A Hardware-Oriented Random Number Generation Method and A Verification System for FPGA

Sansei Hori, Hakaru Tamukoh
Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology,
2-4 Hibikino, Wakamatsu, Kitakyushu, Fukuoka, 808-0196, Japan
E-mail: hori-sansei@edu.brain.kyutech.ac.jp, tamukoh@brain.kyutech.ac.jp
https://www.kyutech.ac.jp

Abstract

Deep learning technology has made remarkable progress in recent years and has been applied to a variety of applications such as smartphones and cloud servers. These systems employ dedicated processors to save power consumptions and process massive data. In this paper, we introduce a hardware-oriented restricted Boltzmann machine and propose a field-programmable gate array (FPGA) infrastructure for easy verification of user circuits. The infrastructure makes it easy to communicate and control between the host PC and the user circuit.

Keywords: FPGA, Hardware Accelerator, Xillybus, Deep Learning, Random Number Generator, RBM

1. Introduction

Recent days, neural network technologies such as deep learning[1] have been utilized many applications such as image processing and natural language processing. Especially, these technologies have been actively applied to various embedded fields, including smartphone applications.

However, most training of deep neural networks (DNNs) require a massive amount of calculation resources and are often performed on high-performance computers with GPUs[2]. On the other hand, some embedded systems have restrictions such as power consumption and physical size to implement high-performance computers. Therefore, application-specific integrated circuits (ASIC) and system-on-a-chip (SoC) dedicated DNNs[3,4] is actively developed to accelerate the processing and reduce power consumptions. These some of the technologies have already yield practical applications.

Field-programmable gate arrays (FPGAs)[5] can also be used to create dedicated circuits and can be rewritten, making it possible to build more general-purpose systems. We have proposed resource-saving hardware implementation of a restricted Boltzmann machine (RBM)[6,7], which is a building block of Deep Belief Networks. However, the hardware implementation of the user circuit on an FPGA is costly work. In this paper, we introduce the hardware-oriented RBM, and we propose an FPGA infrastructure for verifying the modules implemented as IP in FPGA. Also, to verify the operation of the proposed FPGA infrastructure, we implemented a hardware circuit for learning an RBM[8,9] and confirmed that the circuit could be controlled from a host computer.

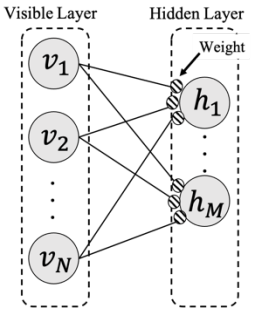

Fig. 1. Restricted Boltzmann machine.

2. Restricted Boltzmann machine

Restricted Boltzmann machine (RBM) is one of the generative model and a part of the element structures DNNs. An RBM has two layers called visible layer and hidden layer, as shown in figure 1. The probability distribution of an RBM calculated by Eq. (1), where $\boldsymbol{v}$ and $\boldsymbol{h}$ represent the states of visible and hidden units, and $\boldsymbol{\theta}$ is a set of the network parameters. Z($\boldsymbol{\theta}$) shown in Eq. (2) is a normalized term, and Φ shown in Eq. (3) is an energy function, where a_i and b_j represent the biases of visible and hidden units, and w_{ij} is the weight.

$$p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} e^{-\Phi(\boldsymbol{v},\boldsymbol{h},\boldsymbol{\theta})}. \quad (1)$$

$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-\Phi(\boldsymbol{v},\boldsymbol{h},\boldsymbol{\theta})}. \quad (2)$$

$$\Phi(\boldsymbol{v},\boldsymbol{h},\boldsymbol{\theta}) = -\sum_j a_i v_i - \sum_j b_j h_j - \sum_i \sum_j w_{ij} v_i h_j. \quad (3)$$

The firing probabilities of the visible and hidden units of the RBM are calculated by the following equations, where σ(x) is a sigmoid function. This firing probability determines the state of each unit in the RBM.

$$p(h_j = 1|\boldsymbol{v},\boldsymbol{\theta}) = \sigma\left(b_j + \sum_i w_{ij} v_i\right). \quad (4)$$

$$p(v_i = 1|\boldsymbol{h},\boldsymbol{\theta}) = \sigma\left(a_i + \sum_j w_{ij} h_j\right). \quad (5)$$

3. Resource-saving random number generator

When training an RBM, it is necessary to sample each state from the firing probability of the visible and hidden units. In this case, a large number of random number generators are required, but it is difficult to implement them in the hardware such as an FPGA. Therefore, the authors have proposed a method using truncated bits generated during fixed-point arithmetic operations as a substitute for random numbers[6,7].

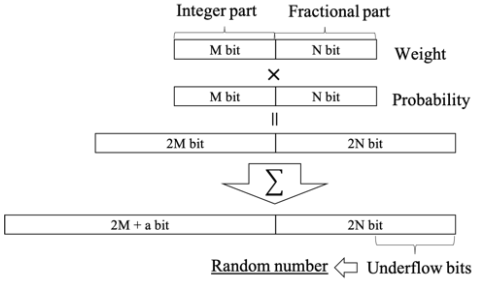

Fig. 2. Resource-saving random number generation method.

In general, when some operations are implemented in digital hardware such as FPGAs, the various operations are implemented as fixed-point operations. In the operations, if there are variables with M bits and N bits in the integer and fractional parts, respectively, the result of multiplication of these values is 2M bits in the integer part and 2N bits in the fractional part, as shown in figure 2. Furthermore, the bit width of the integer part is increased by the addition process. When the result of this operation is stored in the register, the incremented bits are truncated. In the method, this truncated bit is used as a substitute for a random number.

4. Verification Infrastructure on FPGA

Figure 3 shows the configuration of a proposed user logic verification infrastructure. In this system, an FPGA is connected to the host PC via PCI Express bus to communicate and control a user logic. When configuring the FPGA, this system uses a joint test action group (JTAG). The user can connect the user logic by two AXI interfaces to verify the user logic. In this chapter, we describe the main components of this system.

4.1. *Interface between the host PC and the FPGA*

We use PCI Express bus to connect the host PC to the FPGA to control the user circuit and to communicate

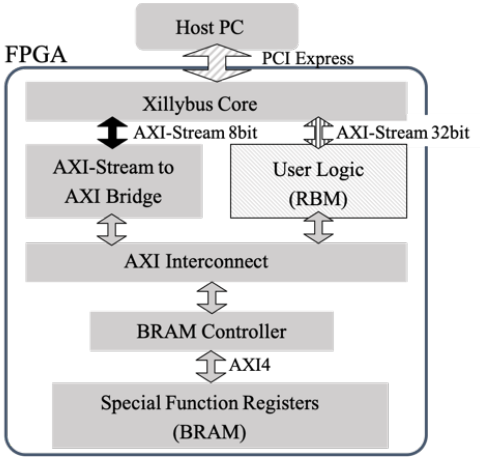

Fig. 3. User logic verification infrastructure.

data with the software on the host PC. This system applies Xillybus[10] to realize the PCI Express connection, which can convert the PCI Express data communication to first in first out (FIFO) or AXI-Stream. The host PC can access to the FPGA by reading or writing device files on its operating system.

4.2. *Internal bus and user logic interface*

This system applies AXI bus as an internal bus to connect all modules except the interfaces for Xillybus core. The user logic has AXI-Stream and AXI interface. The AXI-Stream interface connects to the Xillybus core directly, and this data path is used for data transfer. The AXI interface connects to AXI interconnect is used for the internal bus to control the user logic from the host PC through the special function registers (SFRs).

4.3. *Control registers*

The software running on the host PC can control and monitor the user circuits on the FPGA by accessing the control and status registers called SFRs. The user circuit on the FPGA can also access the SFRs and return its status, such as in-process or completed, to the software.

4.4. *AXI-Stream to AXI bridge*

The AXI-Stream to AXI bridge module extracts addresses and data from the 8-bit stream data sent from the Xillybus core to access the SFR. It is a necessary module to connect the core to the SFRs implemented in block RAM (BRAM) because the Xillybus core provides AXI-Stream interfaces.

5. Verification of the infrastructure

In order to verify the operation of the infrastructure, we synthesized an RBM using a conventional random number generators by Xilinx Vivado HLS, which is a high-level synthesis tool and implemented it as a user logic.

In this verification, the RBM trained the MNIST[11] dataset. The experimental conditions are listed below.

- Visible neuron: 784
- Hidden neuron: 150
- Without HLS optimization options
- Integer bit width: 14 bits
- Fraction bit width: 18 bits

Table 1 shows the resource utilization report of the RBM under the conditions. The target device is Xilinx Kintex 7 evaluation board KC705 (XC7K325T).

Table 1. Resource utilization report.

Resource	Utilization	Available	Utilization %
LUT	13202	203800	6.48
LUTRAM	580	64000	0.91
FF	16279	407600	3.99
BRAM	276.5	445	62.13
DSP	108	840	12.86
IO	5	500	1.00
GT	8	16	50.00
MMCM	2	10	20.00

We trained the RBM logic by the MNIST dataset on 25 epochs, and after the training, we obtained the weights of the RBM to verify the infrastructure. The weights are shown in Figure 4, and we can see the infrastructure is working in this application.

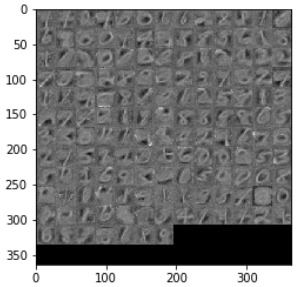

Fig. 4. Visualized weight parameter calculated on the FPGA infrastructure.

6. Conclusions and future works

In this study, we proposed and constructed a system that connects a host PC and an FPGA via PCI Express, and controls a user logic from software on the host PC. By using this system, the user does not need to construct the communication and control part by ownself except the module which user wants to verify, and the verification of the hardware-oriented system becomes possible more quickly.

Our future work is as follows. Firstly, to implement an RBM with the resource-saving random number generators in hardware, and to verify its operation in the verification infrastructure. Secondly, to connect external memory, such as an SDRAM to the verification infrastructure to be able to handle larger network parameters on the FPGA. In addition, since the user circuit is connected to the peripheral circuits only by AXI bus and AXI-Stream bus, we aim to create an environment in which the user circuit can be reconfigured while the peripheral circuits are running by utilizing the partial configuration technology. If this technology becomes available, the circuit can be verified and tested more easily and quickly. The goal of this project is to create an environment that enables easier and faster circuit verification and experimentation.

References

1. G.E. Hinton, S. Osindero, and Y.-W. Teh, *A fast learning algorithm for deep belief nets*, Neural computation, vol.18, no.7, pp.1527–1554, 2006.
2. NVIDIA, "NVIDIA Tesla V100 GPU Architecture", https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, 2017.
3. N.P. Jouppi et al. *In-datacenter performance analysis of a tensor processing unit*, Proceedings of the 44th Annual International Symposium on Computer Architecture. 2017. p. 1-12.
4. S. Wang, P. Kanwar, *BFloat16: The secret to high performance on Cloud TPUs*, https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus, 2019.
5. S. Trimberger, *A reprogrammable gate array and applications*, Proceedings of the IEEE, 1993, 81.7: 1030-1041.
6. S. Hori, T. Morie, and H. Tamukoh, *Restricted boltzmann machines without random number generators for efficient digital hardware implementation*, International Conference on Artificial Neural Networks, Springer, pp.391–398 2016.
7. S. Hori, and H. Tamukoh, *A random number generation method for hardware implemented neural networks*, IEICE Tech. Rep., vol. 119, no. 78, SIS2019-1, pp. 1-4, June 2019.
8. A. Fischer and C. Igel, *An introduction to restricted boltzmann machines*, Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, pp.14–36, Springer, 2012.
9. G.E. Hinton, *A practical guide to training restricted boltzmann machines*, Technical Report UTML TR 2010-003, Department of Computer Science, University of Tronto, 2010.
10. Xillybus PCIe IP, http://xillybus.com, access: 14th December, 2020.
11. Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and LD. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation, vol.1, no.4, pp.541–551, 1989.