electronics

MDPI

Article

Improving FPGA Based Impedance Spectroscopy Measurement Equipment by Means of HLS Described Neural Networks to Apply Edge AI

Jorge Fe †, Rafael Gadea-Gironés *,†, Jose M. Monzo †, Ángel Tebar-Ruiz † and Ricardo Colom-Palero †

Institute for Molecular Imaging Technologies (I3M), Universitat Politècnica de València, 46022 Valencia, Spain; jorfe@posgrado.upv.es (J.F.); jmonfer@upvnet.upv.es (J.M.M.); atebar@upvnet.upv.es (Á.T.-R.); rcolom@eln.upv.es (R.C.-P.)

* Correspondence: rgadea@eln.upv.es

† These authors contributed equally to this work.

Abstract: The artificial intelligence (AI) application in instruments such as impedance spectroscopy highlights the difficulty to choose an electronic technology that correctly solves the basic performance problems, adaptation to the context, flexibility, precision, autonomy, and speed of design. Present work demonstrates that FPGAs, in conjunction with an optimized high-level synthesis (HLS), allow us to have an efficient connection between the signals sensed by the instrument and the artificial neural network-based AI computing block that will analyze them. State-of-the-art comparisons and experimental results also demonstrate that our designed and developed architectures offer the best compromise between performance, efficiency, and system costs in terms of artificial neural networks implementation. In the present work, computational efficiency above 21 Mps/DSP and power efficiency below 1.24 mW/Mps are achieved. It is important to remark that these results are more relevant because the system can be implemented on a low-cost FPGA.

Keywords: FPGA; impedance spectroscopy; artificial neural networks; high-level synthesis; AI edge computing

Citation: Fe, J.; Gadea-Gironés, R.; Monzo, J.M.; Tebar-Ruiz, Á.; Colom-Palero, R. Improving FPGA Based Impedance Spectroscopy Measurement Equipment by Means of HLS Described Neural Networks to Apply Edge AI. *Electronics* **2022**, *11*, 2064. https://doi.org/10.3390/electronics11132064

Academic Editor: Akash Kumar

Received: 20 May 2022
Accepted: 27 June 2022
Published: 30 June 2022

1. Introduction

Impedance Spectroscopy is an instrumentation technique that measures the electrical complex impedance of a biological system when it is excited with different frequency voltage signals. Due to its electrical nature, the physiological behavior and state of a biological component directly affect its impedance. For this reason, a precise impedance measurement on a biological system gives us information on the internal physiological processes variations.

Impedance spectroscopy is a non-invasive, fast, low cost and portable technology. Due to these properties, it is a perfect technique to be applied in the industry to measure food quality and in healthcare and biomedical applications. The technique has been successfully applied in different studies. At food industry, it has been used for quality inspection on fruit [1,2], fish and meat [3,4], and beverages [5,6]. In healthcare and biomedical applications, it has been used in cardiography, cancer diagnosis, tissue analysis, and biosensing [7,8].

Adapting the technique to industrial environments is one of the current impedance spectroscopy challenges. To perform this, the technique should be fast, low-power, reliable, adaptable to detect different quality parameters on various foods, and easily upgradable to improve detection with new algorithms.

To achieve the above objectives, the present paper proposes using an FPGA as the main signal processor on the impedance spectroscopy instrumentation and AI algorithms at the edge to improve quality detection.

FPGA-based instrumentations have several notable properties. They are compact and portable, cross-platform compatible, have low latency, are reconfigurable, and allow edge processing and machine learning hardware acceleration. Therefore, FPGAs have been integrated into different research areas' instrumentation equipment [9]. Regarding Impedance Spectroscopy, FPGA-based systems can be found in different studies [10–13].

The typical FPGA-based impedance spectroscope is composed of an FPGA, one Data Acquisition Converter (DAC), an analog front-end, sensor probes, and two analog to digital converters (ADCs). The measurement process requires two stages: sample excitation and measurement. At the excitation process, the FPGA generates a signal that is converted into a voltage or current through the DAC and the analog front-end. This signal is applied to the sample through the sensor. In the measurement process, the sensor and the analog front-end measure a voltage and the current on the excited sample. These two measures are digitalized using the two ADCs. FPGA processes these signals to obtain the impedance phase and modulus. This process can be repeated several times using different frequency excitation signals. In the end, the system has impedance modulus and phase at the sample for different excitation frequencies.

Several FPGA-based impedance spectrometers are based in commercial data acquisition boards [10,14]. These boards are composed of one FPGA and several high-speed ADCs and DACs. FPGA configuration can be modified with custom-made hardware adapted to achieve the designed system performance specifications. These spectrometers are made up of the commercial board, together with a custom-made analog front-end and the sensor. The presented paper system is based on a commercial data acquisition board Red Pitaya STEMlab 125-14 [15].

It is important to remark that a third process should be performed over the measured information, data analysis. Usually, data analysis is complex and performed outside of the instrumentation system [16]. Several statistical methods such as principal component analysis, linear discriminant analysis, ANOVA, least-squares analysis, and several machine learning methods such as logistic regression, support vector machine, and K-nearest neighbors have been applied successfully to extract information from the measured data [16]. Also, there are some works that have successfully applied data analysis based on Artificial Intelligence (AI) techniques [13,17].

Artificial Neural Networks (ANN) are efficient algorithms to classify and perform estimations over impedance spectrum measured data. It is a methodology that can be implemented in industrial FPGA-based instrumentation systems that require automated analysis [18].

Applying data analysis and performing estimations and classification inside the measurement instrument without or reducing access to cloud services is known as edge computing. Using FPGA-based systems to perform AI edge computing has several advantages over other AI edge computing architectures such as the GPU based. These advantages include providing a throughput independent of the workload, achieving high performance for complex concurrent AI structures, and obtaining better energy efficiency than other non-FPGA-based architectures. On the other hand, developing hardware on FPGA-based edge AI systems is complex for most programmers [19].

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 popularised. HLS tools convert behavioral C/C++ algorithms descriptions into low-level descriptions.

HLS tools infer parallelism from the sequential C/C++ described algorithms and exploit it to achieve higher throughput and low latency. Although you can directly convert any C/C++ program or software function into a synthesizable RTL code, this is not optimal. 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.

HLS tools focus designers on what they need to do instead of specifying how to implement it for a given reconfigurable hardware [20]. These tools reduce the FPGA hardware developing time. Consequently, HLS described systems could be upgraded easily, improving their adaptability.

Using HLS descriptions to design the impedance FPGA-based spectrometer data analysis block is a proper solution to develop a system that can be easily upgraded with new algorithms to improve parameter detection over measured data. Considering ANN, these algorithms require several iterations to process the data. The operations applied to process the data are matrix multiplication or matrix-vector multiplication. The performed calculations involve applying several nested loops to the data.

The HLS compiler will only infer task-level parallelism from function calls. The sequential code blocks (such as loops) which need to be run concurrently in hardware should be put into dedicated functions. One of the desired objectives is to implement data processing algorithms, minimizing the FPGA used resources. To do this, a proper HLS description that divides operations into small functions simplifies path control and improves parallelism. Then, to achieve functions high performance, pipe techniques should be applied to the data loops. For this, using stream-based communication between functions allows consumers to start data processing as soon as producers start data generation, allowing overlapping the execution, which increases parallelism and throughput.

Therefore, using properly HLS as a description language reduces the time needed to implement a time and resource-efficient new ANN topology.

This paper presents an FPGA-based impedance spectrometer with edge AI data analysis using HLS described ANN. The system is based on a 7010 Zynq FPGA Red Pitaya board with custom FPGA firmware. The spectrometer has been developed to be used in the food industry to detect poultry breast anomalies using edge AI computing. The system is fast, low-power, reliable, and adaptable to detect different poultry breast anomalies. Authors have previous experience developing impedance spectrometer systems for poultry breast anomalies detection [3,21]. The presented system architecture is flexible and adaptable, allowing it to perform AI data analysis over impedance measured data on other food or healthcare applications. The developed system is a prototype that allows us to verify the feasibility of the technique in an industrial environment. In order to design a definitive system for an industrial environment, the security of the system should be improved to protect the system from potential threats. Using a zynq-based system allows us to perform these security tasks [22].

The paper is organized as follows: This first section introduces the impedance spectroscopy technique and the drawbacks of adapting and improving the technique for industrial environments. The second section describes the FPGA system architecture. The section is divided into several subsections: Generation and acquisition system, FPGA data analysis, and processing engine ANN. The third section talks about the HLS code optimization applied to improve the system. The fourth section presents the obtained results and their comparison with other edge-AI FPGA-based systems. The last section presents our conclusions.

2. FPGA System Architecture

The system implemented on zynq 7010 red pitaya is presented in Figure 1. The developed hardware is described in the PL (Programmable Logic) section, which is divided into two large blocks: the acquisition, processing, and generation section, with the module called GAP, and the analysis section that is performed by the module ANN PE. Two possible architectures to implement the system are presented. The architecture for solution A uses fewer resources compared to the architecture for the other solutions (B, C, D). This is a tradeoff between resources and processing speed.

It is important to remark that the ANN PE module allows us to carry out this type of configuration to gain processing speed.

GAP and ANN PE are described in more detail in the following sections.

2.1. FPGA Generation, Acquisition and Preprocessing GAP

The application latency is a fundamental factor when AI at the edge is evaluated. In impedance spectroscopy, input signals have a marked sequential character that conditions the rest of the system (including the AI block) and determines much of the overall application latency.

Another remarkable feature of this type of application is that the acquisition trigger is generated by the application itself, either as a result of a user's action or through the response of a pressure sensor that will indicate the right contact with the object of study. In any case, it derives in a start signal that will mark the beginning of the process that generates and acquires the ANN inputs.

The ANN inputs number can be obtained as the number of impedance spectra taken (in our application 4 spectra) times the number of frequencies in each spectrum sweep (in our case, 225 frequencies obtained performing a logarithmic interpolation between 40 Hz and 1 MHz) times the two obtained data for each frequency (impedance modulus and phase). Therefore, our neural network that must classify the studied object has an input layer of 1800 inputs ($4 \times 225 \times 2$). In our acquisition system, each frequency sweep lasts exactly 1130 ms so the two last ANN inputs from the 1800 total needed will be available 4.52 s after the acquisition begins.

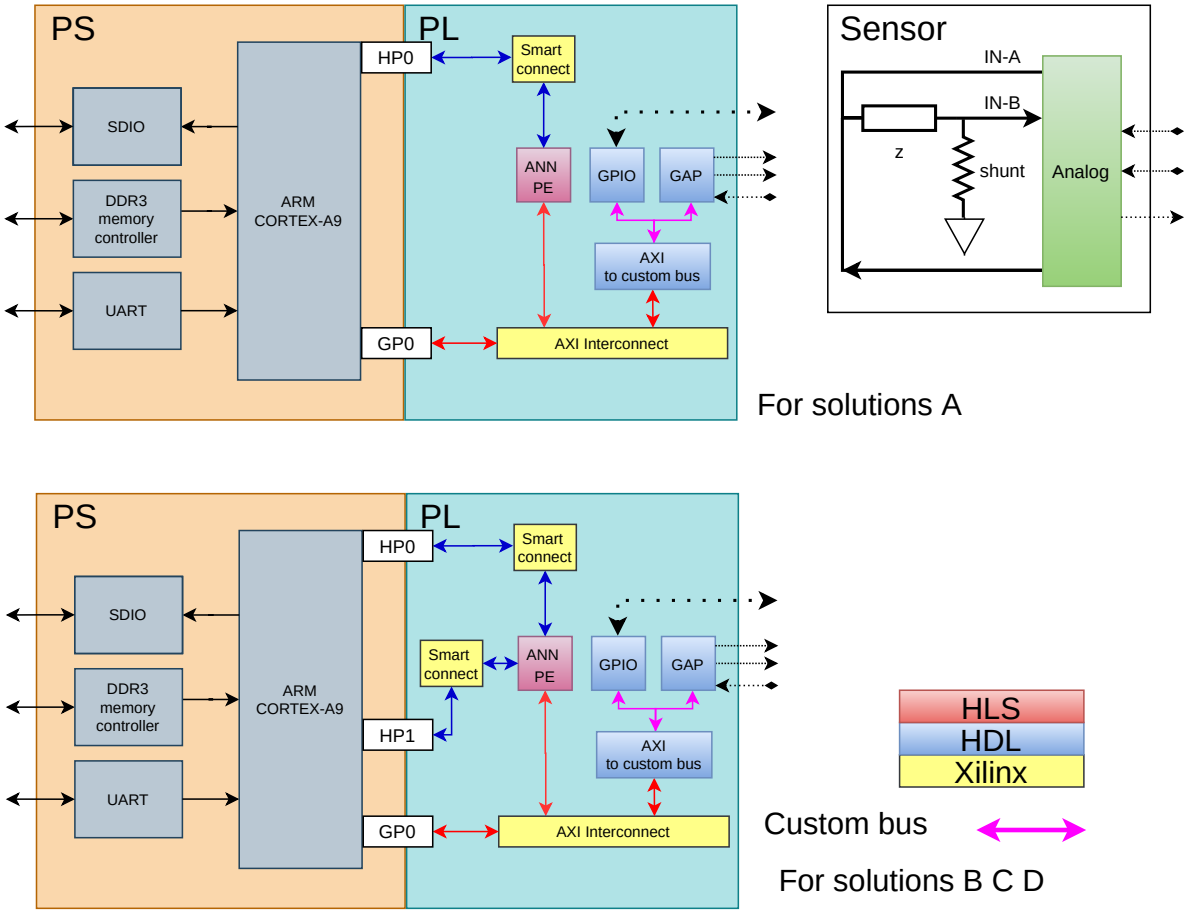

Figure 1. SoC system.

As can be seen in Figure 2, the ANN inputs generation, acquisition, and data preprocessing of block systems have been described at the hardware register transfer logic (RTL) level with HDL. The method used to calculate the impedance modulus and phase in the preprocessing block is the correlation method. The block is controlled by a Finite State Machine to decrease its latency below 5 s. Attempts to perform the generation, acquisition

control, and preprocessing tasks using the FPGA embedded ARM microprocessor increased the signal acquisition times by an order of magnitude.

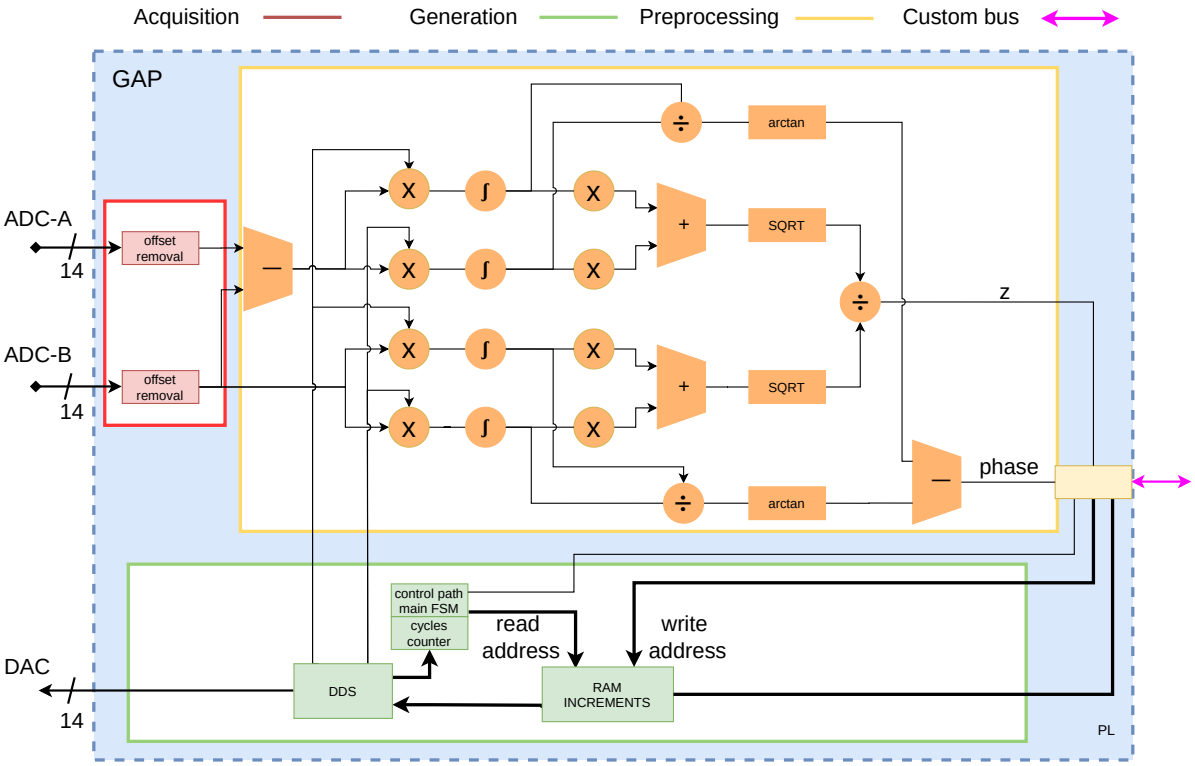

Figure 2. Generation, acquisition and preprocessing implemented GAP details.

2.2. FPGA Data Analysis

In this section, the HLS developed ANN processing block components implemented in the FPGA are described. The presented architecture supports different ANN topologies (with a different number of layers and neurons per layer).

Type of Neural Networks

Different ANN topologies are available depending on the type of function they perform. The best known or used topologies are built by connecting different numbers of layers, with one input layer, N hidden layers, and one output layer. In the present work, Feed-Forward Neural Networks topology is used. Equation (1) describes the computation for each neuron in different layers where $l = 0, 1, \ldots, L$ is the ANN number of layers m_0, m_1, m_2 ,m_L indicates the number of inputs for each layer, the input layer, the first hidden layer, the second hidden layer, and the output layer respectively. Indices i, j refer to different ANN neurons. Forwarding the data from left to the right, the neuron j is located to the right of the neuron i. The index i represents the numbers of entries, where $i = 0, 1, \ldots, m^l$ beginning with zero because it includes bias as input. The index j represents the number of neurons in the layer l, where $j = 1, 2, \ldots, N$. The n index represents the n^{th} ANN input measured samples.

$$v_j^l(n) = \sum_{i=0}^{m^{l-1}} W_{ij}^l . y_i^{l-1}(n), \tag{1}$$

2.3. ANN Processing Engine

Based on the previous section's described topology, our component is named "ANN Processing Engine" (ANN PE). Figure 3 shows the integrated ANN PE with all the components needed to communicate with the microprocessor and DDR memory. The developed

system uses the "Advanced Microcontroller Bus Architecture" (AMBA) protocols to interconnect the different system blocks: AXI-Lite protocol is used to configure each function, AXI-Memory Mapped Full protocol (AXI-MM Full) is used to transfer data between ANN PE and DDR memory, and AXI-STREAM is used to transfer data between internal functions in the ANN PE.

The ANN PE block has been divided into four HLS-developed blocks: DMA WRITE, DMA READ, Matrix-Vector Multiplication (MVM), and Activation Function (AF). DMA READ and DMA WRITE blocks have been developed to perform "Direct Memory Access" (DMA) to the DDR memory. DMA READ is for data reading, and DMA WRITE is for data writing. The reading DMA block uses the AXI-MM Full protocol to read from DDR the raw samples that the GAP section has previously stored in it and to read the ANN weights that are also stored in the DDR. The writing DMA block uses the AXI-MM Full to store ANN PE obtained results in the DDR memory.

The output data from the DMA read function is a stream in fixed-point or floating-point format, depending on the implemented solution.

The read samples and the ANN weights are received by the "Matrix-Vector Multiplication" (MVM) block through the AXI-MM protocol. MVM block should be configured with the number of neurons to be processed before starting the calculation. After MVM configuration, the calculation process begins when there is valid data in the AXI-STREAM input. When a neuron computation is completed, the result is propagated using an AXI-STREAM protocol to be processed by the "Activation Function" (AF). AF block is implemented using a look-up table. Finally, the DMA WRITE block writes the results of the current layer in DDR memory.

This process is repeated for each layer using the previous layer's results as input. The last layer DDR stored results are the ANN analyzed impedance spectra measured data.

The following section presents the different HLS optimizations carried out in the present work. These optimizations allow us to have a fast, efficient, and reconfigurable data analysis system.

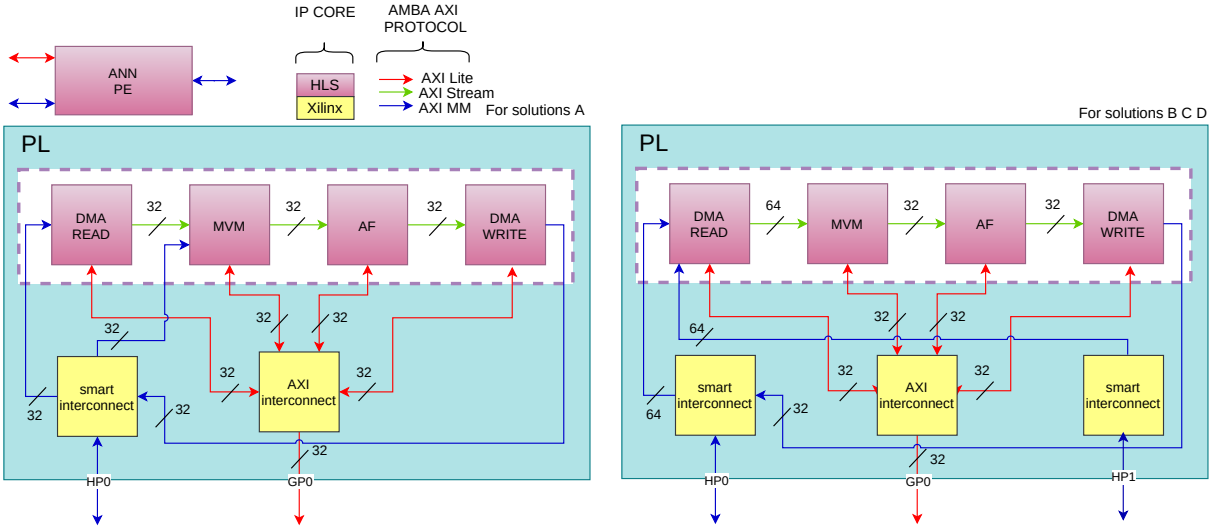

Figure 3. Artificial neural network processing element (ANN PE) in programable logic (PL) section.

3. Optimization and Improvements

Working with FPGA, different levels of hardware optimizations are possible. HLS allows optimizations through its pragmas for performance, latency, area, throughput and interfaces.

The first ANN PE IP Core performed optimization level has been applied at the AXI-STREAM protocol communication between blocks. This optimization allows having a Producer-Consumer architecture. That is, when the producer function delivers an output

result, immediately, the consumer function starts to operate with it. "First Input First Output" (FIFO) is applied between each function. The advantage of applying FIFO queues when AXI-STREAM is used is that each function's control path is reduced in complexity.

Another optimization level applied is to select the right word size for the matrix-vector multiplication operations. 32-bit fixed-point words $<W, I>$ with $I = 8$ and $W = 32$ have been selected. This word size allows us to perform a DMA memory word read in each clock cycle. With this fixed-point word configuration in the MVM operation, it is possible to have a 10^{-5} error compared to performing the same operations using a 32-bit floating point.

The developed read and write DMA functions have been optimized for high throughput and low initialization latency. As is described at the read DMA Algorithm 1 pseudocode, the implementation has been separated into two functions. The main function contains the AXI-MM and AXI-STREAM interfaces. The secondary function is responsible for converting from AXI-MM to AXI-STREAM at the read DMA function and for converting AXI-STREAM to AXI-MM at the write DMA function. If the pragma HLS DATAFLOW (that optimizes throughput and latency) is not applied, the secondary function generates an "Interval Latency" (IL) greater than 2. On the other hand, when the DATAFLOW pragma is applied, it is possible to reduce the IL to 2 clock cycles. In addition, as it is described in previous paragraphs, DMAs can read or write one 32-bit word per clock cycle respectively for solution A or 64-bit for solutions B, C, and D.

Considering the MVM function, different optimizations have been applied. The first decision was to store the weights in FPGA embedded memory. This allows for faster data processing, although it has the drawback that, due to the reduced memory space in FPGA, only ANN structures with small layers can be implemented. The other drawback is the time consumed sending and saving the weights to the Block-RAM (BRAM) memory. Consequently, a second optimization was implemented: remove the BRAM memories and add AXI-MM Full port to read the weights in a similar way to how the read DMA function executes reads from DDR memory. Although, in this optimization, the ANN weights reading begins when the first valid data is present at the input of the MVM AXI-STREAM interface, the cycles latency before the beginning of multiplications and sums calculation is increased, reducing the performance of the MVM function.

The last implemented MVM optimization starts reading the memory weights just before having the first Tvalid signal activation in the AXI-STREAM input data interface. In addition, a single for loop is used together with three pragmas that are applied to this loop. The first implemented pragma is PIPELINE with "Initialization Interval" $II = 1$. This allows the execution of the operation concurrently. Also, the ARRAY PARTITION pragma is applied to divide the samples vector and store it in individual registers without using BRAM. This improves the IP Core throughput. HLS has different pragmas to allocate resources for the executed operations. The low-resource FPGA device used has DSP cores that can perform multiplication and accumulation in one clock cycle. Applying the BIND_OP impl = DSP pragma forces DSP blocks to be used. The MVM implementation has been performed in a function, as described in the Algorithm 2 pseudocode. The sigmoid "Activation Function" (AF) has been implemented with a look-up table which generates one clock cycle latency.

HLS allows defining the IP Core control via hardware or software. By default, Vitis HLS generates several control signals to perform a Hardware IP Core control. To control via software the IP Cores, pragma HLS INTERFACE AXI-LITE port = return is applied to the ports grouped into s_axilite interface. The reason for using a software control is that the processing unit could be reusable to process different neural network layers. This allows us to control via software each of the functions individually.

The sequence of how the data flows from each function is shown in Figure 4, where an example of a 3 inputs layer and 5 neurons is shown. The diagram shows how the AXI-STREAM and AXI-MM signals interact between the functions. It also shows the MVM function latency when a vector multiplied by a column of the weight matrix is computed.

Algorithm 1 READ DMA

MMToStream$(NWords,$
$\quad *dataIn,$
$\quad strmOut)$
#pragma DATAFLOW
$temp$
for $i = 0$ **to** $NWords - 1$ **do**
$\quad temp = *dataIn++$
$\quad strmOut << temp$
end for
dmaRead$(length,$
$\quad *dataInDDR,$
$\quad strmOut)$
#pragma INTERFACE AXI-Stream strmOut
#pragma INTERFACE AXI-Lite dataIn
#pragma INTERFACE AXI-Lite NWords
#pragma INTERFACE AXI-Lite control
MMToStream$(NWords,$
$\quad *dataIn,$
$\quad strmOut)$

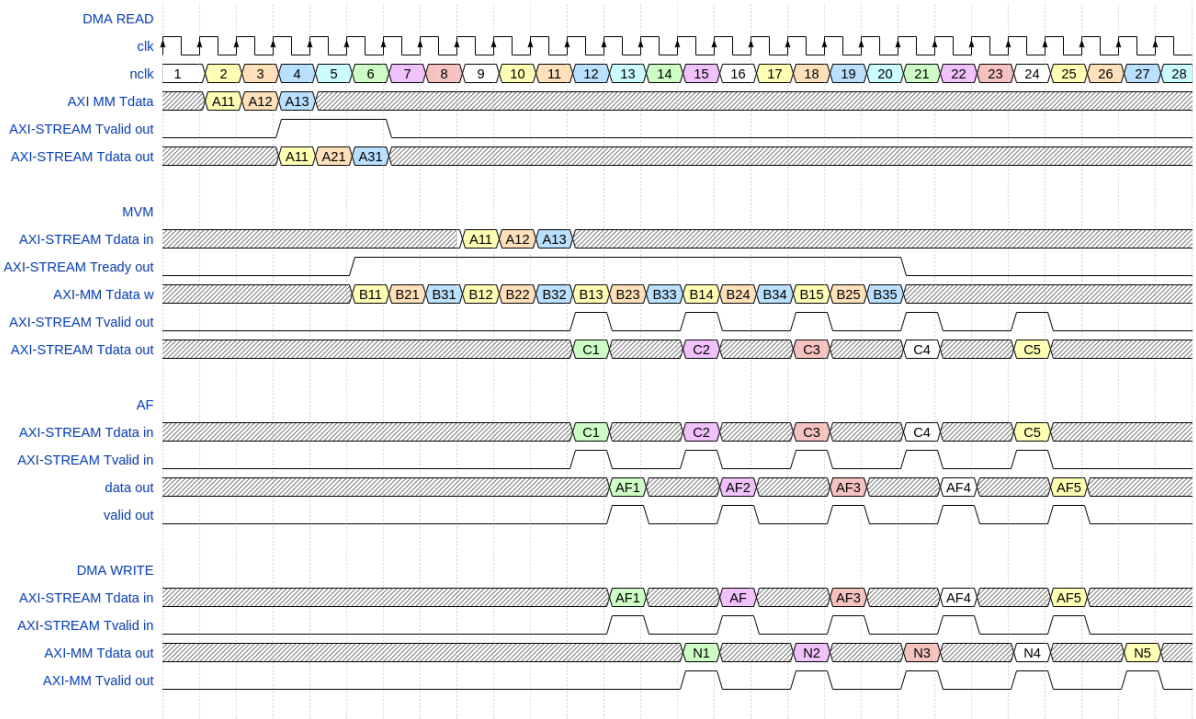

Figure 4. Waveform clock for a ANN PE layer, example of 3 input layer and 5 neurons.

Algorithm 2 Multiplication Vector Matrix

```
1: MVM(strmIn,
2:         strmOut,
3:         *WInDDR,
4:         rowsW,
5:         NWords)
6: #pragma INTERFACE AXI-Stream strmOut
7: #pragma INTERFACE AXI-Stream strmIn
8: #pragma INTERFACE AXI-MM WInDDR
9: #pragma INTERFACE AXI-Lite rowsW
10: #pragma INTERFACE AXI-Lite control
11: inA, outC
12: P, Weight, tmp
13: IndexRW, IndexCP
14: inputBuf[]
15: Weight = *WInDDR++
16: for i = 0 to NWords − 1 do
17:   #pragma PIPELINE II = 1
18:   #pragma ARRAY PARTITION inputBuf //Solutions A, B, C
19:   #pragma ARRAY PARTITION inputBuf factor=2 cyclic//Solution D
20:   inA = StreamIn
21:   if IndexRW < rowsW then
22:     inputBuf[IndexRW] = inA
23:     P = inA.data
24:     IndexRW++
25:   else
26:     inA = inputBuf[IndexCP]
27:     P = inA.data
28:   end if
29:   tmp+ = P * Weight
30:   #pragma BIND_OP impl=dsp
31:   outC.data = tmp
32:   if IndexCP == rowsW − 1 then
33:     IndexCP = 0
34:     tmp = 0
35:     IndexRW++
36:     strmOut << outC
37:   else
38:     IndexCP++
39:   end if
40: end for
```

4. Results

Performance Evaluation

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

Since most of the works are focused on computer vision applications, it is not surprising that frames per second (FPS) are widely used in different publications 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 it is a measurement parameter 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 (mega parameters) calculated per second (Mps), remarking that it is generally understood that each synaptic connection involves one parameter (one Multiply-Accumulate operation which we refer to as MAC).

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 LUTs, and DSPs characteristics) and, fundamentally, by resources available and used in the chosen FPGA. To avoid this drawback, in the present paper, our solution's computational efficiency goodness is measured as network parameters per second per DSP (which corresponds to a MAC operation in almost all manufacturer's FPGA families).

Energy efficiency is also very important when AI at the edge processing is performed in limited battery capacity embedded devices [23]. Energy efficiency is often reported as the number of operations per joule; but, in this work, mW/Mps, which means energy per operation, is used.

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 time-consuming hardware recompilation. And some implementations with dependency on group size (such as the NDrange kernel-based OpenCL implementations), even if they do not require recompilation, have an efficiency very dependent on the exact size of the layers.

The results are summarized in four tables. Tables 1 and 2 show the different iterations performed to reduce FPGA resources and processing time to the ones needed in our application.

Table 1. ANN PE FPGA Resources.

Resources	**Solutions**								
	1A	**2A**	**3A**	**4A**	**5B**	**6B**	**7C**	**8D**	**9C**
LUT	8132	7972	5753	7359	5708	7318	14243	7254	15,264
LUTRAM	364	364	295	297	299	301	889	889	890
FF	11,327	1104	9575	12,479	9669	12,573	35,192	10,309	36,333
BRAM	105	15	15	15	15	15	19	21	21
DSP	6	6	6	5	6	5	10	10	5
BUFG	1	1	1	1	1	1	1	1	1

Different HLS optimizations in the Matrix Vector Multiplication unit for a ZYNQ 7010.

Table 2. Comparison.

Reference	Topology	Clock MHz	Data Types	Thp1 Mps	Thp2 FPS	Energy mW/Mps	CE Mps/DSP
Countinho et al. [24]	784-100-50-10	100	12-bit Fixed	105.062	1250	2.855	0.96
Belabed et al. [25]	784-100-50-10	100	32-bit Float	97.948	1160	3.89	2.129
Our A	784-100-50-10	100	32-bit Fixed	45.84	545.4	3.03	7.64
Our B	784-100-50-10	100	32-bit Fixed	98.49	1173.9	2.43	16.41
Our B	784-100-50-10	100	32-bit Float	19.94	237.7	12.98	3.98
Our C	784-100-50-10	100	32-bit Fixed	213.01	2538.9	2.37	21.30
Our D	784-100-50-10	100	32-bit Fixed	213.14	2540.5	1.24	21.31
Our C	784-100-50-10	100	32-bit Float	48.19	574.4	9.69	9.63
Wang et al. [26]	784-256-256-10	200	32-bit Float	3.346	12.45	69.93	0.104
Our A	784-256-256-10	100	32-bit Fixed	46.98	174.8	2.95	7.83
Our A	784-256-256-10	100	32-bit Float	12.44	46.3	20.08	2.48
Our B	784-256-256-10	100	32-bit Fixed	99.5	370.2	2.41	16.58
Our C	784-256-256-10	100	32-bit Fixed	265.6	988.1	1.9	26.56
Our D	784-256-256-10	100	32-bit Fixed	266.16	990.2	0.99	26.61
Our C	784-256-256-10	100	32-bit Float	50.55	188.07	9.23	10.11

Comparison of the ANN PE with state-of-the-art. CE: Computational Efficiency. Thp1: Throughput in Frames per second. Thp2: Throughput in Mega Parameters per second.

In the first solution, weights are loaded through the AXI-LITE protocol in BRAM memory instantiated in the MVM function. This implementation took up a large number of resources, and it was not feasible to add the other GAP section components. This implementation would be possible for small ANN topology sizes but is not reusable for bigger ANN topologies. Also, this solution has a latency due to the BRAM memory weights load process.

In the second solution, the MVM function reduced the BRAM memory needed because the weights were read from the DDR memory. However, the FPS that are processed is reduced, compared with the previous implementation. Here, at the same time that the DMA begins to read the sample vector and has a latency cycle to start pulling through its axi-stream output, the MVM function reads the weights from memory one by one generating a block with the input data.

The third iteration has been described in detail in the previous sections. It is important to emphasize that this solution can be integrated into the 7010 Zynq FPGA devices (see Table 3), which are the lowest cost Zynq solutions (less than 400 euros). For example, our first iteration, which is the one that has the best results from a speed point of view (550 fps), requires a 7020 Zynq FPGA family device, the cost of which is twice the previous one. Also, a floating-point version, which uses more FPGA resources than the fixed-point version, can be integrated into the 7010 devices, as can been seen in Table 4.

Table 3. FPGA resources.

		PL		PS
		GAP, GPIO, BUS	ANN PE	
Combinational Logic	LUTS	5847 33%	7933 44%	
Sequential Logic	FF	5251 15%	9575 28%	
Memory	BRAM	6 10%	9 15%	
Memory	LUTRAM	48 1%	361 8%	
DSP	Blocks	39 49%	6 8%	
Dynamic Power	W	0.522	0.085	1.292
Static Device Power	W			0.132

Red pitaya 7010 version.

Although in the third solution the speed performance is slightly lower than the first one, it has better energy and computational efficiencies than those shown in the most recent publications (see Table 2).

The scalable accelerator for large-scale DL networks DALU by Wang et al. [26] references 3 topologies (784-64-64-10, 784-128-128-10, and 784-256-256-10) on a 7020 ZYNQ FPGA. Their work presents a poor energy efficiency (69.93 mW/Mps) but, comparatively, it can be considered a low power (234 mW) solution.

The work detailed in [24] proposes an SSAE optimized at the RTL level to achieve the best system performance in terms of throughput and energy efficiency. However, to achieve this performance, they use 12-bit fixed data types that provide lower accuracy (93.3%). The low computational efficiency achieved is the problem with this solution.

Undoubtedly, Belabed et al. [25] is the best solution for the referenced studies in the present paper. Their solution achieves a throughput that stands out above all. A very high number of DSPs have been used to obtain these results. That makes its computational efficiency inferior to our third implemented solution. Their solution cannot be implemented on a 7010 Zynq device, as can be seen in Table 4.

Table 4. FPGA resources for MVM

Reference	Topology	Clock[MHz]	LUT	LUTRAM	BRAM	DSP	FF
[25] 7020	784-100-50-10	100	55%	10%	20%	21%	32%
[25] 7010	784-100-50-10	100	166%	20%	47%	58%	96%
Our 7020 Fixed A	784-100-50-10	100	11%	2%	7%	3%	9%
Our 7020 Float A	784-100-50-10	100	13%	1.71%	11%	2.2%	11.73%
Our 7020 Fixed B	784-100-50-10	100	10.73%	1.72%	11.07%	2.73%	9.09%
Our 7020 Float B	784-100-50-10	100	13.78%	1.73%	11.07%	2.27%	11.82%
Our 7020 Fixed C	784-100-50-10	100	26.77%	5.11%	13.57%	4.55%	33.08%
Our 7020 Float C	784-100-50-10	100	28.69%	5.11%	11.07%	2.27%	34.15%
Our 7020 Fixed D	784-100-50-10	100	13.64%	5.11%	15%	4.55	9.69%
Our 7010 Fixed A	784-100-50-10	100	32%	4%	15%	7%	27%
Our 7010 Float A	784-100-50-10	100	42%	5%	26%	6%	35%

In comparison with Belabed et al. [25].

5. Conclusions and Future Directions

AI at the edge applications must be low power, portable and low cost with high performance and low latency. This paper presents a functional impedance spectroscopy system for the industry. The works demonstrate it is possible to use a low-cost FPGA device to implement a system with the data acquisition, generation, and complex ANN-based data analysis blocks. The ANN PE component has been developed in C++ and can be quickly implemented and optimized using Vivado HLS. According to the results shown in the tables, high computational efficiency and low consumption have been achieved on the system.

In future directions, new AI algorithms will be adapted. To speed up system changes, the acquisition, and generation system will also be migrated to HLS. To modify the system's accuracy, the possibility of generating the components with HLS with other data types such as fp16 will be incorporated.

Also, it is possible to migrate the project to a higher-resource FPGA to work with different ANN implementations in parallel. This would increment the data analysis capabilities having a system cost increment.

Author Contributions: Conceptualization, J.F.; methodology, J.F., R.G.-G. and J.M.M.; software, R.G.-G. and J.F.; validation, R.G.-G. and J.M.M.; investigation, J.F.; resources, R.C.-P. and Á.T.-R.; writing original draft preparation, J.F.; writing review and editing, J.F., R.G.-G. and J.M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is contained within the article.

Acknowledgments: This work was supported in part by the Spanish MCIU under Project PID2020-116816RB-I00 (MCIU/FEDER) and in part by GVA under Project INNEST/2020/248.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ochandio Fernández, A.; Olguín Pinatti, C.A.; Masot Peris, R.; Laguarda-Miró, N. Freeze-damage detection in lemons using electrochemical impedance spectroscopy. *Sensors* **2019**, *19*, 4051. [CrossRef] [PubMed]
2. Islam, M.; Wahid, K.; Dinh, A. Assessment of ripening degree of avocado by electrical impedance spectroscopy and support vector machine. *J. Food Qual.* **2018**, *2018*. [CrossRef]
3. Traffano-Schiffo, M.V.; Castro-Giraldez, M.; Herrero, V.; Colom, R.J.; Fito, P.J. Development of a non-destructive detection system of Deep Pectoral Myopathy in poultry by dielectric spectroscopy. *J. Food Eng.* **2018**, *237*, 137–145. [CrossRef]
4. Zhao, X.; Zhuang, H.; Yoon, S.C.; Dong, Y.; Wang, W.; Zhao, W. Electrical impedance spectroscopy for quality assessment of meat and fish: A review on basic principles, measurement methods, and recent advances. *J. Food Qual.* **2017**, *2017*, 6370739 [CrossRef]
5. Durante, G.; Becari, W.; Lima, F.A.; Peres, H.E. Electrical impedance sensor for real-time detection of bovine milk adulteration. *IEEE Sens. J.* **2015**, *16*, 861–865. [CrossRef]
6. Zhu, H.; Liu, F.; Ye, Y.; Chen, L.; Liu, J.; Gui, A.; Zhang, J.; Dong, C. Application of machine learning algorithms in quality assurance of fermentation process of black tea–based on electrical properties. *J. Food Eng.* **2019**, *263*, 165–172. [CrossRef]
7. Stupin, D.D.; Kuzina, E.A.; Abelit, A.A.; Emelyanov, A.K.; Nikolaev, D.M.; Ryazantsev, M.N.; Koniakhin, S.V.; Dubina, M.V. Bioimpedance spectroscopy: Basics and applications. *ACS Biomater. Sci. Eng.* **2021**, *7*, 1962–1986. [CrossRef]
8. Naranjo-Hernández, D.; Reina-Tosina, J.; Min, M. Fundamentals, recent advances, and future challenges in bioimpedance devices for healthcare applications. *J. Sens.* **2019**, *2019*, 9210258. [CrossRef]
9. Carminati, M.; Scandurra, G. Impact and trends in embedding field programmable gate arrays and microcontrollers in scientific instrumentation. *Rev. Sci. Instrum.* **2021**, *92*, 091501. [CrossRef] [PubMed]
10. Ruiz-Vargas, A.; Arkwright, J.; Ivorra, A. A portable bioimpedance measurement system based on Red Pitaya for monitoring and detecting abnormalities in the gastrointestinal tract. In Proceedings of the 2016 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES), Kuala Lumpur, Malaysia, 4–8 December 2016; pp. 150–154.
11. Yang, Y.; Zhang, F.; Tao, K.; Wang, L.; Wen, H.; Teng, Z. Multi-frequency simultaneous measurement of bioimpedance spectroscopy based on a low crest factor multisine excitation. *Physiol. Meas.* **2015**, *36*, 489. [CrossRef] [PubMed]
12. Jiang, Z.; Yao, J.; Wang, L.; Wu, H.; Huang, J.; Zhao, T.; Takei, M. Development of a portable electrochemical impedance spectroscopy system for bio-detection. *IEEE Sens. J.* **2019**, *19*, 5979–5987. [CrossRef]
13. Luna, J.M.M.; Luna, A.M.; Fernández, R.E.H. Characterization and Differentiation between Olive Varieties through Electrical Impedance Spectroscopy, Neural Networks and IoT. *Sensors* **2020**, *20*, 5932. [CrossRef] [PubMed]

14. Vela, L.M.; Kwon, H.; Rutkove, S.B.; Sanchez, B. Standalone IoT bioimpedance device supporting real-time online data access. *IEEE Internet Things J.* **2019**, *6*, 9545–9554. [CrossRef]
15. RedPitaya Product Comparison Table. Available online: https://redpitaya.readthedocs.io/en/latest/developerGuide/hardware/compares/vs.html (accessed on 18 February 2022).
16. Rivola, M.; Ibba, P.; Lugli, P.; Petti, L. Bioimpedance data statistical modelling for food quality classification and prediction. In Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Korea, 22–28 May 2021; pp. 1–5.
17. Chowdhury, D.; Chattopadhyay, M. Study and Classification of Cell Bio-Impedance Signature for Identification of Malignancy Using Artificial Neural Network. *IEEE Trans. Instrum. Meas.* **2021**, *70*, 1–8. [CrossRef]
18. Paterno, A.; Negri, L.H.; Bertemes-Filho, P. Efficient computational techniques in bioimpedance spectroscopy. *Applied Biological Engineering-Principles and Practice*; InTech-Open Access Publisher: Rijeka, Croatia, 2012, pp. 1–26.
19. Wang, X.; Han, Y.; Leung, V.; Niyato, D.; Yan, X.; Chen, X. Edge Computing for Artificial Intelligence. In *Edge AI*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 97–115.
20. VITIS Hls. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug1399-vitis-hls.pdf (accessed on 18 February 2022).
21. Traffano-Schiffo, M.V.; Castro-Giraldez, M.; Colom, R.J.; Fito, P.J. Development of a spectrophotometric system to detect white striping physiopathy in whole chicken carcasses. *Sensors* **2017**, *17*, 1024. [CrossRef] [PubMed]
22. Trimberger, S.M.; Moore, J.J. FPGA security: Motivations, features, and applications. *Proc. IEEE* **2014**, *102*, 1248–1265. [CrossRef]
23. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. How to Evaluate Deep Neural Network Processors: TOPS/W (Alone) Considered Harmful. *IEEE Solid-State Circuits Mag.* **2020**, *12*, 28–41. [CrossRef]
24. Coutinho, M.G.; Torquato, M.F.; Fernandes, M.A. Deep neural network hardware implementation based on stacked sparse autoencoder. *IEEE Access* **2019**, *7*, 40674–40694. [CrossRef]
25. Belabed, T.; Coutinho, M.G.F.; Fernandes, M.A.; Sakuyama, C.V.; Souani, C. User Driven FPGA-Based Design Automated Framework of Deep Neural Networks for Low-Power Low-Cost Edge Computing. *IEEE Access* **2021**, *9*, 89162–89180. [CrossRef]
26. Wang, C.; Gong, L.; Yu, Q.; Li, X.; Xie, Y.; Zhou, X. DLAU: A scalable deep learning accelerator unit on FPGA. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2016**, *36*, 513–517. [CrossRef]