Received 6 June 2024, accepted 9 July 2024, date of publication 2 August 2024, date of current version 12 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3437663

RESEARCH ARTICLE

An Accelerated FPGA-Based Parallel CNN-LSTM Computing Device

XIN ZHOU, WEI XIE, HAN ZHOU, YONGJING CHENG, XIMING WANG, YUN REN, SHANDONG YUAN, AND LIUWEN LI
College of Information and Communication, National University of Defense Technology, Wuhan, Hubei 430000, China
Corresponding authors: Xin Zhou (13165323626@163.com) and Wei Xie (xiewei197466@163.com)

This work was supported by the National Natural Science Foundation of China under Grant 62201581.

ABSTRACT Recently, the combination of convolutional neural network (CNN) and long short-term memory (LSTM) exhibits better performance than single network architecture. Most of these studies connect LSTM networks behind CNNs. When operating on hardware, the current design of CNN-LSTM is similar to a pipeline architecture. However, the classic structure lead to a feature loss when data is sent to LSTM since CNN is not good at extracting temporal features. At the same time, as the depth and scale increases, it will bring a huge amount of computation, which makes hardware implementation difficult. Based on that, a parallel CNN-LSTM architecture is proposed, in which two networks extract features from the input data synchronously, being proven to be more effective than classical CNN-LSTM. This paper designs a parallel CNN-LSTM computing device based on FPGA. The device is divided into control unit and operation unit. Control stream and data stream transport between the two units, ensuring the proper running of the device. A highly parallel multi-channel convolution layer and pooling layer are designed to improve the calculation efficiency. A 4-stage pipeline structure is adopted to implement the LSTM part. This paper makes full use of on-chip BRAM to design a look-up table for activation function approximation, reducing the resource consumption by 95% compared with the traditional polynomial approximation. Finally, we verify our device under cooperative spectrum sensing (CSS) and handwritten classification scenarios. Proposed device reaches higher accuracy in two scenarios compared with classic CNN-LSTM structure as well as faster calculating speed, and the overall project power is limited below 2W. The scalability and limitation of this computing device are also discussed.

INDEX TERMS CNN-LSTM, field programmable gate array (FPGA), hardware acceleration, deep learning.

I. INTRODUCTION

Deep learning (DL) includes a number of well-established algorithms such as convolutional neural network (CNN), long short-term memory (LSTM) and generative adversarial network (GAN) etc. Each of them has its own application scenarios. Among them, CNN is mainly used for visual tasks [1]. By using different types of neural network layers, CNN can extract relative features from input and being applied to tasks like image classification etc [2]. While LSTM is a special kind of recurrent neural network (RNN), can solve the problem of gradient vanishing and easily learn long-term dependent information [3]. Compared to CNN, LSTM is good at extracting time features from 1D data, and is often used in tasks such as time series prediction. To take advantage of two different networks, recent studies have proposed a variety of hybrid CNN-LSTM structures by combining two networks for more complex feature extraction. In the classic CNN-LSTM structure, the data first feed the CNN module, and after the CNN completes the calculation, the result is passed to the LSTM module for subsequent operation. References [4], [5], [6], [7], [8], [9], and [10] apply hybrid CNN-LSTM in virtual machine workload forecasting, activity recognition, air quality prediction and wind speed retrieval etc. Unlike the classical structure, [11] connects CNN and LSTM in parallel, in which CNN can capture special features while LSTM can capture temporal feature from input synchronously. The results show that the parallel

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Wei.

connection has a better detection effect than that of the classic CNN-LSTM. In the specific applications, the combination of CNN and LSTM is proven to be more robust and efficient than single network.

Despite a variety of algorithms have been proposed, their huge computational requirement have led to the difficulty of implementation on resource-limited hardware [12]. Field-programmable gate arrays (FPGAs) are considered to be one of the most reliable and lightweight platform for implementing computationally intensive algorithms nowadays. With its product term structure, parallel computing properties, and the large number of configurable compute resources and memory blocks, FPGAs have better performance and lower power consumption than GPUs, and higher flexibility than ASICs [13]. In recent years, many researchers have deployed DL algorithms on FPGAs for different purposes. References [14], [15], [16], [17], [18], [19], [20], [21], and [22] use FPGA-based CNN architecture for object detection, image classification and edge computing. There are also many researchers focus on the acceleration of DL algorithms on FPGA, which are divided into two directions: quantization and weight reduction [23]. The quantization method reduces the size of the weights or the input and output data to accelerate hardware. While the weight reduction works on pruning redundant connections and allowing multiple connections to share the same weights to achieve acceleration. Reference [24] summarizes the requirements for memory, computational resources and system flexibility for mapping CNNs on embedded FPGAs, proposing a programmable and flexible CNN accelerator architecture "Angel Eye" as well as a data quantization strategies and compilation tools, which can reduce the bit width to 8 bits with little loss of accuracy. Reference [25] designs a hardware-oriented CNN compression strategy by dividing a deep neural network into "nopruning layers (NP-layers)" and "pruning layers (P-layers)" to improve its computing efficiency.

Based on current research, considering the superiority of parallel CNN-LSTM and the characteristics of FPGA, we design a generalizable parallel CNN-LSTM computing device for different tasks. The device is deployed on FPGA and get accelerated with different ideas. The main contributions of this paper are summarized as follows:

1) A parallel CNN-LSTM computing device is designed on the FPGA platform. The device is divided into control unit and operation unit. Control unit makes sure the proper running of device, while the operation unit controls CNN and LSTM calculating input data parallelly as well as the subsequent calculations.
2) A highly parallel multi-channel convolution layer and pooling layer are designed to improve the calculation efficiency. A 4-stage pipeline structure is adopted to implement the LSTM part. This paper makes full use of on-chip BRAM to design a look-up table to fit activation function, saving on-chip resources and improving the computing speed.
3) The modular design idea is used to realize the computing device. The network scale can be reconfigurable to apply different tasks.
4) The device's scalability is verified under cooperative spectrum sensing (CSS) and handwritten classification scenario. The detection results show better accuracy than classic structure. The versatility is also tested by deploying computing device on different platforms.

II. RELATED WORK

A. CNN-LSTM

CNN can extract and process significant features of input data to fulfill tasks like computer vision, speech processing, face recognition and other fields by applying the weights in each layer [26]. A typical CNN has three types of layers: convolution layer, pooling layer and fully connected layer [27].

Convolution layers are used in CNNs for feature extraction. During the computing, the convolution layer accepts feature graph $X_{W,H,D}$, and uses the convolution kernel (or filter) $F_{F_w,F_h,D,K}$ to generate a new feature graph $Y_{W,H,K}$ through convolution operation, and the calculation formula is written as [25]:

$$Y(i,j,k) = \sum_{d=1}^{D}\sum_{q=1}^{F_w}\sum_{p=1}^{F_h} F(p,q,d,k)X(i \times s + p, j \times s + q, d) \tag{1}$$

where W and H represent the width and height of the feature map. F_w and F_h denotes the size of the convolution kernel. Normally, $F_w = F_h$ when the input is 2D. D, K denotes the number of input and output channels. s denotes the convolution step.

The pooling layer reduces the size of the data by computing a local area of the feature map to output one pixel. Commonly used pooling methods include average pooling and maximum pooling, where average pooling calculates the average value of local fields, and maximum pooling selects the maximum value of local fields as output. The fully connected (FC) layer is often used as the last few layers of a CNN. All input neurons are fully connected to each neuron in the next layer by weights.

LSTM networks can selectively remember important features and discard some relatively unimportant features for a long time during network propagation, which is often used to solve the problem of long sequence modeling [28]. (2) describes the calculation of the four gates of a classical LSTM network as well as the method for generating the cell state and the cell output [14].

$$\begin{cases} i_t = sigmoid(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\ f_t = sigmoid(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\ o_t = sigmoid(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\ g_t = \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \\ c_t = f_t \odot c_{t-1} + i_t \odot g_t \\ h_t = \tanh(c_t) \odot o_t \end{cases} \tag{2}$$

i, f, o, g, c and h are, respectively, input gate, forget gate, output gate, updating gate, cell state and cell output with the same width. The subscript t denotes the current time step and $t-1$ denotes the previous time step. W_* denotes the weight matrix, the magnitude of which is determined by the number of inputs and outputs. b_* denotes the bias vector. The $+$ denotes element-wise addition and the $\odot$ denotes the Hadamard product. Eight pairs of matrix multiplication shown in (2) can be replaced by the transpose formula of the matrix $\begin{pmatrix} x^T & h^T \end{pmatrix} \begin{pmatrix} W_{x*}^T \\ W_{h*}^T \end{pmatrix}$ to facilitate the operation of the matrix by the circuit. That is, x and h can be concatenated into an single input vector to fulfill later calculation.

1) HYBRID CNN-LSTM

In current research, hybrid CNN-LSTM mostly refers to a serial connection of CNN and LSTM. Figure 1 gives the common structure of hybrid CNN-LSTM. It could be seen that the network input is directly sent to the CNN. After multiple rounds of convolution layers, activation functions and pooling layers, results from CNN are flattened to 1D and sent to LSTM for subsequent calculating. When implementing this structure using a hardware description language, it is common to use a pipeline architecture, in which first data is processed by CNN and then moved to the LSTM, in the meantime, new data feed the CNN and so on. However, since CNN is not good at extracting time features, it will lead to the loss of features in the previous rounds of convolution and pooling, leading to a decrease in the network fitting effect.

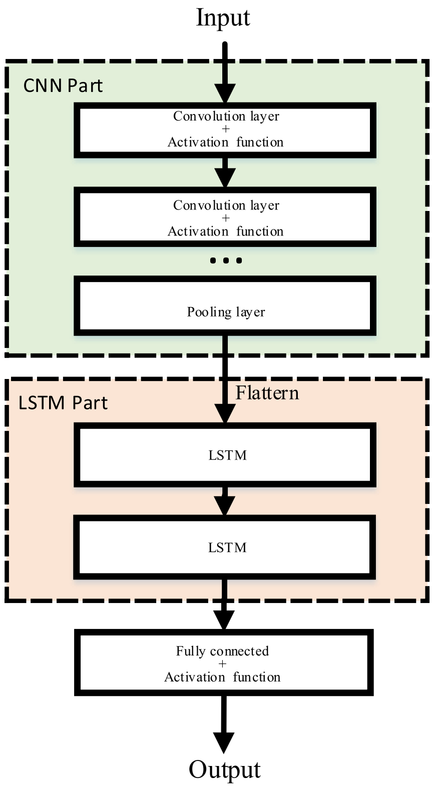
2) PARALLEL CNN-LSTM

Based on the problems with the serial connection, a parallel CNN-LSTM structure is proposed to solve CSS under cognitive radio [11]. By connecting CNN and LSTM parallelly, the structure manages to extract the temporal and relevant features at the same time. As shown in Figure 2, The parallel CNN-LSTM structure starts with CNN and LSTM processing input data synchronously. After both parts finishing their calculation, Their outputs will be flattened to 1D for subsequent data stitching operations so as to fuse the features extracted by two networks. The stitched data will be calculated through fully connected layer, activation function and softmax layer to get the final output.

Parallel structure design avoids the loss of characteristic information that occurs when a serial connection is made. At the same time, the parallel processing method also shortens the time delay of computing data and provide convenience for subsequent hardware implementation. Reference [11] has proven that parallel CNN-LSTM is better than classic CNN-LSTM and single network in terms of sensing effect and convergence speed.

B. FPGA-BASED CNN-LSTM IMPLEMENTATION

Table 1 compares several recent FPGA-based CNN-LSTM design architectures. As it shows, [29] proposes a 1D CNN-LSTM implementation architecture based on FPGA. The front end of this architecture is a 1D CNN. After the CNN module completes the calculation, the data is sent to the LSTM module. The block circulant weight matrix is used to assign the cell value for LSTM cell, and this paper uses this architecture to achieve a highly accurate electrocardiography (ECG) classification. The CNN-LSTM architecture designed


Input
CNN Part
Convolution layer + Activation function
Convolution layer + Activation function
Pooling layer
LSTM Part
Flattern
LSTM
LSTM
Fully connected + Activation function
Output
FIGURE 1. Classic hybrid CNN-LSTM structure.

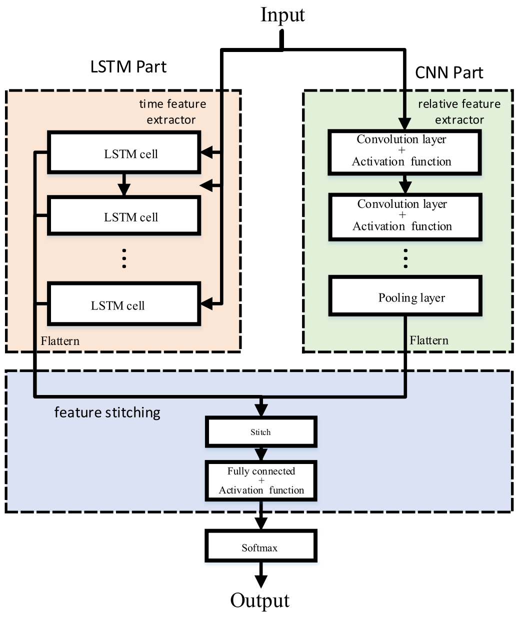

Input
LSTM Part
CNN Part
time feature extractor
relative feature extractor
LSTM cell
LSTM cell
LSTM cell
Convolution layer + Activation function
Convolution layer + Activation function
Pooling layer
Flattern
Flattern
feature stitching
Stitch
Fully connected + Activation function
Softmax
Output
FIGURE 2. Parallel CNN-LSTM structure.

TABLE 1. Comparison of FPGA-based CNN-LSTM design architectures.

work	CNN design(at one time)	LSTM design	CNN-LSTM structure	acceleration strategy
[29]	single channel	sequential execution	pipeline	block circulant weight marix
[19]	single channel	3 operating cycles	pipeline	time-sharing multiplexing, DSP groups
[30]	multiple channel	sequential execution	pipeline	streaming architecture,mapping strategy
[13]	multiple channel	/	/	depthwise convolution
[25]	single channel	/	/	CNN compression
[14]	/	5-stage pipeline	/	fast approximate matrix multiplication
this work	multiple channel	4-stage pipeline	parallel	LUT-based approximation, depthwise convolution

in [19] uses DSP to accelerate network operations, and also realizes the pipeline operations of CNN-LSTM. In the design of CNN, the data of the feature map is divided into four DSP groups and calculate at the same time, which can realize the fast convolution of single channel. In the design of LSTM, the time-sharing multiplexing strategy is adopted, dividing the operation into three cycles. Reference [30] proposes a high-throughput and resource-efficient classic CNN-RNN fusion accelerator on FPGA with commercial OpenCL. Multi-channel convolution is used to improve the speed when CNN is designed, and resource utilization is improved by mapping LSTM operations to the convolution engine when designing LSTM. Reference [13] uses depthwise convolution instead of standard convolutions for embedded platforms based on FPGA. That is, factorizing the standard convolution into a depthwise convolution plus a pointwise convolution, reaching high-speed convolution operations with little loss of accuracy. Reference [25] proposes a hardware-oriented CNN compression strategy, which divides the deep neural network (DNN) models into ‘‘nopruning layers (NP-layers)’’ and ‘‘pruning layers (P-layers)’’ and process separately according to its characteristics to improve the calculation speed. Reference [14] designs a 5-stage pipeline architecture for LSTM implementation, and a fast approximate matrix multiplication technique is used to reduce the delay encountered in the layer calculation of LSTM network.

III. PARALLEL CNN-LSTM COMPUTING DEVICE AND KEY MODULES DESIGN

Although the current CNN-LSTM design is a pipeline structure in terms of hardware implementation, CNN and LSTM can also work ‘‘parallelly’’ after a period of time. However, the data calculated by LSTM is not the original data but the data processed by CNN, so it will inevitably bring about the loss of features. At the same time, because the back-end LSTM is good at processing timing signals, some design architectures are based on 1D data, which is not conducive to application in multiple scenarios. In this paper, a parallel 2D CNN-LSTM architecture is designed to extract features from the input synchronously. In the design of CNN, the idea of multi-channel convolution is drawn to achieve the parallel convolution of all channels. An optimized 4-stage pipeline structure is proposed in the design of the LSTM.

A. OVERALL ARCHITECTURE

So as to design a general parallel CNN-LSTM computing device on FPGAs, the implementation architecture we propose is shown in Figure 3. It is evident that the computing device is divided into control unit and operation unit. The function of control unit is generating control stream, including enable signal, reset signal of different modules and synchronization signal. Especially, the synchronization signal calculates the running time of different modules, making sure that modules are ready when data is exchanged between them. The control stream transmits between two units and ensure the proper running of the whole device.

The operation unit is responsible for the operation, storage, stitching and output of data. The operation module mainly transmits the data stream, including the address information of the on-chip BRAM and the calculation results of each unit. Especially, to improve the speed of calculation and resource utilization, this paper take advantage of the on-chip BRAM resources to store the large amount of data used for computation. The BRAM is divided into different sections, separately storing the network input, network output, the weight data obtained from training and the function value used for activation function approximation. During the calculation, on-chip BRAM outputs the data to the buffer and then send them to corresponding modules. Figure 3 shows the basic parallel CNN-LSTM deployment framework on hardware, where the CNN and LSTM modules calculate input data read from on-chip BRAM simultaneously. The detailed design of key modules is given in the following sections.

B. MULTI-CHANNEL CONVOLUTION WITH A HIGH DEGREE OF PARALLELISM

In this paper, the design idea of convolution layer is to design a general-purpose convolution circuit with high flexibility and high parallelism, so that the parameters like depth, height, width and number of channels can be modified according to specific task without designing new convolution circuits.

The schematic diagram of the convolution layer is shown in Figure 4. Assuming that the network input parameter is $W \times H \times D$, the convolution kernel parameter is $F_w \times F_h \times D \times K$ as described in Section II. Data is stored as a 16-bit floating-point number. When the convolution stride is 1, during convolution, the convolution window needs to slide to the right $W - F_w + 1$ times and down $H - F_h + 1$ times on the feature map, for a total of $(W - F_w + 1) \times (H - F_h + 1)$ convolution operations for one single channel.

In this paper, during the process of sliding right $W - F_w + 1$ times for convolution kernel on the feature map of all channels, all the data in the convolution window is defined

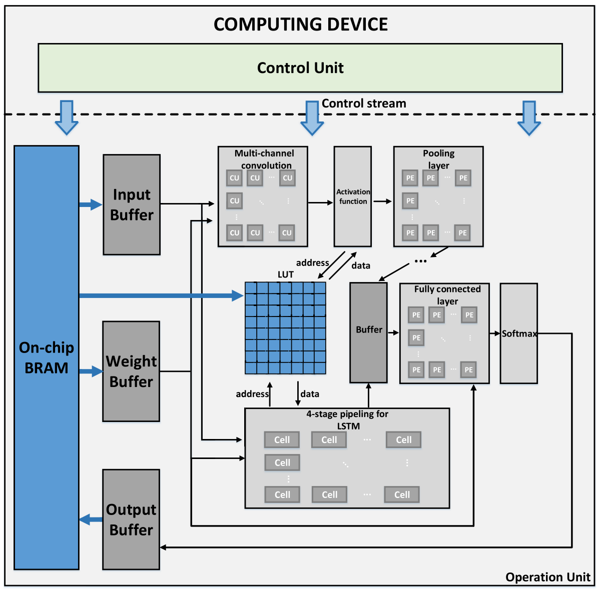


FIGURE 3. The architecture of parallel CNN-LSTM computing device.

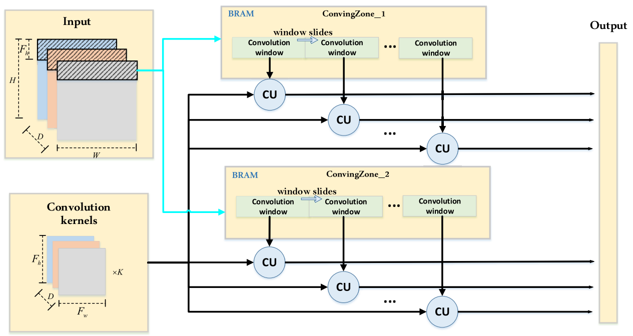


FIGURE 4. Architecture of convolution layer.

as the "Conving Zone". We use on-chip BRAM to store Conving Zone by address and read them sequentially when operating convolution.

In order to prevent insufficient BRAM storage space when the amount of data is too large, Conving Zone is divided into two parts and store separately. To improve the speed of operation, this paper chooses the strategy of exchanging area for speed, generating $D \times (W - F_w + 1)$ convolution units (CUs) to calculate the data stored in two Conving Zones. All these CUs work at the same time to realize the convolution of multi-channel feature map with a high degree of parallelism. N_{CU} denotes the number of clock consumed by one convolution operation on CU. After the convolution of current Conving Zone is completed, the Conving Zone shifts down for the second round of convolution. In this high-parallelism computing mode, it takes total $K \times (H - F_h + 1) \times N_{CU}$ to finish convolution.

An example is given with the input feature map as $6 \times 6 \times 8$ and the convolution kernel as $3 \times 3 \times 8 \times 8$ to verify the function of convolution module. The calculation process of CU is shown in Figure 5. Z represents the data stored in the Conving Zone as shown in Figure 4, C represents the kernel data of the same channel as the convolution data, and b represents the output produced by single CU. Depending on the kernel size, it takes 9 times for single CU to complete a convolution operation. During the calculation, Conving Zone outputs 9 data to the CU at a time and multiplies them with the corresponding kernel data. The data processed by the two adjacent CUs needs to be shifted 9 times. Under the current 2 Conving Zones, there are $D \times (W - F_w + 1)$ same CUs parallel operations. After the calculation is completed, Conving Zone is updated for the next moment. Corresponding data calculating process and timing diagram are shown in Figure 6. It can be seen that when the convolution starts, the filter data is divided into $9 \times 16 = 144$ bits since the data is stored in float16 format. The Conving Zone of the eight channels store the data sequentially in the BRAM and each channel divides the data into $6 - 3 + 1 = 4$ blocks of 144 bits according to the size of the filter. These data are multiplied by the filter at the same time, that is, convolution operations of the filter shifting right 4 times on the 8 channels are completed. At the next time slot, Conving Zone moves downward, the stored data is updated and convoluted with the current filter.

Compared with FPGA-based depthwise separable convolution proposed in [13] and single channel convolution in [19] and [29], the design of convolution layer in this paper ensures the high parallelism of convolution unit while realizing multi-channel convolution, and ensures that it has a faster operation speed when performing large-scale convolution. In order to prove the effect of the convolution module designed in this paper, a feature map of $36 \times 36 \times 8$ is taken as input, the size of the convolution kernel is $4 \times 4 \times 8$, and the step size is 1. As shown in Table 2, the consumption of the module resources designed in this paper is slightly increased due to the high parallelism, but the computing speed is significantly improved.

TABLE 2. Resource usage comparison between different design idea of convolution layer.

ideas	LUT	FF	DSP	Latency
[13]	6134	6730	80	428
[19]	8034	6902	72	512
[29]	6509	7002	81	567
This work	7826	7248	89	108

C. FULLY PARALLEL POOLING LAYER

During pooling operation, the size of input feature map is $D \times H \times W$, and the feature map is "scanned" by a filter with the size of $M \times N$. The value is calculated and output to the next layer. The architecture of pooling layer is shown in Figure 7. Similar to the convolution layer implementation, the pooling layer reads data from the output register of previous layer, then generates $D \times \frac{H \times W}{M \times N}$ pooling units for parallel pooling. We also use an output register to store the pooling result. The architecture of average-pooling unit (PU) is shown in Figure 8. The adder adopts a pipeline structure: previous

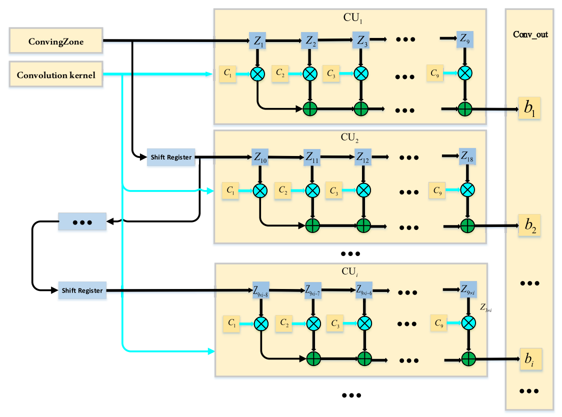


FIGURE 5. Architecture of convolution unit (CU).

adding result is used as the input of the next adder, and a multiplication (averaging) operation is performed to obtain the result after the accumulation is completed.

Figure 9 illustrates the calculation timing of average pooling layer when $H = W = 4$ and $M = N = 2$. When pooling, four identical pooling units are generated in parallel, and each pooling unit costs one pooling period (decided by the size of filter) to obtain the result.

D. 4-STAGE PIPELINE FOR LSTM IMPLEMENTATION

Due to the special structure of LSTM as shown in Figure 10, the calculation within and between LSTM cells can only start after the previous module completing its calculation. Thus it is difficult to design an implementation architecture with a high degree of parallelism. To minimize the clock resources consumption, we use a 4-stage pipeline architecture to deploy LSTM cell as shown in Figure 11. The LSTM cell input includes c_{t-1} and h_{t-1} from last LSTM cell, network input X_t, weights and bias matrix for four gates: W_{h*}, W_{x*} and $bias$.

Stage 1 consists of two blocks for splicing weights and input, two "4 to 1" multiplexors to select the weights and bias matrix corresponding to four gates and an adder to calculate the data prepared for subsequent sigmoid and tanh functions. The prepared data is stored in buffer for stage 2. Stage 2 includes a "1 to 2" demultiplexor to send the input to sigmoid or tanh separately and a block for computing the output of four gates: i_t, f_t, g_t and o_t. Stage 3 is used for the data preparation and calculation for Hardeman product as given in (2). Hardeman product requires the value of tem_3, c_{t-1} and output of gates as shown in Figure 10, while c_t is needed through another tanh block to obtain tem_3, thus a buffer block is synthesized for storing the output of four gates to synchronize the pipeline between stage 2 and stage 3. After completing the Hardeman product and storing the result in buffer, stage 4 consists an accumulator for h_t and a "1 to 2" demultiplexor to divide c_t and h_t of current cell. The timing diagram of LSTM cell is shown in Figure 12.

Between two adjacent cells, the latter cell set the c_{t-1} and h_{t-1} from previous cell as the driving signal, ensuring that the data is instantly transferred and calculated, which saves clock resources to the greatest extent.

We verify the LSTM module designed in this paper by entering an input of $16\times16\times8$, and the resource consumption is given in the Table 3. It could be seen that compared with the 5-stage pipeline structure proposed in [14], the 4-stage pipeline structure in this paper reduces the waiting latency of data. And the stability of the operation is improved by storing data in the buffer between different stages compared with [19].

TABLE 3. Resource usage comparison between different design idea of LSTM cell.

ideas	LUT	FF	DSP	Latency
[14]	6185	3025	12	489
[19]	5017	2967	16	562
This work	4969	2399	7	422

E. LUT-BASED ACTIVATION FUNCTION APPROXIMATION

Activation functions are used in neural networks to introduce nonlinearity and improve the fitting ability of the network. Common activation functions include tanh, sigmoid, relu, leaky relu, etc. The computing device proposed in this paper uses three activation functions: tanh, sigmoid and relu. The common implementation method of activation functions in hardware description language is polynomial approximation. This method divides the function into several intervals. In each interval, the polynomial approximation formula is used to convert the nonlinear calculation into polynomial calculation, eliminating the higher-order terms and retaining low-order terms required to ensure accuracy. This process requires multiple operations on each value to obtain a single result, consuming loads of logic and clock resources.

Therefore, this paper uses the look-up table (LUT) method instead of the polynomial approximation method. Specifically, we use FPGAs' on-chip BRAM to store the corresponding activation function values of all possible values under the 16-bit floating-point number in a certain range, then designs a two-level mapping from input to address and address to function output. The activation function value can be obtained in only two clock cycle and the obtained data can be accurate to 5 decimal places. Algorithm 1 demonstrates the LUT fitting method of sigmoid functions as an example.

According to Algorithm 1, when the input is greater than 8 or less than −8, the output of the sigmoid function is considered to be 1 or 0, and when the input range is [−8,8], the data output range is [0,1], which are symmetrical with (0,0.5) as the center. In this paper, two BRAM data storage segments are constructed. The operation first determines the positive and negative input to correspond to different BRAMs, then compare the data exponential bit (2 to 6 bits of the input signal) with the binary code '10010', and then uses the last 15 bits of the 16-bit floating-point number as the address code (input) of BRAM. This BRAM look-up table-based design takes only two clock cycle from input to output.

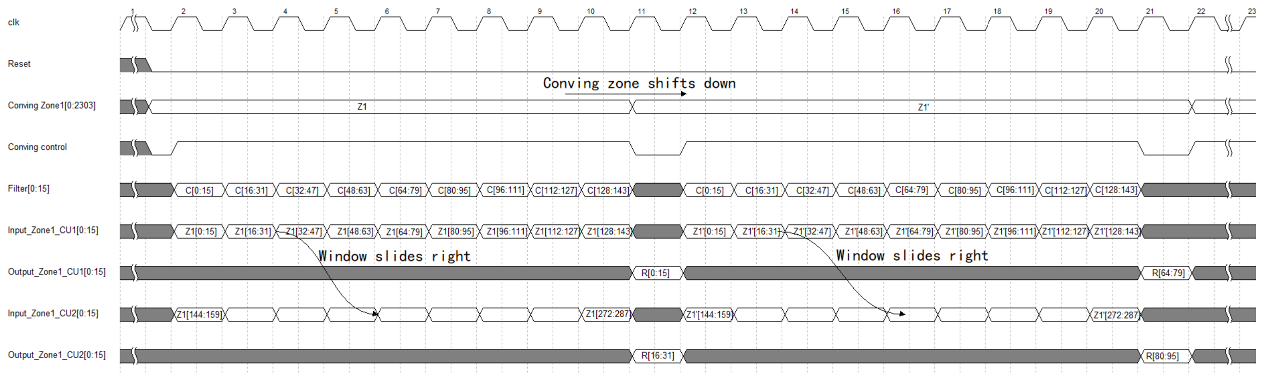


FIGURE 6. The timing diagram of the proposed Convolution Layer.

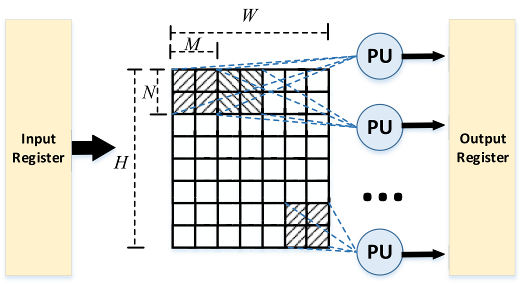


FIGURE 7. Architecture of pooling layer.

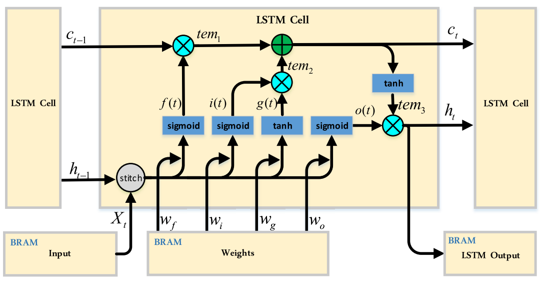


FIGURE 10. Architecture of LSTM cell.

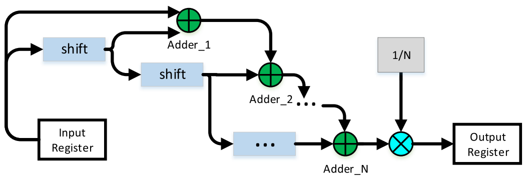


FIGURE 8. Architecture of Average-Pooling unit.

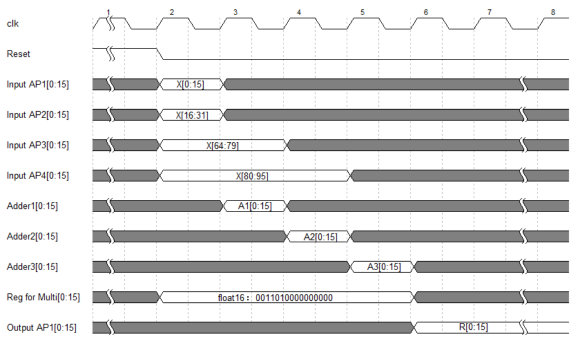


FIGURE 9. The timing diagram of the pooling layer.

Algorithm 1 Sigmoid Approximation Using LUT

```
Input: x in float16
Output: y in float16
 1: if $x[1] = 0$ then
 2:     $BRAM \leftarrow BRAM\_pos$          ▷ select BRAM storing positive data
 3:     if $x[2:6] >' 10010'$ then
 4:         $y \leftarrow' 0011110000000000'$    ▷ if $x > 8$ set $y = 1$
 5:     else
 6:         $address \leftarrow x[2:16]$
 7:         $y \leftarrow$ output of BRAM
 8:     end if
 9: else
10:     $BRAM \leftarrow BRAM\_neg$          ▷ select BRAM storing negtive data
11:     if $x[2:6] >' 10010'$ then
12:         $y \leftarrow' 0000000000000000'$ ▷ if $x < -8$ set $y = 0$
13:     else
14:         $address \leftarrow x[2:16]$
15:         $y \leftarrow$ output of BRAM
16:     end if
17: end if
```

The implementation of the tanh is similar to the idea of sigmoid. Since the tanh function is symmetrical with the origin, it only needs to construct the BRAM of the positive part. When the input is negative, the address code is the same as the positive part, and the output only needs to replace the first bit with '1'.

According to its expression, the Relu activation function does not need to build a LUT to store its data, but only needs to zero the part less than 0 and keep the part greater than zero.

Figure 13 gives the approximate activation functions of Tanh and Sigmoid obtained by LUT method. It can be seen that the approximate function designed in this paper has great fitting effect to the same as the original function. In order to compare the LUT and the polynomial approximation method,

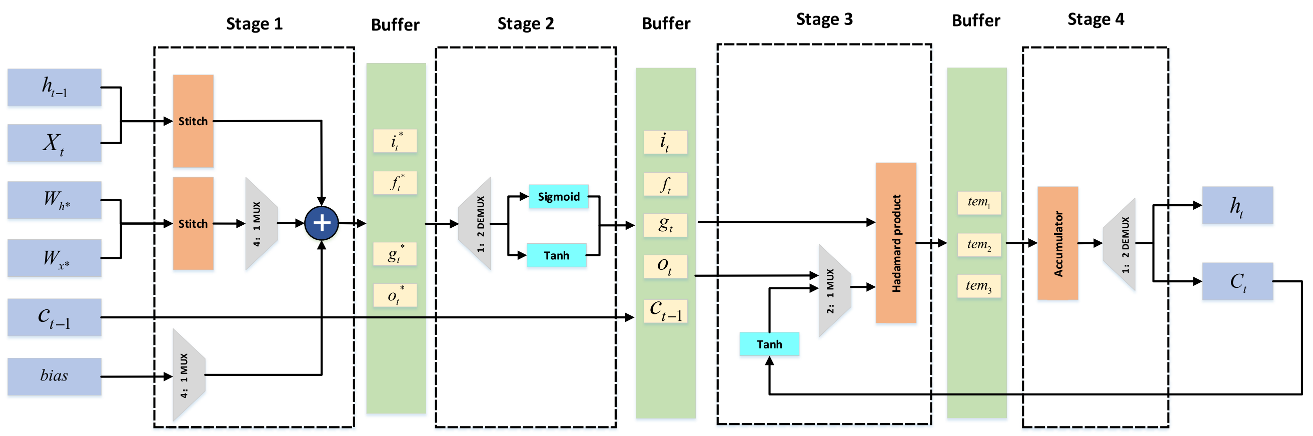


FIGURE 11. 4-stage pipeline architecture for LSTM implementation.

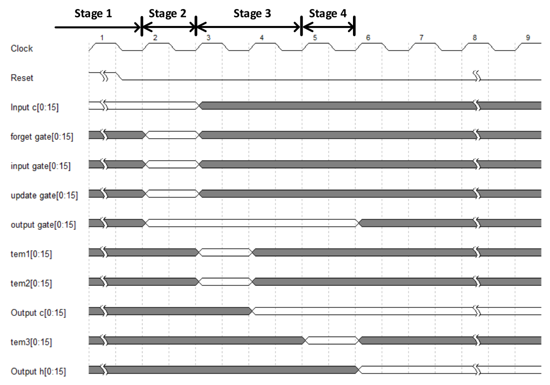


FIGURE 12. The timing diagram of the proposed LSTM model.

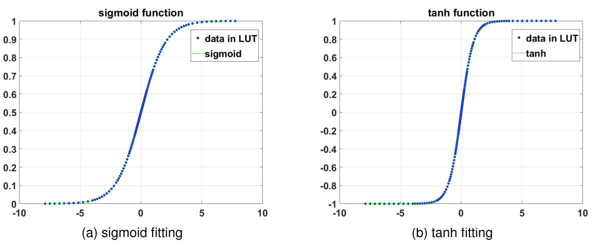


FIGURE 13. Proposed activation function approximation.

we list the logic and clock resources cost by both methods and method proposed in [14] in Table 4. It can be seen that compared with the polynomial approximation, LUT method reduces the resource consumption of the device to a great level. Taking the tanh function as an example, the LUT and FF resources in the FPGA are reduced by 92.5% and 96.0 % respectively. Compared with the [14], which also uses the LUT-based method, this paper uses float16 to represent the data, expanding the accurate range, simplifying the process of data comparison before querying, and can directly use a part of the data as address code, which effectively improves the calculation speed.

F. FULLY CONNECTED LAYER

The fully connected (FC) layer integrates the feature map obtained from previous convolution and pooling layer calculations into a single value, reducing the influence of feature locations on the classification results and improving the robustness of the whole network. It operates in the same way as a convolution operation when implemented on hardware. According to the CNN-LSTM architecture shown in Figure 2 in this paper, the data size processed by the fully connected layer is 1D. The FC design architecture is shown in Figure 14. M and N denotes the number of input and output data. When calculating, N processing elements (PE) are generated in parallel. The input of each PE is the weight data read from BRAM and data of M input nodes. The architecture of PE is similar to that of CU in section I. Multipliers and adders in each PE complete the calculation obtain one pixel of data of the output layer.

TABLE 4. Resource usage comparison between approximation method.

function	method	LUT	FF	Latency
tanh	Polynomial	7287	2036	22
	[14]	2224	1429	10
	This work	317(92.5% ↓)	81(96% ↓)	2
sigmoid	Polynomial	3003	1382	16
	[14]	1608	891	15
	This work	369(87.7% ↓)	144(89.6% ↓)	2

G. SOFTMAX

Finally, the parallel CNN-LSTM network proposed in this paper uses the softmax function to realize classification. The softmax function is defined by the formula [31]:

$$softmax(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{K}\exp(z_j)} for\ i = 1, \ldots, K \tag{3}$$

Softmax is to map each input element to real numbers between 0 and 1, and to normalize the guaranteed sum to 1 by dividing by the sum of all the exponentials. Since after multiple rounds of convolution, pooling, and fully connected layers, the last layer of the parallel CNN-LSTM tends to have a small amount of data. We argue that it is acceptable to use the polynomial approximation method for softmax. Taking binary classification for example, we use 7 taylor

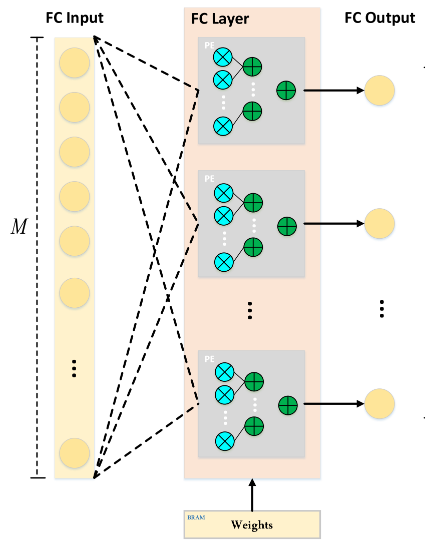


FIGURE 14. Architecture of FC layer.

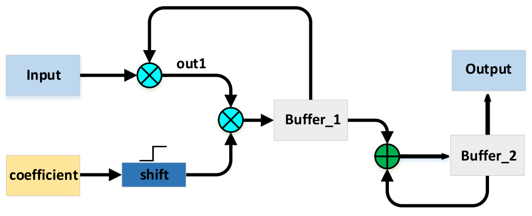


FIGURE 15. Proposed Taylor series unfolding fitting method.

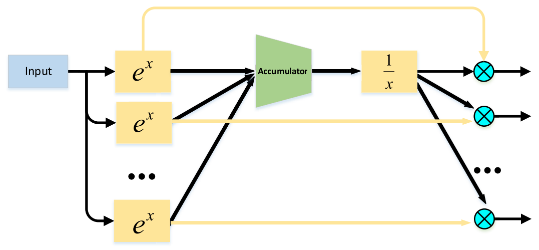


FIGURE 16. Calculation process of Softmax.

series expansions to fit the exponential function as shown below. The derivation of the formula is given in [32].

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \frac{x^6}{6!} \qquad (4)$$

In this paper, 6 coefficients used (1, 1/2, 1/3, 1/4, 1/5, 1/6) are sequentially stored in the coefficient register. When calculating, the register is shifted to the right by 16 bits (corresponding to 16 floating-point numbers), and the calculation is completed when the registers are all 0. Two buffers are used to store the multiplication result and the accumulation result. As shown in Figure 15, the output of buffer1 is the coefficients before x and the output of buffer2 is the fitting result corresponding to the input value after the operation is completed. The implementation structure is given in Figure 16.

Table 5 gives the effect of proposed polynomial approximation method for different input situations. Take the second row as an example, given 0000 (corresponding to decimal 0)

TABLE 5. Polynomial approximation verification.

Input	Output	Fitting result (float16)	Fitting result
0, 0	0.5, 0.5	3800, 3800	0.5, 0.5
0, 0.2	0.450, 0.550	3737, 3866	0.451, 0.549
0, 0.5	0.378, 0.622	360c, 38fb	0.378, 0.623
0, 1	0.269, 0.731	39d9, 344f	0.269, 0.739

and 3265 (corresponding to decimal 0.2) two sets of float16 data as input, the calculation results according to the softmax formula are about 0.450 and 0.650 respectively. The output of the softmax module designed in this paper is 3737 and 3866, which is converted to 0.451 and 0.549 in decimal system. Since we only need to compare two values to classify, the fitting works well.

IV. VERIFICATION UNDER COOPERATIVE SPECTRUM SENSING SCENARIO

In previous section, a FPGA-based computing device for the parallel CNN-LSTM has been proposed, and the implementation method of each layer has been elaborated in detail. In order to verify the proposed device, this section applies it in cognitive radio (CR) by using the device to achieve cooperative spectrum sensing(CSS).

A. BACKGROUND OF COOPERATIVE SPECTRUM SENSING

CSS is the key technology of CR. The communication scenario for centralized CSS is shown in Figure 17, which consists of a primary user (PU), N secondary users (SU) and a fusion center. Primary user, also known as licensed user, is the user who has access to transmit information within a certain frequency band during spectrum allocation. While SU, also known as unlicensed user, needs to perceive the current ''spectrum holes'' that PU is not using to dynamically access the available frequency bands. In centralized CSS, SUs use its own spectrum detection algorithm to complete local spectrum sensing and sends the sensing result to the fusion center through reporting channel. After receiving the local sensing results from all sensing nodes, the fusion center processes all perception information to obtain unified sensing results: whether there is PU signal in frequency band. In this section, we use the proposed parallel CNN-LSTM computing device as the data processing method for the fusion center.

We use simulation tools to generate dataset and divide them into training part and test part. Weight data is obtained through training on the software platform with training part and loaded into the on-chip BRAM. Then we load the test set into the BRAM as well and the result could be observed by running the device on FPGAs.

B. DATA GENERATION

We set the PU's position as the coordinate origin and set different locations for SUs. Assuming that the power of the noise and the distance between SU and PU remained unchanged during spectrum sensing. The dataset is generated using 10000 Monte Carlo methods under SNR in the range of [−20 10]db. The parameters of the dataset are shown in

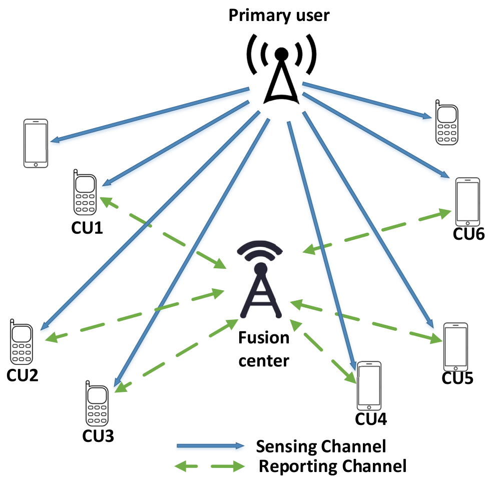


FIGURE 17. Cooperative spectrum sensing scenario for this paper.

TABLE 6. Parameters for data generation.

Modulation	BPSK
sample size	128
Number of SU	6,8,10
Samples per SNR	50000
PU power range(W)	[0 100] step5

Table 6. The noise is additive Gaussian white noise with an average value of 0 and the noise power is −143dbm/Hz. By changing the transmit power of the PU from 0 to 100 with the step of 5, different SNRs are obtained. The received signal of SUs varies according to the distance from the PU.

The generated signal is a complex data with real part and imaginary part. In order to save computing space, the signal is ABS processed before calculation. Figure 18(a) shows the waveform of the transmit signal from PU. When there is no PU signal in sensing channel, the sensing signal is given in Figure 18(b), which is the superposition of channel noise sensed by multiple SUs. When sensing channel is occupied by PU signal, the sensing result is the superposition of the PU signal perceived by SUs and channel noise as well as shown in Figure 18(c) and Figure 18(d). It can be seen that farther the SU is from the PU, the more serious the attenuation of the received signal, the smaller the received signal power, and the smaller the difference between the PU signal and the noise signal. The superimposed value of the sensing data from multiple SUs is converted to floating-point number as parallel CNN-LSTM network input after sampling.

The sensing channel is marked as '1' if it is occupied by PU signal and '0' if available. '0' and '1' are used as the network output for training. 70% of the generated dataset is used to train the network by software to obtain the weights, and the other 30% is used to test the results on FPGA. Before running the network, the test data and weights files (after training) are stored into BRAM so that CNN-LSTM can read out data by address during operation.

C. NETWORK STRUCTURE

When there are 8 SUs, fusion center will receive signal from the 8 users. The structure we select is shown in Table 7. We set signals from 8 users as the eight channels of the input feature map, CNN and LSTM will extract feature parallelly and obtain the sensing result after combining two networks. When the number of SU changes, the number of input channels and LSTM cells change accordingly and the width of input is determined by the sample size when generating dataset as shown in Table 6.

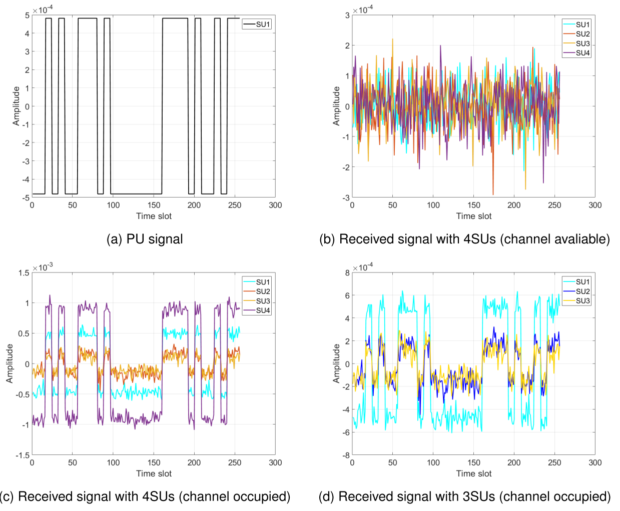


FIGURE 18. Data generation.

D. RESULT

We used Vivado 2018.3 to give a comprehensive report of our computing device. The FPGA platform used is the Xilinx XCZU9EG as given in Figure 19. The available resources on this device are shown in Table 8. Table 9 gives CSS implementation result when there are 6, 8 and 10 SUs. As the number of users increases, the network structure becomes larger, which also brings greater resource consumption. To illustrate the resource consumption in detail, we take the number of users 8 as an example. It could be seen that the maximum available quantities of BRAM, DSP, FF and LUT on the used FPGA are 588.5, 103, 8352, 56955 respectively. The utilization of on-chip BRAM reaches $588.5 \div 912 \approx 64.53\%$ consists of input, output, weights, activation function values as well as Conving Zone, which is about 2.56MB on-chip memory consumption. Set the frequency to 10Mhz, it can be seen that after loading the weights data obtained from training, the accuracy of the CSS under 3 scenarios reaches 96.32%, 96.54% and 94.86% and the project power is limited at 0.717, 0.842 and 1.744W.

Figure 20 illustrates logic and time resources consumed by individual layer with 8 SUs. Due to the large amount of data operation and activation function approximation, the convolution layer and LSTM module occupy 92.4% of the LUT consumption, 86.7% of the DSP consumption and 99.7% of the FF consumption of the overall architecture. It can be seen that with the 4-stage pipeline architecture, compared with CNN part, the LSTM part is 12.3% faster, and the resource consumption is 43.2% of the CNN part with the heavy use of LUT and matrix multiplication.

TABLE 7. Structure of parallel CNN-LSTM for CSS with 8 SUs.

Part	layer	Output shape
CNN-LSTM	Input	[1,128,8]
CNN	Conv_1[5, 5, 8]	[1, 124, 8]
	Relu_1	[1, 124, 8]
	Conv_2[9, 9, 8]	[1, 116, 8]
	Relu_2	[1, 116, 8]
	Pooling	[1, 29, 8]
	Flatten	[1, 232, 1]
LSTM	Hidden cells	8
	Flatten	[1, 8, 1]
CNN-LSTM	Stitich	[1, 240, 1]
	FC_1	[1, 64, 1]
	Sigmoid	[1, 64, 1]
	FC_2	[1, 2, 1]
	Softmax	[1, 2, 1]

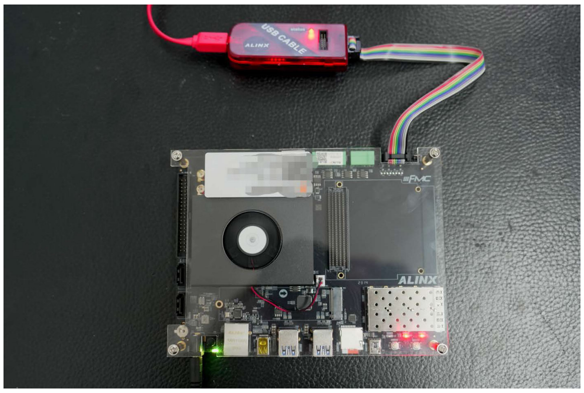


FIGURE 19. XCZU9EG paltform for this paper.

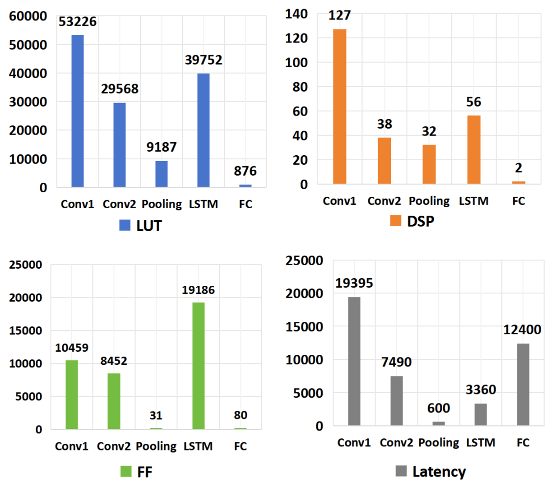


FIGURE 20. Resource consumption of different layers.

However, due to the acceleration from multi-channel high-parallelism convolution, in general, the computing speed and resource consumption of the CNN part and the LSTM part are of the same magnitude, which is convenient for subsequent operations. With the network structure shown in Table 5, when LSTM has completed calculation before the CNN, the result of the LSTM would be saved in the buffer and subsequent operations such as splicing and full

TABLE 8. Available resource on XCZU9EG.

Resources	Number Available
LUTs	274080
Flip-flops	548160
BRAM(36Kb)	912
DSP	2520
I/O	328
BUFG	404

TABLE 9. Implementation under different number of SUs on XCZU9EG.

Number of SU	6 SUs	8 SUs	10 SUs
LUT	37873(13.82%)	56955(20.78%)	63850(23.30%)
DSP	89(3.53%)	103(4.09%)	120(47.62%)
Flip-flops	5204(0.95%)	8352(1.52%)	10105(1.84%)
BRAM	490(53.73%)	588.5(64.53%)	635.5(69.68%)
Frequency	10MHz	10MHz	10MHz
Power	0.717W	0.842W	1.744W
Accuracy	96.32%	96.54%	94.86%

TABLE 10. Comparison with the resource consumption with different network structure (8 SUs) on XCZU9EG.

Structure of CNN-LSTM	Logic resource consumed		Clock resource consumed	Accuracy	Power (10MHz)
Classic	LUTs	52647	69875	89.78%	0.816W
	BRAMs	560			
	Flip-flops	8015			
Parallel	LUTs	56955	57350	96.54%	0.842W
	BRAMs	588.5			
	Flip-flops	8352			

connection would be carried out after CNN part completing its calculation.

In order to verify the superiority of the parallel CNN-LSTM architecture proposed in this paper, we connect LSTM behind CNN and compare the parallel CNN-LSTM model with the classic CNN-LSTM under the same input and output. The comparison results are shown in Table 10. It can be seen that the optimized parallel structure and classic structure are basically the same in terms of logical resource consumption. However, the clock consumption of CNN-LSTM in parallel structure is 82.1% of that of classic structure and the sensing accuracy of parallel CNN-LSTM is about 7% higher. With a classic structure, when calculation starts, the subsequent LSTM module will wait for the CNN to pass the computational data, which will have more delay than the parallel structure, while the parallel structure compresses the network depth and increases the speed of operation. At the same time, parallel computing ensures that CNN and LSTM extract different features directly from the input, which solves the problem of feature loss caused by serial computing to the greatest extent.

V. SCALABILITY AND LIMITATION

In order to verify the scalability of the computing device designed in this paper, it is clear that it is not enough to verify it only in the CSS domain and on only one FPGA platform. At the same time, due to the different types of data that CNN and LSTM are good at processing, the requirements for input are high, which will affect the scalability of this device.

TABLE 11. Resource consumption of parallel CNN-LSTM on different platforms with 8 SUs.

Platform		XCZU15EG	XCZU9EG	XQKU060	XQVU7P
LUTs	Available	341280	274080	331680	788160
	Consumed	52047(15.25%)	56955(20.78%)	56780(17.12%)	56915(7.22%)
Flip-flops	Available	682560	548160	663360	1576320
	Consumed	7533(1.10%)	8352(1.52%)	9471(1.43%)	9487(0.60%)
BRAMs	Available	745	912	1072	1440
	Consumed	587.5(78.97%)	588.5(64.53%)	588.5(54.49%)	588.5(40.87%)
DSP	Available	3528	2520	2760	4560
	Consumed	103(2.92%)	103(4.09%)	103(3.73%)	103(2.26%)
I/O	Available	204	328	520	832
	Consumed	34(16.67%)	34(10.37%)	34(6.54%)	34(4.09%)
BUFG	Available	404	404	624	1200
	Consumed	14(3.47%)	15(3.71%)	15(2.4%)	15(1.25%)

For these reasons, this chapter focuses on the scalability and limitation of this computing device.

We deploy the network structure shown in Table 7 on another 3 different FPGA platforms: XCZU15EG, XQKU060 and XQVU7P to verify the versatility of the computing device. Resource consumption of each is given in Table 11. It can be seen that the proposed computing device can be successfully deployed on different platforms. That is because the computing device designed in this paper and the acceleration ideas adopted use common FPGA components such as DRAM, BRAM, DSP, which do not adopt technologies such as off-chip storage. So it can be deployed on almost all common FPGA platforms. Depending on different tasks, the appropriate platform should be selected according to the resource consumption brought about by the network structure.

To expand the versatility of the computing devices designed in this paper, it is clear that verification in the CSS domain alone is not enough. Therefore, We select handwritten classification using the MINIST dataset to further validate the computing device. The dataset includes a total of 70,000 grayscale handwritten digital images of 28×28, of which 60,000 are the training set and 10,000 are the test set. This task implements a ten-class of handwritten data. After offline training, we adjust the network structure of the computing device according to the training model and load the weight data into BRAM. We take 100 images from the test set for classification testing. The network structure, resource consumption and classification results are shown in Table 12 and Table 13 (we also compared it with the serial structure to prove the superiority of the parallel structure). Table 13 illustrates that the classification accuracy of the parallel structure is still better than that of the serial structure in the new application scenario. Moreover, compared with the CSS scenario, the input in the handwritten scenario becomes 2D and the network scale is expanded, leading to a greater resource and power consumption. It could be seen that the computing device designed in this paper has good results under different data sets.

When it comes to the limitation, although the parallel structure leads to a better effect. However, since the LSTM module calculates the input data directly, there is no previous CNN pruning process compared to serial. At the same time, the gate operation of each cell of LSTM involves four activation functions and the corresponding matrix operation, so the demand for computational resources is greater when the network depth is high and the input sequence is long. Therefore, similar to the CSS and handwriting classification scenarios we give, the computing device designed in this paper is better at processing short time-series-based sequence or handling 2D data with smaller input sizes, but has limited processing ability for high-depth spatial datasets or long sequence issues.

TABLE 12. Structure of parallel CNN-LSTM for handwritten classification.

Part	layer	Output shape
CNN-LSTM	Input	[28,28,1]
CNN	Conv_1[3, 3, 16]	[26, 26, 16]
	Relu_1	[26, 26, 16]
	Pooling	[13, 13, 16]
	Conv_2[3, 3, 16]	[11, 11, 16]
	Relu_2	[11, 11, 16]
	Pooling	[5, 5, 16]
	Flatten	[1, 400, 1]
LSTM	Hidden cells	32
	Flatten	[1, 32, 1]
CNN-LSTM	Stitich	[1, 432, 1]
	FC_1	[1, 64, 1]
	Relu_3	[1, 64, 1]
	FC_2	[1, 10, 1]
	Softmax	[1, 10, 1]

TABLE 13. Comparison of the resource consumption on XCZU9EG for handwritten classification.

Structure of CNN-LSTM	Logic resource consumed		Clock resource consumed	Accuracy	Power (10MHz)
Classic	LUTs	98259	102339	98%	2.893W
	BRAMs	710			
	Flip-flops	10576			
Parallel	LUTs	108653	87438	100%	3.012W
	BRAMs	750			
	Flip-flops	11984			

VI. CONCLUSION

In this paper, a parallel CNN-LSTM computing device based on FPGA is proposed. The device combines the advantages of CNN and LSTM while being more suitable for hardware implementation. The device is optimized with the idea of high parallelism, pipeline and LUT-based acceleration, which is designed for different deep learning tasks. We verify the device under CSS scenario, the results show that compared with the classic CNN-LSTM, parallel structure has a higher sensing accuracy and faster computation speed. We also discuss the versatility and limitation of the device by applying device on different platforms and expanding the scenario to handwritten classification.

REFERENCES

[1] C. Zheng, C. Hu, Y. Chen, and J. Li, "A self-learning-update CNN model for semantic segmentation of remote sensing images," *IEEE Geosci. Remote Sens. Lett.*, vol. 20, pp. 1–5, 2023.

[2] S. Zhao, S. Gao, R. Wang, Y. Wang, F. Zhou, and N. Guo, "Acceleration and implementation of convolutional neural networks based on FPGA," *Digit. Signal Process.*, vol. 141, Sep. 2023, Art. no. 104188.

[3] H. L. Leka, Z. Fengli, A. T. Kenea, A. T. Tegene, P. Atandoh, and N. W. Hundera, "A hybrid CNN-LSTM model for virtual machine workload forecasting in cloud data center," in *Proc. 18th Int. Comput. Conf. Wavelet Act. Media Technol. Inf. Process. (ICCWAMTIP)*, Dec. 2021, pp. 474–478.

[4] Y. Zhang, Y. Li, B. Liang, and R. Ma, "A prediction method for fuel cell degradation based on CNN-LSTM hybrid model," in *Proc. 25th Int. Conf. Electr. Mach. Syst. (ICEMS)*, Nov. 2022, pp. 1–5.

[5] L. Wensheng, W. Kuihua, F. Liang, L. Hao, W. Yanshuo, and C. Can, "A region-level integrated energy load forecasting method based on CNN-LSTM model with user energy label differentiation," in *Proc. 5th Int. Conf. Power Renew. Energy (ICPRE)*, Sep. 2020, pp. 154–159.

[6] S.-G. Lee, G.-Y. Kim, Y.-N. Hwang, J.-Y. Kwon, and S.-M. Kim, "Adaptive undersampling and short clip-based two-stream CNN-LSTM model for surgical phase recognition on cholecystectomy videos," *Biomed. Signal Process. Control*, vol. 88, Feb. 2024, Art. no. 105637.

[7] R. Bao, Y. Zhou, and W. Jiang, "FL-CNN-LSTM: Indoor air quality prediction using fuzzy logic and CNN-LSTM model," in *Proc. 2nd Int. Conf. Electr. Eng. Control Sci.*, Dec. 2022, pp. 986–989.

[8] C. Lu, Z. Wang, Z. Wu, Y. Zheng, and Y. Liu, "Global ocean wind speed retrieval from GNSS reflectometry using CNN-LSTM network," *IEEE Trans. Geosci. Remote Sens.*, vol. 61, 2023, Art. no. 5801112, doi: 10.1109/TGRS.2023.3276173.

[9] W. Xiong, L. Han, and X. Qu, "Bus load forecasting based on maximum information coefficient and CNN-LSTM model," in *Proc. IEEE Int. Conf. Image Process. Comput. Appl. (ICIPCA)*, Aug. 2023, pp. 659–663.

[10] C. Nguyen, T. M. Hoang, and A. A. Cheema, "Channel estimation using CNN-LSTM in RIS-NOMA assisted 6G network," *IEEE Trans. Mach. Learn. Commun. Netw.*, vol. 1, no. 1, pp. 43–60, Jun. 2023.

[11] L. Li, W. Xie, and X. Zhou, "Cooperative spectrum sensing based on LSTM-CNN combination network in cognitive radio system," *IEEE Access*, vol. 11, pp. 87615–87625, 2023.

[12] H. Li, L. Gong, C. Wang, and X. Zhou, "A flexible dataflow CNN accelerator on FPGA," in *Proc. IEEE/ACM 23rd Int. Symp. Cluster, Cloud Internet Comput. Workshops*, May 2023, pp. 302–304.

[13] L. Bai, Y. Zhao, and X. Huang, "A CNN accelerator on FPGA using depthwise separable convolution," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 10, pp. 1415–1419, Oct. 2018.

[14] D. He, J. He, J. Liu, J. Yang, Q. Yan, and Y. Yang, "An FPGA-based LSTM acceleration engine for deep learning frameworks," *Electronics*, vol. 10, no. 6, p. 681, Mar. 2021.

[15] Y. Wang, Y. Liao, J. Yang, H. Wang, Y. Zhao, C. Zhang, B. Xiao, F. Xu, Y. Gao, M. Xu, and J. Zheng, "An FPGA-based online reconfigurable CNN edge computing device for object detection," *Microelectron. J.*, vol. 137, Jul. 2023, Art. no. 105805.

[16] S. Kala and S. Nalesh, "Efficient CNN accelerator on FPGA," *IETE J. Res.*, vol. 66, no. 6, pp. 733–740, Nov. 2020.

[17] A. Baba and T. Bonny, "FPGA-based parallel implementation to classify hyperspectral images by using a convolutional neural network," *Integration*, vol. 92, pp. 15–23, Sep. 2023.

[18] Y. Luo, X. Cai, J. Qi, D. Guo, and W. Che, "FPGA—Accelerated CNN for real-time plant disease identification," *SSRN Electron. J.*, vol. 207, Apr. 2023, Art. no. 107715.

[19] Y. Yang, F. Ge, D. Qiu, X. Yue, Z. Li, F. Zhou, and N. Wu, "Implementation of reconfigurable CNN-LSTM accelerator based on FPGA," in *Proc. IEEE 21st Int. Conf. Commun. Technol. (ICCT)*, Oct. 2021, pp. 1026–1030.

[20] V. R. S. Mani, A. Saravanaselvan, and N. Arumugam, "Performance comparison of CNN, QNN and BNN deep neural networks for real-time object detection using Zynq FPGA node," *Microelectron. J.*, vol. 119, Jan. 2022, Art. no. 105319.

[21] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Performance modeling for CNN inference accelerators on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 4, pp. 843–856, Apr. 2020.

[22] D. T. Kwadjo, E. N. Tchinda, J. M. Mbongue, and C. Bobda, "Towards a component-based acceleration of convolutional neural networks on FPGAs," *J. Parallel Distrib. Comput.*, vol. 167, pp. 123–135, Sep. 2022.

[23] Z. Wang, H. Li, X. Yue, and L. Meng, "Briefly analysis about CNN accelerator based on FPGA," *Proc. Comput. Sci.*, vol. 202, pp. 277–282, Jul. 2022.

[24] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.

[25] T. Yuan, W. Liu, J. Han, and F. Lombardi, "High performance CNN accelerators based on hardware and algorithm co-optimization," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 1, pp. 250–263, Jan. 2021.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[27] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision—ECCV*. Cham, Switzerland: Springer, 2014, pp. 818–833.

[28] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Phys. D, Nonlinear Phenomena*, vol. 404, Mar. 2020, Art. no. 132306.

[29] S.-N. Tang, Y.-H. Chen, Y.-W. Chang, Y.-T. Chen, and S.-H. Chou, "Hybrid CNN-LSTM network for ECG classification and its software-hardware co-design approach," in *Proc. 20th Int. SoC Design Conf. (ISOCC)*, Oct. 2023, pp. 173–174.

[30] Y. Sun, B. Liu, and X. Xu, "An OpenCL-based hybrid CNN-RNN inference accelerator on FPGA," in *Proc. Int. Conf. Field-Programmable Technol. (ICFPT)*, China, Dec. 2019, pp. 283–286.

[31] X. Dong, X. Zhu, and D. Ma, "Hardware implementation of softmax function based on piecewise LUT," in *Proc. IEEE Int. Workshop Future Comput. (IWOFC*, Dec. 2019, pp. 1–3.

[32] J. Wei, A. Kuwana, H. Kobayashi, and K. Kubo, "Divide and conquer: Floating-point exponential calculation based on Taylor-series expansion," in *Proc. IEEE 14th Int. Conf. ASIC (ASICON)*, Oct. 2021, pp. 1–4.

XIN ZHOU received the B.S. degree in information and communication engineering from the College of Information and Communication, National University of Defense Technology, Wuhan, Hubei, China, in 2021, where he is currently pursuing the M.S. degree with the College of Information and Communication. His research interests include cognitive radio, signal process, and the application of artificial intelligence technology in the field of communication.

WEI XIE received the B.S. degree in communication engineering from the College of Communication Engineering, Army Engineering of PLA University, Chongqing, China, in 1997, and the M.S. degree in communication engineering from Chongqing University, China, in 2003. From 2003 to 2012, he was a Lecturer with the College of Information and Communication, National University of Defense Technology, where he was an Associate Professor with the College of Information and Communication, from 2012 to 2019. Since 2019, he has been a Professor. His research interests include data link technology and application, digital signal processing, network planning and management, and the application of artificial intelligence in the field of communication.

HAN ZHOU received the B.S. degree in measurement and control technology and instrumentation from the School of Physics and Mechatronic Engineering, Xiamen University, in 2010, and the master's and Ph.D. degrees in instrument science and technology from the School of Intelligent Science, National University of Defense Technology, in 2012 and 2017, respectively. His research interests include cognitive radio and datalink technology.

YONGJING CHENG received the master's degree in computer application from Northeastern University, in 2013. He is currently a Lecturer with the College of Information and Communication, National University of Defense Technology. His research interests include artificial intelligence and natural language processing.

XIMING WANG received the Ph.D. degree in information and communication engineering from the School of Communication Engineering, Army Engineering University, in 2020. He is currently a Lecturer with the College of Information and Communication, National University of Defense Technology. His research interests include intelligent anti-jamming communications, cognitive radio, and multi-agent decision-making theory.

YUN REN received the Ph.D. degree in information and communication engineering from the National University of Defense Technology, in 2018. He is currently a Lecturer with the School of Information and Communication, National University of Defense Technology. His research interests include image intelligent recognition and computer vision.

SHANDONG YUAN received the master's degree from the School of Computer Science and Technology, National University of Defense Technology, in 2015, where he is currently pursuing the Ph.D. degree in command with the College of Information and Communication. His research interests include natural language processing and optimization algorithms.

LIUWEN LI received the B.E. degree in information and communication engineering from China Army Engineering University, in 2017. He is currently pursuing the M.E. degree with National Defense Technology University. His research interests include cognitive radio networks, machine learning/deep learning for wireless networks, communication signal process, and auto modulation classification.