creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

조선대학교
CHOSUN UNIVERSITY

February 2022
PhD Dissertation

Energy-efficient Online Arithmetic in Domain-Specific Accelerators for Deep Learning Applications

Graduate School of Chosun University

Department of Computer Engineering

Muhammad Usman

조선대학교
CHOSUN UNIVERSITY

Energy-efficient Online Arithmetic in Domain-Specific Accelerators for Deep Learning Applications

딥 러닝 가속기를 위한
에너지 효율적인 온라인 연산기 구조 설계

25, February, 2022

Graduate School of Chosun University

Department of Computer Engineering

Muhammad Usman

조선대학교
CHOSUN UNIVERSITY

Energy-efficient Online Arithmetic in Domain-Specific Accelerators for Deep Learning Applications

Advisor: Lee, Jeong-A

A dissertation submitted in partial fulfillment of the requirements for a Doctoral Degree

October, 2021

Graduate School of Chosun University

Department of Computer Engineering

Muhammad Usman

조선대학교
CHOSUN UNIVERSITY

우스만 무하마드 박사 학위 논문을 인준함

위원장 조선대학교 교 수 신석주 (인)

위원 조선대학교 교 수 강문수 (인)

위원 조선대학교 교 수 정호엽 (인)

위원 한림대학교 교 수 이정근 (인)

위원 조선대학교 교 수 이정아 (인)

2022년 01월

조선대학교 대학원

조선대학교
CHOSUN UNIVERSITY

Dedicated to my loving family.

조선대학교
CHOSUN UNIVERSITY

ACKNOWLEDGEMENTS

All praise is due to the Lord of the Worlds alone.

I would like to sincerely acknowledge my advisor Professor Lee, Jeong-A for all her technical, financial and personal support during my tenure at Chosun University. She has been tremendously supporting me and patiently guiding me during throughout these three years. In the hard times of pandemic, she always kept herself available and conducted countless online meetings upon my request at all times. I am grateful to her for trusting me to conduct this research when I did not have enough background of computer architecture field. I appreciate her for always being humble to answer all my trivial queries and sharing insightful ideas to lay the foundation of my dissertation.

I am grateful to all the committee members for their valuable feedback and comments. I treasure the joint work experience with my fellow lab members Umar Afzaal, Abdus Sami Hassan and specially my friend Shujaat Khan who has always been a great support and source of encouragement and inspiration for me.

Moreover, some of my friends including Ijaz Ahmed, SaidJalol Toshkujaev, Samsuddin Ahmed and Zohaib Nisar have helped me in many ways during my studies.

My prayers and regards to my father Muhammad Younas, whom I lost during my studies. He had been a constant support during his lifetime and I feel no hesitation to credit him for what I am today is just because of him.

조선대학교
CHOSUN UNIVERSITY

My heartfelt gratitude to my loving mother, Rukhsana Younas who made a difficult and selfless decision to let me go the foreign country for my studies. I thank my brothers and sisters for their love, support and continuous prayers for my success.

I would like to thank my wife Misbah Tariq, who firmly believe that I can be a good scholar, and constantly remind me of her confidence whenever I am in doubt. She always had been very patient and stayed beside me in the tough times without complaints. I hope I can support her as much as she did during my studies. Lastly, my son AbdulNafay and my daughter Sameera, the sparkles of my life and little bundles of joy, whose presence alleviated all the stress and worries in my life.

조선대학교
CHOSUN UNIVERSITY

TABLE OF CONTENTS

조선대학교
CHOSUN UNIVERSITY

조선대학교
CHOSUN UNIVERSITY

조선대학교
CHOSUN UNIVERSITY

LIST OF ABBREVIATIONS AND ACRONYMS

CA	Conversion/Append
CNN	Convolutional Neural Network
CSA	Carry-save adder
DNN	Deep Neural Network
LR	Left-to-Right
MSDF	Most Significant Digit First
r	Radix
SD	Signed-Digt
SOP	Sum-of-Product
RD	Repeated digit slices
SEL	Selection unit
SELM	Selection unit for multiplier
FPGA	Field programmable gate array
CLB	Configurable Logic Block
LUT	Look-up table
MUX	Multiplixer
TC	Two's complement
$\hat{v}$	Residual estimate
t	Minimum number of most significant fractional bits required for selecting output digit
w	Residual
ρ	Redundancy factor
δ	Online delay

조선대학교
CHOSUN UNIVERSITY

LIST OF FIGURES

조선대학교
CHOSUN UNIVERSITY

조선대학교
CHOSUN UNIVERSITY

LIST OF TABLES

조선대학교
CHOSUN UNIVERSITY

조선대학교
CHOSUN UNIVERSITY

ABSTRACT

Energy-efficient Online Arithmetic in Domain-Specific Accelerators for Deep Learning Applications

Muhammad Usman
Advisor: Prof. Lee, Jeong A
Department of Computer Engineering
Graduate School of Chosun University

In deep learning architectures, convolution, which is essentially sum-of-products, is a dominant operation and accounts for majority of computation. It is of great interest to minimize the resource consumption of convolution operation and reduce its latency to have a short response time of the network during inference.

This thesis is focused on the utilization of online arithmetic for the computation of inner products in the deep learning architectures. Online or left to right (LR) arithmetic executes in digit-serial manner in which inputs are provided and output is produced from most to least significant digit. It allows digit-level pipelining and early execution of successive operation regardless of data dependency which increases the overall throughput and decreases the latency. It has an inherent property to execute as a variable precision arithmetic where the execution can be stopped upon reaching the desired precision. Furthermore, for a given precision, the total number of digit slices required by the online algorithms are less than that required by the parallel implementations. Therefore, the working precision can be reduced to obtain a full precision result, thus minimizing the area occupancy, interconnects and signal activities.

조선대학교
CHOSUN UNIVERSITY

These properties of online arithmetic are explored to present a low-power pipelined online multiplier which along with the pipelined online adder is utilized to perform sum-of-products hardware unit. The cycle time of the pipelined online units is independent of data precision and is smaller compared to the conventional SoP designs.

The implementation of various precision multipliers and adders has been done using Verilog descriptions and functionally verified using ModelSim. The synthesis have been performed Synopsys Design Compiler on 45nm technology. Furthermore, the designs have been implemented on FPGA to observe the sequential and combinational logic utilization which show significant amount of savings in both area and power.

조선대학교
CHOSUN UNIVERSITY

한 글 요 약

딥 러닝 가속기를 위한 에너지 효율적인 온라인 연산기 구조 설계

우스만 무하마드
지도 교수: 이정아
컴퓨터공학과
조선대학교 대학원

딥 러닝 아키텍처의 대부분의 계산처리는 컨볼루션 연산으로, 궁극적으로 대량의 곱의합 연산 을 수행한다. 딥 러닝 추론 가속기에서 대량의 곱의합 연산을 수행하면서, 짧은 응답시간을 달성하기 위하여 초고속처리에 필요한 하드웨어 자원을 효율적으로 사용하는 문제는 중요한 문제이다.

본 논문은 딥 러닝 가속기를 위한 에너지 효율적인 온라인 연산기 구조 설계를 제시한다. 온 라인 연산으로 알려진 LR (Left 왼쪽에서 Right 오른쪽으로) 연산기법은 기존의 연산처리와 달리, 입력데이터가 LR 디지트 직렬(Digit-Serial) 방식으로, 즉 왼쪽(최상위 숫자)부터 오른쪽 (최하위 숫자)으로 제공되고, 출력도 디지트 직렬방식으로 최상위 숫자에서 최하위 숫자로 생 성된다. 온라인 연산은 최상위 숫자부터 출력하기 때문에, 현재 연산이 완료되지 않아도, 연속된 다음 연산을 시작할 수 있다. 즉, 연산 데이터 종속성에 관계 없이, 연속된 다음 연산을 시작할 수 있어서, Digit-level 파이프라이닝 이 가능하다. 연속 작업의 조기 실행이 가능하 여, 작업 처리량도 늘어나고, latency (처리지연시간)을 줄일 수 있어, response time (응답 시간)이 빨라진다. 온라인 연산은, 원하는 정밀도에 도달하면 실행을 중지할 수 있어서, 필요 한 만큼의 연산을 수행할 수 있는 고유한 속성이 있다. 온라인 연산 알고리즘은 입력 데이터 를 디지트 직렬 방식으로 받아서 출력 디지트를 결정하는데, 주어진 정밀도에 따라 결정되지

조선대학교
CHOSUN UNIVERSITY

만, 온라인 연산 알고리즘은 기존의 연산알고리즘과 달리, 출력 디지트를 결정하기 위하여, 전체 디지트를 다 필요로 하지 않는다. 연산 결과값의 precision(정밀도)보다 작은 작업 정밀도(working precision)만으로 동일한 결과 값을 얻을 수 있으므로, 회로의 크기를 줄일 수 있고, 모듈의 신호 활동을 최소화하여 에너지효율을 높일 수 있다.

본 논문에서는 온라인 연산기의 특성을 살려, 에너지 효율적인 파이프라인 온라인 곱셈기, 파 이프라인 온라인 가산기 구조 그리고 온라인 곱의합 연산기 구조를 제안하고 구현하였다. 파 이프라인 온라인 곱의합 연산기의 사이클 처리시간은 입력 데이터의 정밀도 (operand width) 에 따라 변화되지 않고, 동일하며, 기존의 곱의합 연산기 처리시간에 비하여 빠르기 때문에, 딥러닝 가속기에 적용할 경우의 그 효용성이 기대된다. 다양한 정밀도의 온라인 곱셈기와 가 산기를 Verilog로 구현하고, ModelSim을 사용하여 기능적으로 검증한 후에, Synopsys Design Compiler 를 사용하여 45nm 기술로 합성하였다. FPGA로 구현한 결과에서도 면적과 전력 모두에서 상당한 양의 절감이 가능함을 보였다.

조선대학교
CHOSUN UNIVERSITY

1 Introduction

1.1 Motivation

The development of sophisticated signal processing applications is growing with the passage of time and the scale of integration is increasing. The complex signal processing and machine learning applications demand high computational capacity and the power consumption and area utilization has become one of the major concerns for VLSI system design [1, 27]. There is an urgent need for the development of low power design because (i) the operating frequency is growing with a steady rate that increases the heat dissipation which must be lowered using effective cooling techniques, (ii) modern electronic devices are constrained by the limited power supply and are mostly battery operated. The low power design leads to longer life expectancy of these portable devices.

One of the fundamental operation used in the signal processing and deep learning algorithm is the computation of sum-of-products (SOPs) [25, 31]. It requires larger area, has longer latency and consumes significant power. Considerable amount of work has been done at the technological and circuit level towards development of low power multipliers [4, 23, 38, 39]. However, these generic techniques are not specifically opted for the multiplier design and can be utilizes in any other module. Machine learning algorithms specially the deep neural networks (DNNs) such as convolutional neural network (CNNs) employ massive inner product arrays and the consumption is directly related to the switching activities. It is therefore desirable to utilize the characteristics of arithmetic computations for designing low power operators for computing SOPs.

This dissertation provides the design and evaluation of online arithmetic based operators that are optimized in terms of interconnect and signal activities.

조선대학교
CHOSUN UNIVERSITY

Online arithmetic algorithms introduces parallelism with in the sequential operations because pipelining is possible regardless of data dependency. This greatly reduces the computation time and latency of lengthy sequences which is highly desirable in DNN applications. The model of LR arithmetic based multiplier and adder is presented alongwith the derivation of online algorithms, high level optimization have been carried out in the architecture and algorithm of conventional online operators to explore the opportunities of reducing power and area consumption. Moreover, low level optimization in the digit slices of the multiplier have been performed to minimize the interconnect and further reduce the power utilization.

1.2 Literature Review

The online arithmetic algorithm due to their unique properties have been widely adopted in a number of applications [7, 15, 22, 24]. Early research on the development of online algorithms for division and multiplication was carried out by Ercegovac [32]. The proof for the correctness of online division algorithm was presented in [33] along with two radix-4 online division algorithms. Later a simulator tool was developed for online arithmetic algorithms and presented in [28]. Ercegovac presented numerous online arithmetic based operations including square root, multiplicative vector normalization, computation of rotation factors, evaluation of polynomials etc. [13, 16, 34]. A method for computing fast multiplication without carry propagation array that suits well for the VLSI was presented in [11],

Methods to reduce the online delay have been proposed in [26] in which the algorithms are developed with two rather than one recurrence equation. One of

조선대학교
CHOSUN UNIVERSITY

the recurrence is for approximate computation to the additive or multiplicative inverse of the input variables and the other utilizes the approximation for generating the output. These algorithm have smaller online delay however, suffers from a longer step time for each digit.

Recent advancement in utilizing the online arithmetic were focused on the efficient implementation on reconfigurable devices such as FPGA. To this end a scheme for 3D vector normalization was proposed with its FPGA implementation in [18], which showed significant improvement in throughput when compared with the existing approaches. Galli and Alexandre proposed a design methodology for developing networks of online modules which highlighted the advantages of online arithmetic in designing complex signal processing algorithms and an efficient FPGA implementation was also shown [14]. In [30], the effect of overclocking the radix-2 online arithmetic algorithms was analyzed and the errors due to the timing violations were quantified. In [41], a method for achieving variable precision was presented which used on-chip block RAMs for the storage of intermediate values. The most recent work to utilize the online arithmetic is named as ARCHITECT by [20], which propose the method of computing arbitrary precision result in arbitrary iterations and showed $16x$ performance speedup and $1.9x$ savings in memory.

The use of online arithmetic for computing the inner products of the convolutional neural networks has not studied well. The proposed work could serve as an initiation towards exploration of online arithmetic based algorithms in the machine learning applications.

조선대학교
CHOSUN UNIVERSITY

1.3 Research Approach

The algorithms developed using online or left to right (LR) arithmetic execute serially taking most significant digit first (MSDF). The precision of inputs increases one digit in each iteration/cycle. Therefore, it is possible to gradually increase the digit slices in accordingly. This approach allows to reduce the signal activities consequently reduction in both dynamic and static power can be observed. Online arithmetic allows to reduce the maximum working precision and therefore, the area utilization of the algorithm which is often increased due the use of redundant number system is circumvented. Using these properties, we propose a low power pipelined online multiplier. The unused modules in a particular pipeline stage are not implemented, which minimizes the slice and switching activities leading to savings in both area and power. The low power pipelined online multiplier is utilized with a pipeliend online adder to form a pipelined sum-of-product unit to have an increased throughput which is desired in the computation of massive inner products in CNN applications.

1.4 Contributions

The efficacy of online arithmetic algorithms have been well debated and theoretical estimation of possible savings have been derived. However, none of the work has truly utilized the power reduction properties of online arithmetic to implement on a hardware and observe the actual savings. In this work we present:

- The conventional algorithm of online multiplier and adder has been modified with respect to the input and output generation.
- A 2D array implementation has been proposed in which the algorithm's

조선대학교
CHOSUN UNIVERSITY

steps are pipelined for increased throughput.

- A digit-level pipelined sum-of-product unit is proposed for the computation of inner products and their summation using an adder tree.
- Maximum working precision has been reduced in the online multiplier such that only required number of modules are instantiated in each step.
- Hardware implementation of low power online multiplier and adder with details of internal circuitry of each module involved in the algorithm has been presented.
- The proposed and conventional designs have been developed using technology-independent Verilog descriptions.
- The Verilog designs are subjected to area and power estimation using Synopsys Design Compiler on 45nm technology.
- The designs are implemented on Xilinx FGPA to compare the combinational and logical area utilization.

1.5 Organization of Thesis

The research work presented in the thesis is organized and structured in the form of five chapters, which are briefly described as follows:

i) Chapter 2 provides a formal introduction of online arithmetic. The differences between online and conventional algorithms in terms of timing and structures are highlighted. The basic components and method for developing online algorithm is presented. The method to develop online multiplier and online adder is also discussed.

조선대학교
CHOSUN UNIVERSITY

ii) **Chapter 3** presents the pipelined implementation of online multiplier and online adder and their combination to develop a digi-level pipelined online sum-of-product unit.

iii) **Chapter 4** presents a review of the contemporary hardware accelerators for CNN applications and compares the performance of online SoP in one of these accelerators.

iv) **Chapter 5** highlights the summary and conclusions of thesis.

조선대학교
CHOSUN UNIVERSITY

2 Overview of Online Arithmetic

2.1 Introduction

Online arithmetic or left-to-right algorithms execute serially where the input operands arrive and output is produced digit by digit (serially) most significant digit first (MSDF). Unlike conventional arithmetic computing right to left and requiring full precision input, the first output digit of the online algorithm is generated after a fixed and small initial delay known as the online delay δ. The online delay can be defined as the minimum number of input digits that must be accumulated in order to generate the first digit of output. To be able to generate output on the basis of partial input information, the online algorithms employ redundant number systems to have flexibility and opportunity to correct the upcoming output digit if the preceding output is making the overall result incorrect.

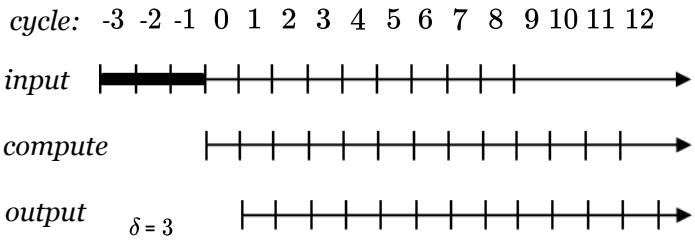

Figure 2.1: Timing depiction of online arithmetic based algorithms with online delay $\delta = 3$.

The working principle of online algorithm is shown in Fig. 2.1. To calculate j^{th} output digit, $j + \delta + 1$ digits of the input operands are required.

조선대학교
CHOSUN UNIVERSITY

2.2 General Properties

Some of the properties of using online arithmetic are listed below:

- The digit level pipelining is possible when employing online arithmetic which helps in execution of parallel execution of arithmetic operation. The execution of successive operation can be started after an online delay suggesting that the online algorithms do not have to wait for the full precision input unlike conventional arithmetic which has to wait for full precision input to start computation as shown in Fig. 2.2.

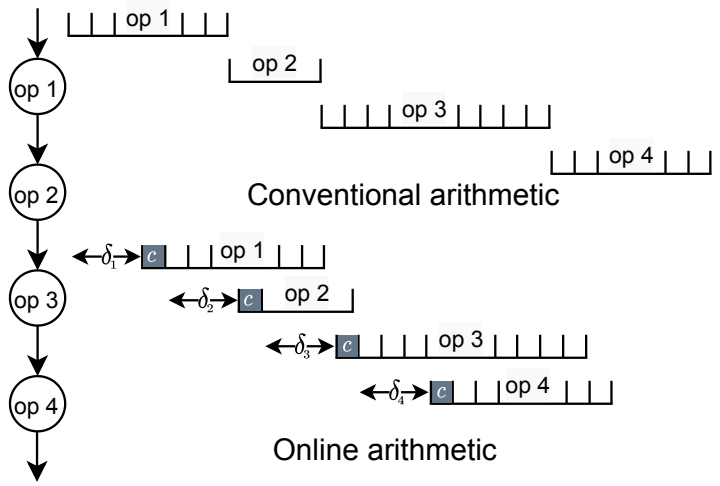

Figure 2.2: Timing comparison of conventional and online arithmetic for sequence of operations assuming $\delta_i = 3$ and a compute cycle $c = 1$ for each operation. Conventional arithmetic has to wait for the completion of previous computation. Using online arithmetic, the successive operation can be started regardless of the data dependency as soon as δ_i digits result of the previous operation have been produced.

- Online arithmetic uses digits in serial manner which reduces the communication lines to give the inputs and the interconnections between

조선대학교
CHOSUN UNIVERSITY

the modules are also reduced as the same set of lines are used to transfer the data. This property leads to the reduction in the area occupied by the interconnecting wires in the parallel arithmetic as well eases the process of routing.

- The output is computed while communication i.e., the output digits are produced while the input digits are still arriving. The generation of the output on the basis of low precision input requires the use of flexibility in the number system which is achieved by employing redundant number systems.

- The full precision output can be obtained by employing fewer digit slices than the conventional parallel arithmetic. The effect of reduction of total number of digit slices reflects in the reduction of switching activity and number of active digit slices. Thus providing the room for savings in both area and power.

2.3 Number System

Usually the online arithmetic operators use fractional numbers to make them compatible with other operations and simplify the alignment. Therefore, the weight of the first digit of the operand is r^{-1}, where r is the radix. The output is computed on the basis of partial information about the inputs, therefore, in case of incorrect digit selection in the most significant part of the result, correction can be applied in the lower significant digit. This is achieved with the use of redundant number systems which allows to represent a number in more than one way. The use of redundant number system results in an increase of cost per bit,

조선대학교
CHOSUN UNIVERSITY

however, the need of carry propagation is eliminated which results in an overall advantage. Commonly used redundant number system are the signed digits (SD). The number is represented in the form of radix where each digit belongs to a set $\{-a,...,-1,0,1,...,a\}$ and $\frac{r}{2} \leq a < r$. The amount of redundancy is denoted by ρ ($\rho = \frac{a}{r-1}$). In this work we employ radix-2 SD numbers to represent the input and output digits belonging to a symmetric digit set $\{-1,0,1\}$.

The representation of digit x_j in a given iteration j uses two bits x_j^+ and x_j^-, and the subtraction of these two bits represents the value of the given digit as shown in relation (2.1).

$$x_j = SUB(x_j^+, x_j^-) \tag{2.1}$$

2.4 Implementation Model

All the online algorithms are composed of similar computation modules and can be designed using a generic model, making their development and implementation a lot easier than the contemporary algorithms. Some of the basic components for the implementation are listed below:

1. Registers are required to store the values of incoming input, residual and the output.

2. The precision of the input is increased gradually in each step, accordingly appending units are required to append the incoming input with the already stored input vector.

3. Multiplication of a vector by a digit, which is often carried out using a multiplexer.

조선대학교
CHOSUN UNIVERSITY

4. Multi-operand adders are required which can be either signed digit or carry save type depending on the number system representation.

5. Converters to convert the redundant numbers such as signed digit or carry save form to conventional representation. This conversion is carried out using on-the-fly converters (OTFC) which are based on simple logical functions and shift registers, therefore, do not add any delay.

6. Output is generated by a selection function for which digit selection units are required.

7. The digit selection function uses an estimate of the residual which is computed using carry-propagate adders of limited precision (3 to 6 bits).

The digit slice organization of an online unit is shown in Fig. 2.3.

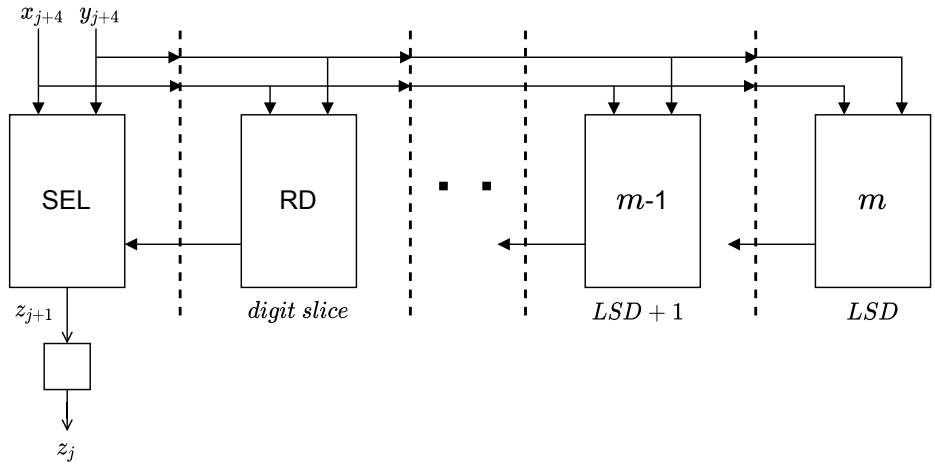

Figure 2.3: Digit slice organization of online arithmetic unit ([10]) SEL refers to the selector unit, RD being the repeated digit slices.

The cycles or steps are labeled from $-\delta, \ldots 0, 1, \ldots, n-1$ where at step j, the

조선대학교
CHOSUN UNIVERSITY

input digits $x_{j+1+\delta}$ and $y_{j+1+\delta}$ are received from the most significant side, an output digit z_{j+1} is computed and z_j is delivered as shown in Fig. 2.1.

The algorithms requires the conversion of input digits to the conventional two's complement (TC) number to have their numerical values to update the recurrence and compute the internal state of the algorithm termed as residual (ω). The numerical values of a digits at iteration j is denoted as $x[j]$, $y[j]$ and $z[j]$ and its corresponding online form are given as follows:

$$\begin{aligned}
x[j] &= \sum_{i=1}^{j+\delta} x_i r^{-i} \\
x[j+1] &= x[j] + x_{(j+1+\delta)} r^{-(j+1+\delta)} \\
y[j] &= \sum_{i=1}^{j+\delta} y_i r^{-i} \\
y[j+1] &= y[j] + y_{(j+1+\delta)} r^{-(j+1+\delta)} \\
z[j] &= \sum_{i=1}^{j} z_i r^{-i} \\
z_{j+1} &= F(w[j], x[j], x_{j+\delta+1}, y[j], y_{j+\delta+1}, z[j]) \\
z[j+1] &= (z[j], z_{j+1}) \\
w[j+1] &= G(w[j], x[j], x_{j+\delta+1}, y[j], y_{j+\delta+1}, z[j], z_{j+1})
\end{aligned} \tag{2.2}$$

2.5 Method for Developing Online Algorithms

The development of online algorithms are composed of two fundamental parts. First is the definition of residual and its recurrence, while the second part is the definition of digit selection function that determines the output digit.

조선대학교
CHOSUN UNIVERSITY

2.5.1 Residual and its Recurrence

The online arithmetic algorithms rely on the recursive computation of residual (w) at each step; generating an output is by a digit selection function which selects an output on the basis of current value of residual. Since the output generation uses partial information of the inputs, it is required to define an error bound. For an operation f, the bound is shown in (2.3) on the recurrence which is maintained at each step when the residual is updated.

$$|f(x[j],y[j]) - z[j]| < r^{-j} \tag{2.3}$$

Transformation is applied on (2.3), such that the recurrence contains the primitive operations only. The scaled residual for the multiplier can be defined as:

$$w[j] = r^j(G(f(x[j],y[j]) - z[j])) \tag{2.4}$$

with $|w[j]|$ being bounded by $\omega = \rho(1-2r^{-\delta})$. The residual, $w[j]$, in the redundant carry-save form actually has a two's complement representation and is represented by the vectors $ws[j]$ and $wc[j]$. The residual can be deduced to obtain the recurrence $w[j+1]$ having bounds $\overline{\omega}$ and $\underline{\omega}$:

$$\begin{aligned} w[j+1] = rw[j] + r^{j+1}(&G(f(x[j+1],y[j+1]) - z_{j+1}) \\ &- G(f(x[j],y[j]) - z[j])) \end{aligned} \tag{2.5}$$

The recurrence is decomposed into:

$$v[j] = rw[j] + H_1 \tag{2.6}$$

$$w[j+1] = v[j] + H_2(z_{j+1}) \tag{2.7}$$

such that (2.6) is independent of z_{j+1}.

조선대학교
CHOSUN UNIVERSITY

2.5.2 Selection Function with Selection Constants

The output digit is selected using a selection function such that the residual $w[j+1]$ remains bounded between $\overline{\omega}$ and $\underline{\omega}$. The selection of the output digit $q_{j+1} = k$ where $k = -a, -a+1, \ldots, a$ depends upon the selection intervals of $v[j]$. However, in actual implementation, only t bits of $v[j]$ are truncated to give its estimate, $\widehat{v}[j]$. The corresponding selection intervals $\widehat{U}_k \leq \widehat{v}[j] \leq \widehat{L}_k$ are given by:

$$\begin{aligned} \widehat{U}_k &= \left\lfloor \rho(1-2^{-\delta}) + k - 2^{-t} \right\rfloor_t \\ \widehat{L}_k &= \left\lceil -\rho(1-2^{-\delta}) + k \right\rceil_t \end{aligned} \tag{2.8}$$

The simplified range of the estimate $\widehat{v}[j]$ is shown in relation (3.4):

$$\left\lfloor -\rho(r-2^{-\delta}) - 2^{-t+1} \right\rfloor_t \leq \widehat{v}[j] \leq \left\lfloor \rho(r-2^{-\delta}) \right\rfloor_t \tag{2.9}$$

2.6 Online Multiplier

Now we present the design methodology of an online arithmetic based multiplier and in particular the derivations of the recurrence and selection function will be presented for the radix-2 online multiplier. The input operands x and y in the signed digit redundant representation are computed to produce the product digit z ranging from $(-1,1)$ from the symmetric signed digit set $\{-a, ..., a\}$. The operands and the resulting product digit at iteration j are given as:

$$x[j] = \sum_{i=1}^{j+\delta} x_i r^{-i}, \quad y[j] = \sum_{i=1}^{j+\delta} y_i r^{-i}, \quad z[j] = \sum_{i=1}^{j} z_i r^{-i}, \tag{2.10}$$

At each iteration j, a SD input is received, which is converted to two's complement representation in digit serial manner using on-the-fly

조선대학교
CHOSUN UNIVERSITY

conversion/append (CA) function as: $x[j] = CA(x[j-1], x_{j+4})$ and $y[j] = CA(y[j-1], y_{j+4})$. An output is produced on the basis of only partial information of the inputs, therefore, an error bound must be defined as follows:

$$|x[j] \cdot y[j] - z[j]| < r^{-j} \tag{2.11}$$

The above relation is subjected to a transformation function to develop the recurrence having primitive functions only, which is then scaled by a factor to have a bound on the error after the computation of j digits. The corresponding scaled residual is given by:

$$w[j] = r^j(x[j] \cdot y[j] - z[j]) \tag{2.12}$$

The residual can be deduced to obtain the recurrence $w[j+1]$:

$$\begin{aligned} w[j+1] = rw[j] + (x[j]y_{j+1+\delta} + y[j+1]x_{j+1+\delta})r^{-\delta} \\ - z_{j+1} \end{aligned} \tag{2.13}$$

The recurrence is decomposed into:

$$\begin{aligned} v[j] &= rw[j] + (x[j]y_{j+1+\delta} + y[j+1]x_{j+1+\delta})r^{-\delta} \\ w[j+1] &= v[j] - z_{j+1} \end{aligned} \tag{2.14}$$

resulting in

$$H_1 = (x[j]y_{j+1+\delta} + y[j+1]x_{j+1+\delta})r^{-\delta} \quad H_2 = -z_j + 1 \tag{2.15}$$

The simplified range of the estimate $\widehat{v}[j]$ is shown in relation (2.16):

$$\left\lfloor -\rho(r - 2^{-\delta}) - 2^{-t+1} \right\rfloor_t \leq \widehat{v}[j] \leq \left\lfloor \rho(r - 2^{-\delta}) \right\rfloor_t \tag{2.16}$$

조선대학교
CHOSUN UNIVERSITY

2.7 Radix-2 Online Multiplier

The multiplication algorithm for radix-2 online multiplier is now presented. The online delay for this multiplier is $\delta = 3$ and the number of most significant fractional bits required in the selection function to select the output are $t = 2$. The recurrence and the selection function for this multiplier is shown now.

2.7.1 Recurrence

The derivation of the recurrence is similar to what presented in section 2.6. For $r = 2$ and $\delta = 3$, $v[j]$ in (2.14) can be rewritten as:

$$v[j] = 2w[j] + (x[j]y_{j+4} + y[j+1]x_{j+4})2^{-3} \tag{2.17}$$

2.7.2 Selection Function

The output digit is selected using a selection function such that the residual $w[j+1]$ remains bounded. The selection of the output digit $z_{j+1} = q$ where $q_i = -a, -a+1, \ldots, a$ depends upon the selection intervals of $v[j]$. Only t most significant fractional bits along with integer bits (*ibs*) of $v[j]$ are used from the result generated by the $[4:2]$ adder in carry-sum pair (WS and WC) to give its estimate, $\widehat{v}[j]$, the range of which is defined as:

$$-2 \leq \widehat{v}[j] \leq \frac{7}{4} \tag{2.18}$$

In case of radix-2 online multiplier, the least significant estimate bit v_2 is not used and the three 3 MSBs i.e., two *ibs* (v_{-1} and v_0) and one fractional bit (v_1) are sufficient to select the output z_{j+1}. The corresponding selection function (SELM) for δ = 3, r = 2 and t = 2 is given as:

조선대학교
CHOSUN UNIVERSITY

Table 2.1: Selection function for radix-2 multiplier.

$\hat{v}$	$v_{-1}v_0.v_1$	z_{j+1}
3/2	00.1	1
1	01.0	1
1/2	00.1	1
0	00.0	0
-1/2	11.1	0
-1	11.0	-1
-3/2	10.1	-1
-2	10.0	-1

$$z_{j+1} = SELM(\hat{v}[j]) = \begin{cases} 1 & \text{if } 1/2 \leq \hat{v}[j] \leq 7/4 \\ 0 & \text{if } -1/2 \leq \hat{v}[j] \leq 1/4 \\ -1 & \text{if } -2 \leq \hat{v}[j] \leq -3/4 \end{cases} \tag{2.19}$$

The product digit z_{j+1} uses similar coding as (2.1) and corresponding selection function is shown in Table. 2.1.

2.7.3 Algorithm

The algorithm of a conventional radix-2 online multiplier has been shown in Algorithm. 2.1. During initialization, δ input digits are received from most significant side and the residual is updated. The output generated during this phase are discarded. During recurrence, the remaining $n - \delta$ input digits are received and an output is generated in each cycle. For the last δ cycles the inputs are set to be zero. The multiplication of terms with 2^{-3} in (2.17) is carried out by arithmetic right shift by 3.

조선대학교
CHOSUN UNIVERSITY

Algorithm 2.1 Online Multiplication

Initialize:

$x[-3] = y[-3] = w[-3] = 0$

for j=$-3,-2,-1$ **do**

$x[j+1] \leftarrow CA\left(x[j], x_{j+4}\right);\ y[j+1] \leftarrow CA(y[j], y_{j+4});$

$v[j] = 2w[j] + \left(x[j]y_{j+4} + y[j+1]x_{j+4}\right)2^{-3}$

$w[j+1] \leftarrow v[j]$

end for

Recurrence:

for $j = 0 \ldots n-1$ **do**

$x[j+1] \leftarrow CA\left(x[j], x_{j+4}\right);\ y[j+1] \leftarrow CA(y[j], y_{j+4});$

$v[j] = 2w[j] + \left(x[j]y_{j+4} + y[j+1]x_{j+4}\right)2^{-3}$

$z_{j+1} = SELM(\widehat{v}[j])$

$w[j+1] \leftarrow v[j] - z_{j+1}$

$Z_{\text{out}} \leftarrow z_{j+1}$

end for

조선대학교
CHOSUN UNIVERSITY

2.7.4 Block Diagram

The block diagram of radix-2 online multiplier has been shown in 2.4. The received inputs are converted to conventional form using OTFC and stored in registers which are shown as carry/append (CA) registers . The residual, $w[j]$, in the redundant carry-save form actually has a two's complement representation and is represented by the vectors $WS[j]$ and $WC[j]$. The estimate of the residual is calculated in the V block and the calculation of the updated residual $w[j+1]$ which requires subtraction of z_{j+1} from $v[j]$ is carried out by the M block. The subtraction is performed using a Boolean expression rather than explicit subtraction [8].

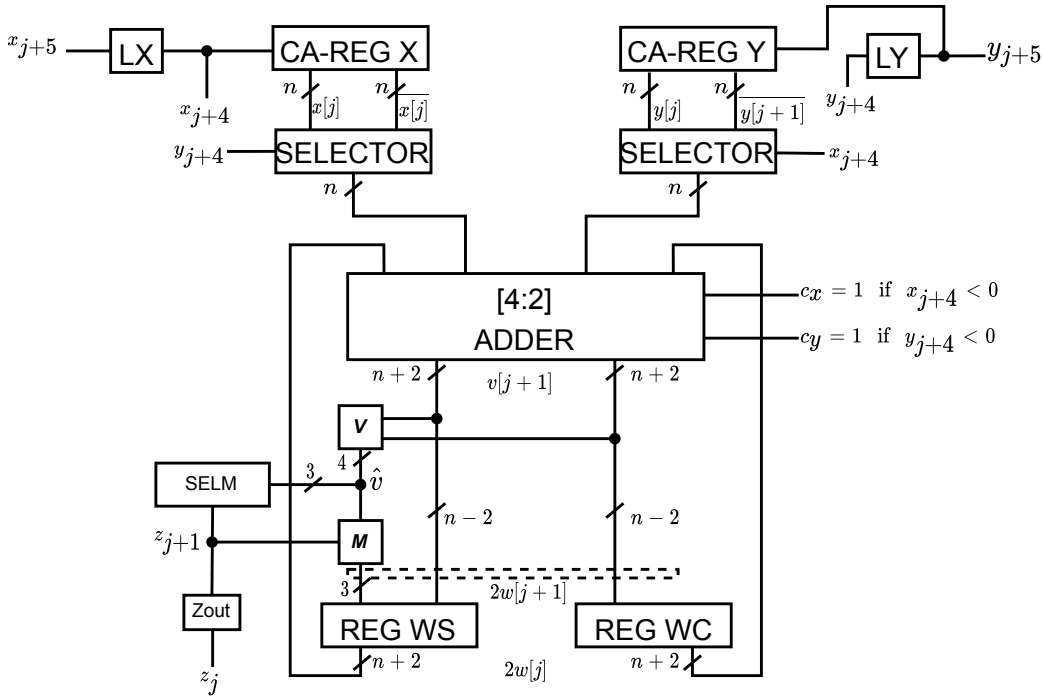

Figure 2.4: Implementation level diagram for radix-2 online multiplier ([10])

조선대학교
CHOSUN UNIVERSITY

2.8 Radix-2 Online Adder

The online adder is obtained from the serialization of a redundant adder. A radix-2 online adder shown in Fig. 2.5 is composed of two full adders and 5 flip-flops. The cycle time corresponds to the delay of one digit radix-2 signed digit adder plus the loading of registers. The online delay of radix-2 online adder is $\delta = 2$.

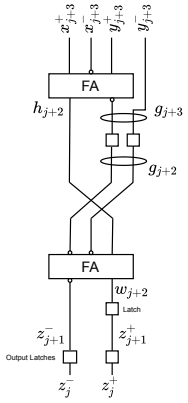

Figure 2.5: Radix-2 online adder

2.9 Chapter Summary

This chapter developed the understanding of online arithmetic and online algorithms. The difference of execution and output generation between online and conventional arithmetic was shown. Basic components of online algorithms were described and methods of developing a typical online algorithms with derivation was described. In particular, the derivation of recurrence and selection function of a radix-2 online multiplier and adder has been presented. The algorithm and block diagram of the radix-2 online multiplier and adder was also presented.

조선대학교
CHOSUN UNIVERSITY

3 Online Sum-of-Product

3.1 Introduction

In recent years convolutional neural network (CNN) which is a type of deep learning have been widely used in variety of fields including image recognition and natural language processing. The CNN is greatly influenced by the number of convolution layers which essentially perform dot products and summation i.e., sum-of-products. Many methods have been proposed to accelerate the computation of convolution as they constitute 90% of computation time [6]. One way to accelerate the CNN computation is to use approximation; which however, results in accuracy degradation. In this work, we propose to use pipelined online sum-of-product (SoP) to perform the convolution operation to take advantage of online arithmetic discussed in Chapter. 2. The pipelined online SoP has been developed by realizing digit-level pipelining in the low-power online multiplier and online adder, the architectures of which are discussed in the forthcoming sections.

3.2 Pipelined Online Multiplier

It is established that the precision of the inputs to the multiplier increases gradually. Consequently the slice activities also increases one digit in each iteration. However, the conventional online multiplier has constant slice activity, i.e., all the n bit slices are activated during all iterations. This corresponds to an unnecessary power supply to the slices which do not contribute to the computation in a certain iteration. It has proposed to use power gating in which the unused slices are remained off until they are required for computation. To

조선대학교
CHOSUN UNIVERSITY

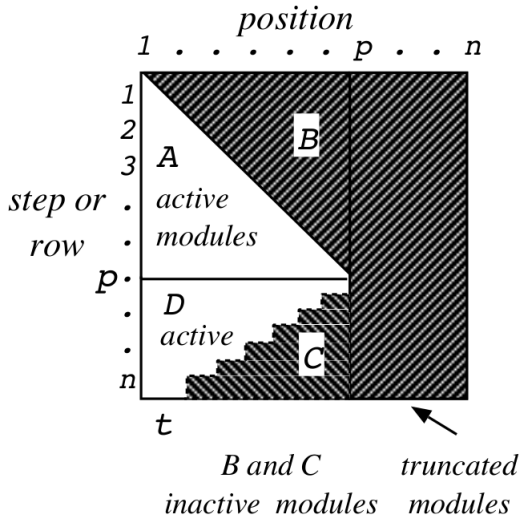

Figure 3.1: Profile of slice activity [9].

which end, extra control circuitry has to be employed which results in an increase in area as well as the dynamic power remains.

Another method is to implement the design as a 2D array and modules that are not required in a certain iteration are not implemented and neither dynamic nor static power is consumed. Furthermore, the n precision result can be obtained by using $p < n$ digit slices [10]. In [9], a depiction of such scheme has been presented which is shown in Fig. 3.1. After p cycles, truncation is applied that introduces an error which propagates in the residual for next cycle. Therefore, the erroneous digit slices can be turned off in the linear implementation or in the 2D implementation they can be entirely removed leading to significant savings in area and power.

조선대학교
CHOSUN UNIVERSITY

Table 3.1: Depiction of the input and output dataflow in the digit-level pipelined online multiplier processing streams of input.

Execution Cycle	Input				Output	
1	x_n^1	y_n^1	-	-	-	-
2	x_{n-1}^1	y_{n-1}^1	x_n^2	y_n^2	-	-
3	x_{n-2}^1	y_{n-2}^1	x_{n-1}^2	y_{n-1}^2	-	-
4	x_{n-3}^1	y_{n-3}^1	x_{n-2}^2	y_{n-2}^2	z_n^1	-
5	x_{n-4}^1	y_{n-4}^1	x_{n-3}^2	y_{n-3}^2	z_{n-1}^1	z_n^2
6	x_{n-5}^1	y_{n-5}^1	x_{n-4}^2	y_{n-4}^2	z_{n-2}^1	z_{n-1}^2
.	.	.	x_{n-5}^2	y_{n-5}^2	.	z_{n-3}^2
.	.	.	.	.	.	.
n	x_0^1	y_0^1	.	.		.
n+1	0	0	x_0^2	y_0^2	z_3^1	
n+2	0	0	0	0	z_2^1	z_3^2
n+δ	0	0	0	0	z_1^1	z_2^2
n+δ+1	0	0	0	0	z_0^1	z_1^2
n+δ+2	0	0	0	0	-	z_0^2

3.2.1 Working Principle of Pipelined Multiplier

As an example two streams on inputs X^1,Y^1 and X^2,Y^2 of n-bits are multiplied using pipelined online multiplier to generate the products Z^1 and Z^2. The inputs X^1,Y^1 are fed to the multiplier and after one cycle, the inputs X2,Y2 are supplied. Likewise the MSD of Z^1 is received after the online delay and the MSD of Z^2 is obtained one cycle later. It takes n+δ cycles to fill the pipeline, after which a complete output vector can be obtained in each cycle resulting in an increased throughput. The dataflow for the digit-level pipelined multiplier is depicted in Table. 3.1.

조선대학교
CHOSUN UNIVERSITY

3.2.2 Maximum Working Precision Reduction

In the conventional design, the working precision of n bits is constant and all digit slices have similar circuit in each iteration. However, in the proposed design, we explore the sources of reducing the active modules which includes gradual use of the input digits and reduction of working precision to $p < n$; defining a dynamic working precision in stage k, as $1 \leq k \leq p$ as the iterations are performed.

3.2.3 Working Precision Reduction Strategy

The output digit of the online algorithm is based on a selection function which utilizes a few most significant bits of the residual comprised of integer and t fractional bits as discussed in section 2.7.2, to obtain the residual's estimate denoted by $\hat{v}$. Therefore, it is possible to achieve n bits accuracy by implementing only p ($p < n$) bit slices and ignoring a few least significant h bit slices.

For $j \leq p$ iterations, j modules are active in j^{th} recurrence step; whereas, for $j > p$, the availability of only p modules introduces an error due to truncation. The algorithm's convergence can be assured if the t bits in the selection function are not affected due to the truncation error. The optimum number of p varies according the type of adder used in the recurrence equation, the number of ignored bit slices h and the initial delay δ. For a valid selection in an online multiplier with $[4:2]$ adder, relation (3.1) has been suggested in [10]:

$$p = \left\lceil \frac{2n + \delta + t}{3} \right\rceil \tag{3.1}$$

Due to the gradual increase in the precision of the incoming digits, the signal activity is not constant and increases gradually in each iteration. Furthermore, if $p < n$ slices are implemented for the given multiplier, the signal activity begins to

조선대학교
CHOSUN UNIVERSITY

decrease after p iteration due the truncation error which affects $(j-2)^{th}$ result bit and is shifted one bit towards left due to the left shift operation in the recurrence. Overall, the error propagates to 3 bit slices, therefore, the 3 least significant bit slices can be turned off in the subsequent stage of the pipeline.

Based on these properties, $8, 16, 24$ and 32 bit designs of the full precision online multiplier have been compared with the respective low-powered designs. According to relation (3.1), the n precision result can be obtained by employing $7, 12, 18$ and 23 modules for $8, 16, 24$ and 32 bit designs respectively. In this correspondence, the conventional online multiplier algorithm shown in Fig. 2.4, and the proposed low power designs have been implemented as a two-dimensional pipeline array, where the bit widths of the registers (*CA-Reg*, *Reg WS*, *Reg WC*), adder and selector are increased till p iteration and then decreased till $n+\delta$ iteration.

3.2.4 Pipelining

The conventional implementation has its throughput limited by its latency and produces one vector in n cycles. In the deep learning applications where large number of multiplications have to be performed, this limitation on the throughput may not be acceptable. Therefore, to process large number of convolution operations in the deep learning applications, it is suitable to unfold and pipeline the multiplier. For n-bit precision, n stages of the multiplier are unfolded and pipelined. The pipeline not only allows the computation of n different vectors in the same hardware, but also the subsequent operations including addition and activation function can be started immediately upon receiving the first MSD of result. This can drastically improve the throughput of the network and decrease

조선대학교
CHOSUN UNIVERSITY

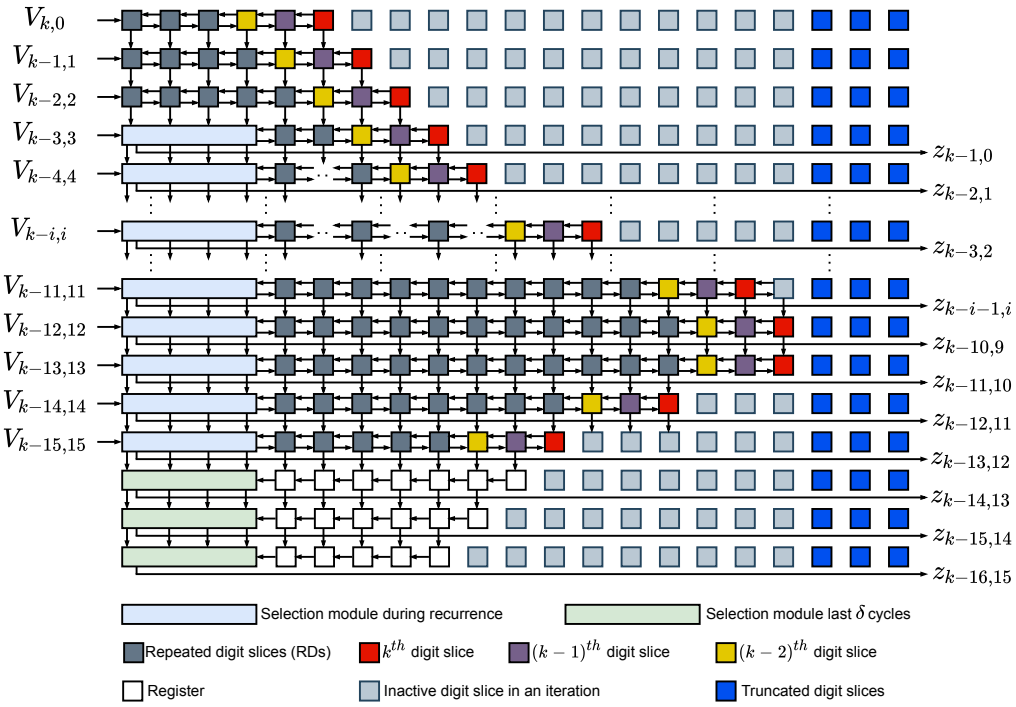

Figure 3.2: Signal activity of radix-2 online multiplication algorithm with maximum truncated precision of p with $\delta = 3$, 2 integer bits (ib) and $t = 2$. Different colors of the digit slice refer to their distinct structure. SEL block evaluates three bits to compute the output and is therefore larger than the rest of digit slices.

조선대학교
CHOSUN UNIVERSITY

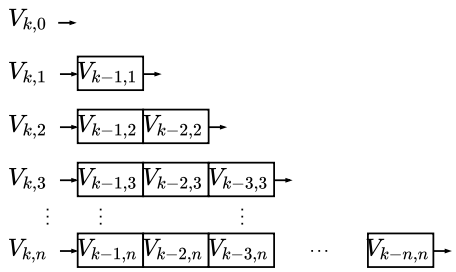

Figure 3.3: Stair-case input shifter array [18].

its latency. As discussed in section 3.2.3, the input bit precision is increased gradually and $p < n$ modules are sufficient to produce n-bit precision result, only the required number of modules can be activated upto p^{th} iteration and after truncation in $(p+1)^{th}$ iteration, the modules can be turned off according to the error profile. In a pipelined scheme however, the inactive modules are not implemented, hence no dynamic/static power is consumed.

A 16-bit piplelined scheme which is a two-dimensional array structure with 16 stages has been depicted in Fig. 3.2. The output digit selection module in the most significant place is instantiated after initialization steps to generate an output digit and the residual signals are transferred vertically to the subsequent linear array instead of left shifting as in the conventional implementation. The input vectors are arranged in a stair-case manner to match the pipeline online flow using a stair-case shifter array shown in Fig. 3.3, which simply adds a delay in the i^{th} digit of a vector using a i-bit shift register [18].

The details of each digit slice has been presented in the forthcoming section.

조선대학교
CHOSUN UNIVERSITY

3.2.5 Algorithm

The conventional algorithm has two steps, however, in the proposed design the algorithm has been divided into three steps [35], including, (1) initialization: having execution length equal to δ, during which the input digits are collected and no output is generated, (2) recurrence: which executes for $n - \delta$ iterations, producing one output digit in each iteration. (3) last δ cycles: having execution length equal to δ, during which the input digits are *zero* and output is generated in each iteration. The pseudocode of the conventional radix-2 online multiplier presented in [10] with initialization and recurrence loops shown in 2.1, has been modified to have three phases as shown in Algorithm 3.1.

3.2.6 Block Diagram

The block diagram of radix-2 online multiplier has been shown in 3.4. Since the implementation is done in 2D array form, the length of each module is increased one digit in each step. During initialization, no output digit is produced consequently, the modules responsible for producing output are not instantiated. The corresponding block diagram for this phase is shown in Fig. 3.4(a). After obtaining sufficient input digits, the algorithm proceeds to the recurrence phase, where the output generation modules are instantiated to produce the output. The block diagram for this phase is shown in Fig. 3.4(b). Finally in the last δ iterations, all the input digits are utilized an no new input is fed to the algorithm, accordingly the modules for taking and converting inputs are removed in this phase as seen in Fig. 3.4(c).

조선대학교
CHOSUN UNIVERSITY

Algorithm 3.1 Online Multiplication

Initialize:

$x[-3] = y[-3] = w[-3] = 0$

for j=$-3,-2,-1$ **do**

  $x[j+1] \leftarrow CA\left(x[j], x_{j+4}\right)$;

  $y[j+1] \leftarrow CA(y[j], y_{j+4})$;

  $v[j] = 2w[j] + \left(x[j]y_{j+4} + y[j+1]x_{j+4}\right)2^{-3}$

  $w[j+1] \leftarrow v[j]$

end for

Recurrence:

for $j = 0 \ldots n-\delta-1$ **do**

  $x[j+1] \leftarrow CA\left(x[j], x_{j+4}\right)$;

  $y[j+1] \leftarrow CA(y[j], y_{j+4})$;

  $v[j] = 2w[j] + \left(x[j]y_{j+4} + y[j+1]x_{j+4}\right)2^{-3}$

  $z_{j+1} = SELM(\widehat{v[j]})$

  $w[j+1] \leftarrow v[j] - z_{j+1}$

  $Z_{\text{out}} \leftarrow z_{j+1}$

end for

Last δ cycles:

for $j = n-\delta \ldots n-1$ **do**

  $x[n-\delta \ldots n-1] = y[n-\delta \ldots n-1] = 0$

  $v[j] = 2w[j]$

  $z_{j+1} = SELM(\widehat{v[j]})$

  $w[j+1] \leftarrow v[j] - z_{j+1}$

  $Z_{\text{out}} \leftarrow z_{j+1}$

end for

조선대학교
CHOSUN UNIVERSITY

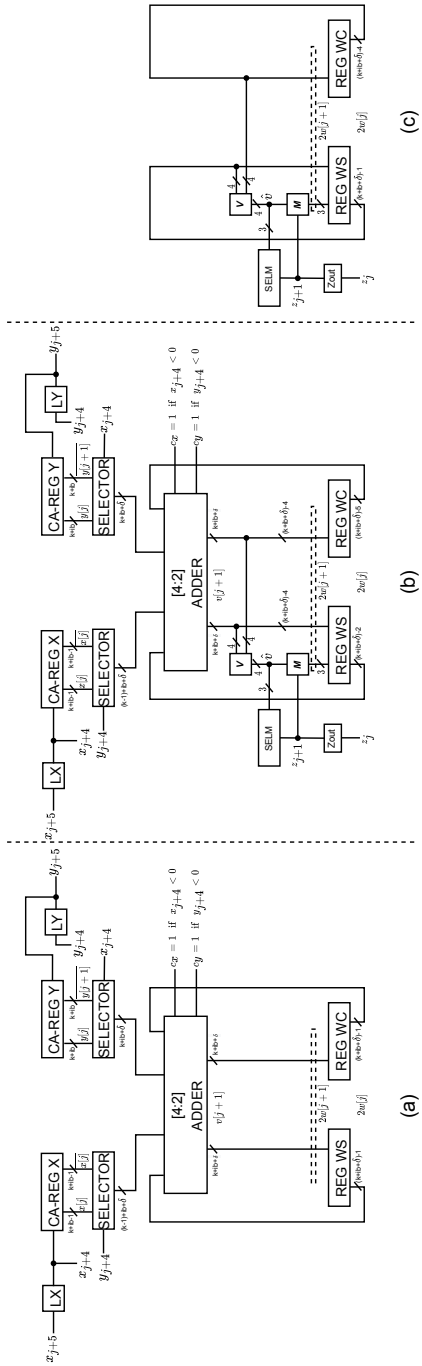

Figure 3.4: Implementation level diagram for the proposed low power radix-2 online multiplier

조선대학교
CHOSUN UNIVERSITY

3.2.7 Implementation Details

In the proposed design, each digit slices have been carefully fine tuned in order to reduce signal activities and save power. Accordingly only useful modules are instantiated in each step of the three sub-loops. We present the details of each module and the corresponding digit slice structure in the following.

3.2.7.1 Initialization

During initialization, the algorithm executes for δ cycles to accumulate sufficient input digits to produce the first output. Since no output digit is produced during initialization, the modules to generate the output digit are not implemented. The digit slices in the initialization consists of OTFC units and selectors. While the presence of adders either half, full or their combination depends on the position of the digit slice. The detail of each unit in the initialization stage is as follows:

3.2.7.2 On-the-Fly Conversion

The redundant SD inputs are required in the conventional form during the recurrence step j, which are obtained without any additional delay using the OTFC module; proposed in [12]. Two OTFC units are instantiated for the two operands during initialization and recurrence, each composed of two $2-to-1$ multiplexers, 2-input *OR* and *AND* gates and two registers to store Q and $QM = Q-1$ as shown in Fig. 3.5. In each iteration, a new incoming digit is appended in the least-significant digit of either Q and QM registers depending on the value of q_{j+1}, increasing its width by one bit upto $n+ib$. Two integer bits are initialized as '00' or '11' representing '0' and '-1' for the first positive or negative fractional bit respectively. The conversion/append (*CA-Reg*) registers shown in Fig. 2.4 correspond to the *Q[j+1]* register of the OTFC unit. According

조선대학교
CHOSUN UNIVERSITY

the online multiplier's algorithm, the computation requires advance availability of one of the operands (in this case operand y), therefore, the bit width for 'y' OTFC unit is one bit longer than 'x' in all iterations.

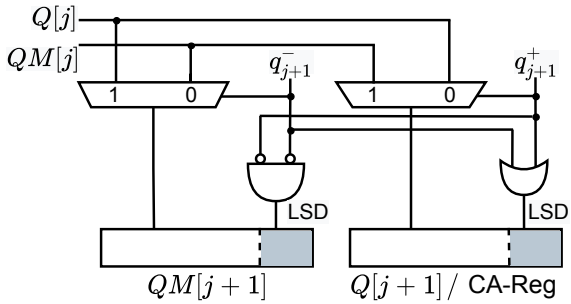

Figure 3.5: Digit slice of on-the-fly converter.

3.2.7.3 Selector

The multiplication is performed by *selector*, which is a four input multiplexer as shown in Fig. 3.6. Since it receives the inputs from the *CA-Reg* registers, the width of its inputs also increases upto p^{th} iteration, and then begins to decrease. As there are no inputs in the last δ cycles, the selector module is instantiated during initialization and recurrence stages only. At each iteration, the signed digit selector can take values from $1, -1$ or 0, encoded as '10','01' and '00', for which the selector outputs $x.y$, $\overline{x.y}$ or 0 respectively.

3.2.7.4 Adder

A $[4:2]$ carry-save adder (CSA) is employed to perform the addition of the input operands and the residual. The functionality of this adder is obtained by utilizing two full adders. The final output vectors of sum and carry are denoted as *vs* and *vc* respectively.

조선대학교
CHOSUN UNIVERSITY

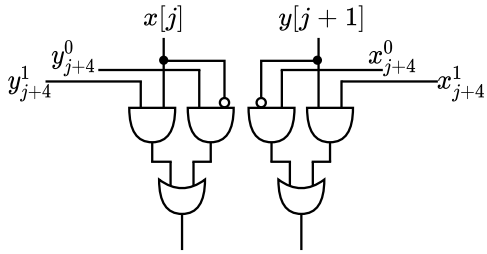

Figure 3.6: Digit slice of selector unit.

At any given iteration *j*, the number of bits in *x* and *y* are $k+ib+\delta$ and $k+ib+\delta+1$ respectively, where *k* are the number of fractional bits in a given iteration. The length of residual registers *WS* and *WC* is $k+ib+\delta-1$ during initialization.

For the low power implementation, a distinct structure of adder is specified according to the bit position, as shown in different colors in Fig. 3.2. For the two's complement representation, multiplication with '-1' can be performed by negating the input bits and adding a logical 1 to the *unit in the last place (ulp)*, therefore, the least significant bit vc_k of the $vc[j]$ vector is designated for c_x ($c_x = x^+_{j+4} \cdot \overline{x^-_{j+4}}$), the corresponding digit slice to obtain the least significant bits of *vc* and *vs* is the red colored slice *k* from the Fig. 3.2 and its internal circuit is depicted in Fig. 3.7 (d). It has no adders because the length of vector $y[j]$ is largest and there are no digits to be added, therefore, y_k and c_x are simply copied to vs_k and vc_k respectively.

For the same reason of achieving correct result of multiplication of a vector by '-1' in the two's complement, the least significant bit VC_k of the $VC[j]$ is accounted for c_y ($c_y = y^+_{j+4} \cdot \overline{y^-_{j+4}}$). Since the length of vector $x[j]$ is one digit smaller than $y[j]$, c_y is present in the $(k-1)^{th}$ digit slice. This is the purple colored

조선대학교
CHOSUN UNIVERSITY

slice in Fig. 3.2 and its internal circuit is depicted in Fig. 3.7 (c). The length of the recurrence registers is one bit smaller than $x[j]$, therefore a single full adder is employed to add the three input digits $x[j]_m$, $y[j]_m$ and c_y. Furthermore, absence of adder in the k^{th} place accounts for no output carry, therefore, a permanent '0' is placed at the vc_{k-1} position. Due to a single full adder in $(k-1)^{th}$ position, there are no intermediate sum or carry digits, instead a final sum vs_{k-1} and a carry vc_{k-2} is produced. This implies that in the $(k-2)^{th}$ digit slice, a full adder in the first stage and a half adder in the second stage is sufficient to produce the outputs. This slice is shown in yellow color in Fig. 3.2 while its logic is shown in Fig. 3.7 (b). The $(k-2)^{th}$ digit slice however, generates both intermediate and final carry digits to the higher digit slice, therefore, the $(k-3)^{th}$ digit slice is composed of two full adders. First full adder evaluates the sum of $x[j]$, $WS[j]$ and $WC[j]$ and produces an intermediate carry and sum vector named as $VC[j]$ and $VS[j]$ respectively. The second full adder evaluates the sum of $VS[j]$, $VC[j]$ and $y[j+1]$ to produce final sum and carry, expressed as, $vs[j]$ and $vc[j]$ respectively and are collectively represented as $v[j]$ shown in Eq. 3.3. This grey shaded digit slice from Fig. 3.2 is implemented using the logic shown in Fig. 3.7 (a). It is named as repeated digit slice (RD) as the same digit slice is repeated $k+ib$ during initialization.

3.2.7.5 Residual Calculation

In the initialization phase, the residual for the next iteration $(2w[j+1])$ corresponds to the left shifting of *vs* and *vc* vectors. The most significant *ibs* of both vectors (i.e., vs_{-1} and vc_{-1}) are discarded and the vectors are left shifted by a simple re-wiring. The updated residual $(2w[j+1])$ is shown in relation (3.2).

조선대학교
CHOSUN UNIVERSITY

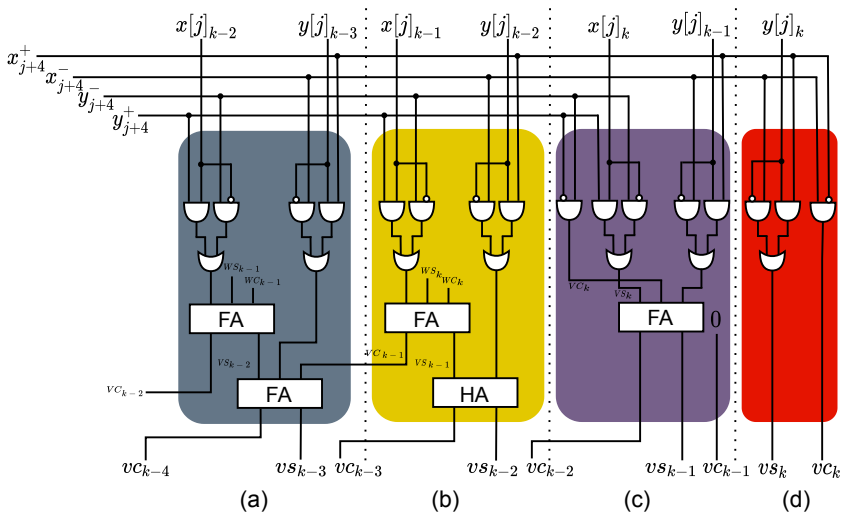

Figure 3.7: Internal structure for the least significant and repeated digit slices.

$$2w[j+1]\left|\begin{array}{l} vs_0\ vs_1 \cdot vs_2\ vs_3\ vs_4\ vs_5 \ldots \\ vc_0\ vc_1 \cdot vc_2\ vc_3\ vc_4\ vc_5\ \ldots \end{array}\right. \tag{3.2}$$

3.2.7.6 Recurrence

After accumulating sufficient number of input digits to generate the output, the algorithm advances to the recurrence stage. SEL digit slice is instantiated to generate the output. The computation of next residual ($2w[j+1]$) involves M block which subtracts the output digit $z[j+1]$ from $\hat{v}[j]$. Similar structures of the OTFC and the selector modules shown in Fig. 3.5 and Fig. 3.6 respectively are utilized for the entire recurrence stage with bit widths corresponding to the signal activity pattern. The blue colored SEL digit slice from Fig. 3.2 uses the logic depicted in Fig. 3.8.

Residual's estimate calculation, selection of the output digit and subtraction of the output digit from residual are performed by distinct modules present in the

조선대학교
CHOSUN UNIVERSITY

SEL digit slice, details of each of these modules are briefed below:

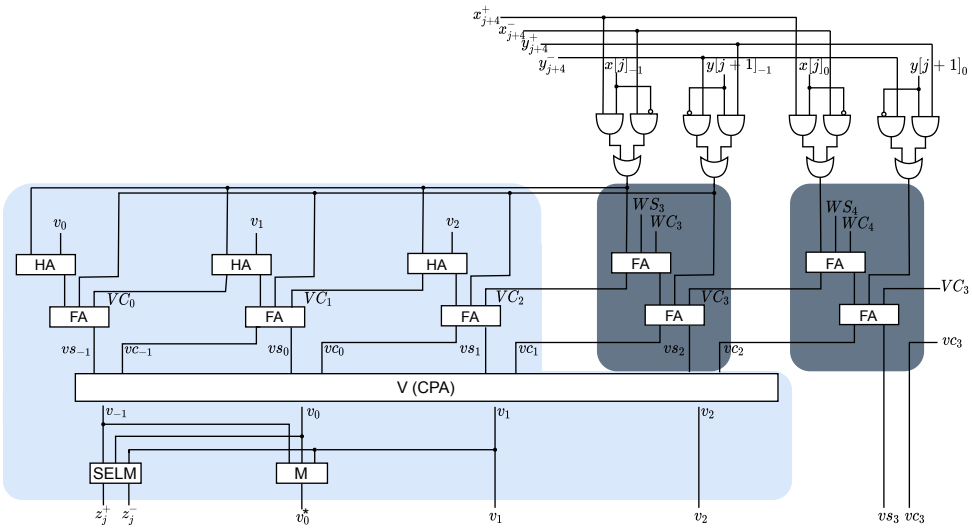

Figure 3.8: Logic for the SEL digit slice during recurrence. The three MSBs are composed of a combination of half and full adders in contrast to two full adders in the repeated digit slices.

3.2.7.7 V Block

The output is based on the estimate ($\hat{v}[j]$) of the residual ($v[jj]$), is evaluated in the V block. It is a carry propagation adder that performs the addition of t most significant fractional bits and the integer bits of $v[j]$ (represented by $vs[j]$ and $vc[j]$ vectors shown in Eq.(3.3)) to generate the estimate of the residual ($\hat{v}$) as shown in Eq. (3.4).

$$v[j] \Big| \begin{array}{l} vs_{-1}\ vs_0 \cdot vs_1\ vs_2\ vs_3\ vs_4 \ldots \\ vc_{-1}\ vc_0 \cdot vc_1\ vc_2\ vc_3\ vc_4 \ldots \end{array} \tag{3.3}$$

$$\hat{v} \Big| \ v_{-1}\ v_0 \cdot v_1\ v_2 \ldots \tag{3.4}$$

조선대학교
CHOSUN UNIVERSITY

3.2.7.8 SELM Block

The result of the V block is subjected to the *SELM* module for selecting the corresponding output from a look-up table shown in Table. 2.1.

3.2.7.9 M Block

It performs the subtraction of z_{j+1} from the residual's estimate ($\hat{v}$) to produce $2w[j+1]$. The subtraction to obtain v_0^* is performed using the following Boolean expression [8]:

$$v_0^* = v_0 \, XOR \, |p_{j+1}|. \tag{3.5}$$

3.2.7.10 Adder

The RDs and the least significant digit slices for adder are similar to the initialization stage, and for k bit precision, $k + ib - 3$ number of RDs are instantiated in a certain iteration. The length of the vector *vc* after being subjected to V block is reduced by 3 bits (refer to Eq. (3.6)). Therefore, in the 3 most significant bit slices, which accounts for the SEL block, instead of two stages of full adders, a half adder is employed in the first stage and a full adder is employed in the second stage. The multiplication of the terms $x[j] \cdot y_{j+4}$ and $y[j+1] \cdot x_{j+4}$ in the recurrence equation with 2^{-3} in both initialization and recurrence stages, corresponds to the sign extension of the MSBs, which is done by performing 3 bit arithmetic right shift operation without additional cost.

3.2.7.11 Residual Calculation

The MSB of $\hat{v}[j]$ i.e., $\hat{v}_{-1}$ is discarded and the remaining 3 bits are vertically transferred to *vs* vector consequently resulting in an updated residual as shown in relation (3.6). In this manner the left shifting of the residual ($2w[j]$) is carried

조선대학교
CHOSUN UNIVERSITY

out.

$$2w[j+1]\Big| \begin{array}{r} v_0\ v_1 \cdot v_2\ v_3\ v_4\ v_5 \ldots \\ v_3\ v_4\ v_5 \ldots \end{array} \tag{3.6}$$

3.2.7.12 Last δ cycles

The remaining output digits are obtained in the last δ iterations which produces

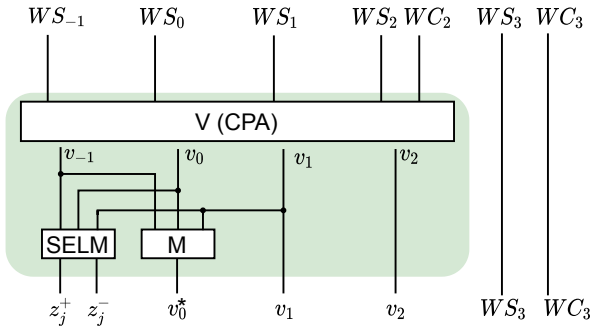

Figure 3.9: Logic for the MSB in last δ cycles. The SEL digit slice is simplified to a carry propagation adder, SELM and M block; taking the shifted residuals as input and producing an output in each cycle.

one output digit in each cycle. All the inputs are utilized in the prior stages and in the conventional online multiplier, three 0*s* are applied in the LSB of the inputs. However, in the proposed low-power design, all unused modules are eliminated and therefore, the OTFC, selector and $[4:2]$ adders are not implemented. The residual containing two vectors *WS* and *WC* are subjected to the *V*, *M* and *SELM* modules to perform their respective tasks and generate the output digit. The digit slice used during the last δ iterations has been depicted in Fig. 3.9.

조선대학교
CHOSUN UNIVERSITY

3.2.8 Synthesis Results and Comparison

The Verilog code has been developed for $n = 8, 16, 24$ and 32 bit precision multipliers and simulated on Modelsim. A case study has been presented for 16-bit precision multiplier with operands

$$x = 00.110\bar{1}0\bar{1}\bar{1}011\bar{1}0\bar{1}100$$
$$y = 00.\bar{1}1\bar{1}100\bar{1}101\bar{1}11\bar{1}0\bar{1}$$

The numerical value of x and y is 0.66644287109375 and -0.31562805175781 respectively. The actual product in conventional form is -0.2103480650112033 and the calculated product from online multiplier is -0.2103424072265625. The difference between the actual and calculated product is $5.657784640789032 \times 10^{-6}$ which is well under the error bound of the last iteration which is $2^{-16} = 1.52587890625 \times 10^{-5}$. Not only the final result, but the result of the online multiplier in each cycle is with in the respective error bound according to relation (2.3). In the event of variable precision requirement, the computation can be stopped upon reaching the desired precision, resulting in an accurate result upto that precision; unlike approximate circuits in which result is not accurate.

For reduced working precision for $n = 16$, $p = 13$ was evaluated from relation (3.1). The digit slices are gradually increased according to the increasing precision of the inputs until $p - \delta$ cycles, in contrast to the conventional implementation which has increment in the input's precision till $n - \delta$ cycles and has constant slice activity, i.e., all n bits remain active in all cycles. Truncation is applied in $p - \delta + 1$ cycle that introduces an error in positions p, $p-1$ and $p-2$ which propagates to the left for residual calculation ($2w[j]$). Consequently, the 3 least significant digit slices affected by the truncation error are not implemented in the forthcoming cycles. For the last δ iterations, the modules for input and

조선대학교
CHOSUN UNIVERSITY

carry-save adder are not present and one bit reduction in the digit slice is due to the left shift operation of the residual. The input digits used in the proposed design are shown in clear fonts in Table. 3.2, whereas the conventional design uses both clear and shaded digits. The $[4:2]$ adder takes two inputs from *Selector* units and residuals WS and WC, generating redundant vectors VS and VC the sum of which is shown as $v[j]$ in Table. 3.2.

Table 3.2: Example of radix-2 online multiplication for $n = 16$ with reduced working precision $p = 13$.

j	x_{j+4}	y_{j+4}	$x[j]$	$y[j+1]$	$v[j]$	p_{j+1}		Error bound
						SD	Conventional	
-3	1	$\bar{1}$	0.0000000000000000	1.10000000000000000	11.1111	-	-	-
-2	1	1	0.1000000000000000	1.11000000000000000	11.11101	-	-	-
-1	0	$\bar{1}$	0.1100000000000000	1.10100000000000000	11.101110	-	-	-
0	$\bar{1}$	1	0.1100000000000000	1.10110000000000000	11.1001001	0	0.0	2^{-1}
1	0	0	0.1011000000000000	1.10110000000000000	11.00100100	$\bar{1}$	-0.25	2^{-2}
2	$\bar{1}$	0	0.1011000000000000	1.10110000000000000	00.010100100	0	-0.25	2^{-3}
3	$\bar{1}$	$\bar{1}$	0.1010110000000000	1.10101110000000000	00.1001100011	1	-0.1875	2^{-4}
4	0	1	0.1010101000000000	1.10101111000000000	11.01000110110	$\bar{1}$	-0.21875	2^{-5}
5	1	0	0.1010101000000000	1.10101111000000000	00.100000110110	0	-0.21875	2^{-6}
6	1	1	0.1010101010000000	1.10101111010000000	01.0001000111111	1	-0.2109375	2^{-7}
7	$\bar{1}$	$\bar{1}$	0.1010101011000000	1.10101111001000000	00.00011000101101	0	-0.2109375	2^{-8}
8	0	1	0.1010101010100000	1.10101111001100000	00.0100011010111110	0	-0.2109375	2^{-9}
9	$\bar{1}$	1	0.1010101010100000	1.10101111001110000	00.10101100111001101	1	-0.2099609375	2^{-10}
10	1	$\bar{1}$	0.1010101010011000	1.10101111001101000	11.00111010010111100	$\bar{1}$	-0.21044921875	2^{-11}
11	0	0	0.1010101010011100	1.10101111001101000	00.0111010010100	0	-0.21044921875	2^{-12}
12	0	$\bar{1}$	0.1010101010011100	1.10101111001100110	00.1101010000	1	-0.2103271484375	2^{-13}
13	-	-	-	-	11.101001100	0	-0.2103271484375	2^{-14}
14	-	-	-	-	11.01010000	$\bar{1}$	-0.210357666015625	2^{-15}
15	-	-	-	-	00.1010000	1	-0.2103424072265625	2^{-16}

For comparative analysis, both conventional and proposed designs have been implemented on FPGA and their synthesis is performed to obtain the power consumption and area utilization.

조선대학교
CHOSUN UNIVERSITY

3.2.8.1 FPGA Implementation

We present the pipelined implementation of both conventional and proposed designs on FPGA. implementation of both conventional and proposed designs have been done on Xilinx Virtex-7 VC707 FPGA evaluation platform with XC7VX485T processor. The utilization of slice registers, slice look up tables (LUTs) and flip flop pairs have been compared. Reduction of working precision and gradual increase and decrease of the slice activity results in the savings of 40% configurable logic blocks (CLBs) decreasing from 715 in conventional to 425 in the proposed design for the 32-bit precision. Further results are presented in It can be observed from Table. 3.3.

Table 3.3: Comparison of FPGA resource utilization of the proposed and conventional design for the pipelined radix-2 online multiplier.

Precision	**8**	**16**	**24**	**32**	**Resources**
Conventional	164	1299	2642	4366	Occupied Slices
Proposed	**103**	**800**	**1529**	**2446**	
Reduction (%)	37.19	38.41	42.12	43.98	
Conventional	270	1030	2123	3225	Slice Registers
Proposed	**238**	**738**	**1434**	**2146**	
Reduction (%)	11.85	28.34	32.45	33.45	
Conventional	445	1874	3836	5713	Slice LUTs
Proposed	**369**	**1202**	**2351**	**3393**	
Reduction (%)	17.07	35.85	38.71	40.60	
Conventional	453	1877	3837	5751	LUT - Flip Flop Pairs
Proposed	**381**	**1211**	**2357**	**3439**	
Reduction (%)	15.89	35.48	38.64	40.20	

3.2.8.2 Synthesis using Yosys, ABC and SIS

We present the results of synthesis to obtain the power and area estimate in terms of digital logic gates. For area estimation, we use relative area of gates

조선대학교
CHOSUN UNIVERSITY

from the dictionary calculated by [36] for MCNC library, $V_{dd} = 5$V and clocking frequency of 20 MHz relative to single NAND gate. The total area therefore, will be the sum of relative areas of the gates. The synthesis to obtain the equivalent logic gates of the multiplier has been performed using Yosys synthesis suite [37]. The area estimation is performed using Berkely ABC tool [3] for the MCNC library, and for power consumption estimation SIS software has been utilized for the MCNC library at $V_{dd} = 5$V and 20 MHz clock.

Table 3.4: Comparison of area and power estimates of n bit pipelined designs for the radix-2 online multiplier with full and reduced working precision.

Working Precision	**n**				**Resources**
	8	**16**	**24**	**32**	
Full	432	1734	2906	4844	Latches
Reduced	315	976	1906	3162	
Savings %	27.08	31.93	31.41	34.72	
Full	2385	1903	18402	30869	Nodes
Reduced	1786	5898	18455	17801	
Savings %	25.11	34.21	37.87	40.21	
Full	4474	16851	34617	58204	Edges
Reduced	3395	11363	22112	35759	
Savings %	24.11	23.56	36.12	38.56	
Full	2629.39	1059.32	21556.31	36217.59	Area
Reduced	1947.91	6432.94	12461.77	20133.69	
Savings %	25.91	38.9	42.18	44.4	
Full	25812.8	95179.7	194340.5	325646.8	Power
Reduced	18695.5	62720.4	122039	199687.7	
Savings %	27.57	34.1	37.2	38.68	

It can be observed that the proposed design consumes less number of latches, has fewer nodes and edges due to reduced signal activity and therefore results in significant area and power savings. The pipelined architecture allows to remove the unused circuitry which is useful for saving the static power. The reduction

조선대학교
CHOSUN UNIVERSITY

follow an increasing trend with the increase in the precision suggesting increased savings for longer word lengths.

3.2.8.3 Synthesis using Design Compiler

For better analysis, the designs have been synthesized using the Synopsys Design Compiler. All design have been synthesized using 45nm TSMC technology file and a constrain have been applied on the critical path. Table. 3.5 presents the synthesis results of a non-pipelined radix-2 multiplier of various bit precision. Table. 3.6 and Table. 3.7 presents the synthesis results of a pipelined radix-2 multiplier with full and reduced working precision respectively.

Table 3.5: Area, power utilization and cycle time of non-pipelined *n* bit radix-2 online multiplier.

Operand Width	Area(μm^2)			Power(mW)	Timing (ns)	
n	Comb	Non-Comb	Total		Required	Critical Path
8	1024.01	590.37	1614.39	1.71	0.90	0.75
16	1221.59	1180.75	2458.66	2.40	0.90	0.75
32	2237.63	2329.60	4567.22	4.41	0.90	0.75

Table 3.6: Area, power utilization and cycle time of pipelined *n* bit radix-2 online multiplier with full working precision.

Operand Width	Area(μm^2)			Power(mW)	Timing (ns)	
n	Comb	Non-Comb	Total		Required	Critical Path
8	4726.39	2273.74	7000.07	7.18	0.90	0.75
16	15830.84	7738.71	23570.01	2.40	0.90	0.75
32	54399.06	26041.40	80397.65	4.41	0.90	0.75

Conventional multipliers of different types including serial-parallel, array and tree have also been developed and synthesized to observe and compare

조선대학교
CHOSUN UNIVERSITY

Table 3.7: Area, power utilization and cycle time of pipelined n bit radix-2 online multiplier with reduced working precision.

Operand Width	Area(μm^2)			Power(mW)	Timing (ns)	
n	Comb	Non-Comb	Total		Required	Critical Path
8	3475.16	1699.335	5174.50	5.38	0.90	0.75
16	10998.88	5409.12	16408.13	16.88	0.90	0.75
32	31708.58	16562.92	48269.85	49.41	0.90	0.75

the resource utilization and cycle time. The synthesis results for serial-parallel, array and tree type multipliers have been shown in Tables. 3.8, 3.9 and 3.10 respectively. Since the shown multipliers are not pipelined, their resource utilization is less than pipelined online multiplier. However, the cycle time for the aforementioned multipliers is larger than the online multipliers (pipelined or non-pipelined) as well as dependent on the bit precision.

Table 3.8: Area, power utilization and cycle time of a serial-parallel multiplier.

Operand Width	Area(μm^2)			Power(mW)	Timing (ns)	
n	Comb	Non-Comb	Total		Required	Critical Path
8	810.94	366.99	1177.94	0.91	0.90	0.84
16	1910.06	694.09	2604.15	1.79	0.90	0.90
32	3467.23	1340.32	4807.50	2.12	0.90	1.44

Table 3.9: Area, power utilization and cycle time of an array type multiplier.

Operand Width	Area(μm^2)			Power(mW)	Timing (ns)	
n	Comb	Non-Comb	Total		Required	Critical Path
8	656.08	0	656.08	2.83E-02	0.90	0.90
16	4149.59	0	4149.5	0.20	1.10	1.10
32	10542.82	0	10542.82	0.58	6.00	5.68

조선대학교
CHOSUN UNIVERSITY

Table 3.10: Area, power utilization and cycle time of a tree type multiplier.

Operand Width	Area(μm^2)			Power(mW)	Timing (ns)	
n	Comb	Non-Comb	Total		Required	Critical Path
8	1414.47	0	1414.47	8.10E-02	0.90	0.88
16	7375.69	0	7375.69	0.49	1.10	1.10
32	30551.45	0	30551.45	2.31	2.00	1.60

3.3 Pipelined Online Adder

The online delay affects the throughput for streams of input because zero padding has to be performed between two consecutive streams. A method to pipeline the online adder for streaming inner products has been proposed in [2]. To achieve digit-level pipelining, the adder in Fig. 2.5 has been divided into two modules, each having a full adder, the upper one called as module 1 and the lower one named as 2 as shown in Fig. 3.10.

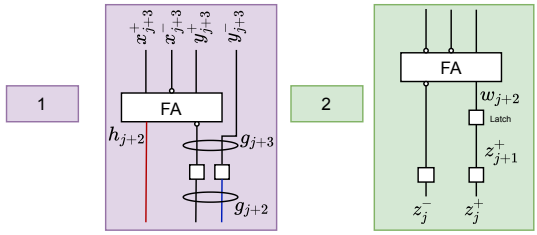

Figure 3.10: Modules in radix-2 onilne adder

The pipelining is done by rearranging the online adders in a 2-D array of modules 1 and 2, such that the module 1 produces the intermediate values and module 2 generates the final sum as shown in Fig. 3.11. The full adder in module 1 generates the carry and sum depicted in red and black connection respectively, out of which the sum is connected to the full adder of module 2 of the same column

조선대학교
CHOSUN UNIVERSITY

along with the latched input shown in blue color, while the carry of module 1 is connected to adjacent module 2 in the higher column position.

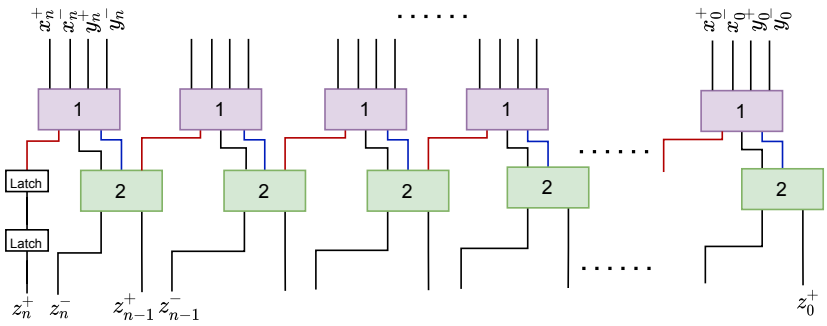

Figure 3.11: Pipelined radix-2 online adder [2]

The proposed pipelined online adder can be supplied with streams of input for addition without any separation cycle. For multiple streams of input, the output of the first stream of the input is received in $n+\delta+1$ cycles and after one more cycle, the output of the second stream is obtained. Therefore, for k input streams the output is obtained in $n+\delta+k$ cycles. It takes n cycles to fill the pipeline after which the output of the stream is obtained in one cycle. For large number of streams when $k >> n$, the initial delay can be ignored.

3.3.1 Adder Tree

Using the pipelined online adders, an adder tree can be developed to compute the partial sums. The resource consumption of the pipelined online adder tree is higher than that of a parallel adder tree, however, the pipelined online adder tree produces result from left-to-right due to which the operation in succession can be started without waiting for the completion of current operation.

조선대학교
CHOSUN UNIVERSITY

3.3.2 Synthesis Results

This section provides the synthesis results of a conventional, pipelined and parallel adder and adder trees. Verilog code have been devloped for all types of adder discussed in this section and the functional verification has been done using ModelSim. Table. 3.11 shows the synthesis results of non-pipelined, pipelined and parallel adder with 8-bits precision.

Table 3.11: Area, power utilization and cycle time of a non-pipelined, pipelined online and parallel adder for n=8.

Adder Type	Area(μm^2)			Power(mW)	Timing (ns)	
	Comb	Non-Comb	Total		Required	Critical Path
Online (Non-piplined)	52.09	39.89	91.98	0.31	0.25	0.25
Online (Pipelined)	397.96	335.08	733.04	2.58	0.25	0.25
Parallel	317.24	0	317.24	0.016	0.25	0.25

As an example, a n=8-bit adder tree to add k=8 partial sums using pipelined online and parallel adders have been developed and its synthesis results are shown in Table. 3.12.

Table 3.12: Area, power utilization and cycle time of a pipelined online and parallel adder tree for n=8 and k=8.

Adder Type	Area(μm^2)			Power(mW)	Timing (ns)	
	Comb	Non-Comb	Total		Required	Critical Path
Online (Pipelined)	6393.85	5329.34	11723.39	42.50	0.25	0.25
Parallel	2537.97	0	2537.97	0.129	0.25	0.25

조선대학교
CHOSUN UNIVERSITY

3.4 Pipelined Online Sum-of-Product

In order to add multi-operand inputs, the pipelined online adder is configured to form an adder tree. This adder tree together with the pipelined online multiplier will form a pipelined online Sum-of-Product (SoP) module as shown in Fig. 3.12. For each stage in the adder tree, the online delay is summed up. And again, for large number of inner products this initial delay is negligible. An adder tree to add P input operands will have an initial delay of $\log_2(P) * \delta_{add} + 1$. This is the time required to produce first MSD of the output, whereas the complete n-bit output is obtained in $\log_2(P) * \delta_{add} + n + 1$. For k streams of input, the output is obtained in $\log_2(P) * \delta_{add} + n + k$ cycles.

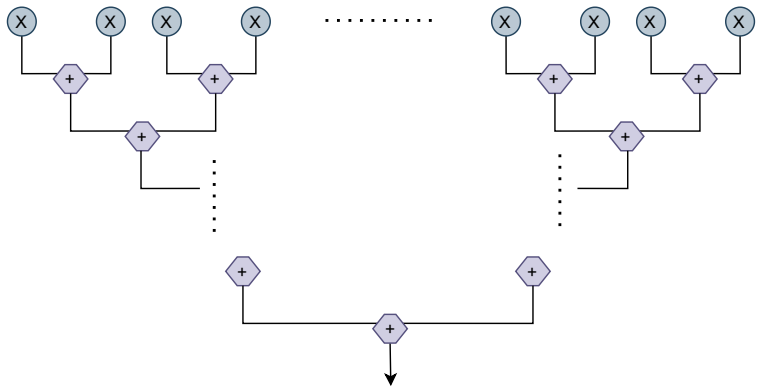

Figure 3.12: Pipelined online SOP

3.4.1 Synthesis Results

In this section, we provide the synthesis results for the sum-of-product units developed using pipelined online multiplier with reduced working precision and pipelined online adder as shown in Fig. 3.12. SoP units are developed for k

조선대학교
CHOSUN UNIVERSITY

multiplications k=16, 32 and 128, each of which has the precision of n=8 bits. Table. 3.13 shows the resource utilization and critical path of different SoP units with n=8 bits precision. It depicts the benefit of pipelined online SoP unit in

Table 3.13: Area, power utilization and cycle time of pipelined online SoP.

No. of multipliers	Area(μm^2)			Power(mW)	Timing (ns)	
n	Comb	Non-Comb	Total		Required	Critical Path
16	63223.19	32519.95	95735.67	99.21	0.90	0.84
32	127241.80	65503.09	192791.75	200.44	0.90	0.84
128	464128.18	262117.98	72664.25	778.48	0.90	0.84

terms of its constant critical path delay. The number of multiplications to be summed determines the depth of adder tree. In conventional arithmetic based SoP is dependent both upon the number of products to be summed as well as the precision of the operands. With pipelined online units, the critical path delay is no more than the individual stage time of pipelined units.

3.5 Chapter Summary

This chapter provided the details of the proposed algorithm for multiplication. The pipelined architecture has been depicted and the internal structures of each module has been shown and discussed in detail. Furthermore, the architecture of a conventional online adder is shown. The online adder has been rearranged to support digit-level pipelining which results in an increased throughput. The pipelined online adder has been utilized to develop a pipelined online adder tree that can be utilized to generate the sum of inner products with an increased throughput. The synthesis results of various implementations multipliers including online multiplier, pipelined online multiplier with full and

조선대학교
CHOSUN UNIVERSITY

reduced working precision and conventional multipliers. The synthesis results are shown for the non-pipelined and pipeline online adder, parallel adder and respective adder trees. Finally the results are presented for the pipelined SoP to be utilized for computing inner products in deep learning applications.

조선대학교
CHOSUN UNIVERSITY

4 Convolutional Neural Network Acceleration

Convolutional neural networks (CNN) is one of the most popular deep learning architecture which has been widely adopted in a variety of applications ranging from image recognition, mobile vision, object detection, surveillance etc. It is inspired by the optic nerves in the living creatures and processes the data by employing a large number of neuron connections to achieve high accuracy. These networks require high computation and massive data storage which pose challenge for both computational performance and energy efficiency. The CNN follow a specific computation pattern, which general computation processors do not perform efficiently. In this regard, several CNN accelerators based on various hardware platforms have been propose in the recent years [5, 17, 21, 29, 40].

Typically, a CNN is composed of two components: feature extraction and classification. The features of the input can include edges, lines, corners etc., which are not affected by the position and distortions. The feature extractor takes the input and extracts these features to map them onto an output feature map which is a low dimensional vector. Generally there are series of such computational layers along with optional sub-sampling layers that feeds the extracted features to the classifier. The classifier is a traditional fully connected neural network which decides the class/category of the input.

The computation in the convolutional layer has been illustrated in Fig. 4.1. The input layer receives N feature maps of size RxC. Each input feature map is convolved with a sliding kernel window of size KxK to produce a output feature map. The sliding of the kernel on input feature map is determined by stride S. A set of M kernels are applied on to the input feature map to obtain M output

조선대학교
CHOSUN UNIVERSITY

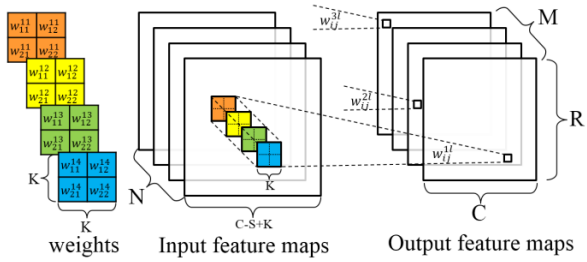

Figure 4.1: Graph of a convolutional layer [40].

feature maps that will form input to the next layer.

The convolution operation occupy over 90% of the computation time [6], therefore, the acceleration of this operation is of paramount importance.

In works [40] and [17], CNN accelerators have been proposed and as a case study the convolution layers of AlexNet [19] have been considered for computing performance improvements. The AlexNet shown in Fig. 4.2, is composed of 8 layers, of which the first 5 layers are convolutional layers and the last two layers are fully connected.

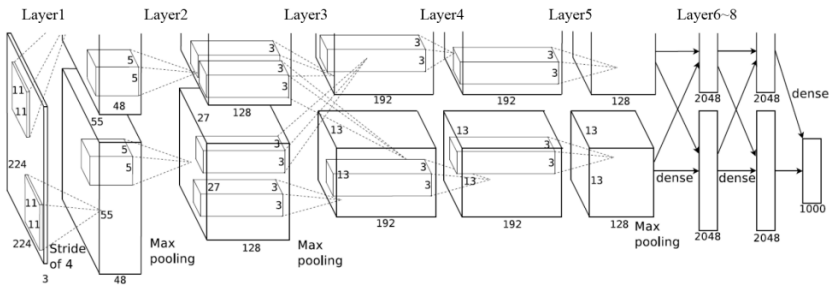

Figure 4.2: Structure of AlexNet [19].

The input layer receives an image having resolution of R_ixC_i (227x227) and 3 channels or input feature maps ($N = 3$). Convolution is performed in Layer1

조선대학교
CHOSUN UNIVERSITY

with a kernel of size KxK (11x11) and stride S of 4 to produce 96 output feature maps/channels ($M = 96$). Both works ([40, 17]) use 2 sets of hardware, each producing 48 maps of resolution R_oxC_o (55x55). The resolution of the output feature map is calculated using the relation $R_o = \frac{R_i - K}{S} + 1$ and $C_o = \frac{C_i - K}{S} + 1$. The configurations of rest of the layers have been shown in Table. 4.1.

Table 4.1: CNN configurations in [40, 17].

Layer	**1**	**2**	**3**	**4**	**5**
Input channels (N)	3	48	256	192	192
Output channels (M)	48	128	192	192	128
Output Rows (R_o)	55	27	13	13	13
Output Columns (C_o)	55	27	13	13	13
Input Rows (R_i)	227	55	27	13	13
Input Columns (C_i)	227	55	27	13	13
Kernel (K)	11	5	3	3	3
Stride (S)	4	1	1	1	1
Sets	2	2	2	2	2

4.1 Design Overview of [40]

The accelerator has been designed to target FPGA and is composed of several components as shown in Fig. 4.3, of which we are interested in what computes the convolution. The convolution is performed by a unit called the processing engine (PE). PE are typically multipliers to compute the inner products followed by an adder tree to obtain the SoPs. The pseudocode for CNN is shown in Fig. 4.4.

In order to efficiently utilize the available hardware resources, it is advantageous to perform loop unrolling by considering the data dependencies.

조선대학교
CHOSUN UNIVERSITY

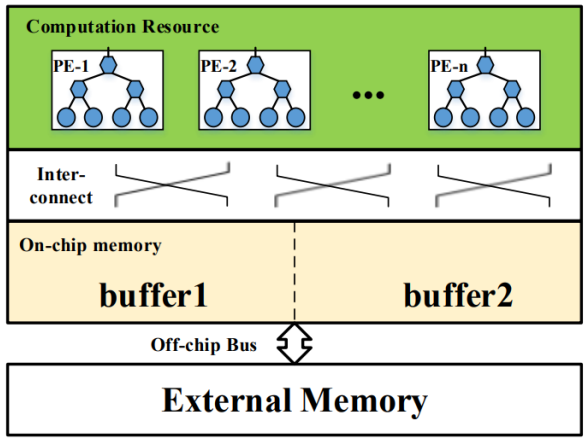

Figure 4.3: Design overview of [40]

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
  L:   output_fm[to][row][col] +=
           weights[to][ti][i][j]*
           input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

Figure 4.4: Pseudocode of CNN

Thus, loop unrolling along different dimensions have been performed as shown in Fig. 4.5 to generate various implementations as shown in Fig. 4.6.

The legal tile sizes for implementation with unrolled loops are shown in (4.1):

$$\begin{cases} 0 < Tm \times Tn \leq (\#\text{ of } PEs) \\ 0 < Tm \leq M \\ 0 < Tn \leq N \\ 0 < Tr \leq R \\ 0 < Tc \leq C \end{cases} \tag{4.1}$$

where T_m and T_n are the tile size for output and input channels.

조선대학교
CHOSUN UNIVERSITY

```
for(trr=row;trr<min(row+Tr,R);trr++){
 for(tcc=col;tcc<min(col+Tc,C);tcc++){
  for(too=to;too<min(to+Tm,M);too++){
   for(tii=ti;tii<min(ti+Tn,N);tii++){
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
     L: output_fm[too][trr][tcc] +=
          weights[too][tii][i][j]*
          input_fm[tii][S*trr+i][S*tcc+j];
} } } } } }
```

Figure 4.5: Pseudocode of tiled convolution layer

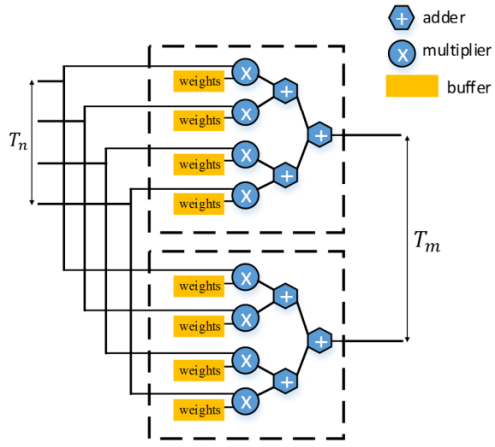

Figure 4.6: Computation Engine [40]

4.2 Attainable Performance [40]

Based on (4.1) the attainable performance can be computed as follows:

$$\begin{aligned} Attainable\ \ Performance &= \frac{total\ number\ of\ operations}{number\ of\ execution\ cycles} \\ &\approx \frac{2*R_o*C_o*M*N*K*K}{\lceil \frac{M}{T_m} \rceil * \lceil \frac{N}{T_n} \rceil * R_o*C_o*K*K} \end{aligned} \tag{4.2}$$

The number of operations are equal to the total number of multiplications and additions to compute the output feature map. The number of multiplications and additions are approximately equal, therefore, the term is multiplied by 2. A set

조선대학교
CHOSUN UNIVERSITY

of M KxK kernels slide on to the input feature map of RxC with N channels. Considering the CNN configuration in Table. 4.1, the number of operations are given as in Table. 4.2. The number of execution cycles depend on the tile sizes

Table 4.2: Number of operations in AlexNet.

Layer	No. of operations
1	105415200
2	223948800
3	149520384
4	112140288
5	74760192

T_m and T_n. The execution cycles required in each layer for Alexnet according to the parameters shown in Table. 4.1 with optimal unroll factors of T_m and T_n are shown in Table. 4.3.

Table 4.3: Number of cycles required to compute convolution in different layers of AlexNet using certain tile sizes in [40].

Layer	$\langle T_m, T_n \rangle$	Execution Cycles
1	$\langle 48, 3 \rangle$	366025
2	$\langle 20, 24 \rangle$	237185
3	$\langle 96, 5 \rangle$	160264
4	$\langle 95, 5 \rangle$	120198
5	$\langle 32, 15 \rangle$	80132

4.3 Design Overview of ESSA [17]

The CNN accelerator named as ESSA is a bit-serial streaming convolutional neural network accelerator that uses several techniques to obtain an energy

조선대학교
CHOSUN UNIVERSITY

efficient design. The bit reduction technique has been applied to reduce the number of operations and loop tiling approach as in [40] has been adopted for better computational performance. The basic computational units in ESSA are the T_mxT_n processing element array (PEA). Each PEA is composed of 9 processing elements (PE) as shown in Fig. 4.7

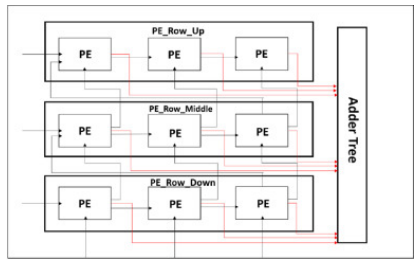

Figure 4.7: Architecture of PE Array in ESSA [17]

The PE are bit-serial and would require more cycles to compute the multiplication result. Therefore, the multiple bit-serial PEs are employed to compute the multiplication in parallel. Each PE in Fig. 4.7 is composed of 8 sub-PEs and each sub-PE is connected to an adder tree to get the sum-of-product. For final result, the output of each PE is summed up using an adder tree. The proposed architecture with sub-PE is shown in Fig. 4.8.

To avoid unnecessary computations, the non-unit stride kernels are mapped to the decomposed unit stride convolution kernels as proposed in [21]. The size of the decomposed kernel (K') is given as $K' = \lceil \frac{K}{S} \rceil$. Moreover, the kernel size in ESSA has been fixed to 3x3 which is mostly used in new CNN architectures, and therefore, to handle the kernel sizes larger than 3, it is decomposed into $(\lceil \frac{K'}{3} \rceil)^2$ sub-kernels. Bit reduction technique has been used and the weight pixels which are fixed in PEs are represented using fewer bits, consequently reducing

조선대학교
CHOSUN UNIVERSITY

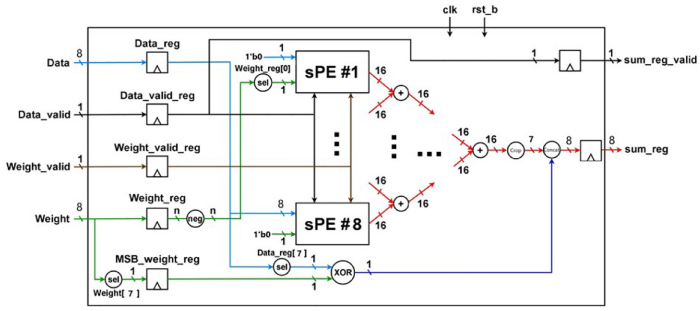

Figure 4.8: Architecture of sub-PE in ESSA [17].

the number of active sub-PEs to perform multiplication between weight and input pixel.

Loop tiling technique has been employed and two tiling factors T_m and T_n are considered. Where T_m is number of processed weights for each occurrence and T_n is the number of processed channels for input feature maps and weights at each occurrence. The number of bits to represent weight pixel are represented by B and the number of sub-PEs to process different bits of the same weight pixel are represented by U.

4.4 Attainable Performance ESSA [17]

Using above notations, the maximum attainable performance of ESSA can be formulated as (4.3).

$$\begin{aligned} Attainable\ \ Performance &= \frac{total\ number\ of\ operations}{number\ of\ execution\ cycles} \\ &= \frac{2*R_o*C_o*M*N*K*K}{(R*C+2)*S^2*\left\lceil\frac{K'}{3}\right\rceil*\left\lceil\frac{K'}{3}\right\rceil*\left\lceil\frac{M}{T_m}\right\rceil*\left\lceil\frac{N}{T_n}\right\rceil*\left\lceil\frac{B}{U}\right\rceil} \end{aligned} \tag{4.3}$$

The accelerator uses the AlexNet to show the performance, therefore, the number of operations are same as computed in Table. 4.2. The number of

조선대학교
CHOSUN UNIVERSITY

execution cycles to compute the convolutions in AlexNet by ESSA are shown in Table. 4.4.

Table 4.4: Number of cycles required to compute convolution in AlexNet for ESSA [17].

Layer	T_m, T_n	R_o	C_o	M	N	K	S	B	U	Execution Cycles
1	48,1	55	55	48	3	11	4	7	7	145296
2	12,7	27	9	128	48	5	1	9	9	225148
3	21,12	13	13	192	256	3	1	7	2	150480
4	47,9	13	13	192	192	3	1	6	1	112860
5	42,9	13	13	128	192	3	1	6	1	75240

4.5 Attainable Performance using Pipelined Online SoP

The dataflow with the pipelined implementation of online modules has been shown earlier in Table. 3.1. If multiple streams of the inputs are supplied to the online module, the MSD of the first input stream is obtained in $\delta + 1$ cycles, and the MSD of second input stream is obtained in $\delta + 2$ cycles, i.e., one cycle later. For the convolutional neural networks, the successive layer operations can be overlapped utilizing the digit-level pipelining as shown in Fig. 4.9 if we consider layer-by-layer operations, as shown by [17] and [40], the performance of online pipelined SoP will be similar. However, upon exploiting the advantages of online arithmetic and starting the computation of the next layer just after receiving the first MSD, the number of cycles to compute the successive layers will be much fewer compared to other works.

Table. 4.5 shows the required number of execution cycles in individual layers using pipelined online SoP assuming we have no limitations on the hardware

조선대학교
CHOSUN UNIVERSITY

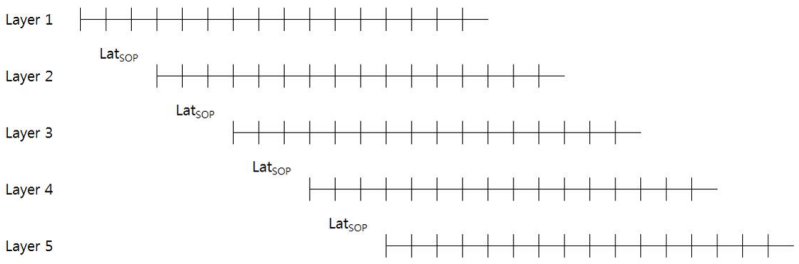

Figure 4.9: Possibility to pipeline successive layers of Alexnet using pipelined online SoP.

resources and can have all hardware to exploit data parallelism available in SoP. To count the maximum number of multipliers needed, we realize the CNN operation in AlexNet in terms of matrix multiplication. For the first layer, the input is 227x227x3 which is to be convolved with 11x11x3 filters at stride 4, then we would take 11x11x3 blocks of pixels in the input and stretch each block into a vector of size $11*11*3 = 363$. Iterating this process in the input at stride of 4 gives ($\frac{227-11}{4}+1 = 55$) locations along both width and height, leading to a matrix of size 363x3025 where every column is a stretched out receptive field and there are $55*55 = 3025$ of them in total. Note that since the receptive fields overlap, every number in the input volume may be duplicated in multiple distinct columns. The weights of the CONV layer are similarly stretched out into rows. For example, if there are 48 filters of size 11x11x3, this would give a weight matrix of size 48x363. The result of a convolution is now equivalent to performing one large matrix multiply and would give a matrix 48x3025 which evaluates the dot product between every filter and every receptive field location. The result must finally be reshaped back to its proper output dimension 55x55x48. Using the same number of T_m and T_n as in [40], the number of multipliers to execute parallel

조선대학교
CHOSUN UNIVERSITY

multiplications would be $T_m * T_N * K * K$. The number of execution cycles can be computed using relation (4.4).

$$= \left\lceil \frac{M}{T_m} \right\rceil * \left\lceil \frac{N}{T_n} \right\rceil * R * C + \lceil log_2(P) \rceil * \delta_{add} + \delta_{mul} + n, \tag{4.4}$$

where $P = K * K * T_n$.

Table 4.5: Number of execution cycles to compute convolution in AlexNet using the proposed pipelined online SoP for certain tile size.

Layer	T_m, T_n	K	**No. of operations**	**Execution Cycles**
1	48,3	11	105415200	3054
2	20,24	5	223948800	774
3	96,5	3	149520384	296
4	95,5	3	112140288	309
5	32,15	3	74760192	248

It is noteworthy that unlike conventional multipliers used in other works, the cycle time of pipelined online multiplier and adder is independent of bit precision. The cycle time or the critical path for different bit precision of online and conventional multipliers are shown in Table. 4.6.

Table 4.6: The critical path delay of various multipliers for different precision in nanoseconds (ns) computed using Synopsys Design Compiler on 45nm technology.

Precision	**Non-Pipelined Online**	**Pipelined Online**	**Serial Parallel**	**Array**	**Tree**
8	0.75	0.75	0.84	0.90	0.88
16	0.75	0.75	0.90	1.10	1.10
32	0.75	0.75	1.44	2.0	1.60

조선대학교
CHOSUN UNIVERSITY

If the online SoP design uses similar hardware configurations as in the works [17] and [40] which uses serial-parallel and array type multiplier respectively, the online SoP will have faster response time.

Moreover, the proposed reduced working precision pipelined online multiplier occupies less area and consumes less power compared to the pipelined online multiplier with full working precision. Here, we compare the resource utilization of full and reduced working precision multiplier when employed in CNN accelerator. Table. 4.7 shows the comparison of area utilization and power consumption between the pipelined online multipliers with full and reduced working precision considering the configurations of output and input tiles (T_m,T_n) provided in [40].

Table 4.7: Comparison of resource utilization for 8-bits full and reduced working precision pipelined online multipliers to perform convolution in AlexNet. Synthesis carried on Synopsys Design Compiler using 45nm TSMC technology.

Layer	T_m,T_n	No. of multipliers	Area(μm^2)		Power(mW)	
			Full working precision	Reduced working precision	Full working precision	Reduced working precision
1	48,3	17424	121969370.70	90160488.00	125243.71	93810.81
2	20,24	12000	8400943.99	62094000.00	86256.00	64608.00
3	96,5	4320	30240339.84	22353840.00	31052.16	23258.88
4	95,5	4275	29925336.3	22120987.50	30728.70	23016.60
5	32,15	4320	30240339.84	22353840.00	31052.16	23258.88
Reduction %				26.07		25.09

4.6 Chapter Summary

This chapter presented the discussion of utilizing pipelined online SoP in convolutional neural network accelerators. The benefits of the proposed architecture has been shown in terms of latency. The comparison of the proposed

조선대학교
CHOSUN UNIVERSITY

reduced working precision multiplier has been compared with full working precision multiplier which show significant savings when utilized in a CNN accelerator.

조선대학교
CHOSUN UNIVERSITY

5 Summary and Conclusion

Online arithmetic provides opportunities to pipeline the data at digit level. The serial computation of digits from most significant to least significant side make them suitable for use in area constraint devices. They require fewer modules for the generation of result as compared to the conventional arithmetic algorithms. Online arithmetic algorithms have been utilized for many signal processing applications and can be revisited for their adoption in the deep learning algorithms.

This dissertation presents the design and evaluation of online arithmetic based low power pipelined multiplier in which the switching activity has been reduced by instantiating only the required number of modules in a given step. The scheme of reducing the maximum working precision has also been employed due to which the total number of digit slices are reduced.

To compute the sum of products generated by the multiplier, online adder has been employed which is also pipelined. The fine grained optimization has been applied in which each individual module has been carefully observed for the possible reduction in the switching activity. A complete design for the computation of SoP has been discussed which can be utilized for the computation of convolution in the CNNs.

The results for synthesis and FPGA implementation of the proposed low power design have been presented which show significant reduction of area and power utilization when compared to the pipelined multiplier with full working precision. Furthermore, the cycle time of online modules have been reported and compared with the conventional arithmetic based modules which show that the proposed methodology has a constant cycle time i.e., independent of

조선대학교
CHOSUN UNIVERSITY

precision. This makes the proposed design a favorable candidate to be employed in computation of massive inner product arrays in convolutional neural networks with reduced response time.

조선대학교
CHOSUN UNIVERSITY

PUBLICATIONS

1. M. Usman, L. Jeong-A, and M. D. Ercegovac, “Multiplier with reduced activities and minimized interconnect for inner product arrays,” in 2021 55th Asilomar Conference on Signals, Systems, and Computers. IEEE, 2021.

2. Umar Afzaal, Abdus Sami Hassan, Muhammad Usman and Jeong A. Lee. ”On the Evolutionary Synthesis of Fault-resilient Arithmetic Circuits”. IEEE Transactions on Evolutionary Computation. (2021) (Under review at the time of thesis submission)

조선대학교
CHOSUN UNIVERSITY

BIBLIOGRAPHY

[1] Abderazek Ben Abdallah. “Power Optimization Techniques for Multicore SoCs”. In: *Advanced Multicore Systems-On-Chip*. Springer, 2017, pp. 225–244.

[2] Tooba Arifeen et al. “Adder with Reduced Activities and Minimized Interconnect for Streaming Inner Products”. In: *2021 55th Asilomar Conference on Signals, Systems, and Computers*. IEEE. 2021, pp. –.

[3] Robert Brayton and Alan Mishchenko. “ABC: An academic industrial-strength verification tool”. In: *International Conference on Computer Aided Verification*. Springer. 2010, pp. 24–40.

[4] Yen-Jen Chang et al. “A Low Power Radix-4 Booth Multiplier With Pre-Encoded Mechanism”. In: *IEEE Access* 8 (2020), pp. 114842–114853.

[5] Yu-Hsin Chen et al. “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks”. In: *IEEE journal of solid-state circuits* 52.1 (2016), pp. 127–138.

[6] Jason Cong and Bingjun Xiao. “Minimizing computation in convolutional neural networks”. In: *International conference on artificial neural networks*. Springer. 2014, pp. 281–290.

[7] Martin Dimmler et al. “On-line arithmetic for real-time control of microsystems”. In: *IEEE/ASME transactions on mechatronics* 4.2 (1999), pp. 213–217.

[8] Pouya Dormiani et al. “A design of online scheme for evaluation of multinomials”. In: *Advanced Signal Processing Algorithms, Architectures,*

조선대학교
CHOSUN UNIVERSITY

and Implementations XV. Vol. 5910. International Society for Optics and Photonics. 2005, 59100S.

[9] Miloš D Ercegovac. "On Reducing Module Activities in Online Arithmetic Operations". In: *2020 54th Asilomar Conference on Signals, Systems, and Computers*. IEEE. 2020, pp. 524–528.

[10] Milos D Ercegovac and Tomas Lang. *Digital arithmetic*. Elsevier, 2004.

[11] Milos D. Ercegovac and Tomas Lang. "Fast multiplication without carry-propagate addition". In: *IEEE Transactions on Computers* 39.11 (1990), pp. 1385–1390.

[12] Milos D Ercegovac and Tomas Lang. "On-the-fly conversion of redundant into conventional representations". In: *IEEE Transactions on Computers* 7 (1987), pp. 895–897.

[13] MilošD Ercegovac and Tomas Lang. "On-line scheme for computing rotation factors". In: *Journal of Parallel and distributed computing* 5.3 (1988), pp. 209–227.

[14] Reto Galli and Alexandre F Tenca. "A design methodology for networks of online modules and its application to the levinson-durbin algorithm". In: *IEEE transactions on very large scale integration (VLSI) systems* 12.1 (2004), pp. 52–66.

[15] Reto Galli and Alexandre F Tenca. "Design and evaluation of online arithmetic for signal processing applications on FPGAs". In: *Advanced Signal Processing Algorithms, Architectures, and Implementations XI*. Vol. 4474. International Society for Optics and Photonics. 2001, pp. 134–144.

조선대학교
CHOSUN UNIVERSITY

[16] Aksenti L Grnarov and Milos D Ercegovac. “On-line multiplicative normalization”. In: *1983 IEEE 6th Symposium on Computer Arithmetic (ARITH)*. IEEE. 1983, pp. 151–155.

[17] Lien-Chih Hsu et al. “Essa: An energy-aware bit-serial streaming deep convolutional neural network accelerator”. In: *Journal of Systems Architecture* 111 (2020), p. 101831.

[18] Zhijun Huang and Milos D Ercegovac. “FPGA implementation of pipelined on-line scheme for 3-D vector normalization”. In: *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM’01)*. IEEE. 2001, pp. 61–70.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[20] He Li et al. “ARCHITECT: Arbitrary-precision hardware with digit elision for efficient iterative compute”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.2 (2019), pp. 516–529.

[21] Yue-Jin Lin and Tian Sheuan Chang. “Data and hardware efficient design for convolutional neural network”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.5 (2017), pp. 1642–1651.

[22] Emeka Mosanya and Eduardo Sanchez. “A FPGA-based hardware implementation of generalized profile search using online arithmetic”. In: *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*. 1999, pp. 101–111.

조선대학교
CHOSUN UNIVERSITY

[23] Muteen Munawar et al. “Low power and high speed Dadda multiplier using carry select adder with binary to excess-1 converter”. In: *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*. IEEE. 2020, pp. 1–4.

[24] William G Natter and Behrouz Nowrouzian. “Digit-serial online arithmetic for high-speed digital signal processing applications”. In: *Conference Record of Thirty-Fifth Asilomar Conference on Signals, Systems and Computers (Cat. No. 01CH37256)*. Vol. 1. IEEE. 2001, pp. 171–176.

[25] Alan V Oppenheim and Ronald W Schafer. “DISCRETE-TIME SIGNAL PROCESSING”. In: (1989).

[26] Robert Michael Owens. “Compound algorithms for digit online arithmetic”. In: *1981 IEEE 5th Symposium on Computer Arithmetic (ARITH)*. IEEE. 1981, pp. 64–71.

[27] Christian Piguet. *Low-power electronics design*. CRC press, 2018.

[28] Cauligi S Raghavendra and Milos D Ercegovac. “A simulator for on-line arithmetic”. In: *1981 IEEE 5th Symposium on Computer Arithmetic (ARITH)*. IEEE. 1981, pp. 92–98.

[29] Charbel Sakr and Naresh Shanbhag. “An analytical method to determine minimum per-layer precision of deep neural networks”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 1090–1094.

[30] Kan Shi, David Boland, and George A Constantinides. “Efficient FPGA implementation of digit parallel online arithmetic operators”. In: *2014*

조선대학교
CHOSUN UNIVERSITY

International Conference on Field-Programmable Technology (FPT). IEEE. 2014, pp. 115–122.

[31] Xian Su et al. “Local re-encoding for coded matrix multiplication”. In: *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2020, pp. 221–226.

[32] Kishor S Trivedi and Milos D Ercegovac. “On-line algorithms for division and multiplication”. In: *Computers, IEEE Transactions on* 100.7 (2006), pp. 681–687.

[33] Kishor S Trivedi and Joseph G Rusnak. “Higher radix on-line division”. In: *1978 IEEE 4th Symposium onomputer Arithmetic (ARITH)*. IEEE. 1978, pp. 164–174.

[34] Dean M Tullsen and Milos D Ercegovac. “Design and VLSI implementation of an on-line algorithm”. In: *Real-Time Signal Processing IX*. Vol. 698. International Society for Optics and Photonics. 1986, pp. 92–99.

[35] Muhammad Usman, Lee Jeong-A, and Miloš D Ercegovac. “Multiplier with Reduced Activities and Minimized Interconnect for Inner Product Arrays”. In: *2021 55th Asilomar Conference on Signals, Systems, and Computers*. IEEE. 2021, pp. –.

[36] Zdenek Vasicek and Lukas Sekanina. “Evolutionary approach to approximate digital circuits design”. In: *IEEE Transactions on Evolutionary Computation* 19.3 (2014), pp. 432–444.

조선대학교
CHOSUN UNIVERSITY

[37] Clifford Wolf, Johann Glaser, and Johannes Kepler. “Yosys-a free Verilog synthesis suite”. In: *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*. 2013.

[38] Melih Yildirim. “Design of Low-Voltage and Low-Power DTMOS Based Analog Multiplier Utilizing Current Squarer”. In: *International Journal of Electronics Letters* 9.1 (2021), pp. 1–13.

[39] Pegah Zakian and Rahebeh Niaraki Asli. “An efficient design of low-power and high-speed approximate compressor in FinFET technology”. In: *Computers & Electrical Engineering* 86 (2020), p. 106651.

[40] Chen Zhang et al. “Optimizing fpga-based accelerator design for deep convolutional neural networks”. In: *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*. 2015, pp. 161–170.

[41] Yiren Zhao, John Wickerson, and George A Constantinides. “An efficient implementation of online arithmetic”. In: *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE. 2016, pp. 69–76.