A Hiding External Memory Access Latency by Prefetch Architecture for DNN Accelerator Available on High-Level Synthesis

Ryota Yamamoto
Graduate School of Informatics
Nagoya University
Nagoya, Aichi 464-8603, Japan
muku@ertl.jp

Shinya Honda
Faculty of Science and Engineering
Nanzan University
Nagoya, Aichi 466-8673, Japan
shonda@nanzan-u.ac.jp

Masato Edahiro
Graduate School of Informatics
Nagoya University
Nagoya, Aichi 464-8603, Japan
eda@ertl.jp

Abstract— In recent years, there has been a demand for deep neural network (DNN) inference applications in embedded systems. We are developing a framework to design hardware (HW) on an FPGA for DNN inference. Although low-end FPGAs are needed to reduce the cost, FPGAs have limited internal memory (Block RAM, BRAM). Therefore, it is necessary to use SDRAM instead of BRAM, but to speed up the access to SDRAM, a prefetcher is needed that can be used in the system-level design. In this study, we designed a prefetch architecture in a system-level design environment that can be easily implemented in C code for high-level synthesis. We propose a method of storing data in the BRAM by transferring the data in a burst. We designed DNN inference HW with external memory (SDRAM) access using prefetch architecture. As a result, the prefetch design is faster than cases using BRAM or SDRAM. In particular, it is found that the performance is up to a factor of 10 faster than that of SDRAM access without prefetch.

I. Introduction

In recent years, there has been a demand for deep neural network (DNN) inference applications in embedded systems. For embedded systems, owing to performance requirements, FPGAs are expected to be used for DNN inference processing. However, there are resource constraints in terms of memory capacity and power consumption. Hardware (HW) for specific operations can be realized using FPGAs, and applications can be developed with high computational cost. However, there is an issue that the development work time increase due to the high expertise of FPGA development. Given this background, DNN frameworks based on FPGAs have been actively studied [1, 2, 3, 4, 5].

To decrease the development cost and degree of expertise required in FPGA development and to facilitate the co-design of HW and software (SW), we have developed SystemBuilder, a system-level design environment [6, 7, 8]. SystemBuilder takes C source code for high-level synthesis (HLS) and system configuration files as input, and it outputs an executable for a CPU and a bitstream for an FPGA. An overview of SystemBuilder is shown in Fig. 1. As described in detail in Section 2, SystemBuilder provides communication primitives that enable communication between processes. The communication

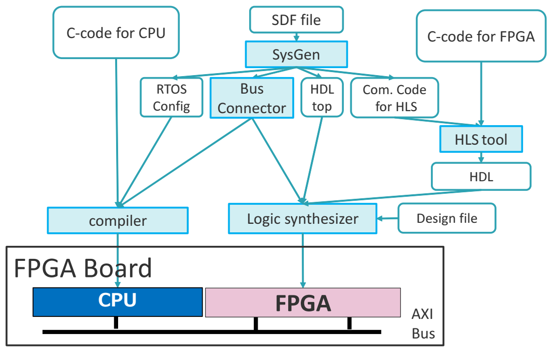


Fig. 1.: SystemBuilder flowchart

primitives have interprocess communication and memory access communication.

Table I presents an example of FPGA specifications for each grade[1]. As this table shows, it is difficult to place all the weights on a low-end chip with less than a few megabit capacity. Table II lists the number of weights of the popular DNNs [9]. Here, by converting the number of weights in the table to data size, for example, by assuming that the weights are single-precision floating-point (32-bit), the most numerous parameter in this table is 4,672 Mb for Overfeatfast, which means that even high-end FPGAs cannot store the data in their BRAM. Even Lenet5, which has the smallest number of parameters in this table, has 1.875 Mb, which means that the BRAM is insufficient in some cases in low-end FPGAs. Therefore, although some high-end FPGAs have several hundred MB of BRAM, a large DNN may require the use of SDRAM in some cases. As a result, SDRAM accesses are required, and the overhead of such accesses has a negative effect on the overall system latency and throughput. Therefore, because a DNN application needs to store a large amount of weights, they are placed in SDRAM to execute the inference, SDRAM access slows down the execution speed, even when using FPGAs.

We have developed a communication primitive to hide the

[1]Table I was created by referring to `https://www.digikey.com/` (accessed on October 26, 2020).

TABLE I
: FPGA specifications

Vender	Chip	BRAM Size (Mb)	Price
Xilinx	XCVU190	132.9	$32,605
Intel	1SD280PT	240.1	$209,662
Xilinx	XC7Z020	4.9	$114
Intel	5CSXFC2C	512	$89

TABLE II
: Number of weights in popular DNNs

Name	Size
Lenet5	60k
AlexNet	61M
Overfeatfast	146M
VGG16	138M
GoogLeNetv1	7M
ResNet50	25.5M

memory access latency by a prefetch architecture to solve the SDRAM access bottleneck in DNN inference. This communication primitive can be called from C with hiding the HDL, so that it can be easily used in high-level synthesis. Prefetch architecture takes advantage of the fact that the weights of the DNN are used fixed value cyclically. The hardware can receive data from the buffer rather than SDRAM accesses, thus reducing the latency of SDRAM accesses.

The contributions of this study are as follows:

- Prefetch architecture is implemented so that SystemBuilder can automatically generate HDL and communication interfaces for it, which can be easily generated from C.
- A case study demonstrates that SDRAM accesses using prefetch architecture are faster than BRAM and SDRAM accesses without it.
- The case study shows that, even if the number of master interfaces (MIFs) is reduced, the increase in HW latency can be reduced.

II. SystemBuilder

A. Overview

SystemBuilder is a system-level design environment that enables the co-design of SW and HW [8]. The level of design abstraction between SW and HW can be increased, and the design can be performed without considering the design as HW, compared to the design with HDL. The advantages of SystemBuilder are that the developer does not have to implement communication for inter SW-SW, HW-HW, and SW-SW, and that the HW is designed without considering it as a HW design compared to the design with HDL by increasing the abstractness of the design.

SystemBuilder provides the following features:

- C simulation,
- SW–HW co-simulation,
- a communicator module and a communication I/F generator,
- an executable for the CPU generator, and
- a bitstream for the FPGA generator.

As shown above, SystemBuilder can be integrated with various simulation environments and can be verified without using an actual device.

SystemBuilder generates the files for HLS and logic synthesis (for HW) and compilation (for SW) from the system description, as shown in Fig. 1. The HW and SW specifications are defined by the process. The ‘process’ in this paper is defined as each module that divides the system into functional units.

Next, there are the input files of SystemBuilder. The input files are the C source file and the system definition file (SDF). The HW or SW process is described in C source code, and the SDF defines the configuration of the entire system, including processes and channel definitions. SDFs are written in YAML format and contain the following information:

- process type (HW or SW) definition,
- shared memory definition,
- communication primitives (channel) definition, and
- process definition.

Process Type Definition Developers can specify whether the process is executed as HW or SW. This definition determines whether each portion of C code is treated as input to the HLS or to the compiler.
Shared Memory Definition When using the memory (MEM) channel or the prefetch channel proposed in this study, the developers can define the SharedMemory. It specifies the address of the memory to be treated as the common memory and defines the access port required to create the HW routing information. The definition of shared memory includes the start address and the depth of the shared memory that can be used in the MEM channel.
Communication Primitives (Channel) Definition Developers can describe the definition of communication primitives described in Section B. This definition includes the data size (depth) and memory size to be allocated.
Process Definition Developers can specify the top function to be specified as a process and the source file name that contains the function. The channels used in the process with the direction can also be defined.

By inputting these definitions to SystemBuilder and C files for HLS, or for compilation, the simulation files can be generated.

The generated C files for HLS are input to HLS tools such as CyberWorkBench (NEC) and eXCite (YXI) to perform HLS. Additionally, we have enabled Vivado HLS (Xilinx), an HLS tool that can be used free, on SystemBuilder. The HLS tool generates the HDL corresponding to the input C file. SystemBuilder also generates HDLs for bus access and communication I/F and hardware-top HDLs, and these HDL files are used for logic synthesis.

Logic synthesis is currently supported by Quartus Prime (Intel) and Vivado (Xilinx). With these, HW processes can be designed by these tools.

Moreover, developers can build the files generated for the SW by using the compiler of the target CPU. In SystemBuilder, the SW process is built as a μITRON-based or an AUTOSAR-based RTOS application, and an executable file is generated.

It can be executed by writing the bitstream for the HW and the SW executable file generated by the above to the actual device.

TABLE III
: Communication primitives available in SystemBuilder

Primitive Name	Description
BC (Blocking Channel)	This channel can send and receive one data at a time and is equivalent to a FIFO. The receiver waits for reception while the FIFO is empty, and the sender waits for transmission while the FIFO is full.
NBC (Non-Blocking Channel)	This channel can send and receive one data at a time and is equivalent to a register. There is no waiting for transmission and reception.
MEM (MEMory Channel)	This channel is given an offset value to the address specified in the SDF and can send and receive data at the specified address in a single communication. There is no waiting for transmission and reception.

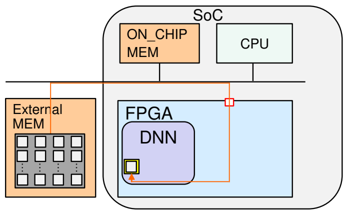


Fig. 2.: SDRAM access by the MEM channel

B. *Communication Primitives*

The communication primitives for interprocess communication provided by SystemBuilder are given in Table III.
Blocking Channel (BC) The BC is a channel that can communicate one data of 8, 16, or 32 bits. This channel is regarded as a FIFO in HW, and the depth of the FIFO may be specified. If the FIFO is empty, the receiver process waits to receive. However, if the FIFO is full, the sender process waits to send. To use the BC from C, developers can use two APIs, one for transmission and the other for reception. In each case, the argument is passed as data for transmission or as the address of a variable that stores the data to be received.
Nonblocking Channel (NBC) The NBC is a channel that can communicate one data of 8, 16, or 32 bits. This channel is regarded as a register in HW. The NBC is just a variable in SW, and the receiver process obtains the value stored in the NBC at a time that is not synchronized with the sending process. Moreover, the sending process writes the value asynchronously. The NBC is the same API as the BC in the C language.
MEM Channel The MEM channel is a channel that can communicate one data of 8, 16, or 32 bits, and this communication is asynchronous. The MEM channel differs from the NBC in that an offset value is given for a statically assigned address, and data access is provided to a specific address. This channel can access both SDRAM and BRAM. The MEM channel transmits and receives only one data at a time in the case of SDRAM access (even BRAM access), as shown in Fig. 2. This leads to the disadvantage of increasing the latency of memory accesses because of the latency of the MEM channel.

III. Characteristics and Requirements for SDRAM Access in DNN Inference

As described in Section I, to implement DNN inference in an FPGA, it is sometimes necessary to place trained data, such as weights, in SDRAM. Therefore, it is important to consider the characteristics of weights and SDRAM accesses to provide a background to the HW for better DNN inference. In terms of weights and SDRAM access, DNN inference has the following characteristics:

C1 In each case, the weights are not able to be stored in the BRAM.
C2 The order of weights is constant for each layer.
C3 Each layer in the DNN inference reuses weights cyclically each time it finishes processing all the input given for that layer.
C4 To perform the filtering, only the weights of the filter size are used at a time.
C5 After convolving to the end of the input, the weights are unnecessary until the next input is given.

From these characteristics, the requirements for SDRAM access in DNN inference are as follows:

R1 A part of the weights is temporarily stored in the BRAM.
R2 The weights stored in the SDRAM are acquired in the order in which they are stored.
R3 When the weight to be used in the layer has been read to the end address, then reload weight from the start address.
R4 The number of communication cycles can be reduced by acquiring a filter size or one side of a filter size at a time in a single communication.
R5 It is unnecessary to store the weights that have already been used.

In addition to these requirements, we use high-level synthesis to consider its use in SystemBuilder. If HLS is assumed, the following is required:

R6 The hardware modules for memory access are hidden and should be easily available through the C API.

As an architecture that satisfies these requirements, a cache architecture could be implemented; however, because of them, a mechanism as complex as a cache mechanism is unnecessary.

IV. Prefetch Mechanism

This section describes the specifications of the prefetch blocking channel (PFBC) introduced into SystemBuilder. Figure 5d shows the mechanism of the PFBC for HW processes, and Figure 5c is SDF for the PFBC. PFBC is introduced as one of the communication primitives of SystemBuilder. As mentioned above, the purpose of the PFBC is to fulfill **R1–R6**.
There are the following parameters for the PFBC:

- data size (selectable from 8, 16, and 32 bits),

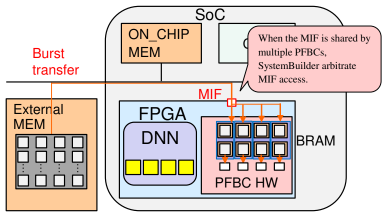

SoC
ON_CHIP MEM
When the MIF is shared by multiple PFBCs, SystemBuilder arbitrate MIF access.
Burst transfer
MIF
External MEM
FPGA
DNN
BRAM
PFBC HW

Fig. 3.: MIF collision because of sharing of the MIF by PFBCs

- vector size (can be specified from 2 to 16),
- data block depth,
- FIFO (internal buffer) depth,
- bit width for burst transfer (up to 256 in power of 2),
- enable/disable auto reload feature,
- data location (e.g., SDRAM), and
- MIF assignment.

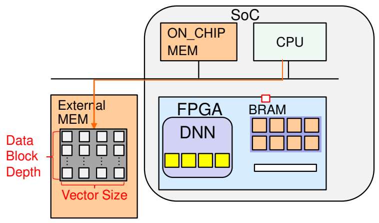
In Fig. 5d, it is assumed that the data are placed in the SDRAM by the CPU pictured in Fig. 4a, and the number of data of the size specified by the data size multiplied by the vector size and data block depth is placed in the SDRAM. After that, the FIFO with the depth specified by the FIFO depth exists for the vector size, and the data are stored in the FIFO by burst transfer, as shown in Fig. 4b. Finally, the data stored in the BRAM are transferred by the BC to the HW process by the number of vectors at a point in time, as shown in Fig. 4c. At this time, the data retrieved by the process are deleted from the FIFO, and the vacant area is further loaded with data from the SDRAM by burst transfer. The PFBC also has a function to automatically reload the data when the final data (the (ID-1)th data) is read; after the final data is read, it automatically reads the first data. Note that the Master Interface (MIF) for accessing the memory must be specified. And multiple PFBCs can be assigned to a single MIF (Fig. 3). SystemBuilder automatically determines when each PFBC uses the MIF.

The reading data from SDRAM by the PFBC is performed in the background of the HW process operations. Therefore, for DNN inference, the HW process only retrieves the data from the BRAM because the data necessary for the operation are already stored in the BRAM. The PFBC thus takes advantage of the fact that the parameters of DNNs are always used in a certain order (**C3**). Therefore, continuous data acquisition by burst transfer is effective; however, this feature is not considered to be effective for operations used in a random order.

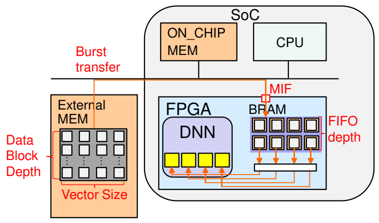
Figure 5 shows an example of the SDF and C description for the PFBC. In the SDF, developers can describe the configuration for the PFBC and MIF. Developers can also describe the HW description in C. In the figure, L0_COEF_PF_READ is an API to read vector data from SDRAN by HW; this corresponds to Fig. 4c.

Next, we explain how to use the PFBC for both SW and HW processes. The PFBC is a different interface from SW and HW processes. For the SW process, it is possible to read and write as well as use MEM; however, for the HW process, it is only possible to read. Therefore, the SW process writes the data that HW reads by using the PFBC, and the HW retrieves those data when the process needs them.

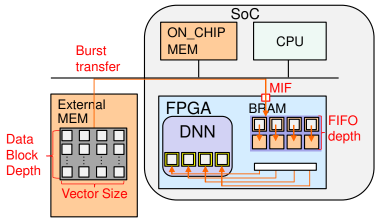
The required usage of BRAM can be adjusted according to the FIFO depth and vector size. Therefore, it may be possible to implement large-scale DNNs that have not been possible in the past on smaller FPGAs in the future.



(a) Writing weights from the CPU to an SDRAM (with a vector size of 4)



(b) Burst transfer to BRAM



(c) Sending to the HW process and prefetch from an SDRAM

Fig. 4.: Overview of the PFBC mechanism for the HW process

Compared with the method proposed in this study, Wei et al. proposed a DNN inference for FPGAs that implements weight buffer prefetching and focuses on memory access [10]. They devised a method to hide the memory access time while executing other layers of processing, focusing on the execution time of each layer. In this study, we do not use the hiding method of memory access latency as in Wei et al. However, prefetch is automatically performed when the value is taken from the internal buffer, and users can use the prefetch architecture with simple notation. Although the possibility of consuming a large amount of memory access bandwidth is considered a disadvantage of our method, we can adjust the number of data retrieved from memory when prefetching by setting upper limits on vector size and FIFO size. Specifically, the vector size is limited to a maximum of 10, and the internal buffer depth is limited to a maximum of 1024. However, by setting these numbers, the PFBC used at each layer can also limit excessive memory accesses.

V. Case Study

We used a handwritten numeric image (MNIST) inference program. This program was created by Nakahara et al.[2]. This program has a five-layered DNN with binary activations and 8-bit integer weights. The baseline for this case study is an optimized description with loop folding and other optimizations. The structure of the network is given in Table IV. Addition-

[2]http://www.cqpub.co.jp/interface/download/contents.htm

```
MIF:
 - {id: 1, mem: [l0_coef_pf]}
MemoryChannel:
 - {name: l0_coef, size: 8, depth: 150, loc: sdram}
```

(a) SDF for MEM

```
for (i = 0; i < L0_KSIZE; i++){
    for (j = 0; j < L0_KSIZE; j++){
        L0_COEF_READ(dmap * L0_KSIZE * L0_KSIZE + i * L0_KSIZE + j, &ui);
        coef_w_fmap[i * L0_KSIZE + j] = (int8)ui;
    }
}
```

(b) C for MEM

```
MIF:
 - {id: 1, mem: [l0_coef_pf]}
PreFetchBlockingChannel:
 - {name: l0_coef_pf,    size: 8, vsize: 5, depth: 30, burst_len: 256, fifo_depth: 15, loc: sdram, auto_reload: }
```

(c) SDF for PFBC

```
for (i = 0; i < L0_KSIZE; i++){
    ui = i * L0_KSIZE;
    L0_COEF_PF_READ((uint8 *)&si[0], (uint8 *)&si[1], (uint8 *)&si[2], (uint8 *)&si[3], (uint8 *)&si[4]);
    for (j = 0; j < L0_KSIZE; j++){
        coef_w_fmap[ ui + j ] = si[ j ];
    }
}
```

(d) C for PFBC

Fig. 5.: Example of HW description for memory access MEM and PFBC

TABLE IV
: Number of weights in the target network

Name	Weights
Layer 0 (convolution)	150
Layer 1 (average pooling)	24
Layer 2 (convolution)	2,400
Layer 3 (average pooling)	64
Layer 4 (convolution)	48,000
Layer 5 (fully connected)	1,200

ally, Fig. 5 shows an example description of the SDF and HW C code, which defines and uses MEM and the PFBC.

Although the original implementations can store all weights and feature image data in the BRAM, to evaluate the effectiveness of the PFBC, in this study, all the weights of these layers, except layer 2 and layer 5, are placed in SDRAM. There are several reasons for excluding layer 2 and layer 5. In layer 2, the original source code is designed to skip some images for speed-up, and the order of parameter access may not be continuous. Therefore, it was decided not to apply the PFBC for layer 2. In the case of layer 5, the layer is excluded from the application of the PFBC because it is a fully connected layer, and applying the PFBC makes it impossible to apply loop folding and loop unrolling.

To compare the design with and without PFBCs, three designs were implemented:

- with BRAM as a baseline and without PFBCs,
- with SDRAM and without PFBCs, and
- with SDRAM and with PFBCs.

We measured the execution time, circuit area, and BRAM usage. In order to investigate the effect of changing the number of MIFs, the design of ZYBO with MIFs of 1, 2 and 4 was also investigated. We used ZYBO Z7-20 (Xilinx, hereinafter called ZYBO) and DE4 Education Board (Intel, hereinafter called DE4).

In the baseline, the weights are defined as a two-dimensional constant array, which is rewritten using MEM to create a design with BRAM. The design with SDRAM is the design with the BRAM rewritten for SDRAM. Compared to the baseline, this design has four lines or eight lines of changes and additions, but no difference in terms of the C description. In Fig. 5a, the MEM channel l0_coef is defined and associated with MIF 1. The channel has 150 data block depth and 8-bits per one data block placed on SDRAM. In Fig. 5b, the HW process read weight by L0_COEF_READ. The first argument is the index, and the second argument is the address of the variable to be read.

Finally, the design with PFBCs is a modified design in which the SDRAM is used to obtain the weights by PFBCs. Compared to the original implementation, this implementation has eight line changes and additions in the SDF and 20 line changes in the C description. In Fig. 5c and Fig. 5d are the PFBC definition described in Section IV.

VI. Discussion

We consider the results of the logic synthesis and execution presented in Table V. In terms of latency, the design with PFBCs was the most successful in reducing the latency of memory accesses. The speeds of both DE4 and ZYBO were factors of 2–3 faster than that of the base case. When SDRAM was used without a PFBC, the speeds of DE4 and ZYBO were factors of approximately 1/4 and 1/10 that of the base case, re-

TABLE V
: Case study results

Target	Com. I/F	Latency (μs)	Throughput (fps)	BRAM usage (KB)	Area (LEs, slice logics)
DE4 @50 MHz	BRAM (base)	24,065	249.3	211.6	14,202
	SDRAM	103,754	57.8	166.5	13,969
	PFBC (4 MIFs)	9,141	656.4	176.8	15,603
ZYBO @100 MHz	BRAM (base)	12,470	249.3	371.3	14,430
	SDRAM	118,566	57.8	303.8	14,304
	PFBC (4 MIFs)	4,787	1253.4	317.3	15,535
	PFBC (2 MIFs)	4,845	1238.4	317.3	14,997
	PFBC (1 MIF)	4,986	1203.4	317.3	14,742

Latency is the execution time for six images.

spectively. The reason for the difference between the boards is the difference in the bus width: DE4 has bus access with a width of 256 bits, while ZYBO has bus access with a width of 32 bits. These results suggest that the PFBC can reduce the memory access latency compared to other implementations.

Next, we will discuss BRAM usage. BRAM is internal memory and the baseline design requires a large capacity. The design of the PFBC also uses more BRAM than the SDRAM design; however, this is because FIFO is located in the BRAM, which requires more BRAM than simply getting data from the SDRAM.

Finally, the PFBC had the largest circuit area. The PFBC has an interface for each layer. The scale of the circuitry is larger than that of other designs because of the automatic reading of data and other functions.

There was no difference in BRAM usage, as the number of used MIFs increases, the circuit area increased. Therefore, we can increase the number of PFBCs and reduce the increase in latency from MIF contention, even if the number of MIFs is insufficient for PFBCs.

These results show that PFBC design can reduce the amount of BRAM used and is faster than other designs with memory accesses. As described in Section I, most of the FPGAs currently available from FPGA vendors have a large circuit area but low BRAM. The choice of low-end FPGAs becomes impractical when considering high-speed implementation with BRAM, and this can be a major problem for embedded system development in which reduced costs is a goal. Therefore, even if the circuit area is large, PFBCs are considered to be useful because they reduce the BRAM usage and allow faster implementation than using SDRAM.

VII. Summary and Conclusions

In this study, we proposed PFBC and using it as a method to hide SDRAM access latency. The results show that PFBCs can be used to infer DNNs faster by a factor of about 10 than simple access to SDRAM.

One of the challenges for the future is to rethink the design of the DNN layer operations themselves to gain more insight into the effective use of PFBCs. To apply PFBCs to DNNs, we believe that there are some cases in which PFBCs can be more effective for designs where loop expansion or loop folding could not be applied in the past (e.g., by modifying the order of layer operations).

References

[1] H. Nakahara, T. Fujii, and S. Sato, "A fully connected layer elimination for a binarizec convolutional neural network on an FPGA," in *FPL 2017*. IEEE, 2017, pp. 1–4.

[2] A. Prost-Boucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on FPGA," *FPL 2017*, 2017.

[3] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs," in *FPGA 2017*. ACM, 2017, pp. 15–24.

[4] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 99, pp. 1–14, 2018.

[5] F. Li and B. Liu, "Ternary Weight Networks," *arXiv preprint arXiv:1605.04711*, 2016.

[6] Y. Ando, Y. Ishida, S. Honda, H. Takada, and M. Edahiro, "Automatic synthesis of inter-heterogeneous-processor communication for programmable system-on-chip," *IPSJ Transactions on System LSI Design Methodology*, vol. 8, pp. 95–99, Aug 2015.

[7] Y. Ando, S. Honda, H. Takada, and M. Edahiro, "System-level design method for control systems with hardware-implemented interrupt handler," *Journal of Information Processing*, vol. 23, no. 5, pp. 532–541, Sep 2015.

[8] S. Honda, H. Tomiyama, and H. Takada, "Systembuilder: A system level design environment," *IEICE Trans. Information & Systems*, vol. 88, pp. 163–174, 2005.

[9] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[10] X. Wei, Y. Liang, and J. Cong, "Overcoming data transfer bottlenecks in FPGA-based DNN accelerators via layer conscious memory management," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.