International Journal of Electrical and Computer Engineering (IJECE)
Vol. 14, No. 4, August 2024, pp. 3863~3875
ISSN: 2088-8708, DOI: 10.11591/ijece.v14i4.pp3863-3875

Field-programmable gate array implementation of efficient deep neural network architecture

Pottipati Dileep Kumar Reddy, Kota Venkata Ramanaiah
Department of Electronics and Communication Engineering, Faculty of Engineering, Yogi Vemana University/YSR Engineering College of YVU, Proddatur, India

Article Info

Article history:

Received Dec 12, 2023
Revised Mar 16, 2024
Accepted Apr 30, 2024

Keywords:

Deep neural network
Field-programmable gate array
High-speed
Parallel architecture
Pipeline architecture

ABSTRACT

Deep neural network (DNN) comprises multiple stages of data processing sub-systems with one of the primary sub-systems is a fully connected neural network (FCNN) model. This fully connected neural network model has multiple layers of neurons that need to be implemented using arithmetic units with suitable number representation to optimize area, power, and speed. In this work, the network parameters are analyzed, and redundancy in weights is eliminated. A pipelined and parallel structure is designed for the fully connected network information. The proposed FCNN structure has 16 inputs, 3 hidden layers, and an output layer. Each hidden layer consists of 4 neurons and describes how the inputs are connected to hidden layer neurons to process the raw data. A hardware description language (HDL) model is developed for the proposed structure and the verified model is implemented on Xilinx field-programmable gate array (FPGA). The modified structure comprises registers, demultiplexers, weight registers, multipliers, adders, and read-only memory lookup table (ROM/LUT). The modified architecture implemented on FPGA is estimated to reduce area by 87.5% and improve timing by 3x compared with direct implementation methods.

Corresponding Author:

Pottipati Dileep Kumar Reddy
Department of Electronics and Communication Engineering, Faculty of Engineering, Yogi Vemana University/YSR Engineering College of YVU
Korrapadu Road, Proddatur, Kadapa District, Andhra Pradesh 516360, India
Email: dkr.pottipati@gmail.com

1. INTRODUCTION

Deep learning systems have been used successfully in the past few years for industrial applications including image processing, speech processing, language translation, and object recognition. Deep neural networks (DNN) are trained to identify raw inputs from input data and accelerate feature identification. Training of DNN models with large data sets and the use of trained models for inferences need to be efficient with higher accuracy. Computing platforms such as central processing units (CPUs), graphics processing units (GPUs) and field-programmable gate arrays (FPGAs) for the implementation of DNN models are becoming popular for real-time use of DNN models. Edge computing is a key driving factor for DNN models to achieve real-time data processing. To speed up data processing in DNNs, parallelization, batch processing, sub-processing, and partitioning are used to make the best use of hardware resources more quickly. Several methods have been reported in the literature for reducing computation complexity in convolutional neural networks and DNNs. Techniques such as pruning [1], quantization [2], and decomposition [3] have been demonstrated to reduce complexity. Hardware accelerators that reduce computation complexity and are efficient for hardware implementation have also been reported in the literature [4], [5]. The convolutional layer that performs feature extraction is the most complex sub-system in convolutional neural network

(CNN) and it is required to have a trade-off of complexity, accuracy, and resource utilization. The last stage of the CNN model is the fully connected model that achieves the end objective of CNN by processing extracted features from the convolutional layers. The fully connected (FC) layer comprises matrix multiplication operations and the number of weights required for multiplication is based on the network configuration, input, and accuracy. The FC layer requires a large number of multiply and accumulate (MAC) operations and storage units for weights and biases. Storing and accessing the FC model parameters for computation is a challenge when used for edge computing. Several techniques are reported in the literature for efficient hardware implementation of the FC structure and CNN model. Intermediate internal storage units are incorporated to achieve parallelism and improve throughput [6]. Winograd algorithm is used to reduce the number of computations leading to a reduction in the use of DSP blocks for hardware implementation [7].

Fast Fourier transform (FFT)-based approaches have also been used to carry out computation in the frequency domain demonstrating improvement in computing performance but are observed to be suitable for large kernel logic [8]. Resource multiplexing algorithm is another method that is developed to optimize complexity in the number of arithmetic operations and use of lookup tables (LUTs) for CNN architecture implementation. Traditional structures for CNN implementation use cores for computing convolutional layers and the core sizes are set according to different stages. The use of CNN cores increases power dissipation and hardware resources [9]. Xiaokang [10] has reported the design of the CNN model considering XOR operations for multiplications, pipeline structure, parallelism, and intermediate storage for reducing delay in data access. The model is implemented on Artix-7 xc7z020clg400-1 FPGA and is found to operate at a maximum frequency of 150 MHz. Binfeng [11] has developed a CNN accelerator by developing the algorithm on Zynq FPGA for hardware configuration and ARM CORTEX-A9 for software processing. The operating frequency of the CNN model is 100 MHz with an improvement of 25 units compared with software implementation and power dissipation of less than 1.59 W. From the literature studies it is observed that the convolutional layer is complex for implementation, and to reduce the complexity, several methods are being used such as parallelism, intermediate storage, dedicated cores, and multiplier-less operations. The methods proposed have advantages for hardware implementation but have poor regularity and are not flexible for hardware implementation. The fully connected layer that is the last stage of CNN is also a complex structure when it has more no. of hidden layers. Hardware design optimization considering area and power is required for the implementation of the DNN model. In this work, parallelism and intermediate storage logic are used to improve latency and throughput for the implementation of a fully connected network of the DNN model. CNN model combined with the advanced encryption standard (AES) algorithm is used for data encryption. In this work, the AES algorithm that encodes and decodes the data requires an encryption key which is generated using the CNN model. To accelerate the computation process, the complexity in the CNN model has been optimized by suitable training methods [12]. Video encoding methods such as high-efficiency video coding (HEVC) require high-speed accelerators for the processing of sub-modules. Coding units are used in parallel and each of these units is pipelined to improve throughput and latency. The techniques proposed in this work are utilized for the local processing of CNN module computation [13].

Orthogonal frequency division multiplexing (OFDM) architecture using a wavelet-based approach provides faster processing time and increases the number of sub-bands, avoiding overlapping. wavelet architecture, which is a complex model implemented using sub-band modules that require arithmetic units for low-power operation, integer arithmetic, and resource-sharing methods, has been proposed in this work [14]. The optimum implementation of arithmetic units is suitable for CNN architecture implementation.

Proportional–integral–derivative (PID) controllers are developed with self-tuning capability, and the neural network model is used for self-tuning purposes. The artificial neural network (ANN) and long-short-term-memory (LSTM) models are developed considering 11 or 21 sampling data points to evaluate PID controller performance. The model developed by studies [15] and [16] reports a 92.9% improvement in the performance of PID controller tuning. Hardware implementation of the NN controller module is not carried out. In their work, they have developed a PID controller module for the X-Y table using a self-tuning algorithm and implemented it on FPGA [17], [18]. The radial basis function neural network (RBF NN) model is used for adjusting the PID controller performances and is demonstrated to be implemented on NIOS II FPGA.

The neural engineering framework (NEF) is adopted for neuromorphic computing with a spiking neural network model implemented using the large model network. PID controller is designed using the neuromorphic model and 6 degrees of freedom are analyzed for control of the robotic arm. The work reports software simulation results and has limitations for hardware performance [19].

A fully connected DNN architecture is implemented on FPGA, considering the stochastic bit stream and synchronization techniques. The proposed architecture in their work was implemented on vertex FPGA with an 82% area reduction [20]. Vaziri and Jahanrid [21] have presented a CNN accelerator based on stochastic computing with parallel processing, accelerating CNN computation. The proposed architecture in their work is reconfigurable with a low-power method and reduces area by 4.36%. The model is evaluated with the VGG 16 structure and CIFAR dataset [21]. From the studies carried out, it was identified that most of the CNN and DNN architectures are focused on area reduction for FPGA implementation. Considering hardware implementation metrics such as processing time, throughput, and latency, there is a need for high-speed architectures.

A fully connected neural network model comprises a large no. of neurons and each neuron has multiplier modules for multiplying parameters with weight vectors. The input and weight vectors are stored in memories get frequently accessed to perform arithmetic operations. Performing parallel arithmetic operations occupies more area on the FPGA platform. Parallel processing reduces latency with huge area requirements. To have a trade-off between delay and area utilization, there is a need to develop an optimum architecture that addresses area requirements and also meets the delay requirement for real-time processing. The number of redundancies among weight vectors can be estimated to minimize the no. of multiplication operations. Pipelining methods can be incorporated to improve throughput at the cost of latency. In this work, a fully connected neural network architecture with two layers is designed to optimize the arithmetic operations with trade-offs with delay. The fully connected network model is presented in this work, considering a 16:4:16 network, and can be extended to complex models.

2. FULLY CONNECTED NEURAL NETWORK

Figure 1 presents the structure of a fully connected neural network that has an input layer, hidden layers, and an output layer. The size of the input layer and output layer depends upon the features or attributes in the data set and the expected output respectively. The number of hidden layers can be varied based on the requirement [22], and the selection of the number of hidden layers is a challenging task [23]. The number of hidden layers is a trade-off between complexity and lack of learning capability by the fully connected model. A simple mathematical expression for the selection of the number of hidden layers ‘n’ is given as $n =\backslash sqrt\{(m + n)\} + 1$ [24]. Where m is the number of input nodes, k is the output nodes and l is a constant from 1 to 10. The total number of neurons ‘N’ in a given fully connected network is computed as $N = m + n + k$. The total number of weights ‘D’ that are required for a fully connected neural network with N nodes is $D = mn + nk + n + k$.

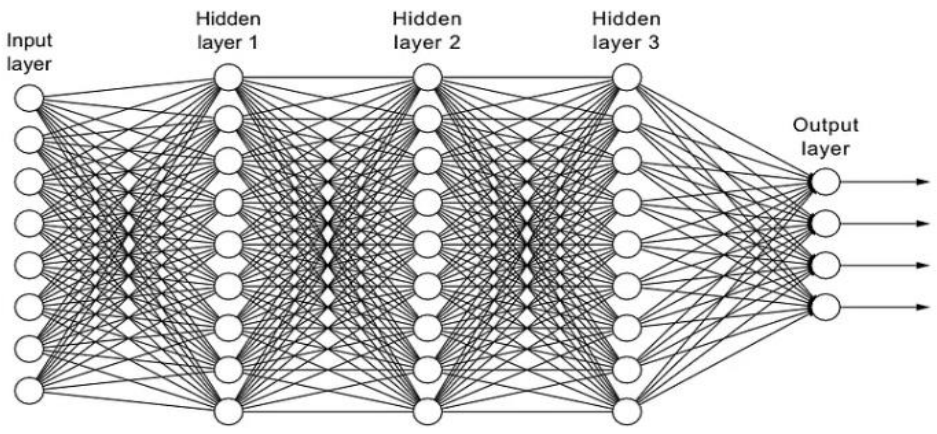

Figure 1. Fully connected neural network model

Figure 2 presents the structure of a fully connected layer that has an input layer, hidden layer, and output layer. The functionality of the FC layer is to extract the features from the feature map generated from convolutional layers and map these features to a known set of data spaces. Hardware implementation of FC structure requires input data, corresponding weights, and biases of the neurons to be stored in the memory and use of these parameters for multiply and accumulation operations. The complexity of computation depends upon the weight vector and the number of weights. If the weight vector is less than 0.1 then the multiplication will lead to a product term of smaller magnitude. Observing the weight magnitude multiplication operation can be controlled to reduce computation complexities.

With a large number of weights vectors the multiplication operations are also large in number increasing the number of arithmetic operations, storage elements, and resource utilization. Serial architecture implementation of the FC model reduces resources by reusing the arithmetic operations but increases delay.

It is required to develop hybrid methods with a trade-off between delay and area utilization. In this work, a fully connected neural network model with three layers is designed to demonstrate hardware performance optimization.

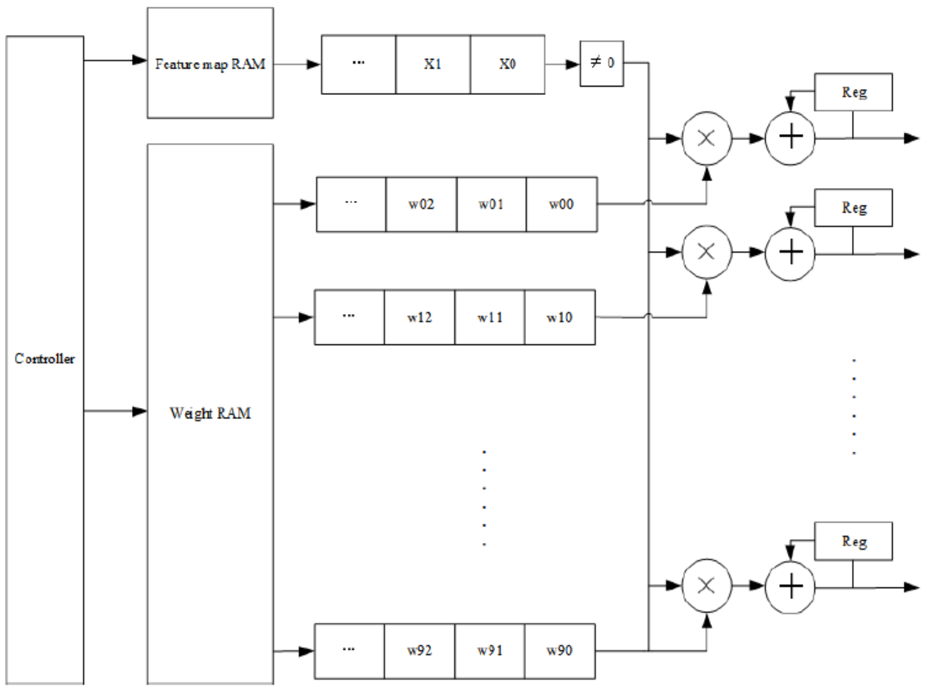

Figure 2. Architecture of FC network

3. PROPOSED STRUCTURE FOR FULLY CONNECTED MODEL

Figure 3 shows the building blocks of the generic feed-forward neural network (FFNN) architecture with 16 input vectors, represented by X1–X16 with four neurons in the hidden layer, with each neuron processing 16 inputs. Each of the neurons in the hidden layer has an array of multipliers, adders, and activation functions. The input X (16 elements) is multiplied by neuron weights represented by $W_{n,m}$ and is accumulated in the adder along with the bias bn. The output of the adder after multiplication and addition is represented by C_n.

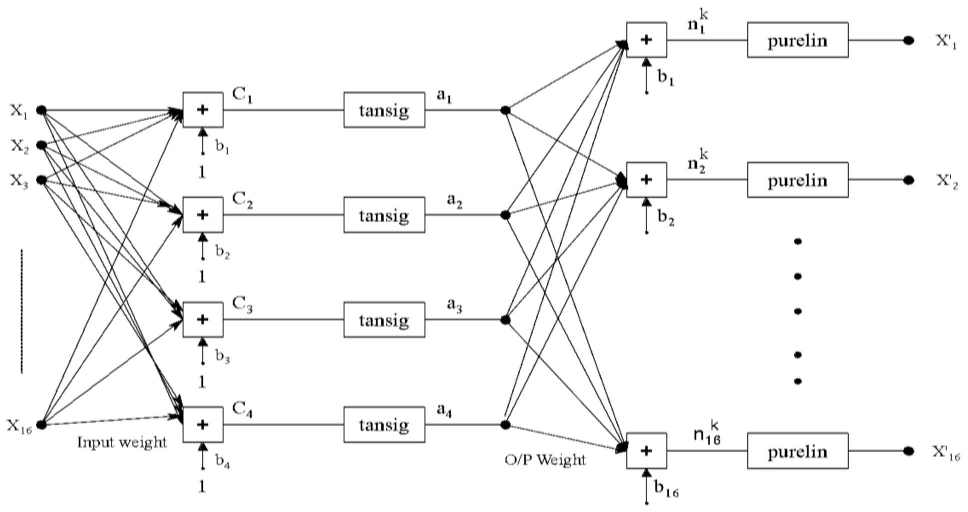

Figure 3. Two-layered (4 hidden layers and 16 output layers) FCNN architecture

The network activation function ‘tansig’ processes the intermediate output Cn to obtain the hidden layer output a_n. The output layer has 16 neurons that process the four hidden layer outputs to produce the output of the FFNN network, represented by X’. Each neuron in the output layer multiplies four inputs with four corresponding weight elements. The multiplied partial products are accumulated in the adder unit to generate intermediate output. The activation function purelin is used as the network function and the final output is generated. The input from the input layer consisting of 16 elements is provided as input for all four neurons, each input is multiplied with corresponding weights as given below, and the multiplied products are accumulated and stored in a register. C1 to C4 are the intermediate outputs generated by the hidden layer network and are expressed as in (1). The network activation function further processes the intermediate outputs represented by C1 to C4 to generate the final output an [a1 = tansig (C1), a2 = tansig (C2), a3 = tansig (C3), a4 = tansig (C4)]. The hidden layer outputs are stored in registers R1 to R4 and are represented as Y1 to Y4.

$$C1 = W1{,}1\,X1 + W1{,}2\,X2 + W1{,}3\,X3 + \ldots\ldots\ldots\ldots\ldots\ldots + W1{,}16\,X16 + b1 \tag{1a}$$

$$C2 = W2{,}1\,X1 + W2{,}2\,X2 + W2{,}3\,X3 + \ldots\ldots\ldots\ldots\ldots\ldots + W2{,}16\,X16 + b2 \tag{1b}$$

$$C3 = W3{,}1\,X1 + W3{,}2\,X2 + W3{,}3\,X3 + \ldots\ldots\ldots\ldots\ldots\ldots + W3{,}16\,X16 + b3 \tag{1c}$$

$$C4 = W4{,}1\,X1 + W4{,}2\,X2 + W4{,}3\,X3 + \ldots\ldots\ldots\ldots\ldots\ldots + W4{,}16\,X16 + b4 \tag{1d}$$

Figure 4 presents the structure of four neurons with each neuron processing 16 inputs and generating one output represented by Y. Each neuron requires 16 multipliers, a_n adder array to add 17 inputs (16 products generated by a multiplier and one bias, adder array consists of 16 adders), and one activation function. For a hidden layer consisting of 4 neurons and 16 inputs, the total number of multipliers, and adders is 64 with 4 activation functions; each of them realized using the read-only memory (ROM) structure.

Each of the input data X as shown in Figure 4 is represented by N-bits 2’s complement format with most significant bit (MSB) for sign bit and N-1 bits for magnitude. The weights of the network, after training, are also represented by N bits 2’s complement representation. Each multiplier generates 2N-1 bits and requires 2N-1 clock cycles to perform multiplication. The output of each multiplier is accumulated in the adder array structure; each adder will perform the addition of two inputs each of size 2N-1 bits to generate 2N bits. As there are four neurons in the hidden layer to generate four outputs Y1 to Y4, a structure similar to one in Figure 4 is used.

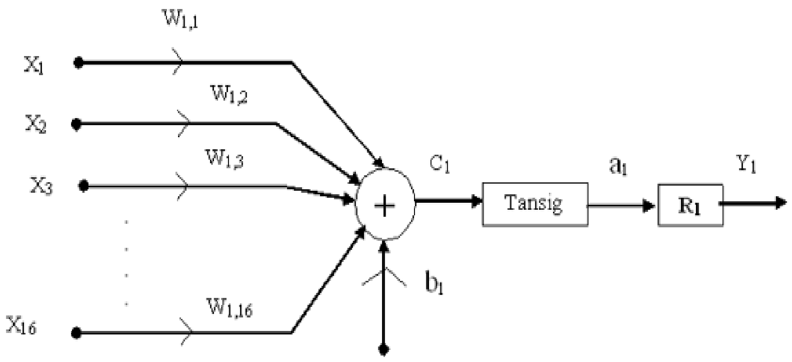

Figure 4. FCNN hidden layer architecture with arithmetic blocks

The adder array is a three-stage structure with each array consisting of 8 adders in the first stage, 4 adders in the second stage, 2 adders in the third stage, and 1 adder in the fourth stage. The final stage adder is used to add the bias. The adder array structure is represented as an (8-4-2-1-1) structure. The first stage adder outputs are represented by 2N bits, the second stage adder outputs are represented by 2N+1 bits, the third stage adder requires 2N+2 bits, and fourth stage adder requires 2N+3 bits, and the adder that adds bias requires 2N+4 bits for output representation. With N bits entering the hidden layer, the intermediate output generated requires 2N+4 bits for representation. For N=8, there is a 60% increase in bit width as there are six stages of the data path (one multiplier stage and five addition stages). The network activation function which is tansig is designed using ROM. The min-max value of tansig is +/- 1 and the digital equivalent of tansig is computed by considering 512 samples between +/-1 and each sample scaled by 256 and rounded to the nearest integer. Tansig function output for 512 possible inputs is represented by N bits in 2's complement format. The ROM depth is 512 with a width of 2N, at each memory location or address starting from "0 0000 0000" to "1 1111 1111" there is a_n N-bit tansig equivalent stored in the ROM. With the master clock represented by 'f', the multiplier is operated at clock frequency 'f', adder is operated at clock frequency $f1$, where $f1$=f/N. For the computation of every output in the hidden layer, the input data 'X' is loaded into the input register, and each neuron is designed to operate in parallel to compute the final outputs. Loading of input data requires 16 clock cycles ($f1$ is the reference clock frequency). Multiplication requires N clock cycles (multiplication is successive addition, and N is the number of bits), the adder array requires 5 clock cycle and the activation function requires 1 clock cycle. From the loading of data to the generation of output requires a total of 22+N clock cycles. The latency of the hidden layer network is 22+N clock cycles.

Figure 5 presents the structure of the output layer, comprising four inputs being processed by each of the 16 neurons to generate 16 outputs represented by X'. The four inputs represented by Y are processed by each of the neurons to generate the intermediate output C. The network activation function 'purelin' is used to process the output layer intermediate function to generate output represented by 'a' which is stored in output register R whose output is represented by X'. The total number of multipliers and adders required for realizing 16 neurons in the output layer is 64. As the activation function used is purelin, the output adders are directly stored in the output register. The input to output layer is loaded in four clock cycles, multiplication is carried out in N clock cycles, and addition (adder array represented by 2-1-1-1) requires 4 clock cycles. In total computation of one output in the output layer requires 8+N clock cycles and hence the latency is 8+N clock cycles. All 16 neurons are designed to process data in parallel. The hardware implementation of FFNN shown in Figure 3 requires 128 multipliers and 128 adders. With the hidden layer requiring an additional 4 ROMs of size 512×N, the number of adders and multipliers are dependent upon the number of neurons selected in the hidden layer and the number of neurons selected in the output layer. If the purelin activation function is used, then ROMs can be avoided. The structure in Figure 5 will be used 16 times to generate all 16 outputs of the proposed network.

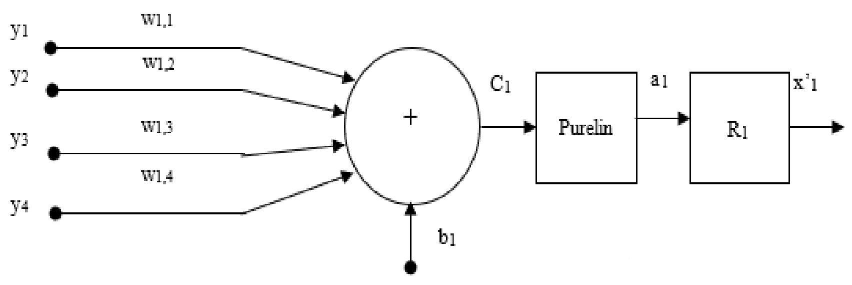

Figure 5. FCNN output layer architecture with arithmetic blocks

4. MODIFIED STRUCTURE FOR FULLY CONNECTED NETWORK

Figure 6 presents the proposed FFNN architecture, designed by optimizing the number of arithmetic elements required for hardware implementation. The architecture consists of first-stage input registers, second-stage de-multiplexers, third-stage de-multiplexers, fourth-stage weight registers, fifth-stage multipliers, sixth-stage adder array, seventh-stage bias adder unit, eight-stage ROM or LUT unit, and final stage output register. In the first stage register, there are 16 registers with a data width of N-bits, and the data is loaded from the top register to the bottom register. The output of each register is connected to the corresponding de-multiplexer in the second stage, the select signal 'C' transfers the input data to either multiplier input or to the successive register input with control input 'C' set to '0' or '1' respectively. During load operation 'C' is set to '1' the input register is activated, and 16 inputs are loaded into the register array. After 16 clocks, 'C' is set to '0' so that the contents of the register are connected to the multiplier for the multiplication process. Each neuron has 16 weights, and for four neurons there are 64 weights, which are stored in a 64-depth register shown in Figure 6. The 16 weights corresponding to each neuron are loaded into the weight register, represented by 'W'. The third stage de-multiplexer unit control input 'D' is set to '0' for 16 clock cycles to read the weights from the 64-depth register and load the 16 weights into the weight registers. After 16 clock cycles, 'D' is disabled by setting it to '0' enabling the weights to feed the multiplier input. At the 17th clock cycle, the multiplier array comprising 16 multipliers is enabled to perform multiplication.

The output of each multiplier is to be added and fed into the adder array for accumulation operations. As there are 16 products generated, 15 adders are required to accumulate these results; the adder array designed accomplishes the addition operation using 7 adders. Figure 7 is the reduced multiplier array structure for hidden layer realization of a single neuron network.

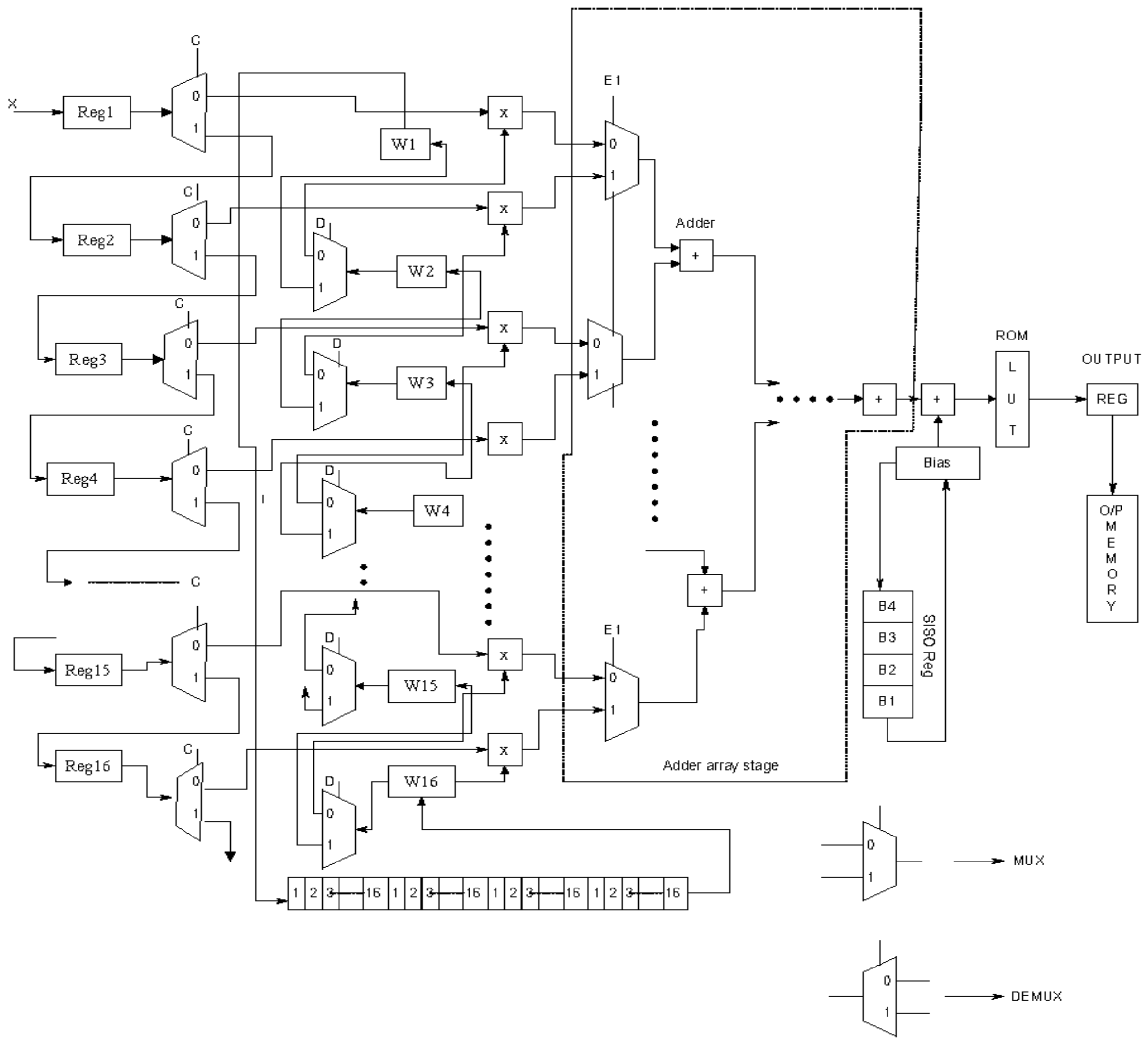

Figure 6. Pipelined architecture of single neuron of hidden unit

As discussed, 16 multipliers perform multiplication in parallel and increase the area requirement for FFNN implementation. The present work addresses this issue by developing a pipelined logic that comprises 8 multipliers instead of 16 multipliers that are activated in parallel. Each of the multipliers is driven by a serial in serial out (SISO) register of depth 2, in which the weights are stored as shown in Figure 7. The weights in each of the SISOs are loaded from the 64-depth SISO register at the bottom of the structure, which stores the network weights. During the first 16 clock cycles, the weights are loaded from the bottom SISO, and each two-stage SISO stores two alternate weight elements. The demultiplexer unit reads out the corresponding weights from the SISO and performs multiplication with the input data. The demultiplexer unit at the output routes the multiplied data into respective registers V1-V16. The multiplication operation is carried out in two clock cycles, in the first clock cycle eight products are computed (V1, V3, V5, V7, V9, V11, V13, and V15), and in the second stage, another eight products are computed (V2, V4, V6, V8, V10, V12, V14, and V16).

Figure 8 presents the array adder structure designed with 7 adders for the addition of 16 inputs. The adder array unit consists of a first-stage multiplexer with E1 control, second-stage adders (four adders), third-stage de-multiplexers and registers, fourth-stage multiplexers, fifth-stage adders (2 adders), sixth-stage de-multiplexers with registers, seventh stage multiplexers, and last stage adders (1 adder). In the first clock, ‘E1’ is set to ‘0’, and out of 16 inputs, only 8 are selected, which are added by the corresponding four adders to generate four outputs. The de-multiplexer with control signal ‘E’1’ is set to ‘0’ to store the output of the adder into registers R1, R3, R5, and R6. In the second clock cycle, ‘E1’ is set to ‘1’ and the next eight inputs are added by four adder units and the results are stored in the registers R2, R4, R6, and R8, by routing the output of adder through the de-multiplexer by setting the control input ‘E’1’ to ‘1’. The eight outputs generated and stored in R1-R8 are added in two successive clock cycles to generate four outputs. The multiplexer unit in the fourth stage, the adder unit in the fifth stage, and the de-multiplexer unit in the sixth stage are activated to perform the addition operations. The four registers R9 to R12 store the corresponding outputs. Four outputs are further added to generate one output by the seventh and eighth stage units as shown in Figure 7. The FIFO stage with a depth of 4 stores the corresponding bias values of each neuron as shown in Figure 8, which are sequentially loaded into the adder unit, and the final addition operations, are performed to add the bias with the multiplied weights. The result of the final adder stage is fed into the ROM, the input data forms the address for the ROM, and the corresponding memory content is read out which is stored in the output register.

The output register which is of depth four stores the four outputs of each neuron. The addition operation is carried out in 7 clock cycles, each stage of addition requires 2 clock cycles, and the last stage of bias addition requires an additional clock cycle. The total clock cycles (latency) required to compute the hidden layer outputs are $2N$+23 ($2N$ for multiplication operation, 7 for addition operation, 16 for input data loading). The present work reduces the hardware requirement by 75% in terms of area and the computation delay is increased by 23% as compared with direct implementation for the hidden layer architecture.

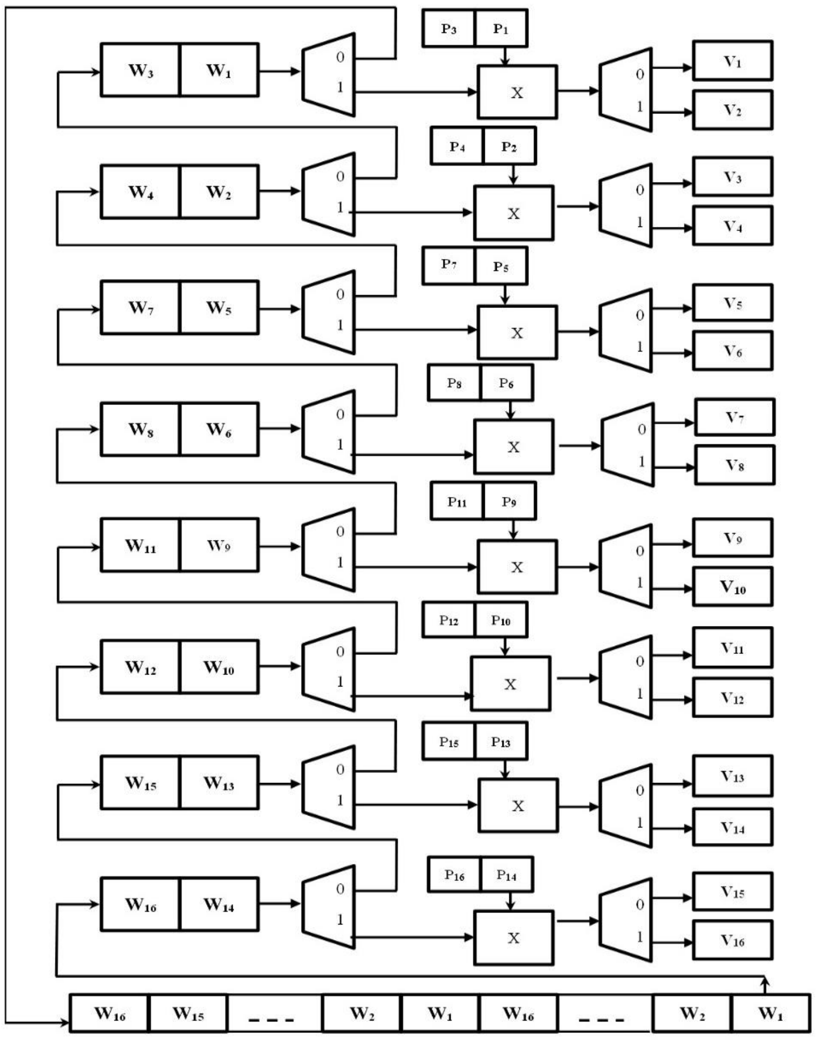

Figure 7. Reduced multiplier array structure

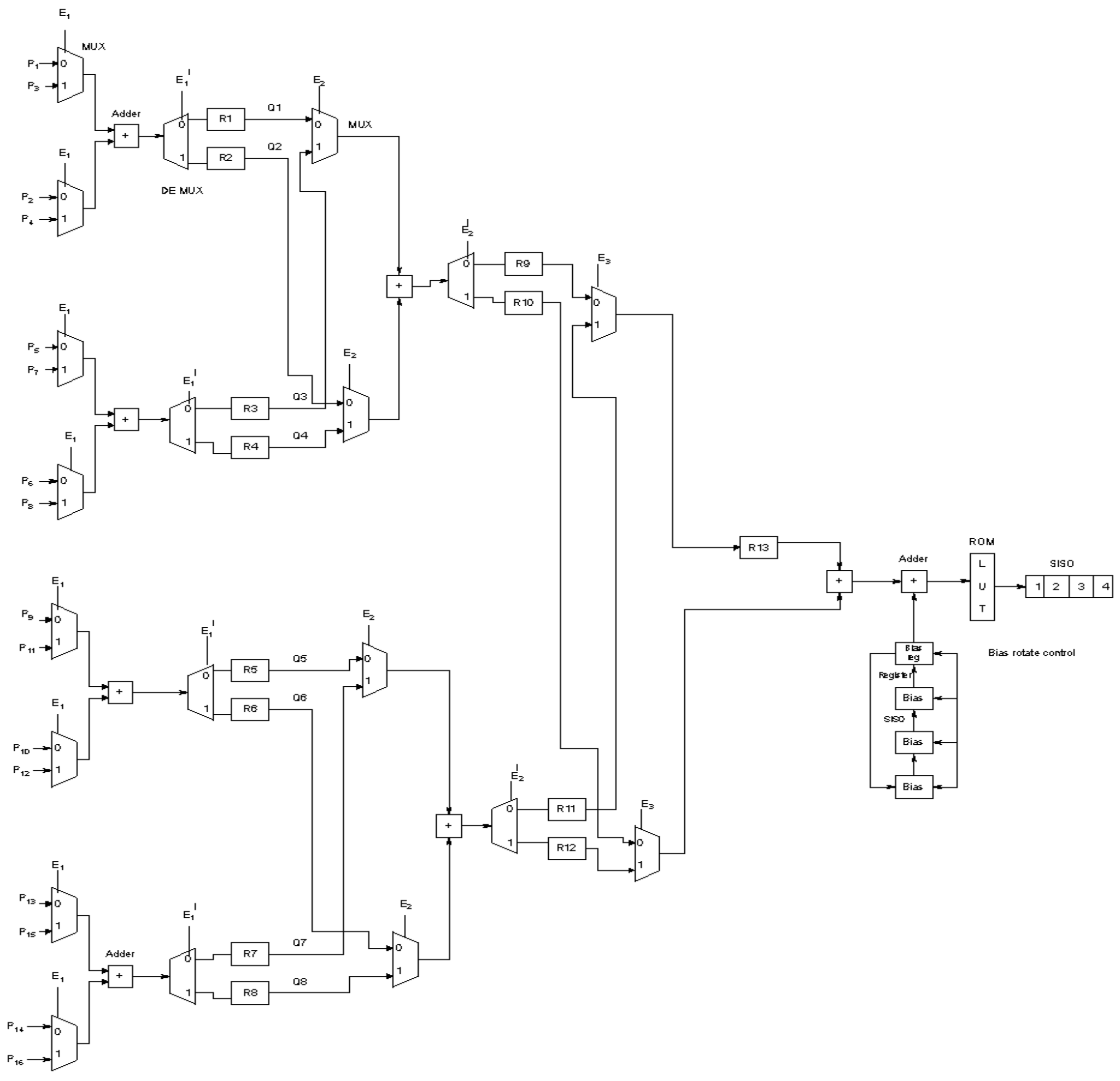

Figure 8. Pipelined adder array structure

Figure 9 presents the proposed architecture for the output layer. The first stage is the input register stage with four registers (Reg1–Reg4) and the weight registers (W1-W4), the second stage is the multiplier array, the third stage is the adder array, the fourth is the bias adder array with corresponding FIFO register with depth 16 (represented by B1 to B16), fifth stage is the LUT or ROM unit with output register; and the last stage is the output memory unit with depth 16 to store the outputs. The output layer comprises 16 neurons, with each neuron having 4 weights, these weights are stored in the weight memory as shown at the bottom of Figure 9. During the first clock cycle, the inputs, corresponding weights, and bias are loaded into the input register, weight register, and bias register. The multiplication and corresponding addition operations are performed as discussed in the previous section for the hidden layer structure. As there are 16 neurons in the hidden layer, using a single neuron to compute all 16 outputs will increase the computation delay. The work presented in this is the design of a parallel architecture that is shown in Figure 10. In the parallel architecture, two pipelined architectures are designed as shown in Figure 9, which is used to compute the 16 neuron outputs. The top architecture in Figure 10 is a single neuron structure that is designed to compute eight outputs (O1–O8), similarly, the bottom architecture computes the remaining eight neuron outputs (O9–O16). The delay in the computation of output layer outputs is increased by 33% and the area requirement is reduced by 87.5% as compared with the direct implementation structure. The latency in the computation of outputs is 2N+8 clock cycles. A detailed discussion on ANN implantation on FPGA is presented in [25]–[27]. Optimization regarding area, power, and speed is carried out by a suitable selection of an arithmetic unit, data synchronization, and memory operations. the memory operation techniques proposed in this work use pipeline operations.

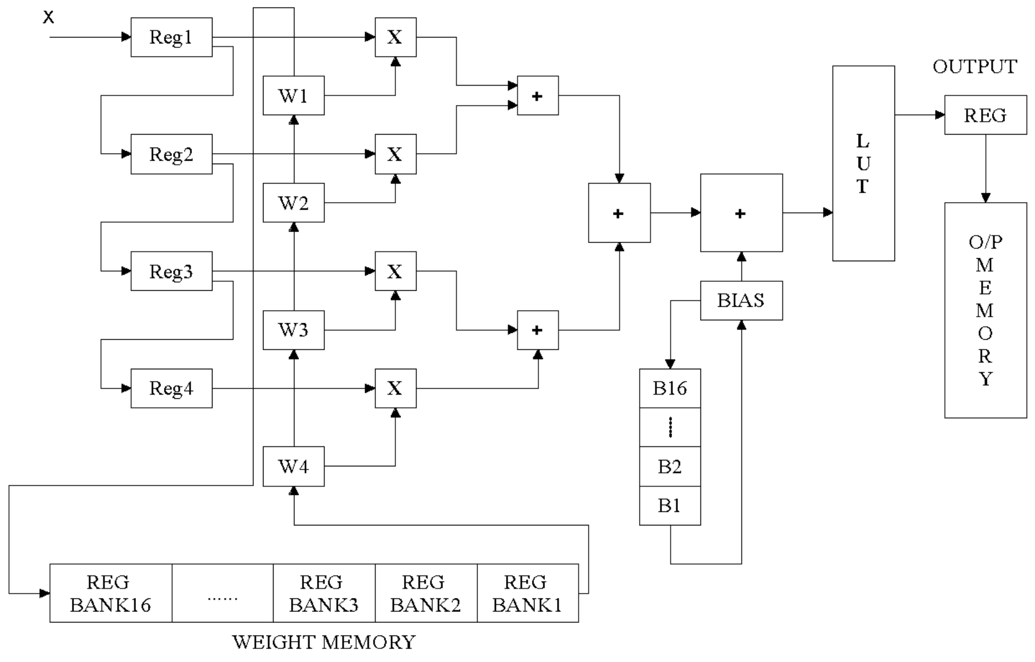

Figure 9. Pipelined architecture of single neuron of output layer unit

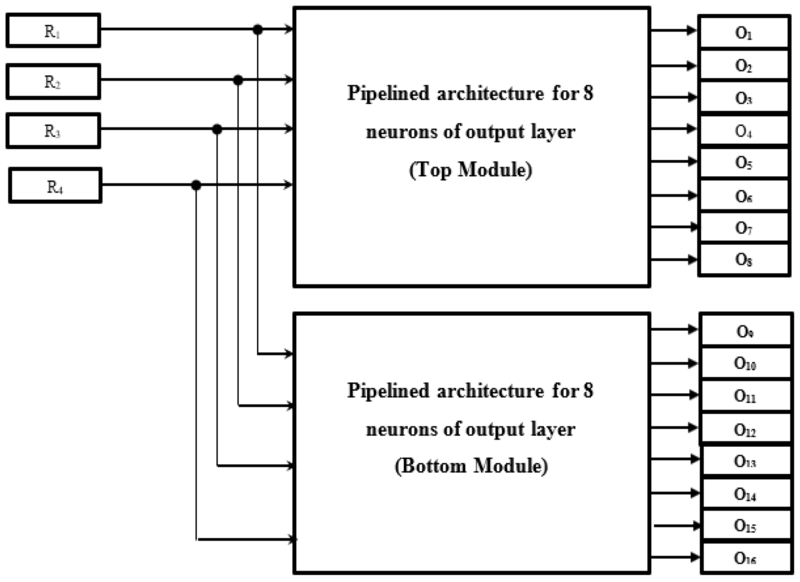
R1
R2
R3
R4
Pipelined architecture for 8 neurons of output layer (Top Module)
Pipelined architecture for 8 neurons of output layer (Bottom Module)
O1
O2
O3
O4
O5
O6
O7
O8
O9
O10
O11
O12
O13
O14
O15
O16

Figure 10. Parallel architecture of output layer unit

5. FPGA IMPLEMENTATION

It is possible to model the fully connected network (FCN) architecture shown in Figure 11 in HDL using both the direct structure form and the reduced structure with pipelined architecture. The developed HDL model is verified for its functionality with a known set of test vectors and validated against known outputs. The HDL code that works properly is made with Xilinx ISE and is meant to be put into action on the Xilinx Virtex-5 FPGA XCVLX110 device. The internal building blocks of the FCN are designed as sub-systems, considering modular logic and reusable logic. The modelled sub-blocks are verified for their functionality and logical correctness. The modelled sub-blocks are integrated into the top module, and HDL code for the test bench is developed to verify the FCN structure. The developed structure is implemented on Virtex-5 FPGA and synthesized using Xilinx ISE.

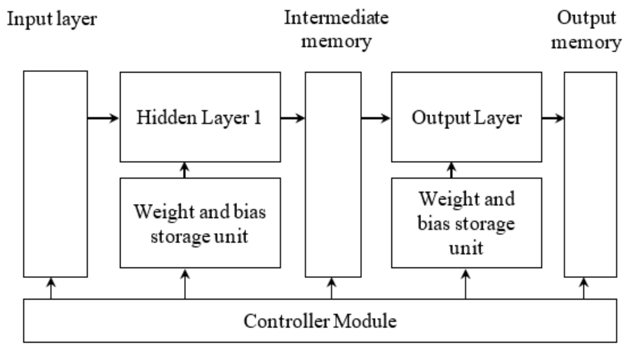

Figure 11. Top-level block diagram of FCN

The first hidden layer processes an input layer that is part of the designed FCNAs every hidden layer consists of arithmetic units for performing multiplication with weights and addition of multiplied weights the hidden layer structure is considered a data path structure. The data path operator is designed to work on 2's complement number system, and the weights of the FCN are represented using 9-bit signed numbers. Every multiplication result in an increase in product bit width to 18 bits. In this data path operation design, as the weights are scaled by 256 to represent them as integers, after every multiplication the scaling down operation is performed by discarding the 8 LSBs thus limiting the product width to be 9-bit. The adder operation increases the output width by one bit and hence the network output generates the final output of 10 bits in 2's complement representation. The data path structure for the RFFNN is implemented considering the Virtex-5 FPGA, and the resource utilization is evaluated from the synthesis report.

The RTL synthesis schematic of the hidden layer structure is obtained from the synthesis report after implementation, and it is observed for multipliers and adder stages. The output stage is designed using LUTs. The output layer as discussed in the earlier section is designed using a data path unit comprising multipliers and adders that operate on a 10-bit number system. As the input from the hidden layer is 10-bit, which is further processed by the output layer, the data path unit both multiplier and adder is designed to work with a 10-bit number system.

The overflow of data output is avoided as the scaling of input is performed to limit the input data range between +/- 64. The LUT width is also limited to 10-bit and hence the FCN structure design processes 9-bit input data and generates 10-bit output data. The HDL code is verified and implemented to identify FPGA resources and the synthesized netlist. The synthesized report is obtained, and the hardware implementation performance parameters such as the number of slices, gates, frequency of operation, and power dissipation are noted and tabulated in Tables 1 and 2.

Table 1. Comparisons between conventional FCN and modified FCN arithmetic units

	Conventional FCN		Modified FCN	
Parameters	Hidden unit	Output unit	Hidden unit	Output unit
Multipliers	64	64	8	4
Adders	64	64	8	6
Latency	N+22	2N+23	N+8	2N+8
Delay	T	T	T+23%	T+33%
Area	100%	100%	25%	12.50%

Table 2. Comparisons between conventional FCN and modified FCN FPGA implementation

FPGA implementation – Virtex 5 FPGA		
Parameter	Conventional FCN	Modified FCN
No. of slices	42364	11,235
No. of gates	45968K	734K
Max. clock speed	104 MHz	285 MHz
Power dissipation	1.3 W	0.49 W

The comparison of FCN architecture with direct implementation shows that the current work is 3 times faster than direct implementation, reduces the total area required by 73% in terms of the number of slices, and reduces power dissipation by 58%. Synthesized results show that the FCN structure operates at a frequency of 285 MHz, with 11,235 slices utilized, and power consumption of less than 0.49 W.

In the proposed method the area requirement per neuron is 561.75, which is observed to be a reduction of 39% [28]. The operating frequency of the proposed work is 285 MHz, which is a 100x improvement compared to the work reported in reference [28]. The total power dissipation for the proposed work is 0.49 W which is 38% less as compared to the reference [29] as shown in Table 3.

Table 3. Comparison with other hardware implementation models

Parameters	Area	Clock	Time	Power (W)
[20]	12964	65.3 kHz	15.3 µs	0.74
[28]	1013002	0.56 MHz	1.705 µs	0.767
[29]	144450	0.11 MHz	8.561 µs	0.798
[30]	62695	819 clock cycle	819 clock cycle	4.982
Proposed	11235	285 MHz	0.003 µs	0.49

6. CONCLUSION

This paper proposes the FPGA implementation of a modified structure for the FCN model, optimizing area, power, and speed. The proposed model uses intermediate memories and pipelining methods to improve throughput and reduce latency. The reuse logic that controls the data flow into the designed structure synchronizes the movement of data and weight vectors, reducing latency in the design. The parallel structure that has been developed provides a trade-off between resource utilization and speed. The proposed architecture is implemented on FPGA, demonstrating a maximum operating frequency of 285 MHz, and consumes less than 0.49 W power, occupying less than 12% of FPGA resources. The developed model is suitable for high-speed implementation of CNN structures and can be used as a hardware accelerator soft IP for both DNN and CNN models.

REFERENCES

[1] J. Liu, S. Tripathi, U. Kurup, and M. Shah, "Pruning algorithms to accelerate convolutional neural networks for edge applications: a survey," *arXiv preprint arXiv:2005.04275*, 2020.

[2] J. Cheng, J. Wu, C. Leng, Y. Wang, and Q. Hu, "Quantized conn: a unified approach to accelerate and compress convolutional networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 10, pp. 4730–4743, Oct. 2018, doi: 10.1109/TNNLS.2017.2774288.

[3] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[4] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, Jun. 2017, pp. 1–12, doi: 10.1145/3079856.3080246.

[5] T. Chen *et al.*, "DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 269–284, Apr. 2014, doi: 10.1145/2654822.2541967.

[6] Y. Chen *et al.*, "DaDianNao: a machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2014, pp. 609–622, doi: 10.1109/MICRO.2014.58.

[7] J. Yu *et al.*, "Instruction driven cross-layer CNN accelerator for fast detection on FPGA," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 11, no. 3, pp. 1–23, Sep. 2018, doi: 10.1145/3283452.

[8] C. Zhang and V. Prasanna, "Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Feb. 2017, pp. 35–44, doi: 10.1145/3020078.3021727.

[9] F. Yan, Z. Zhang, Y. Liu, and J. Liu, "Design of convolutional neural network processor based on FPGA resource multiplexing architecture," *Sensors*, vol. 22, no. 16, Aug. 2022, doi: 10.3390/s22165967.

[10] B. Xiaokang, "Design and implementation of FPGA-based hardware acceleration circuit for convolutional neural network image classification algorithm," Nanjing, China, 2019.

[11] L. Binfeng, "Design and FPGA implementation of a convolutional neural network acceleration circuit," Nanjing, China, 2019.

[12] I. Negabi, S. Ait El Asri, S. El Adib, and N. Raissouni, "Convolutional neural network based key generation for security of data through encryption with advanced encryption standard," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, no. 3, pp. 2589–2599, Jun. 2023, doi: 10.11591/ijece.v13i3.pp2589-2599.

[13] E. A. Abdessamad, N. Bahri, A. Mansouri, N. Masmoud, and A. Ali, "Area and power efficient VLSI architecture of mode decision in integer motion estimation for HEVC video coding standard," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 4, pp. 2469–2480, Aug. 2019, doi: 10.11591/ijece.v9i4.pp2469-2480.

[14] S. Q. Hadi, A. A. Jafaar, B. M. Alameri, and S. A. Malallah, "Field programmable gate array implementation of multiwavelet transform based orthogonal frequency division multiplexing system," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 12, no. 5, pp. 5136–5144, Oct. 2022, doi: 10.11591/ijece.v12i5.pp5136-5144.

[15] Y.-S. Lee and D.-W. Jang, "Optimization of neural network-based self-tuning PID controllers for second order mechanical systems," *Applied Sciences*, vol. 11, no. 17, Aug. 2021, doi: 10.3390/app11178002.

[16] J. Berner, K. Soltesz, T. Hägglund, and K. J. Åström, "An experimental comparison of PID autotuners," *Control Engineering Practice*, vol. 73, pp. 124–133, Apr. 2018, doi: 10.1016/j.conengprac.2018.01.006.

[17] Y.-S. Kung, H. Than, and T.-Y. Chuang, "FPGA-realization of a self-tuning PID controller for X–Y table with RBF neural network identification," *Microsystem Technologies*, vol. 24, no. 1, pp. 243–253, Jan. 2018, doi: 10.1007/s00542-016-3248-x.

[18] D. Xu, W. Yan, and N. Ji, "RBF neural network based adaptive constrained PID control of a solid oxide fuel cell," in *2016 Chinese Control and Decision Conference (CCDC)*, May 2016, pp. 3986–3991, doi: 10.1109/CCDC.2016.7531681.

[19] Y. Zaidel, A. Shalumov, A. Volinski, L. Supic, and E. Ezra Tsur, “Neuromorphic NEF-based inverse kinematics and PID control,” *Frontiers in Neurorobotics*, vol. 15, Feb. 2021, doi: 10.3389/fnbot.2021.631159.

[20] M. Nobari and H. Jahanirad, “FPGA-based implementation of deep neural network using stochastic computing,” *Applied Soft Computing*, vol. 137, Apr. 2023, doi: 10.1016/j.asoc.2023.110166.

[21] M. Vaziri and H. Jahanirad, “Low-Cost and hardware efficient implementation of pooling layers for stochastic CNN accelerators,” in *2022 12th International Conference on Computer and Knowledge Engineering (ICCKE)*, Nov. 2022, pp. 24–29, doi: 10.1109/ICCKE57176.2022.9960139.

[22] R. HECHT-NIELSEN, “Theory of the backpropagation neural network,” in *Neural Networks for Perception*, Elsevier, 1992, pp. 65–93.

[23] D. T. Larose and C. D. Larose, *Discovering knowledge in data: an introduction to data mining*. John Wiley & Sons, 2014.

[24] Guang-Bin Huang, “Learning capability and storage capacity of two-hidden-layer feedforward networks,” *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 274–281, Mar. 2003, doi: 10.1109/TNN.2003.809401.

[25] N. Perryman, C. Wilson, and A. George, “Evaluation of Xilinx versal architecture for next-gen edge computing in space,” in *2023 IEEE Aerospace Conference*, Mar. 2023, pp. 1–11, doi: 10.1109/AERO55745.2023.10115906.

[26] A. K. Jameil and H. Al-Raweshidy, “Efficient CNN architecture on FPGA using high level module for healthcare devices,” *IEEE Access*, vol. 10, pp. 60486–60495, 2022, doi: 10.1109/ACCESS.2022.3180829.

[27] R. O. Hassan and H. Mostafa, “Implementation of deep neural networks on FPGA-CPU platform using Xilinx SDSOC,” *Analog Integrated Circuits and Signal Processing*, vol. 106, no. 2, pp. 399–408, Feb. 2021, doi: 10.1007/s10470-020-01638-5.

[28] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, “VLSI implementation of deep neural network using integral stochastic computing,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2688–2699, Oct. 2017, doi: 10.1109/TVLSI.2017.2654298.

[29] B. Li, M. H. Najafi, and D. J. Lilja, “An FPGA implementation of a restricted Boltzmann machine classifier using stochastic bit streams,” in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Jul. 2015, pp. 68–69, doi: 10.1109/ASAP.2015.7245709.

[30] J. Park and W. Sung, “FPGA based implementation of deep neural networks using on-chip memory only,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016, pp. 1011–1015, doi: 10.1109/ICASSP.2016.7471828.

BIOGRAPHIES OF AUTHORS

Pottipati Dileep Kumar Reddy currently working as a research scholar at YSR Engineering College of Yogi Vemana University in the Electronics and Communication Engineering Department. He received his M.Tech degree in VLSI design from SRM University India, in 2014. In 2012, he completed his B.Tech degree in electronics and communication engineering From JNTUA, Anantapur, India through the 4-year program. His research interests include quantum computing, image processing, neural networks, and VLSI Design. He can be contacted at email: dkr.pottipati@gmail.com.

Kota Venkata Ramanaiah is a professor in the Electronics and Communication Engineering Department. Currently working as Dean Faculty of Engineering, of Y.S.R Engineering College of Yogi Vemana University, Proddatur. He has more than 28 years of experience in teaching. His areas of research interests include low-power VLSI design architectures, image processing, and neural network-based image compression. He published papers in more than 115 International and National journals. He received the Adarsh Vidya Saraswati Rashtriya Puraskar National Award. Under his guidance, nine students received Ph.D from Yogi Vemana University Kadapa, JNTUA Ananthapuram, and JNTUK Kakinada. He obtained his Ph.D from JNT University of Hyderabad in 2009. M.Tech from JNTU College of Engineering Kukatpally Hyderabad in 1998 and B.E from KBNCE Gulbarga University Gulbarga, 1992. He can be contacted at email: ramanaiahkota@gmail.com.