Exploring NEURAghe: A Highly Parameterized APSoC-based CNN Inference Accelerator

Paolo Meloni
Gianfranco Deriu
Daniela Loi
Marco Carreras
Luigi Raffo
Università di Cagliari
via Marengo 2
09123 Cagliari, Italy
paolo.meloni@diee.unica.it

Alessandro Capotondi
Davide Rossi
Università di Bologna
viale del Risorgimento 2
240136 Bologna, Italy
alessandro.capotondi@unibo.it
davide.rossi@unibo.it

Michele Brian
T3Lab
via Sario Bassanelli 9/11
40129 Bologna, Italy
michele.brian@t3lab.it

Francesco Conti
Luca Benini
ETH Zurich
Ramistrasse 101
8092 Zurich, Switzerland
fconti@iis.ee.ethz.ch
lbenini@iis.ee.ethz.ch

Abstract
The NEURAghe architecture has been presented in [3], as a powerful accelerator for Deep Convolutional Neural Network running on FPGA-assisted SoC devices like XILINX Zynq-7000. NEURAghe exploits at the same time the ARM-based processing system and the programmable logic in these kind of devices, to improve performance through parallelism and to widen the scope of CNN actors that can be supported, and, thus, the scope of complex algorithms that can be accelerated. To further improve flexibility, the architecture is highly parameterized. The general template can be customized at design time to fit in different SoCs of different size and costs, ranging from pricey high-end chips down to very low-cost and low-power entry-level devices. In this paper we present the general summary of NEURAghe's architectural features and report performance on different use-cases for different configurations to provide an overview of the available possibilities and prospective use-cases.

Author Keywords
FPGA; CNN inference; Zynq.

ACM Classification Keywords
500 [Computer systems organization]: Embedded systems; 100 [Hardware]: Reconfigurable logic and FPGAs.

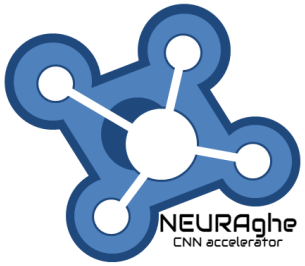

Figure 1: Logo of the NEURAghe processing architecture. The name of the accelerator template derives from the ancient megalithic edifices named *nuraghes*, typical of the prehistoric culture in Sardinia. The different configurations are named after most important *nuraghes*. https://en.wikipedia.org/wiki/Nuraghe

Introduction

Convolutional Neural Networks (CNNs) are one of the most promising classes of deep learning algorithms, due to remarkable performance achieved in a broad area of applications, ranging from speech recognition to computer vision and natural language processing. The execution of CNN algorithms involves a huge number of Multiply Accumulate (MAC) operations, representing the core of the convolution kernels. To accelerate these operations, a wide community of developers is exploiting reconfigurable logic inside modern FPGAs, taking profit of the perfect match between the intrinsic parallelism in the application and the significant number of MAC-supporting hardware primitives available/implementable in FPGA fabrics (e.g Xilinx DSP slices) [1, 2, 4, 5].

NEURAghe platform architecture

The NEURAghe architectural template is composed as a hierarchical structure that overlaps with the hardware organization in Xilinx Zynq SoCs (adaptable to other SoCs on the market). It contains a general-purpose processor (i.e. a hard wired processor corresponding to the ARM-based processing system in the Zynq, a dual ARM Cortex A9 processor in most cases), a memory-mapped off-chip DDR (as available on the Zynq), and a Convolution Specific Processor (CSP), hosted on the reconfigurable logic, acting as accelerator. Figure 2 illustrates the overall organization of NEURAghe. The accelerator is composed of the following major functional blocks:

- A RISC microcontroller (uC) executing a middleware synchronizing transfers and convolution computation.
- A Convolution engine (CE) that represents the computational core of the accelerator, with a private weight memory and several port to read the input pixels and write the results of convolution.

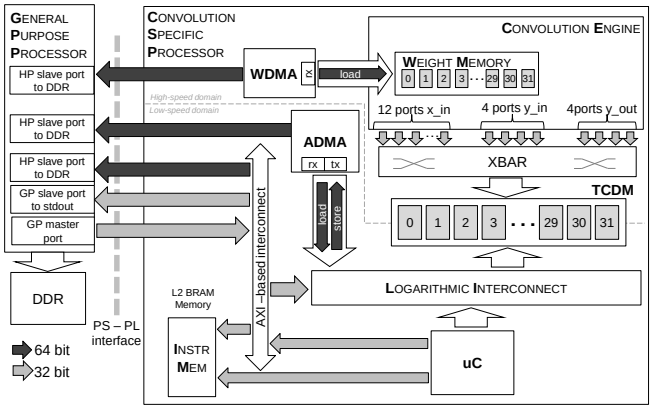

Figure 2: NEURAghe overall architectural template.

- A pair of Direct Memory Access (DMA), WDMA for the transfer of weights and ADMA for the transfer of the activations.
- A Tight Coupled Data Memory (TCDM) to store the activations and an instruction memory to store the code for the RISC controller.

The Convolution Engine embeds a set of Sum-of-Product (SoP) units, used to deploy convolutions on the reconfigurable logic, which are organized as a matrix of N columns and M rows. The SoP matrix is the most resource-consuming component and defines the number of input features (IF) maps and output features (OF) maps processed in each iteration.

Figure 3 shows the internal organization of the CE module, when a SoP matrix with 4 columns and 4 rows is used. We will refer to this configuration as LOSA.

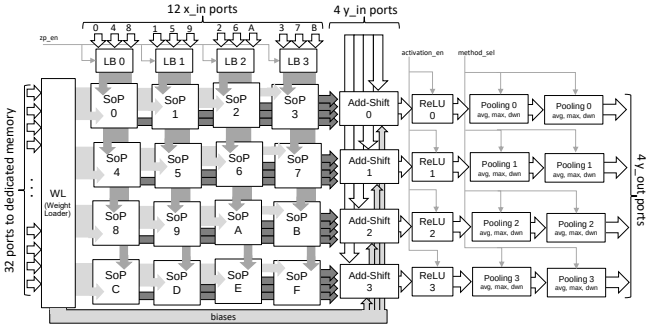

Figure 3: NEURAghe SoP matrix in LOSA configuration.

In each cycle of activity, the CE collects up to 12 input features and computes their contributions to 4 output features. The input features are loaded through a set of line buffers, indicated with LB in the diagram. The LB blocks are realized by means of shift registers and are used to cache the value of a few lines of the input image. In this way, by loading a single new pixel per cycle, an entire new window of the image can be dispatched to the SoP matrix to be convoluted with the weight filters. Considering 16-bit pixel data, each LB is fed with two pixels obtained from the input port and produces two square convolution windows per cycle. The convolution windows are then consumed by the SoP units. To mapping efficiency on the FPGA DSP resources, the SoP units can be configured at design time to be partitioned in multi-trellis structures of multiply and add operations, whose outputs are summed together using a dedicated Adder. The output pixels produced by each Adder-shifter module are then streamed into a ReLU (Rectifier Linear Unit) block, that, when enabled, performs rectifier activation function on each pixel. The CE block integrates also a pooling layer, implemented by means of a shift register, that temporarily stores output pixels and compares values of pixels in square pooling windows. After comparison, according to the selected operating mode, the pooling layer outputs one single pixel per window. Different types of pooling functions, implementing max pooling, average pooling or a simple down-sampling, can be chosen at design time.

Design-time configurable parameters

NEURAghe is designed to be fully parametric. Several parameters can be tuned by the designer to fit on specific target chips and to achieve the desired performance. In the following section, a description of each parameter is provided.

SoP matrix size

The SoP matrix is the main scalable computing module of the convolution engine. Designers can change the size of the matrix acting on the number of rows and columns that it contains, creating configurations that may fit on devices featuring different number of DSP slices. The number of rows and columns in the matrix define, respectively, the number of IF maps consumed by the engine and the number of OF maps produced by it.

Number of clusters

In NEURAghe, each cluster is independent by others and can have its own SoP matrix configuration. Thus, the RISC microcontroller can execute a private program enabling developers to map different layers on each cluster.

Data precision

The designer can choose to set data precision to 16 bits or 8 bits, to trade off accuracy for performance. All data (i.e. input pixels, output pixels, biases and weights) will be represented with the same number of bits. It is possible to instantiate support for run-time selection of the data precision.

	Price	DSP	BRAM
Z-7045	2500	900	545
Z-7020	450	220	140
Z-7010	100	80	60

Table 1: Price in dollars and FPGA resources available for the target Xilinx Zynq SoCs.

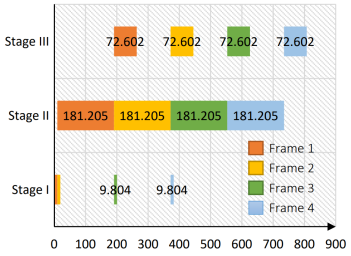

Figure 4: VGG-16 four frame execution timeline. Stage I consists on input data preparation, executed by the GPP. Stage II consists of all the computational blocks executed on the CSP. Stage III consists all the rest, mainly fully-connected layers, executed on the General Purpose Processor. Execution stages related to different frames can overlap.

Cluster memory size

The cluster hosts two main memories: the activations memory (AMEM) and the weights memory (WMEM). Both are served by dedicated DMA modules and are organized in banks. The memory size is limited by the total amount of BRAMs in the target device. Tiling and feature grouping are used to enable a chosen memory size to be used for the execution of arbitrary CNN layers.

Hardware implementation

As mentioned, the NEURAghe template can be implemented on different devices of the Xilinx Zynq-7000 SoC family, to conveniently select the best performance/power/cost trade-off for a specific use-case. As an example, we have tested implementation on three products, which integrate an ARM-based processing system with different sizes of FPGA circuitry: the Zynq Z-7045, the Zynq Z-7020 and the Zynq Z-7010, see Table 1. We chose three configurations of the NEURAghe architecture fitting respectively in the three target devices: LOSA (configured with a $4x4$ matrix of SoP modules), SABINA (featuring a $2x2$ matrix) and BANZOS (with one single SoP module). Moreover, we have implemented ARRUBIU, that fits on the Z-7045 SoC but has a different cluster-related organization, featuring two clusters with a $2x4$ matrix per cluster.

Performance Evaluation

Table 2 reports performance and efficiency levels achievable by the four configurations on several benchmarks, in terms of actual computational power (GOps/s), energy efficiency (GOps/sec per Watt), and price efficiency (GOps/sec per k$). As previously mentioned, NEURAghe exploits the interaction between the GPP and the CSP. As an example, you may notice in Figure 4, how different execution stages on different image frames overlap with each other in a task-level pipeline fashion.

Exploring SoP matrix size

While, obviously, implementations on more expensive devices reach higher absolute performance levels, smaller configurations, e.g. SABINA, expose a higher efficiency of SoP utilization with respect to peak performance, since smaller matrices work more effectively, especially in initial and final layers of common CNN models, see Figure 5.

Implementing more efficient configurations on low-cost hardware opens a whole range of possibilities, involving distributed processing and multi-board designs (e.g. distributed smart camera systems, near-sensor processing in sensor networks etc.). For example, if allowed by the use case, a designer may choose to replace a single LOSA with two SABINA configurations, to execute a task based on ResNet-18, achieving around 80% of the original performance, saving around 65% of costs.

Exploring number of clusters

Since smaller matrix can be more efficient, when allowed by the use case, multi-cluster configurations can be used to combine the benefits of a bigger device with those of a reduced matrix size. As shown in Table 2, the ARRUBIU configuration can be used to run VGG-16, alternately sending input frames to each of the clusters. In this way it is possible to improve the overall efficiency of the configuration.

Exploring data precision

In NEURAghe configurations processing 16 bit activation and weights, each DSP slice in the programmable logic performs one MAC operation per cycle. NEURAghe allows the programmer to select at runtime an operating mode processing 8 bit data, allowing to execute 2 MAC operations per cycle on each slice, prospectively improving performance by a factor of 2. As may be noticed in Table 2, when testing 8 bit operating modes in some of our configurations, measured speed-up is very close to the theoretical limit.

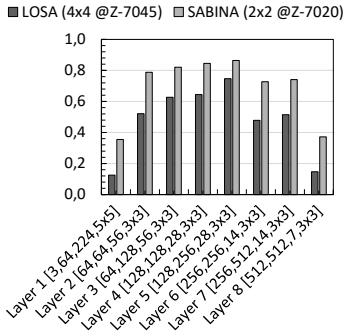

Figure 5: Comparison of the efficiency per peak achieved across all the convolution layers by LOSA and SABINA configurations on ResNet-18, estimated as the ratio between actual performance (GOps/s) and peak performance (GOps/s). For each layer, we reported four parameters: [number of input features, number of output features, input image size with 1:1 aspect ratio, kernel size]. As may be noticed, SABINA compensates some inefficiencies, especially in first layers (reducing mismatch between layer size and matrix size) and in last layers (weigth transfers are better handled by the DMA).

BANZOS: NEURAghe at the lowest cost
A 1x1 matrix implementation, providing very good performance over cost efficiency has been tested on lower complexity benchmarks, such as Squeezenet and Darknet.

	LOSA single 4x4	ARRUBIU dual 2x4	SABINA single 2x2	LOSA single 4x4	ARRUBIU dual 2x4	SABINA single 2x2	BANZOS single 1x1
Device	Z-7045	Z-7045	Z-7020	Z-7045	Z-7045	Z-7020	Z-7007s
DSP [#]; Freq [MHz]	864; 140	864; 140	216; 120	864; 140	864; 140	216; 120	54; 80
Benchmark net	ResNet-18	ResNet-18	ResNet-18	VGG-16	VGG-16	VGG-16	SqueezeNet
GOps/s (16 bit)	61.91	103	27.51	172.67	184	42.48	7.46
GOps/s per Watt (16 bit)	6.19	10.3	7.86	17.26	18.4	12.56	2.98
GOps/s per k$ (16 bit)	25	41	61.13	69	74	97.5	95
GOps/s (8 bit)	111.12	-	49.61	335.09	-	84.77	-

Table 2: Main features of different NEURAghe configurations.

Conclusion

In this paper we have presented a general overview of NEURAghe, a scalable architectural template for CNN acceleration on FPGA-based SoCs. Changing parameters of the NEURAghe architecture leads to a variety of configurations that may be used in different use-cases to fit in different devices, ranging from entry-level to high-end. This offers a vast set of trade-off optimization possibilities with respect to costs, computation efficiency and power consumption.

Acknowledgements

This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 780788.

REFERENCES

1. Michaela Blott, Thomas Preusser, Nicholas Fraser, Giulio Gambardella, Kenneth O'Brien, and Yaman Umuroglu. 2018. FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. (2018). https://arxiv.org/abs/1809.04570

2. Griffin Lacey, Graham W. Taylor, and Shawki Areibi. 2016. Deep Learning on FPGAs: Past, Present, and Future. (2016). https://arxiv.org/abs/1602.04283

3. Paolo Meloni, Alessandro Capotondi, Gianfranco Deriu, Michele Brian, Francesco Conti, Davide Rossi, Luigi Raffo, and Luca Benini. 2018. NEURAghe: Exploiting CPU-FPGA Synergies for Efficient and Flexible CNN Inference Acceleration on Zynq SoCs. *ACM Transactions on Reconfigurable Technology and Systems* 11, 3 (2018), 1–24. DOI: http://dx.doi.org/10.1145/3284357

4. Sparsh Mittal. 2018. A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computing and Applications* (2018). DOI: http://dx.doi.org/10.1007/s00521-018-3761-1

5. Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. 2017. DLAU: A Scalable Deep Learning Accelerator Unit on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 3 (2017), 513–517. DOI: http://dx.doi.org/10.1109/TCAD.2016.2587683