UNIVERSITY OF CALIFORNIA,
IRVINE

Approximate and Bit-width Configurable Arithmetic Logic Unit Design for Deep Learning Accelerator

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Xiaoliang Chen

2020

DEDICATION

To the memory of my grandparents,
To my parents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my research advisor, Professor Fadi J. Kurdahi, who provided me with support and guidance in all aspects of my research. He always encourages me to think creatively and the discussions with him always helped me to overcome challenges I faced in research. This dissertation would not have been possible without his persistent support and guidance.

I would also like to thank my co-advisor Prof. Ahmed M. Eltawail. The discussions with him always inspired me to think deeper.

I would also like to thank Prof. Rainer Doemer for dedicating his time to be in my committee.

I would especially like to thank all friends met in UCI for making the time at Irvine a great experience, and my colleagues at Broadcom for providing generous help during work.

Last but not least, I would like to thank my parents, brothers and sisters who were always supporting me and encouraging me with their best wishes.

VITA

Xiaoliang Chen

EDUCATION

Doctor of Philosophy in Electrical and Computer Engineering **2020**
University of California, Irvine *Irvine, CA*

M.Sc. in Computerl Sciences **2010**
Peking University *Beijing, China*

B.Sc. in Electrical Engineering **2007**
Tsinghua University *Beijing, China*

EXPERIENCE

Staff Engineer **2012 –present**
Broadcom Inc. *Irvine, CA*

Design Engineer **Feb 2010 – Dec 2010**
MediaTek Inc (Beijing). *Beijing, China*

PUBLICATIONS

- Xiaoliang Chen, and F. J. Kurdahi, "BC-MAC: Bit-width Configurable MAC for Deep Learning Accelerator", *To be submitted to IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*
- Xiaoliang Chen, and F. J. Kurdahi, "Approximate Compressor based Speculative Multi-operand Adder with Error Recovery", *To be submitted to 2021 Design Automation & Test in Europe Conference (DATE).*
- Xiaoliang Chen, A. M. Eltawil and F. J. Kurdahi, "Low Latency Approximate Adder for Highly Correlated Input Streams", *2017 IEEE International Conference on Computer Design (ICCD)*, Boston, MA 2017.

ABSTRACT OF THE DISSERTATION

Approximate and Bit-width Configurable Arithmetic Logic Unit Design for Deep Learning Accelerator

By

Xiaoliang Chen

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2020

Professor Fadi J. Kurdahi, Chair

As key building blocks for digital signal processing, image processing and deep learning etc, adders, multi-operand adders and multiply-accumulator unit (MAC) have drawn lots of attention recently. Two popular ways to improve arithmetic logic unit (ALU) performance and energy efficiency are approximate computing and precision scalable design. Approximate computing helps achieve better performance or energy efficiency by trading accuracy. Precision scalable design provides the capability of allocating just-enough hardware resources to meet the application requirements.

In this thesis, we first present a correlation aware predictor (CAP) based approximate adder, which utilizes spatial-temporal correlation information of input streams to predict carry-in signals for sub-block adders. CAP uses less prediction bits to reduce the overall adder delay. For highly correlated input streams, we found that CAP can reduce adder delay by ∼23.33% and save ∼15.9% area at the same error rate compared to prior works.

Inspired by the success of approximate multipliers using approximate compressors, we proposed a pipelined approximate compressor based speculative multi-operand adder (AC-MOA). All compressors are replaced with approximate ones to reduce the overall delay of the bit-array reduction tree. An efficient error detection and correction block is designed to compensate the errors with one extra cycle. Experimental results showed the proposed 8-bit 8-operand AC-MOA achieved 1.47X $\sim$ 1.66X speedup over conventional baseline design.

Recent research works on deep learning algorithms showed that bit-width can be reduced without losing accuracy. To benefit from the fact that bit-width requirement varies across deep learning applications, bit-width configurable designs can be used to improve hardware efficiency. In this thesis a bit-width configurable MAC (BC-MAC) is proposed. BC-MAC uses spatial-temporal approach to support variable precision requirements for both of activations and weights. The basic processing element (PE) of BC-MAC is a multi-operand adder. Multiple multi-operand adders can be combined together to support input operands of any precision. Bit-serial summation is used to accumulate partial addition results to perform MAC operations. Booth encoding is employed to further boost the throughput. Synthesis results on TSMC 16nm technology and simulation results show the proposed MAC achieves higher area efficiency and energy efficiency than the state-of-the-art designs, making it a promising ALU for deep learning accelerators.

Chapter 1

Introduction

1.1 Motivation

High performance and energy efficient computing technologies are always needed for future. With the development of semiconductor technologies, some computation intensive algorithms and applications become popular. As an example, deep learning algorithms, which were considered to be too computation intensive to realize, now gained lots of attentions due to its unprecedented accuracy on many AI applications such as face recognition, image classification, object detection, speech recognition and translation etc [1, 2]. The success motivates people to explore computing technologies to achieve higher performance and energy efficiency.

Approximate computing has become a well-known computing technique in recent years [3, 4, 5]. Recently many research works on applying approximate computing techniques to key arithmetic logic units are published, including adders, multipliers, multi-operand adders, multiply-accumulate unit etc [6, 7]. Approximate computing relies on the ability of many systems and applications to tolerate some loss of quality or optimality in the computed result.

The main idea is to exploit the inherent error resiliency or error tolerance of the system to achieve energy efficiency, or trading accuracy with energy consumption. Such trade-off, in most cases, is also associated with performance improvements like faster operations, area reduction etc.

Bit-width scalable design is another direction to improve energy efficiency. Bit-width scalable design can allocate just enough hardware resources for computation. Many applications may not necessarily need full precision. Run time configurable design provides the flexibility to reallocate hardware resources and therefore improve overall computing efficiency. Deep learning gained lots of attentions due to its unprecedented accuracy on many AI applications. The demands of high computations, mostly multiply-accumulate (MAC) operations, and the need for high energy efficiency motivated designing dedicated hardware accelerators to accommodate DNN computations. Recent research work shows that the bit-width of MAC operations can be reduced without loss on accuracy [8]. While many research works exploit high degree of parallelism at word level with fixed-bit-width design hardware accelerators supporting variable bit-width are introduced to benefit from the fact that bit-width requirements varies significantly across applications and layers of DNNs [9].

As a result of these challenges and opportunities, arithmetic logic units need to be revisited with new design techniques to pave the way for the need of emerging computation intensive applications.

1.2 Organization and Contributions

Chapter 1 is introduction and organization.

In chapter 2, we focused on approximate adders and proposed new carry predictor to reduce error rate as well as latency for highly correlated input streams. We first studied the correlations of real input streams as well as carry signals. Results showed that the correlation information can be used to reduce the error rate and latency for carry prediction. We proposed a Correlation Aware Predictor (CAP) utilizing correlation information of input streams to predict carry-in signal value for sub-adders. Analysis and simulation results showed that a CAP based approximate adder can reduce latency significantly for highly correlated input streams compared with state-of-art approximate adders. We experimented with a CAP based approximate adder processing 24-bit audio streams. Simulation results show that the proposed approximate adder can significantly reduce latency and area for real applications.

In chapter 3, we proposed an approximate compressor based pipelined speculative multi-operand adder (AC-MOA). All compressors are replaced with approximate ones to reduce the overall delay of the bit-array reduction tree. A light error correction scheme is adopted to cover the cases when error distance is 1 without stalling the pipeline. Otherwise an extra cycle is required to correct the final results. To further improve the overall performance, the proposed AC-MOA can operate in approximate mode when error correction signals of the least significant bit-blocks are masked. We implemented an 8-bit 8-operand AC-MOA in Verilog and synthesized with TSMC 16nm library. Simulation results show that the proposed AC-MOA achieves 1.25X$\sim$1.49X and 1.68X$\sim$1.78X improvement on latency in accurate mode and approximate mode respectively, compared with the conventional design.

In chapter 4, we present BC-MAC, a bit-width configurable MAC array which can be reconfigured at run time to match the precision requirements. BC-MAC uses spatial-temporal approach to support variable precision requirements for both of activations and weights. The basic processing element (PE) of BC-MAC is a multi-operand adder. Multiple PEs can be combined together to support input operands of any precisions. Bit-serial summation is used to accumulate partial addition results to perform MAC operations. In this way the accelerator can save either hardware resources or execution cycles when the required bit-width is reduced. Area overheads on reconfiguration logic are minimal compared with state-of-the-art designs. Synthesis results on TSMC 16nm technology and simulation results show the proposed MAC achieves higher area efficiency and energy efficiency than the state-of-the-art designs, making it a promising ALU for deep learning accelerators.

Conclusion and future work are presented in chapter 5.

Chapter 2

Low Latency Approximate Adder for Highly Correlated Input Streams

As a key arithmetic component, approximate addition [3, 4, 5] has drawn lots of attentions recently and many approximate adders have been proposed and studied [10, 11, 12, 13, 14, 15]. Most approximate adders are composed of multiple sub-adders and long carry chains are split to reduce latency, thus benefiting from the fact that carry propagation across long carry chains is rare for uniformly distributed inputs. One key tradeoff of these approximate adders is between latency and error rate. The more prediction bits are used, the lower is the error rate, but the latency is longer. However, the assumption that input streams follow a uniform distribution is not always true. As pointed out in [16], multimedia signals like music have signal distributions that are closer to Gaussian shape, and therefore addition error probability can be very different compared with the uniform case.

To show the importance of input characteristics, we take a counter as an example. For an 8-bit counter with two 4-bit sub-adders, the upper sub-adder's carry-in signal is predicted by the lower neighboring bits. If the counter increments by 1 in each cycle, the carry-in signal for upper sub-adder is 1 only when lower sub-adder counts to 15, otherwise it's always 0. The actual error rate is 6.25% if the carry input is tied to 0. However, if we assume uniformly distributed signal values, the error rate would be 46.8% according to [12] when the number of prediction bits is 0. This example clearly shows that input distribution has a strong impact on error rate. Uniform distribution assumption is insufficient and may incur over/under design for real applications [16].

We propose new carry predictor to reduce error rate as well as latency for highly correlated input streams. Our major contributions are as follows:

1. We study correlations of real input streams as well as carry signals. We show that correlation information can be used to reduce the error rate and latency for carry prediction.

2. We propose a Correlation Aware Predictor (CAP) utilizing correlation information of input streams to predict carry-in signal value for sub-adders. Analysis and simulation results show that a CAP based approximate adder can reduce latency significantly for highly correlated input streams compared with state-of-art approximate adders.

3. We experimented with a CAP based approximate adder processing 24-bit audio streams. Simulation results show that the proposed approximate adder can significantly reduce latency and area for real applications.

The rest of the chapter is organized as follows. In Section 2.1, we present background information on approximate adders and related work in the area is reviewed; Section 2.2 presents our study of correlations of input streams and carry-out signals; Section 2.3 capitalizes on the correlation study results to introduce the proposed correlation-aware predictor (CAP) and compared the performance with previous work; Section 2.4 reports the experimental results of proposed adder for 24-bit audio signal processing; Section 2.5 is the conclusion.

2.1 Background

In this section, we review the basics of approximate adders including basic structures, error conditions, and error detection and correction mechanisms. Without loss of generality, we will examine the Generic Accuracy configurable adder (GeAr) [15] as a representative approximate adder design in more detail.

2.1.1 Block-based Approximate Adders

Most block-based approximate adder designs are designed by utilizing the fact that carry propagation across long carry chains is rare. This implies that breaking long carry chain doesn't induce errors frequently. In many designs, a long addition is divided into smaller sub-adders which are executed in parallel. Therefore the longest carry propagation chain is related to the size of sub-adders.

Figure 2.1 illustrates the circuit structures of group adders in GeAr. A GeAr adder can be completely defined by three parameters **(N, R, P)** where

- **N**: the length of operands to be added
- **R**: the number of result bits contributing to final sum
- **P**: the number of prediction bits of each block adder

GeAr adder uses **K (R+P)**-bit sub-adders in parallel to perform the approximate addition and **K** can be calculated by

$$K = \frac{N - R - P}{R} + 1 \tag{2.1}$$

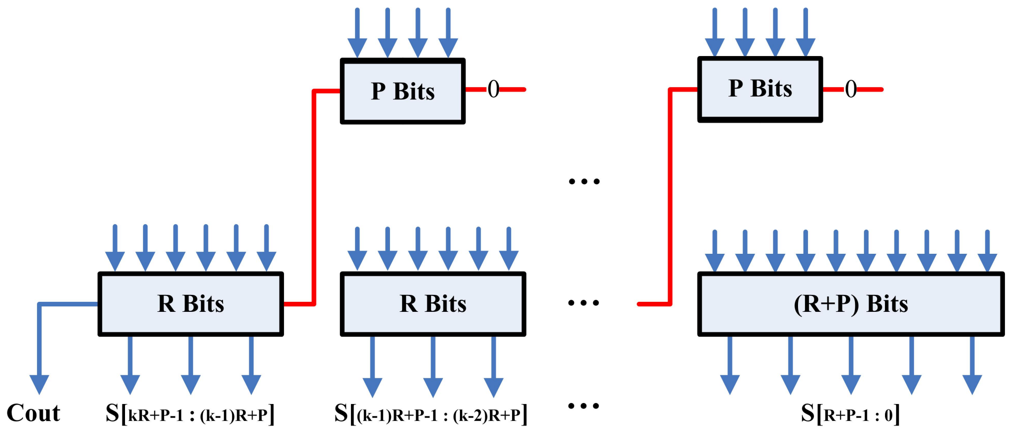


Figure 2.1: Block diagram of GeAr adder

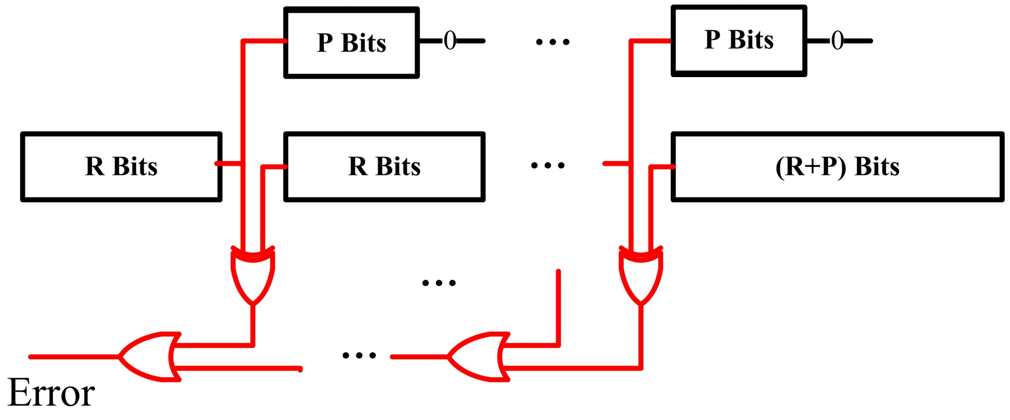


Figure 2.2: Error Detection Unit

Compared to an N-bit adder, the delay is significantly reduced since the longest carry propagation is no more than (R+P) bits. The fewer predictive bits are used, the shorter is the delay.

2.1.2 Error Condition

From the approximate adder model, it's easy to figure out that the approximate adder has errors when for any of sub-adders $k (0 \leq k \leq N)$, the carry-in signal is mispredicted. An error occurs in an individual sub-adder when both of the following conditions occur:

1. The carry-in of sub-adder k is 1, not 0;

2. All P predictive bits of sub-adder k propagate carry-in

For each bit i ($0 \leq i \leq N-1$) in the adder, carry generation, kill and propagation signals are defined as

- g_i: $g_i = 1$ if both operands are 1 at position i;
- k_i: $k_i = 1$ if both operands are 0 at position i;
- p_i: $p_i = 1$ if two operands differ at position i.

Similarly, for a group of input bits range from i to j, group carry generation, kill and propagation can be defined as

- $G(i:j)$: $G(i:j) = 1$ if carry-out of group inputs $(i:j)$ is 1
- $K(i:j)$: $K(i:j) = 1$ if carry-out of group inputs $(i:j)$ is 0
- $P(i:j)$: $P(i:j) = 1$ if for every bit k in $(i:j)$ p_k is 1

With the above definition, the error condition can be presented as

1. $G_{(k-1)R}$ =1 and
2. $P((k-1)R:(k-1)R+P) = p_{(k-1)R+1}{*}p_{(k-1)R+2} \, \ldots \, p_{(k-1)R+P} = 1$

Many research works [15, 17, 18] reported approximate adder error rate under uniformly distributed inputs. Generally, the more prediction bits are used, the lower is the error rate, but the longer is the adder delay.

2.1.3 Error Detection and Correction

For each group addition, the error detection circuit compares the carryout value from the previous window with the carry-out value generated by the predictor, as shown in Figure 2.2. If the circuit detects a mismatch between these two values in any of the group adders, it means that the predictor of the group adder fails to predict the correct carry-in value. Thus the final result will be wrong and error detection circuit will flag an error. Otherwise the result is guaranteed to be correct.

2.1.4 Related Work

In [10] the Almost Correct Adder (ACA-I) is proposed where R=1. The Error Tolerant Adder Type II (ETA-II), proposed in [11], is the case that R=P. In Accuracy Configurable Approximate Adder (ACAA) [13], can be configured at runtime to achieve different accuracy levels. In [14], a reconfigurable approximate adder is proposed and computation quality degrades gracefully. In [19] the author proposed correlation-aware speculative adder (CASA) based on the observation there is a high chance that carry-in to groups adder equals 1 when xor(msb) =1. In [20] the author proposed history aware adder by exploiting the temporal operand correlation of each instruction location. In [15], the author proposed a low-latency Generic Accuracy Configurable Adder (GeAr) , which is a general model covering larger design space. Since the GeAr topology is configurable and covers both of ETA-II and ACAA as special cases , in this paper, without lack of generality, we will use GeAr approximate adders as baseline design.

2.2 Study of Correlated Input Streams

In this section, we study correlations of real input streams as well as carry-in signal correlations.

2.2.1 Correlations of Input Streams

In digital signal processing, random signals with uniform distribution are often considered as white noise. Real signals typically have certain correlations. Correlations of input streams can be classified as spatial correlation and temporal correlation. To better explain input stream correlations, we use a 20s pop song with 24-bit in 2's complement encoding as an example [21]. Figure 2.3(a) is the signal waveform and figure 2.3(b) is the histogram of the signal which is close to a Gaussian distribution.

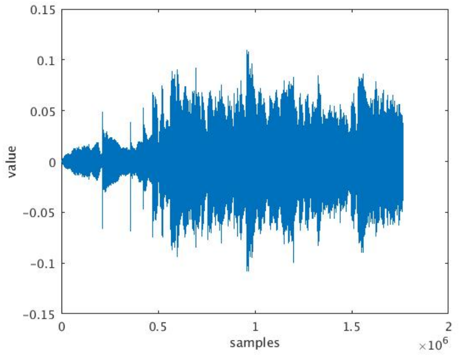
1. **Spatial correlation** Spatial correlation measures the similarity of switching behavior between neighboring bits. As shown in Figure 2.3(b), the input signal follows a zero-mean Gaussian distribution with a standard deviation of 0.022. Small deviation indicates that the range of input signal is limited and there are fewer "active" bits. Figure 2.3(c) shows the switching probability of audio signal for each bit. While lower bits have high switching activity and behave like random signals, the switching probabilities of upper bits switch much less often. Besides, neighboring bits usually have similar switching probability. For example, switching probabilities of the most significant 8 bits are less than 6% and for MSB, it is only 1.66%. In contrast, the switching probabilities of the lowest 10 bits are 50%. We refer to this similarity of switching behavior between neighboring bits as spatial correlation. In this case, the upper bits have higher spatial correlation than lower bits.

2. **Temporal correlation** Temporal correlation of a signal measures the deviation of

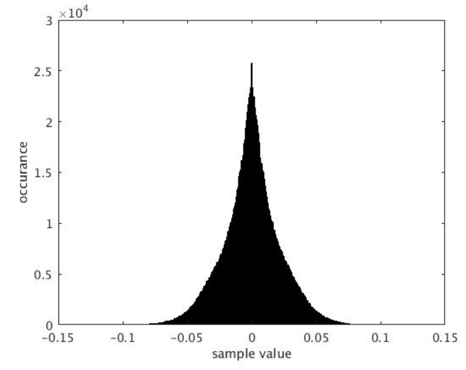
differences between two successive inputs. Fig 2.3(d) is the histogram of differences of consecutive inputs. Results show that differences between consecutive inputs are very small and follow a Gaussian distribution whose is 0 and standard deviation is 0.0013, which is much smaller than the input signal's standard deviation (0.022). Another observation is the signs of differences don't change frequently. This is due to the fact that low frequency components are dominant and signal value switches from increasing to decreasing (or vice versa) are infrequent.

2.2.2 Correlations of Carry Signals

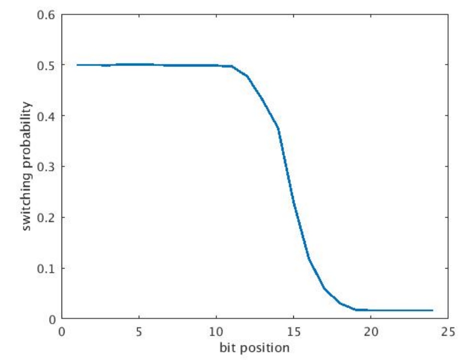
Given the observation that spatial and temporal correlations do exist in real input streams especially for MSBs, we hypothesize there is a high probablility that internal carry value keeps unchanged for additions (or substractions) on highly correlated input streams. Consider addition operations on two correlated input streams, $a_{t0} + b_{t0}$ and $a_{t1} + b_{t1}$. If Δa and Δb are small, only the lower bits of input operands switch and carry-in value of upper sub-adders rarely changes. These additions only impact internal carries of sub-adder for LSBs. Another case is signed subtraction on delayed signal, $a_{t1} - a_{t0}$, $a_{t2} - a_{t1}$. When the input stream keeps increasing (or decreasing), the signs of subtraction results don't change hence internal carry value doesn't switch. Figure 2.4(a) and Figure 2.4(b) show the probability that xor results of successive carry-in signals of each bit for unsigned addition and signed subtraction, respectively. For MSBs, there is a strong correlation between successive carry signal values. For LSBs, the correlation is weak. The temporal correlation of carry signals motivates us to exploit the correlations in carry prediction.



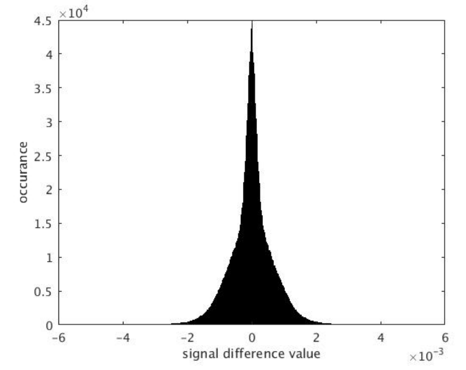
(a) Audio signal waveform



(b) Histogram of audio signal

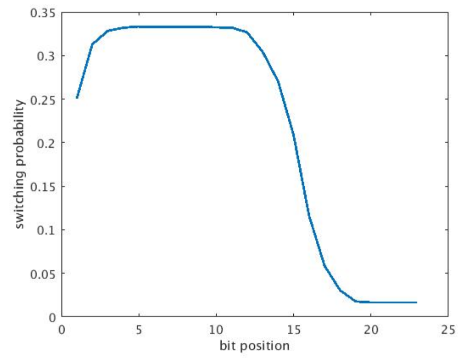


(c) Input switching probability at differnt bit position

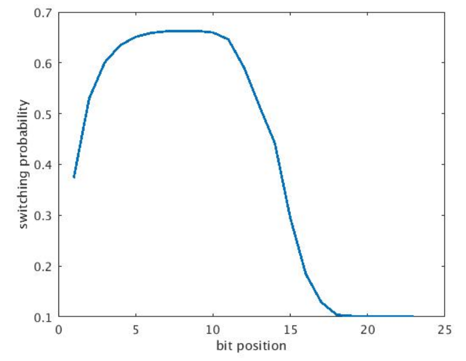


(d) Histogram of differences between successive inputs

Figure 2.3: Input signal characteristics



(a) Carry signal bit switching probability of addition operation



(b) Carry signal bit switching probability of subtraction operation

Figure 2.4: Carry signal characteristics

2.3 CAP Based Approximate Adder Design

In this section, we present the design of correlation aware predictor (CAP) based approximate adder and evaluate its performance. CAP is an extension to existing carry-in predictors such as GeAr [15], which is considered as baseline design for comparison.

2.3.1 CAP Based Approximate Adder

We proposed correlation aware predictor (CAP) as shown in Fig. 2.5 which is a mux based predictor. Control signals are group generate (G) and kill (K) results from prediction block. Predicted carry-in gives correct value when G (or K) is 1. When the value of both of G and K are zero, an estimated value is needs since predictive bits are not sufficient to get correct value. In most existing predictors such as GeAr, the estimated value is fixed at zero. To reduce error rate and latency, CAP uses a flip-flop to store the carry-in result of the prior operation, which can be used as estimated value in this case. As discussed in section III, the prior result could be a better guess for strongly correlated bits of highly correlated input streams. The error detection and correction mechanism of the CAP based approximate adder is similar to GeAr [15]. Fig. 2.6 shows the structural diagram of CAP based approximate adder. For each carry predictor, a XOR gate is used to compare predicted carry-in with the carry-out of neighboring sub-adder. Once a mismatch is found, the error detection circuit will flag an error and extra cycles are needed to correct the output. Flip-flops storing prior results will also be updated. The result is considered to be correct as long as no mismatch is detected.

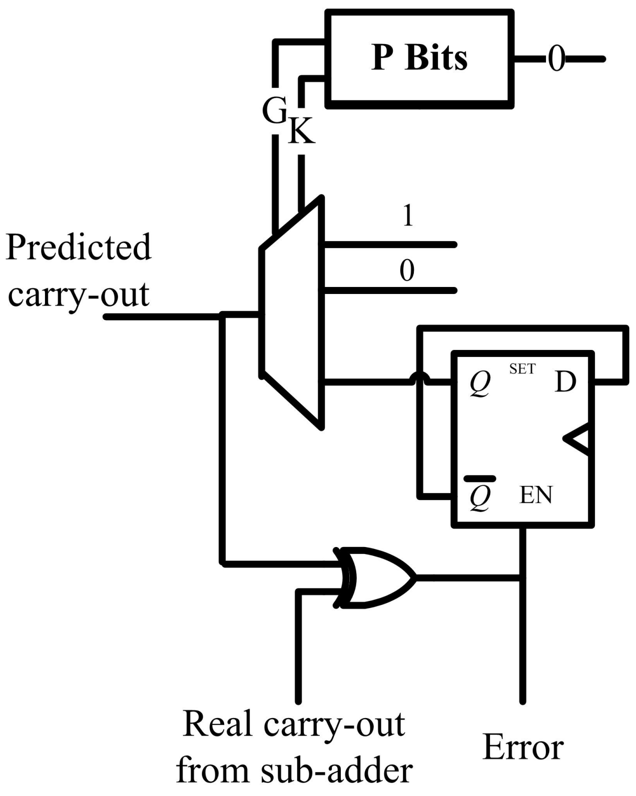


Figure 2.5: Implementation of correlation aware predictor

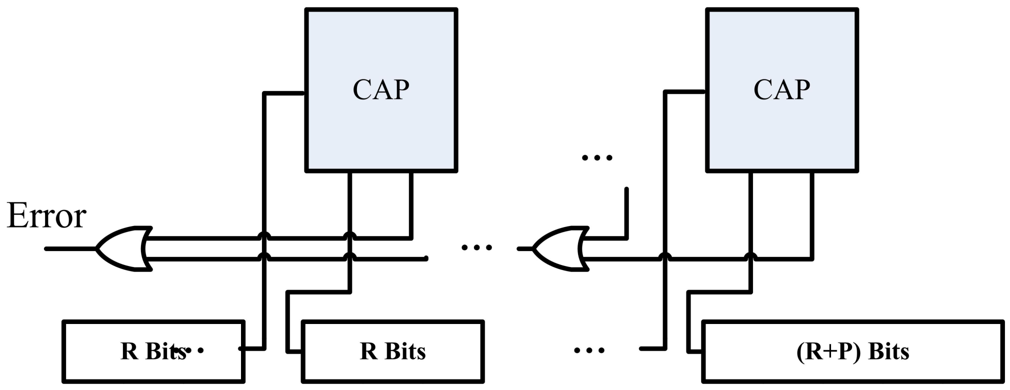


Figure 2.6: Diagram of CAP based approximate adder

2.3.2 Error Rate Analysis

To verify the effectiveness of CAP, we analyze error rate for the following cases, including 1) unsigned addition under uniform distributed inputs 2) unsigned addition under spatial correlated inputs and 3) signed subtraction under sine wave inputs.

Unsigned addition under uniformly distributed inputs

Although CAP is designed for highly correlated input streams, the performance is not degraded under uniformly distributed random inputs. In [15, 17, 18] the error probability of GeAr adder for uniformly distributed inputs is discussed with the help of a complex model. In [16] an approximate closed-form equation is proposed which can simplify error rate analysis. Take an GeAr adder with configuration (N,R,P) as an example, there are $K = (N - P)/R$ sub-adders. The first sub-adder is accurate. For the rest sub-adders, error occurs when

1. $\mathbf{G}(KR - P - 1)$==1
2. $\mathbf{P}(KR - P - 1 : KR - 1)$==1.

Therefore the error probability of the kth sub-adder is approximately equal to $2^{-P+1} * (1 - 2^{-R})$. The overall error probability of aproximate adder is

$$ER_{uniform} = 1 - (1 - 2^{-(p+1)} * (1 - 2^{-R}))^{K-1} \tag{2.2}$$

The closed form equation implies that all predictors contribute to final error rate equally. For CAP based approximate adders with the same configuration, the probability of condition 1 is the same. For the condition 2, error occurs when prior carry result is not the same as

	$P = 1$	$P = 2$	$P = 3$	$P = 6$	$P = 8$
GeAr	25.01%	12.6%	3.15%	0.77%	0.2%
CAP	24.67%	12.45%	3.23%	0.78%	0.18%

Table 2.1: Carry prediction error rate under uniform input distribution

carry state transition		Switch of input a			
		00	01	10	11
Switch of input b	00	KK	KP	PK	PP
	01	KP	KG	PP	PG
	10	PK	PP	GK	GP
	11	PP	PG	GP	GG

Table 2.2: Carry-out state transition table

current value which can be expressed as

$$XOR(carry(t-1), carry(t)) == 1$$

The probability is also 1/2 since correlation of successive carry values is 0 under uniformly distributed inputs. The overall error rate of CAP based on approximate adder is the same as GeAr under uniformly distributed inputs. Table I shows the error rate of a carry predictor with different number of prediction bits. For each case, we use 50000 random input vectors for simulation. Results confirm that CAP based approximate adder is no worse than state-of-art GeAr under uniformly distributed inputs.

Unsigned addition under switching activity biased inputs

To analyze the impact of spatial correlation on CAP, we use switching activity biased inputs to calculate error rate. For each bit of spatially correlated signal, there are four switching cases 00/11/01/10. For carry-out signal of each bit, there are 16 cases as shown in 2.2 where $G/P/K$ represent carry-out status. For example, when switches of inputs are a_{00} and b_{01}, carry-out state changes from K to P and the probability equals to $prob(a_{00})*prob(b_{01})$.

For GeAr, the probability of error condition 1 and 2 are the same as uniformly distribution case when 0/1 appears evenly for all input bits. For CAP, the probability of error condition 2 is the same as GeAr which equals to 2^{-P}, while the probability of error condition 1 $XOR(carry(t-1), carry(t)) == 1$ depends on switching activity of carry bits. For the ith bit, condition 1 is met when the group generation signal of $[i-P-R : i-P]$ under successive input vectors changes, which can be expressed as

$$XOR(carry(i,t-1), carry(i,t)) = XOR(G(i-P-R : i-P, t-1), G(i-P-R : i-P, t)) \quad (2.3)$$

Though there are nine possbile state transitions as shown in Table 2.2, only the following three ay evaluate condition 2 to true. For transitions with ending state G/K, predictor always give accurate carry values.

- **GP**: Prior carry-out is 1 and current state is P, current output will be 0 when G(i-P-R:i-P-1)==0 which evaluates (2) to true.
- **KP**: Prior carry-out is 0 and current state is P, current output will be 1 when G(i-P-R:i-P-1)==1 which evaluates (2) to true.
- **PP**: Both of prior state and current state is P, current output equals to XOR(G(i-P-R:i-P-1,t-1),G(i-P-R:i-P-1,t)) and (2) is evaluated to true if the output is 1.

To simplify the notation, we define XORG as

$$XORG(i-P-R : i-P) = XOR(G(i-P-R : i-P, t-1), G(i-P-R : i-P, t))$$

Equation 2.3 can be rewritten as

$$\begin{aligned} XORG(i-P-R:i-P) &= GP_{i-P} * G(i-P-R:i-P-1,t) \\ &+ KP_{i-P} * G(i-P-R:i-P-1,t) \\ &+ PP_{i-P} * XORG(i-P-R:i-P-1) \end{aligned}$$

To simplify analysis, we assume that switching is unbiased so that

$$prob(01) = prob(10), prob(00) = prob(11)$$

Thus we have

$$\begin{aligned} prob(GPi) &= prob(a_{10}) * prob(b_{11}) + prob(a_{11}) * prob(b_{10}) \\ &= prob(a_{00}) * prob(b_{01}) + prob(a_{01}) * prob(b_{00}) \\ &= prob(KP_i) \end{aligned}$$

The probability of condition (i) at location i can be simplified to

$$\begin{aligned} & prob(XORG(i-P-R:i-P)) \\ & = prob(GP_{i-P}) * prob(G(i-P-R:i-P-1,t) == 0) \\ & + prob(KP_{i-P}) * prob(G(i-P-R:i-P-1,t) == 1) \\ & + prob(PP_{i-P}) * prob(XORG(i-P-R:i-P-1)) \\ & = prob(GP_{i-P}) + prob(PP_{i-P}) * prob(XORG(i-P-R:i-P-1)) \\ & = prob(GP_{i-P}) + prob(PP_{i-P} * GP_{i-P-1}) + prob(PP_{i-P} * PP_{i-P-1} * GP_{i-P-2}) + \ldots \\ & + prob(PP_{i-P} * PP_{i-P-1} * \ldots * PP_{i-P-R-1} * GP_{i-P-R}) \end{aligned}$$

For sub-adder whose input bits showing high spatial correlation, we assume they have the same state transition probability,

$$prob(GP) = prob(GP_{i-P}) = \ldots = prob(GPi - P - R),$$
$$prob(PP) = prob(PP_{i-P}) = \ldots = prob(PPi - P - R),$$

Then the equation can be represented as

$$prob(XORG(i - P - R : i - P)) = prob(GP) * (1 - prob(PP)^R)/(1 - prob(PP))$$

So the probability that CAP has an error at the ith bit is

$$prob(PP)^P * prob(GP) * \frac{1 - prob(PP)^R}{1 - prob(PP)} \tag{2.4}$$

Equation 2.4 shows that for input streams with high spatial correlation, the switching activities of prediction bits $prob$(GP) and $prob$(KP) are small, and $prob$(PP) is large, the error rate of CAP can be greatly reduced. Equation 2.4 is the general form of calculating CAP carry-out error. For uniformly distributed inputs, replacing $prob$(GP), $prob$(KP) and $prob$(PP) with 0.25, 0.25 and 0.5 respectively, we get the same results in previous subsection.

To verify the analysis, we generate 32-bit highly correlated input signals with switching probability of the kth bit equals to $prob(k) = 0.5^k$, the LSB has the highest switching activity and the MSB has the lowest switching activity. Figure 2.7 shows the error rate of CAP predicted carry-out at bit 16 and bit 24 under highly correlated inputs. Simulation results show the error rate of CAP is much smaller than GeAr.

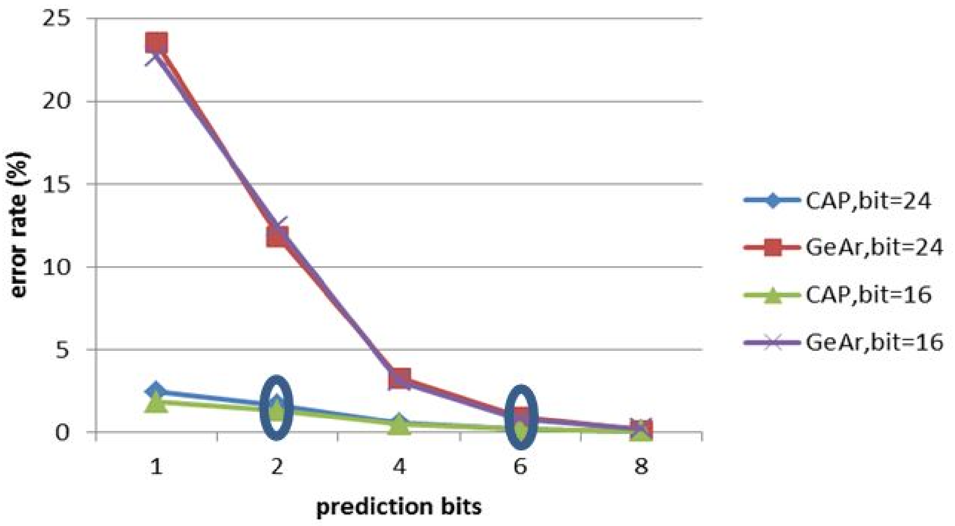


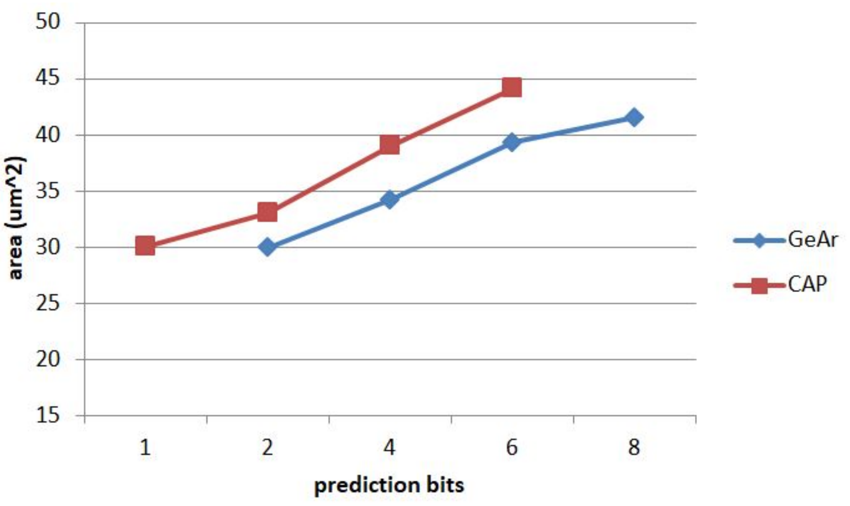
Figure 2.7: Error rate comparison under spatially correclated inputs

	GeAr(32,8,6)	CAP(32,6,2)	Reduction
Delay(ns)	0.30	0.23	23.33%
Area(um^2)	39.35	33.074	15.9%

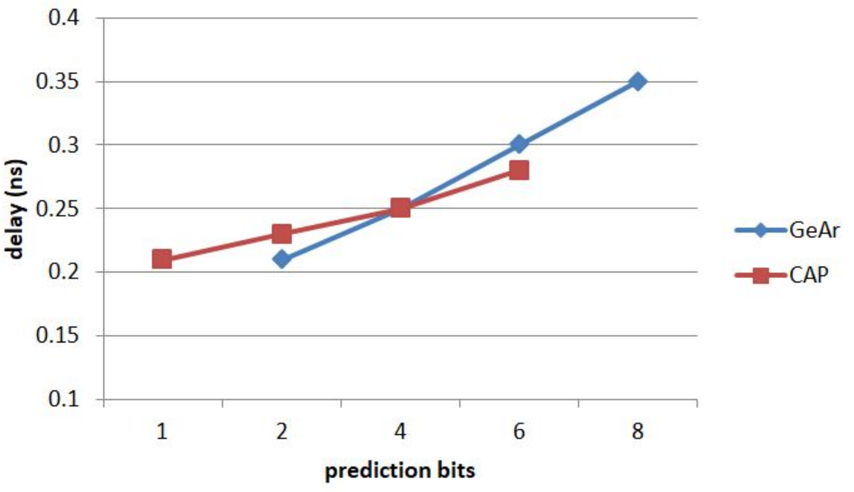
Table 2.3: Synthesis Results of GeAr(32,8,6) and CAP(32,8,2)

Signed subtraction under sine wave inputs

Signed subtraction is a common operation in digital signal processing. In this case, we analyze the performance of CAP for signed subtraction. As mentioned in the previous section, the signs of differences between consecutive inputs with strong temporal correlation seldom change. Intuitively, signs of differences between consecutive inputs reflect the tendency of signal changing, either increasing or decreasing. CAP can benefit a lot if signal frequency is low. For example, for a sine wave input with signal frequency $K1$, sampling frequency $K2$, signs of signal difference changes twice in $K2/K1$ operations so the error rate of CAP is $2 * K1/K2$.



(a) GeAr(32,8) and CAP(32,8) area compasion with differnt number of prediction bits



(b) GeAr(32,8) and CAP(32,8) delay comparison with differnt number of prediction bits

Figure 2.8: CAP(32,8) and GeAr(32,8) synthesis results comparison

2.3.3 Synthesis Result

We implemented two 32-bit approximate adders (GeAr-based and CAP-based) in Verilog and synthesized them using Synopsys Design Compiler with TSMC 16nm library. Both adders have four 8-bit sub-adders. Figure 2.8 shows the area and latency results with different predictive bits. With the same number of predictive bits, CAP consumes 12.5% more area and has slightly higher delay when $P > 4$. However, CAP can achieve the same error rate with much less prediction bits for highly correlated input streams. As shown in Figure 2.8, CAP with 2 predictive bits get the same error rate as GeAr with 6 predictive bits. Table 2.3 shows the corresponding latency, area and power results of GeAr(32,8,6) and CAP(32,8,2). Based on Table 2.3, CAP is able to reduce latency by 23.33% and save as much as 15.9% in silicon area.

2.4 Experimental Results

In this section, we experiment a 24-bit CAP based approximate adder and a GeAr adder for interchannel decorrelation used in audio encoding standards FLAC [22] to reduce redundancy and save bandwidth. Comparison results on delay, area and power are also reported.

Most of the existing approximate adders [11, 12, 13, 15] use fixed result bits R and predictive bits P for all sub-adders since for uniform distributed inputs, the error rate is only decided by the number of prediction bits. This may produce non-optimal design for non-uniformly distributed inputs. To explore design space more efficiently, we implemented a procedure which assigns just enough prediction bits for each sub-adder thus produce better design in terms of latency and area. The procedure produced CAP based adder which can be represented as

$$(N, P, R) = (24, [8, 6, 4, 6], [0, 2, 4, 1])$$

Note that previously we use (N, R, P) to describe adder configuration which implies all sub-adders have the same result bit R and predictive bits P. They become array with K elements, where K is the number of sub-adders. The adder employs a hybrid prediction scheme where the last predictor (where signal bits are highly correlated) is CAP based while the other predictors (where signal bits are less correlated) are GeAr based. GeAr based

	GeAr(24,8,5)	CAP(24,[8,6,4,6],[0,2,4,1])	Reduction
Delay(ns)	0.27	0.2	25.92%
Area(um^2)	26.5	23.01	13.17%
Error Rate	7.57%	15.9%	-110.03%
Effectiv Delay(ns)	0.2904	0.2318	20.18%

Table 2.4: Synthesis Results of GeAr(24,8,5) and CAP(24,[8,6,4,6],[0,2,4,1])

predictor suffers from high error rate when predicting for the last sub-adder. According to [23], 99% of predicted errors can be fixed in two cycles and therefore effectiveness delay can be estimated as (1+ER)*delay. To achieve the best effective delay, GeAr(24,8,5) is selected for comparison.

Table 2.4 is the comparison results between GeAr(24,8,5) and CAP(24,[8,6,4,6],[0,2,4,1]) on delay, area, error rate and effective delay. CAP based approximate adder consumes 13.17% less in silicon area and reduces latency by 25.92% compared to GeAr. It's interesting to note though CAP has higher error rate, its effective delay is still lower than GeAr due to latency reduction. By scaling supply voltage from 0.9V down to 0.71V so that CAP has the same effective delay as GeAr, our simulation results show that 35% power saving can be achieved.

2.5 Conclusion

In this section, we first studied the spatial/temporal correlations existing in real input streams as well as carry-in signals. For highly correlated input streams, we proposed correlation aware predictor which unitizes correlation information to predict carry-in value to reduce latency. Analysis and simulation results confirmed its effectiveness. Finally we implemented a 24-bit CAP based approximate adder and demonstrated its usefulness for real application.

Chapter 3

Approximate Compressor based Speculative Multi-Operand Adder

Multi-operand adder (MOA) is a basic block widely used in digital signal processing systems. Multiplier can be considered as a special case whose input bit-array is non-rectangular. Recently many approximate multipliers are proposed to reduce the critical path delay by replacing the compressors on the timing critical paths with the approximate ones. However, research works on approximate multi-operand adders with rectangular input bit-array are rarely found to our knowledge. In this paper, we proposed an approximate compressor based pipelined multi-operand adder (AC-MOA) macro-architecture with error recovery mechanism. Approximate compressors (AC) are utilized in bit-array reduction tree to reduce the overall delay and error detection and correction units are used to compensate the errors. A light error correction scheme is adopted to cover the cases when error distance is 1 without stalling the pipeline. Otherwise an extra cycle is required to correct the final results. To further improve the overall performance, the proposed AC-MOA can operate in approximate mode when error correction signals of the least significant bit-blocks are masked. We implemented an 8-bit 8-operand AC-MOA in Verilog and synthesized with TSMC 16nm library. Simulation results show that the proposed AC-MOA achieves 1.25X∼1.49X and 1.68X∼1.78X improvement on performance in accurate mode and approximate mode respectively, compared with the conventional design. Synthesis results show the proposed AC-MOA has the best area-efficiency and energy-efficiency.

3.1 Background

3.1.1 Multi-opereand Adder

A multi-operand adder performs additions for more than two operands. The input vectors of multi-operand form a bit-array and reduction operation is needed to produce an output bit-array with less number of bits. A carry propagation adder is used to calculate final results until only 2 bit-vectors are left. [p:2] column compressors (also known as [p:2] adder), play key roles on multi-operand adder design since all of them are on the critical timing paths. Multi-operand adders can be implemented with adder tree or compressors. Recenet research works on interpolation filter [24] and sum-of-product [25] show that compressor based multi-operand adder has advantage on power dissipation over conventional adders from a state-of-the-art synthesis tool. Therefore we will focus on compresssor based multi-operand adder.

Fig. 3.1 shows the input bit-array reduction process of an 8-bit 8-operate adder. In reduction stage 1, two 4:2 compressors are used which reduce the bit-array into 4 rows. One 4:2 compressor is needed in reduction stage 2 to further reduce it into two rows. A final carry propagate adder is used to calculate final results. As highlighted in Fig. 3.1, the critical path of the multi-operand adder consists of 'bit-array reduction' and 'final addition'. For high performance applications, pipelines are used to split these two steps into two stages to reduce critical path delay. Techniques on reducing the delay of carry-propagate adders (CPA) have been well developed [26]. In this paper we assume final addition is not the bottle neck and

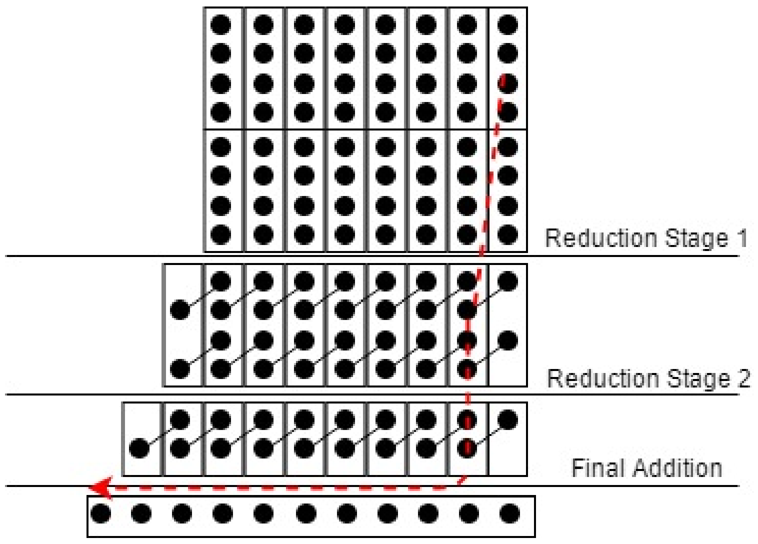


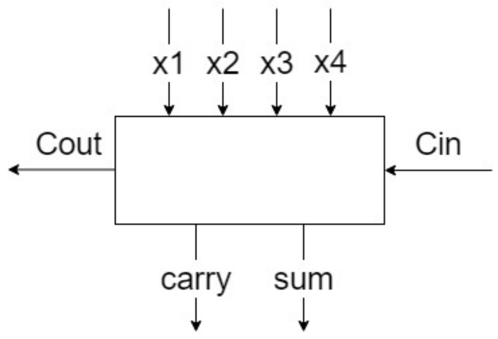
Figure 3.1: Bit-array reduction process of a multi-operands adder with 8 operaends

focus on bit-array reduction stage only. For a pipelined 8-operand adder, the critical paths of the bit-array reduction tree are the cascaded 4:2 compressors.

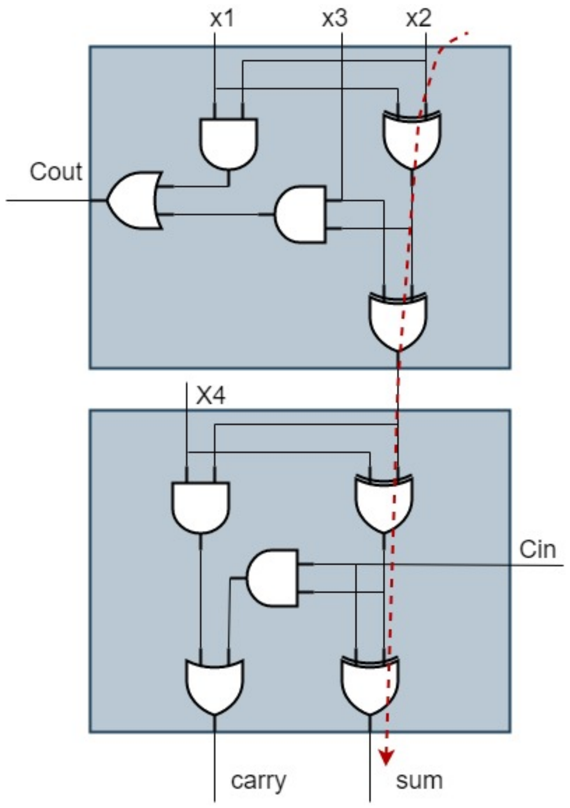
3.1.2 4:2 Compressor

As shown in Fig. 3.2a, a 4:2 compressor has five inputs and three outputs. Inputs $x1, x2, x3, x4$ and output sum are of the same weight, while output $carry$ and C_{out} are of heavier weight which would be passed to its upper neighboring bit. C_{in} is from the lower neighboring bit. A classic implementation of 4:2 compressor is composed of two full adders connected in serial, which can be described as follows:

$$\begin{aligned} C_{out} &= (x1 \oplus x2) \wedge x3 + (x1 \wedge x2) \\ sum &= x1 \oplus x2 \oplus x3 \oplus x4 \oplus C_{in} \\ carry &= (x1 \oplus x2 \oplus x3 \oplus x4) \wedge C_{in} + (x1 \oplus x2 \oplus x3) \wedge x4 \end{aligned} \tag{3.1}$$



(a) A 4:2 compressor



(b) Full-adder based 4:2 compressor

Figure 3.2: 4:2 compressor and the conventional implementation with full adders

Fig. 3.2b is the gate level schematic. An XOR gate has longer delay than AND and OR gates with the same area [27]. It is clear that the critical path of 4:2 compressor consists of 4 XOR gates [28]. It is worth mentioning that though some enhanced 4:2 compressors with 3 XOR delay are proposed for ultra-low-voltage design [28], their actually delays under normal voltage are close to the conventional design.

3.1.3 Approximate 4:2 Compressor

Approximate compressor (AC) uses simplified logic by modifying partial logic values in the truth table in order to reduce delay, area and power consumption. Inevitably it introduces errors in some cases. We present an implementation of 4:2 AC without using C_{in} and C_{out} below as an example. With two outputs $carry, sum$, the proposed AC can only represents

x1	x2	x3	x4	carry	sum	Err	$P(Err)$
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	0
0	0	1	1	1	0	0	0
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	0
0	1	1	1	1	1	0	0
1	0	0	0	0	1	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	1	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	1	0	0
1	1	1	0	1	1	0	0
1	1	1	1	1	1	1	1/16

Table 3.1: Truth Table of The Proposed Approximate 4:2 Compressor

$\{0, 1, 2, 3\}$. An error is introduced when all four inputs are 1s. The value of *carry* and *sum* are modified from '0' to '1' when all one inputs are 1. The error distance (ED) is reduced to 1. The truth table of the proposed 4:2 AC is Table 3.1.

The logic expression of the proposed approximate 4:2 compressor can be described as

$$\begin{aligned} err &= x1 \wedge x2 \wedge x3 \wedge x4 \\ sum &= (x1 \oplus x2 \oplus x3 \oplus x4) + err \\ carry &= (x1 \oplus x2) \wedge (x3 \oplus x4) + (x1 \wedge x2) \oplus (x3 \wedge x4) + err \end{aligned} \tag{3.2}$$

The gate level schematic are shown in Figure 3.3. The critical path of the proposed AC consists of 2 XOR gates and 1 OR gates, which is ∼35% shorter than conventional design according to [27]. Assuming all inputs obtain uniform distribution, it produces errors in 1 out of 16 cases with error distance $ED = 1$.

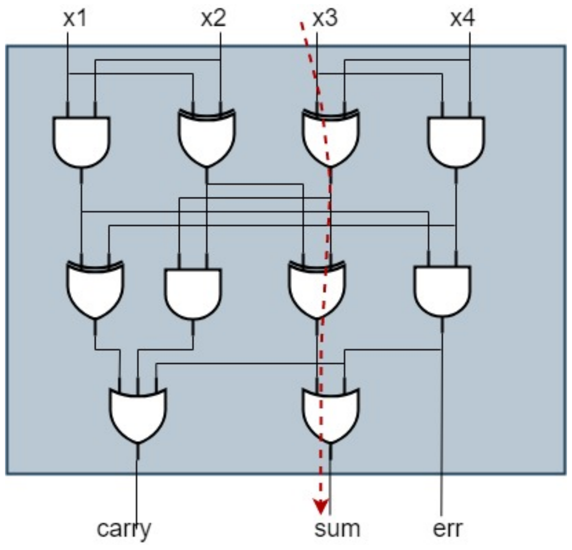


Figure 3.3: Gate level implementation of the proposed approximate 4:2 compressor

3.1.4 Related Work

Recently many approximate compressors were proposed [29, 30, 31, 32, 33] to build approximate multipliers. As explained in Section I, in almost all of the proposed designs, AC introduced errors are ignored or only partially fixed and exact results are not guaranteed. However, the runtime accuracy reconfigurability is a useful feature providing the flexibility to tradeoff output quality with power and performance at run time. Besides, for general digital signal process systems, both of approximate and exact computation modes are needed for applications with different accuracy requirement. A common approach to achieve this feature is adding error detection and correction (EDC) units, which may consume extra area, power or cycles when exact results are required [34]. [33] proposed dual-quality compressors which can be configured at run time to switch between exact and approximate operating modes. However, the error rate and area overhead are fairly large, preventing it from being massively used in a system. [35] proposed a error detection and correction mechanism to fix in-accurate compressor results which fits for the case when very limited ACs are used.

3.2 Approximate Compressor based Speculative Multi-operand Adder

The success of approximate compressor based multipliers motivated us to build multi-operand adders with approximate compressors to reduce the delay of rectangular input bit-array reduction tree. As shown in Fig. 3.1, a bit-array reduction tree has multiple critical paths with the equal delays. The challenge is, replacing only part of the compressors with approximate compressors won't improve the overall performance significantly, while massive replacements would introduce too many errors and produce unacceptable results. To resolve this issue, we deployed error detection and correction (EDC) block to compensate the errors. EDC block must be carefully designed to catch all potential errors without introducing significant overhead.

To make EDC more efficient, we first consolidate all errors of the same bit-block and calculate correction value for compensaton. At the same time, a "light error correction" (LEC) value with simplified logic is also generated to cover the most frequenct errors. When the LEC value is the same as accurate one, it can be used to calculate exact results without losing performance gains of approximations. The overall system performance can be significantly improved when the miss rate of LEC is low.

Fig. 3.4 is the diagram of the proposed approximate compressor based pipelined multi-operand adder (AC-MOA). It has multiple bit-blocks connected in serial to support input operands with multiple bits. Bit-array reduction and final addition are split into different stages. Optimized carry propagate adder (CPA) can be automated generated with commercial logic synthesis tool under given constraints so the final addition is not the bottle neck. In the proposed AC-MOA, approximate compressors are used in bit-array reduction tree to generate $Carry$ and Sun. EDC is used to generate error signal ERR and exact error compensation value EC_v to fix final results. The signal $STALL$ is enabled to stall the pipeline

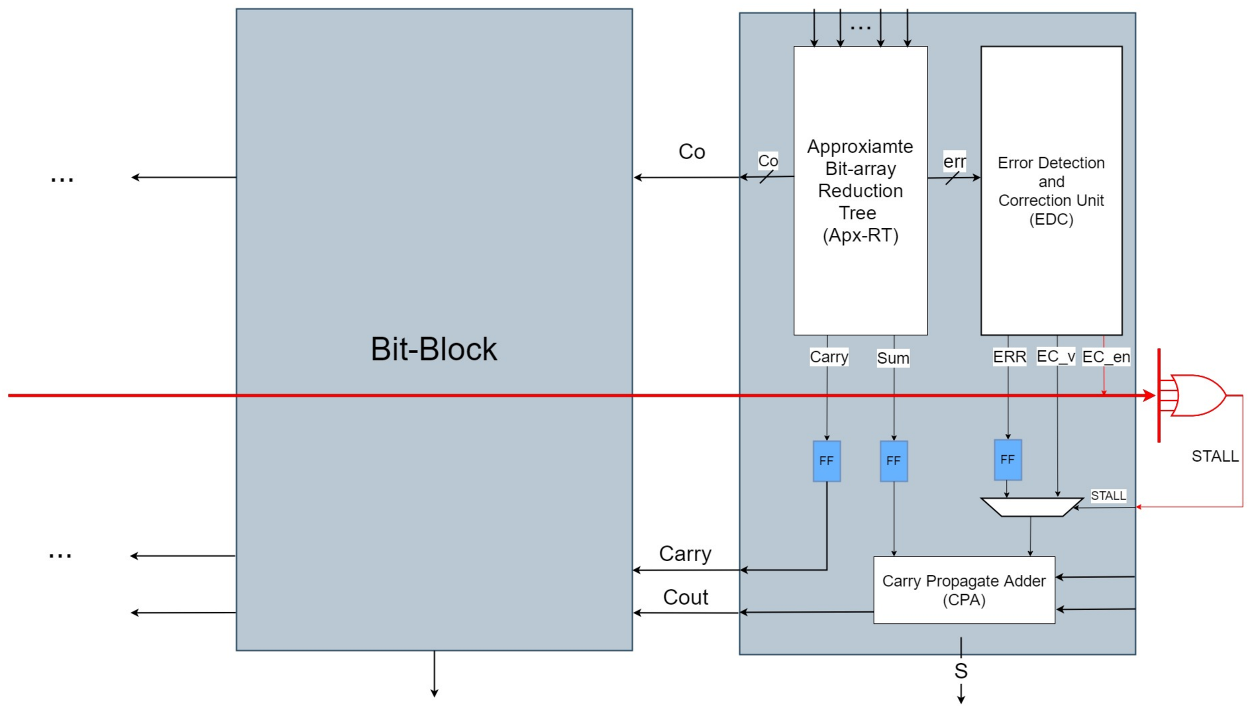

...
Bit-Block
Co
Co
Approxiamte Bit-array Reduction Tree (Apx-RT)
err
Error Detection and Correction Unit (EDC)
Carry
Sum
ERR
EC_v
EC_en
FF
FF
FF
STALL
STALL
...
Carry
Cout
Carry Propagate Adder (CPA)
S

Figure 3.4: Diagram of the proposed AC-MOA

when any of the bit-blocks require an extra cycle to correct errors with error distance greater than one.

Figure 3.5 shows the bit-block diagram of the proposed AC-MOA with EDC. Each bit-block consists of three major sub-blocks as described below.

3.2.1 Bit-array Reduction Tree with Approximate Compressors

Bit-array reduction tree consists lots of serially connected compressors. Multiple rows of input vectors are reduced into two, $Carry$ and Sum. The results are saved in flip-flops for final addition. In the proposed design, the compressors are replaced with AC to reduce the critical path delay. Error signals of ACs err_k are passed to EDC block for further processing.

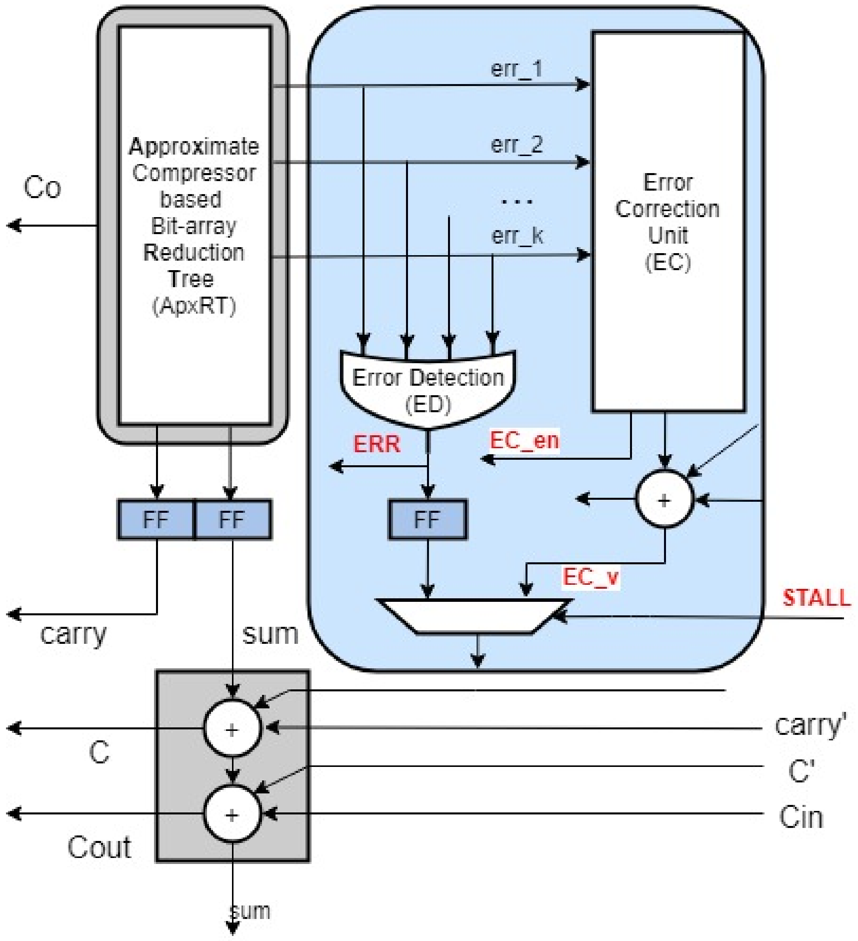


Figure 3.5: Bit-block diagram of the proposed AC-MOA

3.2.2 Error Detection and Correction Block

EDC takes error signals of ACs as input and generates the following outputs:

- *ERR*

  ERR is error detection result indicating if any of the ACs of the bit-block generates errors. It can be obtained by "OR" all *err* signals from ACs. When there is no more than 1 error in current bit-block, 'err' can be used as correction value to fix final results. This is an important feature to reduce penalty cycles since the probability that two or more errors occur in the same cycle and the same bit-block is low.

- *EC_v*

  EC_v is the error correction value for the current bit-block. It's implemented by adding all errors of the bit-block. It is worth mentioning that *EC_v* results don't need to be generated within one clock cycle. When *EC_v* is needed for correction, the pipeline would be stalled for 1 extra cycle. This makes designing complex error compensation possible since some *err* signals arrive late.

- *EC_en*

  EC_en is error detection result indicating *EC_v* should be used for compensation and one extra cycle is needed. When there are more than one error in the current cycle, *EC_en* is enabled to request extra cycles for error correction. *STALL* is a global signal and it's enabled when any of the bit-blocks *EC_en* is on. EDC should be designed to reduce the probability of $EC\_en == 1$ so *err* can be used for error correction which doesn't incur penalty cycle.

3.2.3 Carry Propagate Adder

Carry propagate adder calculates the summation of the reduced bit-vectors to get final results. In the proposed design, error compensation results need to be added in this stage. CPA results can be used as partial results by loading back as inputs of final addition to support more operands. Various types of carry propagation adders can be used and a detailed comparision is included in [25].

3.3 Implementation of an 8-Operands Pipelined AC-MOA

We implemented an 8-bit 8-operand AC-MOA. Figure 3.6 shows the schematic diagram of the proposed AC-MOA. The bit-blocks used at different bit positions are the same and only the bit-blocks for 3 LSBs are shown here due to space limitation.

In each bit-block, the approximate bit-array reduction tree has three 4:2 approximate compressors, which generates three error signals $e1, e2, e3$. A 3-input OR gate is used to generate error detection result err. A 3:2 compressor is used to generate EC_en indicating if there are more than 1 error in this bit-block, which enables global signal $STALL$ to stall the pipeline for one extra cycle to correct final results with EC_v. An adder is used to calculate accurate compensation results EC_v. Extra AND/OR gates are employed to reduce the error probability of $prob(e1 == 1)$ and $prob(e2 == 1)$ without changing the final results.

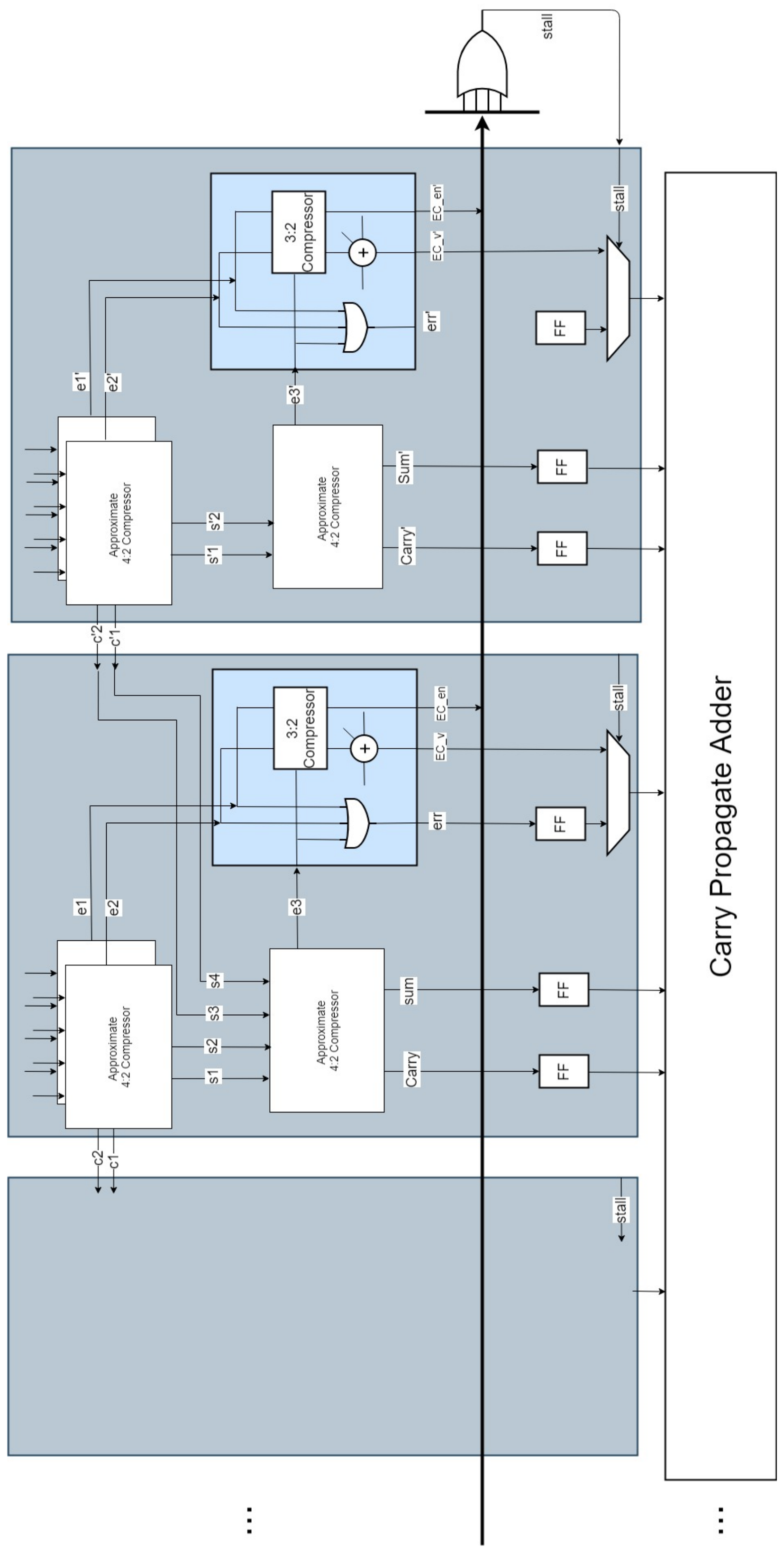


Figure 3.6: Schematic diagram of the proposed 8-operand AC-MOA

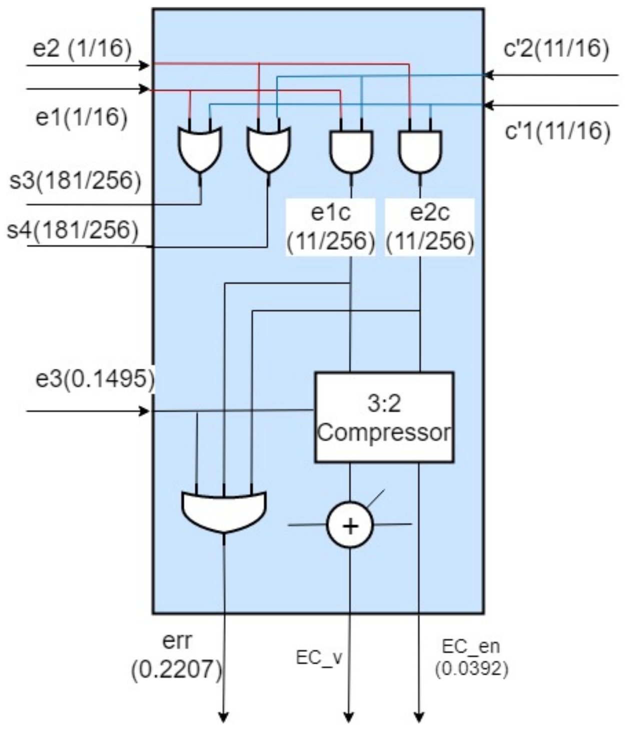


Figure 3.7: Error probability under uniformly distributed inputs

3.3.1 Error Probability Analysis

Error probability determines the system performance since error correction may require extra cycles to finish. The overhead is significant if the error probability is high. $prob(EC\_en == 1)$ is a key parameter for EDC design. Here is the analysis results on error probability of proposed bit-block.

Assuming the input bit-array value obtains uniform distribution, then
$P(EC\_en) = prob(EC\_en == 1)$ can be derived as following:

$$P(x1) = P(x2) = ... = P(x8) = 1/2$$

The corresponding error probabilities of $s1, s2, c1, c2, e1, e2$ are

$$P(s1) = P(s2) = 9/16$$
$$P(c1) = P(c2) = 11/16$$
$$P(e1) = P(e2) = 1/16$$

As shown in Fig. 3.7, extra AND/OR gates are used to exchange signals values between $e1, e2$ and $c1, c2$ to reduce potential "1"s to EDC. To simplify error probability calculation, we assume the internal signals are independent and the error probability can be calculated as

$$\begin{aligned}
P(s3) = P(s4) &= P(c'1) + P(e1) - P(c'1) * P(e1) \\
&= 181/256 \\
P(e1c) = P(e2c) &= P(c'1) * P(e1) = 11/256 \\
P(e3) &= P(s1) * P(s2) * P(s3) * P(s4) \approx 0.1582 \\
P(err) &= P(e1c) + P(e2c) + P(e3) \\
&- P(e1c) * P(e3) * 2 - P(e1c) * P(e2c) \\
&+ P(e1c) * P(e2c) * P(e3) \\
&\approx 0.2207
\end{aligned}$$

$P(EC\_en)$, the error probability that the error distance of the current bit-block is greater than 1, can be calculated as

$$\begin{aligned}
P(EC\_en) \approx\ & P(e1c) * P(e2c) \\
&+ P(e1c) * P(e3|e1c == 1, e2c == 0) \\
&+ P(e2c) * P(e3|e1c == 0, e2c == 1) \\
&\approx 0.0392
\end{aligned}$$

For an 8-bit 8-operand AC-MOA, the probability of pipeline stalling $P(STALL)$ is

$$\begin{aligned} P(STALL) \approx & \sum_{k=1}^{7} P_k(EC\_en) \\ & - \sum_{\substack{p\in 1\sim 7 \\ q\in 1\sim 7 \\ p\neq q}} P_p(EC\_en) * P_q(EC\_en) \\ & \approx 24.36\% \end{aligned}$$

The error probability analysis results on single bit-block shows that most errors introduced by ACs can be compensated since the occurence of multiple errors in a single cycle rarely happens. Only 3.92% of the cases require extra cycle. However, the probability of stalling pipelines increases fast when multiple bit-blocks are connected in serial. For 8-bit operands, the overall $P(STALL)$ is 24.36%, indicating 24.36% penalty cycles are used to correct the final results with uniform distributed inputs. The proposed AC-MOA can achieve better performance if the critical path can be reduced at least by 25%. One way to mitigate this effect is to correct errors with penalty cycles for MSBs only, which would introduce errors in LSBs.

3.3.2 Simulation Results

Previous error rate analysis is under the assumption that the input value obtains uniform distribution, which may not be the case in real applications. In order to validate the effectiveness of the proposed macro-architecture, we setup simulation environment in Matlab to get error rate information with real applications. We apply Gaussian filtering and mean filtering with 3x3 kernels on images for evaluation. We assume multiplications are implemented using precise components and the proposed AC-MOA is used for the additions only.

Table 3.2 shows the simulation results, including

Operations	$P(err)$	$P(EC\_en)$	$P(STALL)$
Gaussian filtering on image	15.54%	5.19%	30.28%
Mean filtering on image	30.89%	8.87%	43.61%
Mean filtering on random inputs	21.49%	3.33%	20.48%
Average	**22.63%**	**5.79%**	**31.45%**

Table 3.2: Simulated Error Rate Results in Real Applications

- $P(err)$: the probability that an error occurs at EDC level
- $P(EC\_en)$: the probability that extra cycle is needed at EDC level
- $P(STALL)$: the probability that AC-MOA stalls pipeline for error correction

From the simulation results, we find that distribution of input values has a significantly impact on the overall performance due to data correlations. The error probability of mean filtering on random inputs is close to our analysis result. However, mean filtering operation on images showed much higher error rate. The reason is that many of the input value of neighboring pixels are very close, causing high $P(err)$ on some bit-blocks since all of the input values are 1 at that bit position, which results in frequent pipeline stalling at top level.

3.3.3 Synthesis Results

We implemented the proposed 8-operands pipelined AC-MOA in Verilog and synthesized with TSMC 16nm library. Accurate 4:2 compressor based pipelined 8-bit 8-operands adders are also implemented for comparison. We generated low performance (LP), medium performance (MP) and high performance (HP) versions under different constraints with commercial synthesis tool. LP design consumes the least area and is considered as the baseline design for comparison.

Table 3.3 shows the synthesis results. Compared with the LP design, the high performance implementation (HP Design) consumes ~60% more area and 10X power to get 50% per-

Design	Delay(ns)	Area(um^2)	Power(mW)
LP Design	0.18	112.4952	0.1289
MP Design	0.14	141.1782	0.9017
HP Design	0.10	179.1852	1.4489
Proposed AC-MOA	0.10	136.9014	0.1355

Table 3.3: Synthesis results of 8-bit 8-operand multi-operand adders

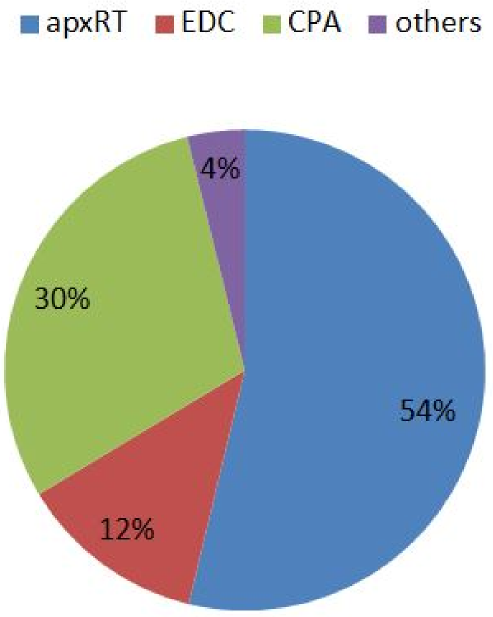


Figure 3.8: Area breakdown of the proposed 8-bit 8-operand AC-MOA

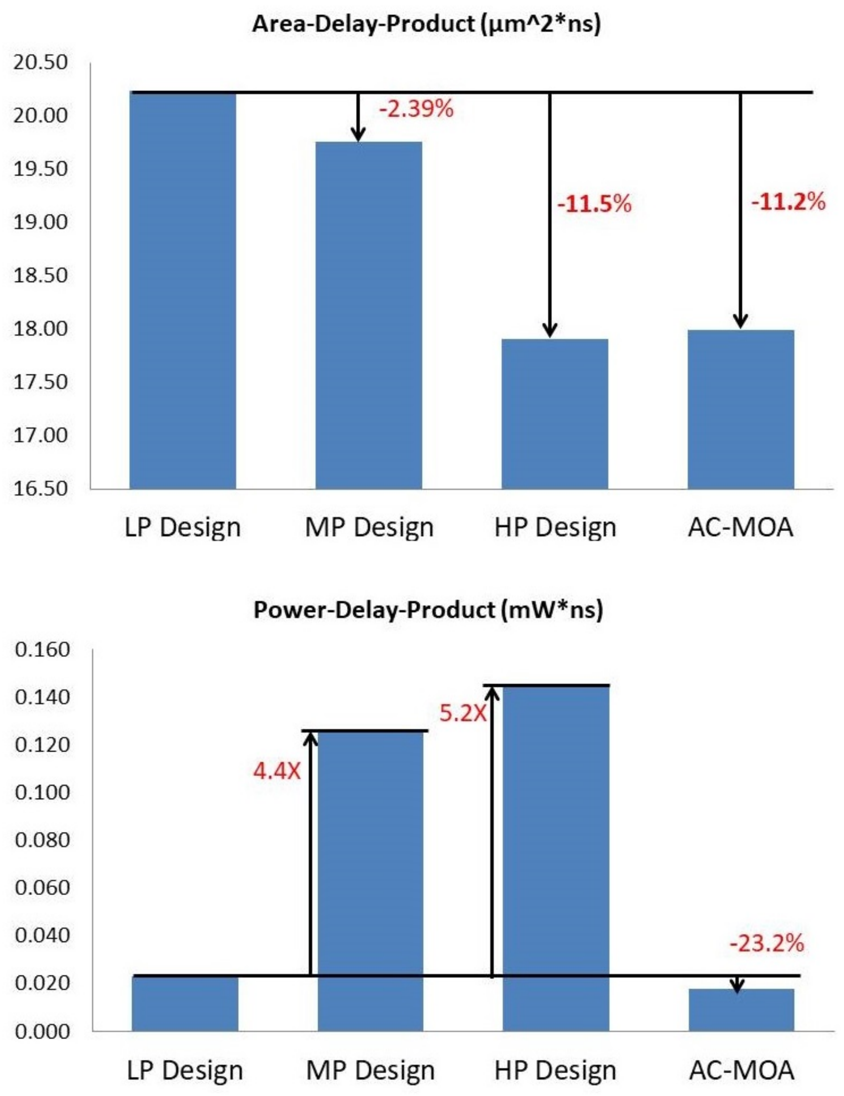


Figure 3.9: Area-delay-product and Power-delay-product comparison results

formance improvement. Our proposed AC-MOA achieves this target with 20% more area and 5% more power. Fig. 3.8 shows the area breakdown of the proposed 8-bit 8-operand AC-MOA. The area overhead of EDC is 12%.

We also calculate the Area-Delay-Product (ADP) and Power-Deay-Product (PDP) of different designs to compare their area and energy efficiency. To make fair comparisons, we assume the probability of stalling the pipeline is 31.45%, thus the effective delay of the proposed

AC-MOA is

$$T_{eff} = T_{clk} * (1 + prob(STALL)) = 0.13145$$

Fig. 3.9 shows the ADP and PDP comparison results. The ADP of all 4 designs are close indicating they have comparable area efficiency. HP design and the proposed AC-MOA get lower ADP due to the improvement on delay. In terms of PDP, we saw 8X differences across the designs. While the proposed AC-MOA and LP design maintain low PDP ranging from 0.018$\sim$0.023, the PDP of MP design and HP design are almost one order higher from 0.126$\sim$0.145. The results clearly show that the energy overhead of gate sizing is significant.

Approximate results are acceptable for error-tolerant applications. Without introducing significant errors in final results, the proposed AC-MOA can be switched to approximate mode by masking EC_en of LSB bit-blocks to reduce the probability of pipeline stalling thus improve performance. To explore the performance of the proposed AC-MOA in approximate mode, we conducted experiment to mask 0$\sim$5 EC_*en* signals. Mode 0 represents the exact mode and mode 5 has the most number of errors. Three operations are evaluated in different approximate modes. Mean error distance (MED) is used to reflect the error level. Table 3.4 shows the simulation results. When more EC_en signals are masked, mean error distance gets larger, the probability of pipeline stalling is reduced and the overall performance is improved. Simulation results showed the proposed AC-MOA can achieve 1.68X $\sim$ 1.78X speedup over the conventional design in approximate mode.

Based on the results, we can conclude that improving the performance of multi-operand adders with accurate logic representation is difficult since gate sizing seems to be the only way. Gating sizing does not only requires more area, the overhead on energy consumption is significant since extra effort is needed to drive large size transistors. Approximate computing is an effective way to improve overall performance. Low cost EDC can be used to effectively

Gaussian Filtering on Images						
Appx Mode	0	1	2	3	4	5
P(STALL)	0.303	0.278	0.209	0.160	0.049	0.001
MED	0	2.15	4.76	7.24	17.13	21.43
T_{eff}	0.130	0.127	0.121	0.116	0.105	0.101
Speedup	1.38	1.41	1.49	1.55	1.71	1.78
Mean Filtering on Images						
Appx Mode	0	1	2	3	4	5
P(STALL)	0.436	0.415	0.393	0.270	0.206	0.069
MED	0	2.161	3.367	11.303	16.446	34.775
T_{eff}	0.144	0.141	0.139	0.127	0.121	0.107
Speedup	1.25	1.27	1.29	1.41	1.49	1.68
Mean Filtering on Random Inputs						
Appx Mode	0	1	2	3	4	5
P(STALL)	0.205	0.178	0.151	0.122	0.094	0.065
MED	0	2.11	3.21	5.14	8.26	13.77
T_{eff}	0.120	0.118	0.115	0.112	0.109	0.106
Speedup	1.49	1.52	1.56	1.60	1.64	1.69

Table 3.4: Simulation results in approximate mode

reduce the penalty cycles. The proposed AC-MOA helps reduce the overall effective delay and the area and power overhead on error detection and correction units are marginal.

3.4 Conclusion

In this chapter, we reviewed the conventional design of multi-operand adder and 4:2 compressors. Motivated by the success of approximate compressor based multipliers, we proposed AC-MOA, an approximate compressor based hardware macro-architecture for multi-operand adder design. The critical paths of input bit-array reduction tree can be greatly reduced with approximate compressors and error detection and compensation blocks are used to correct errors introduced by AC. We implemented an 8-bit 8-operand AC-MOA and synthesis in TSMC 16nm. Simulation results showed it can achieve 1.25X$\sim$1.49X and 1.68X$\sim$1.78X performance improvement in accurate mode and approximate mode respectively compared with the conventional design. Synthesis results show the proposed AC-MOA has the best area efficiency and energy efficiency.

Chapter 4

Bit-Width Configurable MAC for Deep Learning Accelerator

Deep learning gained lots of attentions due to its unprecedented accuracy on many AI applications such as face recognition, image classification, object detection, speech recognition and translation etc [1, 2]. The demands of high computations, mostly multiply-accumulate (MAC) operations, and the need for high energy efficiency motivated designing dedicated hardware accelerators to accommodate DNN computations. Recent research work shows that the bit-width of MAC operations can be reduced without loss on accuracy [8]. While many research works exploit high degree of parallelism at word level with fixed-bit-width design [36, 37, 38, 39, 40, 41], recently hardware accelerators supporting variable bit-width [42, 43, 44, 45, 9, 46, 47] are introduced to benefit from the fact that bit-width requirements varies significantly across applications and layers of DNNs [9]. Bit-width scalable (also known as precision-scalable) hardware accelerators can allocate just enough processing resources or execution cycles to meet the precision requirements, therefore improve overall hardware efficiency.

In this chapter, we present BC-MAC, a bit-width configurable MAC array which can be reconfigured at run time to match the precision requirements. BC-MAC exploits bit-level parallelism [48] and narrow datapaths for accumulation to improve hardware efficiency. The basic processing element (PE) of BC-MAC is a pipelined multi-operand adder. Multiple PEs can be combined together to support input operands of any precisions. Bit-serial summation is used to accumulate partial addition results to perform MAC operations. In this way the accelerator can save either hardware resources or execution cycles when the required bit-width is reduced. Area overheads on reconfiguration logic are minimal compared with state-of-the-art designs [9, 46, 47].

The remainder of this chapter is organized as follows. We first give a brief introduction on deep learning networks and key operations. In section 4.1, several state-of-the-art bit-width scalable DNN accelerators are reviewed. In section 4.2, we introduce the proposed bit-width configurable MAC based deep learning accelerator. In section 4.3 simulation results and comparison results with other designs are presented. Section 4.4 is the conclusion.

4.1 Background

4.1.1 Deep Neural Networks

DNN algorithm uses multiple computation layers to extract intermediate features and get classified results [38]. Convolutional neural networks (CNN), as a common form of DNNs, have multiple computation layers, including convolution layer (CONV), activation layer (ACT), pooling layer (POOL), normalization layer (NORM) and fully connected layer (FC) etc. CONV layers are used to extract features from low-level to high-level, and a couple of FC layers to get classified results. Activation layers and normalization layers are applied in the middle of CNN layers to introduce nonlinearity. POOL layers are inserted after each

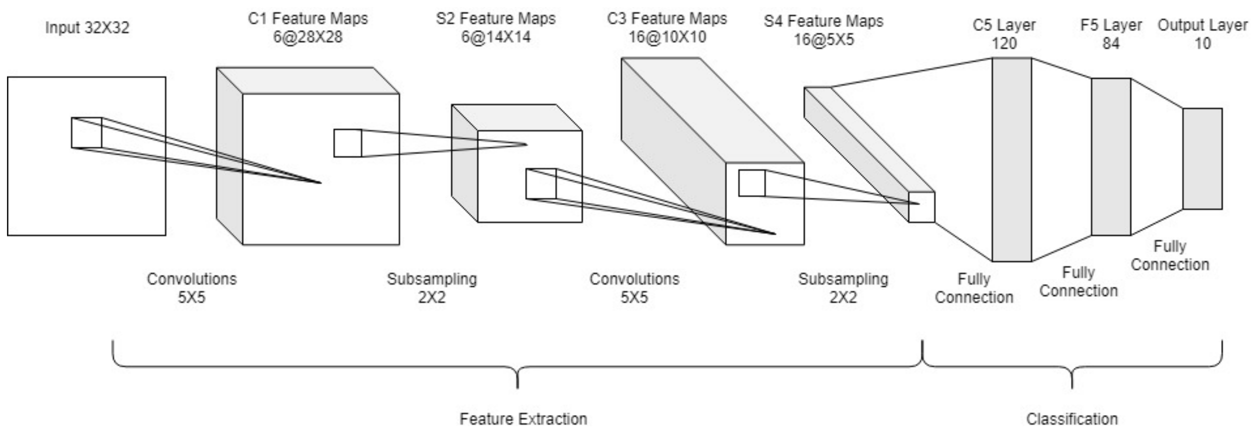


Figure 4.1: LeNet-5 Convolutional Neural Network Architecture

convolution layer to reduce the feature map size and help with overfitting problem.

LeNet-5 [49], which was one of the first CNN approaches proposed by Yann LeCun in 1989, is a representative of the convolution neural network (CNN) for recognizing handwritten digits and characters from pixel images with minimal preprocessing. As shown in Figure 4.1, LeNet-5 CNN architecture is made up of 7 layers, including 2 CONV layers, 2 POOL layers and 3 FC layers. Layer C1 has 6 convolutional filters of 5x5 and layer C3 uses 16 convolutional filters of the same size. Layer S2 and S4 are pooling layers of 2x2, which are used after CONV layers to reduce the dimensionality of intermediate feature maps. Three FC layers are deployed to get classified results based on feature maps extracted from previous layers.

Modern DNN models use very deep hierarchy of layers to achieve high accuracy. AlexNet [50], the first CNN to win the ImageNet Challenge in 2012, has 5 CONV layers and 3 FC layers. For CONV layers, the filter size ranges from 3x3 to 11x11 and the filter numbers are from 96 to 384. VGG-16 [51], the runner up of the ILSVRC-2014, makes the improvement over AlexNet by utilizing multiple stacked 3x3 filters. VGG-16 has 16 layers, consisting of 13 CONV layers and 3 FC layers. GoogLeNet [52], the winner of the ILSVRC-2014, has 22 layers with inception module. Different sized filters, including 1x1, 3x3 and 5x5, are used to produce intermediate features at multiple scales to further improve accuracy. ResNet [53], the winner of the ILSVRC-2015, goes even deeper with 34 layers or more. The core idea of ResNet is the introduction of "identity shortcut connection" layer which skips one or more weight layers to resolve the vanishing gradient problem during training. ResNet also uses 1x1 filters to reduce the number of weight parameters. A summary of popular DNNs with detailed layer information can be found in [54]. The trends can be observed that the depth of DNN models get deeper to achieve high accuracy and a wider range of filter shapes are needed across CONV layers to keep network flexibility. Most computations of deep learning algorithms would be placed on CONV layers due to the increasing number of CONV layers and filters. Therefore efficient hardware implementations on CONV layers are increasingly important for deep learning accelerators.

Another major form of DNN networks are recurrent neural networks (RNNs) [54]. Unlike feedforward networks that all of the computations are performed on the outputs of a previous layer, RNNs have internal memory to store temporal information and allow long-term dependencies to affect the output. RNNs and its popular variant, Long-Short-Term-Memory (LSTM) networks, are widely used for applications such as natural language processing and translation, where keeping a track of input history is important. The details of these models won't be reviewed since this thesis focuses on convolutional layers and full-connected layers. However, the major computation in RNNs is still the weighted sum and the proposed MAC can also be used for RNN accelerators.

4.1.2 Convolutional Neural Network Layers

In CONV layers, the major computation is high-dimensional convolution. Figure 4.2 demonstrates the operation of a convolutional layer. Input feature maps ($ifmaps$) and output feature maps ($ofmaps$) are 3D arrays. Both of them have multiple channels, each of which is a 2D array of neurons. The dimensions of input feature maps is represented by (C,H,W),

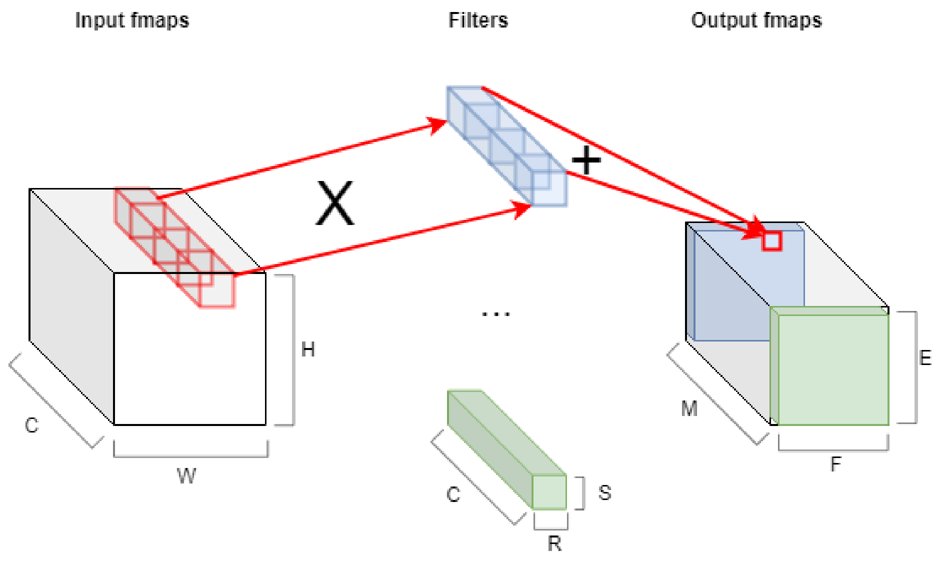


Figure 4.2: Convolutional (CONV) Layer

where C is the number of channels and (H,W) are the height and width of the 2D array respectively. M groups of 3D convolution filters, denoted as $\mathbf{W}[0]$,$\mathbf{W}[1]$,...,$\mathbf{W}[\text{M-1}]$, are applied on the $ifmaps$ to generate the $ofmaps$ by calculating the inner product of each of these M filters and a sub-array of 3D $ifmaps$ of the same size. The dimensions of filters are represented by (M,C,R,S), where M equals to the channel number of $ofmaps$, C is the channel number of $ifmaps$, and (R,S) defines the inner product sub-array sizes. The convolutional filters slide across the $ifmaps$ with a fixed stride U until the whole $ifmaps$ have been filtered. In addition, there is a 1-D bias that is added to the filtering results. The M 2D output filtered results forms $ofmaps$ which can be used as input for the next layer. The dimensions of $ofmaps$ are represented by (M,E,F), where M is the number of $ofmaps$ channels and (E,F) are the height and width of the 2D array, which can be calculated as

$$\begin{aligned} E &= \frac{H-R}{U}+1 \\ F &= \frac{W-S}{U}+1 \end{aligned} \tag{4.1}$$

With the definition in [38] where O, I, W, and B are the matrices of the $ofmaps$, $ifmaps$, $filters$, and $biases$ respectively. U is a given stride size. The computation of a CONV layer

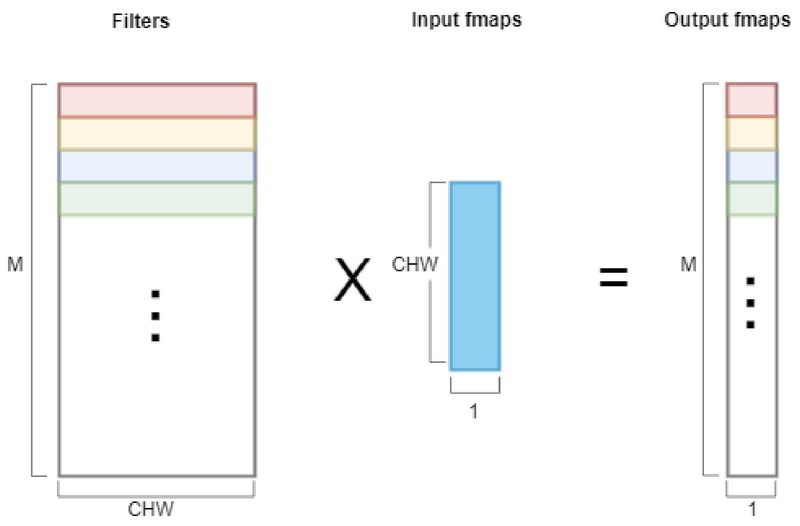


Figure 4.3: Fully-Connected (FC) Layer

can be defined as following:

$$\mathbf{O}[u][x][y] = \mathbf{B}[u] + \sum_{k=0}^{C-1}\sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\mathbf{I}[k][Ux+i][Uy+j] * \mathbf{W}[u][k][i][j] \tag{4.2}$$

Fully-connected layers are normally applied after the CONV layers to generate classified results. Figure 4.3 shows the operation of FC layers. Similar to the CONV layers, FC layer also applies filters on the $ifmaps$ but the filters are of the same size as the $ifmaps$. Multiple filters are applied to $ifmaps$ to generate classified results. Therefore the FC layer can be considered as a special case without weight sharing.

In addition to CONV and FC layers, CNNs also consist other layers such as the nonlinear, pooling and normalization. Nonlinear functions, such as ReLU, Sigmoid etc, are typically applied after CONV or FC layer to introduce nonlinearity. Pooling layer is used to reduce dimensionality of feature maps and increase the robustness of neural network. Normalization layers help improve accuracy by controlling the input distribution across layers. More information on these layers can be found from [54]. These layers are out of the scope of the thesis since they are not computation intensive and cannot be handled with MAC array.

4.2 Overview of Prior Work

Many dedicated deep learning accelerators are proposed to run deep learning applications in a more energy efficient way. This section provided an overview of prior work on deep learning accelerators. DNNs require huge computations and the majority are multiply-accumulate (MAC) operations. Deep learning accelerator (DLA) is essentially massive parallel multiply-add accelerator. We first reviewed data-parallel designs which can be considered as baseline designs for comparison. Then we overviewed some state-of-the-art bit-width precision scalable designs. Discussions on precision scalable design are included in the end of the section.

4.2.1 Data-Parallel Design

Diannao [39], is one of the earliest published deep learning accelerators with a data-parallel engine, which is the de facto standard used for comparison in most accelerator studies [42, 47]. We consider a data-parallel engine similar to Diannao as baseline design. Figure 4.4 shows the architecture that uses N-bit fixed-point input features and filters. Depending on the network, the bit-width N can be 8 or 16. As shown in 4.4, each processing element PE consists of multiple multipliers and a adder tree to perform 2-D convolution operations. A final shift-adder is used to accumulate partial results. Multiple PEs can be deployed to share activations. Every cycle each PE accepts multiple activations and the corresponding weights. The activations are broadcast to all PEs. Each PE unit multiplies the activations with its weights. An adder tree is used to calculate the summation of resulting products, and accumulates the result into an output register that stores the partial output. The adder tree helps reduce the cost of saving partial sum results compared with single MAC IPs, which need to load and save partial sum for all multiply results, making adder-tree based IPs more energy efficient.

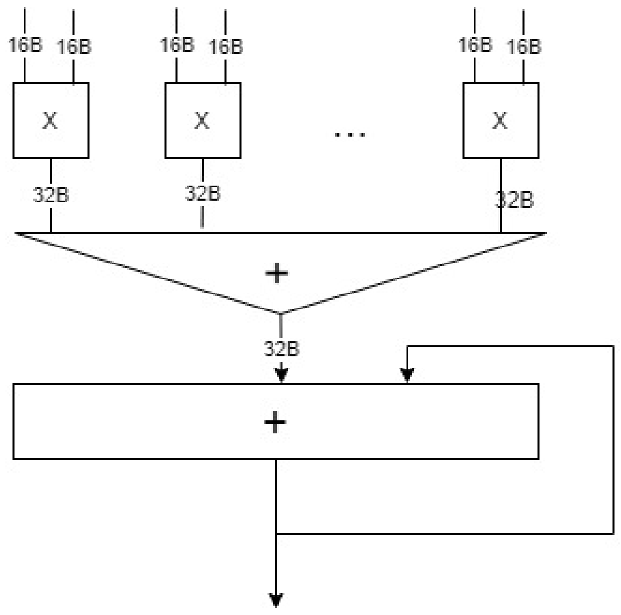


Figure 4.4: Data-parallel Baseline Architecture

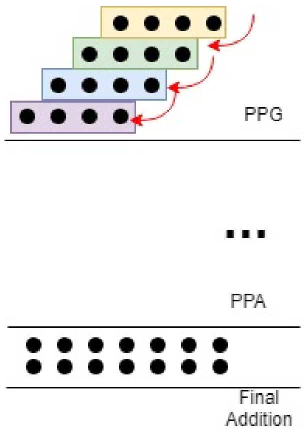
Systolic MAC array is another popular architecture to accelerate 2-D convolutions [38, 9]. A PE usually consists of a multiplier and a final adder. All PEs are connected to their neighbors from three directions: horizontal, vertical and diagonal. The whole MAC array accepts multiple activations and weights from the boundary. Activations, weights and partial sum results of each PE are passed to its neighbors from different three directions respectively to save the IO bandwidth. The multipliers and adders are of fixed bit-width. Architectures have an significant impact on memory access which is out of the scope of this thesis though.

4.2.2 Temporal Fusion and Spatial Fusion

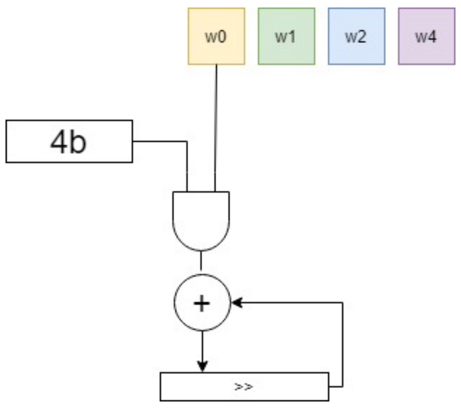
Recent research works on DNN algorithm show that bit-width requirements of activations and weights can be reduced with no loss of accuracy. The actual requirements varies significantly across DNNs for different individual layers [55, 56]. This provides an opportunity to exploit bit-width configurable accelerrator for higher hardware efficiency. Precision-scalable hardware accelerators allocate just enough processing resources or execution cycles to meet the bit-width requirement of individual layers, therefore improve hardware efficiency. What's more, fewer memory access is required if less bits are needed which further reduces power consumption.

Two common approaches to realize variable bit-width designs are spatial fusion and temporal fusion [9]. Prior works on precision scalable designs [9, 46**?** , 57, 47, 44, 45] use one or both of them at different granularities. In this section we first take multiplier as an example to briefly explain these two approaches. A review on state-of-the-art precision scalable deep learning accelerators is also included.

Temporal Fusion



(a) Temporal fusion diagram of 4x4 multiplication



(b) Temporal fusion schematic of 4x4 multiplication

Figure 4.5: Temporal Fusion diagram and schematic of 4x4 multiplication

As shown in Figure 4.5(a), a 4x4 multiplier has three major stages: partial product generation (PPG), partial product accumulation (PPA) and final addition (FA). Different fusion schemes group partial products at different granularity or process them in different cycles. We found fusion diagram is helpful to show the differences of various bit-width scalable designs. And mapping a given fusion diagram to hardware schematics is straightforward. A fusion diagram uses

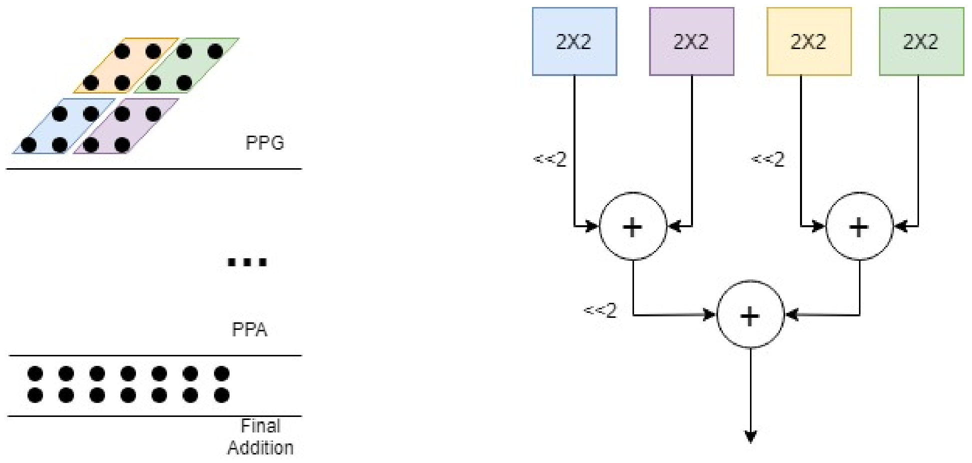
1. rectangles with different colors to show if partial products are processed on the same PE. Partial products in the same rectangle are assigned to the same PE and are processed in the same cycle. Partial products belonging to different rectangles are assigned to different PEs or are processed in different cycles. Shifters are needed to align partial products bit-positions to calculate the summation results.

2. curved arrow to show if partial products are processed in the same cycle. A series of curved arrows means multiple 'shift-add' operations are needed to accumulate partial products with partial sum results. Figure 4.5 shows a temporal fusion diagram and schematic to calculate 4X4 multiplication results.

For the temporal design, the multiplier performs 'shift-add' operations in multiple cycles to get final results. For example, the temporal design requires 4 cycles to execute a 4-bit x 4-bit multiplication. Figure 4.5(b) is the hardware schematic. Compared to a fixed 4-bit multiplier, the temporal design uses much smaller multiply units for PPG and PPA. However, the hardware resources required for the FA (the shifter and the accumulator) depend on the highest supported bit-width which is not scalable when less bit-width is needed. According to [9], to support up to 16-bits using a temporal design, the shifter and the accumulator use up around 90% of the area, which limits the benefits provided by this approach.

Spatial Fusion

Spatial fusion multiplier splits the 2D partial products array to multiple smaller bit-blocks (of size 2x2 in this case) and combines them over a single cycle to execute either one 4-bit x 4-bit multiplication, two 4-bit x 2- bit multiplications, or four 2-bit x 2-bit multiplications. Figure 4.6 shows the fusion diagram and schematic of a spatial multiplier that supports up to 4 bits for either of the two operands. Different from temporal design, the spatial multiplier adds explicit shifters for each small bit-blocks. An adder tree is used to perform PPA. Each



(a) Spatial fusion diagram of 4x4 multiplication (b) Spatial fusion schematic of 4x4 multiplication

Figure 4.6: Spatial Fusion diagram and schematic of 4x4 multiplication

level of the shift-add tree consists of three shift-units and a four-input adder. Compared to a 4-bit fixed bit-width multiplier the spatial multiplier requires more area but delivers higher performance when bit-width requirement is low.

4.2.3 Bit-width Scalable Design

Recently, many bit-width scalable hardware accelerators have been proposed for a variety of neural networks enabling the optimization of power consumption and accuracy [7, 8, 10, 13, 14]. In this section, we reviewed some state-of-the-art designs with fusion diagrams.

Stripes [42] is a 1D bit-width scalable design supporting variable bit-width for activations only. As shown in Figure 4.7, temporal fusion approach is used. An enhanced version, called dstripes [57], can reduce the execution cycles on a finer granularity then a layer. In UNPU [45], variable precision configurations are supported for weights only. As shown in Figure 4.8, temporal fusion is adopted and a lookup-table is used to pre-calculate partial sum results for reuse. As discussed before, 1D temporal design having bit-width flexibility only for activations or weights, lead to limitation to choose optimal bit widths. And full

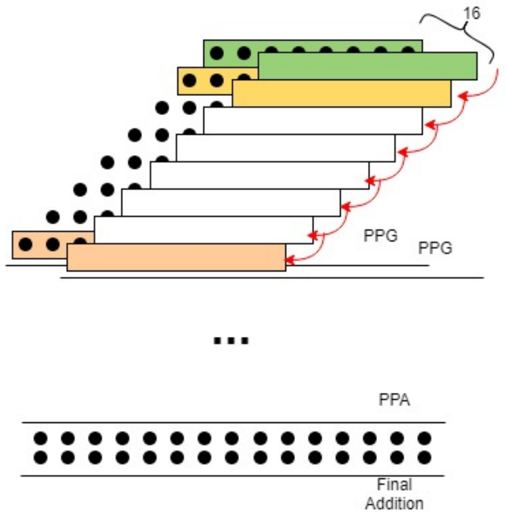


Figure 4.7: Stripes fusion diagram (1D Temporal Fusion)

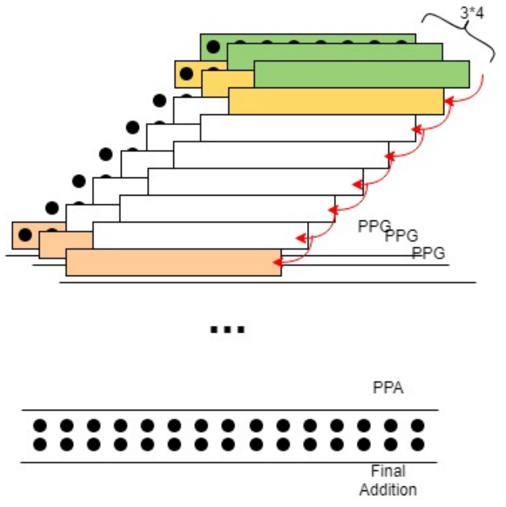


Figure 4.8: UNPU fusion diagram (1D* Temporal Fusion)

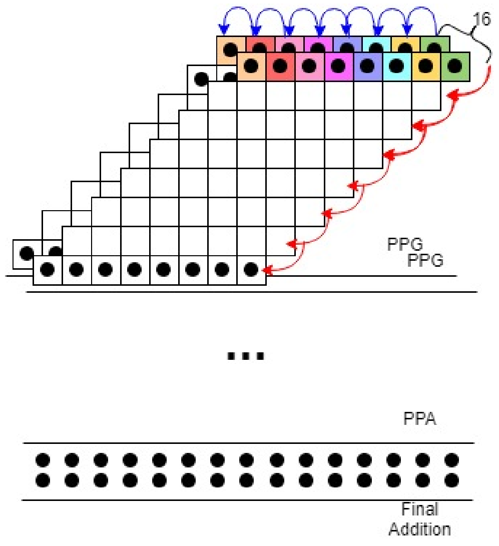


Figure 4.9: Loom fusion diagram (2D Temporal Fusion)

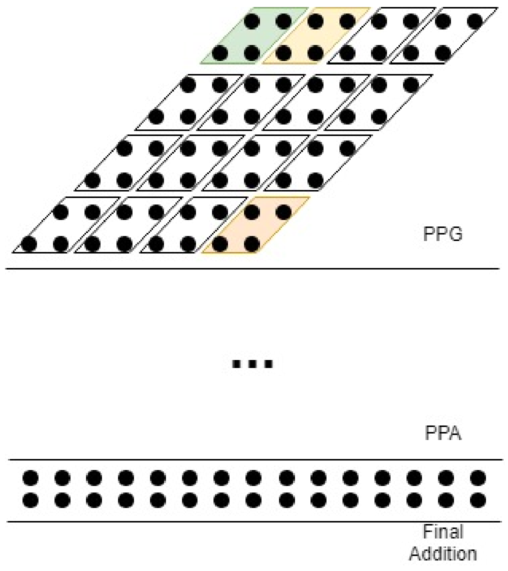


Figure 4.10: Bit-fusion fusion diagram (2D Spatial Fusion)

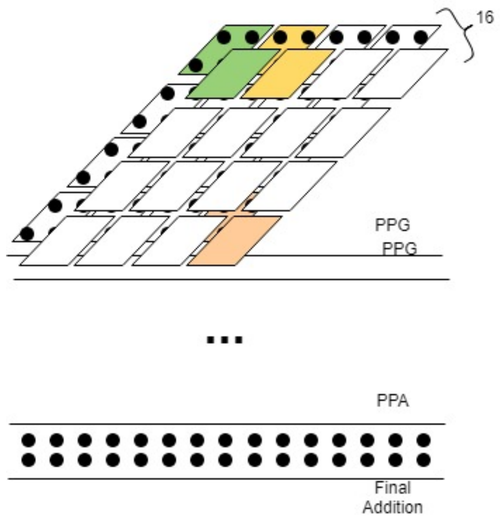


Figure 4.11: Bit-blade fusion diagram (2D* Spatial Fusion)

bit-width is needed for FA which is a waste when low precision is needed. Loom [47] adopts bit-serial multiplication to achieve the flexibility of bit widths. As shown in Figure 4.9, 2D temporal fusion is used to support variable bit-width for both of weight and activation. However, it requires two level FA which requires much wider bit-width to store parital sum results, which significantly decreases hardware efficiency. Bit Fusion [9] is a 2D spatial fusion based bit-width scalable design. As shown in Figure 4.10, it performs multiplication with variable bit-width by partitioning it into several 2-bit x 2-bit multiplications and then by performing shift-addition operations. By doing so, it can always fully utilize all the sub-multipliers. However, the shift-addition logic for the reconfigurable bit-width occupies large area (67%) and consumes a lot of power (79%) [9]. Bit-Blade [46] reduces the overhead for variable shift-addition using bitwise summation method, which has much smaller overhead for shift-add logic compared to Bit Fusion. As shown in Figure 4.11, aligned 2-bit x 2-bit bit-blocks from multiple multiplications are mapped to the same PE for processing, which eliminates all of the shifters at 2x2 bit-block level. However, the overhead of supporting high bit-width, 16-bit for example, is still significant.

Prior works on bit-width scalable designs showed that both of temporal fusion and spatial fusion are effective ways to support various precision requirements. When the bit-width requirement can be reduced, temporal fusion helps save execution cycles while spatial fusion saves hardware resources. However, the overhead of reconfiguration logic can offset the benefits.

4.3 BC-MAC: Bit-width Configurable MAC for Deep Learning Accelerator

In this section, we present Bit-width Configurable MAC which can be used to build deep learning accelerators, which exploits both of spatial and temporal fusion approaches to save hardware resources and execution cycles. We first explain the execution model with a simplified diagram. Then we present the design considerations and implementation with details from bit-block design to top level structure.

4.3.1 BC-MAC Execution Model

Figure 4.12 is the fusion diagram of the proposed BC-MAC, showing that the proposed BC-MAC is a 2D bit-width scalable design with a spatial-temporal fusion approach. Spatial fusion approach is used to support variable precisions of input feature maps. The input partial products are split into groups of 4-bit and multiple aligned 4-bit partial product blocks are allocated to a multi-operand adder to calculate partial sum results. 4-bit multi-operand adders can be combined together to support input feature maps of wider bit-width. Hardware resources can be saved when the input feature maps bit-width can be reduced. Similar to Stripes, temporal fusion approach is used to accumulate partial sum results. Depending on the bit-width of filters, the number shift-add operations can be saved when the precision requirement is reduced. It's worth mentioning that a "narrow datapath" is used at the final addition stage to accumulate partial sum results, only 3-bits more than input

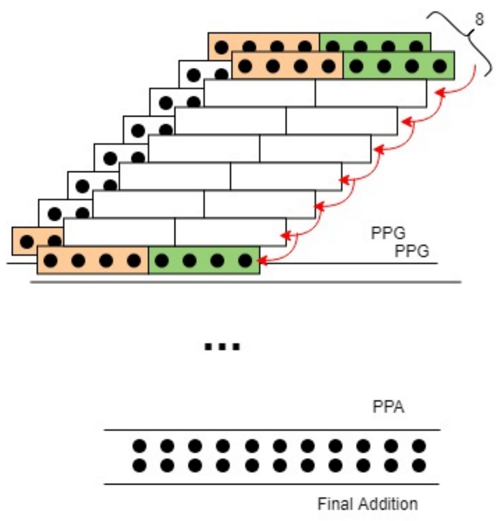


Figure 4.12: BC-MAC fusion diagram (Spatial-temporal fusion)

feature bit-width. This is for bit-width extension to accommodate addition of 8 operands. The LSB of previous partial sum is always shifted out which keeps the final summation bit-width unchanged.

Figure 4.13 is the diagram of a BC-MAC that consists of 4 MOAs. Each MOA has a config signal to control if the neighboring MOA is combined to support wide input feature maps. Here we explain how the proposed BC-MAC works.

1. Before processing, config bits are assigned to either '0' or '1' to match the precision requirement of the input feature maps. Figure 4.14 shows three different configurations to support input feature maps with different precisions, which are 4x4-bit, 2x8-bit and 1x16bit respectively. If input feature maps bit-width is 4, all config bits are set to '0' which means all signals from neighboring MOA are ignored. All MOAs work independently to accumulate partial sum results. If input feature maps bit-width is 16, all config bit are set to '1' which enables all signals
2. At the same time, input feature buffer should be ready to provide 8 input feature maps to MOAs. Sign extension is needed if the bit-width of input feature map is not multiple of 4. Filter buffer should be ready to provide filter value.

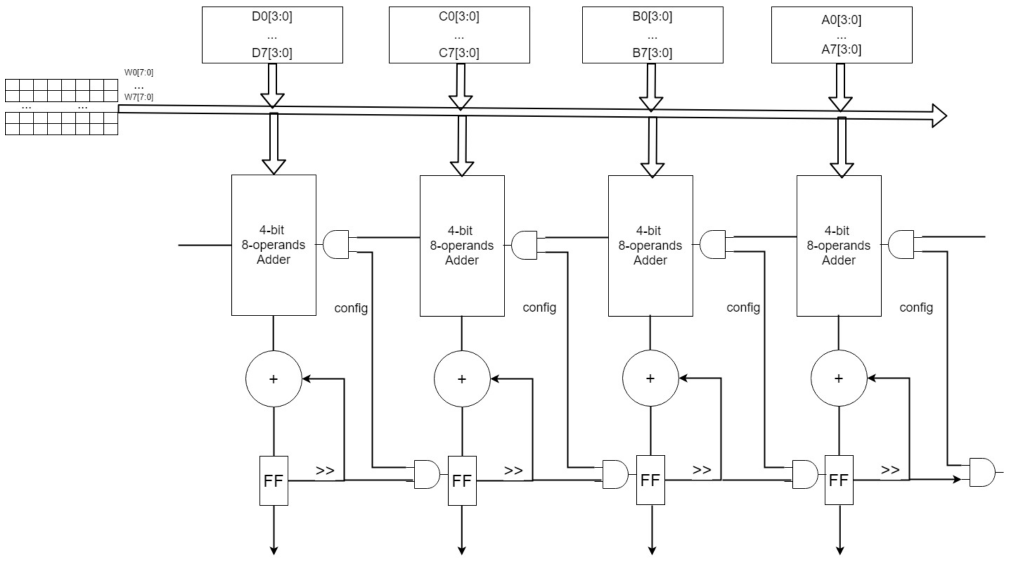


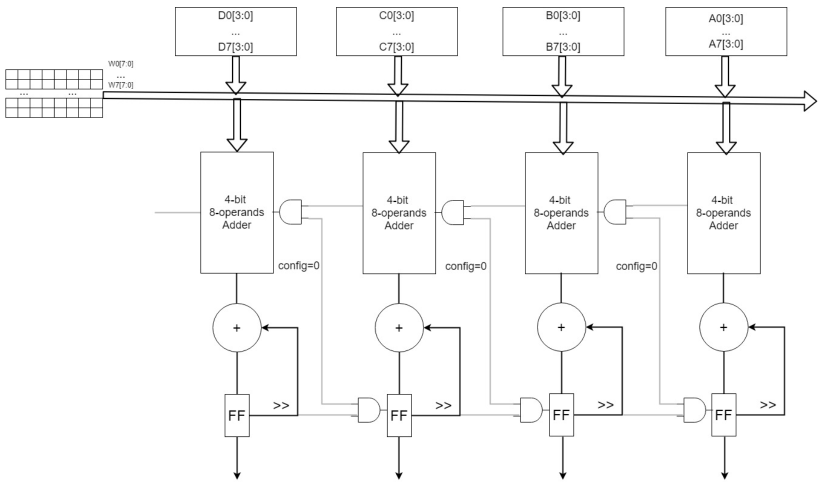
Figure 4.13: Diagram of the proposed BC-MAC

3. While keep input feature map unchanged, a fast clock is used to trigger filter buffer to release 8-bit number at each cycle for partial sum accumulation. If the neighboring config bit is set to 1, the 'cout' results would be sent to the MOA on the left side, and the final adder results will be shifted to the MOA on the right side for accumulation.

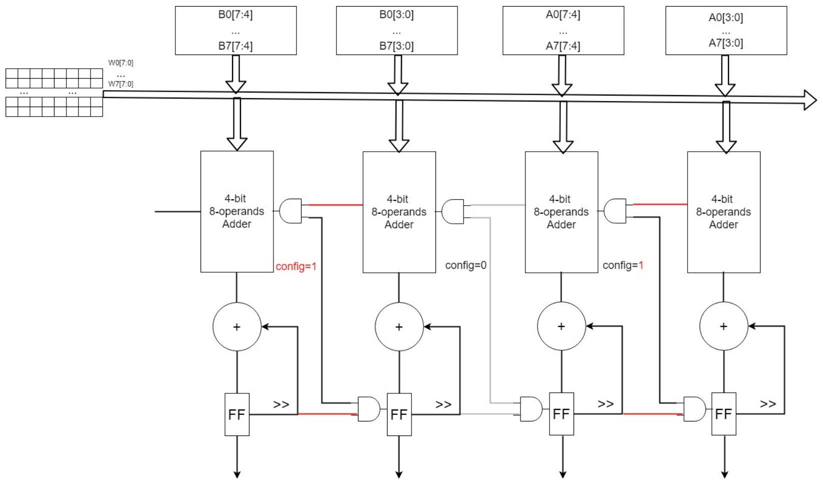
4.3.2 BC-MAC Implementation

As explained in the previous section, BC-MAC consists of multiple 4-bit 8-operands MOAs which can be combined or split per precision requirement. In this section, we discuss the detailed implementation from 1-bit 8:2 compressor tree, 4-bit bit-block, bit-plane and top MAC architecture.

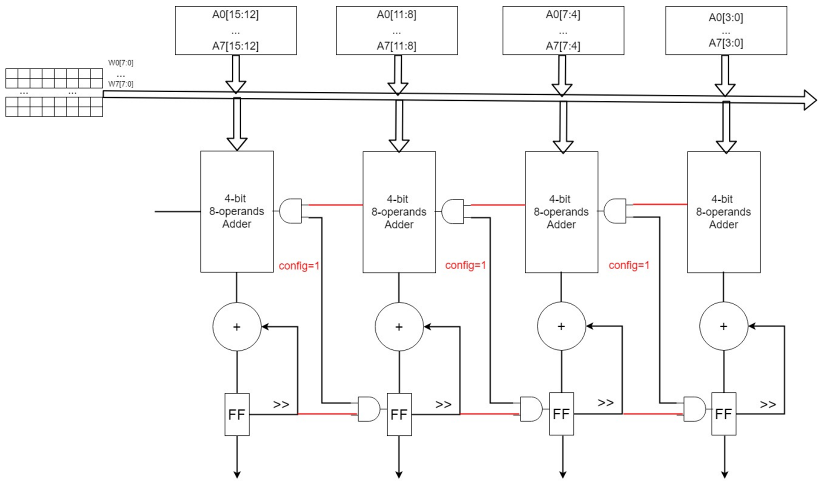
Figure 4.15 shows the schematic of 1-bit bit-block. It's essentially a pipelined multi-operand adder. The partial product reduction tree and the final summation are split into 2 stages. An 8:2 compressor is used to reduce the partial products until only 2-bit results *carry* and *sum* are obtained. As discussed in chapter 3, the critical path of compressor based reduction tree is fixed. When multiple 1-bit bit-block are connected together to form wider MOA,



(a) Configuration to support 4 x 4-bit MACs



(b) Configuration to support 2 x 8-bit MACs



(c) Configuration to support 1 x 16bit MAC

Figure 4.14: Configurations to support input features with different bit-widths

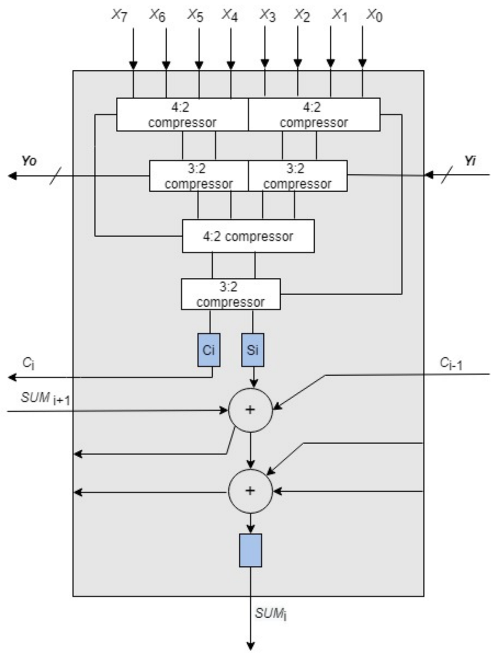


Figure 4.15: bit-block schematic

there are no new timing critical paths coming out which make the structure very scalable to support wide inputs. However, the timing critical path of final summation in pipeline stage 2, from 'cin' to 'cout', would get longer and eventually raise timing violations if too many bit-blocks are connected together.

To resolve the critical timing path issue in FA, a 4-bit bit-block with carry lookahead logic is designed to reduce the critical path delay for wide inputs. As shown in Figure 4.16, carry lookahead block is added on the LSB side of 4-bit bit-block. There is also a config signal IS_LSB_BLOCK to control 'cin' selection depending on if it's combined with neighboring 4-bit-block. Figure 4.17 shows that there are two potential critical timing paths: one path from compressor tree which has a fixed delay, and another one from carry lookahead and final addition. With the carry lookahead logic, the longest timing path to support 16-bit input feature maps would be reduced from $16 * T_{FA}$ to $4 * T_{FA} + 3 * T_{cla}$, which is much shorter, making BC-MAC scalable with spatial fusion approach.

To simplify the configuration and reduce control logic overhead, we proposed bit-plane which

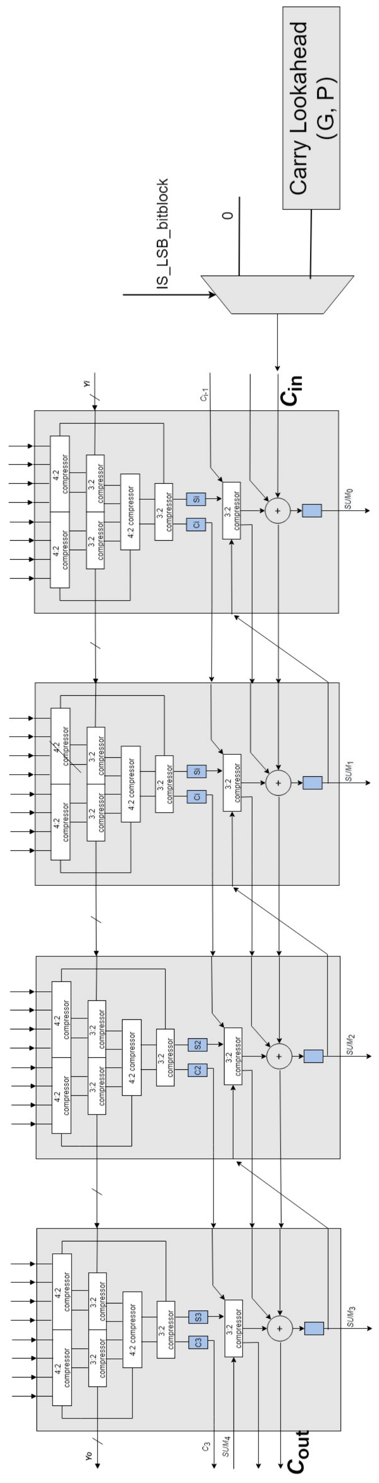


Figure 4.16: 4-bit-block schematic with carry lookahead logic

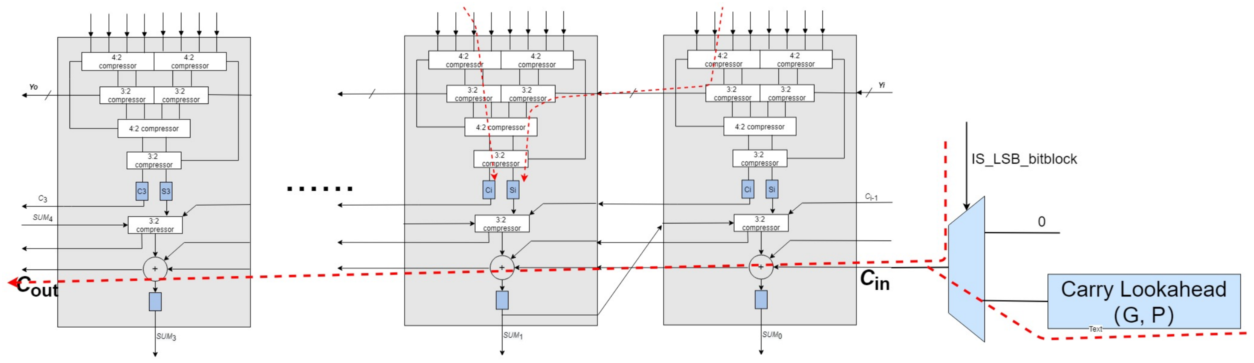


Figure 4.17: Critical timing paths of 4-bit-block

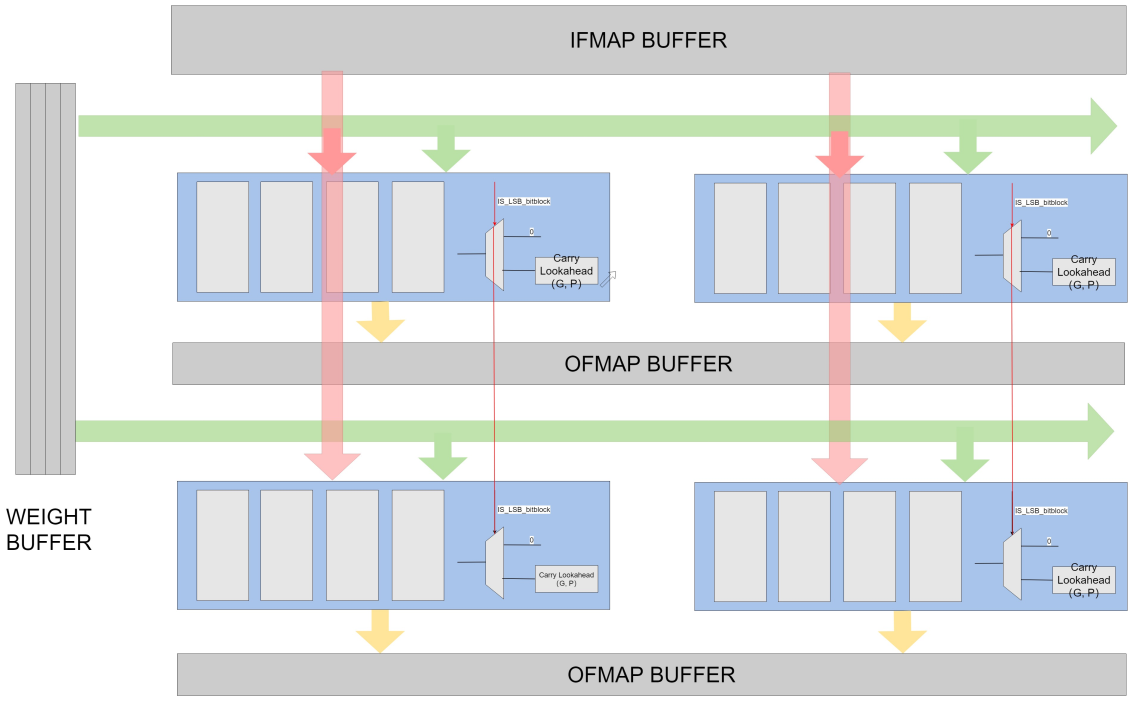


Figure 4.18: Bit-plane of 4-bit-blocks

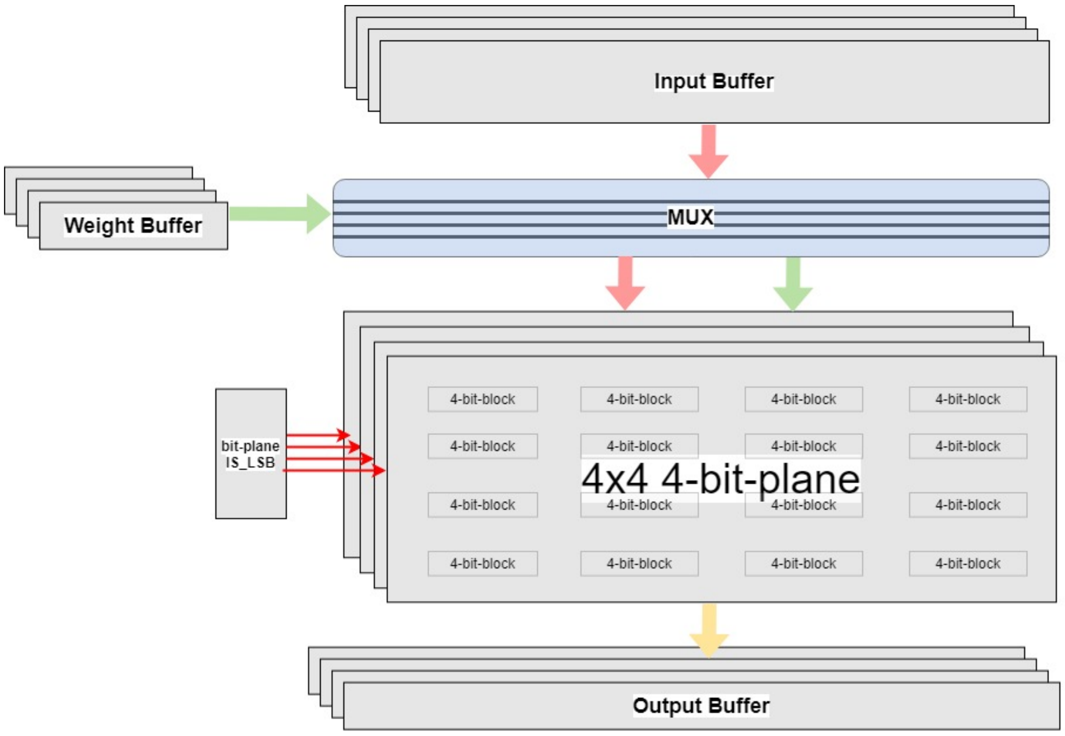


Figure 4.19: Use multiple bit-planes to support wide input feature maps

is a parallel 2D 4-bit-block array. Each bit-plane has a *ifmap* and a *weight*. Similar to Diannao tiles, both of the buffer data can be shared across 4-bit-blocks. As shown in Figure 4.18, the 4-bit-blocks within the same bit-plane share the same config value. No signal connections exist in the same bit-plane. Multiple bit-planes can be combined together to support wide input feature maps 4.19.

In summary, the proposed BC-MAC is based on 4-bit pipelined MOA. It adopts spatial-temporal fusion approach to support variable bit-width requirements. The number of 4-bit-blocks can be saved when input feature maps require less bits. Execution cycles can be reduced when the precision requirement of filters are low. Compared with temporal fusion designs which requires a final adder with full precision [42, 45], BC-MAC reduces the overhead by using narrow datapath. Unlike adder tree based structure in spatial fusion designs [9, 46], where sign extension logics are needed for the smallest 2x2 multipliers, BC-MAC requires only one 4-bit-block for both of bit-width/sign extension which helps save

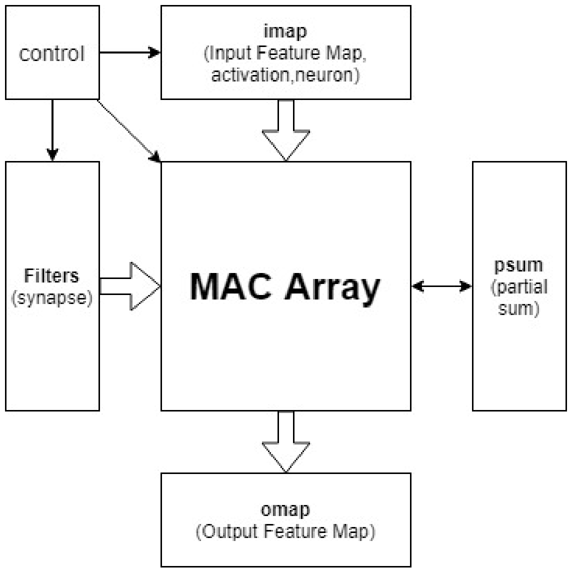


Figure 4.20: Top level diagram of the deep learning accelerator

area when input feature map bit-width is wide. Beside, Booth encoding can be used to boost MAC performance simply by replacing the input feature maps in activation buffer with Booth encoded data.

4.3.3 Deep Learning Accelerator Architecture

To evaluate the performance and compare it with state-of-the-art designs, we integrated the proposed BC-MAC into a deep learning accelerator. Figure 4.20 is the top level diagram of the deep learning accelerator, which consists of 1 control block, 4 buffers (ifmap,ofmap,filters and psum) and a 2D MAC array, which is similar to the design in [39, 47, 45].

4.4 Experimental Results

4.4.1 Simulation Environment Setup

The proposed BC-MAC array was implemented in Verilog and synthesized in TSMC 16nm technology. A version supporting Booth-encoding, BC-MAC-booth, was also implemented. For comparison, we also implemented the MAC arrays used in other state-of-the-other designs in Verilog and generated gate-level netlist. We developed a deep learning accelerator model in Matlab for results validation. Matlab simulation data were saved as stimulus to drive gate-level netlist simulation to get power information.

4.4.2 Comparison Methods and Results

To make fair comparison between designs with different fusion approaches, we use area-efficiency method and energy-efficiency method instead of comparing peak performance or energy consumption. Area efficiency indicates the performance under given silicon budget ($Mops/um^2$), while energy-efficiency evaluates the performance a MAC array can deliver under given power constraint ($Gops/mW$).

Figure 4.21 shows the area-efficiency comparason results under TSMC 16nm technologies. Performances under different bit-width are scaled based on synthesis results. The area-efficiencies of all bit-width scalable designs get higher except the baseline data-parallel design Diannao. Bit-width scalable designs show great advantage over the fixed bit-width design especially when the precision requirement is lower than 4. However, the advantages diminish quickly when precision requirement get higher due to the overhead of reconfigurable logic at low level. The proposed BC-MAC show advantages at the high precision requirement case (8-bitx8-bit), mainly due to that area efficiency of the compressor tree structure over adder

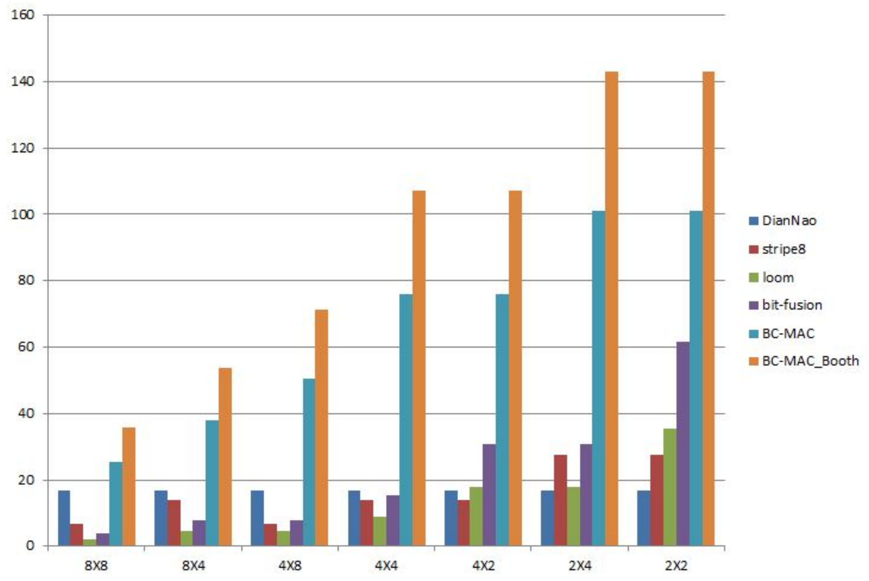


Figure 4.21: Area efficiency comparison results ($Mops/um^2$)

tree [25, 24].

Figure 4.22 shows the energy-efficiency comparason results under different bit-width requirements. Similar to area-efficiency comparison results, the energy-efficiency of all bit-width scalable designs get higher. However, the advantages over the fixed bit-width baseline design are not significant. When the precision requirement is high, it's not surprising to see the fixed bit-width design gets highest energy efficiency since it is designed to work at 8bitx8bit and the logic gates are fully in use. All bit-width scalable designs have lower energy efficiency due to the overhead introduced by the reconfiguration logic. It's worth mentioning that designs using temporal fusion approach are not as energy efficient as spatial fusion based designs, indicating the overhead introduced in final addition is significant. For the 2x2 case, Bit-Fusion shows highest energy efficiency since the smallest processing element is 2x2 multiplier. Bit-blade, the enhanced version 2D spatial fusion MAC, shows similar results as Bit-Fusion. However, this becomes overhead when precision requirement increases. The proposed BC-MAC achieves highest energy efficiency among all bit-width scalable designs except the 2-bitx2-bit case, which shows it's a promising ALU for deep learning accelerators.

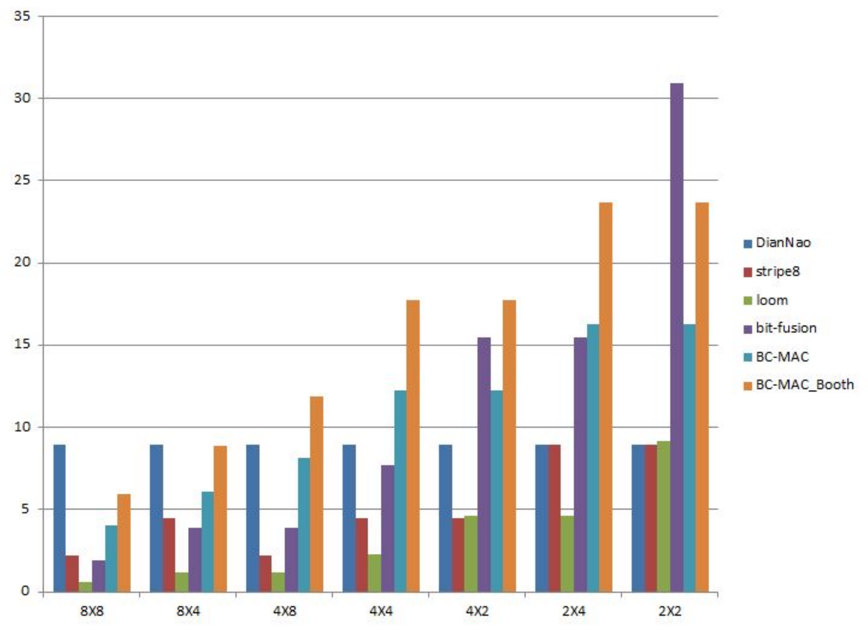


Figure 4.22: Energy efficiency comparison results ($Gops/mW$)

4.5 Conclusion

In this chapter, we presented a bit-width configurable MAC for deep learning accelerators. We first introduced precision-scalable design uArch with spatial fusion and temporal fusion approaches. Prior works are also reviewed. We proposed a multi-operand adder based MAC. BC-MAC uses spatial-temporal approach to support variable precision requirements for both of activations and weights. The bit-width can be reconfigureed at runtime to meet different precision requirements. Experimental results showed the proposed BC-MAC achieved higher area and energy efficiency than state-of-the-art designs.

Chapter 5

Conclusions and Future Work

In this thesis, we have discussed implementations of key arithmetic logic units using approximate computing techniques, including adders, multi-operand adders. We also explored bit-width scalable design techniques on multiply-accumulators for deep learning accelerators. However there are some challenges that need to be addressed.

In chapter 2, we implemented a new carry predictor for the carry signal value based on the correlation of input streams. Analysis and simulation results show that the latency of CAP based approximate adder can be reduced significantly for highly correlated input streams. We experimented with a CAP based approximate adder processing 24-bit audio streams. Simulation results show that the proposed approximate adder can significantly reduce latency and area for real applications. However, this structure is specific to an input stream with known correlations. In the future, we can examine other sources of correlations, for example structural correlations, and use the history based predictor to improve approximate adder performance.

In chapter 3, an approximate compressor based multi-operand adder was proposed. All compressors are replaced with approximate ones to reduce the overall latency. A light error detection and correction unit is used to compensate most of the errors without introducing extra cycles. An error probability analysis to calculate MOA error rate with the proposed approximate compressor was also presented. However, there is a lack of systematic and automatic method to explore the results of other approximate compressors, which prevents the exploration in large design space. In the future, a systematic program that can explore new approximate compressors and MOA performance can be developed.

In chapter 4, we present BC-MAC, which can be reconfigured at run time to match the precision requirements. BC-MAC uses spatial-temporal approach to support variable precision requirements for both of activations and weights. The basic processing element (PE) of BC-MAC is a multi-operand adder. Multiple PEs can be combined together to support input operands of any precisions. Bit-serial summation is used to accumulate partial addition results to perform MAC operations. In this way the accelerator can save either hardware resources or execution cycles when the required bit-width is reduced. In the future we can implement a BC-MAC based deep learning accelerator and evaluate the performance at system level.

Bibliography

[1] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. A Survey on Deep Learning: Algorithms, Techniques, and Applications, *ACM Computing Surveys (CSUR)* , 51(5):92:1–92:36, 2018.

[2] A. Shrestha and A. Mahmood. Review of Deep Learning Algorithms and Architectures, *IEEE Access* , vol. 7, pp. 53040-53065, 2019.

[3] J. Han and M. Orshansky. Approximate computing: An merging paradigm for energy-efficient design, *European Test Symposium (ETS)* , pp.1-6, 2013.

[4] V. Chippa, S. Chakradhar, K. Roy, and A. Raghunatha. Analysis and characterization of inherenet application resilience for approximate computing, *Design Automaton Conference (DAC)* , 2013.

[5] K. Roy and A. Raghunathan. Approximate Computing: An EnergyEfficient Computing Technique for Error Resilient Applications, *Proc. IEEE Comput. Soc. Annual Symp. VLSI* , pp. 473-475, 2015.

[6] J. Han H. Jiang and F. Lombardi. A comparative review and evaluation of approximate adders, *In Proc. Great Lakes Symp. VLSI* , pp. 343-348, 2015.

[7] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han. A comparative evaluation of approximate multipliers, *In 2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)* , pp. 191-196, 2016.

[8] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Proteus: Exploiting Numerical Precision Variability in Deep Neural Networks, *In Proceedings of the 30th Annual ACM International Conference on Supercomputing, ICS-30 '16* , pages 23–34, 2016.

[9] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. BitFusion: Bit-level Dynamically Composable Architecture for Accelerating Deep Neural Networks, *In Proceedings of the 45th Annual IEEE/ACM International Symposium on Computer Architecture, ISCA-45 '18* , pages 764–775, 2018.

[10] P. Brisk A. K. Verma and P. Ienne. Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design, *Proc. of Design Automation and Test in Europe DATE* , pp. 1250-1255, March 2008.

[11] G. Wang N. Zhu, W. L. Goh and K. S. Yeo. An enhanced low-power high-speed adder for error tolerant applications, *Proc. of International Symp. on Integrated Circuits (ISIC)* , pp. 67-69, 2009.

[12] K. Du. High performance reliable variable latency carry select addition, *Proc. of Design Automation and Test in Europe DATE* , 2012.

[13] A. B. Kahng and S. Kang. Accuracy configurable adder for approximate arithmetic design, *Proc. of Design Automation Conf. (DAC)* , pp. 820-825, June 2012.

[14] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu. On reconfiguration-oriented approximate adder design and its application, *International Conference on Computer-Aided Design (ICCAD)* , 2013.

[15] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. A Low Latency Generic Accuracy Configurable Adder, *In ACM/IEEE Design Automation Conference (DAC)* , 2015.

[16] G . Cas etl D. Esposito. Approximate Adder With Output Correction for Error Tolerant Applications and Gaussian Distributed Inputs, *IEEE International Symposium on Circuits and Systems (ISCAS)* , 2016.

[17] J. Han C. Liu and F. Lombardi. An analytical framework for evaluating the error characteristics of approximate adders, *IEEE Trans. Comput., vol. 64, no. 5* , pp. 1268-1281, May 2015.

[18] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel. Probabilistic Error Modeling for Approximate Adders, *IEEE Transactions on Computers* , pp. 515-530, 2016.

[19] M. Tan G. Liu, Y. Tao and Z. Zhang. CASA: Correlation-aware speculative adders, *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)* , 2014.

[20] Ali Mural Gok and Nikos Hardavellas. VaLHALLA: variable Latency History Aware Local-carry Lazy Adder, *proc. of the on Creat Lakes Synposium on VLSI (GLSVLSI)* , pp. 17-22, 2017.

[21] Samples of High-Resolution music files . `http://helpguide.sony.net/high-res/sample1/v1/en/index.html`, 2014.

[22] Free Lossless Audio Codec . `https://xiph.org/flac/format.html#interchannel`, 2009.

[23] A. A. Del Barrio, R. Hermida, S. O. Memik, J. M. Mendias, and M. C. Molina. Multi-speculative addition applied to datapath synthesis, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)* , vol. 31, no. 12, pp. 1817-1830, Dec. 2012.

[24] da Costa C.M. Diniz, M.B. Fonseca and Sergio Bampi. Evaluating the use of adder compressors for power-efficient HEVC interpolation filter architecture, *Analog Integr Circ Sig Process* , 89:111–120, 2016.

[25] B. Silveira, G. Paim, B. Abreu, M. Grellert, C. M. Diniz, E. A. C. da Costa, and S. Bampi. Power-Efficient Sum of Absolute Differences Hardware Architecture Using Adder Compressors for Integer Motion Estimation Design, *IEEE Transactions on Circuits and Systems I: Regular Papers* , vol. 64, no. 12, pp. 3126-3137, 2017.

[26] M. D. Ercegovac and T. Lang. Digital Arithmetic, *Amsterdam, The Netherlands: Elsevier* , 2003.

[27] W.C. Yeh and C.W. Jen. High-speed booth encoded parallel multiplier design, *IEEE Transaction. Computer* , Vol. 49, issue. 7, Jul. 2000, pp. 692-701.

[28] C. Chang, J. Gu, and M. Zhang. Ultra low-voltage low- power CMOS 4-2 and 5-2 compressors for fast arithmetic circuits, *IEEE Trans. Circuits Syst.* , vol. 51, No. 10, pp. 1985-1997, Oct. 2004.

[29] A. Momeni, J. Han, P. Montuschi, and F. Lombardi. Design and analysis of approximate compressors for multiplication, *IEEE Trans. Comput.* , vol. 64, no. 4, pp. 984-994, Apr. 2015.

[30] A. Gorantla and P. Deepa. Design of approximate compressors for multiplication, *ACM J. Emerg. Technol. Comput. Syst.* , vol. 13, no. 3, pp. 44, May 2017.

[31] M. Ha and S. Lee. Multipliers with approximate 4:2 compressors and error recovery modules, *IEEE Embedded Syst. Lett.* , vol. 10, no. 1, pp. 6-9, Mar. 2018.

[32] D. Esposito, A.G.M. Strollo, and E. Napoli et al. Approximate multipliers based on new approximate compressors, *IEEE Trans. Circuits Syst. I Regul. Pap.* , vol. 65, no. 12, pp. 4169-4182, 2018.

[33] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram. Dual-quality 4:2 compressors for utilizing in dynamic accuracy configurable multipliers, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* , vol. 25, no. 4, pp. 1352-1361, Apr. 2017.

[34] J Castro-Godínez, M. Shafique, and J. Henkel. ECAx: Balancing Error Correction Costs in Approximate Accelerators, *ACM Transactions on Embedded Computing Systems* , Vol. 18, No. 5s, Article 48. Oct. 2019.

[35] C.-H. Lin and C. Lin. High accuracy approximate multiplier with error correction, *In Proc. IEEE 31st Int. Conf. on Computer Design (ICCD)* , pp. 33-38, 2013.

[36] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A Spatial Architecture for Energy-efficient Dataflow for Convolutional Neural Networks, *In Proceedings of the 43rd Annual IEEE/ACM International Symposium on Computer Architecture, ISCA-43 '16* , pages 367–379, 2016.

[37] Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks, *In Proceedings of the 63rd IEEE International Solid-State Circuits Conference, ISSCC-63 '16* , pages 262–263, 2016.

[38] Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks, *IEEE Journal of Solid-State Circuits* , 52(1):127–138, 2016.

[39] Tianshi Chen, Zidong Du, Ninghui Sun, JiaWang, ChengyongWu, Yunji Chen, and Olivier Temam. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning, *In Proceedings of the 19th Annual ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS-19 '14* , pages 269–284, 2014.

[40] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and O. Temam. DaDianNao: A Machine-Learning Supercomputer, *In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-47 '14* , pages 609–622, 2014.

[41] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. ShiDianNao: Shifting Vision Processing Closer to the Sensor, *In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Computer Architecture, ISCA-42 '15* , pages 92–104, 2015.

[42] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, and Andreas Moshovos. Stripes: Bit-Serial Deep Neural Network Computing, *In Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-49 '16* , pages 1–12, 2016.

[43] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst. Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI, *In 2017 IEEE International Solid-State Circuits Conference, ISSCC-'17* , pp. 246-247, 2017.

[44] D. Shin, J. Lee, J. Lee, J. Lee, and H. Yoo. DNPU: An Energy-Efficient Deep-Learning Processor with Heterogeneous Multi-Core Architecture, *In IEEE Micro* , vol. 38, no. 5, pp. 85-93, Sep./Oct. 2018.

[45] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo. UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision, *In IEEE Journal of Solid-State Circuits* , vol. 54, no. 1, pp. 173-185, Jan. 2019.

[46] S. Ryu, H. Kim, W. Yi, and J.-J. Kim. BitBlade: Area and energyefficient precision-scalable neural network accelerator with bitwise summation, *In Proceedings of the 56th Annual Design Automation Conference, DAC-56 '19* , pp. 84:1–84:6, 2019.

[47] Sayeh Sharify, Alberto Delmas Lascorz, Kevin Siu, Patrick Judd, and Andreas Moshovos. Loom: Exploiting Weight and Activation Precisions to Accelerate Convolutional Neural Networks, *In Proceedings of the 55th Annual Design Automation Conference, DAC-55 '18* , page 20, 2018.

[48] Soroush Ghodrati, Hardik Sharma, Cliff Young, Nam Sung Kim, and Hadi Esmaeilzadeh. Bit-Parallel Vector Composability for Neural Acceleration, *arXiv preprint arXiv:2004.05333* , 2000.

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition, *In Proceedings of the IEEE* , vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks, *In Neural Information Processing Systems Conference, NIPS-25 '12* , pp. 1097–1105, 2012.

[51] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, *In International Conference on Learning Representations, ICLR '15* , 2015.

[52] C. Szegedy et al. Going deeper with convolutions, *2015 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '15* , pp. 1-9, 2015.

[53] S. Ren K. He, X. Zhang and J. Sun. Deep Residual Learning for Image Recognition, *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '16* , pp. 770-778, 2016.

[54] T. Yang V. Sze, Y. Chen and J. S. Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey,, *In Proceedings of the IEEE* , vol. 105, no. 12, pp. 2295-2329, Dec. 2017.

[55] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, Raquel Urtasun, and Andreas Moshovos. Reduced-Precision Strategies for Bounded Memory in Deep Neural Nets, *arXiv preprint arXiv:1511.05236v4* , 2015.

[56] Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. WRPN: Wide Reduced-Precision Networks, *arXiv preprint arXiv:1709.01134* , 2017.

[57] Alberto Delmas Lascorz, Patrick Judd, Sayeh Sharify, and Andreas Moshovos. Dynamic Stripes: Exploiting the Dynamic Precision Requirements of Activation Values in Neural Networks, *arXiv preprint arXiv:1706.00504* , 2017.