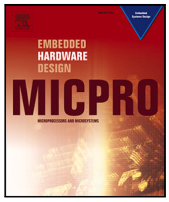
Microprocessors and Microsystems 98 (2023) 104824

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro



Task parallelism-based architectures on FPGA to optimize the energy efficiency of AI at the edge

Rafael Gadea-Gironés *, Jorge Fe, Jose M. Monzo

Institute for Molecular Imaging Technologies (I3M), Universitat Politècnica de València, Valencia, 46022, Spain

ARTICLE INFO

Keywords:
FPGA
Deep neural network
Hardware co-design
Energy efficiency
Systolic architectures

ABSTRACT

In the world of artificial intelligence (AI) at the edge, we need to focus primarily on the energy efficiency with which we approach deep neural network (DNN) applications. In many applications, the speed of obtaining an inference can be critical; but many applications easily meet their time requirements, and the energy needed to calculate the huge numbers of multiplication and addition operations of DNNs becomes the essential element. We have provided systolic architectural solutions written in C++ and OpenCL that are highly flexible and easily tunable to take full advantage of the resources of programmable devices and achieve superior energy efficiencies. We focused on low-cost solutions with soft macro microprocessors (Nios2) and hard macro microprocessors (ARM cortex A9).

1. Introduction

Bringing artificial intelligence (AI) to electronic devices is a growing need. This type of machine-learning solution could be used for many applications: Predictive maintenance of machines where the device can recognize its status, detect anomalies or loss of calibration, and act accordingly within a few milliseconds. Food sorting in the supply chains either by artificial vision or other types of sensing (impedance analyzers). Non-invasive medical instrumentation to determine anomalies in a scan can alter the patient's quality of life. Smoke and fire detection in forested areas. Hand gesture recognition using low-power radar, capacitive touch sensors, or accelerometers.

The choice of the electronic device will depend on many factors, but we will summarize them in three: performance, cost and a compendium of size, weight and power consumption (SWaP).

In this choice, field-programmable gate arrays (FPGAs) are an interesting alternative, but they usually have a fourth factor that can be detrimental, which is the difficulty of design. FPGA manufacturers provide a series of applications to their customers so that the developments made in frameworks dedicated to AI (i.e., Tensorflow, Caffe, Pytorch) and have been used to train these learning machines can easily transfer to the devices that will host the AI and perform the inference tasks. Although they communicate easily with training environments and allow heterogeneity of technological alternatives (CPU, GPU, FPGA), they lack architectural vision when dedicated to FPGA solutions.

For instance, OpenVINO (Intel FPGA), Vitis AI (AMD Xilinx), Lattice sensAI (Lattice), and Smart embedded Vision (Microsemi) provide a direct implementation for FPGA-based solutions. Developments from the first two manufacturers (Intel FPGA and AMD Xilinx) are not intended for low-cost devices such as the ones we focus on in this article (Cyclone V SoC and Zynq-7000 SoC).

In the opposite direction (bringing the hardware world closer to the software), the use of high-level compilers that can create hardware (based on FPGA) from a software language very close to C++ is also interesting. In this line, we have compilers that work from OpenCL, DPC++, C++, or SystemC. In principle, this option was not created with AI applications in mind, but it is now also the focus of many resources and studies.

Against this background, the contribution of this article is to put energy efficiency in the first place when assessing neural network implementations in inference actions. We want to demonstrate the validity of fully configurable FPGA-based solutions in which both the soft macro microprocessor and the accelerator coexist. The study will focus on low-cost devices that allow us to create intelligent, autonomous, and energy-efficient sensors.

Methodologically, we will use high-level compilers combined with the creation of register transfer level (RTL)-based libraries, thus combining the experience of low-level design and the power of new environments based on high-level synthesizers. FPGA, traditionally, has been programmed using low-level "Hardware Description Languages" (HDL) such as VHDL or SystemVerilog. However, in the last few years, "High-Level Synthesis" (HLS) tools have been developed and popularized. HLS tools convert behavioral C/C++ algorithms descriptions into low-level descriptions.

* Corresponding author.
E-mail address: rgadea@eln.upv.es (R. Gadea-Gironés).

https://doi.org/10.1016/j.micpro.2023.104824
Received 25 August 2022; Received in revised form 13 March 2023; Accepted 23 March 2023
Available online 27 March 2023

HLS tools infer parallelism from the sequential C/C++ described algorithms and exploit it to achieve higher throughput and low latency. HLS tools provide different options and pragmas to drive the C/C++ code conversion into RTL description. Proper use of these HLS options and pragmas is essential to achieve an optimal hardware implementation in speed and area.

Architecturally, our contribution will be based on the implementation of a systolic architecture with two main objectives: to exploit the advantage of implementing two chained layers of neurons through alternative projections that facilitate communication between them and to have the possibility of folding internally to be able to implement the number of layers we want without resorting to global memory and without the microprocessor intervening in the successive call.

The rest of this paper is organized as follows: AI-at-the-edge design infrastructures currently used with FPGA as the target technology are reviewed in Section 2. In Section 3, the elements used in the proposed methodology and architecture are determined. In Section 4, the obtained implementation results for a multilayer perceptron (MLP) are analyzed and compared with other available solutions. The conclusions are presented in Section 5.

2. State of the art

The first thing we must do in this review of the state of the art is focus clearly on the type of implementations we are interested in to not become lost in a huge number of implementations that have proliferated in recent years.

We are interested in DNN implementations that can be hosted on low-cost FPGA devices. We will focus primarily on those SOC-type technology families based on 28 nm process devices and will, of course, look for hardware–software co-design solutions where a microprocessor and accelerator hardware work collaboratively within the same package. We believe that this type of device is the most suitable to deal with AI at the edge.

In general terms, we are talking about FPGA devices with a limited number of arithmetic resources (mainly represented by hard macros called digital signal processors (DSP)) and whose raison d'être is far from the handling of floating point data types. These limited resources lead to solutions based on the reuse of an accelerator by means of successive calls exercised and controlled by a microprocessor to meet the sizes of existing networks within the DNN typology.

Of course, we will focus on solutions of the inference phase, and although we will demonstrate that we can change the type of data used in the operations, we will take as a base reference the operations in IEEE754 single precision. This last aspect is important because the handling of other types of data (fixed point of different precisions for activations, weights, pixels, etc.) would lead us to greatly complicate the goodness of the architectural solutions and to perform an exhaustive accuracy analysis of the solved applications (whether classifications, regressions, compressions, or predictions).

We begin our review of solutions with the work of Coutinho et al. [1]. They use processing units (PEs) that compute the basic neural operations, and from a weight communication point of view, they use a systolic architecture based on streamings. It is a low-level RTL implementation with very specific fixed-point data types. Its results are very good both in terms of throughput and energy efficiency, but the type of FPGA used is not low-cost, and its low energy efficiency is striking. A solution using the same type of device (Virtex 6) can be found at [2]. Also implemented with HDL and with impressive throughput performance (working with fixed points less than 18 bits), and although we cannot know its energy efficiency, we can suspect from its computational efficiency that it will not be good.

In a similar line of not using a low-cost device, we have the studies of Maria et al. [3]. In this case, very high values of MAC operations per second are obtained. The energy efficiency can be improved, but in their favor, we must consider that they strictly measure the power consumed by the host and the FPGA device. The use of OpenCL is introduced in this work, which we also do in some of our proposals. However, we use the existing ARM in the Cyclone V family as host so there is the possibility of performing the inference completely autonomously without the need for a host computer connected with a peripheral component interconnect (PCI). We cannot end this review of large FPGA devices without the study of Westby et al. [4], which is currently the fastest in processing a single digit image as it only requires 1.55 μs. The implementation is done in RTL Verilog, and although the network is an MLP with only one hidden layer (784-12-10), it achieves an accuracy of 93.5% and beats all records in throughput. However, the computational efficiency (it uses 604 DSPs) and energy efficiency results are not very convincing.

In the arena of low-cost family devices, we can highlight the studies of Acosta et al. [5] using a Cyclone IV. They use the largest device of the series, and their throughput results are very remarkable. Acosta et al. design It works with a 32-bit fixed point precision, and it is not possible to know both the energy and computational efficiency of the solution. The design flow is by means of a proprietary framework, as is that of Mazouz et al. [6], which must use the largest device size (Zynq 7100) within the Zynq FPGA family. The energy efficiency of this solution is moderate. It is clear in these examples that either high-performance families are used, or the largest devices of the families classified as low power and low cost are used, resulting in implementations that are far from being suitable for AI edge computing.

Considering solutions closer to our proposal, the works of Belabed et al. [7,8], Wang et al. [9] and Jorge Fe et al. [10] are worth mentioning. All use devices from the Zynq family; the first two work with a 7020, and the last one works with a 7010. The underlying idea is to reuse certain functional blocks using hardware–software co-design techniques to tackle deeper neural networks. The results [9] obtained with an RTL design flow are quite modest in terms of throughput values and computational and energy efficiency values. The results on [10] stand out when working with 32-bit fixed point; as we have already mentioned, fixed point solutions must be carefully evaluated for the effects on the limited precision of the solution. In the case of [7,8], we have the best results for a fixed point in terms of throughput, albeit at the cost of lower computational efficiency and energy efficiency based on unrealistic consumption figures.

3. Methodology and architecture

3.1. Methodology

The following methodological aspects must be described:

1. Generation of the systolic architectures: From this point of view, a classic systolization system is used;
2. Transfer the obtained architecture to C++ code compatible with the HLS compiler of Intel FPGA and to OpenCL compatible with the OpenCL compiler of Intel FPGA;
3. Function implementation refinement using RTL libraries.

Fundamental contributions in these three areas have already been provided by our group, as follows:

1. Systolic generation
    - The first contribution is the implementation of the dense layers in pairs rather than individually. Both multilayer perceptrons and the dense layers of CNNs require at least two layers based on matrix–vector multiplication. With this technique, we can improve the communication between layers by performing it internally in the developed IP.

- The second contribution is to demonstrate the implementation of a larger number of dense layers by means of a looped data flow within IP. With this approach, we have solved the MLP implementation of two hidden layers and one output layer or the final stages of CNN (LeNet, AlexNet, VGGNet).
- Projections are needed to allow several processing elements (PEs) to be executed by the same kernel. This fact will help save resources in the hardware implementation of the systolic array generation. The best projections are chosen based on their efficiency in layer-to-layer communication.

2. Implementation of the systolic architecture using C++ and OpenCL:

   - C++: Use of a system of tasks implementation, in which the projected processing units are assumed by a function with the property launch_always_run. From now on, this element will be referred to as the projected processing element (PPE). In the case of OpenCL, we use autorun kernels.
   - Use of functions for data entry and data extraction based on two techniques: buffers and queues.
   - Use of two control units from which it will distribute its control signals to the remaining functions (C++) or kernels (OpenCL).
   - Use of control flow and data flow through pipes or streaming interfaces (C++) and through channels (OpenCL) that will play a key role in the synchronization of all processing units.

3. RTL refinement:

   - Implementation of the activation functions by means of embedded memories, and
   - Implementation by direct instantiation of the operations output = A × B + C and Accu = A × B + Accu using the embedded DSPs.
   - Both techniques are common in low level, working with HDL and performing manufacturer's own ip instantiations. Incorporating these techniques in the flow with HLS and OpenCL is possible, but requires a correct parameterization according to the streaming interface handled by the RTL libraries used.

3.2. Systolic generation

A systematic method was applied for the design of systolic arrays [11] to obtain the architecture of two adjacent dense layers. This method is suitable for algorithms that can be expressed as a set of uniform recurrence equations on a convex set of whole D coordinates [11]. Once the recurring equations have been obtained, the method follows two steps: (1) finding a plan for the operations (schedule) that is compatible with the dependencies introduced by the equations, and (2) mapping the D domain within another finite set of coordinates. Each of these coordinates will represent a processor of the systolic array. As a consequence, the concurrent operations are mapped within different processors (allocation). The schedule and allocation functions must fulfill conditions that allow the method to be fully automated. This fact enables the transition to a systolic array of equations that intervene in the operation of the MLP.

The resulting architecture was obtained using different procedures than the ones shown in [12–14]. The latter procedures start from the dependency network and conduct a mapping operation on a systolic array. Projection operations in a certain direction are carried out for each PPE. The proposed architecture is called an "alternative orthogonal systolic array", and the complete method of extraction can be seen in [15],

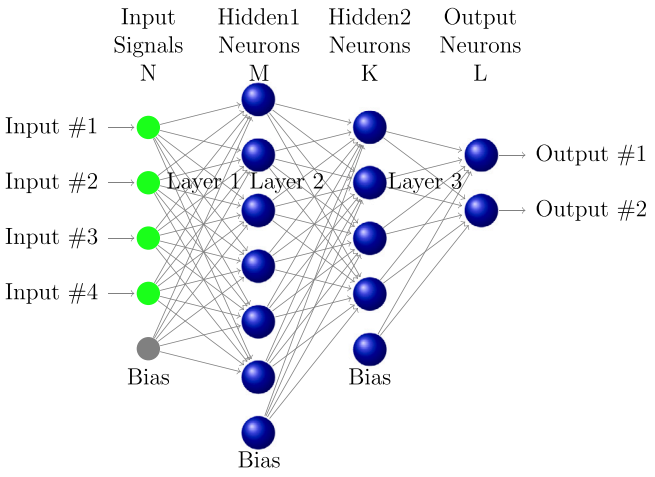


Fig. 1. Multilayer perceptron.

although it is not systematic and is, therefore, difficult to transfer to other cases. In the case presented herein, the systematic procedure seen in [11] is used. Even though the same configuration already obtained by Murtagh, Tsoi, and Bergmann is achieved here, this methodology can be executed for the computation of different configurations and can be applied to other types of problems (algorithms).

3.2.1. Multilayer perceptron

An MLP with the same layer structure as the one shown in Fig. 1 is used. In turn, this structure is shared by several CNNs in their final stage (LeNet, AlexNet, VGGNet).

Here, the layer connection between j (neuron source layer) and i (neuron destination layer) is discussed. Throughout the text, we will use the terms DESTINATION for the number of neurons in the target layer and ORIGIN for the number of neurons in the source layer.

Uniform recurrence equations. From Eqs. (1) and (2), uniform iterative Eqs. (3), (4), (5), (6), (7) and (8) can be obtained:

$$u_i^l = \sum_{j=0}^{ORIGIN+1} w_{ij}^l y_j^{l-1} \tag{1}$$

$$y_i^l = f(u_i^l) \tag{2}$$

$$1 \le i \le DESTINATION,\ 1 \le l \le L,$$

where y stands for activation, w for weight, f for nonlinear function, and j, i for the indexes in the source and destination layers.

1. Layer 1

$$layer1 \begin{cases} [l]x^l(i,j) = x^l(i,j-1) + w^l(i,j) \cdot y^{l-1}(i-1,j) \\ y^{l-1}(i,j) = y^{l-1}(i-1,j) \\ y^l(i,N+2) = f(x^l(i,N+1)) \\ 1 \le i \le M \\ 1 \le j \le N+1 \end{cases} \tag{3}$$

and the boundary conditions:

$$layer1 \begin{cases} y^{l-1}(0,j) = inputs \quad \text{for} \quad \text{j} = 1....\text{N} \\ y^{l-1}(0,N+1) = 1 \qquad \text{Bias} \\ x^l(i,0) = 0 \quad \text{for} \quad \text{i} = 1....\text{M} \end{cases} \tag{4}$$

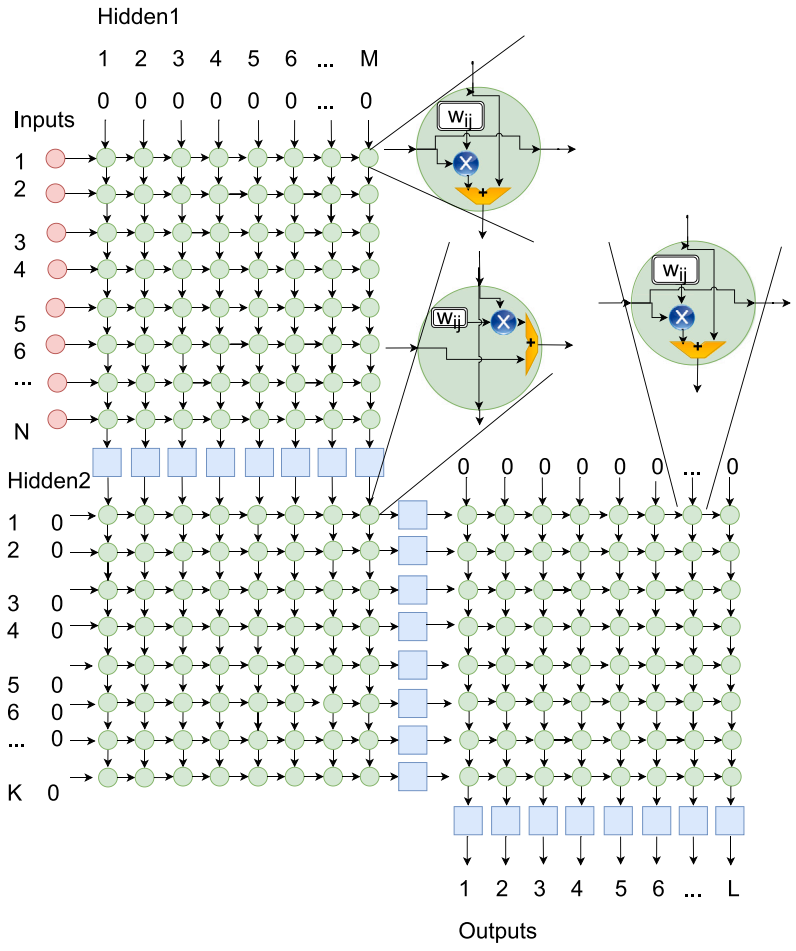


Fig. 2. Multi layer interconnection.

2. Layer 2

$$layer2\begin{cases} x^l(i,j) = x^l(i-1,j) + w^l(i,j)\cdot y^{l-1}(i,j+1) \\ y^{l-1}(i,j) = y^{l-1}(i,j+1) \\ y^l(M+2,j) = f(x^l(M+1,j)) \\ 1 \le i \le K \\ 1 \le j \le M+1 \end{cases} \quad (5)$$

and the boundary conditions:

$$layer2\begin{cases} y^{l-1}(i,0) = f(x^{l-1}(i,N+1)) \quad \text{for} \quad \text{i = 1....M} \\ y^{l-1}(M+1,0) = 1 \qquad \text{Bias} \\ x^l(0,j) = 0 \quad \text{for} \quad \text{i = 1....K} \end{cases} \quad (6)$$

3. Layer 3

$$layer3\begin{cases} x^l(i,j) = x^l(i,j-1) + w^l(i,j)\cdot y^{l-1}(i-1,j) \\ y^{l-1}(i,j) = y^{l-1}(i-1,j) \\ y^l(i,K+2) = f(x^l(i,K+1)) \\ 1 \le i \le L \\ 1 \le j \le K+1 \end{cases} \quad (7)$$

and the boundary conditions:

$$layer3\begin{cases} y^{l-1}(0,j) = f(x^{l-1}(M+1,j)) \quad \text{for} \quad \text{j = 1....K} \\ y^{l-1}(0,K+1) = 1 \qquad \text{Bias} \\ x^l(i,0) = 0 \quad \text{for} \quad \text{i = 1....L} \end{cases} \quad (8)$$

The geometry of the domain and the interconnection are shown in Fig. 2.

Resulting systolic array. The final result of the systolization procedure for layer 1 with a direction (1,0) projection is the systolic array shown in Fig. 3. The same applies to layer3 in Fig. 4.

- The number of functional blocks called "vertical synapses" equals the number of neurons in the source layer. Those functional blocks basically perform the function $F = A \times B + C$, and
- The number of functional blocks (neurons) is 1. This functional block implements the activation function.

The final result of the systolization procedure with a direction (1,0) projection of the systolic array of layer 2 is shown in Fig. 5.

The fundamental projection characteristics for an interconnection layer are:

- The number of functional blocks, called "horizontal synapses", equals the number of neurons in the target layer. Those functional blocks basically perform the function Accu = A ×B + Accu; and
- The number of functional blocks (neurons) equals the number of neurons in the target layer. This functional block implements the activation function.

3.3. Architecture

We will list the developed architectures.

1. V-H-V
   - Non flexible (VHV)
   - Flexible(F-VHV)

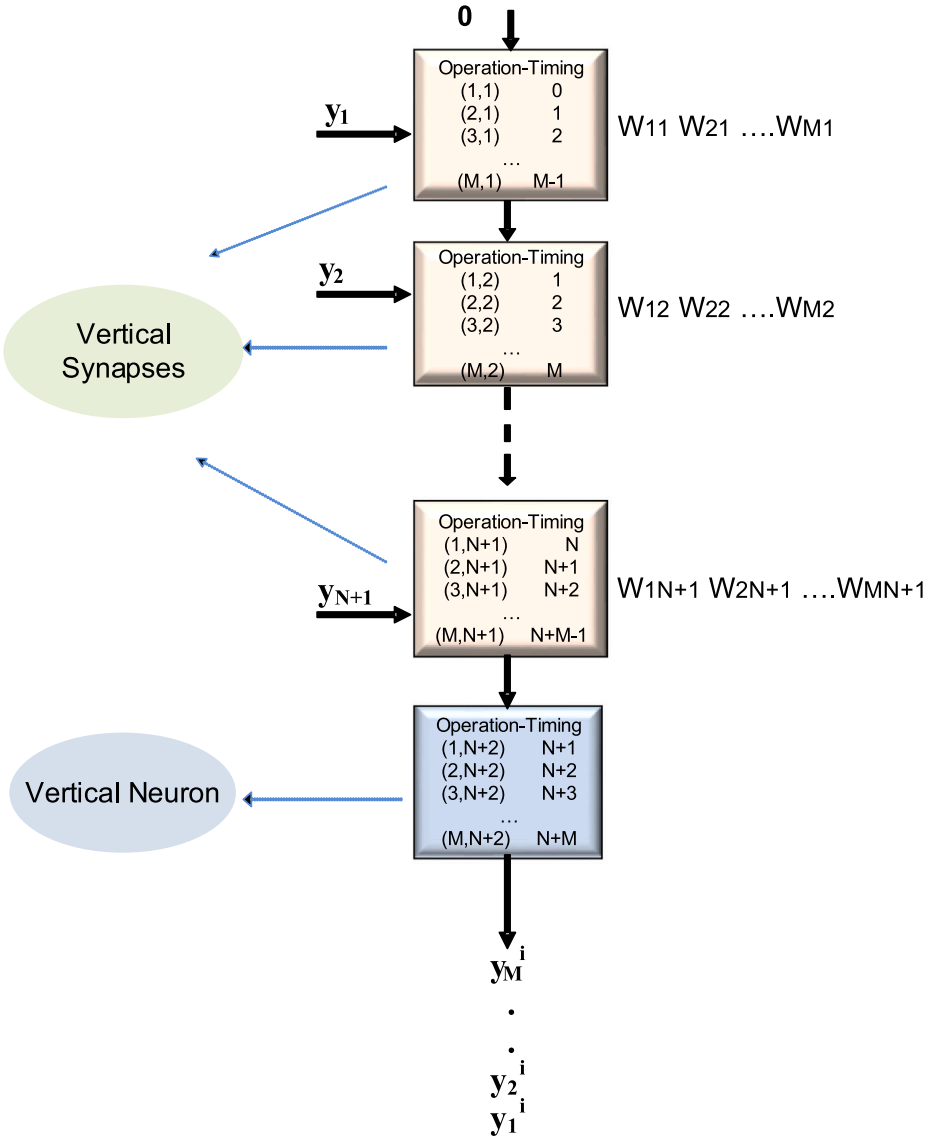


Fig. 3. Systolic array layer 1.

- Batch Flexible(BF-VHV)

2. V-H folded

- Non flexible (VH)
- Flexible(F-VH)
- Batch Flexible(BF-VH)

The most significant architectural solutions are described below.

3.3.1. *Alternative V-H-V*

The first proposed architecture is shown in Fig. 6.

These architectures are the combination of layers 1, 2, and 3 discussed in the previous subsection:

- All the data streams have been implemented by channels(OpenCL) or pipes(C++). Those channels perform two important tasks: the transmission of data and control signals and the synchronization of the systolic architecture. Each processing unit (synapses and neurons) can only operate when all of the data and control channels have data available. Otherwise, these processing units are blocked while queues (channels or pipes) are empty. Therefore, all of the processing units (synapses and neurons) act autonomously and do not need to be commanded by the host. The difficulty in this approach lies in making the systolic network adaptable to different numbers of inputs, outputs, and neurons in the hidden layers without having to hardware recompile the origin code (C++ or OpenCL). This task is performed by the control generation units that distribute control commands through the associated queues (see 7).
- The layers are constructed based on a BLOCK-SIZE parameter (Bs1 and Bs2 in Fig. 6). The Bs1 parameter is used several times to cover the number of inputs, and the Bs2 parameter is used to cover the number of hidden2 neurons, as shown in Fig. 6. In our experiments, both parameters will have the same value (BLOCK-SIZE), and given the size limitation of the used devices, we will test values of 8, 16, and 32, which respectively provide 24, 48, and 96 floating point DSPs. For a neural network such as the one depicted in Fig. 1, this architectural solution requires that the number of inputs and the number of neurons in the second hidden layer are multiples of BLOCK-SIZE.
- All the operations are in single precision format. We have also made implementations with other data types of 16 bits (FPHalf

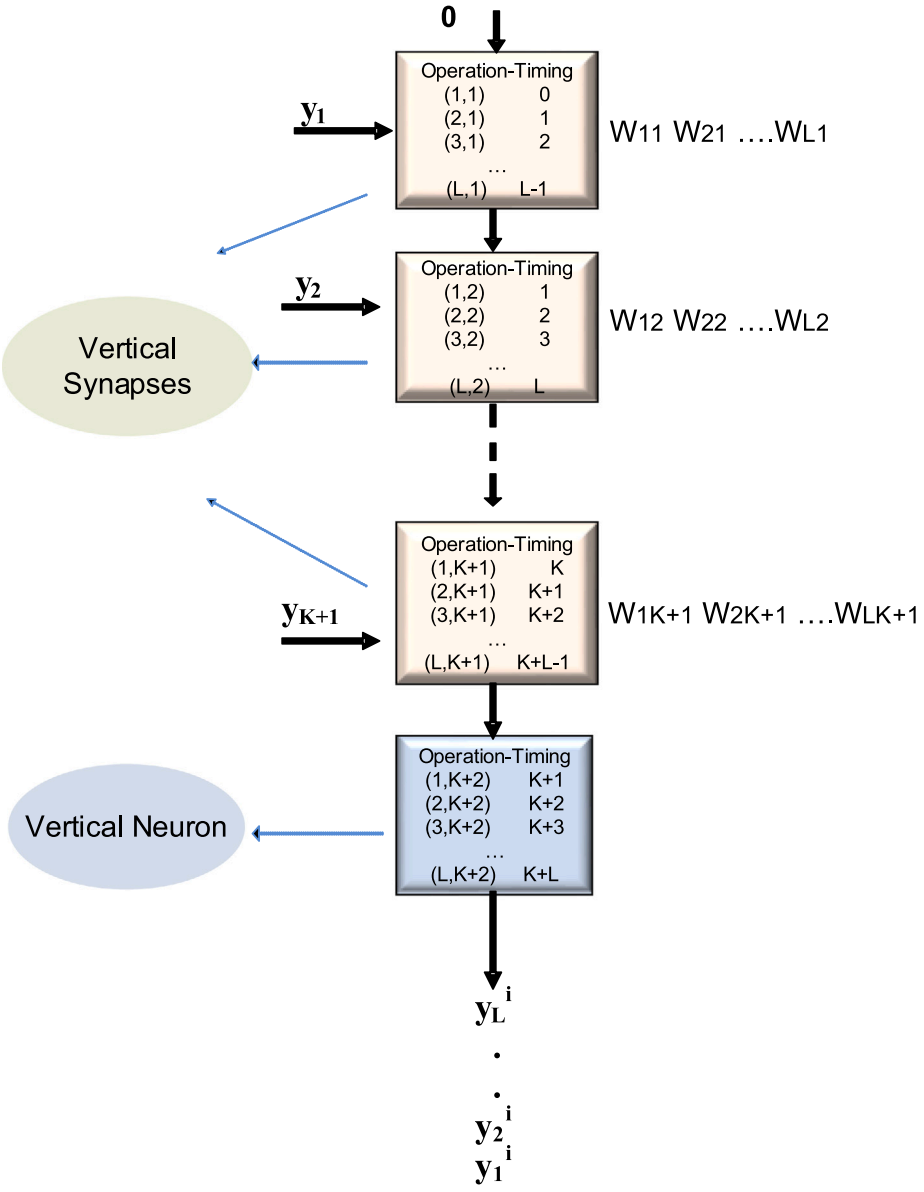


Fig. 4. Systolic array layer 3.

and Bfloa16) to evaluate occupancy and throughput values without loss of accuracy. The results are interesting, and we will include some of these architectures with these types of data.

- There is the possibility of a distributed control in each of the PPEs (see [16]) that would allow us to improve the performance characteristics. However, any change in the topology (changes in the number of inputs, outputs, and neurons of each layer) would force us to recompile the hardware.

3.3.2. Alternative V-H folded

The second proposed architecture is shown in Fig. 8.

- The main advantage of this architecture over the first one is the saving of resources. By making Bs1 and Bs2 the same, we can infer that layer 1 can be reused, making the datapath much simpler (because we save a layer) and also lightening the control, which is now governed only by two functions that handle four control flows instead of three (see Fig. 9).

3.3.3. Alternative flexible

As indicated above, the main drawback of the folded V-H-V and V-H alternatives is that the dimension of the number of inputs and the number of neurons of the second hidden layer must be multiples of Bs1 and Bs2, respectively. We needed a mechanism to overcome this limitation. Let us see this mechanism on an architecture V-H folded as an example; but it is equally applicable to the V-H-V architecture.

From a control point of view, we needed the control units in Fig. 11 to generate a new char type signal that we will call *rest*, which should be transmitted as shown in Fig. 11. Each processing unit has an ID that indicates its position in the systolic array so that when it is hit by a control signal resting with that identifier, it stops transmitting data and control line flow to the successive processing units. This technique is sufficient for the horizontal layer; however, for the vertical layer, it is necessary that each vertical synapse has a bypass mechanism to the vertical neuron (see Fig. 10). This bypass mechanism cannot be solved with a single channel or pipe because in both compilers (HLS and OpenCL SDK), these communication mechanisms can only be written or read from a single kernel (or function); therefore, it is necessary to have BLOCK-SIZE channels that must be read by the PPE vertical

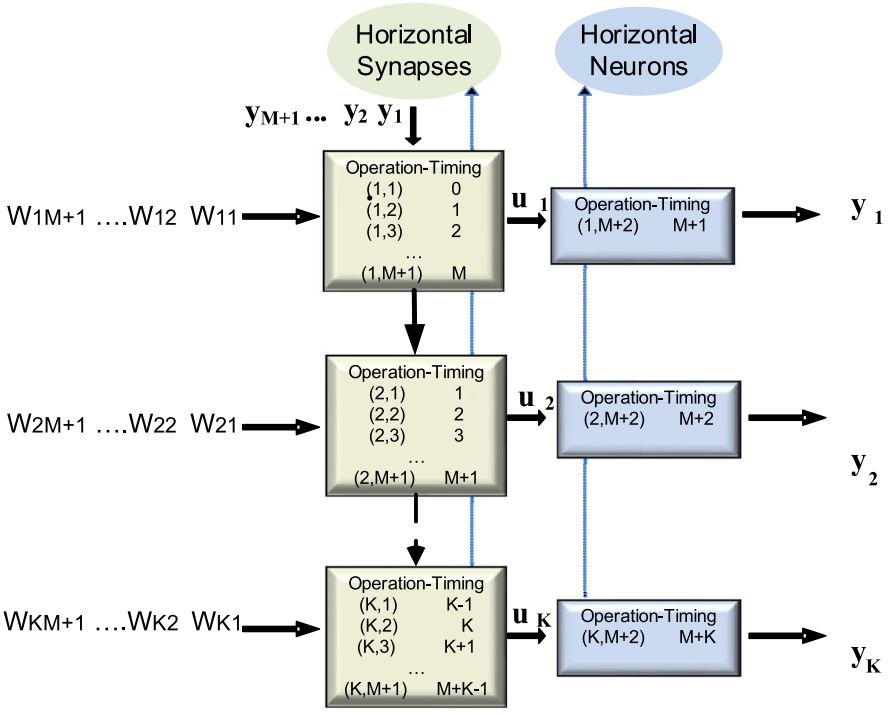


Fig. 5. Systolic array layer 2.

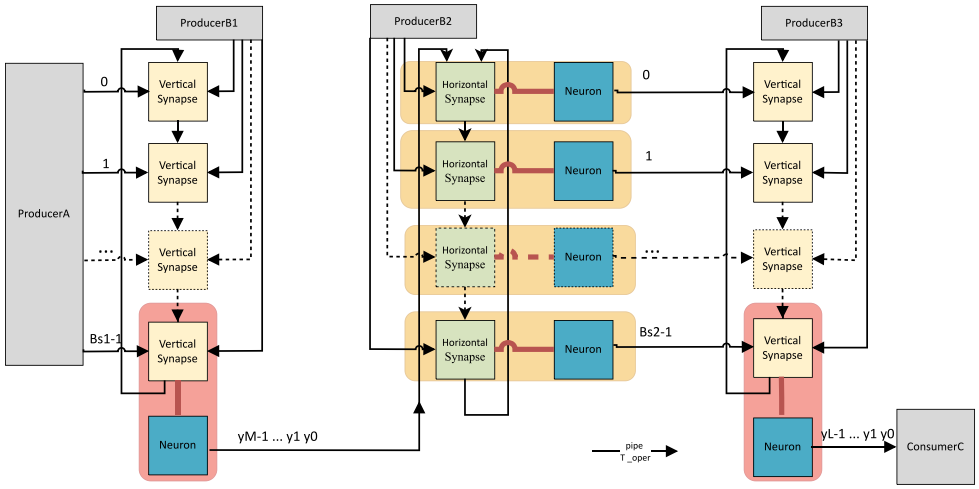


Fig. 6. Systolic architecture data flow with vertical–horizontal–vertical combination.

neuron through non-blocking reads with the intention of choosing the first channel that is not empty (see Algorithm 1).

The implementation details of the vertical and horizontal synapses can be seen in Fig. 12.

3.3.4. Alternative batch

The latter mechanism affects both the control blocks and the so-called "producer" blocks. The main idea is to take full advantage of the pipeline capabilities of the systolic architecture by introducing successive inputs into the input layer in a number that we will call from now on batch size. Evidently in the world of inference, this succession of inputs is not usual due to its dependence on the sensing and acquisition system. However, in the training world, it is quite common, given the mini-batch concept of most learning algorithms. Another reason that pushes us to this type of mechanism is to be able to adapt in the future this systolic architecture to convolutional layers in which the input instead of being a vector is a matrix. In any of the

Algorithm 1 Vertical Neuron

```
1:  exito ⇐ false;
2:  while 1 do
3:      while exito ≠ true do
4:          for i = 0 to BLOCK_SIZE do
5:              if exito = true then
6:                  input ⇐ read_channel_nb_intel(bypass[i], &success[i]);
7:              end if
8:              exito ⇐ exito or success[i];
9:          end for
10:     end while
11:     output ⇐ tanh(input);
12:     write_channel_intel(sal, output);
13:     exito ⇐ false;
14: end while
```

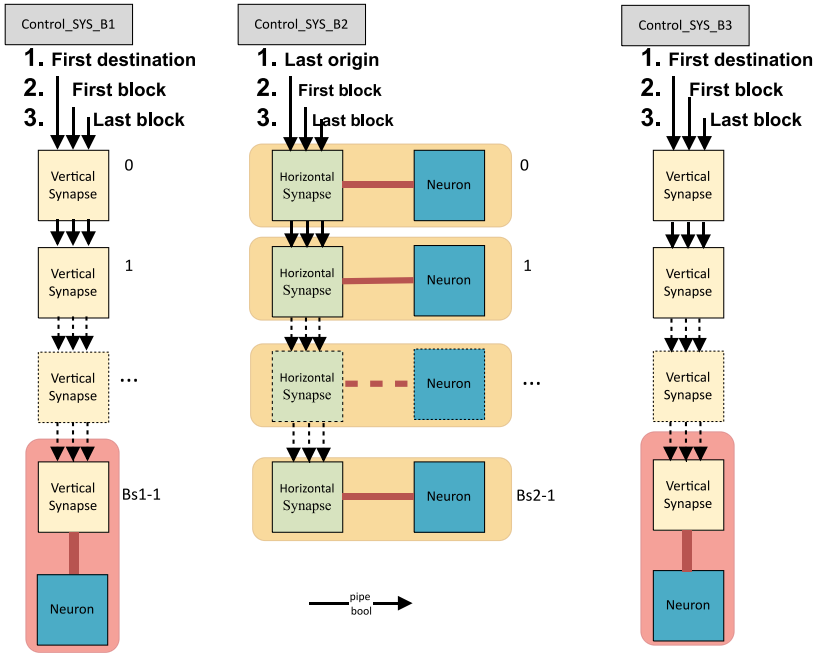


Fig. 7. Systolic architecture control flow with vertical–horizontal–vertical combination.

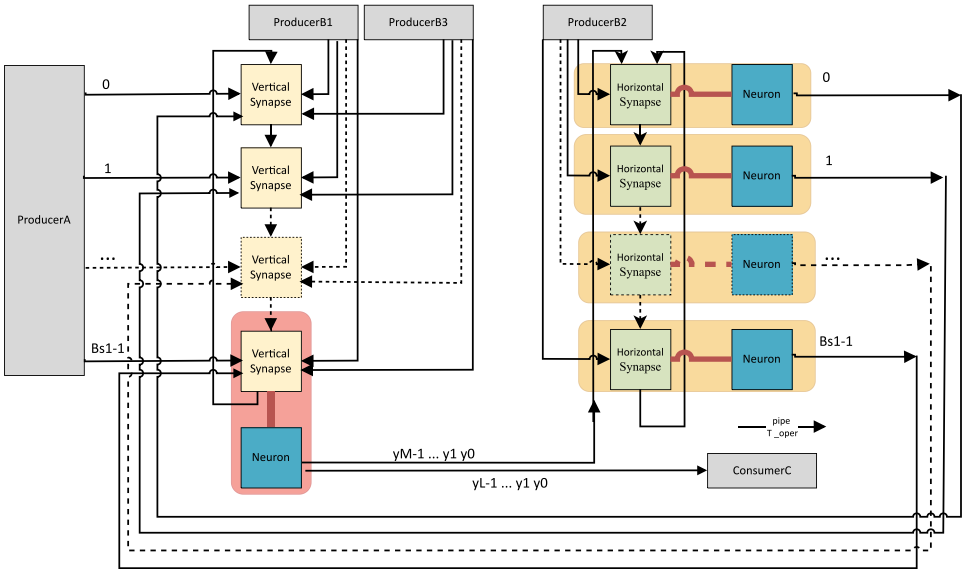


Fig. 8. Systolic architecture data flow with vertical–horizontal–folded combination.

cases we have checked in this way the throughput limits that can be reached by this type of architectures.

3.4. RTL tuning

3.4.1. Introduction of the nonlinear function inside the kernel

Our experience in designing backpropagation (BP) algorithm accelerators [17,18] along with other works [19] indicates that one of the most effective means of achieving acceleration is usually to replace the software implementation of the hyperbolic tangent with a hardware implementation.

The processing units depicted in Figs. 6, 8, and 10, called neurons, necessarily include the nonlinear activation function. The proposed hardware implementation uses lookup tables (LUTs) with a memory capacity that could be implemented using the Cyclone V devices' embedded memories. An organization of $2^{10}\times 20$ is used in the proposed implementation.

For instance, when the hyperbolic tangent is used as the activation function, a direct implementation through the HLS compiler or OpenCL compiler is highly unsatisfactory in terms of area and speed. To observe this effect, we will take a vertical layer like the one in Fig. 6 with BLOCK_SIZE = 3, and we will implement it using the libraries hls_float_math using Eq. (9) or our RTL IP (LUT based). Latency results are obtained when this vertical layer implements layer 3.

$$tanh = (e^{2x} - 1)/(e^{2x} + 1) \tag{9}$$

Our results regarding the resources being used and speed performance (latency) are shown in the row labeled "RTL IP tanh" in Table 1. This approach is outstanding in terms of area and speed; moreover, it has no impact on the inference mode of the trained MLP from the precision point of view.

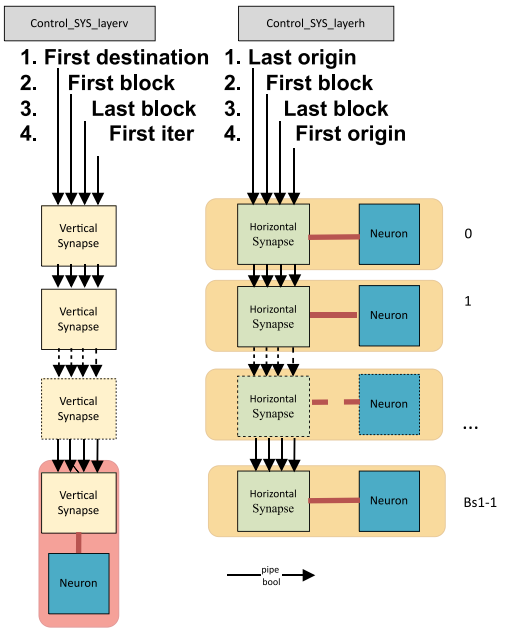


Fig. 9. Systolic architecture control flow with vertical–horizontal–folded combination.

Table 1
Comparison between the implementations of the vertical layer.

	ALM	Register	Memory 20 Kb	MLAB	DSP Blocks	Latency Cycles
HLS tanh	7568	15941	86	86	23	1123
RTL IP tanh	4803	9421	56	46	5	1061

Table 2
Resource comparison between $F = A \times B + C$ implementations of the vertical layer.

	ALM	Register	Memory 20 Kb	MLAB	DSP Blocks	Latency Cycles
HLS inference	4803	9421	56	46	5	1061
RTL IP	5407	11005	57	47	5	1177

3.4.2. Instantiation of DSPs versus direct inference

The same procedure used for the implementation of hyperbolic tangent functions inside PPE neurons was applied to PPE synapses to achieve further refinement based on RTL libraries written in Verilog-SystemVerilog.

Results in Table 2 demonstrate that for the $F = A \times B + C$ function's implementation, the RTL code-based instantiation versus HLS code inference has no remarkable impact in terms of efficiency. We can see the results in Table 2, where we show the implementation differences of the vertical layer with size BLOCK_SIZE=3.

However, in the case of the function $Accu = A \times B + Accu$, the number of resources saved by using direct instantiations of horizontal synapses is remarkable. The major contribution of these IPs is to perform the operation with a throughput of 1. In the case of more modern and powerful technologies such as Arria 10, the internal DSPs can be configured to achieve this property natively when working with floating point and single precision. However, in the case of the Cyclone V family that is the subject of this work, the only way to achieve it is with IP RTL instantiation in HLS by means of a fixed-point accumulator kernel. This leads to problems in layers with a very large number of source neurons (as in the case of the first interconnection layer with 784 neurons).

An overall improvement in both area and performance can be observed for alternative flex V_H folded with 784-100-50-10 topology (Tables 3 and 4). The improvements in terms of performance are not remarkable, but in the case of resources, it may mean that our accelerator may or may not fit in our device.

Table 3
Comparison between two implementations of MLP (784-100-50-10) with flex V-H folded architecture and BLOCK-SIZE = 8.

	ALM	Register	Memory 20 Kb	MLAB	DSP Blocks	Latency Cycles
HLS inference	27465	55020	246	275	22	42119
RTL IP	19186	36721	145	214	22	37813

Table 4
Performance comparison between two implementations of MLP (784-100-50-10) with flex V-H folded architecture and BLOCK-SIZE = 8.

	Thr[a] Mps	Thr[b] FPS	Energy efficiency mW/Mps	Comput. efficiency Mps/DSP
HLS inference	199.197	2374.2	5.72	9.05
RTL IP	221.881	2644.6	4.91	10.085

[a]Throughput in mega parameters per second.
[b]Throughput in frames per second.

4. Results and performance evaluation

4.1. Performance evaluation

In AI-at-the-edge applications, system latency and, derivatively, application throughput are often some of the most requested performance metrics.

Because most studies focus on computer vision applications, it is not surprising that frames per second is widely used as the throughput measurement. This measure has been used in the present work to compare our results with other works, but it is essential to keep in mind that this measurement parameter is very dependent on the network topology, which includes the input layer that would have the size of the starting image.

It is more interesting and independent of the topology to calculate the throughput as a function of the number of network parameters calculated per second, given that it is generally understood that each synaptic connection involves one parameter (one MAC operation).

However, it should be noted that results using these throughput metrics are often masked by the programmable device architecture characteristics (granularity, number of variables per LUT, and DSP characteristics), and, fundamentally, by resources available and used in the chosen FPGA. To avoid this drawback, our solution's computational efficiency goodness is measured as network parameters per second per DSP (which corresponds to a MAC operation on almost all manufacturers' FPGA families). Comparing solutions on the same device family will be a very valuable measure, but comparing solutions between families or between vendors is almost meaningless considering the different characteristics of the DSPs.

Nevertheless, whether these operations are done in fixed or floating point, and exactly the type of representation used should be made clear. We can evaluate the accuracy impact at different accuracies, but it would always depend on the application used.

Energy efficiency is also very important when AI-at-the-edge processing is performed in limited battery capacity embedded devices [20]. Energy efficiency is often reported as the number of operations per joule, but in this work, mW/Mps, or energy per operation, is used. It is a very significant value when comparing different manufacturers. The only problem with this measurement is knowing how the power has been evaluated (estimated or measured); the values found in many works are estimates of the design environments with FPGAs (Vivado or Quartus), and it can often be difficult to discern what has been done with the power consumed by the PS in SOCs.

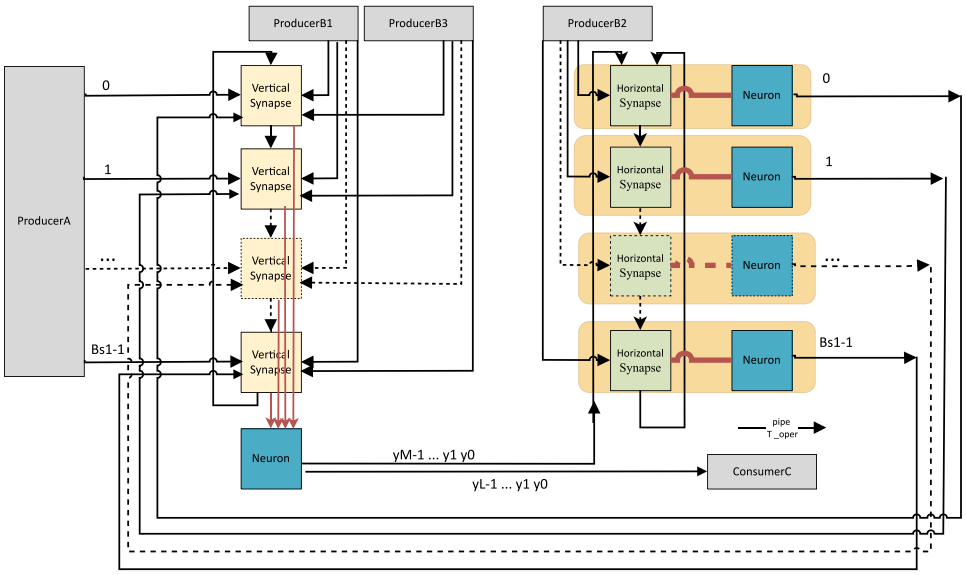


Fig. 10. Systolic architecture data flow with flexible vertical–horizontal–folded combination.

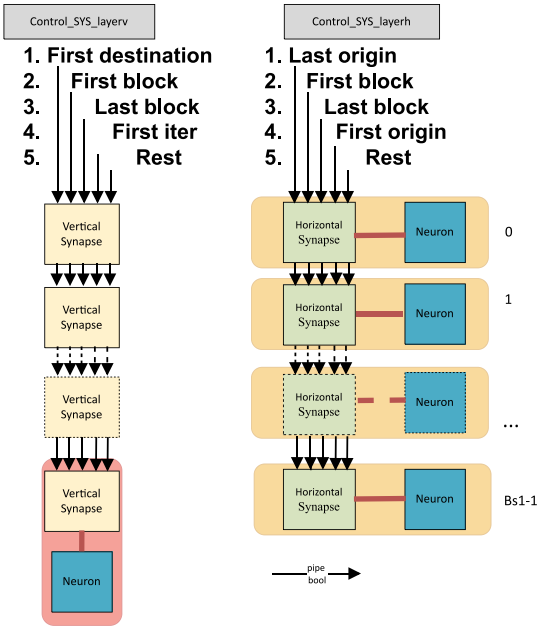


Fig. 11. Systolic architecture control flow with flexible vertical–horizontal–folded combination.

Finally, it is necessary to include two last aspects that our implementation perfectly fulfills and that cannot be ignored: flexibility and architectural adaptation. Some other implementations are unbeatable at the previous metrics but at the cost of hardware time compilation for these specifically implemented topologies. This means that any topology change requires a time-consuming hardware recompilation. Some implementations with dependency on group size (as the NDrange kernel-based OpenCL implementations), even if they do not require recompilation, have an efficiency that is very dependent on the exact size of the layers. These implementations, derived from fantastic implementations of matrix multiplication, are very efficient with a batch of input vectors, but in inference, such an input situation is very unrealistic.

4.2. Results and discussion

First, we will establish a comparison with the implementations reviewed and referenced in the literature (Table 5). We will highlight the implementation identified as *VH*, which represents our solution with the alternative flexible V-H folded 3.3.3 architecture. This implementation has been achieved with a BLOCK = 8. We achieve the highest computational efficiency working with 32-bit floating point precision. In terms of energy efficiency, it is only surpassed by the work of Coutinho [1] and Belabed [7] but has throughputs 50% lower.

At this point, it is important to establish that many of these results are obtained with the tools of the design environment: in our case, an Intel HLS compiler in collaboration with Quartus (to obtain the implementation results and power analysis) and with Questasim (to obtain the clock cycles necessary to drive the designed accelerator). Therefore, we are talking about results obtained by cosimulation and establishing that the input of 784 inputs is unique (there is no batch).

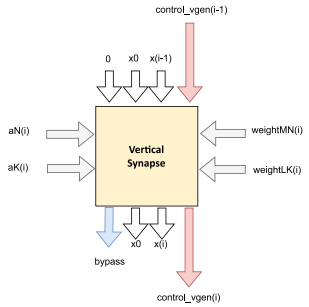
In Table 6 we already introduce performance values obtained in hardware verifications of the proposed architectures. For this purpose we build a complete system in which the proposed accelerator is one of many necessary elements: memories, embedded, external memory controllers, profiling peripherals (performance counters) and, above all, the system that controls the accelerator (a microprocessor). These results show situations that only physical verification can provide and that are often masked in the results of the co-simulators. These circumstances can be:

- The implementation does not fit directly into the device.
- The latency values obtained by co-simulation by the HLS compiler are reduced when working with the final system.
- Power consumption values are not complete.
- The use of inputs in batch mode.

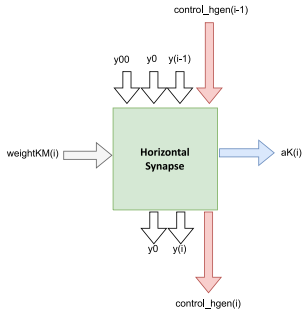
An example of the first point above is included in Table 5. The implementation identified as *VHV* is the most energy and computationally efficient and has been achieved with variants of the alternative V-H-V 3.3.1 architecture that allow us to have a different Bs1 and Bs2 (16-8). In addition, the granularity of each PPE of the vertical layer has been modified to implement Eq. (10).

$$result = A \times B + C \times D + F \tag{10}$$

However, when we introduce the whole system (accelerator together with the microcontroller and necessary peripherals), it is not possible to fit this implementation into the device.



(a) Vertical Synapse



(b) Horizontal Synapse

Algorithm 2 Vertical Synapse

```
1: while 1 do
2:     aux ⇐ read_channel(control_vgen[i]);
3:     First_dest ⇐ aux.control1;
4:     First_block ⇐ aux.control2;
5:     Last_block ⇐ aux, control3;
6:     First_iter ⇐ aux.control4;
7:     Rest ⇐ aux.control5;
8:     if First_iter = true then
9:         weight ⇐ read_channel(weightMN[i]);
10:    else
11:        weight ⇐ read_channel(weightLK[i]);
12:    end if
13:    if i = 0 then
14:        if First_block = true then
15:            operand1 ⇐ read_channel(x0);
16:        else
17:            operand1 ⇐ 0.0;
18:        end if
19:    else
20:        operand1 ⇐ read_channel(x[i − 1]);
21:    end if
22:    if First_iter = true & First_dest = true then
23:        operand1 ⇐ read_channel((aN[cols]);
24:    else
25:        operand1 ⇐ read_channel(aK[cols]);
26:    end if
27:    result ⇐ operand2 ∗ weight + operand1;
28:    if i < Bs − 1 then
29:        if Rest ≠ i then
30:            write_channel(x[cols], result);
31:            write_channel(control_vgen[i + 1], aux);
32:        else
33:            write_channel(bypass[i], result);
34:        end if
35:    else
36:        if First_block = true then
37:            write_channel(bypass[i], result);
38:        else
39:            write_channel(x0, result);
40:        end if
41:    end if
42: end while
```

Algorithm 3 Horizontal Synapse

```
1: while 1 do
2:     aux ⇐ read_channel(control_hgen[i + 1]);
3:     Last_origin ⇐ aux.control1;
4:     First_block ⇐ aux.control2;
5:     Last_block ⇐ aux, control3;
6:     First_origin ⇐ aux.control4;
7:     Rest ⇐ aux.control5;
8:     weight ⇐ read_channel(weightKM[i]);
9:     if i = 0 then
10:        if First_block = True then
11:            operand2 ⇐ read_channel(y00);
12:        else
13:            operand2 ⇐ read_channel(y0);
14:        end if
15:    else
16:        operand2 ⇐ read_channel(y[i − 1]);
17:    end if
18:    if i < Bs − 1 then
19:        if Rest ≠ i then
20:            write_channel(y[i], operand2);
21:            write_channel(control_hgen[i + 2], aux);
22:        end if
23:    end if
24:    if i = 0 then
25:        if Last_block = Falsei then
26:            write_channel(y0, operand2);
27:        end if
28:    end if
29:    if RTL = False then
30:        acc ⇐ operand2 ∗ weight + acc;
31:    else

32:        acc ⇐ myAcc(First_origin, operand2, weight);
33:    end if
34:    if Last_origin = True then
35:        temp2 ⇐ my_tanhFPsingle(acc);
36:        write_channel(aK[i], temp2);
37:        if RTL = False then
38:            accu ⇐ 0;
39:        end if
40:    end if
41: end while
```

Fig. 12. Detail of the synapses.

For the second and third situations, we include Table 6. In this table, we observe different implementations of the MNIST problem on board DE10-nano, equipped with a Cyclone V of size A6.

In this table, we seek to evaluate our solution with the HLS compiler using the Intel soft macro (NIOS2) as host and with the OpenCL SDK Compiler using the ARM cortex-A9 as host. In either case, the intervention of the microprocessor is merely a testimonial to the inference work of the neural network because the only thing it does is configure the accelerator. In both cases, due to the size of the weights, the DDR3 memory must be shared with the microprocessor. In HLS, the reading of the weights will be in charge of a soft DMA defined in the accelerator, and in the case of OpenCL, we will make use of the existing base infrastructure in the board support package of the board manufacturer in which the action of the existing hard DMA in the processing subsystem (PS) stands out. There is a considerable loss of performance in both cases. In the case of working with NIOS2 and the Avalon master defined in the accelerator HLS itself, we include some implementation derived from version *VH* in which we have modified the characteristics of the interface with the RAM and the sequentiality of the weight readings by changing the granularity of the processing units (row 2 version *VH-256*) to achieve better results. However, the results are still far from the performance estimated by the high-level synthesis environment. This degradation of measured versus co-simulated throughput makes more conventional solutions, such as the one in row 4, better in computational efficiency and similar in energy efficiency, even though the throughput is far from 836.44 Mps, which is our best result for the NIOS2-Systolic array with HLS.

Undoubtedly, the main problem in the evaluation of energy-efficient architectures for neural network inference is the evaluation of the power consumed. Many authors use the estimates of the FPGA synthesis and implementation tools without mentioning the value of the toggle rates used and how they were obtained. In many of the articles reviewed, no mention is made of the power consumed by the control element, even if it is in the same package as the FPGA, and of course,

Table 5
Comparison with other implementations.

	Topology	Device	Clk MHz	Data types	Thr[a] Mps	Thr[b] FPS	Energy efficiency mW/Mps	Comput. efficiency Mps/DSP
R. Acosta et al. [5]	LeNet-5	CycloneIV-115LE	100	32-bit Fixed	368.28	926		
Westby et al. [4]	784-12-10	Kintex Ultrascale	100	32-bit Float	6147.1	645161	92.72	10.17
C. Wang et al. [9]	784-256-256-10	Zynq 7020	200	32-bit Float	3.346	12.45	69.93	
Maria et al. [3]	3072-2000-750-10	Stratix V		32-bit Float	345.34	45	46.33	
Suzuki et al. [2]	24-10-24	Virtex 6	242	18-bit Fixed	856.5	1784375		1.142
Mazouz et al. [6]	1-2-4	Zynq 7100		16-bit Fixed	21.93	526	26.35	
Coutinho et al. [1]	784-100-50-10	Virtex 6	100	12-bit Fixed	105.06	1250	2.855	
Belabed et al. [8]	784-100-50-10	Zynq 7020	100	32-bit Float	97.948	1160	3.89	2.129
J. Fe et al. [10]	784-100-50-10	Zynq7010	100	32-bit Fixed	213.14	2540	1.24	21.31
J. Fe et al. [10]	784-100-50-10	Zynq7010	100	32-bit Float	45.84	545.4	9.69	9.63
Our CycloneV RTL(VHV)	784-100-50-10	Cyclone V A6	100	32-bit Float	416.26	4922	2.63	10.954
Our CycloneV RTL (VH)	784-100-52-10	Cyclone V A6	100	32-bit Float	225.86	2671	4.46	10.266
Our CycloneV RTL(VHV)	784-100-52-10	Cyclone V A6	100	16-bit Float	462.177	5465.67	2.13	12.162

[a]Throughput in mega parameters per second.
[b]Throughput in frames per second.

the power consumption measured at the chip or board level is rarely included. To make a quick overview of this situation, we focus on raw 5 of Table 6. If we do not include the power consumed by the ARM, it seems a faster, more computationally and energy efficient solution than the same solution (architecture *VH*) implemented with NIOS2 (raw 1). If we include the power consumed by the PS, the energy efficiency does not improve the performance obtained with a NIOS2 and that accelerator architecture (52.38 mW/Mps versus 34.97 mW/Mps). However, we have measured the power consumption at the board level and it seems evident that the ARM consumption, even when not used, makes this board with this embedded hard macro lose energy efficiency.

Regarding the fourth aspect mentioned above, it is where we can observe the most promising results. The one in row 7 of Table 6 stands out above all of them, where the 3.26 mW/Mps exceeds, working with 32-bit float and performing power estimation, any work referenced in Table 5. This estimated power consumption then corresponds to about 15 mW/Mps measured at the board level. This aspect corroborates the notion that the implemented systolic architecture reaches its full efficiency when we can take advantage of all its pipeline features. In order to explore the performance characteristics of our BF-VH and BF-VHV architectural solution, we have modified the sizes of the different layers and the Batch to see how it behaves with different topologies and compared it with the ARM solution with the Neon SIMD extension (see Figs. 13–16). Our base topology on which we have made variations has been 784-100-50-10 with a batch size of 2000, and all these implementations have been tested and measured on the DE10-nano board. Several interesting aspects can be observed in these figures:

- The BF-VH and BF-VHV architectures compared have similar characteristics from a throughput point of view and also from an energy efficiency point of view. In both architectures, there is a performance drop when the number of inputs or the number of neurons of the second hidden layer are not multiples of the corresponding BLOCK_SIZE. These results may make the necessity of the folded architecture debatable. It uses less resources but this circumstance does not seem in principle exploitable because implementations with higher BLOCK_SIZE are only efficient if they are suited to the characteristics of the connection port with the global RAM (8 and 16 for example) and unfortunately with BLOCK_SIZE of 16 not fit in the programmable device working with 32-bit floating point.
- It is also evident that RTL tuning is absolutely necessary to extract all the possibilities of the proposed architecture. We had already anticipated this with the results of the Table 4; but now it is much more evident. If we let the HLS compiler infer the use of DSPs, the throughput drops below even by using the ARM with its Neon SIMD extension. This behavior can be seen in the Figs. 13–16 in the implementation called FB VH bs=8 without RTL.
- Finally, we would like to highlight the characteristics of our architecture working with half-precision floating point format (fp16). With this precision and with the DE10-nano board we have surpassed 1000 Mps and energy efficiency below 6 mW/Mps as can be seen in Fig. 15. In this case, the usefulness of the folded architecture (FB V-H) has been demonstrated, since it is the only one that allows an implementation with BLOCK_SIZE of 16 for both the vertical and horizontal layers.

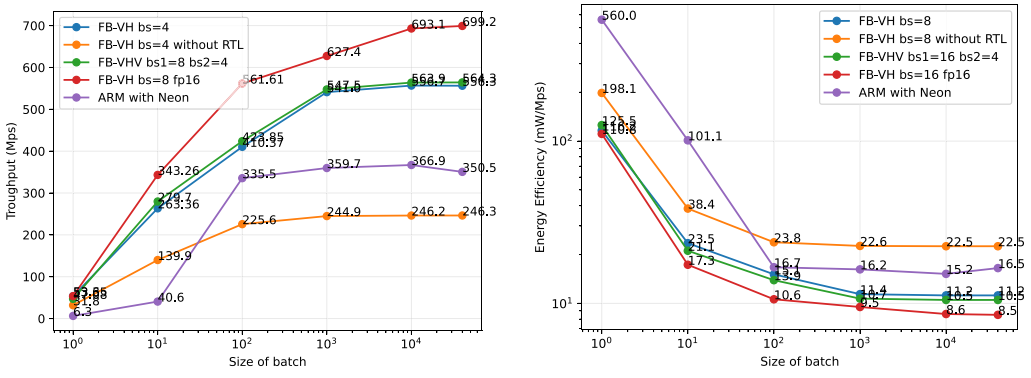
5. Conclusions

In the article, we have developed an architecture based on task parallelism that allows us to achieve efficient systolic architectures for

Table 6
Comparison of implementations.

	Version	Topology batch size	Clk MHz	Data types	Thr[a] Mps	Thr[b] FPS	Energy efficiency mW/Mps	Comput. efficiency Mps/DSP
1	Our NIOS2 CycloneV (VH)	784-100-52-10 1	100	32-bit Float	35.83	423.72	34.97[c] 109.69[e]	1.628
2	Our NIOS2 CycloneV (VH-256)	784-100-52-10 1	100	32-bit Float	69.157	826.44	22.16[c] 96.88[e]	1.686
3	Our NIOS2 CycloneV (conventional)	784-100-52-10 1	100	32-bit Float	37.80	450.450	23.25[c] 97.98[e]	6.30
4	Our NIOS2 CycloneV (BF-VH)	784-100-50-10 10	100	32-bit Float	70.150	836.44	21.60[c] 95.08[e]	2.598
5	Our ARM CycloneV (VH)	784-100-52-10 1	105	32-bit Float	57.40	678.886	27.86[c] 52.38[d] 87.10[e]	2.609
6	Our ARM CycloneV (BF-VHV)	784-100-50-10 1	103	32-bit Float	47.08	561.14	35.08[c] 64.7[d] 169.8[e]	1.8
7	Our ARM CycloneV (BF-VH)	784-100-50-10 2000	103	32-bit Float	527.59	6288.31	3.28[c] 5.92[d] 15.02[e]	20.29
8	Our ARM CycloneV (BF-VHV)	784-100-50-10 2000	103	32-bit Float	510.17	6080.69	3.24[c] 5.975[d] 15.68[e]	19.62
9	ARM	784-100-50-10 2000	—	32-bit Float	21.17	252.32	299[e]	—
10	ARM Neon	784-100-50-10 2000	—	32-bit Float	360.1	4292.01	17.16[e]	—

[a]Throughput in mega parameters per second.
[b]Throughput in frames per second.
[c]Predicted power without a PS.
[d]Predicted power with a PS.
[e]Power measured at the board level.



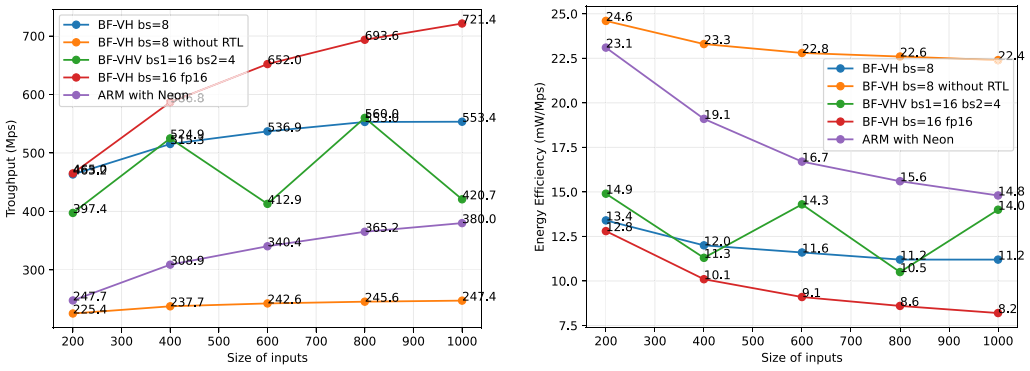
(a) Throughput (b) Energy Efficiency

Fig. 13. Throughput and Energy Efficiency with batch size changes.

the implementation of MLP-type networks with two hidden layers in a simple way based on a block_size parameter. We have achieved a measured energy efficiency of 15 mW/Mps at the board level that has not been achieved so far in other work. The systolic architectures have two remarkable particularities: the ability to have a folded structure and to achieve total flexibility in the topology of each of the layers. All this is achieved by means of a completely systolic data and control flow.

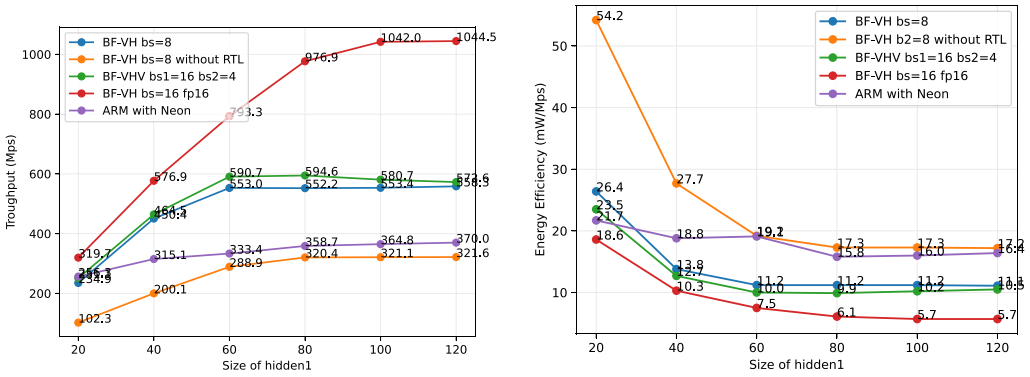
The possibilities of developing these architectures both in C++ based on communications through streaming and pipes and in OpenCL based on channels have been demonstrated. We have optimized the use of the corresponding compilers through the use of RTL libraries written in HDL that make optimal use of the existing DSPs in the programmable device and optimal implementation of the nonlinear activation functions in this type of network.

From a co-design point of view, we have opened a line of discussion on the use of hard macro and soft macro microprocessors that we wish to deepen in future work by means of RISC-V or similar. We have also not forgotten to explore new floating-point accuracies (fp16) and even though the DSPs of the technology family evaluated were not primarily designed for this type of data, we have achieved throughputs of 1000 Mps.



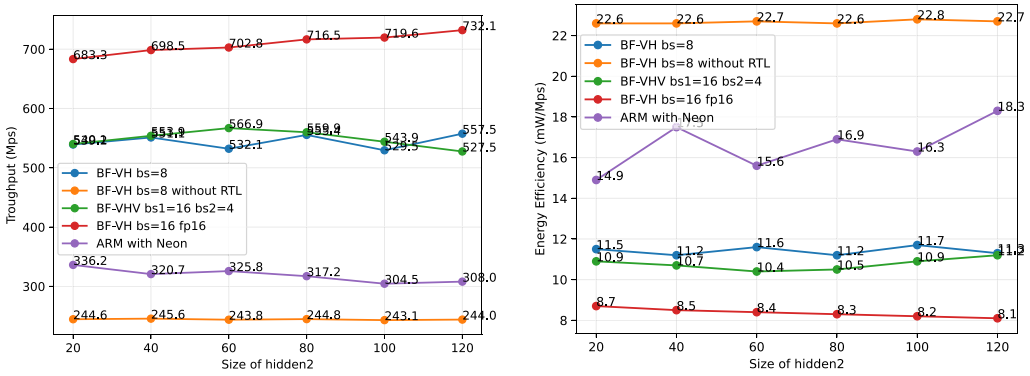
(a) Throughput (b) Energy Efficiency

Fig. 14. Throughput and Energy Efficiency with input size changes.



(a) Throughput (b) Energy Efficiency

Fig. 15. Throughput and Energy Efficiency with hidden1 size changes.



(a) Throughput (b) Energy Efficiency

Fig. 16. Throughput and Energy Efficiency with hidden2 size changes.

Finally, we try to indicate that energy efficiency must be one of the priority objectives in the application of AI at the edge; the means used to measure the values involved must be described in more detail than those in most of the existing contributions.

Funding

This research received no external funding.

CRediT authorship contribution statement

Rafael Gadea-Gironés: Conceptualization, Software, Methodology, Validation, Investigation, Writing – original draft, Writing – review & editing. **Jorge Fe:** Software, Methodology, Writing – review & editing. **Jose M. Monzo:** Methodology, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] M.G. Coutinho, M.F. Torquato, M.A. Fernandes, Deep neural network hardware implementation based on stacked sparse autoencoder, IEEE Access 7 (2019) 40674–40694, http://dx.doi.org/10.1109/ACCESS.2019.2907261.

[2] A. Suzuki, T. Morie, H. Tamukoh, A shared synapse architecture for efficient FPGA implementation of autoencoders, PLoS One 13 (3) (2018) http://dx.doi.org/10.1371/journal.pone.0194049.

[3] J. Maria, J. Amaro, G. Falcao, L.A. Alexandre, Stacked autoencoders using low-power accelerated architectures for object recognition in autonomous systems, Neural Process. Lett. 43 (2) (2016) 445–458, http://dx.doi.org/10.1007/s11063-015-9430-9.

[4] I. Westby, X. Yang, T. Liu, H. Xu, FPGA acceleration on a multi-layer perceptron neural network for digit recognition, J. Supercomput. 77 (12) (2021) 14356–14373, http://dx.doi.org/10.1007/s11227-021-03849-7.

[5] M. Rivera-Acosta, S. Ortega-Cisneros, J. Rivera, Automatic tool for fast generation of custom convolutional neural networks accelerators for FPGA, Electronics 8 (6) (2019) http://dx.doi.org/10.3390/electronics8060641, URL https://www.mdpi.com/2079-9292/8/6/641.

[6] A. Mazouz, C.P. Bridges, Automated offline design-space exploration and online design reconfiguration for CNNs, in: 2020 IEEE Conference on Evolving and Adaptive Intelligent Systems, EAIS, 2020, pp. 1–9, http://dx.doi.org/10.1109/EAIS48028.2020.9122697.

[7] T. Belabed, M.G.F. Coutinho, M.A.C. Fernandes, C.V. Sakuyama, C. Souani, User driven FPGA-based design automated framework of deep neural networks for low-power low-cost edge computing, IEEE Access 9 (2021) 89162–89180, http://dx.doi.org/10.1109/ACCESS.2021.3090196.

[8] T. Belabed, V. Ramos Gomes da Silva, A. Quenon, C. Valderamma, C. Souani, A novel automate python edge-to-edge: From automated generation on cloud to user application deployment on edge of deep neural networks for low power IoT systems FPGA-based acceleration, Sensors 21 (18) (2021) http://dx.doi.org/10.3390/s21186050, URL https://www.mdpi.com/1424-8220/21/18/6050.

[9] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, X. Zhou, DLAU: A scalable deep learning accelerator unit on FPGA, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 36 (3) (2016) 513–517.

[10] J. Fe, R. Gadea-Gironés, J.M. Monzo, Á. Tebar-Ruiz, R. Colom-Palero, Improving FPGA based impedance spectroscopy measurement equipment by means of HLS described neural networks to apply edge AI, Electronics 11 (13) (2022) http://dx.doi.org/10.3390/electronics11132064, URL https://www.mdpi.com/2079-9292/11/13/2064.

[11] P. Quinton, The systematic design of systolic arrays, in: Centre National de Recherche Scientifique on Automata Networks in Computer Science: Theory and Applications, Princeton University Press, USA, 1987, pp. 229–260.

[12] H. Kung, Leiserson, Systolic Arrays for (VLSI), CMU-CS- Carnegie-Mellon University, Department of Computer Science, 1978, URL https://books.google.es/books?id=pAKfHAAACAAJ.

[13] D.I. Moldovan, On the design of algorithms for VLSI systolic arrays, Proc. IEEE 71 (1) (1983) 113–120, http://dx.doi.org/10.1109/PROC.1983.12532.

[14] M. Zargham, Computer Architecture: Single and Parallel Systems, in: Prentice Hall International Series in, Prentice Hall, 1996, URL https://books.google.es/books?id=_7d5QgAACAAJ.

[15] P. Murtagh, A.C. Tsoi, N. Bergmann, Bit-serial array implementation af a multilayer perceptron, IEEE Proc.-E 140 (5) (1993) 277–288.

[16] R. Gadea-Gironés, V. Herrero-Bosch, J. Monzó-Ferrer, R. Colom-Palero, Implementation of autoencoders with systolic arrays through openCL, Electronics 10 (1) (2021) http://dx.doi.org/10.3390/electronics10010070, URL https://www.mdpi.com/2079-9292/10/1/70.

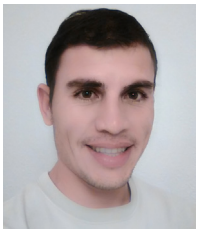
[17] R. Gadea-Gironés, V. Herrero, A. Sebastia, A.M. Salcedo, The role of the embedded memories in the implementation of artificial neural networks, in: Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications, FPL '00, Springer-Verlag, London, UK, 2000, pp. 785–788, URL http://portal.acm.org/citation.cfm?id=647927.739394.

[18] R. Gadea-Gironés, R. Colom-Palero, V. Herrero-Bosch, Optimization of deep neural networks using SoCs with openCL, Sensors (Switzerland) 18 (5) (2018) http://dx.doi.org/10.3390/s18051384.

[19] P. Kumar Meher, An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks, in: 2010 18th IEEE/IFIP International Conference on VLSI and System-on-Chip, 2010, pp. 91–95.

[20] V. Sze, Y.-H. Chen, T.-J. Yang, J.S. Emer, How to evaluate deep neural network processors: TOPS/W (alone) considered harmful, IEEE Solid-State Circuits Mag. 12 (3) (2020) 28–41, http://dx.doi.org/10.1109/MSSC.2020.3002140.

Rafael Gadea-Gironé is currently an Associate Professor with Universitat Politecnica de Valencia, Valencia, Spain. He has authored/coauthored around 45 refereed papers in journals and high-ranked conferences, has guided 5 doctoral theses, and has participated in more than 30 research projects. His research interest focuses on microelectronic design, implementation of artificial neural networks, design and verification of FPGA-based systems. GadeaGirones has a Ph.D. in industrial engineering from' Universitat Politecnica de Valencia. Contact him at rgadea@eln.upv.es.



Jorge Fe is currently a Ph.D. candidate from Universitat Politecnica de Valencia. His research interests include artificial intelligence, digital signal processing, embedded systems, reconfigurable hardware and the implementation of artificial neural networks in FPGA-based systems. Contact him at jorfe@posgrado.upv.es.

Jose M. Monzo is currently an Associate Professor with Universitat Politecnica de Valencia, Valencia, Spain. He has authored/coauthored around 30 refereed papers in journals and high-ranked conferences, and has participated in more than 16 research projects. His research interest focuses on FPGA based data acquisition systems for instrumentation, digital signal processing, and microelectronic design. Monzo has a Ph.D. in telecommunication engineering from Universitat Politecnica de Valencia. Contact him at jmonfer@upvnet.upv.es.