SUGISAKA MASANORI

Journal of Robotics, Networking and Artificial Life
Vol. 9(1); June (2022), pp. 37–42
ISSN (Online): 2352-6386; ISSN (Print): 2405-9021
https://alife-robotics.org/jrnal.html

Research Article

A Control and Data Transfer Platform for FPGA Applications

Sansei Hori[1], Hakaru Tamukoh[2]
[1]*Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology,2-4 Hibikino, Wakamatsu, Kitakyushu, Fukuoka, 808-0196, Japan*
[2] *Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, Research Center for Neuromorphic AI Hardware, 2-4 Hibikino, Wakamatsu, Kitakyushu, Fukuoka, 808-0196, Japan*

ARTICLE INFO

Article History

Received 25 November 2020
Accepted 08 November 2021

Keywords

FPGA
Hardware Accelerator
Xillybus
RBM

ABSTRACT

Deep learning technology has made remarkable progress in recent years and has been adopted for a variety of applications such as smartphones and cloud servers. These systems employ dedicated processors to reduce power consumption and process massive amounts of data. In this paper, we propose a field-programmable gate array (FPGA) infrastructure for easy verification of user logic. The infrastructure makes it easy to communicate and control a host PC and user logic. We implemented two example logics, which were simple image processing and a restricted Boltzmann machine, into the infrastructure to confirm these features

1. Introduction

In recent years, neural network technologies such as deep learning[1] have been utilized in many applications[2], such as image processing and natural language processing. In particular, these technologies have been actively applied to various embedded fields, including smartphone applications.

However, most deep neural network (DNN) trainings require a significant amount of calculation resources and are often performed on high-performance computers with graphics processing units (GPUs)[3]. On the other hand, some embedded systems have restrictions, such as power consumption and physical size, to implement applications that use DNNs. Therefore, application-specific integrated circuits (ASICs) and system-on-a-chip (SoC) dedicated DNNs[4,5] have been actively developed to accelerate the processing and reduce power consumption. Some of these technologies have already yielded practical applications.

Field-programmable gate arrays (FPGAs)[6] can also be used to create dedicated logic 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)[7,8], which is a building block of deep belief networks (DBNs) and deep Boltzmann machines (DBMs)[9]. However, hardware implementation of a user circuit on an FPGA is costly work. In this paper, we propose an FPGA infrastructure for verifying the logic implemented in an FPGA. In addition, to verify the operation of the proposed FPGA infrastructure, we implemented two types of hardware logic: a simple image processing logic and a conventional RBM[10,11] logic. We confirm that the logic can be controlled by a host computer using the infrastructure.

This paper is an extended version of Ref. 12.

Corresponding author's E-mail: hori-sansei@edu.brain.kyutech.ac.jp, tamukoh@brain.kyutech.ac.jp
https://www.kyutech.ac.jp/english/, https://www.brain.kyutech.ac.jp/~neuro/?lang=en

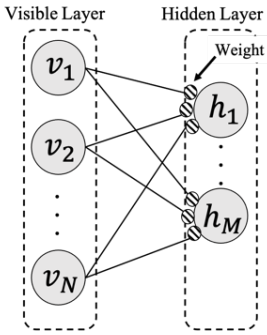


Fig. 1. Restricted Boltzmann machine.

2. Restricted Boltzmann Machine

An RBM is a generative model and forms part of the structure of DNNs[13]. An RBM has two layers: a visible layer and a hidden layer, as shown in Fig. 1. The probability distribution of an RBM is calculated by Eq. (1), where $\boldsymbol{v}$ and $\boldsymbol{h}$ represent the states of the visible and hidden units, respectively, and $\boldsymbol{\theta}$ is a set of network parameters. $Z(\boldsymbol{\theta})$, as shown in Eq. (2), is a normalized term, and Φ as shown in Eq. (3), is an energy function, where a_i and b_j represent the biases of the visible and hidden units, respectively, and w_{ij} is the weight.

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

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

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

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

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

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

3. Verification Infrastructure on FPGA

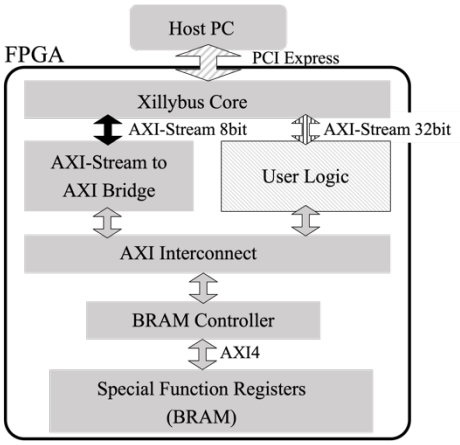


Fig. 2. User logic verification infrastructure.

Figure 2 shows the configuration of the proposed user logic verification infrastructure. In this system, an FPGA is connected to the host PC via a PCI Express bus to communicate and control the user logic. When configuring the FPGA, this system uses a joint test action group (JTAG). The user can connect the user logic with two AXI interfaces to communicate with the user logic. In this section, we describe the main components of the system.

3.1. *The interface between the host PC and the FPGA*

We use a PCI Express bus to connect the host PC to the FPGA to control the user logic and communicate data using software on the host PC. This system applies Xillybus[14] to realize the PCI Express connections, which can convert the PCI Express data communications to first in first out (FIFO) or AXI-Stream. The host PC can access the FPGA by reading or writing device files on its operating system.

3.2. *User logic*

User logic is shown in Fig. 2, which is a central component of the FPGA system, and this implements the processes of one part of a user application. The user application consists of two parts: software on the host PC and the user logic on the FPGA system. Hardware-optimized processing logic is implemented in the user logic to realize high-performance processing, compared to a software-only system.

The minimum requirements of the user logic are interfaces that connect special function registers (SFRs) and the Xillybus Core, and the two states of processing.

The interfaces are described in this paper. The processing states include both standby and active states. In the standby state, the logic waits for an execution signal from the host PC. In the case of the active state, the logic runs. User logic developers can implement more than two states in the active state, if necessary.

3.3. *Internal bus and user logic interface*

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

3.4. *Control registers*

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

The SFR example is shown in Fig. 3, where mode bits are parameters to switch the user logic behaviors, and E is an execution flag of user logic. The layout shown in Fig. 3 is an example, and the user can design the form as necessary.

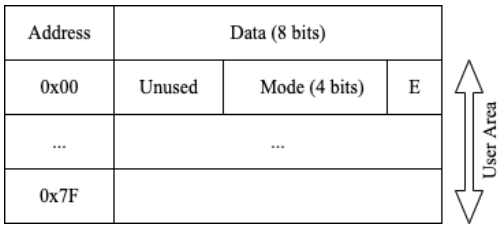


Fig. 3. Layout example of the SFR.

3.5. *AXI-Stream to AXI bridge*

The AXI-Stream to AXI bridge module extracts an address and data from the 8-bit stream data, sent from the Xillybus core, to access the SFRs. It is necessary to connect the core to the SFRs, implemented in block RAMs (BRAMs), because the Xillybus core provides AXI-Stream interfaces.

3.6. *Access sequences for SFRs via Xillybus*

This section explains the details of the access procedure from the host PC to the SFRs via AXI-Stream interfaces provided by Xillybus.

The system requires two transactions to access the SFR, as shown in Fig. 4. The first transmits a destination address and mode information, which has read and write modes, from the host PC to the system. The read and write mode bit indicate the direction of data transfer. When it is set to zero, the mode is the read mode. Conversely, when the mode bit is set to one, the mode is the write mode. The second sends or receives an SFR parameter. In the read mode, the transaction is for reading the parameter from the SFRs to the host PC. By contrast, in the case of the writing mode the transaction is for writing the parameter from the host PC to the SFRs. The AXI-Stream to AXI bridge controls these two transactions.

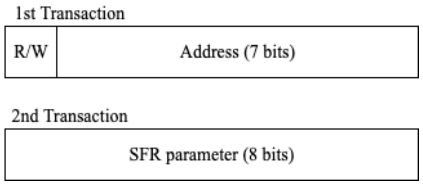


Fig. 4. Data structures to access the SFRs. R/W means a read and write mode bit.

3.7. *A control method for the user logic*

Steps to start the user logic (Fig. 2) via the SFRs (Fig. 3) from the host PC.

(i) An FPGA is configured.
(ii) The user logic wakes up in a standby state, and the logic starts watching an “execution flag”, designed on the SFR.
(iii) The host PC sets parameters to control the user logic, for example, mode bits for switching behaviors of the user logic, on the designed address of the SFR.
(iv) The host PC sets the execution flag on the SFR.
(v) User logic processes wake up.
(vi) The user logic reads the mode bits on the SFR, and the logic starts a process specified by the bits.
(vii) After finishing the user logic process, the logic negates the execution flag on the SFR.
(viii) The host PC ensure that the logic completes its process by watching the negation of the flag.

As mentioned above, the user logic always watches the execution flag in a standby state after the FPGA configuration. By contrast, the host PC can also watch the execution flag through Xillybus to check the user logic state. Furthermore, the data transfer through the 32-bit AXI-Stream interface can be executed separately from

the control steps, as shown above, and, if necessary, during the processing of the user logic.

4. APIs for Software on the Host PC

This section explains the method for communicating software on the host PC with the proposed infrastructure and application programming interfaces (APIs) that we created to facilitate communication.

4.1. *Device files for Xillybus*

Linux operating systems provide interfaces, called device files, to access peripheral devices such as USB devices and PCI Express devices. The software reads or writes device files to communicate with peripheral devices.

In the case of the Xillybus, the device files commonly appear under the "/dev" directory as xillybus_read_8, xillybus_read_32, xillybus_write_8, and xillybus_write_32. In the C++ programming environment, the software must first open the device file. Second, the software uses system calls to write or read device files.

4.2. *APIs for Xillybus communication*

To access the SFRs, the software must make two transactions to send the read or write mode bit, an SFR address, and an SFR parameter, which are explained in Section 3.6. These steps are complicated for software developers. Therefore, we created APIs, named xillybus_tools and sfr_tools, to make it easy to communicate the user logic and the SFRs via Xillybus from the software.

The xillybus_tools provide basic data communication functions to the software on the host PC. The API provides two classes: "xillybus8" and "xillybus32." The software uses the xillybus32 class to receive or transmit data (i.e., an unsigned int value, an array, and a std::vector) to the user logic on the FPGA system through the 32-bit AXI-Stream interface.

The sfr_tools provide functions to make address-based transactions, explained in Section 3.6, to the software on the host PC. It requires a "xillybus8" class object to access Xillybus. The software only needs to call one function to access the SFRs and does not require the generation of transactions, such as a read and write flag, an address, or the SFR parameter.

We opened these APIs on GitHub[15, 16]

5. Verification of the Infrastructure

We implemented two applications on the system to verify the operation of the infrastructure. The first is a simple image processing application that reverses the colors of an input grayscale image to validate the SFR and data communication between the host PC and the FPGA system. The second is the conventional RBM. This application is implemented to validate the system using full-fledged user logic, which has many processing modes, parameters, and data. Note that these applications were developed by a high-level synthesis[17] environment, provided by Xilinx Inc.[18]

5.1. *A simple image processing logic*

This is a very simple image processing application that reverses the colors of an input grayscale image to validate transmitting parameters to the SFRs from the software on the host PC, controlling the user logic and data communication via the Xillybus.

This application reverses the colors of the input grayscale image, as shown in Eq. (6).

$$y = 255 - x. \tag{6}$$

In the equation, y represents the converted pixel value, and x is the original pixel value. All pixels have 8-bits values.

User logic has three modes for operating the application. Mode zero is the receiving mode of the input value from the host PC, mode one is the reverse mode of the image, and mode two transmits the results to the host PC. The user logic has two BRAMs as buffers to save the original image and the resulting image. An overview of the logic layout and data flow is presented in Fig. 5.

The SFR layout is shown in Fig. 6. Address 0x00 has mode-select bits and an execution bit. Address 0x01 and 0x02 have input image sizes.

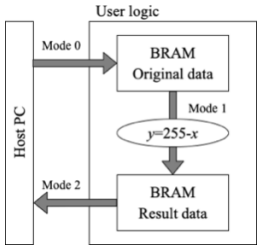
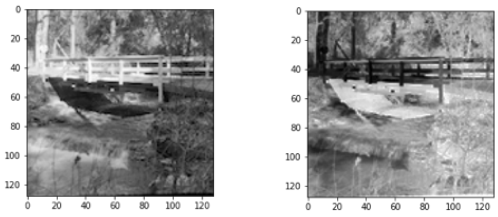


Fig. 5. Overview of the data flow between the host PC and the user logic. This figure only focuses on the data flow of each mode.

Address	Data (8 bits)		
0x00	Unused	Mode (4 bits)	E
0x01	Width (8 bits)		
0x02	Height (8 bits)		

Fig. 6. A layout of the SFRs for the user logic. Mode bits switch the user logic mode, and E is an execution bit.

We input an image, selected from the standard image database[19] (SIDBA), which was reduced down to 128×128 pixels. The input and resulting images are shown in Fig. 7, from which we can see that the application achieves the desired outcome.



(a) Original image (b) Result image

Fig. 7. Input and output images comparison.

5.2. *An RBM logic*

We synthesized an RBM using Xilinx Vivado HLS, which is a high-level synthesis tool and implemented it as user logic.

In this verification, the RBM was trained by the MNIST[20] dataset. The experimental conditions are as follows:

- Visible unit: 784
- Hidden unit: 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 these conditions. The target device was 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 Fig. 8, and it can be observed that the infrastructure works well in this application.

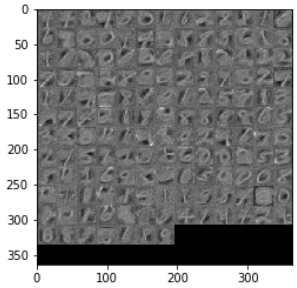


Fig. 8. Visualized weight parameters calculated on the FPGA infrastructure.

6. Conclusions and Future Works

In this study, we proposed and constructed an infrastructure that connects a host PC and an FPGA via PCI Express using the Xillybus, and controls the user logic from software on the host PC. We have also created APIs for infrastructure and published them on GitHub. Furthermore, we implemented two applications using the infrastructure to validate the infrastructure and confirm that it works correctly. Using this system, the user does not need to construct the communication and control part by themselves, except for modules that require verification. Verification of the hardware-oriented system becomes possible more quickly.

In future work, first, we will implement an RBM with resource-saving random number generators[7] in the hardware and verify its operation. Second, we will focus on connecting external memory, such as an SDRAM, to the infrastructure to enable the handling of larger network parameters on the FPGA. In addition, because the user logic is connected to the peripheral circuits only by the AXI bus and AXI-Stream bus, we aim to create an environment in which the user logic can be reconfigured while the peripheral logics are running by utilizing the partial configuration technology. With the availability of this technology, logic verification and testing can be achieved in a much more straightforward manner, and with improved speed. 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. S. Dargan, M. Kumar, M.R. Ayyagari, and G Kumar, *A survey of deep learning and its applications: a new paradigm to machine learning*, Archives of Computational Methods in Engineering, vol. 27, no. 4, pp.1071-1092, 2020.
3. NVIDIA, *NVIDIA Tesla V100 GPU Architecture*, https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, 2017, access: August 16th , 2021.
4. N.P. Jouppi et al., *In-datacenter performance analysis of a tensor processing unit*, Proceedings of the 44th Annual International Symposium on Computer Architecture, vol. 45, no. 2, pp. 1-12, 2017.
5. S. Wang, and 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, accessed: August 19th , 2021.
6. S. Trimberger, *A reprogrammable gate array and applications*, Proceedings of the IEEE, vol. 81, no. 7, pp. 1030-1041, 1993.
7. 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.
8. 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.
9. R. Salakhutdinov, G.E. Hinton, *Deep Boltzmann machines*, Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics, vol. 5, pp. 448-455, 2009.
10. 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.
11. G.E. Hinton, *A practical guide to training restricted Boltzmann machines*, Neural Networks: Tricks of the Trade, pp. 599-619, 2012.
12. Sansei Hori, and Hakaru Tamukoh, *A hardware-oriented random number generation method and a verification system for FPGA,* Proc. 2021 Int. Conf. on Artificial Life and Robotics (ICAROB 2021), pp. 122-125, 2021.
13. N. Zhang, S. Ding, J. Zhang, Y. Xue, *An overview on restricted Boltzmann machines,* Neurocomputing, vol. 275, pp. 1186-1199, 2018.
14. Xillybus PCIe IP, http://xillybus.com, access: December 14th , 2020.
15. xillybus_tools, https://github.com/HoriThe3rd/xillybus_tools, accessed: August 16th, 2021.
16. sfr_tools, https://github.com/HoriThe3rd/sfr_tools, accessed: August 16th, 2021.
17. VIVADO High Level Design, https://www.xilinx.com/products/design-tools/vivado/high-level-design.html, accessed: August 22th, 2021.
18. Xilinx, https://www.xilinx.com, accessed: August 22th, 2021.
19. Standard image database, http://www.ess.ic.kanagawa-it.ac.jp/app_images_j.html, access: August 16th, 2021.
20. 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.

Authors Introduction

Mr. Sansei Hori

He received his Bachelor's degree from Tokyo University of Agriculture and Technology, Japan in 2012. He received his Master's degree from Kyushu Institute of Technology, Japan in 2014. He is currently a Doctoral course student in Kyushu Institute of Technology, Japan.

Dr. Hakaru Tamukoh

He received his B.Eng. degree from Miyazaki University, Japan, in 2001. He received his M.Eng. and Ph.D. degrees from Kyushu Institute of Technology, Japan, in 2003 and 2006, respectively. He was a postdoctoral research fellow at Kyushu Institute of Technology, from 2006 to 2007. He was an assistant professor at Tokyo University of Agriculture and Technology, from 2007 to 2013. He is currently a professor in the graduate school of Life Science and Systems Engineering, Kyushu Institute of Technology, Japan. His research interest includes digital hardware design, soft-computing and home service robots.