is ready.

4.2. DNN architecture and layers'computation mechanism

In order to initiate the DNN computation, the `DNNInit` signal becomes active once the weight and bias constants are loaded in the local registers of the MAC units. Each layer has the `LayerInit` input signal used to initiate the layers computation, whereas `LayerDone` signal is generated by the layer which is used to acknowledge that the output of the layer is ready. The top-level DNN Core has two additional sub-modules which interacts with the control module:

- Weight loading module: Due to the limited number of DMA channels available in Zynq, we found a workaround for loading the weight in sequence into corresponding FIFOs by multiplexing the handshake signals based on the index of the DMA data transfer request, i.e., neuron's FIFO that accepts the data is selected based on index of loading DMA request.
- Computational module: It is built from Layer modules which perform calculations for individual DNN layers and are daisy-chained by data and handshake signals. Each layer module is controlled by two handshake signals: `LayerInit` and `LayerDone`. First is the input signal of the layer module that initiates layer computation while latter is the output signal of the layer module and indicates that the layer computation has finished. Since the computation of subsequent layer has to start after the computation of current layer has finished, the `LayerInit` signal of the subsequent layer is connected to the `LayerDone` signal of the current layer. Similarly, the output of the current layer is connected to the input of the subsequent layer as depicted in Fig. 7. The `LayerInit` signal of the first layer is connected to the top-level `DNNInit` control signal, which initiates the DNN computation. On the other hand the `LayerDone` signal of the last layer is connected to the top-level `DNNDone` control signal, which acknowledges the master *AXI stream*, which is used to access the output of the DNN core as shown in Fig. 6, that the DNN computation is finished.

In DNNs, each layer may have many parallel neurons. Each neuron consists of a MAC computational unit followed by an AF. Thus, in a fully parallel architecture, the layer with N neurons has N number of MACs and AFs. In contrast to the fully parallel architecture, the proposed architecture uses N number of MAC units and a single AF unit shared through Parallel-In Serial-Out (PISO) unit placed in between the MACs and AF. The serial output stream of the current layer is the input feature $i_1, i_2, \ldots i_j$ for the subsequent layer, i.e number of

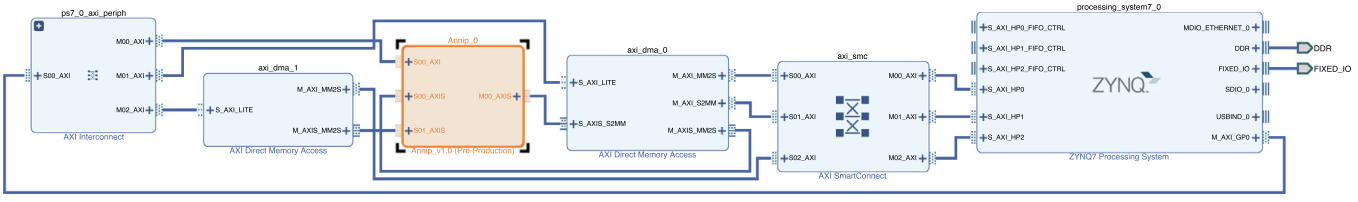

Fig. 6. Block design of system architecture with use of AXI interconnects. The proposed design DNN IP (Annip_0) is shown in orange color. Processing system (PL) is used for the external communication.

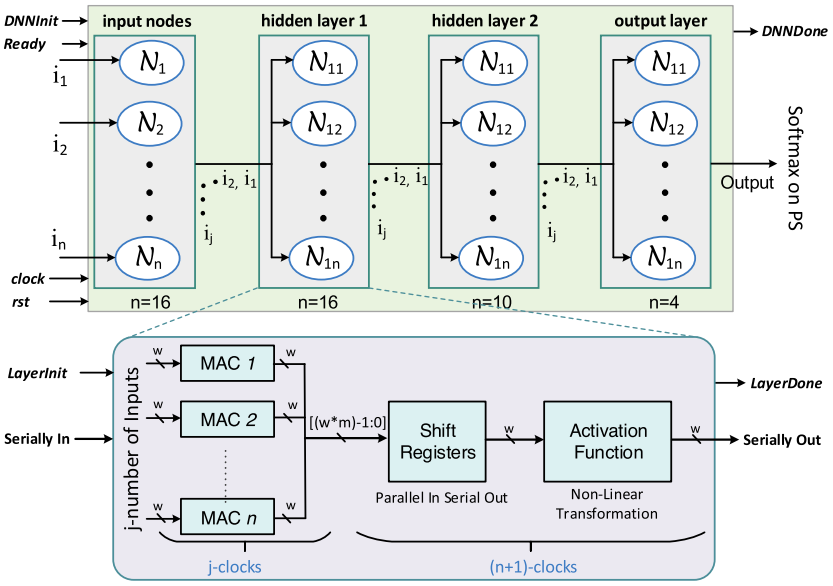

Fig. 7. DNN architecture with efficient design of hidden layer's architecture that reuses resources hungry Activation Function. The parallel output of array of MAC is serialised using shift register and pass it in packets to the AF.

inputs of the next layer is the number of neurons in the current layer, as shown in Fig. 7. In order to achieve better performance, the proposed design for each layer has the following key features:

1. FIFO Buffer: Each multiply-accumulate unit has an input buffer and an output buffer to receive and send the input, weight, and output data in FIFO.
2. Pipeline Accelerator: We use a stream-like data passing mechanism (AXI stream) to transfer data between the input data and multiply-accumulate unit.
3. Neural Unit: Contains one FIFO and one MAC unit that is used for computation of one neuron in correspondence with its assigned index.
4. Data Quantization: The AF is implemented for limited precision in order to have a minimum resources utilization. Hence the output of the processing unit is resized into 'Fixed $\langle 8,7 \rangle$'.
5. Data Serialize: The output of the array of MAC unit available in the single-layer are serialized to reused the single AF within the layer.

The timing complexity of the proposed DNN code has been elaborated by determining the number of clock cycles needed to perform the computation. Let T_R and T_P represent the number of clock cycles delay required to compute outputs for proposed AF *Reused* and fully *Parallel* DNN architecture, respectively. The proposed data-multiplexed layer architecture computes the MAC output in j clocks, where j is the number of layers' input features, and computes the first neuron output at $j+2$ clock cycles while the outputs of the following neurons within the layer are computed in every clock cycle. One can notice that it is due to the sharing of a single AF in each layer. Whereas in fully parallel layer architecture, it computes outputs for all MAC in j clock cycles and computes outputs of all neurons in $j+1$ clock cycles. It is important to note that while parallel architecture computes all outputs at $j+1$ clock cycles, the next layer reads them serially one per clock cycle due to the iterative computational nature of the used MAC unit. This implies that in the case of hidden layers the data multiplexed architecture introduces the delay of the PISO module, which is one clock cycle as described in Eq. 6b.

$$T_R(l) = T_{MAC}(l) + T_{PISO} + T_{AF} = j(l) + 1 + 1 \tag{6a}$$

$$T_P(l) = T_{MAC}(l) + T_{AF} = j(l) + 1 \tag{6b}$$

On the other hand, the output layer does not have the next layer and the serial computation of the AF unit must be taken into account. Therefore, the computation time of the output layer of the proposed architecture output requires additional $n-1$ clock cycles to compute all outputs. The computation time of the output layer for both architectures is given in Eq. 7a and Eq. 7b.

$$T_R(L) = T_{MAC}(L) + T_{PISO} + T_{AF} = j(L) + 1 + n(L) \tag{7a}$$

$$T_P(L) = T_{MAC}(L) + T_{AF} = j(L) + 1 \tag{7b}$$

The computation time T_R of the whole AF reused DNN architecture is computed by summing the computational time of the individual layers. The first layer is input layer and does not perform computations and is omitted from the sum illustrated in Eq. 8.

$$\begin{aligned} T_R &= \sum_{l=2}^{L-1} T_R(l) + T_R(L) \\ &= \sum_{l=2}^{L-1} (j(l)+2)) + j(L) + 1 + n(L) \end{aligned} \tag{8}$$

By applying the property that number of neurons in a layer is equal to the number of inputs in next the layer (Eq. 3) the equation can be simplified to Eq. 9.

$$\begin{aligned} T_R &= \sum_{l=1}^{L-2} n(l) + 2(L-2) + n(L-1) + 1 + n(L) \\ &= \sum_{l=1}^{L} n(l) + 2L - 3 \end{aligned} \tag{9}$$

Similarly, the computation time T_P of the fully parallel DNN architecture is evaluated using Eq. 10.

$$T_P = \sum_{l=1}^{L-1} n(l) + L - 1 \tag{10}$$

By comparing T_R and T_P computation time, it can easily be seen that the computation delay of the AF Reused DNN architecture requires more clocks which is equals to number of hidden layers $h = L - 2$ and by the number of DNN outputs $n(L)$ as given in Eq. 11. Further, time delay of accelerator output can be computed by simply multiplying clock time t_{CLK} with the number of clock periods T_R evaluated by using Eq. 9.

$$T_R = T_P + h + n(L) \tag{11}$$

Let us illustrate how the computation time can be determined on the evaluation example of the `16:16:10:4` feed-forward DNN using proposed DNN architecture. First hidden layer with 16 neurons has $j(2) = 16$ inputs and its MAC units require 16 clock cycles to compute weighted sum. The processing of the output of the first neuron requires two additional clock periods: one by PISO unit and the other for AF unit, hence the output of the first neuron is computed in 18^{th} clock cycle as given in Eq. 6a. Second hidden layer has 10 neurons and 16 inputs and can start processing after 18 clock cycles. By the analogy, it requires additional 18 clock cycles. Thus, the output of its first neuron is available after 36 clock cycles. The output layer has four neurons and 10 inputs. Therefore, 10 clock cycles are used for MAC units to compute weighted sum and one clock is used by PISO unit. However, to complete the DNN computation, the outputs of all neurons must be determined. Due to serial AF computation, it requires four clock cycles. Hence, 15 clock cycles are required for the output layer. For this Example, 51 clock cycles are required for DNN computation in this example. Similarly, 45 clocks (i.e.. 17 + 17 + 11) are required to compute the output of the fully parallel DNN.

4.3. Architecture design and implementation of w-bit precision Multiply-Accumulate unit

Each neuron has a MAC computational unit followed by an AF unit. The MAC unit with parallel multipliers and adder tree in each neuron are hardware costly and burdensome on tiny FPGAs. Hence, iterative MAC architecture that uses minimal resources utilization at the cost of throughput loss [12] is preferred. The resources-efficient iterative MAC unit used in this work is shown in Fig. 8. It requires j clocks for the computation of weighted sum, where j is the number of inputs at the MAC computation. The standard multiplier and adder defined in IEEE library package are used, and implemented using logic slices (CLBs) on FPGA. Here, we used fixed-point 8-bit precision 'Fixed $\langle 8,7 \rangle$' arithmetic for input feature and weights/biases i.e, `input[8:0]`, `weight[8:0]` and `bias[8:0]`. The output of the MAC unit is the sum of input bits, weight memory bits, and overflow bits required for the weighted sum accumulations. Therefore, output of multiplier gets accumulated using adder with extra overhead bits (k), i.e.. k=$\lceil log_2 j \rceil$. We use `select_b` to load the bias constant that gets added in first multiplication, i.e.. at the first clock initiating the MAC computation. Afterwards, it selects the accumulation path using the same `select_b` signal.

The neurons have MAC followed by non-linear transformation over the weighted accumulated sum, and therefore, the precision

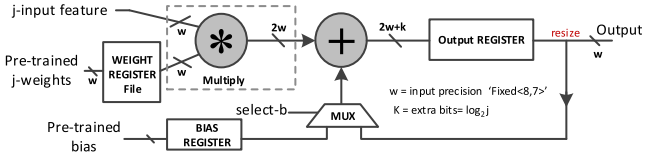

Fig. 8. Iterative Multiply-Accumulate (MAC) computation unit with resized output data.

of an AF depends on the output bit-width of the MAC unit within the neuron. The MAC used in the proposed design is shown in Fig. 8. The output of the MAC unit has $[2w + k]$-bits, whereas practically it is not desirable to keep the $[2w + k]$ bits precision AF for hardware implementation. Hence, we have resized MAC output into w-bits (9-bits) using *numeric_std* library package of VHDL. It allows us to keep the same input and output precision for MAC implementation with dynamic fixed-point arithmetic 'fixed $\langle 8,7 \rangle$' as shown in Fig. 8. In most cases, it is beneficial to have the same input and output format that we use and implies $i_{b\_in} = i_{b\_out} = i_b$, $f_{b\_in} = f_{b\_out} = f_b$ and $w_{in} = w_{out} = w$, where i_b represents the integer bits excluding the sign bit, f_b represents the fractional bits and w represents the total number of bits at input/output. However, in the case of fixed-point format, final choice must be based on a target application that gives maximum accuracy with desired precision [37].

4.4. Design methodology and implementation of activation function

In DNN, MAC output is the input for a non-linear transformation function AF. Due to the non-linear nature of the AF, it demands more hardware resources for implementation, and resources utilization increases exponentially for higher precision. In addition, the AF with 16-bit and higher precision requires more memory elements for PWL implementation, and therefore, it is very costly for hardware utilization as the number of required memory elements increases exponentially with the quantization precision at the input of the AF. The sigmoid function is the extended version of an exponential function, and the exponential function can be evaluated using Taylor series expansion. For example, the Taylor series expansion for a hyperbolic and exponential function is shown in Eq. 12c. Whereas, sigmoid AF can be implemented using these Taylor series expansions, the higher order of Taylor series expansions returns better accuracy but comes with high precision representation. Hence, performance-centric evaluations have been done between the Taylor series order used in AF evaluation and DNN accuracy. The analysis of the impact of Taylor series order on inference accuracy is shown in Fig. 9.

$$sinh(z) = z + \frac{z^3}{3!} + \frac{z^5}{5!} + \frac{z^7}{7!} + \frac{z^9}{9!} + ... \tag{12a}$$

$$cosh(z) = 1 + \frac{z^2}{2!} + \frac{z^4}{4!} + \frac{z^6}{6!} + \frac{z^8}{8!} + ... \tag{12b}$$

$$e^z = sinh(z) + cosh(z) \tag{12c}$$

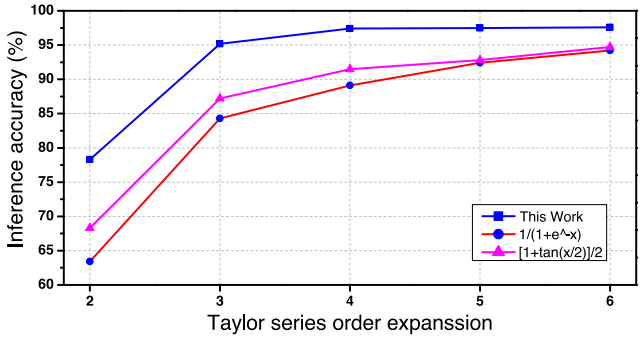

Fig. 9. Inference accuracy for fully connected neural network with sigmoid activation function. The network 784:256:128:128:10 is trained for MNIST dataset and used sigmoid with different order of Taylor expansion. The results shown for 'fixed $\langle 8,7 \rangle$' precision.

An exponential function has been evaluated using the sum of the Taylor series for *sinh* and *cosh* function, as shown in Eq. 12c. We have used an optimized sigmoid AF evaluation technique for both negative and positive inputs. In this work, sigmoid function is represented using positive exponential function as given *f(z)* for positive and negative inputs. The elaborated equations for the positive and negative half are shown in Eq. 13. The Pareto study between series expansion order and accuracy shows that accuracy gets saturated after the fifth-order of series expansion as shown in Fig. 9. Hence, fifth-order Taylor series expansion has been used for the memory elements extraction in the 'Fixed $\langle 8,7\rangle$' representation. Further, extracted memory elements for sigmoid AF are stored in the BRAM for real-time function evaluation. The proposed approach can easily be extended for *tanh* AF implementation simply by dividing the *sinh* and *cosh*. However, sigmoid and *tanh* AF can also be implemented using a different approach as discussed in Table 1.

$$f(z) = sigmoid(z) = \begin{cases} \frac{e^z}{1+e^z}, & \text{if } z \geqslant 0 \\ \frac{1}{1+e^{-z}}, & \text{otherwise} \end{cases} \quad (13)$$

The implemented sigmoid AF design is efficient in terms of physical performance parameters such as area, power, and delay, where we have used only positive exponential function for sigmoid AF evaluation. We have analyzed the order representation of Taylor series expansion for sigmoid AF. The order of Taylor series expansion for AF versus inference accuracy for the proposed scheme and state of the art is shown in Fig. 9. The accuracy results are evaluated for the MNIST dataset on DNN configuration `784:256:256:128:10`. The arithmetic representation of the integer bits and the fractional bits have been presented using the 'Fixed $\langle 8,7\rangle$' scheme. Here, we trained the model for 20 epochs and stopped the training when the validation accuracy decreased consecutively for two epochs. Then,we evaluated the trained model on a test set of MNIST and observed the accuracy. Conventionally these equations computations represent the same output as shown in Fig. 9; however, due to 8-bit fixed-point representation, numbers get quantized at every math operation involved in AF computations. Thus, results show the sigmoid with the proposed technique has better accuracy compared with equations used in previous works [48,52].

5. Experimental setup

The system architecture is represented at the RTL level using VHDL language to evaluate the performance of the proposed design. The RTL for our system architecture is synthesized using Xilinx *Vivado,* and implementation results are produced for Zybo FPGA hardware. Inference accuracy for sigmoid AF on DNN is evaluated through software evaluation. Further, for ASIC design, synthesis results for an optimized sigmoid AF are extracted using Synopsys *Design Vision*. Following experiments are done to validate our proposed design:

1. The inference accuracy has been evaluated for different precision 8, 16, and 32-bit fixed-point arithmetic representations of sigmoid AF implementation with different approaches to finalize the precision selection for hardware implementation.
2. The DNN configuration `784:256:128:128:10` is designed for 'Fixed $\langle 8,7\rangle$' precision using QKeras. The Pareto study tuned the Taylor series order expansion for the sigmoid AF implementation, which returns better accuracy and physical performance parameters. MNIST dataset is used for evaluation and validation of AF. Further exploration for ASIC, synthesis results for 8-bit precision are presented at *45 nm* technology.
3. The proposed hardware efficient data multiplexed DNN architecture implantation is validated on the Zybo board for network size `16:16:10:4`. Here, we have designed a custom IP for the proposed architecture and show the implementation results for the FPGA prototyping of the proposed resources-reused (i.e., AF) design architecture.

6. Experimental results and discussion

We have evaluated the system architecture at different abstraction levels for setting the design parameter for hardware implementation. Finally, the embedded design of the DNN has been implemented in hardware. The detailed analysis and evaluation results are given in the following subsections.

6.1. Accuracy for multi-bit precision and Pareto study for Taylor series expansion

The QKeras model is used for the training of the DNN as it allows drop-in replacement for the quantized versions of the conventional *Dense* and *Activation* layers. The accuracy has been observed for the MNIST dataset on DNN configuration `784:256:256:128:10` with 8, 16, and 32-bit precision, as shown in Table 2. Furthermore, our sigmoid AF design's evaluated accuracy using MNIST dataset has been compared with the accurate AF function of the Tensor library [53] and Piece-Wise Linear Activation Function [47]. From Table 2, it can be observed that there is an insignificant accuracy loss ($\leqslant$2%) between 8-bit and 32-bit precision computation. However, 8-bit precision computation leads to reduce memory computation demand by a factor of 4. Hence, we have chosen the fixed-point 8-bit precision for hardware implementation. We have also evaluated the optimal position of the decimal point, i.e., one integer bit and the rest fractional bits in the dynamic fixed-point arithmetic.

6.2. Data multiplexed DNN implementation and results comparison

The Xilinx Zybo (xc7z010clg400) FPGA hardware contains 4,400 logic slices; each logic slice includes four 6-input LUTs and 8-FF. Additionally, it provides 240 KB of BRAM. In order to evaluate the resources utilization and power consumption, we used Xilinx *Vivado* tool. Through an efficient addressing scheme, local memory registers with weights and bases are loaded. Architecture has been implemented on Zybo. Furthermore, the DSP blocks have not been used for logic implementation due to their fixed architecture for higher precision and power consumption constraints.

We have implemented data multiplexed hardware reused architecture and results are extracted. For the sake of comparison with state-of-the-art approaches, we used the same design parameters which are 'Fixed $\langle 8,7\rangle$' arithmetic, DNN size, and evaluation kit for the implementation of all the previous and proposed work. We used DNN configuration `16:16:10:4` for the implementation and results comparison. The hardware architecture has been designed using *VHDL* and for DNN core extracted post-implementation results on the *Zybo* FPGA board are shown in Table 3.

In [22], authors have designed DNN with fully parallel layers. The network has high throughput and flexibility to change the number of nodes in the network, allowing us to configure the network for different applications. However, the design architecture has further scope to make it hardware efficient using the data multiplexed and AF reused scheme. An efficient hardware design has been proposed where authors have reused the AF [33]. This work has used a multiplexer between the parallel MAC units and AF in

Table 2
Network inference accuracy of DNN size 784:256:128:128:10 for Piece-Wise Linear (PWL) AF [47] and TenserFlow library based AF [53] at different bit-precision with dynamic fixed-point representation @MNIST dataset.

Activation function	Inference accuracy (%) for different AFs					
	ReLU	Sigmoid			tanh	
precision	Tensor [53]	Proposed	PWL [47]	Tensor [53]	PWL [47]	Tensor [53]
32-bit	97.51	98.20	97.94	98.30	97.14	98.58
16-bit	96.71	97.63	97.86	97.58	96.90	98.53
8-bit	97.50	97.06	97.58	96.68	95.54	97.93

Table 3
Resource utilization and performance parameters for 4-layer (*16:16:10:4*) on xc7z010clg400 at 100 MHz.

Hardware Avail. Logic Resources	DNN with AF Reused [33]	DNN with Layer Multiplexed [36]	DNN with Layer Reused [11]	DNN with Fully Parallel AF [22]	Hardware Reused (AF) Proposed	% Utilized Relative [22]
Slice LUTs (17600)	10439	6733	6127	9968	8858	88.86
Slice FFs (35200)	7573	6018	5904	10638	6219	58.46
BRAM (60)	6.5	9	13.5	11	6.5	59.09
BUFG (32)	8	8	8	8	8	–
On-chip Power (*mW*)	158.5	88.12 × L	74.04 × L	120.46	106.05	88.33
I/O Critical Delay (*ns*)	11.6	4.07 × L	3.28 × L	9.23	9.86	−6.82
Throughput (*samples/s*)	T	T/L[1]	T/L[1]	T	T	–

[1]L is the number of layers available in the implemented DNN.

which MAC address is the input select line for the MUX. The hardware implementation results show less LUT utilization as authors have reused the AF through multiplexing shown in Table 3. However, these techniques are not efficient for a larger size of a DNN as multiplexer increases the time complexity of the logic architecture, and further bigger size multiplexer requires more hardware resources. We have addressed this issue and proposed an efficient AF reused technique using shift register following the Parallel-In Serial-Out's mechanism.

The proposed technique is efficient for any deep neural network at very insignificant throughput loss, as we discussed in Section 4.2. For the performance parameter comparison, we have synthesized the different DNN architectures and proposed DNN core for `16:16:10:4`. The implementation report for DNN core is shown in Table 3. The hardware implementation results show that our method uses *25%* fewer resources and requires *12%* less power, without performance loss compared to the work proposed in [22] at the cost of delay overhead compared to parallel architecture in DNN as discussed in Section 4. Here, L is the number of layers available in the DNN. Therefore, L-1 clocks period delay occurs for the first output in comparison with the fully parallel architecture. However, after the first output, the network computes the output at every clock, and hence, proposed hardware-reused architecture has insignificant throughput loss, as we have discussed in Section 4.2. Compared with AF reused architecture design in [33], implementation results show 17.8% and 21.7% fewer LUT and FF utilization respectively. Further, the proposed design has 17.6% less critical delay. For the proposed design, the comparison numbers will be adequate for the bigger sizes of DNNs.

In order to design resource-efficient architecture, state-of-the-art works have proposed layer-reused architectures [11,36]. Though the design is hardware efficient, from Table 3, one can see that layer-reused designs have $L\times$ lower throughput as compared to fully parallel architecture. This architecture implementation will be efficient for less resources utilization at the cost of low throughput. In [36], the author has used the multiplexer for passing the output data of the current layer to the input of the same layer; using this, one can reuse the single layer at any number of times within the DNN. However, the design suffers from low throughput. Moreover, the design has used parallel AFs within the layer. Therefore, this design has further scope to reduce resources utilization using the AF reused technique by compromising throughput loss of one clock period in each layer computation. The resources utilization for fixed parameters is shown in Table 3. It can be observed that [11] has higher BRAM utilization compared the [36]; as no quantization scheme was used in [11]. However, these designs can adopt the proposed AF reused scheme to further optimize hardware utilization.

The design's performance parameters are in the trade-off for area/power and throughput. However, the proposed design will be the best choice for area/power and throughput-centered applications where resources and power are on a tight budget, such as Mobile, edge-AI, IoT applications, etc.

6.3. Implementation results and Comparison of proposed AF at multi-bit precision

In an AF, moving from 8-bit to 12-bit and 16-bit, the number of quantization states is equal to $2^8 = 256, 2^{12} = 4096$, and $2^{16} = 65,536$ which leads to an exponential increase in the memory elements required. Therefore, we used the pre-calculated memory element for AF implementation using the Taylor series expansion approach and efficiently stored it in BRAM. Table 4 shows the resource utilization comparison at different precision for our optimized architecture with the utilization of BRAM and LUT-based approach [43]. Generally, tiny FPGA have limited slice resources utilization, whereas it can be seen that 16-bit precision sigmoid AF needs *2111* LUTs. Here, it can be observed that higher precision AF implementation for LUT based approach is not the appropriate method. Hence, our design uses BRAM, which is scalable for higher precision, has better access time, and saves LUTs for other logic computations.

6.4. Comparison of AFs physical performance parameters at 45 nm technology node

In order to evaluate the design performance of our optimized AF for ASICs, we have synthesized design for fixed-point 8-bit precision at *45 nm* TT Process Corner. The efficiently extracted memory elements for sigmoid AF have been compared with sigmoid using

Table 4
Activation function implementation using LUT based and proposed technique for Zybo xc7z010clg400.

Resources	Sigmoid Activation Function Resources Utilization					
Utilization	8-bit		12-bit		16-bit	
(Available)	LUT [46]	Proposed	LUT [46]	Proposed	LUT [46]	Proposed
LUT	16	1	370	1	2111	20
FF	0	1	0	1	0	5
BRAM	0	0.5	0	1.5	0	32
BUBG	0	1	0	1	0	1

Table 5
Performance parameter metrics of 8-bit precision proposed design for sigmoid AF and state-of-the-art @45 nm TT Process Corner.

Sigmoid AF Physical Parameters	Digital Implement [41]	Negative Input [48]	Iterative CORDIC [12]	Memory Elements Proposed
Chip Area (μm^2)	846	483	377	671
St. Power (μW)	27.6	121.7	96.77	21.6
Dy. Power (μW)	121.3	209.4	189.3	38.2
Delay (ns)	7.41	5.93	4.38	4.86
EDP	1105	11,780	6,264	287
No. of Clock	1	6	5	1

different design techniques. The *RTL* for our AF architecture is synthesized, and results are produced by Synopsis *Design Vision*. The physical performance parameters comparison is shown in Table 5. Results show that compared to CORDIC-based AF, we have *4.8*× higher throughput at the cost of *1.9*× of area overhead. Further, it can be seen that the memory elements-based proposed method has better physical parameters compared to the digital design technique [41].

7. Conclusion

We presented a novel DNN architecture that reduces the hardware-resource requirements and on-chip power consumption without the loss in computational rate. Our architecture has struck a demand of resource overhead in a Deep Neural Network by exploiting the hardware-reused context. The contribution of the work is twofold. Firstly, the activation function is designed efficiently for higher accuracy, and quantized memory elements are stored in BRAM for better utilization and higher throughput. Secondly, our experimental results demonstrate reused AF by serializing the output of the array of MAC in each layer. The efficient memory addressing scheme is used to overcome the *SoC* throughput limits. The implementation was verified at 100*MHz* for thermometer level to digital conversion. Using extensive evaluation, we show that our proposed design gives better results as compared to other designs. Thus, the embedded system design with lower hardware resources and power consumption can significantly benefit edge computing solutions.

CRediT authorship contribution statement

Gopal Raut: Conceptualization, Methodology, Software, Writing – original draft. **Anton Biasizzo:** Visualization, Investigation, Writing – original draft. **Narendra Dhakad:** Validation, Formal analysis. **Neha Gupta:** Writing – review & editing. **Gregor Papa:** Visualization, Writing – review & editing, Supervision. **Santosh Kumar Vishvakarma:** Project administration, Resources, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors would like to thank the University Grant Commission (UGC) New Delhi, Government of India under SRF scheme with award No. 22745/(NET-DEC. 2015) for providing financial support. And Special Manpower Development Program Chip to System Design (SMDP), Department of Electronics and Information Technology (DeitY) under the Ministry of Communication and Information Technology, Government of India for providing necessary Research Facilities. The authors also acknowledge the financial support from the Slovenian Research Agency (research core funding No. P2-0098). The work is also part of a projects that have received funding from the ECSEL Joint Undertaking under grant agreements No 876038 (InSecTT) and No 101007273 (DAIS).

References

[1] J. Gama, R. Sebasti ao, P.P. Rodrigues, Issues in evaluation of stream learning algorithms, in: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2009, pp. 329–338.
[2] K. Sun, J. Zhang, C. Zhang, J. Hu, Generalized extreme learning machine autoencoder and a new deep neural network, Neurocomputing 230 (2017) 374–381.
[3] J. Ma, R.P. Sheridan, A. Liaw, G.E. Dahl, V. Svetnik, Deep neural nets as a method for quantitative structure–activity relationships, J. Chem. Inf. Model. 55 (2) (2015) 263–274.
[4] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, F.E. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing 234 (2017) 11–26.
[5] E. Monmasson, L. Idkhajine, M.N. Cirstea, I. Bahri, A. Tisan, M.W. Naouar, FPGAs in industrial control applications, IEEE Trans. Ind. Inf. 7 (2) (2011) 224–243.
[6] G. Raut, S. Rai, S.K. Vishvakarma, A. Kumar, A CORDIC based configurable activation function for ANN applications, in: 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), IEEE, 2020, pp. 78–83.
[7] M.C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, Achieving high performance with FPGA-based computing, Computer 40(3) (2007)..
[8] E. Wu, X. Zhang, D. Berman, I. Cho, J. Thendean, Compute-efficient neural-network acceleration, in: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019, pp. 191–200.
[9] A.R. Omondi, J.C. Rajapakse, FPGA implementations of neural networks, Springer, vol. 365, 2006..
[10] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, D. Marr, Accelerating binarized neural networks: comparison of FPGA, CPU, GPU, and ASIC, in: Field-Programmable Technology (FPT), 2016 International Conference on, IEEE, 2016, pp. 77–84.
[11] S. Himavathi, D. Anitha, A. Muthuramalingam, Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization, IEEE Trans. Neural Networks 18 (3) (2007) 880–888.
[12] G. Raut, S. Rai, S.K. Vishvakarma, A. Kumar, Recon: Resource-efficient cordic-based neuron architecture, IEEE Open J. Circuits Syst. 2 (2021) 170–181. doi:10.1109/OJCAS.2020.3042743..

[13] K. Khalil, O. Eldash, A. Kumar, M. Bayoumi, An efficient approach for neural network architecture, in: 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS), IEEE, 2018, pp. 745–748.
[14] H. Zhu, C. Chen, S. Liu, Q. Zou, M. Wang, L. Zhang, X. Zeng, C.-J.R. Shi, A communication-aware DNN accelerator on imagenet using in-memory entry-counting based algorithm-circuit-architecture co-design in 65-nm CMOS, IEEE J. Emerg. Sel. Top. Circuits Syst. 10 (3) (2020) 283–294.
[15] Y. Zhang, C. Hu, B. Jiang, Embedded atom neural network potentials: Efficient and accurate machine learning with a physically inspired representation, J. Phys. Chem. Lett. 10 (17) (2019) 4962–4967.
[16] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, B. Yu, Recent advances in convolutional neural network acceleration, Neurocomputing 323 (2019) 37–51.
[17] N.P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al., In-datacenter performance analysis of a tensor processing unit, in: Proceedings of the 44th annual international symposium on computer architecture, 2017, pp. 1–12.
[18] Y.-H. Chen, T.-J. Yang, J. Emer, V. Sze, Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices, IEEE J. Emerg. Sel. Top. Circuits Syst. 9 (2) (2019) 292–308.
[19] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, et al., Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration, in: Proceedings of the 58th Annual Design Automation Conference (DAC), 2021.
[20] Y.S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, et al., Simba: Scaling deep-learning inference with multi-chip-module-based architecture, in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp. 14–27.
[21] Y. Qian, D. Wu, W. Bao, P. Lorenz, The internet of things for smart cities: Technologies and applications, IEEE Network 33 (2) (2019) 4–5.
[22] K. Khalil, O. Eldash, B. Dey, A. Kumar, M. Bayoumi, A novel reconfigurable hardware architecture of neural network, in: 2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS), IEEE, 2019, pp. 618–621.
[23] S. Liang, S. Yin, L. Liu, W. Luk, S. Wei, FP-BNN: Binarized neural network on FPGA, Neurocomputing 275 (2018) 1072–1086.
[24] J. Vreca, I. Ivanov, G. Papa, A. Biasizzo, Detecting network intrusion using binarized neural networks, in: 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), 2021, pp. 1–6 (in press).
[25] J. Han, Z. Li, W. Zheng, Y. Zhang, Hardware implementation of spiking neural networks on FPGA, Tsinghua Sci. Technol. 25 (4) (2020) 479–486.
[26] G. Baccelli, D. Stathis, A. Hemani, M. Martina, NACU: a non-linear arithmetic unit for neural networks, in: 2020 57th ACM/IEEE Design Automation Conference (DAC), IEEE, 2020, pp. 1–6.
[27] T. Yang, Y. Wei, Z. Tu, H. Zeng, M.A. Kinsy, N. Zheng, P. Ren, Design space exploration of neural network activation function circuits, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 38 (10) (2018) 1974–1978.
[28] X. Fan, S. Zhang, T. Gemmeke, Approximation of transcendental functions with guaranteed algorithmic QoS by multilayer pareto optimization, IEEE Trans. Very Large Scale Integr. VLSI Syst. 28 (12) (2020) 2495–2508.
[29] F. Kästner, B. Janßen, F. Kautz, M. Hübner, G. Corradi, Hardware/software codesign for convolutional neural networks exploiting dynamic partial reconfiguration on PYNQ, in: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2018, pp. 154–161.
[30] J. Han, M. Orshansky, Approximate computing: An emerging paradigm for energy-efficient design, in: Test Symposium (ETS), 2013 18th IEEE European, IEEE, 2013, pp. 1–6..
[31] V. Pejovic, Towards approximate mobile computing, CoRR, vol. abs/1901.08972, 2019. [Online]. Available: http://arxiv.org/abs/1901.08972..
[32] K. Khalil, O. Eldash, A. Kumar, M. Bayoumi, N2OC: Neural-network-on-chip architecture, in: 32nd IEEE International System-on-Chip Conference (SOCC), IEEE, 2019, pp. 272–277.
[33] L. Prono, A. Marchioni, M. Mangia, F. Pareschi, R. Rovatti, G. Setti, A high-level implementation framework for non-recurrent artificial neural networks on FPGA, in: 2019 15th Conference on Ph. D Research in Microelectronics and Electronics (PRIME), IEEE, 2019, pp. 77–80.
[34] J.G. Oliveira, R.L. Moreno, O. de Oliveira Dutra, T.C. Pimenta, Implementation of a reconfigurable neural network in fpga, in: 2017 International Caribbean Conference on Devices, Circuits and Systems (ICCDCS), IEEE, 2017, pp. 41–44.
[35] K. Khalil, O. Eldash, B. Dey, A. Kumar, M. Bayoumi, Architecture of a novel low-cost hardware neural network, in: 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), IEEE, 2020, pp. 1060–1063.
[36] T.V. Huynh, Deep neural network accelerator based on FPGA, in: 2017 4th NAFOSTED Conference on Information and Computer Science, IEEE, 2017, pp. 254–257..
[37] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision, in: International conference on machine learning. PMLR, 2015, pp. 1737–1746.
[38] I. Tsmots, O. Skorokhoda, V. Rabyk, Hardware implementation of sigmoid activation functions using fpga, in: 2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), IEEE, 2019, pp. 34–38.
[39] I. del Campo, R. Finker, J. Echanobe, K. Basterretxea, Controlled accuracy approximation of sigmoid function for efficient FPGA-based implementation of artificial neurons, Electron. Lett. 49 (25) (2013) 1598–1600.
[40] K. Basterretxea, Recursive sigmoidal neurons for adaptive accuracy neural network implementations, in: Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on, IEEE, 2012, pp. 152–158.
[41] S. Gomar, M. Mirhassani, M. Ahmadi, Precise digital implementations of hyperbolic tanh and sigmoid function, in: 2016 50th Asilomar Conference on Signals, Systems and Computers, IEEE, 2016, pp. 1586–1589..
[42] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, Optimizing FPGA-based accelerator design for deep convolutional neural networks, in: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ACM, pp. 161–170..
[43] V. Saichand, S. Arumugam, N. Mohankumar, et al., Fpga realization of activation function for artificial neural networks, in: 2008 Eighth International Conference on Intelligent Systems Design and Applications, vol. 3, IEEE, 2008, pp. 159–164..
[44] G. Rajput, G. Raut, M. Chandra, S.K. Vishvakarma, VLSI implementation of transcendental function hyperbolic tangent for deep neural network accelerators, Microprocess. Microsyst. 84 (2021) 104270.
[45] G. Raut, V. Bhartiy, S.K. Vishvakarma, et al., Efficient low-precision CORDIC algorithm for hardware implementation of artificial neural network, in: International Symposium on VLSI Design and Test, Springer, 2019, pp. 321–333..
[46] K. Leboeuf, A.H. Namin, R. Muscedere, H. Wu, M. Ahmadi, High speed VLSI implementation of the hyperbolic tangent sigmoid function, in: 2008 Third international conference on convergence and hybrid information technology, vol. 1, IEEE, 2008, pp. 1070–1073..
[47] A. Armato, L. Fanucci, E.P. Scilingo, D. De Rossi, Low-error digital hardware implementation of artificial neuron activation functions and their derivative, Microprocess. Microsyst. 35 (6) (2011) 557–567.
[48] M. Alçın, İ. Pehlivan, İ. Koyuncu, Hardware design and implementation of a novel ANN-based chaotic generator in FPGA, Optik-Int. J. Light Electron. Opt. 127 (13) (2016) 5500–5505.
[49] U. Legat, A. Biasizzo, F. Novak, SEU recovery mechanism for SRAM-based FPGAs, IEEE Trans. Nucl. Sci. 59 (5) (2012) 2562–2571.
[50] H. Le, W. Jiang, V.K. Prasanna, Scalable high-throughput sram-based architecture for IP-lookup using FPGA, in: International Conference on Field Programmable Logic and Applications, 2008, pp. 137–142.
[51] N. Shah, P. Chaudhari, K. Varghese, Runtime programmable and memory bandwidth optimized FPGA-based coprocessor for deep convolutional neural network, IEEE Trans. Neural Networks Learn. Syst. 29 (12) (2018) 5922–5934.
[52] A. Kundu, A. Heinecke, D. Kalamkar, S. Srinivasan, E.C. Qin, N.K. Mellempudi, D. Das, K. Banerjee, B. Kaul, P. Dubey, K-tanh: Efficient tanh for deep learning, arXiv preprint arXiv:1909.07729, 2019..
[53] M. Abadi, A. Agarwal, et al., Tensorflow: Large-scale machine learning on heterogeneous systems, 2015..

Gopal Raut received the B.Engg. in electronic engineering and M.Tech. in VLSI Design from G H Raisoni College of Engineering Nagpur, India, in 2015. He is currently pursuing the Ph.D. degree with the Electrical Engineering Department, Indian Institute of Technology Indore, India. His research focus is compute-efficient and configurable VLSI circuit design for low power IoT and edge-AI applications.

Anton Biasizzo is a researcher at the Jožef Stefan Institute, and an Assisting Professor at the Jožef Stefan International Postgraduate School. He received his BSc, MSc, and PhD in Electrical Engineering from the University of Ljubljana in 1991, 1995, and 1998, respectively. His research interests include design and test of digital systems, reconfigurable systems and embedded systems.

Narendra Dhakad recieved B. E. degree in Electronics and Communication Engineering from Indira Gandhi Engineering College, Sagar in 2015 and M.Tech. degree in Microelectronics and VLSI Design from Shri G. S. Institute of Technology and Science, Indore in 2017. He is currently pursuing PhD degree at IIT Indore, India. His research interest includes in-memory computation, edge AI Computing, SoC/FPGA based CNN hardware accelerator.

Neha Gupta received the B.Tech in electronics and telecommunication engineering from RGPV University Bhopal and the M.E. degree in electronics engineering from the Institute of Engineering and Technology, Devi Ahilya University, Indore, India, in 2017. She got her Ph. D. degree from IIT Indore, India, in 2021. Her research interests include reliable VLSI circuit and SRAM design, and low power high performance digital circuit designs.

Gregor Papa is a Senior researcher and a Head of Computer Systems Department at the Jožef Stefan Institute, and an Associate Professor at the Jožef Stefan International Postgraduate School. He received his BSc, MSc, and PhD degrees in Electrical Engineering from the University of Ljubljana in 1997, 2000 and 2002, respectively. His research interests include meta-heuristic optimization methods and hardware implementations of high-complexity algorithms.

Santosh Kumar Vishvakarma is currently an Associate Professor with the Department of Electrical Engineering, Indian Institute of Technology Indore, India, where he is leading Nanoscale Devices and VLSI Circuit and System Design Lab. He got his Ph.D. degree from Indian Institute of Technology Roorkee, India, in 2010. From 2009 to 2010, he was with the University Graduate Center, Kjeller, Norway, as an Post-Doctoral Fellow under European Union Project "COMON." His current research interests include nanoscale devices, reliable SRAM memory designs, and configurable circuits design for IoT application.