PNeuro: a scalable energy-efficient programmable hardware accelerator for neural networks

A. Carbon, J.-M. Philippe, O. Bichler,
R. Schmit, B. Tain, D. Briand and N. Ventroux
CEA, LIST
F-91191 Gif sur Yvette, France
alexandre.carbon@cea.fr

M. Paindavoine, O. Brousse
GlobalSensing Technologies
14 rue Pierre de Coubertin
F-21000 Dijon, France
michel.paindavoine@gsensing.eu

***Abstract*—Artificial intelligence and especially Machine Learning recently gained a lot of interest from the industry. Indeed, new generation of neural networks built with a large number of successive computing layers enables a large amount of new applications and services implemented from smart sensors to data centers. These Deep Neural Networks (DNN) can interpret signals to recognize objects or situations to drive decision processes. However, their integration into embedded systems remains challenging due to their high computing needs.**

This paper presents PNeuro, a scalable energy-efficient hardware accelerator for the inference phase of DNN processing chains. Simple programmable processing elements architectured in SIMD clusters perform all the operations needed by DNN (convolutions, pooling, non-linear functions, etc.). An FDSOI 28 nm prototype shows an energy efficiency of 700 GMACS/s/W at 800 MHz. These results open important perspectives regarding the development of smart energy-efficient solutions based on Deep Neural Networks.

***Index Terms*—Neural networks; Neural network hardware; Computer architectures; FPGA; ASIC; Low power**

I. Introduction

Artificial intelligence and especially Machine Learning gained recently a lot of interest from the industry thanks to deep learning algorithms. Some of them are getting their inspiration in the structure and function of the visual cortex and the brain. By cascading feature extraction filters and multiple neuron-based layers of nonlinear processing units, these algorithms have the capacity to efficiently segregate data (inference process) and to answer the need of advanced classification applications. This property is mainly provided by learning algorithms that enable the modification of processing chain parameters based on new data. In this field, Deep Neural Networks (DNNs) have a large number of successive computing layers and can be used to interpret signals to recognize objects or situations to drive decision processes. They can be exploited in complex cognitive systems, enabling a large amount of new applications and services. One of the most known example of DNN in image processing is the Convolutional Neural Network (CNN) [1] composed of convolution and pooling layers followed by fully connected layers (Figure 1).

Unfortunately, the integration of DNN applications into embedded systems remains challenging because of their high complexity. Indeed, due to their structure and the amount of parameters obtained in the learning phase, the inference phase is compute- and memory-intensive. In addition, even if their computation only requires the use of simple energy-efficient operators, their number is tremendous and requires a complex interconnection and memory hierarchy architecture to remain energy efficient. Thus, designing an area and energy-efficient accelerator to compute DNN is both strategic and complex.

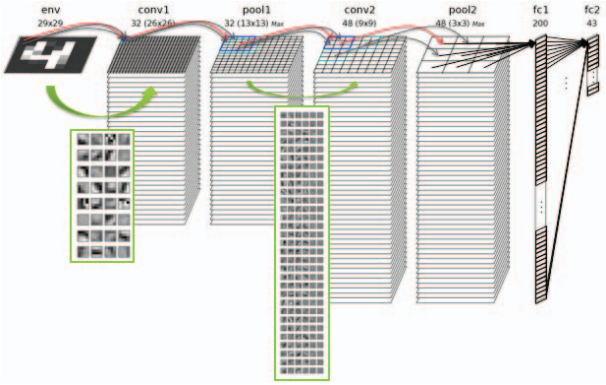

Fig. 1. Typical structure of a Deep Neural Network with successive computing layers (convolution, pooling and fully connected layers).

Contributions. This paper presents a new scalable energy-efficient programmable hardware accelerator for DNN inference phase. This dedicated processor named PNeuro is a clusterized SIMD (Single Instruction Multiple Data) architecture able to perform all DNN operations (convolutions, pooling, non-linear functions, etc.). Compared to existing solutions, PNeuro uses a smart data management, combining on-the-fly routing and transformation units as well as flexible and optimized computing datapaths, enabling an efficient execution of many different DNN applications. FDSOI 28 nm synthesis shows an energy efficiency of 700 GMAC/s/W at 800 MHz.

This paper is organized as follows. The second section presents the existing work. Section III details the PNeuro architecture and highlights our main contributions. Section IV presents the programming and software tools. Section V discusses the validation of PNeuro and presents FPGA and ASIC evaluations. Finally, Section VI concludes the paper and discusses the future work.

978-3-9819263-0-9/DATE18/©2018 EDAA

II. Related Work

Efficient processing of modern neural networks can be performed by three kinds of hardware accelerators: Graphics Processing Units (GPUs), Field-Programmable Gate Arrays (FPGAs) and Application-Specific Instruction set Processors (ASIPs).

General purpose GPUs are widely used for the learning and inference of DNNs, especially on the server and High-Performance Computing sides. For the inference, GPUs represent a direct solution to accelerate the execution of neural applications with a large support of DNN design environments (e.g. TensorFlow [2] or Caffe2 [3]). Their deeply-parallel SIMD structure and high-data bandwidth are well-adapted to neural computing. However, these accelerators initially designed for graphical processing come with a lower energy efficiency than specific solutions targeting DNN accelerations [4] and their costs make them unsuitable for high-volume integration.

In the meantime, FPGAs are also gaining in interest as powerful and energy-efficient solutions for low-volume embedded systems. Recent works [5], [6] demonstrate their capacity to adapt their structure to dataflow and massively parallel implementations of DNNs. Nevertheless, DNN structures are rapidly evolving and new neural network layers are included in existing DNN so as to perform fine tuning. Architectures exploiting FPGA abilities heavily rely on hardware datapaths which are not flexible enough and necessitate to be deeply modified to support new topologies, activation layers, etc. In addition, the dataflow approach limits the DNN complexity to the FPGA resources availability. Although time sharing of operators can help to overcome the limitation of FPGA resources for dataflow implementations, this solution comes at the expense of computation latency.

Finally, ASIPs bring dedicated and flexible solutions for the execution of DNN applications. They can support a large set of different DNNs thanks to their high programmability or reconfigurability. Their structure, optimized for DNN, makes these solutions highly efficient in terms of area and energy efficiency. In addition, ASIPs are well-adapted for a deep integration into constrained high-volume embedded systems. Many different architectures can be considered like 2D-mesh dataflows, multi-cores, etc. depending on the application domain. Additionnaly, a trade-off can be found between flexibility and efficiency for many parts of the overall architecture such as the computing elements or the interconnect. Another interesting property of these solutions is that specific low-power mechanisms can be added to improve the performance per watt ratio. From the academic point of view, we can cite Neuflow [8], DianNao [9] or Eyeriss [10]. The industry is also very active in this research field, with for example the Orlando [7] platform from STMicroelectronics which focuses on very low-power neural network execution. The Synopsys EV6x Vision Processor [11], the CEVA XM6 IVP [12], the Intel Myriad X [13] or the Google TPU [14] are some other examples among commercial solutions.

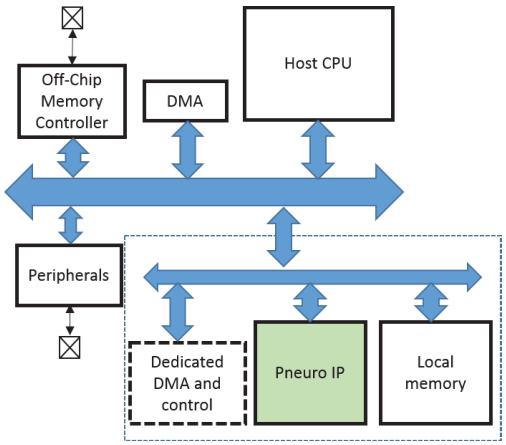

Fig. 2. Typical system integration for PNeuro. A local memory stores data for fast accesses (partial results, programs, network parameters, etc.).

III. PNeuro Architecture

PNeuro is a scalable energy-efficient programmable architecture designed for the acceleration of neural network operations. It is an IP, targeting both FPGA or ASIC technologies. PNeuro was specified and designed to accelerate various types of neural network layers and even image processing algorithms.

A. System integration

The PNeuro accelerator connects to the host system through an AXI4 interface. All the internal memories (program and data) are exposed to the overall system (e.g. host processor, DMA). Synchronisations between the host and PNeuro are done through a set of registers. An optional interrupt subsystem can be used to avoid costly polling. PNeuro is composed of independant clusters, which will be described in the next subsection. Two operation modes are available. The clusters can execute different programs so as to implement different layers or they can execute the same program in a synchronized way to increase data parallelism during layer execution.

A typical system integration for the PNeuro IP is presented in Figure 2. A local memory is dedicated to data that require fast accesses, such as intermediate results and the ones that cannot fit into PNeuro memories (e.g. network parameters and partial programs for large networks, etc.). An optional dedicated control subsystem comprising a DMA (Direct Memory Access) can be embedded into the PNeuro scope to lower the workload of the host processor. Particular attention must be paid to data management since most efficient PNeuro kernels use interleaved data to efficiently compute convolutions. Consequently, a dedicated control subsystem can be usefully used to manage the communications with the upper-level of the system, to synchronize the operations of the different clusters, to transfer data, to compute some complex serial or irregular operations and to interpret the results processed by the accelerator.

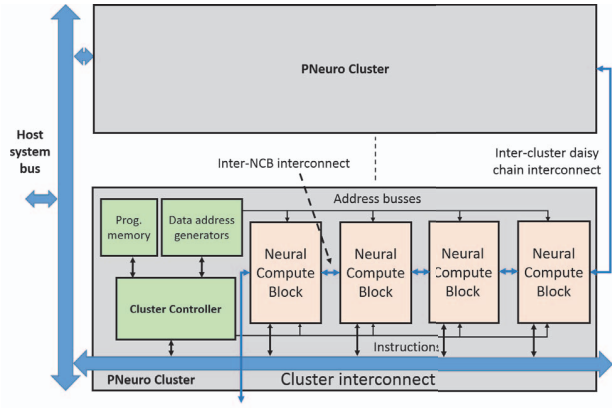

Fig. 3. Global architecture of PNeuro. The cluster controller sends intructions to the NCBs and address generators send addresses to the memories inside the NCBs. The inter-NCB interconnect enables neighbour data sharing.

To use the PNeuro accelerator, the host CPU loads the programs into the program memories of the clusters (or asks the dedicated control subsystem to do it). Data are then loaded into the data memories of the clusters (input data, network parameters, etc.) and in the local memory near the PNeuro. Once this process is done, the host CPU or the dedicated control subsystem sends the start signal to the cluster controllers. Synchronisation instructions executed by the cluster controllers can stall the PNeuro execution for critical parts of the application. For example, they can be used to wait for the results or until the next set of network parameters have been loaded from the local memory.

B. General architecture

PNeuro is a clustered SIMD architecture (Figure 3). The different clusters are connected to an AXI4 host-system bus. Each cluster is composed of Neural Computing Blocks (NCBs) that will be described in more details in the next subsection. All the NCBs belonging to a cluster are controlled by a local cluster controller. It fetches instructions from the local program memory of the cluster, decodes them and sends control signals to all NCBs of the cluster. The local controller contains all the configuration and status registers of the cluster, which are accessed thanks to dedicated control instructions.

Three dedicated data address generators per cluster automatically compute address patterns to be broadcasted to the internal memories of the cluster. While they are not specialized by nature, two of them usually generate adresses for the operands (e.g. pixels and filter coefficents) and the last one generates addresses for storing results. Each address generator has two sets of registers to quickly switch between two patterns. These registers comprise the index register (the address), the modifier register (to be added or substract to the index register during generation) and two registers for the definition of address generation bounds. Thus, these address generators can provide the right data and coefficient / synaptic weight addresses at each cycle so as to efficiently feed the computation units inside the NCBs. This is a key feature for the PNeuro architecture since these computation units are directly connected to the internal memories without any additional register operations.

Each NCB can quickly exchange data with its immediate neighbors via an inter-NCB interconnect. This fast network connects the external ports of routing units inside the NCBs (see subsection III-C). Thus, it allows to directly compute an external data coming from the memory of a neighbored NCB with local NCB filter coefficients or synaptic weights. This interconnect creates a daisy chain network connecting all clusters together. Combined with synchronized control programs, it can be used to configure the whole PNeuro as a wide parallel architecture able to process larger pictures for example.

C. Neural Computing Block

The Neural Computing Block is composed of a multi-banked SRAM (Static Random Access Memory) connected to Processing Elements (PE) working in SIMD mode, as illustrated by Figure 4. The multi-banked SRAM consists in multiple independent memories used to store data for PEs. A typical usage of this configuration is two memories for operands, one for temporary results and one for storing the final results (used by the next layer or to be retrieved by the host). Contrary to other approaches, there is no dedicated memory for filter coefficients or synaptic weights. This choice was done to flatten the memory hierachy in order to maximize the usable memories in either traditionnal image processing or neural network operations. All SRAM blocks can be targeted by the address generators of the cluster.

The connection between memories and PEs is ensured by a routing / data transformation module performing on-the-fly operations with the incoming data. Data from the NCB memories can be accessed by any individual PE within one cycle, despite the transformation performed by this module. It can introduce zeros or ones for padding operations, copy or multicast values, perform bit-shifting for data alignment between PEs, and support neighbour data accesses. The multicast property is very important for neural computing, since it is necessary to send convolution or neural network parameters to multiple PEs at the same time. It allows direct data exchanges between the different PEs for efficient local transfers without using the memories. This interconnect consequently enables the programmer to transfer data of different precisions (pixels, coefficient or weights, multiplication results and accumulations) between the PEs with ease. For this purpose, two interconnection paths are available. As it will be described later, PNeuro operators are optimized for 8-bit operations. Thus, a 8-bit path enables the transfer of native PNeuro data. The other path is 32-bit wide and is mainly used to transfer accumulations or groups of 8-bit data. The routing modules of the NCBs are connected to other NCB routing modules, to more efficiently compute large input data, as shown in Figure 3. These connections are fully configurable by the programmer using dedicated cluster controller instructions.

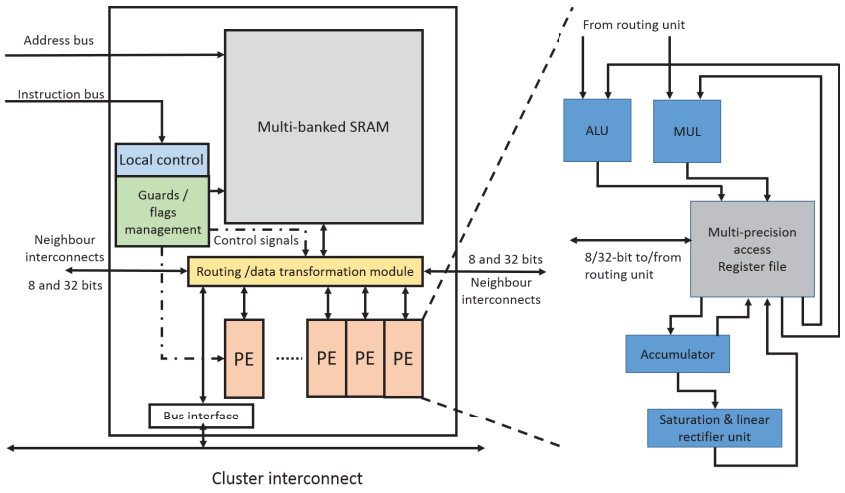

Fig. 4. Architecture of the Neural Computing block (NCB) with details on a processing element.

PEs within NCBs heavily work with cluster memories. But each of them also has a small configurable register file for improving local data management. It can be accessed to read or write data of 8, 16 or 32 bits. 8-bit accesses are mainly used for operands and final results, 16-bit accesses for multiplication results and 32-bit accesses for targeting the accumulator and groups of 8-bit data. As a result, PNeuro is very flexible regarding the operand sources. Computing instructions can work on data coming directly from the cluster memory, from the internal register file and even from neighbored resources, while keeping the same instruction format.

The NCB is composed of PEs supporting the execution of the full processing chain (i.e. there is no shared operator inside each NCB). The PE is composed of a MAC (Multiply Accumulate) operator, an ALU (Arithmetic and Logic Unit) and a dedicated non-linear-operation unit based on approximated functions. PNeuro is mainly an 8-bit architecture and thus was optimized for this data width from interconnect and computation points of view. Specific sequence of instructions together with the use of the internal register file can bring the support of 16-bit data if needed. The PE includes a 9-bit multiplier and a 32-bit accumulator. The multiplier uses automatic sign extension units keeping full-precision of 8-bit data operations in a signed or unsigned format. The output of the accumulator is connected to a specific unit, which performs saturation and non-linear approximation functions. Saturation units can transform 32-bit data coming from the accumulator to 8-bit data before storing them. Thanks to an automatic MSB (Most Significant Bit) detector, this transformation can be automatically made. In addition, the saturation parameters can be fully controlled by the programmer. PNeuro supports approximations of non-linear functions such as hyperbolic tangent (tanh), sigmoïde, linear rectifier and even radial-basis functions. This means that PNeuro is not restricted to pure CNN operations. Each non-linear funtion is approximated using a sequence of instructions using the internal register file, the MSB detector and shift operators. Being an SIMD accelerator, each NCB is managed using guards (related to operation flags) to enable *if-then-else* and similar control structures. Particular PEs can also be disabled in case of complex control flows by using a specific instruction enabling the independent control of the different PEs in the whole cluster.

IV. Programming and Software Tools

PNeuro uses 32-bit instructions for both control and computing. There are 28 instructions for the control and 40 instructions for the computation. The instructions follow the same pattern. The MSB indicates whether it is a control or a computation instruction, whereas the following 6 bits encode the opcode of the instruction. Contrary to the control instructions, the computation instructions use guards (next bits for this type of instructions). Then, operand sources are encoded. They can come from the PE register file, the neighbored PEs, the memory or an immediate. Signed and unsigned operand types can be specified, allowing sign extension modules of the datapath to automatically transform the input data.

These instructions have been used to derive an assembly language. There are more than 150 mnemonics to enable a fine tuning of the programs. These mnemonics expose all the mechanisms of the PNeuro to the programmer allowing to bypass all automatic features of the architecture. An assembly-to-binary generator has been developped to produce an executable program for each cluster.

In order to ease the programming of PNeuro, a complete toolchain will be available and integrated into our inhouse open source N2D2 framework [15]. N2D2 (Neural Network Design and Deployement) is a comprehensive framework solution for fast and accurate DNN simulation with fully automated DNN-based-application building and benchmarking capabilities. This platform integrates a complete set of pre- and post-processing functions that can be applied on

databases. Thus, N2D2 allows to build complete processing chains, including neural network topologies, and to generate the code for various hardware platforms. At the moment, it can generate codes for multi- and many-core processors (OpenMP/OpenCL), GPUs (OpenCL, Cuda, cuDNN, TensorRT) and FPGAs.

For building the complete PNeuro toochain, a specific PNeuro code generator is currently under development and will be integrated into the N2D2 platform as a plug-in. It will allow the description of state-of-the-art DNNs through an input text file and the generation of the corresponding assembly code after application validation. The assembly code will then be compiled by the already-developed assembly-to-binary generator to build the PNeuro machine instructions.

V. Evaluation and results

In order to evaluate the PNeuro IP, an FPGA implementation and ASIC syntheses have been carried out. The following subsections detail the results and present some comparisons with respect to the state of the art. A typical configuration of the PNeuro IP was chosen for the experiments. Table I details the different parameters. The PNeuro IP is composed of 2 clusters gathering 64 processing elements and 264 KBytes of embedded memory. The IP has been integrated into a complete low-power automotive subsystem composed of two locksteped AntX processors configured in a monothreaded way [16], an UART, SRAM controllers, and GPIOs.

TABLE I
PNeuro parameters for the evaluations.

Parameter	Value
Number of clusters	2
NCBs per cluster	4
PEs per NCB	8
Total number of PEs	64
Data memory size per cluster (KB)	128
Program memory size per cluster (KB)	4
Total memory (KB)	264

A. FPGA target

The overall system has been implemented using Vivado 2017.3.1 on a Xilinx Virtex-7 XC7V2000T FPGA (-2 speed-grade) at 100 MHz. Table II depicts the resources used for implementing the full system including the PNeuro IP on the FPGA. No DSP Slice can be exploited by the synthesizer since 9-bit multipliers are used to keep full-precision of 8-bit data, in a signed or unsigned format (an 8-bit pixel is known to be always positive, contrary to a weight or a convolution parameter).

TABLE II
FPGA resources (full system and 2-cluster PNeuro IP).

Resources	Used by the full system	Used by the PNeuro IP
LUT (Logic)	144153	76261 (52.9 %)
FF	183427	11522 (6.3 %)
RAM36	102	66 (64.7 %)

B. ASIC target

The overall system has also been synthesized using a ST Microelectronics 28 nm FDSOI technology (TT, 25C, 0.9V). A frequency of 800MHz has been achieved. The two clusters of the PNeuro IP occupy 0.93 mm^2 and have a total power consumption of 73 mW (the memories consume 35.2 mW). Regarding the area breakdown, the memory accounts for the largest area of the cluster with approximately 80 % of the total area. Thus, the energy efficiency of the PNeuro reaches 700 GMAC/s/W, whereas the area efficiency is about 54 GMAC/s/mm^2. Table III shows some comparisons with existing commercial neural IPs. It highlights that PNeuro is both more energy and area efficient.

TABLE III
Comparisons with some commercial DNN accelerators.

Architecture	CEVA XM6	Synopsys EV64	PNeuro
Clock Speed (MHz)	690	500	800
Technology node (nm)	20	28	28
Data types (bit)	16-32-64	8-16-32	8-16
Integer MACs per cycle	512	800	64
Energy consumption (mW)	800	1300	73
Area (mm^2)	NC	8.2	0.93
Energy efficiency (GMAC/s/W)	441	307	700
Area efficiency (GMAC/s/mm^2)	NC	49	54

C. Performance evaluations

For the PNeuro evaluation, a simple CNN was manually programmed in assembly code and executed on the FPGA prototype. This small network is able to classify four categories of pictures from the Caltech-101 database [17]. The topology of the CNN is illustrated Figure 5. An input picture of 48x48 pixels is processed by four convolution filters (two 3x3 and two 5x5 coefficients). The results are then pooled using Max operators. The outputs of this layer are sent to fully connected layers consisting of 60 neurons in the hidden layer and four neurons in the output layer, corresponding to the four chosen categories (face, car, motorcycle and plane). The network has a complexity of 450 KMACs. Coefficients and synaptic weights are learned using backpropagation of errors in the N2D2 design environment. Data sizing are required to meet the architecture requirements. After the learning phase using floating-point numbers, a precision reduction to 8 bits has been carried out for the feed-forward phase.

With approximately 1000 lines of binary instructions for the entire processing chain, a PNeuro cluster is then able to simultaneously categorize four images in about 70,000 clock cycles with a recognition rate of 96 %. This timing includes the synchronizations with the host processor. The same processing chain was executed on a 900 MHz quad-core ARM Cortex-A7 CPU and a 2 GHz quad-core ARM Cortex-A15 CPU using OpenMP. Table IV shows the energy efficiency of the different platforms. The 100 MHz FPGA implementation of PNeuro is almost five times more energy-efficient than the multicore

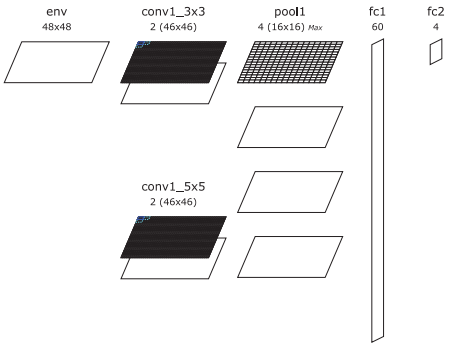

Fig. 5. Structure of the neural network for the classification application.

processors. ASIC simulations after synthesis show the interest of ASIPs compared to general-purpose CPU solutions.

TABLE IV
ENERGY EFFICIENCY OF DIFFERENT PLATFORMS EXECUTING A SMALL CNN APPLICATION.

Platforms	Frequency (MHz)	Energy efficiency (images/s/W)
PNeuro (FPGA)	100	2,000
PNeuro (ASIC estimation)	800	125,000
Quad-core ARM Cortex-A7	900	380
Quad-core ARM Cortex-15	2000	350

D. Scalability to large networks

In order to evaluate the scalability of PNeuro for larger networks, performance projections were performed using the Squeezenet neural network [18]. The Squeezenet DNN has a complexity of 739 MMACs and works with 224x224 input images. To execute this application, PNeuro internal memories must be enlarged to 64 KB per NCB and 16 KB per program memory. Thus, input images can be splitted in all the NCBs, enabling a full parallel computation. The optional connection between the two clusters is also used to synchronize them while the cluster controllers execute the same programming code.

The total area obtained after synthesis is about 1.76 mm^2 for an operational frequency of 800 MHz. The logic part of the clusters occupies 0.28 mm^2, whereas the memories need 1.48 mm^2. The total power consumption remains under 260 mW. By maximizing the data reuse along the different layers and then hidding most of data exchanges, the processing time was calculated at about 20.8 ms per input frame. As a comparison, an implementation of Squeezenet on a Nvidia Tegra X1 using cuDNN and 256 cuda cores reaches 11.1 ms while consumming approximately 8 W.

VI. CONCLUSION

We presented the PNeuro architecture, an accelerator for signal processing and bio-inspired deep networks. This scalable energy-efficient programmable hardware accelerator is specifically designed for the execution of DNN processing chains. PNeuro is made of several programmable clusters composed of processing elements dedicated to neural network implementations. It uses a smart data management, combining on-the-fly routing and transformation units as well as flexible and optimized computing datapaths. Our FPGA prototype and demonstrator is able to execute a real embedded image recognition application while maintaining a low enery consumption. The FDSOI 28 nm synthesis shows an energy efficiency of 700 GMAC/s/W at 800 MHz, consuming only 73 mW. We are currently working on a specific N2D2 code generation module in order to build a fully automatic binary generation flow to ease the programming of PNeuro.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, 1998, pp. 2278–2324.
[2] Tensorflow - an open-source software library for machine intelligence. [Online]. Available: https://www.tensorflow.org/
[3] Caffe 2 deep learning framework. [Online]. Available: https://caffe2.ai/
[4] L. Cavigelli, M. Magno, and L. Benini, "Accelerating real-time embedded scene labeling with convolutional networks," in *In ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.
[5] K. Ovtcharov, O. Ruwase, J. Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Toward accelerating deep learning at scale using specialized hardware in the datacenter," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, Aug 2015, pp. 1–38.
[6] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, February 2015, pp. 161–170.
[7] G. D. et al., "A 2.9 TOPS/W DeepConvolutional Neural Network SoC in FD-SOI 28nm forIntelligent Embedded Systems," in *IEEE Journal of Solid-State Circuits*, vol. PP, no. 99, November 2017, pp. 1–14.
[8] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in *IEEE Computer Vision and Pattern Recognition Workshop (CVPRW)*, June 2011.
[9] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014, pp. 269–284.
[10] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan 2017.
[11] SYNOPSYS DesignWare EV Processors for Embedded Vision. [Online]. Available: https://www.synopsys.com/designware-ip/processor-solutions/ev-processors.html
[12] CEVA-XM6. [Online]. Available: https://www.ceva-dsp.com/product/ceva-xm6/
[13] Intel Movidius Myriad X VPU. [Online]. Available: https://www.movidius.com/myriadx
[14] Google TPU. [Online]. Available: https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu
[15] CEA LIST. (2017) N2D2 (Neural Network Design and Deployement). [Online]. Available: https://github.com/CEA-LIST/N2D2
[16] C. Bechara, A. Berhault, N. Ventroux, S. Chevobbe, Y. Lhuillier, R. David, and D. Etiemble, "A small footprint interleaved multithreaded processor for embedded systems," in *IEEE International Conference on Electronics, Circuits, and Systems*, Dec 2011, pp. 685–690.
[17] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories," in *IEEE Workshop on Generative-Model Based Vision*, 2004.
[18] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," *arXiv:1602.07360*, 2016.