IMPLEMENTATION of DEEP NEURAL NETWORK ACCELERATOR USING FPGA

S. Pradeep[1], P. Sridevi[2], N. Seshu Kumar[3], P. Jyostna[4]

[1] *P. G Scholar , Electronics and Communication Engineering ,Miracle Educational Society Group of Institutions ,Andhra Pradesh ,India*
[2] *Assistant Professor, Electronics and Communication Engineering ,Miracle Educational Society Group of Institutions ,Andhra Pradesh ,India*

[3] *Associate Professor, Electronics and Communication Engineering ,Miracle Educational Society Group of Institutions ,Andhra Pradesh ,India*

[4] *Assistant Professor, Electronics and Communication Engineering ,Miracle Educational Society Group of Institutions ,Andhra Pradesh ,India*

ABSTRACT

Low-precision arithmetic operations to accelerate deep-learning applications on field- programmable gate arrays (FPGAs) have been studied extensively, because they offer the potential to save silicon area. However, these benefits come at the cost of a decrease in accuracy. Neural network-based methods for image processing are becoming widely used in practical applications. Modern neural networks are computationally expensive and require specialized hardware, such as graphics processing units. Since such hardware is not always available in real life applications, there is a compelling need for the design of neural networks for mobile devices. Mobile neural networks typically have reduced number of parameters and require a relatively small number of arithmetic operations. However, they usually still are executed at the software level and use floating-point calculations. The use of mobile networks without further optimization may not provide sufficient performance when high processing speed is required, for example, in real-time video processing (30 frames per second). In this study, we suggest optimizations to speed up computations in order to efficiently use already trained neural networks on a mobile device.

Keyword : - *Field programmable gate arrays, Neural network hardware, Fixed-point arithmetic, 2D convolution, Digital arithmetic.*

1. INTRODUCTION

Convolutional neural networks (CNNs) have been widely adopted in recent computer vision applications due to their superior prediction capabilities, with researchers gravitating toward larger networks with higher computational complexity and memory requirements. Field-programmable gate array (FPGA) implementations have demonstrated improved latency and power efficiency compared with central processing unit (CPU) and graphics processing unit(GPU) technologies.

In contrast to CPU/GPU technologies, they allow customized data paths, enabling improved parallelism and less data movement. This design flexibility poses an opportunity to optimize system performance through custom hardware tailored to the application. Optimizations via compression, quantization, and neural network layer explorations have been utilized to reduce complexity and boost performance. In particular, quantizing inference networks to very low precision, such as constraining weight representations to binary or ternary values, both reduces memory requirements and enables multiplications to be replaced with the exclusive NOR operation. However, the disadvantage of extreme quantization is that the networks typically incur significant accuracy degradation for very low precisions, especially for complex problems.
One limitation with traditional fixed-point quantization is that it has a uniform distribution. However, it has been demonstrated that a non-uniform distribution with the same number of potential weights can result in

better accuracy, provided the distribution appropriately matches the desired full-precision neural network weight distribution. It follows that reducing precision may not be the best method to save silicon area. Reconfigurable constant coefficient multipliers (RCCMs) are an alternative method to reduce FPGA resources through time multiplexing and resource sharing. They are usually realized using additions; subtractions, bit shifts, and multiplexers, meaning that multiplies are implemented without requiring digital signal processing (DSP) blocks on an FPGA. However, RCCMs are restricted to a given number of target coefficients; this has restricted their use to DSP application domains including digital filtering and linear transformations.

Following properties of many modern high-performing CNN architectures make their hardware implementation feasible:

- High regularity: all commonly used layers have similar structure (Conv3x3, Conv1x1, Max-Pooling);
- Typically small size of convolutional filters: 3 × 3;
- ReLU activation function (comparison of the value with zero): easier to compute compared to previously used Sigmoid and Tanh functions.

Due to high regularity, size of the network can be easily varied, for example, by changing the number of convolutional blocks. In the case of field programmable gate arrays (FPGAs),thisallows to program the network on different types of FPGAs, providing different processing speed.

For example, implementation of higher number of convolutional blocks on an FPGA can directly lead to a speed-up in processing. Related direction in neural network research considers adapting them for the use on mobile devices. However, they are still executed at the software level and use floating-point calculations. For some tasks such as real-time video analysis that requires processing of 30 frames per second mobile networks still can be not fast enough without further optimization

In order to use an already trained neural network in a mobile device, a set of optimizations can be used to speed up computation. There exist a number of approaches to do so, including weight compression or computation using low-bit data representations. Since hardware requirements for neural networks keep increasing, there is a need for design and development of specialized hardware block for the use in ASIC and FPGA. The speed up can be achieved by following:

- Hardware implementation of the convolution operation, which is faster than software convolution;
- using fixed-point arithmetic instead of floating-point calculations;
- reducing the network size while preserving the performance;
- Modifying the structure of network architecture while preserving the same level of performance and decreasing the footprint of the hardware implementation and saved weights.

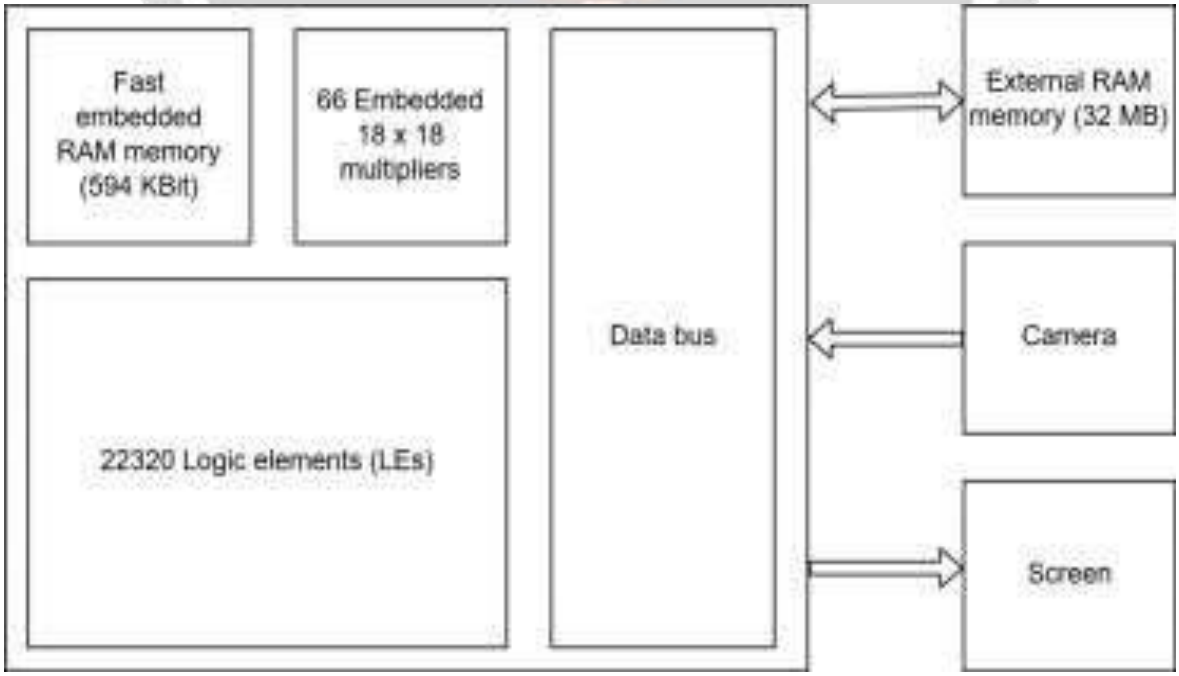

Figure 1. DE0-Nano development board and external devices

2. BACKGROUND

2.1 Convolutional Neural Networks

CNNs are biologically inspired networks that process input tensors (multidimensional arrays) of (w, h, d) dimensions in a transnationally invariant manner [1]. Typically, w and h are spatial dimensions and d is the number of channels. Processing operations, such as convolution, pooling, and activation functions, are applied in a series of layers, each of which transforms the input tensors from dimensions (w, h, d) → (w, h, d).

Pooling layers are downsamplers of 2-D images. Max pooling layers provide a spatial maximum function, which divides an input image into small sub tiles of a given window size and then replaces these with the maximum value in the subtile. An average pooling layer is similar; however, it finds the average in the subtile rather than the maximum.

2.2 Implementation requirements

To demonstrate our approach, we implement a solution for the problem of recognizing handwritten digits received from a camera in real time. The results are displayed on an electronic LED screen. The minimal speed of digit recognition should exceed 30 FPS, that is, a neural network should be able to process a single image in 33ms. The resulting hardware implementation should be ready for transfer to separate custom VLSI device for mass production.

3. PROPOSED DESIGN

3.1 CNN architecture design

We searched for building blocks that efficiently map to the logic fabric of an FPGA. Our designs are optimized for the latest Xilinx FPGAs. For these devices, a slice provides either six-input LUTs with a single output (used in Topology A) or two five-input LUTs with shared inputs (used in Topology B). As such, we designed our base topologies to ensure the MUX'es fit into the same LUTs that are required for the adders. Fig. 2 shows the two base topologies used to build the RCCM units in this article. Each of these consists of an adder with at least one input being the output of a MUX. However, to ensure the topology fits into a single LUT, this comes at a cost of less select inputs. Through our experimentation, we found that the chosen topologies were sufficient for creating RCCMs with a desired coefficient set to simplify the training process.

All contemporary FPGA devices are similar in that their logic blocks consist of LUTs followed by a fast carry chain. Hence, a simple adder can be extended by MUXes with no additional cost for certain MUX sizes when carefully selected for the target device. The detailed slice mappings of our base topologies are shown in Fig. 3, highlighting how our design consumes exactly the same silicon area as a traditional ripplecarry adder with the same word size on that FPGA (which would only implement the XOR gate to complete the carry logic to a full adder).

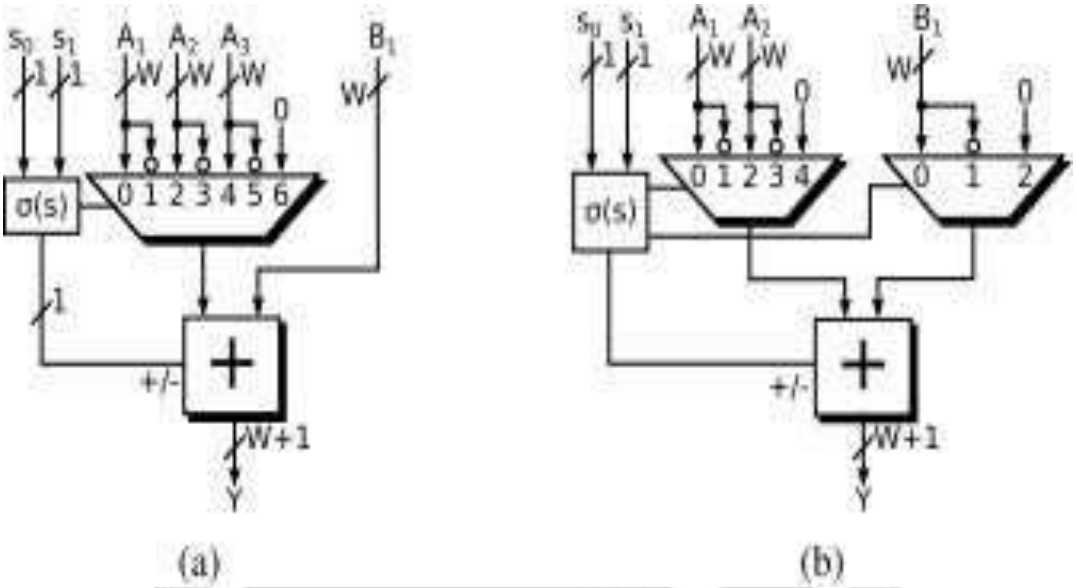

Figure 2. Base topologies used to build reconfigurable multipliers.

a) Topology A.
b) Topology B.

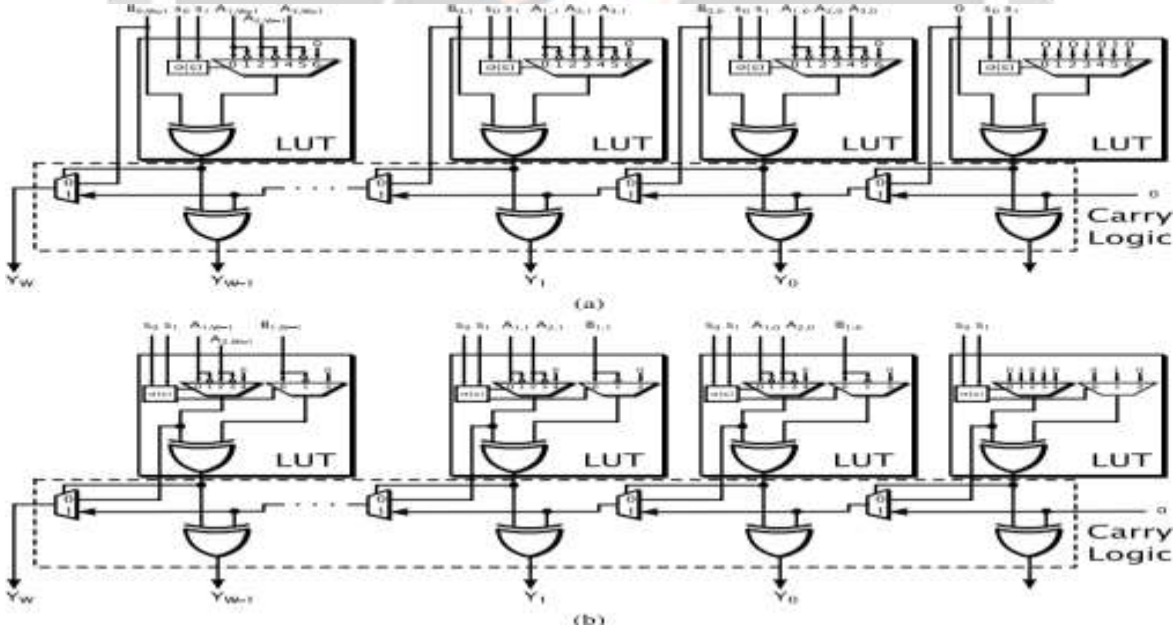

Figure 3. Bit-level FPGA slice mapping of base topologies

(a) Topology A.
(b) Topology B.

3.2 FPGA-based hardware implementation

In FPGA-based realization, SDRAM is used to store a video frame from camera. In SDRAM memory on De0-Nano card used in this study, two equal areas for two frames are allocated - current frame is recorded in the first area, and previous frame is read from the other memory area. And after the output is finished, these areas change their roles. When using SDRAM memory in this study, we consider two important issues. First, memory operates at high frequency of 143MHz, thus, we face one more problem of transferring of data from the clock domain of camera to the clock domain of SDRAM. Second, in order to achieve maximum speed, writing to SDRAM should be performed by whole transactions, or in "burst". FIFO directly built in FPGA memory is the best way to solve both of these problems. Basic idea is that camera fills FIFO at low frequency, then SDRAM controller reads data at high frequency, and immediately writes them to memory in one transaction. Data output to TFT screen is organized in the same way. Data from SDRAM are written to screen FIFO, and then are read at the frequency of10MHz. After FIFO has been cleared, the operation is repeated. Process of frame data transfer through different memory locations is shown in Fig.4A, and the functional design of the project using Verilog modules is shown in Fig.4B.

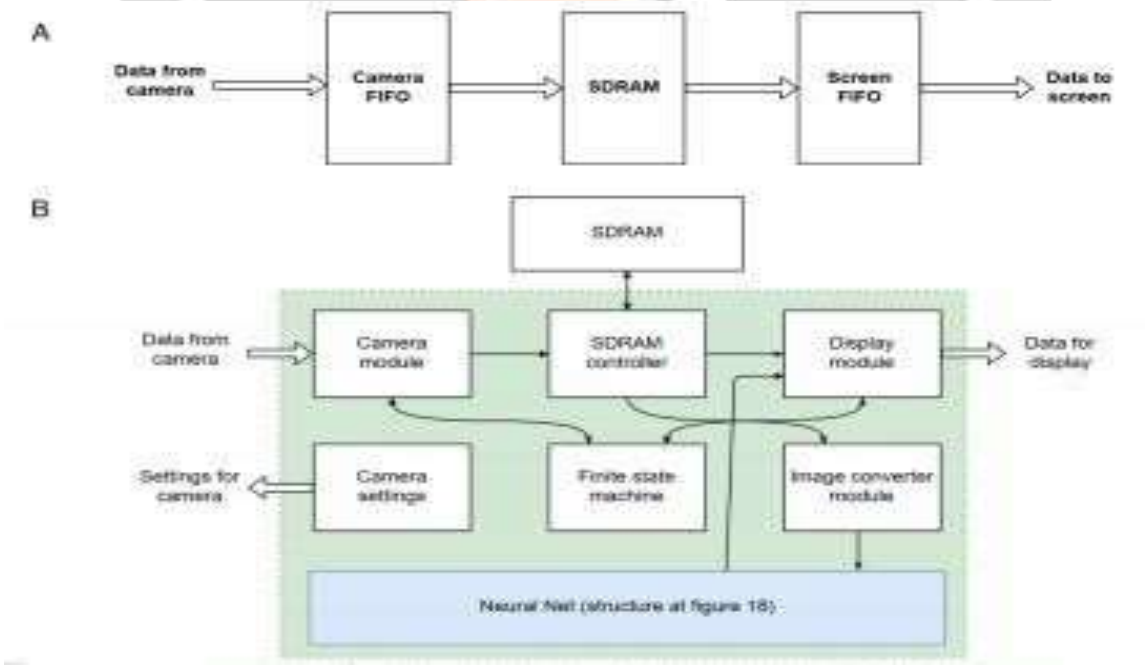

Figure 4. Data flow in FPGA implementation:

A. Frame data transfer; B. Functional design of the project using Verilog modules.

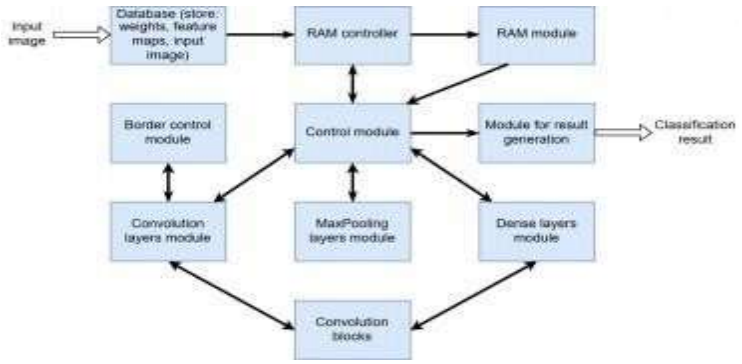

Figure 5. Schematic representation of the hardware implementation

A picture from the camera, after passing through SDRAM, is displayed on the screen as is, and also is fed to neural network for its recognition through block that converts image to grayscale and decreases resolution. When neural network operation is finished, the result is also output directly to the screen. Since the camera has a large number of operation parameters, they are incorporated in a separate module that uploads them to the camera before the operation starts. Hardware implementation of the neural network is presented schematically in Fig.5.

After conversion, an input image is stored in the database, which also stores weight coefficients for each layer that were calculated and wired-in beforehand. As necessary, data from there is downloaded through the controller to the small memory unit for the further use. Everything in neural network is controlled by the main module that keeps the sequence and parameters of network layers. In the hardware realization, not all layers of neural network under test are used; some of them are replaced by other functions. For example, there is no ZeroPadding layer, instead of it module of intermediate image edge detection is applied, which allows to reduce chip memory usage. GlobalMaxPooling layer is replaced by the function from the Convolution layer that immediately gets GlobalMaxPooling layer result by finding the largest value in the intermediate image. The rest of the layers are implemented as separate modules. Since Convolution and Dense layers can use convolutional blocks for calculations, both of them have access to these blocks. Modules contain ReLU activation function, which is used as needed. This modified activation is realized as a separate module, from which results of the neural network operation are received. To implement the neural network, the specialized Convolution block is used, which performs convolution of 3 × 3 in one clock cycle (we can make such block for other dimensions 4×4, 5×5, etc., if we have those in the network). This block is a scalar product of vectors and contains 9 multiplications and 8 additions. The same block is used for calculations in the fully connected.

4. PERFORMANCE RESULTS

	Number of LUT's Utilized	Number of DSP slices utilized	Delay	Static power dissipation	Dynamic power dissipation	Total power dissipation
CNN	112	22	18.936 ns	0.791 W	27.181 W	27.972 W
DCNN	56	26	16.779ns	0.791 W	25.952 W	26.743 W
Modified DCNN	8	28	14.156 ns	0.791 W	23.780 W	24.571 W

Table -I Comparison of different CNN architectures

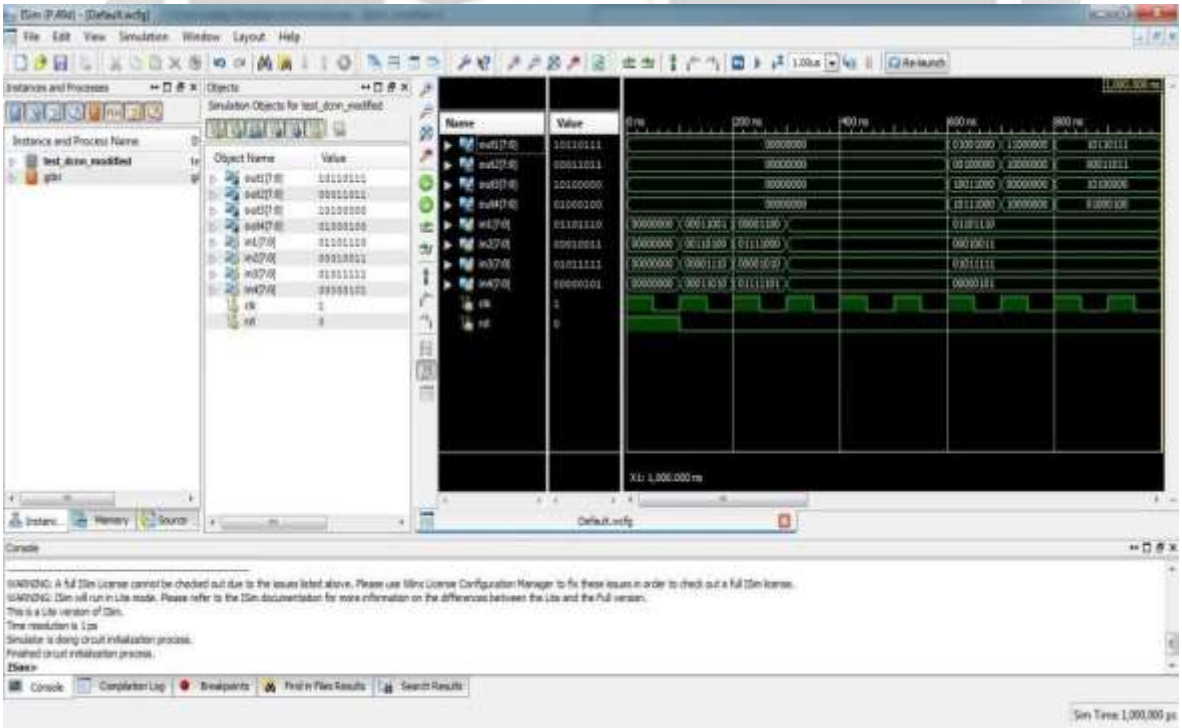

Figure 6. Simulation result of the proposed design

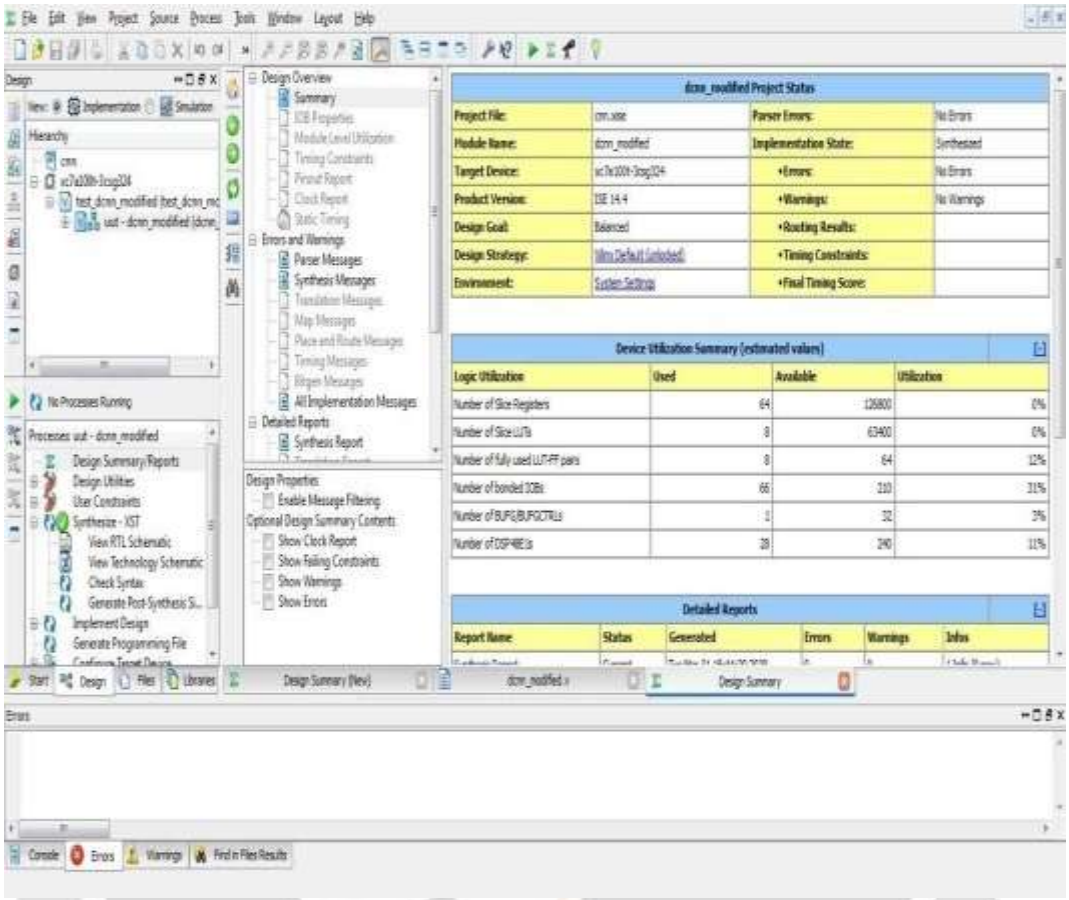

Figure 7. Summary report of the proposed design

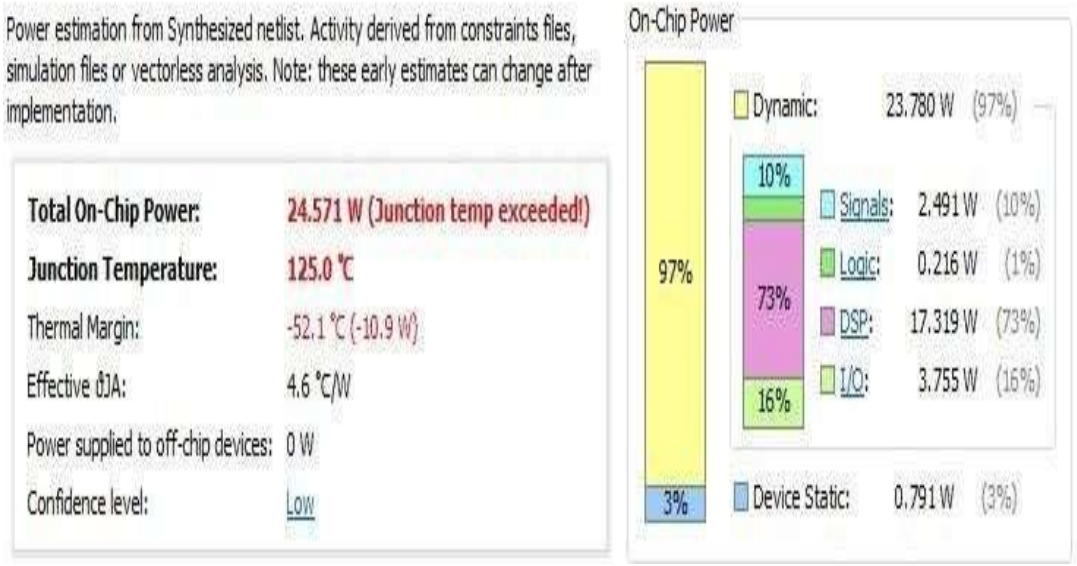

Figure 8. Power report of the proposed design

5. CONCLUSION

There are many possible ways to improve performance of hardware implementations of neural networks. While we explored and implemented some of them in this work, only relatively shallow deep neural networks were considered, without additional architectural features, such as skip connections. Implementing even deeper networks with multiple dozens of layers is problematic, since all layer weights would not fit into the FPGA memory and will require the use of the external RAM, which can lead to the decrease in performance. Moreover, due to the large number of layers, error accumulation will increase and will require wider bit range to store fixed- point weight values. In the future, we plan to consider implementing on FPGA's specialized lightweight neural network architectures that are currently successfully used on mobile devices. This will allow using the same hardware implementation for different tasks by fine-tuning the architecture using pre-trained weights.

REFERENCES

[1] AysegulDundar, Jonghoon Jin, Berin Martini, and Eugenio Culurciello," Embedded Streaming Deep Neural Networks Accelerator With Applications" Senior Member, Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., vol. 2. Jun. 2016, pp. 2169–2178.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Proc. Adv. Neural Inf. Process. Syst., vol. 25. 2012, pp. 1097–1105.

[3] H.Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," Comput. Vis. Image Understand., vol. 110, no. 3, pp. 346–359, 2011.

[4] R. Socher, B. Huval, B. Bath, C. D. Manning, and A. Y. Ng, "Convolutional-recursive deep learning for 3D object classification," in Proc. ,oogfq`. Neural Inf. Process. Syst., 2012, pp. 665–673.

[5] D. Grangier, L. Bottou, and R. Collobert, "Deep convolutional networks for scene parsing," in Proc. ICML Deep Learn. Workshop, 2009, pp. 1–2.

[6] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex- like mechanisms," IEEE Trans. Pattern Anal. Mach. Intell., vol. 29, no. 3, pp. 411–426, Mar. 2007.

[7] M. D. Zeiler and R. Fergus. (2013). "Visualizing and understanding convolutional networks." [Online]. Available:thttp://arxiv.org/abs/1311.2901

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. (2012). "Improvingneuralnetworksbypreventingco-adaptationoffeaturedetectors."[Online]. Available: http://arxiv.org/abs/1207.0580

[9] M. A. Erdogdu and A. Montanari, "Convergence rates of sub-sampled Newton methods," in Proc. Adv. Neural Inf. Process. Syst., 2015, pp. 3034–3042.

[10] M. A. Erdogdu, "Newton–Stein method: A second order method for GLMs via Stein's lemma," in Proc. Adv. Neural Inf. Process. Syst., vol. 28. 2015, pp. 1216–1224.

[11] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May/Jun. 2010, pp. 257–260.

[12] A. Krizhevsky. (2014). "One weird trick for parallelizing convolutional neural networks." [Online]. Available:http://arxiv.org/abs/1404.5997

[13] S.Chetluretal.(2014)."cuDNN:Efficientprimitivesfordeeplearning."[Online]. Available: http://arxiv.org/abs/1410.0759

[14] C. Farabet, C. Poulet, and Y. LeCun, "An FPGA-based stream processor for embedded real- time vision with convolutional networks," in Proc. IEEE 12th Int. Conf. Comput. Vis. Workshops (ICCV Workshops), Sep./Oct. 2009, pp. 878–885.