Article

RISC-V Virtual Platform-Based Convolutional Neural Network Accelerator Implemented in SystemC

Seung-Ho Lim 1,*1, WoonSik William Suh 2, Jin-Young Kim 1 and Sang-Young Cho 1

1 Division of Computer Engineering, Hankuk University of Foreign Studies, Yongin 17035, Korea; kestro960408@gmail.com (J.-Y.K.); sycho@hufs.ac.kr (S.-Y.C.)
2 Genesys Logic, Section 3, Beixin Rd., Xindian Dist. 231, New Taipei City 23143, Taiwan; William.Suh@genesyslogic.com.tw
* Correspondence: slim@hufs.ac.kr

Abstract: The optimization for hardware processor and system for performing deep learning operations such as Convolutional Neural Networks (CNN) in resource limited embedded devices are recent active research area. In order to perform an optimized deep neural network model using the limited computational unit and memory of an embedded device, it is necessary to quickly apply various configurations of hardware modules to various deep neural network models and find the optimal combination. The Electronic System Level (ESL) Simulator based on SystemC is very useful for rapid hardware modeling and verification. In this paper, we designed and implemented a Deep Learning Accelerator (DLA) that performs Deep Neural Network (DNN) operation based on the RISC-V Virtual Platform implemented in SystemC in order to enable rapid and diverse analysis of deep learning operations in an embedded device based on the RISC-V processor, which is a recently emerging embedded processor. The developed RISC-V based DLA prototype can analyze the hardware requirements according to the CNN data set through the configuration of the CNN DLA architecture, and it is possible to run RISC-V compiled software on the platform, can perform a real neural network model like Darknet. We performed the Darknet CNN model on the developed DLA prototype, and confirmed that computational overhead and inference errors can be analyzed with the DLA prototype developed by analyzing the DLA architecture for various data sets.

Keywords: convolutional neural network; deep learning accelerator; RISC-V; virtual platform; SystemC

1. Introduction

Recently, DNN in the field of Artificial Intelligence (AI) has been applied to various application fields with the high accurate inference abilities such as object detection, based on the learning of huge data set and rich computational resources. The DNN algorithm has been also applied to the application of Internet of Things (IoT) system that collects data, learning and inferences for various IoT environments. Since DNN requires a lot of data collection and computation due to its characteristics, as a result, in IoT systems, learning is mainly performed on a server, and inference is performed on an edge device [1].

Performing DNN inference on the edge computing device also requires a large number of computational resources and memory footprint compared to the limited resources of the edge device [2], so edge device suffers from processing DNN inference. Many researchers have made efforts to solve this problem in edge devices [3]. These studies have been done on CNN, which is the most widely used among many neural network models. Some studies have been conducted to optimize the neural network model such as Mobilenet [4] and Darknet [5,6], or reduce the computational overhead such as quantization of float point values for parameters [7–11]. Some studies have done the work of using an embedded GPU or adding a hardware specialized accelerator for DNN acceleration [12–14], which is so called Deep Learning Accelerator (DLA). As such, various network model optimizations and hardware accelerator designs are being conducted for CNN inference in embedded...
edge devices. To optimize the combination of these approaches in a rapidly changing field of AI applications, a system level simulator is preferred that can quickly verify a hardware accelerator at a software level, and an integrated simulation system that can apply various network models and parameters on the simulator is required.

Meanwhile, according to the diversity of IoT systems, an open-source based RISC-V processor platform [15,16] has emerged for individual customization of specific IoT systems. Since RISC-V is an open-source ISA (Instruction Set Architecture) and is maintained at a non-profit foundation, it is a processor whose development is rapidly evolving from various HW development to System Level Simulator. In the near future, RISC-V-based processors are expected to be widely applied from embedded systems to IoT edge devices. Recently, the RISC-V VP (Virtual Platform) [17] implemented by SystemC was developed, and an environment in which the RISC-V system prototype can be quickly verified at the ESL is established.

In this paper, for rapid prototyping of embedded edge device with RISC-V-based DLA, CNN DLA system was designed and implemented with SystemC at ESL level on RISC-V-based Virtual Platform (VP). The RISC-V-based DLA prototype designed in this paper has the following features. First, it is possible to analyze the requirements of hardware resource according to the CNN dataset through the configuration of the CNN DLA architecture. Specifically, it is possible to analyze the required amount of DLA internal buffer/cache according to the dataset, internal parallelism of processing element in DLA architecture, and quantization efficiency by analyzing computational overhead and inference error according to the unit of the model parameter. Second, because the designed RISC-V-based DLA prototype can run RISC-V software, it is possible to perform an actual DNN or object detection applications such as Darknet [5,18]. Using this, it is possible to analyze the performance of RISC-V up to the performance analysis of DLA according to the CNN model. In the experiment, we performed the Darknet CNN model on the RISC-V VP based DLA system prototype, and through this, we confirmed that the RISC-V based DLA prototype could execute the actual CNN model. In addition, we analyzed the performance of DLA according to buffer/cache size for various datasets, parallelism issues of processing element, and quantization efficiency, and identified that computational overhead and inference error according to model parameters can be analyzed.

2. Related Work

There are various existing studies on the system structure and accelerator structure for deep learning operations such as CNN in embedded systems such as edge devices. The DLA development at the processor architecture has an issue of whether it is better to apply the development at the RTL (Register Transfer Level) such as VHSIC Hardware Description Language (VHDL) or at the behavior level such as SystemC, in terms of its development speed and verification. Ref. [19] implemented artificial neural network architecture in both levels and showed the advantages and disadvantages of each level of abstraction. The RTL level development can take advantage of low-level details and efficient parameterization, whereas the disadvantages are as follows. It has more code lines and design time, there exists gap between algorithm design and RTL design. In addition, the redesign is required for different area versus performance tradeoffs. On the other hand, the advantages of behavior level design with SystemC are less code length and fast design time, and relatively small gap between algorithm design and hardware level system design, and easy to use, while the disadvantages of behavior level design with SystemC are limited parameterization mechanisms and limited flexibility for hardware synthesis.

Designing and implementing a deep learning algorithm is largely divided into training and inference. Training still requires high computing power and high algorithm complexity to perform deep learning, so it is difficult to perform it in an embedded edge device, whereas inference can be performed in a recent advanced embedded edge device. A lot of research at this level has been conducted.
There are several research topics that have designed and implemented an architecture for deep learning algorithm execution [13,20–24]. These studies designed a deep learning processing unit in consideration of the limited energy consumption, area, and cost to performance of the embedded edge system. At first, they designed and implemented a reconfigurable processing unit specialized for deep learning operations such as CNN and Recurrent Neural Network (RNN) at the Field-Programmable Gate Array (FGPA) level. Eyeriss [13] designed a coarse-grain reconfigurable accelerator for convolutional neural networks, and [21] is designed for executing both of convolutional neural networks and recurrent neural networks. Ref. [22] offered reconfigurable IP for artificial intelligence, and [23] proposed dynamic reconfigurable processor for deep learning models. V. Gokhanl [25] designed a deep learning platform that has a general processor and NN accelerator with interfacing to external memory.

In addition, there is an open framework for modeling and simulating various aspects of DNN accelerators such as data path, interconnection networks, performance and energy at the cycle-accurate level [26–29] without RTL simulation. Timeloop [26] is a kind of open framework that evaluates DNN accelerators architectures to obtain inference performance and energy consumption. ScaleSim [27] presented a cycle-accurate simulator on systolic arrays to model both scale up and out systems as well as analytical model to estimate the optimal scaling ratio given hardware constraints. MAESTRO [28] analyzes various forms of DNN data in an accelerator based on pre-defined inputs and generates statistics such as total latency, energy, throughput, as outputs. LAMBDA [29] is an extension of Timeloop framework focusing on the modeling of the communication and memory sub-system that greatest impact on the energy and performance of a DNN accelerator.

There are existing studies that designed and implemented a neural network accelerator at the FGPA level based on the RISC-V processor [30–33]. In [30], they designed a RISC-V core-interconnected accelerator structure which forms a coprocessor of RISC-V core. They also designed the corresponding instruction for the CNN accelerator coprocessor and the compiling environment for the instructions is developed. Ref. [31] also designed a CNN processor based on the RISC-V core, in which the CNN coprocessor has custom instructions designed to accelerate the convolution processing. Ref. [32] has investigated the possibility of extension of RISC-V ISA to accelerate the CNN processing and inference of neural network algorithm with the advantage of in-pipeline hardware and custom instructions. Ref. [33] designed the accelerator customized for a You Only Look Once (YOLO) neural network, which was operated as co-processor of RISC-V core. The hardware was verified using Xilinx Virtex-7 FGPA. There are some researches for DLAs systems [34–36] that incorporate with NVDLA accelerator [14], an open source hardware deep learning accelerator developed by NVIDIA.

Several researches were conducted at the ESL level simulation based on SystemC for neural network accelerators [37–39]. In [38], they designed and implemented deep CNN accelerator with SystemC language and Transaction-Level Modeling (TLM) with the adaption of the electronic system-level (ESL) design methodology. The designed simulator has raised abstraction level by omits of the detailed hardware design which has achieved fast and accurate simulation results in comparison with the RTL approach. In [39], they proposed AccTLMSim that provided cycle-accurate-based CNN accelerator simulator with SystemC and validated the implementation results against on the Xilinx Zynq. In addition, they proposed a performance estimation model that can speed up the design of space exploration. Table 1 summarizes various research results of deep learning accelerators.
### Table 1. List of various DNN accelerators and their approaches for architectural exploration.

| Approach                                    | List                                                                 | Features                                           |
|---------------------------------------------|----------------------------------------------------------------------|----------------------------------------------------|
| RTL with FPGA                               | Y. Chen [13]                                                          | course-grain reconfigurable                        |
|                                             | D. Shin [21]                                                          | both CNN and RNN                                   |
|                                             | Flex [22]                                                             | reconfigurable IP                                  |
|                                             | T. Fujii [23]                                                         | dynamic reconfigurable                             |
| Coprocessor with GP or RISC-V              | V. Gokhale [25]                                                       | interface with general processor and external memory|
|                                             | N. Wu [30]                                                            | core-interconnected accelerator                    |
|                                             | Z. Li [31]                                                            | support custom instruction for convolution          |
|                                             | R. Porter [32]                                                        | extension for in-pipeline hardware                 |
|                                             | G Zhang [33]                                                          | customized for Yolo                                |
| Framework                                   | Timeloop [26]                                                         | inference performance and energy                   |
|                                             | ScaleSim [27]                                                         | cycle-accurate and analytical model for scaling    |
|                                             | MAESTRO [28]                                                          | various data form for generating statistics        |
|                                             | LAMBDA [29]                                                           | support modeling for communication and memory sub-system |
| ESL with SystemC                            | Y. Lee [38]                                                           | raised abstraction level                           |
|                                             | S. Kim [39]                                                           | cycle-accurate-based                               |
|                                             | S. Lim (This Paper)                                                   | support RISC-V VP and software interface           |

### 3. RISC-V VP Based CNN DLA

#### 3.1. SystemC-Based RISC-V VP

RISC-V is an open ISS (Instruction Set Architecture) originated in U.C. Berkeley [15] and is a promising RISC-based processor architecture maintained by the open foundation. RISC-V core has already been developed and released as Core IP, and RISC-V Core IP-based processor chips are already available. However, since RISC-V is still in the developing stage, RISC-V is widely used in research based on FPGA or RTL. The research and development of the RISC-V-based CNN Architecture has also been performed based on FPGA or RTL level design, however, this approach has a disadvantage which takes relatively long time in verifying, analyzing, and optimizing the system.

Recently, the RISC-V Virtual Platform was developed based on SystemC [17], which is efficient for system verification in a relatively short time. The Virtual Platform was designed and implemented with a generic bus system using TLM 2.0 around RISC-V RV32IM core, so it is very expandable and configurable platform that can extend other TLM-connected modules for verifying special functions in the RISC-V VP environment.

The architectural overview of RISC-V virtual platform is described in Figure 1. This virtual platform basically includes a CPU Core, a TLM 2.0 Bus module, an Interrupt controller, and a Memory interface with DMA (Direct Memory Access) controlling. The CPU core module supports RISC-V’s RV32IM instruction set and provides interrupt handling and a system call interface to software running on a virtual platform. All modules are connected to TLM 2.0 based SystemC Bus module by port. The master (initiator) of the port is the CPU and its memory interface, and other modules operate as port slaves (target). The bus routes the initiator’s transaction based on the memory-mapped address and delivers it to the target port. In addition, the CPU can handle local or external interrupts. The local interrupt is handled by the CLINT (Core Local Interrupt Controller), and the external interrupt is handled by the PLIC-based IC (Interrupt Controller). On this virtual platform, software compiled with RISC-V cross-compiler can be executed. The SW is compiled by cross-compiler and then created as an ELF format executable file, which is loaded into the main memory and executed in the form of firmware. The main memory module is also connected to Bus and is used as a memory area of the software.

We developed a DLA prototype system based on the RISC-V VP platform [17] by integrating the CNN DLA module into the virtual platform. As shown in Figure 1, the CNN DLA module is connected to the TLM 2.0 Bus through the target port and is allocated to a
part of the address range of the RISC-V CPU core. The CNN DLA module is composed of an internal module that performs CNN operations such as convolution, activation, and pooling through read/write via TLM bus. The data for DNN is first loaded into main memory and transferred to DLA module through DMA operations. The CPU core controls the CNN module by writing and reading the registers allocated to the CNN address range. A virtual platform including CNN DLA can perform deep learning inference by executing a DNN applications such as Darknet. The DNN applications is compiled by RISC-V cross compiler and loaded into main memory as an ELF format executable file.

![Architecture Overview](image)

**Figure 1.** Architecture Overview of RISC-V Virtual Platform with CNN Deep Learning Accelerator.

### 3.2. CNN DLA Overview

The overall structure of the CNN DLA module is shown in Figure 2. As shown in the figure, the CNN DLA module is connected to the TLM 2.0 Bus and interacts with the CPU core and main memory through the bus. The internal architecture of CNN DLA is composed of two data structure area, that is register and buffer, and three modules, that is Data Loader, CPIPE and APIPE, in which CPIPE and APIPE is abbreviation of Convolution PIPE and Activation-Pooling PIPE, respectively.

![Internal Structure](image)

**Figure 2.** The overall structure and Interface of the CNN DLA module.

The internal data structure of the CNN DLA module includes Global Special Function Register (GSFR) and Buffer. GSFR is the memory-mapped register area that the CPU core can access by being mapped to the RISC-V address range. The DNN applications running on the virtual platform can control the CNN DLA module through the memory...
mapped GSFR area, as well as operate interrupt processing with the specific values of GSFR. The internal Buffer is a buffering space used to temporarily buffering data for CNN operation. The buffer is subdivided into image, parameter, temp, CPIPE and APIPE data buffers according to the characteristics of the data. The temp, CPIPE, and APIPE are space to temporarily store intermediate data and results of convolution, activation, or pooling.

Internal modules include Data Loader module, CPIPE module, and APIPE module. The Data Loader module reads data from the main memory to process deep learning operations such as convolution and activation, or writes back the resulting data to main memory. The CPIPE module performs the convolution operation with loaded window and weight parameters, and the APIPE module performs the activation or pooling operations. The buffer module stores data for deep learning operations or the results of deep learning operations. To parameterize and analyze the features of the buffer, we subdivide the buffer into 5 pieces; IMG, Parameter, Temp, CPIPE and APIPE.

3.3. GFSR Register Set in DLA

The DLA module operates like the controller of the RISC-V Core in the RISC-V VP platform, and the RISC-V Core can control the DLA module by accessing the DLA registers set allocated in the RISC-V memory map area, in which we call the register set as Global Function Set Register (GFSR). Accordingly, applications of deep learning network model running on RISC-V VP execute DNN with DLA module by accessing the allocated GFSR register set. The memory-mapped DLA registers and its access from software application in RISC-V VP platform is described in Figure 3.

![Figure 3](image)

**Figure 3.** The memory-mapped DLA Registers and its access from software application in RISC-V VP platform.

The register values that can be set by software are as follows. For convolution settings, filter size, stride length, parameter type, and bias type can be set. The filter size can be set from $1 \times 1$ to $3 \times 3$, and the stride length can be set to 1 or 2. In addition, the width, height, and size for the image are set, and the locations of the memory address of image and parameters such as weight and bias value are set in register set. The registers for activation and pooling configuration are also exist within the register set. We can specify the activation type as a register value, in which the activation types supported by DLA are ReLU and leaky ReLU. The pooling type also can be specified as a register value, and pooling supports min, max, and average pooling. The pooling sizes supported by the DLA are $2 \times 2$ and $3 \times 3$. 

The DNN applications running in RISC-V VP can control the operation of each sub-module of DLA through a register, so called G_CNTL in our system, and returns the result of the operation through the status register. The 32-bit G_CNTL register is configured to perform detailed operations of the DLA module by defining a value in each bit unit like as follows. The 0th bit is used as a reset bit. The first bit is used to set activate the data loader module, which means application sets it to 1 to cause the data loader module to load data from main memory to the DLA buffers. The second bit is used to set activate the CPIPE module in which CPIPE runs convolution operation of the corresponding configuration of the GFSR registers and dataset in the DLA buffers. The third bit is the bit set by DLA when convolution by CPIPE is finished. The fourth bit is the bit that activate the APIPE module in which APIPE runs activation or pooling operation for the corresponding configuration of GFSR resister values. The fifth bit is set after activation or pooling are completed by APIPE. In addition, when the operation of CPIPE and APIPE is completed, an interrupt can be generated separately to notify the software application that the corresponding job is done.

3.4. Data Loader Module and Buffer

The loader module is an interface module that is connected to the outside of the CNN DLA module. Between DLA and memory, all data for CNN is transmitted and received through the loader module, by DMA operation. Inside the loader module is described in Figure 4, in which it is internally divided into router module, data module, and requester module. Each internal module operates as an independent SystemC thread module. Those are connected to each other through a SystemC channel to perform signal sending or to synchronize with each other. This subdivided data loader module allows individual analysis of the independent behavior of the modules. We can analyze those modules in the aspect of data access pattern, data transfer time and amount, effects of buffers and pipeline issues for the data path for various DNN configurations.

![Figure 4. The internal modules decomposition of Data Loader Module.](image)

The router module is connected to the TLM bus through the target port, and it transmits and receives data through the bus in connection with the external memory module and DMA controller module. In addition, since the router module is exposed to the TLM bus, it is also responsible for handling accesses of CPU’s GFSR registers. In the router module, the neural network data such as image, parameters for weight and bias are received from memory module through external TLM bus, then those are temporarily buffered in the Buffer according to its data type. If proper data are received and buffered ready to process CNN operations, the router module sends notify signal to data module or request module. Furthermore, reversely, when the resulting data of convolution or activation operations of CPIPE or APIPE are buffered to CPIPE or APIPE buffers, the router module writes those back to memory through the TLM bus.
The data module is a module that manages the internal buffer memory. It stores data received from the router module in Buffer according to the type. When the data in the buffer is suitable for proper DNN operation such as convolution or activation, the data module sends a signal to the requester module so that it let CPIPE or APIPE module, which is outside of the data module, perform convolution operation or activation/pooling operation. Or, when the completed deep learning data are received from the requester module, it is sent to the router module to memory back.

The requester module is connected to the CPIPE and APIPE modules, outside the data loader module, through the SystemC channel to send request signals for DNN operations and to receive resulting data of DNN operations. When the requester module receives an appropriate DNN operation signal from the data module, it requests a deep learning operation to the CPIPE or APIPE module. Specifically, it requests to the CPIPE module for convolution operation, or requests to the APIPE module for activation or pooling operation.

### 3.5. CPIPE and APIPE Module

CPIPE is a module that receives data from the data loader module and performs actual convolution operation. The CPIPE module is internally composed of con2CPIPE, series of PE module, and CPIPEDone module, as shown in Figure 4. The con2CPIPE module receives data to perform convolution through a channel from the data loader module and distributes the data to the PEs to perform parallel convolution operations. Each PE in the CPIPE module performs a partial convolution operation on the part allocated to it, then it notifies the CPIPEDone module when the operation is finished.

The PE module performs partial convolution of fixed size region allocated from con2CPIPE module. Figure 5 describes the data allocation to perform partial convolution operation in each PE in CPIPE module, in which partial data allocation is dynamically adjusted by the configuration parameter such as size of filter, stride number and quantization level. Multiple PE modules can be configured to execute partial convolutions in parallel by receiving notifications through independent channels from the con2CPIPE module. The number of independent PEs can vary depending on the configuration. Since PE can perform convolution operation of a fixed size, the number of times PE is required to executed to complete convolution operations for the corresponding image depends on the size of the dataset. So, if there is data remaining to perform the convolution operation even if the PE completes its current unit convolution, the CPIPEDone module requests con2PIPE to distribute the remaining data of convolution operation to the PEs. The CPIPEDone module returns the operation result to the data loader when all partial convolution operations are completed.

![Figure 5. The internal structure for convolution processing of PE in CPIPE module.](image)
data, the load of convolution operation is differently calculated, as a result, the analysis of operational load and error are achieved in the CPIPE module.

The APIPE module performs activation and pooling operations in the neural network model. The activation operations available in the APIPE module are ReLU (Rectified Linear Unit) and Leaky ReLU, and it supports $2 \times 2$ and $3 \times 3$ pooling for pooling operation. The structure and operation method of the APIPE module is similar to that of the CPIPE module. That is, the APIPE module is also internally divided into convAPIPE module, PE modules for activation and pooling, and APIPEDone module. The convAPIPE module receives data from the Data Loader module to perform activation or pooling operation and distributes the data to the PE module to perform activation or pooling operation according to the configuration at the corresponding network layer. The PE module performs the corresponding activation or pooling operation, and when this is done, it delivers a complete signal to the APIPEDone module. The APIPEDone module requests the convAPIPE module to perform the next operation if there is data to be processed. Or, when the operation is finished, it informs the Data Loader Module that the activation or pooling operation is complete and transmits the processed data.

3.6. DNN Applications on the RISC-V DLA System

The developed RISC-V DLA system can execute ELF binary programs generated by RISC-V cross compiler. We have performed verification the developed DLA system with the Darknet [5,18] deep learning neural network application, which shows excellent performance in object detection of images. Darknet is a state-of-art neural network model for real-time object detection program implemented in C, which started to develop several years ago and continues to evolve. It is composed of dozens of network layers, and each layer performs object classification or prediction of an image while mainly performing configured convolution, activation, or pooling operations.

Although Darknet is implemented in C language, proper porting job for Darknet programming code is required to make it work in our DLA architecture. To execute each layer of the Darknet neural network model in our developed DLA, the application set the configuration values, such as image size, number of filters, stride, filter size, pooling size, required by each layer to the GFSR registers through the RISC-V VP software API. It also loads the image and parameter values used in the experiments into the main memory area of RISC-V VP, at this time the preprocessed image and parameter values having appropriate padding region are loaded. Then, it runs convolution or activation/pooling work with CPIPE or APIPE module by setting appropriate value to the G_CNTL register for the corresponding layer of the neural network. When the operation of the DLA module is finished, the application receives an interrupt signal from the DLA, identifies the memory address having result dataset from the GFSR and extracts the result data of the DNN operation.

3.7. Extension Issues

One of the important considerations in the architectural exploration phase of the DNN accelerator is power and energy consumption. Fortunately, there are SystemC add-on libraries that supports power modeling of each module and estimates their energy usage at the SystemC TLM level [40,41]. The library [41] supports static and dynamic level of power modeling of each building block, as well as estimation of wiring energy consumption by count of hamming distance of bit value changes. In addition, it is possible to set physical units for voltage, distance and area along with the operators. We can build the power modeling of each module block by using the add-on power library APIs for the developed RISC-V VP platform, operations that models the energy of the TLM payload according to changes in each module and associated bus wiring during system operation can be added and extended.

Another important consideration is architectural exploration of data communication and overhead between the memory and PEs within the accelerator, since such a large-scale NN operation is limited by communication and storage for huge amounts of data [29,42].
Interconnection network for moving data between memory and PEs greatly affects the performance of the accelerator, so it is necessary to consider this in the development of simulation. To date, we have implemented the simulator system that considers the pipelined bus for data movement between memory and PEs, which can explore the pipelined architecture between memory and PEs according to the dataset. It is also needed to consider the simulator extension of other different interconnection methods such as mesh for architectural exploration and its network analysis.

4. Verification and Analysis with Experiments

To date, we have explained the implementation of the SystemC based RISC-V DLA system prototype and running of the DNN application on the RISC-V DLA system. With several experiments for the developed DLA, first, we verified whether the actual DNN application can be performed on the RISC-V VP based DLA system. Next, we investigated whether it is possible to analyze the internal behaviors that could be critical issues in the design of the DLA architecture, such as analysis of buffer usage, parallelism of processing elements, and quantization effects within DLA architecture. In the experiment, Darknet DNN operations were performed while changing various configurations for several layers of Darknet, and the results were verified and was analyzed.

In addition, the developed SystemC-based DLA system provides methods and metrics that analyze DLA architecture quickly and easily in terms of electronic system level for AI applications. We have added the following several performance metrics to modules of the DLA to analyze the behaviors of DLA for various datasets of neural network operations. First, metrics for buffer access count and amount of data transferred were added to analyze the effect of buffers. Second, to analyze the number of for convolution, activation or pooling operations and their execution time for processing element of CPIPE or APIPE module, the execution count and execution time for each module were added as metrics. Specifically, the execution time for each module was calculated by carefully considering the modeling behavior and computational complexity of the module. While running various datasets of neural network applications on the system, we analyzed the following optimization points of DLA architecture such as buffer effects, parallelism aspect of processing elements according to dataset and quantization effects.

The developed DLA system can be executed in the operating system in which SystemC compiler and RISC-V cross-compiler are installed. Basically, the RISC-V VP is created by building with cmake development environment in Linux system with SystemC and RISC-V cross-compiler. In addition to the Linux system, we also ported the DLA system to Visual Studio to be able to setup a buildable environment on Windows systems. Therefore, it is possible to use the DLA system in both Linux and Windows environments. For verification and analysis of the DLA system, we installed Ubuntu 18.04 Linux on a computer system with Intel Core-i9 3.6 GHz processor and 32 GB DRAM, and setup a SystemC compiler and RISC-V VP cross-compiler.

4.1. Darknet Running on DLA with RISC-V VP

First, we present the results of the Darknet DNN operations. In order to show the execution result of the Darknet NNs, we selected layers 0, 1, and 13 as representative layer among of dozens of Darknet Yolo tiny v3 layers [6]. The reasons for selecting layers 0, 1, and 3 are as follows. The layer0 and layer1 perform a convolution operation and max pooling operation, respectively, using 16 filter and 3 channels for the image having largest size with 416 × 416 pixel in the Darknet Yolo tiny v3, so the effect of multi-channel and many convolutional operation of DLA with a large-sized image can be investigated with layer 0 and 1. On the contrary, the layer 13 performs a convolution operation for 256 filters and 1024 channels for small-sized image having 13 × 13 pixel for each image. It can be said that the computational load of layer 0, 1 and 13 has opposite characteristic among the layers and is heavy loads among them.
To verify the results, we compared the result images of CNN performed in RISC-V VP DLA with the result image of CNN based on original Darknet library. Figure 6 shows Experimental Results of running of layer 0, 1 and 13 of Darknet Yolo tiny v3 Neural Network Model on RISC-V VP DLA system and Original Darknet C library. In the figure, the convolution results of 416 × 416 3 channel images with 16 filters, which is layer 0, running on original Darknet C library and RISC-V VP DLA are depicted in Figure 6a,b, respectively. The result images of max pooling operations of 16 416 × 416 images of layer 0, which is layer 1, is depicted in Figure 6c,d, respectively. Figure 6e,f show the convolution results of 3 × 13 1024 channel images for original Darknet C library and RISC-V VP DLA system and, respectively. As shown in the figures, we identify that the Darknet yolo tiny v3 neural network model is properly performed by the RISC-V-compiled application in the developed RISC-V VP DLA architecture and relatively accurate results are derived. However, comparing the results of each layer with those of the original Darknet, we can see that the some of the images are slightly different. The difference comes from the difference in parameters used to perform the neural network operation.

In order to quantitatively analyze the differences between results of neural network operations, we plotted the pixel values along with the pixel position from 0 to end of pixel for the images and analyzed the similarity of the images using cosine similarity method [43]. Figure 7 shows plots of each pixel values of resulting images of original Darknet library and RISC-V VP DLA for layer 0, 1 and 13. In the figures, the x-axis represents the pixel position of images and y-axis represents the pixel value of the position. For each image, the pixel values of original Darknet library are plotted as blue points, and pixel values of RISC-V VP DLA are plotted as orange points. As shown in the figures, many pixels overlap each other, and there are a few differences at the very small region. When comparing the similarity by the cosine similarity method, the similarity is approximately 85% or higher. Particularly, the differences are made in the part where the pixel value is high or low. Compared to Figure 7a,b, the similarity is lower and the pixel differences between pixels larger in Figure 7c, because the pixel value spreads more diversely than in the previous two cases. Thus, we identify that as the absolute pixel value is getting larger, the quantification error increases.

![Figure 6](image-url)

**Figure 6.** Experimental results of running on RISC-V VP DLA system and running of Original Darknet library for layer 0, 1 and 13 of Darknet Yolo tiny v3 Nueral Network Model, which are convolution of 416 × 416 image with 16 filters (a,b), max pooling of 16 416 × 416 images (c,d), and convolution of 13 × 13 image with 1024 filters (e,f), respectively.
Figure 7. Comparison of each pixel values between RISC-V VP DLA system and Original Darknet Model running for layer 0, 1 and 13 of Darknet Yolo tiny v3 Nueral Network Model, which are convolution of $416 \times 416$ image with 16 filters (a), max pooling of $16 \times 416 \times 416$ images (b), and convolution of $13 \times 13$ image with 1024 filters (c), respectively. Blue points represent pixels for original Darknet and orange points represent pixels for RISC-V VP DLA system.

In summary, we identify from the results of running of Darknet DNN application, the RISC-V VP DLA system could perform a proper DNN application, which means it can be used for generic AI operations such as inference with the result dataset generated from the neural network operations.

4.2. Buffer Effects in DLA System

With the developed RISC-V VP DLA system, we analyzed the performance effects of DLA buffers according to NN operations. To analyze buffer effects on various configurations of NN operation, we measured the performance in the aspect of buffer access count and amount data transferred from and to memory for the configured NN operations. For the analysis, The CNN operations including convolution and max pooling are executed based on the $416 \times 416$ image and $3 \times 3$ filter size which are applied to the Darknet layer 0. To analyze the performance of buffers according to various datasets and configurations of CNN operations, the input channel number was changed to 3, 9, and 15, respectively, while the output number for each input number was also changed to 4, 8, and 16, so there are 9 datasets for NN operations.

Figure 8 plots update count of each buffer in developed DLA according to the change in each buffer size from 1 MB to 8 MB for each CNN configuration. In the DLA system, there are five separate buffers: image, parameter, temp, CPIPE, and APIPE. Among those, the image buffer is used for buffering the source image data used for NN operation. At first, the buffer effect of the image data is as follows. As shown in the figure, the image can be buffered at once regardless of the buffer size for 3 channel input and any output numbers. In case of input channel of 9, there are many buffer updates if there is only 1 MB image buffer, while all dataset is buffered at once if image buffer size is over 2 MB. Likewise, for input 15, there are several buffer updates for buffer whose size is under 2 MB, while the input can be buffered at once if buffer size is over 4 MB. If the buffer size is not as large as this input increases, buffer flushing occurs.
The Temp buffer is used for storing temporary data of convolution or activation operations, so the buffer update count increases in proportion to the number of output channels rather than number of input channels. This phenomenon can also be seen in CPIPE and APIPE buffers, in which CPIPE buffer is used for storing results of convolution operation and APIPE buffer is used for storing activation operation. For all cases, buffer update count reduces as buffer size increases. Having sufficient buffering size enough to have output results in accordance input number reduce buffer flushing overhead. We can infer the appropriate buffer size according to the configuration of the CNN operation.

The amount of data transferred between memory and each buffer according to the buffer size is shown in Figure 9. As identified from Figure 8, the image buffer is related to the number of input channel, and accordingly from Figure 9, we identify that if the image buffer has a sufficient size to hold input data, buffer flushing does not occur, so the amount of data movement is be minimized. On the other hand, for Temp, CPIPE, and APIPE, the amount of CNN operation increases in proportion to the output number, so the amount of data movement is also proportional to the output number. However, as shown in the result of Figure 8, the larger the buffer size, the lower the buffer update frequency, so the bandwidth is be increased when memory access count is reduced. In summary, we can easily analyze the buffer efficiency of DLA for various neural network configurations, and based on this, an optimal buffer architecture such as buffer size for each data set can be setup when a specific neural network is applied.
channels of Darknet layers 0 and estimated the overall execution time for each submodule with the table. For Darknet’s layer 0, the CNN job is run with inputs 3, 9, 15 and outputs 4, 8, 16 for $416 \times 416$ images and $3 \times 3$ filter sizes.

### Table 2. Estimated execution time for each submodule in DLA.

|                      | Loader Module |
|----------------------|---------------|
|                      | Memory Delay per Byte | Router | Data | Requester |
|                      | 5              | 11     | 43   | 10         |
| CIPIPE               |                |        |      |            |
| con2CIPIPE           | 4 PEs          |        |      |            |
| CIPIPEDone           |                |        |      |            |
| APIPE                |                |        |      |            |
| con2APIPE            | 4 PEs          |        |      |            |
| APIPEDone            |                |        |      |            |
|                      | 555            | 1114   | 67   | 208        |
|                      |                |        |      | 399        |
|                      |                |        |      | 100        |

The simulated execution times of each submodule of DLA for layer 0 shown in Figure 10. In the simulation results, for Darknet layer 0 with 3 input channels, we identify that the time to perform the convolution operation is similar to that of data transmission time for 1 MB buffer size, while it takes about twice as much as the data transmission time for over 2 MB buffer size. For Darknet layer 0 with 9 input channels, when the buffer size is over 8 MB, the time to perform convolution becomes about four times than that of data transmission time. However, there is no changes of ratio of time for data transmission and NN operation when input channels are 16, that is, the execution time for convolution is almost twice as that of data transmission time.

From the results we can see the following architectural inspection. If the number of input channel is small, there is no significant difference between the data movement time of dataset and the NN operation time consumed. The total execution time of consumed in each module increases as the number of output channel increases. The larger the buffer size, the larger the time difference between the two. We can estimate the ratio of data movement time and convolution operation time according to the layer of the neural network from the developed DLA and use these characteristics for architecture parallelism of DLA. In our experiment, in the case of layer 0, depending on the input–output settings and buffer size, the number of processing elements (PEs) of the convolution module can be adjusted from 1 to 4 and executed in parallel to optimize the execution time of the layer. In addition, the structure of the optimal parallelization module can be designed through pipeline operation of data movement path and operation unit considering the buffer size.

![Figure 10](image-url) It plots simulated execution times of each submodule in DLA for CNN running with inputs 256, 512, 1024 and outputs 32, 128, 245 for $13 \times 13$ images and $1 \times 1$ filter sizes.

The main purpose of the DLA system is to analyze the performance of the accelerator according to the network configuration of the DNN model. By the way, from the point of view of the user who analyzes using the actual simulator system, the wall-clock time required for the actual simulation can greatly affect the efficiency of the simulator. We measured the actual wall-clock times of the according to the various numbers of input and output for Darknet layer 0, and the measurement results are shown in Table 3. As shown in the table, actual simulation times are consumed from several seconds to several tens of seconds depending on the NN workload. It is also shown that the simulation time increases...
in proportion to the number of outputs for the same number of inputs. This is because, in our DLA system, the actual data are loaded into DRAM memory, goes through DLA module to perform CNN and activation/pooling operation, and then the results moved back to DRAM. In some cases, the simulation time may take a long time, however, most NN operations are completed within tens of seconds. Besides, it is important that our DLA system can run real Neural Network models using real data.

Table 3. Wall-Clock Simulation Time for Execution of various NN workloads.

| (# of input, # of output) | (3, 4) | (3, 8) | (3, 16) | (9, 4) | (9, 8) | (9, 16) | (15, 4) | (15, 8) | (15, 16) |
|--------------------------|--------|--------|---------|--------|--------|---------|---------|--------|---------|
| wall-clock time (s)      | 7.62   | 14.08  | 27.03   | 18.22  | 34.7   | 65.98   | 29.2    | 55.6   | 107.91  |

4.4. Quantization Effect

Quantization is applied as one of the methods to efficiently reduce computational complexity of DNN for embedded edge devices, in which it reduces data precision by quantizing the floating-point data into integer data [9,10] or fixed-point data [11] for weight parameters of NN models, then performing matrix multiply and addition with the quantized parameters. The quantization affects not only the computational complexity of the accelerator, but also data flow between memory and DLA, buffering within the DLA module, accelerator architecture of PEs operation.

The developed DLA system can be used to analyze the architectural issues of the accelerator according to the precision level, i.e., size and type of the quantized parameters. To apply the analysis of quantization in the developed DLA system, it is necessary to convert the parameters of the existing floating-point NN model into quantized data before the NN operation is executed by DLA system. When the quantized parameters are ready, we set the precision level and data length to the DLA according to the prepared quantization level. Then, NN operation is performed on the DLA using the quantized data parameter.

To show how we analyze the architectural issues with the quantized data, we extracted quantized data with integer level. Although, there may be effects on inference accuracy depending on the precision level, such as integer or fixed-point, recently, TensorRT [9] proved that 32-bit floating point data of popular NN models is possible to be quantized to 8-bits integer with little loss of inference accuracy. We extracted integer parameters for the Darknet layer 0, in which the quantization method [9] to convert from floating point data to integer data with quantization algorithm, so called log2 distribution and KL divergence. Although the quantization algorithm is an important factor in inference accuracy, it is not the main subject of this paper so detailed description of the quantization algorithm is omitted. Please refer to [9]. The quantized parameters were applied to the DLA system NN operations of Darknet layer 0 with various input/output numbers was performed, and the simulated execution times were measured for each NN operation running.

Figure 11 shows the simulated execution time in each sub-module by performing convolution and maxpooling operations of 3 NN operations according to precision level with 1M and 8M buffers, respectively; (a) input3/output16, (b) input9/output8, (c) input15/output4. As the experiments were done with 1MB and 8MB buffer size, in the case of 8bit, we identify from results (a) and (b) that the amount of data transferring is much less than PE operations if buffer is enough for buffering for some NN layer. Since the time of PE operation unit is larger than the time for data transferring, the unit operation time of the PEs is less than data movement, which means that complexity of PEs can be reduced by optimizing the number of running PEs according to the dataflow of data movement. On the other hand, for the case of 32 bit, we identify that the time consuming of PIPE operations is much more than the dataflow movement, in contrast to that of 8 bit quantized parameters. If the buffer size is sufficient and the data required for the corresponding NN operation are sufficiently buffered, the amount of data movement can be significantly reduced and the NN computation time is relatively increased. In this case, more optimized DNN can
be performed by increasing the number of PEs for the CPIPE module to increase parallel operation and overall throughput. In summary, it provides insight to determine the number of PEs to be executed according to the quantization level for the specific NN configuration. Figure 12 plots the ratio of actual data transfer to the amount of data required for NN operations according to the precision level with 1 M and 8 M buffers, respectively; (a) input3/output (4, 8, 16) and (b) input9/output (4, 8, 16) As shown in the results, when the input channel is small, the buffering efficiency is high even if the buffer size is small, whereas when the there are many input channels, it is necessary to increase the buffering efficiency by increasing the buffer size. From the results, we could obtain the insights from the DLA to determine the required buffer size and the number of running PEs for various NN configurations for the specific precision level.

5. Conclusions
Recently, the use of AI applications such as DNNs in Embedded Edge devices for IoT systems is gradually increasing. For efficient DNN operation in resource-limited edge device, embedded-optimized DLAs are widely applied. For efficient running of DNNs composed of complex networks with different characteristics for each layer, analytical design of a reconfigurable DLA structure is required.

In this paper, we developed the SystemC DLA Simulator that can easily and efficiently analyze the DLA of an embedded edge system based on RISC-V embedded processor. The developed DLA system is built into the RISC-V processor-based VP, memory interface module, buffer module, and PE modules that can execute CNN through convolution, ReLU pooling, activation, etc. We can execute RISC-V-compiled DNN applications on the RISC-V VP having DLA, and while executing each layer, the various architectural issues of DLA, such as buffer size required in DLA, execution time for each module along with the data path, number PEs for parallel operations, can be analyzed. In addition, it is possible to analyze DNN optimization techniques for data quantization. The developed DLA system can provide an easy and efficient analysis method for various DNN applications. We developed the DLA system based on the open-source RISC-V VP. In the future, we plan to
apply more DNN applications using the developed DLA system and do research on a DLA architecture that can optimize DNN applications.

The developed system has some limited features. First, it cannot provide power modeling and energy estimation feature of the accelerator. Based on the aforementioned TLM add-on library that provides power modeling and energy estimation at various modes, we plan to expand it to model the power of each module and estimate energy according to usability.

Author Contributions: Conceptualization, S.-H.L. and W.W.S.; methodology, S.-H.L.; software, J.-Y.K.; validation, S.-H.L. and J.-Y.K.; formal analysis, S.-H.L. and S.-Y.C.; investigation, S.-Y.C.; resources, S.-H.L., W.W.S. and J.-Y.K.; data curation, J.-Y.K.; writing—original draft preparation, S.-H.L.; writing—review and editing, S.-H.L. and W.W.S.; visualization, S.-H.L.; supervision, S.-H.L. and W.W.S.; project administration, W.W.S. and S.-Y.C.; funding acquisition, S.-Y.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Ministry of Science and MSIT (ICT), Korea, under the National Program for Excellence in SW, supervised by the IITP (Institute of Information & communication Technology Planning & Evaluation) (2019-0-01816). This work was funded by Genesys Logic, Inc. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2019R1F1A1057503, NRF-2021R1F1A1048026). This work was supported by Hankuk University of Foreign Studies Research Fund.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations
The following abbreviations are used in this manuscript:

| Abbreviation | Description |
|--------------|-------------|
| CNN          | Convolutional Neural Network |
| ESL          | Electronic System Level |
| DLA          | Deep Learning Accelerator |
| DNN          | Deep Neural Network |
| AI           | Artificial Intelligence |
| IoT          | Internet of Thing |
| ISA          | Instruction Set Architecture |
| VP           | Virtual Platform |
| RTL          | Register Translation Level |
| VHDL         | VHSIC Hardware Description Language |
| RNN          | Recurrent Neural Network |
| FPGA         | Field-Programmable Gate Array |
| GP           | General purpose Processor |
| YOLO         | You only look once |
| TLM          | Transaction-Level Modeling |
| GFSR         | Global Function Set Register |
| CPipe        | Convolution PIPE |
| APIPE        | Activation-Pooling PIPE |
| PE           | Processing Element |

References
1. Yang, Q.; Luo, X.; Li, P.; Miyazaki, T.; Wang, X. Computation offloading for fast CNN inference in edge computing. In Proceedings of the ACM Conference on Research in Adaptive and Convergent Systems (RACS’19), Chongqing, China, 24–27 September 2019; pp. 101–106.
2. Véstias, M.P. A Survey of Convolutional Neural Networks on Edge with Reconfigurable Computing. *Algorithms* 2019, 12, 154. [CrossRef]
3. Zhang, Q.; Zhang, M.; Chen, T.; Sun, Z.; Ma, Y.; Yu, B. Recent advances in convolutional neural network acceleration. *Neurocomputing* 2019, 323, 37–51. [CrossRef]
4. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* 2017, arXiv:1704.04861.
5. Redmon, J.; Farhadi, A. Yolo9000: Better, faster, stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
6. Darknet. Available online: https://pjreddie.com/darknet/ (accessed on 1 July 2020).

7. Marchisio, A.; Hanif, M.A.; Khalid, F.; Plastiras, G.; Kyrkou, C.; Theoecharides, T.; Shafique, M. Deep Learning for Edge Computing: Current Trends, Cross-Layer Optimization, and Open Research Challenges. In Proceedings of the 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Miami, FL, USA, 15–17 July 2019; pp. 553–559.

8. Chen, Y.; Xie, Y.; Song, L.; Chen, F.; Tang, T. A Survey of Accelerator Architectures for Deep Neural Networks. *Engineering* **2020**, *6*, 264–274. [CrossRef]

9. Migacz, S. 8-bit Inference with TensorRT. In Proceedings of the NVIDIA GPU Technology Conference, Silicon Valley, CA, USA, 8–11 May 2017.

10. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 2704–2713.

11. Jain, S.; Venkataramani, S.; Srinivasan, V.; Choi, J.; Chuang, P.; Chang, L. Compensated-DNN: Energy efficient low-precision deep neural networks by compensating quantization errors. In Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference, San Francisco, CA, USA, 24–28 June 2018.

12. Shawahna, A.; Sait, S.M.; El-Maleh, A. FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. *IEEE Access* **2019**, *7*, 7823–7859. [CrossRef]

13. Chen, Y.; Yang, T.; Emer, J.; Sze, V. Eyriress v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2019**, *9*, 292–308. [CrossRef]

14. NVIDIA. NVIDIA Deep Learning Accelerator. Available online: https://nvdla.org (accessed on 1 July 2020).

15. Waterman, A.; Asanović, K. *The RISC-V Instruction Set Manual; Volume I: User-Level ISA*; SiFive Inc. and CS Division, EECs Department, University of California, Berkeley: 2017.

16. Waterman, A.; Asanović, K. *The RISC-V Instruction Set Manual; Volume II: Privileged Architecture*; CS Division, EECs Department, University of California: Berkeley, CA, USA, 2017.

17. Herdt, V.; Große, D.; Le, H.M.; Drechsler, R. Extensible and Configurable RISC-V Based Virtual Prototype. In Proceedings of the 2018 Forum on Specification and Design Languages (FDL), Garching, Germany, 10–12 September 2018; pp. 5–16.

18. Redmon, J.; Farhadi, A. Yolov3: An Incremental Improvement. *arXiv 2018*, arXiv:1804.02767.

19. Aledo, D.; Schafer, B.C.; Moreno, F. VHDLS vs. SystemC: Design of Highly Parameterizable Artificial Neural Networks. *IEICE Trans. Inf. Syst.* **2019**, *E102.D*, 512–521. [CrossRef]

20. Abdelouahab, K.; Pelcat, M.; Sérot, J.; Berry, F. Accelerating CNN inference on FPGAs: A Survey. *arXiv 2018*, arXiv:1806.01683.

21. Shin, D.; Lee, J.; Lee, J.; Yoo, H. 14.2 DNPU An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks. In Proceedings of the 2017 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 5–9 February 2017; pp 240–241.

22. Flex Logix Technologies, Inc. *Flex Logic Improves Deep Learning Performance by 10X with New EFLX4K AI eFPGA Core*; Flex Logix Technologies, Inc.: Mountain View, CA, USA, 2018.

23. Fujii, T.; Toi, T.; Tanaka, T.; Togawa, K.; Kitaoka, T.; Nishino, K.; Nakamura, N.; Nakahara, H.; Motomura, M. New Generation Dynamically Reconfigurable Processor Technology for Accelerating Embedded AI Applications. In Proceedings of the 2018 IEEE Symposium on VLSI Circuits, Honolulu, HI, USA, 18–22 June 2018; pp. 41–42.

24. Wang, Y.; Xu, J.; Han, Y.; Li, H.; Li, X. DeepBurning: Automatic generation of FPGA-based learning accelerators for the Neural Network family. In Proceedings of the 2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, 5–9 June 2016.

25. Gokhale, V.; Jin, J.; Dundar, A.; Martini, B.; Culurciello, E. A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops, Columbus, OH, USA, 23–28 June 2014; pp. 696–701.

26. Parashar, A.; Raina, P.; Shao, Y.S.; Chen, Y.-H.; Ying, V.A.; Mukkara, A.; Venkatesan, R.; Khailany, B.; Keckler, S.W.; Emer, J. Timeloop: A systematic approach to dnn accelerator evaluation. In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Madison, WI, USA, 24–26 March 2019; pp. 304–315.

27. Samajdar, A.; Joseph, J.M.; Zhu, Y.; Whatmough, P.; Mattina, M.; Krishna, T. A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim. In Proceedings of the 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Boston, MA, USA, 23–25 August 2020; pp. 58–68.

28. Kwon, H.; Chatrasi, P.; Sarkar, V.; Krishna, T.; Pellauer, M.; Parashar, A. MAESTRO: A Data-Centric Approach to Understanding Reuse, Performance, and Hardware Cost of DNN Mappings. *IEEE Micro* **2020**, *40*, 20–29. [CrossRef]

29. Russo, E.; Palesi, M.; Monteleone, S.; Patti, D.; Ascia, G.; Catania, V. LAMBDA: An Open Framework for Deep Neural Network Accelerators Simulation. In Proceedings of the 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), Kassel, Germany, 22–26 March 2021; pp. 161–166.

30. Wu, N.; Jiang, T.; Zhang, L.; Zhou, F.; Ge, F. A Reconfigurable Convolutional Neural Network-Accelerated Coprocessor Based on RISC-V Instruction Set. *Electronics* **2020**, *9*, 1005. [CrossRef]
31. Li, Z.; Hu, W.; Chen, S. Design and Implementation of CNN Custom Processor Based on RISC-V Architecture. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications, IEEE 17th International Conference on Smart City, IEEE 9th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019; pp. 1945–1950.

32. Porter, R.; Morgan, S.; Biglari-Abhari, M. Extending a Soft-Core RISC-V Processor to Accelerate CNN Inference. In Proceedings of the 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 5–7 December 2019; pp. 694–697.

33. Zhang, G.; Zhao, K.; Wu, B.; Sun, Y.; Sun, L.; Liang, F. A RISC-V based hardware accelerator designed for Yolo object detection system. In Proceedings of the 2019 IEEE International Conference of Intelligent Applied Systems on Engineering (ICIASE), Fuzhou, China, 26–29 April 2019.

34. Venkatesan, R.; Shao, Y.S.; Zimmer, B.; Clemons, J.; Fojtk, M.; Jiang, N.; Keller, B.; Klinefelter, A.; Pinckney, N.; Raina, P.; et al. A 0.11 PJ/OP, 0.32-128 Tops, Scalable Multi-Chip-Module-Based Deep Neural Network Accelerator Designed with A High-Productivity VLSI Methodology. In Proceedings of the 2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, 18–20 August 2019; IEEE Computer Society: Washington, DC, USA, 2019.

35. Feng, S.; Wu, J.; Zhou, S.; Li, R. The Implementation of LeNet-5 with NVDLA on RISC-V SoC. In Proceedings of the 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 18–20 October 2019.

36. Giri, D.; Chiu, K.-L.; Eichler, G.; Mantovani, P.; Chandramoorthy, N.; Carloni, L.P. Ariane+ NVDLA: Seamless third-party IP integration with ESP. In Proceedings of the Workshop on Computer Architecture Research with RISC-V (CARRV), Valencia, Spain, 29 May 2020.

37. Bailey, B.; Martin, G.; Piziali, A. ESL Design and Verification: A Prescription for Electronic System Level Methodology; Morgan Kaufmann/Elsevier: Amsterdam, The Netherlands, 2007.

38. Lee, Y.; Hsu, T.; Chen, C.; Liou, J.; Lu, J. NNSim: A Fast and Accurate SystemC/TLM Simulator for Deep Convolutional Neural Network Accelerators. In Proceedings of the 2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, Taiwan, 22–25 April 2019; pp. 1–4.

39. Kim, S.; Wang, J.; Seo, Y.; Lee, S.; Park, Y.; Park, S.; Park, C.S. Transaction-level Model Simulator for Communication-Limited Accelerators. arXiv 2020, arXiv:2007.14897.

40. Vece, G.B.; Conti, M. Power estimation in embedded systems within a systemc-based design context: The pktool environment. In Proceedings of the 2009 Seventh Workshop on Intelligent solutions in Embedded Systems, Ancona, Italy, 25–26 June 2009; pp. 179–184.

41. Greaves, D.; Yasin, M. TLM POWER3: Power estimation methodology for SystemC TLM 2.0. In Proceeding of the 2012 Forum on Specification and Design Languages, Vienna, Austria, 18–20 September 2012; pp. 106–111.

42. Nabavinejad, S.M.; Baharloo, M.; Chen, K.-C.; Palesi, M.; Kogel, T.; Ebrahimi, M. An Overview of Efficient Interconnection Networks for Deep Neural Network Accelerators. IEEE J. Emerg. Sel. Top. Circuits Syst. 2020, 10, 268–282. [CrossRef]

43. Cosine Similarity. Available online: https://en.wikipedia.org/wiki/Cosine_similarity (accessed on 1 December 2020).