electronics

MDPI

Article

FPGA-Based BNN Architecture in Time Domain with Low Storage and Power Consumption

Longlong Zhang, Xuebin Tang, Xiang Hu, Tong Zhou * and Yuanxi Peng *

College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China; zll@nudt.edu.cn (L.Z.); xbtang@nudt.edu.cn (X.T.); huxiang@nudt.edu.cn (X.H.)
* Correspondence: zhoutong09@nudt.edu.cn (T.Z.); pyx@nudt.edu.cn (Y.P.)

Abstract: With the increasing demand for convolutional neural networks (CNNs) in many edge computing scenarios and resource-limited settings, researchers have made efforts to apply lightweight neural networks on hardware platforms. While binarized neural networks (BNNs) perform excellently in such tasks, many implementations still face challenges such as an imbalance between accuracy and computational complexity, as well as the requirement for low power and storage consumption. This paper first proposes a novel binary convolution structure based on the time domain to reduce resource and power consumption for the convolution process. Furthermore, through the joint design of binary convolution, batch normalization, and activation function in the time domain, we propose a full-BNN model and hardware architecture (Model I), which keeps the values of all intermediate results as binary (1 bit) to reduce storage requirements by 75%. At the same time, we propose a mixed-precision BNN structure (model II) based on the sensitivity of different layers of the network to the calculation accuracy; that is, the layer sensitive to the classification result uses fixed-point data, and the other layers use binary data in the time domain. This can achieve a balance between accuracy and computing resources. Lastly, we take the MNIST dataset as an example to test the above two models on the field-programmable gate array (FPGA) platform. The results show that the two models can be used as neural network acceleration units with low storage requirements and low power consumption for classification tasks under the condition that the accuracy decline is small. The joint design method in the time domain may further inspire other computing architectures. In addition, the design of Model II has certain reference significance for the design of more complex classification tasks.

Keywords: BNN; time domain; mixed-precision; low storage; low power consumption; FPGAs

Citation: Zhang, L.; Tang, X.; Hu, X.; Zhou, T.; Peng, Y. FPGA-Based BNN Architecture in Time Domain with Low Storage and Power Consumption. *Electronics* **2022**, *11*, 1421. https://doi.org/10.3390/electronics11091421

Academic Editor: Akash Kumar

Received: 1 April 2022
Accepted: 27 April 2022
Published: 28 April 2022

1. Introduction

In recent years, convolutional neural networks (CNNs) have been widely used in image classification, speech recognition, face recognition, autonomous driving, and other areas [1–3]. As the ability of CNNs increases, the number of parameters and the computational complexity increase, limiting the application of neural networks. On edge computing devices such as mobile devices and embedded systems [4,5], there is an urgent need for lightweight neural networks with low storage and low power consumption to facilitate rapid deployment and efficient implementation.

To better realize intelligent computing acceleration on hardware devices with limited resources, it is necessary to compress and optimize neural networks. Commonly used model compression methods include quantization, pruning, and sparsity. The way to achieve the ultimate quantification method is the binary neural network (BNN). At present, there are many related studies on BNNs. For example, BinaryNet [6] first proposed the binary concept with neural networks which used +1 and −1 to represent input values and weights. They used bit-wise operations instead of multiplication operations. XNOR-

Net [7] used a scaling factor on the binarization parameters to approximate floating-point parameters, which further reduced the quantization error. Subsequently, there have been many similar related studies based on CPU or GPU platforms [8–10]. For instance, BinaryDenseNet [11] and Bi-Real Net [12] achieved certain results on the ImageNet dataset. Most of the above research work on BNN called library functions in neural network frameworks (such as TensorFlow, Pytorch) to complete training and testing.

Neural network accelerators based on reconfigurable devices can achieve a compromise between general-purpose processor platforms and ASIC-based dedicated processors. In particular, FPGA's flexible programmability and high-density logic resources provide parallel memory access capabilities and high on-chip memory access bandwidth, which is very suitable for the development of neural network accelerators. There have been many research results on the FPGA platform. For example, Zhao et al. first used FPGA to implement BNN [13]. Xilinx Research Labs (Dublin, Ireland)designed an FPGA accelerator framework (Finn [14]), which uses HLS to flexibly allocate resources and modules, with each layer having its computing module. On this basis, they launched the second-generation Finn framework (Finn-r [15]), which can support design space exploration and automatically create a customized inference engine on FPGA. The work on Fp-bnn [16] modified the original multilevel addition tree structure to a special compressed tree structure to save the consumption of logic resources. Cheng et al. [17] made full use of the similarity between the input value and the weight after the binary operation to reduce the number of MAC operations in the inference process. Han et al. [18] proposed a scalable full-pipeline BNN architecture whose goal is to maximize throughput in large FPGAs and maintain energy and resource efficiency. A multiscale BNN topology with few nonarithmetic operations was proposed in [19]. In addition, Simons et al. [20] applied binary neural network layers to visual detection tasks.

The above algorithms and structures were based on neural networks implemented in the digital domain, which consumes more energy and area than the analog domain. In the analog domain, there have been studies using voltage [21,22] and frequency [23,24] to represent the size of the data. However, affected by noise, the data range and precision that the voltage or frequency can represent is limited, and it is not easy to control. At the same time, conversion modules such as analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) may need to be used.

In the research process of BNN, the method of expressing data size by pulse width has attracted more and more attention. For example, Miyashita et al. [25] used time as an analog signal to test the second layer of BNN. Another study [26] used a bidirectional memory delay line unit to complete the multiply–accumulate operation. They also expanded the bit width to multi-bits by encoding [27]. The abovementioned time domain-based designs have the limitation of nonidealities of analog circuits and may require delay calibration units for error correction.

However, the current FPGA-based BNN structure still has several challenges. Firstly, XNOR logic has been used in the BNN to replace the multiplier, which greatly reduces the computational complexity and resource consumption, but there still is room for optimization. Computing units such as adder, subtractor, and pop-count operations need to be improved. Secondly, although BNN has reduced the value of the weight and input activation to one bit, the intermediate accumulation results after convolution are still multi-bits, which need to be processed to further reduce storage requirements. Thirdly, research results show that the key to the accuracy of neural networks lies in the calculation precision of certain important layers. For key layers, higher data precision is required to ensure less information loss. Therefore, it is necessary to consider how to efficiently support the diversified requirements for data precision of different layers or different parts in the same layer. Lastly, the dot multiplication of BNN becomes a simple XNOR operation, and its hardware implementation only requires a small number of LUTs. As the network becomes larger, this may lead to excessive LUT resource usage. However, almost all DSPs

embedded in the accelerator are in an idle state. How to balance the utilization of logic resources and DSP resources on the hardware is also an issue worthy of discussion.

Hence, our aim was to explore the BNN inference process based on the FPGA platform and propose two models to meet different application requirements. The major contributions of this work are as follows:

- By constructing a time-domain processing elements (PE) prototype, the proposed time domain-based binary convolutional structure effectively avoids the traditional addition and subtraction operations, which saves resource consumption and reduces power consumption, while reducing the amount of storage required in the accumulation process.
- The proposed time-domain data path optimization scheme and multidimensional array structure can achieve parallelism in time and space dimensions to maximize computing efficiency.
- Through the joint design of binary convolution, batch normalization, and activation function, all intermediate results of the BNN based on the time domain use binary values instead of integer values, which enables the parameters and intermediate results to be stored on chip. This avoids the overhead of on-chip and off-chip memory access for intermediate data, thereby reducing the total computational cost.
- Through the test of the impact of data precision on classification accuracy, we summarize the sensitivity of different layers to accuracy and consider the mixed-precision calculation structure to achieve full hardware utilization and ensure classification accuracy.
- We develop the full-BNN accelerating system based on the DSP-FPGA platform. Compared to some state-of-the-art accelerators, the proposed system has the characteristics of both low storage and low power consumption.

The remaining sections are organized as follows: in Section 2, the principle and hardware implementation of BNN is introduced. The PE arrays based on the time domain and the details of parallel optimization from two dimensions are discussed in Section 3. Section 4 introduces the hardware acceleration design of two BNN models based on the time domain. The analysis results and comparisons with previous work are shown in Section 5. Section 6 concludes the work.

2. Background

Usually, in CNN, we need to calculate the multiply–accumulate operation of multiple weights and input activations to get an output feature value. As shown in Equation (1), w_i, x_i, and y_{CNN} respectively represent the weight, input activation, and output feature value in the convolution operation.

$$y_{CNN} = \sum_{i=0}^{k-1} w_i x_i \quad (1)$$

In BNN, since the values of weight and input activation become +1 and −1, the multiplication operation of convolution becomes a bit-wise operation, which can be realized by a simple XNOR circuit [6]. We know that there are only high-level and low-level signals in the circuit. Thus, it can be completed by simply performing data mapping. For example, "+1" is represented by a high-level signal, and "−1" is represented by a low-level signal.

The cumulative result of the binary convolution process is generally integer data. We need to binarize the result of the previous layer before it can be input to the next convolution layer as the input activation, i.e., the Bin function, as shown in Equation (2).

$$y_{BNN} = Bin(\sum_{i=0}^{k-1} w_i x_i) \quad (2)$$

As shown in Equation (3), the Bin function is bounded by zero. If it is greater than or equal to 0, the output is 1; otherwise, the output is −1.

$$Bin(x) = Binarize(x) = \begin{cases} +1, if\ x \geq 0 \\ -1, if\ x < 0 \end{cases}. \tag{3}$$

In the case of binarization, the operation of batch normalization on the hardware becomes easier. The batch normalization used in the training process is shown in Equation (4), where γ and β respectively represent the scale parameter and the shift parameter. By default, γ = 1, and β = 0. These two parameters give the ability to learn during the batch normalization process.

$$BatchNorm(x) = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \xi}} \times \gamma + \beta \tag{4}$$

After combining Equations (3) and (4) (ignoring the value of ε), we can proceed as follows [25]:

$$\begin{aligned} &Bin(BatchNorm(x)) \\ &= Binarize\left(\frac{\gamma}{\sqrt{\sigma_B^2}} \left(x - \mu_B + \frac{\sqrt{\sigma_B^2}}{\gamma} \times \beta \right) \right) \\ &= Binarize\left(x - \mu_B + \frac{\sqrt{\sigma_B^2}}{\gamma} \times \beta \right) \\ &= Binarize\left(x + offset \right) \end{aligned}. \tag{5}$$

It can be seen from Equation (5) that, when we use the scale and the shift parameters (setting $affine = True$), then $offset = \sqrt{\sigma^2} \times \beta \div \gamma - \mu_B$. On the contrary, when we set $\gamma = 1$ and $\beta = 0$ ($affine = False$), the value of the offset is $(-\mu_B)$.

Since the model parameters of the trained network are determined, the offset value of each layer can be calculated in advance. For hardware implementation, this method turns the original multiplication and division operations in the batch normalization process into simpler addition operations.

It should be noted that, when there is a batch normalization operation, the value of bias is so small that it can be considered to not affect the final accuracy. Therefore, we usually ignore the bias when doing hardware inference [13].

Data can be represented not only in voltage, current, and frequency but also in variable pulse widths. As shown in Figure 1, the main idea of the time-domain model is to cascade a series of delay elements, where the delay value of each unit depends on the result of multiplying the weights and activations [28].

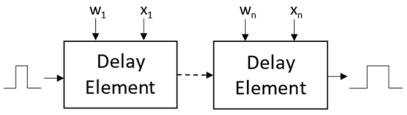

Figure 1. Illustration of time-domain model [28].

We can see that a pulse is used as the input of the first delay element, and the last output pulse represents the result of the MAC operation. The main advantage of this approach is that the addition operation is free as the pulse propagates from one delay element to another.

3. The Architecture of Binary Convolution Based on Time Domain

In this section, we first construct a PE array to complete the binary convolution operation of the convolutional layer based on the time domain. Then, we introduce the details of the single PE and detector structure. Lastly, we perform parallel optimization in the spatial and temporal dimensions.

3.1. The Design of PE Arrays

First, we construct a row array as shown in Figure 2. This structure is mainly composed of several time-domain PEs, XNOR logic units, and detector parts, which can complete binary convolution, batch normalization, and the activation function.

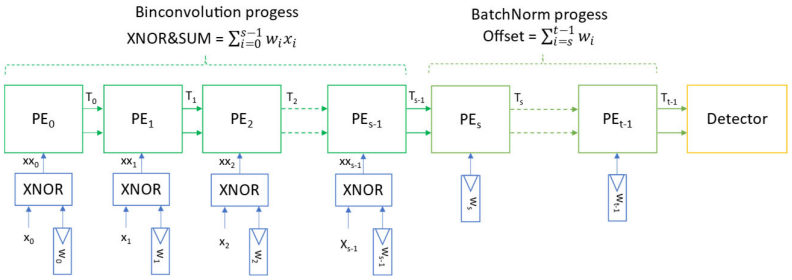

Figure 2. PE array based on time domain.

The left part completes the binary convolution process. Here, we use the weight reuse mode, where the weight can be stored in the on-chip register before the calculation starts. After the computing begins, the output feature values of the upper layer are input in order, and XNOR operations are performed with the weights. The delay difference is transmitted from left to right to complete the "multiply–accumulate" function, i.e., $XNOR\&SUM = \sum_{i=0}^{s-1} w_i x_i$. The middle part is the process of batch normalization, which improves the batch normalization operation to avoid the participation of adders and subtractors. The process of batch normalization is described in detail in Section 4.1.2. After several PEs, the delayed signal finally reaches the detector.

3.2. PE Structure and Detector Based on Time-Domain

Inspired by the work of Miyashita et al. [25], we designed a time-domain prototype based on FPGA, which can convert multiply–accumulate operations from the digital domain into the time domain. The prototype of PE is shown in Figure 3a, where V_{A-in}, V_{B-in}, V_{A-out}, and V_{B-out} represent the input signals and output signals of PE, respectively. Except for the input signal of the first PE which is provided by an external signal, the input signals of other PEs are all transmitted by the output signal of the previous PE. The output signal of the last PE serves as the basis for the detection signal of the final result. The value of xx_s entering each PE is the result of the XNOR operation of the weight w_s and the input activation x_s. The truth table of the XNOR logic is shown in Figure 3b.

We use pulse width to represent the value of data, and we use delay difference to represent data of different values. T_{in} and T_{out} represent the input delay difference and the output delay difference, respectively, and the initial input delay difference is zero.

For a single PE, we define the delay difference as follows: when the rising edge of V_{A-in} or V_{B-in} arrives, as shown in Figure 3c, if $w_s = +1$, $x_s = +1$ or $w_s = -1$, $x_s = -1$, then V_{A-out} is one cycle later than V_{A-in}, but V_{B-out} is two cycles later than V_{B-in}. We can get the result $T_{out} = T_{in} + 1$ where the "1" represents a clock pulse. Correspondingly, as shown in Figure 3d, if $w_s = +1$, $x_s = -1$ or $w_s = -1$, $x_s = +1$, then V_{A-out} is two cycles later than V_{A-in}, but V_{B-out} is one cycle later than V_{B-in}. Then, we can get the result $T_{out} = T_{in} - 1$. After this prototype design, the input and output signals of the PE carry the data information needed for the calculation.

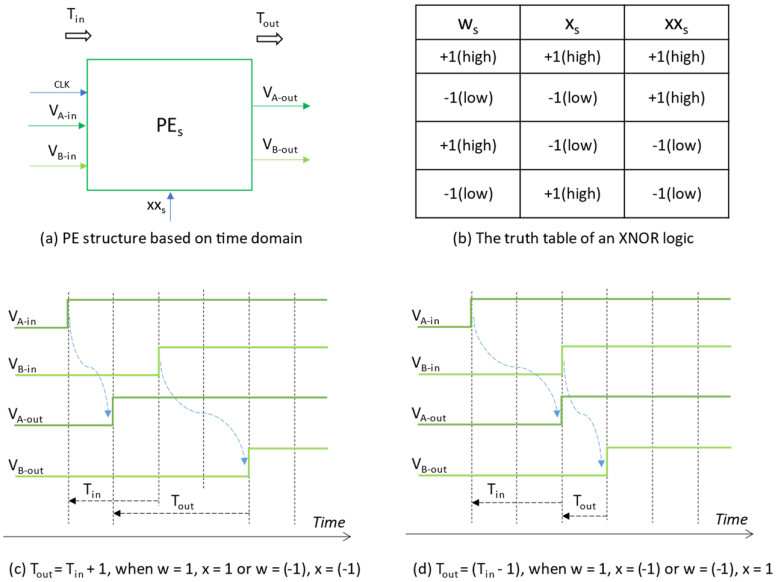

Figure 3. Time-domain PE and timing relationship.

At the end of the array, we set up a detector to pick up the result. As shown in Figure 4, we only need to compare the rising edge arrival sequence of V_{A-in} and V_{B-in} to the detector, regardless of the delay difference between V_{A-in} and V_{B-in}. The specific method is that, when the rising edge of V_{B-in} arrives, we detect the signal value of V_{A-in}. If V_{A-in} is in a high-level state at this time, then D_{out} is 1 (representing "+1"). If the value of V_{A-in} is in a low-level state, D_{out} is 0 (representing "−1"). We use simple logic to implement this detection function. In addition, D_{valid} is the D_{out} data valid signal.

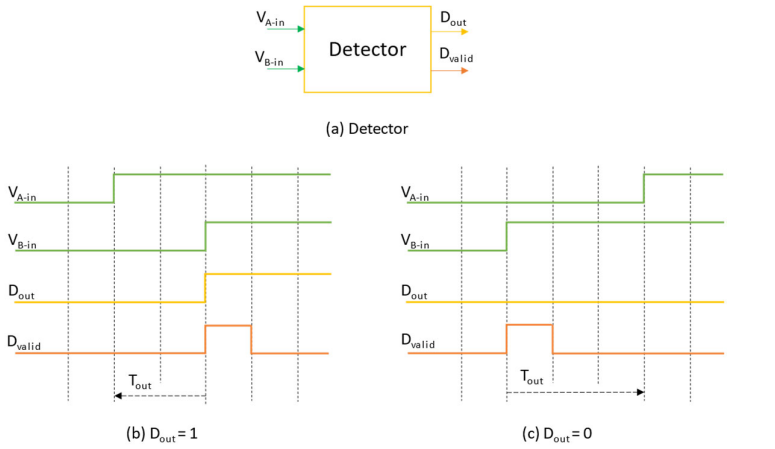

Figure 4. Results detection structure based on time domain.

3.3. Parallel Processing in Spatial Dimension

Furthermore, we construct a two-dimensional PE array processing structure based on a single array, as shown in Figure 5. On the one hand, we can make the input value x_i flow from the bottom to the top of the PE chain by fixing the input value, to achieve the purpose of data multiplexing for the input value x_i. On the other hand, since the number of convolution kernels used by neural networks is generally dozens, we adopt this spatially parallel method to operate multiple convolution kernels at the same time. The degree of parallelism of a certain layer needs to be comprehensively determined according to the size of the convolutional layer and the logical resources of the platform used.

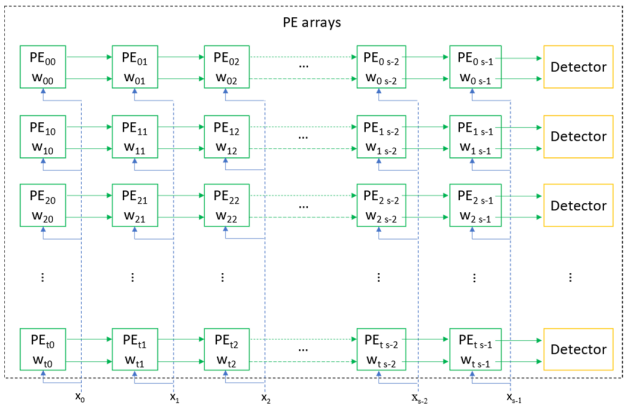

Figure 5. Convolutional layer parallel PE arrays based on time domain.

The above binary convolution structure based on the time domain can replace the addition tree or compressed addition tree structure in the previous work, further reducing resource overhead and storage requirements.

3.4. Data Path Optimization in the Time Dimension

In addition to considering parallel processing in spatial dimension as described in Figure 5, we also optimize the time-domain model in parallel in the temporal dimension to achieve computational overlap and increase data throughput.

For arrays based on the time domain, we need to consider possible data overflow. If the time width is not enough, when the time information flows to the detector, the correct result may not be detected. If the time width is too large, the overall calculation efficiency will be affected.

We only need to consider the case of a row of PE arrays. What we are discussing here is the V_{A-in} and V_{B-in} signals that enter the first PE. These are also the signals we need to control, and the related signals of the subsequent PE are automatically transmitted from the front.

We define the hold time as the number of clock cycles during which the V_{A-in} or V_{B-in} signal is in a high-level state. Similarly, we define the interval time as the number of clock cycles when the V_{A-in} or V_{B-in} signal is in a low-level state. For the subsequent PE, if V_{A-in} is reached earlier than V_{B-in}, then the high-level signal of V_{A-in} is maintained until the rising edge of V_{B-in} arrives. That is, the hold time is greater than their delay difference. If V_{B-in} arrives earlier than V_{A-in}, then the low-level signal of V_{A-in} is maintained until the rising edge of V_{B-in} arrives. That is, the interval time must be greater than the delay difference.

If we consider the extreme case, i.e., when the symbols of x_i and w_i are all equal to "+1", the maximum hold time and interval time should be set to ensure that there is no error in the process. In fact, we can also reasonably set the hold time and interval time according to the distribution characteristics of the dataset and the number of PEs, as shown in Figure 6. The ultimate goal is to find an optimal combination point in the temporal dimension under the premise of ensuring correctness. The hold time and interval time are set under the specific situation in Section 5.3.

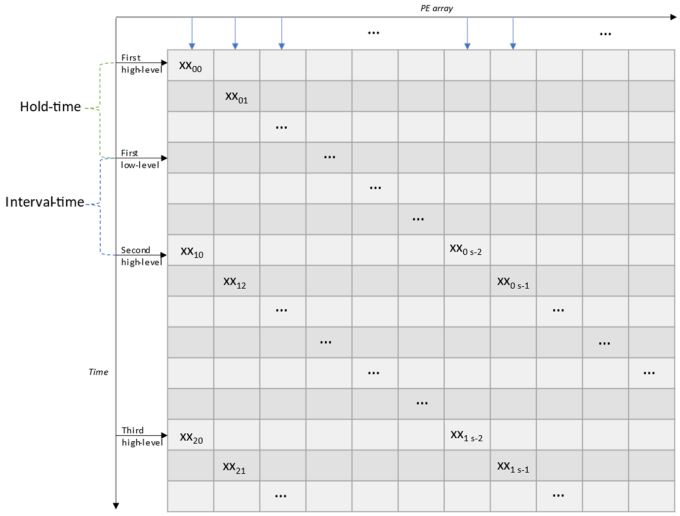

Figure 6. Data path optimization in the time dimension.

4. BNN Accelerator Design

Each layer of the BNN has its characteristics in terms of sensitivity to accuracy and calculation requirements, which need to be considered comprehensively. In this section, on the basis of the binary convolution structure in the time domain, we complete the design and implementation of the full-BNN and the mixed-precision BNN.

4.1. Design of Full-BNN Accelerator

On the basis of the idea of full binarization, we first introduce the full binary model design based on time-domain signal processing. The meaning of full binarization here is that, except for the last fully connected (FC) layer for classification, the entire process of the remaining layers is fully binarized, including weights, input values, and all intermediate results. Then, we introduce the joint design of convolution, batch normalization, and activation function. Lastly, we perform a binary design on the middle FC layer (not the last).

4.1.1. Full-BNN Model

The overall design of the full-BNN model is shown in Figure 7, which is roughly divided into three parts: the first layer, the middle layer, and the last layer.

In the data preprocessing stage, we batch normalize the input image data and quantize them to one bit, and then input them to the first layer of the binary convolution module.

Both the first layer and the middle layer use one bit. The weights and input activations are originally in the digital domain. When we load them into the time-domain PE array, they undergo the process of digital-to-time conversion (DTC). It is the detector shown in Figure 4 that completes the time-to-digital conversion (TDC). We complete the pooling operation in the digital domain, and then enter the time domain through the DTC on the next layer, executing it in a loop.

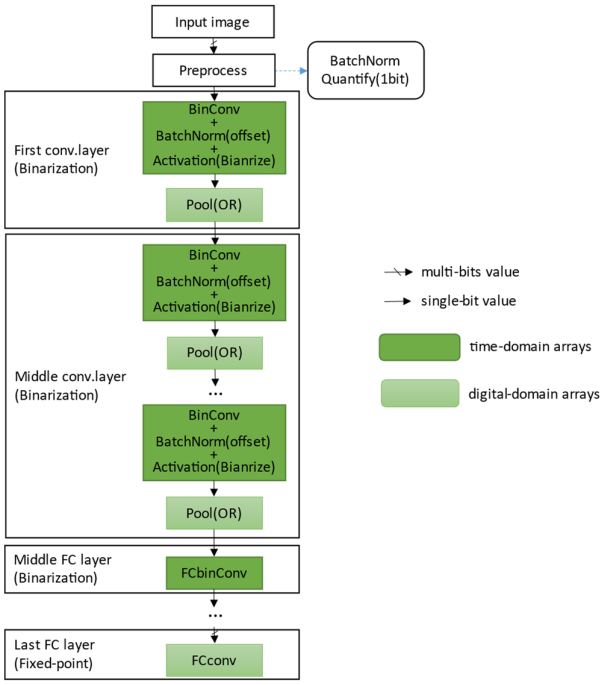

Figure 7. Full-BNN model based on time domain.

We can also use the binary mode for the operation of the middle FC layer. Unlike the previous convolutional layer, it only needs one PE chain. The specific design is described in Section 4.1.3.

4.1.2. Joint Design of Convolution, Batch Normalization, and Activation Function

As mentioned earlier, when the input activation of the last layer needs to be binarized, batch normalization can be converted into simple addition operations. Furthermore, we incorporate batch normalization, activation function, and binary convolution into the time domain to complete them together, as shown in Figure 8. This method not only avoids the addition operations in the batch normalization process but also completes the binarization of the input activation of the next convolution layer. Note that this joint design does not affect the pooling layer.

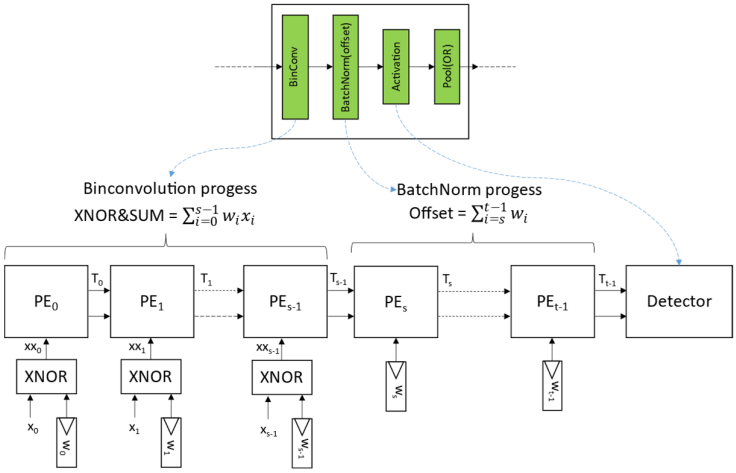

Figure 8. The process of joint design.

As shown in Figure 8, after the binary convolution array, we set up some time-domain PEs to complete the batch normalization process. Unlike the previous method, the input value of these PEs is directly preconfigured with the corresponding w_i value through the offset value of each channel, i.e., $Offset = \sum_{i=s}^{t-1} w_i$, where $w_i \in \{-1, +1\}$.

As shown in Figure 9a,b, the intermediate results of traditional CNN are real values throughout the process. The input activation of convolution in BinaryNet [6] is binary, while the outputs of convolution are integer values. Thus, traditional BNN deals with integer values during the pooling operation and batch normalization operation.

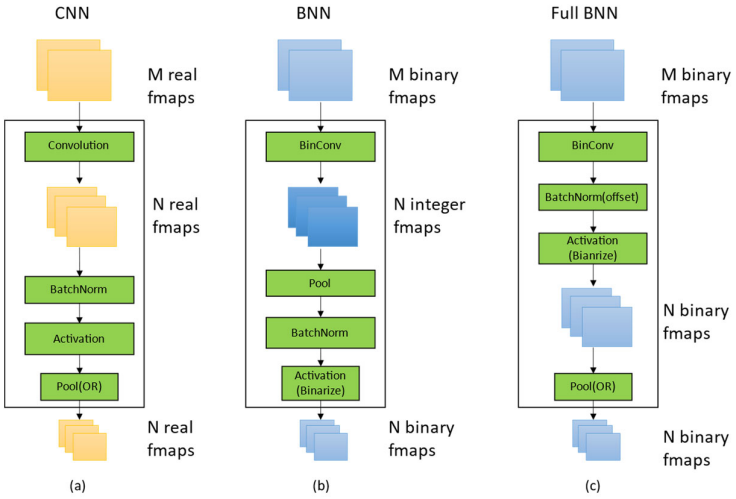

Figure 9. Comparison of three models: (**a**) traditional CNN; (**b**) BinaryNet [6]; (**c**) our full BNN.

As shown in Figure 9c, the intermediate results after binary convolution, batch normalization, and activation function are 1 bit output feature maps. It can be seen that there are no intermediate results of real or integer values within layers and between each layer, which largely reduces the storage space required for intermediate results and improves computational efficiency.

In addition, compared to the pooling operation of integer values in Figure 9b, the input of the pooling operation shown in Figure 9c is binary; hence, so max pooling can be determined by OR logic and efficient realization [14], i.e., $max\{x_1, x_2, \ldots x_8, x_9\} = OR\{x_1, x_2, \ldots x_8, x_9\}$, where $x_i \in \{+1, -1\}$. Similarly, min pooling can be easily implemented by AND logic.

4.1.3. Binarization Design of the Middle FC Layer

The output feature map of the convolutional layer becomes a vector after being tiled. Thus, the convolution operation of the middle FC layer is essentially a matrix–vector multiplication operation, as shown in Figure 10a. If there are many continuous feature maps from the previous convolution layer, the convolution of the middle FC layer can also be regarded as a matrix–matrix multiplication operation, as shown in Figure 10b.

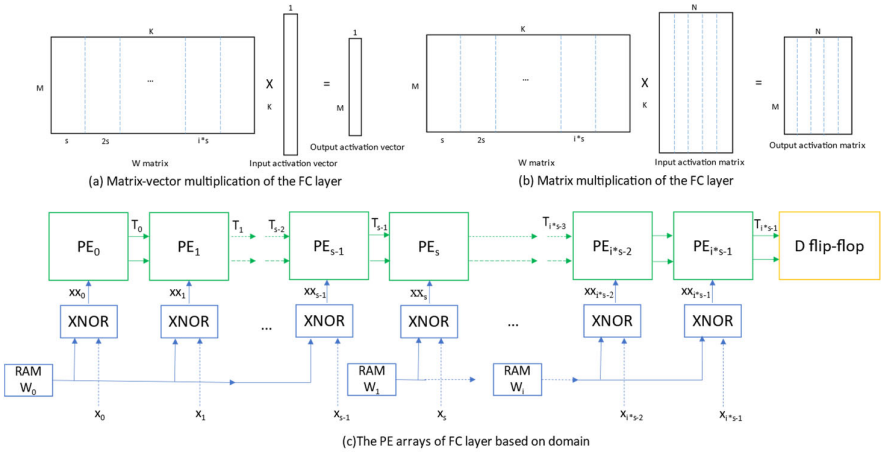

Figure 10. The middle FC layer structure based on time domain.

Of course, although the middle FC layer is also a convolution operation, there are some differences from the convolution layer; that is, the weight matrix of the middle FC layer convolution operation is relatively large, and the number of input eigenvalues is relatively small. Combining this feature, we design a binary convolution structure for the middle FC layer based on the time domain, as shown in Figure 10.

Although the weight matrix is relatively large, this part of the value is fixed, which we can obtain in advance. Thence, we put the weight in the on-chip RAM in advance so that this part of the data can be reused.

We also consider the use of multiple RAMs. Each RAM stores several columns of data for the weight matrix. The data are distributed according to the address in the same RAM, and different RAMs give enable signals in order, which can make full use of storage resources and achieve overlapping calculations in the temporal dimension.

4.2. Design of Mixed-Precision BNN Accelerator

Under the trend of low precision, the appropriate precision of each layer of the CNN may be different. Therefore, it is necessary to consider mixed precision to achieve a balance between accuracy and resource consumption.

4.2.1. Motivation for the Design of Mixed-Precision Model

Studies have shown that the combination of fixed-point and binary data is feasible, and, if implemented properly, it can bring substantial savings and improve processing efficiency. Wei et al. [29] divided the convolution into two groups: the more important convolution kernel that is not binarized and the other group that is binarized. Although the discussion was about the training process, this also has reference significance for the inference process. Mocerino et al. [30] trained two networks on a CPU, including a binary network and an 8 bit quantization network. If the accuracy of the binary network is not enough, then the 8 bit quantization network is revisited to complete computing. This constructs an adaptive scheme that realizes dynamic cooperation between binary networks and fixed-point networks. In addition, Fasfous et al. [31] proposed a configurable PE that uses the existing DSP48 block on the FPGA to calculate binary convolution and fixed-point arithmetic operations, such as scaling, shifting, batch norm, and nonbinary layers.

After analyzing and testing the dataset and network model, we found that the CNN has common characteristics. Firstly, the first layer of CNN often has fewer input channels, the amount of calculation is the least compared to other layers, and the overall complexity is relatively low. Secondly, whether it is an XNOR or "multiply and accumulate" operation in the time domain, it needs to be implemented with LUTs, and there are other modules that also need to use LUT resources, such as FIFO and control logic. There may be a

situation where logic resources are in short supply, and the DSP resources in the accelerator are in an idle state. Lastly, the first layer has a greater impact on the accuracy of the neural network. In more complex classification tasks, the importance of the data precision of the first layer is more obvious. Therefore, we chose to quantize the input activation and weight of the first layer to a fixed point (8 bit).

This design has two advantages. On the one hand, it can make full use of on-chip resources. The fixed-point multiplier can be mainly completed by DSPs embedded in the FPGA, and the binary convolution part of the middle layer uses LUT resources. In this way, in the case of the same logic resources, the middle layer part can appropriately expand the degree of parallelism. On the other hand, it can ensure that more feature information is extracted at the beginning. This reduces the error in the subsequent convolution process to alleviate the serious deterioration of accuracy caused by binarization.

4.2.2. Mixed-Precision BNN Model

The CNNs need to use different precisions on different data types. For inference, the data types that need to be considered include weights, input activations, and partial sums. In different networks, the accuracy of these three data types has different effects on overall accuracy.

The mixed-precision model we used is shown in Figure 11, where the first layer of convolution uses an 8 bit fixed-point mode. This can be designed as a separate calculation structure. In the data preprocessing stage, input activation and weighting need to be converted to fixed-point data. Excluding the first convolutional layer, the middle layers and FC layers are the same as those described in Section 4.1.1.

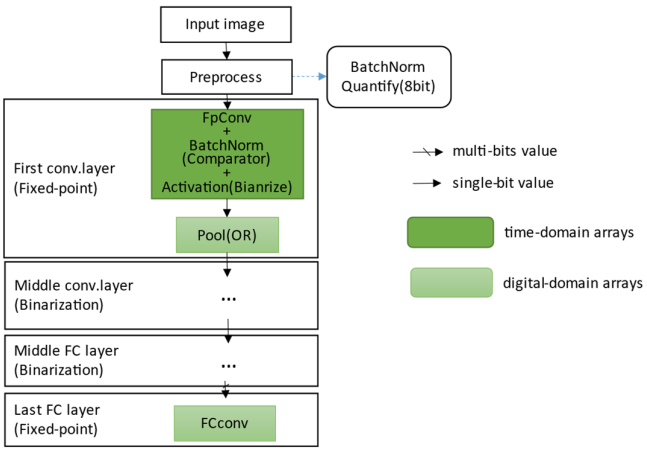

Figure 11. Mixed-precision model.

Note that the result of the first layer of convolution is fixed-point data. Here, we also perform joint design in conjunction with the binarization operation of the next convolutional layer.

Similar to the concept of a threshold [14,32], we use a comparator to implement the batch normalization and activation function of the first layer. One value of the comparator can be determined, i.e., the offset, while the other value is the output feature map of the first convolution layer. Thence, when we complete the batch normalization and activation function, we also complete the binarization of the input activation for the second convolution layer.

Therefore, this mixed-precision model only needs to use fixed-point values in the first convolutional layer and uses binary values for the subsequent pooling layer. Batch normalization only requires the use of comparators, without the use of addition operations.

4.2.3. Fixed-Point Computing Array

As shown in Figure 12, we took the MNIST dataset [33] as an example to design a fixed-point calculation structure based on the systolic array. Similar design ideas can be adopted for different datasets.

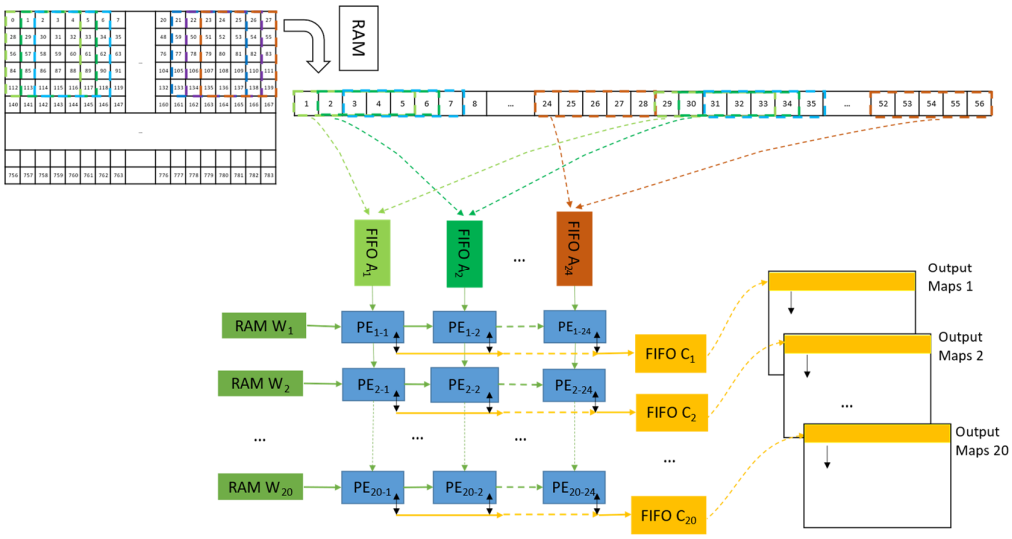

Figure 12. The first layer of the convolution computing structure.

According to the number and size of convolution kernels in the first layer, we construct a 20×24 systolic array. Our structure generally consists of a set of PE arrays for convolution operations, FIFO A for buffering input feature maps, RAM W for storing weights, and FIFO C for storing output feature maps.

When the sliding window slides on the input feature map, the address of the data is not continuous. This memory access method greatly reduces the utilization of memory bandwidth. To ensure the continuity of the memory access address, we load multiple feature maps into the on-chip RAM in sequence at one time, and then we fetch the data according to the address in the preconfigured address generator and put it into the corresponding FIFO A. This is equivalent to tiling the 28×28 picture data on the chip, but it does not occupy the memory access bandwidth.

We choose the multiply–adder IP core to complete the fixed-point multiplication operation because it can complete the operation of signed data, and an IP core uses only one DSP. Note that the original data involved in the calculation generally contain decimals, such as one sign bit, two integer bits, and five decimal bits. We input the original data as fixed-point data into the calculation module (equivalent to shifting the decimal point to the right by five digits). Then, we intercept the effective number of digits in the final result data.

The weight of the trained model is deployed to the corresponding on-chip RAM in advance. When the data of the input feature map are prestored to a certain amount, we read FIFO A_1 and RAM W_1 at the same time to start the calculation of the first PE. After reading FIFO A_1, we read FIFO A_2 − FIFO A_{24} in turn. Similarly, after reading RAM W_1, we read RAM W_2 − RAM W_{20} (each RAM W is read in a loop every 25 clock cycles). Accordingly, the entire calculation array starts calculating in turn.

In the calculation process, the input value x flows between the PEs in the same column from top to bottom, and the weight w flows between the PEs in the same row from left to right. The value of the intermediate result C is equal to the multiplication and accumulation result of the previous clock cycle and participates in the calculation of the current clock cycle. The value of C is cleared every 25 clock cycles. The system detects the

result valid signal every 25 clock cycles and stores the valid result in FIFO C. After completing a round of calculations, each FIFO C corresponds to an output characteristic map.

Considering the joint design with batch normalization, it is necessary to add a comparator at the output of the systolic array to complete the batch normalization operation. We store the result in RAM so that the next layer can fetch the number according to the address. Although the systolic array has a relatively high demand for memory access bandwidth, in general, except for the first load of image data, the time for subsequent loading of data can be hidden between the overall calculations.

Another advantage of this mixed-precision scheme is that the calculation structure of the final FC layer can reuse the calculation structure of the fixed-point convolution of the first layer. The specific method is described in Section 5.1.

4.3. The Architecture of Accelerating System

For different data sources, we need to design the network structure and configure the model parameters to achieve the best performance. At the same time, to efficiently map the neural network to the hardware for execution, it is necessary to coordinate optimization in both software and hardware.

Figure 13 presents the proposed overall system structure, which can be dynamically configured and implemented on the FPGA according to the network structure. The whole system consists of the DSP part (the host processor) and the FPGA part (the coprocessor). The FPGA part mainly includes controller unit, BNN accelerating unit, SRIO unit, and MCDMA unit used for data transmission.

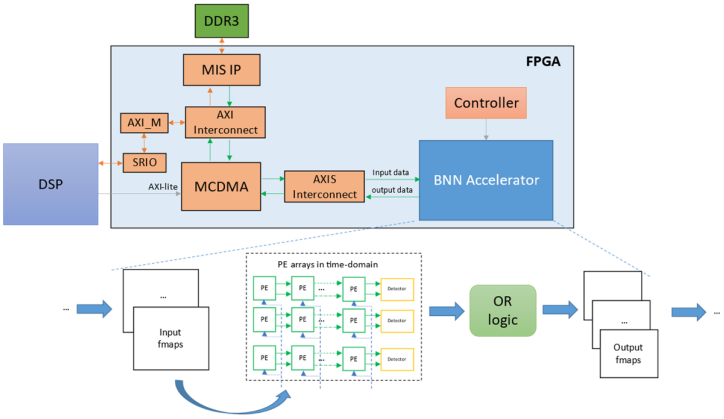

Figure 13. The overall architecture of the BNN accelerator system.

5. Experimental Results and Discussions

We used Verilog hardware description language (VHDL) to program the BNN acceleration unit, used Vivado 2019.2 for simulation and testing, and deployed the optimized two models on the DSP + Xilinx 325T development board.

5.1. Network Structure Settings and Experimental Results

For different data sources, we need to design the network structure and configure the model parameters to achieve the best performance. At the same time, to efficiently map the neural network to the hardware for execution, it is necessary to coordinate optimization in both software and hardware.

Table 1 shows the network structure and model parameter settings that we used in Model I to process the MNIST dataset. We put the max-pooling operation before the binarization operation in the training phase to reduce information loss. In the inference phase, we adjusted the input activation binarization operation of the next convolutional layer to the front of the max-pooling layer of the current layer. This is because, in the

inference phase, the final results of these two methods are the same, but the method of putting the binarization operation before the pooling operation is convenient for deployment on the hardware.

Table 1. Parameter configuration of the model I network.

	Training	Inference	Input Shape	Kernel Shape	
Preprocess	BatchNorm	BatchNorm	(1,28,28)		
	Quantify	Quantif (1 bit)	(1,28,28)		
1st conv layer	BnnConv	BnnConv	(1,28,28)	(20,1,5,5)	
	BatchNorm	Offset	(20,24,24)		affine = True
		Binarize	(20,24,24)		
	MaxPool	OR	(20,24,24)	(2,2)	Stride = 2
2nd conv layer	BnnConv	Conv	(20,12,12)	(50,20,3,3)	
	BatchNorm	Offset	(50,10,10)		affine = True
		Binarize	(50,10,10)		
	MaxPool	OR	(50,10,10)	(3,3)	Stride = 3
FC1 layer	BNNLinear	Linear	(450,1)	(120,450)	
	ReLU	ReLU	(120,1)		
FC2 layer	Linear	Linear	(120,1)	(10,120)	
	Softmax	Softmax			

In Model I, the first convolution layer used 20 convolution kernels with a size of 5 × 5. It took 25 PEs to complete the convolution operation and 10 PEs to complete the batch normalization function; thus, we set the single-row PE chain to 35 PEs. The second convolution layer used 50 convolution kernels with a size of 20 × 3 × 3. It required 180 PEs to complete the convolution function and 30 PEs to complete the batch normalization function. Therefore, a single-row PE chain included 210 PEs. The parameter settings of the batch normalization layer and the max-pooling layer are shown in Table 1. The FC1 layer used a convolution kernel with a size of 120 × 450, and the PE array had 450 time-domain PEs.

Similarly, the parameter settings of Model II corresponding to the MNIST dataset are shown in Table 2. Only the parts that were different from Model I are listed here. For example, the preprocessing stage quantized the data to 8 bits, where the input data were composed of one sign bit and seven decimal bits, and the weight was composed of one sign bit, three integer bits, and four decimal bits.

Table 2. Network parameter configuration in Model II (only parameters different from Model I are listed).

	Training	Inference	Input Shape	Kernel Shape	
Preprocess	BatchNorm	BatchNorm	(1,28,28)		
	Quantify	Quantify (8 bit)	(1,28,28)		
1st layer	Conv	Conv	(1,28,28)	(20,1,5,5)	
	BatchNorm	Comparator	(20,24,24)		affine = False
		Binarize	(20,24,24)		
	MaxPool	OR	(20,24,24)	(2,2)	Stride = 2

In Model II, the first layer of convolution operation used a 20 × 24 systolic array. This structure required a total of 480 fixed-point MACs, of which one fixed-point MAC used one DSP and eight LUTs. On other datasets, we can refer to this method to design the first layer separately to give priority to the use of DSP resources inside the FPGA.

The FC2 layers of Model I and Model II were the matrix multiplication of the weight matrix B (10 × 120) and the input activation matrix A (120 × N), where N represents the number of output feature maps of the previous layer. We controlled the first 10 rows of

PE of the systolic array to complete this work by adding a data selector, an output data selector, and a enable signal of PE, while other PEs were set to be inactive.

After testing, the accuracy of Model I and Model II was 97.5% and 98.8%, respectively. Since the first layer used fixed-point data, compared to Model I, the accuracy of our Model II was improved by 1.3%. In fact, in larger datasets, the accuracy of the first layer would have a greater impact on the final result.

5.2. The Impact of the Time-Domain Model on the Accelerator

First of all, benefiting from replacing the multiplication and accumulation in the digital domain with the delay difference of the time signal in the time domain, we used fewer logic resources compared with the addition tree structure. For example, a single PE in the time domain used only three LUTs, and only 540 LUTs were needed to complete a 20 × 3 × 3 convolution operation. More importantly, the intermediate results of the addition tree [16] or pop-count accumulation operation [14] are all multi-bit digital signals. All the intermediate results of our structure were represented by time signals; that is, only pairs of wires contained multi-bit values. This method greatly reduces power consumption.

Secondly, we analyzed the impact of the joint design technology of convolution and batch normalization in the time domain on the accelerator. On the one hand, this design can further reduce resource consumption, because the usual batch normalization requires a multi-bit adder, which we do not need. On the other hand, the joint design technology reduces the storage requirements for intermediate results. We took Model I and Model II as objects to compare the storage requirements in Figure 9b,c. As shown in Figure 14, the storage requirements of the first layer of Model I, the first layer of Model II, and the second layer were reduced to 25%, 25.1%, and 18.4%, respectively, using joint design technology.

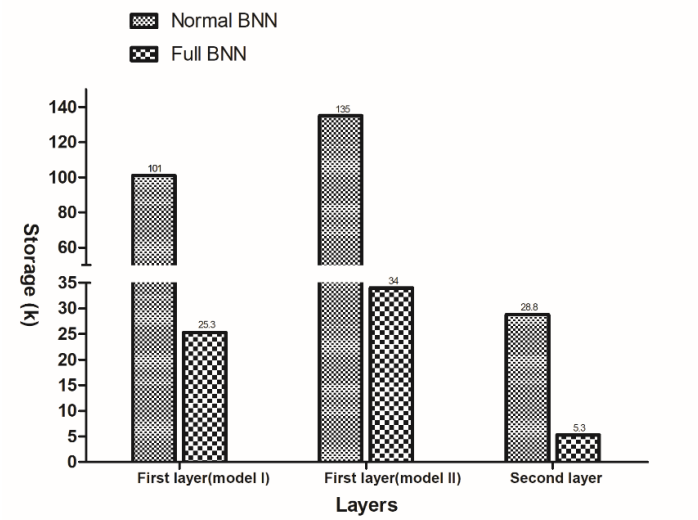

Figure 14. The impact of the codesign technology of convolution and batch normalization on storage requirements.

In the case of a relatively large network structure, the previous method of processing interlayer data was to first output the results of the previous layer to the off-chip memory, before importing these data in batches to participate in the calculation of the next layer, which caused a large amount of data transfer. Since this data transmission consumes more energy and time, our design hopes to avoid off-chip and on-chip data transmission in a larger network by reducing the storage requirements of each layer.

5.3. The Impact of Parallel Optimization on Accelerators

The parallel optimization parameters for the MNIST dataset are shown in Table 3. First of all, in terms of spatial dimension parallelism, some parallelism parameters were determined. For example, the parallelism of the convolution operation of the first layer of Model I was 20; thus, it could calculate an input feature map at one time. Some parallelism needed to be determined according to the resources of the platform. For instance, the parallelism of the second layer of convolution was selected as 10, and the calculation of 50 convolution kernels needed to be completed five times. Of course, in the case of sufficient resources, it is easy to increase the degree of parallelism to improve the calculation speed.

Secondly, in terms of time dimension parallelism, we determined the reasonable hold time and interval time according to the characteristics of the MNIST dataset and network model, as shown in Table 3.

Lastly, whereas the above discussion referred to parallel processing within layers, parallel processing between layers is also possible. For model I, this was equivalent to using a separate structure for each layer, which could achieve parallelism between layers. For Model II, although the first layer and the last layer used the same structure, they were separated by two layers; hence, their parallelism was not affected.

Table 3. Parallelization parameter setting.

	Layers	**Parallelism**	**Hold Time**	**Interval Time**
Model I	First layer	20 × 25	20	20
	Second layer	10 × 180	60	60
	Third layer	1 × 450	200	200
Model II	First layer	20 × 24	/	/
	Second layer	10 × 180	60	60
	Third layer	1 × 450	200	200

5.4. Related Work

We compared the two optimized model designs with related work, including BNNs implemented by traditional digital circuits and analog circuits.

As shown in Table 4, compared with the conventional digital-domain computing structure [14,16,34], our design had an advantage in storage consumption. Benefiting from the time domain and the joint design of convolution and batch normalization, our structure could reduce the total storage requirements by 75%. Our design minimized the on-chip and off-chip transfer of intermediate data, which is more meaningful for applications with limited storage space.

At the same time, since the switching activity of the circuit decreased during the "multiply–accumulate" operation in the time domain, this could promote the saving of power consumption. Thus, the time-domain model has great development potential. In addition, compared with the HLS method, we could use the embedded DSP resources to complete the calculation requirements of the key layer, thereby reducing the use of other logic resources, such as LUTs.

Compared with the HLS method, the flexibility of our structure still needs to be strengthened. For example, Finn [14] can flexibly choose the bit width of the weights and input activations (1 bit or 2 bit). Our structure can also be easily extended to multi-bit data using another mapping method.

Table 4. Deployment performance of binary neural networks.

	Precision	Accuracy	Topology	Resource Consuming			Performance	
				LUTs	DSP	BRAMs (18 k)	Clk (MHz)	Power (W)
Model I	Full BNN	97.5	CNV-4	30,075	100	127	125	3.5
Model II	BNN [1]	98.8	CNV-4	26,899	480	135	125	4.3
Finn [14]	BNN [1]	98.4	LFC [2]	82,988	-	396	200	22.6
FP-BNN [16]	BNN [1]	98.24	LFC [2]	182,301	20	2210	150	26.2
BinaryEye [34]	BNN [1]	98.4	MLP-4	40,000	-	110	100	12.2
FCA-BNN [35]	BNN [1]	98.79	LFC [2]	152,800	10	1384	166	5.7

[1] Partial binarization; [2] LFCs are three-layer fully connected networks.

The work of Miyashita et al. [25] first realized the second-layer structure of BNN through time-domain arrays and achieved good results. Their structure was characterized by the use of basic analog circuits, with the advantage of low power consumption.

However, this basic analog circuit had nonideal effects. Although some technical means can be used to reduce the error rate of a certain layer, this kind of error may not be ignored after the accumulation of several layers.

Their fully spatially unrolled structure divided the channels of the convolution kernel in a convolution layer into several groups for processing. After the convolution operation is performed, these groups of data need to be fused, which increases the calculation process. We chose to calculate all channels of a convolution kernel at one time, so that the output feature map of the convolution could be directly obtained. In addition, they did not consider parallelism in time, while we added parallelism in the time dimension to increase the processing speed of the time-domain PE array.

More importantly, despite their clever design, the complexity of the logical structure was also increased, making it not very easy to deploy on a large scale. This complexity is mainly reflected by the aspects described below.

On one hand, due to the design of $\tau_i = (\tau_{i-1} + w_i)x_i'$, the signal enters and exits the PE at an interval of 2–3 clock cycles, whereas our PE only needs 1–2 clock cycles. On the other hand, in the process of streamlining, additional calculation of the input activation value is required, i.e., $x_i' = x_i \; XNOR \; x_{i+1}$. It is necessary to additionally control the timing of the input values x_i and x_{i+1}. Experiments have found that this is more difficult, and it is generally necessary to calculate x_i' in advance. Therefore, compared with their structure, our design is more concise and easier to implement on a large scale. Thus, we can fully implement the entire BNN structure.

6. Conclusions

This article mainly focused on the computing structure of lightweight CNNs. By introducing the time-domain method, we designed a time-domain PE array that integrates binary convolution, batch normalization, and activation function operation. Then, we built a full-BNN model and a mixed-precision BNN model based on the time domain. Compared with existing BNN hardware structures, the full-BNN model has the advantages of low storage, low power consumption, and high efficiency. The mixed-precision BNN can use fixed-point data to ensure accuracy and use binary data in the time domain to improve calculation efficiency and reduce power consumption. This provides a tradeoff between complexity and accuracy.

In future work, we will consider adding a scaling factor to the end of the time-domain PE array, by adding a small number of multipliers to improve its accuracy in complex tasks. We will also consider multi-bit width (i.e., a weight of two or three digits) or a time

domain based on three-value representation to improve the processing accuracy of binary neural networks in more scenarios.

Author Contributions: Conceptualization, L.Z., Y.P. and T.Z.; methodology, L.Z. and T.Z.; validation, L.Z., T.Z. and X.T.; formal analysis, L.Z., X.H. and X.T.; investigation, L.Z.; resources, Y.P. and T.Z.; data curation, L.Z.; writing—original draft preparation, L.Z.; writing—review and editing, Y.P. and T.Z.; visualization, L.Z.; supervision, Y.P. and T.Z.; project administration, Y.P.; funding acquisition, Y.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the National Natural Science Foundation of China (No. 91948303-1, No. 61803375, No. 12002380, No. 62106278, No. 62101575, and No. 61906210) and the National University of Defense Technology Foundation (No. ZK20-52).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The evaluation for the BNN was based on MNIST database. Please refer to MNIST testset at http://yann.lecun.com/exdb/mnist/ (accessed on 10 December 2021).

Acknowledgments: The authors would like to thank the anonymous reviewers for their special effort.

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

References

1. Li, H.; Fan, X.; Li, J.; Wei, C.; Wang, L. A high performance fpga-based accelerator for large-scale convolutional neural networks, In Proceedings of the 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2016.
2. Xu, J.; Li, S.; Jiang, J.; Dou, Y. A simplified speaker recognition system based on fpga platform. *IEEE Access* **2020**, *8*, 1507–1516.
3. Shehzad, F.; Rashid, M.; Sinky, M.H.; Alotaibi, S.S.; Zia, M.Y.I. A scalable system-on-chip acceleration for deep neural networks. *IEEE Access* **2021**, *9*, 95412–95426.
4. Shi, W.; Jie, C.; Quan, Z.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *Internet Things J. IEEE* **2016**, *3*, 637–646.
5. Li, S.; Shen, X.; Dou, Y.; Ni, S.; Xu, J.; Yang, K.; Wang, Q.; Niu, X. A novel memory-scheduling strategy for large convolutional neural network on memory-limited devices. *Comput. Intell. Neurosci.* **2019**, 4328653.
6. Courbariaux, M.; Bengio, Y. Binarynet: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv* **2016**. https://arxiv.org/pdf/1602.02830v1.pdf.
7. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. Xnor-Net: Imagenet Classification Using Binary Convolutional Neural Networks. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Cham, Switzerland, 2016.
8. Zhou, S.; Wu, Y.; Ni, Z.; Zhou, X.; Wen, H.; Zou, Y. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv* **2016**, arXiv:1606.06160.
9. Hu, Q.; Wang, P.; Cheng, J. From hashing to cnns: Training binaryweight networks via hashing. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018.
10. Mishra, A.; Cook, J.J.; Nurvitadhi, E.; Marr, D. Wrpn: Training and inference using wide reduced-precision networks. *arXiv* **2017**, arXiv:1704.03079.
11. Bethge, J.; Yang, H.; Bornstein, M.; Meinel, C. Binarydensenet: Developing an architecture for binary neural networks. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), Seoul, Korea, 27–28 October, 2019; pp 1951–1960.
12. Liu, Z.; Luo, W.; Wu, B.; Yang, X.; Liu, W.; Cheng, K.T. Bi-real net: Binarizing deep network towards real-network performance. *Int. J. Comput. Vis.* **2020**, *128*, 202–219.
13. Zhao, R.; Song, W.; Zhang, W.; Xing, T.; Lin, J.-H.; Srivastava, M.; Gupta, R.; Zhang, Z. Accelerating binarized convolutional neural networks with software-programmable fpgas. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017.
14. Umuroglu, Y.; Fraser, N.J.; Gambardella, G.; Blott, M.; Leong, P.; Jahre, M.; Vissers, K. Finn: A framework for fast, scalable binarized neural network inference. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; Association for Computing Machinery: Monterey, CA, USA, 2017; pp. 65–74.
15. Blott, M.; Preußer, T.B.; Fraser, N.J.; Gambardella, G.; O'brien, K.; Umuroglu, Y.; Leeser, M.; Vissers, K. Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks. ACM Trans. *Reconfigurable Technol. Syst.* **2018**, *11*, 16.
16. Liang, S.; Yin, S.; Liu, L.; Luk, W.; Wei, S. Fp-bnn: Binarized neural network on fpga. *Neurocomputing* **2018**, *275*, 1072–1086.

17. Cheng, F.; Zhu, S.; Hao, S.; Lee, C.E.; Zhao, J. Towards fast and energy-efficient binarized neural network inference on fpga. *arXiv* **2018**, arXiv:1810.02068.
18. Han, Z.; Jiang, J.; Xu, J.; Zhang, P.; Zhao, X.; Wen, D.; Dou, Y. A high-throughput scalable bnn accelerator with fully pipelined architecture. *CCF Trans. High Perform. Comput.* **2021**, *3*, 17–30.
19. Xiang, M.; Teo, T.H. Implementation of binarized neural networks in all-programmable system-on-chip platforms. *Electronics* **2022**, *11*, 663.
20. Simons, T.; Lee, D.-J. Efficient binarized convolutional layers for visual inspection applications on resource-limited fpgas and asics. *Electronics* **2021**, *10*, 1511.
21. Biswas, A.; Chandrakasan, A.P. Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications. In Proceedings of the 2018 IEEE International Solid-State Circuits Conference(ISSCC), San Francisco, CA, USA, 11–15 February 2018.
22. Gonugondla, S.K.; Kang, M.; Shanbhag, N. A 42pj/decision 3.12tops/w robust in-memory machine learning classifier with on-chip training. In Proceedings of the 2018 IEEE International Solid-State Circuits Conference(ISSCC), San Francisco, CA, USA, 11–15 February 2018.
23. Liu, M.; Everson, L.R.; Kim, C.H. A scalable time-based integrate-and-fire neuromorphic core with brain-inspired leak and local lateral inhibition capabilities. In Proceedings of the 2017 IEEE Custom Integrated Circuits Conference (CICC), Austin, TX, USA, 30 April–3 May 2017; pp. 1–4.
24. Amravati, A.; Nasir, S.B.; Thangadurai, S.; Yoon, I.; Raychowdhury, A. A 55nm time-domain mixed-signal neuromorphic accelerator with stochastic synapses and embedded reinforcement learning for autonomous micro-robots. In Proceedings of the 2018 IEEE International Solid-State Circuits Conference(ISSCC), San Francisco, CA, USA, 5–9 February 2018.
25. Miyashita, D.; Kousai, S.; Suzuki, T.; Deguchi, J. A neuromorphic chip optimized for deep learning and cmos technology with time-domain analog and digital mixed-signal processing. *IEEE J. Solid-State Circuits* **2017**, *52*, 2679–2689.
26. Sayal, A.; Nibhanupudi, S.S.T.; Fathima, S.; Kulkarni, J.P. A 12.08-tops/w all-digital time-domain cnn engine using bi-directional memory delay lines for energy efficient edge computing. *IEEE J. Solid-State Circuits* **2020**, *55*, 60–75.
27. Sayal, A.; Fathima, S.; Nibhanupudi, S.T.; Kulkarni, J.P. Compac: Compressed time-domain, pooling-aware convolution cnn engine with reduced data movement for energy-efficient ai computing. *IEEE J. Solid-State Circuits* **2021**, *56*, 2205–2220.
28. Maharmeh, H.A.; Sarhan, N.J.; Hung, C.C.; Ismail, M.; Alhawari, M. A comparative analysis of time-domain and digital-domain hardware accelerators for neural networks. In Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Korea, 22–28 May 2021; pp. 1–5.
29. Tang, W.; Hua, G.; Wang, L. How to train a compact binary neural network with high accuracy. In Proceedings of the 31st AAAI Conference on Artificial Intelligence, AAAI 2017, San Francisco, CA, USA, 4–10 February 2017; AAAI Press: San Francisco, CA, USA; pp. 2625–2631.
30. Mocerino, L.; Calimera, A. Fast and accurate inference on microcontrollers with boosted cooperative convolutional neural networks (bc-net). *IEEE Trans. Circuits Syst. I Regul. Pap.* **2021**, *68*, 77–88.
31. Fasfous, N.; Vemparala, M.R.; Frickenstein, A.; Stechele, W. Orthruspe: Runtime reconfigurable processing elements for binary neural networks. In Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 1662–1667.
32. Li, Y.; Liu, Z.; Xu, K.; Yu, H.; Ren, F. A 7.663-tops 8.2-w energy-efficient fpga accelerator for binary convolutional neural networks. In Proceedings of the FPGA, Monterey, CA, USA, 22–24 February 2017. Available online: http://yann.lecun.com/exdb/mnist (accessed on 31 March 2022).
33. Lecun, Y.; Cortes, C. The Mnist Database of Handwritten Digits. 1998. Available online: http://yann.lecun.com/exdb/mnist (accessed on 31 March 2022).
34. Gao, J.; Yao, Y.; Li, Z.; Lai, J. Fca-bnn: Flexible and configurable accelerator for binarized neural networks on fpga. *IEICE Trans. Inf. Syst.* **2021**, *E104.D*, 1367–1377.
35. Jokic, P.; Emery, S.; Benini, L. Binaryeye: A 20 kfps streaming camera system on fpga with real-time on-device image recognition using binary neural networks. In Proceedings of the 2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES), Graz, Austria, 6–8 June 2018, 2018; pp. 1–7.