applied sciences

MDPI

Article

Implementation of the SoftMax Activation for Reconfigurable Neural Network Hardware Accelerators

Vladislav Shatravin *, Dmitriy Shashev and Stanislav Shidlovskiy

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

* Correspondence: shatravin@stud.tsu.ru

Abstract: In recent decades, machine-learning algorithms have been extensively utilized to tackle various complex tasks. To achieve the high performance and efficiency of these algorithms, various hardware accelerators are used. Typically, these devices are specialized for specific neural network architectures and activation functions. However, state-of-the-art complex autonomous and mobile systems may require different algorithms for different tasks. Reconfigurable accelerators can be used to resolve this problem. They possess the capability to support diverse neural network architectures and allow for significant alterations to the implemented model at runtime. Thus, a single device can be used to address entirely different tasks. Our research focuses on dynamically reconfigurable accelerators based on reconfigurable computing environments (RCE). To implement the required neural networks on such devices, their algorithms need to be adapted to the homogeneous structure of RCE. This article proposes the first implementation of the widely used SoftMax activation for hardware accelerators based on RCE. The implementation leverages spatial distribution and incorporates several optimizations to enhance its performance. The timing simulation of the proposed implementation on FPGA shows a high throughput of 1.12 Gbps at 23 MHz. The result is comparable to counterparts lacking reconfiguration capability. However, this flexibility comes at the expense of the increased consumption of logic elements.

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

Citation: Shatravin, V.; Shashev, D.; Shidlovskiy, S. Implementation of the SoftMax Activation for Reconfigurable Neural Network Hardware Accelerators. *Appl. Sci.* **2023**, *13*, 12784. https://doi.org/10.3390/app132312784

Academic Editors: Paris Kitsos and Vincent A. Cicirello

Received: 20 April 2023
Revised: 15 June 2023
Accepted: 24 June 2023
Published: 28 November 2023

1. Introduction

Machine-learning algorithms, particularly artificial neural networks (NN), are one of the most promising technologies these days. NNs possess the capability to extract relationships between properties of source data objects, making them highly applicable for numerous tasks that cannot be efficiently addressed by traditional rigid algorithms: object classification, image recognition, predictive analysis, natural language processing, and many others. The training process, which involves extracting the relationships, enables NNs to solve such tasks for unknown input data.

The complexity of solving tasks requires the use of large NN models with a substantial number of parameters. These models have high computational complexity and, therefore, require specialized computing systems known as neural network hardware accelerators. Hardware accelerators are well optimized specifically for machine-learning algorithms, performance, and power efficiency. The significance of hardware accelerators is particularly pronounced in mobile and standalone systems, such as mobile robots, smartphones, unmanned aerial vehicles (UAVs), self-driving cars, satellites, smart Internet of Things (IoT) sensors, edge computing nodes, and various other devices [1–8]. These systems impose stringent requirements on factors such as durability, weight, and battery life.

The issue of the computational complexity of neural networks is well-recognized, and numerous implementations of hardware accelerators have been proposed. All of

them are based on devices that support parallel computing: graphics processing units (GPU), field-programmable gate arrays (FPGA), and application-specific integrated circuits (ASIC) [2,8–12]. GPU consists of numerous small processing units working independently of each other; consequently, it ensures high performance in machine-learning tasks [1,9]. Nevertheless, the high power consumption of GPUs limits their use primarily to cloud, server, and desktop systems. FPGAs and ASICs are more widespread in low-power and autonomous devices. They allow one to develop a computing unit with an architecture customized for a particular task. The customizations and low-level implementation ensures the high performance and power efficiency of such accelerators. To use the benefits of the chosen platform, the required algorithms need to be adapted to its peculiarities.

In addition to the specialized hardware, numerous optimizations are employed. Due to the redundancy of neural networks, some weights can be eliminated without any significant loss [13]. The high sustainability of NNs to distorted input data allows one to reduce the bit depth of both input data and parameters [14–18].

These approaches exhibit enhanced performance and efficiency for specific tasks. However, complex intelligent systems encounter various challenges, each of which may require an NN model with a specific architecture. Furthermore, in numerous cases, there is a lack of prior knowledge regarding the algorithms that will be required in the future. This issue is particularly relevant for remote systems with low availability, where limitations in power consumption, weight, and reliability make it challenging to carry multiple task-specific devices for any possible task.

A more promising solution involves hardware accelerators with the reconfiguration ability [19–22]. Such accelerators can drastically change the implemented NN model by an external signal at runtime. At the same time, reconfigurable accelerators offer comparable performance, energy efficiency, and reliability to their rigid counterparts. These features are of primary relevance to engineers, leading to numerous studies in this field. The concept of reconfigurable computing environments (RCEs) can be utilized to build hardware accelerators with such capabilities. The RCE concept proposes implementing computing devices as a homogeneous grid of independently configurable processing elements (PE).

A significant challenge of applying RCE-based accelerators is the adaptation of the required machine-learning algorithms to the multi-cell homogeneous structure of RCE. Each element of the RCE only supports a small set of specific operations. At the same time, all elements are interconnected and work in parallel. Thus, the desired algorithms must be implemented as a spatially distributed composition of the given operations. In the research, we propose the PE with 10 simple operations: signal source, signal transfer, multiply-accumulate, parametric ReLU, maximum, minimum, gate, union, delay, and block [21,23–25]. There is currently no implementation of SoftMax activation based on spatial distribution and these operations. This paper considers the implementation of SoftMax activation for the RCE. Activation has a lot in common with sigmoid activation, the implementation of which is presented in [25]. However, due to its higher computational complexity, SoftMax activation requires more optimizations, as will be discussed further.

2. Reconfigurable Computing Environment

A reconfigurable computing environment (RCE) is a mathematical model of a computing system built as a regular grid of similar small processing elements (PEs). Each element is connected with its neighbors. The shape of a PE corresponds to the number of its neighbors. Each PE supports several operations; however, it only performs one of them at any given time. The performing operation can be set by an external signal or by some predefined internal rule. This setting is called the configuration of the PE. To ensure the high flexibility and performance of an RCE, its PEs should be small and simple. This can be achieved by implementing only a small set of straightforward reusable operations. Although each element is very simple, a large group of such elements working together has many advantages. The collaborative work of many independent processors allows for the implementation of complex distributed and parallel algorithms [26,27]. Since all elements

are similar, in the case of partial damage to the RCE, the current algorithm can be restored to an intact area. The complexity of the algorithms supported by an RCE depends only on the number of PEs in the RCE and their set of operations. In some references, RCE is also referred to as a homogeneous structure [28].

In general, a computing environment can be one-, two- or three-dimensional, and its elements can have a different number of neighbors. This paper discusses two-dimensional environments in which elements have up to four neighbors (Figure 1).

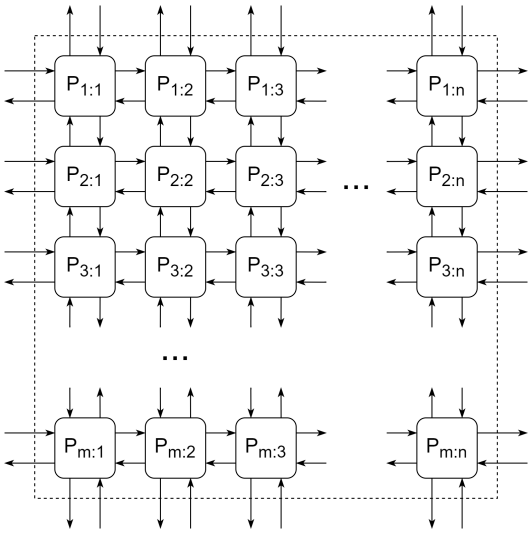

Figure 1. Reconfigurable computing environment.

3. Implementation of Neural Networks in RCE

As previously noted, an RCE is a general design paradigm for computing systems with high parallelism and reconfiguration ability. The capabilities and characteristics of such systems essentially depend on the predefined set of PE operations, as well as the interaction between their elements. Consequently, there are many ways to implement the required algorithms.

This paper explores the implementation of neural networks in an RCE that supports fine-grained reconfiguration and highly distributed computing. In accordance with our architecture, an artificial neuron can be implemented in the form of a processing elements chain (Figure 2). Separate elements perform different operations, such as storing the bias, weighting and adding each input, and applying the activation. Therefore, neurons with any number of inputs can be implemented by altering the length of this chain.

Obviously, the size and complexity of an RCE processing element is highly dependent on the number and complexity of operations it supports. Thus, complex operations, such as the sigmoid or SoftMax activations, should be decomposed into simple, reusable ones. In other words, the complex operations should be implemented by a collaboration of several processing elements. The literature review allowed us to identify the following set of operations: “signal source” (SRC), “signal transfer” (TRS), “multiply and accumulate” (MAC), “parametric ReLU” (PRL) “maximum” (MAX), “minimum” (MIN), “gate” (GAT), “union” (U), “delay” (DEL), and “block” (BLK) (Figure 3). These operations are sufficient for implementing the key layers of neural networks (dense, convolution, pooling) as well as activations (sigmoid, tanh, ELU, and other) [21,23–25]. This paper focuses on the implementation of SoftMax activation.

Certain operations require an additional argument, which can be set during the configuration process and stored in the internal memory of the PE. For instance, the MAC operation treats this argument as a weight value, while the PRL employs it for its coefficient.

Furthermore, each PE is configured with one of four possible signal flow directions. As will be demonstrated later, the direction setting enables the implementation of the pipeline processing in an RCE.

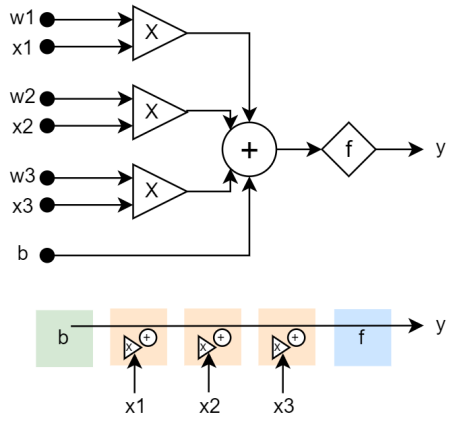

Figure 2. Implementation of a neuron with RCE processing elements. Reprinted from [23], with permission 2021 IEEE.

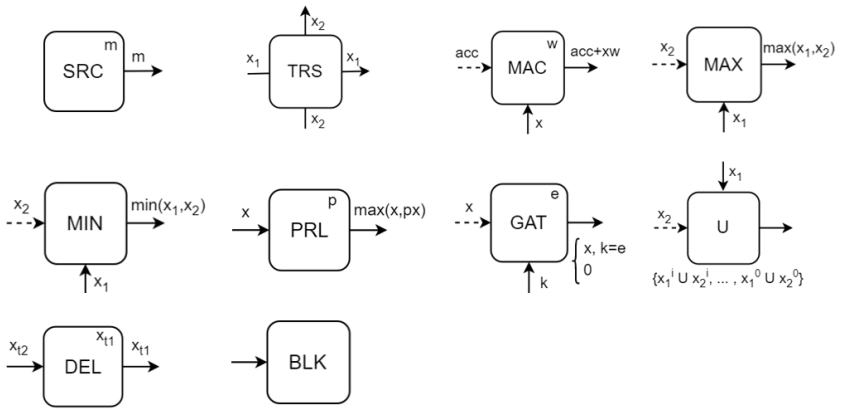

Figure 3. Processing element operation set.

By utilizing the proposed approach, the layers of a neural network can be implemented by connecting several chains (Figure 4). The values *b1*, *b2*, *b3* correspond to the neuron biases. Neuron weights are stored in the elements that are configured for the MAC operation (depicted as orange boxes). The symbol *f* denotes an activation function. Since all processing elements work independently, all neurons of the layer are computed in parallel, resulting in enhanced performance.

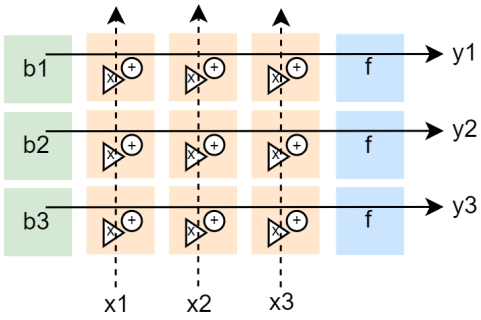

Figure 4. Dense layer in RCE.

As is evident from Figure 4, the proposed implementation of the neuron rotates the signal by 90 degrees. This rotation serves two purposes: to stack the neuron chains together and to compute multiple layers concurrently. The simultaneous processing of several neural network layers can be accomplished through pipelining. To process n layers, the RCE needs to be divided into $n + 1$ segments. At each moment of time, n segments implement different layers while the last segment is reconfigured to compute the next layer (Figure 5). Each segment processes signals that arrive at different points in time and passes the results to

the following segment, which is configured for the next layer of the network. At each step, the reconfiguration process is performed for the subsequent segment. Thus, the full reconfiguration of the RCE requires n steps. Through the simultaneous computation and dynamic reconfiguration, this implementation of RCE is capable of processing NNs with an arbitrary number of layers. Furthermore, the implementation has decreased time and power consumption since all intermediate results of signal processing are kept in the RCE, and data exchange with an external memory is mostly eliminated.

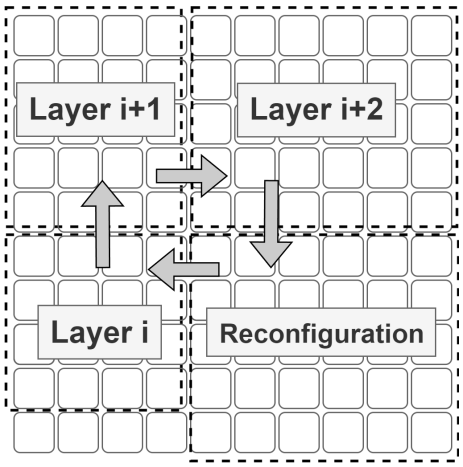

Figure 5. Pipeline processing in RCE.

4. Implementation of the SoftMax Activation in RCE

4.1. Challenges of the Hardware Implementation of SoftMax Activation

The SoftMax is a neuron activation widely used in the output layers of neural networks for classification tasks if the number of output classes exceeds one. It is also known as the softargmax or normalized exponential function. The result of this activation is the probability distribution of the input vector values among several classes. Consequently, SoftMax is a multi-class alternative to the sigmoid activation. The SoftMax activation has the following form:

$$S(x_1, \ldots x_N)_i = \exp(x_i) / \sum_{1}^{N} \exp(x_n) \quad (1)$$

where N is the number of inputs.

As can be seen, there are several challenges to hardware implementation of this function:

- It is based on the non-linear fast-growing exponential function (Figure 6);
- In general, the input values of the activation can lie within a wide range, limiting possible optimizations;
- It utilizes division to produce the final result.

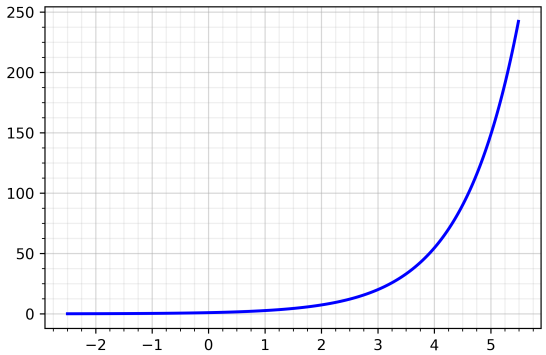

Figure 6. The exponential function.

The wide usage of the SoftMax activation has led to numerous studies addressing these difficulties.

4.2. Current State of Hardware Implementation of the SoftMax Activation

According to Equation (1), the SoftMax activation incorporates the exponential function. This is the non-linear fast-growing function (Figure 6). Since the hardware implementation of this function is resource-intensive, researchers have proposed various approaches to optimize it. In papers [29–32], the exponential function is implemented by storing its points in an internal memory of FPGA (look-up table, LUT). This is a simple and effective solution. However, to achieve sufficient accuracy, the number of stored points must be large enough. As a result, this solution is convenient for sequential processing only when the device computes its inputs one by one. To perform the calculations in parallel, each branch needs its own table of points, leading to high memory consumption. To reduce the memory footprint, in [29], the power of e is decomposed into digits. The exponential function values for these points are stored in the device memory. These values are then multiplied to produce the final result.

More promising solutions utilize bitwise shift operation [13,33]. They propose replacing e^x with 2^{y+z}, where y and z are the integer and fractional parts, respectively. Since the bitwise shift has a well-known efficient hardware implementation, the solution provides good results. Nevertheless, performing the bitwise shift operation requires several clock cycles, leading to difficulties for asynchronous devices. The papers [22,30] propose a similar solution based on the Taylor expansion and bitwise shift.

Unfortunately, both presented solutions cannot be efficiently implemented in an RCE. The LUT-based implementations are not applicable due to the homogeneity of RCE—all its processing elements must be identical. Storing the entire look-up table of the exponential function in each PE results in low efficiency and high memory consumption. Furthermore, the stored points are applicable for the exponential function only and cannot be reused for other tasks. This fact violates the key design principle of RCE processing elements that imply high reusability of operations in an operation set. The implementations based on bitwise shift cannot be employed in an RCE for several reasons. As mentioned before, the bitwise shift operation implies synchronous data processing and takes several clock cycles. Nevertheless, the RCE processing elements work asynchronously and do not rely on the clock cycle. The second obstacle is the lack of bitwise shift operation in the proposed set of PE operations. This operation can be introduced into the set; however, at the date, the necessity for this operation in PEs is debatable.

The wide input range of the SoftMax activation increases the complexity of its hardware implementation. To reduce the range, most of the mentioned papers suggest subtracting the maximum input value from all inputs [29,31,32,34]. This is possible due to Equation (2). If c is equal to $-max(x_1, \ldots x_n)$, the results of the exponential function are always between 0 and 1. However, the subtraction leads to negative powers and small values of the exponential function, resulting in low precision for fixed-point numbers. Since the proposed implementation of RCE processes 16-bit fixed-point numbers, this solution has a significant impact on the results obtained. In this paper, we have modified this approach to provide greater accuracy.

$$S(x_1, \ldots x_N)_i = \frac{\exp(x_i)}{\sum_1^N \exp(x_n)} = \frac{\exp(c) * \exp(x_i)}{\exp(c) * \sum_1^N \exp(x_n)} = \frac{\exp(x_n + c)}{\sum_1^N \exp(x_n + c)} \quad (2)$$

where c is any constant value.

The hardware implementation of division is a well-recognized problem that increases the complexity of the design. A simplified implementation of this operation can be achieved using subtraction and bitwise shift, as shown in [29,30,33]; or by addition and bitwise shift, according to [34]. The problem of applying bitwise shift in RCE was discussed above. The research [31] proposes avoiding division through rounding due to the primary impact of the maximum input value. Obviously, this approach has limited use.

In summary, although the modern implementations of SoftMax activation ensure high performance and resource utilization, they cannot be effectively used for reconfigurable computing environments. Consequently, to benefit from the reconfiguration capability, a novel RCE-specific implementation of SoftMax activation is required.

4.3. Implementation of the Exponential Function

As discussed earlier, in RCE, complex functions need to be decomposed among a group of processing elements; thus, each element performs a simple operation. One of the most popular approaches to defining a function in a simple general way is piecewise linear approximation (PLA). This approach can be used to implement the exponential function.

We briefly discuss the implementation of PLA in the RCE using the example of sigmoid activation described in detail in the paper [25]. This implementation is based on the approximation with equal subranges—the entire approximation range is divided into several subranges of equal width. Then, in each subrange, the value of the original function is replaced by the straight line defined by the Equation (3). The a and b values are chosen to reduce the difference between the original and approximated functions in this subrange. The piecewise linear approximation is a popular approach to implement complex functions in hardware accelerators [25,35,36].

$$f(x) = ax + b \tag{3}$$

where a and b are the multiplier and addend of the subrange in which the x is located.

Due to the shape of sigmoid activation (Figure 7), the approximation in the range $[-5, 5]$ with 10 subranges provides acceptable results [25,35]. The input values below -5 lead to zero output; the values above 5 result in one. Since all the subranges are calculated independently of each other, they can be processed simultaneously by different processing elements (Figure 8). Thus, each subrange is implemented in the form of a chain of PEs. Each chain has the GAT element that passes the computed value if the input value lies within the subrange of this chain. Otherwise, it outputs zero.

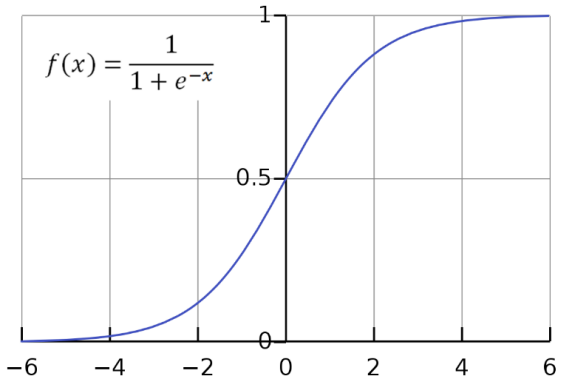

Figure 7. Sigmoid activation.

To ensure whether the input value belongs to a certain subrange, it is necessary to compare this value with the reference value of the subrange, ignoring insignificant bits. The insignificant bits are the least significant bits that cannot affect the subrange of the signal. This can be achieved by applying a special mask to the input value. In this paper, it is proposed to use a mask containing ones in insignificant bits and zeroes in the rest of the bits. The application of this mask implies performing the bitwise OR operation on the mask and the input signal. Since the proposed models work with 16-bit fixed-point numbers with eight fractional bits, as well as approximation points are integer values (-5, -4, ... 4, 5), the eight least significant bits (fractional part) of the mask are equal to one, the rest are equal to zero. After applying this mask, the input value keeps its integer part unmodified, and its fractional part is always "11111111". Similarly, the subrange reference value is the

result of masking any point of the subrange. Thus, the GAT elements compare the masked input values with masked approximation points.

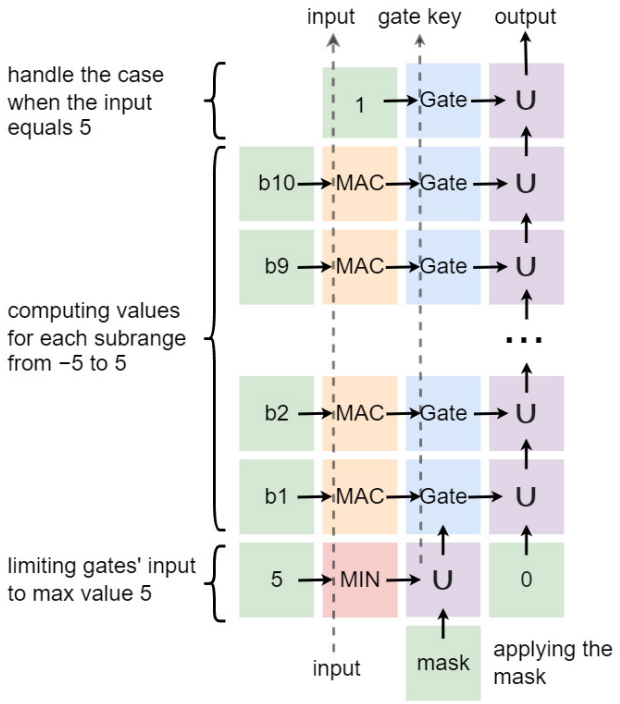

Figure 8. Sigmoid activation in the RCE.

Obviously, this approach can be applied to other functions as well. It is possible to use any approximation step with a length equal to a power of two (2, 1, 1/2, 1/4, etc.) by varying the applied mask. An approximation with any required number of subranges can be accomplished by altering the number of the involved PE rows.

Nevertheless, as noted earlier, the exponential function is non-linear and fast-growing (Figure 6). As a result, the piecewise linear approximation of this function with equal steps introduces a significant error. One possible solution to this problem is an approximation with a variable step. In this paper, an approximation of the exponential function in the range $[-2.5, 2.5]$ with 12 varying steps is proposed. The reasons for choosing such an interval will be discussed in the next subsection. The lengths of these steps are distributed as follows: from -2.5 to 1.5, the step is 0.5, and from 1.5 to 2.5, the step is 0.25. Both step lengths are a power of two, so they can be implemented using the masking operation discussed above. The corresponding approximation parameters are presented in Table 1. The evaluation of the proposed approximation shows acceptable results: the average absolute error is 0.027, the relative error is 1.9%, the maximum absolute error is 0.1, and the relative error is 5.8%.

The masks should be applied to the signal in a certain order: from the least restrictive (for the smallest step size) to the most restrictive in ascending order. Among the mentioned masks, the least restrictive is the mask for step 0.25, which has the value 0000 0000 0011 1111. The second mask corresponds to the step size 0.5, whose value is 0000 0000 0100 0000, since the least significant bits are already set to one by the first mask. The resulting masked values for the GAT elements are shown in Table 1.

The proposed approximation of the exponential function for the RCE is presented in Figure 9. It utilizes 59 processing elements. The main idea is similar to the previously discussed sigmoid activation, except that one additional mask is applied. All the approximation subranges are computed in parallel; however, at most, one GAT element is opened for any input value. If all these gates are closed, the entire design outputs zero, meaning that the input is below the chosen approximation range $[-2.5, 2.5]$. As will be shown in the next subsection, the processing of input values greater than 2.5 is not required. Neverthe-

less, it is possible to introduce it in a manner similar to the above sigmoid implementation, where the input is constrained by a PE specifically configured for the MIN operation.

Table 1. The parameters of the exponential function approximation.

Step	Range	a	b	Masked Point
1	[−2.5, −2.0)	0.1015625	0.3359375	1111 1101 1111 1111
2	[−2.0, −1.5)	0.1796875	0.4921875	1111 1110 0111 1111
3	[−1.5, −1.0)	0.2890625	0.65625	1111 1110 1111 1111
4	[−1.0, −0.5)	0.4765625	0.84375	1111 1111 0111 1111
5	[−0.5, 0.0)	0.7890625	1	1111 1111 1111 1111
6	[0.0, 0.5)	1.296875	1	0000 0000 0111 1111
7	[0.5, 1.0)	2.1328125	0.58203125	0000 0000 1111 1111
8	[1.0, 1.5)	3.53125	−0.81640625	0000 0001 0111 1111
9	[1.5, 1.75)	5.09375	−3.16015625	0000 0001 1011 1111
10	[1.75, 2.0)	6.53125	−5.67578125	0000 0001 1111 1111
11	[2.0, 2.25)	8.390625	−9.39453125	0000 0010 0011 1111
12	[2.25, 2.5]	10.78125	−14.7734375	0000 0010 0111 1111

The values of the applied masks are underlined.

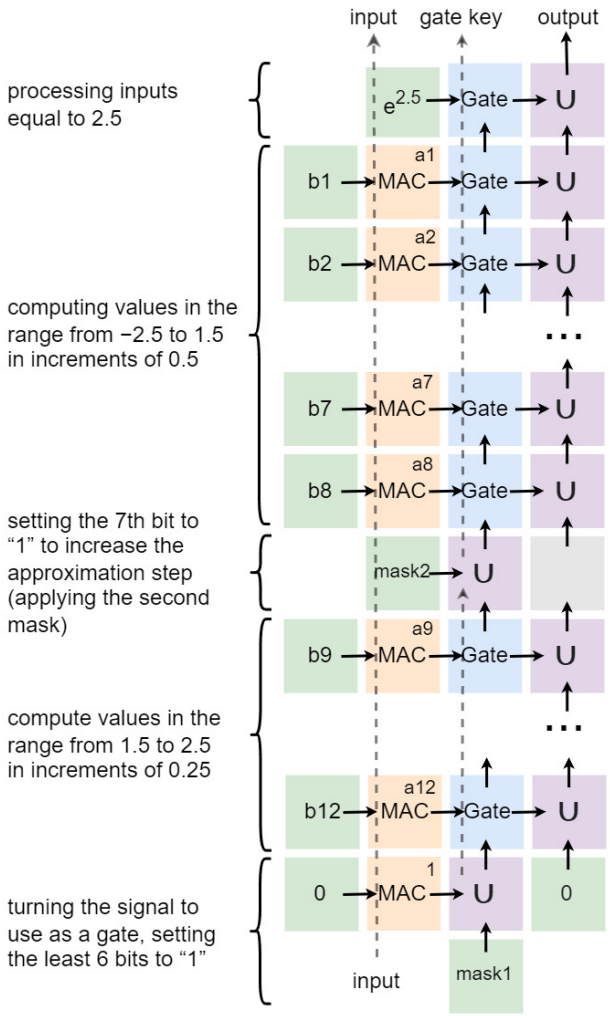

Figure 9. Implementation of the exponential function in the RCE.

4.4. Reduction of the Range Width of Input Values

As mentioned in the previous subsection, the proposed implementation of the SoftMax activation supports inputs in the range from −2.5 to 2.5. This range was chosen for two reasons—it is centered around zero, and its width is equal to five. According to Equation (3),

the result of SoftMax activation is a ratio between the results of the exponential function. The reduction of the input range leads to the exclusion of some values, introducing an error in the results. To make this effect less considerable, the range should be wide enough to exclude the least significant values only. Insofar as the result of SoftMax activation is probability distribution, the introduced error can be defined as the ratio between the exponential function at the maximum point and the exponential function at the minimum point:

$$s = \exp(x_l) / \exp(x_h) \tag{4}$$

where x_l and x_r—the lowest and highest points of the chosen range, respectively.

In this paper, the error is assumed equal to 0.0075, so the difference between the highest and the lowest possible inputs equals to:

$$x_h - x_l = -ln(0.0075) = 4.89 \approx 5 \tag{5}$$

To avoid the range of rapid growth of the exponential function, as well as processing small numbers in the negative subrange, it is proposed to select the approximation range endpoints equidistant from zero. Therefore, in this paper, the exponential function is approximated in the range from -2.5 to 2.5.

Obviously, if some of the input values exceed the upper endpoint, omitting those values will introduce a significant error to the result. As mentioned previously, a special shift can be applied to all input values to address this issue (Equation (2)). The shift value can be chosen as the difference between the upper endpoint of the approximation range and the maximum input value:

$$c(x_1, \ldots x_N) = x_h - max(x_1, \ldots x_N) = 2.5 - max(x_1, \ldots x_N) \tag{6}$$

After applying this shift, all the inputs will be limited to the upper endpoint of the approximation range.

4.5. Division of Output Values

The last problem mentioned is the division of the exponential function results by their total sum. As noted before, it can be accomplished using addition, subtraction, and shift operations [29,33,34]. However, the division is not widely used in other algorithms. Furthermore, even in the SoftMax algorithm, the division is performed at the end stage, introducing the least impact on the overall performance. Since each processing element of the RCE implements the whole set of operations, adding a new operation significantly influences the complexity of the entire RCE, even if this operation is rarely used. Consequently, it is important to implement processing elements only in simple, reusable operations. As a result, we suggest not supporting division in the RCE. This operation can be performed by some external computing block. Thus, the output of the proposed RCE implementation is a set of the exponential function results and their total sum.

It should be mentioned that in case of necessity, the division can be accomplished directly in the RCE. This can be achieved by approximating the hyperbolic function $f(x) = 1/x$ according to the example shown earlier or by introducing a bit shift operation in processing elements. Obviously, these solutions significantly increase the number of required processing elements or their complexity.

4.6. Complete Implementation

The complete implementation of the SoftMax activation in the RCE is presented in Figure 10. It processes an input vector of three values. The bottom part of the proposed design computes and applies a shift value to all the input signals to limit their range. The middle part consists of three blocks, performing the same exponential function approximation discussed above. The top part accumulates the exponential function results and computes the total sum. This implementation utilizes 215 processing elements, 177 of them

for the exponential function units. Obviously, the proposed implementation can be easily scaled for the required number of inputs.

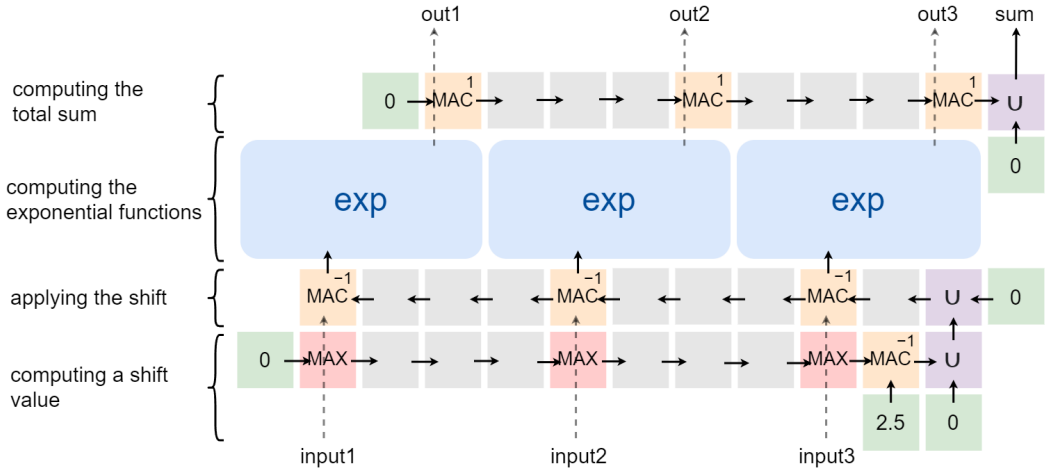

Figure 10. SoftMax activation in the RCE.

5. Experimental Results and Discussion

The characteristics of the proposed SoftMax implementation were evaluated through simulations on Field-Programmable Gate Arrays (FPGA). FPGAs are widely used in the research and development of state-of-the-art computing systems, and numerous useful software tools are available [6,7,23].

The models were implemented in the form of digital design modules using Verilog HDL and Quartus Prime 20.1.0 software. The modules operate on 16-bit fixed-point numbers (8 fractional bits). All the simulations were performed on a Cyclone V 5CGXFC9E7F35C8 device with disabled Digital Signal Processing (DSP) blocks. To avoid unintentional optimizations, the "synthesis keep" directive was applied.

We evaluated the resource utilization and performance of the proposed models. In this paper, resource utilization refers to the number of FPGA logic elements (LEs) required to implement the module. The Quartus software provides this value after a successful compilation. The model performance was measured as the largest processing delay. To evaluate this value, the Timing Analyzer tool (included in the Quartus software pack) was used. This tool analyzes all available paths between the inputs and the outputs, computes their signal propagation delay, and determines the longest one. Due to the large number of interconnections between processing elements, Quartus treats them as combinational loops. To tackle this problem, some unused connections were removed during the timing simulation.

The simulation results are presented in Table 2. As can be seen, the proposed models have high performance. The total processing delay of the SoftMax activation for three inputs is 43 ns. Exponential blocks are processed in 14 ns. Since these blocks work in parallel, their total delay is independent of the number of inputs. The processing delay of the other stages (applying the shift and computing the sum) linearly depends on the number of inputs.

At the same time, the models consume a large amount of FPGA resources. The entire implementation of the SoftMax activation for three inputs utilizes about 55,000 LEs, including 15,000 LEs per exponential block. An explanation of these results will be provided further.

The achieved results were compared with other implementations lacking the reconfiguration capability (Table 3).

Table 2. Simulation results of the SoftMax activation with three inputs.

Property	Value
Number of FPGA Logic Elements (LE) per exponential unit	15,069
Number of LEs for the entire implementation	54,718
Shift computing delay, ns	14.5
Shift applying delay, ns	3.5
Exponential block computing delay, ns	14
Outputs sum computing delay, ns	11
Total processing delay, ns	43

Table 3. Comparison of the obtained results with other studies.

Property	RCE	[29]	[30]	[33]
Total number of FPGA LEs	54,718	17,870	2229	646
Clock frequency, MHz	23.26	150	154	265
Throughput, Gbps	1.12	2.4	1.2	0.73

As is evident from Table 3, the proposed in this paper implementation provides the comparable throughput (about 1.12 Gbps) despite the lower clock frequency (about 23.3 MHz). In other words, it processes the commensurate amount of input data per unit of time, performing more calculations per clock cycle. However, the FPGA resource utilization is noticeably high (about 55,000 LEs compared to 18,000 of the counterpart). Thus, the proposed model consumes more space (in terms of a semiconductor area or the number of transistors) but does not provide a significant increase in performance.

The results presented in Tables 2 and 3 can be explained by the following reasons:

- Specialized computing devices are capable of achieving the best results for specific tasks due to particular optimization. Reconfigurable devices, by contrast, must sacrifice some properties to provide acceptable results for a wide variety of tasks. Our implementation of a reconfigurable accelerator sacrifices resource utilization (number of logic elements) to provide comparable performance.
- The proposed algorithms are spatially distributed. Different parts of the computing structure calculate different branches or stages of these algorithms. Spatial distribution leads to better performance (for some algorithms) at the expense of increased resource consumption;
- To provide the reconfiguration ability, each processing element supports the entire set of operations, although it performs only one of them at any given moment;
- The PEs must support simple reusable operations to be small and versatile. As a result, algorithms need much more PEs;
- The presented results were obtained for FPGAs that already have their own internal design. This design does not correspond to the proposed one. This incompatibility results in a violation of some geometric properties of the RCE and extra consumption of resources. We chose the FPGA to obtain the results that can be compared with other studies. In real cases, the best option to accomplish such accelerators is ASIC.

Consequently, the high resource utilization mentioned before is the price to obtain the desired reconfiguration capability. The listed analogs have a fixed implementation; therefore, they are capable of solving only one particular problem. In contrast, the suggested implementation works in a reconfigurable environment that can be customized to resolve a wide variety of other tasks. The importance of the reconfiguration ability for hardware accelerators is especially noticeable for hard-to-reach mobile and autonomous devices for four reasons:

- Such systems may require several neural networks with different architectures and activation functions. To implement them without the reconfiguration capability, it is essential to either employ a single universal device or multiple specialized devices (one per architecture) with activation computing units. Both approaches have

disadvantages: the universal devices typically have lower performance; the set of specialized devices affects the weight, reliability, and power consumption of the system. At the same time, reconfigurable accelerators are designed to support a wide variety of architectures and activation functions.

- Changes in the environment of the device may require alterations to the implemented algorithms. If the new algorithm relies on functions that are not introduced in the specialized accelerator, this algorithm cannot be used in this device. In the case of reconfigurable accelerators, the new functions can be built as a group of processing elements if their operations allow it. Numerous complex functions can be implemented by combining simple operations.
- An RCE-based accelerator only implements a particular algorithm when needed. The rest of the time, the same processing elements can be employed to perform other tasks. Such a reuse increases the efficiency of the reconfigurable devices.
- In the case of partial damage to the reconfigurable accelerator, there is a chance to restore key functions by redistributing computations to an undamaged area of the RCE. This requires a special control unit and an excessive number of processing elements.

In summary, the proposed implementation of the SoftMax activation, as well as the entire paradigm of RCE-based accelerators, provides significant benefits for specific devices.

6. Future Work

As is evident from the simulation results, the presented solution has shown high performance. At the same time, the proposed implementation requires many more FPGA logic elements. This can be explained as the expected cost of the reconfiguration ability. Nevertheless, it can cause problems in implementing complex neural networks. There are several promising approaches to decrease the logic utilization, including reduction of the bit depth of the processed signals, eliminating redundant operations from the processing elements, and simplifying the approximation of the exponential function. Another direction of further research is the implementation of the proposed models in an ASIC device. ASICs enable more detailed implementation of the desired circuit design, while FPGAs have limitations due to predefined structure.

7. Conclusions

Presently, machine-learning algorithms are becoming an important part of many modern intelligent systems. However, the variety and computational complexity of these algorithms is a challenge for low-power devices, such as smartphones, IoT sensors, satellites, and many others. A possible solution to this problem is reconfigurable hardware accelerators. Their key feature is the capability to support a wide variety of algorithms and dynamically alter them by an external signal. This paper explores the reconfigurable hardware accelerators based on reconfigurable computing environments. RCE is a class of computing systems that consists of numerous similar processing elements connected in a grid pattern. To use RCE efficiently, the required algorithms need to be adapted to its particularities.

This paper discusses the implementation of the SoftMax activation in the RCE. The implementation leverages several optimizations: using a piecewise linear approximation of the exponential function, modifying input values to limit their range, spatial distribution, and parallelization of computations. The proposed implementation of the piecewise linear approximation with variable step in the RCE is of great importance. It is a generic approach that can be easily reused to perform a wide variety of other functions in the RCE, greatly expanding its capabilities. The spatial distribution and parallel computing ensures high performance of the considered implementation of the SoftMax function.

The characteristics of the proposed model were evaluated by simulations on FPGA. The simulation results were compared to other implementations, lacking the reconfiguration capability. The proposed implementation of the SoftMax demonstrates high performance (about 43 ns for three inputs or 1.12 Gbps throughput) and high accuracy (the maximum

and average absolute errors are 0.002 and 0.00078, respectively). Only one of the three counterparts presented has a throughput of 2.4 Gbps, which noticeably outperforms the proposed implementation. The resource usage is also high—the model utilizes 54,718 FPGA logic elements, compared to 17,870 logic elements in the previously mentioned research.

The high performance of the developed models proves the efficiency of using the RCE-based accelerators as a replacement for rigid ones in tasks that require variability and high flexibility of the implemented algorithms. The noticeably high resource consumption of the proposed model is the cost of its spatial distribution and the flexibility of the RCE. This is the price for all the previously mentioned benefits of the reconfiguration ability. The proposed implementation of the SoftMax function allows RCE-based accelerators to tackle a wide variety of classification tasks. The RCE-based reconfigurable accelerators seem to be a promising direction for the development of AI hardware.

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: This study was supported by the Tomsk State University Development Programme (Priority-2030).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available in article [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].

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

References

1. 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.
2. Ghimire, D.; Kil, D.; Kim, S.-H. A Survey on Efficient Convolutional Neural Networks and Hardware Acceleration. *Electronics* **2022**, *11*, 945. [CrossRef]
3. 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]
4. 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]
5. 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]
6. 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]
7. 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]
8. 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.
9. Nabavinejad, S.M.; Reda, S.; Ebrahimi, M. Coordinated Batching and DVFS for DNN Inference on GPU Accelerators. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 2496–2508. [CrossRef]
10. 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.
11. 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.
12. Dhilleswararao, P.; Boppu, S.; Manikandan, M.S.; Cenkeramaddi, L.R. Efficient Hardware Architectures for Accelerating Deep Neural Networks: Survey. *IEEE Access* **2022**, *10*, 131788–131828. [CrossRef]

13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. Armeniakos, G.; Zervakis, G.; Soudris, D.; Henkel, J. Hardware Approximate Techniques for Deep Neural Network Accelerators: A Survey. *ACM Comput. Surv.* **2022**, *55*, 1–36. [CrossRef]
19. 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.
20. 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.
21. Shatravin, V.; Shashev, D.V. Designing high performance, power-efficient, reconfigurable compute structures for specialized applications. *J. Phys. Conf. Ser.* **2020**, *1611*, 012071. [CrossRef]
22. Shao, H.; Lu, J.; Lin, J.; Wang, Z. An FPGA-Based Reconfigurable Accelerator for Low-Bit DNN Training. In Proceedings of the 2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Tampa, FL, USA, 7–9 July 2021; pp. 254–259. [CrossRef]
23. 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.
24. 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.
25. 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. [CrossRef]
26. 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]
27. Kung, S.Y. *VLSI Array Processors*; Prentice Hall Information and System Sciences Series; Englewood Cliffs: Bergen, NJ, USA, 1988; 600p.
28. Evreinov, E.V. *Homogeneous Computing Systems, Structures and Environments*; Radio and Communication: Moscow, Russia, 1981; 208p.
29. Sun, Q.; Di, Z.; Lv, Z.; Song, F.; Xiang, Q.; Feng, Q.; Fan, Y.; Yu, X.; Wang, W. A High Speed SoftMax VLSI Architecture Based on Basic-Split. In Proceedings of the 2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), Qingdao, China, 31 October 2018–3 November 2018; pp. 1–3. [CrossRef]
30. Gao, Y.; Liu, W.; Lombardi, F. Design and Implementation of an Approximate Softmax Layer for Deep Neural Networks. In Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS) , Seville, Spain, 12–14 October 2020; pp. 1–5. [CrossRef]
31. Kouretas, I.; Paliouras, V. Hardware Implementation of a Softmax-Like Function for Deep Learning. *Technologies* **2020**, *8*, 46. [CrossRef]
32. Yang, X.; Su, T. EFA-Trans: An Efficient and Flexible Acceleration Architecture for Transformers. *Electronics* **2022**, *11*, 3550. [CrossRef]
33. Hussain, M.A.; Tsai, T.-H. An Efficient and Fast Softmax Hardware Architecture (EFSHA) for Deep Neural Networks. In Proceedings of the 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), Washington, DC, USA, 6–9 June 2021; pp. 1–4. [CrossRef]
34. Li, T.; Zhang, F.; Xie, G.; Fan, X.; Gao, Y.; Sun, M. A high speed reconfigurable architecture for softmax and GELU in vision transformer. *Electron. Lett.* **2023**, *59*, 5. [CrossRef]
35. 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.
36. 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]