applied sciences

MDPI

Article

Sigmoid Activation Implementation for Neural Networks Hardware Accelerators Based on Reconfigurable Computing Environments for Low-Power Intelligent Systems

Vladislav Shatravin *, Dmitriy Shashev and Stanislav Shidlovskiy

Faculty of Innovative Technologies, Tomsk State University, 634050 Tomsk, Russia; dshashev@mail.tsu.ru (D.S.); ssv@mail.tsu.ru (S.S.)

* Correspondence: shatravin@stud.tsu.ru

Abstract: The remarkable results of applying machine learning algorithms to complex tasks are well known. They open wide opportunities in natural language processing, image recognition, and predictive analysis. However, their use in low-power intelligent systems is restricted because of high computational complexity and memory requirements. This group includes a wide variety of devices, from smartphones and Internet of Things (IoT)smart sensors to unmanned aerial vehicles (UAVs), self-driving cars, and nodes of Edge Computing systems. All of these devices have severe limitations to their weight and power consumption. To apply neural networks in these systems efficiently, specialized hardware accelerators are used. However, hardware implementation of some neural network operations is a challenging task. Sigmoid activation is popular in the classification problem and is a notable example of such a complex operation because it uses division and exponentiation. The paper proposes efficient implementations of this activation for dynamically reconfigurable accelerators. Reconfigurable computing environments (RCE) allow achieving reconfigurability of accelerators. The paper shows the advantages of applying such accelerators in low-power systems, proposes the centralized and distributed hardware implementations of the sigmoid, presents comparisons with the results of other studies, and describes application of the proposed approaches to other activation functions. Timing simulations of the developed Verilog modules show low delay (14–18.5 ns) with acceptable accuracy (average absolute error is 4×10^{-3}).

Keywords: deep neural networks; hardware accelerators; low-power systems; homogeneous structures; reconfigurable environments; parallel processing

Citation: Shatravin, V.; Shashev, D.; Shidlovskiy, S. Sigmoid Activation Implementation for Neural Networks Hardware Accelerators Based on Reconfigurable Computing Environments for Low-Power Intelligent Systems. *Appl. Sci.* **2022**, *12*, 5216. https://doi.org/10.3390/app12105216

Academic Editors: Deliang Fan and Zhezhi He

Received: 6 April 2022
Accepted: 19 May 2022
Published: 21 May 2022

1. Introduction

At present, artificial neural networks (NN) are actively used in various intelligent systems for tasks that cannot be effectively solved by any classical approach: natural language processing, image recognition, complex classification, predictive analysis, and many others. It is possible because of the ability of NN to extract domain-specific information from a large set of input data, which can be used later to process new input data. The main disadvantage of NN is their high computational complexity. A complex task requires a large NN model with a huge number of parameters, which means that many operations must be performed to calculate the result. The problem is especially acute for deep convolutional neural networks (CNN) because their models can include hundreds of billions of parameters [1,2].

In cloud and desktop systems, the problem of computational complexity can be partially solved by scaling. However, low-power and autonomous devices impose strict requirements on their weight and battery life [3]. Some examples of such systems include unmanned aerial vehicles (UAVs), self-driving cars, satellites, smart Internet of Things (IoT) sensors, Edge Computing nodes, mobile robots, smartphones, gadgets, and many others.

Such devices require specialized hardware accelerators and fine-tuned algorithms to use machine learning algorithms effectively.

There are many papers about various optimizations of hardware and machine learning algorithms for low-power applications. Hardware optimizations mainly mean replacing powerful but energy-intensive graphics processing units (GPU), which are popular in the cloud and desktop systems, with more efficient devices, such as field-programmable gate array (FPGA) and application-specific integrated circuits (ASIC) [4–11]. At the same time, to use hardware resources efficiently, applied algorithms are adapted to the features of the chosen platform. In addition, the quality of NN models is often acceptably downgraded to reduce computational complexity. For example, authors of [12,13] propose to reduce the bit-lengths of numbers in a model to 8 and 4 bits, respectively, and in [14,15], binary NNs are presented. In [16], the complexity of a NN is reduced using sparsing, which is the elimination of some weights to reduce the number of parameters.

Such comprehensive solutions show good results in the implementation of concrete NN architectures. However, in practice, complex intelligent systems can face various tasks that require different NN architectures. Sometimes, it is impossible to predict which architectures will be required in the future. The problem is especially acute if the autonomous system is remote and difficult to access. The simplest solution is to implement many hardware subsystems for the most probable architectures, although this approach affects weight, power consumption, reliability, and complexity of the device.

Another way to solve the problem is to use dynamically reconfigurable hardware accelerators that can change the current model by an external signal at runtime. Application of such accelerators offers great opportunities for performance, energy efficiency, and reliability. Therefore, numerous research has been conducted in this area. We explore dynamically reconfigurable accelerators based on the concept of reconfigurable computing environments (RCEs).

One of the significant parts in developing RCE-based hardware accelerators is the implementation of neuron activation functions. There are many different activations now, and one of the most popular among them is the sigmoid activation (logistic function), which is widely used in an output layer of NNs for classification tasks. However, the original form of the activation is difficult to compute in hardware, so simplified implementations are usually used. This paper proposes two implementations of sigmoid activation for dynamically reconfigurable hardware accelerators.

2. Artificial Neural Networks

An artificial neural network is a mathematical model of a computing system, inspired by the structure and basic principles of biological neural networks. By analogy with natural NNs, artificial networks consist of many simple nodes (neurons) and connections between them (synapses). In artificial NNs, a neuron is a simple computing unit, which sums weighted inputs, adds its own bias value, applies an activation function to the sum, and sends the result to neurons of the next layer (Figure 1). Neurons are distributed among several layers. In the classical dense architecture, inputs of each neuron are connected to the output of each neuron of a previous layer. Therefore, each NN has one input layer, one output layer, and one or more hidden layers. If the network has many hidden layers, this network is called “deep” (DNN). This architecture is called a feedforward neural network (Figure 2) [17].

All values of synapses (weights) and neurons (biases) are parameters of the NN model. To solve a specific task by NN, it is necessary to determine all parameters of the model. This process of determining the parameters is called “learning”. There are many learning techniques. The classical supervised learning process requires input dataset with known expected results (marked dataset). During training, the parameters are corrected to reduce the difference between the actual and expected results at the network output for the training samples. Using a large and representative training set, a sufficiently complex

model and a sufficient number of iterations allows obtaining a model with high accuracy on new samples.

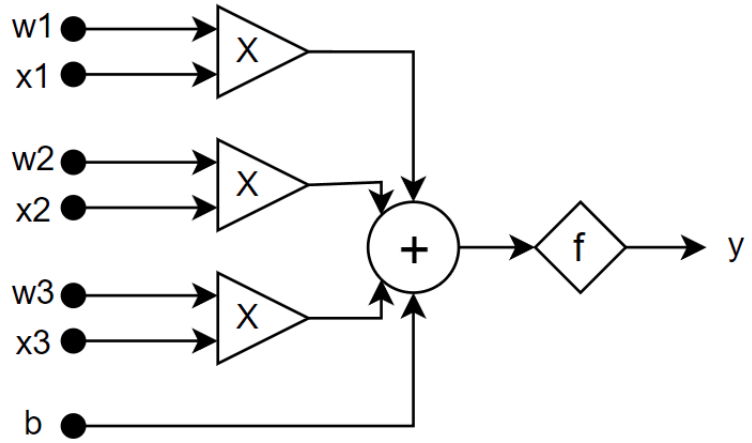

Figure 1. Artificial neuron with three inputs.

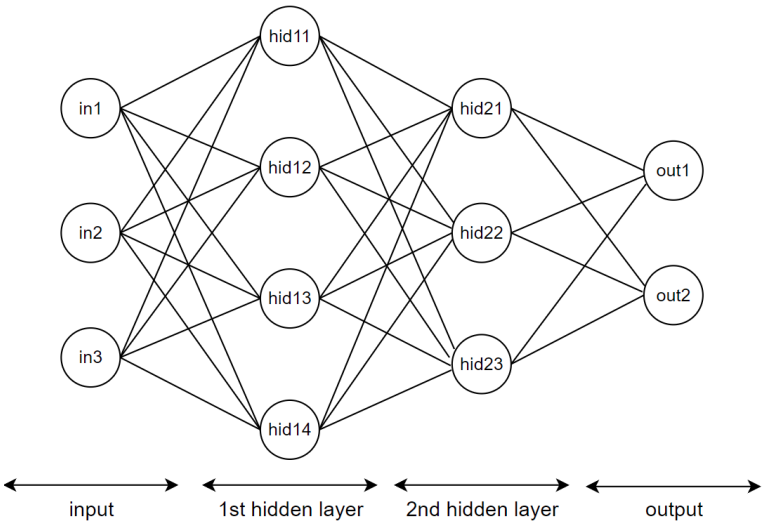

Figure 2. Feedforward neural network with two hidden layers.

An activation function of a neuron is a non-linear function that transforms a weighted sum of the neuron's inputs to some distribution. In practice, many different activations are used, and they depend on the task and the chosen architecture [18–21]. In classification tasks, rectified linear units (ReLU) in hidden layers and a sigmoid activation in the output layer are very popular. This allows getting a result in the form of the probability that the input object belongs to a particular class. Because the original sigmoid activation uses division and exponentiation, it is often replaced with simplified analogs. In this paper, the natural sigmoid is replaced by its piecewise linear approximation.

3. Neural Networks Hardware Accelerators

The applying of neural network models in real systems leads to the need to use specialized computing devices. The huge number of simple operations inherent to NN models is a challenge, even for a state-of-the-art CPU. At the same time, the dataflow in NN allows high parallelization, so computing systems that support simultaneous execution of large amounts of calculations are worth using.

A simple and powerful solution to the problem is the use of GPUs [4,5]. Due to their multicore architecture, GPUs can significantly reduce time costs of training and using DNNs in many applications. In addition, the simplicity of the development on the GPU

makes it easy to implement and maintain solutions. Today, GPU-based accelerators are widely used in desktop, server, and cloud systems, as well as in some large mobile systems, such as self-driving cars. The main disadvantage of these accelerators is their high power consumption, which limits their use in many autonomous and mobile systems.

Further research to improve the characteristics of accelerators has led to the development of highly specialized devices based on FPGA and ASIC. Due to their flexible architecture, low-level implementation, and fine-tuning optimization, these solutions outperform GPU-based counterparts while consuming less power [7–11]. The disadvantages of such solutions are the complexity and cost of developing, embedding, and maintaining. In addition, the accelerators often support only a certain model or architecture of NN, which limits their reuse for other tasks.

In recent years, hybrid computing systems have become popular. These systems include a CPU and an auxiliary neuroprocessor simultaneously. The CPU performs common tasks and delegates machine learning tasks to the neuroprocessor [22]. Because of the division of responsibilities, hybrid systems effectively solve a wide class of applied tasks. But the neuroprocessors are designed to fit prevalent needs, and the problem of the narrow focus of accelerators has not been completely eliminated yet.

One possible solution to this problem is applying dynamically reconfigurable hardware accelerators.

4. Dynamically Reconfigurable Hardware Accelerators

A key feature of dynamically reconfigurable accelerators is their ability to change the current NN model at runtime. A single computing system is able to implement different models and even architectures of NN at different points in time. This offers wide opportunities for target systems:

- Support for different architectures of NN without hardware changes;
- Usage of different NN models for different operating modes. For example, the device can use an energy saving model with low accuracy in standby mode and switch to the main model with high accuracy and power consumption in a key working mode;
- Remote modification or complete replacement of NN models of the accelerator, which can be very useful in cases where there is not enough information to set models in advance;
- Ability to be recovered by reconfiguration and redistribution of computations over the intact area of the accelerator.

There are many different approaches to designing reconfigurable accelerators. The paper [23] proposes a hierarchical multi-grained architecture based on bisection neural networks with several levels of configuration. In [24], reconfigurability is achieved by using some amount of floating neurons that can move between layers of NN to create the required model. The accelerators presented in [25–27] do not use reconfigurability in its classical meaning, but modularity and homogeneous structures with an excessive number of elements make it possible to solve a wide range of applied tasks.

We propose the development of reconfigurable hardware accelerators based on the concept of reconfigurable computing environments.

5. Reconfigurable Computing Environments

A reconfigurable computing environment (RCE) is a mathematical model of a wide range of computing systems based on the idea of a well-organized complex behavior of many similar small processing elements (PE) [28,29]. In some references, RCE is called a homogeneous structure [29]. The PEs are arranged in a regular grid and are connected to neighboring PEs. Each PE can be individually configured by an external signal or internal rule to perform some operation from a predefined set. The collaborative operating of many independent processors allows implementing complex parallel algorithms. Thus, the computational capabilities of RCE are limited only by its size and operations set.

In general, an RCE can have an arbitrary number of dimensions (from one to three), and its PEs can be of any shape. In this paper, we discuss a two dimensional RCE with square PEs. Therefore, each non-boundary element is connected with four neighbors (Figure 3).

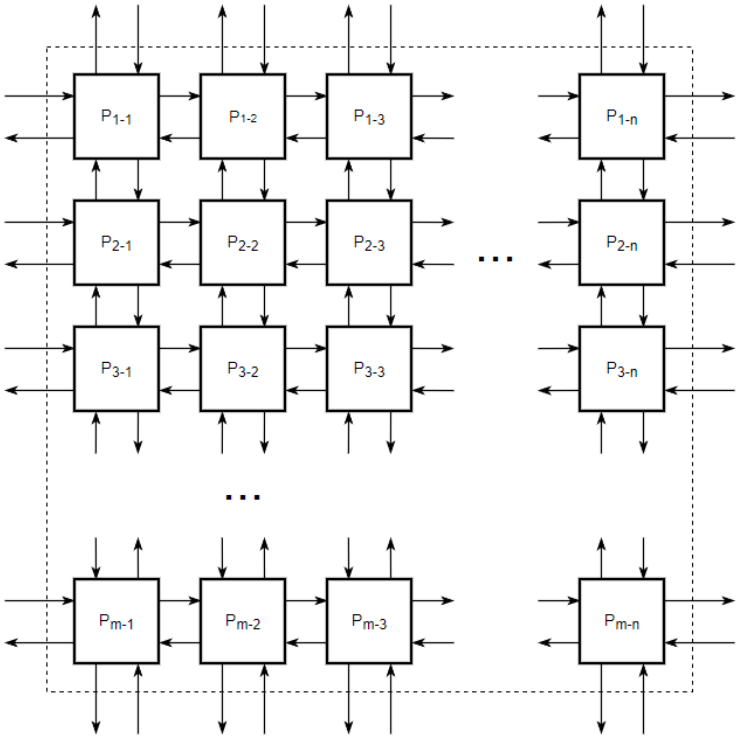

Figure 3. Reconfigurable computing environment.

6. Hardware Accelerators on Homogeneous Structures

We are not pioneers of applying homogeneous structures in the hardware accelerators. Numerous research in this area is based on the use of systolic arrays [25–27]. The systolic array, or systolic processor, is a subclass of homogeneous computing structures, for which additional restrictions are set [30]:

- All PEs perform the same operation;
- After completion of the operation (usually on each clock cycle), the PE transmits the result to one or more neighboring PEs in accordance with the specified direction of signal propagation;
- The signal passes through the entire structure in one direction.

Systolic processors are known for their simplicity and high performance. The homogeneity and modularity of their structure facilitates scaling and manufacturing. But there are some disadvantages:

- Narrow focus on specific tasks, since all PEs perform the same simple operation;
- Low fault tolerance due to a large amount of processors and interconnections, as well as a fixed direction of signal propagation;
- Some algorithms are difficult to adapt to the features of the processing flow in systolic arrays.

A classic task for systolic arrays is matrix multiplication (Figure 4) [31]. Since most of the computations in NN are matrices multiplication, modern hardware accelerators efficiently use computing units based on systolic arrays. All other calculations, such as activation or subsampling, are performed by separate specialized units.

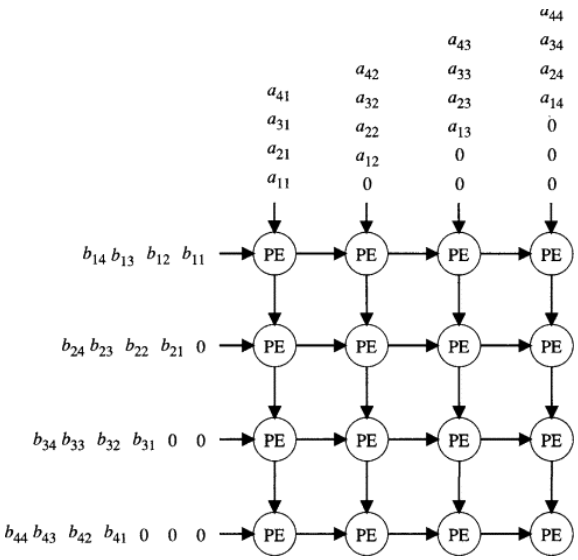

Figure 4. Matrix multiplication in the systolic array [31]. 1999 Academic Press, with permission from Elsevier.

One of the popular solutions for machine learning tasks is the tensor processing unit (TPU), based on the systolic array (Figure 5) [27]. This is a hardware accelerator on ASIC developed by Google. The TPU systolic array has a size of 256 × 256 PEs. It performs matrix multiplication and shows very good results (Figure 6). The accumulation of sums, subsampling, and activation are performed by dedicated blocks outside the array.

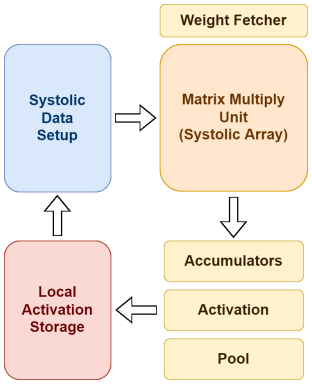

Figure 5. Google Tensor processing unit architecture.

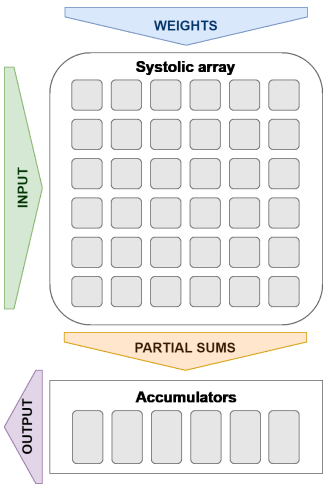

Figure 6. Systolic array in the Google TPU.

Another example of the efficiency of homogeneous structures in neural networks hardware accelerators is presented in [25,26]. The proposed Eyeriss accelerator uses a homogeneous computing environment consisting of 12 × 14 relatively large PEs (Figure 7). Each PE receives one row of input data and a vector of weights and performs convolution over several clock cycles using a sliding window. Accordingly, the accelerator's dataflow is called "row-stationary".

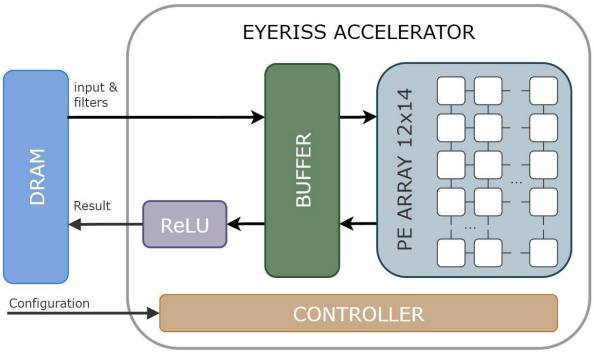

Figure 7. Eyeriss accelerator architecture.

One of the important tasks in the development of accelerators is to reduce the data exchange with other subsystems of the device (memory, additional computing units, etc.) due to the high impact on performance and power consumption. This can be achieved through data reuse. There are four types of dataflows with data reuse: weight-stationary, input-stationary, output-stationary, and row-stationary [32]. In a weight-stationary dataflow, each PE stores the weight values in its internal memory and applies them to each input vector. Google TPU uses this dataflow [27]. The PE of input-stationary dataflow stores the input vector and receives different weight vectors from the outside. The accelerators with output-stationary dataflow accumulate partial sums in PE, while the input and weights move around the environment. The above-described Eyeriss accelerator uses a row-stationary dataflow since each PE stores one row of input data and one vector of weights to perform multicycle convolution [25].

The accelerator proposed in this paper uses a hybrid dataflow. It operates in weight-stationary dataflow when input data are too large to be handled at once; otherwise, it does not reuse data at all. The refusal to reuse data is compensated by an inherent ability of our architecture to avoid any exchange of intermediate results with other subsystems by performing complete processing within the computing structure. This allows eliminating time cost of access to memory and auxiliary blocks. However, temporary weight reuse is supported as part of pipeline processing.

In contrast to most counterparts [23–26], the proposed accelerator architecture is based on the principle of atomic implementation of processing elements. This means that PE has a simple structure and performs very simple operations. This decision makes it possible to achieve high flexibility of the computing environment and allows fine-tuning of each parameter. By analyzing classical feedforward networks, a set of PE operations was selected: "signal source", "signal transfer", "MAC", "ReLU", "sigmoid" [33]. With these operations, a neuron can be implemented as a chain of PEs (Figure 8) [34]. The length of the chain determines the number of inputs, so a neuron of any configuration can be built. The PE of the proposed model operates with 16-bit fixed-point numbers. Table 1 presents the comparison of the proposed model and the mentioned counterparts. The decision to implement activation functions in RCE instead of using a dedicated unit is based on the following considerations:

- RCE is more flexible. This allows introducing new activations or modifying existing ones after the system is deployed;
- RCE is more reliable. The dedicated unit can become a point of failure, while the RCE can be reconfigured to partially restore functionality;

- Data exchange between RCE and dedicated unit may be longer. It is more efficient to keep all intermediate results inside the RCE.

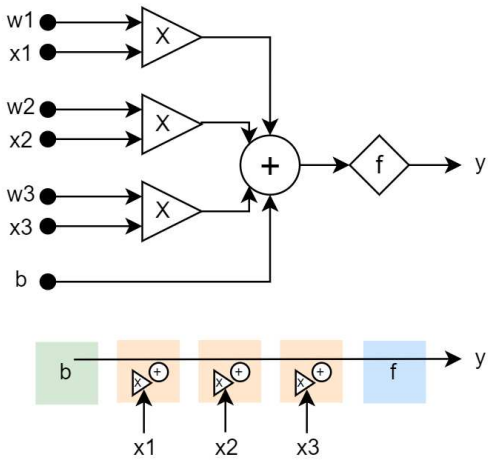

Figure 8. Neuron on the proposed RCE architecture. Reprinted from [34], with permission 2021 IEEE.

Table 1. Comparison of accelerators based on homogeneous structures.

Parameter	TPU Systolic Array	Eyeriss	Proposed Model
Processing element (PE)	8-bit MAC	16-bit MAC with memory block and partial sums accumulator	16-bit unit, supporting 7 simple operations
PE memory	None	448KB SRAM + 72KB Registry	21 bits
PE size	Very small	Relatively large	Average
Number of PEs	A lot of $256 \times 256 = 65{,}536$	A few $12 \times 14 = 168$	A lot of depends on the task
Reconfigurable PE	No	No	Yes
A role of homogeneous structure	Matrix multiplication	Matrix multiplication	Complete processing
Dataflow	Weight-stationary	Row-stationary	Hybrid (weight-stationary or no reuse)
Intermediate results storage	Buffer	Buffer	PEs (inside the environment)
Post-processing units (activation, subsampling)	Dedicated blocks	Dedicated blocks	PEs (inside the environment)

To implement deep networks and perform pipelining, the proposed model supports multi-cycle processing using internal rotation of the signal [35]. The RCE is divided into several segments; each segment is configured to implement one layer of the required NN model. When an input signal arrives at a segment, the segment calculates an output of the layer and passes it on to the next segment. After the signal leaves the segment, the segment is reconfigured to the layer, following the layer implemented in the previous segment. It means that, in case of four segments, the second segment at the i-th step will implement the $(4 \times (i-1) + 2)$-th layer of the neural network. Therefore, the signal is spinning inside the structure until the final result is obtained (Figure 9).

The proposed architecture allows implementing a DNN of arbitrary depth. The only limitation is the size of the configuration memory. Parallel flow processing of inputs in many PEs significantly reduces computation time. In addition, it is possible to use pipelining to improve further performance. The structure with n segments can process $n-1$ signals simultaneously, while the n-th segment can be reconfigured by a sliding window. The pipelining solves three problems:

- Improves resource utilization;
- Increases performance by processing multiple signals simultaneously and eliminating reconfiguration time cost;
- Introduces temporary reuse of weights (short-term stationary dataflow).

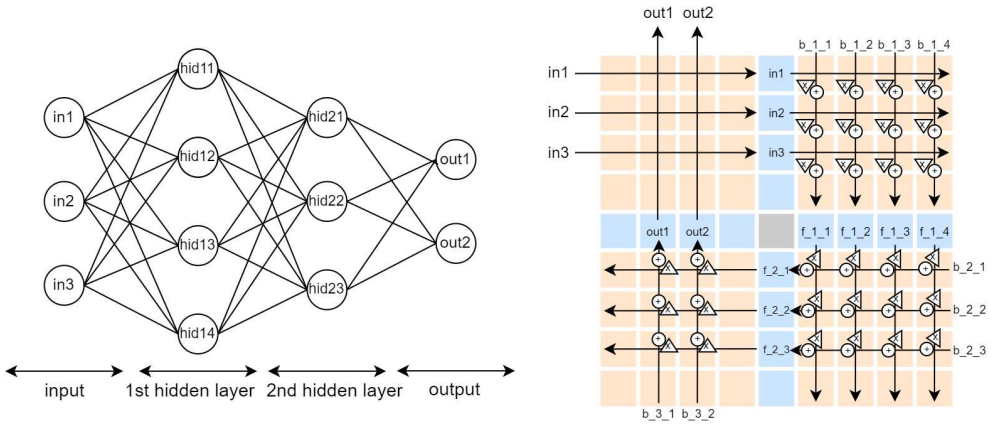

Figure 9. Multi-cycle processing of NN in the segmented RCE.

To implement a multi-cycle architecture, a "1-cycle latch" operation must be added to the operations set. This is necessary to store intermediate results between segments. A 90° signal rotation operation is included in the MAC.

The key disadvantage of the proposed segmented architecture is the limitation of the maximum supported layer size. After segmentation, only some parts of the RCE can be used to implement a layer. This is a significant drawback for the first hidden layers of CNNs, which consist of a large number of neurons. To handle this case, an integral operating mode can be used. In this mode, the entire RCE implements only one layer of a NN and the signal is transmitted in one direction. Disadvantages of the mode are the lack of pipelining and need for external control unit to route the input signal and store intermediate results (Figure 10).

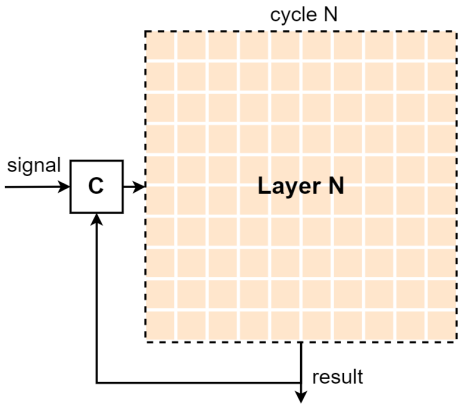

Figure 10. RCE in intergal mode.

Timing simulations of the proposed models showed acceptable results [34,35]. However, one of the operations, sigmoid activation, is computationally difficult in its natural form (Figure 11). As a result, the processing elements and the entire environment become large, complex, and slow. The piecewise linear approximation proposed in [36] partially solves the problem, but the operation remains complicated compared to others. This paper presents centralized and distributed modifications of the sigmoid activation for implementation in the described RCE architecture.

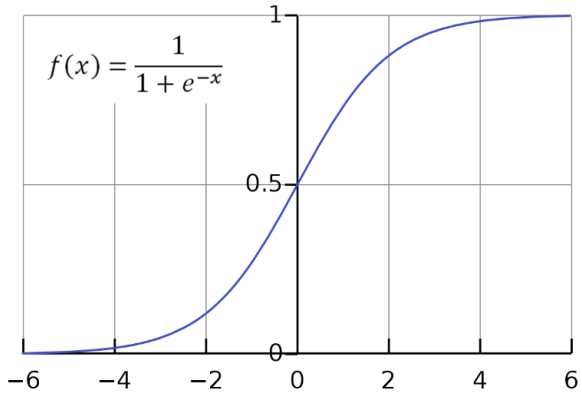

Figure 11. Sigmoid activation.

7. Implementations of Sigmoid Activation

7.1. Centralized Implementation

In the centralized implementation of the sigmoid activation, all calculations are performed in one PE, which allows to minimize and optimize it efficiently. These optimizations are possible because of the features of the fixed-point number format and the applied approximation [36]. The piecewise linear approximation is described as:

$$f(x) = ax + b \tag{1}$$

where a and b are the constants related to the subrange in which x is located.

One of the most difficult parts of the piecewise linear approximation is finding the range where the input value lies. Approximation nodes are the integers ranging from -5 to 5 (Figure 12). The Δ symbol denotes the smallest step for this number format, and it is equal to $1/256$ for numbers with 8 fractional bits. Therefore, the PE must include 11 comparisons. At the same time, the fixed-point format puts the integer part of the number in high order bits. Our models use 16-bit fixed-point numbers, and the 8 high order bits contain an integer part.

Decimal value	Fixed-point value
-5	1111 1011 0000 0000
$-4-\Delta$	1111 1011 1111 1111
-4	1111 1100 0000 0000
$-3-\Delta$	1111 1100 1111 1111
-3	1111 1101 0000 0000
$-2-\Delta$	1111 1101 1111 1111
-2	1111 1110 0000 0000
$-1-\Delta$	1111 1110 1111 1111
-1	1111 1111 0000 0000
$0-\Delta$	1111 1111 1111 1111
0	0000 0000 0000 0000
$1-\Delta$	0000 0000 1111 1111
1	0000 0001 0000 0000
$2-\Delta$	0000 0001 1111 1111
2	0000 0010 0000 0000
$3-\Delta$	0000 0010 1111 1111
3	0000 0011 0000 0000
$4-\Delta$	0000 0011 1111 1111
4	0000 0100 0000 0000
$5-\Delta$	0000 0100 1111 1111
5	0000 0101 0000 0000
$128-\Delta$ (max)	0111 1111 1111 1111

Figure 12. Approximation nodes. Dotted boxes indicate the bits used in Equations (2) (blue box), (3) (green box) and (4) (orange box).

The analysis of these values shows that the four high order bits are the same for all positive and all negative values (within the approximation range):

$$is\_neg = x_{15} \,\&\, x_{14} \,\&\, x_{13} \,\&\, x_{12} \tag{2}$$

$$is\_pos = \overline{x_{15}} \,\&\, \overline{x_{14}} \,\&\, \overline{x_{13}} \,\&\, \overline{x_{12}} \tag{3}$$

where x_n is the n-th bit of x.

The next four bits help to determine the specific range. For example, to check if the x value is in the range from -5 (inclusive) to -4 (excluding), the following formula can be used:

$$is\_neg5\_to\_neg4 = is\_neg \,\&\, x_{11} \,\&\, \overline{x_{10}} \,\&\, x_9 \,\&\, x_8 \tag{4}$$

For all values above 5, use the following function:

$$is\_pos5\_to\_inf = \overline{x_{15}} \,\&\, \overline{(is\_pos \,\&\, \overline{x_{11}}) \,\&\, (\overline{x_{10}} \mid (x_{10} \,\&\, \overline{x_9} \,\&\, \overline{x_8}))} \tag{5}$$

As a result, the approximation coefficients can be found as:

$$a = (is\_neg5\_to\_neg4 \,\&\, a1) \mid (is\_neg4\_to\_neg3 \,\&\, a2) \mid \cdots \mid (is\_pos4\_to\_pos5 \,\&\, a10) \mid (is\_pos5\_to\_inf \,\&\, 0) \tag{6}$$

$$b = (is\_neg5\_to\_neg4 \,\&\, b1) \mid (is\_neg4\_to\_neg3 \,\&\, b2) \mid \cdots \mid (is\_pos4\_to\_pos5 \,\&\, b10) \mid (is\_pos5\_to\_inf \,\&\, 1) \tag{7}$$

Thus, 11 comparisons of 16-bit numbers can be replaced by 11 comparisons of 5-bit numbers, which leads to the significant simplification of this implementation.

The centralized implementation has two major disadvantages. Firstly, a PE with the proposed implementation of the sigmoid occupies a larger area on a chip. Secondly, the focus on the sigmoid function does not allow to reuse this implementation for another activations.

7.2. Distributed Implementation

The distributed implementation is based on the ability to approximate each subrange of the sigmoid function independently. In other words, each subrange can be computed in parallel using separate chain of the PEs.

The complete distributed implementation is presented in Figure 13. The following color differentiation of operations is used here: green—"signal source", red—"minimum", yellow—"MAC" (the input value comes from the bottom, accumulating-from the left, and the weight is stored in the internal memory of the PE), blue—"gate", purple—"union". Dashed lines indicate the signals passing through the PEs without changes. Thus, distributed implementation requires three new operations, such as "minimum", "gate", and "union". However, these operations are the basic transformations, which can be effectively used in different computations. It is appropriate to include them in the final operations set.

The "gate" operation controls the propagation of the signal. It compares a value from the "key" input with the expected value stored in internal memory. If both values match, the PE passes the value from the main input to the main output. Otherwise, the main output value is zero. In the presented figure, the "key" input is on the bottom, the main input is on the left, and the main output is on the right. The key value keeps moving forward regardless of the comparison result. Since all gates expect different keys, at the most, one gate will have a non-zero output value.

As mentioned earlier, due to the integer nodes of the approximation, we only need to compare the 8 high order bits. However, "gate" is a general-purpose operation that realizes a comparison of the entire key. To fix this contradiction, the key shifting logic is introduced. It sets the 8 low order bits (fractional part) of the key value to "1" to generalize all possible keys. The expected gate values are shifted in the same way.

The "union" operation applies a bitwise OR to both inputs. It is used to merge the results of all gates and to generalize values of the key.

The "minimum" operation calculates the smallest of two numbers. It helps to exclude values of the key greater than 5, because according to the approximation used, all input values equal to or greater than 5 result in an output value of 1. So it is not necessary to process any key greater than 5. Input values below -5 are ignored in the implementation since the output of the approximation in this range is zero.

Thus, the distributed implementation of sigmoid activation requires 48 processing elements.

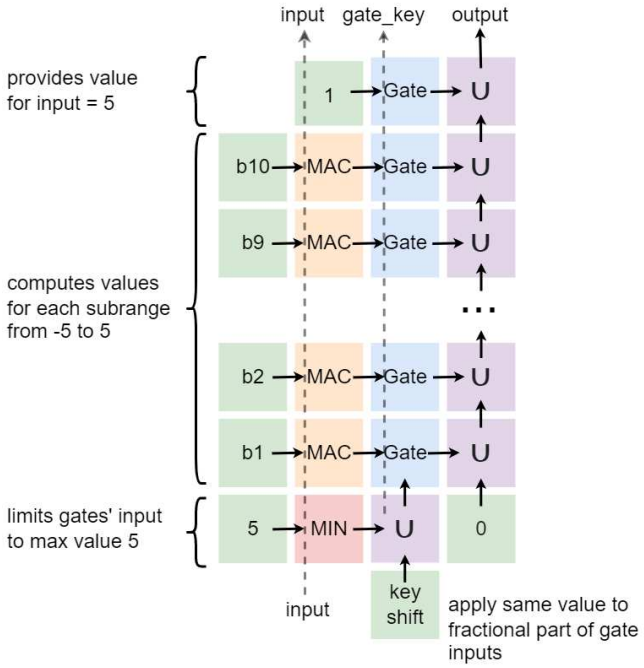

Figure 13. Distributed implementation of sigmoid activation.

8. Experimental Results

Two simulation models were developed to compare both presented implementations of sigmoid activation. The models are Verilog HDL modules designed in the Quartus Prime software. The modules support the entire operations set of the PE. Two key parameters of the modules were measured: the size (the number of required logic elements (LE) of the FPGA) and the maximum processing delay.

To evaluate these parameters, the developed modules were synthesized in the Quartus Prime (version 20.1.0, build 711 SJ Edition) for the Cyclone V (5CGFXC9E7F35C8) FPGA device. Usage of DSP blocks was disabled in the settings. Unwanted deletion of submodules during the optimization phase was prevented by the "synthesis keep" directive.

To measure processing delays, the Timing Analyzer tool was used. The Timing Analyzer is part of the Quartus software. All simulations were carried out with the predefined mode of the analyzer "Fast 1000mV 0C". During these simulations, the largest signal delay between the input and output of the module was measured. Both modules were pre-configured; thus, the configuration delays are eliminated from the results.

Due to bidirectional connections between PEs, the Timing Analyzer gets stuck in combinational loops. To avoid this problem, we removed unnecessary interconnections from the distributed sigmoid module for the duration of simulations.

In addition to the mentioned parameters, we measured the absolute error of the sigmoid implementation. The measurements were realized in a special PC application because the error value is not related to the hardware implementation and depends on the number format and the approximation algorithm. The application selects a random value from the approximation range $(-5, 5)$, rounds it to the nearest 16-bit fixed-point number, and evaluates the difference with the general sigmoid implementation at the original (before rounding) point. The algorithm is repeated one million times to get reliable statistics.

The results of all experiments are presented in Table 2.

Table 2. Experimental results.

Implementation	Total Size, LE	PE Size, LE	Max Delay, ns	Average Absolute Error	Max Absolute Error
Centralized	312	312	14	4×10^{-3}	1×10^{-2}
Distributed	13,175	296	18.5		

Note, the centralized implementation is 24.4% faster than the distributed one. However, in fact, the difference is only 4.5 ns, which is not important for most real systems. The sigmoid is usually used in the output layer of NN, and its contribution to the overall processing time is relatively small. The second advantage of the centralized implementation is a convenient configuration. This approach requires only 1 PE to be configured to perform the entire approximation algorithm, in contrast to the distributed approach, where 48 PEs must be configured. Another advantage is the bit-length of the configuration signal. The more operations the PE supports, the more bits are required to encode them all. The distributed implementation introduces three new operations, while the centralized uses only one. However, the operations used in the distributed implementation can be efficiently reused to compute different functions (not only the sigmoid function), in contrast to the centralized implementation.

The key advantage of the distributed implementation is the area on a chip occupied by PE. With this approach, each PE occupies 5.1% less area. Thus, many more PEs can be placed in an RCE of the same size. It is a significant improvement since a typical RCE can contain thousands of PEs. The second benefit is the simplicity of the PE, which leads to improved reliability. In addition, the parallel processing inherent to the distributed realization is more consistent to the key principles of the reconfigurable computing environments design.

The results of comparison with alternative research are presented in Table 3. Data about counterparts are taken from [37]. Presented implementations of sigmoid activation have an acceptable average error and very high performance. However, the distributed implementation requires more logic elements, since the calculations are distributed among many processing elements, each of which supports a complete set of operations. In addition, our models use combinational logic, which provides high performance at the expense of a larger area on a chip.

Table 3. Comparison of the developed models and alternative research.

Implementation	E_{max}	E_{avg}	Input Format	Output Format	LUT	DSP	Delay, ns
Hajduk (McLaurin)	1.192×10^{-7}	1.453×10^{-8}	32b FP	32b FP	1916	4	940.8
Hajduk (Pade)	1.192×10^{-7}	3.268×10^{-9}	32b FP	32b FP	2624	8	494.4
Zaki et al.	1×10^{-2}	N/A	32b FP	32b FP	363	2	N/A
Tiwari et al.	4.77×10^{-5}	N/A	32b FXP	32b FXP	1388	22	1590–2130
Tsmots et al.	1.85×10^{-2}	5.87×10^{-3}	16b FXP	16b FXP	N/A	N/A	N/A
Wei et al.	1.25×10^{-2}	4.2×10^{-3}	16b FXP	12b FXP	140	0	9.856
PLAN	1.89×10^{-2}	5.87×10^{-3}	16b FXP	16b FXP	235	0	30
NRA	5.72×10^{-4}	8.6×10^{-5}	16b FXP	16b FXP	351	6	85
Centralized	1×10^{-2}	4×10^{-3}	16b FXP	16b FXP	312	0	14
Distributed	1×10^{-2}	4×10^{-3}	16b FXP	16b FXP	13175	0	18.5

N/A—not assessed, FP—floating-point number, FXP—fixed-point number.

9. Application for Other Activation Functions

Sigmoid activation is one of the most popular and well known, but many other activations are used in practice. The simplest of them (ReLU, LeakyReLU, and PReLU) can be implemented in the proposed RCE by "maximum" and "MAC" operations and do not require a complex design. Swish activation [18] is based on the sigmoid and requires minor

modifications to the proposed design. P-Swish activation [19] is a combination of Swish and ReLU functions, so it can be implemented with "minimum" and "gate" operations.

As mentioned above, the distributed implementation of the sigmoid activation can be effectively reused to perform approximations of another functions. Thus, the proposed RCE is able to support a wide variety of activations. The approximation of the exponential function makes it possible to implement ELU [20] and Softmax [21] activations. The piecewise linear approximations of Tanh and Softplus activations can be introduced according to the given design. To improve the accuracy of these approximations, intervals of small or variable length can be used. To support variable-length intervals, the key-shift operation must be used several times with different shift values.

10. Conclusions

Modern intelligent systems are increasingly faced with the need to use computationally complex machine learning algorithms. However, low-power systems have severe restrictions to their weight and power consumption. To solve this issue, dynamically reconfigurable hardware accelerators based on the reconfigurable computing environments can be used. However, the efficiency of accelerators depends on the implementation of a set of operations of the PEs. The sigmoid function is one of the most popular activations in neural networks. However, its general form is computationally complex. Due to this, in practice, it is replaced by simplified analogues.

This paper proposes two hardware implementations of the sigmoid activation on the RCE. The centralized implementation has high performance and a simple configuration process, but it leads to an increase in the size of each PE. The distributed implementation has lower performance, requires more LEs, and uses only simple operations. As a result, each PE has a smaller size.

The experimental results show high performance (the largest signal delay is 14–18.5 ns) and acceptable accuracy (average and maximum errors are 4×10^{-3} and 1×10^{-2}, respectively) of the proposed sigmoid activation implementations compared to the existing alternatives.

Author Contributions: Conceptualization, V.S. and S.S.; data curation, D.S.; formal analysis, S.S.; funding acquisition, D.S.; investigation, V.S.; methodology, D.S.; project administration, S.S.; resources, S.S.; software, V.S.; supervision, D.S.; validation, D.S.; visualization, V.S.; writing—original draft, V.S.; writing—review and editing, S.S. All authors have read and agreed to the published version of the manuscript.

Funding: The research was supported by the Russian Science Foundation, grant No. 21-71-00012, https://rscf.ru/project/21-71-00012/ (accessed on 18 May 2022).

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Chen, J.; Li, J.; Majumder, R. Make Every Feature Binary: A 135B Parameter Sparse Neural Network for Massively Improved Search Relevance. Available online: https://www.microsoft.com/en-us/research/blog/make-every-feature-binary-a-135b-parameter-sparse-neural-network-for-massively-improved-search-relevance/ (accessed on 20 March 2022).
2. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Amodei, D. Language Models are Few-Shot Learners. *arXiv* **2020**, arXiv:2005.14165v4.
3. Carrio, A.; Sampedro, C.; Rodriguez-Ramos, A.; Campoy, P. A Review of Deep Learning Methods and Applications for Unmanned Aerial Vehicles. *J. Sens.* **2017**, *2017*, 3296874. [CrossRef]
4. Nabavinejad, S.M.; Reda, S.; Ebrahimi, M. Coordinated Batching and DVFS for DNN Inference on GPU Accelerators. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 1–12. [CrossRef]
5. Guo, J.; Liu, W.; Wang, W.; Yao, C.; Han, J.; Li, R.; Hu, S. AccUDNN: A GPU Memory Efficient Accelerator for Training Ultra-Deep Neural Networks. In Proceedings of the 2019 IEEE 37th International Conference on Computer Design (ICCD), Abu Dhabi, United Arab Emirates, 17–20 November 2019; pp. 65–72.

6. Chajan, E.; Schulte-Tigges, J.; Reke, M.; Ferrein, A.; Matheis, D.; Walter, T. GPU based model-predictive path control for self-driving vehicles. In Proceedings of the 2021 IEEE Intelligent Vehicles Symposium (IV), Nagoya , Japan, 11–15 July 2021; pp. 1243–1248.
7. Chang, K.C.; Fan, C.P. Cost-Efficient Adaboost-based Face Detection with FPGA Hardware Accelerator. In Proceedings of the 2019 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Yilan, Taiwan, 20–22 May 2019; pp. 1–2.
8. Lee, J.; He, J.; Wang, K. Neural Networks and FPGA Hardware Accelerators for Millimeter-Wave Radio-over-Fiber Systems. In Proceedings of the 2020 22nd International Conference on Transparent Optical Networks (ICTON), Bari, Italy, 19–23 July 2020; pp. 1–4.
9. Yu, L.; Zhang, S.; Wu, N.; Yu, C FPGA-Based Hardware-in-the-Loop Simulation of User Selection Algorithms for Cooperative Transmission Technology Over LOS Channel on Geosynchronous Satellites. *IEEE Access.* **2022**, *10*, 6071–6083. [CrossRef]
10. Kyriakos, A.; Papatheofanous, E.-A.; Bezaitis, C.; Reisis, D. Resources and Power Efficient FPGA Accelerators for Real-Time Image Classification. *J. Imaging* **2022**, *8*, 114. [CrossRef] [PubMed]
11. Lamoral Coines, A.; Jiménez, V.P.G. CCSDS 131.2-B-1 Transmitter Design on FPGA with Adaptive Coding and Modulation Schemes for Satellite Communications. *Electronics* **2021** , *10*, 2476. [CrossRef]
12. Sakai, Y. Quantizaiton for Deep Neural Network Training with 8-bit Dynamic Fixed Point. In Proceedings of the 2020 7th International Conference on Soft Computing and Machine Intelligence (ISCMI), Stockholm, Sweden, 14–15 November 2020; pp. 126–130.
13. Trusov, A.; Limonova, E.; Slugin, D.; Nikolaev, D.; Arlazarov, V.V. Fast Implementation of 4-bit Convolutional Neural Networks for Mobile Devices. In Proceedings of the 2020 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, 10–15 January 2021; pp. 9897–9903.
14. Liu, Z.; Zhang, H.; Su, Z.; Zhu, X. Adaptive Binarization Method for Binary Neural Network. In Proceedings of the 2021 40th Chinese Control Conference (CCC), Shanghai, China, 26–28 July 2021; pp. 8123–8127.
15. Zhu, B.; Al-Ars, Z.; Hofstee, H.P. NASB: Neural Architecture Search for Binary Convolutional Neural Networks. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020; pp. 1–8.
16. Tang, Z.; Luo, L.; Xie, B.; Zhu, Y.; Zhao, R.; Bi, L.; Lu, C. Automatic Sparse Connectivity Learning for Neural Networks. In Proceedings of the 2022 IEEE Transactions on Neural Networks and Learning Systems, Padua, Italy, 22 April 2022; pp. 1–15.
17. Haykin, S. *Neural Network: A Comprehensive Foundation*, 2nd ed.; Prentice Hall International, Inc.: Hoboken, NJ, USA, 1999; 842p.
18. Ramachandran, P.; Zoph, B.; Le, Q.V. Searching for Activation Functions. In Proceedings of the ICLR 2018 Conference, Vancouver, BC, Canada, 30 April–3 May 2018; pp. 1–13.
19. Mercioni, M. A.; Holban, S. P-Swish: Activation Function with Learnable Parameters Based on Swish Activation Function in Deep Learning. In Proceedings of the 2020 International Symposium on Electronics and Telecommunications (ISETC), Timisoara, Romania, 5–6 November 2020; pp. 1–4. [CrossRef]
20. Devi, T.; Deepa, N. A novel intervention method for aspect-based emotion Using Exponential Linear Unit (ELU) activation function in a Deep Neural Network. In Proceedings of the 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 6–8 May 2021; pp. 1671–1675. [CrossRef]
21. Hu, R.; Tian, B.; Yin, S.; Wei, S. Efficient Hardware Architecture of Softmax Layer in Deep Neural Network. In Proceedings of the 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP), Shanghai, China, 19–21 November 2018; pp. 1–5. [CrossRef]
22. Lee, K.J.; Lee, J.; Choi, S.; Yoo, H.-J. The Development of Silicon for AI: Different Design Approaches. *IEEE Trans. Circuits Syst.* **2020**, *67*, 4719–4732. [CrossRef]
23. Kan, Y.; Wu, M.; Zhang, R.; Nakashima, Y. A multi-grained reconfigurable accelerator for approximate computing. In Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Limassol, Cyprus, 6–8 July 2020; pp. 90–95.
24. Khalil, K.; Eldash, O.; Dey, B.; Kumar, A.; Bayoumi, M. A Novel Reconfigurable Hardware Architecture of Neural Network. In Proceedings of the IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS), Dallas, TX, USA, 4–7 August 2019; pp. 618–621.
25. Chen, Y.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE J.-Solid-State Circuits* **2017**, *52*, 127–138. [CrossRef]
26. Chen, Y.H.; Yang, T.J.; Emer, J.; Sze, V. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE J. Emerg. Sel. Top. Circuits Syst. (Jetcas)* **2019**, *9*, 292–308. [CrossRef]
27. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Yoon, D.H. In-Datacenter Performance Analysis of a Tensor Processing Unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17), Toronto, ON, Canada, 24–28 June 2017; pp. 1–12.
28. Bondarchuk, A.S.; Shashev, D.V.; Shidlovskiy, S.V. Design of a Model of a Reconfigurable Computing Environment for Determining Image Gradient Characteristics. *Optoelectron. Instrum. Data Process.* **2021**, *57*, 132–140. [CrossRef]
29. Evreinov, E.V. *Homogeneous Computing Systems, Structures and Environments*; Radio and Communication: Moscow, Russia, 1981; 208p.
30. Kung, S.Y. *VLSI Array Processors*; Prentice Hall Information and System Sciences Series; Englewood Cliffs: Bergen, NJ, USA, 1988; 600p.
31. Wanhammar, L. *DSP Integrated Circuits*; Academic Press Series in Engineering: Cambridge, MA, USA, 1999; 561p.
32. Ghimire, D.; Kil, D.; Kim, S.-H. A Survey on Efficient Convolutional Neural Networks and Hardware Acceleration. *Electronics* **2022**, *945*, 945. [CrossRef]

33. Shatravin, V.; Shashev, D.V. Designing high performance, power-efficient, reconfigurable compute structures for specialized applications. *J. Phys. Conf. Ser.* **2020**, *1611*, 1–6. [CrossRef]
34. Shatravin, V.; Shashev, D.V.; Shidlovskiy S.V. Applying the Reconfigurable Computing Environment Concept to the Deep Neural Network Accelerators Development. In Proceedings of the International Conference on Information Technology (ICIT), Guangzhou, China, 15–17 January 2021; Volume 1611, pp. 842–845.
35. Shatravin, V.; Shashev, D.V.; Shidlovskiy S.V. Developing of models of dynamically reconfigurable neural network accelerators based on homogeneous computing environments. In Proceedings of the XXIV International Scientific Conference Distributed Computer and Communication Networks: Control, Computation, Communications (DCCN), Moscow, Russia, 26–30 September 2021; pp. 102–107.
36. Faiedh, H.; Gafsi, Z.; Besbes, K. Digital Hardware Implementation of Sigmoid Function and its Derivative for Artificial Neural Networks. In Proceedings of the 13 International Conference on Microelectronics, Rabat, Morocco, 29–31 October 2001; pp. 189–192.
37. Pan, Z.; Gu, Z.; Jiang, X.; Zhu, G.; Ma, D. A Modular Approximation Methodology for Efficient Fixed-Point Hardware Implementation of the Sigmoid Function. *IEEE Trans. Ind. Electron.* **2022**, *69*, 10694–10703. [CrossRef]