Ph.D. Dissertation

A Design of Neural Network Processing Element Array with Mixed-Signal Operations

혼합 신호 구동을 포함한

신경망 연산 요소 어레이의 설계

August 2022

Department of Electrical and Computer Engineering

Seoul National University

Seung-Heon Baek

A Design of Neural Network Processing Element Array with Mixed-Signal Operations

지도 교수 김 재 하

이 논문을 공학박사 학위논문으로 제출함
2022 년 8 월

서울대학교 대학원
전기정보공학부
백 승 헌

백승헌의 박사 학위논문을 인준함
2022 년 8 월

Abstract

This work presents a method to mitigate deep neural network (DNN) accuracy drop and energy consumption increase of DNN accelerator hardware by utilizing mixed-signal operations. The proposed accelerator includes an array of 16×16 mixed-signal processing elements (MPEs), which implements signed upper 4-bit of a signed 9-bit input that significantly influences a DNN accuracy with digital operations, and latter unsigned 5-bit that frequently appears with energy-efficient analog operations. The proposed MPE array supports weight precision from signed 1-bit to 9-bit in a bit-serial manner. In addition, this dissertation proposes a cyclic multiply-accumulate scheme that fixes an output precision and performs analog-to-digital conversion by only 2 bits for each cycle to maintain the number of analog-to-digital converted bits regardless of the weight precision.

The efficacy of the proposed accelerator was verified by results obtained from the measurement environment, including a prototype IC fabricated with a 28nm CMOS process and a Xilinx Kintex-7 FPGA KC705 board. The simulations for IC design steps were performed in a verification environment that could quickly and accurately process numerous calculations in DNN examples by combining three-level simulators. A 4-layer MNIST CNN, a 5-layer CIFAR-10 CNN, and a 7-layer CIFAR-100 CNN were used for simulations and tests. The accuracy of the example CNNs on a prototype IC was measured by applying the MPE array calibration method using the gradient descent optimization technique, and tiny MNIST/CIFAR-10 CNN accuracy changes of -0.42~0.33%p was recorded. The energy consumption for each DNN layer decreased by 20.4-46.1% compared to the equivalent case with all digital computations.

Keywords: Analog-to-digital converting, Array architecture, Deep neural network, Neural computing, Neural network accelerator, Reconfigurable architecture

Student Number: 2014-21726

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

With the rapid development of deep neural network (DNN) software, many studies have proposed accelerator hardware for DNN computations. DNNs utilize simple but numerous operations such as multiply-accumulate (MAC), rectified linear unit (ReLU) activation, pooling, and normalization. For example, the VGG-19 model [1], widely used as a benchmark, classifies one 224x224 image and performs multiplications more than 17 billion times. Therefore, since DNNs require a lot of data movement between memory and processing elements (PEs), conventional computing systems with the Von Neumann architecture generate a bottleneck between separated memory and PEs, causing a lot of energy and latency consumption.

Various studies have introduced novel DNN accelerator architectures for replacing the Von Neumann architecture. Many studies like [2-4] minimized the number of memory accesses by placing memory and PEs on the same die, including memory in a PE, or rearranging the

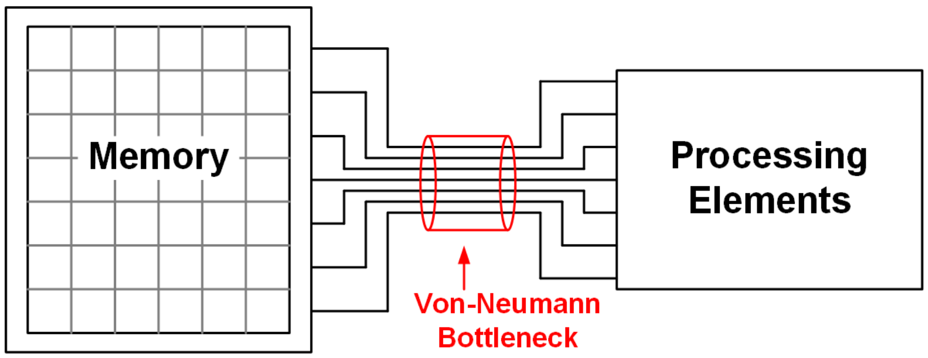


Figure 1.1. Bottleneck on Von-Neumann architecture.

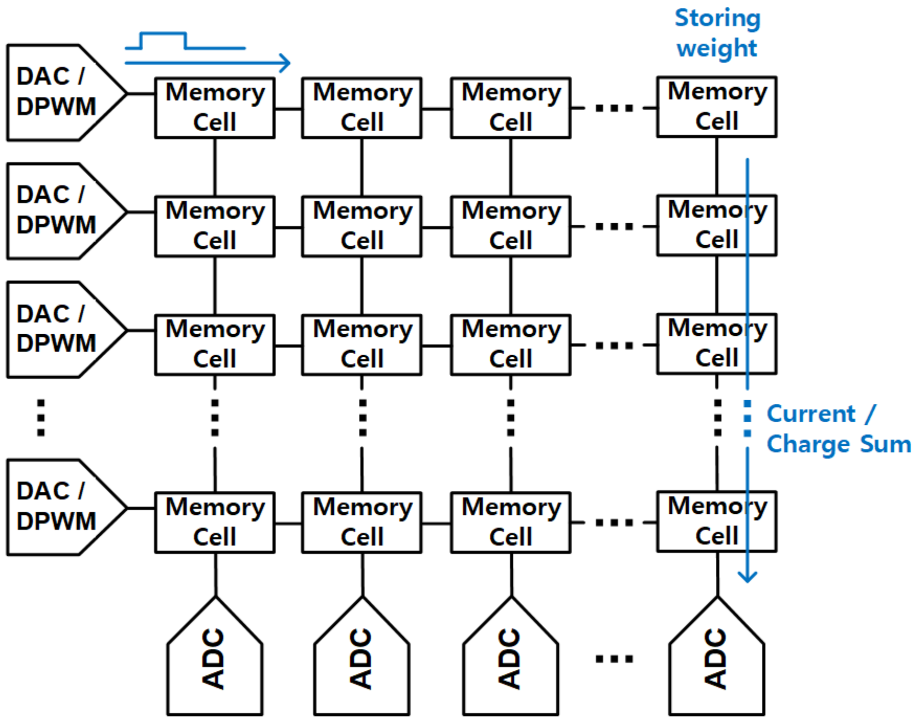


Figure 1.2. Conventional compute-in-memory architecture.

sequences of computations. In addition, various techniques have been proposed, such as quantization [5-6], model compression [7], approximate computing [8], utilizing sparsity [9-10], and voltage-frequency scaling [11-12] to increase hardware performance with sacrificing accuracy of DNNs. Some researchers introduced analog operations to achieve higher energy efficiency [13-15]. Furthermore, a computation-in-memory (CIM) architecture was proposed for increasing energy efficiency and density by reducing the movements

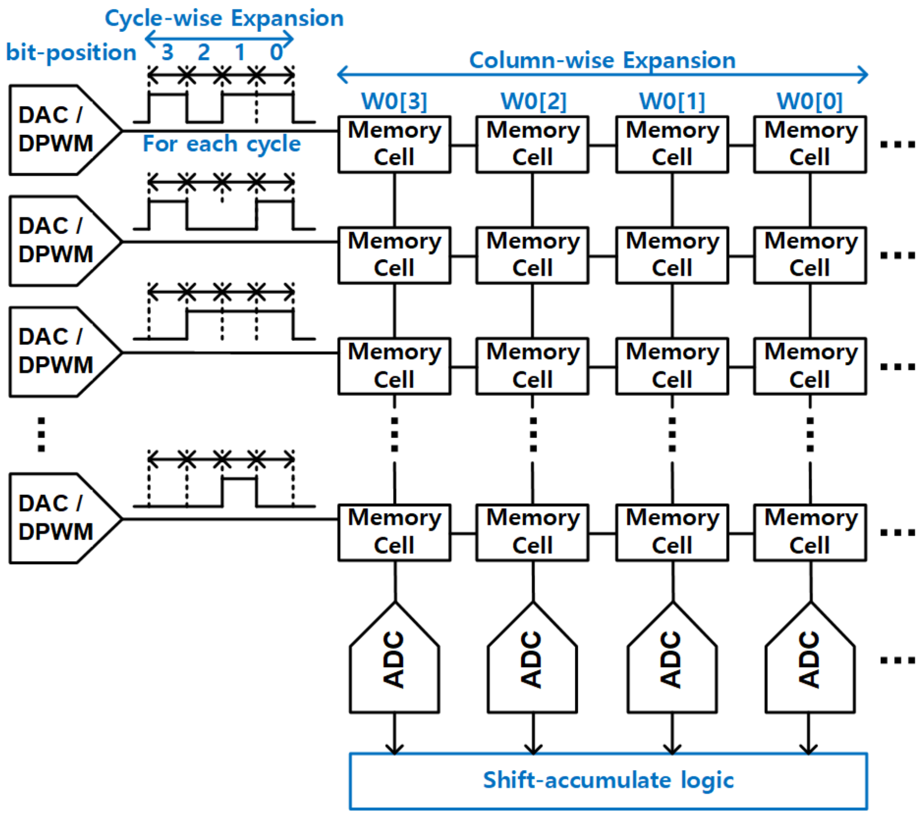


Figure 1.3. Realization of high precision MACs on CIM macro.

of repeatedly computed weights. A CIM architecture realizes a vector-to-matrix multiplication by accumulating weighted-sum represented with analog physical values like currents or charges on memory bit-lines [16-22]. For example, [19] recorded a high energy efficiency of 75.9 TOPS/W with a 2.24-3.14%p decrease in accuracy of DNNs classifying CIFAR-10 datasets.

However, the following difficulties arose due to the inflexibility of CIM architectures fixed after fabrication processes. First, it is difficult to attenuate accuracy drops caused by the uncertainties of analog operations. Area consumption is significant for adding a calibration function for each computing element, like a CIM architecture's memory cell. For example, a 5-bit programmable current source with the ability to calibrate element-wise multiplication in the proposed architecture of this dissertation is 278 times larger than a 6T SRAM cell. Some studies have proposed a method of post-training including the non-ideal characteristics of each fabricated IC [23-24], but additional time-consuming processes are required whenever the target IC or DNN change. Alternatively, there is a way to increase accuracy by utilizing high precision [25-27]. Some researchers implemented high-precision operations by placing input and weight bits separately for each column and cycle [18,21]. However, since all input and weight bits share the same computation path, there is a problem that energy consumptions and distortions from an analog domain are experienced equally regardless of bit position. Therefore, LSB that has less influence on

DNN accuracy [28] than MSB consumes the same energy as MSB. In addition, depending on the size of the DNN, utilization ratios of computation elements or conversion circuits like DACs and ADCs decrease. As shown in Table.1.1, the PE utilization ratio decreases with increasing the PE array size when running the VGG-19 DNN model. As shown in Fig. 1.4, the PE utilization ratio of the 16×16 PE array differs within each layer of the VGG-19 DNN model.

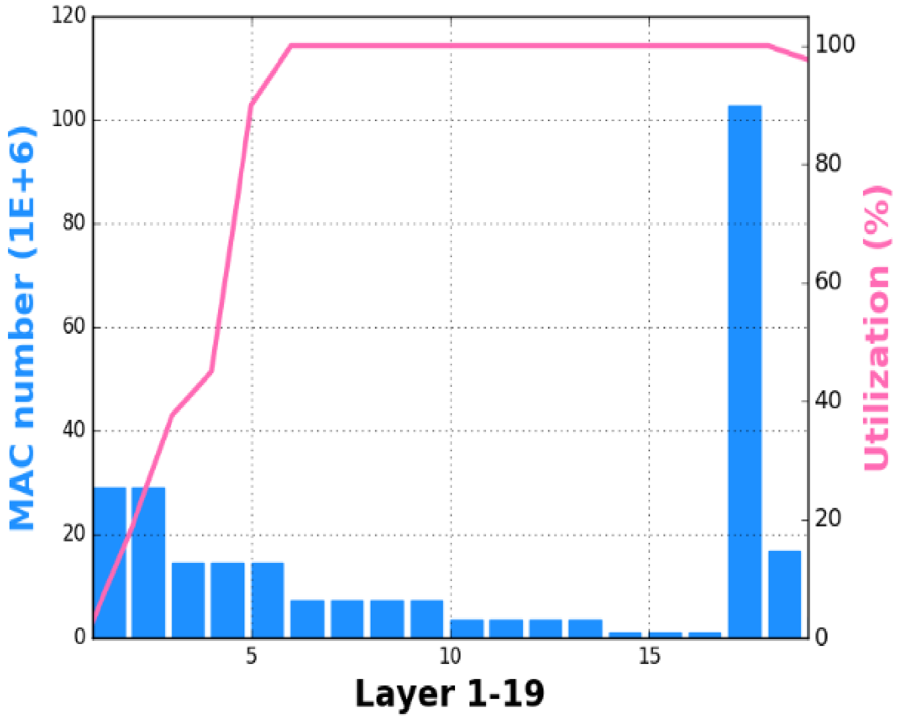


Figure 1.4. PE utilization of each layer of the VGG-19 model on a 16×16 PE array.

Table 1.1. PE Utilization of the VGG-16 model for different PE array sizes.

Col / Row	**16**	**64**	**256**	**1024**
16	98.32%	89.94%	86.76%	81.09%
64	96.07%	90.58%	85.80%	79.28%
256	92.50%	89.83%	84.73%	77.66%
1024	87.85%	87.83%	83.18%	76.06%

1.2 Thesis Contribution and Organization

This dissertation proposes a DNN accelerator IC with an array architecture composed of mixed-signal processing elements (MPEs) to address these problems. An MPE has an SRAM storing 1152 weight bits and supports analog and digital MACs.

The proposed IC has the following characteristics. First, An MPE array computes the upper 4-bit of a signed 9-bit input on an accurate digital domain and the lower 5-bit on an analog domain with high energy efficiency. Second, An MPE adopts a bit-serial manner to load a weight bit for each cycle. In a bit-serial manner, weight bits on an MPE share various circuits for calibration, weight precision reconfigurability, and domain conversions reducing area overhead and sustaining a high PE utilization ratio. Third, this dissertation proposed a cyclic MAC scheme that quantizes the results of analog operations by only 2-bit for each cycle for a total of 8 cycles and accumulates residue on the analog domain. Since the cyclic MAC scheme fixes the number of analog-to-digital converted bits, it is possible to maintain consistent energy efficiency for variable weight precision from signed 1-bit to 9-bit, thereby reducing energy consumed in the lower weight bits.

In addition, a verification environment was implemented using a DNN framework software, a System-Verilog-based simulator, and a circuit-level simulator. Combining each verification level allows accurate and fast verification of numerous DNN operations in the hardware design stage.

Instead of manually calibrating the accelerator of the array structure for each PE, this dissertation also introduces a calibration technique that uses a gradient descent algorithm to adjust the output currents of an MPE array in a direction, offsetting the effect of process variations.

This paper is described in the following order. Chapter 2 introduces the features of the proposed accelerator with an MPE array and the cyclic MAC scheme. Chapter 3 introduces the verification environment and simulation results for the IC designing stage. Chapter 4 introduces an array calibration technique utilizing a gradient descent algorithm and the simulation results. Chapter 5 describes the detailed circuit design of the proposed accelerator. Chapter 6 reports the measurement results with a fabricated prototype IC running example DNNs.

Chapter 2

Neural Network Accelerator with Mixed-Signal Processing Element Array

2.1 Top Architecture

As shown in Fig. 2.1, the proposed IC comprises a 16×16 array of MPEs, array-peripheral circuits on each side, and a current mirror array for distributing the bias currents. On the left side of the MPE array, there are 16 pairs of digital input drivers and digital pulse-width modulators (DPWMs). Each signed 9-bit input is split into an upper signed 4-bit signal and a lower unsigned 5-bit signal, and the digital input driver drives the MSB-lines (MLs) with the upper 4 bits, and the DPWM drives a pair of word-lines (WLs) with two pulses of which timing difference expresses the lower 5 bits.

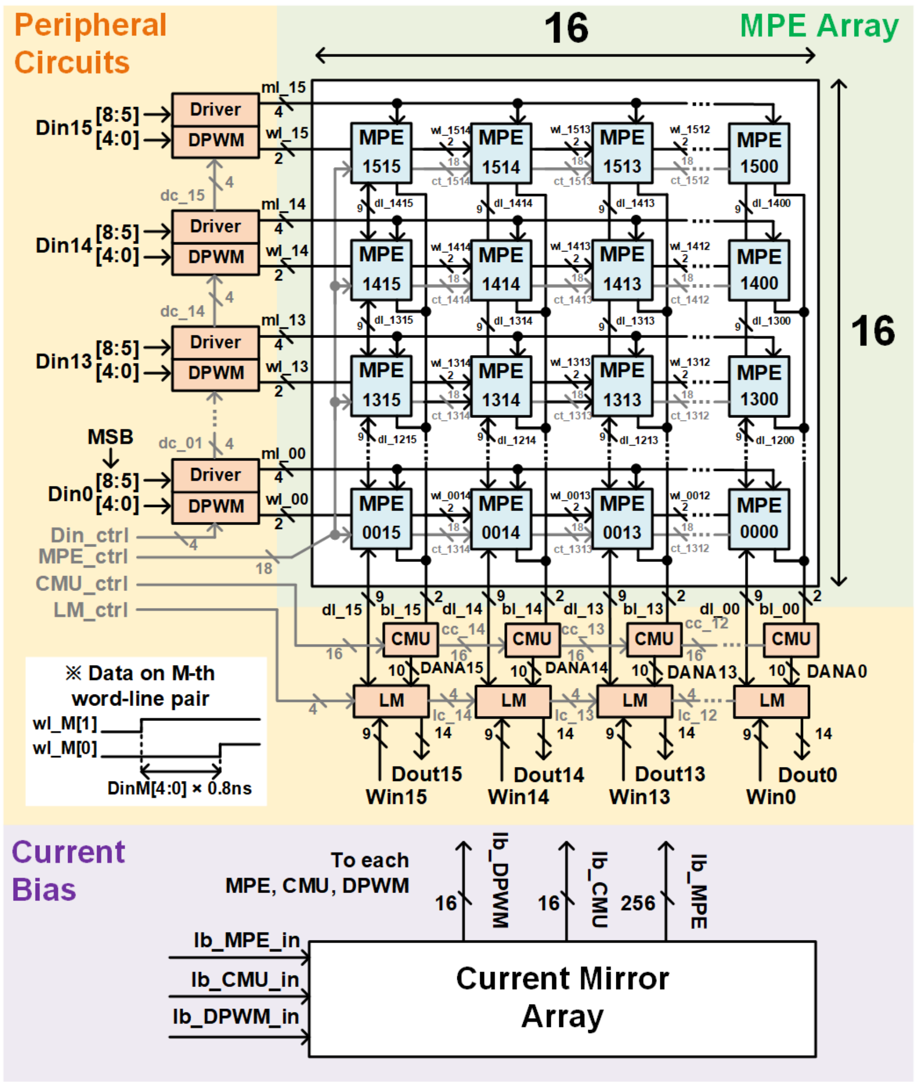


Figure 2.1. The overall architecture of the proposed mixed-signal NN accelerator.

Each MPE in the array performs a mixed-signal MAC operation between an input and a weight bit stored in an 1152-bit MPE SRAM in a bit-serial manner. First, it computes the multiplication between the upper 4-bit input and one of the weight magnitude bits in a digital fashion and propagates the accumulated sum towards the bottom of the array via the data lines (DLs). Second, it steers discharging currents to a pair of capacitively-loaded bit lines (BLs), of which duration is equal to the timing difference of the two WL pulses, and the sign of the weight sets polarity and magnitude and one of its magnitude bits, respectively. The weighted sum of the inputs can be effectively computed through a sequential, bit-serial operation in conjunction with the cyclic MAC unit and logic module located at the bottom.

On the bottom side of the MPE array, there are 16 pairs of cyclic MAC units (CMUs) and logic modules (LMs). Each CMU produces a signed 10-bit output as the weighted sum of the lower 5-bit inputs via the sequential, bit-serial operation. Moreover, each LM combines the results from the CMU and the weighted sum of the upper 4-bit inputs propagated through the DLs and produces the final signed 14-bit output.

The control signals for MPEs, DPWMs, CMUs, and LMs are propagated with daisy chain connections to avoid long interconnects. Similarly, each LM can update the weights stored on the MPEs by propagating the values via the bidirectional DLs. The current mirror array distributes the bias currents to 256 MPEs, 16 DPWMs, and 16 CMUs, all identical copies of the corresponding external bias currents.

2.2 Mixed-Signal MAC Realization on an MPE Array Architecture

Fig. 2.2 illustrates the mixed-signal MAC operation of the MPE in more detail. The MAC operation is carried out in a bit-serial manner over 9 control cycles. During the first 8 cycles, the analog part of the MPE and CMU computes the sum of the lower 5-bit inputs scaled by one magnitude bit of the weight and performs the shift-and-add operation of the resulting signed 3-bit, 4-level values (±3, ±1) each cycle, producing a signed 10-bit result (DANA). During the same cycles, the digital part of the MPE and LM performs a similar shift-and-add operation in the digital domain and produces a signed 16-bit result (DDIG) of the weighted sum of the upper 4-bit inputs. And at the last ninth cycle, the LM produces a signed 17-bit weighted sum (DMAC) by combining DDIG with DANA scaled by 8. The final signed 14-bit output (DOUT) is produced by truncating the lowest 3 bits of DMAC.

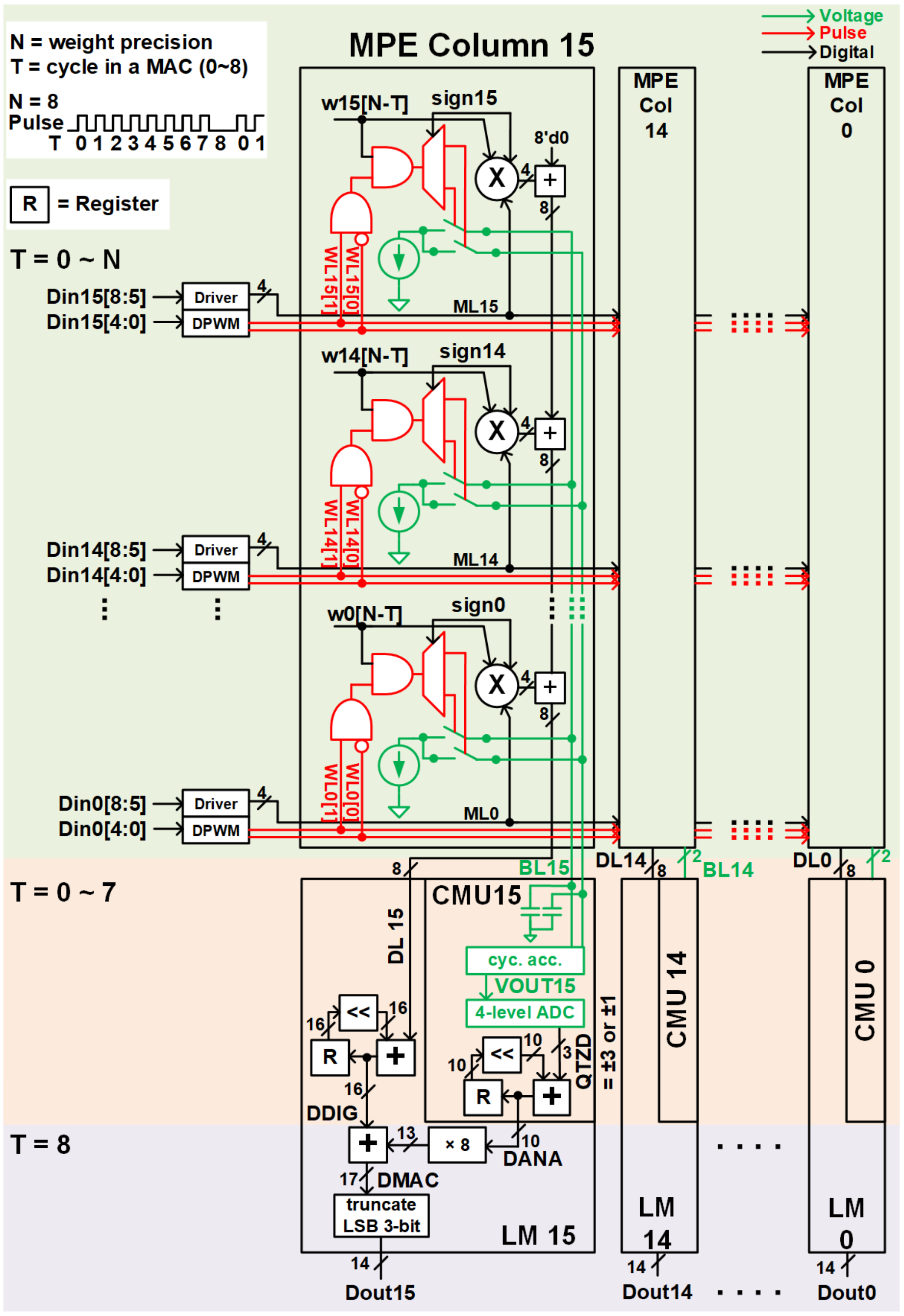
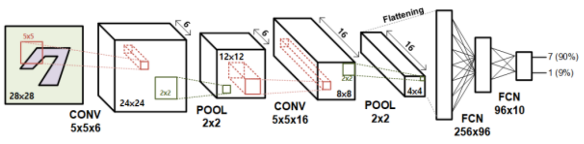
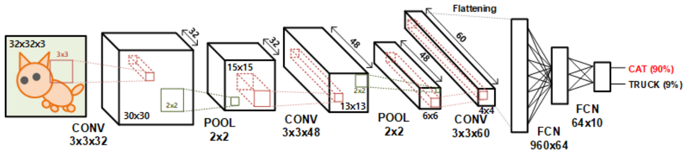


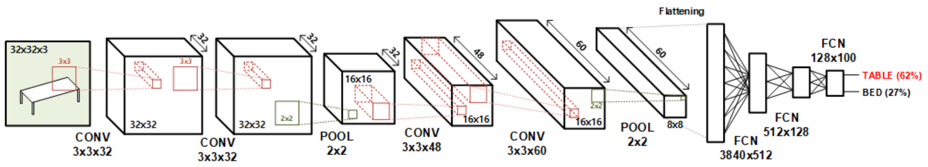
Figure 2.2. The realization of mixed-signal MACs on the proposed architecture.



(a)



(b)



(c)

Figure 2.3. Example (a) four-layer CNN with MNIST dataset, (b) five-layer CNN with CIFAR-10 dataset (c) seven-layer CNN with CIFAR-100 dataset.

MPEs process the upper input bits in digital to preserve the accuracy and process the lower input bits in analog to save the power dissipation. To provide a supporting argument, the median accuracy of running 1000 epochs of the MNIST convolutional neural network (CNN) example with signed 9-bit weights shown in Fig. 2.3(a) is measured while adding Gaussian additive noise to each bit position of the activation, emulating the effects of variation when computing each bit value in the analog domain. Fig. 2.4 plots the resulting degradation in accuracy, indicating that the noises added to the upper bits significantly impact the accuracy than those added to the lower bits. On the other hand, Fig. 2.5 plots the number of transitions that occurred at each bit position when all the computation for a 5-layer CIFAR-10 CNN example in Fig. 2.3(b) is done in the digital domain. The number of transitions is higher for the lower bits, implying that computing the lower bits in the analog domain can save the most energy while minimizing the accuracy's impact. This hybrid approach also relaxes the required precision of the DPWM and reduces the required dynamic range of the bit-line signal.

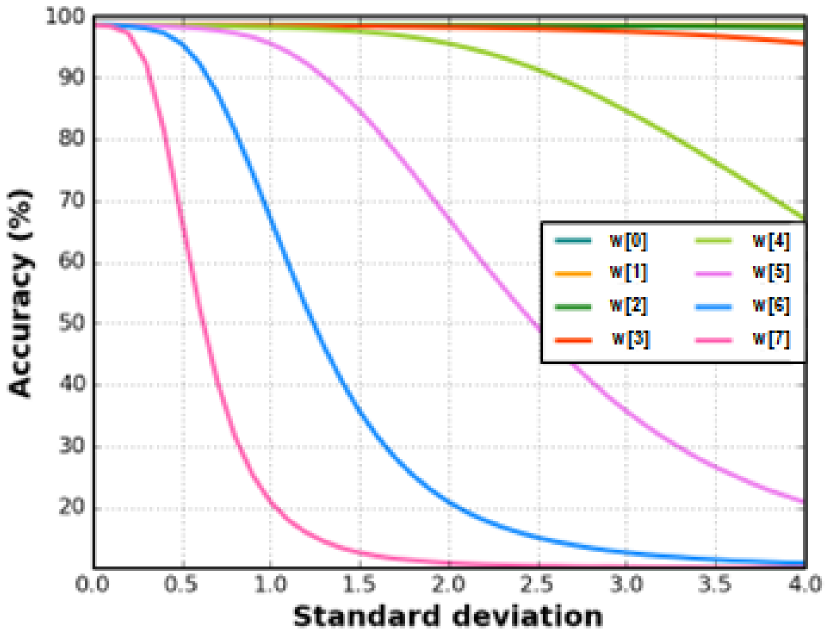


Figure 2.4. Accuracy degradation of an example MNIST CNN with Gaussian random variations added on each bit position of the 8-bit activation result.

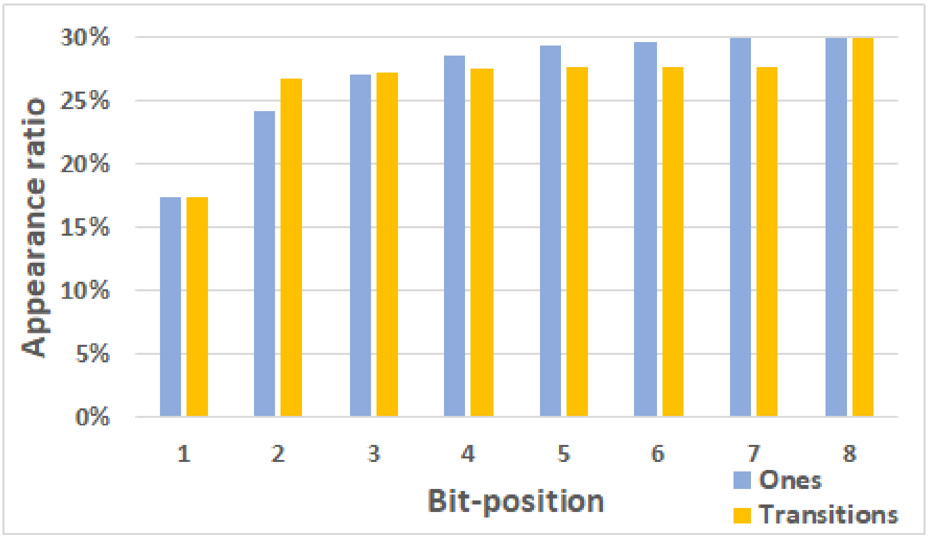


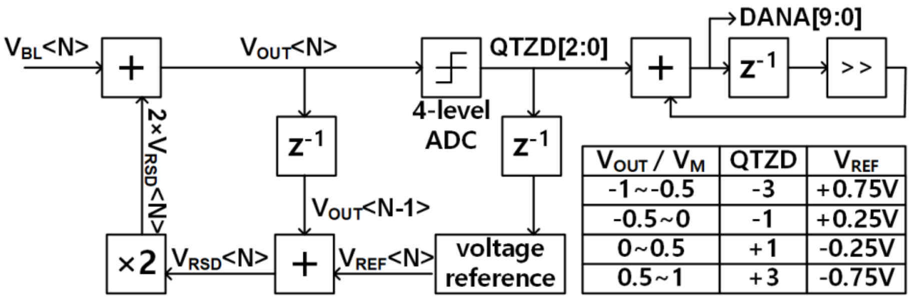
Figure 2.5. The number of appearances of ones and transitions at each bit-position of 8-bit activations on the example CIFAR-10 CNN network.

2.3 Cyclic MAC Scheme with Bit-Serial Operation

Fig. 2.6 illustrates the cyclic MAC unit (CMU) operation using a signal flow model. The CMU computes the weighted sum of the 16 lower 5-bit inputs in the analog domain while producing signed 3-bit outputs each cycle for eight cycles.

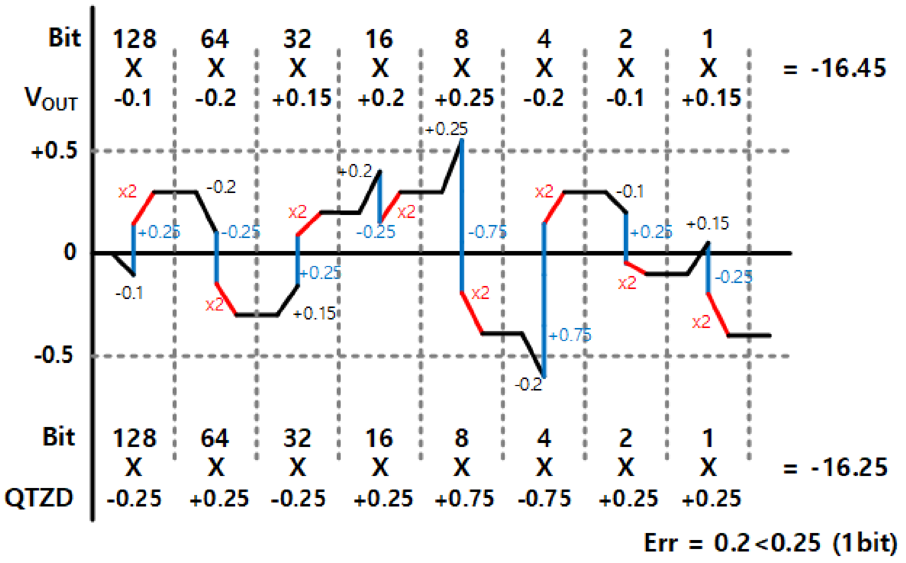
First, each MPE column generates a voltage difference on a bit-line pair (V_{BL}). V_{BL} has a value between $\pm V_M$ when V_M represents maximum output voltage. Next, V_{BL} is transmitted to a 4-level ADC generating signed 3-bit quantized results (QTZD) with thresholds at 0V and $\pm 0.5 \times V_M$. A digital-to-analog converter (DAC) generates a reference voltage (VREF) from QTZD, and a residue (V_{RSD}) is computed. In the next cycle, the residue V_{RSD} is scaled by a factor of 2 and combined with the new V_{BL} value. By repeating this process for eight cycles and shifting and adding the signed 3-bit quantized output of each cycle, the CMU can produce a signed 10-bit result of the weighted sum.

Note that this architecture performs only the minimum operations required to produce the final output bits. The sum of 16 unsigned 5-bit inputs scaled by signed 9-bit weights would produce a 17-bit result. Rather than first computing this total 17-bit result and then truncating its seven least significant bits (LSBs) to produce the final 10-bit result, the CMU computes the weighted sum in a bit-serial manner starting



V_{OUT} / V_M	QTZD	V_{REF}
-1~-0.5	-3	+0.75V
-0.5~0	-1	+0.25V
0~0.5	+1	-0.25V
0.5~1	+3	-0.75V

(a)

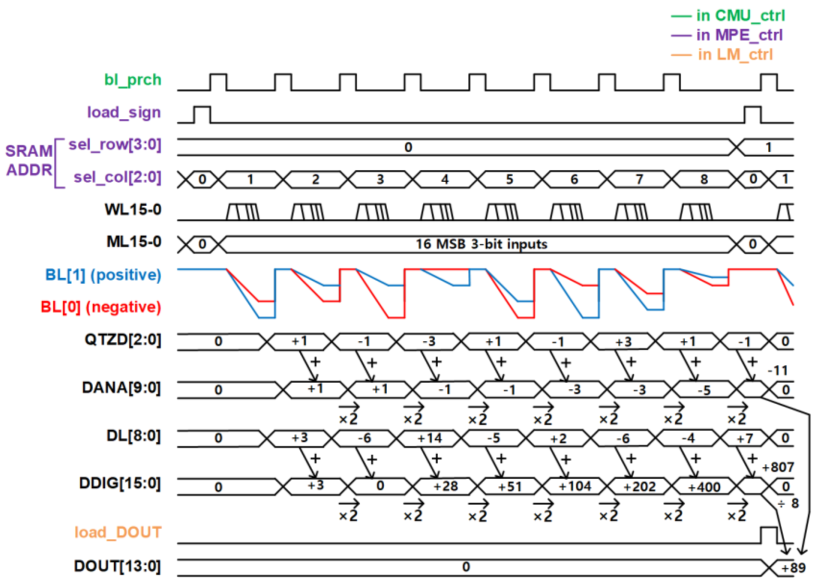


(b)

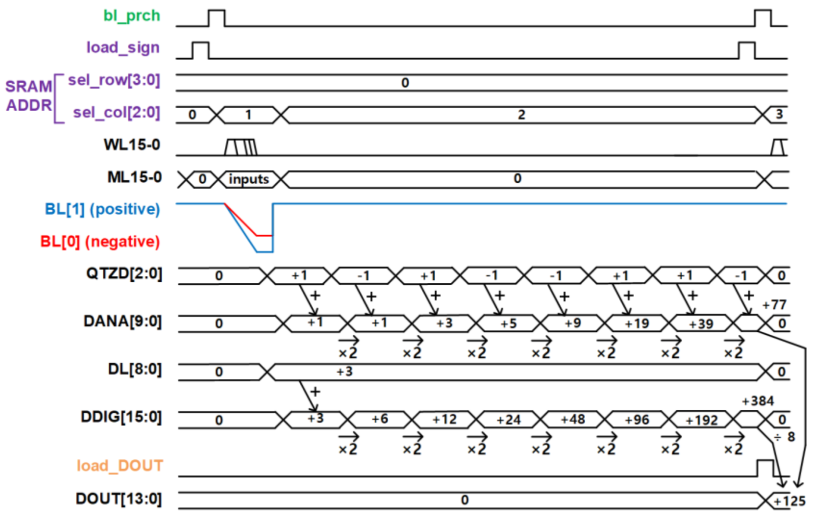
Figure 2.6. (a) A signal flow model illustrating the operation of the CMU (b) An example case of the operation of the CMU.

from the most significant bit (MSB) and stops when the remaining computations do not affect the final 10-bit result obtained so far. In other words, the number of CMU cycles only varies with the number of output bits required, but not with the number of input or weight bits as long as the output precision is lower than the total result with a truncation.

The bit-serial operation of the proposed architecture allows it to support variable weight bit precision. Fig. 2.7 illustrates the operations with a signed 9-bit weight and a signed 2-bit weight. First, the sign bit of the weight is loaded to the sign register. Second, the weighted sum of the inputs is computed for each weight bit. Third, the CMU accumulates the analog weighted sum while scaling the residue from the previous cycle by two and produces the signed 3-bit quantized output. At the same time, an LM accumulates the digital weighted sum. Fourth, regardless of the weight bit precision, the signed 3-bit quantized outputs are collected for eight cycles, and the combined results via shift-and-add yield the final signed 14-bit result.



(a)



(b)

Figure 2.7. Illustration of (a) signed 9-bit × signed 9-bit MAC operation and (b) signed 9-bit × signed 2-bit MAC operation of the proposed bit-serial architecture.

2.4 Realization of DNN layers on an MPE Array

As shown in Fig. 2.8, since 1152 internal weight bits on an MPE share array peripheral circuits with the bit-serial manner, the MPE array can map the example CIFAR-10 CNN with high PE utilization of 78%. Since all MPEs on the array share the same SRAM access address, it can be understood as an MPE array containing 1152 16x16 weight bit matrices. Because the number of weight bits activated at a time is small as 16x16, the MPE array can maintain high PE utilization and require only 16 peripheral circuits for each side to reduce area consumption. With the bit-serial manner, an MPE array can efficiently map a weight matrix of convolution, or a fully-connected layer is larger than a 16×16 array on a separated time domain. As shown in Fig. 2.9(a), the mapping space of each kernel in a convolution layer was expanded from rows [29] to SRAM addresses using multiple cycles. A fully connected layer was mapped by dividing a weight matrix into several 16×16 sub-matrices, achieving a column expansion by adding all results after sub-matrix calculations and a row expansion by separating results at different registers, as shown in Fig. 2.9(b). For operations of a layer larger than a 16×16, a CMU accumulates intermediate results on signed 16-bit registers before ReLU activations.

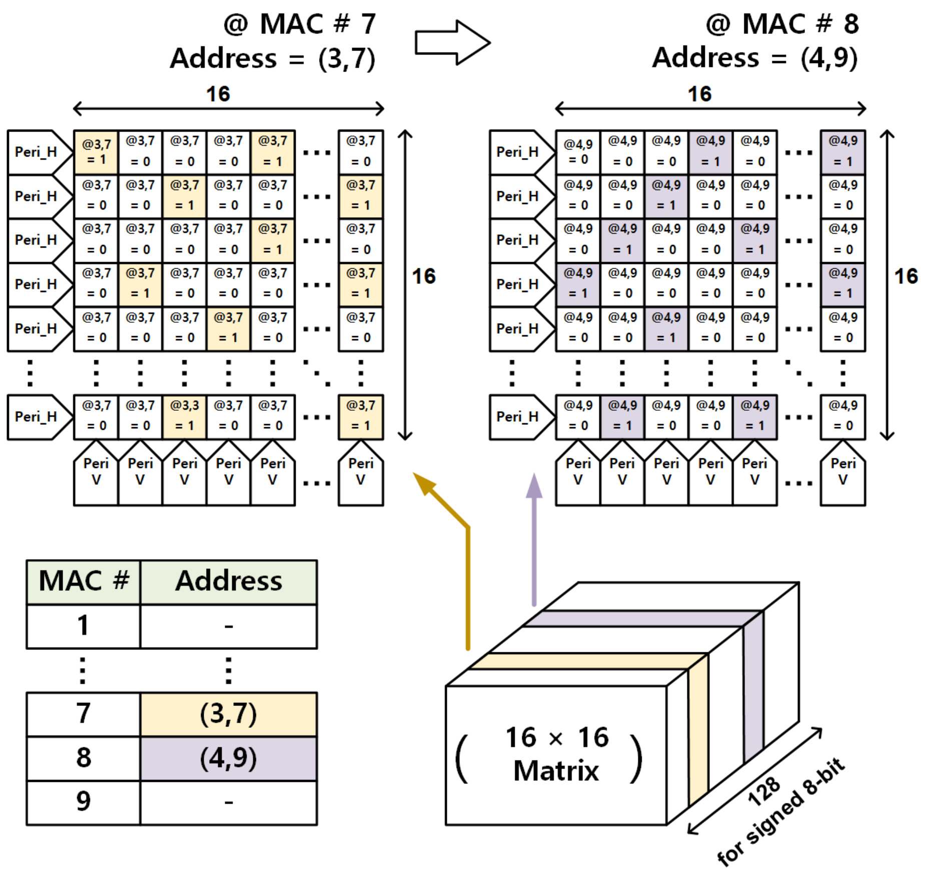
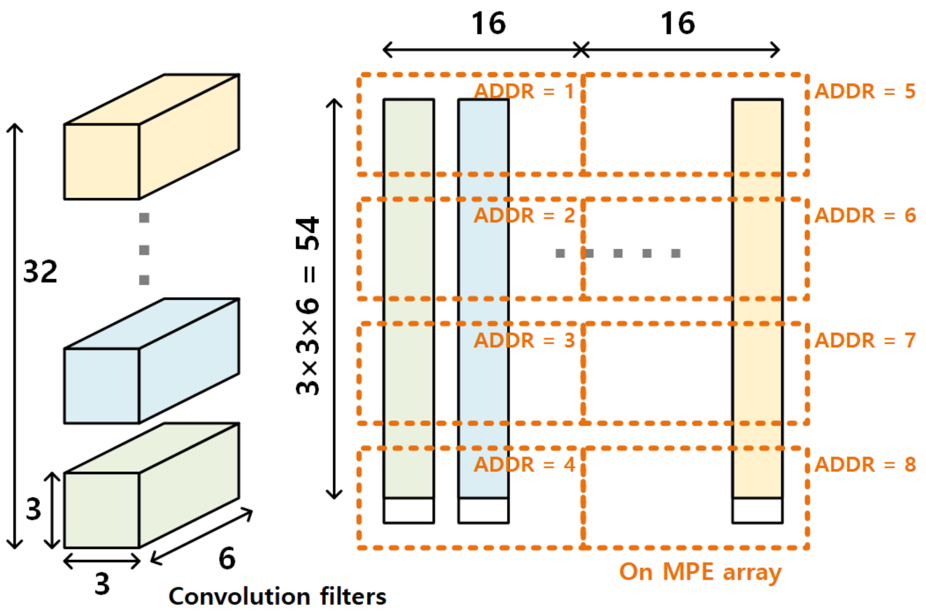
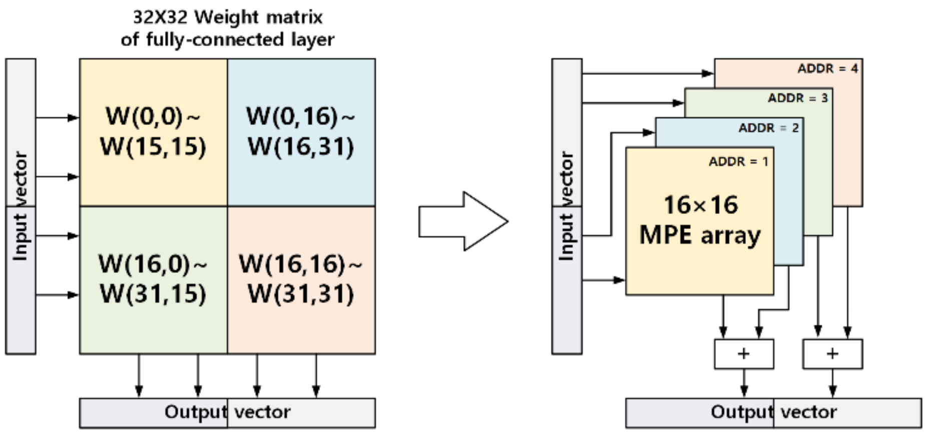


Figure 2.8. Array operation selects a 16 × 16 weight matrix by adding addressing logic to each MPE containing 1152 weight bits.



(a)



(b)

Figure 2.9. Mapping strategies on a 16×16 MPE array for (a) a convolution layer and (b) a fully connected layer.

Chapter 3

Array Calibration Scheme with Gradient Descent Optimization

3.1 Gradient Descent Optimization

Gradient descent is an optimization technique for finding the minimum value of a target cost function with multiple variables [30]; it is widely used as a neural network training method. The optimization process obtains a differential value for the variable to be changed at the current point; it iteratively moves at a velocity proportional to the differential value.

The i_{th} iteration step in the process is where α represents a step size to move per iteration.

$$x_{i+1} = x_i - \alpha \nabla f(x_i) \tag{3.1}$$

If the step size is too large, it is possible to bypass the optimum point; if it is too small, many iterations are required to reach the optimum point. A squared error of a MAC is used as the target cost function for the gradient descent method to achieve neural network learning.

$$\sum_i (y_i - t_i)^2 = \sum_i \left(f\left(\sum_j w_{ij} x_j\right) - t_i\right)^2 \tag{3.2}$$

At this point, the weight changes through the k_{th} iteration as Eq. 3.3.

$$\Delta w_{ij,k} = w_{ij,k+1} - w_{ij,k} = -2\alpha \times (y_i - t_i) \times \nabla f(w_{ij} x_j) \times x_j \tag{3.3}$$

The error can propagate when the product of the weight and input is positive if a ReLU is used for the activation function f(x). A gradient matrix is obtained by multiplying the input vector with the error vector using the sign of the input elementwise as Eq. 3.4.

$$W = X^T E \odot sign(X^T W) \tag{3.4}$$

3.2 Calibration Scheme

The proposed calibration scheme repeatedly learns the parameters of a small 16×16 MPE array to process large layers according to process variations. One fully connected layer with 256 weight parameters is learned using a 16×16 MPE array as an example. The input vector is generated randomly, and the ideal answer is the sum of input vector elements with all output nodes sharing the same value. In this case, the probability distribution of each input element follows a uniform distribution between 0 and 1/16 such that the output value does not exceed one and the gradient value does not increase excessively. The weight bit of an MPE is fixed to a maximum value of 1. Furthermore, the activation function is not used, and the sign term matrix in Eq. 3.4 disappears because the primary purpose is to check only whether the results of multiplication and addition are accurate.

Fig.3.1 shows the proposed technique's implementation on an MPE array. First, an input vector is generated using a PRBS pattern; Since only 1/16 of the maximum value of signed 9-bit activation is required to represent the number of bits of PRBS, unsigned 4-bit is sufficient for the PRBS generator. A DPWM supplies the generated digital vector input to the MPE array in the pulse width proportional to the input. In addition, a CMU and an LM quantize generates a signed 9-bit digital data, and an error vector is generated compared to the sum of the elements of the supplied input vector. The last step is multiplying the

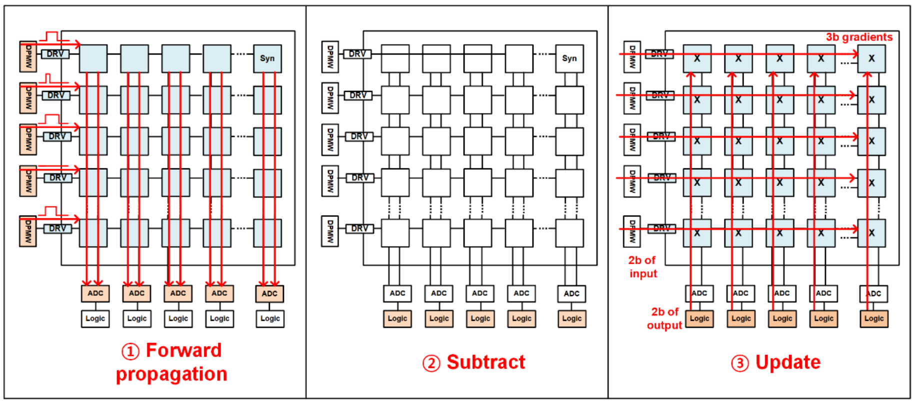


Figure 3.1. The steps of implementing the proposed calibration technique.

input vector by the error vector to calculate the gradient matrix and add it to the input parameter of the programmable current source. The stored calibration parameter in an MPE is unsigned 8-bit digital data, and it uses 8'b1000_0000, the median value of digital values expressed in eight bits, as the initial value. A 5-bit programmable current source uses the upper 5-bit of a calibration parameter as an input.

The generated gradient matrix has the same connection with the MPE array between the inputs, outputs, and gradient elements; therefore, the gradient matrix can be calculated internally on the MPE array instead of using an external computation space. For realizing an internal calibration scheme, additional modules are required inside an MPE circuit, such as a multiplier for an input and error value and an adder for adjusting the parameter of the current source. The input and error data were clipped during the gradient operation to a maximum of 2'b11 using two bits to adopt this internal calibration scheme with a tiny hardware addition. Furthermore, the generated gradient value was limited to a maximum of 3'b111, and it could be expressed in three bits.

3.3 Simulation Results

We modeled an example neural network and the effect of process variation using TensorFlow, a machine learning framework. The neural network was modeled first and changed to reflect the same effect of variation on parts calculated by the same weight because TensorFlow is not a framework for describing hardware. Process variation is modeled as two 16×16 arrays representing the scale and offset terms. This variation array pair was randomly generated through a Gaussian distribution with an average of 0 and a standard deviation of 0.5.

We implemented a set of sub-computations practical to only 16×16 units of weight left using TensorFlow built-in functions to include this variation model in a neural layer operation. After calculating the original layer for each sub-weight array, in the same manner, the outputs were summed up, and a ReLU activation function was applied to create the final result. The process variation was implemented by multiplying the scale term and adding the offset term for each sub-weight-array operation.

The accuracy improvement proceeds separately from the inference process and goes through learning 16 × 16 variation terms with a random input generated by a uniform distribution. A process comprising forward propagation, multiplication, and addition was performed without using the TensorFlow built-in optimizer function to implement the synapse circuit clipping of input, weight, and gradient.

We generated a set of 1000 different random variation arrays and validated how classification accuracy changed when the proposed techniques were applied. A four-layer convolution neural network for handwritten digit classification based on MNIST data shown in Fig. 2.3.(a) was used. The ideal accuracy when the effect of process variation is not applied to this network is 98.2%. Fig. 3.2 shows the test results, including the process variation model. The red histogram shows that the accuracy of the network decreases to 17.3~95.2% due to process variations. Fig. 3.2(b) shows that the accuracy is almost fully restored to the ideal level, ranging from 96.5% to 98.3%, after 500 epochs of learning with the proposed technique. The required number of iterations to recover the accuracy to higher than 97% was measured for each test; 150 and 250 times were sufficient for 92.7% and all tests, respectively.

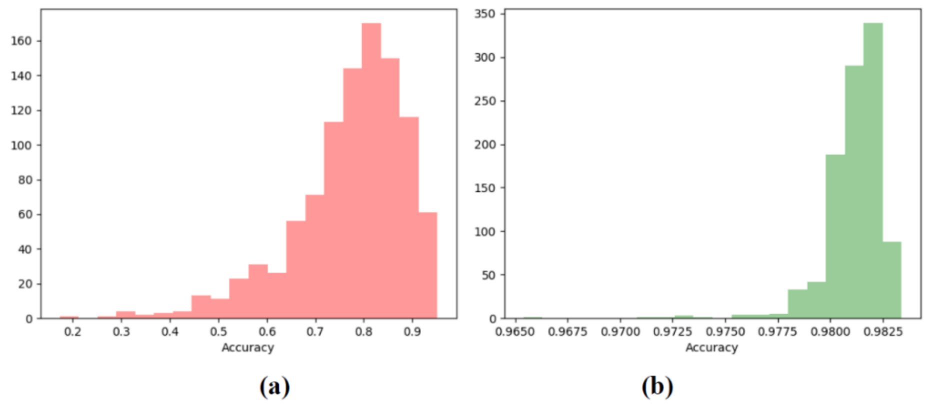


Figure 3.2. The accuracy of the example DNN with variation model (a) without and (b) with the proposed calibration technique.

We conducted the following tests to confirm that this technique was valid under several conditions. It was swept from three to eight bits to check whether the number of bits used in the input code of the current source was appropriate; then, as shown in Fig. 3.3, the technique could not fully restore accuracy using a bit width lower than 5, which indicates that finding the required minimum number of bits is essential. The required number of bits may vary depending on the expected scale or offset error distribution confirmed during the design process. A total of 25 cases were tested by changing the standard deviation of Gaussian distributions to view the scale distribution and offset error effect. Table 3.1 summarizes the average drop and improvement of accuracy for each test. For each cell, the value on the left indicates the accuracy before applying the technique; the value on the right represents the accuracy after applying the technique. Accuracy is more affected by the scale error, and if the standard deviation is less than 0.5, it can be recovered sufficiently using this technique.

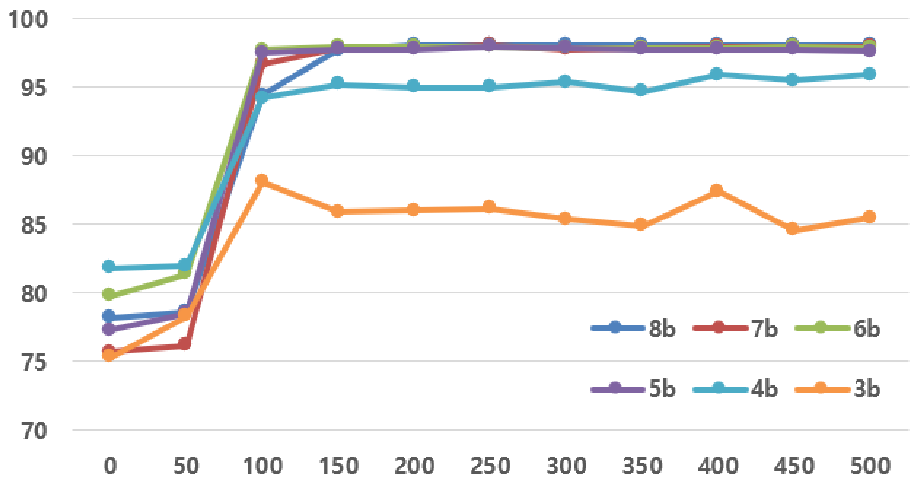


Figure 3.3. Average accuracy improvement by sweeping the number of gradient bits.

Table 3.1. The average accuracy of the example DNN with and without the proposed calibration technique for different variation conditions.

offset \ scale	**σ = 0**	**σ = 0.25**	**σ = 0.5**	**σ = 0.75**	**σ = 1**
σ = 0	98.2% / 98.2%	96.1% / 98.2%	82.3% / 98.1%	47.9% / 96.6%	25.0% / 89.1%
σ = 0.25	98.2% / 98.2%	96.0% / 98.2%	75.3% / 98.1%	44.0% / 97.0%	27.7% / 84.8%
σ = 0.5	98.2% / 98.2%	96.3% / 98.2%	78.1% / 98.1%	46.7% / 96.9%	26.6% / 84.3%
σ = 0.75	98.2% / 98.2%	95.9% / 98.2%	77.8% / 98.1%	46.1% / 96.9%	27.9% / 86.6%
σ = 1	98.2% / 98.2%	94.9% / 98.2%	71.5% / 98.1%	45.3% / 96.8%	27.9% / 87.3%

Chapter 4

Implementation of A Computation Core and Test Logics

4.1 Mixed-Signal Processing Element

Fig. 4.1 shows the schematic of an MPE having functions of storing weight, computing a MAC on the mixed-signal domain, and calibrating analog MAC current. First, an MPE uses a 6T SRAM block for storing weight. An SRAM block can store a total of 1152 bits as 9 bits for each 7-bit address. A 9-bit mux selects a bit of weight for a MAC operation within the 9-bit output of an SRAM. Second, for digital MAC computation, an MPE chains an adder with adders of neighboring MPEs using DLs. An adder adds a signed 4-bit digital input on ML to signed 7-bit data from the upper MPE's DL[7:0]. And the output of an adder is propagated to the downside MPE. Third, for analog MAC computation, a WL logic decides whether to transmit or not the timing

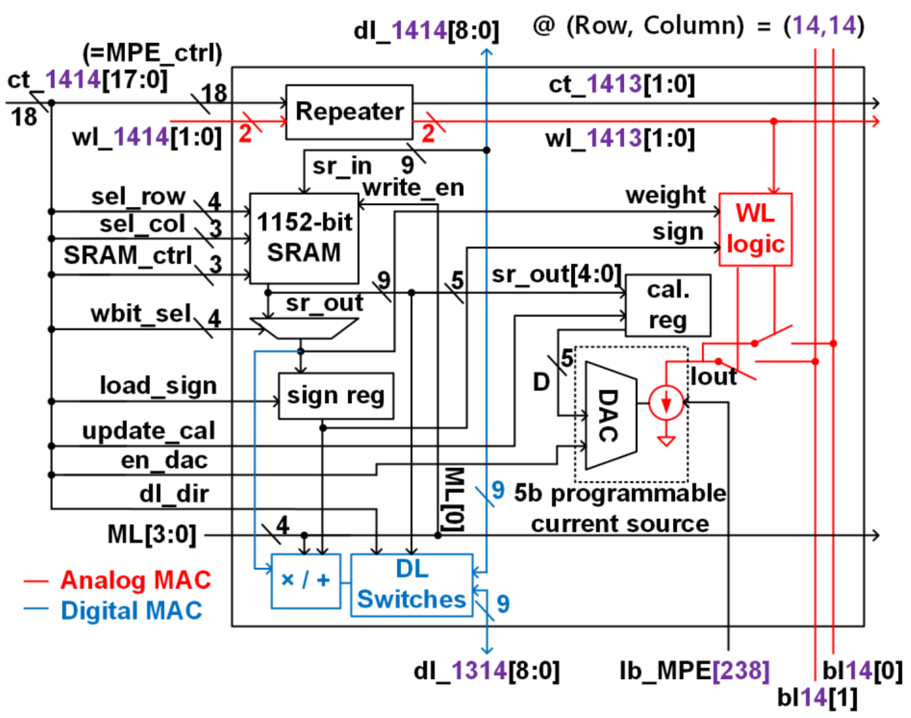


Figure 4.1. Schematic of the MPE.

difference of WLs according to the weight bit currently selected. And a WL logic decides the direction of transmitting with a sign bit stored in a sign-bit register. Moreover, the analog computation uses a 5-bit programmable current source to control the amount of output current with a calibration term.

Fig. 4.2 shows the port information of an MPE array and the detailed connection between the MPEs when constructing the array. To prevent a pulse from shrinking due to the propagation delay difference between a rising and a falling edge during propagation, a DPWM generates only rising edges and sends them separately to two WLs. An MPE has repeaters for pulses and control signals to drive the large parasitic capacitance of WLs and control signal paths. An MPE has a bit-line pair for each column, and each bit-line pair consists of a positive and a negative line.

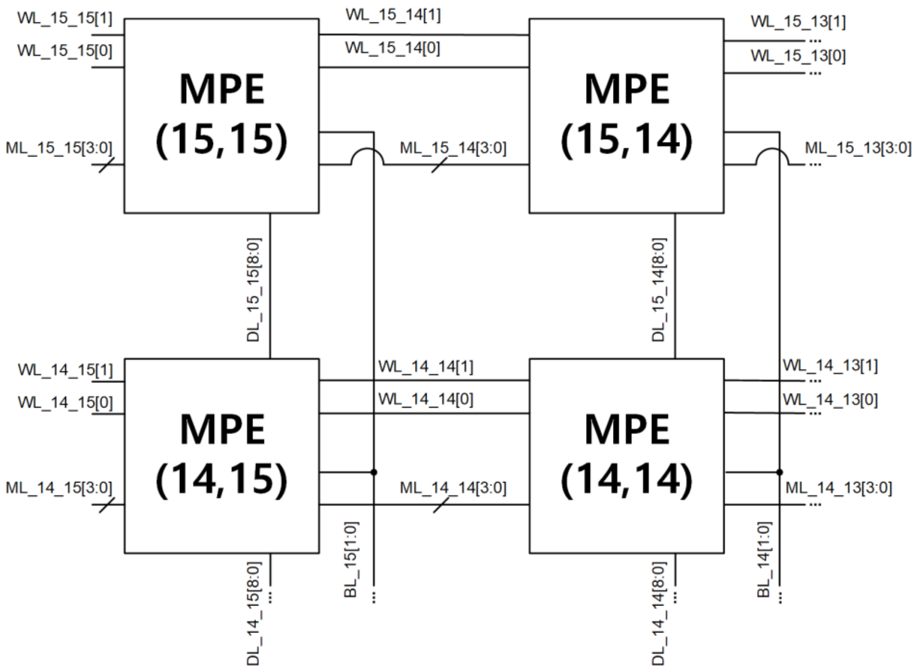
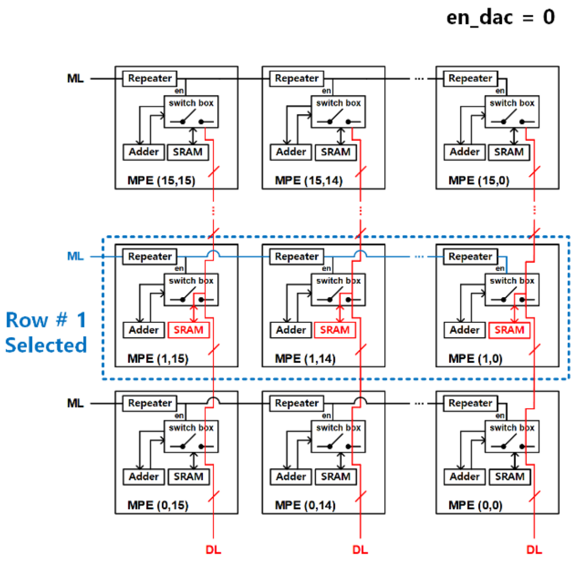
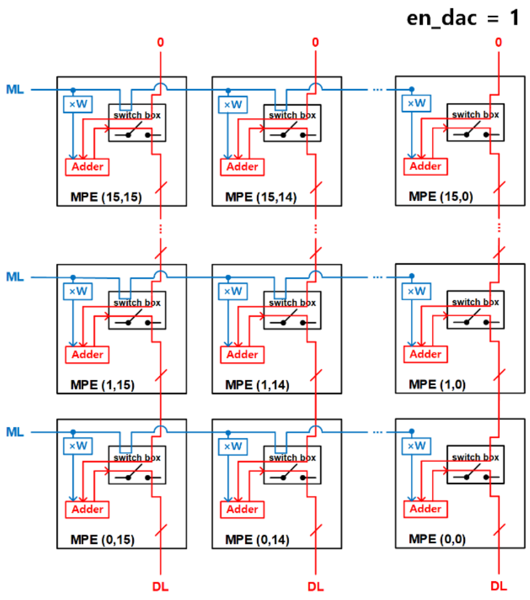


Figure 4.2. The detailed port connection on an MPE array.



(a)



(b)

Figure 4.3. The detailed port connections of DLs and MLs for accessing MPE SRAMs and operating digital MACs.

An MPE array loads and saves weights by accessing MPE SRAMs and operates digital MACs using MLs and DLs. As shown in Fig. 4.3(a), a MPE row is selected using an ML to handle weights. All DLs in the target row configures a 9b-bidirectional bus connected and transmits 9-bit weight data. For realizing digital MACs, MPEs serially connect DLs to form an adder chain, such as Fig. 4.3(b).

The ability to calibrate the uncertainty of a 5-bit programmable current source increases the accuracy of analog operations. Since the 5-bit programmable current source can adjust the output current range from 300nA to 900nA, an MPE can reduce the errors from the effect of process variation on a multiplication below 19.35nA by a calibration process. Fig. 4.4 shows the accuracy change of the MNIST CNN example according to the standard deviation for generating random Gaussian distribution for the variation model on output currents. The NN accuracy drop is 1%p with a standard deviation of 0.053×600nA, 1.64 times larger than the MPE's calibration unit step. Additionally, with the contents of Chapter 3, an easy calibration method to recover network accuracy by regarding output currents of MPEs as trainable parameters of a gradient descent training algorithm was used. Circuit design complexity was reduced by inputting only MSB 5-bit of an 8-bit parameter stored in a calibration term register into a programmable current source.

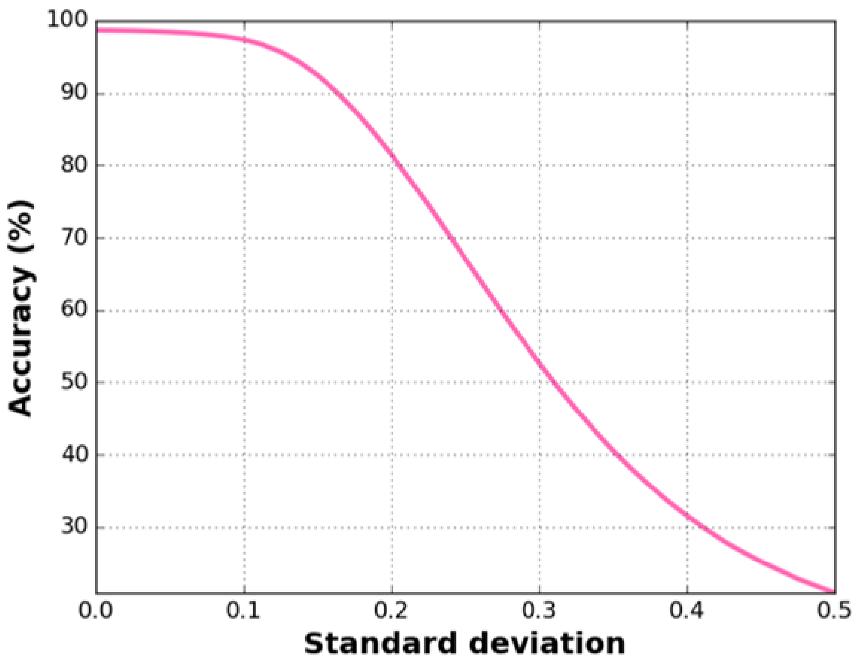
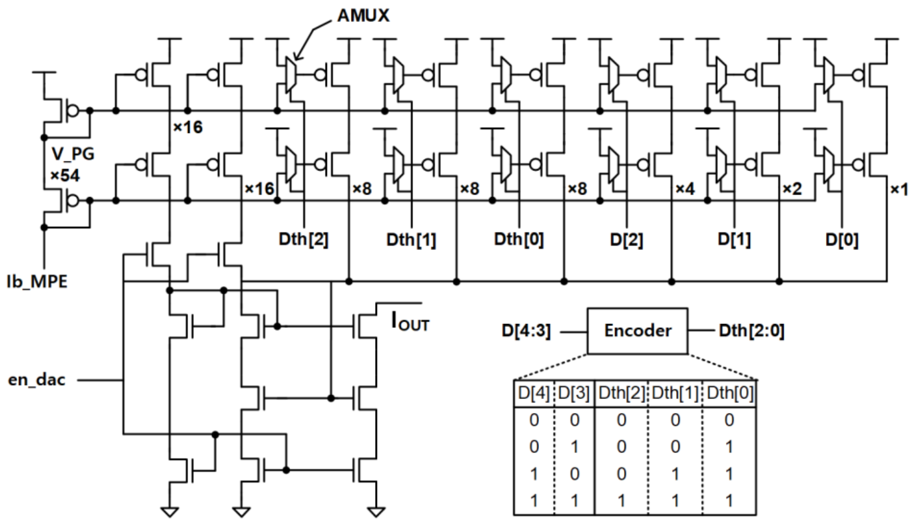


Figure 4.4. Example 4-layer MNIST CNN accuracy with a standard Gaussian distribution standard deviation for randomly generated output current variation.

As shown in Fig. 4.5, a programmable current source uses a current digital-to-analog converter (DAC) structure and generates a current output by mirroring the current supplied from current bias circuits proportional to the digital code. The MSB 2-bit uses unary coding to reduce the non-linearity, while the LSB 3-bit uses binary coding to reduce area consumption. The output current range is from 300nA to 900nA; a DAC constantly turns on offset 300nA and divides the remaining 600nA by 31 levels. When a DAC enable signal (en_dac) is off, analog switches (AMUXs) charge the input node with the supply voltage, which is closer to PMOS gate voltages (V_PG, Ib_MPE) at run-time than ground, for fast responses. The current mirror of the output stage was implemented in a cascode form to reduce the output current change due to the change of bit-line voltage.



D[4]	D[3]	Dth[2]	Dth[1]	Dth[0]
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

Figure 4.5. Schematic of the 5-bit programmable current source.

4.2 Digital Pulse-Width Modulator

Fig. 4.6 shows a detailed schematic of a DPWM, including a pair of integrate and fire (I&F) cells and steps for creating a pulse signal whose width is proportional to the digital code. Like [32], a pair of I&F units marking firing timing information proportional to a digital code transmitted to an MPE row. A DPWM ctrl logic converts a 5-bit digital code to a 31-bit thermometer code controlling switches connected to each unit capacitor to guarantee monotonicity. Instead of using a single large crossing detector, DPWM uses a miniature crossing detector for each unit capacitor to reduce errors caused by the parasitic capacitance

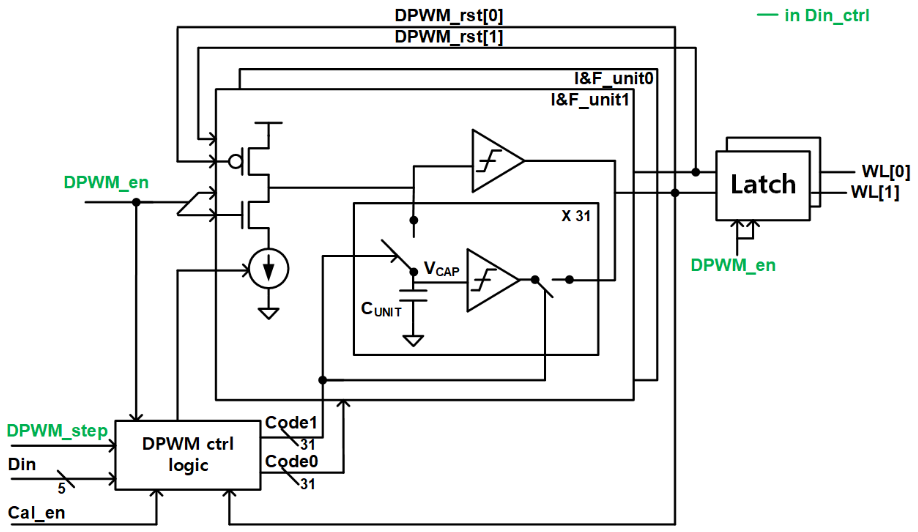
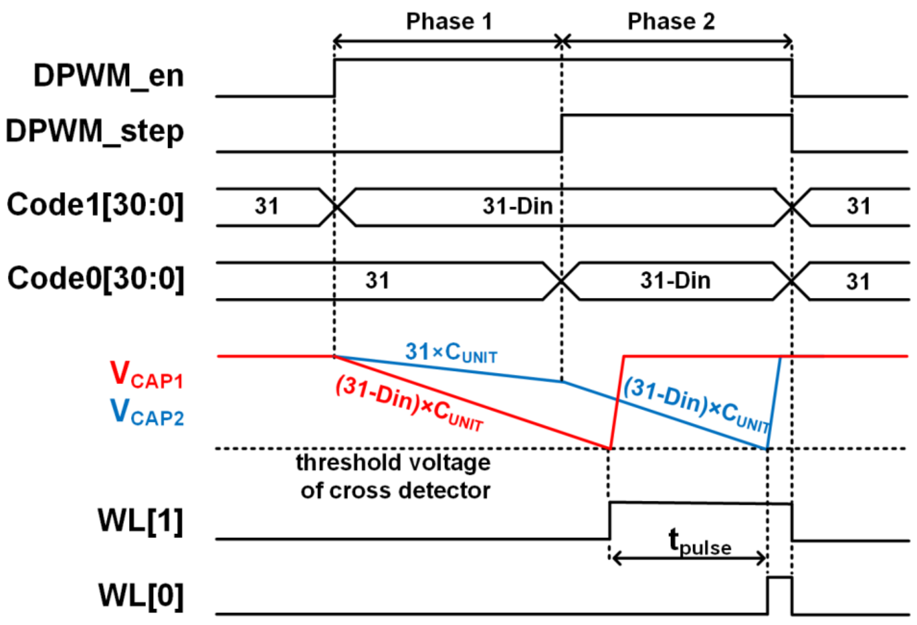


Figure 4.6. Schematic of a DPWM.

of the crossing detector when the number of selected capacitors is small. In addition, to prevent an offset term with parasitic capacitance, for converting the digital value of an LSB 5-bit input equal to N, one I&F unit connects 32-N capacitors, and another I&F unit connects all 32 capacitors in phase 1, 32-N in phase 2, as shown in Fig. 4.7.



$$t_{pulse} = t_{MAX} - \frac{((31-N)*C_{UNIT}+C_{OFFSET})*t_{MAX}}{31C_{UNIT}+C_{OFFSET}} = \frac{N*C_{UNIT}*t_{MAX}}{31C_{UNIT}+C_{OFFSET}}$$

Figure 4.7. Operation of a DPWM.

The HSPICE simulation confirms that the ratio between capacitance and current of an I&F cell can vary from 70% to 140% of the typical corner case by process variations on the designed DPWM circuit. Due to the wide range of variations, DPWMs with a large capacitance can not fire within two cycles, which may cause non-linearity of input data. To prevent the non-firing situation, a calibration function that adjusts the magnitude of the current by checking whether firing occurs for the maximum capacitance case was implemented in the DPWM logic.

4.3 Cyclic MAC Unit

As shown in Fig. 4.8, a CMU has six main components: voltage reference, 2-bit ADC, cyclic accumulator, capacitor bank, bit-line pre-charger, and accumulation logic. The capacitor bank calibrates bit-line capacitance to an appropriate value for supply voltage and maximum pulse-width with 63 adjustable levels in 5.5fF units. A cyclic accumulator receives two bit-line voltages as inputs and performs the reference subtraction and residue doubling processes required for the cyclic MAC scheme with generating differential output voltages. A 4-level flash ADC quantizes output voltages from a cyclic accumulator and generates a signed 3-bit output whose value is one of ±3 and ±1. A voltage reference is a voltage divider composed of nine 1kΩ resistors providing a reference voltage to the cyclic accumulator and the 4-level ADC. An accumulator logic generates signed 10-bit output by shifting and adding signed 3-bit ADC output. The 16-bit control signal for a CMU is separated into control signals for capacitor bank (cb_ctrl), cyclic accumulator (swcap_ctrl), and bit-line pre-charger (bl_prch).

The quantization level of a CMU is related to the dynamic range of an analog input and output voltage of a CMU. As shown in Fig. 4.9, when the V_{OUT} range is restricted to $\pm V_M$, each quantization level has a size of $2V_M/(N+1)$ with N comparators. Using midpoints of each quantization level as reference voltages, the residue after subtraction has a value between $\pm V_M/(N+1)$. Since the sum of input and doubled residue

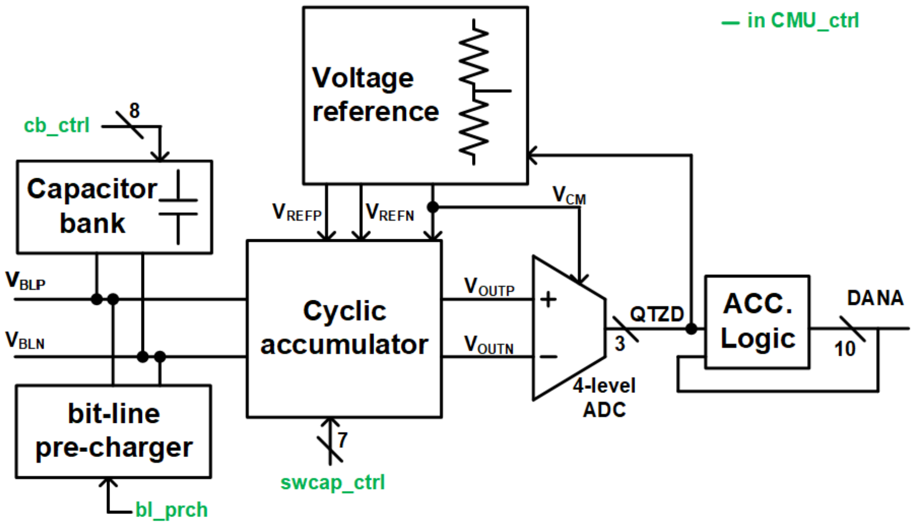
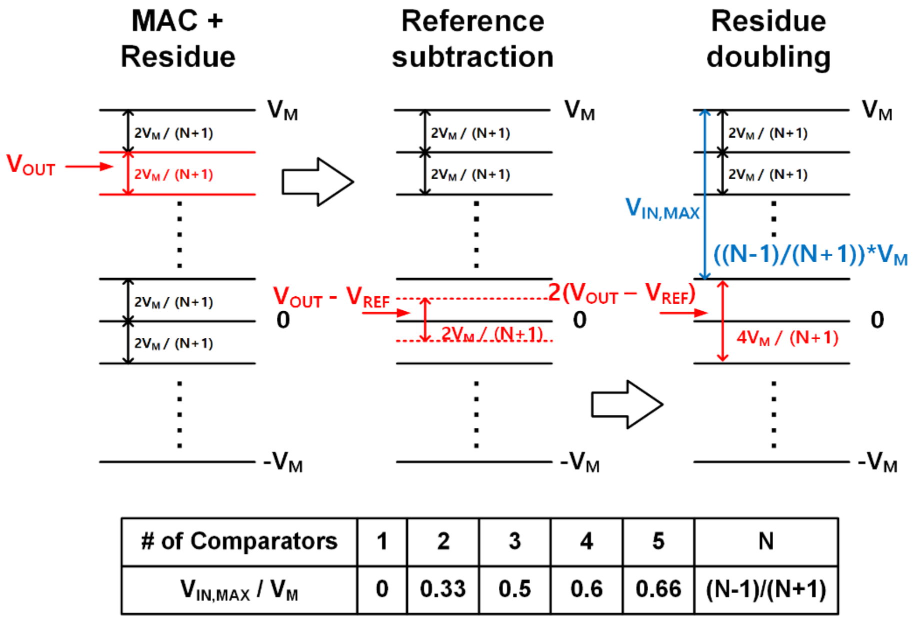


Figure 4.8. Schematic of a CMU.



# of Comparators	1	2	3	4	5	N
$V_{IN,MAX} / V_M$	0	0.33	0.5	0.6	0.66	(N-1)/(N+1)

Figure 4.9. The number of comparators and dynamic range of input voltage.

transferred to the V_{OUT} node cannot exceed V_M, the dynamic range of the input is limited to $V_M \times (N-1)/(N+1)$. The dynamic range of input increases, but quantizer bits increase together, and three comparators were selected for a 4-level quantization as the optimal point. When N is 1, it is the same as a cyclic ADC [33] with zero input range without a MAC for the next weight bit.

Fig. 4.10 shows the schematic of the cyclic accumulator with detailed connections at each operation phase. The cyclic accumulator consists of a differential op-amp, three pairs of capacitors, and switches. There is one more capacitor pair than a conventional differential switched capacitor structure because a CMU receives the computation result of the following weight bit for every cycle during multi-bit MAC operation. As shown in Fig. 4.11, the cyclic accumulator has five operation phases, runs phases 1-2 for the first cycle, and repeats phases 3-5 for the successive seven cycles. The swcap_ctrl signal is separated into seven flags as Φs, Φa, Φr, and $\Phi 1 \sim 4$. In phase 1, turn on Φs and $\Phi 1$ to sample bit-line voltages to C1. In phase 2, turn on Φa and $\Phi 2$ to move charges stored in C1 to C3. In phase 3, turn on Φr to reset C1 and C2. In phase 4, turn on $\Phi 1$, transfer the charge stored in C3 to C1, doubling the residue voltage, and simultaneously turn on $\Phi 3$ to sample a bit-line voltage to C2. In phase 5, turn on $\Phi 2$, transfer the charge stored in C1 and C2 to C3, and turn on $\Phi 4$ to subtract a reference voltage.

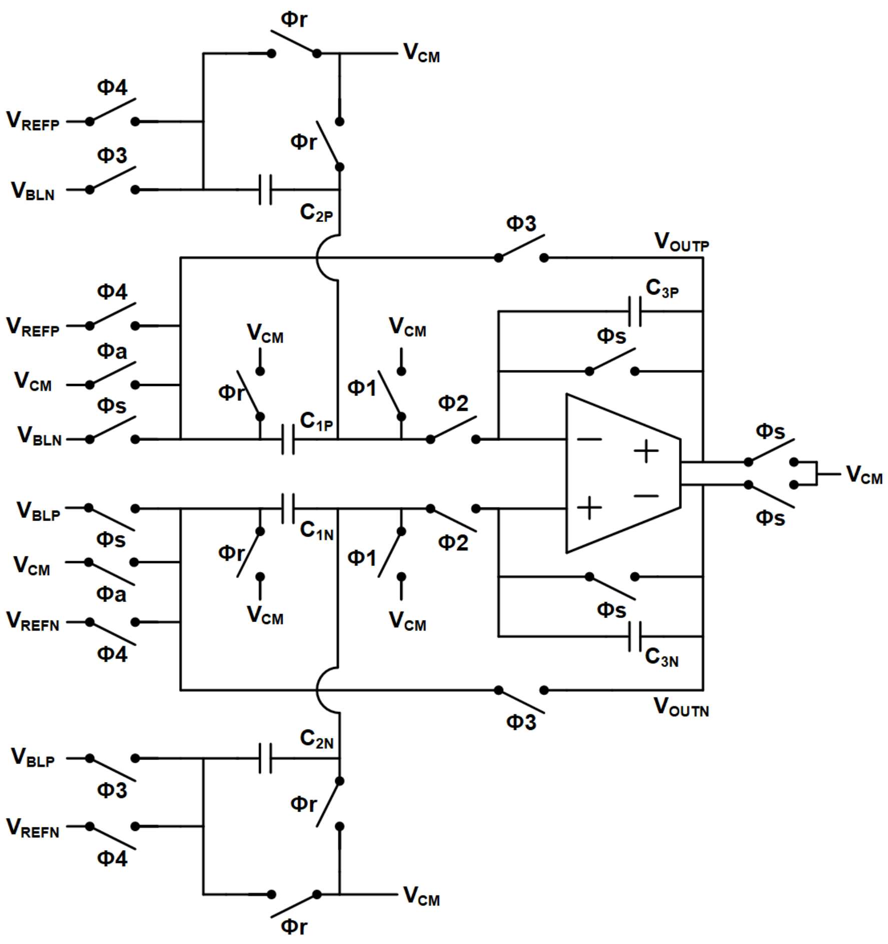


Figure 4.10. Schematic of a cyclic accumulator.

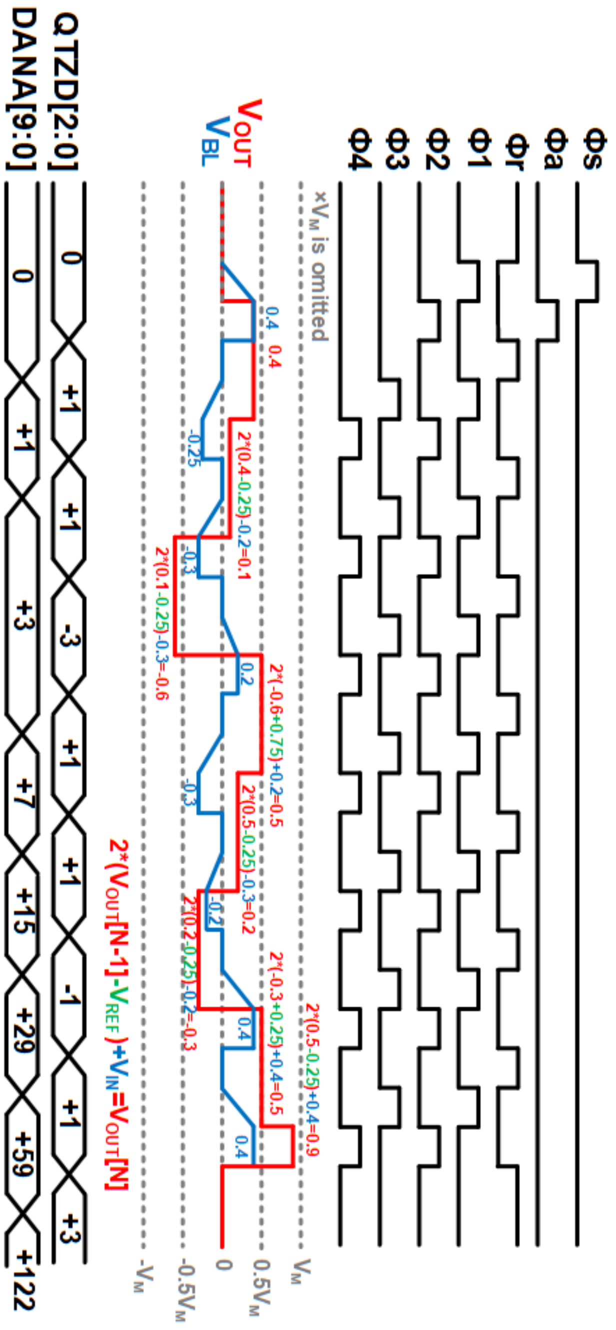


Figure 4.11. The waveform of a cyclic accumulator in a MAC with a signed 9-bit weight.

4.4 Test logics

Since MPEs, CMUs, and DPWMs of the computation core require many control signals and data ports, a test logic is required to generate control signals in a die due to insufficient IC I/O pads. As shown in Fig. 4.12, the test logic consists of instruction SRAM and finite state machine. The test logic sequentially generates a control signal with an input instruction code. There are two operating phases for test logic: First, the programming phase with instruction programming enabling signal (prg_en) turned on. Second instruction running phase, with prg_en turned off. As shown in Fig. 4.13, The test logic can load instructions at both phases. In the programming phase, the test logic

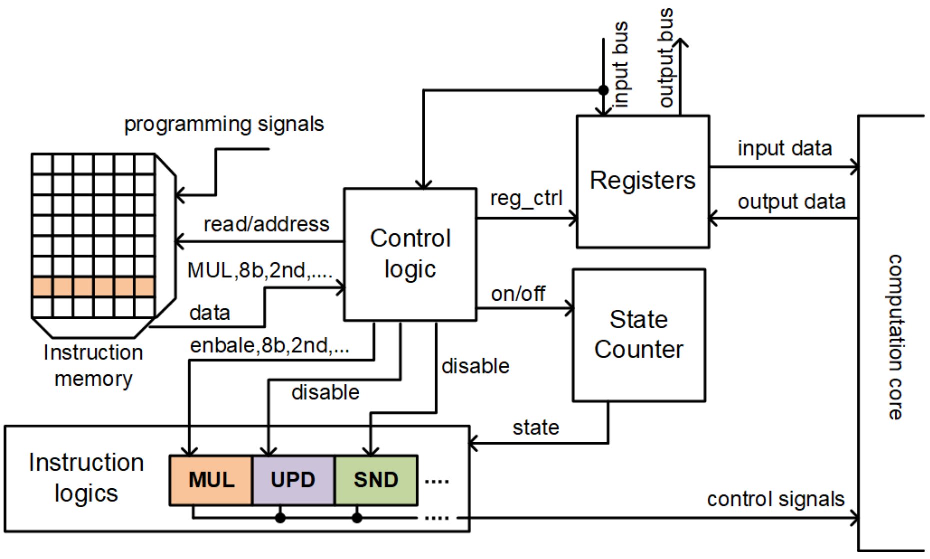


Figure 4.12. The structure of a test logic for creating control signals for a computations core with instruction codes.

loads instruction codes from an instruction SRAM storing instruction codes provided through programming signal ports. During the instruction running phase, the test logic receives instruction codes through input bus ports in the instruction running phase. After loading an instruction code, a state counter determines the current state for each clock cycle, and a combinational instruction logic activated by the current instruction code generates control signals for each state and

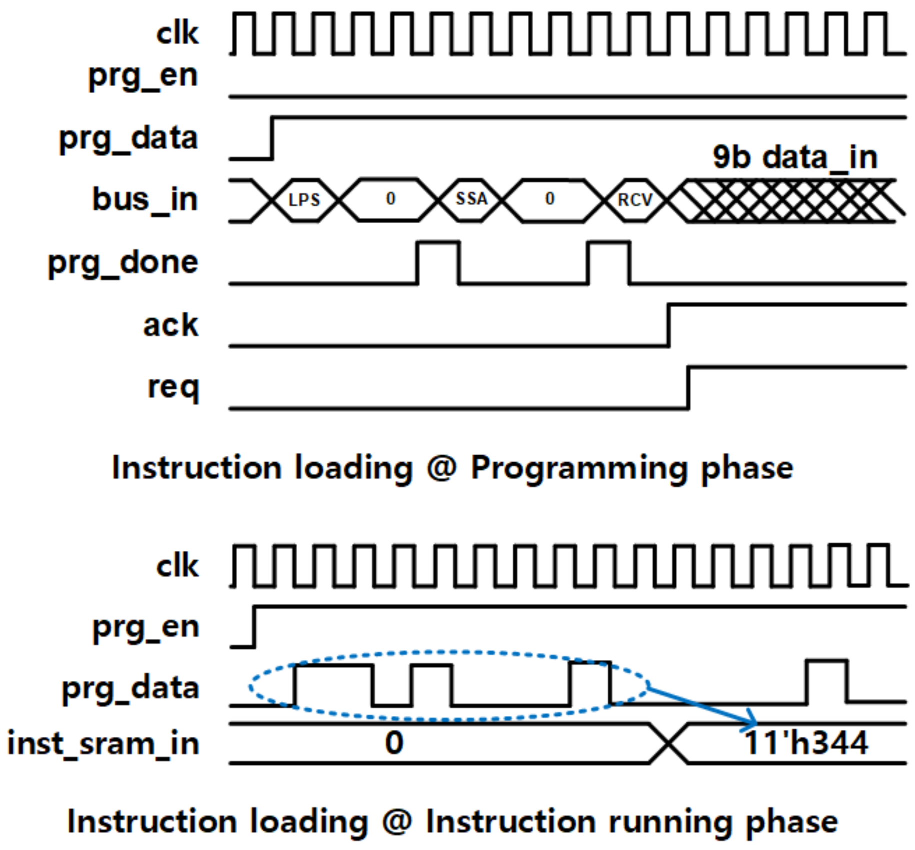


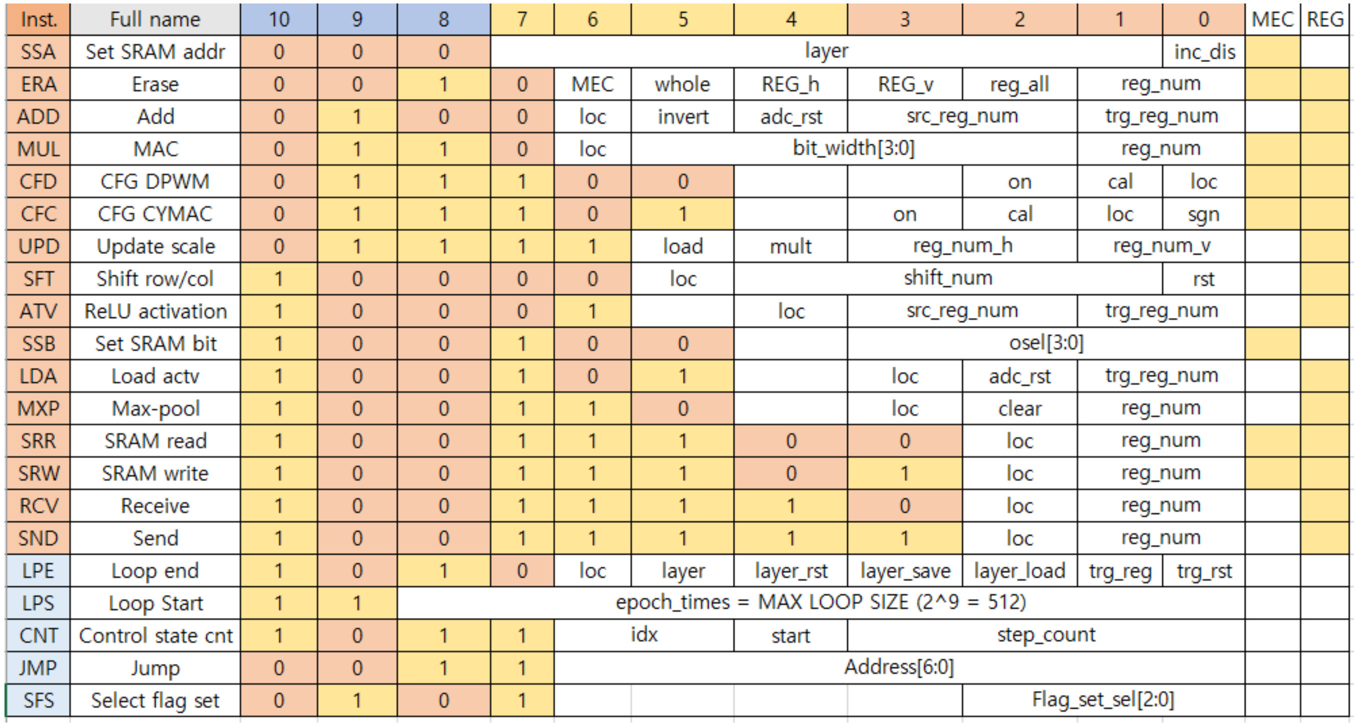
Figure 4.13. An example of instruction loading processes at programming and an instruction running phase.

transmits control signals to the computation core at the instruction running phase. Moreover, the test logic includes registers for handling four 9-bit data of each row and column of the computation core.

Fig.4.14 lists 21 instructions provided by the test logic of the prototype IC. The detailed types of instruction codes are as follows: first, nine configuration instructions (SSA, ERA, CFD, CFC, UPD, SFT, SSB, SRR, and SRW). Second, two communication instructions (RCV, SND). Third, five DNN instructions (MUL, ADD, ATV, LDA, MXP). Fourth, two for-loop creation instructions (LPS, LPE). Finally, three debugging instructions (CNT, JMP, SFS).

The functions of the configuration instructions are as follows. SSA configures an MPE SRAM address for accessing MPE SRAMs for MAC, reading, or writing weights. SSB is used to select the target weight bit for a MAC manually. ERA initializes weights stored in MPE SRAMs to zero. CFD and CFC calibrate or turn on DPWM and CMU, respectively. SRW and SRR are used to write and read weights of MPE SRAMs, respectively. SFT selects a row of an MPE array for accessing MPE SRAMs via DL before using SRW and SRR. Finally, UPD is used to load calibration terms for 5-bit programmable current sources of MPEs.

With communication instructions, RCV receives external data, and SND sends data to the outside of the prototype IC. As shown in Fig. 4.15, each 9-bit data is serially received or sent per instruction via 16 input or output bus ports for each row or column of an MPE array.



Inst.	Full name	10	9	8	7	6	5	4	3	2	1	0	MEC	REG
SSA	Set SRAM addr	0	0	0	layer							inc_dis	🟨	
ERA	Erase	0	0	1	0	MEC	whole	REG_h	REG_v	reg_all	reg_num		🟨	🟨
ADD	Add	0	1	0	0	loc	invert	adc_rst	src_reg_num		trg_reg_num			🟨
MUL	MAC	0	1	1	0	loc	bit_width[3:0]				reg_num		🟨	🟨
CFD	CFG DPWM	0	1	1	1	0	0			on	cal	loc	🟨	🟨
CFC	CFG CYMAC	0	1	1	1	0	1		on	cal	loc	sgn	🟨	🟨
UPD	Update scale	0	1	1	1	1	load	mult	reg_num_h		reg_num_v			🟨
SFT	Shift row/col	1	0	0	0	0	loc	shift_num				rst		🟨
ATV	ReLU activation	1	0	0	0	1		loc	src_reg_num		trg_reg_num			🟨
SSB	Set SRAM bit	1	0	0	1	0	0		osel[3:0]				🟨	
LDA	Load actv	1	0	0	1	0	1		loc	adc_rst	trg_reg_num			🟨
MXP	Max-pool	1	0	0	1	1	0		loc	clear	reg_num			🟨
SRR	SRAM read	1	0	0	1	1	1	0	0	loc	reg_num		🟨	🟨
SRW	SRAM write	1	0	0	1	1	1	0	1	loc	reg_num		🟨	🟨
RCV	Receive	1	0	0	1	1	1	1	0	loc	reg_num			🟨
SND	Send	1	0	0	1	1	1	1	1	loc	reg_num			🟨
LPE	Loop end	1	0	1	0	loc	layer	layer_rst	layer_save	layer_load	trg_reg	trg_rst		
LPS	Loop Start	1	1	epoch_times = MAX LOOP SIZE (2^9 = 512)										
CNT	Control state cnt	1	0	1	1	idx		start	step_count					
JMP	Jump	0	0	1	1	Address[6:0]								
SFS	Select flag set	0	1	0	1					Flag_set_sel[2:0]				

Figure 4.14. The instruction set of test logic includes 21 instruction codes.

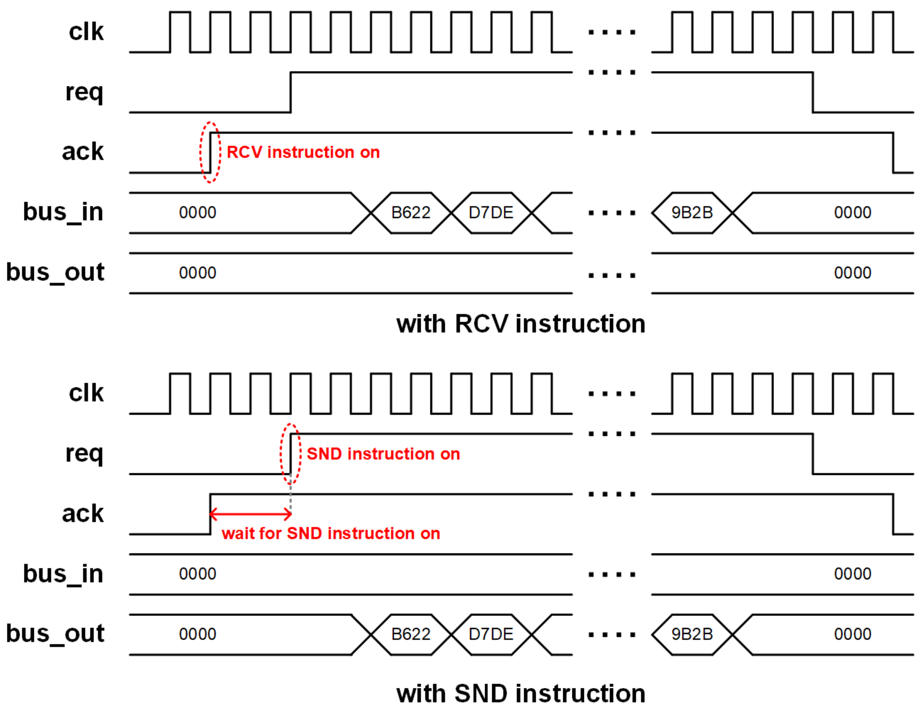


Figure 4.15. The examples of operating an RCV and an SND instruction.

The functions of DNN instructions are as follows. MUL implements MACs by providing control signals for pulse generation to DPWMs, weight bit loading to MPEs, and cyclic accumulator steps to CMUs. ADD provides additions between data in registers. ATV supports ReLU activation for output data. LDA determines an accumulation time point between MAC results. MXP supports max-pooling operations.

The for-loop creation instructions operate as follows. LPS determines a loop's size and stores the loop's starting point. The LPE specifies the point at which the loop ends and returns the instruction SRAM address after the LPS with the same loop index. LPS and LPE support a total of eight loop index creations. LPE also provides incremental controls of MPE SRAM addresses, target weight bit positions, and test logic register orders.

The functions of debugging instructions are as follows. CNT provides manual adjustment of an instruction state. JMP provides manual adjustment of an instruction SRAM address. As shown in Fig. 4.16, SFS selects control signals for a computation core to be sent outside a prototype IC using bus_out ports.

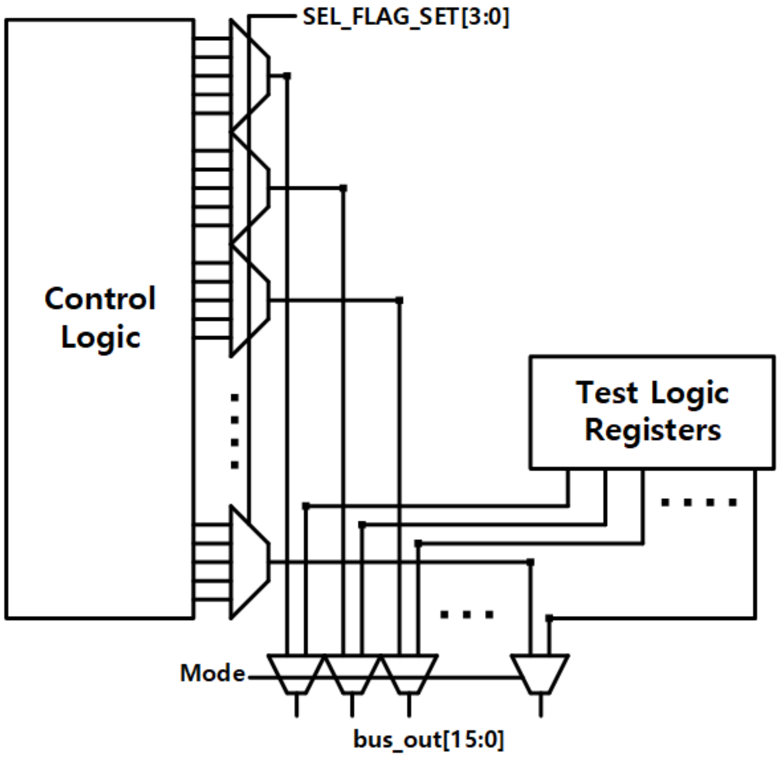


Figure 4.16. The logic for monitoring control signals from the control logic to the computation core.

Chapter 5

Verification Environment for the IC Design Step

5.1 Verification Environment with Three Types of Simulators

Design steps of a mixed-signal DNN accelerator require a wide range of verifications, including circuit-level, system-level, and DNN-level. The following simulators were used for each verification level. First, A HSPICE simulator was used to accurately predict non-ideal characteristics of analog circuits such as a 5-bit programmable current source of an MPE, a CMU, and a DPWM at the circuit level. Second, XMODEL, a System-Verilog-based simulator, was used for system-level verification. Since XMODEL is an event-driven simulator that performs calculations only when any input changes, it is possible to quickly and accurately verify interactions between numerous analog circuits placed as an array by actively utilizing the sparse nature of DNNs. In addition to verifying the functionality at the system level, it is possible to predict the effect on MAC computation accuracy by modeling non-ideal characteristics of analog circuits obtained from the circuit level simulations. Finally, for DNN-level verification, a verification environment using Tensorflow, one of the deep learning frameworks, was prepared to verify factors of changing DNN accuracies, such as analog circuits and calibration schemes proposed in this dissertation. Like system-level verifications, the non-ideality characteristics of analog circuits obtained from circuit-level simulations were modeled using Python to realize the effect on computations in a DNN. DNN-level simulations can determine whether the accuracy drop caused by designed analog circuits is acceptable and, conversely, help determine specifications for the degree of non-ideal characteristics of analog circuits.

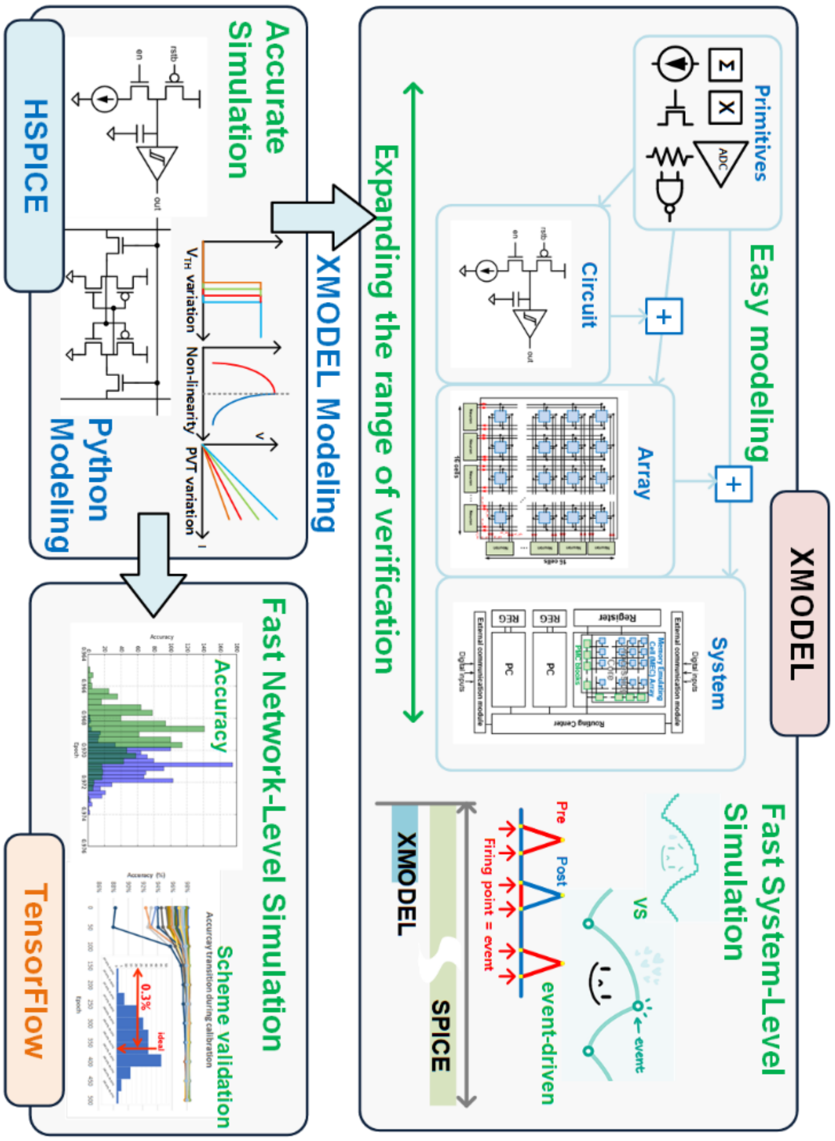


Figure 5.1. An overview of the verification environment with three types of simulators.

5.2 Verification Environment with System-Level Simulations

The system-level verification is performed for wide ranges between circuit-level verifications for a single analog circuit and DNN-level verifications for measuring accuracy for thousands of input images. Specifically, the targets of the system-level verification are as follows. First, the functionalities of test logic and analog and digital operations of a computational core are verified for each instruction. Second, the interactions between test logic and a computational core are verified

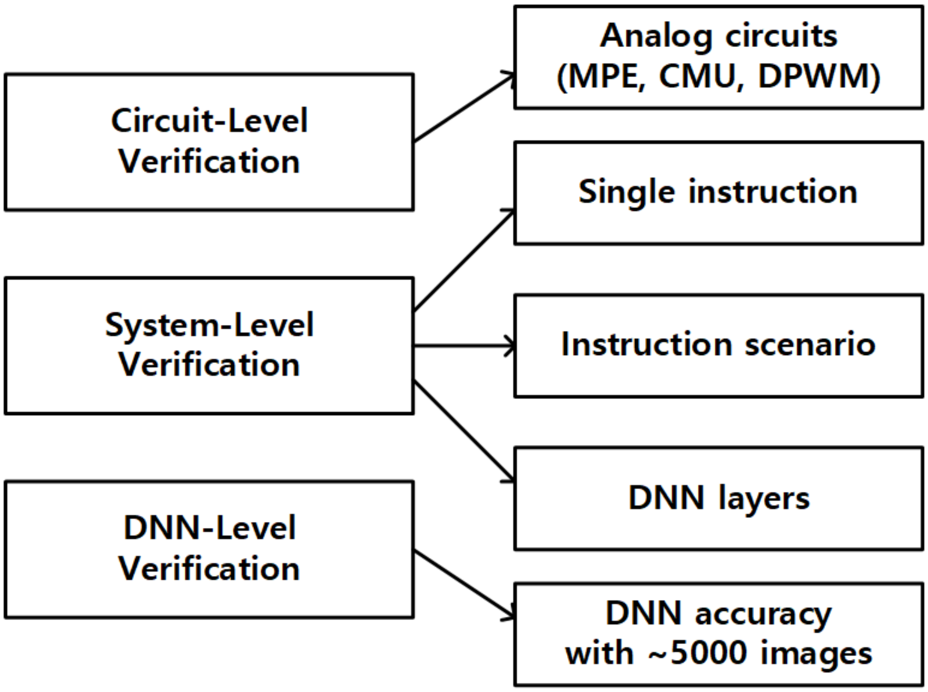

Circuit-Level Verification
System-Level Verification
DNN-Level Verification
Analog circuits (MPE, CMU, DPWM)
Single instruction
Instruction scenario
DNN layers
DNN accuracy with ~5000 images

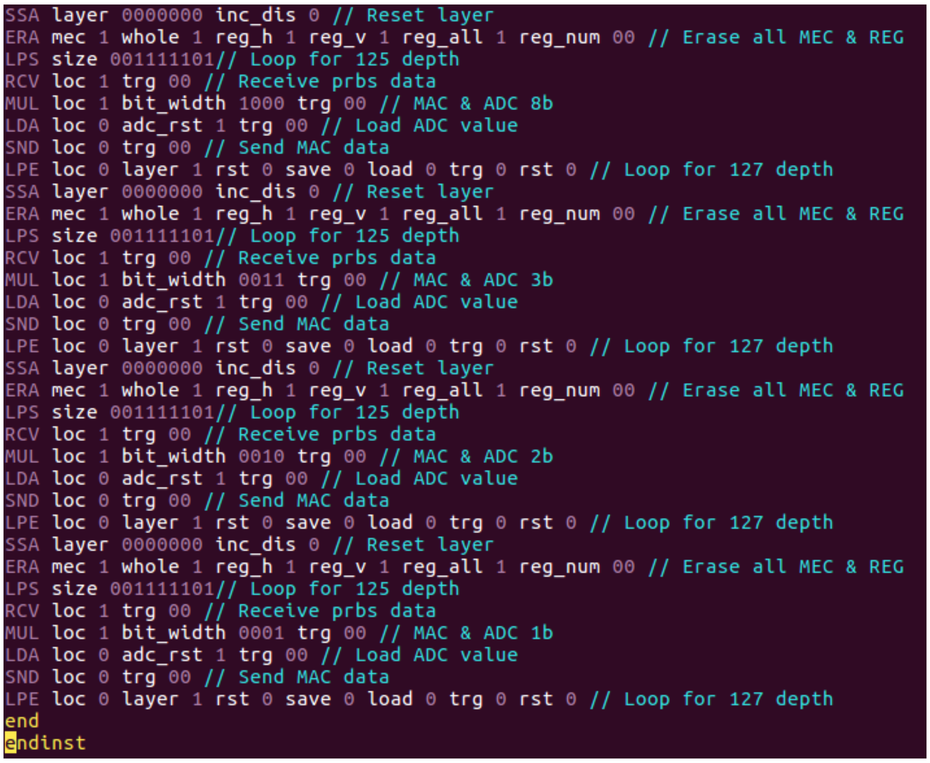
Figure 5.2. The specific range of the system-level verification between the circuit-level and the DNN-level verification.

with instruction scenarios frequently appearing in preparing or running a DNN example, such as a MAC computation or an SRAM read. Finally, a layer computation procedure from an input image to an output label is verified by checking the error of each MAC computation between ideal MAC results extracted from Tensorflow.

Table 5.1 shows the list of instruction scenarios for the system-level verification. The numbering of each scenario was given following the test order to be performed with a prototype IC in a lab. Each scenario sweeps all cases or generates thousands of random data by performing equivalence checks comparing the results of each scenario with ideal values for covering a variety of DNN layer configurations with different inputs, weights, and order of computations. Fig.5.3 shows an example instruction code and system-level simulation result of the 05_mac instruction scenario.

Table 5.1. The list of instruction scenarios for the system-level verifications.

Scenario	Verification Target
00_prgsram	Instruction loading at the programming phase.
01_prgbus	Instruction loading at the instruction running phase.
02_rcvsnd	Identity between received and sent data.
03_era	The functionality of initialization of MPE SRAMs
04_wla	Whether written and read, weights are identical.
05_mac	MAC errors with random data.
06_actv	The functionality of ReLU.
07_mxp	The functionality of max-pooling.
08_dpwm	DPWM configuration and calibration.
09_cnt	Changing an instruction state.
10_mnt	The functionality of transmitting selected control signals to the outside of the IC.
11_jmp	Changing an instruction SRAM address.
12_cal	Decreasing of MAC error along epochs for gradient descent calibration process.



```
SSA layer 0000000 inc_dis 0 // Reset layer
ERA mec 1 whole 1 reg_h 1 reg_v 1 reg_all 1 reg_num 00 // Erase all MEC & REG
LPS size 001111101// Loop for 125 depth
RCV loc 1 trg 00 // Receive prbs data
MUL loc 1 bit_width 1000 trg 00 // MAC & ADC 8b
LDA loc 0 adc_rst 1 trg 00 // Load ADC value
SND loc 0 trg 00 // Send MAC data
LPE loc 0 layer 1 rst 0 save 0 load 0 trg 0 rst 0 // Loop for 127 depth
SSA layer 0000000 inc_dis 0 // Reset layer
ERA mec 1 whole 1 reg_h 1 reg_v 1 reg_all 1 reg_num 00 // Erase all MEC & REG
LPS size 001111101// Loop for 125 depth
RCV loc 1 trg 00 // Receive prbs data
MUL loc 1 bit_width 0011 trg 00 // MAC & ADC 3b
LDA loc 0 adc_rst 1 trg 00 // Load ADC value
SND loc 0 trg 00 // Send MAC data
LPE loc 0 layer 1 rst 0 save 0 load 0 trg 0 rst 0 // Loop for 127 depth
SSA layer 0000000 inc_dis 0 // Reset layer
ERA mec 1 whole 1 reg_h 1 reg_v 1 reg_all 1 reg_num 00 // Erase all MEC & REG
LPS size 001111101// Loop for 125 depth
RCV loc 1 trg 00 // Receive prbs data
MUL loc 1 bit_width 0010 trg 00 // MAC & ADC 2b
LDA loc 0 adc_rst 1 trg 00 // Load ADC value
SND loc 0 trg 00 // Send MAC data
LPE loc 0 layer 1 rst 0 save 0 load 0 trg 0 rst 0 // Loop for 127 depth
SSA layer 0000000 inc_dis 0 // Reset layer
ERA mec 1 whole 1 reg_h 1 reg_v 1 reg_all 1 reg_num 00 // Erase all MEC & REG
LPS size 001111101// Loop for 125 depth
RCV loc 1 trg 00 // Receive prbs data
MUL loc 1 bit_width 0001 trg 00 // MAC & ADC 1b
LDA loc 0 adc_rst 1 trg 00 // Load ADC value
SND loc 0 trg 00 // Send MAC data
LPE loc 0 layer 1 rst 0 save 0 load 0 trg 0 rst 0 // Loop for 127 depth
end
endinst
```

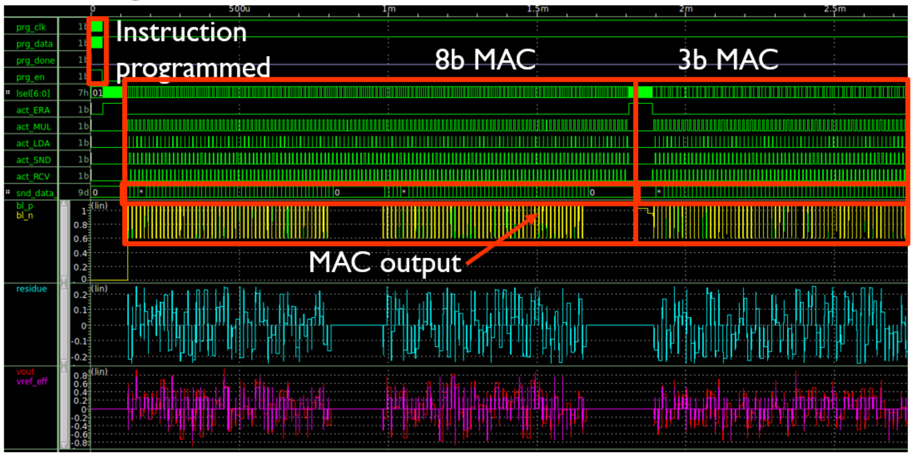


Figure 5.3. The instruction code and simulation result of the 05_mac instruction scenario.

DNN layers on a prototype IC are implemented by combining instruction sequences including 02_rcvsnd, 05_mac, 06_actv, and 07_mxp scenarios with DNN layer configurations. Scenarios as 00_prgsram, 01_prgbus, 03_era, 04_wla, 08_dpwm, and 12_cal are used to verify pre-requisite tasks for setting a prototype IC before running DNN layers. The remaining scenarios (09_cnt, 10_mnt, 11_jmp) are used for debugging a prototype IC. The system-level verifications verified the gradient descent calibration method by processing MAC results from the computation core inside a prototype IC. Weights were used as max (255 = 9'b0_1111_1111), and randomized 6-bit inputs were generated using XMODEL PRBS primitives. The randomly generated effects of process variations were applied to XMODEL models for each MPE current source. Input and error terms were clipped to 2-bit in test logic, and gradients generated at each MPE were also clipped at 3-bit through vector multiplication to lower the area consumption of an MPE. As shown in Fig. 5.4, MAC errors could be minimized through the repeated calibration epochs of 5 ms.

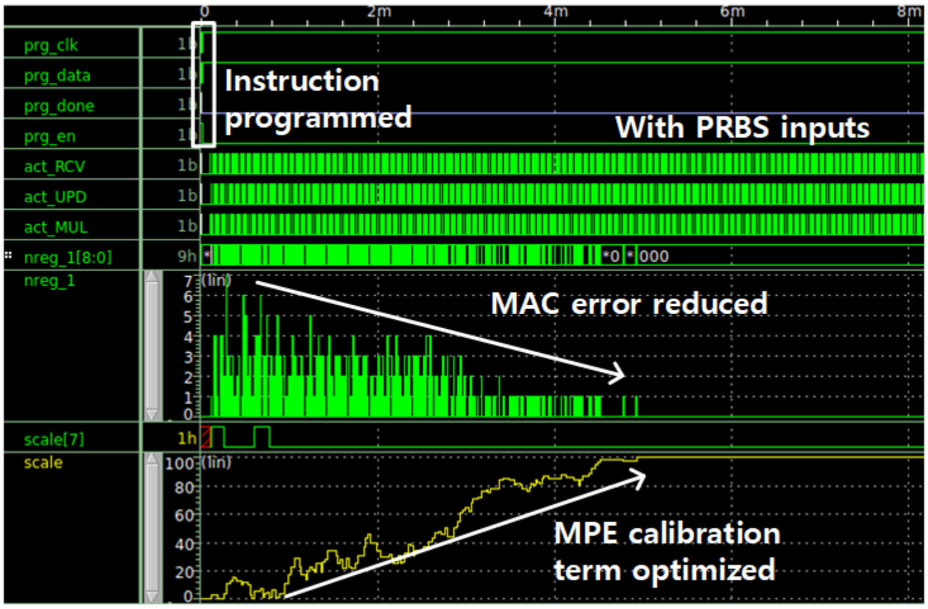


Figure 5.4. The simulation result of gradient descent calibration method on prototype IC.

XMODEL testbenches are automatically generated with layer descriptions and input datasets of DNNs to reduce the difficulties of building system-level verification environments for various DNN layers and weight precision configurations. The generated testbench consists of an XMODEL model of the prototype IC, weight information stored in MPE SRAMs, and an FPGA part controlling a prototype IC in a measurement process.

Weights were realized with the Verilog force statement assuming that weights were stored in MPE SRAMs before MAC calculations for fast simulation operation. Weight values were generated with training processes with Tensorflow and reshaped to fit the form of MPE SRAM addresses.

An equivalence check module of the FPGA part (eq_chk.sv) calculates errors for each MAC result by comparing values from the ideal DNN model acquired from Tensorflow for each image input. In

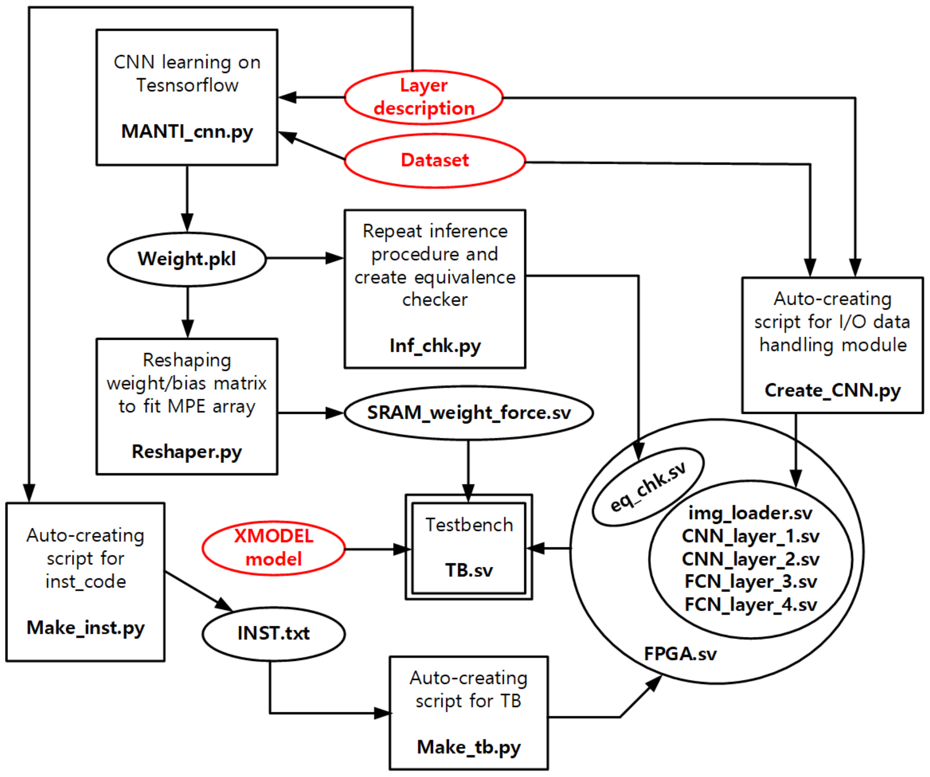


Figure 5.5. The task flow of testbench auto-creation with neural network information and XMODEL models.

addition, the FPGA part includes modules that store, reshape, and transmit input and output data between layers to fit the shape of an MPE array. The instruction sequence for running a prototype IC is automatically generated by adjusting the size of for-loops and the order of instructions according to layer descriptions and is provided to the FPGA part.

As shown in Fig 5.5, the functionality of a prototype IC running a 9-bit MNIST CNN example could be verified with the testbench auto-generating task flow. The errors in MAC results could be confirmed. The error was accumulated along with layers, but the maximum value of 4 is sufficiently small. The simulation time required for each image was 15 minutes and 9 seconds, making it possible to verify a mixed-signal system faster than other circuit-level simulators.

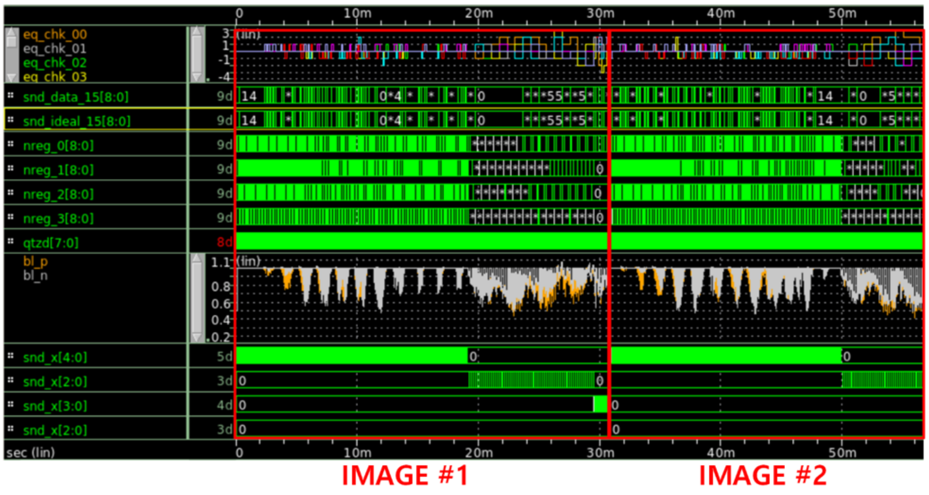


Figure 5.6. The XMODEL simulation result for the MNIST CNN example with the 9-bit weight precision configuration.

With 9-bit CIFAR-10 CNN and CIFAR-100 examples, since the memory capacity required to store all network weights is greater than 288 kb of the prototype IC, four weight reloading processes are required, as shown in Fig 5.6. For XMODEL simulations for simplifying the simulation process, the reloading processes were simplified by using Verilog force statements. For measurements on prototype ICs requiring more time consumption for data UART communications than running instructions, weights are reloaded after 5,000 images are processed, and the intermediate results are stored in FPGA BRAMs. Fig 5.7 shows the result of XMODEL simulation for checking the functionality of a prototype IC with running a 9-bit CIFAR-10 example, and the simulation time of 2-hour 34-minute 22-second per image was consumed.

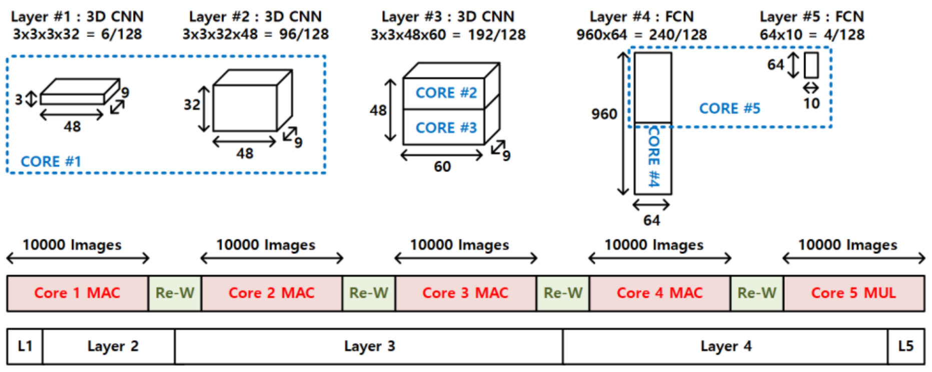


Figure 5.7. Weight loading scheme for the example CIFAR-10 CNN on a computation core.

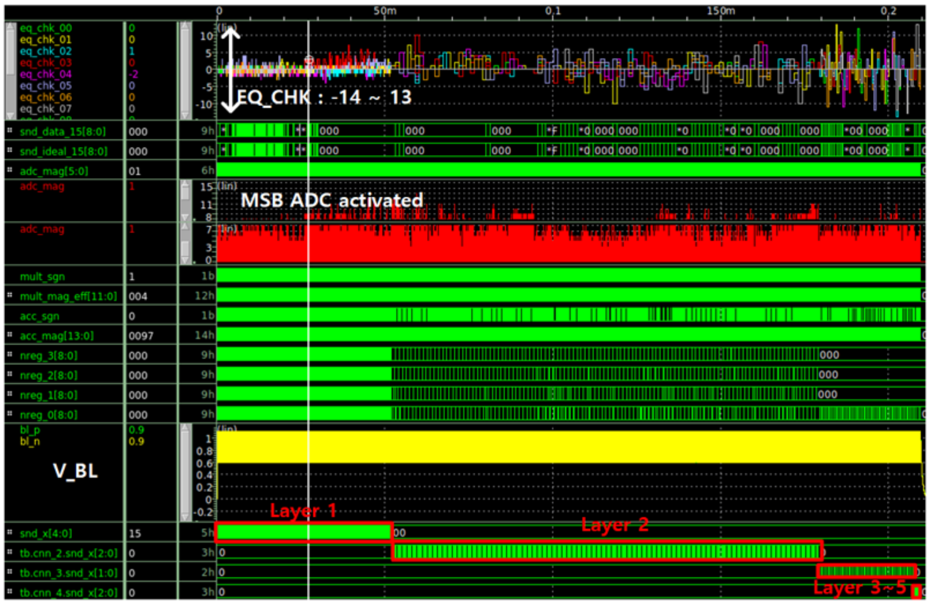


Figure 5.8. The XMODEL simulation result for the CIFAR-10 CNN example with the 9-bit weight precision configuration.

5.3 Verification for DNN examples with DNN-Level Simulations

The DNN-level simulations provide predictions of DNN accuracy by modeling non-ideality characteristics of various analog circuits with Python on the Tensorflow domain. Compared to the circuit- and system-level simulations, which take a long time, from 16 minutes to several days, to classify an image, the DNN level simulations require simple models for computation parts associated with DNN accuracy. Moreover, the DNN level simulations maximize the parallel computing

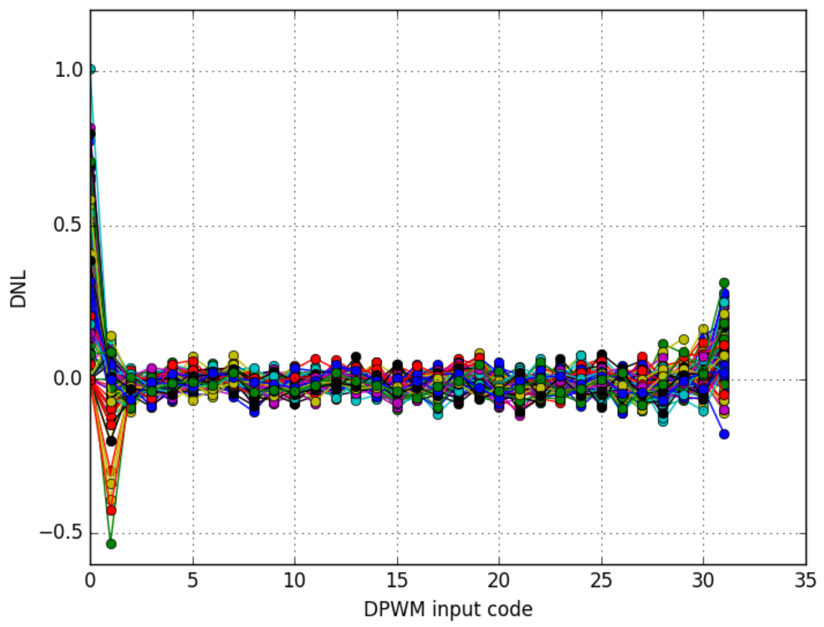


Figure 5.9. The differential non-linearity of a designed DPWM was extracted with 100 HSPICE Monte-Carlo simulations.

capability of GPUs and allow for the prediction of DNN accuracy with a small time consumption of less than 1 minute for 5000 images in the validation dataset. In this dissertation, the hardware used for the DNN level simulations was NVIDIA GeForce GTX 1080 GPU. The fast speed of DNN level simulation can significantly reduce the effort required in predicting and determining accuracy-related performance specifications in the IC design step. The followings are two examples of determining the maximum allowable range of each non-ideality characteristic in the analog circuit design steps.

First, it analyzes the effect on the DNN accuracy of DPWM offset terms. In Chapter 4, a DPWM control method was proposed to reduce

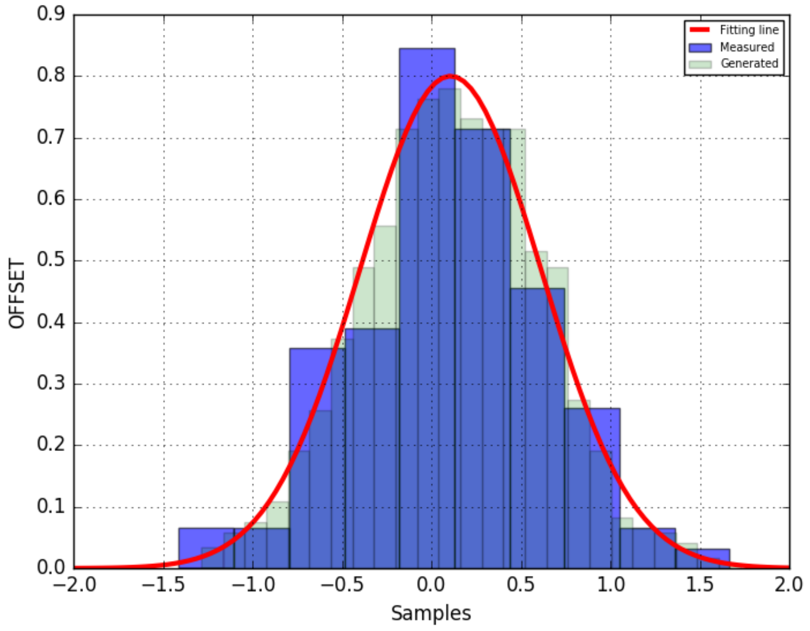


Figure 5.10. The result of curve fitting with Gaussian distribution for the offset variations of a DPWM.

offset terms caused by parasitic capacitors of DPWM I&F units, but as shown in Fig.5.9, offset terms are not completely reduced due to the comparably significant effect of process variations in cases with small manner of an MPE array that reuses 16 DPWMs for each computation and then repeatedly tiling 16 process variation terms for every 16 inputs. In addition, the offset term model was implemented only to the analog-computed latter 5-bit of inputs.

As shown in the DNN level simulation results in the blue histogram of Fig.5.11, there are only tiny accuracy drops of up to 0.2%p for a 9-bit MNIST CNN example with DPWM offset term models obtained

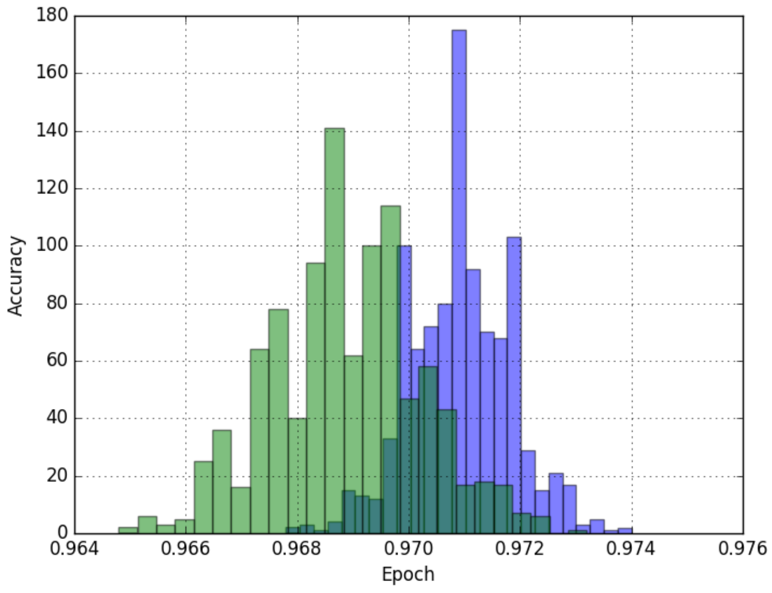
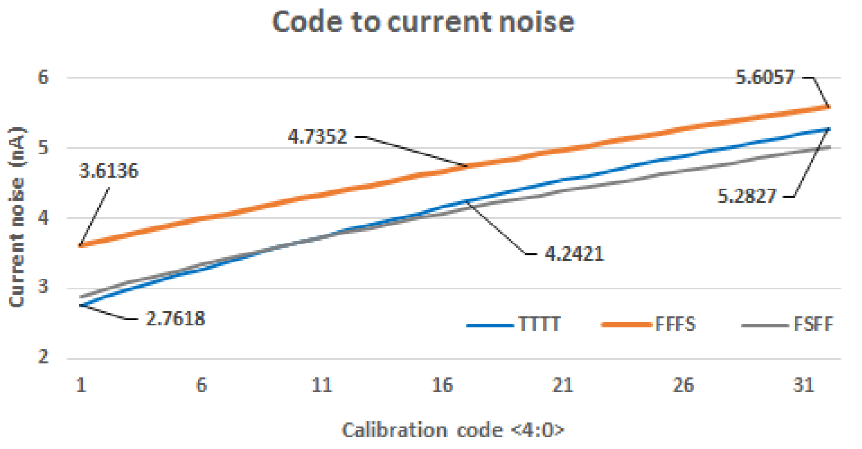


Figure 5.11. The Tensorflow simulation result for predicting the accuracy of the MNIST CNN example with offset variations of 16 DPWMs extracted from HSPICE Monte-Carlo simulations.

from the HSPICE Monte-Carlo simulation. The green histogram is the DNN level simulation results with a ten times larger standard deviation of the Gaussian distribution for generating DPWM offset term than the blue histogram, and there is only a tiny accuracy drop of up to 0.6%p. Therefore, the offset characteristics of the DPWM circuit design do not



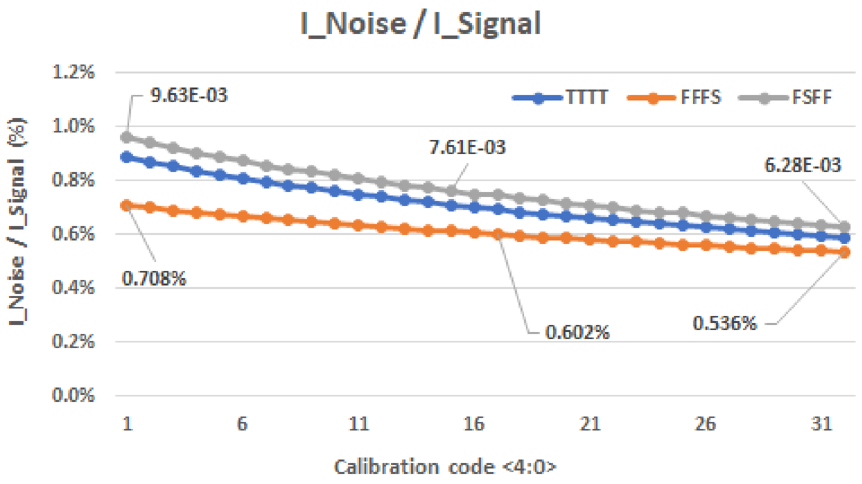


Figure 5.12. HSPICE simulation results for noise and ratio between output current and noise of an MPE's 5-bit programmable current source at three representative PVT variation corners with sweeping an input code.

cause a major problem in the accuracy drop, so more design steps were not required to reduce non-ideality, enabling to move quickly to the next design steps.

Another DNN-level simulation example is an analysis of the relationship between DNN accuracy and noises on the output currents of 5-bit programmable current sources of MPEs. As shown in Fig.5.12, HSPICE noise simulations were performed for sweeping input codes and process-voltage-temperature (PVT) variation corners to find the maximum noise case. The largest standard deviation of the noise to output current ratio is 9.63e-3 with an input code of 0 at the FSFF corner.

Random noise terms were generated using Gaussian distribution with a standard deviation of 1e-2 greater than the maximum value obtained from HSPICE simulations to determine the acceptable noise range on MPEs' 5-bit programmable current sources. Since noise affects each weight bit multiplication on an MPE array, noise terms are added to each weight bit computation in the Tensorflow computation flow model.

Fig. 5.13 shows the DNN level simulation results of an MNIST CNN example with 500 repetitions. Since the noises on MPEs' 5-bit programmable current sources cause an acceptable drop in DNN accuracy of 0.1%p compared to ideal accuracy, a designed 5-bit programmable current source was determined to use for a prototype IC. Simulations with a doubled standard deviation of 2e-2 of the Gaussian

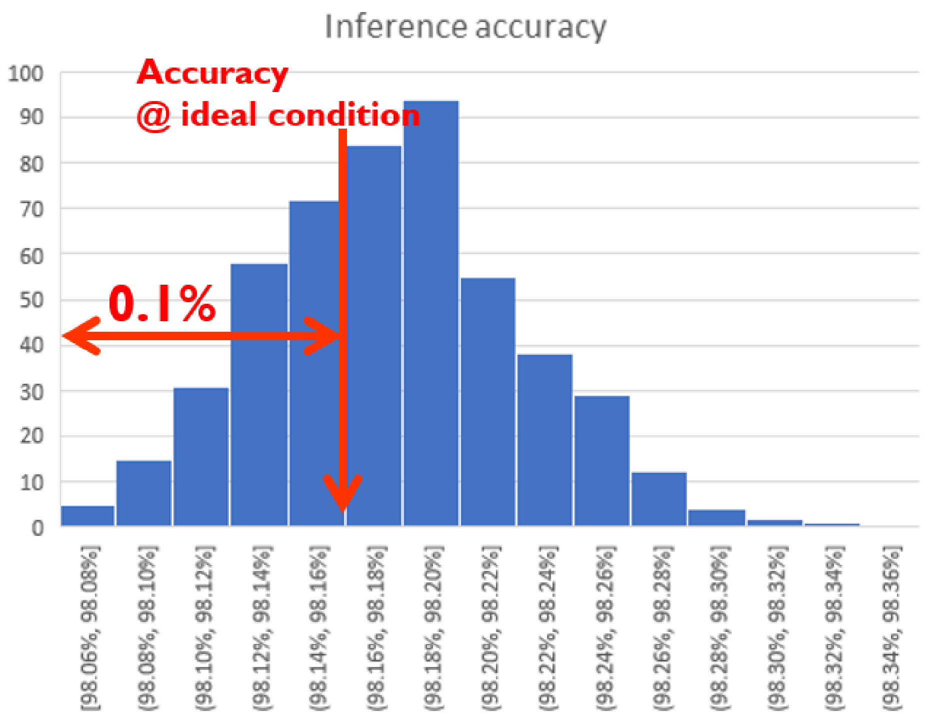


Figure 5.13. Tensorflow simulation results for MNIST CNN accuracy drop with randomly generated MPE's 5-bit programmable current source noise model extracted from HSPICE simulations with the designed circuit.

distribution used to generate noise randomly were performed to increase the reliability of the design. As Fig.5.14 shows, when repeatedly simulating for 200 times, a significant accuracy drop of 10%p occurs. However, simulating by implementing a process of performing 500 epochs of gradient descent calibration of Chapter 3 on Tensorflow, the DNN accuracy drop was reduced to 0.3%p.

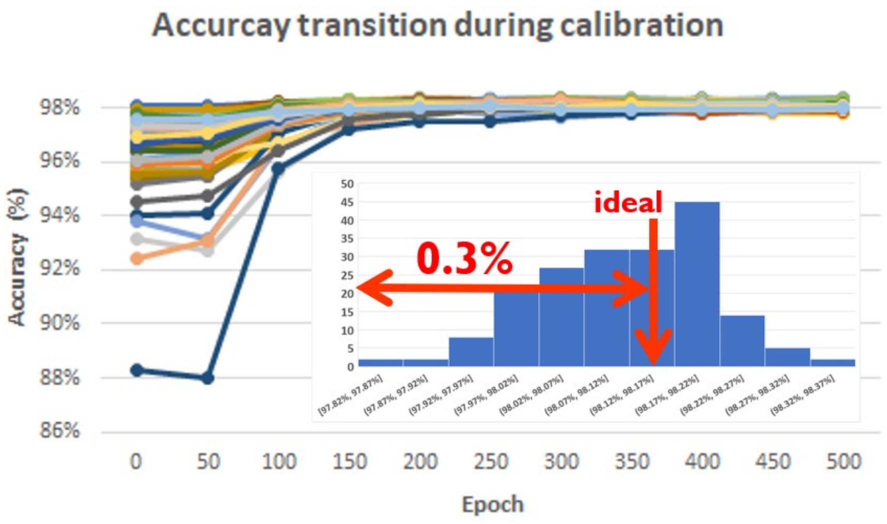


Figure 5.14. Tensorflow simulation results for MNIST CNN accuracy drop with the gradient descent calibration process with MPE's 5-bit programmable current source noise model with a doubled standard deviation of HSPICE simulation results.

Chapter 6

Measurement Results of Proposed Neural Network Accelerator IC

6.1 Measurement Environment and Prototype IC

We constructed a test environment with a 3mm × 3mm IC prototype fabricated with a 28nm CMOS process for evaluation. As shown in Fig. 6.1, a Xilinx Kintex-7 FPGA KC705 evaluation kit was used to handle control signals and data for the prototype IC via a UART serial port.

Fig. 6.2 shows the detailed equipment connection for the measurement environment. A Jupyter-Notebook-based environment on a personal computer (PC) was built to provide input and weight data and analyze output data. A PC and FPGA communicate at a baud rate of 115200 bps over a UART serial port. Since the UART baud rate is significantly lower than the frequency of 40 MHz of a prototype IC, communication between the PC and the FPGA takes place only at the

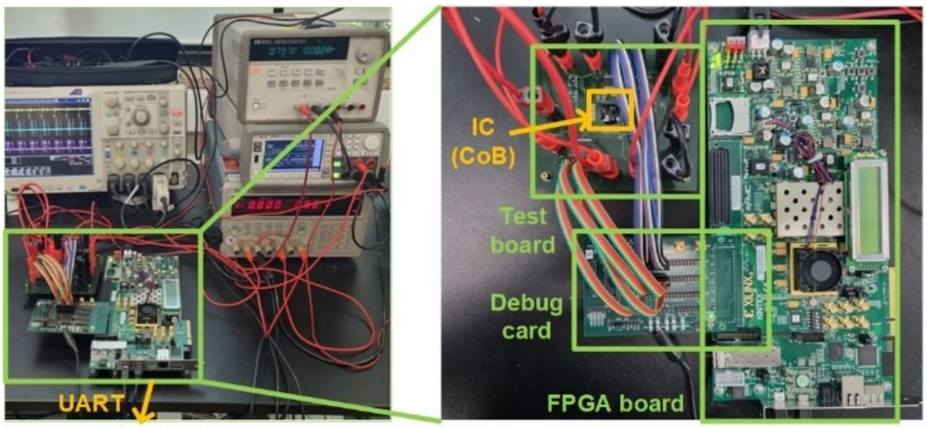


Figure 6.1. The measurement environment with the FPGA board.

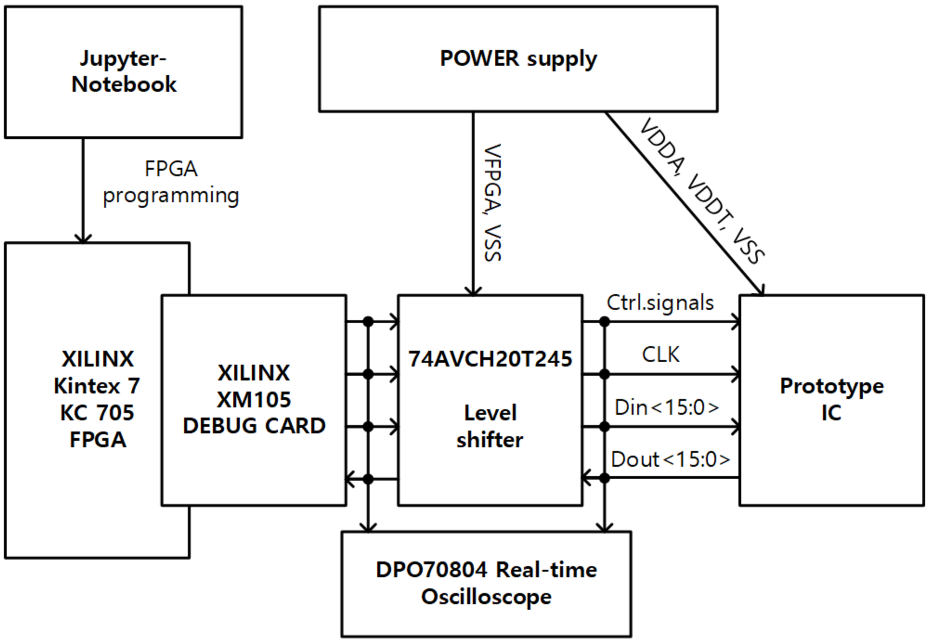
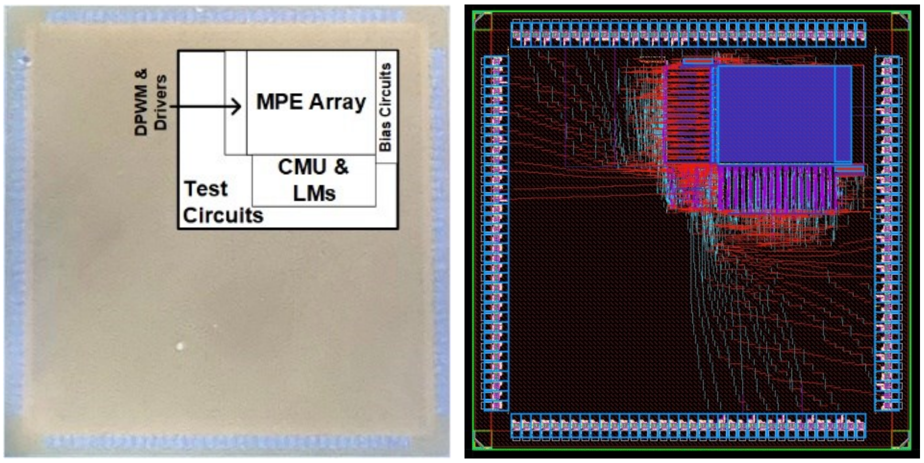


Figure 6.2. The schematic of the measurement environment.

first and last stages of the measurement process. Most of the operations required to construct a DNN were implemented on the FPGA. The Xilinx XM105 Debug Card was attached to the FPGA and connected to a printed circuit board (PCB), including a prototype IC using jumper cables. A 74AVCH20T245 level shifter capable of converting a 2.5V signal supplied from the debug card into a 0.8-1.2V signal required by a prototype IC was attached to the PCB. A power supply provides supply voltages and bias currents to a prototype IC and level shifters. The oscilloscope provides various detection points for debugging the measurement environment.

Fig. 6.3 shows the die photo of the prototype IC and the layout under power straps, and the computation core area is 0.97 mm2. The prototype IC includes a 0.57 mm2 test logic with 21 instruction sets for generating control signals for the computation core and supporting functions for on-chip NN realization such as max-pooling, ReLU activation, weight loading, and for-loop creation. A 4-layer MNIST CNN example and a 5-layer CIFAR-10 CNN example in Fig. 2.3 were used. Table 6.1 shows the detailed specification and measurement results of the prototype IC. With a clock frequency of 40MHz, the prototype IC achieves a throughput of 0.59 GOPS for instruction loading, weight loading, and MAC operations. As shown in Fig. 6.4, for the supply voltage of 0.8~1.2V, the peak energy efficiency of 0.17~0.54 TOPS/W for signed 9-bit and 0.22~0.75 TOPS/W for binary weight precision were measured.



(a) (b)

Figure 6.3. (a) The die photo of the prototype IC (b) the layout of the IC under power straps.

Table 6.1. Detailed Specification and Measurement Results of the Prototype IC.

Die area	9mm^2
Core area	0.97mm^2
Test logic area	0.57mm^2
Supply voltage	0.8-1.2V
The number of MPE	256
Clock frequency	40MHz
Memory Capacity	288kb
Weight density	0.307bit/μm^2
Weight precision	binary - signed 9-bit
Data precision	signed 9-bit
Peak energy efficiency	0.22-0.75 TOPS/W for binary weights 0.17-0.54 TOPS/W for signed 9-bit weights
Throughput	0.59 GOPS
Accuracy with s8-bit weight	MNIST: 98.32 % (98.74% for ideal) CIFAR-10: 74.18 % (73.85% for ideal)

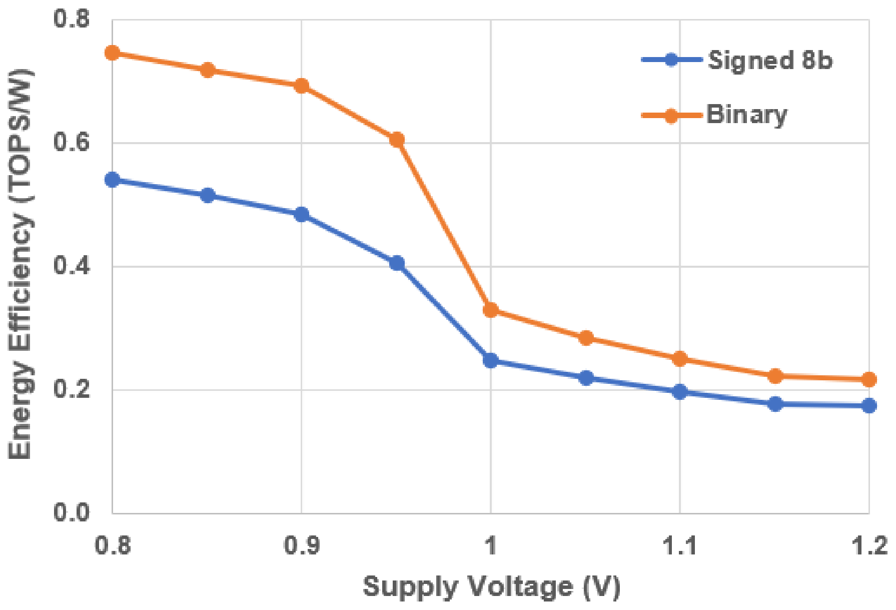


Figure 6.4. Peak energy efficiency with signed 9-bit weight precision sweeping supply voltage.

6.2 Measurement Results

Fig. 6.5 compares the MPE utilization ratio and energy efficiencies between the mixed-signal and digital-only MAC operations for each layer of CNN examples. The energy efficiency of the mixed-signal MAC was measured directly from the prototype IC and all-8-bit digital input separately three times, 3-bit, 3-bit, and 2-bit, and then added up each increased energy consumption to the zero-input case. The input and biasing current of analog circuits were gated to exclude energy consumption on the analog domain for measuring the energy of digital computation blocks. With the example MNIST/CIFAR-10 CNNs, the total energy consumption with mixed-signal inputs is reduced by 27.72/40.44%, respectively, compared to the all-digital input case. Fig. 6.6 provides the comparison ratio of additional energy consumption compared to the case with zero-inputs and gating analog circuits for two cases: First, mixed-signal 8-bit vs. all-digital 8-bit. Second, analog 5-bit vs. all-digital 5-bit with excluding the effect of common digital 3-bit computations. The layer with the highest energy consumption improvement of 46.12% is the input layer of the CIFAR-10 CNN because zero input is scarce for input RGB images, and the number of transitions on logic gates is the largest for the digital domain.

Conversely, the layer with the lowest energy consumption improvement of 20.44% is the first fully-connected layer in the CIFAR-

10 CNN, with very sparse non-zero inputs. In addition, the first layer of the MNIST CNN with the lowest column utilization among all layers also showed a low energy consumption improvement of 24.51% since the prototype IC has no column-wise gating function for analog operations for unused columns. The IC can improve energy efficiency for low column utilization layers if the CMU at unused columns can be turned off.

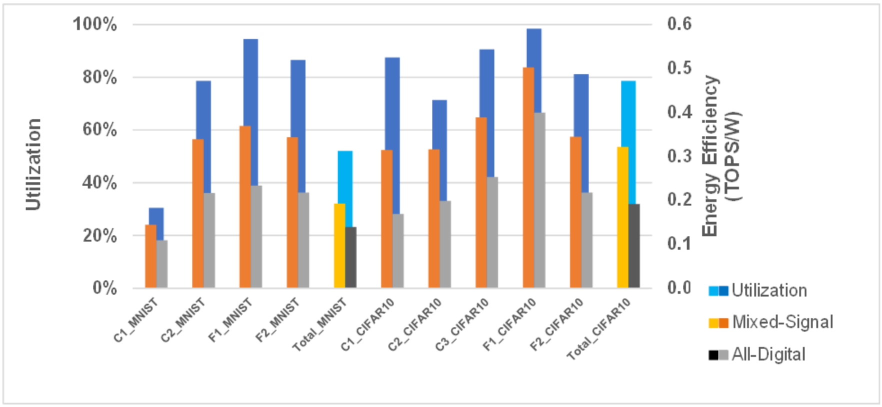


Figure 6.5. The utilization and energy efficiency with mixed-signal and digital-only MAC operations for each layer of CNN examples.

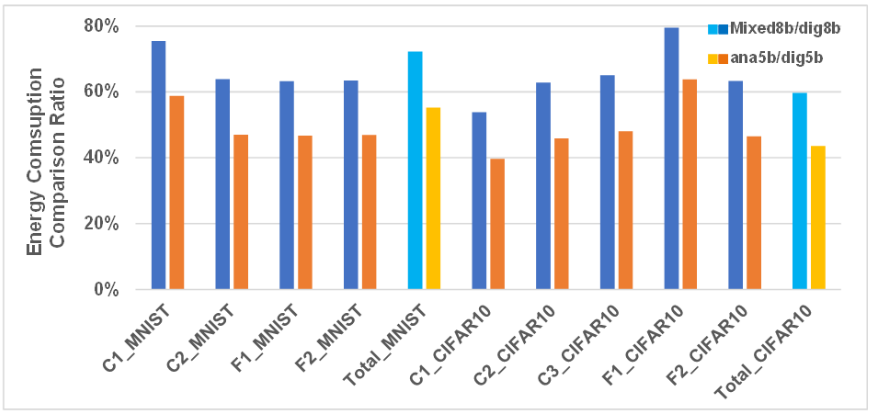


Figure 6.6. The comparison ratio of additional energy consumption compared to the case with zero-inputs and gating analog circuits.

Fig. 6.7 shows that the prototype IC achieves a minimal accuracy drop of -0.42 to 0.33%p compared to the ideal accuracy, for example, MNIST/CIFAR-10 CNN with signed 9-bit weights applying a calibration process. The input parameters of the 5-bit programmable current source on MPEs were calibrated from the initial value, the midpoint of the adjustable range. The 16×16 calibration parameters were trained by gradient descent optimization for cost function, the sum of squared errors. All weights were fixed to 255 and used random 16 inputs whose sum did not exceed 64 for 300 epochs. The trained parameters are quantized to 5-bit for programmable current sources. Even without calibration, MSB 3-bit digital computations guarantee tiny accuracy drops of 2.44/2.80%p compared to the ideal accuracy.

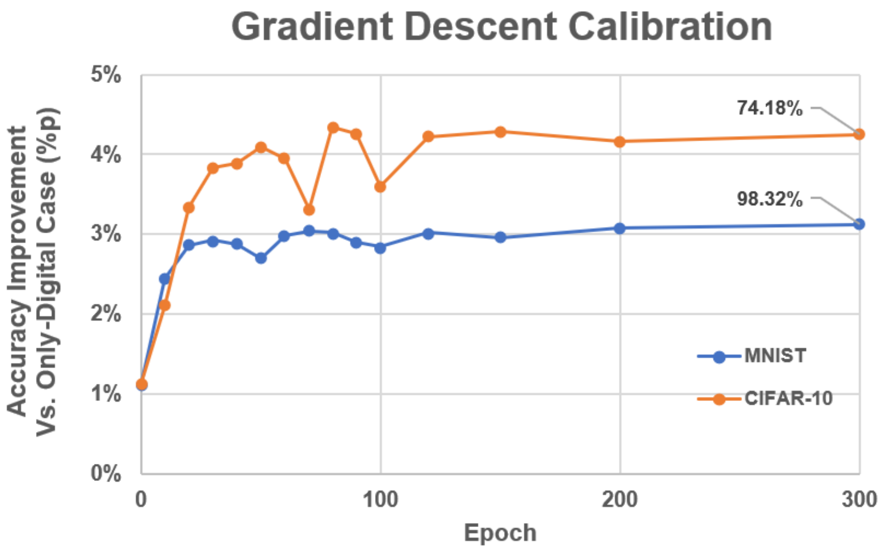


Figure 6.7. The accuracy improvement with the calibration process for the example MNIST/CIFAR-10 CNNs.

The accuracy improvement with the calibration process saturates around 150 epochs, and it was confirmed that the result accuracy was 2.02/3.13%p higher than the case without applying calibration.

Fig. 6.8 shows top-1 and top-5 accuracy changes when the CIFAR-100 CNN in Fig. 2.3(c) is tested under the same conditions. After completing the calibration process, the final top-1/top-5 accuracy drop is 2.07/-0.77%p, and the top-1 accuracy drop is higher than that of MNIST/CIFAR-10 CNN examples. In addition, the accuracy drop for the only-digital-input case is 10.54/7.76p%, which is significantly higher than the MNIST/CIFAR-10 CNN examples. The primary cause of the high accuracy drop is the unstable learning state of the example

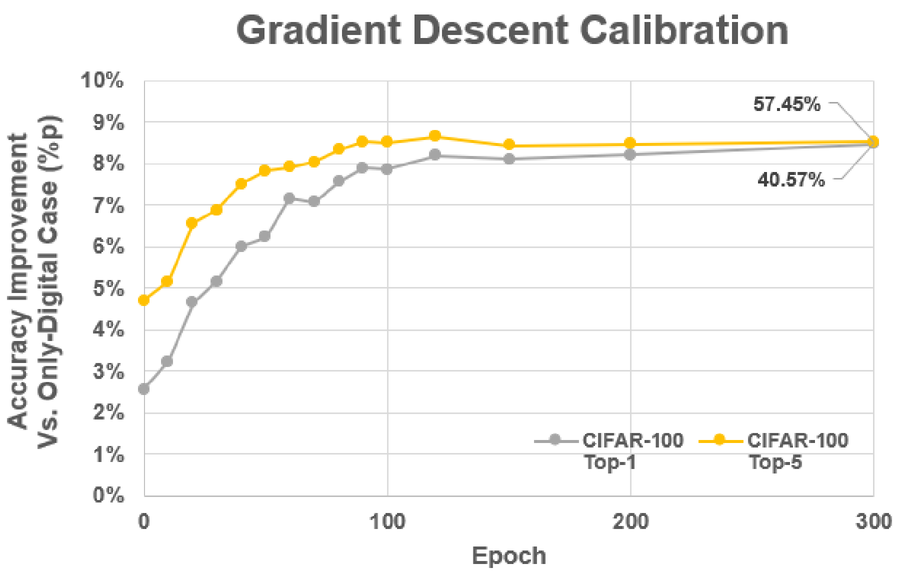
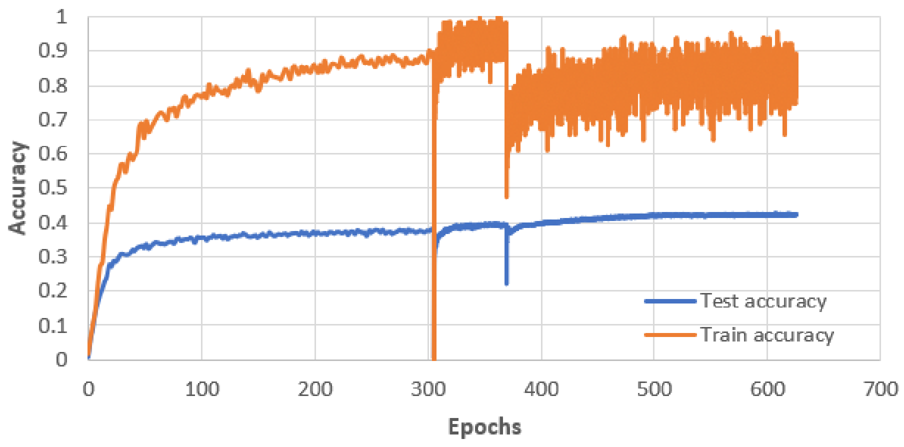


Figure 6.8. The accuracy improvement with the calibration process for the example CIFAR-100 CNN.

CNN. Fig. 6.9 shows the training process of the example CIFAR-100 CNN. Despite three long training processes, the test accuracy sustained much higher than the training accuracy. Therefore the network is overfitted. In addition, since the test cost tends to increase at later iterations of the training process, it can be presumed that the probability



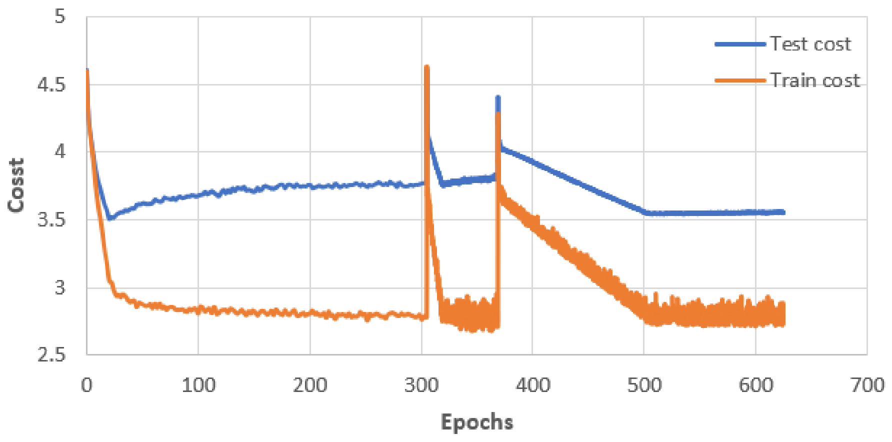


Figure 6.9. The training and test accuracy and training and test cost change during three training processes.

of the network selecting similar-but-incorrect classes increases. It can also be thought that this network's characteristic helped the top-5 accuracy record a higher accuracy than the ideal accuracy with the non-ideality from the analog operations of the prototype IC. And the top-5 accuracy increases significantly with activating analog operations even without the calibration process because the outputs of the last fully connected layer with relatively low values for similar-but-incorrect classes can produce non-zero values with latter bit computations.

For checking less additional energy consumption for the latter weight bit-position than MSB, the energy consumption for 1 MAC operation was measured with increasing weight precision from binary to signed 9-bit. Fig. 6.10 shows the peak energy efficiency with changing weight precision configurations for four cases of inputs for 16 WLs. The four cases are as follows; First, all inputs are zero (zero case). Second, all inputs are 31 (8'b0001_1111), the maximum of LSB 5-bit (analog-max case). Third, all inputs are 224 (8'b1110_0000), the maximum MSB 3-bit (digital-max case). Fourth, all inputs are maximum of 255 (8'b1111_1111, max case). The weights for all MPEs are the same as +170 (=9'b0_1010_1010), which causes many transitions of eight. For reference, it was confirmed that when only the analog operation is used, there is no difference from using 255 (9'b0_1111_1111) for weights with the number of transitions of two.

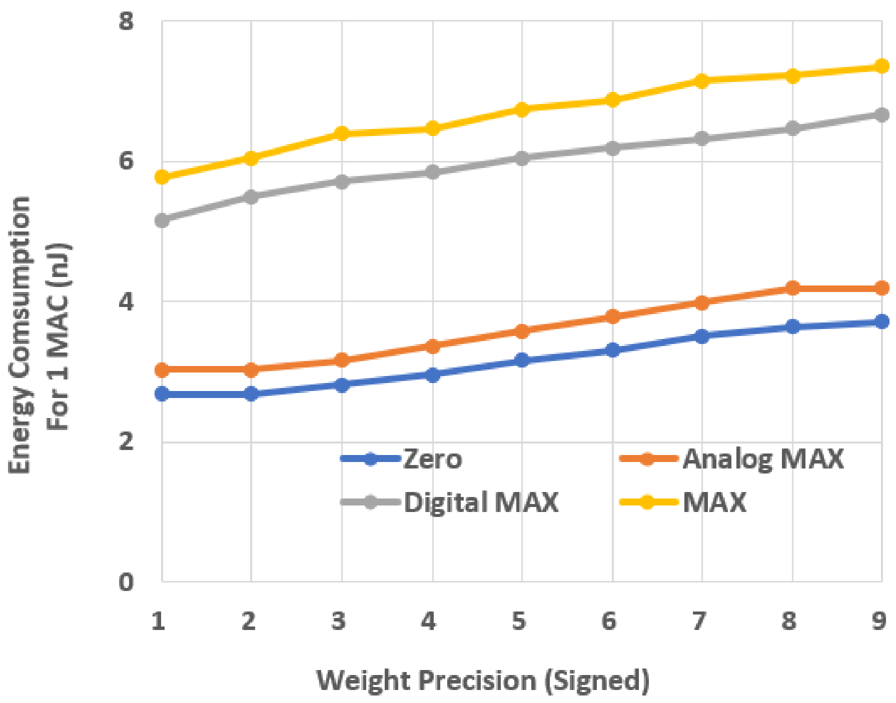


Figure 6.10. The peak energy efficiency with different types of inputs with changing weight precision configurations.

As the weight precision increases, energy consumption increases as a linear relationship with the large constant term. Since the constant term is larger than the differential term, the energy consumption for the latter weight bit-position consumes less energy than the MSB. The 4-level quantization and accumulation of CMU repeated eight times to generate a signed 10-bit output generates the constant term, and the digital computation, bit-line pre-charge, DPWM pulse generation, and weight bit shifting create the differential term for each weight bit configuration. In addition, due to the energy consumed during the digital transitions, the energy consumption increased by weight precision of the digital-max case is 1.47 times greater than that of the analog-max case.

Fig. 6.11 shows the energy breakdown, which shows the energy consumption ratio of each circuit in the case of maximum energy consumption. Energy consumption in MPE SRAM and the LM shared by analog and digital domains does not affect the difference in energy efficiency improvement between mixed-signal and all-digital cases. The

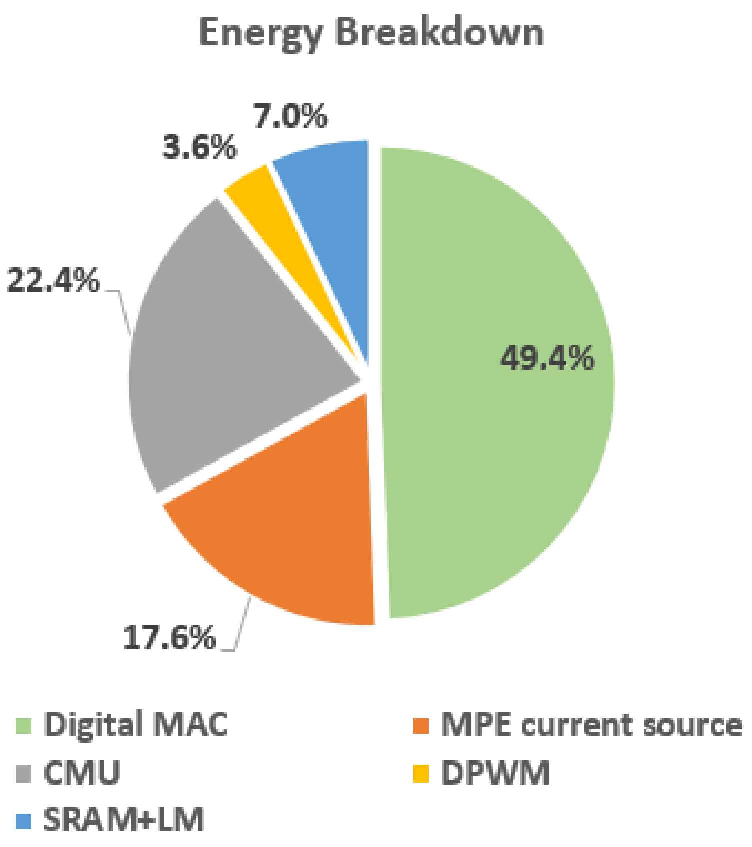


Figure 6.11. Energy breakdown.

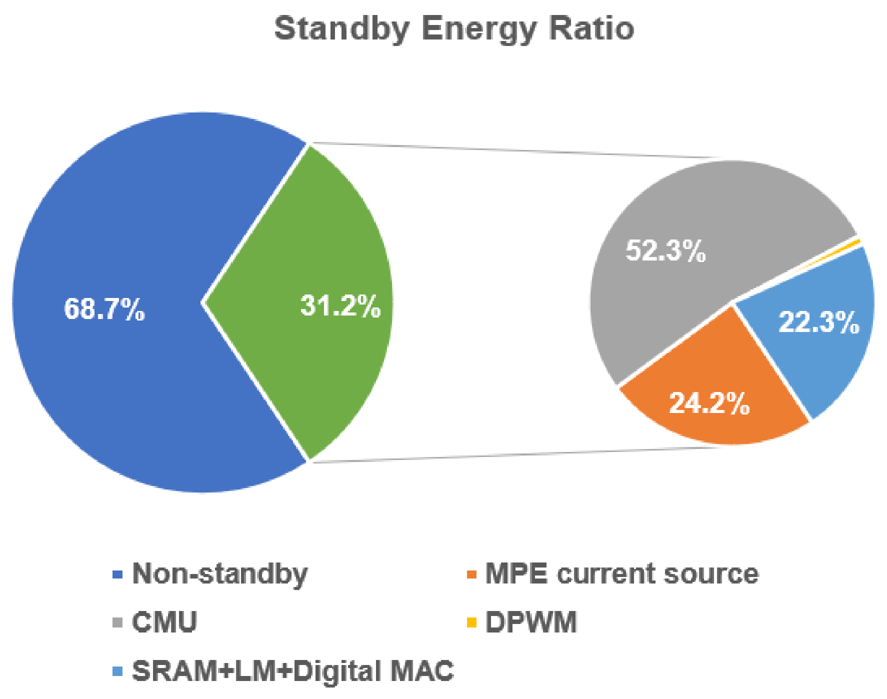


Figure 6.12. Standby energy ratio with the max case input.

analog domain, occupying 5/8 of the operations, accounts for 43.6% of the total energy consumption by adding the figures of the DPWM, the MPE current source, and the CMU. On the other hand, the digital domain, occupying 3/8 of the operations, accounts for 49.4% of the total energy consumption, and the energy consumption per bit is 1.89 times higher than the analog operation. However, the energy efficiency of the digital operation in the example CNNs is better than in the maximum energy case since the bit sparsity of input and weight reduces energy consumption. In addition, the energy consumption of a CMU is more than 20 times that of an MPE current source, indicating that the CMU is the primary energy consumption source of the analog computations. As shown in Fig. 6.12, the standby energy consumption is 31.3% of the maximum energy consumption, of which the entire analog circuit and 16 CMUs generate 77.6% and 52.4%, respectively. Therefore, reducing the standby current of an op-amp in a CMU is necessary to improve the analog domain energy efficiency of the prototype IC.

Fig. 6.13 shows the area breakdown, which shows the area consumption ratio of each circuit. The area of the MPE array is 0.74 mm2, 77.29% of the area of the computation core. Moreover, the accounted area of SRAM, 5-bit programmable current source, and auxiliary logic is 55.90%, 10.45%, and 33.66% of the MPE area,

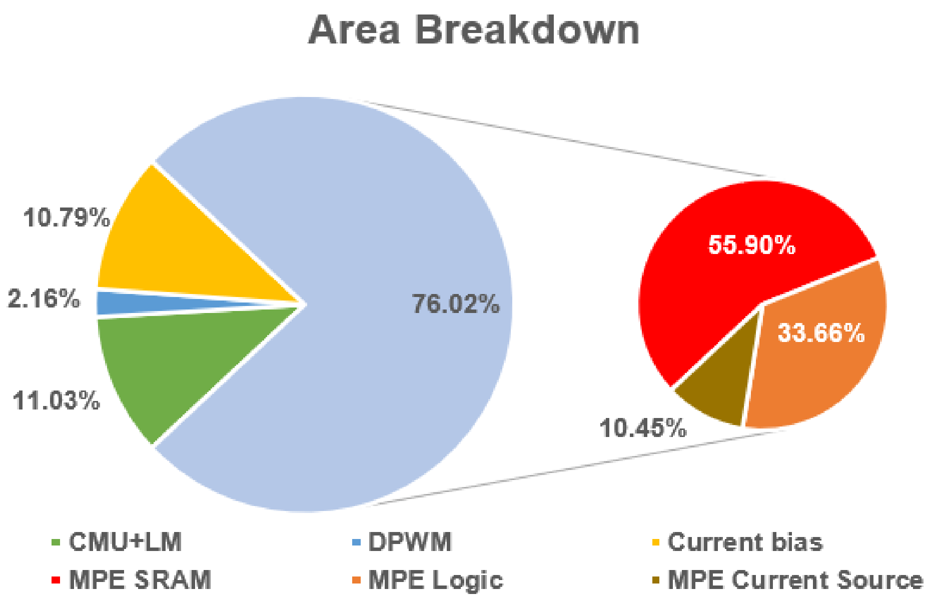


Figure 6.13. Area breakdown.

respectively. To improve the weight bit density of 0.307bit/µm2, increasing memory capacity and simplifying logic for an MPE is necessary. The area sum of a CMU and an LM was 6369 µm2, respectively, and the area of one DPWM was 656 µm2.

Chapter 7

Conclusion

This dissertation proposes a DNN accelerator hardware based on an MPE array that supports mixed-signal domain operations combining the advantages of accurate digital and energy-efficient analog domain operations. The proposed accelerator accurately computes the upper signed 4-bit, significantly affecting DNN accuracy among signed 9-bit inputs in the digital domain via an MPE column adder chain. The remaining lower unsigned 5-bit is modulated as pulse width with a DPWM and multiplied by weights in a bit-serial manner to generate a current sum on MPE bit-lines. Weight precision reconfigurability is implemented, and a cyclic MAC scheme is proposed to fix the number of analog-to-digital converted bits regardless of weight precision in a bit-serial manner.

The efficacy of the proposed DNN accelerator is verified by the results from a measurement environment, including a prototype IC fabricated with a 28 nm CMOS process and a Xilinx Kintex-7 FPGA KC705 evaluation kit. A 4-layer MNIST CNN example and a 5-layer CIFAR-10 CNN example were used for testing, and classification accuracy changes of -0.42 to 0.33%p were recorded when 300 calibration epochs were performed. In addition, the energy consumption of each layer decreased by 20.4 to 46.1% compared to the case with only digital calculations.

A technique of simultaneously calibrating all 16 × 16 current sources on an MPE array with a gradient descent training method was proposed to simplify calibration processes for improving DNN accuracy with analog operations. With Tensorflow simulation, the calibration process of 500 epochs could recover classification accuracy to 96.5-98.3% from defected accuracy of 17.3-95.5% by a process variation model added to MNIST CNN with an ideal accuracy of 98.2%.

A verification environment including three-level simulations was established for fast and accurate verifications for IC design steps. The system-level simulation built with the XMODEL simulator validates the functionality of the prototype IC for running DNN examples, consuming a short simulation time of 15 minutes per input image. In addition, the non-ideal characteristics obtained from analog circuit simulations were modeled on Tensorflow to predict the effect on DNN accuracy, and the acceptable range for each non-ideal characteristic could be determined early in the circuit design steps.

Bibliography

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in arXiv preprint arXiv:1409.1556, Sep. 2014.

[2] T. Luo, et al., "DaDianNao: A Neural Network Supercomputer," IEEE Transactions on Computers, pp. 73-88, May. 2016.

[3] Y. -H. Chen, et al., " Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," IEEE J. Solid-State Circuits, pp. 127-138, Apr. 2017.

[4] J. Lee, et al., "UNPU: An Energy-Efficient Deep Neural Network Accelerator with Fully Variable Weight Bit Precision," IEEE J. Solid-State Circuits, pp. 173-185, Jan. 2019.

[5] P. C. Knag, et al., " A 617-TOPS/W All-Digital Binary Neural Network Accelerator in 10-nm FinFET CMOS," IEEE J. Solid-State Circuits, pp. 1082-1092, Apr. 2021.

[6] S. Yin, et al., "An Energy-Efficient Reconfigurable Processor for Binary- and Ternary-Weight Neural Networks with Flexible Data Bit Width," IEEE J. Solid-State Circuits, pp. 1120-1136, Apr. 2019.

[7] J. Yue, et al., "STICKER-T: An Energy-Efficient Neural Network Processor Using Block-Circulant Algorithm and Unified Frequency-Domain Acceleration," IEEE J. Solid-State Circuits, pp. 1936-1948, Jun. 2021.

[8] M. S. Ansari, et al., "Improving the Accuracy and Hardware Efficiency of Neural Networks Using Approximate Multipliers" IEEE Trans. Very Large Scale Integr. (VLSI) Syst., pp. 317–328, Feb. 2020.

[9] A. Parashar et al., "SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks," ACM SIGARCH Computer Architecture News, pp. 27–40, May. 2017.

[10] J. -F. Zhang, et al., "SNAP: An Efficient Sparse Neural Acceleration Processor for Unstructured Sparse Deep Neural Network Inference," IEEE J. Solid-State Circuits, pp. 636-647, Feb. 2021.

[11] B. Moons, et al., "Envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural

network processor in 28nm FDSOI," in IEEE Int. Solid State Circuit Conf. (ISSCC), pp. 246–247, Feb. 2017.

[12] F. Tu, et al., "Evolver: A Deep Learning Processor with On-Device Quantization–Voltage–Frequency Tuning," IEEE J. Solid-State Circuits, pp. 658-673, Feb. 2021.

[13] D. Bankman, et al., "An Always-On 3.8 μJ/86% CIFAR-10 Mixed-Signal Binary CNN Processor with All Memory on Chip in 28-nm CMOS," IEEE J. Solid-State Circuits, pp. 158-172, Jan. 2019.

[14] D. Chang, et al., "Compact Mixed-Signal Convolutional Neural Network Using a Single Modular Neuron," IEEE Trans. on Circuits and Systems I: Regular Papers, pp.5189-5199, Dec. 2020.

[15] B. Murmann, "Mixed-Signal Computing for Deep Neural Network Inference," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., pp. 3–13, Jan. 2021.

[16] J. Zhang, et al., "In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array," IEEE J. Solid-State Circuits, pp. 915-924, Apr. 2017.

[17] A. Biswas and A. P. Chandrakasan, "CONV-SRAM: An Energy-Efficient SRAM With In-Memory Dot-Product Computation for Low-Power Convolutional Neural Networks," IEEE J. Solid-State Circuits, pp. 217-230, Jan. 2019.

[18] H. Jia, et al., "A Programmable Heterogeneous Microprocessor Based on Bit-Scalable In-Memory Computing," IEEE J. Solid-State Circuits, pp. 2609-2621, Sep. 2020.

[19] J. Yue, et al., "A 2.75-to-75.9TOPS/W Computing-in-Memory NN Processor Supporting Set-Associate Block-Wise Zero Skipping and Ping-Pong CIM with Simultaneous Computation and Weight Updating," in IEEE Int. Solid State Circuit Conf. (ISSCC), pp. 238–240, Feb. 2021.

[20] J.-W. Su, et al., "A 28nm 384kb 6T-SRAM Computation-in-Memory Macro with 8b Precision for AI Edge Chips," in IEEE Int. Solid State Circuit Conf. (ISSCC), pp. 250–252, Feb. 2021.

[21] J.-H. Kim, et al., "Z-PIM: A Sparsity-Aware Processing-in-Memory Architecture with Fully Variable Weight Bit-Precision for Energy-Efficient Deep Neural Networks," IEEE J. Solid-State Circuits, pp. 1093-1104, Apr. 2021.

[22] H. Jia, et al., "Scalable and Programmable Neural Network Inference

Accelerator Based on In-Memory Computing," IEEE J. Solid-State Circuits, pp. 198-211, Jan. 2022.

[23] K. Zia, et al., "Calibrating Process Variation at System Level with In-Situ Low-Precision Transfer Learning for Analog Neural Network Processors," in the 28th ACM/IEEE Design Automation Conf. (DAC), pp. 1–6, Jun. 2018.

[24] S. K. Cherupally et al., "Improving DNN Hardware Accuracy by In-Memory Computing Noise Injection," IEEE Design & Test, pp.1-8, Dec. 2021.

[25] S. K. Esser, et al., "Learned Step Size Quantization," in The Intl. Conf. on Learning Representations (ICLR), pp. 1–12, Apr. 2020.

[26] L. Deng, et al., "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey," Proceedings of the IEEE, pp. 485-532, Apr. 2020.

[27] A. N. Mazumder, et al., "A Survey on the Optimization of Neural Network Accelerators for Micro-AI On-Device Inference," IEEE J. Emerging and Selected Topics in Circuits and Systems, pp. 532-547, Dec. 2021

[28] E. Park, et al., "Value-aware Quantization for Training and Inference of Neural Networks," in European Conf. on Computer Vision (ECCV), pp. 580-595, Sep. 2018.

[29] L. Song, et al., "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in Pro. of IEEE Int. Symp. on High Performance Computer Architecture (HPCA), pp. 541-552, Feb. 2017.

[30] S. Ruder, "An Overview of Gradient Descent Optimization Algorithms," arXiv preprint arXiv:1609.04747, Sep. 2016.

[31] S. Baek and J. Kim, "A Gradient-Descent Calibration Method to Mitigate Process Variations in Analog Synapse Arrays," in Int. Conf. on Electronics, Information, and Communication (ICEIC), pp.1-4, Feb. 2022.

[32] Y. Lee, et al., "A 9-11 bits Phase-Interpolating Digital Pulse-Width Modulator with 1000X Frequency Range," in Pro. of IEEE Energy Conversion Congress and Exposition (ECCE), pp. 2172-2176, Nov. 2014.

[33] B. Murmann and B. E. Boser, "A 12-bit 75-MS/s Pipelined ADC Using Open-Loop Residue Amplification," IEEE J. Solid-State Circuits, pp. 2040-2050, Dec. 2003.

초 록

본 학위 논문은 혼합 신호 연산을 활용하여 심층신경망 연산기 하드웨어의 정확도 하락과 에너지 소모 증가를 완화하는 방법을 제시한다. 제안하는 하드웨어 구조는 16×16 혼합 신호 연산 유닛 어레이를 포함하며, 심층신경망 정확도에 큰 영향을 주는 입력의 앞 네 자리를 디지털 연산으로 구현하고 자주 발생해 에너지 소모가 큰 뒤 다섯 자리를 아날로그 연산으로 구현하여 입력 자릿수에 따라 불필요하게 소모하는 에너지를 줄이며 높은 심층신경망 정확도를 유지할 수 있다. 제안하는 혼합 신호 연산 유닛은 가중치의 비트를 직렬로 연산하여 다양한 가중치 자릿수 연산을 지원한다. 또, 출력의 자릿수를 고정하고 가중치 자릿수 별로 2 비트씩만 아날로그-디지털 변환을 수행하여 가중치 자릿수와 관계없이 동일한 변환 수를 유지할 수 있는 주기 연산 방식도 제안하였다.

제안한 하드웨어의 유효성은 28nm CMOS 공정으로 제작한 프로토타입 IC 과 Xilinx Kintex-7 FPGA KC705 보드를 포함하는 측정 환경에서 얻은 측정 결과를 통해 검증되었으며, IC 의 설계 과정은 세 가지 시뮬레이터를 조합하여 예제 단위의 수많은 연산을 빠르고 정확하게 처리할 수 있는 검증 환경을 통해 검증되었다. 검증에는 4-layer MNIST CNN, 5-layer CIFAR-10 CNN, 7-layer CIFAR-100 CNN 이 사용되었다. 별도로 제안된 경사하강법을 활용한 어레이 캘리브레이션 방식을 적용하여 프로토타입 IC 상에서의 예제 CNN 의 정확도를 측정하였을 때, -0.42~0.33%p 의 MNIST/CIFAR-10 예제의 작은 정확도 변화를 기록하였다. 각 심

층신경망 레이어 별 에너지 소모는 모든 연산을 디지털로 수행한 등가 상황과 비교하였을 때 20.4~46.1%만큼 줄어들었음을 확인하였다.

주요어 : 딥 뉴럴 네트워크, 신경망 연산, 신경망 연산 가속기, 아날로그 디지털 간 변환, 어레이 구조, 재구성 가능한 구조
학 번 : 2014-21726