2018 IEEE International Conference on Smart Cloud

FPGA acceleration of LSTM based on data for test flight

Zhanrui Sun
School of Microelectronics
Shanghai Jiao Tong University
Email: KUIflying@sjtu.edu.cn

Yongxin Zhu ✉
Shanghai Advanced Research Institute
Chinese Academy of Sciences
Email: zhuyongxin@sari.ac.cn
School of Microelectronics
Shanghai Jiao Tong University
Email: zhuyongxin@sjtu.edu.cn

Yu Zheng
School of Microelectronics
Shanghai Jiao Tong University
Email: sea_7seclouds@sjtu.edu.cn

Hao Wu
School of Microelectronics
Shanghai Jiao Tong University
Email: wh_sjtu@sjtu.edu.cn

Zihao Cao
School of Microelectronics
Shanghai Jiao Tong University
Email: zhcaoic@sjtu.edu.cn

Peng Xiong
School of Microelectronics
Shanghai Jiao Tong University
Email: woshixp@sjtu.edu.cn

Junjie Hou
School of Microelectronics
Shanghai Jiao Tong University
Email: hjj123@sjtu.edu.cn

Tian Huang
Department of Physics
University of Cambridge
Email: th523@cam.ac.uk

Zhiqiang Que
Department of Computing
Imperial College
Email: z.que@imperial.ac.uk

***Abstract*—Long Short-Term Memory Recurrent neural networks are generally used in speech recognition, machine translation and other fields. And LSTM-RNN also performs well in data anomaly detection. However, Due to the repeatability of LSTM-RNN, general-purpose processors such as CPU and GPGPU cannot efficiently implement LSTM-RNN, most of the existing model optimizations on FPGA are aimed at LSTM cells or large-scale model accelerations that do not require high accuracy (such as speech recognition). For the model of aircraft anomaly detection, which models with short data sampling intervals, high speed requirements and high precision requirements, the accuracy and speed of existing models are insufficient. Therefore, we proposed an FPGA-based LSTM-RNN accelerator to optimize the accuracy and speed of existing models. We achieve the optimization in the computation speed without sacrificing the accuracy, and balance performance and resources utilized in FPGA. The peak performance of our accelerator reaches 13.45 GOP/s, which is superior to other existing methods.**

***Keywords*—LSTM, RNN, FPGA, Deep Learning**

I. Introduction

In recent years, the study of artificial neutral network is rapidly emerging whose progress has drawn the attention of more and more researchers. With many successful applications in machine learning with different types of artificial neutral network such as CNN and RNN, it is easy to realize that artificial neutral network is gradually making a real difference in society and people's life. Artificial neutral network can function extensively in various machine learning domains, including prediction, classification, recognition[1] etc.

To cater for different practical problems, artificial neutral network can be divided into different categories, and recursive neutral network(RNN) is an important component among them. By taking advantage of previous outputs as inputs for current prediction, RNNs show a strong ability to learn and predict sequential data. To further improve the prediction accuracy of RNNs, Long Short-Term Memory (LSTM), a learned memory controller, is combined with standard RNN designs[2].

Since the LSTM is confirmed to be an effective model in data prediction in most cases, we make use of the LSTM model to implement the prediction in flight data. Different from other hot prediction issues in LSTM model such as speech recognition [3][4], human emotion perception[5] , question answering[6] and traffic monitoring[7], the flight data prediction puts forward requirements in two aspects. On the one hand, it acquires for high accuracy in prediction to guarantee the quality the flight monitoring; on the other hand, further indicators in prediction time should be taken into consideration to satisfy real-time processing. Besides, the flight data is in high dimension, so a delicate LSTM network with multiple layers is supposed to build up to solve the problem.

In our research, we intend to implement the LSTM prediction model and accelerate the model in high performance with FPGA. There are many research making use of FPGA to achieve hardware acceleration and their results have shown that the architecture of FPGA is appropriate for machine learning models based on neutral network[3].

LSTM-RNN is widely used in aspects such as speech recognition, machine translation, etc. which need to learn and remember a large amount of content. The computation speed by CPU for large-scale LSTM-RNN networks is slow, so the model on FPGA is proposed to accelerate large-scale LSTM-RNN networks. Among them, the reference [8] proposed an FPGA-based accelerator for a 2-layer LSTM-RNN and achieved a peak speed of 0.47 GOP/s. Then the reference [3] in speech recognition, for example, to achieve the LSTM model on the FPGA, and achieve a peak speed of 7.26GOP / s, mainly based on large-scale communications optimization and optimization of

978-1-5386-8000-1/18/$31.00 ©2018 IEEE
DOI 10.1109/SmartCloud.2018.00009

operations, which to some extent sacrifice accuracy. This paper optimizes computation and communication without sacrificing accuracy, and achieves a small-scale, high-precision LSTM model with a peak speed of 13.45 GOP/s.

The contributions of this paper can be summarized as:

1) we built up an RNN-LSTM model based on the python reference model;

2) we optimized the computation speed without sacrificing the accuracy, and reached a balance between performance and resources utilized in FPGA.

3) We finally achieved a 28.76x acceleration compared with the CPU and the peak performance of our accelerator reaches 13.45 GOP/s, which is superior to other existing methods.

II. PRELIMINARIES OF RNN AND LSTM

In this section, we will introduce some basic concepts and structure of RNN and LSTM cell.

A. RNN

Recurrent neural network (RNN) was proposed to solve the input sequence time information in 1980s. In 1993, a neural history compressor system solved a "Very Deep Learning" task that required more than 1000 subsequent layers in an RNN unfolded in time[9]. As Fig. 1 illustrates, we obviously would see an fixed network of fixed input size to fixed output size in traditional modeling activities such as Feed-forward neural network. Under the enfoldment of Feed-forward neural network, we obtain the Fig. 1.b to reveal the RNN distinguishable.

RNNs change this input dynamic to include multiple input vectors, one for each time-step, and each vector can have many columns.

For the RNN network shown in Fig. 2, given that the input sequence is X = {x_1, x_2 ... x_t}, then the corresponding time vectors of the hidden layer nodes are H = {h_1, h_2 ... h_t}, and the output nodes correspond to Y respectively. ={y_1, y_2 ... y_t}, where

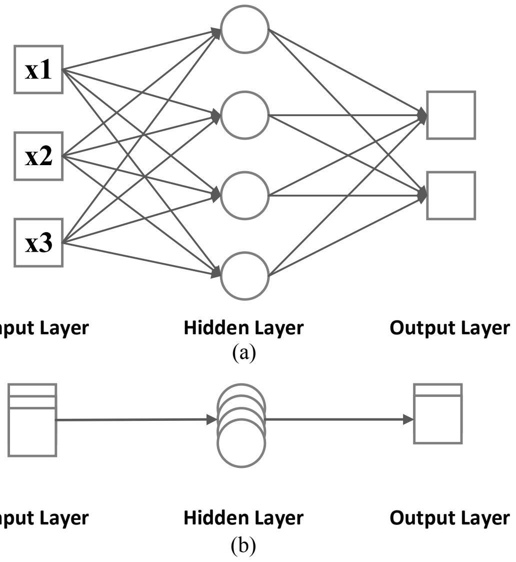

Fig. 1. Feed-forward neural network

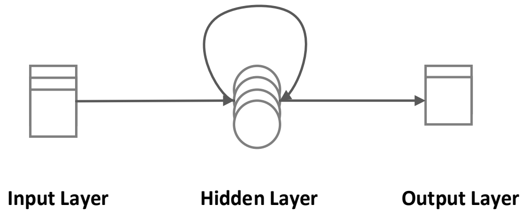

Fig. 2. Recurrent neural network(RNN)

x_i, h_i, y_i (i=1, 2 ... t) are the node vectors of the corresponding layers of the neural network.

In the ordinary Feed-forward neural network, the forward propagation formula is as follows:

$$h_t = \sigma(W_{xh}x_t + b_h) \tag{1}$$

$$y_t = W_{hy}h_t + b_y \tag{2}$$

In the RNN of Feed-forward neural network, the forward propagation formula is as follows:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{3}$$

$$y_t = W_{hy}h_t + b_y \tag{4}$$

where W_{xh} is the weight matrix between the input layer and the hidden layer, b_h is the bias vector of the hidden layer. W_{hy} is the weight matrix between the hidden layer and the output layer, b_y is the bias vector of the output layer, and W_{hh} is the weight matrix of the hidden layer from moment of the previous time to the next moment, σ is the activation function of *sigmoid()*.

Since the RNN iteratively multiplexes the results of the hidden layer, the RNN can effectively preserve the time information of the sequence. As shown in Fig. 1, the structure of the general neural network is from the input layer through the hidden layer to the output layer, and the output of each time is not related to the previous input and the subsequent input. As shown in Fig. 2, the RNN will be circularly multiplexed in the hidden layer, so it can effectively preserve the time information of the sequence, and each result will be affected by all previous inputs.

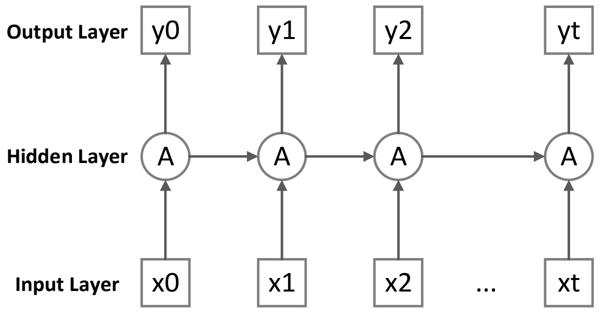

Fig. 3. The unroll of recurrent neural network

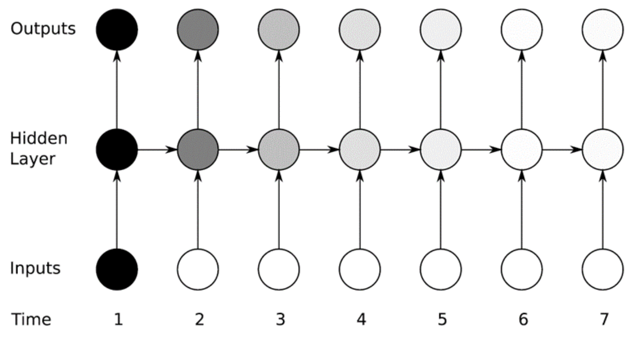

Fig. 4. The "vanishing gradient problem"

In Fig. 3, the RNN structure is expanded to explain the hidden layer multiplexing of the RNN. In addition to the first layer, the RNN will use the result of the previous hidden layer together with the new input as the input for this time, thus affecting the subsequent results and achieving the purpose of saving the time information of the sequence.

But RNNs are known to have issues with the "vanishing gradient problem"[10]. Such as Fig. 4, This issue occurs when the gradients become too large or too small and make it difficult to model long-range dependencies (10 time-steps or more) in the structure of the input dataset.

B. LSTM

To overcome the "vanishing gradient problem" of RNN, the Long Short-Term Memory(LSTM) networks with RNN architecture was proposed by Hochreiter and Schmidhuber in 1997[11].

As shown in Fig. 5, the difference between LSTM and other neural network models is its unique LSTM unit. When combined with RNN, the h_t of LSTM, which propagates in different time phases, transmits the internal memory state C_t[12]. Although there are many improved structure for LSTM cell[13][14], but it can be ignored for the Prediction Accuracy of

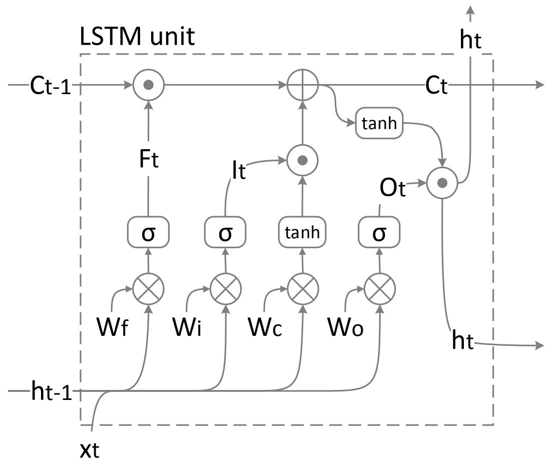

Fig. 5. The unit of Long Short-Term Memory network

the model[15]. In Fig. 5, the four parts from left to right are the forget gate, input gate, cell gate, and output gate. The input of each gate is h_{t-1} , x_t, and their corresponding weight matrix. The forget gate is responsible for selectively forgetting the previous state stored in the original cell to avoid the "vanishing gradient problem." The result of the input gate and the cell gate are multiplied and added to the previous cell C_{t-1}. Finally, to calculate the current output h_t, we use the current cell gate result and the output gate result. Specific related formulas are as follows[1][11]:

$$F_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right) \tag{5}$$

$$I_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{6}$$

$$\tilde{C}_t = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{7}$$

$$O_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{8}$$

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t \tag{9}$$

$$h_t = O_t \odot tanh(C_t) \tag{10}$$

where F_t, I_t, C_t, and O_t are the output results of the four gates respectively. W is the corresponding weight from the input to the corresponding gate. For example, W_{xf} is the weight from the input layer to the oblivious gate, and W_{hi} is hidden from the previous moment. The weights from the layer to the input gate, b_f, b_i, b_c and b_o is the bias vector of the four gates, and $\odot$ is the two vector or matrix dot multiplication. σ and *tanh()* are commonly used activation functions.

III. Hardware Design Optimization and Analysis

A. Operand estimation

The model built in this paper consists of the LSTM layer and the dense layer. Among them, the LSTM unit is the most important factor that effects the hardware operating speed. In order to better optimize and estimate performance, this paper gives a rough estimate of the operands of the entire model. Assume that the number of input nodes is N_i, the number of hidden layer nodes is N_h, the number of output layer nodes is N_o, and the model operand is N_{ops}. In reference [3], the number of operations of the LSTM single layer can be approximately estimated as follow:

$$N_{ops} = (N_i + N_h) * N_h * 4 * 2 + N_h * (8 + 4 + 4) \tag{11}$$

In this paper, no *tanh()* is approximated for the purpose of improving accuracy. At the same time, the number of output nodes is large, and operations from the hidden layer to the output layer are increased at the same time. Therefore, the estimation is performed again. As shown in Figure 5, starting from the input side, the four-W multiplication will change the vector dimension from $\mathrm{N}_i + \mathrm{N}_h$ to N_h, so $(\mathrm{N}_i + \mathrm{N}_h) * \mathrm{N}_h$ multiplications and the same number of addition are performed because The *sigmoid()* function is linearized to *hard_sigmoid()*[16](This will be discussed in the next section), so the sum of the four activation functions and the bias vector amounts to $N_h * (3 + 3 + 6 + 3)$. The next three points multiply and add total $N_h * 4$, the activation function *tanh()* is $N_h * 6$, and from ht to the final output level $N_h * N_o * 2 + N_o$. The number of operations of the entire model can be approximately estimated as follow:

$$N_{ops} = (N_i + N_h) * N_h * 4 * 2 + N_h * (3 + 3 + 6 + 3 + 4 + 6) + N_h * N_o * 2 + N_o \qquad (12)$$

B. *Loop computation optimization*

Obviously each logic gate is internally matrix multiplied, so *Pipeline* and *Unroll* are very effective methods for reducing the model latency[17]. Taking into account the computational resource constraints of the hardware platform, we unroll all loops in Fig. 6, the loop is collapsed into many small parts, and each part performs a small block of calculations. For top-level functions, we use the *Pipeline* to speed up and avoiding excessive hardware reads and writes. At the same time, the tile based on the unfolding direction is performed for the input output unit and the memory unit corresponding to each gate inside. However, since full expansion will result in an instance of more than 900 floating-point multiplications and floating-point additions, it will occupy a large amount of hardware space. Therefore, the function for cell unit is only expanded, and the I_t and F_t with less calls are not tiled.

The main influence on latency is the calculation of two activation functions of *sigmoid()* and *tanh()*. The operations of the activation function are exponential and floating-point division, so they consume a lot of hardware resource for implementation. The activation function *sigmoid()* can generally be replaced with *hard_sigmoid()*. The corresponding calculation formula is as follows:

$$y = sigmoid(x) = \frac{e^x}{e^x + 1} \qquad (13)$$

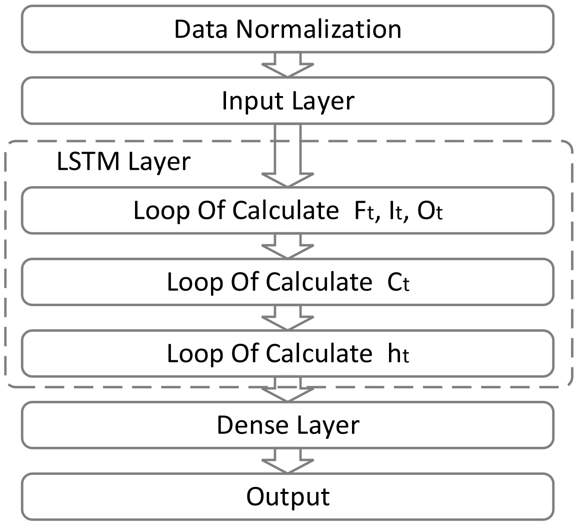

Fig. 6. Flow chart of the model implementation

$$y = hard\_sigmoid(x) = \begin{cases} 1 & (x \geq 2.5) \\ 0.2x + 0.5 & (-2.5 < x < 2.5) \\ 0 & (x \leq -2.5) \end{cases} \qquad (14)$$

Because the activation function provided in the model software can use *hard_sigmoid()*[13], this activation function can be directly used during training of the model, and the average error rate brought by our activation function is zero.

The activation function *tanh()* is generally considered to be replaced with *hard_tanh()*, and the two functions are calculated as follows:

$$y = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad (15)$$

$$y = hard\_tanh(x) = \begin{cases} 1 & (x \geq 1) \\ x & (-1 < x < 1) \\ -1 & (x \leq -1) \end{cases} \qquad (16)$$

However, the *hard_tanh()* activation function is not provided in the software model, so the use of *hard_tanh()* for replacement needs to consider the possible increase in the error caused by the results. Because the model in this paper is not a relatively non-demanding model for accuracy requirements such as speech recognition, but for aircraft anomaly detection, the acceptable fluctuation range of many data is only 5%, so accuracy is very important. This article counts 132,925 data in the model results, and the average data change brought by it is 8.5%. The impact on the data is relatively large, so we can only give up this alternative optimization.

IV. IMPLEMENTATION

In this section, we will first briefly introduce the software model framework, and then introduce the implementation of the hardware part.

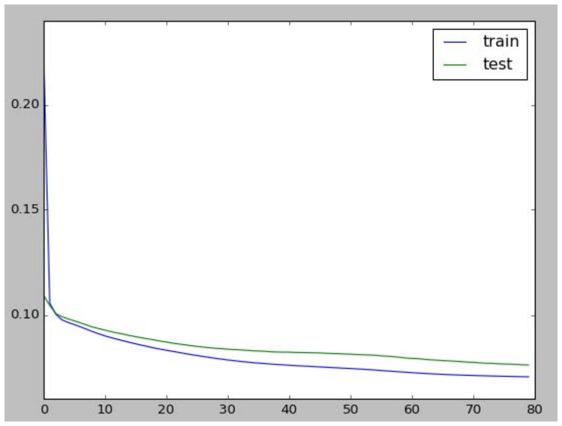

Fig. 7. The loss of the training set and test set

A. Software model

The software model is built on the basis of python 3.5 and anaconda3 platform, and the time-based forecasting model is constructed through the LSTM function provided by *Keras*. The number of input layer nodes is 19, the number of hidden layer nodes is 10, and the number of output layer nodes is 19. The entire model consists of an LSTM layer and a dense layer. The activation function is *hard_sigmoid()* and the recurrent activation function is *tanh()*.

The mean squared error is 0.098, and the loss of the training set and test set is shown in Fig. 7.

B. Hardware implementation

The structure of the LSTM in this paper is shown in Fig. 8. The on-chip data is divided into two groups: *input* and *output*. The input is called in parallel by the four gate modules, so the input and output are tiled and optimized. Forget gates, input gates, unit gates, and output gates are gated to get data from *input* and perform calculations. Since multiple groups of data are input at the same time, the gate module must perform matrix multiplication of three cycles. Therefore, the data communication time of the multiple-used *input* x_{in} and the hidden layer state h_{t-1} must be taken into consideration, and the loop is selected to perform the pipeline and unroll.

At the same time, for the internal array that is used multiple times, the corresponding reshape and segmentation are performed, and the data is transferred to the gate module in parallel for calculation. We have parallelized the multiplication and addition in the gate module. The two *tanh()* activation functions use overmuch hardware resources because of exponent and floating-point division, so only one is parallelized and the other is partially parallel optimized. The output of the gate module enters the LSTM functional logic to perform the remaining matrix dot multiplication, matrix addition, and *tanh()* activation. The LSTM functional logic takes the operation result h_t of the hidden layer as an output, performs weighting and dimension conversion through the dense layer, and finally acts as a result of the output layer.

V. Evaluation Results and Discussion

A. Data and model

Based on the data of COMAC's test flight, we implemented a data prediction LSTM model using PYTHON 3.5 and implemented the trained model on the FPGA.

The input of the model is 19-dimensional, and the data is the result of normalization from the original data. Part of the original data is shown in TABLE I. The number of model input nodes is 19, the number of hidden layer nodes is 10, and the number of output layer nodes is 19.

B. Comparison with Software

The model of this paper is built on a hardware simulation software vivado HLS (v2015.4), using Xilinx's Virtex-7 VC707 FPGA, based on the parameters previously used PYTHON training to build the model. In order to verify the accelerated performance of the FPGA model, the software code is implemented on the Intel Core i7-6700HQ CPU. The CPU

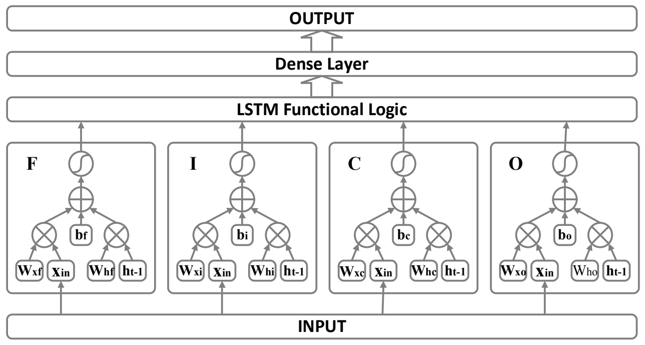

Fig. 8. The structure of the LSTM

TABLE I. Part of the original data

Time	VIB1	VIB2	VIB3	VIB4	VIB5	VIB6	VIB7	VIB8
06:41:19:051756	1.0227	0.844	-0.78186	1.5313	-1.5852	-0.4048	-0.6645	-0.0189
06:41:19:051878	1.0227	0.8458	-0.78866	1.5551	-1.5836	-0.376	-0.6152	-0.0223
06:41:19:052000	1.0227	0.8458	-0.79036	1.567	-1.5804	-0.3808	-0.6152	-0.0223
06:41:19:052122	1.021	0.8458	-0.79036	1.55	-1.5836	-0.416	-0.6645	-0.0223
06:41:19:052244	1.0193	0.844	-0.79036	1.5449	-1.5852	-0.44	-0.6373	-0.0206
06:41:19:052366	1.021	0.8404	-0.79376	1.55	-1.5772	-0.4368	-0.6118	-0.0189
06:41:19:052488	1.021	0.8386	-0.80056	1.5449	-1.5788	-0.4384	-0.6713	-0.0155
06:41:19:052610	1.0227	0.8476	-0.79376	1.5483	-1.582	-0.464	-0.6679	-0.0189
06:41:19:052732	1.0261	0.862	-0.77676	1.5636	-1.5804	-0.4832	-0.6152	-0.0308
06:41:19:052854	1.0261	0.8674	-0.77166	1.5534	-1.5836	-0.4816	-0.6577	-0.0325

TABLE II. Performance Comparison with Software

	CPU(ms)	FPGA(ms)
LSTM ex1	0.06858	0.002725
LSTM ex2	0.1338	0.004311
total	0.20238	0.007036
speedup	1.00 x	28.76 x

operating frequency is 2.6GHz. The result is shown in TABLE II.

As shown in TABLE II. , our FPGA implementation is 28.76x faster than CPU. While observing TABLE I, the sampling interval of the data is only about 0.12ms. Therefore, no acceleration may cause a problem that the new data arrived before the previous data finish the calculation. The FPGA of this article only uses 0.0043ms, which can effectively avoid the data accumulation problem.

C. Calculated performance

In the reference [8], the estimation of the LSTM structure operands and the evaluation criteria of the model calculation performance (operand/time) are proposed. The LSTM model has been further improved and optimized in reference [3]. For a fair comparison, we have listed the runtime frequency, the total number of operations in each model, and overall performance. We also use VC707 FPGA, while setting the clock frequency to

TABLE III. Comparison with Others Implementation

	Ref.[8]	Ref.[3]	Ours
Frequency	142 MHz	150 MHz	150 MHz
Model Size	0.48 MOP	2.76 MOP	2.90KOP
Performance	0.47 GOP/S[a] 3.78 GOP/S[b]	7.26GOP/S[a] 10.67GOP/S[b]	10.64 GOP/S[10] 13.45 GOP/S[20]

[a] implemented

[b] predicted

[10] $N_{sample} = 10$

[20] $N_{sample} = 20$

TABLE IV. Resource Utilization of 10 input

Resource	BRAM_18K	DSP48E	FF	LUT
Used	24	1508	192477	235669
Available	2060	2800	607200	303600
Utilization	1	53	31	77

TABLE V. Resource used of division and exponential

Resource	BRAM_18K	DSP48E	FF	LUT
FL-division	0	0	555	810
Exponential	0	26	1219	2595
Available	2060	2800	607200	303600

TABLE VI. Resource Utilization of 20 input

Resource	BRAM_18K	DSP48E	FF	LUT
Used	6	1730	194318	285558
Available	2060	2800	607200	303600
Utilization	0	61	32	94

150MHz. After comparing in TABLE III. , we found that each batch of 10 data inputs can be increased by 1.47x. Each batch of 20 data inputs can be increased by 1.85x, but each batch of 20 data inputs, the occupancy of data input is slightly higher.

TABLE IV. and TABLE V. is the resource utilization of the model. Compared to the reference [3], the number of DSP, FF and LUT usages has increased significantly, because in order to improve the model accuracy, *tanh()* is not linearized, so the model will increase the N_{sample} instance of floating-point division and exponential. The resources it occupies are shown in TABLE V. Therefore, after removing the instance of this part, resource utilization is similar to that of the reference [3].

VI. Conclusions and Future Work

In this paper we presented an optimized hardware design and implementation of an ultra-fast small LSTM model to meet the forecasting needs of aircraft test flight data. We implemented the LSTM-RNN model based on FPGA and optimized it in terms of resources and speed. The final simulation results show that our model has accelerated by 28.76x and reached a speed of 13.45GOP/s.

In the future research, we plan to further improve acceleration and model accuracy.

VII. Acknowledgment

This work was supported in part by a grant (175111105000) from Shanghai Municipal Science and Technology Commission and Commercial Aircraft Corporation of China, Ltd. (COMAC).

This research is partially sponsored by National Key Research & Development Program of China (2016YFE0100600).

References

[1] Han S, Kang J, Mao H, et al. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA[J]. 2017.

[2] Gers F A, Schmidhuber J, Cummins F. Learning to Forget: Continual Prediction with LSTM. Neural Computation 12(10): 2451-2471[J]. Neural Computation, 2000, 12(10):2451-2471.

[3] Guan Y, Yuan Z, Sun G, et al. FPGA-based accelerator for long short-term memory recurrent neural networks[C]// Design Automation Conference. IEEE, 2017:629-634.

[4] Sundermeyer M, Schlüter R, Ney H. LSTM Neural Networks for Language Modeling[C]// Interspeech. 2012:601-608.

[5] Ringeval F, Eyben F, Kroupi E, et al. Prediction of asynchronous dimensional emotion ratings from audiovisual and physiological data[J]. Pattern Recognition Letters, 2015, 66(C):22-30.

[6] Chen Q, Zhu X, Ling Z H, et al. Enhanced LSTM for Natural Language Inference[C]// Meeting of the Association for Computational Linguistics. 2017:1657-1668.

[7] Ma X, Tao Z, Wang Y, et al. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data[J]. Transportation Research Part C, 2015, 54:187-197.

[8] Chang A X M, Martini B, Culurciello E. Recurrent Neural Networks Hardware Implementation on FPGA[J]. Computer Science, 2015.

[9] Jürgen Schmidhuber. Netzwerkarchitekturen, Zielfunktionen und Kettenregel[J]. 1993.

[10] Pascanu R, Mikolov T, Bengio Y. On the difficulty of training recurrent neural networks[C]// International Conference on International Conference on Machine Learning. JMLR.org, 2013:III-1310.

[11] Hochreiter, Sepp; Schmidhuber, Jürgen. Long Short-Term Memory[J]. Neural Computation, 1997, 9(8):1735-1780.

[12] Ferreira J C, Fonseca J. An FPGA implementation of a long short-term memory neural network[C]// International Conference on Reconfigurable Computing and Fpgas. IEEE, 2017:1-8.

[13] Otte S, Liwicki M, Zell A. Dynamic Cortex Memory: Enhancing Recurrent Neural Networks for Gradient-Based Sequence Learning[C]// International Conference on Artificial Neural Networks. Springer, Cham, 2014:1-8.

[14] Sak H, Senior A, Beaufays F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling[J]. Computer Science, 2014:338-342.

[15] Greff K, Srivastava R K, Koutník J, et al. LSTM: A Search Space Odyssey[J]. IEEE Transactions on Neural Networks & Learning Systems, 2015, 28(10):2222-2232.

[16] Amin H, Curtis K M, Hayes-Gill B R. Piecewise linear approximation applied to nonlinear function of a neural network[J]. IEE Proceedings - Circuits, Devices and Systems, 1997, 144(6):313-317

[17] Gai K, Qiu M, Zhao H. Cost-Aware Multimedia Data Allocation for Heterogeneous Memory Using Genetic Algorithm in Cloud Computing[J]. IEEE Transactions on Cloud Computing, 2016, PP(99):1-1.