The Journal of Supercomputing
https://doi.org/10.1007/s11227-021-03849-7

FPGA acceleration on a multi-layer perceptron neural network for digit recognition

Isaac Westby[1] · Xiaokun Yang[1] · Tao Liu[2] · Hailu Xu[3]

Accepted: 27 April 2021

Abstract
This paper proposes field-programmable gate array (FPGA) acceleration on a scalable multi-layer perceptron (MLP) neural network for classifying handwritten digits. First, an investigation to the network architectures is conducted to find the optimal FPGA design corresponding to different classification rates. As a case study, then a specific single-hidden-layer MLP network is implemented with an eight-stage pipelined structure on Xilinx Ultrascale FPGA. It mainly contains a timing controller designed by Verilog Hardware Description Language (HDL) and sigmoid neurons integrated by Xilinx IPs. Finally, experimental results show a greater than ×10 speedup compared with prior implementations. The proposed FPGA architecture is expandable to other specifications on different accuracy (up to 95.82%) and hardware cost.

Keywords Digit recognition · Field-programmable gate array (FPGA) · Multi-layer perceptron (MLP) · Modified national institute of standards and technology (MNIST) · Neural network (NN)

1 Introduction

Neural networks (NNs) have become an indispensable technique for a wide range of applications such as image classification, natural language processing, speech recognition, and many more [1–4]. Specifically in the era of edge computing, to limit the complexity of NN in the power-constrained and latency-critical scenarios is a big challenge. Many researches thus focused on mapping NN models onto

✉ Xiaokun Yang
YangXia@uhcl.edu
https://sceweb.sce.uhcl.edu/xiaokun/

[1] University of Houston Clear Lake, 2700 Bay Area Blvd, Houston, TX 77058, USA

[2] Lawrence Technological University, 21000 West Ten Mile Road, Southfield, MI 48075, USA

[3] California State University Long Beach, 1250 Bellflower Blvd., Long Beach, CA 90840, USA

Published online: 13 May 2021

field-programmable gate array (FPGA) with the benefits of high parallelism and energy efficiency [5, 6].

The implementation of NN on FPGA is much harder than that on CPUs and GPUs. The development framework like Caffe and Tensorflow for CPU and GPU is absent for FPGA [7, 8]. Most of existing FPGA designs of NN are based on software-hardware co-design platforms such as Xilinx Zynq FPGA and Intel HARPv2, where a CPU host and an FPGA in the same chip or package are integrated [9–11]. For example, the Zynq system-on-chip (SoC) family contains software programmability of an ARM-based processor with the hardware programmability of an FPGA [30, 31]. The flexibility to use the programming system (PS) can significantly reduce the design work on Hardware Description Language (HDL); nonetheless, the utilization of the ARM core is very costly in terms of FPGA slice count and delay.

Another research direction to the FPGA accelerator on NNs is based on the high-level synthesis (HLS) tools like Xilinx Vivado [12–14]. HLS allows users to build a network by using high level language like C or C++, and then convert the design into register-transfer level (RTL). Comparing with adoptions of early generations of HLS, the latest applications to HLS tools have made significant progress to the design on FPGA. However, optimized manual design in RTL is still necessary particularly to complex designs on SoCs [15].

Therefore, in this paper, a multi-layer perceptron (MLP) neural network, including RTL design to the controller and an integration with multiple Vivado IPs, is presented to perform a practical application of handwritten digits recognition. The database of Modified National Institute of Standards and Technology (MNIST), which was developed by Yann LeCun, Corinna Cortes and Christopher Burges, is used to build the MLP network and evaluate the accuracy of the NN models [28]. Specifically, the contributions are below.

- This paper first conducts an investigation to several design architectures of the MLP neural network related to different quality results. To show a case study, a single-hidden-layer MLP network is implemented with an eight-stage pipelined structure on Xilinx Ultrascale FPGA. Though a specific design is demonstrated in this paper, the proposed design structure is expandable and scalable to different accuracy constraints and hardware specifications.
- Experimental results show that our proposed work can achieve a latency of 1.55 microseconds per digit recognition with an accuracy of 93.25%. To the best of our knowledge, this is the minimum inference latency compared to those of existing works. Additionally, the FPGA slice count and energy consumption are evaluated as well.

The remainder of this paper is organized as follows: Sect. 2 introduces the related works to the application of handwritten digit recognition with FPGA, and Sect. 3 presents the background of MLP network. In Sect. 4, the design architecture of the MLP is discussed. The implementation of the NN is further described in Sect. 5. In Sect. 6, the FPGA design performance is evaluated in terms of latency, slice count, energy consumption. Finally, Sect. 7 concludes this paper.

2 Related works

The practical application of handwritten digits recognition has been performed by numerous researches to overcome challenges such as reducing the computational complexity [16–18] and increasing the calcification correctness [19–21]. However, this paper focuses on finding the minimum latency corresponding to different quality bounds of classifying images.

The computational speed of handwritten numeral digit recognition has been greatly improved by using hardware accelerator in the past few years. For example, two HLS FPGA designs on LeNet-5 CNN were presented in [22] and [23], achieving a latency of 3.58 ms and 3.2 ms with accuracy of 98.64% and 96%, respectively. The implementations on LeNet-5 CNN contained three convolutional layers, two average pooling layers, and two fully connected layers, in addition to the input and output layer.

Logic design on FPGA can further reduce the latency of NNs with the benefit of computational parallelism. As an example, a deep neural network (DNN) was implemented on Xilinx Zync-7020 FPGA in [24]. By optimizing the scheduling of input memory and weight memory, the proposed work can reach a latency of 640 us with 100 MHz clock. Additionally, in [25], a CNN network was built end-to-end using a reconfigurable IP core and then implemented on an lntel Cyclone10 FPGA. Experimental result showed that the design spent 17.6 us to recognize a handwritten digital picture with an accuracy rate of 97.57%.

To reduce the design complexity on NNs, authors of [26] presented a Super-Skinny CNN (SS-CNN) with 39,541 parameters and only three layers in addition to input and output layer. The implementation on a Cyclone IVE FPGA achieved a latency of 2.2 seconds including both training (55,000 images) and inference (10,000 images). Another two-layer MLP network, containing only one input layer and one output layer, was presented by the same authors [27]. Using a 25 MHz clock frequency, the design took 3.8 seconds to train 55,000 images and recognize 10,000 handwritten digits.

In this paper, a scalable MLP network architecture is proposed, aiming to significantly reduce the latency by slightly decreasing the classification rate. The MNIST data base is used to evaluate different design structures corresponding to different quality constrains. Though a case study on a single-hidden-layer design is finally implemented on FPGA, the proposed network architecture is expandable to meet different specifications on latency, accuracy, and hardware cost.

Specifically, the network structure is expandable with more neural layers to achieve a higher accuracy for digit classification. Further, the design on the sigmoid neuron can be instantiated into multiple neurons in each layer to improve the classification rate. Additionally, the design architecture can be expanded into different stages of pipelining structures by trading off latency with FPGA cost in terms of slice count and power consumption. Specifically, a 16-stage structure can achieve a significant reduction to slice count and power consumption on FPGA compared with using an eight-state design architecture. However, the computation latency using a 16-stage design would be larger than that of the

implementation using an eight-stage structure. From hardware designers' perspective, generally the more resource cost on FPGA, the higher accuracy and speed could be achieved to the design on an MLP network.

3 Fundamental theorem of MLP neural network

This section discusses the fundamental knowledge of MLP networks, including the sigmoid neurons and the data set used for training the network. Finally, the way for finding the optimal design structure is depicted.

3.1 Sigmoid neurons

Due to the benefit that a sigmoid neuron is much smoother than the step functional output from perception, a sigmoid neural network is performed in our work [29]. Generally, the output for the sigmoid neuron can be written as

$$sigmoid\_neuron\_output = \frac{1}{1 + exp\left(-\sum_{i=1}^{n} w_i \times x_i - b\right)} \tag{1}$$

where w_i denotes the weight corresponding to the input x_i, and b represents the bias. The parameter n demotes the number of input neurons. By making small changes to the weights and biases of the sigmoid neuron, small changes to the output would occur, eventually converging on a 'correct' or most effective set of weights and biases.

From the hardware perspective, the design on each sigmoid neuron needs the hardware designs on multiplication, addition, subtraction, accumulation, exponential, and reciprocal.

3.2 Finding the design structure of the MLP network

In this paper, The MNIST handwritten digit data set is used to train the MLP network [28]. This data set has 60,000 handwritten digits with corresponding labels that can be used to train the network. There is then a separate set of 10,000 different handwritten digits with labels that can be used to estimate the network. Once the method for finding the accuracy of the network with a trained set of weights and biases has been established, the goal is to decide on a network design.

First, the input layer would require 784 neurons due to the fact that the MNIST digit images to train the network are 28×28 input pixels in size. The second thing that would need to remain static is the fact there would be 10 output neurons because there are 10 possible outputs (0-9) that would converge to a value of around 1. This leaves the number of hidden layers, as well as the number of neurons in each hidden layer as the values that can be adjusted.

Once a network design (number of hidden layers and number of nodes in each layer) has been decided, the values of epochs used, mini_batch_size, and learning rate can be tweaked in order to find the most accurate set of weights and biases.

These values are static with values of epoch = 30, mini_batch_size = 10, and learning rate = 3.0, when comparing different network designs.

4 Proposed design architecture

This section discusses the design methodology of choosing a network. First, we start an investigation to several single-hidden-layer and two-hidden-layer networks corresponding to different quality specifications. In what follows, one of the network structures is chosen as a case study to the design on FPGA acceleration.

4.1 Comparing different networks

As stated above, various network designs are considered to find a network that would be able to be implemented with relative simplicity but still attain a high accuracy. When testing different networks, the epoch, mini_batch size, and learning rate all remain the same, but the number of hidden layers and neurons in each layer is changed. First, the one-hidden layer networks are tested: [784, 50, 10], [784, 30, 10], [784, 20, 10], [784, 16, 10], [784, 12, 10], [784, 10, 10], [784, 8, 10], [784, 5, 10], [784, 4, 10], [784, 3, 10], [784, 2, 10], [784, 1, 10], where the first number is the input layer number, the second number is the number of neurons in the hidden layer, and the third number is the output layer neurons.

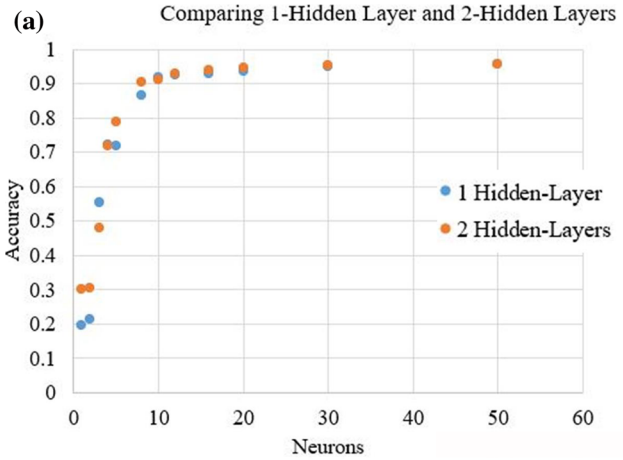
When using two-hidden layers, the following networks are tested: [784, 50, 50, 10], [784, 30, 30, 10], [784, 20, 20, 10], [784, 16, 16, 10], [784, 12, 12, 10], [784, 10, 10, 10], [784, 8, 8, 10], [784, 5, 5, 10], [784, 4, 4, 10], [784, 3, 3, 10], [784, 2, 2, 10], [784, 1, 1, 10], where the second and third numbers denote the value of neurons used in the first and second hidden layers.

The accuracy of the one-hidden layer (blue dots) and two-hidden layer (orange dots) networks is summarized in Fig. 1a. It can be observed that as the number of neurons in the hidden layer(s) increases, the accuracy increases exponentially and asymptotically approaches a value of 1.0. In Fig. 1b, c, it shows that for networks that have 10 neurons or greater in the hidden-layer(s), the difference in accuracy is especially small. This is taken into consideration when deciding the final network design.

4.2 Adjusting Epoch, mini_batch_size, and learning rate

In what follows, three parameters—epoch, mini_batch_size, and learning rate are adjusted when training the network. These values are important to how quickly the network's weights and biases can be trained to the highest attainable digit recognition accuracy.

The mini_batch is used to set the number size of the batches that are used to train the weights and biases. When setting learning rate = 3.0, the result in Fig. 2a from changing the mini_batch shows that the highest digit recognition accuracy is achieved by the networks of mini_batch = 30 after 10 epochs.



(b)

Neurons	Accuracy
50	0.9568
30	0.9489
20	0.9366
16	0.9277
12	0.9264
10	0.9167
8	0.8639
5	0.7176
4	0.7213
3	0.552
2	0.2132
1	0.1961

(c)

Hidden Layer 1	Hidden Layer 2	Accuracy
50	50	0.9582
30	30	0.952
20	20	0.9453
16	16	0.9379
12	12	0.9296
10	10	0.9121
8	8	0.9032
5	5	0.7882
4	4	0.7194
3	3	0.4781
2	2	0.3055
1	1	0.3019

(a) A graph comparing networks with a single hidden layer and two hidden layers.

(b) The data corresponding to single hidden layer NNs.

(c) The data corresponding to two hidden layer NNs.

Fig. 1 Comparing accuracy of MLP networks with a single hidden layer and two hidden layers (color figure online)

When setting mini_batch = 30, the accuracy result from changing learning rate is shown in Fig. 2b. It can be observed that the highest digit recognition accuracy occurs when learning rate is 3.0. For these results, there is virtually no discernible difference in the performance of training for learning rates between 2.0 and 10.

The final test is conducted in Fig. 2c using four different combinations across 60 epochs. The combinations used here are, mini_batch = 10 & learning rate = 3.0, mini_batch = 10 & learning rate = 0.1, mini_batch = 30 & learning rate = 3.0, and finally mini_batch = 30 & learning rate = 3.0. By testing these varying parameters, it is able to figure out which combination will lead to the most accurate set of weights and biases in the shortest amount of time. Since the biases and weights are only going to be generated one time, then used in the network after that point, this study trains the network for many more epoch, so that the accuracy can be as high as possible.

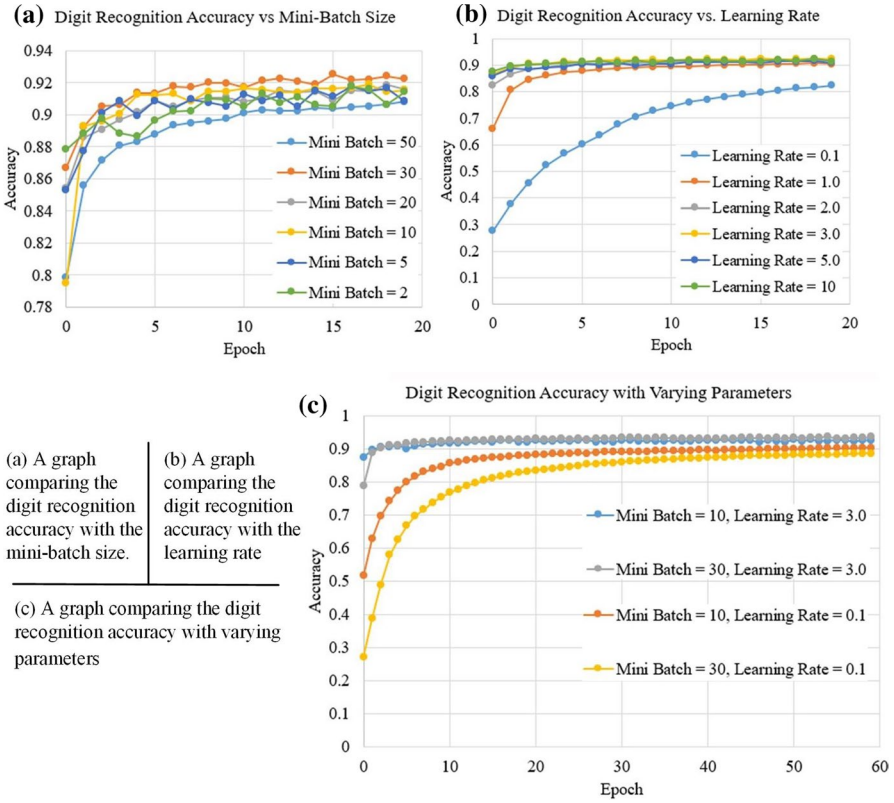


Fig. 2 A graph comparing the digit recognition accuracy with **a** the mini-batch size, **b** learning rate, and **c** the combination

4.3 Final network design

In conclusion, networks with two hidden layers perform better than networks with single hidden layer, but this increase is very limited. Second, at least 10 neurons in the hidden layer are needed to reach a classification rate over 90%. Therefore, a single-hidden-layer MLP network is chosen as a case study, and further 12 sigmoid neurons in the hidden layer are instantiated. Notice that the design structure is expandable to achieve higher accuracy with more hardware cost, or reversely, to trade the design accuracy for less hardware consumption.

Figure 3 shows the network structure, containing 784 input neurons, 12 hidden neurons, and 10 output neurons. It results in $784 \times 12 = 9408$ weights and 12 biases in the hidden layer, and $12 \times 10 = 120$ weights and 10 biases in the output layer. Totally there would be thus 9550 parameters stored into FPGA buffers.

Once a network has been chosen, the weights and biases are generated by running the python program [29]. For this task, the techniques with Epoch = 60,

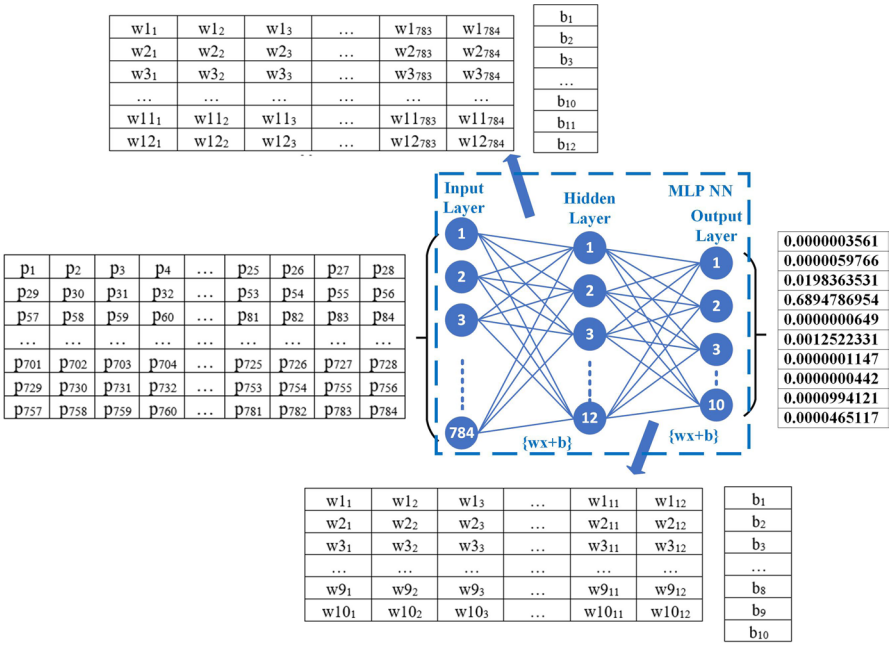


Fig. 3 A figure showing the final network design: 784 input pixels to the input layer, 784×12 weights and 12 biases to the hidden layer, and 12×10 weights and 10 biases to the output layer

mini_batch_size = 30, and learning rate = 3.0 are used. There is a random nature to these numbers, so the system is run multiple times until a set of weights and biases are obtained which achieves the highest accuracy of 93.25%.

5 Implementation

In this section, the ML algorithm is broken down by hardware operations. Further, different architectural designs to the algorithm are analyzed and evaluated by hardware implementations.

5.1 Non-pipelined hardware design architecture

Generally, the NN can be divided into the hidden layer neurons and the output layer neurons. First, the equation for the output of the hidden layer neuron j can be rewritten:

$$hidden\_neuron\_output(j) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{784} w_{i,j} \times x_i - b_j\right)} \tag{2}$$

where i specifies the number of input pixels (ranges from 1 to 784) and j denotes the number of the hidden neurons (ranges from 1 to 12). This allows us to break the problem of finding the output of a hidden layer neuron down into two parts. The first part is to multiply all 784 input pixels (x_i) by their corresponding weights ($w_{i,j}$), then sum those values into one result which can be formulated as $\sum_{i=1}^{784} w_{i,j} \times x_i$. A visual representation of this process is shown in Fig. 4(a). Assuming that each hardware operation has a latency of one clock cycle, in the first clock cycle, 784 multipliers are needed to multiply the input pixels by their correct weights, and then 10 cycles of cascading adders to sum the results up into one final value.

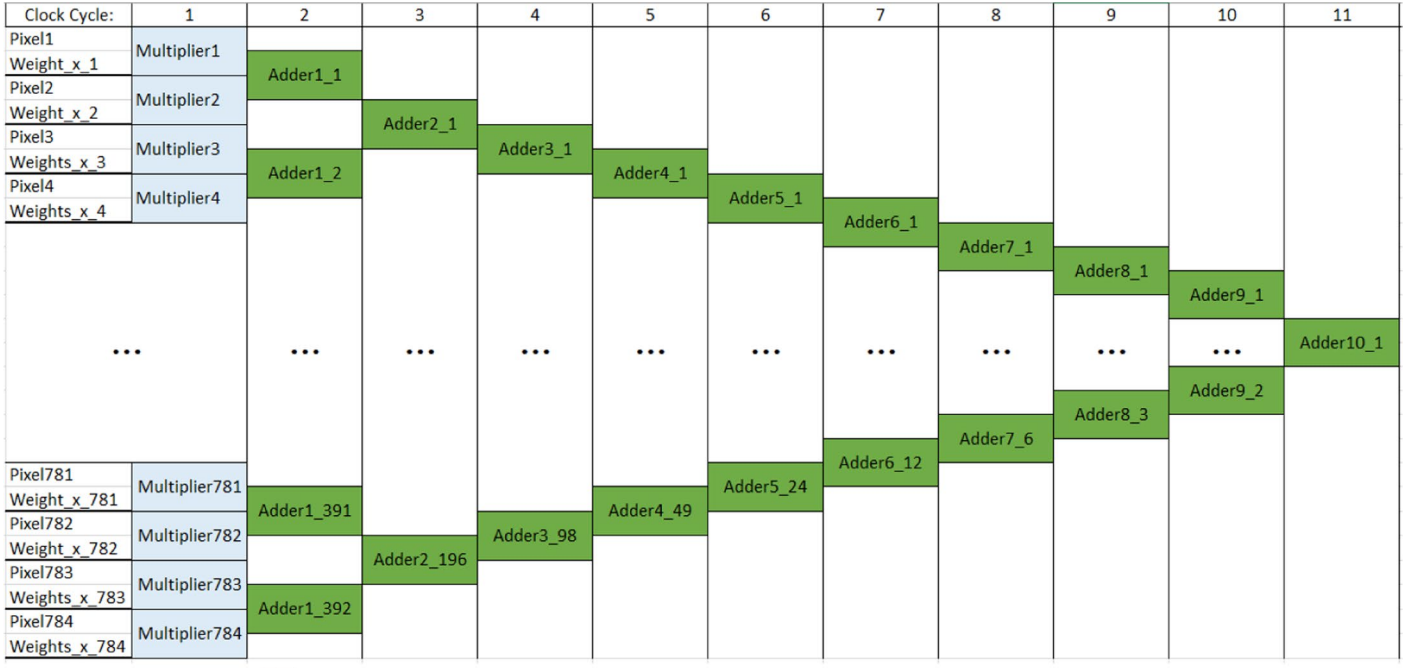
The second part is to take that summed value, denoted as $S_j = \sum_{i=1}^{784} w_{i,j} \times x_i$), and plug it into the sigmoid function formulated as $\frac{1}{1+\exp(-S_j - b_j)}$. This involves five different operations: (1) taking the negative value of the summed result, (2) subtracting bias, (3) taking exponential to that value, (4) adding a value of 1, and finally (5) taking the reciprocal. As shown in Fig. 4b, this stage takes another 5 clock cycles by assuming a latency of one cycle for every operation. Putting the two stages together, it spends a total of 16 clock cycles in order to find the output of a hidden layer neuron.

Then, the output layer neurons are considered. Specifically, the hardware involves taking the outputs of the hidden-layer neurons, then plugging those values into the output layers below:

$$final\_neuron\_output(k) = \frac{1}{1 + \exp\left(-\sum_{j=1}^{12} w_{j,k} \times hidden\_neuron\_output_j - b_k\right)} \tag{3}$$

where k ranges from 1 to 10 as the number of output neurons. This output layer is implemented in a similar manner as the hidden layer. A visual representation is shown in Fig. 5. In this stage, it takes only 4 cycles of adders to sum the multiplication results, allowing the entire process of calculating the result of the final layer neurons to take only 10 clock cycles.

In summary, the design described above contains the operations needed for one neuron in the hidden layer and one neuron in the output layer. The hardware design on a non-pipelined network thus can be implemented for every neuron in the hidden layer, as well as every neuron in the output layer by repeating the operations 12 and 10 times, respectively. Specifically, one neuron in the hidden layer spends 784 floating-point multiplyers, and one neuron in the output layer spends 12 floating-point multiplyers; hence, the non-pipelined architecture will use $784 \times 12 + 12 \times 10 = 9528$ floating-point multiplyers. Similarly, 9528 adders are needed including 9,408 adders in the hidden layer and 120 adders in the output layer. Though the latency is only 26 cycles, the non-pipelined architecture is very costly on hardware and not affordable by most advanced FPGA boards.



(a) A visual representation of the multipliers and adders with the hidden neuron

Clock Cycle:	12	13	14	15	16	17
Adder10_1_result	Negative	sub bias	exp	add 1.0	Recip	Neuron_final_result

(b) A visual representation of the sigmod with the hiden neuron

Fig. 4 A visual representation of the operations of a single hidden layer neuron

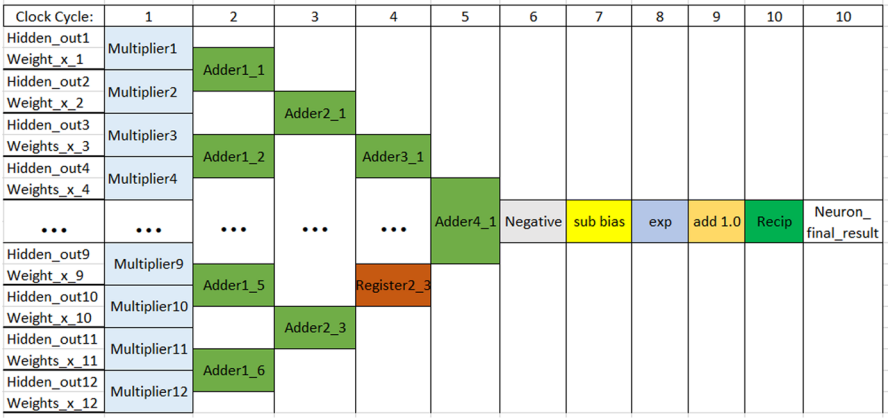


Fig. 5 A visual representation of the operations of a single output layer neuron

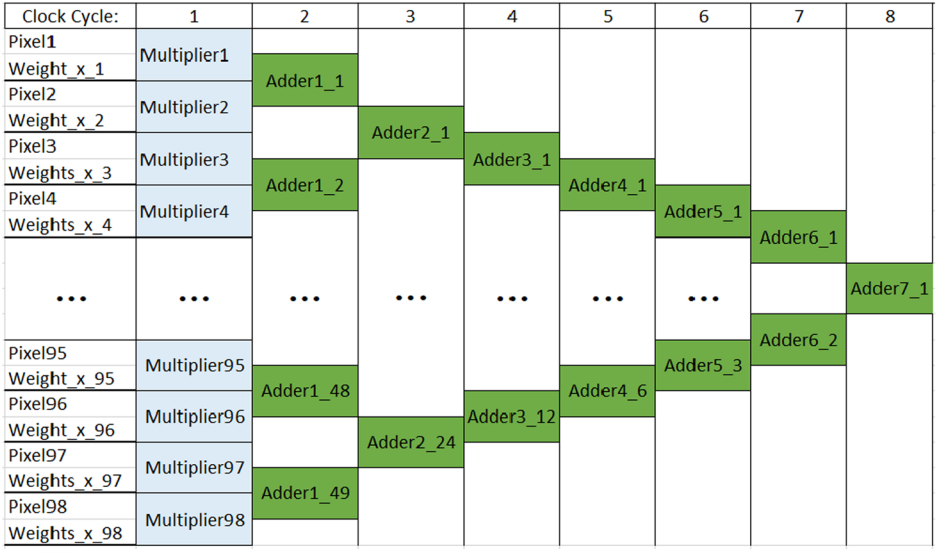
5.2 Pipelined hardware design architecture

To reduce the hardware cost, a fully pipelined design architecture is further presented. Instead of duplicating the design neuron 12 times in the hidden layer and 10 times in the output layer, the design on the single neuron can be reused over clock cycles. The comparison between resource cost of the non-pipelined design and pipelined design is shown in Table 1. It can be observed that even though the non-pipelined approach is nearly twice the speed of the pipelined approach, it uses nearly 12 times the resources.

In order to further reduce the hardware cost, the multiplication of the first layer weights and inputs within the hidden layer neuron can be continuously broken down. For example, an eight-stage pipelined structure for the hidden layer neurons is shown in Fig. 6a. This design keeps the same pipelined structure for the output level neurons, but for the hidden layer neurons, instead of 784 multipliers followed by adders it will only need 98. Hence, it is able to process 98 inputs with each iteration. So, eight iterations will be executed in order to process all the inputs for a neuron. As the results from each eighth of the multiplications come through, they are added together in an accumulator.

Table 1 A comparison of the resources needed for a pipelined, non-pipelined design, and pipelined with 98 multipliers designs

Design structures	Latency (cycles)	Hardware cost				
		MUL	ADD	SUB	EXP	REC
Non-pipelined	26	9528	9528	44	22	22
Fully pipelined	48	796	796	4	2	2
8-statge pipelined	129	110	110	4	2	2



(a) A visual representation of the 98 multipliers and adders with the hidden neuron

Clock Cycle:	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Adder7_1_result	Accum1	Accum2	Accum3	Accum4	Accum5	Accum6	Accum7	Accum8	Negative	sub bias	exp	add 1.0	Recip	Neuron_final_result

(b) A visual representation of the accumulation and sigmod with the hiden neuron

Fig. 6 A visual representation of the operations of a single hidden layer neuron

An idea of how the accumulator fits into the output of the hidden layer neurons is shown in Fig. 6b. Specifically, it can be seen that at cycle nine the first value from the multiplier-adder comes in. For eight cycles, these values are accumulated to get one final value for all 784 inputs. Once this value has been found, the result is just sent into the same sequence, of taking the negative, subtracting bias, taking exponential, adding 1.0, then taking the reciprocal. This entire process ends up taking 21 cycles to find the result.

A comparison of the resource utilization and latency between pipelined and non-pipelined designs is shown in Table 1. It can be observed that the eight-stage pipelined structure consumes more clock cycles but only spends 13.8% the number of hardware components as the fully-pipelined design, and 1.15% the number of hardware as the non-pipelined design.

The proposed design structure can be expanded to different pipelined levels with different specifications to resource cost and computational speed. The higher of the pipelined levels, the less hardware resource is needed but more clock cycles will be taken.

5.3 Design and simulation

In what follows, the case study of the eight-stage structure of the MLP network is designed by Verilog HDL and integrated with multiple Vivado IPs. Basically, the design structure can be divided into two stages, as shown in Fig. 7. Stage one is used for finding the output of the hidden layer neurons, and stage two is used for finding the output of the output layer neurons.

In stage one, the 98 multipliers and cascading adders are executed eight times, feeding the result each time into the accumulator. This allows the network to process all 784 input pixels and their corresponding weights. Once the accumulator has accumulated all eight summed values, the results propagate sequentially through the rest of stage one. Once all 12 of the stage one outputs have been calculated, stage two processes the results of the output layer neurons sequentially.

The design on the timing controller is verified by Mentor Graphic Mondelsim, and the final functionality of the digit recognition is tested by using Xilinx Vivado. As an example shown in Fig. 8, the final results of the network can be obtained by looking at the value of the signal 'final_neuron_result_value', when 'cnt6' spanning over hexadecimal '0xa - 0x13'. These results come in such that the value of 'final_neuron_result_value' at 'cnt6 = 0xa' corresponds to detection

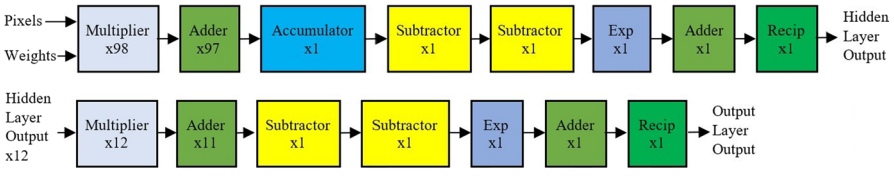


Fig. 7 A visual representation of all the components of the design put together

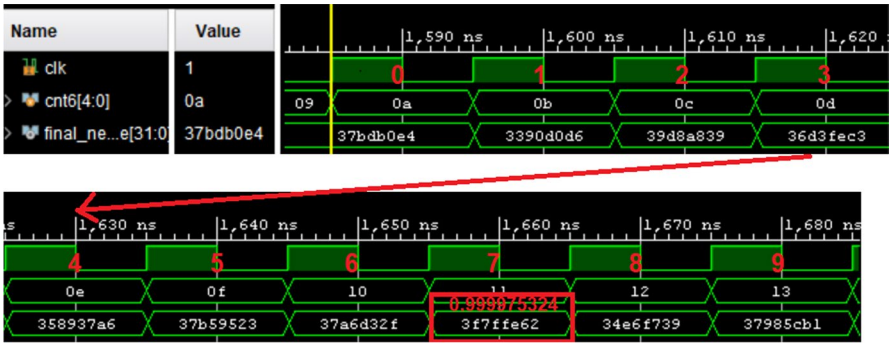


Fig. 8 Vivado waveform showing the final results of the network

of a handwritten digit '0', at 'cnt6 = 0xb' corresponds to a detection of a digit '1', etc. Thus, the final detection results of the network show that the network has strongly detected a digit '7', with that output being equal to '0.999975324', and all other outputs being nearly '0'.

6 Experimental results

In this section, the performance evaluation is further discussed in terms of speed, slice count, and energy cost. Xilinx Vivado is applied as the synthesis tool with FPGA part xcku035-sfva784-1LV-I.

6.1 Execution time on FPGA

For the FPGA execution, the latency of a single digit recognition is 1.55 us running on a 100 MHz clock. An equivalent design on Matlab is performed in order to compare the execution time between hardware and software. Generally, the software latency is taken on a personal computer that has the following processor, Intel(R) Core(TM) i7-7500U CPU @ 2.70 GHz, 2904 MHz, 2 Core(s), 4 Logical Processor(s).

Table 2 A comparison of the execution times in software and hardware

Comparison	Time for 10,000 images (s)	Time per image (us)	Speedup	
Software (Matlab)				
Fastest	0.6261488	62.61	n/a	
Longest	1.9718928	197.19		
Hardware (FPGA)	n/a	1.55	40.4	Over fastest
			127.2	Over longest

Table 3 A further breakdown of the utilization results

Resource	CLB LUTs	CLB registers
digitrec (top)	44,668	14,274
acc_add	1880	656
mul_98	30,100	11,255
mul_12	3473	1330
u0_neuron_finish	1575	311
u1_neuron_finish	1543	311
all blk_mem (total)	6097	411

Table 4 The energy consumption on FPGA

	Dynamic energy				Static energy	Total
	Signals	Logic	DSP			
Energy (mJ)	0.35	0.49	0.03	0.00	0.01	0.88
Percentage (%)	40	56	3	0	1	100

As shown in Table 2, the execution time in software varies every time running it with the longest total execution time taken into consideration, and the fastest taken into consideration. The execution time for the software implementation is based upon 10,000 input images, so the total execution time is divided by 10,000 in order to find the execution time per image. The speedup is found by a simple formula $Speedup_A = \frac{(ExecutionTime)_B}{(ExecutionTime)_A}$.

Specifically shown in the fourth column, the longest measured time per image in software is 197.19 us, and the fastest is 62.61 us. This leads to a speedup in hardware of 127.2× over the longest software run, and a speedup of 40.4× over the fastest software run. When further taking the difference of the clock frequency (100 MHz in hardware vs. 2.7 GHz in software) into consideration, the implementation of a solution in hardware allows significant speedup over a software implementation.

6.2 Resource cost on FPGA

After synthesis, the hardware utilization is shown in Table 3. From the overall summary in the second row, it shows that the design on the MLP network mainly spends 44,668 LUTs and 14,274 FFs. In the further breakdown of the utilization from the third to the eighth row, it can be observed that most of the resource utilization comes from the multiplication modules, which includes 98 single-precision floating-point multipliers in the hidden layer (mul_98) and additional 12 in the output layer (mul_12).

6.3 Energy consumption on FPGA

In what follows, the energy dissipation is summarized in Table 4. The total on-chip energy is 0.88 uJ, including 0.87 dynamic energy and 0.01 static energy. This gives a breakdown of 99% dynamic power and 1% static power consumption. Further, in the second and third column, it can be observed that the high switching activities on signals and logic take most of the dynamic energy (96%).

6.4 Comparison to related works

Finally, the comparison to prior works is emphasized in Table 5. The accuracy shown in the third column is based on the 10,000 test images from the MNIST data set. Our case study to the eight-stage MLP network achieves an accuracy of 93.25%, between those of existing works. By using our proposed MLP architectures, the classification rate can reach 95.82% with a two-hidden-layer network and 50 neurons in each hidden layer.

Then, the hardware cost is summarized in the fourth and fifth columns in terms of LUTs and FFs. For all the implementations except for [23], the slice count of LUTs and FFs is similar. The results of [23] in the fourth row do not include many hardware functions like sigmoid neurons; thus the slice number is less than those of others. The resource cost is highly dependent on the pipelined levels with our proposed work. For example, the hardware cost on multiplications and additions can be reduced by half with a 16-stage pipelined design. In other words, the slice count can be reduced by half for 2× level of the pipelined design. It is a trade off between hardware cost and recognition accuracy.

Focusing on FPGA acceleration, the latency to recognize handwritten digits is mainly compared in the last column. It can be observed that our case study achieves the highest speed by using a single-hidden-layer MLP network and logic-only implementation. Specifically, the latency of [22, 23] is very high due to the limitation of timing and speed optimization using an HLS design. Compared to the DNN and CNN designs in the fifth and sixth rows, our proposed work achieves 411× and 11× speedup, respectively. Finally, the execution time

Table 5 Comparison to related works

Comparison	FPGA—Clock (MHz)	Accuracy (%)	Hardware Cost		Latency
			LUTs	FFs	
[22] LeNet-5 CNN-HLS	Xilinx ZCU102-100	98.64	32,589	33,585	3.58 ms
[23] LeNet-5 CNN	Xilinx Zync 7Z020-100	90–96	18,426	8,264	3.2 ms
[24] DNN	Xilinx Zync 7Z020-100	94.67	38,899	40,534	637 us
[25] CNN	Intel Cyclone10-150	97.57	12,588	48,765	17.6 us
[26] SS-CNN	Intel Cyclone IVE-30	98.8	98,000		220 us
[27] MLP NN	Intel Cyclone IVE-25	89	34,000		380 us
Our case study	Xilinx Ultrascale-100	93.25	44,668	14,274	1.55 us

for our proposed work is greatly less than [27] and [26], because the weights and biases for our work are trained beforehand, while the total execution time for [26, 27] also includes training operations.

In summary, our proposed logic-only design on a single-hidden-layer network is able to detect handwritten digits with a significant speedup, and maintain a 93.25% accuracy and similar utilization to exiting works.

7 Conclusion

This paper proposes a scalable MLP network for the recognition of handwritten digits. As a case study, a single-hidden-layer design structure is implemented on FPGA, achieving 93% classification rate with just 12 hidden-layer neurons and 9,550 weight and bias parameters. The logic-only design and off-board training parameters enable to significantly decrease the complexity of the final network implementation and provide a low latency when classifying images. Experimental results show a $> 10\times$ acceleration over existing works.

References

1. Benidis K, Rangapuram S, Flunkert V (2020) Neural forecasting: introduction and literature overview. arXiv:2004.10240
2. Ismayilov G, Topcuoglu HR (2020) Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing. Future Gen Comput Syst 102:307–322
3. Molchanov P, Mallya A, Tyree S, Frosio I, Kautz J (2019) Importance estimation for neural network pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp 11264–11272
4. Son C, Park S, Lee J, Paik J (2019) Deep learning-based number detection and recognition for gas meter reading. IEIE Trans Smart Process Comput 8(5):367–372
5. Shawahna A, Sait SM, El-Maleh A (2019) FPGA-based accelerators of deep learning networks for learning and classification: a review. IEEE Access 7:7823–7859
6. Nurvitadhi E, Kwon D, Jafari A et al (2019) Evaluating and enhancing Intel Stratix 10 FPGAs for persistent real-time AI. In: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, p 119
7. Guo K, Zeng S, Yu J, Wang Y, Yang H (2017) A survey of FPGA based neural network accelerator. arXiv:1712.08934
8. Albert R et al (2019) Survey and benchmarking of machine learning accelerators. arXiv:1908.11348v1
9. Gschwend D (2020) ZynqNet: an FPGA-accelerated embedded convolutional neural network. arXiv:2005.06892
10. Gao C, Braun S, Kiselev I, Anumula J, Delbruck T, Liu S (2019) Real-time speech recognition for IoT purpose using a delta recurrent neural network accelerator. In: 2019 IEEE International Symposium on Circuits and Systems (ISCAS), pp 1–5
11. Vaca K, Gajjar A, Yang X (2019) Real-time automatic music transcription (AMT) with Zync FPGA. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp 378–384
12. Li Q, Zhang X, Xiong J, Hwu W, Chen D (2019) Implementing neural machine translation with bi-directional GRU and attention mechanism on FPGAs using HLS. In: Proceedings of the 24th Asia and South Pacific Design Automation Conference, pp 693–698

13. Rongshi D, Yongming T (2019) Accelerator implementation of Lenet-5 convolution neural network based on FPGA with HLS. 2019 3rd International Conference on Circuits, System and Simulation (ICCSS). Nanjing, China, pp 64–67
14. Zhang Q, Cao J, Zhang Y, Zhang S, Zhang Q, Yu D (2019) FPGA implementation of quantized convolutional neural networks. In: 2019 IEEE 19th International Conference on Communication Technology (ICCT), pp 1605–1610
15. Cong J, Liu B, Neuendorffer S et al (2011) High-level synthesis for FPGAs: from prototyping to deployment. IEEE Trans Comput Aided Des Integr Circuits Syst 30(4):473–491
16. Akgun OC, Mei J (2020) An energy efficient time-mode digit classification neural network implementation. Philos Trans R Soc A 37820190163
17. Xiang Y et al (2019) Hardware implementation of energy efficient deep learning neural network based on nanoscale flash computing array. Adv Mater Technol 4(5):1800720
18. Ma Y, Guo J, Wei W (2019) An exceedingly fast model for low resolution handwritten digit string recognition. In: IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), pp 282–288
19. Abdulrazzaq MB, Saeed JN (2019) A comparison of three classification algorithms for handwritten digit recognition. In: International Conference on Advanced Science and Engineering (ICOASE), pp 58–63
20. Ahlawat S, Choudhary A, Nayyar A, Singh S, Yoon B (2020) Improved handwritten digit recognition using convolutional neural networks (CNN). Sensors 20:3344
21. Ali S, Shaukat Z, Azeem M et al (2019) An efficient and improved scheme for handwritten digit recognition based on convolutional neural network. SN Appl Sci 1:1125
22. Cho M, Kim Y (2020) Implementation of data-optimized FPGA-based accelerator for convolutional neural network. In: International Conference on Electronics, Information, and Communication (ICEIC), pp 1–2
23. Madadum H, Becerikli Y (2019) FPGA-based optimized convolutional neural network framework for handwritten digit recognition. In: 1st International Informatics and Software Engineering Conference (UBMYK), pp 1–6
24. Tsai T-H, Ho Y-C, Sheu M-H (2019) Implementation of FPGA-based accelerator for deep neural networks. In: 2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)
25. Xiao R, Shi J, Zhang C (2020) FPGA implementation of CNN for handwritten digit recognition. In: IEEE Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), pp 1128–1133
26. Si J, Yfantis E, Harris SL (2019) A SS-CNN on an FPGA for handwritten digit recognition. In: 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), pp 88–93
27. Si J, Harris SL (2018) Handwritten digit recognition system on an FPGA. In: 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), pp 402–407
28. LeCun Y, Cortes C, Burges C (2010) MNIST handwritten digit database
29. Nielsen M (2019) Neural Networks and Deep Learning. Determination Press, neuralnetworksanddeeplearning.com, Neural Networks and Deep Learning
30. Zynq-7000 SoC Data Sheet: Overview. V1.11.1, Xilinx, July 2 (2018)
31. Zynq-7000 SoC Technical Reference Manual. V1.12.2, Xilinx, July (2018)