International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

Simulation and High Level Synthesis of Gated Recurrent Unit for Field Programmable Gate Array

Neil Derick
PG Student,
Electronics and Communication Engineering Dept.
TKM Institute of Technology
Kerala, India

Lakshmy G B
Head of the Dept., Electronics and Communication Engineering Dept.
TKM Institute of Technology
Kerala, India

***Abstract*—Data like biometric signals, weather monitoring, stock prices etc. are sequential, patterned and based on time. Accurate predictions on time series data helps in understanding the future outcome of a given data. Gated Recurrent Unit (GRU) is a type of neural network that can be used to predict time series data. GRU is usually implemented in devices with higher computing resources like a CPU or GPU. Hence, GRU cannot be used with devices having low computing resources (like microcontrollers or embedded computers). Implementing GRU on Field Programmable Gate Array (FPGA) would allow devices having low computing resources to use GRU. However, FPGA implementation of any such network should meet with the requirement of optimum resource utilization, speed, and accuracy. GRU was designed in C++ with pre-trained weights obtained from PyTorch. High Level Synthesis for two different GRU architectures were done - one-to-one and many-to-one. This gave an insight on the trade-off between resource utilization, speed, and accuracy for both the architectures. One-to-one architecture had lower resource utilization and provided outputs at faster rate than many-to-one architecture. However, many-to-one architecture was found to have better average accuracy than one-to-one architecture.**

Keywords— High Level Synthesis, Gated Recurrent Unit, Field Programmable Gate Array, One-To-One GRU Architecture, Many-To-One GRU Architecture

I. INTRODUCTION

Data like biometric signals, weather monitoring, stock prices etc. are sequential, patterned and based on time. An accurate prediction on time series data helps in understanding the future outcome of a given data. For such kind of prediction Recurrent Neural Networks (RNN) are used. Recurrent neural networks recognize data's sequential characteristics and use patterns to predict the next likely scenario. Vanishing gradient problem is a major drawback of RNN. So, Long Short-Term Memory (LSTM) is the most widely used recurrent neural network representation for modelling sequential data. LSTM has feedback connections, unlike conventional feed-forward neural networks, i.e., it can process the entire sequence of data, apart from single data points such as images [6]. This finds application in speech recognition, machine translation, etc. LSTM shows outstanding performance on a large variety of problems. However, it is assumed an expensive modelling architecture as it requires many hardware components to be implemented, which is inefficient in solving problems based on small datasets. GRU is like LSTM but uses a lesser number of parameters. So, it is faster. GRU is found to be best suitable for applications having fewer datasets. RNN and similar networks (LSTM and GRU) are usually implemented using software and used in personal or super computers. Thus, they are limited to devices with high performance specifications. Hence, it becomes difficult to use RNN or similar networks in applications that use devices having lower computing resources. Field Programmable Gate Arrays (FPGA) are semiconductor devices that can be used to implement digital circuit. They are faster than a CPU for executing programs specific to the desired application. Hence, implementing RNN, LSTM or GRU in an FPGA can allow these networks to be used with low end devices [3][4][9]. Optimum resource utilization and speed are one of the important factors to be considered while implementing such networks on FPGA. There is always a trade-off between resource utilization and speed. So, it is up to the designer on how the design can be optimized.

II. BACKGROUND

Devices used for embedded applications often have limited hardware resources. Using GRU along with such devices becomes difficult. This difficulty is due to the latency and memory utilization of the network. Hence, GRU is usually implemented in devices having higher computing capabilities like a high-end CPU or GPU. Evolution of FPGA has helped with implementing complex digital circuits in an easier manner. Furthermore, with the help of High Level Synthesis (HLS) one can easily implement algorithmic structures (like FFT or neural networks) into digital circuit by writing C or C++ code. Implementing GRU in an FPGA will allow GRU and similar networks to be used with devices with low computing resources.

A. Gated Recurrent Unit

The Gated Recurrent Unit (GRU) is a type of RNN. GRU like LSTM, aims to solve the vanishing gradient problem which comes with a standard RNN. GRU has lesser number of parameters than LSTM. A GRU cell has an update gate and a reset gate. Basically, these are two vectors which decide what information should be passed to the output. GRU is best suitable for application with fewer of dataset.

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{ir}) \quad (1)$$

r_t is the reset gate.

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hr}h_{t-1} + b_{iz}) \quad (2)$$

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

z_t is the update gate.

$$n_t = tanh(W_{in}x_t + b_{in} + r_t \otimes (W_{hr}h_{t-1} + b_{ir})) \quad (3)$$

n_t is the memory content.

$$h_t = (1 - z) \otimes n_t + z_t \otimes h_{t-1} \quad (4)$$

h_t is the output. Input to the GRU is denoted as x_t. h_t also serves as an input to the next layer if there exist multiple layers. σ is used to represent sigmoid activation function. Other activation function used is tanh.

B. High Level Synthesis

Algorithms are more difficult to be expressed by using Hardware Description Languages (HDL). High Level Syn thesis (HLS) enables algorithms to be expressed more easily, with faster development time. HLS allows the designer to use C or C++ to implement algorithms as a digital circuit. GRU is defined as an algorithm, so it can be implemented using C++. This effectively reduces the time to design the network using an HDL. HLS tool runs the RTL implementation along with code optimization. The designer has better control over how the design should be optimized. HLS would give an insight into the resource utilization and latency of the GRU network. HLS tool generates IP core for the synthesized code. This IP core can later be used for FPGA implementation.

III. HIGH LEVEL SYNTHESIS OF GATED RECURRENT UNIT

A feed-forward GRU network is used for synthesis and simulation. The weights should be obtained from a pretrained network. Pre-training can be done with PyTorch. PyTorch is an open-source machine learning framework. PyTorch comes with GRU model that can be used to create a two-layer network. After training, the trained weights can be obtained. These weights correspond to the equations 1, 2, 3 and 4. High Level Synthesis can be done for two different architectures - One-to-one and Many-to-one. A feed-forward GRU network is created using C++. Figure 1 shows the proposed GRU network. The pre-trained weights obtained from PyTorch GRU network is added to the feed-forward network in C++. Activation functions can be implemented in C++ by two methods. One is by using the functions available in the built-in math library. Second method is by directly framing each component that makes up the activation function into the C++ code. The two approaches have different results in terms of resource utilization and speed.

Generalized equations are:

$$tanh(x) = (e^x - e^{-x} / e^x + e^{-x}) \quad (5)$$

$$\sigma(x) = 1 / (1 + e^{-x}) \quad (6)$$

Equations 5 and 6, both utilize exponential function e^x in generating the output. $e\ x$ can also be generalized.

$$e^x = 1 + x + x^2/2 + x^3/6 + ... \quad (7)$$

A. One-to-One architecture

One-to-one architecture has a single input and a single output. The layers are arranged in such a way that the output from one layer forms the input for the next layer. Here, as shown in figure 1(a) output of GRU layer 1 forms the input for GRU layer 2. Finally, the output is given by a linear layer. Since, there are two GRU layers, pipelining can be applied to optimize performance.

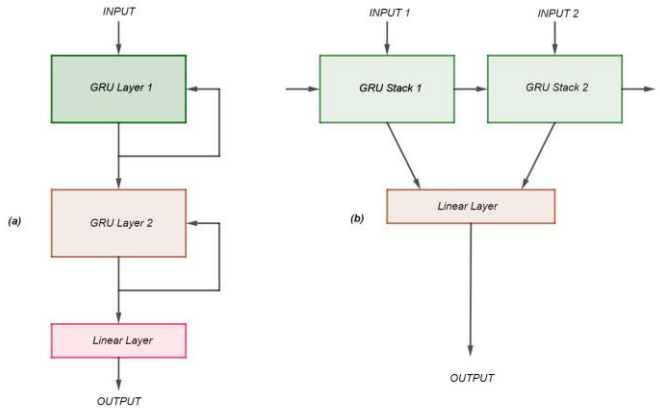

Figure 1. Proposed GRU networks. (a) One to One Architecture with 2 GRU layer. (b) Many to One Architecture with a single GRU layer

1) Pipelined GRU network

A pipelined architecture always provides better performance that a non-pipelined architecture. HLS tool can be used to implement pipelined architecture of the designed GRU network. The synthesized network has clock intervals 111 cycles for layer 1, 105 cycles for layer 2 and 5 cycles for the final linear layer. The total time interval would be 221 clock cycles. Figure 2 shows non-pipelined GRU network. Once an input is given the output is obtained after a time interval of 221 clock cycles. In this type of architecture, there is a need to wait for an interval of 221 clock cycles to give the next input. Figure 3 shows a pipelined GRU network. Considering an initiation interval (II) for pipelining, input can be given at every interval equal to II and can expect an output after an interval equal to II. Thus, the pipelined network provides output at a faster rate as compared to the non-pipelined network. Figure 3 shows an II of 105 clock cycles. Thus, input can be given at every 105 clock cycles and expecting an output after every 105 clock cycles.

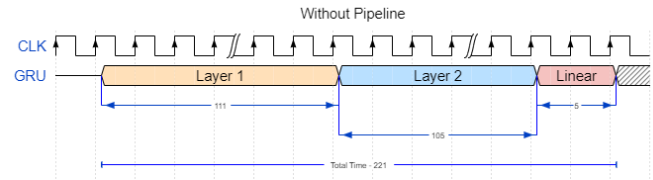

Figure 2. Non-Pipelined Network

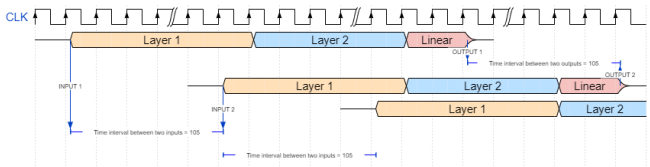

Figure 3. Pipelined Network

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

B. Many-to-One architecture

Many-to-One architecture can have multiple inputs (and also multiple GRU layers) and a single output. Here, as in figure 1(b) two input GRU is used. The two outputs appearing are combined using a linear layer. For such architecture weights are no more 1 dimensional (as in section 3-A), but 2 dimensional matrices. Hence, each weight corresponding to equations 1, 2, 3 and 4 are 2x2 matrices. Hence, for computation matrix multiplication must be done. This can use up more resources as compared to one-to-one architecture. Pipelining was not done as there is only a single layer. Although, a more optimized code can provide better results.

IV. RESULTS

The FPGA based GRU network (GRU-FPGA) should be able to predict output based on the given input (or inputs). A comparison between the output of GRU-FPGA and GRU-PyTorch gave the accuracy of the proposed networks. For this a simulation testbench was created to simulate the GRU-FPGA network against test inputs [1]. Simulation and synthesis of the two GRU networks as in figure 1 gave a detailed comparison of resource utilization and speed of computation as discussed in section 3. Simulation provided timing analysis and synthesis provided resource utilization of the proposed network. Simulation was done considering a system clock of 10ns.

A. Resource utilization

Resource utilization is different for both the proposed architectures (one-to-one and many-to-one). Many-to-one architecture consumed more resources than one-to-one architecture. Table I shows a comparison in resource utilization when one-to-one architecture is synthesized using built-in math library and general equation (Eq 5, 6 and 7). Table III also shows the resource utilization when using built-in math library and general equation for many-to-one architecture. When using built-in math library, more resources were utilized than implementation using general equation. Comparing table 1 and table 2, resource utilization was significantly larger for many-to-one architecture. Realizing the activation functions using built-in math library utilized more resources, while implementation using general equations 5, 6 and 7, the resource utilization got reduced.

1) Resource utilization in pipelined network (One-to-One)

Pipelining was applied to the one-to-one GRU network with activation functions realized using general equation (because of lesser utilization). There is a further reduction in resource utilization when pipelining (as in section 3-A.1) the network. Table III shows the comparison in resource utilization for non-pipelined and different pipelined networks. Without pipeline, the utilization remains the same. However, with a pipeline interval of 111 clock cycles, resource utilization is less than non-pipelined network. With a pipeline interval of 105 clock cycles, resource utilization gets further reduced.

TABLE I.

RESOURCE UTILIZATION (ONE-TO-ONE) - MATH LIBRARY VS GENERAL EQUATION.			
	DSP	***Flip-Flops***	***LUTs***
Built-in Math	27	2326	3322
General Equation	15	2313	3215

TABLE II.

RESOURCE UTILIZATION (MANY-TO-ONE) - MATH LIBRARY VS GENERAL EQUATION.				
	DSP	***Flip-Flops***	***LUTs***	***BRAM***
Built-in Math	104	12497	15907	7
General Equation	20	8073	11196	1

TABLE III.

RESOURCE UTILIZATION (ONE-TO-ONE) - NON-PIPELINED VS PIPELINED			
	DSP	***Flip-Flops***	***LUTs***
Non-pipelined	15	2313	3215
Pipelined - II 105	5	1612	2491
Pipelined - II 111	5	1650	2440

B. Time Analysis

Time analysis for the network was done same as in section 4.1. One-to-one architecture with built-in math library yields faster result (1550ns) for a single computation as compared to generalized mathematical functions (2200ns). Similarly, many-to-one with built-in math library yields faster result (3310ns) for a single computation as compared to generalized mathematical functions (3450ns). This can be seen as a trade-off between resource utilization and speed. Time analysis for one-to-one pipelined architecture was done using test bench containing test input dataset. The output of the simulation was used to calculate the accuracy of the network.

1) Non-Pipelined vs Pipelined Network (One-to-One)

The output appeared at a faster rate than a non-pipelined network as in figure 3. For getting the total computation time, 19 consecutive test inputs were given to the non-pipelined and pipelined network. It was observed that the non-pipelined network took 39.8μs for providing all the inputs whereas, it took 21.2μs for a pipelined network of II 111 and 20.2μs for pipelined network of II 105. Hence, a pipelined network provided all the outputs in a faster rate for the corresponding inputs. Noting the time taken for many-to-one architecture with the same test inputs took 59.8μs when using built-in math library and 62.3μs when using general equation.

C. Accuracy

Resource budgeting and speed optimization of GRU-FPGA along with good prediction accuracy is need for a usable FPGA architecture. Hence, the output of GRU-FPGA was compared against GRU-PyTorch. Prediction for one-to-one architecture using GRU-PyTorch had an average accuracy of 98% and GRU-FPGA had 97% accuracy. Prediction for many-to-one architecture using GRU-PyTorch and GRU-FPGA had an average accuracy of 98%.

V. CONCLUSION

High Level Synthesis and simulation of one-to-one and many-to-one GRU architecture was done to realize resource utilization, speed, and accuracy of prediction for FPGA implementation. Techniques for resource budgeting includes implementing activation functions using built-in math library and as a general equation. Implementation using general equation gave lesser resource utilization but more time for

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

prediction. For one-to-one architecture pipeline interval of 105 clock cycles gave lesser resource utilization and with results appearing at faster rate. Many-to-one architecture had more resource utilization and lesser speed than one-to-one architecture. But, many-to-one architecture (98% average accuracy) was more accurate than one-to-one architecture (97% average accuracy). Therefore, from the results obtained it can be concluded that, there is always a trade-off between resource utilization, speed, and accuracy. If one is looking for accuracy, many-to-one architecture seems to be the choice.

REFERENCES

[1] Neil Derick, "Dataset and simulation results of hls," 2023, mendeley Data, V1. [Online]. Available: https://dx.doi.org/10.17632/hv8b5yjtzb.

[2] B. James Romaine and M. P. Mart ´ın, "High-throughput low power area efficient 17-bit 2's complement multilayer perceptron components and architecture for on-chip machine learning in implantable devices," IEEE Access, vol. 10, pp. 92 516–92 531, 2022.

[3] Z. S. Zaghloul and N. Elsayed, "The fpga hardware implementation of the gated recurrent unit architecture," in SoutheastCon 2021, 2021, pp. 1–5.

[4] U. Yoshimura, T. Inoue, A. Tsuchiya, and K. Kishine, "Implementation of low-energy lstm with parallel and pipelined algorithm in small-scale fpga," in 2021 International Conference on Electronics, Information, and Communication (ICEIC), 2021, pp. 1–4.

[5] S. Bouguezzi, H. Faiedh, and C. Souani, "Hardware implementation of tanh exponential activation function using fpga," in 2021 18th International Multi-Conference on Systems, Signals & Devices (SSD), 2021, pp. 1020–1025.

[6] S. Saadatnejad, M. Oveisi, and M. Hashemi, "Lstm-based ecg classification for continuous monitoring on personal wearable devices," IEEE Journal of Biomedical and Health Informatics, vol. 24, no. 2, pp. 515– 523, 2020.

[7] S. Saadatnejad, M. Oveisi, and M. Hashemi, "Lstm-based ecg classifi cation for continuous monitoring on personal wearable devices," IEEE Journal of Biomedical and Health Informatics, vol. 24, no. 2, pp. 515– 523, 2020.

[8] A. E. Syahrulanuar Ngah, Rohani Abu Bakar and S. Razali, "Two-steps implementation of sigmoid function for artificial neural network in field programmable gate array," April 2016.

[9] A. X. M. C. et al., "Recurrent neural networks hardware implementation on fpga," November 2015.