Field Programmable Neural Array

Artificial Intelligence at the Edge

Peter Gadfort and Oluseyi A. Ayorinde
Sensors and Electron Devices Directorate
US Army Research Laboratory
Adelphi, MD 20783–1138
Email: peter.gadfort.civ@mail.mil

Melissa Bezandry and Max Yu
Department of Electrical and Computer Engineering
University of Maryland
College Park, MD 20742

***Abstract*—This paper will present the architecture of an Field Programmable Neural Array (FPNA) for AI applications at the tactical computing edge. This platform combines domain specific accelerators for AI with a reconfiguable interconect to permit any Deep Neural Network to be mapped into the FPNA. The FPNA includes domain specific accelerators that perform inference tasks with higher computing efficiency than CPUs and GPUs, approaching that of ASICs designed specifically for AI applications, and a reconfigurable interconnect providing the flexibility and connectivity of an FPGA.**

***Keywords*—artificial intelligence; machine learning; domain specific accelerators; field programmable**

I. Introduction

Conventional approaches to efficiently implementing Deep Neural Networks (DNNs) on-chip result in relatively rigid implementations and while these approaches achieve high computational efficiencies[1], they do not permit system engineers to change the fundamental architecture (Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), etc.) once the hardware has been deployed. As a result, new network structures and varying sizes of DNNs require new hardware to be developed and fabricated, which is costly and not compatible with rapid deployment schedules. Field Programmable Gate Array (FPGA) implementations of DNNs are also possible[2], and even though they allow for greater flexibility, they are not able to reach the efficiencies required for Army edge computing applications. Fig. 1 shows the energy efficiency of different computing platforms, including the proposed solution – Field Programmable Neural Array (FPNA) – which will target a computing efficiency of 200 GOPs/W.

Many Army systems have extremely stringent size, weight, and power (SWaP) constraints, therefore they are especially sensitive to increases in the total power load, and thus require low-power additions to their platforms[3], [4]. Focusing on the most common platforms in Fig. 1, we tabulate the power required for several different DNNs (GoogleNet, AlexNet, and automatic speech recognition (ASR)), based on the operations count from [5] and efficiency from [1], [6], in Table I. From this table, it is apparent that solutions using CPUs, GPUs, and FPGAs will require too much power to make them practical for most edge computing applications.

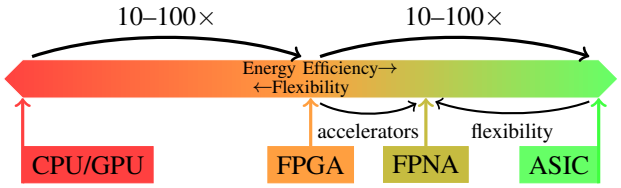


Fig. 1: Relative Computing Efficiency for AI platforms

TABLE I: Added power burden for common AI algorithms

	Efficiency GOPs/W	GoogleNet 16.33 GOPs	AlexNet 7.33 GOPs	ASR 502.65 MOPs
CPU	5.1	3 W	1 W	98 mW
GPU	18.3	891 mW	400 mW	27 mW
FPGA	47.00	347 mW	156 mW	11 mW
ASIC	8100.00	2 mW	905 µW	62 µW
FPNA	200.00	82 mW	37 mW	3 mW

Considering a more concrete example of the 72 h mission length requirement for the Soldier, we can compute the loss of mission time when adding additional AI applications to their mission. Here we will assume that GoogleNet, AlexNet, and LeNet will all need to be performing 15 inferences per second, causing the power to increase by 15× from Table I, and ASR is doing 200 inferences per second. Table II shows how much the mission time would be reduced if the Soldiers battery load is not increased to accommodate the additional processing. From the table, it is apparent that CPUs, GPUs, and FPGAs, will not be able to support the mission requirements for the Soldier. The ASIC example would be ideal as it does not impact the Soldier mission by a noticeable amount; however, this approach would be cost prohibitive. While the FPNA still reduces the mission length by a few hours in some cases, these are not likely to the cases that would be deployed on the Soldier and ideally, the AI deployed on the Soldier would be better tailored to their mission and the supported hardware.

TABLE II: Mission length reduction for Soldier

	GoogleNet	AlexNet	LeNet	ASR
CPU	60.4 h	50.4 h	3.3 h	49.1 h
GPU	42.7 h	28.5 h	56.8 s	27 h
FPGA	26.1 h	14.6 h	22.3 s	13.6 h
ASIC	14.2 s	6.4 s	7.8 ms	5.8 s
FPNA	8.5 h	4.1 h	5.3 s	3.7 h

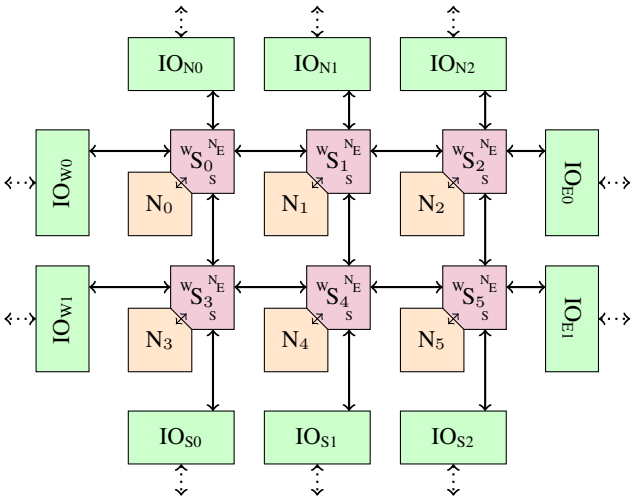


Fig. 2: Example 3×2 FPNA

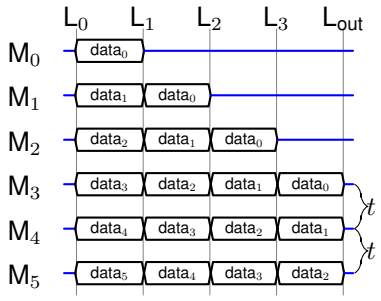


Fig. 3: Data movement in FPNA fabric.

II. Field Programmable Neural Array

This efficiency and flexibility challenge can be overcome by borrowing interconnect fabrics from FPGAs (synapses) and implementing dedicated accelerators (neurons) to perform the required mathematics, resulting in the FPNA architecture. By using a programmable fabric, composite neurons can be created in the hardware to implement more complex functions like the Long Short-Term Memory (LSTM) cells or cells with larger input spaces than would be physically possible with an individual neuron. This will open up new computing opportunities by providing the Army with a cost-effective, energy-efficient machine learning platform, which can be deployed with the Warfighter.

Fig. 2 shows an example of the FPNA fabric, illustrating how the neurons and synapses are connected to each other and how synapses are tied together with each other and with off-chip communications. The FPNA will be moving the data around on the chip as a pipeline, as shown in Fig. 3. Here, each step of the data movement is described by $M_{0\ldots5}$ and each layer of neurons is described by $L_{0\ldots3,\text{out}}$. By pipelining the FPNA, we can keep the neurons full of information and better utilize the total functionality of the chip. This is a significant departure from other solutions which rely heavily on an external memory to provide the weights and biases. The FPNA exclusively uses on-chip memories to store the weights and biases, and thus we can eliminate costly, off-chip memory accesses.

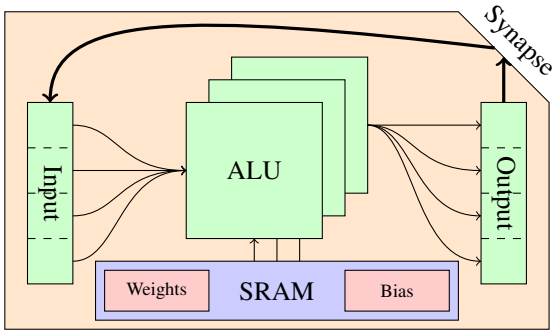


Fig. 4: Neuron internal structure

This paper will discuss the two structures of the FPNA: Neurons (Sec. III), which perform the necessary mathematics to compute each layer of a DNN, and Synapses (Sec. IV), which support the neurons and move data between them. In Sec. V, we describe the programming of each neuron and synapse to be able to perform machine learning (ML) inferencing tasks.

III. Neurons

The neuron is the computational engine in the FPNA that performs all the required mathematics and data organization necessary to complete the computation of a full DNN. A diagram of the neuron is shown in Fig. 4. The inputs are fed from outside the neuron and stored into a local memory. The outputs are similarly stored in a local memory, and each of these storage banks contains simple Arithmetic Logic Units (ALUs), which can perform additions and other simple single-cycle operations. The core of the neuron is the ALU, where most of the complex math is performed. It primarily consists of a set of vector/matrix multipliers and can be configured to use different word sizes to maximize the efficiency and utilization of the neuron. Finally, the configuration memory contains all the preprogrammed weights and algorithm selection for the neuron.

For reference, two networks were compiled into the FPNA structure, LeNet and a Mathworks example speech command network[7]. Fig. 5 shows how many individual neurons would be required to construct that network on the FPNA. Here we can see that if the neuron memory is too small, a large number of neurons would be required, which would increase the complexity of the routing in the synapses. For very large memories, the neurons level off because a single neuron can only perform a few tasks, so adding more memory to a given neuron does not improve the overall network packing.

To help determine the best size of memory for each neuron, Fig. 5 would suggest it would include approximately 2048 to 4096 words. To better evaluate this, the size of the neuron logic and the resultant memory were combined with the data in Fig. 5 to generate Fig. 6. This figure shows the effective FPNA required to implement the desired network. In this case,

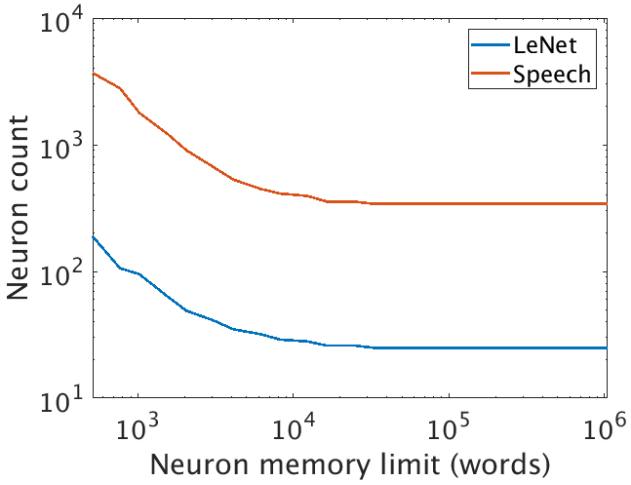


Fig. 5: Neuron count

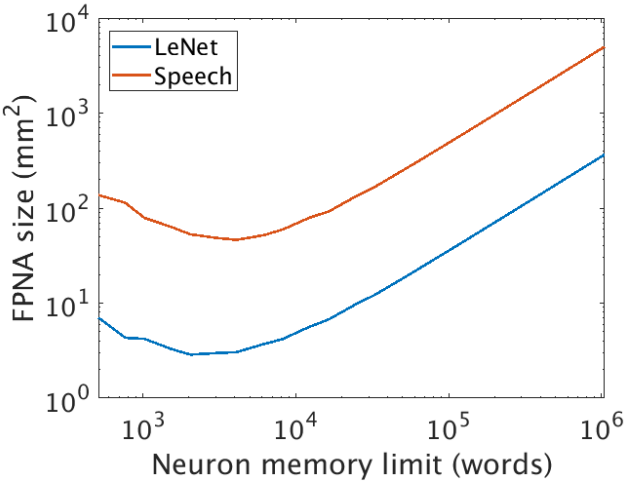


Fig. 6: FPNA size

we are using GlobalFoundries 55lpx and would ideally build the neurons in the valley that appears at 3072 words. At this size the FPNA would be 2.98 mm^2 and 48.38 mm^2 for the LeNet and speech networks, respectively. While this is a large discrepancy, the FPNA is designed to be tiled together to build a larger effective FPNA, therefore, we would be able to build a single chip that could cover the LeNet example, and then use 16 chips to build a larger system to cover the speech network example.

IV. Synapses

The synapse is the reconfigurable interconnect block in the FPNA, responsible for routing the data from a neuron to other neurons. The connectivity of computational blocks varies widely in neural network implementations, ranging from relatively sparse connectivity in CNNs to the dense connectivity of fully-connected layers used for classification. As a result, the synapse needs to support varying levels of connectivity efficiently. A diagram of the synapse is shown in

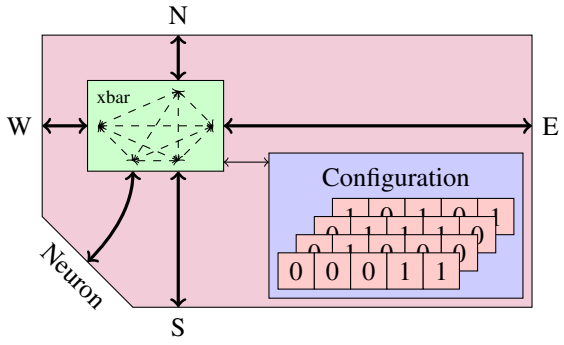


Fig. 7: Synapse internal structure

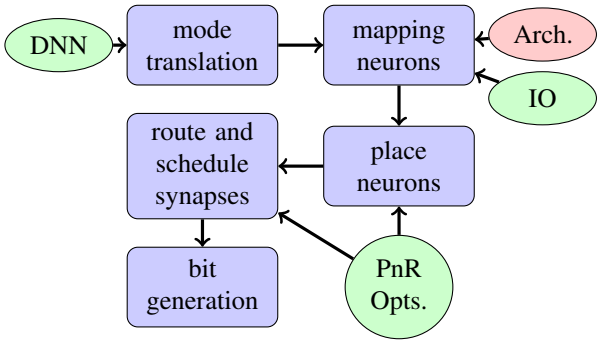


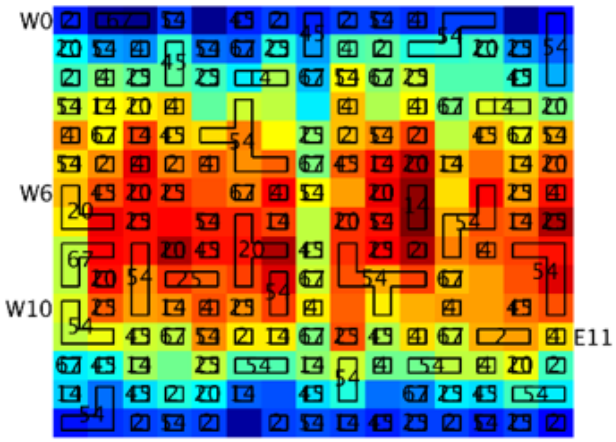
Fig. 8: Programming flow

Fig. 7, which shows the cross-bar structure in the synapse and the routing configuration table. This architecture is similar to a Network-on-Chip (NoC); however, by recognizing that data always moves between the same set of neurons at the same time, the inter-block traffic can be scheduled to avoid collisions and known ahead of time. This allows the synapses in the FPNA to be far less complex than routing nodes in a NoC, by removing arbitration and buffering circuitry altogether.

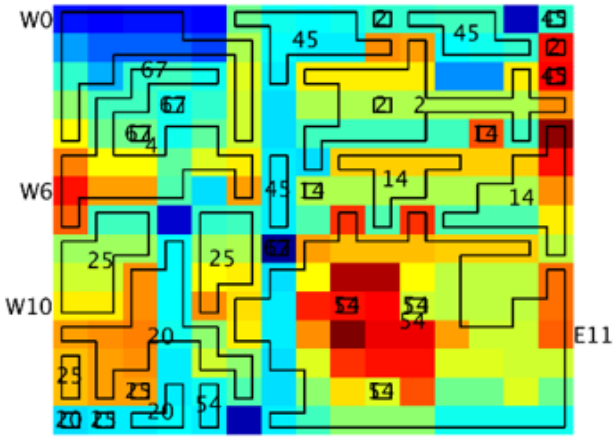
V. Programming

For the FPNA to be useful, it must be programmed with the desired, trained DNN. To make sure this step has the lowest possible impact on utility of the FPNA, the programming flow is similar to that of an FPGA, as illustrated in Fig. 8. This flow takes in a trained DNN and converts the network into smaller mathematical operations that are then mapped into the neurons. The neurons are then placed into the FPNA fabric, the synapses are routed and scheduled, and finally the bit stream for that DNN is produced. This flow is modeled after Verilog-to-Route (VTR)[8]; however, instead of FPNA synthesis, place, and route, we do mapping, placing, and routing.

To show some example of the flow, we ran LeNet through the flow to build show some different aspects of the tools. Fig. 9 shows the routing heatmap of a network before annealing the placement of neurons on the FPNA and after. Because we start with a randomly placed mapping, annealing is necessary to improve the routing performance of the network. Neurons



(a) Before anneal



(b) After anneal

Fig. 9: Impact of anneal

are first placed by minimizing the distance between neurons within the same layer, which creates large congenstion points. Additional annealing steps are required to help smoothe those out. In the figure, prior to annealing (Fig. 9a) the routing shows several hotspots which makes scheduling the synapses difficult; however, after the annealing (Fig. 9b) step the layers are better packed and the hotspots are reduced.

Fig. 10 shows an example of a single route for a given neuron to all its destination neurons. While there is still work to be done here to better pack the routes to reduce parallel routes which block those resources from other routes, this route shows how the programming software correctly build the routes for the FPNA.

Conclusions

In this paper, we present a new architecture – the Field Programmable Neural Array – for AI at the tactical edge,

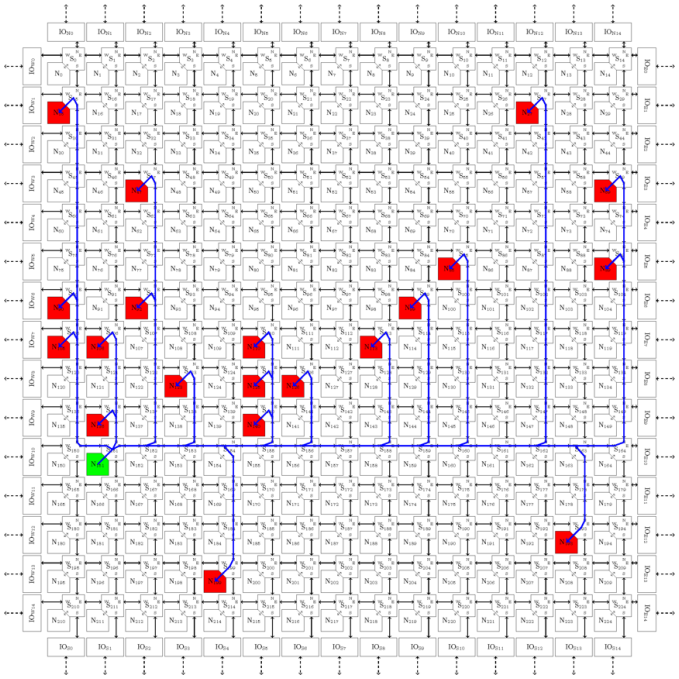


Fig. 10: Example of neuron connectivity

which provides the required energy efficiency and flexibility to allow for the deployment of AI at the tactical edge.

References

[1] D. Shin, J. Lee, J. Lee, and H. J. Yoo, "14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *2017 IEEE ISSCC*, Feb 2017, pp. 240–241.

[2] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *2015 ACM/SIGDA FPGAs*, 2015, pp. 161–170.

[3] "Circuit Realization at Faster Timescales (CRAFT)," https://www.darpa.mil/attachments/ObscurationandMarking_Slides.pdf, accessed: September 24, 2018.

[4] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Feb 2007.

[5] "Quick Start - Netscope CNN Analyzer," https://dgschwend.github.io/netscope, accessed: May 17, 2018.

[6] "GPU-Based Deep Learning Inference: A Performance and Power Analysis," https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf, November 2015.

[7] "Speech Command Recognition Using Deep Learning," https://www.mathworks.com/help/deeplearning/examples/deep-learning-speech-recognition.html, accessed: January 10, 2019.

[8] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "Vtr 7.0: Next generation architecture and cad system for fpgas," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 2, pp. 6:1–6:30, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2617593