Energy-efficient Hardware Accelerator Design for Convolutional Neural Network

Yesung Kang

Department of Electrical Engineering

Graduate School of UNIST

Energy-efficient Hardware Accelerator Design for Convolutional Neural Network

A dissertation
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Yesung Kang

6 / 18 / 2019

Approved by

Advisor
Kyung Rok Kim

Energy-efficient Hardware Accelerator Design for Convolutional Neural Network

Yesung Kang

This certifies that the dissertation of Yesung Kang is approved.

6 / 18 / 2019

signature

Advisor: Kyung Rok Kim

signature

Co-Advisor: Seokhyeong Kang

signature

Seong-Jin Kim

signature

Jongeun Lee

signature

Youngmin Kim

DEDICATION

- *To my wife, Jisoo, without whose sacrifice, love and prayer I would not have finished my dissertation.*
- *To my beloved two sons, Eunchan and Eungyeom, I would like to express my thanks for their smiles and cheering.*

Abstract

Convolutional neural network (CNN) is a class of deep neural networks, which shows a superior performance in handling images. Due to its performance, CNN is widely used to classify an object or detect the position of object from an image. CNN can be implemented on either edge devices or cloud servers. Since the cloud servers have high computational capabilities, CNN on cloud can perform a large number of tasks at once with a high throughput. However, CNN on the cloud requires a long round-trip time. To infer an image picture, data from a sensor should be uploaded to the cloud server, and processed information from CNN is transferred to the user. If an application requires a rapid response in a certain situation, the long round-trip time of cloud is a critical issue. On the other hand, an edge device has a very short latency, even though it has limited computing resources. In addition, since the edge device does not require the transmission of images over network, its performance would not be affected by the bandwidth of network. Because of these features, it is efficient to use the cloud for CNN computing in most cases, but the edge device is preferred in some applications. For example, CNN algorithm for autonomous car requires rapid responses. CNN on cloud requires transmission and reception of images through network, and it cannot respond quickly to users. This problem becomes more serious when a high-resolution input image is required. On the other hand, the edge device does not require the data transmission, and it can response very quickly. Edge devices would be also suitable for CNN applications involving privacy or security. However, the edge device has limited energy resource, the energy efficiency of the CNN accelerator is a very important issue. Embedded CNN accelerator consists of off-chip memory, host CPU and a hardware accelerator. The hardware accelerator consists of the main controller, global buffer and arrays of processing elements (PE). It also has a separate compression module and activation module. In this dissertation, we propose energy-efficient design in three different parts.

First, we propose a time-multiplexing PE to increase the energy efficiency of multipliers. From the fact the feature maps have small values which are defined as non-outliers, we increase the energy efficiency for computing non-outliers. For further improving the energy effciency of PE, approximate computing is also introduced. Method to optimize the trade-off between accuracy and energy is also proposed.

Second, we investigate the energy-efficient accuracy recovery circuit. For the implementation of CNN on edge, CNN loops are usually tiled. During tiling of CNN loops, accuracy can be degraded. We analyze the accuracy reduction due to tiling and recover accuracy by extending et al. of partial sums with very small energy overhead.

Third, we reduce energy consumption for DRAM accessing. CNN requires massive data transmission between on-chip and off-chip memory. The energy consumption of data transmission accounts

for a large portion of total energy consumption. We propose a spatial correlation-aware compression algorithm to reduce the transmission of feature maps.

In each of these three levels, this dissertation proposes novel optimization and design flows which increase the energy efficiency of CNN accelerator on edge.

Contents

List of Figures

List of Tables

ACKNOWLEDGMENT

Throughout the writing of this dissertation I have received a great deal of support and assistance.

First and foremost, I thank my academic advisor, Professor Seokhyeong Kang, for his patience, motivation and enthusiasm. He has provided insightful discussions about the research. He is someone you will instantly love and never forget once you meet him. I could not have imagined having a better mentor for my Ph.D course.

I would like to express my deepest gratitude to my advisor, Professor Kyung Rok Kim, for his guidance and encouragement. I met him in my junior year. His guidance helped to broaden my understanding of device physics.

I gave my sincere thanks to Professor Younmin Kim. He is my first mentor in digital circuit design. I was encouraged by him to pursue a Ph.D. He taught me to perceive the philosophy.

I would like to express my very great appreciation to my dissertation committee member, Prof. Seong-Jin Kim and Prof. Jongeun Lee for their encouragement and professional guidance. I am grateful for their valuable feedback.

I would also like to thank colleagues in Professor Kang's CAD & SoC Design Lab: Dr. Seungwon Kim, Sunmean Kim, Daeyeon Kim, Yoonho Park, Sunghoon Kim, Sungyun Lee, Sunghye Park, Eunji Kwon, Taeho Lim, Jaewoo Kim, Mingyu Woo, Sanggi Do and Jaemin Lee, for the stimulating discussions, for the sleepless nights we were working together, and for all the memory we have had in the last six years. Also I thank my friends: Dasom Jeong, Muyoung Lee, Moohyeon Nam, Byeongju Han and Kiyoung Jo. I wish them the best in their future.

Last but not the least, I would like to thank my family: my parents, Wonjung Kang and Sunhyang Park, parents-in-law, Youyoung Kim and Sunghyun Paik, and the rest of my families, Yewon Kang and Yechan Kang, for their endless love, prayers and caring. I am very thankful to my wife, Jisoo Kim, whose sacrifice, love and prayer allowed me to finish my Ph.D. course. To my beloved two sons, Eunchan Kang and Eungyeom Kang, I would like to express my thanks for their smiles and cheering. Finally I thank God. I will keep on trusting you for my future.

VITA

1991	Born, Busan, South Korea
2013	B.Sc., Electrical Engineering, Ulsan National Institute of Science and Technology, Ulsan, South Korea
2019	Ph.D., Electrical Engineering, Ulsan National Institute of Science and Technology, Ulsan, South Korea

- I. J. Chang, **Yesung Kang**, and Y. Kim, "Channel Length Biasing for Improving Read Margin of the 8T SRAM at Near Threshold Operation", *Electronics*, 8(6), (2019) pp.611.
- **Yesung Kang**, J. Kim, S. Kim, S. Shin, E. Jang, J. Jeong, K. Kim and S. Kang, "A Novel Ternary Multiplier based on Ternary CMOS Compact Model", *Proc. ISMVL*, 2017, pp.25-30.
- **Yesung Kang**, J. Kim and S. Kang, "A Novel Approximate Synthesis Flow for the Energy-Efficient FIR filter", *Proc. ICCD*, 2016, pp.96-102.
- S.-Y. Kim, K. Kim, Y. H. Hwang, J. Park, J. Jang, Y. Nam, **Yesung Kang**, H. J. Park, Z. Lee, J. Choi, Y. Kim, S. Jeong, B.-S. Bae and J.-U. Park, "High-resolution Electrohydrodynamic Inkjet Printing of Stretchable Metal Oxide Semiconductor Transistors with High Performance", *Nanoscale*, 8(39), (2016) pp.17113-17121.
- **Yesung Kang**, J. Choi and Y. Kim, "A Wide-Range On-Chip Leakage Sensor Using a Current–Frequency Converting Technique in 65-nm Technology Node", *IEEE Trans. on CAS II*, 62(9), (2015) pp. 846-850.
- J. Lee, **Yesung Kang** and Y. Kim, "Analysis of On Chip Decoupling Capacitor in the Double-gate FinFETs with PEEC-based Power Delivery Network", *Proc. ISOCC*, 2014, pp.290-291.
- **Yesung Kang** and Y. Kim, "Intra-gate Length Biasing for Leakage Optimization in 45nm Technology Node", *IEICE Trans. Fundamentals*, 96(5), (2013) pp. 947-952.
- M. Ryu, **Yesung Kang** and Y. Kim, "Transistor Layout Optimization for Leakage Saving", *Proc. ISOCC*, 2013, pp.253-254.
- D. Kim, **Yesung Kang**, M. Ryu and Y. Kim, "Simple and Accurate Capacitance Modeling of 32nm Multi-fin FinFET", *Proc. ISOCC*, 2013, pp.392-393.

- D. Kim, **Yesung Kang** and Y. Kim, “Simple and Accurate Modeling of Double-Gate FinFET Fin Body Variations”, *Proc. SMACD*, 2012, pp.265-268.

Chapter I

Introduction

1.1 Developments of Artificial Neural Networks (ANNs)

1.1.1 Neural networks exploiting perceptron

Artificial neural networks (ANNs) are statistical learning algorithms inspired by the brain of the animal. ANN consists of a set of artificial neurons. Each artificial neuron is connected to each other by synapses. Using this synapse, artificial neurons transmits the signal. The transmitted signals are weighted by the neurons by adjustable weights. Adjustable weights mean the coupling strength between neurons, which operate during training or forecasting. Artificial neural network is trained by changing the weight of the synapses to have the ability to perform tasks.

McCulloch et al. first proposed a computational model for neural networks [3]. In the late 1940s, psychologist Donald Hebb proposed "hebbian learning" [4]. Hebbian learning is a typical self-study, and its variations become early models of long term enhancement. This idea began in 1948 with the application of the computational model to Turing's B-type machine. Pali et al. first used a computational model to simulate a hebbian network at MIT [5]. Frank Rosenblatt created an algorithm for pattern recognition based on the perceptron [6] which can perform simple computational functions like addition and subtraction. The perceptron consists of neurons, weights, biases, and activation functions. The neuron is the smallest element that constitutes an artificial neural network. If a net value is larger than the threshold, '1' is output while it is activated, and in the opposite case, '0' is output. The weight is a value that indicates the direction or shape of this linear boundary. The perceptron can classify into two classes using the pre-trained weight. The bias represents the y-intercept of a linear boundary. The activation function returns '1' if the weighted sum is greater than the threshold value. Otherwise, it

returns ‘0’.

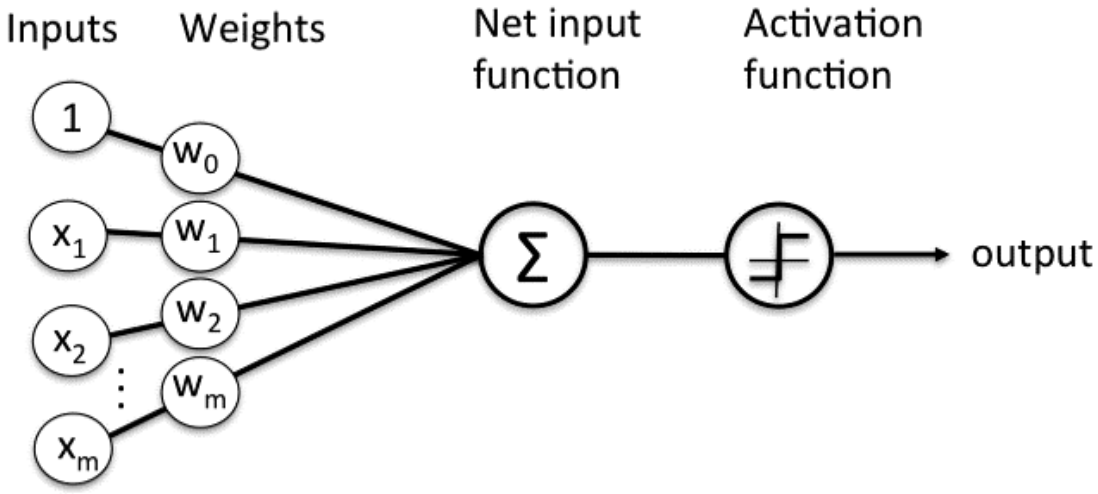


Figure 1.1: Schematic of the perceptron [9].

The schematic of the perceptron is shown in Figure 1.1. As shown in this figure, the inputs are weighted and summed by the perceptron. The function of the perceptron can be described by the following equation:

$$Z_i = f(\sum_{k=0}^{M} x^k w^k + b) \tag{1.1}$$

, while x^k and w^k are input, weight from k-th input neuron. b and f are bias and activation function.

M. Marvin et al. published two problems of the perceptron. The first was that the computer didn’t have enough computing resource to handle huge neural networks. As technology developed, computing performance also improved, and this problem was solved naturally. The second problem was that the single-layer neural network cannot handle exclusive-OR (XOR) problem.

To solve XOR problem, multilayer perceptron (MLP) was proposed. The previous single layer perceptron has only an input layer and an output layer, while an MLP has one or more hidden layers. MLP has the following characteristics. MLP has no connections within a layer. All input and output layers are connected through the hidden layers and no direct connections between them. Two consecutive layers are fully connected. The perceptrons of MLP is identical to that of a single layer. In 1975, the training method for MLP is proposed by P. Werbos. He adjust the weights at each node by considering the back-propagation of the error.

The accuracy of MLP is highly affected by the number of hidden layers. If the number of hidden layers is small, the CNN will quickly converge on local optima, but it will show low accuracy. We can increase the accuracy by increasing the number of hidden layers. However, the increase in the number

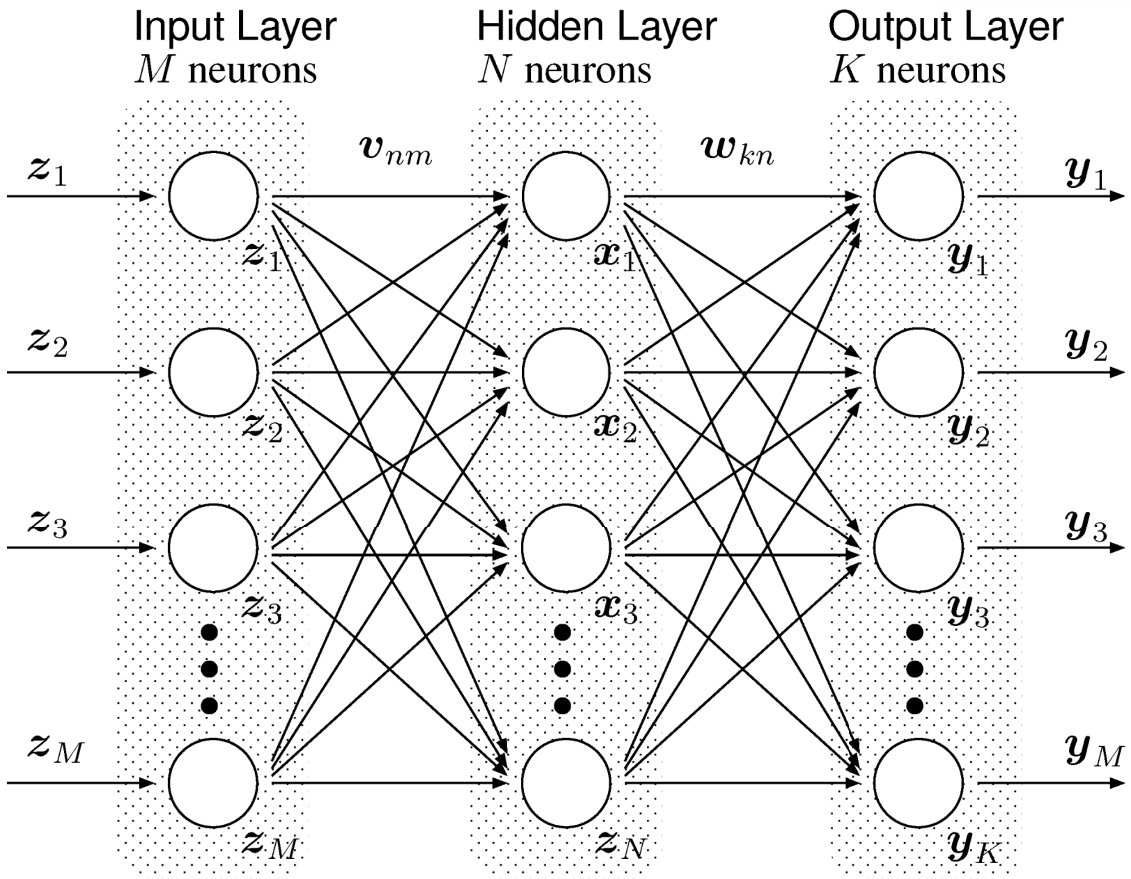


Figure 1.2: Architecture of multilayer perceptron (MLP) [7].

of hidden layers requires more operations and more time for training.

1.1.2 Convolutional neural networks

In 1989, Y. LeCun et al. first proposed the convolutional neural network (CNN), which can alleviate the above problem by weight sharing [11, 12]. The CNN typically consists of convolution layers, pooling layers, and fully connected layer. The parameters of the convolution layer consist of a series of trainable weights and biases. Usually, the size of weights are less than 3×3 because the output feature maps of CNN are connected only to the local region of the input feature maps, while the output feature maps of the MLP are connected to all input feature maps. This is why the CNN can reduce the size of weights by sharing. During the forward propagation, each weight slides the horizontal and vertical dimensions of the input volume and creates two-dimensional output feature maps. When sliding the weights over the input, dot products are made between the filter and the elements of the input. Intuitively, weights activates on a specific pattern at a specific location in the input feature maps.

The accumulation of this activation map along the depth dimension is the output feature maps. The size of output feature maps is determined by three hyper-parameters: depth, stride, and zero-padding. The width of the output volume, E, is calculated as a function of the width of the input feature maps (W), the width of weights (R), stride (S) of the CONV layer, and the zero-padding size (P):

$$E = (W - R + 2P)/S + 1 \tag{1.2}$$

If this value is not an integer, stride is incorrectly set. In this case, it is impossible for the neurons to be symmetrical and neatly arranged. The following example will make this equation more intuitive: The height of the output feature maps also can be calculated by similar method.

Recently, CNNs have shown great performance in various fields, such as computer vision, speech recognition, natural language processing, and the autonomous car. In ImageNet 2012 challenge, a CNN called AlexNet [40], has achieved a top-5 accuracy of 84.7% while the accuracy of second place, which was not a CNN, was around 26.2%. From 2012, various CNN architectures have been proposed. Although the accuracy of CNN has continued to improve, but the size of the network has also continued to grow. Figure 1.3 shows the accuracy and the number of operations of state-of-the-art CNNs, which is trained for ImageNet classification [75]. ResNet-101, which has the highest top-5 accuracy, 97.6%, requires 19 billions of operations and 829 millions of parameters [2].

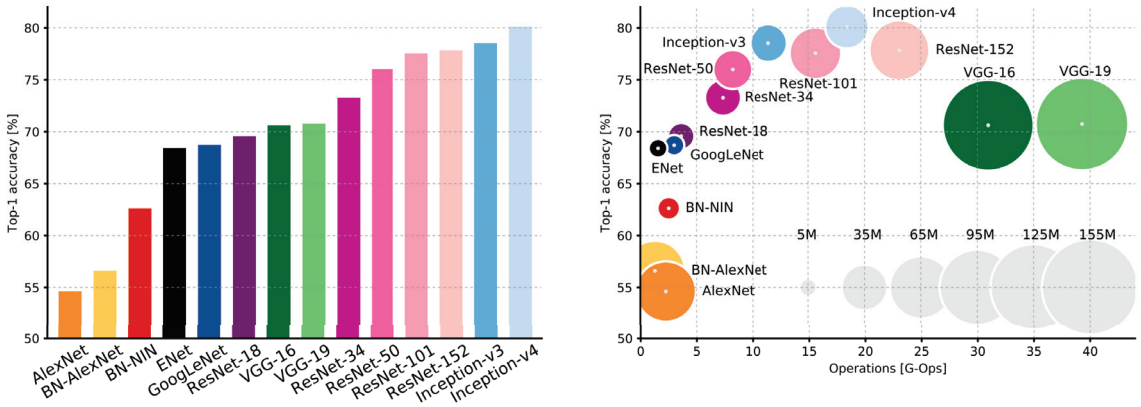


Figure 1.3: Accuracy vs. network size [1].

1.2 Deploying CNN on Edge Devices

CNNs can be implemented on either edge or cloud. Usually, the cloud has rich computing resource and memory. Due to its rich computing resources, training CNN is much faster. It is easier to collect data from sensor nodes. The inference on the cloud is also much faster than on the edge. Although cloud computing has the aforementioned advantage, edge devices are preferred in some application where the following features are emphasized.

(1) Latency and Bandwidth. For the inference of CNN with cloud computing, captured data from sensors is required to be sent to a remote server. Transferred data is inferred by cloud computer and the inferred results should be send back to the local device. The latency can be critical for the applications which require rapid response. Take an autonomous car as an example. In case of emergency, the car should immediately handle the situations, like braking car or avoiding obstacles. If the round trip time (RTT) is long, the car cannot provide an immediate response. On the other hand, an edge device can provide rapid response because it does not need to send or receive data over the network. For the application which has limited bandwidth or numerous node devices, edge devices are preferred. If the number of node device becomes larger, the effective bandwidth decreases which result in performance reduction of CNN tasks. Edge devices can alleviate the problem.

(2) Security and Privacy. Compared to edge devices, the central server is more prone to attacks or hacks. Edge devices are also effective for CNN tasks that deal with information that the user would not want to disclose, such as medical privacy.

(3) Customization. If CNN task needs customization, an edge device is more effective, because hosting multiple classifiers on the cloud would be expensive.

For the aforementioned applications, CNN on edge device has several advantages. However, CNN on edge devices is always thirst for energy. The-state-of-the-art CNNs have been developed with the primary goal of improving accuracy, so that network size becomes increasingly larger. As the network size increases, CNN requires more energy consumption due to increased computations and memory access.

1.3 Energy-efficient CNN Accelerator

1.3.1 Circuit level approach

CNNs contains a lot of multiplication and accumulation operations. Inference of state-of-the-art CNNs require from few giga operations to few tens of giga operations. To accelerate the CNN on edge devices with a limited power resource, the energy efficiency of massive operations is a very important problem. The analysis on statics of feature maps and weights are helpful for designing energy-efficient circuits. Feature maps and weights consist of outlier and non-outlier, of which magnitude is large and small respectively. Most of feature maps and weights are non-outliers and only few of them are outliers. Time-multiplexing multiplier can greatly reduce the energy consumption used for the computation of non-outlier. Approximate computing can further reduce energy consumption. Due to CNN's error-resilient properties, appropriately configured approximate circuit can save a lot of energy consumption without CNN accuracy degradation.

1.3.2 System level approach

The-state-of-the-art CNN requires a larger amount of memory than the on-chip memory that the typical CNN accelerator has. To implement CNN on embedded CNN accelerator, CNN loops are appropriately tiled. During the tiling of channel loop, CNN accuracy can be reduced. If we increase the bit width for partial sums, accuracy would increase for additional memory access and power consumptions. To minimize the energy overhead for accuracy recover, we propose an accuracy recovery method with lower memory overhead and energy consumptions.

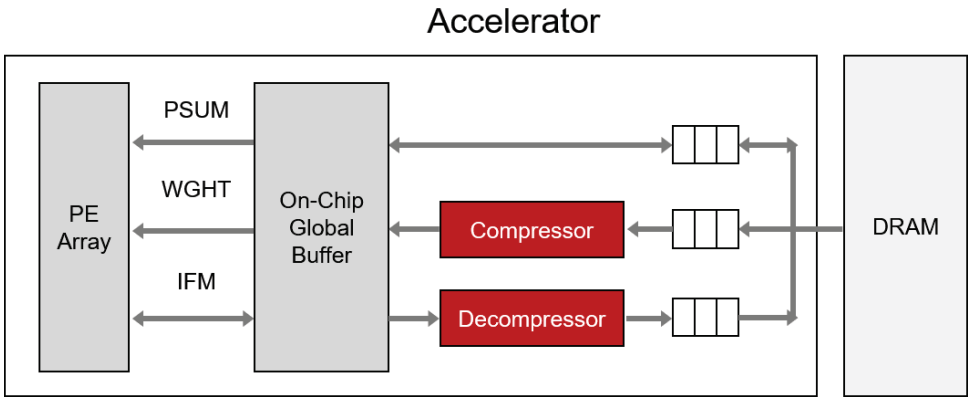


Figure 1.4: Memory compression for reducing DRAM access.

1.3.3 Memory level approach

CNN operations require massive traffic between on-chip and off-chip memory. Since accessing off-chip memory requires about hundreds of times more energy than accessing local memory, energy consumption for off-chip memory occupies a large part of the overall CNN energy consumption. Compressing data before sending to off-chip memory can effectively save DRAM access as shown in Fig.1.4

1.4 This Dissertation

In this dissertation, several innovative techniques are proposed to improve the energy efficiency of CNN accelerator for edge device in three different levels. The remainder of this thesis is organized as follows.

- Chapter II proposes two circuit level method to increase energy efficiency of a CNN accelerator. First, we propose a time-multiplexing multiply-and-accumulator (MAC) for which bit width is adjustable according to filter and feature map. The proposed MAC has relatively small bit width and bit width is adjustable by time-multiplexing. If the both filter and feature map are non-outliers, MAC computes the result in a cycle with small-bit width to increase the energy efficiency. If either filter or feature map, or both are outliers, MAC computes in multiple cycles. The proposed multipliers consume more energy for the computation of outliers. However, outliers occur very rarely, the average energy consumption of CNNs reduces. Chapter II also proposes a method of adjusting the accuracy of the subcircuit to improve energy efficiency while maintaining the accuracy of the circuit. By reducing design space effectively, the proposed method successfully find the optimum results.

- Chapter III proposes an error analysis due to loop tiling. We describes the process of error generation by loop tiling. We also proposes a error recovery circuit which can be executed with very lower energy overhead.

- Chapter IV proposes a memory compression algorithm for feature map to reduce energy consumption for DRAM access. The proposed algorithm compress data by considering the spatial correlation of feature maps with small circuit overhead.

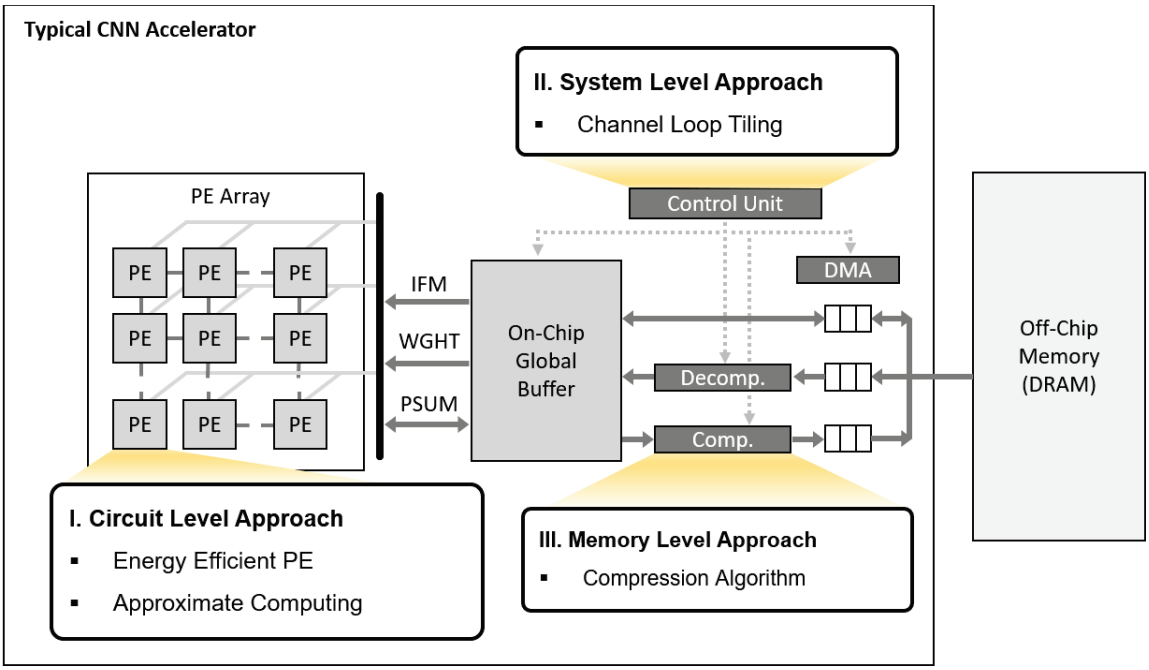


Figure 1.5: Scope and organization of this dissertation.

Chapter II

Circuit Level Approach

2.1 Energy-efficient Processing Element

Convolutional neural networks (CNNs) contain a lot of multiplication and accumulation operations. The computation of state-of-the-art CNNs require from few giga operations to few tens of giga operations. To accelerate the CNN on embedded devices with a limited power resource, energy efficiency of massive operations is an important issue. The problem has been extensively studied and numerous designs for hardware accelerator have been proposed. Reducing precision is one of solution to reduce energy consumption.

According to our observations, most of feature maps and filters have small absolute values except a few large absolute values, which are defined as 'outliers'. The number of outliers is small but has a large effect on the network quality [13]. Although the most of feature maps and weights can be computed by small bit-width multiply-and-accumulators (MACs), the bit width of MACs is chosen by the considering range of outliers, due to its effect on network accuracy. For most of cycles, large bit-width MACs is wasting area and energy. To handle this problem, B. Moons et al. and H. Sharma et al. have proposed precision scalable accelerators [14, 15].

However, these approaches demand decrease of accuracy and investigation of required precision of each layer for a target accuracy. Additionally, adjusting bit width of MACs in a layer is impossible. E. Park et al. have proposed outlier-aware CNN accelerator. They have proposed an accelerator which have dedicate PEs and MACs to compute outliers [16]. Because the ratio of outliers is different for each layer, the ratio of outlier PEs and outlier MACs must be carefully determined. Otherwise, MACs and PEs would stall. We propose a novel energy-efficient accelerator, which fully exploiting MACs. Non-outliers are computed in a cycle with reduced bit-width MACs. By reducing the bit-width of MACs, we

increase the energy efficiency of the computation of non-outliers. Outliers are computed in multi-cycles by time-multiplexing. The contributions of this paper are as follows:

- We propose a novel time-multiplexing MAC which minimizes stall and maximizes energy efficiency.
- We propose a CNN accelerator exploiting the proposed MACs.
- We evaluate the energy efficiency of the proposed CNN accelerator on different CNNs trained for classification of ImageNet dataset.

2.2 Motivation

2.2.1 Sparsity of CNN's feature map

To reduce the power consumption of CNN accelerator, floating point precision are quantized into fixed-point precision. After the quantization of feature maps, the number of feature maps which have large magnitude is small. The large-magnitude data is defined as 'outlier'. In this paper, if a pixel of feature map cannot be expressed with 4-bits, it is called as an 'outlier'. In other words, outlier is greater than $2^3 - 1$ or smaller than -2^3. Figure 2.1 shows the ratio of outliers in ResNet. Although the number of outliers is usually small except for first layer, we cannot ignore outliers. Since a few outliers have a significant effect on the accuracy of image classification, the bit width of multipliers is determined by the magnitude of outliers. To solve the problem, we propose the outlier-aware time-multiplexing MAC which can compute non-outlier without waste of energy.

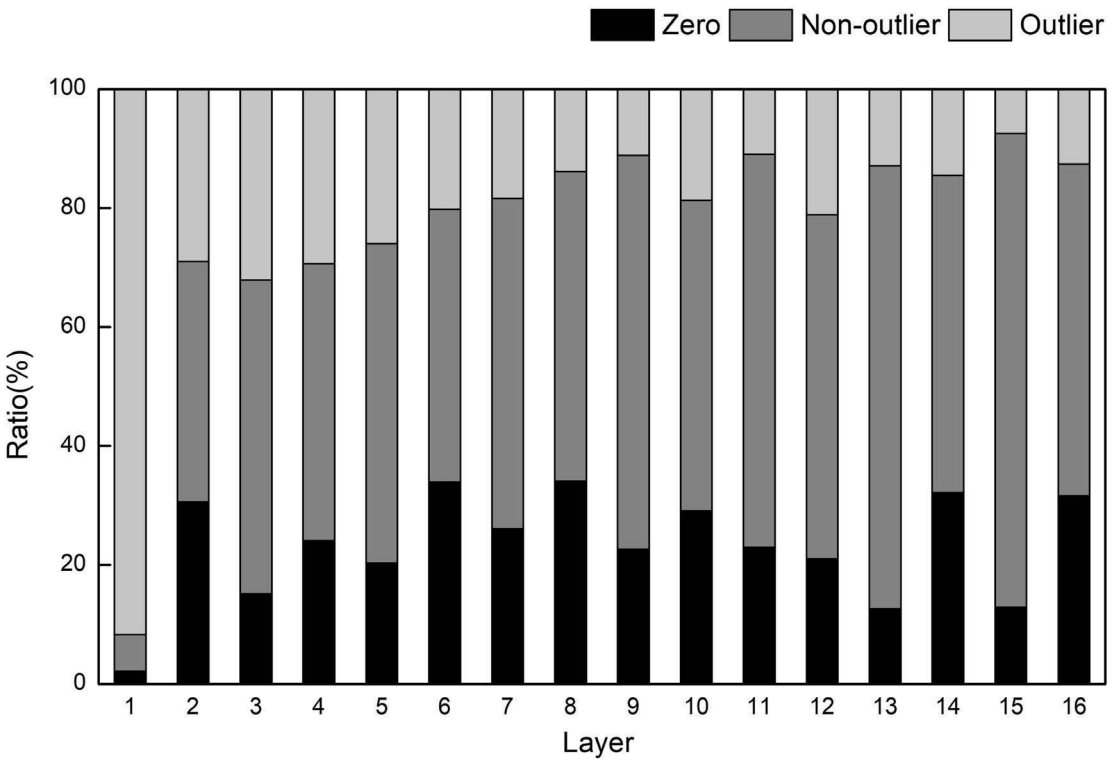


Figure 2.1: Ratio of outlier, non-outlier and zero feature maps

2.2.2 Time multiplexing multiplier

Figure 2.2 shows the architecture of the proposed CNN accelerator. The proposed CNN accelerator consists of encoder, decoder, activation module (ReLU), on-chip global buffer, indicator module and PE array. The encoder compresses the data before writing feature maps to off-chip memory and decoder restore the feature map from compressed data. Indicator module generates 1-bit zero-indicators and 1-bit outlier-indicators for feature map and weight, respectively. Partial sums, input feature maps and weights are transferred to PE by 8-bit bus. PE receives feature maps and weights with indicators from indicator module. Transferred feature maps and weights are fetched to 256×8-b weight buffer and 256×8-b input feature map buffer, respectively. PE has time-multiplexing (TMx) multipliers which can compute non-outlier in a cycle with reduced power consumption as shown in Figure 2.3. The operation of non-outlier is completed in a cycle with relatively small-bit width multiplier, while operation of outlier consumes two cycles.

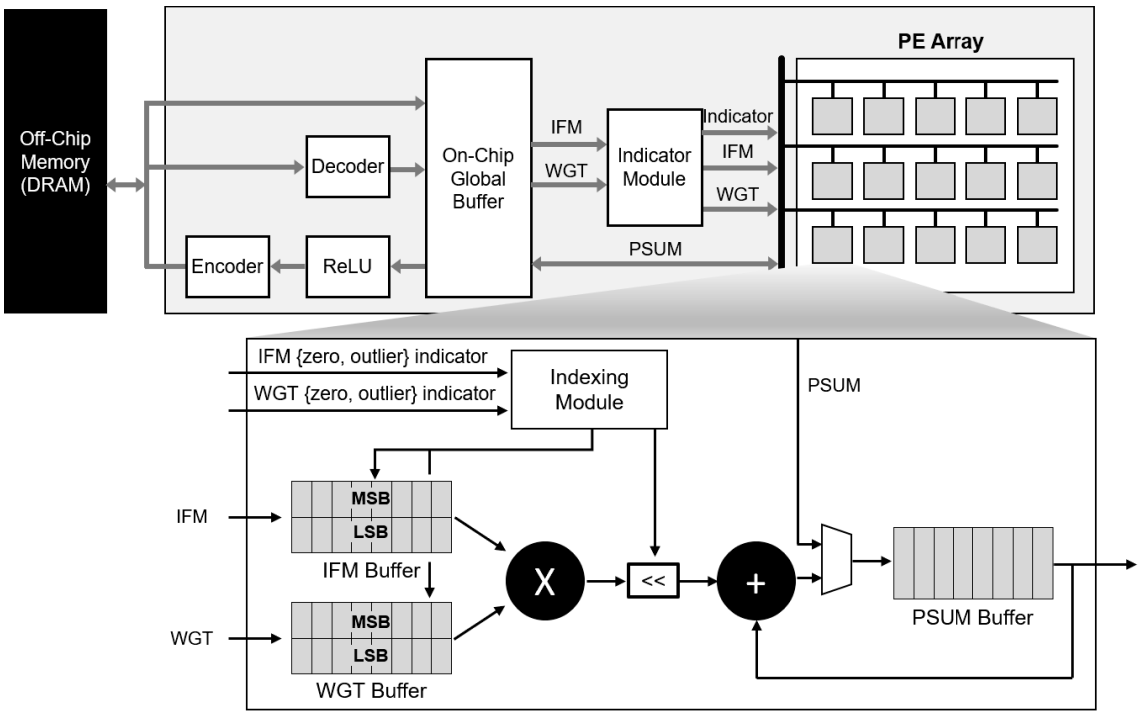


Figure 2.2: Architecture of the proposed time-multiplexing CNN accelerator.

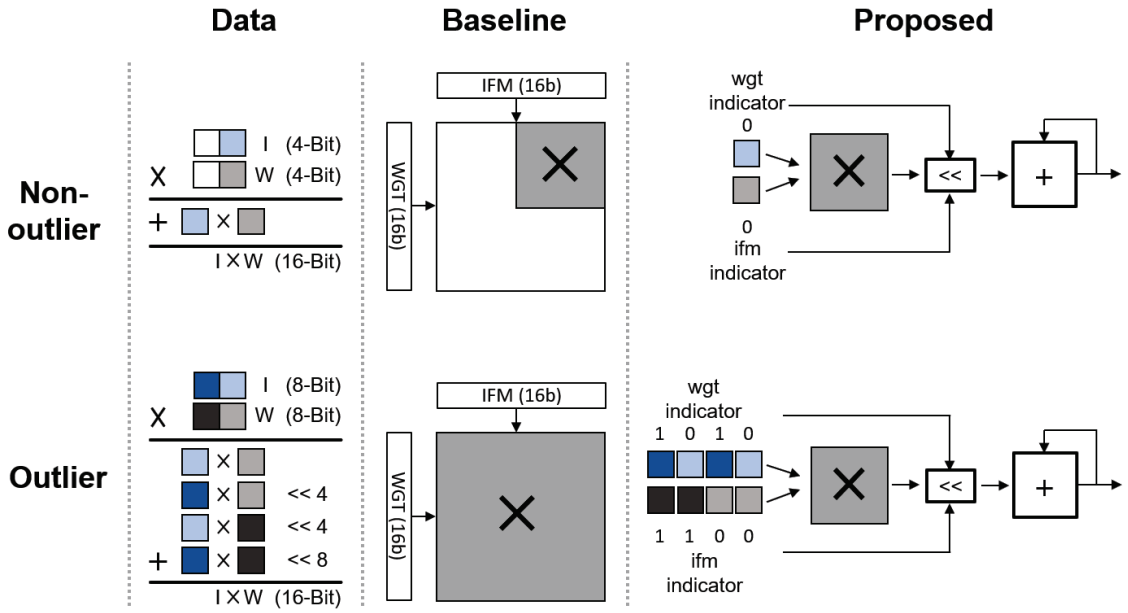


Figure 2.3: Proposed time-multiplexing multiplier.

2.3 Evaluation of Energy Consumption

PDP of the MAC module is used as an indicator for the energy efficiency of a newly proposed hardware architecture. PDP is defined as the multiplication of average power and the worst delay. Lowering PDP serves as the most important performance indicator for digital systems, as it means having benefits in both terms of power and delay. An 8-bit×8-bit MAC was established as a baseline MAC. RTL design of the proposed MAC and baseline MAC module is synthesized to measure energy efficiency. Compare to baseline MAC, the proposed MAC consumes 37.3% less computation energy in a cycle. Since computation cycles of the propose MAC are dependent on the ratio of outliers, the final energy consumption is estimated by multiplying the total cycle to obtain PDP as shown in Figure 2.4. In this experiment, we sweep the bit width of outliers from 4-bit to 7-bit. 7-bit TMx PE shows the lowest energy consumption for the inference. Using 7-bit TMx PE, we can save 9% of energy cosumption. The proposed is compatible with zero-skipping technique. TMxZ denotes a time-multiplexing multiplier with zero-skipping technique. Using 6-bit TMxZ PE, we can 34% energy consumption and

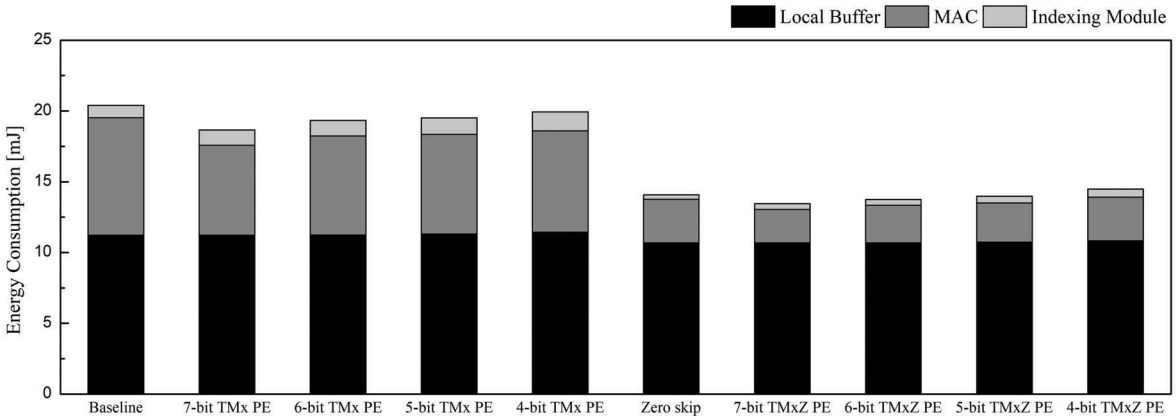


Figure 2.4: Energy consumption of the proposed PE.

2.4 Approximate Computing

As semiconductor technologies continue to develop, electronic devices are becoming smaller and more portable. Consequently, as the battery size of Internet of Things (IoT) devices decreases and power consumption increases, the urgent need for energy-efficient systems has generated research interests in approximate computing techniques. Approximate computing can be applied to vision, search, and image processing, which do not require a 100% of accurate results. In this paper, we apply approximate computing to a digital filter for image processing. The digital filter can be implemented through an infinite impulse response (IIR) filter and a finite impulse response (FIR) filter. The FIR filter shows better phase linearity and stability than the IIR filter. However, it consumes more power because of its complex design, and hence reduces the overall energy efficiency of the system. To improve the energy efficiency of the FIR filter, several proposals have sought to reduce their design complexity [18, 19, 20, 21, 22]. However, these approaches only focused on reducing the number of adder steps [18, 19, 20], providing an accuracy estimation model [21], or developing an approximate adder [22], separately.

Figure 2.5 shows the conventional multiply-and-accumulate (MAC) structure of the FIR filter. A popular idea for complexity reduction here is a multiplier-less FIR filter [18], where multiplication is implemented with shifters and adders rather than multipliers. Integer coefficients are transformed into a proper one for shift and addition operations. In conventional FIR filters, all coefficients are expressed in signed-power-of-two (SPT) space rather than signed binary, since SPT can reduce the number of nonzero digits. In the SPT codes, a canonical signed digit (CSD) code is well known to effectively reduce the complexity of FIR filters.

Another key idea in conventional FIR filters is a common subexpression elimination (CSE) algorithm.

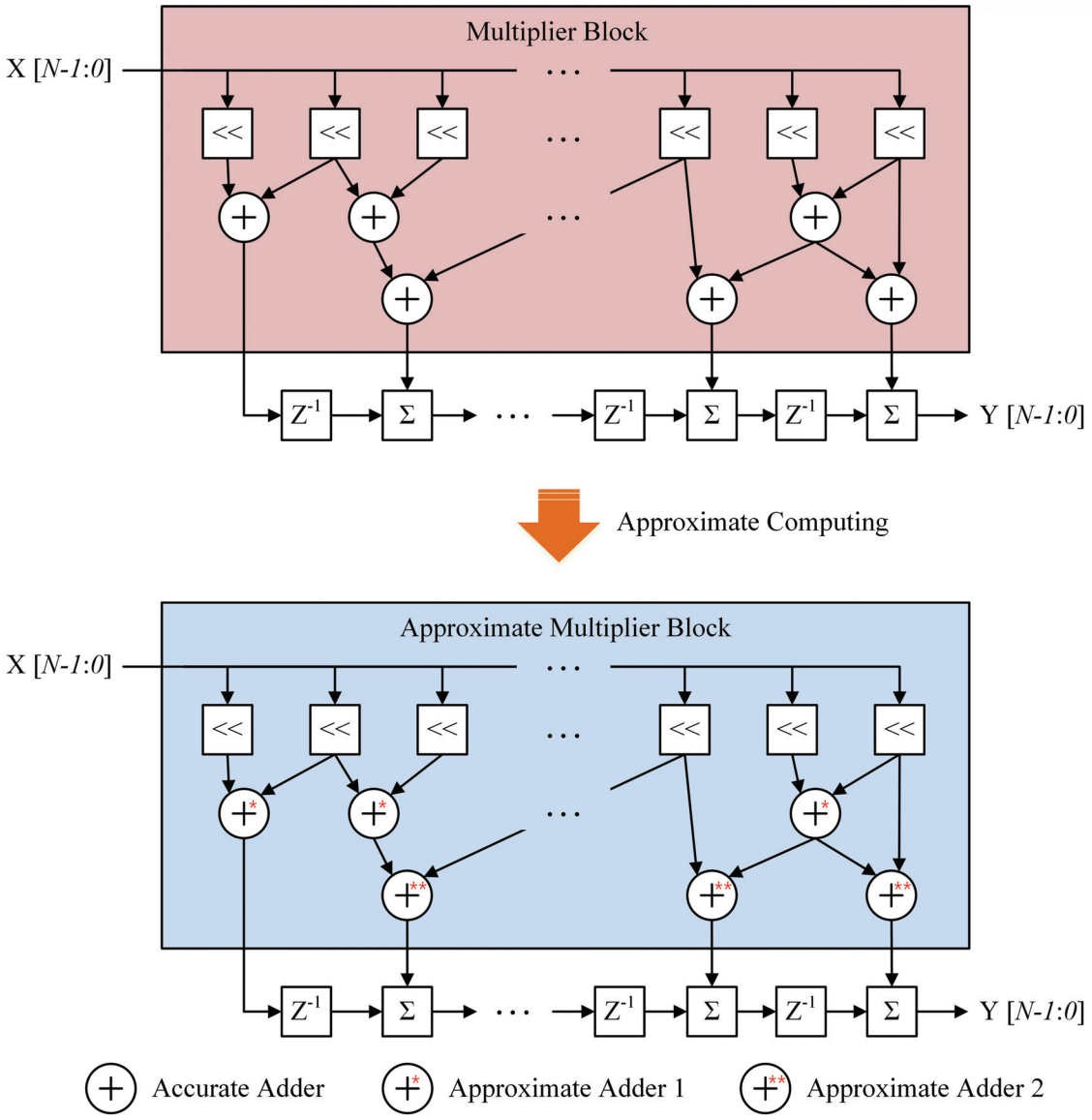


Figure 2.5: The structure of the conventional FIR filter and the proposed approximate FIR filter.

Chia et al. [19] proposed a CSE algorithm to reduce redundancy among CSD coefficients. Choi et al. [20] analyzed the criticality of each coefficient of an FIR filter and applied tighter constraints on more critical coefficients during the CSE algorithm. Choi's FIR filter yielded 25%-30% power saving at low voltages with minor passband/stopband ripples. Kahng et al. [21] implemented an FIR filter using an approximation at the synthesis level. They replaced certain modules with approximated ones based on lookup tables in order to reduce power consumption with only a small degradation in the quality of output. Gupta et al. [22] implemented an FIR filter using an approximated circuit. They proposed mathematical models for error and the power consumption of the approximate adders.

Chia et al. [19] and Malcolm et al. [18] only focused on reducing the number of adder steps. Choi

et al. [20] considered voltage scaling to save power, but the errors incurred along the critical path were observed to usually be more critical than those due to approximations. Kahng et al. [21] and Gupta et al. [22] applied approximate computing to an FIR filter but did not provide any automated synthesis flow for the approximation. If the size of the design of the FIR filter becomes larger, it becomes difficult to find optimum configurations for the approximate adders.

In this dissertation, we propose a novel approximate synthesis technique that reduces energy consumption by replacing conventional adders/subtractors in the FIR filter with approximated adders/subtractors with automated synthesis flow, as shown in Figure 2.5. The following are the main contributions of our paper:

- An *accuracy*-configurable adder/subtractor is proposed, which is energy efficient and has relatively high *accuracy*.
- The maximum error due to the configurations of the proposed adder/subtractor is analyzed to estimate output quality.
- A novel approximate synthesis flow for the FIR filter is proposed. Using the proposed approximate synthesis flow, we can save energy/power consumption and improve performance to yield a reasonable level of *accuracy*.

2.5 Related Works

2.5.1 Common subexpression elimination (CSE)

As discussed in Section 1, the CSE algorithm can reduce the design complexity of the FIR filter. In this section, we briefly introduce the CSE algorithm proposed in [19]. The following terms are used to explain it.

- Adder Step (*AS*): the number of adders that are used to implement the coefficients of the FIR filter.
- Filter Adder Step (*FAS*): the number of adders along the critical path of the FIR filter. *FAS* is always greater than or equal to $max(log_2 k)$ where k is the number of non-zero bits of the coefficients.

At the beginning of the CSE algorithm, all coefficients are converted into canonical signed-digit codes and their consecutive zeros are eliminated using a right-shift operation. Set C_N is constructed from the converted coefficients, and another set N_C is constructed by decomposing C_N. At the first iteration of the CSE algorithm, each value in C_N is checked to determine if it is decomposable by the other values in $C_N \cup \{1\}$. If the value is decomposable, it moves into a set C_P. Otherwise, the algorithm checks if the value is decomposable using values in $C_N \cup N_C \cup \{1\}$, and the decomposed value moves to C_P. The values in N_C, which are used in the decomposition, are moved to C_N. These procedures are repeated until C_N is empty. Following the CSE algorithm, the CSD values in C_P are used to synthesize the multiplier block in Figure 2.5 (b). For further explanation, we use an example. Let *FAS* = 4; the coefficients are

$h_0 = 105_{(10)} = 10\underline{1}01001_{(2)}$ $\quad h_1 = 831_{(10)} = 10\underline{1}01000001\underline{1}_{(2)}$

$h_2 = 621_{(10)} = 1010010\underline{1}01_{(2)}$ $\quad h_3 = 815_{(10)} = 10\underline{1}010\underline{1}0001_{(2)}$

For simplicity, the CSD coefficients are expressed in integer format. Prior to the first iteration,

$C_P = \phi$

$C_N = \{105, 831, 621, 815\}$

$N_C = \{3, 5, 7, 9, 13, 15, 17, 19, 23, 27, 31, 39, 47, 51, 63, 67, 97, 109, 113, 123, 125, 127, 129, 137, 155, 159, 193, 209, 257, 273, 493, 497, 509, 513, 625, 637, 641, 751, 767, 1007, 1023, 1071, 1087\}$

At the first iteration, 815 and 621 are decomposed by 831 and 105, respectively: $815 = 831 - 1 \times 2^4$, $621 = 831 - 105 \times 2^1$. At the next step, 105 and 831 are decomposed. The result of the decomposition is

$105 = 15 \times 2^3 - 15$ and $831 = 15 \times 2^6 - 129$, respectively. At the last step, 15 and 129 are decomposed: $15 = 1 \times 2^4 - 1$ and $129 = 1 \times 2^7 + 1$. Following the iteration,

$C_P = \{105, 831, 621, 815, 15, 129\}$

$C_N = \phi$

$N_C = \{3, 5, 7, 9, 13, 17, 19, 23, 27, 31, 39, 47, 51, 63, 67, 97, 109, 113, 123, 125, 127, 137, 155, 159, 193, 209, 257, 273, 493, 497, 509, 513, 625, 637, 641, 751, 767, 1007, 1023, 1071, 1087\}$

The iterations terminate when C_N is empty. The synthesized FIR filter from the CSE algorithm is shown in Figure 2.6.

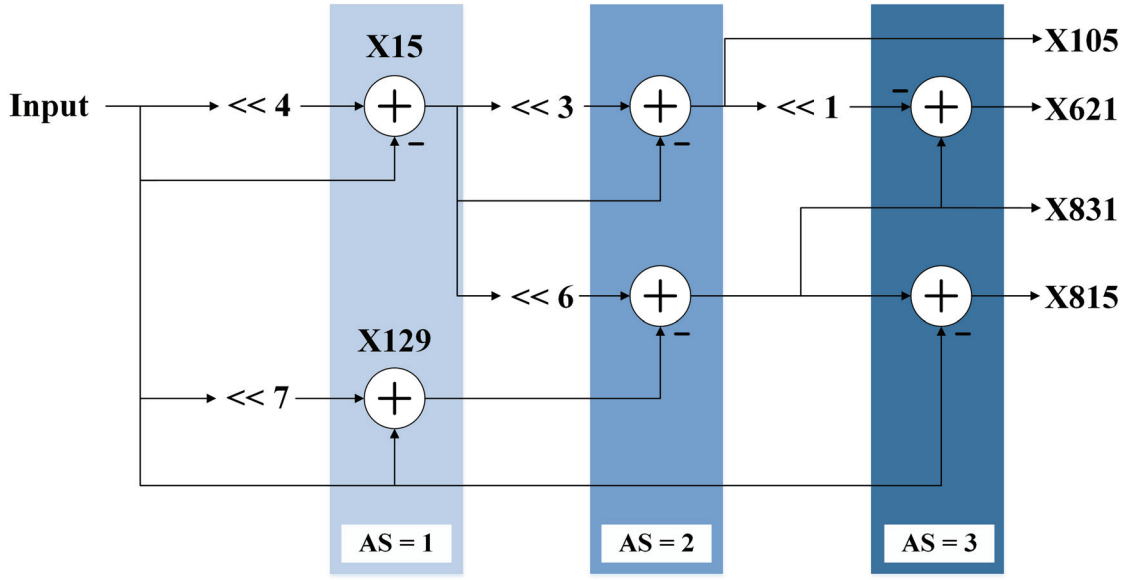


Figure 2.6: The schematic of the FIR filter. The coefficients of the FIR filter are (105, 831, 621, 815), and $FAS = 3$.

2.5.2 Circuit for approximate computing

Approximate computing generates sufficiently good results using low power rather than exact results. It can be used for noise-tolerant applications. Various approximate arithmetic designs have been proposed in past research. Lu et al. [24] introduced a fast adder with shorter carry chains that considers only the previous k bits of input in computing a carry bit. Verma et al. [25] proposed a variable-latency speculative adder (VLSA), which is a reliable version of the Lu adder [24] with error detection and correction. Shin et al. [26] also proposed a data path redesign technique for various adders that reduces the lengths of critical paths in the carry chain. Zhu et al. [23] proposed three approximate adders—ETAI,

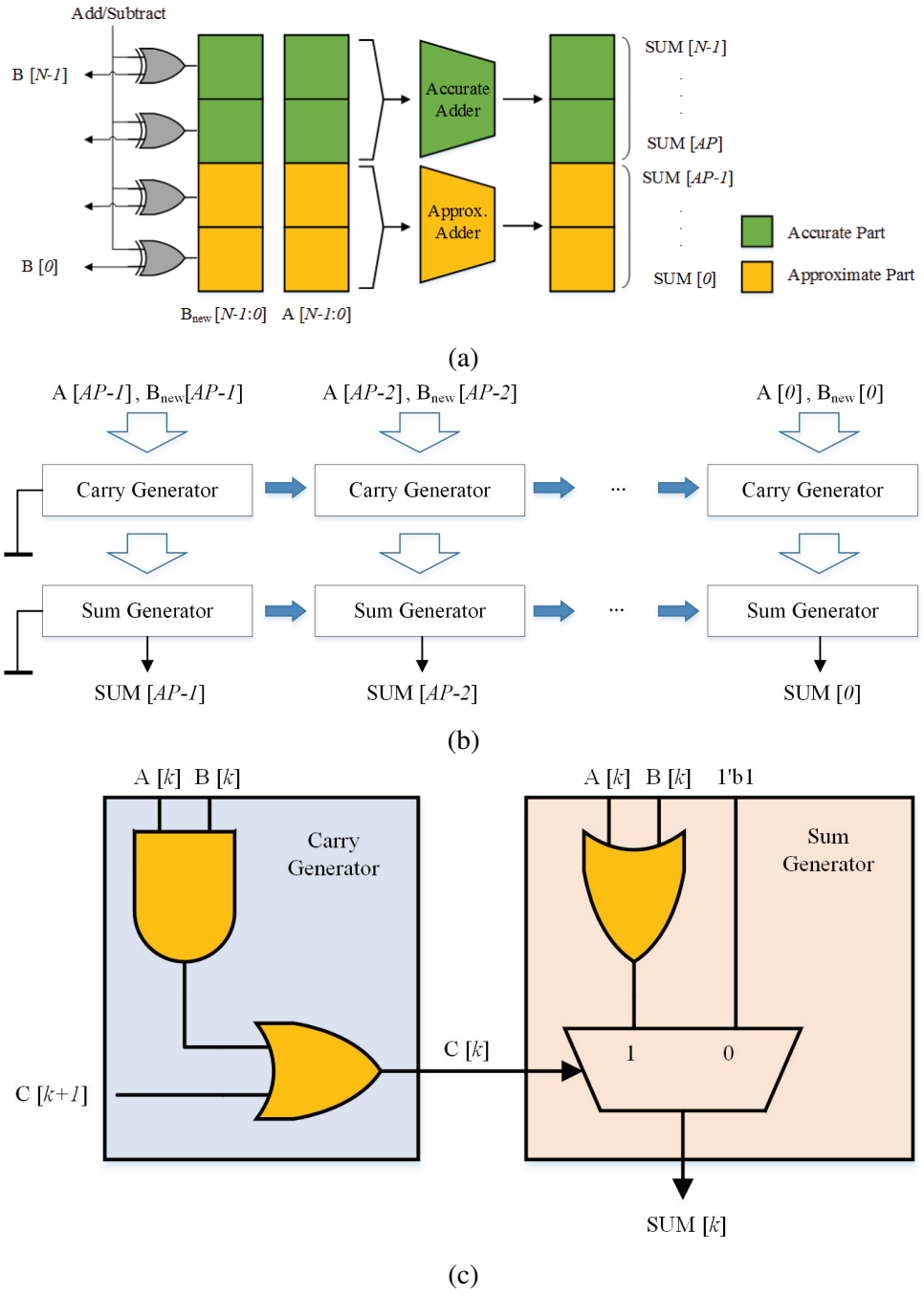


Figure 2.7: (a) Proposed approximate adder/subtractor. (b) Structure of the approximate part. (c) Schematic of the *k*-th carry generator and the sum generator in the approximate part.

ETAII, and ETAIIM. ETAI is divided into an accurate part and an inaccurate part to achieve approximate results. ETAII reduces carry propagation to speed up the adder, and ETAIIM modifies ETAII by connecting carry chains in accurate MSB parts. Gupta et al. [22] conducted approximations at the transistor level, and proposed approximate full adder cells to design multi-bit adders for video applica-

tions to save power and area. Kahng et al. [27] proposed an accuracy-configurable approximate (ACA) adder. In an approximate mode, it carries out approximations by cutting carry chains. In an accurate mode, it recovers accuracy by error detection and correction circuits. The ACA adder can save power consumption in the approximate mode and provide precise results in the accurate mode. Venkatesan et al. [28] proposed a systemic design methodology for approximation computing that eliminates certain nodes from the original set of nodes, and analyzes how the eliminated nodes affect accuracy and power consumption through approximation. Several studies have been devoted to approximate multipliers [30, 31, 32, 33, 29, 34]. For DSP applications, fixed-width approximate multipliers have been proposed in [30, 31, 32]. They eliminate (W-1) LSBs of (2W - 1) partial products obtained from a W$\times$W multiplication. Cho et al. [30] and Wand et al. [32] proposed carry approximation techniques in multiplication. Lu et al. [33] proposed a broken-booth multiplier, but this has a low probability of yielding the correct result rate. Kulkarni et al. [29] introduced an approximate multiplier based on 2×2 approximate multiplication with an error probability of 1/16. The simplified 2×2 approximate multiplier only has five unit cells, whereas the accurate one has eight unit cells. Not only does the simplification reduce the lengths of the critical paths of approximate multipliers, it also consumes less power and outperforms accurate multipliers.

2.6 Approximate Synthesis for FIR Filter

2.6.1 The proposed approximate adder/subtractor

For the approximation of the FIR filter, we propose an accuracy-configurable adder/subtractor. The basic principle of the proposed adder/subtractor is similar to that underlying Zhu's adder [23]. This adder detects carry generation conditions and generates '1' in all lower-sum bits without carry propagation to upper bits. To implement MAC circuits, both adders and subtractors are required. XOR gates are added in front of the adder to switch between it and the subtractor. For exact subtract operations, we should take 2's complement of the subtrahend by adding '1' to the 1's complement. The proposed approximate adder/subtractor, however, takes the 1's complement of the subtrahend as input because a carry in the approximate part is not propagated to the accurate part.

Our proposed adder is divided into two parts: an accurate part and an approximate part, as shown in Figure 2.7 (a). The bit width of the adder is N and that of the approximate part is AP. The operating principle of the accurate part is identical to that of conventional adders. The structure of the approximate part is shown in Figure 2.7 (b). It consists of AP-bit carry generators and AP-bit sum generators. As shown in Figure 2.7 (b), the carry in the approximate parts is propagated from the most significant bit (MSB) of the approximate part to the least significant bit (LSB). The direction of carry propagation is the reverse of that in conventional adders. Figure 2.7 (c) shows a schematic diagram of the carry generator and the sum generator. If the carry is generated from previous carry generators, it passes to the next one. Otherwise, two input operands are compared, and the carry is generated if both are '1.' The sum generator receives a carry from the carry generator. If a carry exists, the sum generator returns '1.' Otherwise, it adds two input operands and returns the sum value. The accuracy of the adder/subtractor is configurable by changing parameter AP, the bit width of the approximate part. AP can be configured from 0 to N. If AP is 0, the result of the proposed adder/subtractor is identical to that of the conventional adder/subtractor. If AP increases, the accuracy of the output is degraded, but power consumption is reduced or performance is improved. However, if AP is larger than a certain value, the propagation delay of the approximate part becomes that of the accurate part, and the benefits of further approximation are diminished. Hence, the AP value should be appropriately configured during approximate synthesis flow.

The maximum error in approximation occurs when all input bits in the approximate part are '1'. In this case, the two input operands are $2^{AP}-1$. The outputs from the conventional adders are $(2^{AP}-1)\times 2$, whereas the approximate adder returns $2^{AP}-1$. In the results, the maximum error that can occur in the approximate adder is $2^{AP}-1$. On the contrary, if the approximate part is truncated, the maximum error

Algorithm 1: Sensitivity-based approximate synthesis flow.

```
Classify adders according to AS
AP_i ← 0, where i = 1, ... ,FAS.
while SF_best > 0 do
    for i := 1 to FAS do
        AP_i ← AP_i + 1
        Synthesis {newAP_0, newAP_1, ..., newAP_FAS}
        Calculate delay_i
        Gate level simulation
        Calculate accuracy_i
        Power analysis
        Calculate power_i
        Calculate SF_i
        Recover design AP_i ← AP_i − 1
    end for
    SF_best = max(SF_1, SF_2, ..., SF_FAS)
    if SF_best > 0 then
        Select {AP_1, AP_2, ..., AP_FAS}_best
    end if
end while
Return {AP_1, AP_2, ..., AP_FAS}
```

is $(2^{AP}-1)\times 2$, which is twice that incurred by the proposed adder. For example, if N, AP, and the two inputs are '8', '4', $01101111_{(2)}$, and $00011111_{(2)}$, respectively, four MSBs are computed in the conventional part and four LSBs are added in the approximate part. The outputs from the accurate and approximate parts are $0111_{(2)}\times 2^4$ and $1111_{(2)}$, respectively, and the result is $01111111_{(2)}$, $127_{(10)}$. Since the golden result of this addition is $10001110_{(2)}$, $142_{(10)}$, the error is 15, which is equal to 2^4-1. From the results, the amount of error can be reduced by using approximate adders when it compares to the truncation of some input bits.

To verify the quality of the output obtained by approximate computing, we use the accuracy metric proposed in [23], defined as follows:

$$accuracy = \min_{k=1,\ldots,M}\left(1-\frac{|result_k - ref_k|}{|ref_k|}\right)\times 100\% \tag{2.1}$$

where M is the number of input patterns. The $result_k$ is an approximate result generated from the k-th input pattern, and ref_k is the correct result.

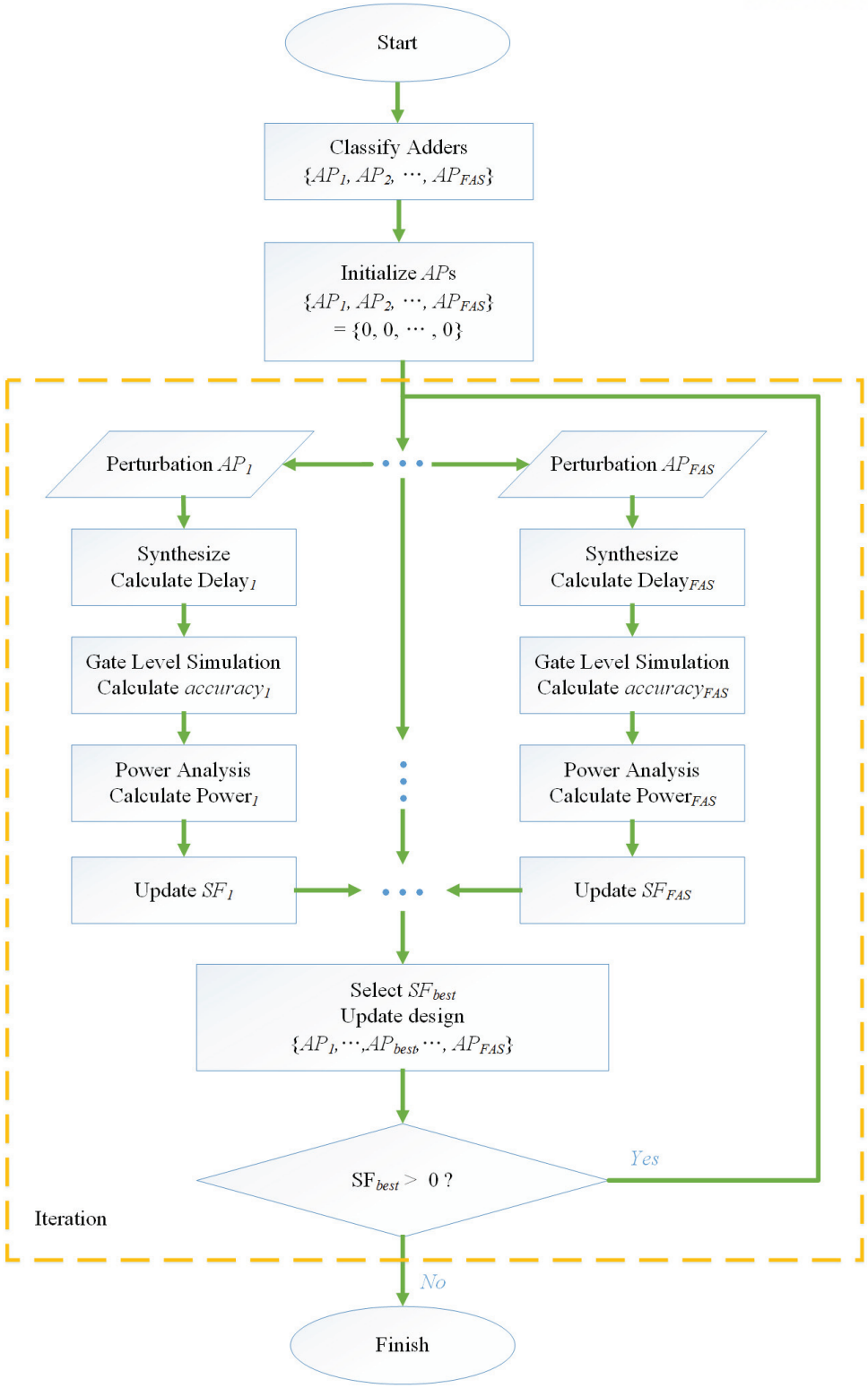


Figure 2.8: Proposed synthesis flow.

2.6.2 Approximate synthesis flow

In this subsection, we describe the proposed approximate synthesis flow. The purpose of the synthesis flow is to find the optimum *AP* configurations of approximate adders. Using these optimum configurations, we can save energy/power consumption and improve performance while maintaining a higher accuracy than a certain minimum constraint, $accuracy_{min}$. However, finding the optimally configured *AP*s of the adders is difficult because the number of possible combinations of configurations is proportional to ${M_{adder}}^{N}$, where M_{adder} is the number of adders and N is the bit width of the adders. For further explanation, we use the example in Figure 2.6. The bit width of the input, the coefficients, and the output in the example are 15, 12, and 28 bits, respectively. The coefficients are (105, 831, 621, 815), synthesized from the CSE algorithm introduced in Section 2.1, with $FAS = 3$. Assuming that the M_{adder} is 6 and N is 28 bits, the number of possible combinations of the *AP*s is approximately 6.14×10^{21}. Since the size of the design of the example is small and the number of adders is conventionally greater than six, the possible combinations of *AP* configurations in conventional FIR filters are considerably more in number than in this example. Searching all combinations is time and resource consuming, and is impossible in cases of larger designs.

To handle this problem, we make two assumptions. First, the delays in the adders are comparable to those in the subtractors. Second, the actual arrival time of an adder/subtractor is comparable to that of another adder/subtractor with the same *AS*. Hence, we can conclude that changing *AP*s in only one path is less effective than simultaneously changing the *AP*s of adders. The number of possible combinations is then proportional to FAS^{M}. Considering that the *FAS* of the FIR filter is much smaller than that of M_{adder}, we can significantly reduce design space. Assuming *FAS* is 3 and N is 28 bits, the number of possible combinations of *AP*s is 2.28×10^{13}. During approximate synthesis flow, *AP* is usually less than the half N, where the practical design space is approximately $FAS^{M/2}$ (4.7 million in this case), which is a more reasonable value than the number of all possible combinations, 6.14×10^{21}.

Algorithm 1 describes the procedure of our proposed approximate synthesis flow. The flow finds an approximate design with the minimum delay and the required accuracy (i.e., higher than $accuracy_{min}$). In the first step, the baseline design is loaded and all adders are classified according to their *AS* (Line 2). All *AP*s of the *AS* are then set to 0 (Line 2). Following this, the *AP* in each *AS* is perturbed by adding 1 (Line 5). The perturbed Verilog design is synthesized, and the delay in the design is calculated (Lines 6-7). Using the synthesized design, a gate-level simulation and static timing analysis are performed to calculate the power and accuracy (Lines 8-11). From the slack and accuracy, the sensitivity factor (SF)

is calculated (Lines 12). The SF is defined as

$$SF = \begin{cases} \frac{accuracy - accuracy_{min}}{delay} & \text{, if } accuracy > accuracy_{min} \\ 0 & \text{, else} \end{cases} \tag{2.2}$$

where *accuracy* is defined in Equation 2.1. The calculated SF_i is added to the SF list. Following calculations, the perturbed design is reverted to the original one (Line 13). If all perturbations and SF calculations are complete from the SF list, the design with the highest SF is selected (Line 15). The selected design is used as a seed for the next iteration (Line 16). If the highest SF is zero or negative, the flow returns a final solution, and ends. The proposed synthesis flow is summarized in Figure 2.8. Low-power or highly energy-efficient design, which are our main concerns here, can be achieved by re-synthesizing the final solution of the synthesis flow with an appropriate clock constraint, i.e., the minimum available clock of the baseline design.

2.7 Experimental Setup and Results

2.7.1 Experimental setup

The proposed synthesis flow is written in Tcl and executed on a 2.6 GHz Intel Xeon E7-4860 Linux workstation. The FIR filter is implemented using the worst corner library of the TSMC 65-nm technology node and an RTL compiler [77]. A tight timing constraint[2] is used to synthesize the approximate design with minimum delay. Following the synthesis, the minimum delay in the FIR filters is calculated by the summation of the worst negative slack and the clock period.

For *accuracy* simulations, Cadence NC Verilog is used [78]. We generate 10,000 random patterns for RTL simulations and compare the output patterns with the correct ones. The *accuracy* value is calculated according to Equation (2.1). We set $accuracy_{min}$ to 95%.

Power consumption is reported using Synopsys PrimeTime-PX [79]. We calculate total power consumption, which includes static and dynamic power. The value change dump file generated from the previous gate-level simulation is used to calculate the switching activity of each net and the minimum clock period for each design is used to report the dynamic power.

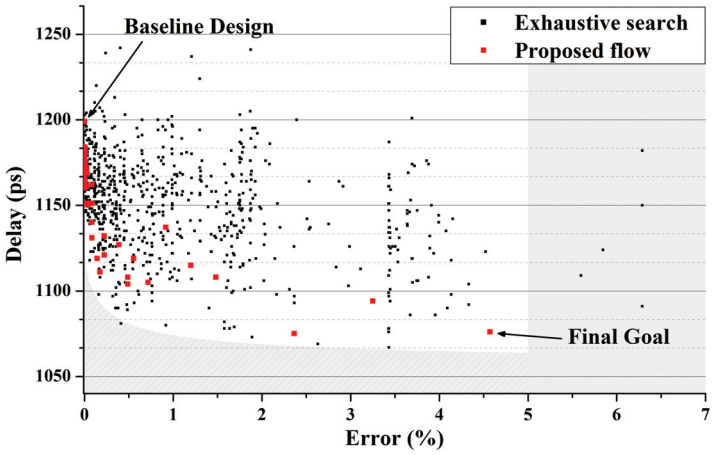
Table 2.1: Approximation results in 4-tap FIR filter with *FAS* = 3.

	Delay [ps]	Power [uW]	Energy [fJ]
Baseline	1199	2796	3352
Flow result	1076	1687	1815
Min. Energy design	1198	1379	1652
	Improvement [%]		
	Delay	Power	Energy
Flow result	10.3	39.7	44.7
Min. Energy design	0.0	50.7	50.7

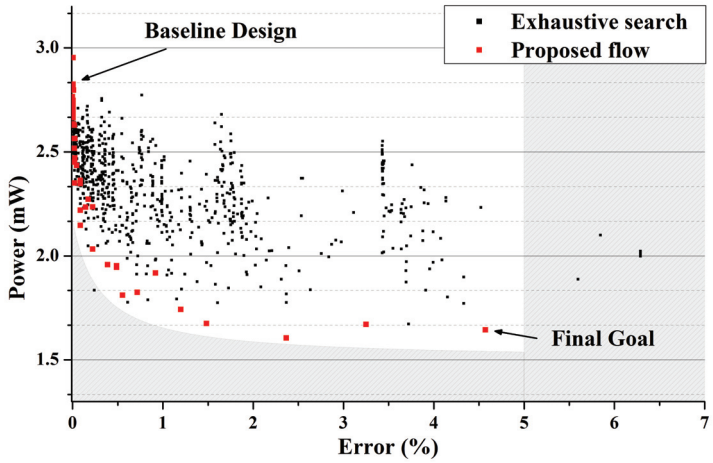
2.7.2 FIR filter implementation

We implement an FIR filter using our proposed approximate synthesis flow. We synthesize a four-tap FIR filter with the coefficient set {105, 831, 621, 815}. Figure 2.6 shows the structure of the implemented FIR filter. In this experiment, the bit width of the coefficients is set to 12. Since the largest coefficient is 831 in the four-tap FIR filter, 12 bits are sufficient to represent four coefficients in SPT. The

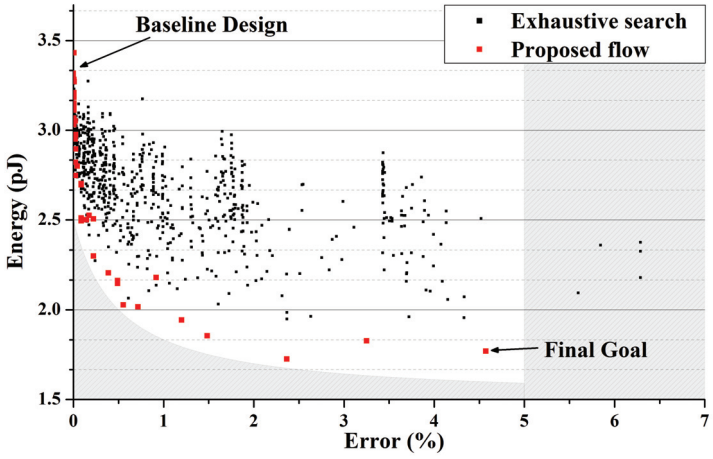
[2]In this paper, 500 *ps* is used for the timing constraint.



(a)



(b)



(c)

Figure 2.9: The proposed synthesis flow (red) and the exhaustive (black) are visualized in terms of (a) *accuracy* vs. the delay domain, (b) *accuracy* vs. the power domain, and (c) *accuracy* vs. the energy domain.

Table 2.2: Approximation results in 25-tap filter with *FAS* = 4.

	Delay [ps]	Power [uW]	Energy [fJ]
Baseline	1988	10.7	21.3
Flow result	1876	8.9	16.7
Min. Energy design	1983	8.2	16.3
	Improvement [%]		
	Delay	Power	Energy
Flow result	5.6	16.8	21.6
Min. Energy design	0.0	23.3	23.5

bit widths of the input and output are set to 15 bits and 28 bits, respectively. For addition, 28-bit adders are used. The four given coefficients are implemented using six adders according to the previously introduced CSE algorithm. The *AS* of each coefficient is different. The *AS*s of $\{15, 129\}$ are 1, those of $\{105, 831\}$ are 2, and the *AS*s of $\{621, 815\}$ are 3. In the first iteration of the synthesis flow, the accuracy configurations of the adders with the same *AS* are perturbed one by one. The perturbed designs ($\{1,0,0\}$, $\{0,1,0\}$, and $\{0,0,1\}$) are synthesized and simulated. $\{0,0,1\}$, which have the highest SF, is selected and set as seed of the following iteration. After several iterations, the final output is $\{11,16,14\}$. Figure 2.9 (a), (b), and (c) show an implemented design space using the proposed synthesis flow. The black dots are generated by randomly but separately configuring the *AP* of all adders. The red dots represent the results from iterations of the approximate synthesis flow. The white space shows the reachable design space with lower *accuracy* than $accuracy_{min}$ by configuring the *AP*s of each adder. As shown in Figure 2.9 (a), the proposed synthesis flow can successfully follow the minimum delay design. Moreover, it can be shown that the proposed synthesis flow can effectively reduce power and energy consumption.

Table 2.3: Specifications of the FIR filters.

FIR Filter	Tap	FAS	Delay [ns]	Power [mW]	Energy [pJ]
[35]	15	3	0.98	5.68	5.5
[36]	15	4	1.27	4.06	5.1
[37]	28	4	1.15	11.6	13.4
[38]	34	3	1.17	13.4	15.7
[39]	49	3	1.20	18.4	22.1

Since the main concern of our work is obtaining high energy efficiency, we re-synthesize the design

Table 2.4: Following the proposed synthesis flow

FIR Filter	Accuracy [%]	Delay [ns]	Power [mW]	Energy [pJ]	Energy Reduction [%]
[35]	97.83	0.93	4.34	4.06	26.9
[36]	95.32	1.14	3.17	3.62	29.5
[37]	96.03	1.15	8.13	9.34	30.1
[38]	95.66	1.15	8.34	9.59	38.9
[39]	95.19	1.12	13.60	15.27	30.8

acquired from the synthesis flow and implement it using different timing constraints. We then select the result with the lowest energy consumption with a delay not exceeding that of the baseline design. In Figure 2.9 (a), due to EDA tool noise, one design with close to 97.5% accuracy shows slightly lower delay and power consumption than the final solution design. Following re-synthesis, however, the energy consumption of the point is greater than that of the final solution.

Table 2.1 summarizes the results of the approximate synthesis flow. Performance improves by 10.3%, and power consumption is reduced by 39.7% over conventional FIR filter design. The energy is calculated by multiplying delay and power. Energy consumption per operation is reduced by 44.7%. To achieve further energy reduction, we change the timing constraint and find the minimum energy design for which delay is shorter than the baseline design. In this way, we achieve up to 50.7% reduction in energy consumption. The runtime of the proposed synthesis flow is 84 minutes for the four-tap FIR filter.

We apply the approximate synthesis flow to a 25-tap FIR filter, the coefficients of which are {-2423, -113, 1564, 762, -1816, -1517, 2276, 3140, -2434, -6205, 2726, 20680, 30093, 20680, 2726, -6205, -2434, 3140, 2276, -1517, -1816, 762, 1564, -113, -2423}. The results are shown in Table 2.2. In the 25-tap case, we can improve the performance by 5.6% with power and energy savings of up to 23.3% and 23.5%, respectively. The runtime of the proposed synthesis flow is 407 minutes for the 25-tap FIR filter.

2.7.3 Image FIR filter experiment

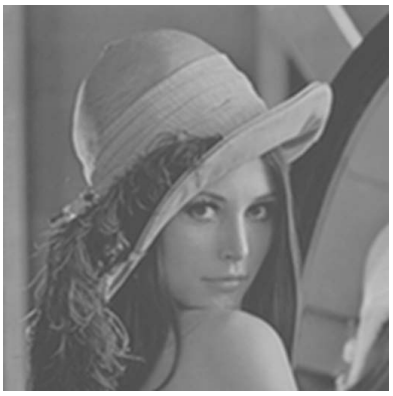
To verify our methodology, we apply the proposed synthesis flow to five different FIR filters [35, 36, 37, 38, 39]. The specifications of the FIR filters are summarized in Table 2.3. The delay, power, and energy information of the baseline designs of the FIR filters are also summarized in Table 2.3.

Original

(a)

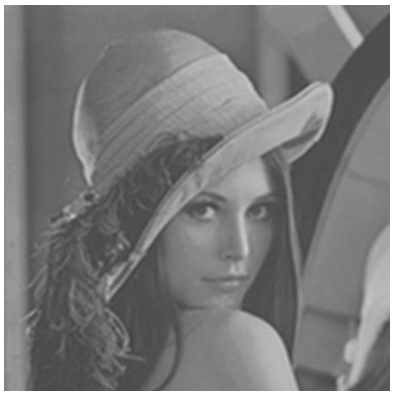
Baseline design

(b)



Proposed
PSNR = 46.3 dB
30.0 % Energy reduced

(c)



89.2 % Accuracy
PSNR = 40.6 dB
36.9 % Energy reduced

(d)

65.4 % Accuracy
PSNR = 14.7 dB
45.7 % Energy reduced

(e)

45.6 % Accuracy
PSNR = -9.5 dB
48.8 % Energy reduced

(f)

Figure 2.10: Filtered images using optimized FIR filters.

The FIR filter is synthesized using the proposed synthesis flow, whereas the bit width of the inputs, the coefficients, and the output width are set to eight, 16, and 24 bits, respectively. The results of the synthesis flow are shown in Table 2.4. The accuracies of the filters are higher than the threshold of 95%. The energy consumptions of the FIR filters are reduced by up to 38.9% and 31.2% on average.

An FIR low-pass filter is implemented in [37] for blurred images. Since the image used is two-dimensional, we apply the FIR filter first in the vertical direction, and divide the output by filter gain. Following this, the FIR filter is applied in the horizontal direction, and the output is divided by filter gain once again. Figure 6 (a) shows the original image and Figure 6 (b) shows the blurred image processed by the baseline FIR filter. Figure 6 (c) shows the image processed by the proposed FIR filter. To verify the output quality of the processed image, peak signal-to-noise-ratio (PSNR) is used. PSNR is defined as

$$PSNR = 10 \times log(\frac{255^2}{\sigma^2_{noise}}) \tag{2.3}$$

where σ^2_{noise} is the variance of the difference between Figure 6 (b) and others. FIR filters with varying accuracies are simulated. As accuracy decreases, the image becomes dark. This is because the proposed adder approximates the previous carry and the approximation error renders the result lower in value than the exact result. If the approximation error continues to increase, the results assume negative values, which are expressed as white dots. We find that we are able to achieve 30.8% energy saving with 46.3 dB PSNR.

2.8 Conclusion

In this chapter, we propose the energy-efficient PE. We have investigate the ratio of outlier of feature maps and weight in three different CNN. To reduce the energy consumption for computing non-outlier, we reduce the bit width of MACs. With the proposed TMxPE, we can save 39 % of energy consumption without any CNN accuracy loss. We also apply approximate computing to a FIR filter to enhance efficient energy consumption. The FIR filter has a MAC structure, and multipliers are replaced by shifters and adders/subtractors that are approximated. For the approximation, we propose an approximate adder/subtractor in order that the accuracy of the approximate adder/subtractor is configurable and switching between the adder and the subtractor is possible. The error in the proposed approximate adder is analyzed. Moreover, we propose a novel approximate synthesis flow that can find the optimal configurations of approximate adders. Using the proposed synthesis flow, we achieve up to 10.3% in terms of performance improvement and 50.7% in terms of power and energy saving over conventional FIR filter design. Our future research in the area will seek to reduce the runtime of the synthesis flow by developing an accuracy and power estimation model. Moreover, we intend to modify the synthesis flow to apply it to general computation blocks.

Chapter III

System Level Approach

Convolution neural networks (CNNs) are widely used in many AI applications, especially in image recognition, detection and segmentation [40, 41, 42]. The accuracy of CNNs has been improved rapidly, but this improvement has entailed increases in network size, number of computations, and memory usage [43]. Despite many attempts to reduce network size, state-of-the-art CNNs require tens to hundreds of megabytes of memory. The on-chip memory capacity of hardware accelerators is increasing, is still far too small for this purpose. Furthermore, many hardware accelerators use multiple buffering to reduce latency between memory and computing cores, and this process decreases that available on-chip memory in a cycle.

Before implementation on a hardware platform, CNNs are quantized for the purpose of reducing their size [48]. The precision of quantized CNNs is reduced from 32-bit single-float to 8-bit or 16-bit fixed-point precision. Even after this quantization, CNNs remain large. Therefore, loop tiling is performed to allow implementation of a CNN on a hardware accelerator.

Loop tiling divides a convolution layer into multiple blocks, which can be accommodated in on-chip memory. Loop tiling allows relatively large CNNs to be implemented on hardware accelerators, but it generates repetitive data movements between on-chip and off-chip memories. The data movements are so huge that they have become a bottleneck to improvements in throughput and energy efficiency. However, well-designed dataflow can maximize data reuse and reduce data movements between on-chip and off-chip memory. Therefore, much research has been focused on finding optimal dataflow adjusting loop tile size and loop order.

The procedures by which a hardware accelerator generates an output feature map depends on whether the channel loop is not tiled (Figure 3.1a) and or tiled (Figure 3.1b). If the channel loop is not tiled, the accelerator loads input feature maps and filters from external memory for convolution. To prevent

overflow of the accumulation, the bit width of accumulators is set to higher than the bit width of feature maps. After the convolution, which consists of several multiplications and accumulations, the results are rounded to the bit width. This rounding of output feature maps leads to a *quantization error*, which is analyzed and controlled during the quantization procedure of CNNs. If the channel loop is tiled (Figure 3.1b), tiles of input feature maps and filters are loaded, and then computed partial sums are stored in external memory. They are reloaded several times until the complete output feature maps are generated. During the process of reading and writing the partial sum, it is rounded and this process generates an additional error, named *channel loop tiling-error*. Because *channel loop tiling-error* deteriorates CNN accuracy, it should be understood and controlled.

Output stationary dataflow can prevent the partial sums from being transferred to external memory, but saving partial sums without rounding requires more than twice the on-chip memory. The increase of bit width of partial sums comes with increase of data traffic between on-chip memory and processing elements. In addition, the output stationary is not the optimal data flow for several convolutional layers of CNNs [44]. Therefore, breaks of channel loops are almost inevitable; they degrade the accuracy, and make another impediment to implementation of CNNs on hardware accelerators. To the best of our knowledge, this is the first work that analyzes the effects of channel loop tiling on accuracy of CNNs.

The main contributions of our work are:

- We explain the mechanism by which channel loop tiling causes errors, and analyze the effect of error on accuracy of CNNs.
- We propose a method which reduces *channel loop tiling-error* by using extended partial sums while minimizing circuit and memory overhead by compressing the absolute values of most significant bits (MSBs).
- We improve the accuracy under channel tiling, and quantify the circuit overhead.

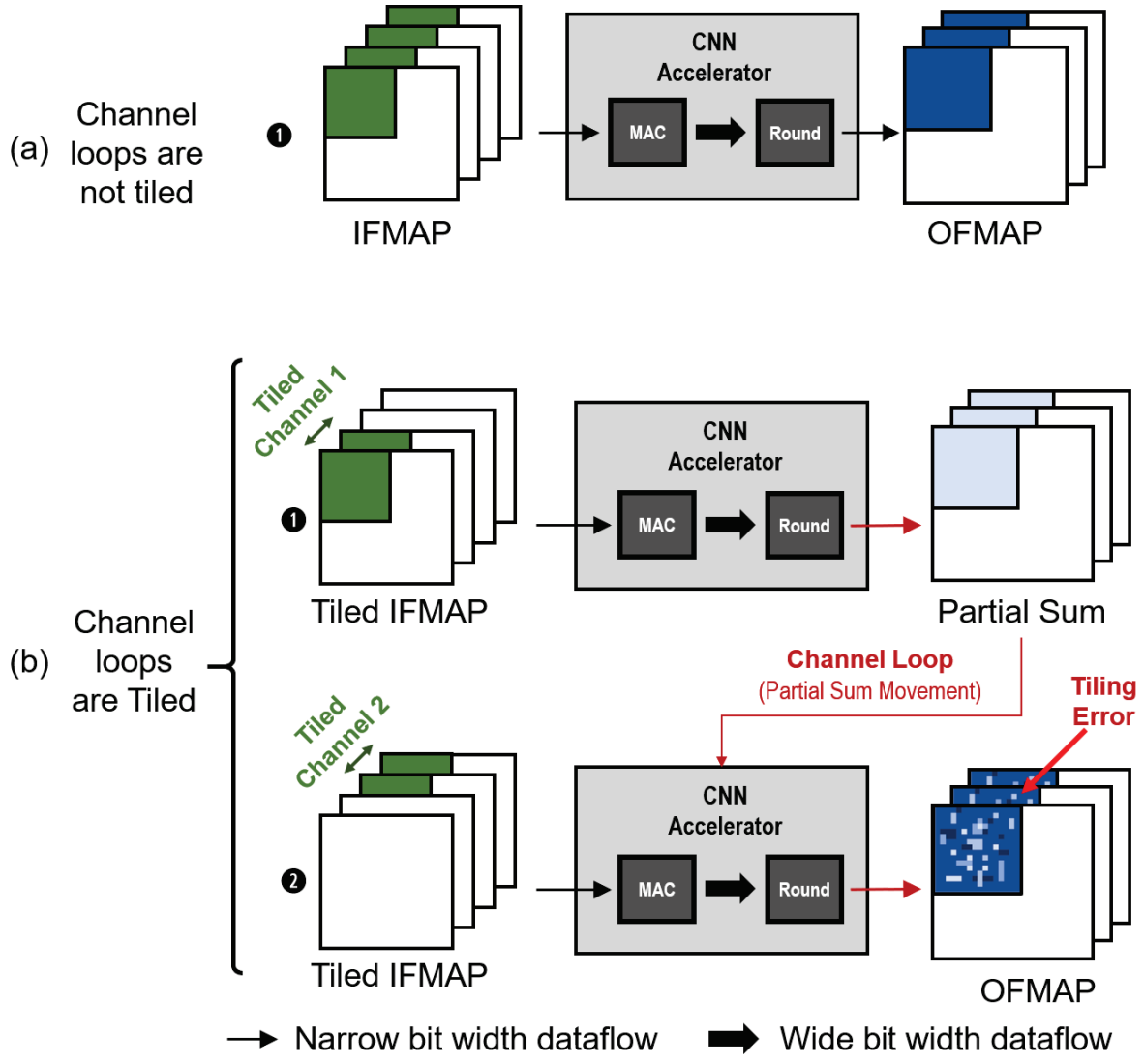


Figure 3.1: CNN implementations of a CNN accelerator. (a) Channel loop is not tiled. (b) Channel loop is tiled by two; additional errors, *channel loop tiling-errors*, occur.

3.1 Related Works

Data movement and precision of hardware accelerators significantly affect the accuracy and energy efficiency of CNN computations. Row-stationary dataflow [44] have been proposed and implemented on their hardware platform. The row-stationary data flow have increased energy efficiency by 2.5 times. Some dataflows consider FPGA hardware accelerators [45, 46, 47]; the authors have developed a roofline model that have considered CNN memory bandwidth and computational resources, and optimized loop tile size and loop order. However, they have not analyzed the interference between loop tiling and CNN quantization; this interaction causes additional degradation of CNN quality. In contrast to these studies, we analyze the reason for this additional accuracy loss, and propose a solution.

Reduction of bit width of CNNs can increase energy efficiency, so several quantization methods have been proposed [48, 49, 50]; the reduction of bit width is accompanied by reduction in CNN accuracy, so the authors focused on minimizing it. However, they have focused reduction of bit width without consideration of hardware.

A proposed hardware accelerator [51] coupling relatively low-energy-consuming processing elements (PEs) with a relatively high accurate outlier PE. Each PE has normal multiply-and-accumulators (MACs) and an outlier MAC. Most of the feature maps and filters having small values, are computed in normal MACs with low energy consumption. The outlier filters are computed in outlier MACs, and the outlier feature maps are computed in outlier PEs. This approach can reduce energy consumption. However, the process of indexing the outlier imposes design complexity and circuit overhead, so the proposed accelerator shows a large energy consumption in logic. Furthermore, the approach needs dedicated CNN architecture. A fixed-point representation with error compensation has been proposed [52] to reduce the length of computation bits. The authors also proposed a sparse compensation scheme to minimize energy overhead that is required to compensate for quantization errors. However, compensation frequency must be carefully managed by considering the tradeoff between accuracy and energy consumption. Also, adjusting bit width to access memory is not easy. Complementary to these studies, our proposed method accesses the memory with a fixed bit width and can be applied in any CNN architecture.

3.2 Loop Tiling on CNN

3.2.1 Role of loop tiling

In this subsection, we introduce procedures of implementing CNNs on a hardware accelerator. Generally, trained CNNs are too large to be implemented in an accelerator, so the networks are quantized and tiled.

CNN quantization entails two procedures: (1) determining the bit width of feature maps and filters; and (2) choosing the precision. Because each hardware accelerator has a different bit width for CNN convolution, the bit width should be determined by considering the target accelerator. For example, if the accelerator supports 8-bit multiplication, the bit width of input feature maps and filters, BW_I and BW_F, should be quantized to 8-bit. Because the output feature maps are the input feature maps of the following layer, the bit width of output feature maps, BW_O, is usually equal to BW_I.

Then we should choose the range and precision of each CNN and each layer. A fixed-point representation consists of *sign*, *integer part* and *fractional part*. *sign* is located on the MSB. The *integer part* determines the range of numbers that the fixed-point number can represent. The *fractional part* determines the precision of the fixed-point representation. The range is more important than the precision in most cases, so we determine the bit width of the *integer part* first, and use the remaining bit as the *fractional part*. The range and precision are determined by investigating the feature maps and filters during pre-inferences of each CNN.

If the required memory size of a CNN layer is still larger than on-chip memory size after quantization, we must appropriately split each layer of CNNs by considering both the size of the quantized CNNs and the memory capacity of the accelerators. The convolution layer consists of six for-loops. Each layer takes C input feature maps of width W and height H. Input feature maps are convolved with M filters which have a size of $C \times K \times K$, where K is the height and the width of a filter. As results of convolution, M output feature maps are generated. Each design parameter of six loops can be tiled independently, so the design space for loop tiling has six dimensions. Considering that the filter size of state-of-the-art CNNs is small (usually three or five), design space can be reduced to four dimensions, T_m, T_c, T_w and T_h as shown in Figure 3.2. The four parameters are adjusted appropriately by considering target CNNs and varying memory size.

3.2.2 Necessity of channel loop tiling

Before discussing effect of *channel loop tiling-error*, we address whether *channel loop tiling-error* is inevitable. Our basic assumption is that the size of a CNN layer is bigger than on-chip memory size so

```
for(w=0; w<W; w+=Tw)
  for(h=0; h<H; h+=Th)
    for(c=0; c<C; c+=Tc)
      for(m=0; m<M; m+=Tm)
```

Computation in Host CPU

```
        /* load partial sum */
        /* load input feature maps. */
        /* load filters. */
        for(tm=m; tm<min(m+Tm,M); tm++)
          for(tc=c; tc<min(c+Tc,C); tc++)
            for(tw=w; tw<min(w+Tw,W); tw++)
              for(th=h; th< min(h+Th,H); th++)
                O[tm][tw][th] +=
                ∑_{i=0}^{K-1} ∑_{j=0}^{K-1} W[tm][tc][i][j]*
                        I[tc][tw+S+i][th+S+j]

        /* Round Partial Sum */
        /* Save Partial Sum */
```

Computation in Core

Figure 3.2: Pseudo code of convolution layer.

that loop tiling is mandatory. The CNN loops are tiled by considering both computational speed of the CNN accelerator and bandwidth between off-chip and on-chip memory. In real implementation of large CNN layers, *channel loop tiling-error* is avoidable in following three cases.

1) Exclusion of channel loop tiling.

In this case, other CNN loops (M, W, H) are tiled except for channel loops. When the number of input channel is small as in first few CNN layers, the computational speed of CNN hardware will no be degraded. On the other hand, when the channel is deep as in rear CNN layers, the computational speed will be greatly reduced without the channel loop tiling. For further explanation, we provide a hardware example with following conditions.

- Layer parameter: $W = 16$, $H = 16$, $C = 1024$, $M = 256$ (38-th layer in ResNet50)
- Variable : T_w, T_h, T_c, T_m (number of tile of each loops)
- Constraints : on-chip memory size $\leq 200\ kB$ (usually scratch pad, local memory or global buffer)
- Goal: minimize the number of cycles.

- With a channel loop tiling, we can compute this layer in 18 cycles ($T_w = 16,\ T_h = 16,\ T_c = 114,\ T_m = 128$).

In this tiling example, channel loop is split into nine pieces and the partial sums are extracted by eight times. Excluding channel loop tiling, however, we need 63 cycles ($T_w = 6,\ T_h = 16,\ T_c = 1024,\ T_m = 22$). We can avoid the accuracy reduction without the channel loop tiling, but more than three times of cycles are required to compute layer. Therefore, channel loop tiling is essential for an efficient CNN computation.

2) Extension of *psum* storage.

Another approach to eliminate *channel loop tiling-error* is to store full-bit *psum* without extraction. This approach requires larger size of on-chip memory for *psum* tile so that relatively smaller size of on-chip memory are assigned to save input feature map and filter so that the on-chip and off-chip memory access would be increased. Output reuse pattern which updates input feature map and filter feature map tile and reuses *psum* can avoid transferring *psum* between on-chip and off-chip memory. However, in some CNN layers, output reuse pattern shows poor energy efficiency compared to input reuse pattern and filter reuse pattern [53].

3) Mapping output feature map to PE.

If the number of *psum* in a tile is smaller than PEs, PEs can temporally hold *psum* in accumulators without the extraction data. In this circumstance, the tiling of channel loop does not affect accuracy of CNNs. However, this approach is forced to use data reuse pattern which is not optimum reuse pattern. Moreover, the constraint that the tile size of the output feature map must be smaller than the number of PEs greatly reduces the computational efficiency.

3.2.3 Channel loop tiling-errors

During the aforementioned procedures for the implementation of CNNs, the error can be occurred because of the interaction between the quantization and channel tiling. In this paper, we focus the CNN accuracy reduction from the *channel loop tiling-error*. In Figure 3.3, we suppose that the channel loop is split into three tiles. max_{accum} and min_{accum} are respectively the largest and smallest number that can be represented with bit width of BW_{accum}, and max_{quant} and min_{quant} are respectively the largest and smallest number that can be represented with bit width of BW_O. After the bias is loaded, the PE starts convolution, and a partial sum is accumulated. When convolution of the first tile is completed, the PE saves the partial sum in the output buffer after rounding. In the second tile and third tile, the stored partial sum is loaded and then the remaining process is similar to the first tile. During the rounding and

reloading a partial sum, *psum*, two types of errors can occur: an **(1) exceeding error** and a **(2) rounding error**.

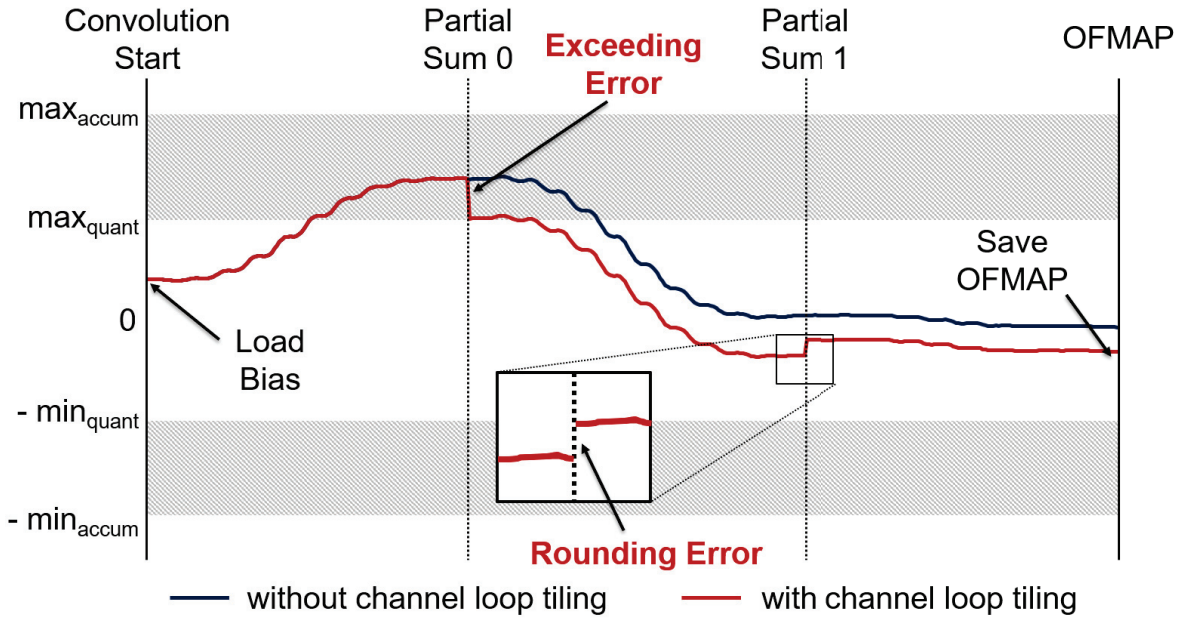


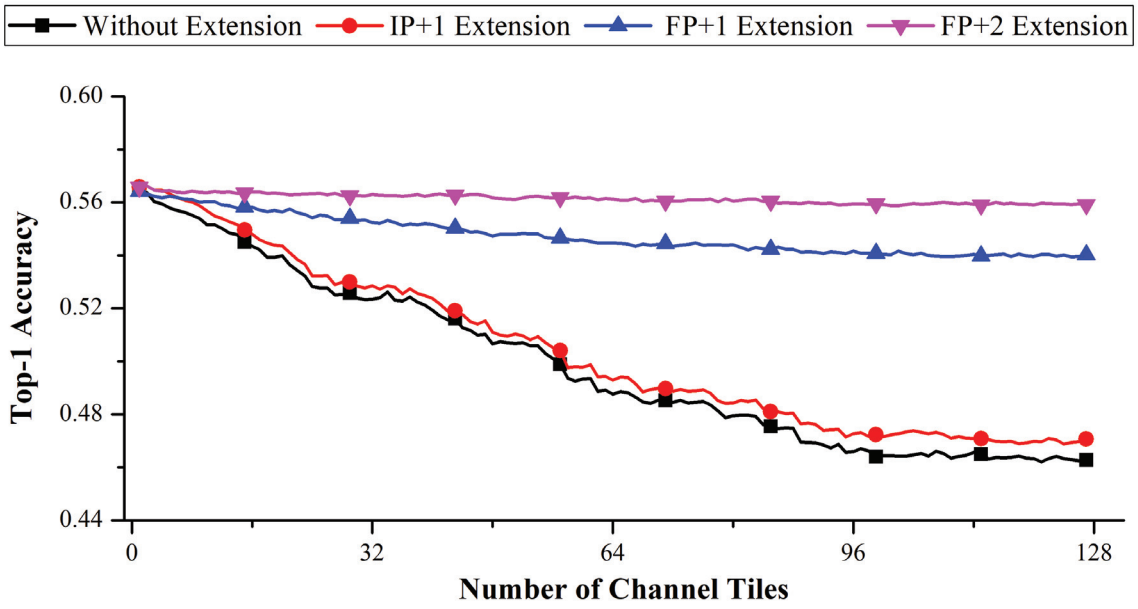
Figure 3.3: Two types of *channel loop tiling-errors*. The value of partial sums (y-axis) are accumulated during three channel loops. If a value of partial sum stored to memory is larger than max_{quant}, exceeding error occurs. The round of the partial sum generates a rounding error.

The exceeding error occurs when *psum* is larger than max_{quant}. At the end of the first tile, *psum* is rounded to max_{quant}, the process yields an exceeding error. Exceeding errors cannot be predicted during the quantization procedure because the final output feature map can be in the range $[min_{quant} - max_{quant}]$. Although exceeding errors are rare, they are significant when they occur, so they can degrade the accuracy of CNNs.

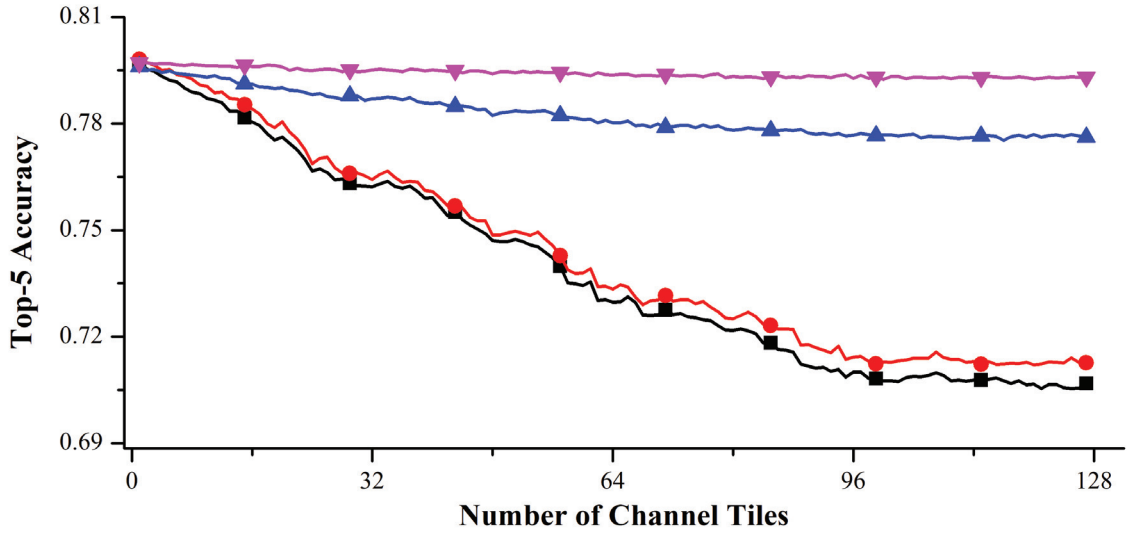
A rounding error is smaller than half of the minimum precision of BW_O. Each rounding error is relatively small, but they occur in almost all outputs and channel tiles. In addition, the accumulated rounding error is much larger than the quantization error. We have counted the frequency and quantity of exceeding errors and rounding errors and Table 3.1 shows the results for image classifications using AlexNet.

The numbers of exceeding errors and rounding errors can be reduced by increasing the bit widths of the integer and fractional parts (Table 3.1). For the analysis, we set the number of channel tile as the number of input channels in each layer. Extending the bit width of the integer part reduced both the

[1] *Freq* is a frequency of errors, (# of error)/(# of total *psum*)×100%.
[2] *Avg* is an average error, (sum of error)/(# of error).
[3] *Exp* is an expected value of error, (sum of error)/(# of total *psum*)×1,000.



(a)



(b)

Figure 3.4: (a) Top-1 and (b) Top-5 accuracy of quantized AlexNet of 50,000 ImageNet dataset with different number of channel tile. To analyze the effect the bit width of partial sum on CNN accuracy, we extend both the bit width of *integer part* (IP) and *fractional part* (FP). If the number of channel tile is '1', channel loops are not tiled.

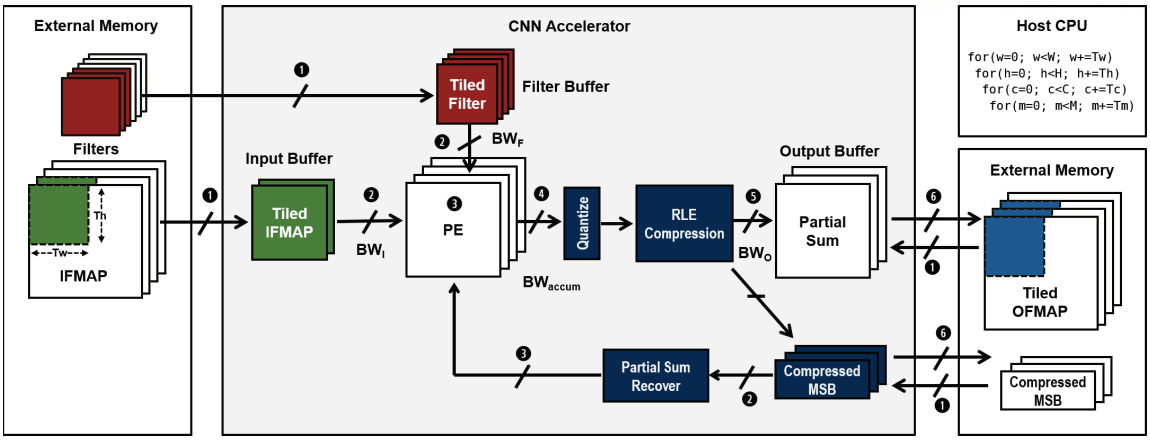


Figure 3.5: Channel loop tiling-aware hardware accelerator.

Table 3.1: Frequency and quantity of exceeding error and rounding error in BW_0 of various configurations (additional bits on integer and fractional parts)

Extension		Exceeding Error			Rounding Error		
IP	FP	$Freq^1$	Avg^2	Exp^3	$Freq$	Avg	Exp
+0	+0	0.011	10.8	1.18	98.2	0.018	18.1
+1	+0	0.000	2.7	0.00	98.2	0.018	18.1
+0	+1	0.011	10.6	1.16	97.8	0.009	9.5
+0	+2	0.011	10.6	1.15	97.0	0.005	4.9

probability and the significance of exceeding errors. On the other hand, The extending bit width of the fractional part can reduce both of the probability and the significance of rounding errors.

The *channel loop tiling-error* degrades the accuracy of a CNN. We have measured the classification rates of 50,000 ImageNet validation dataset [75] using quantized *AlexNet* as shown in Figure 3.4. We have changed the number of channel tiles from one to 127. If the number of channel tiles exceeds the number of input channels of a certain convolution layer, the number of channel tiles of the corresponding convolution layer is set to the number of input channels. For example, the number of input channels of the first convolution layer is three and the number of channel tile is greater than three, the number of channel tiles of the first layer is set to one. Then we have plotted the same graph by extending the bit widths of the integer and fractional parts. As the number of channel tiles increases, the recognition rate decreases; the trend is a result of the exceeding and rounding errors. Also, extending the bit width of *psum* can improve the accuracy of the CNN.

3.3 Channel Loop Tiling-aware Hardware Accelerator

To minimize the overhead for bit extension, we propose a channel loop tiling-aware hardware accelerator which is described in Figure 3.5. A host CPU properly tiles CNN loops and configures the hardware accelerator. Tiles for an input feature map and filter are fetched to the hardware accelerator. If a partial sum exists, the partial sum tile is also loaded. After accumulating bias or partial sum, processing elements (PEs) bring the input feature map and filter that they need for convolution. The bit width of input feature maps and filter are BW_I and BW_F, respectively. The PEs accumulate the multiplication results, of which bit width is $BW_I + BW_F$. To avoid overflow, the accumulator usually has a larger bit width; e.g., $BW_{accum} > (BW_I + BW_F)$. After the partial sum is generated from PEs, it is quantized to BW_O before being saved in external memory. To reduce the error caused by the quantization of the partial sum, we quantize the partial sum that has bit width BW_I.

The increase in the bit width of partial sums requires additional memory space and bandwidth. Moreover, bus widths of memory interfaces are usually multiples of eight bits, so dynamically adjusting bit width is a complex procedure, and circuit overhead is very large. To compress the partial sums, we add a run-length encoding (RLE) compressing and decompressing circuit. RLE saves data and data count as *Run* and *Length*, respectively. In a previous work [44], the RLE have been used to reduce external memory access. They have supposed feature maps are sparse due to activation function of rectified linear unit (ReLu). However, because the partial sums are not rectified using the ReLu, it is not that sparse. Consecutive dense partial sums are less likely to have the same value, and this trait is critical for RLE. If the expected value of *Length* is lower than a certain value, RLE has a difficulty in compressing the partial sum, and can even extend it. To address this problem, we apply RLE at the bit level.

To compress the output feature map in bit level, we analyze a distribution of output feature map. Figure 3.6 shows the distribution of output feature map and absolute value of output feature map in the second layer of a trained AlexNet. The magnitudes of most of the output feature maps are small. When the magnitude of a value is very small, the bits near the MSB are likely to have the same bit value as *sign*. As shown in Figure 3.7, the probability of '1' near MSB is close to 50%, because the probability that the partial sum is positive is near 50%. If the sparsity of consecutive bits is near 50%, it is mostly hard to compress using RLE. To reduce the probability that bits near MSBs will be '1', we take the absolute value of partial sums. The density of absolute values of output feature maps of MSBs is almost zero. Therefore, if we encode the absolute value of MSBs using RLE, we can reduce the memory overhead that is required to store of partial sums of which bit width is extended.

Figure 3.8 shows the procedure of compressing the output partial sums. As mentioned in the previous section, a hardware accelerator rounds the partial sums before saving them. To improve the CNN

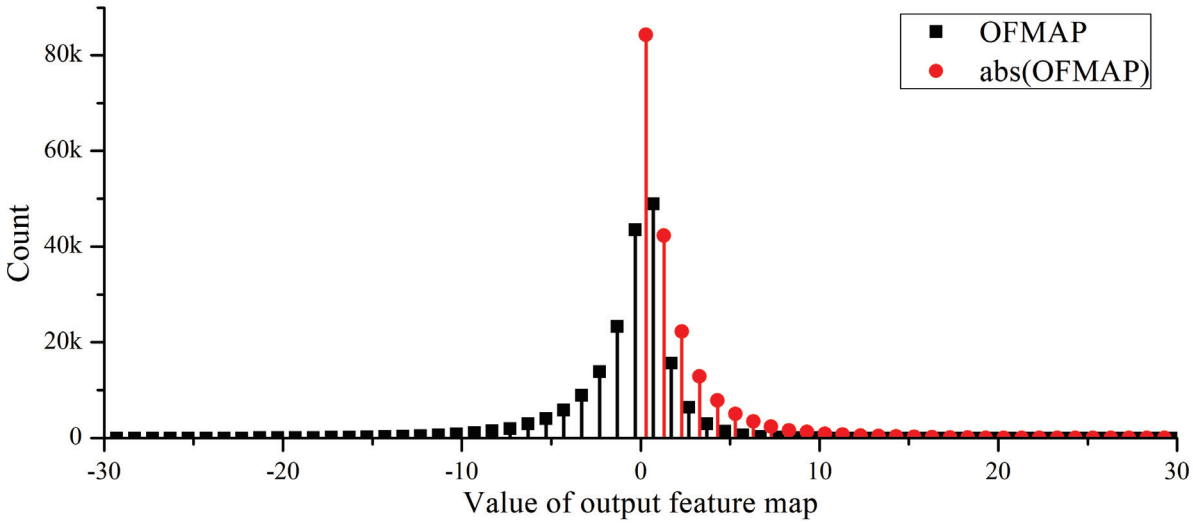


Figure 3.6: Distributions of output feature map and absolute output feature map.

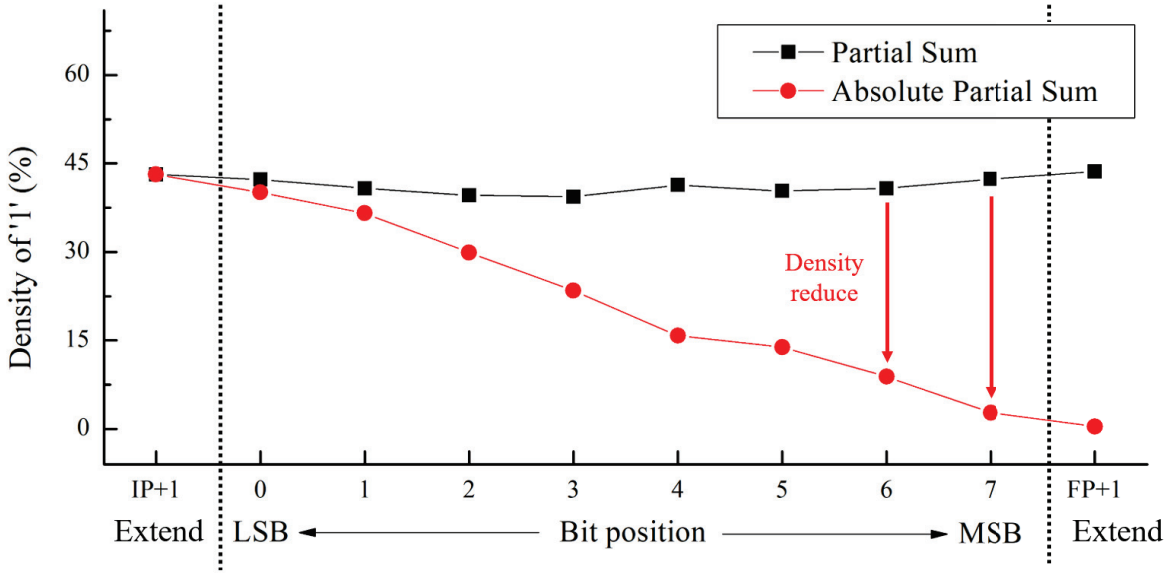


Figure 3.7: Density of ’1’ in each bit position. The absolute values of MSBs are sparse, and the sparse MSBs have an advantage in compression.

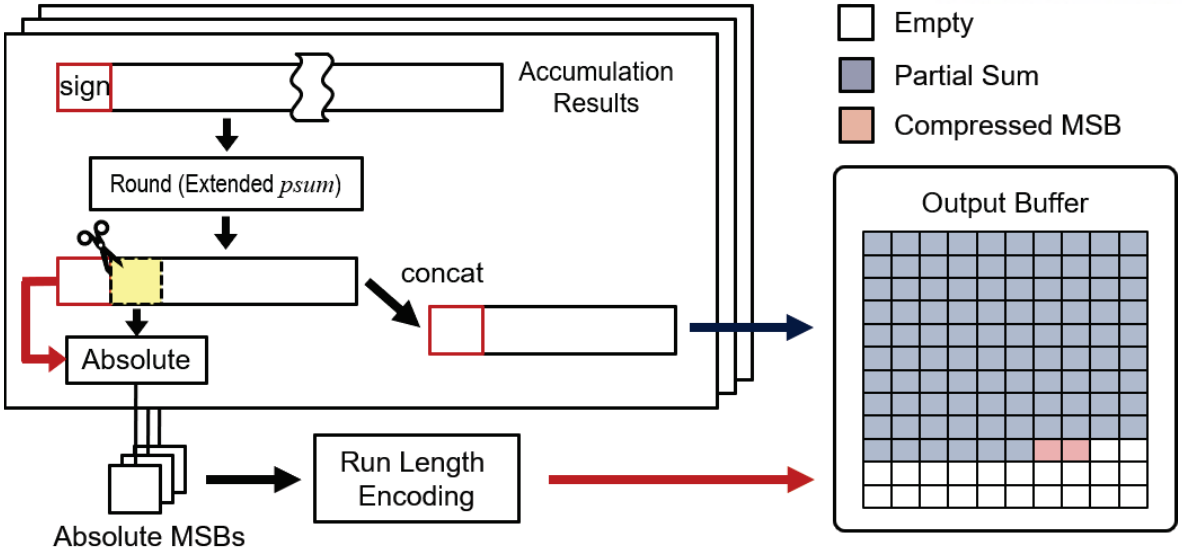


Figure 3.8: Compressing MSBs using the RLE (run-length encoding).

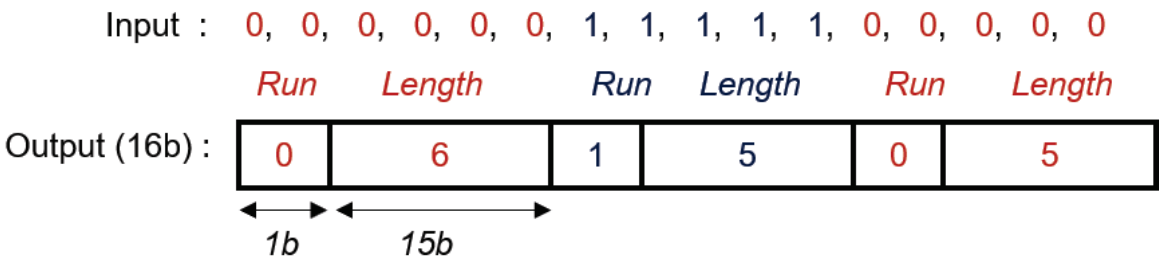


Figure 3.9: An example of the 16-bit RLE operation.

accuracy, we round partial sums to eight bits and some extra bits (one or two-bit extension in IP or FP). To preserve sign of partial sums, seven LSBs and the sign of the rounded partial sums are concatenated. Then the rounded partial sums are stored in an output buffer in the same way that a general hardware accelerator saves an output feature map. Then, to increase the sparsity, we take the absolute values of the remaining outputs. The absolute values are buffered in RLE circuit, which compresses partial sums. The operation principle of the RLE circuit is shown in Figure 3.9. The RLE circuit records the value of consecutive MSBs with a maximum length of 128 (=2^7), 32,768 (=2^{15}), or 2.1 billion (=2^{31}) for 8-bit, 16-bit and 32-bit compression, respectively. The compression ratio and circuit overhead of the RLE circuit will be described in Section 3.4.

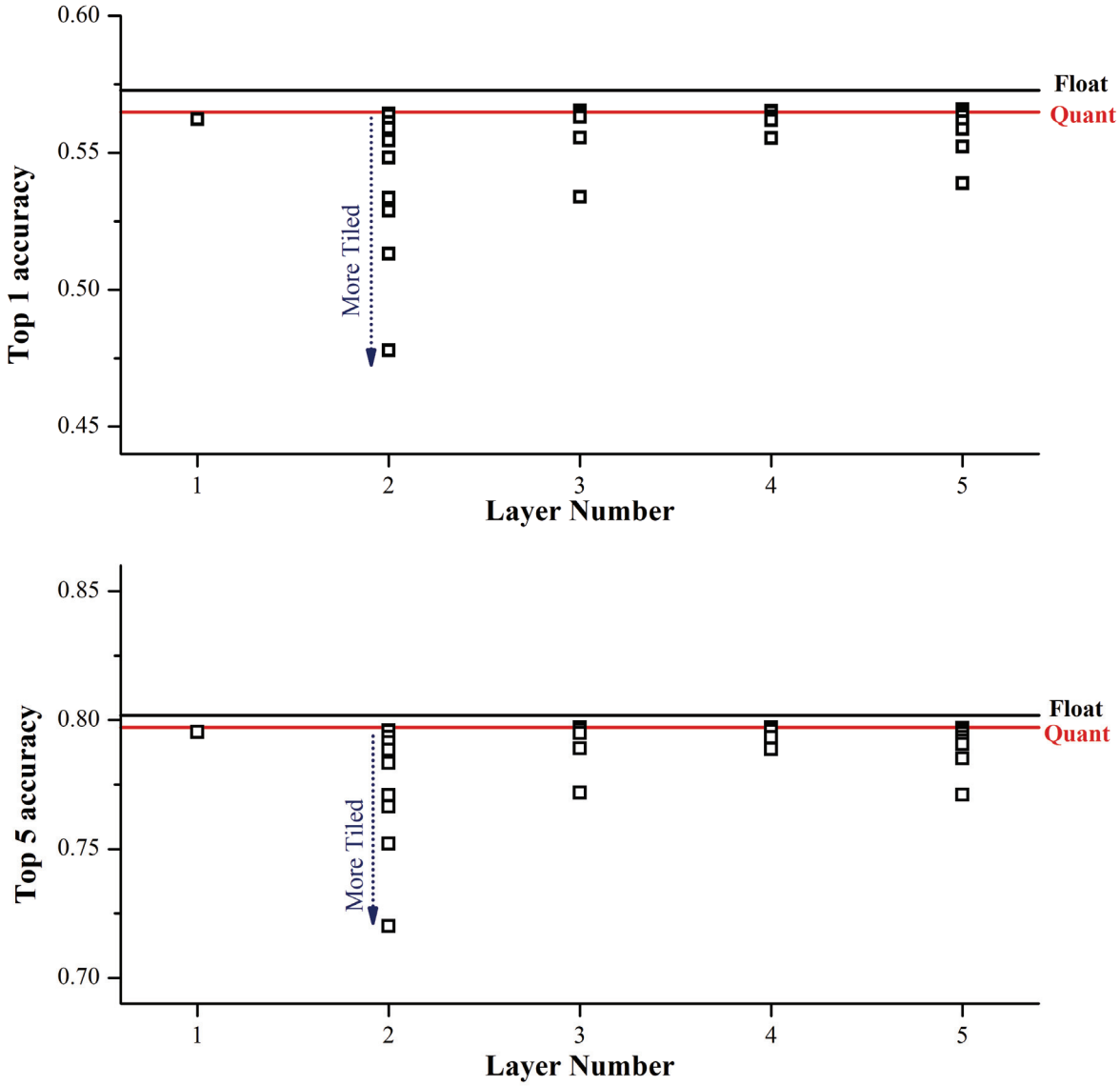


Figure 3.10: *Channel loop tiling-error* effects on accuracy of *AlexNet*. Each layer is tiled without tiling channel loops of the other layers. Red lines: floating-point convolution results, blue: 8-bit quantized convolution results.

3.4 Experimental Setup and Results

3.4.1 Experimental environment

Accuracy Evaluation. To evaluate accuracy loss of channel loop tiled CNNs, we implement CNNs in *C* within darknet [76] and CUDA programming. To mimic channel loop tiling, we quantize partial sums, input feature maps and filters for every channel tile convolution. CNNs are pre-trained using the ImageNet training set and evaluated using 50,000 validation sets. To quantize pre-trained CNNs, we investigate the range of feature maps and filters for all layers of four CNNs (*AlexNet, DarkNet19, ResNet50, Extraction*) [76] during 50,000 inferences. Considering the investigated range, the feature maps and filters are quantized to 8-bit signed integers.

Circuit Overhead. For the estimation of area and power overhead of the proposed methodology, we design a CNN accelerator in RTL. The accelerator has a controller and six PEs which have eight MACs, respectively. The designed RTLs are synthesized to a TSMC 65GP cell library at $1GHz$ clock frequency using *Synopsys Design Compiler* [80]. The supply voltage is set to $0.9V$. The total power consumption is the sum of dynamic power and leakage power.

FPGA Implementation. For the verification of our methodology, we also demonstrates a CNN accelerator by implementing on the PYNQ-Z1 (XC7Z020-1CLG400C), which consists of 13,300 logic slices, 630 kB of BRAM and 220 DSP slices (Figure 3.12). The FPGA communicates with DDR3 at 8,400 Mbps.

3.4.2 Accuracy of channel loop tiled CNN

We have investigated how *channel loop tiling-errors* in each layer of CNNs affect the accuracy of CNNs. Figure 3.10 shows the accuracy of AlexNet. We have tiled a channel loop of each channel without tiling channel loops of the other layers. We have changed the number of channel tiles to all divisors of channel size, *C*, for each layer (e.g., if *C* is eight, we breaks channel loops in one, two, four, and eight tiles).

As the number of channel tiles increases, the accuracy of *AlexNet* decreases owing to *channel loop tiling-error*.

We have also investigated the relationship between memory size of a hardware accelerator and accuracy of CNNs. We have considered 20, 40, 60, 80, 100, 200, and $300kB$ as the memory size of a hardware accelerator. By considering the on-chip memory size, each channel loop of each CNNs has been tiled; the average numbers of channel tiles differ among the CNNs (Table 3.2).

Then we have measured the accuracy of CNNs for each condition, and Figure 3.11 shows the results.

Table 3.2: The average number of channel tiles in each layer of CNNs.

Memory Size [*kB*]	*AlexNet*	*DarkNet19*	*ResNet50*	*Extraction*
20	224.6	133.1	446.4	429.9
40	224.6	83.6	369.3	335.9
60	158.8	54.8	339.5	210.4
80	57.0	29.3	240.4	170.6
100	36.2	25.8	232.5	125.8
200	15.0	11.9	112.1	58.5
300	4.6	6.5	31.3	21.7

The limited on-chip memory size induces *channel loop tiling-error*, which degrades Top-1 and Top-5 accuracy. This degradation is relatively large in *ResNet50* and *Extraction*, because these CNNs are very large, so the number of channel tiles is high. When on-chip memory size is 100*kB*, Top-5 accuracy of *AlexNet*, *DarkNet19*, *ResNet50* and *Extraction* decrease by 8.0%, 1.2%, 15.7% and 36.2%, respectively.

We have considered three extension cases; one bit extension for *integer part*, and one bit and two bit extension in *fractional part*. Figure 3.13 shows the accuracy results on different on-chip memory size for each bit extension case. From the results, we can observe that the bit extension of *fractional part* is more effective than that of *integer part* in most cases.

By extending one bit of *fractional part*, we can recover Top-5 accuracy of *AlexNet* by 5.7%, *DarkNet19* by 1.2%, *ResNet50* by 14.3%, and *Extraction* by 27.9% considering 100*kB* of on-chip memory size. When the fractional part is extended to two bits, the accuracy of CNNs becomes the same level as without channel tiling.

3.4.3 Evaluation of the proposed method

To minimize the memory overhead due to the bit extension, we have encoded partial sum with three different *Length* (8-bit, 16-bit and 32-bit). Without compression, 12.5% of memory overhead per every one bit extension occurs. Table 3.3 shows the average memory overhead of different CNNs. The proposed method shows superior compression ratio. The average memory overheads of compressed MSBs with 8-bit, 16-bit and 32-bit RLE are 0.136%, 0.012%, and 0.022%, respectively. Among three *Length*, 16-bit compression shows the best compression ratio. In most cases, utilization of on-chip memory is not 100%, so we can store 0.012% of additional data without increasing the on-chip memory size.

We have also investigated circuit overhead of the 16-bit run-length encoder and decoder circuit (Ta-

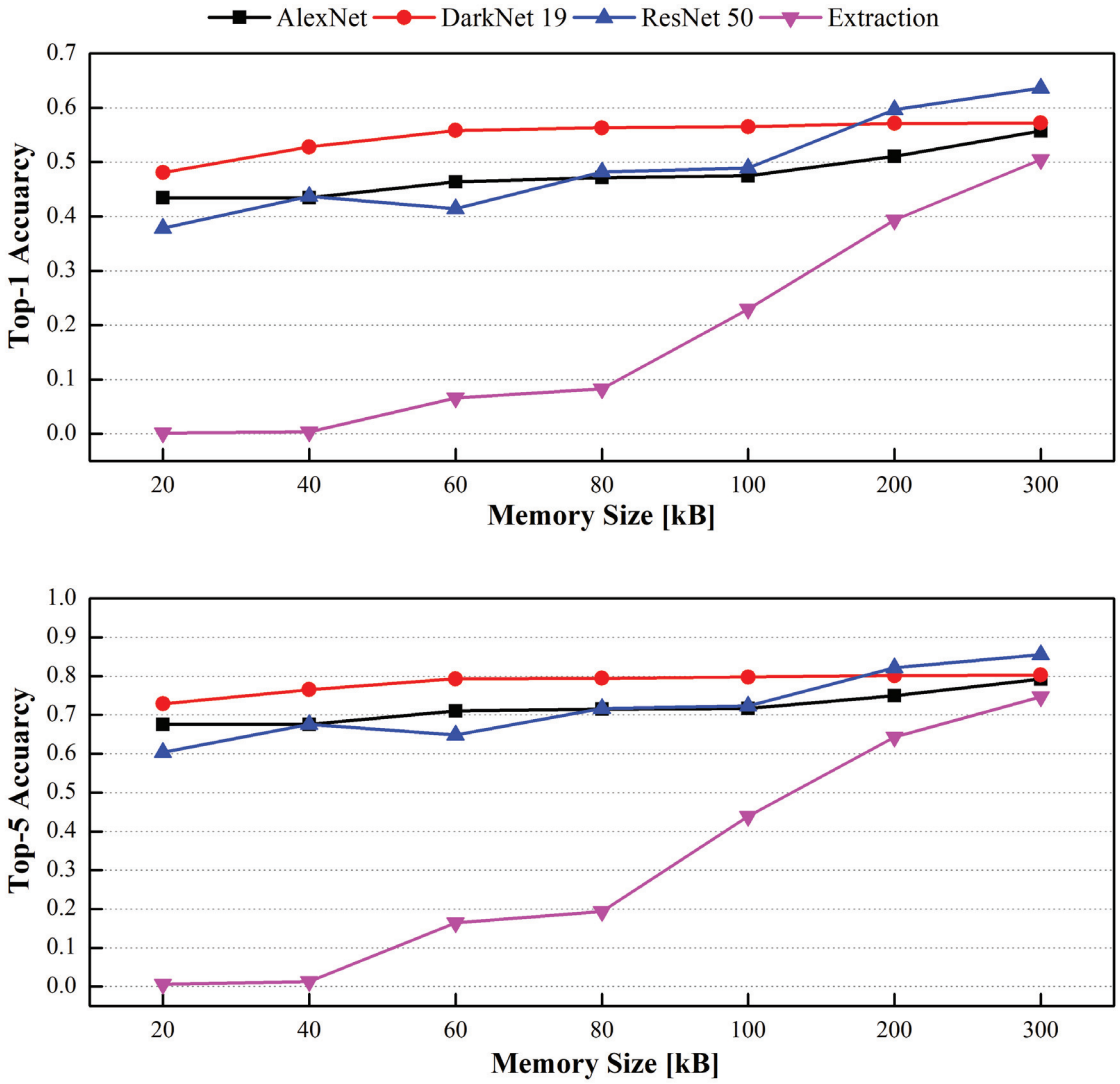


Figure 3.11: The relationship between on-chip memory size (x-axis) and the accuracy (y-axis) of different CNNs. If the available memory size is small, channel loops are splits into much more small tiles.

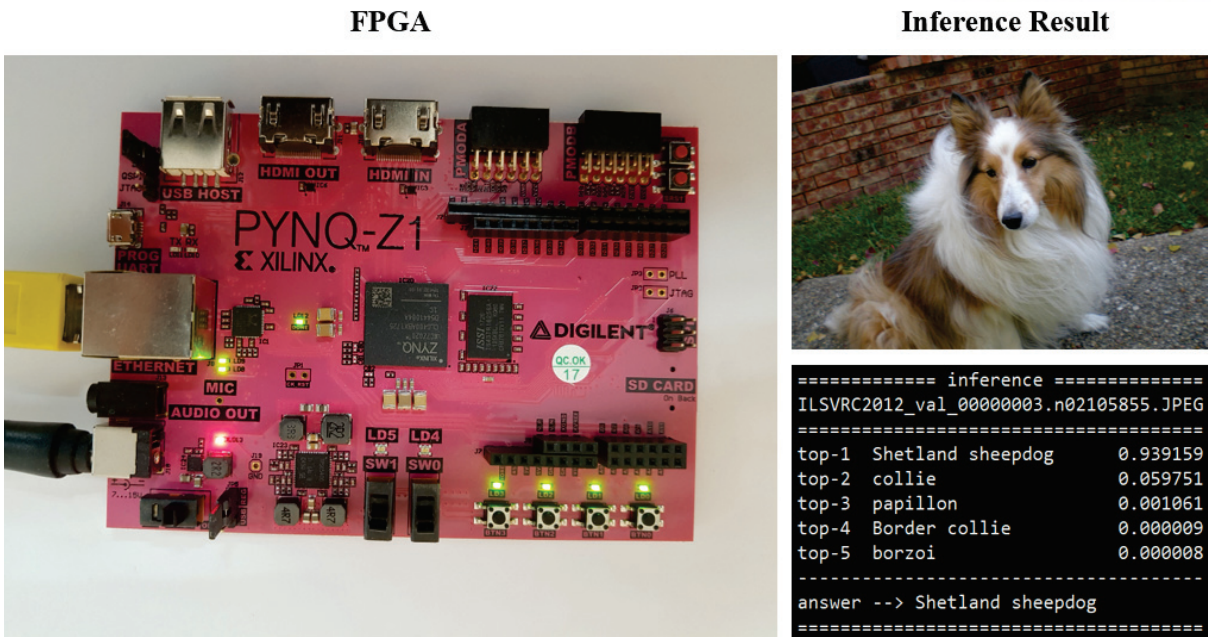
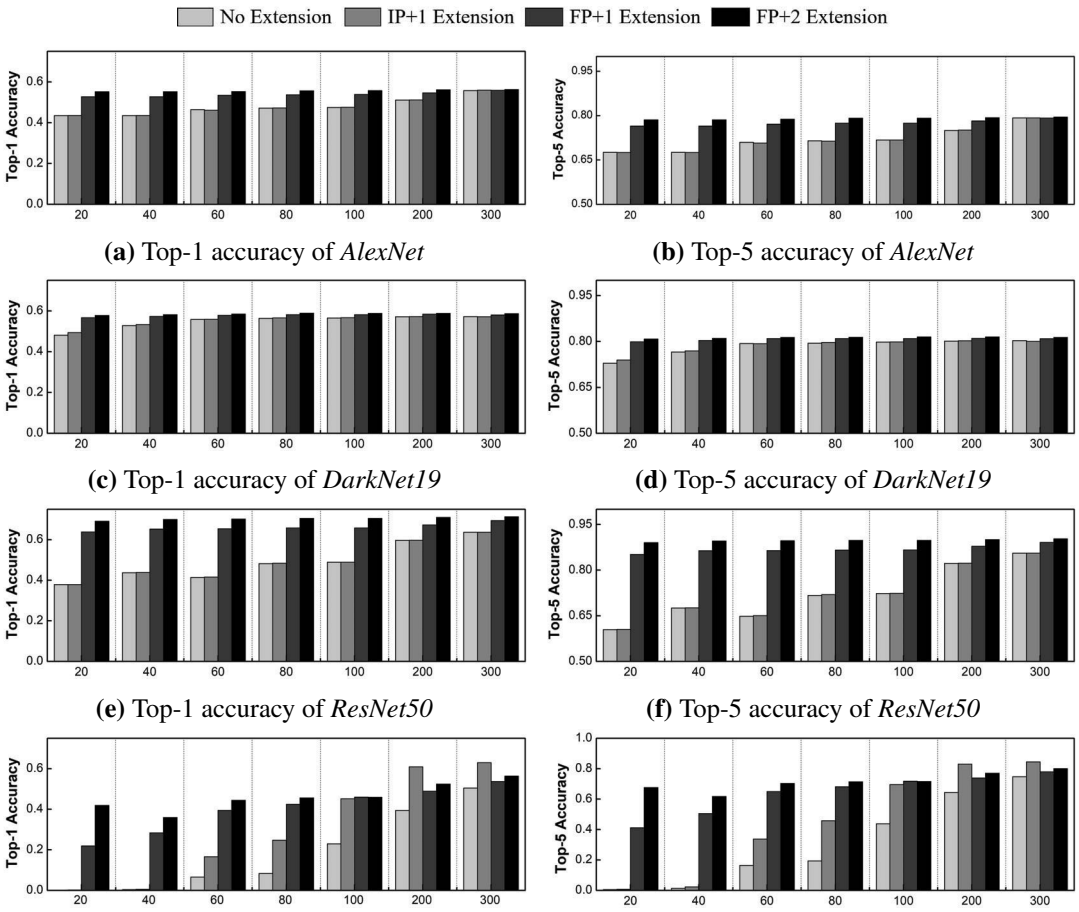


Figure 3.12: Pre-trained CNN for image classification is implemented on PYNQ-Z1 (XC7Z020-1CLG400C).

ble 3.4). The area and power of the encoder and the decoder are very small compared to that of the others. The proposed 16-bit encoder and decoder circuit only consumes 2.0% of additional power with 0.95% of circuit area overhead. Our design is also implemented on FPGA shown in Table 3.5. Compared to CNN accelerator, the area and power overhead of run-length encoder and decoder is also small.



(a) Top-1 accuracy of *AlexNet*

(b) Top-5 accuracy of *AlexNet*

(c) Top-1 accuracy of *DarkNet19*

(d) Top-5 accuracy of *DarkNet19*

(e) Top-1 accuracy of *ResNet50*

(f) Top-5 accuracy of *ResNet50*

(g) Top-1 accuracy of *Extraction*

(h) Top-5 accuracy of *Extraction*

Figure 3.13: Restoration of Top-1 and Top-5 accuracy (y-axis) on different on-chip memory size (x-axis, unit : kB). Top-1 accuracy of (a) *AlexNet*, (c) *DarkNet19*, (e) *ResNet50*, and (g) *Extraction*, and Top-5 accuracy of (b) *AlexNet*, (d) *DarkNet19*, (f) *ResNet50*, and (h) *Extraction*.

3.5 Conclusion

When pre-trained CNNs are implemented on an accelerator (e.g., FPGA or implemented ASIC), accuracy of CNNs can be reduced. This paper has introduced channel loop tiling-error, which is a reason for the accuracy reduction. We have partitioned this error into exceeding error and rounding error, then analyzed separately how these errors affect the accuracy of four state-of-the-art CNNs (*AlexNet, DarkNet19, ResNet50, Extraction*). Channel loop tiling caused 13.2% of Top-1 accuracy and 15.2% of Top-5 accuracy loss on average. We also proposed a solution to recover accuracy loss caused by channel loop tiling. By compressing the extended bits, we minimize memory and circuit overhead. On average, 12.0% of Top-1 accuracy and 12.3% of Top-5 accuracy can be recovered at the cost of only 0.012% of additional memory and 1% of circuit area overhead.

Table 3.3: Comparison of memory overhead due to bit extension. The memory overhead is defined by *(additional memory size) / (original memory size)* $\times$ 100%.

Compression with 8-bit run

Extension	*AlexNet*	*DarkNet19*	*ResNet50*	*Extraction*
(IP+1, FP)	0.0986	0.0988	0.1053	0.0986
(IP, FP+1)	0.0991	0.0998	0.1106	0.0993
(IP, FP+2)	0.1989	0.2063	0.2317	0.2035

Compression with 16-bit run

Extension	*AlexNet*	*DarkNet19*	*ResNet50*	*Extraction*
(IP+1, FP)	0.0011	0.0016	0.0151	0.0010
(IP, FP+1)	0.0022	0.0038	0.0262	0.0027
(IP, FP+2)	0.0058	0.0218	0.0756	0.0155

Compression with 32-bit run

Extension	*AlexNet*	*DarkNet19*	*ResNet50*	*Extraction*
(IP+1, FP)	0.0008	0.0017	0.0287	0.0006
(IP, FP+1)	0.0029	0.0062	0.0509	0.0039
(IP, FP+2)	0.0088	0.0407	0.1484	0.0283

Table 3.4: Area and power overhead of run-length encoder and decoder of different length (eight, 16 and 32-bit).

Component	Circuit Area [μm^2]	Total Power [mW]
8-bit Run-length Encoder	471	0.26
16-bit Run-length Encoder	713	0.38
32-bit Run-length Encoder	1178	0.63
8-bit Run-length Decoder	471	0.26
16-bit Run-length Decoder	665	0.48
32-bit Run-length Decoder	1178	0.63
CNN Accelerator	145,731	43.1

Table 3.5: FPGA implementation results.

	LUT slice	Register	DSP slice
8-bit Run-length Encoder	29	33	0
16-bit Run-length Encoder	44	49	0
32-bit Run-length Encoder	63	81	0
8-bit Run-length Decoder	32	41	0
16-bit Run-length Decoder	44	47	0
32-bit Run-length Decoder	65	89	0
CNN Accelerator	1627	2311	8

Chapter IV

Memory Level Approach

Convolutional neural networks (CNNs) are widely used in various applications, such as segmentation [54], classification [55] and detection[56]. Recently, accuracy of CNNs has continued to improve at the cost of huge network size. At the result, the state-of-the CNNs require millions computations and hundreds mega byte of parameters [57, 58, 59]. A number of operations and parameter movements consume a lot of energy. According to a previous research [60], DRAM access accounts for most of energy consumption of CNN accelerator. To reduce energy consumption of CNN accelerator, it is most efficient way to minimize DRAM access.

To minimize DRAM access for feature maps, we can exploit statics of feature maps. The state-of-the-art CNNs use rectified linear units (ReLU) [69] as an activation function because of low computational complexity and ease of "vanishing gradient" problem. ReLU is a piece-wise linear function that returns zero if the input is negative, otherwise, it returns the input directly. As a consequence, the feature maps of CNN using ReLU have many zeros. By exploiting the inherent sparsity, output feature maps are usually compressed in CNN accelerator.

Figure 4.1 shows the architecture of a typical CNN accelerator [61]. At first, an input image is loaded to on-chip global buffer and the accelerator computes convolution operations using multiple processing elements (PEs). The results of convolution operations, output feature maps, are saved to on-chip global to buffer. Before output feature maps are transmitted to off-chip DRAM, they are compressed by the compressor. Although, the compression of feature maps requires additional energy consumption, the energy consumption for compressing feature maps is much smaller than that of accessing DRAM.

Chen et al. adopt run-length compression (RLC) to compress feature maps [61]. RLC compresses consecutive zeros into a single run. The size of run should be precisely designed considering sparsity of feature maps, because RLC shows very different compression ratio depending on the size of run.

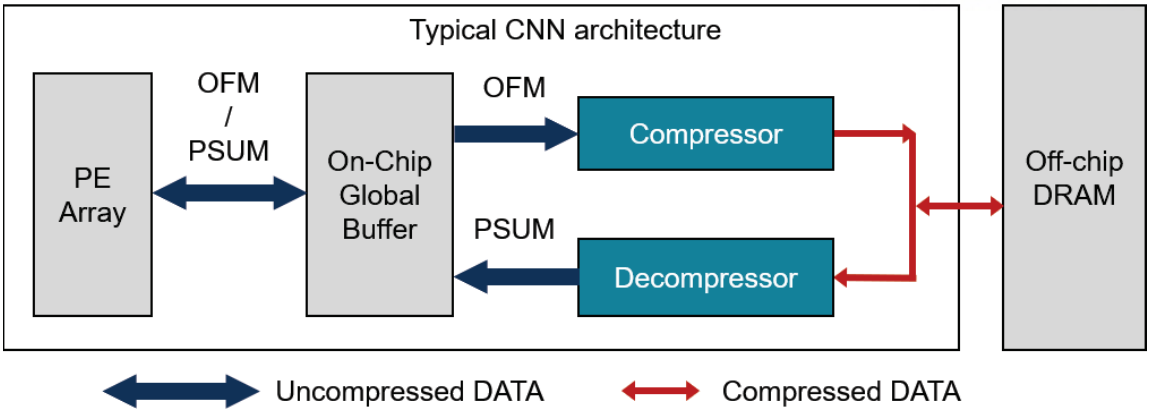


Figure 4.1: Feature map compression circuit of the typical CNN accelerator.

However, the sparsity of features maps varies according to each layer, and it is not easy to find an optimum size of run through all layers. For example, if a RLC is specialized for a certain layer of CNN, which has extremely sparse feature maps, it would show a low compression ratio in the other layers which have less sparse feature maps.

Ryu et al. have adopted a zero-value compression (ZVC) algorithm for the compression of feature maps [62]. ZVC stores feature maps as mask bits and non-zero data. Although ZVC shows relatively high compression ratio regardless of sparsity, it has fixed size of mask. When bit-width of feature maps is long enough, such as 32-bit floating point or 32-bit fixed point precision, the overhead of mask does not matter. However, when the CNN is quantized to 8-bit or less, the mask size is not negligible. According to our observation, ZVC works poorly for the quantized CNN.

In this chapter, we propose a novel compression algorithm, grid-based run-length compression (GRLC), which shows higher compression ratio regardless of sparsity. Furthermore, the proposed GRLC works well for the quantized CNN. To improve the compression ratio, we exploit the spatially correlated property of feature maps. The feature maps, which have non-zero values, are clustered in spatial domain. We group the clustered non-zero feature maps to reduce the size of the mask, which is used for indicating non-zero data. The followings are main contributions of our work.

- We investigate the spatial correlations of output feature maps with spatial auto-correlation.
- We propose a novel compression algorithm by exploiting spatial correlation of output feature maps. For the evaluation, we have evaluated the compression ratio of our algorithm and compared with ZVC and RLC.
- We also implement hardware for the compression of feature maps using Verilog. The area and the power overhead of the proposed compression circuit are also evaluated.

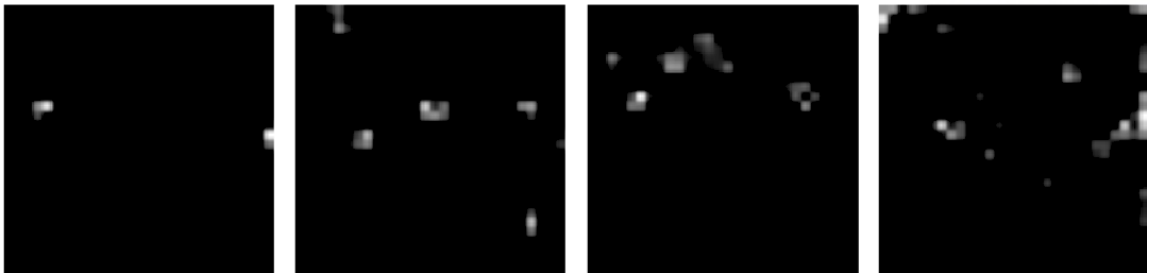


Figure 4.2: The feature maps of 12-th layer of VGG-16. The 8-bit feature maps are mapped to grayscale image. '0' and '255' are converted to black and white pixel, respectively.

4.1 Grid-based Run-length Compression

4.1.1 Motivation

Convolution layers consists of normalization, convolution and activation function. Activation function of CNNs is nonlinear and this nonlinear property allows CNNs to perform non trivial task such as image classification, detection and segmentation. In early CNNs such as multi-layer perceptron, sigmoid function was used for activation function. However, due to "vanishing gradient" problems and hardware complexity of sigmoid function, the-state-of-the-art CNNs use rectified linear unit (ReLU) for activation function [57, 58, 59]. The ReLU is defined by

$$O_i = \begin{cases} I_i & \text{if } I_I \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

while I_i and O_i are i-th weighted sum, generated by the convolution operation, and corresponding activation results, respectively. The output feature maps become sparse, because the negative convolution results become '0' after ReLU as shown in the above equation. If we compress the feature maps by exploiting sparsity of the feature maps, we can significantly reduce energy consumption for DRAM access.

To investigate the characteristics of the feature maps, we extract the feature maps from intermediate layer (12-th layer) of VGG-16, which is trained for ImageNet 2012 classification challenge. The extracted feature maps are converted to 8-bit grayscale images, which is plotted in Figure 4.2. As shown in Figure 4.2, the feature maps are almost dark and a few white pixels are exists, due to inherent sparse property of CNNs. Another characteristic of the feature maps is that white or gray pixels, non-zero output feature maps, are clustered. This means that the feature maps of CNN are "spatially correlated". In other words, if a feature map has non-zero values, its neighbor feature maps are likely to have non-zero

values, and vice versa.

To investigate quantitatively the the spatial correlation, the feature maps of intermediate layers are generated during inferences of 1,000 images of ImageNet 2012 validation set. For the inference, we use VGG-16, which is quantized to 8-bit. We define the spatial correlation in distance i as

$$\rho_i = \frac{\rho_{i,hor} + \rho_{i,ver} + \rho_{i,dia} + \rho_{i,adia}}{4}, \tag{4.2}$$

where $\rho_{i,hor}$, $\rho_{i,ver}$, $\rho_{i,dia}$ and $\rho_{i,adia}$ are spatial autocorrelations between a feature map and copies of itself, which are shifted in the horizontal, vertical, diagonal and anti-diagonal direction, respectively. The correlation, ρ_i, is defined as

$$\rho_i = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma^2}, \tag{4.3}$$

where X and Y are the output feature maps and copies of itself, which are shifted in each direction by distance '1'. Figure 4.3 shows the correlation of the output feature maps of different convolution layers. As shown in this figure, the feature maps are spatially correlated. The graph also shows that the spatial correlation decreases as the distance increases.

4.1.2 Proposed compression algorithm

Figure 4.4 shows the overview of the grid-based run-length compression algorithm (GRLC). First, GRLC divides the feature maps into rectangular tiles, of which size is 2×3. The size of tile is heuristically determined. Then, we classify the tiles with at least one non-zero value as 'non-zero tiles' and the other tiles as 'zero tiles'. GRLC encodes the number of continuous non-activated tiles as 2-bit run. For the non-zero tile, 6-bit mask is generated. '0' in given position indicates that the corresponding value is zero, while '1' indicates that the corresponding value is non-zero. After 6-bit mask, non-zero elements are appended. For entire feature maps, GRLC iterates these procedures.

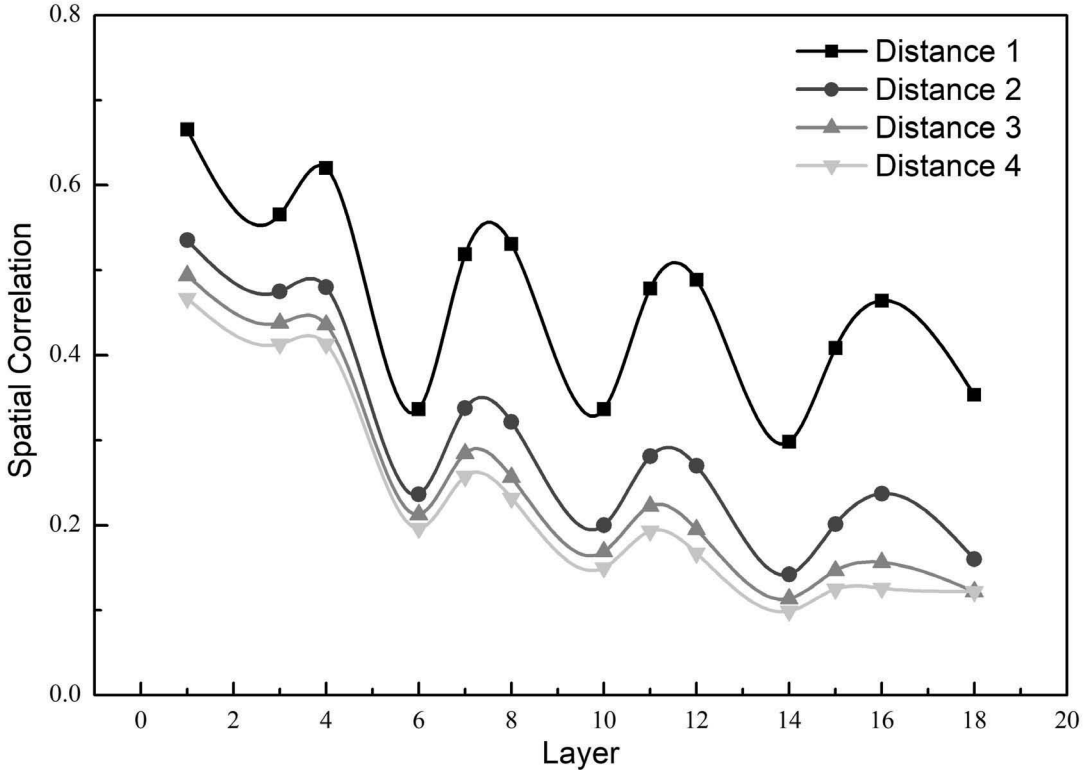


Figure 4.3: Spatial correlation of output feature maps between different distances of VGG-16. For the investigation, the feature maps of intermediate layers are generated during inferences of 1,000 images of ImageNet 2012 validation set. We takes averages of horizontal, vertical, diagonal and anti-diagonal spatial auto-correlation.

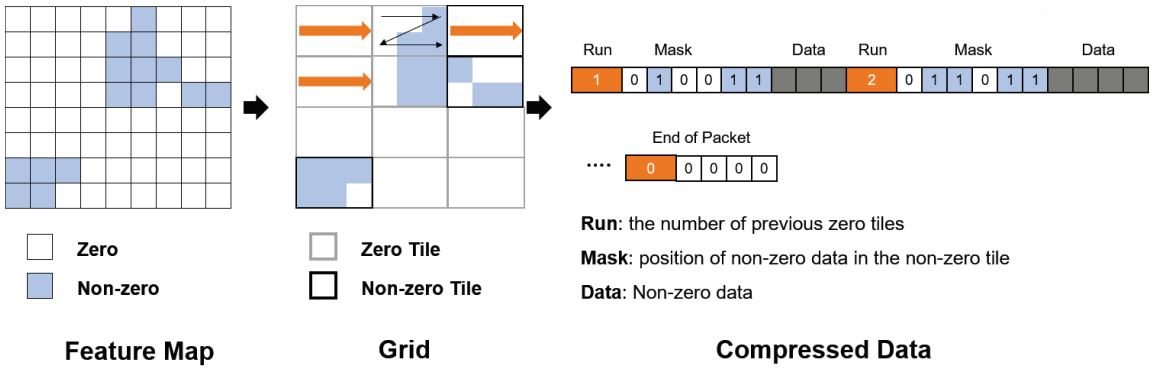


Figure 4.4: Grid-based run-length compression

4.2 Experimental Setup and Results

For the evaluation of compression ratio, RLC, ZVC and GRLC algorithm are implemented using C++. We used VGG16, AlexNet and ResNet18 for the generation of output feature maps. The three CNNs are trained for the classification of the ImageNet ILSVRC-2012 dataset [75]. We obtained intermediate output feature maps using 1k validation examples in ImageNet ILSVRC-2012 dataset using a deep learning framework, Darknet[76].

The compression ratio is defined as

$$Compression\ ratio = \frac{Uncompressed\ Size}{Compressed\ Size}. \tag{4.4}$$

The higher the compression ratio, the higher the performance of the compression algorithm. The compression ratio of uncompressed data equals to '1'. Figure 4.5 shows the compression ratio of the three different compression algorithm using the feature maps of VGG-16. As the sparsity increases, the compression ratio of three algorithms also increases. From the first to the seventh convolution layer, RLC cannot compress the feature maps at all. The average compression ratio of RLC, ZVC and GRLC are 1.09×, 2.21× and 2.34×, respectively. We can improve 1.05× and 0.13× of the compression ratio compared RLC and ZVC, respectively. We also evaluate the compression ratio of the feature maps of ResNet-18 and results are shown in Figure 4.6. The average compression ratios of RLC, ZVC and GRLC are 1.51×, 2.67× and 3.01×, accordingly. GRLC also shows the best compression ratio for ResNet-18.

We also evaluate the performance of the algorithm for further quantized CNN and the results are shown in Figure 4.7. DRAM access is estimated by the reciprocal of compression ratio and it is normalized by the DRAM access of uncompressed data. As the bit-width decreases, DRAM access is

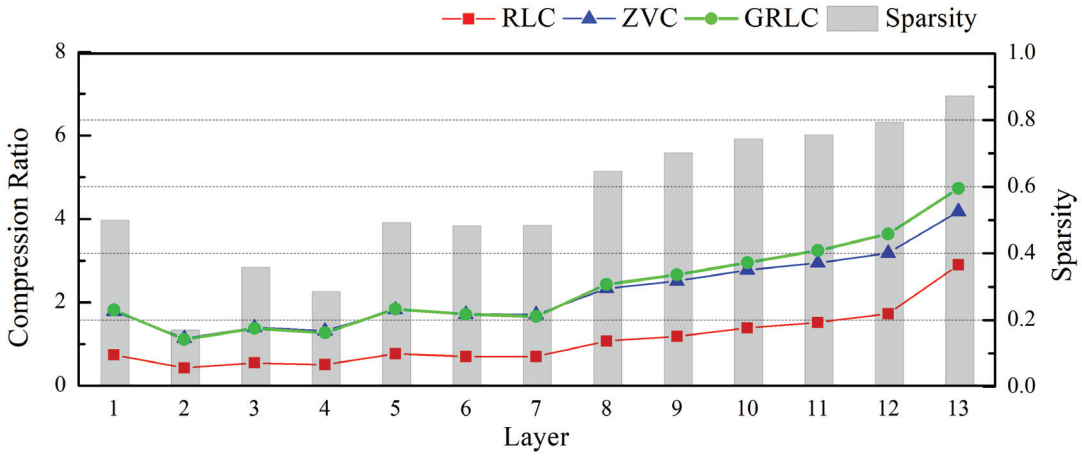


Figure 4.5: The compression ratio (left y-axis) and the sparsity (left y-axis) of the output feature map in each layer of VGG-16.

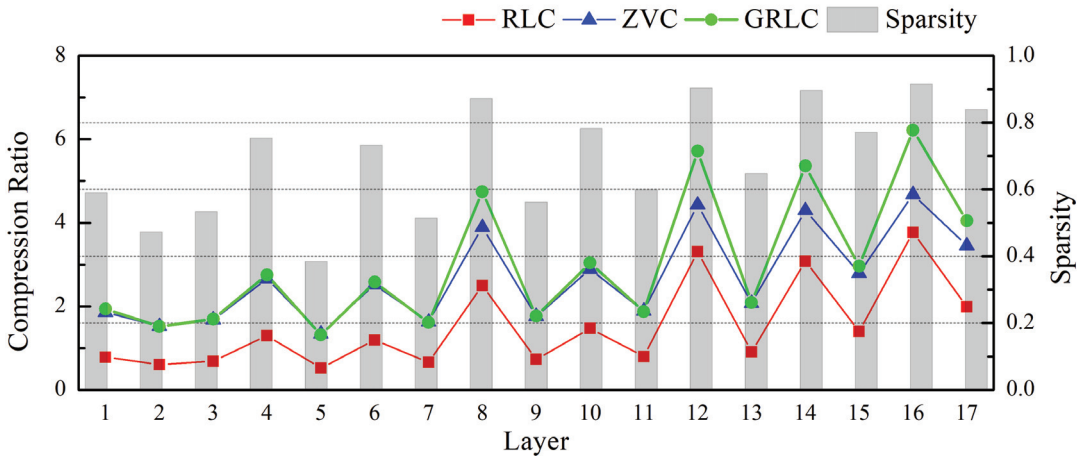


Figure 4.6: The compression ratio (left y-axis) and the sparsity (left y-axis) of the output feature map in each layer of ResNet-18.

also reduced virtue of the increased sparsity. DRAM access of RLC shows the greatest decline, while DRAM accesses of ZVC decreases slightly. This is because ZVC algorithm has fixed size of mask which is fixed regardless of the sparsity and the bit width. Among three algorithms, GRLC shows the lowest DRAM access regardless of bit width. For 4-bit feature maps, GRLC can save 77% of DRAM access, while RLC and ZVC can save 71% and 65% of DRAM access. We also investigate the DRAM access for ResNet-18 in Figure 4.8. We can save maximum 75% of DRAM access (maximum 71% and 65% for RLC and ZVC, each).

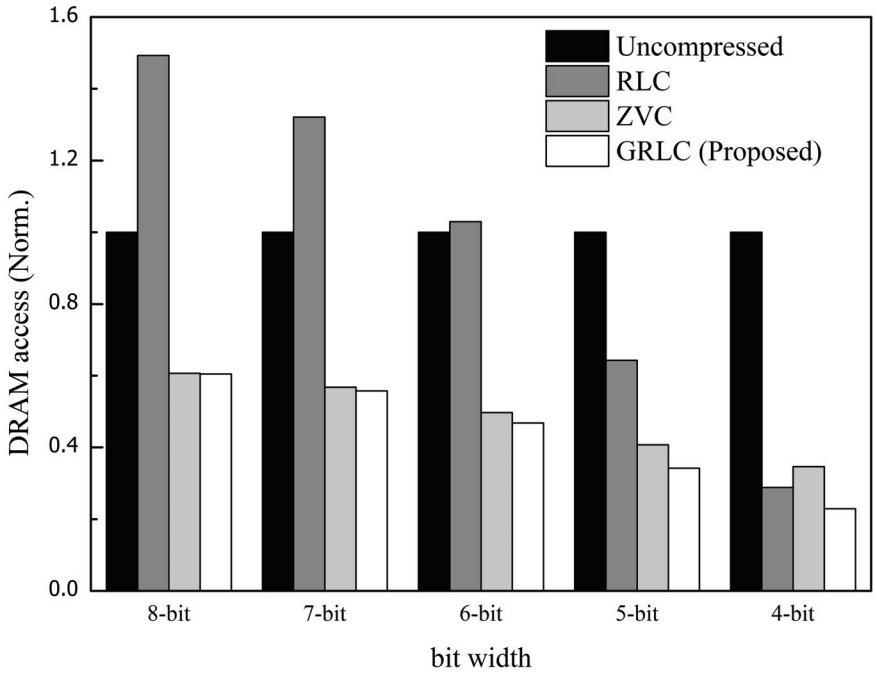


Figure 4.7: The reduced DRAM access by the three different compression algorithm: RLC, ZVC and GRLC in VGG16. The DRAM access is normalized by DRAM access of uncompressed feature maps.

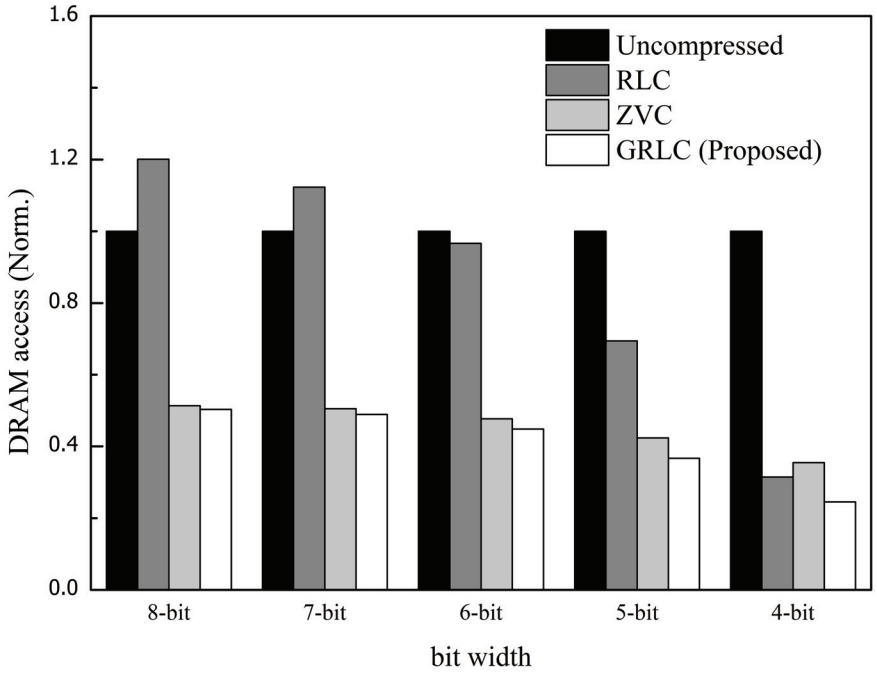


Figure 4.8: The reduced DRAM access by the three different compression algorithm: RLC, ZVC and GRLC in ResNet-18. The DRAM access is normalized by DRAM access of uncompressed feature maps.

4.3 Conclusion

The energy consumption for the DRAM access accounts for most of the energy consumption of the deep learning hardware. To reduce energy consumption for the DRAM access, we propose a compression algorithm for the output feature maps. We investigate the spatial correlations of the output feature maps of VGG-16 and ResNet-18. Our proposed algorithm, GRLC, compresses the feature maps by exploiting the spatially correlated property. GRLC shows higher compression ratio compared to RLC and ZVC both VGG-16 and ResNet-18. Finally, GRLC can save maximum 77% of DRAM access for VGG-16.

Bibliography

[1] A. Canziani, E. Culurciello, and A. Paszke, “Evaluation of Neural Network Architectures for embedded systems.”, *Proc. ISCAS*, 2017, pp.1-4.

[2] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. V. D. Maaten “Exploring the Limits of Weakly Supervised Pretraining.”, *Proc. ECCV*, 2018, pp.181-196.

[3] McCulloch, S. Warren, and P. Walter, “A Logical Calculus of the Ideas Immanent in Nervous Activity.”, *The Bulletin of Mathematical Biophysics*, 5(4), (1943) pp.115-133.

[4] D. O. Hebb, “The Organization of Behavior: A Neuropsychological Theory.”, *John Wiley*, New York, 1964.

[5] B. W. A. C. Farley, and W. Clark, “Simulation of Self-organizing Systems by Digital Computer.”, *Transactions of the IRE Professional Group on Information Theory*, 4(4), (1954) pp.76-84.

[6] F. Rosenblatt, “The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain.”, *Psychological Review*, 65(6), (1958) pp.386.

[7] T. Isokawa, H. Nishimura, and N. Matsui, “Quaternionic Multilayer Perceptron with Local Analyticity.”, *Information*, 3(4), (2012) pp.756-770.

[8] M. Minsky, and S. A. Papert, “An Introduction to Computational Geometry.”, *MIT Press*, 1969.

[9] F. Camillo, “Neural Representation of Logic Gates.”, *Data Science*, 2017.

[10] P. Werbors, “Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.”, *Ph. D. dissertation*, Harvard University, 1974.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition.”, *Proceedings of the IEEE*, 86(11), (1998) pp.2278-2324.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning", *Nature*, 521(7553), (2015) pp.436-444.

[13] E. Park, S. Yoo, and P. Vajda, "Value-aware Quantization for Training and Inference of Neural Networks", *Proc. IJCV*, 2018, pp.580-595.

[14] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI", *Proc. ISSCC*, 2017, pp.246-247.

[15] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks", *Proc. ISCA*, 2018, pp.764-775.

[16] E. Park, D. Kim, and S. Yoo, "Energy-Efficient Neural Network Accelerator Based on Outlier-Aware Low-Precision Computation", *Proc. ISCA*, 2018, pp.688-698.

[17] D. Kim, J. Ahn, and S. Yoo, "Zena: Zero-aware neural network accelerator", *IEEE Design & Test*, 35(1), (2017) pp.39-46.

[18] D. M. Malcolm and G. D. Andrew, "Multiplierless FIR Filter Design Algorithms", *Proc. SPL*, 2005, pp.186-189.

[19] C. Y. Yao, H. H. Chen, T. F. Lin, C. J. Chien and X. T. Hsu, "A Novel Common-Subexpresiion-Elimination Method for Synthesizing Fixed-Point, FIR Filters", *IEEE Trans. CAS I*, 51(11), (2004) pp.2215-2221.

[20] J. H. Choi, N. Banerjee and K. Roy, "Variation-Aware Low-Power Synthesis Methodology for Fixed-Point FIR Filters", *IEEE. Trans. CAD*, 28(1) (2009) pp.87-97.

[21] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Statistical Analysis and Modeling for Error Composition in Approximate Computation Circuits", *Proc. ICCD*, 2013, pp.47-53.

[22] V. Gupta, D. Mohapatra, P. P. Sang, A. Raghunathan and K. Roy, "IMPACT:IMPrecise Adders for Low-Power Approximate Computing", *Proc. ISLPED*, 2011, pp.409-414.

[23] N. Zhu, W. L. Goh and K. S. Yeo, "An Enhanced Low-Power High-Speed Adder for Error-Tolerant Application", *Proc. ISIC*, 2009, pp.69-72.

[24] S. L. Lu, "Speeding Up Processing with Approximation Circuits", *IEEE Computer*, 37(3) (2004) pp.67-73.

[25] A. K. Verma, P. Brisk and P. lenne, "Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design", *Proc. DATE*, 2008, pp.1250-1255.

[26] D. Shin and S. K. Gupta, "A Re-design Technique for Datapath Modules in Error Tolerant Applications", *Proc. ATS*, 2008, pp.431-437.

[27] A. B. Khang and S. H. Kang, "Accuracy-Configurable Adder for Approximate Arithmetic Designs", *Proc. DAC*, 2012, pp.820-825.

[28] R. Venkatesan, A. Agarwal, K. Roy and A. Raghunathan, "MACACO: Modeling and Analysis of Circuits for Approximate Computing", *Proc. ICCAD*, 2011, pp.667-673.

[29] P. Kulkarni, P. Gupta and M. Ercegovac, "Trading Accuracy for Power with an Underdesigned Multiplier Architecture", *Proc. VLSI Design*, 2011, pp.346-351.

[30] K. J. Cho, K. C. Lee, J. G. Chung and K. K. Parhi, "Design of Low-Error Fixed-Width Modified Booth Multiplier", *IEEE Trans. VLSI*, 12(5) (2004) pp.522-531.

[31] C. H. Chang and R. K. Satzoda, "A Low Error and High Performance Multiplexer-Based Truncated Multiplier", *IEEE Trans. VLSI*, 18(12) (2010) pp.1767-1771.

[32] J. P. Wang, S. R. Kuang and S. C. Liang, "High-Accuracy Fixed-Width Modified Booth Multipliers for Lossy Applications", *IEEE Trans. VLSI*, 19(11) (2011) pp.52-60.

[33] C. Liu, J. Han and F. Lombardi, "A Low-Power, High-Performance Approximate Multiplier with Configurable Partial Error Recovery", *Proc. DATE*, 2014, pp.95.

[34] F. Farshchi, M. S. Abrishami and S. M. Fakhraie, "New Approximate Multiplier for Low Power Digital Signal Processing", *Proc. CADS*, 2013, pp.25-30.

[35] D. Goodman and M. Carey, "Nine Digital Filters for Decimation and Interpolation", *Proc. TASSP*, 1977, pp.121-126.

[36] F. Xu , C. H. Chang and C. C. Jong, "Design of Low-Complexity FIR Filters based on Signed-Powers-of-Two Coefficients with Reusable Common Subexpression", *Proc. TCAD*, 2007, pp.1898-1907.

[37] K. Johansson, "Low Power and Low Complexity Shift-and-Add Based Computations ", *Ph. D dissertion*, 2008.

[38] D. Shi and Y. J. Yu, "Design of Linear Phase FIR Filters With High Probability of Achieving Minimum Number of Adders", *Proc. TCAS I*, 2011, pp.126-136.

[39] S. Rosa, Vagner, E. Costa, J. C. Monteiro and S. Bampi, "An Improved Synthesis Method for Low Power Hardwired FIR Filters", *proc. SBCCI*, 2004, pp.237-241.

[40] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Proc. NIPS*, 2012, pp.1097-1105.

[41] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation", *Proc. CVPR*, 2015, pp.3431-3440.

[42] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", *Proc. CVPR*, 2016, pp.170-171.

[43] A. Canziani, A. Paszke and E. Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications", *arXiv preprint arXiv:1605.07678*, 2016.

[44] Y.-H. Chen, T. Krishna, J. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks", *IEEE JSSC*, 52(1), (2017) pp.127-138.

[45] Y. Shen, M. Ferdman and P. Milder, "Maximizing CNN Accelerator Efficiency Through Resource Partitioning", *Proc. ISCA*, 2017, pp.535-547.

[46] A. Rahman, S. Oh, J. Lee and K. Choi, "Design Space Exploration of FPGA Accelerators for Convolutional Neural Networks", *Proc. DATE*, 2017, pp.1147-1152.

[47] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao and J. Cong, "Optimizing Fpga-based Accelerator Design for Deep Convolutional Neural Networks", *Proc. ISFPGA*, 2015, pp.161-170.

[48] D. Lin, S. Talathi and V. Annapureddy, "Fixed Point Quantization of Deep Convolutional Networks", *Proc. ICML*, 2016, pp.2849-2858.

[49] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients", *arXiv preprint arXiv:1606.06160*, 2016.

[50] A. Zhou, A. Yao, Y. Guo, L. Xu and Y. Chen, "Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights", *arXiv preprint arXiv:1702.03044*, 2017.

[51] E. Park, D. Kim and S. Yoo, "Energy-efficient Neural Network Accelerator Based on Outlier-aware Low-precision Computation", *Proc. ISCA*, 2018, pp.688-698.

[52] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, P. Chuang and L. Chang, "Compensated-DNN: Energy Efficient Low-Precision Deep Neural Networks by Compensating Quantization Errors", *Proc. DAC*, 2018, pp.1-6.

[53] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu and S. Wei, "Deep Convolutional Neural Network Architecture with Reconfigurable Computation Patterns", *IEEE Trans. on CAD* 25(8) (2017), pp.2220-2233.

[54] Y. Lyu, L. Bai and X. Huang, "Road Segmentation using CNN and Distributed LSTM", *Proc. ISCAS*, 2019, pp.1-5.

[55] A. Rattani, N. Reddy and R. Derakhshani, "Convolutional Neural Network for Age Classification from Smart-Phone based Ocular Images", *Proc. IJCB*, 2017, pp.756-761.

[56] A. Dimou, P. Medentzidou, F. A. Garcia and P. Daras, "Multi-target Detection in CCTV Footage for Tracking Applications using Deep Learning Techniques", *Proc. ICIP*, 2016, pp.928-932.

[57] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Proc. In Advances in NIPS*, 2012, pp.1097-1105.

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", *Proc. CVPR*, 2016, pp.770-778.

[59] K. Simonyan, and Z. Andrew, "Very Deep Convolutional Networks for Large-Scale Image Recognition.", *arXiv preprint :1409.1556*, 2014.

[60] F. Tu, W. Wu, S. Yin, L. Liu and S. Wei, "RANA: Towards Efficient Neural Acceleration with Refresh-Optimized Embedded DRAM", *Proc. ISCA*, 2018, pp.340-352.

[61] Y. Chen, T. Krishna, J. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks", *Proc. ISSCC*, 2016, pp.262-263.

[62] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon and S. W. Keckler, "Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks", *Proc. International Symposium on HPCA*, 2018, pp.78-91.

[63] Q. Deng, L. Jiang, Y. Zhang, M. Zhang and J. Yang, "DrAcc: a DRAM based Accelerator for Accurate CNN Inference", *Proc. DAC*, 2018, pp.1-6.

[64] A. H. Robinson and C. Cherry, "Results of a Prototype Television Bandwidth Compression Scheme", *Proceedings of the IEEE*, 55(3), (1967) pp.356-364.

[65] S. Han, H. Mao and W.J. Dally, "Deep compression: Compressing Deep Neural Networks with Pruning, trained Quantization and Huffman Coding", *preprint arXiv:1510.00149*, 2015.

[66] "G. Shomron and U. Weiser, "Spatial Correlation and Value Prediction in Convolutional Neural Networks", *IEEE Computer Architecture Letters*, 18(1), (2019) pp.10-13.

[67] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Kecler, W. J. Dally, "SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks" *Proc. ISCA*, 2017, pp.27-40.

[68] Y. Wang, J. Lin and Z. Wang, "An Energy-Efficient Architecture for Binary Weight Convolutional Neural Networks", *IEEE Trans. on VLSI*, 26(2), (2018) pp.280-293.

[69] X. Glorot, A. Borders, Y. BENGIO, "Deep Sparse Rectifier Neural Networks", *Proc. International Conference on AiStats*, 2011, p.315-323.

[70] G. Georgiadis, "Accelerating Convolutional Neural Networks via Activation Map Compression", *Proc. CVPR*, 2019, pp.7085-7095.

[71] T. Wiegand, G. J. Sullivan, G. Bjontegaard and A. Luthra, "Overview of the H. 264/AVC video coding standard", *IEEE Trans. on CSVT*, 13(7), (2003) pp.560-576.

[72] G. J. Sullivan, J. Ohm, W. Han and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard", *IEEE Trans. on CSVT*, 22(12), (2012) pp.1649-1668.

[73] J. Teuhola, "A Compression Method for Clustered Bit-vectors", *Information Processing Letters*, 7(6), (1978) pp.308-311

[74] J. Wen and J. D. Villasenor, "Reversible Variable Length Codes for Efficient and Robust Image and Video Coding", *Proc. IEEE conference on DCC*, 1998, pp.471-480.

[75] O. Russakovsky, J. Deng, H. Su, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *IJCV*, 115(3), (2015) pp.211-252.

[76] J. Redmon, “Darknet: Open Source Neural Networks in C”, http://pjreddie.com/darknet.

[77] *Cadence RTL Compiler User Guide.* http://www.cadence.com.

[78] *Cadence NCVerilog User Guide.* http://www.cadence.com.

[79] *Synopsys PrimeTime User Guide.* http://www.synopsys.com.

[80] *Synopsys Design Compiler User’s Manual.* http://www.synopsys.com.