Title
Power Optimization of Sum-of-Products Design for Signal Processing Applications

Author
Heo, Seok Won

Publication Date
2014

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Power Optimization of Sum-of-Products Design for Signal Processing Applications

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in Computer Science

by

Seok Won Heo

2014

ABSTRACT OF THE DISSERTATION

Power Optimization of Sum-of-Products Design for Signal Processing Applications

by

Seok Won Heo

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2014

Professor Miloš D. Ercegovac, Chair

Power consumption is a critical aspect in today's mobile environment, while higher performance remains a major design goal. In recent mobile devices, the signal processing applications are power-consuming due to the frequent use of arithmetic computations; hence they have a large impact on the overall power dissipation. Specifically, a sum-of-products is a frequently used arithmetic operation in signal processing applications. Conventional designs use separate multipliers and adders in implementing sum-of-products. In this dissertation, we focus on developing a low-power arithmetic unit to perform a sum-of-products operation. The objective of this research is to investigate the algorithmic and architectural approaches for low-power and high-performance design of a sum-of-products with multi-functional computing ability, SIMD and approximate operations, and to demonstrate its capabilities in representative signal processing applications. The key distinguishing features of our approach is to develop a sum-of-products systematically from two aspects: internal efforts considering the arithmetic architecture and external efforts considering input data characteristics. We evaluate the power, delay and area of our solutions, and then compare our designs with similar

arithmetic schemes. The benchmark evaluations are used to identify benefits and limitations of our solutions in signal processing applications.

The dissertation of Seok Won Heo is approved.

Jingsheng Jason Cong

Yuval Tamir

Dejan Marković

Miloš D. Ercegovac, Committee Chair

University of California, Los Angeles

2014

To my parents, brother and wife

Table of Contents

List of Figures

List of Tables

Acknowledgements

After a long formal education journey, I finally have a chance to thank everyone who taught me, helped me, and shared their friendship with me during my Ph.D. program. This dissertation cannot be completed without support from them. Here I am trying to express my gratitude to everyone although these words are never enough.

First of all, I would like to thank my advisor, Miloš D. Ercegovac, for being such a wonderful advisor. His passion during CS252A and CS259 helped me find what I want to study for my Ph.D. He invited me to join his research group when I did not have enough background in the computer architecture field. Through his invaluable guidance, advice and encouragement, he has transformed a struggling graduate student into an experienced researcher. His insight and ideas formed the foundation of my dissertation as much as mine did, and his guidance and care helped me get over a lot of hurdles during my graduate years. He gave me the opportunity to participate in amazing research and inspired me to actively investigate many interesting and challenging problems in computer architecture technologies. When I had a hard time making good research, he always helped me to choose the best path. His advice, support and encouragement helped me to finish this dissertation. I would also like to thank my other dissertation committee, Jason Cong, Yuval Tamir and Dejan Marković. I was fortunate to have them as committee. Their inspiring advice and research guidance made my dissertation more thorough. I wish to thank Professors Yong Surk Lee, Hong-Goo Kang and Jae Won Kim for their inspirational advises.

Second, this work would not have been possible without the support and interest of many individuals at UCLA and Samsung. I thank every member of our group at UCLA. I have felt lucky to be a member of such a wonderful research group. Especially, I thank Seung Hyun Pan for providing a cheerful office environment. I would also like to thank everyone at Samsung for their valuable discussion and suggestions. Especially, I would also like to thank the senior members of Samsung, Chil-Hee Chung. I was lucky to have the opportunity to work with him. More importantly, I am always inspired by his passion for insights. I would like to thank Jin-Hyuck Choi for giving me a chance to work with him at Samsung. He invited me to join his team when I did not have enough background in the IC design field. And then, he encouraged me to solve important and difficult problems after building a strong background. Kyu-hyun Shim gave me good advice and wonderful suggestions for my paper drafts, which eventually became parts of my dissertation. His guidance helped me to define the scope of this research. I also thank Min-Soo Kang and Tae-Keun Jeon for useful discussions and many different kinds of support including system-level simulation. Especially I thank Soong-Man Shim and Dong-Ryul Lee for always being helpful in not only

RTL-level technical support but also helping me with various kinds of compile and synthesis errors. These comments made my dissertation much better, and I learned a lot from their comments.

I would also like to thank the UCLA computer science graduate students for their valuable comments and contribution: Soon Young Oh, Wooseong Kim, Jonathan Park, EunKyu Lee, Sungwon Yang, Seongwon Han, Jihyoung Kim and Jae-Han Lim. My Ph.D. life would have been very different without them. Especially Soon Young, Wooseong and EunKyu had always open to strong technical discussions and showed me how to write strong papers. I thank Sungwon, Seongwon, Jihyoung and Jae-Han for many productive and joyful coffee meetings especially during several years of graduate school. I would also like to thank the UCLA memebers: Moonsoo, Eun-Taek, Jong Jin and Yool. They always gave me kindly advice and great help. I would also like to thank high school friends: Jin Kuk, Minchul, Ho Kwon, Jong Uk and Hyun Jun and many others for their encouragement, friendship and support. Especially, Jin Kuk gave me insightful suggestions whenever I made a decision, and showed me how to write strong papers. He has been my role model in life. In my first year, I stayed with Jong Uk very closely, and I learned how to live as an international student. I also thank Minchul for his comments and suggestions, and Ho Kwon for accompanying me in several trips in California and helping my father health care. Thanks also to the many friends who made it all worthwhile LG internship members: Deuk, Mi Hye and Yonsei Processor Lab. Members: Woo Kyung, Do kyun, Inpyo, Yongjoo and Jin Woo.

I learned a lot during my internships in industry. Mark, Ryan and Vipul were my mentors when I was an intern at Broadcom. They also provided stimulating environment during the internship. I especially thank Mark giving me a chance to intern with his group. Ryan immediately solved problems whenever I was faced with a difficult question. Hwisung walked me through the first steps of being a low-power engineer. Thanks also to the many friends: Seunghwan, Abraham, Stephan and Woojoo. They provided valuable comments on my research and life. I was very fortunate to meet such great people outside of UCLA. Special thanks to my uncle, aunt, cousin: Jim Song, Jacqueline Chiu and Angel, Shashi for educating me in the ways of the American culture, and for being a good friend.

Finally, I wish to express my most sincere gratitude and appreciation to my parents who always had faith in me and gave me endless love and support, my younger brother who always encouraged me and gave me insightful suggestion. My parents firmly believe that I can be a great scholar, and whenever I am in doubt, they constantly remind me of their confidence. Thinking back, it must have been a difficult decision for them to send their first son over thirty years old to a foreign country, and I am grateful for their selfless decision. My younger brother, Seok-Joong, always encouraged me to solve important and difficult problems and guided me to model our design at system level. This dissertation became much stronger and clearer with his contributions. Most importantly, throughout my Ph.D., my wife,

Soh Youn, encouraged me at all times with her love, patience, and understanding. Frankly, it is quite tough to be the fiancé of a graduate student who lives far away from family and her, but she went through the last 4 years without many complaints. I hope that I will be able to give her as much support as I got during my graduate study. All my friends, many UCLA and Samsung folks and many other friends helped me directly or indirectly during my Ph.D. studies. Once again, I gratefully thank everybody and thank my parents, brother and wife. One day I hope that our family and friends will live somewhere nearby and we will all have a peaceful life together.

Seok Won Heo
May 2014
Los Angeles, California

Vita

2002	B.S. in Electrical Engineering and Computer Science
	Hanyang University, Korea
2004	M.S. in Electrical and Electronic Engineering
	Yonsei University, Korea
2012	M.S. in Computer Science
	University of California, Los Angeles

Publications

S. W. Heo, S. J. Huh and M. D. Ercegovac, "Power optimization of sum-of-products design for signal processing applications, in *Proc. ASAP*, Jun. 2013, pp. 192–197.

S. W. Heo, S. J. Huh, and M. D. Ercegovac, "Power optimization ina parallel multiplier using voltage islands," in *Proc. ISCAS*, May 2013, pp. 345–348.

S. Yu, M. Kim, S. W. Heo, J. Song and Y. S. Lee, "An Efficient Scalable and Hybrid Arithmetic Unit for Public Key Cryptographic Applications," *IEICE Electronics Express*, vol. 4, no. 14, pp. 461–466, Jun. 2007.

S. W. Heo, Y. S. Lee, "An Optimal RSA Crypto-processor Design Based on Montgomery Algorithm," in *Proc. ISOCC*, Oct. 2006, pp. 279–282

S. W. Heo, M. Kim, Y. S. Lee, "Study of Optimized Adder Selection", in *Proc. ASIC*, Oct, 2003, pp.1265–1268.

Chapter 1 Introduction

Power consumption has become a critical concern in recent mobile system design, while maximizing performance and minimizing area remain two major design goals. Multiplication is a fundamental arithmetic operation in most signal processing applications, but a multiplier has a large area, long latency and consumes considerable power. Previous works have focused on developing low-power multipliers and have not considered composite arithmetic operations, such as sum-of-products, in reducing power consumption. However, signal processing applications often require many numerical calculations, which may take a large number of clock cycles using a conventional multiplier even when they include a low-latency multiplier. This poses an interesting problem in power optimization of arithmetic operations: what can be gained by using composite arithmetic, i.e., by fusing several basic operations? We propose to investigate power optimization of a well-known composite operation, sum-of-products, which often appears, for example, in computing inner product and Finite Impulse Response (FIR) filtering. In this chapter, we address motivation, define the main research problem, discuss related works and outline our research approach.

1.1 The Main Research Problem

The sum-of-products is found in many digital signal processing and multimedia applications such as a FIR filter, a high pass filter and an inner product. This computation is implemented using a summation of two products. It can be described by

$$S = a \times b + x \times \mathrm{y} \tag{1-1}$$

The inner-product is usually computed by repeatedly using a sum-of-products.

$$S[i + 1] = a[i] \times b[i] + x[i] \times y[i] + S[i] \tag{1-2}$$

Previous research has mainly focused on designs for low-power multipliers, but recent studies have shown that a conventional multiplier design cannot efficiently support increasing high-throughput and low-power requirements [1]. However, the sum-of-products unit can offer an opportunity to satisfy these requirements.

Many Digital Signal Processors (DSPs) and Graphics Processing Units (GPUs) provide multiply and/or Multiply-and-ACcumulate (MAC) instructions because of the frequent use of multiplication and related arithmetic calculations in signal processing applications. To execute sum-of-products operations, they use an existing multiplier or a MAC unit. Conventional processors take more clock cycles when using a single multiplier or a MAC unit to perform sum-of-products. Clearly, by including a sum-of-products operation, one expects that fewer cycles are needed. We want to show that the energy-delay product is also reduced.

Consider a typical FIR filter:

$$y[n] = \sum_{k=-\infty}^{\infty} c[k] \times x[n-k] \tag{1-3}$$

This equation can be implemented in two ways in a high-level language, such as C. One way to do so is as follows:

```
for (k = 0; k < N; k++)
{
    y[n] = y[n] + c[k] × x[n - k]
}
```
(1-4)

The last line corresponds to a MAC operation: $x = x + y \times z$. This equation can be translated into a single MAC instruction.

The FIR filter can also be implemented in C in the following way:

```
for (k = 0; k < N; k+=2)
{
        y[n] = y[n] + c[k] × x[n - k] + c[k + 1] × x[n – k + 1];
 }
```
(1-5)

The last line corresponds to an accumulated sum-of-products: $x = x + y_0 \times z_0 + y_1 \times z_1$. This equation can be translated into a single instruction using the sum-of-products design. This approach is to reduce the total number of clock cycles by half.

Consider another example: matrix multiplication.

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nm} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & \cdots & B_{1p} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mp} \end{bmatrix} A \cdot B = \begin{bmatrix} AB_{11} & \cdots & AB_{1p} \\ \vdots & \ddots & \vdots \\ AB_{n1} & \cdots & AB_{np} \end{bmatrix} \tag{1-6}$$

where the number of columns in A equals the number of rows in B.

This matrix multiplication can be implemented in two ways by using multiplications and sum-of-products operations. One way to use multiplications is as follows:

```
for (i = 0; i < n; i++) {
    for (j = 0; j < p; j++) {
        for (k = 0; k < m; k++) {
            AB[i][j] += A[i][k] × B[k][j]
        }
    }
}
```
(1-7)

The expression (1-7) corresponds to the multiply-add operation which could be executed as a single instruction. Matrix multiplication can be also implemented as

```
for (i = 0; i < n; i++) {
        for (j = 0; j < p; j++) {
                for (k = 0; k < m; k+=2) {
                        AB[i][j] += A[i][k] × B[k][j] + A[i][k+1] × B[k+1][j]        (1-8)
                }
        }
}
```

The last line corresponds to an accumulated sum-of-products operation. In the best case scenario, the sum-of-products operations require only half the number of clock cycles using sum-of-products hardware compared to using a single multiplier, as shown in the example above.

1.2 Motivation

There is a fundamental technological shift taking place in the electronics industry. It is moving from the wired era driven by the Personal Computer (PC) to the wireless era driven by portable devices. As the scale of integration keeps growing, more sophisticated signal processing algorithms are being implemented on mobile Very Large Scale Integration (VLSI) chips [1][3][4]. These signal processing applications not only demand great computation capacity but also consume considerable amounts of power. With a growing number of mobile devices, minimizing the power consumption has become of great importance in today's VLSI system design while performance and area remain the other two major design goals. According to the 2012 International Technology Roadmap for Semiconductors (ITRS), the power demands

are far outstripping the power requirements of typically used designs. Thus, further research in low-power designs is needed to help close this gap [5].

Low-power systems have several advantages over those that do not employ power-saving strategies. Portable systems require the use of low-power design because such a design allows the use of lighter batteries and directly leads a prolonged operation time. Reducing power consumption is also important for high-performance systems that do not need to be portable. High-power dissipation requires the use of more complex cooling and packaging techniques, which are costly to build, operate and maintain. In addition, digital circuits tend to become much less reliable at high operating temperatures; thus low-power technology can directly improve the robustness of Complementary Metal Oxide Semiconductor (CMOS) circuits [6].

Multiplication and MAC operations are frequently executed arithmetic operations in conventional signal processing applications. However, signal processing applications often take many clock cycles using a conventional multiplier even when they include a high-performance parallel multiplier. This is the critical problem for the arithmetic operations in recent signal processing applications which require heavy numerical calculations. Moreover, studies on power dissipation in DSPs and GPUs indicate that a multiplier is one of the most power-hungry components on these chips [7]. Therefore, research on new arithmetic units is needed to satisfy low-power and high-throughput requirements in mobile systems. A great deal of effort has been expended in recent years on the development of techniques to reduce power dissipation while minimizing the throughput degradation. Parallel organization mitigates such throughput degradation [8]. This dissertation proposes a new design for a combined arithmetic unit based on sum-of-products operation for signal processing applications and develops a scheme to achieve power savings in the sum-of-products operation by utilizing parallel organization.

There have been extensive works on multipliers, which are core components of sum-of-products, at all levels from technology through algorithm level. However, low-level

techniques such as circuit or technology-level are not unique to specific structures, and they are generally applicable to all structures. However, the characteristics of arithmetic computation are not considered well at low levels. The power and performance of arithmetic computations are generally determined by specific algorithm and architecture. Moreover, power consumption is directly related to data switching patterns. However, it is difficult to consider application-specific data characteristics in low-level optimization. Therefore, this dissertation addresses high-level optimization techniques for a low-power sum-of-products. High-level techniques refer to algorithm, architecture and logic level techniques that consider arithmetic features and input data characteristics. The main research hypothesis of this dissertation is that high-level optimization of sum-of-products designs produces low-power solutions while maintaining overall throughput. Specifically, we consider how to optimize the internal algorithm and architecture of sum-of-products and how to control active internal resources to match external data characteristics. The primary objective is power reduction without a significant delay overhead. The tradeoff between power, area and delay is also considered in several cases.

1.3 Power Optimization

Power is defined as the rate at which work is performed, whereas energy is a measure of the total amount of power dissipated. Strictly speaking, developing a low-power design is a slightly different goal from creating a low-energy design, even though they are closely related [9][10]. Power is a primary problem when heat dissipation and thermal management are concerns. The peak power is often used for power and ground wiring design, signal noise margin and reliability analysis. Energy is a metric of the energy efficiency of systems, especially in the domain of maximizing battery lifetime.

The total power consumed by CMOS circuits is composed of two main components: dynamic and static power [10][11][12][13]. Dynamic power represents the power consumed by the intended work of the circuit to switch states and thus execute logic functions. Dynamic power is primarily composed of :1) the power dissipation associated with the charging or discharging of the capacitance of the switching nodes and 2) the power dissipation due to short circuit current. The other component is static power dissipation. In a CMOS circuit, either reverse biased PN junction current or sub-threshold channel conduction current is the only source of unintended static current. The total power consumption is summarized in the following equations [10]:

$$P_{total} = P_{dynamic.scc} + P_{static} = P_{dynamic} + P_{scc} + P_{static} \quad (1\text{-}9)$$

$$P_{dynamic} = 0.5 \times C_L \times V_{DD}^2 \times f_p \times N \quad (1\text{-}10)$$

$$P_{scc} = 0.5 \times V_{DD} \times f_p \times N \times I_{peak} \times (t_r + t_f) \quad (1\text{-}11)$$

$$P_{static} = I_{static} \times V_{DD} \quad (1\text{-}12)$$

$P_{dynamic}$ in equation (1-2) represents the dynamic power dissipation because of the charging and discharging of a circuit and wire capacitance loads, where C_L is the load capacitance, V_{DD} is the power supply voltage, f_p is the clock frequency, and N is the switching activity, which is defined as the sum of the $0 \rightarrow 1$ transition probabilities of the node switching in one clock period. P_{scc} in equation (1-3) is the power dissipation due to short circuit current, where I_{peak} is the peak current, and t_r and t_f are the rising and falling time of short circuit current, respectively. The peak current is determined by the saturation current of the devices and thus is directly

proportional to the number of the transistors. P_{static} in equation (1-4) is the static power dissipation, where $I_{static}$is the static current. This static power dissipation is primarily determined by fabrication technology considerations.

Dynamic power dissipation is the dominant factor in the total power consumption of CMOS circuits [6]. Static power dissipation is usually several orders of magnitude smaller than the dynamic power dissipation. Furthermore, many researchers assert that optimizing dynamic power has definitely become more important than optimizing static power because optimizing dynamic power is actually more difficult than static power and optimizing static power is conceptually straightforward. Optimizing static power heavily depends on circuit or technology-level techniques such as dual V_t partitioning and multi-threshold CMOS [14]. Thus, a consideration of static power is neglected. Also, we consider optimizing short circuit power along with optimizing dynamic power. The parameters of P_{scc} in equation (1-3) are remarkably consistent with those of $P_{dynamic}$ in equation (1-2). Thus, reducing dynamic power will decrease the short circuit power consumption as well. Furthermore, the power consumed by the short circuit currents is typically less than 10% of the total dynamic power. Even though dynamic power dissipation is the dominant source of total power consumption, the effect of static power dissipation increases significantly and the static power dissipation will dominate as VLSI manufacturing technology shrinks. Current technology trends indicate that the contribution of static power dissipation will increase rapidly. However, dynamic power optimization of arithmetic unit will still be critical in the future, because dynamic power optimization techniques consider arithmetic computation features and specific input data characteristics. However, the static power is proportional to the leakage current which flows regardless of gate switching. It is difficult to consider specific data characteristics of arithmetic computations in static power optimization. Therefore, we will consider only dynamic power reduction.

The designs to reduce dynamic power dissipation of CMOS circuits can be explored at five levels: technology, circuit, logic, architecture and algorithm levels. Power optimization has been studied at different abstract levels [10][11][12][15]. At the lowest technology level, power reduction can be achieved by improving manufacturing process technology such as small feature size, copper interconnects and insulators with low dielectric constants [12][16]. With the technology support of multiple supply voltages, the lowest supply voltage can be applied on non-critical modules. Wire capacitance and delay imbalances can be reduced during the layout process [17][18]. At circuit level, transistor sizing, transistor restructuring and different circuit logic styles can reduce the power dissipation. At gate level, several techniques have been proposed. Cell sizing and composition, equivalent pin swapping and buffer insertion can achieve power reduction with slight area increase using the Synopsys Power Compiler [19][20]. Gate-level techniques also include signal gating [21][22][23], delay balancing [24], input synchronization [25] and signal polarity optimization [9]. Combinational or sequential blocks not used can be disabled using the clock gating technique during a particular period [26][27][28]. The register bank can be disabled using the Synopsys Power Compiler [29]. To stop propagation of spurious transitions, the retiming technique makes repositions of registers in sequential circuits [20]. At architecture level, there is a large amount of freedom in power optimization. Parallelism and pipeline are two main techniques to achieve high throughput and then trade clock frequency for supply voltage reduction [11]. Synchronous design has prevailed because of its ease of design, tractability of analysis and predictability of performance. However, in terms of a low-power design, the increasing overhead from the clock distribution network essential in synchronous design is posing more and more challenges. The higher speed of modern digital circuits sets the limits in distributing the clock signal to all the required points. Thus, asynchronous systems are investigated to avoid a global clock distribution network and reduce useless signal transitions [30][31][32]. In event-driven systems, components are disabled when they are in idle states [26][33].

Although power optimization can be achieved at all levels, the high-level techniques at algorithm and architecture levels are more efficient than techniques at the middle and the lowest technology levels. At algorithm and architecture level, the optimization affects all four factors, but the middle-level optimization usually affects one or two factors in a limited way. Technology level optimization also affects three important factors: load capacitance, power supply voltage and clock frequency. Not much research has been conducted at high level, even though such optimization achieves the greatest potential power savings. Thus, most designers struggle with an apparent conflict between fast decision-making and accuracy at high level [34]. Furthermore, a sum-of-products operation has many computational features. These features have not been considered well at circuit and technology levels. It is also difficult to consider input data characteristics in low-level power optimization. Therefore, it is desirable to develop algorithm, architecture and gate level power optimization techniques that consider the sum-of-products arithmetic features and operands' characteristics. Therefore, this dissertation reflects the concepts at algorithm, architecture and logic design level. Circuit and technology level designs are outside the scope of this dissertation.

As mentioned above, the goal of this dissertation is dynamic power optimization of a sum-of-products design at algorithm, architecture and logic levels. Based on the equation for dynamic power dissipation in CMOS digital circuits, reduced dynamic power can be achieved by decreasing one or more factors. The dynamic power consumption can be reduced by minimizing the capacitance of circuit nodes wherever possible. The parasitic capacitance in CMOS digital circuits can be reduced by using fewer and smaller devices as well as sparser and shorter interconnection. However, the capacitance of CMOS circuits is determined by the characteristics of the CMOS technology used to fabricate the circuit, which are outside the scope of our research. We do not consider the techniques intended to reduce capacitance. The equation also indicates the power supply voltage has the largest impact on the power

dissipation due to its squared term factor. Unfortunately, lowering power supply voltage causes speed penalties. A great deal of effort has been expended in recent years on the development of techniques to utilize a smaller supply voltage while minimizing performance degradation. Using voltage islands is one way to mitigate such performance degradation by architectural changes of the circuit [35]. Lowering the power supply voltage is a highly effective method for a low-power design. Unfortunately, reducing clock frequency decreases the performance as well as power dissipation. Thus, it might not be expected to meet performance in case of a given performance requirement and increase power-delay product. However, the implementation of composite operations such as a sum-of-products can be used to achieve high throughput while reducing power dissipation by lowering power supply voltage because they provide internal parallelism. In this dissertation, we apply both the design sensitivity and power-delay curve [36][37], and compare our designs with separate recent multipliers. The power consumption also depends on the switching activity of circuits. Techniques to reduce switching activity are effective because 1) they cover many layers of design methodologies from the logic and the architectural levels up to the algorithm, and 2) unnecessary spurious transitions consume in excess of 30% of total energy [38]. Therefore, we also focus on the minimization of switching activities. In summary, this dissertation focuses on reducing power supply voltage and switching activity and demonstrating power-delay optimization.

1.4 Low-Power Multiplier Design

The sum-of-products hardware consists of two multipliers and a single adder. The multipliers consume much more power than the adders; thus, a low-power multiplier designs are critical. In this section, we examine a prior work in algorithm, architecture and gate-level techniques to reduce multiplier power.

There are two main types of multipliers: serial (sequential) and parallel (combinational) multipliers. We consider only parallel multipliers because of their potential for high-performance and prevalent use in digital systems. Parallel multipliers consist of three parts: 1) Partial Products Generators (PPGs), 2) Partial Products Reduction (PPR) and 3) the Carry-Propagate Adder (CPA) [39][40][41].

At the algorithm level, multiplication algorithms differ in the design of PPG, PPR and CPA parts. For PPG, radix-2 digit-vector multiplication is the simplest form because the Partial Products (PPs) are generated by a set of AND gates. To reduce the number of PPs and consequently reduce the area and delay of PP reduction, a higher radix such as 4, 8 or 16 is considered [41][42]. A most popular approach is to reduce the standard radix-4 digit set {0, 1, 2, 3} to a signed-digit set {-2. -1. 0, 1, 2} known as Booth recoding. Radix-4 multipliers require parallel recoders for a tree reduction and multiplexors (MUXs) to form PPs. A potential disadvantage is that they may use long wires and high fan-out required for implementation. It is hard to ensure that signals arrive at the same time; thus, PPGs may generate many glitches, which propagate through the whole multiplier. Higher radix multipliers require sign extension of the partial products. The effects of sign extension techniques and recoder design on energy dissipation were analyzed in [43][44]. The number of sign extension bits can be significantly compressed using the Modified Sign Generate (MSG) algorithm; thus, the number of unnecessary adders used to compress the sign extension bits can be reduced. Another approach to minimize the switching activity in the Most Significant Bit (MSB) is to use a sign magnitude representation [45][46][47]. In the sign magnitude representation, only one bit is allocated for the sign and the rest for the magnitude. In this case, only one bit toggles when the sign is changed. The lack of implementation-specific information limits the optimization of power consumption at the algorithm level. More accurate results can be obtained at the architecture level once the data path and interconnections are fully defined. There are several approaches at

architecture level. For the PPR step, two alternatives exist [41]: reduction by rows performed by an array of adders [48], and reduction by columns performed by an array of counters [49]. In reduction by rows, there are two extreme classes: tree and linear array. The tree structure is a better solution to construct the high-speed multiplier for large operands because the critical path delay of the tree array is proportional to the logarithm of the number of bits in the multiplier [48][49]. Moreover, it has the advantage of lower power dissipation. However, its physical design is rather complicated, due to its complex interconnections. The linear array multiplier has a regular structure and a local interconnection. This translates into a small and dense layout in actual VLSI implementation. However, it cannot achieve high performance because the delay of a linear array is linearly proportional to the operand precision size. Also, this multiplier consumes more power since unbalanced paths exist [50]. However, power and delay can be reduced by using a split array structure [51]. Compared to a non-split Left-to-Right (LR) array multiplier, a 2-level split LR array multiplier consumes approximately 20% less power and has a 15% delay with similar area. Parallelism and pipelining are main techniques to achieve a higher operating frequency at a given supply voltage or, alternatively, a lower supply voltage for a desired throughput [12][52]. Such properties come at the expense of a large area. To exploit parallelism with a scaled power supply voltage, the clustering/partitioning technique was proposed in [53][54]. The cluster width is defined as the distance between the first and the last nonzero bits. Ignoring the positions outside the cluster and performing multiplication with a collection of smaller multipliers in parallel with scaled supply voltages while maintaining given throughput can achieve significant power savings. The scheme to use an ensemble of multipliers with different widths was also proposed in [55]. Only one multiplier is adaptively enabled according to input precision. The power aware multipliers allow the users to select the operational policy: for example, users can select between higher quality and longer battery life. Despite flexibility, the major drawback is the large area overhead. To reduce switching activities, a dynamic range determination unit was proposed in [56][57][58]. Before

computation, the input operand data with the smaller range is used as a recoded multiplier, so the probability of PPs being zero can be increased. The pipelining also reduces the power dissipation [52]. Compared to non-pipelined schemes, the pipelined technique can achieve a higher operating frequency at a given supply voltage or, alternatively, a lower supply voltage for a desired throughput. These low-power schemes for parallel multipliers, however, have larger areas.

At gate level, a number of different techniques such as signal gating and signal bypassing have been proposed. Gate level techniques are more efficient than other techniques because signal gating and bypassing cannot be used at architecture level. The 2-Dimensional (2D) signal gating techniques can achieve power savings for low-precision input data with large dynamic range [31][59][60]. Using a typically large fraction of zero and small valued input, a signal gating approach can achieve power savings by deactivating slices. Compared to previous work, a 2D signal gating reduces the power dissipation by up to 35% [60]. The multiplier divided into several slices detects parts of operands with zero values. Low-power sign extension schemes and self-timed design with bypassing logic for zero PPs in radix-4 multipliers have been proposed in [17]. Another technique is bypassing, which disables the operations in some rows (or columns) [61][62][63]. If the bits of a multiplier (or multiplicand) are zero, the multiplier need not perform the summation of zero PPs, and thus can bypass inputs to outputs. This can save power dissipation with little area penalty. Experimental results show that a bypassing scheme saves 10% of power with 20% area increase [63]. The other technique is signal balancing [24][64][65]. The imbalance of signal delays is reduced by inserting auxiliary logic such as latches and buffers [65].

1.5 Research Approach

The research approach followed in this dissertation is briefly described next. Our research goal is to develop a sum-of-products design that allows the optimization of power dissipation and performance and allows flexible tradeoffs in practical implementations.

First, we identify important factors that affect dynamic power consumption at the algorithm, architecture and gate levels. These factors include internal architectures of a sum-of-products and external data characteristics. Specially, we consider 1) how to control hardware resources to match external data characteristics and 2) how to optimize the internal algorithm and architecture of the sum-of-products unit.

Second, the primary objective is power optimization with reduced area and delay overhead. By using new algorithms or architectures, it is even possible to achieve both power reduction and area/delay reduction. We consider the optimal points between power and delay, and provide comparisons with other designs. This dissertation shows our approaches are superior to other recent designs.

Third, we consider an approximate design with tunable error characteristics for round-off error-tolerant applications as well as an accurate design for error-intolerant applications. Recent mobile systems can tolerate a reasonable amount of computation errors. Thus, we consider the design to allow for error-tolerant operation as well as for correct operation for error-intolerant applications. After conducting statistical error analysis, we design our modules using error-tolerant techniques.

Fourth, we implement the proposed and related previous approaches in technology independent structural Verilog descriptions. The designs are verified using Cadence NC-Verilog and synthesized using the Synopsys Design Compiler and Power Compiler in a Samsung 65 nanometer CMOS standard cell low-power library. Placement and routing process

is performed to obtain more precise results using Synopsys IC Compiler. Interconnect parameters are extracted and back-annotated into Synopsys for more precise delay and power calculation. Delay is obtained from Synopsys PrimeTime, and power is obtained from the Samsung in-house tool CubicWare. Experimental results with comparisons of different schemes are finally presented and analyzed.

1.6 Organization of Dissertation

This dissertation is organized as follows. Chapter 2 presents optimization techniques of reduction structure for array. To reduce power dissipation without performance degradation compared to recent multipliers, structure optimization techniques for PP reduction in LR array multipliers are used here. These techniques include [4:2] adder for PP reduction, 4-level Upper/Lower (UL) split structure and voltage islands. Experiment results show that both power and delay are improved considerably with these techniques.

Chapter 3 proposes a high-performance and low-power CPA. This chapter addresses the problem of adding four carry-save vectors (each two carry-save vectors of two arrays). To improve the speed, reduction structure optimization techniques (see Chapter 2) are combined to develop a high-performance lower-power sum-of-products unit. We present a specific design to match arrival time profiles generated by two arrays, and propose a high-performance and low-power final CPA. Experiments indicate that the sum-of-products with the proposed CPA has less area and power than optimized structures with a conventional fast CPA while keeping the same delay.

Chapter 4 proposes a new arithmetic architecture model for signal processing applications and develops a scheme to reduce power dissipation of a sum-of-products unit by utilizing a parallel organization. This proposed design is compared with an existing high-performance

multiplier and the ARM multiplier. With a proposed sum-of-products design, the effects of a parallel organization versus a solution with a single multiplier are experimentally investigated. The direct implementation of the sum-of-products increases power dissipation and latency because the multipliers are the main cause of power dissipation and the adder contributes significantly to the overall delay in a sum-of-products unit. We address these components in Chapter 2 and Chapter 3, respectively.

Chapter 5 proposes the design for a sum-of-products unit that supports multiplication, multiply-add, square, sum-of-squares and add-multiply operations based on an input control signal. Most DSPs and GPUs include separate dedicated arithmetic units for supporting these arithmetic operations. Such an implementation is less suitable for these chips, in which the frequency of arithmetic operations is application dependent. Thus, we focus on developing the sum-of-products unit capable of supporting multiple arithmetic operations using essentially the same hardware. Compared to a conventional sum-of-products, the proposed multi-functional unit has a modest increase in power, area and delay, but allows several multiplication-related arithmetic computations to be performed on the same hardware.

Chapter 6 presents an approximate and Single Instruction Multiple Data (SIMD) sum-of-products unit capable of supporting several arithmetic operations with multiple precisions. To further reduce power dissipation in a sum-of-products with large-dynamic-range input data, multiple-precision and SIMD and approximate operation techniques are proposed. This unit can perform multiple-precision sum-of-products and multiplication for SIMD, approximate and accurate versions using essentially the same hardware as a sum-of-products with only a small increase in delay compared to a conventional sum-of-products. This technique does not change the basic structure of a sum-of-products. Instead, the fundamental components -multipliers and adders- are partitioned and ancillary logic gates are inserted along the gating

boundaries. For input data with a large dynamic range, significant power reduction can be achieved in the experiments.

Chapter 7 summarizes the contributions of this research, discusses conclusions and proposes future directions.

Finally, Appendix A gives a detailed description of the design and experimental methodologies used in our research, and Appendix B defines abbreviations.

Chapter 2 Power Optimization of an Array Multiplier

In this dissertation, we propose a new design for a sum-of-products unit suitable for signal processing applications and present an approach to reducing power dissipation in the design of sum-of-products operation by utilizing two multiplier arrays and CPA. The optimization of the PPR array and the final CPAs is necessary because they are core components. Chapter 2 and Chapter 3 present optimization techniques for the PPR array and the CPA, respectively. Chapter 4 experimentally investigates the effects of a parallel organization with the optimized multiplier array and CPA versus a solution with a single multiplier.

This chapter considers how to optimize the core component, a multiplier for a low-power design. Our goal is to reduce the power consumption without significant increase in the latency and the complexities of multipliers. We present different methods for the multiplier power savings. The following structure optimization techniques are considered: radix-4 recoding scheme, split structure, [4:2] adder and voltage islands. These techniques reduce the power of multiplier arrays significantly without large delay, area overhead and increase in design complexity. When exploring these power optimization techniques, we consider only LR array multipliers with the final CPA. The previous studies demonstrate that LR array multipliers have the potential of saving power and delay, because glitches in LR array multipliers are fewer than in the conventional RL array multipliers, especially for data with a large dynamic range. We will consider the delay optimized final CPA in Chapter 3 on a high-performance low-power sum-of-products. For simplicity, we consider a 32 × 32-bit multiplier. All of these methods can be easily extended to deal with other types of fixed-point operands.

2.1 Introduction

To meet the application's performance requirements, the parallel multipliers are commonly used in high-performance signal processing applications [42][48][49][66]. The parallel multipliers require a large amount of logic, but can compute a product much more quickly than the serial method of shifting and adding that was typical of earlier computers. The parallel multipliers consist of three main computational blocks: a PPG, a PPR array and a CPA. The first stage is the generation of the PPs. The simplest way of PP generation is that the multiplicand and the multiplier are multiplied bit by bit to generate the PPs, and implemented using two-input AND gates. The advanced approach for high-speed is the radix-4 algorithm, which has been used to reduce the number of PPs at the expense of more complex radix-4 recoders and PPG circuitry [42]. The next stage is the PP reduction. In this stage, two-bit vectors are added up repeatedly until bit vectors (carry, sum) are obtained. Two reduction approaches are common in current implementations: trees and linear arrays. The tree structure is the best solution to construct the high-speed multiplier for large operands because critical path delays are proportional to the logarithm of the number of bits in the multiplier [48][49]. It adds the PPs in parallel using redundant adders. Moreover, the tree structure has a low probability of occurrence of glitches because most inputs to Full Adders (FAs) at each stage are naturally synchronized. The linear array structure, however, has a high probability of occurrence of spurious transitions because all FAs start computation at the same time without waiting for the propagation of sum and carry signals from the previous stage [24]. Most input signals of the adder in the tree structure arrive simultaneously; hence a tree structure includes inherently more balanced delay paths in a PPR module. However, its physical design is rather complicated, due to its complex interconnections; thus, the tree structure occupies more area than a corresponding array structure. Because of its high wiring density, its area increases. Specifically, this problem becomes critical in deep sub-micron designs. In contrast, the array structure has a regular and

local interconnection in the reduction array. This regularity translates into a small and dense layout in VLSI implementation. Moreover, as the interconnection becomes critical in deep sub-micron designs, an architecture with regular and local interconnection is highly desirable. However, the array structure has an architectural disadvantage in terms of power dissipation. It has more unbalanced delay paths in a PPR module and thus introduces many glitches [67][89]. Large latency and high-power dissipation limit its use in applications with large size operands.

Among three main components, the PPR module determines the overall multiplier power. The effect of the radix-4 recoder on the overall power dissipation is not obvious because it is additionally implemented and introduces a large number of spurious transitions, while a large amount of arrays implemented can be reduced. For accurate results, we first implemented two types of 32 × 32-bit multipliers: trees and arrays with/without the radix-4 recoder. Power has been measured using only random test data. The results are shown in Table 2.1. The power dissipation introduced by the PPR module is about 60% of the total power dissipation in parallel multipliers; hence, power savings in the PPR modules will result in major enhancement of the power reduction of the parallel multiplier. The power consumption of the PPG circuitry and the radix-4 recoding logic are also critical. Although they consume less than 20% of the total power dissipation, these modules affect the power dissipation in subsequent PPR and CPA significantly. They are the first stage on the long path in parallel multipliers; hence, they introduce extra unbalanced signal propagation, due to the additional delay on the path from operand to the product. In this chapter, we focus on reducing the power of the PPG and PPR parts.

TABLE 2.1: POWER DISTRIBUTION IN A PARALLEL MULTIPLIER

(A) POWER DISSIPATION OF TREE MULTIPLIER COMPONENTS

Components	Power Distribution (%)
PPG, Radix-4 recoder	18.68
PPR Tree (Wallace, [3:2]Adder)	58.47
CPA (Carry-Select Adder)	22.85

(B) POWER DISSIPATION OF ARRAY MULTIPLIER COMPONENTS

Components	Power Distribution (%)
PPG (Radix-2)	4.54
PPR Array ([4:2]Adder)	68.88
CPA (Carry-Select Adder)	26.58

2.2 Related Work

Various approaches have been proposed to reduce the power consumption of multipliers from the algorithm level to the gate level. Recently, LR array multipliers have been proposed and developed. LR array multiplication provides a competitive alternative to the conventional Right-to-Left (RL) array multiplication as LR computation has the potential of saving power and delay. It was discovered that glitches in LR computation are fewer than in the conventional RL computation, especially for data with a large dynamic range. In [68], the power consumption in the LR PPR array for radix-4 recoded multiplication is studied in detail for DSP applications. A low-power LR array multiplier without final CPA is designed using strategically placed (3,2), (5,3) and (7,4) counters [69] and the modified on-the-fly converter [70]. An asynchronous array multiplier with split RL upper array and LR lower array is proposed to make the computation time faster with relatively lower power consumption [32]. The previous studies demonstrate

that the LR linear array multipliers that integrate an array splitting technique are better than tree multipliers in terms of power while keeping similar delay and area for up to 32 bits [51][87][88]. Therefore, we focus on developing a multiplier based on the split LR array structure.

2.3 The Left-to-Right Array Multiplier

In conventional RL array multipliers, the PPs are added sequentially from the rightmost multiplier bit. In contrast, in LR array multipliers, the PPs are added in series starting from $b_{n-1}A$, as shown in Figure 2.1 [71]. Of the two designs, LR array multipliers have the potential of saving power and delay because the carry signals propagate through fewer stages, which reduces the power consumption in the Most Significant (MS) region. LR array multipliers are superior for data with large range because PPs corresponding to sign bits with low switching activities are located in the upper region of the array [72]. Figure 2.2 shows the implementation of an 8 × 8 LR array multiplier [72]. The black dots correspond to the bit matrix in Figure 2.1, obtained with two-input AND gates. Each "3" symbol is a FA and each "2" symbol is a Half Adder (HA). The numbers associated with wires are signal arrival times assuming a unit delay model. For theoretical analysis, the delay of a 2-input XOR gate, T_{XOR2}, is used as the base unit delay. The brown cells on the left are used to add three bits each column from the array into two bits. The gray cells in the last row comprise a Carry-Ripple Adder (CRA) which generates the Least Significant (LS) half of the final product and carry-in of the final CPA. There is no delay penalty due to the use of CRA, as the arrival times of these carry and sum bits match the computation direction and speed of the CRA. The final CPA generates the MS half of the final product. Using a fast CPA, final addition is conducted, and thus the delay can be reduced.

2.4 Structure Optimization

Several power reduction techniques have been originally proposed for RL multiplication. Recently, these low-power techniques which followed the tradition in RL multiplication have also been proposed for LR array multiplication. It was obvious that the use of these techniques in a LR array multiplier was also efficient in power [51][72]. However, it is unknown if there are other better candidates from the perspective of low power. Detailed studies are desirable to explore the potential advantages of LR array multipliers. In this section, we present several structure optimization techniques for low-power LR array multipliers.

FIGURE 2.1: BIT MATRIX FOR LR MULTIPLICATION EXAMPLE (RADIX-2, N = 8)

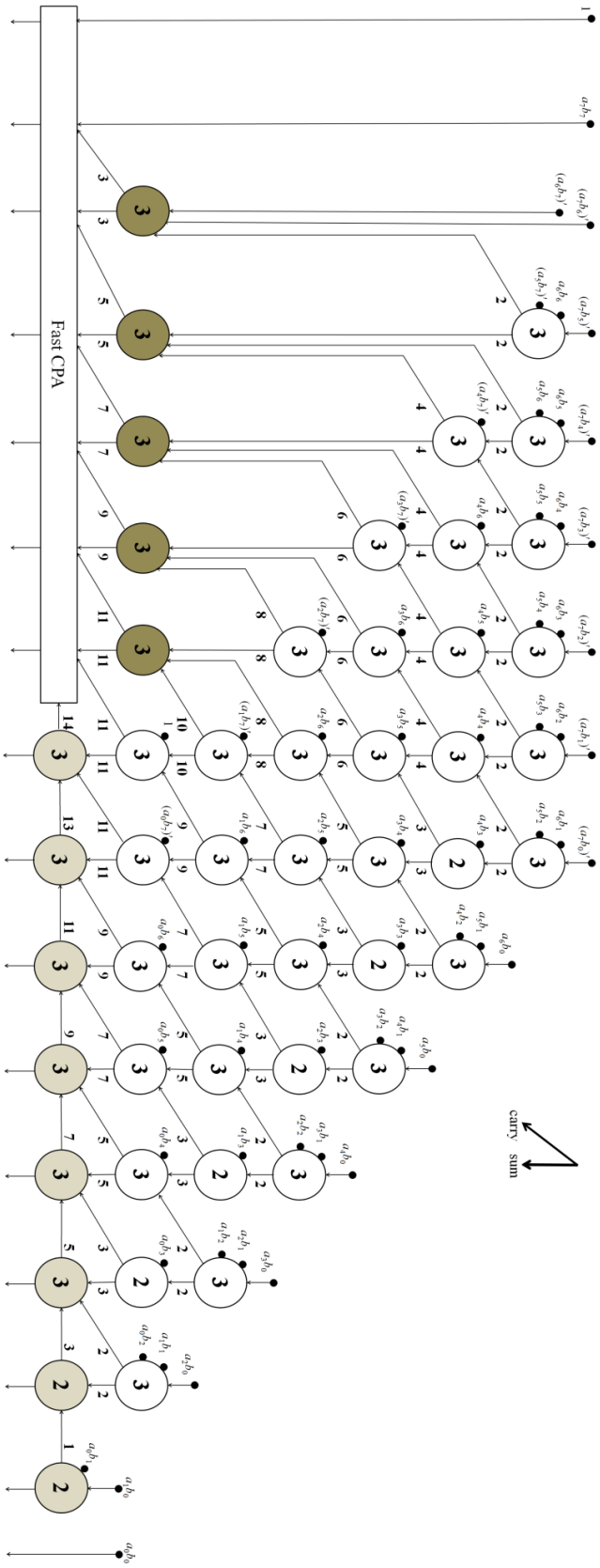


FIGURE 2.2: LR ARRAY MULTIPLIER BASED ON A [3:2] ADDER (RADIX-2, $N = 8$) (ADAPTED FROM [72])

2.4.1 Partial Product Generation with Radix-4 Recoding

We present here several different methods for generating PPs. The simplest method to produce n PPs is to use two-input AND gates, where n is the length of the input operands. However, another recoding scheme introduced by Booth [42] reduces the number of PPs. Among several Booth algorithms, radix-4 recoder with digit set {-2, -1, 0, 1, 2} and radix-8 recoder with digit set {-4, -3, -2, -1, 0, 1, 2, 3, 4} are commonly used in parallel multipliers. Radix-4 and radix-8 algorithms generate $\lceil n/2 \rceil$ and $\lceil n/3 \rceil$ PPs, respectively. Intuitively, the radix-4 and radix-8 recoding scheme may reduce the power consumption, hardware cost and improve performance because the amount of power, area and delay depends on the number of PPs to be added. Specifically, if the number of PPs is reduced in the array multiplier, glitches can be significantly reduced because the lower portion is polluted by frequent switches in the upper portion. However, the PP reduction is obtained at the expense of the extra recoding logic and a more complex PPG circuitry. Moreover, radix-4 (or radix-8) recoding and PPG modules are the first stage on the long path in multipliers, and thus they introduce extra unbalanced signal propagation due to the additional delay on the path from operand to the product. Therefore, radix-4 (or radix-8) recoding and PPG modules affect the power dissipation in subsequent PPR array and CPA significantly, even though they consume only a small portion of the total power.

We want to determine which recoding method is efficient for a low-power multiplier. We first study the characteristics of the most common algorithms: radix-4 and radix-8 algorithms. In the radix-4 recoding, only the multiples +1/–1 and +2/–2 of the multiplicand will be required, all of which are efficiently generated through simple shifts and negation. This simple requirement to generate PPs leads to significant delay savings. On the other hand, the radix-8 recoding scheme can further reduce the number of PPs, but requires a time-consuming extra addition to generate the +3/–3 of the multiplicand. This requirement leads to a large delay penalty, on the order of 15% ~ 20%, as compared with a radix-4 recoding . The extra addition stage of the

radix-8 recoder introduces more unbalanced signals. Specifically, as a recoder and PPG circuitry are the first stage on the long path, glitches are significantly increased as signals propagate through the path. This is a significantly negative factor for power and delay. Therefore, we will apply the radix-4 modified Booth algorithm.

2.4.2 The [4:2] Adder for PP Reduction

A [4:2] adder has been widely used in parallel multipliers. Figure 2.3 illustrates [4:2] adder structure. It has the same gate complexity as two cascaded [3:2] adders, but is faster than its counterpart because it has $3 \times T_{XOR2}$ delay while each single [3:2] adder has $2 \times T_{XOR2}$ delay. Thus, by using the [4:2] adder, the PPR delay is reduced by about 25% without area penalties. The delay reduction is useful for power savings as less switching activities can be generated when signals propagate fewer stages. Moreover, compared to two cascaded [3:2] adders, a [4:2] adder has a more balanced structure and regular interconnection. It reduces the physical complexity. At gate level, the area of the [4:2] adder is very close to that of the [3:2] adder, but will become smaller after placement and routing, due to regular structures.

Figure 2.4 illustrates an 8 × 8-bit LR array reduction using a [4:2] adder. Dark dots are PP bits. Gray dots are carry/sum vectors from adders. Figure 2.5 shows an 8 × 8-bit LR array multiplier using a [4:2] adder. Blue rectangles are [4:2] adders, and each "+" symbol is an inner FA of the [4:2] adder. The first [4:2] adder row accepts four PPs and generates two carry-save vectors. Each subsequent [4:2] adder row accepts two previous carry-save vectors and two new PPs, and then generates two current carry-save vectors. In a LR array multiplier using a [4:2] adder, the CRA is no longer suitable to add the right half carry/sum vectors from the reduction array because the vector bits arrive faster than the CRA computation. To avoid becoming the critical path, CRA should be replaced by a faster CPA. For vectors from the left part of the reduction array, the brown cells on the left which comprise adders are still needed because about

half of the cells on the left have three bits. The previous research demonstrated that a 32 × 32-bit LR linear array multipliers using a [4:2] adder are better than using a [3:2] adder in terms of power and delay [51]; thus, we will utilize a [4:2] adder here.

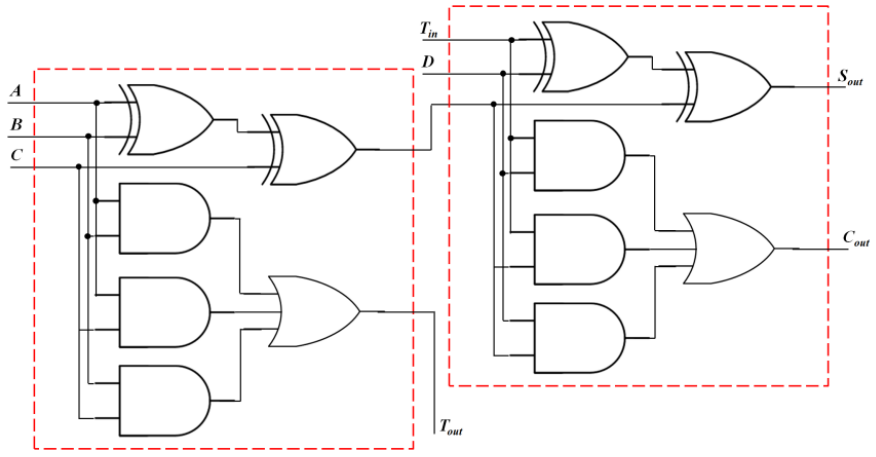


FIGURE 2.3: [4:2] ADDER STRUCTURE

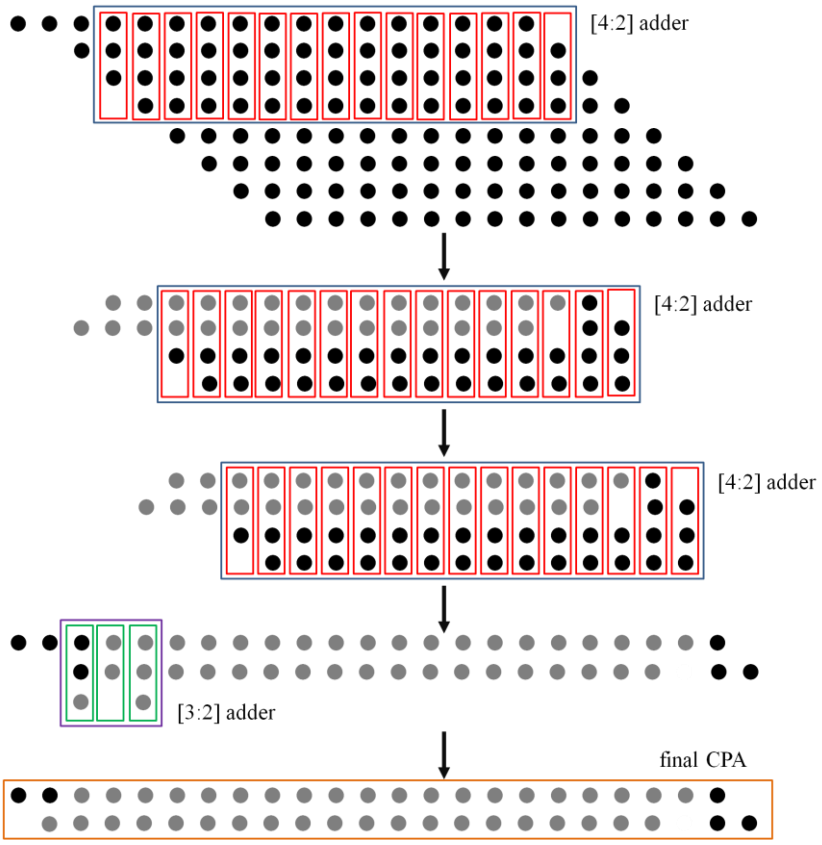


FIGURE 2.4: LR ARRAY REDUCTION USING [4:2] ADDER (RADIX-2, N = 8)

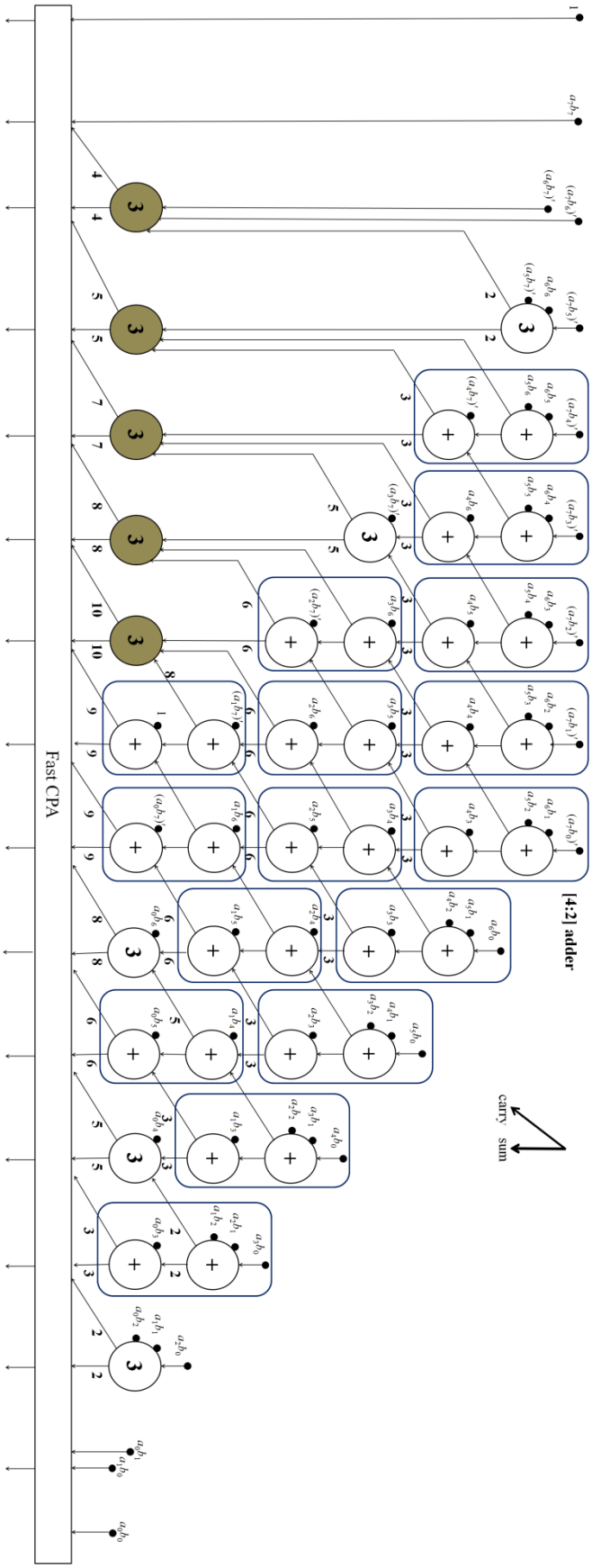


FIGURE 2.5: LR ARRAY MULTIPLIER BASED ON A [4:2] ADDER (RADIX-2, $N = 8$)

2.4.3 The Split Array: Even/Odd and Upper/Lower

The main reason for large power consumption in an array multiplier is unbalanced signal arrival change in the adders. Unbalanced arrival of the signals at the adder is just a consequence of the above. In any case, adder is not the main reason for large power consumption. All FAs in array start computing at the same time without waiting for the propagation of sum and carry signals from the previous stage. This results in many glitches and consumes large power. Carry and sum inputs arrive at different times; thus, this structure introduces a large number of glitches. Specifically, the lower rows consume much more power than the upper rows in the PPR array because glitches cause a snow ball effect as signals propagate through the array [24]. Therefore, if the length of the array could be reduced, there would be power savings. The way to reduce the length of the array is to split the PPR array into several parts and reduce each part in parallel. The final vectors from each part are reduced to two vectors using a [4:2] adder. The previous studies have mainly focused on developing 2-level split array designs. These techniques split the PPR array into two parts with each part having half as many rows. Potentially, there would be greater power and delay savings if each part were split further because a shorter length of array reduces the number of glitch transitions. However, it is still unknown how much would be gained by further splitting. Thus, theoretical analysis and experimental results are desirable in order to explore the potential advantages of multi-level split array design. Considering a 32 × 32-bit multiplier, we predict that a 4-level split array structure will be a good example. In the case of 8-level splitting, each part has only one row. It is obvious we avoid further splitting here. There are two types of split array structures: Even/Odd (EO) and UL structures. In the EO split array structure, the array is broken into two parts: even rows in one part and odd rows in another part, as shown in Figure 2.6(a) [74]. The other possibility is to separate the array into upper and lower parts, as shown in Figure 2.6(b). Among two existing split structures, the 2-level UL split array multiplier presents the less power

consumption result compared to its 2-level EO counterpart [72]. Furthermore, the UL split structure allows simpler interconnection; thus, the physical regularity of array multipliers will also be maintained by interleaved placement and routing. It will still be a good choice for 4-level array splitting. Thus, we will use the UL structure for a 4-level array structure here. All designs analyzed here assume 32-bit integer operands and a [4:2] adder are used in all multiplier designs. A [4:2] adder has the same gate complexity as two cascaded [3:2] adders, and it has $3 \times T_{XOR2}$ delay.The estimates do not include buffers.

The total cell area and delay of the baseline radix-2 non-split array structure is estimated as

$$\begin{aligned} \text{Area} &= 480 \times A_{[4:2]ADDER} + A_{CPA} \\ \text{Delay} &= 45 \times T_{XOR2} + T_{CPA} \end{aligned} \tag{2-1}$$

In a 32 × 32 radix-2 non-split array structure, a linear array has 30 rows because the first adder row accepts three PPs, and subsequent adder row accepts one PP. This array has 15 [4:2] adder, and each [4:2] adder has $3 \times T_{XOR2}$ delay. We estimate the cell area and delay of radix-2 2-level and 4-level UL split array structures based on Figure 2.7.

The total cell area and delay of the radix-2 2-level UL split array structure is also estimated as

$$\begin{aligned} \text{Area} &= 524 \times A_{[4:2]ADDER} + A_{CPA} \\ \text{Delay} &= 24 \times T_{XOR2} + T_{CPA} \end{aligned} \tag{2-2}$$

In a 32 × 32 radix-2 2-level UL split array structure, each array has 14 rows because the first adder row accepts three PPs and subsequent adder row accepts one PP. The total delay is calculated by 7 × [4:2] adder delay + 1 × additional [4:2] delay.

Compared to the radix-2 non-split array structure, the delay of the radix-2 2-level UL split array structure (LR_42_Split2) is reduced about 40% with about 10% area penalties, due to a shorter critical path in the PPR array and extra summation stage.

The total cell area and delay of the radix-2 4-level UL split array structure (LR_42_Split4) is also estimated as

$$\begin{aligned} \text{Area} &= 576 \times A_{[4:2]ADDER} + A_{CPA} \\ \text{Delay} &= 16 \times T_{XOR2} + T_{CPA} \end{aligned} \tag{2-3}$$

Compared with a 2-level, a 4-level UL split array structure has two main advantages. Each part has only 8 PPs instead of 16. It reduces glitches significantly, due to the shorter array length. Also, 4-level splitting reduces the critical path delay. The critical path of one part of the PPR array in a LR_42_Split4 is about 10 × T_{XOR2} due to two [4:2] and two CRA stages, while that in a LR_42_Split2 is 21 × T_{XOR2} due to seven [4:2] stages. This delay reduction is positive for power savings, as a lower supply voltage is used in conjunction with lower clock frequencies to

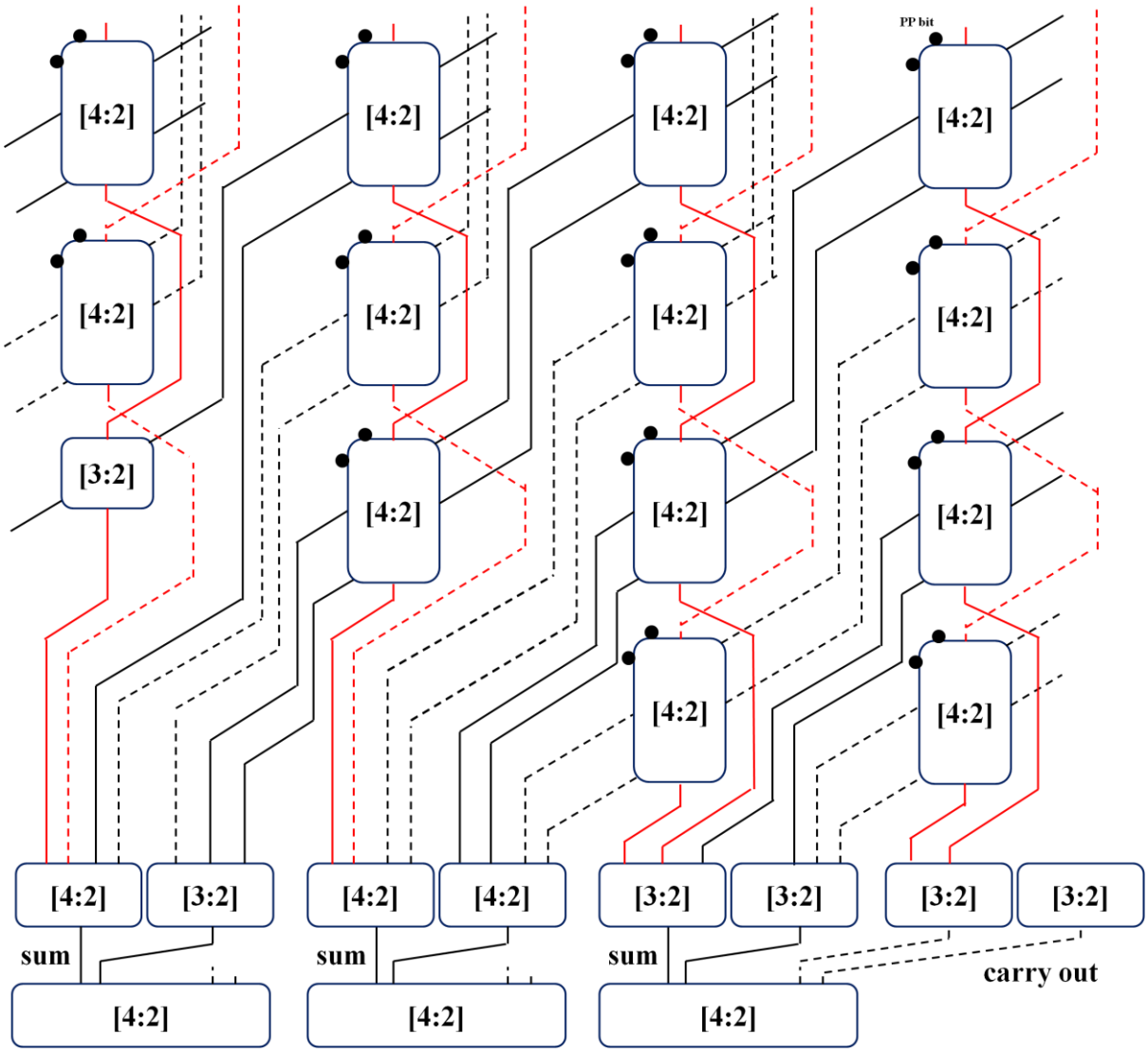
minimize power consumption. On the other hand, a LR_42_Split4 has one more [4:2] addition stage. This addition leads to only an additional $3 \times T_{XOR2}$.

The other way to reduce the length of the PPR array is to use the radix-4 Booth recoding scheme. It can decrease the power dissipation of a multiplier by reducing the number of PPs generated. The total cell area and delay of the radix-4 2-level UL split array structure (LR_42_Split2_ Radix4) is estimated as

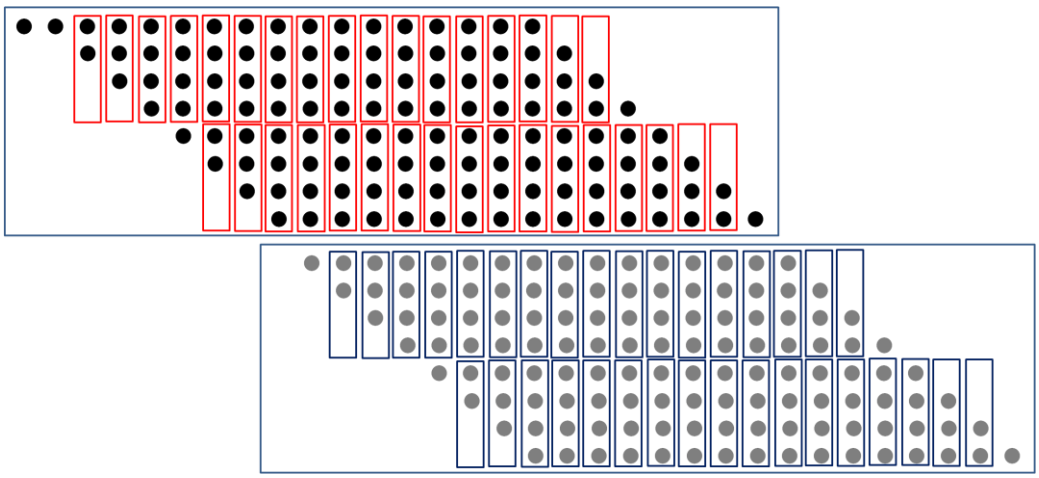
$$\begin{aligned} \text{Area} &= 442 \times A_{[4:2]ADDER} + A_{CPA} \\ \text{Delay} &= 14 \times T_{XOR2} + T_{CPA} \end{aligned} \tag{2-4}$$

Additionally, consider the power dissipation of each structure. Based on the equation for dynamic power dissipation in CMOS digital circuits in Chapter 1, reduced dynamic power can be achieved by decreasing one or more of these factors: the load capacitance, the power supply voltage, the clock frequency and the switching activity. We assume that all designs were executed with the same supply voltages. Dynamic power is proportional to the amount of hardware used to implement the design. Furthermore, the delay reduction is effective for power savings. With regard to switching activities, a 4-level split design eliminates significant glitches, but the radix-4 Booth recoder may introduce a lot of glitches to subsequent modules. All things considered, the radix-2 4-level split or the radix-4 2-level split structures will be the best power-saving structures.

A 16 × 16-bit 2-level and 4-level UL LR array reduction using a [4:2] adder is illustrated in Figure 2.7. Dark dots are PPs. Gold, gray and pink dots are carry and sum vectors from [4:2] adders. A portion of a LR_42_Split4 for uppermost PPs is illustrated in Figure 2.8. A high-level description of a 4-level UL LR array reduction scheme is given in Figure 2.9.

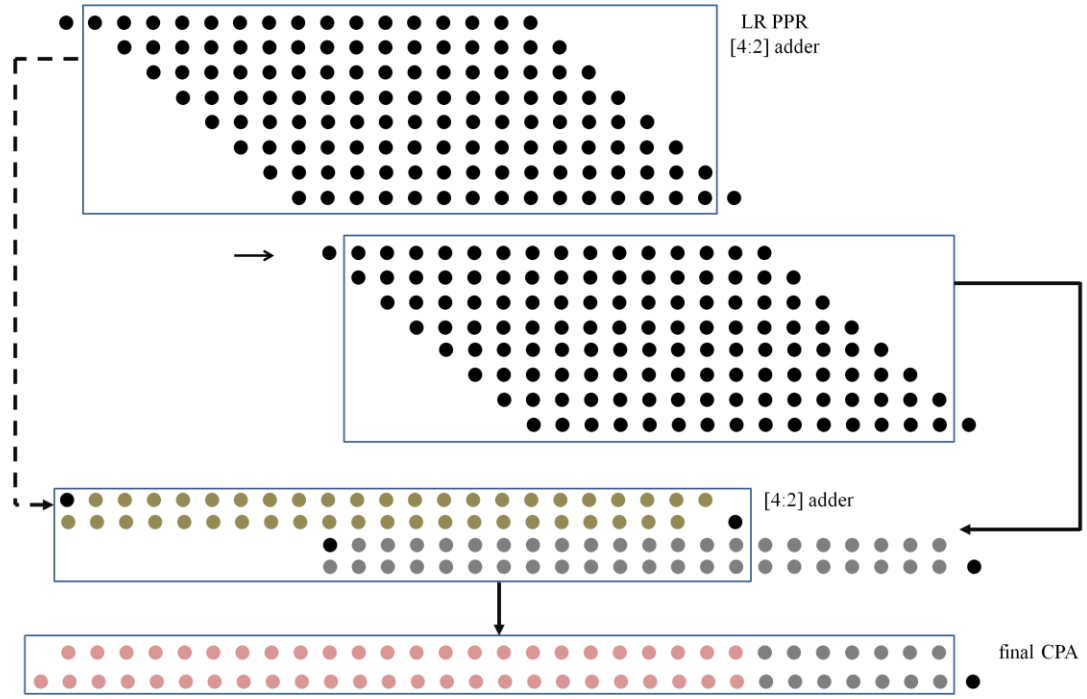


(A) EO SPLIT ARRAY STRUCTURE (ADAPTED FROM [24])

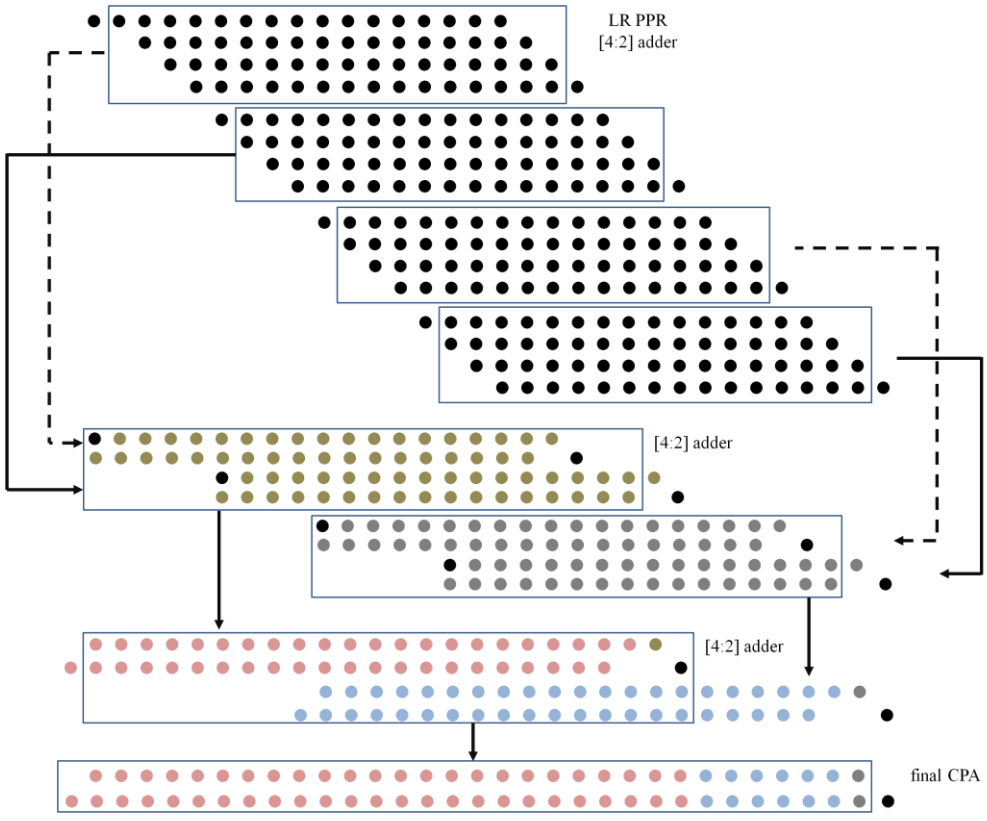


(B) UL SPLIT ARRAY STRUCTURE (ADAPTED FROM [72])

FIGURE 2.6: SPLIT ARRAY MULTIPLIER



(A) 2-LEVEL SPLIT STRUCTURE (ADAPTED FROM [51])



(B) 4-LEVEL SPLIT STRUCTURE

FIGURE 2.7: UL LR ARRAY REDUCTION USING A [4:2] ADDER (RADIX-2, N = 16)

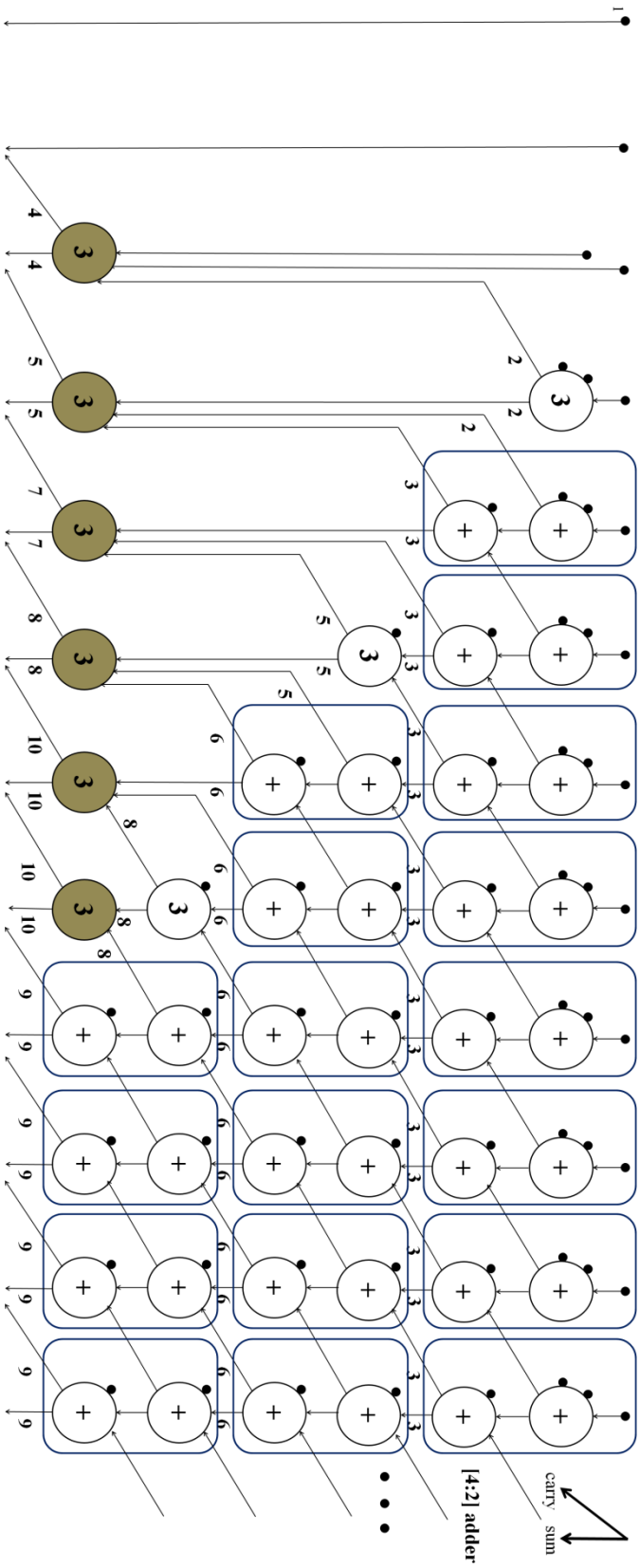


FIGURE 2.8: PORTION OF A 4-LEVEL UL LR ARRAY STRUCTURE FOR UPPERMOST PPS (RADIX-2, $N = 32$)

h: the number of PPs;
$q = h / 4$: quarter of h (assume to be even)
$r = 2\ h / 4$: quarter of h (assume to be even)
$s = 3\ h / 4$: quarter of h (assume to be even)

$(1S, 1C)_i$: sum and carry vectors in the uppermost part
$(2S, 2C)_i$: sum and carry vectors in the second part
$(3S, 3C)_i$: sum and carry vectors in the third part
$(4S, 4C)_i$: sum and carry vectors in the lowermost part

/ the uppermost part using [4:2] adder */*
$(1S, 1C)_0 = \text{ADDER42}(\text{PP}_0, \text{PP}_1, \text{PP}_2, \text{PP}_3)$;
for i from 1 to $q - 6$ do
    $(1S, 1C)_i = \text{ADDER42}(\text{PP}_{i+3}, \text{PP}_{i+4}, 1S_{i-1}, 1C_{i-1})$;
end for

/ the second part using [4:2] adder */*
$(2S, 2C)_0 = \text{ADDER42}(\text{PP}_q, \text{PP}_{q+1}, \text{PP}_{q+2}, \text{PP}_{q+3})$;
for i from 1 to $q - 6$ do
    $(2S, 2C)_i = \text{ADDER42}(\text{PP}_{i+q+3}, \text{PP}_{i+q+4}, 2S_{i-1}, 2C_{i-1})$;
end for

/ the third part using [4:2] adder */*
$(3S, 3C)_0 = \text{ADDER42}(\text{PP}_r, \text{PP}_{r+1}, \text{PP}_{r+2}, \text{PP}_{r+3})$;
for i from 1 to $q - 6$ do
    $(3S, 3C)_i = \text{ADDER42}(\text{PP}_{i+r+3}, \text{PP}_{i+r+4}, 3S_{i-1}, 3C_{i-1})$;
end for

/ the lowermost part using [4:2] adder */*
$(4S, 4C)_0 = \text{ADDER42}(\text{PP}_s, \text{PP}_{s+1}, \text{PP}_{s+2}, \text{PP}_{s+3})$;
for i from 1 to $q - 6$ do
    $(4S, 4C)_i = \text{ADDER42}(\text{PP}_{i+s+3}, \text{PP}_{i+s+4}, 4S_{i-1}, 4C_{i-1})$;
end for

/ add the uppermost and second vectors using [4:2] adder */*
$(P1S, P1C) = \text{ADDER42}((1S, 1C)_{q-6}, (2S, 2C)_{q-6})$;

/ add the third and lowermost vectors using [4:2] adder*/*
$(P2S, P2C) = \text{ADDER42}((3S, 3C)_{q-6}, (4S, 4C)_{q-6})$;

/ add the upper and lower vectors using [4:2] adder */*
$(PS, PC) = \text{ADDER42}((P1S, P1C), (P2S, P2C))$;

FIGURE 2.9: HIGH-LEVEL 4-LEVEL UL SPLIT LR ARRAY REDUCTION ALGORITHM

2.4.4 Voltage Islands

The dynamic power equation indicates that the power supply voltage has the largest impact on the dynamic power dissipation due to its squared term factor. Unfortunately, the lowering power supply voltage causes speed penalties. A great deal of effort has been expended in recent years to develop techniques to utilize the low power supply voltage while minimizing the performance degradation. Using voltage islands is one way to mitigate such performance degradation by architectural changes of the circuit [35].

The problem of constructing the final adder when all input bits arrive at the same time has been well studied [75][76]. However, starting with the input bits for Least Significant Bit (LSB), the delays first increase with the bit numbers and then decrease, as can be seen from Figure 2.10. This non-uniform arrival of inputs to the adder produced by the PPR array has been used in reducing the power of the multiplier by decomposing it into several parts.

We investigate the non-uniform arrival time profiles of the array multiplier to achieve power savings with minimal performance degradation. Specifically, we apply the voltage islands technique to the regions of non-uniform input generated by the PPR array [77]. That is, adders are partitioned into blocks that operate with different power supply voltages. A voltage island occupies a contiguous physical space and operates at one supply voltage. Such voltage island techniques are applied to the array multiplier so that the units of the multiplier get different levels of voltage support, as profiled by their performance requirements. The slowest region of the array multiplier is the middle region at which the arrival time is large and constant. It requires a higher supply voltage level in order to maximize the element's performance. On the other hand, the other regions may run at a lower level of the supply voltage because they are not on the critical path. These regions are 1) the LS part at which arrival time increases from the LSB toward the middle region and 2) the MS part where arrival time decreases from the middle

region toward the MSB. An example of a partition into Low-High-Low islands is shown in Figure 2.11.

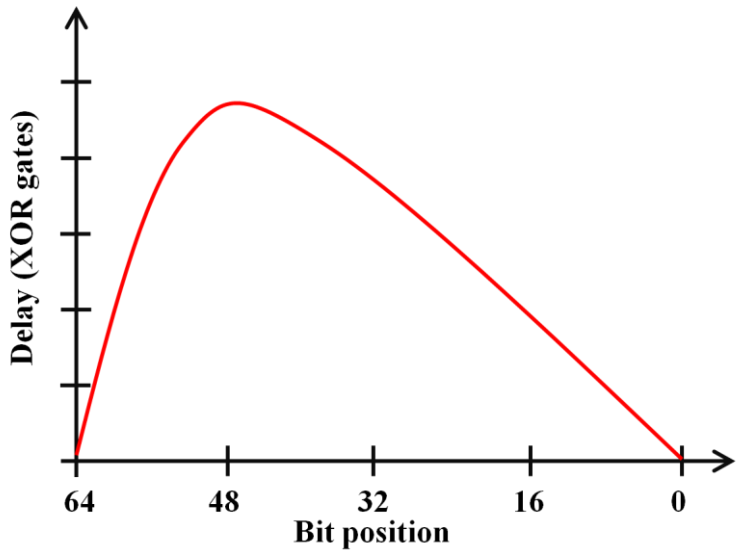


FIGURE 2.10: THE EXAMPLE OF NON-UNIFORM ARRIVAL TIME FOR A 32 × 32 MULTIPLIER

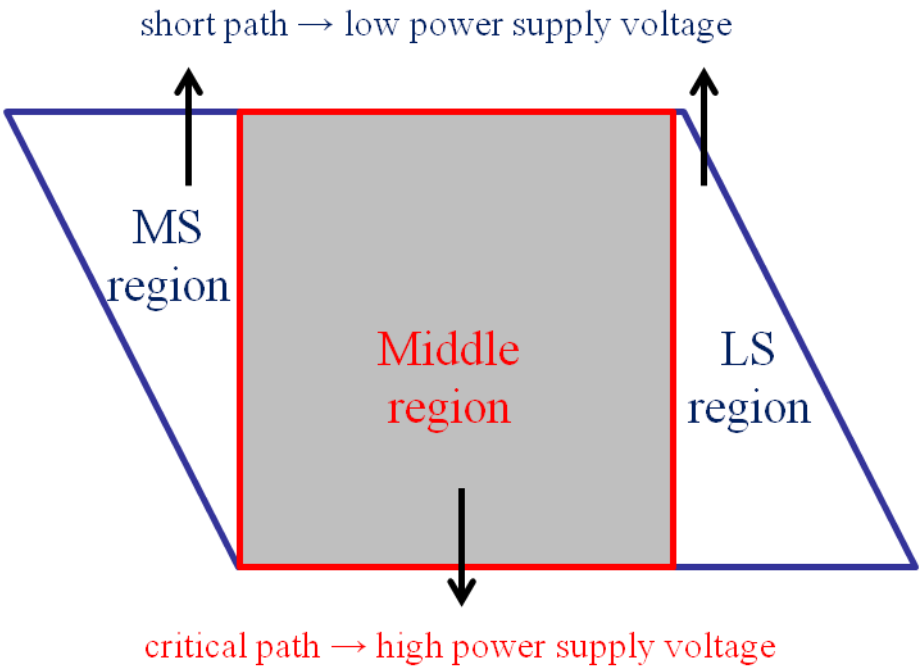


FIGURE 2.11: PARTITION OF THE PPR ARRAY FOR VOLTAGE ISLANDS

In order to get more uniform signal arrival profiles, the PPR array need to be partitioned into more blocks that operate with different power supply voltages. The signal arrival profiles can be finely controlled by more supply voltages. However, the voltage level shifters are needed whenever circuits convert a source of the supply voltage from one voltage to another; thus, more uniform signal arrival profiles might increase area and power dissipation due to additional voltage level shifters.

2.5 Experimental Evaluation

We have implemented 32 × 32-bit LR array multipliers with different structure optimization techniques. The design and simulation methodologies are described in Appendix B. As our major focus is on the PP reduction step, the final CPAs presented in this chapter are not optimized for different input arrival scenarios. Fortunately, the CPAs are the final modules in all multipliers, and the results on PPG and PPR array modules are not affected. Therefore, the non-optimized final CPAs will not significantly affect the relative difference of actual measured values, although they may affect absolute power consumption values. A Carry Skip Adder (CSK) is used for the final CPA in all designs. These adders have good topological regularity and layout simplicity which are considered a good compromise in terms of area and performance. This adder presented in this chapter is not in the final design; thus, we will exploit fast adder design when inputs to the adder will arrive simultaneously in Chapter 3.

2.5.1 Results for Split Array Multipliers

Five schemes with different structure optimization techniques are implemented. For radix-2 multipliers, four schemes are implemented: 1) a LR using the default [3:2] adder (LR_32), 2) a LR using a [4:2] adder (LR_42), 3) a 2-level UL split array multiplier using a [4:2] adder

(LR_42_Split2) and 4) a 4-level UL split array multiplier using a [4:2] adder (LR_42_Split4). For radix-4 array multipliers, only a radix-4 modified Booth 2-level UL split array multiplier using a [4:2] adder (LR_42_Split2_ Radix4) is implemented. Different from the radix-2 LR array multiplier, radix-4 4-level splitting will not be used here because each part accepts 4 PPs and only 1 row using a [4:2] adder in a 32 × 32-bit linear array multiplier, but it can easily be used if a longer fixed-point result is needed.

The comparison results of power, delay and area estimates are shown in Table 2.2. The smallest value of each characteristic is highlighted in boldface. The baseline structure is a LR_32. Compared to a LR_32, a LR_42 achieves 14% less power, 18% less delay and 7% less area, as we expected. This is because the [4:2] adder has a shorter critical path and more regular structures than the [3:2] adder. For the UL split structure, a LR_42_Split2 dissipates 9% less power and 35% less delay with 5% area increase compared to a LR_42. The LR_42_Split4 achieves 10% less power and 26% less delay, but has a 7% area increase compared to the LR_42_Split2. This result well matches our theoretical analysis in Section 2.4.3. The power savings are mainly because the signal propagation paths are shorter. This result implies that we can achieve more power reduction with a slight area increase if each part is split further with larger operand size. The area increase is due to an additional [4:2] stage. The most interesting result is from the LR_42_Split4 which reduces the power by 12% compared to a LR_42_Split2_Radix4. The delay and area of a LR_42_Split4 are close to those of a LR_42_Split2_Radix4. However, the experimental results indicate that the radix-4 Booth recoder has a negative effect on power dissipation, as it is the first stage on the long path in multipliers and thus provides subsequent modules a number of glitches. The LR_42_Split4 has a similar area as the LR_42_Split2_Radix4. These results imply that the increased area in the Booth recoder is roughly the same as the reduced area in the PPR array. Among all schemes, a LR_42_Split4 presents the lowest power consumption and power-delay product results under

experiment. These results indicate that a multi-level split structure is a useful power-saving technique in LR array multipliers.

TABLE 2.2: POWER, DELAY AND AREA FOR LR ARRAY MULTIPLIERS

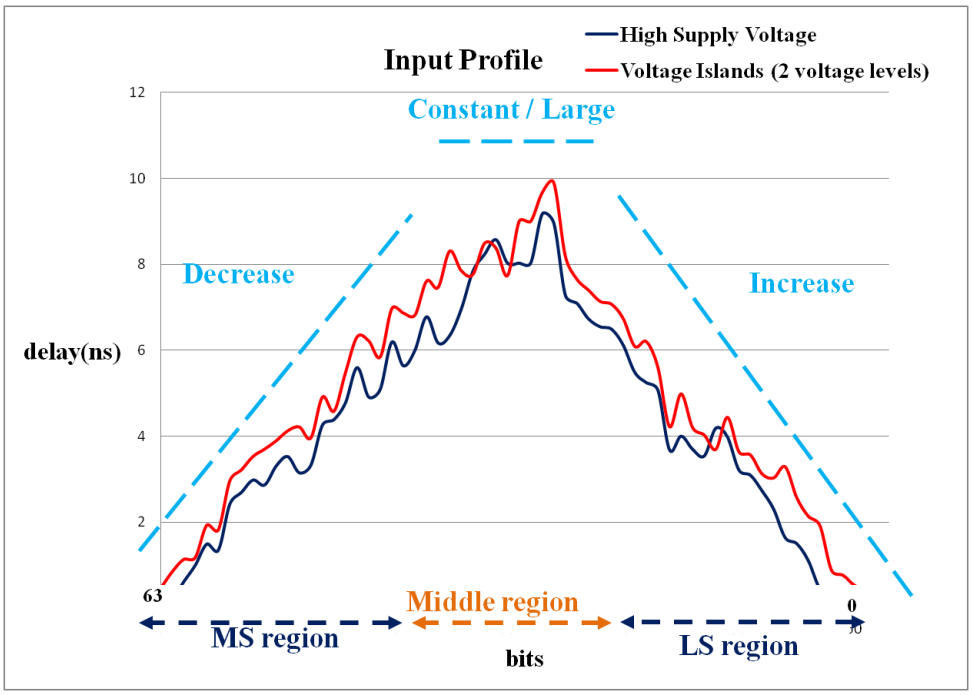
Multiplier	Power (μW)		Delay (ns)		Area (μm^2)		Power-Delay Product (pJ)	
LR_32	6898	1.00	11.20	1.00	13501	1.00	77.26	1.00
LR_42	5932	0.86	9.18	0.82	12559	**0.93**	54.46	0.70
LR_42_Split2	5381	0.78	6.05	0.54	13231	0.98	32.55	0.42
LR_42_Split4	**4829**	**0.70**	4.43	0.40	14177	1.05	**21.39**	**0.27**
LR_42_Split2_Radix4	5450	0.79	**4.38**	**0.39**	13772	1.02	23.87	0.31

2.5.2 Results for Voltage Islands Technique

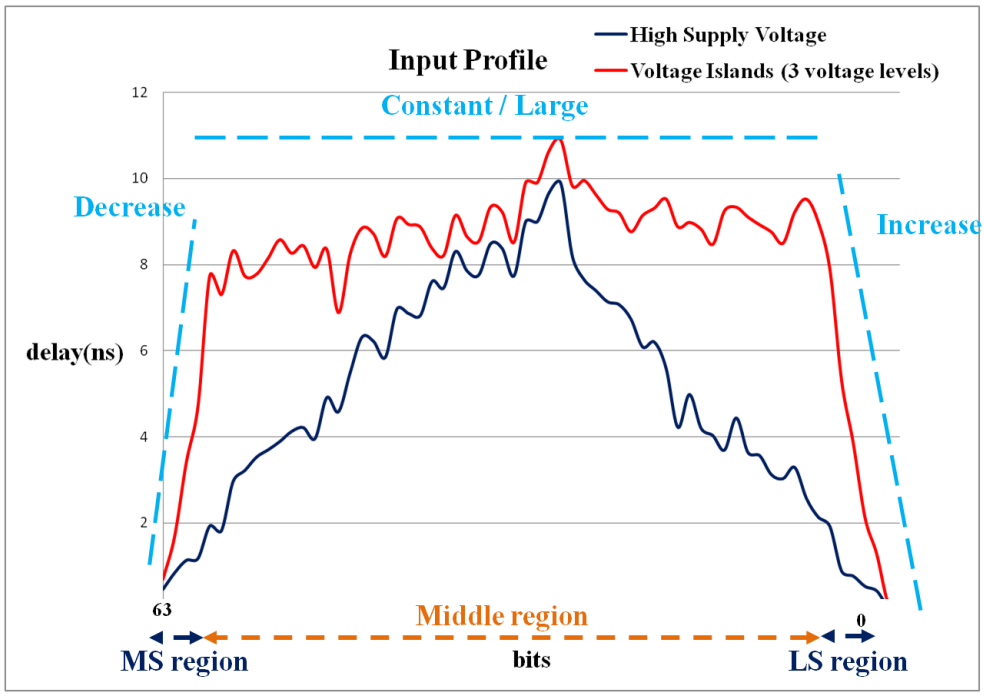
We implemented conventional and the proposed LR linear array multipliers using Verilog and a top-down methodology. The proposed designs were synthesized with two supply voltages 1.20V, 1.32V and three supply voltages 1.08V, 1.20V, 1.32V supported by technology. The voltage level shifters are needed whenever circuits convert a source of the supply voltage from one voltage to another. In recent years, the voltage level shifters can be automatically inserted to support voltage islands by the Synopsys Power Compiler. Figure 2.12 shows the non-uniform arrival time profiles generated by the PPR array with high supply voltage and voltage islands in a 32 × 32-bit multiplier. To partition the PPR array, we analyzed the slopes of arrival time of signals. For the LS region, the delay between two consecutive bits starting at the LSB toward the MSB is linearly increasing. For the middle region, delay is constant and large. For the MS region, the slope of the signal delay profile is negative.

Table 2.3 shows the simulation results for partition in three regions of multipliers. The LS and MS regions may not require higher supply voltage; thus, the power can be significantly reduced in the PPR array. Table 2.4 summarizes the results for the proposed and conventional multipliers. This result includes the power, delay and area of additional implementation for voltage islands. Compared to conventional multipliers, the proposed multipliers dissipate between 20% and 30% less power with between 8% and 11% increase in delay. Furthermore, our designs are better than conventional multipliers in terms of power-delay product.

Unfortunately, the voltage islands technique does not match the split array structure, which has very narrow MS and LS regions compared to non-split array structure, as shown in Figure 2.13. Therefore, in the split array structure, the negative effect due to the extra cost of the power dissipation of level shifters will be more dominating than the positive effect due to power reduction in MS and LS regions. Furthermore, as the operand size increases in the split array multiplier, the relative reduction in power dissipation will decrease because the rate of the middle region of the PPR array which requires a higher supply voltage level increases. Thus, the voltage islands technique cannot be applied efficiently to a split array multiplier, but it is still useful to apply a non-split array multiplier for power reduction. The overall results indicate that the proposed design approach has a good potential for power savings while maintaining the multiplier latency.



(A) TWO VOLTAGE LEVELS



(B) THREE VOLTAGE LEVELS

FIGURE 2.12: INPUT ARRIVAL PROFILES OF THE PPR ARRAY WITH HIGH SUPPLY VOLTAGES AND VOLTAGE ISLANDS IN A 32 × 32-BIT LR ARRAY MULTIPLIER

TABLE 2.3: RESULTS FOR PARTITION

Region	Two Voltage Levels		Three Voltage Levels		Characteristics
	Bit	Rate	Bit	Rate	
LS	0 ~ 27	0.44	0 ~ 8	0.14	Linear increase
Middle	28 ~ 38	0.17	9 ~ 59	0.80	Constant / Large
MS	39 ~ 63	0.39	60 ~ 63	0.06	Rapid decrease

TABLE 2.4: POWER, DELAY AND AREA COMPARISONS OF THE ARRAY IN A NON-SPLIT LR 32 × 32-BIT MULTIPLIER UTILIZING HIGH SUPPLY VOLTAGE AND VOLTAGE ISLANDS

Size		Multiplier	
Power (μW)	Conventional (only 1.32V)	5932	1.30
	Voltage Islands (1.20, 1.32V)	4926	1.08
	Voltage Islands (1.08, 1.20, 1.32V)	**4564**	**1.00**
Delay (ns)	Conventional (only 1.32V)	**9.18**	**1.00**
	Voltage Islands (1.20, 1.32V)	9.90	1.08
	Voltage Islands (1.08, 1.20, 1.32V)	10.19	1.11
Area (μm^2)	Conventional (only 1.32V)	**11881**	**1.00**
	Voltage Islands (1.20, 1.32V)	12288	1.03
	Voltage Islands (1.08, 1.20, 1.32V)	12355	1.04
Power-Delay Product (pJ)	Conventional (only 1.32V)	54.46	1.17
	Voltage Islands (1.20, 1.32V)	48.77	1.05
	Voltage Islands (1.08, 1.20, 1.32V)	**46.50**	**1.00**

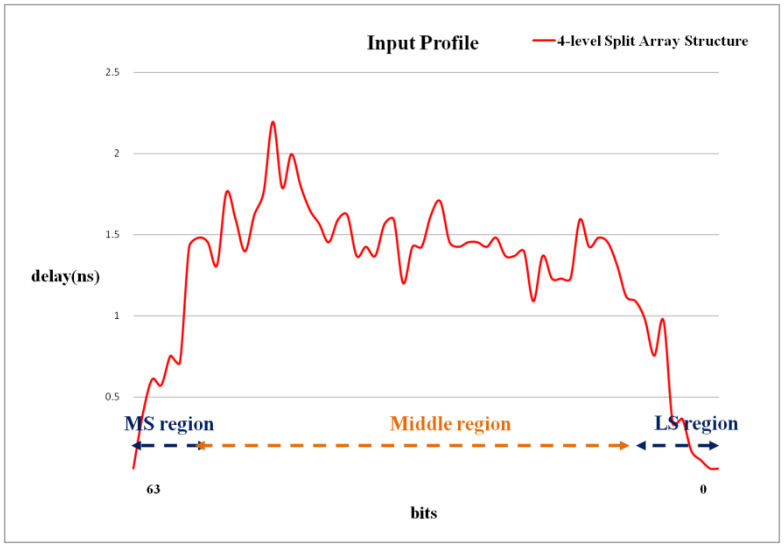


FIGURE 2.13: ADDER INPUT PROFILES OF THE PPR ARRAY IN A 32 × 32-BIT 4-LEVEL UL SPLIT LR ARRAY MULTIPLIER

2.6 Summary

In this chapter, we have proposed a multiplier used for a major component of a low-power sum-of-products in Chapter 4. We have presented several power reduction structure optimization techniques for radix-2 and radix-4 LR linear array multipliers. These techniques include a [4:2] adder for PP reduction, a 4-level UL split structure and voltage islands. A LR_42_Split4 provides a more powerful alternative to the conventional LR_42_Split2 as this structure has the potential of saving power and delay. Detailed experimental results are given to compare the power, delay and area characteristics of each 32 × 32-bit LR multiplication scheme. Among different optimization techniques for LR array multipliers, the 4-level UL split structure is a primary choice if power is the critical concern. The LR_42_Split4 achieves the least power consumption in most cases with relatively small delay. When a small area is the main goal, the simpler LR_42 is a better candidate.

For real-world comparison, we introduce other multipliers used in mobile chips. The ARM core is used to perform real-time digital signal processing in most mobile systems and the ARM7TDMI includes an enhanced 32 × 8 single tree multiplier with a radix-4 modified Booth's algorithm and supports the 64-bit results multiply and MAC instructions. We cannot directly compare the power dissipation and delay of two structures with different structure, due to the difference of features and the problem caused by lack of information. However, the previous studies demonstrate that a 2-level UL LR array multiplier is less power than tree multipliers without delay and area overhead with a maximum of 32-bit [51]. In this chapter, we also find that a 4-level UL split array multiplier using a [4:2] adder has less power consumption and smaller delay than a 2-level counterpart. Therefore, if we replace a tree multiplier in the ARM7 core with the proposed multiplier, a modified ARM processor will consume less power than a conventional one without delay overhead.

In addition, our proposed designs scale well in terms of power reduction and thus use suitably less power when applied to high precision. It probably would gain a similar power reduction in 64 × 64-bit or larger precision. The techniques presented in this chapter can also be applied to other arithmetic units with unbalanced structures.

Chapter 3 Power and Delay Optimization of the Carry-Propagate Adder

In Chapter 2 we have shown that power reduction could come from array structure optimizations at the algorithm and architecture levels. In this chapter we address the problem of adding two carry-save vectors obtained by two PPR arrays. We present a design strategy specific to arrival time profiles generated by two PPR arrays, and propose the power- and delay-optimal final CPA. Finally, we show that our design consume less power over another fast adders with a little delay increase. The proposed adder can achieve performance gains with small power increase when supply voltage is increased.

3.1 Introduction

The sum-of-products unit can be largely divided into three parts: 1) the PPG, 2) the PPR array and 3) the [4:2] adder and the CPA. PPs are created in parallel by the PPG, and the PPR array reduces the number of PPs to be added into two carry-save vectors. The [4:2] adders reduce 4 bit-vectors to 2 bit-vectors, and the final CPA produces the result. The total delay is generally determined by the carry-propagate addition, and thus reducing latency in the final CPA will decrease the total delay of a sum-of-products unit.

We combine structure optimization techniques in Chapter 2 and propose an optimal final CPA. Many studies have focused on reducing a latency of fast adders [40][41]. In this chapter, we focus on reduce the power consumption of fast adders. Lower latency can be achieved using reduced power dissipation when supply voltage is increased. Consequently, we can reduce a delay of the final CPA.

3.2 Problem and Related Work

In Chapter 2, we have found that the 4-level UL LR split array structure and the array structure utilizing voltage islands have the least power consumption with relatively small delay. It is not clear what are good adders and how to design an optimal adder under condition of signal arrival time generated from the proposed reduction structure. This section addresses these problems and related research.

3.2.1 Problem

In previous chapter, we have proposed a multiplier array used for a major component of a low-power sum-of-products. The 4-level UL split array structure with [4:2] adder achieves the least power consumption. The delay of a 4-level UL split array structure is decreased in the middle region while maintaining the delays in the LS and MS regions compared to that of a non-split array structure. Thus, non-uniform input arrival time to the CPA is transformed into mostly uniform input arrival time. This chapter examines the design of the CPA under the condition of uniform (but not perfectly constant) input signals arrival.

3.2.2 Related Work

Previous work on the final CPA focused on the hybrid CPA. The hybrid CPA has been optimized to match the non-uniform input arrival profiles [75][76]. It was obvious that the use of these techniques in multipliers is an efficient in delay. However, the hybrid adder would not be efficient for a sum-of-products unit, when we use a 4-level split array structure or a structure utilizing voltage islands because most input arrival bits to the CPA arrive at the same time.

We consider fast adders under the condition of uniform input signals arrival. One of fast adders is a Carry-SELect Adder (CSELA). The idea of the CSELA is to compute in parallel

two conditional sums: one for a 0-carry and the other for 1-carry, and then select one sum when the carry is available. The basic principle is to divide the adder into groups of *m*-bit and to compute for each group two conditional sums and carry outs. A generic group in which we label the bit from 0 to (*m*-1)-bit is as below.

$$(c^0_m, S^0) = \text{ADD}\ (X, Y, c_0 = 0) \tag{3-1}$$

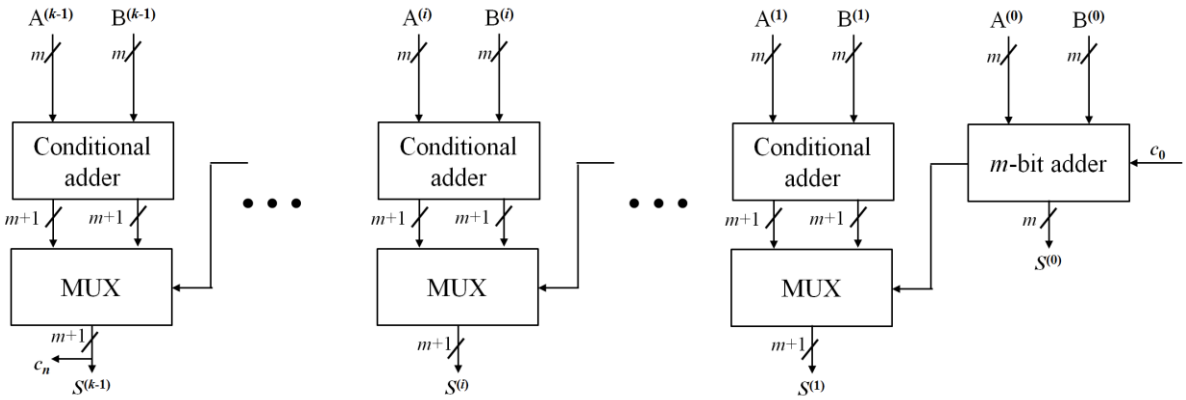
$$(c^1_m, S^1) = \text{ADD}\ (X, Y, c_0 = 1) \tag{3-2}$$

where, *X*, *Y* and *S* are m-bit vectors

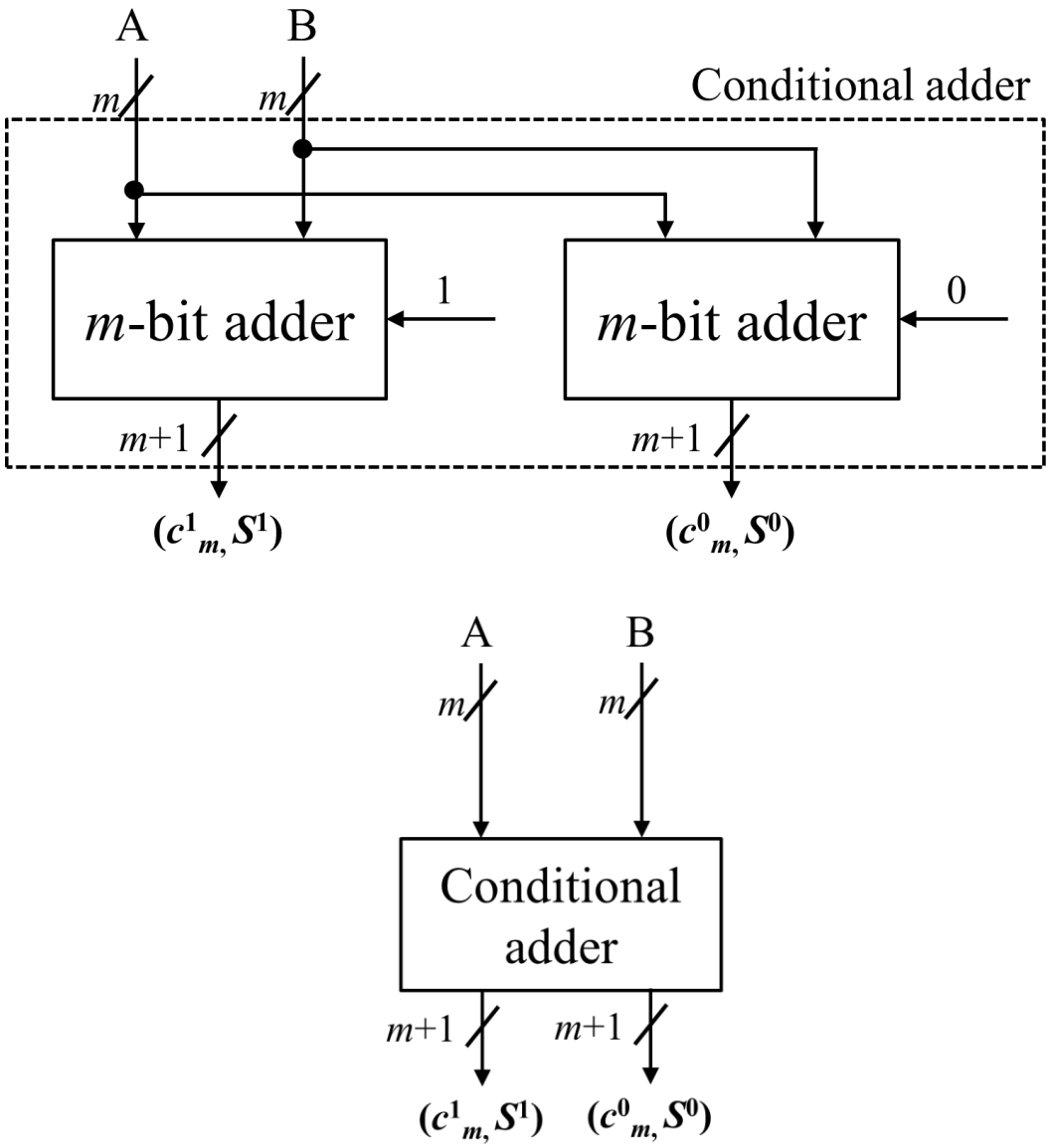
Then, we can select from these two forms when the carry-in of the group is known.

$$(c_m, S) = \begin{cases} (c^0_m, S^0) & \text{if } c_0 = 0 \\ (c^1_m, S^1) & \text{if } c_0 = 1 \end{cases} \tag{3-3}$$

We use a module that has as input the two *m*-bit operands and produces two (*m*+1)-bit result because two *m*-bit adders for the same group can share components, as shown in Figure 3.3 [41].



(A) CARRY-SELECT ADDER

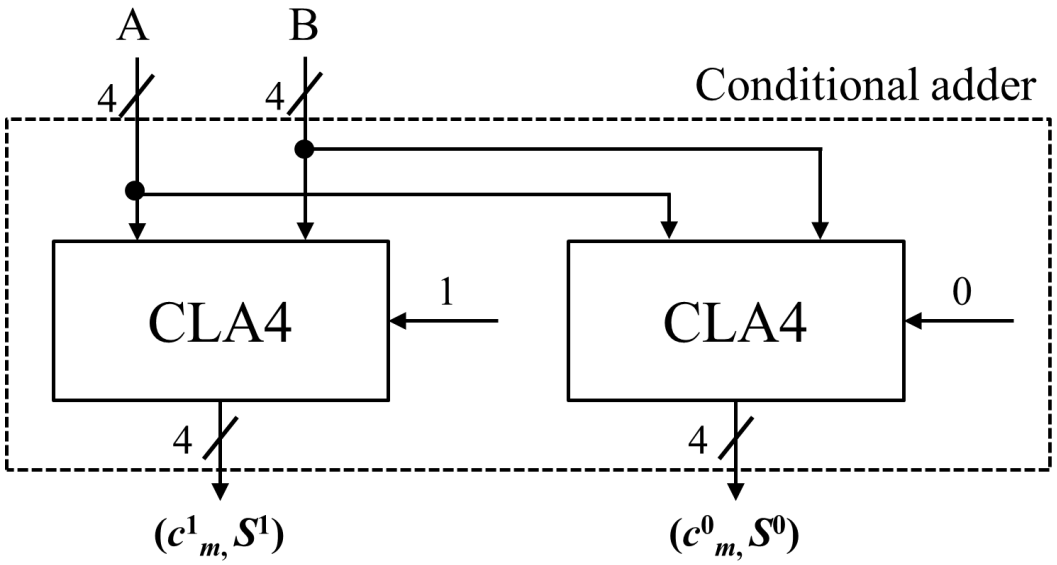


(B) CONDITIONAL ADDER

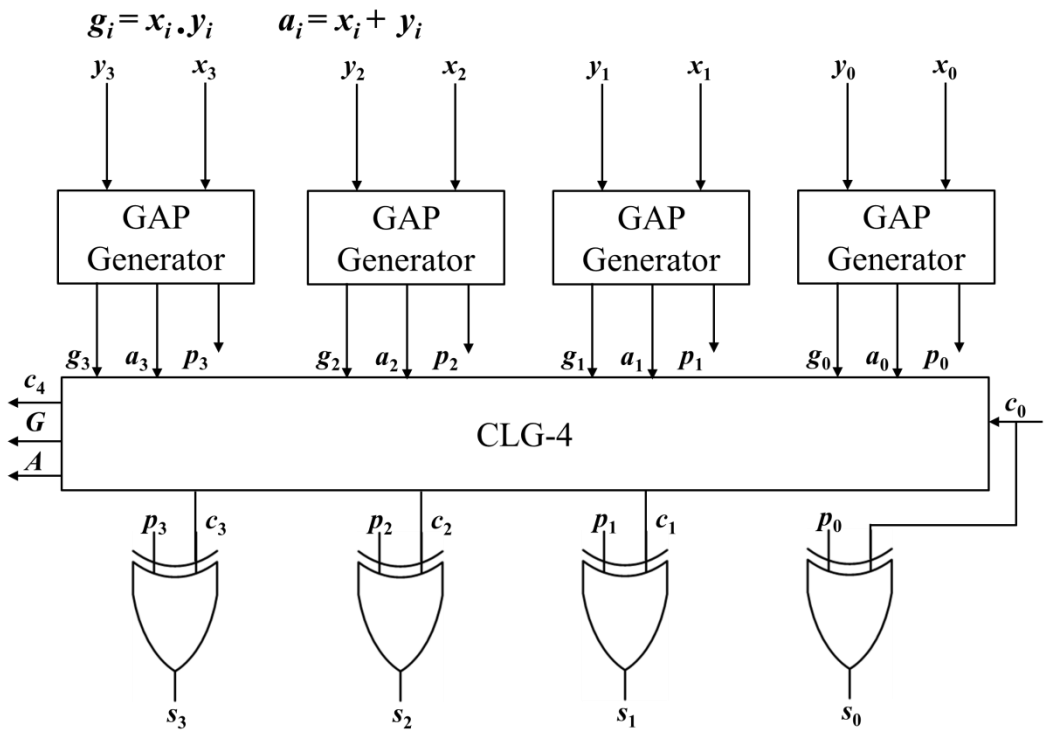
FIGURE 3.1: BLOCK DIAGRAM OF A CARRY-SELECT ADDER (ADAPTED FROM [41])

We can use several types of adders such as CRA, Carry-Lookahead Adder (CLA), the Conditional-Carry Adder (CCA) in a conditional adder. The basic idea of CLA is to compute several carries simultaneously. Ideally, all carries can be computed at the same time, but it is not practical because the implementation has the large number of gates with large number of inputs for large size [41]. The CLA consists of three computation parts: 1) the computation of p_i, g_i and a_i, 2) the computation of the carries in the Carry-Lookahead Generator (CLG) and the computation of sums, as shown in Figure 3.3. In CCA, instead of generating conditional sums, one can obtain only conditional carries using a simple group design, because the sum outputs are worthless for the carry selection [78][79]. To reduce the gates, the MUXs for sum selection are discarded, and only selected carry bits are generated. We know all the carries once controlling carries are determined. One extra XOR level produces the final sums.

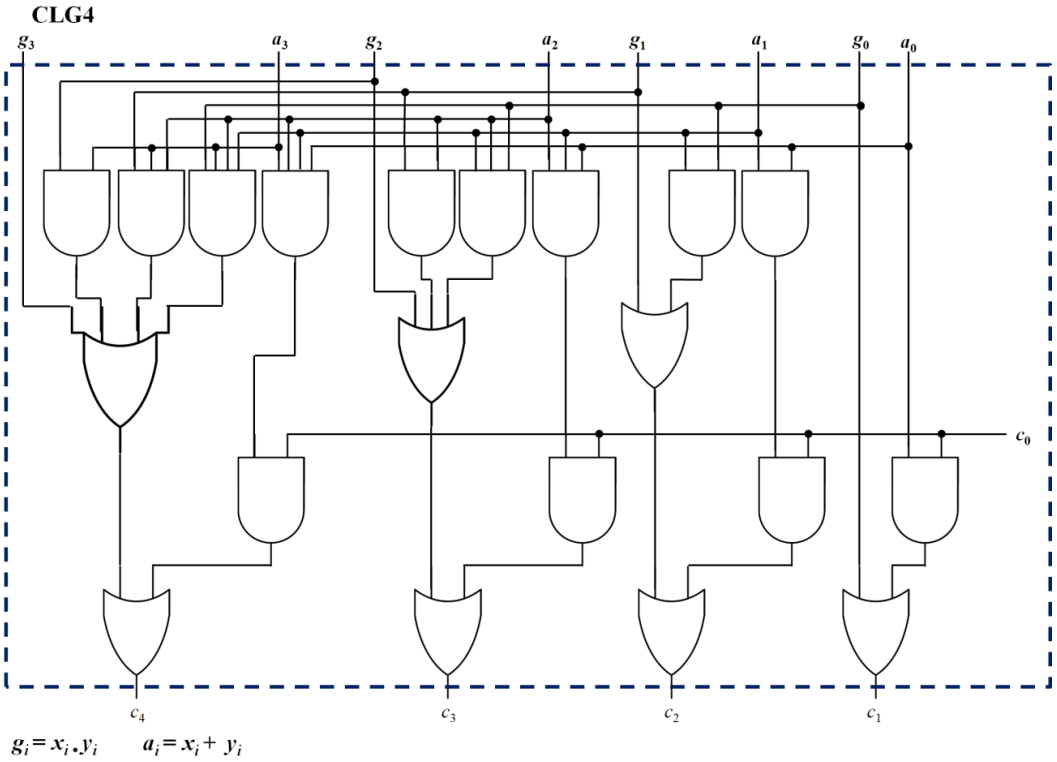
We can use the variable block size of each block. When designing these types of adders, it is important that delays of groups and carry generation are balanced. The idea of varying block sizes can further reduce adder delay.



(A) CONDITIONAL ADDER BASED ON CLA4



(B) Carry-Lookahead Adder module (CLA-4)



(C) 4-BIT CLG GENERATOR (CLG-4)

FIGURE 3.2: BLOCK DIAGRAM OF A CARRY-SELECT ADDER BASED ON CLG4 (ADAPTED FROM [41])

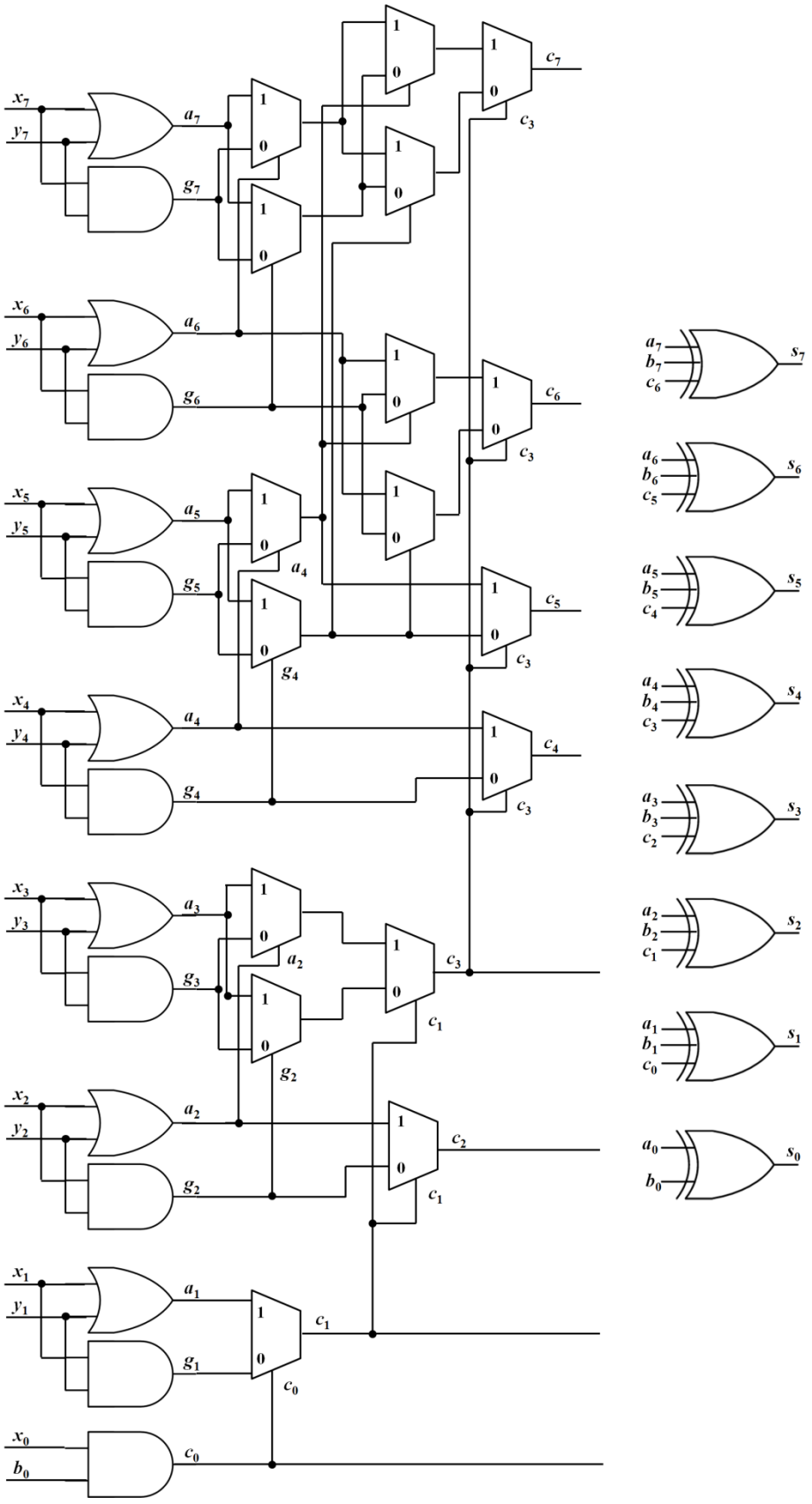


FIGURE 3.3: A CONDITIONAL CARRY ADDER (ADAPTED FROM [79])

3.3 Baseline Design

We design here the fast adder under the assumption of uniform arrivals. We focus on power optimization techniques of fast adders because lower latency of adders can be achieved when supply voltage is increased. The CSELA is one of the fast adders, but the duplicated adder results in high power consumption. However, the recent implementations of the CSELA have been proposed in [81][82][83]. This adder can be implemented by using a single CRA and a MUX-based add-one circuit instead of another CRA with a carry input. The number of connections can be also reduced. We can use several adders as a conditional adder of CSELAs. We first focus on a CRA structure because the input arrival time to the final CPA is not perfectly constant. In a 4-level UL LR split structure, delay is increased from LSB through 7-bit, and in an array structure with voltage islands technique, delay is increased from LSB through 8-bit. In the positive slope region, a CRA would be appropriate because this need not wait for the incoming input from reduction array, while any fast adder would need to wait for higher bits from PPR arrays. Another structure is to use a CLA structure as a conditional adder of a CSELA. The CLA has been widely used in CSELA, due to a fast structure. The CLA calculates several carries simultaneously, and thus reduces the wait time to calculate the result of the larger bits. However, this has a large number of gates with a large number of inputs and long interconnection. Thus we consider here only a simple one-level CLA. The other is to use a CCA as a conditional adder of a CSELA. The CCA is only used for the carry output selections of every bit and the sum bits can be produced by a CSELA.

3.3.1 Preliminaries

To evaluate the effectiveness of our proposed design, we provide a theoretical analysis and conduct experiments. The delay and area evaluation methodology considers a XOR and a MUX

to be made up of basic digital logic gates, shown in Figure 3.4. Based on this approach, the CSELA made up of 2:1 MUXs, HAs, and FAs are evaluated and listed in Table 3.1. The delay of one basic logic gate such as NOR2, NAND2 is $0.5 \times T_{XOR2}$, and a MUX21 gate has the same delay as a XOR2 gate. The area is evaluated by counting the total number of gates required for module. These delay and area characteristics are true in Samsung 65nm standard cell library. Numeric suffixes with gates specify the number of input and output. e.g., AND2 gate means 2-input AND gate, and a MUX21 gate means a 2-input MUX gate with a single output. For theoretical analysis, we assume that 1) basic logic gates such as AND2, OR2, and inverter gates have the same gate delay and 2) positive logic gates such as AND, OR could be optimized to be negative logic NAND, NOR without delay and area penalty, and thus there is no difference between positive and negative logic gates in actual implementation. 3) One additional input for a given gate increases 50% area and 20% delay, and 4) this delay figures exclude sum buffering delays, which depend on the particular application. Figure 3.5 shows two popular FA structures. Figure 3.5(a) is a NAND2-based structure, Figure 3.5(b) and is a MUX-based structure. For the worst-case delays and area, a MUX-based structure is better. Furthermore, a NAND2-based structure has complex interconnection. We will focus on a MUX-based structure here.

NAND2-based structure

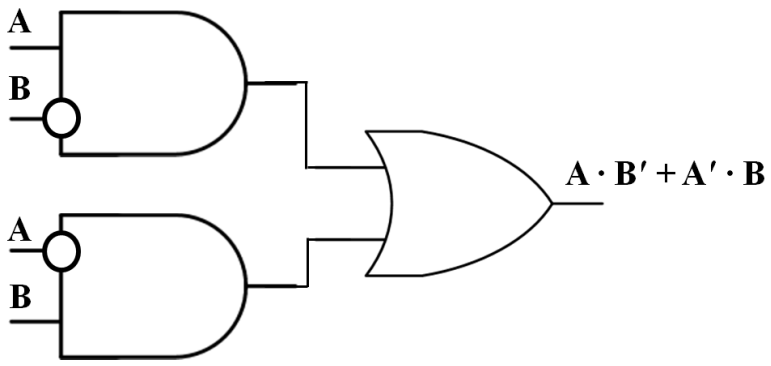
Area: (3 × AND2 + OR3 + 2 ×XOR2) = 3.35 × A_{XOR2}

Delay: 2 × T_{XOR}

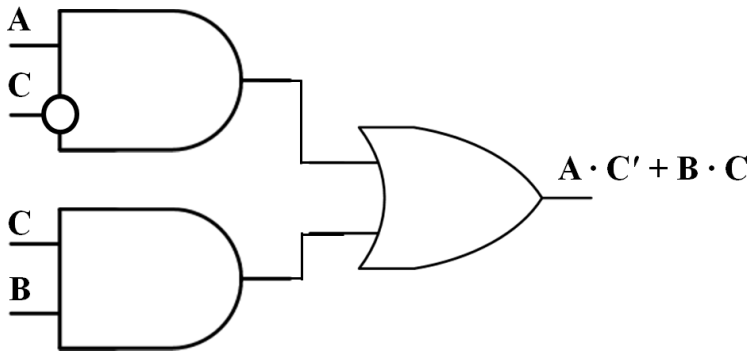
MUX-based structure

Area: (MUX21 + 2 × XOR2) = 3 × A_{XOR2}

Delay: 2 × T_{XOR}



(A) XOR GATE

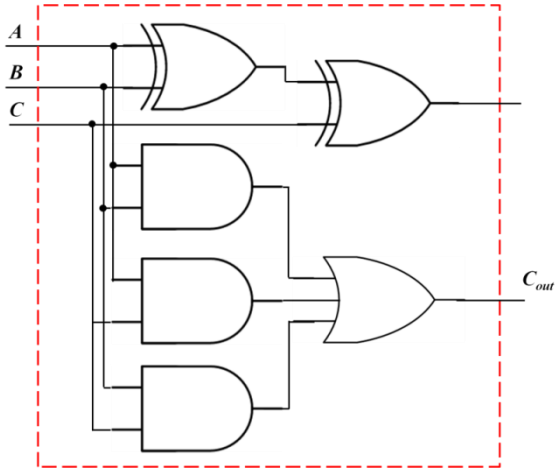


(B) 2:1 MUX GATE

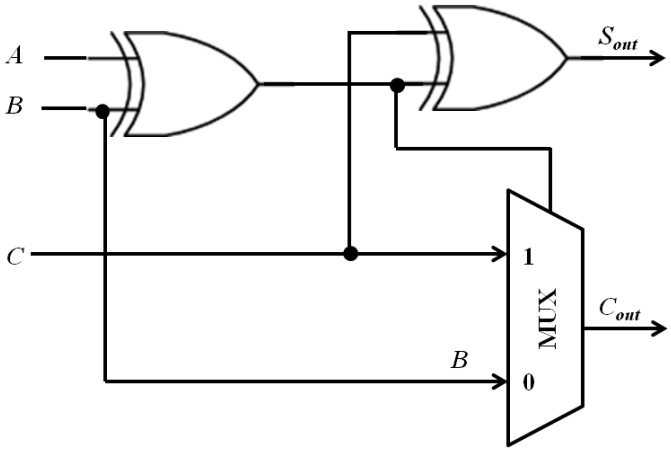
FIGURE 3.4: THE IMPLEMENTATION OF XOR AND MUX

TABLE 3.1: DELAY AND AREA COUNT OF THE BASIC BLOCKS OF CSELA

Module	Delay	Area
XOR2, MUX21	T_{XOR2}	A_{XOR2}
XOR3	$1.2 \times T_{XOR2}$	$1.2 \times A_{XOR2}$
XOR4	$1.4 \times T_{XOR2}$	$1.4 \times A_{XOR2}$
NAND2, NOR2, AND2, OR2	$0.5 \times T_{XOR2}$	$0.3 \times A_{XOR2}$
NAND3, NOR3, AND3, OR3	$0.6 \times T_{XOR2}$	$0.45 \times A_{XOR2}$
NAND4, NOR4, AND4, OR4	$0.7 \times T_{XOR2}$	$0.6 \times A_{XOR2}$
INV	$0.3 \times T_{XOR2}$	$0.3 \times A_{XOR2}$
HA	T_{XOR2}	$1.3 \times A_{XOR2}$
FA	$2 \times T_{XOR2}$	$3 \times A_{XOR2}$



(A) NAND2-BASED STRUCTURE



(B) MUX-BASED STRUCTURE

FIGURE 3.5: DESIGNS OF A FA

3.3.2 Basic Schemes and Architecture of the CSELA

In this section we follow the idea of the modified CSELA with an add-one circuit. Previous studies have mainly focused on area and power optimizations and have not considered well input signal arrival characteristics. Thus we develop an improved version for the modified CSELA design for corresponding 4-level UL LR array structure. The structure of the 64-bit conventional CSELA is shown in Figure 3.6. The numbers within gates ( ) and wires [ ] specify gate delay and signal arrival times, respectively, assuming a unit delay model. e.g., a FA requires 2 unit delays. The delay of the longest path is the sum of all the gate delays. Because carry-in is known at the beginning of computation, a carry-select block is not needed for the first group1. The delay evaluation of group2 is shown in Figure 3.7. The one set of an 8-bit CRA with C_{in} = 0 has 7 FAs and a 1 HA, and the other set of an 8-bit CRA with C_{in} = 1 has 7 FAs and a 1 modified HA, as shown in Figure 3.8. The arrival time of data outputs from a CRA with C_{in} = 1 is the same as the arrival time of data output from a CRA with C_{in} = 0. Based on the consideration of delay values of Table 3.1, the arrival time of data output ($s_0(t) = 2 \times T_{XOR} \sim s_7(t) = 14 \times T_{XOR}$) from a CRA with C_{in} = 0 and a CRA with C_{in} = 1 is earlier than or equal to the arrival time of selection input ($C_{in}(t) = 15 \times T_{XOR}$) of a MUX. Thus, the final sum output from s_0 to s_7 in group2 is summation ($s_0(t) \sim s_7(t) = 16 \times T_{XOR}$) of the arrival time of MUX selection input ($C_{in}(t) = 15 \times T_{XOR}$) and MUX gate delay ($T_{MUX} = 1 \times T_{XOR}$). In order to avoid waiting for the data input, we adjust the group size from group3 to group8, and hence the arrival time of MUX selection input is always greater than or equal to the arrival time of data outputs from CRAs. The total delay of group3 to group8 is estimated as the sum of the arrival time of MUX selection input and MUX gate delay.

Based on the gate count of Table 3.1, the area of group2 is determined as follows:

Group 2 → (8-bit CRA with C_{in} = 0) + (8-bit CRA with C_{in} = 1) + (18:9 MUX) = 53.6 × A_{XOR2}

8-bit CRA with C_{in} = 0 → 7 × FA + HA = 22.3 × A_{XOR2}

8-bit CRA with C_{in} = 1 → 7 × FA + HA = 22.3 × A_{XOR2}

18:9 MUX → 9 × 2:1 MUX = 9 × A_{XOR2}

Similarly, the delay and area of the other groups in the conventional CSELA are estimated.

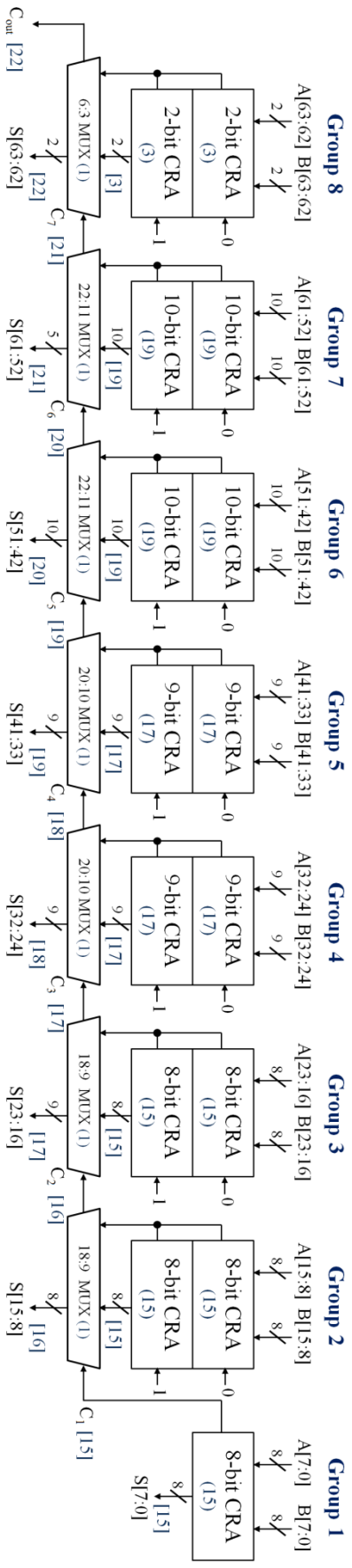


FIGURE 3.6: A CONVENTIONAL CSELA WITH VARIABLE BLOCK SIZE

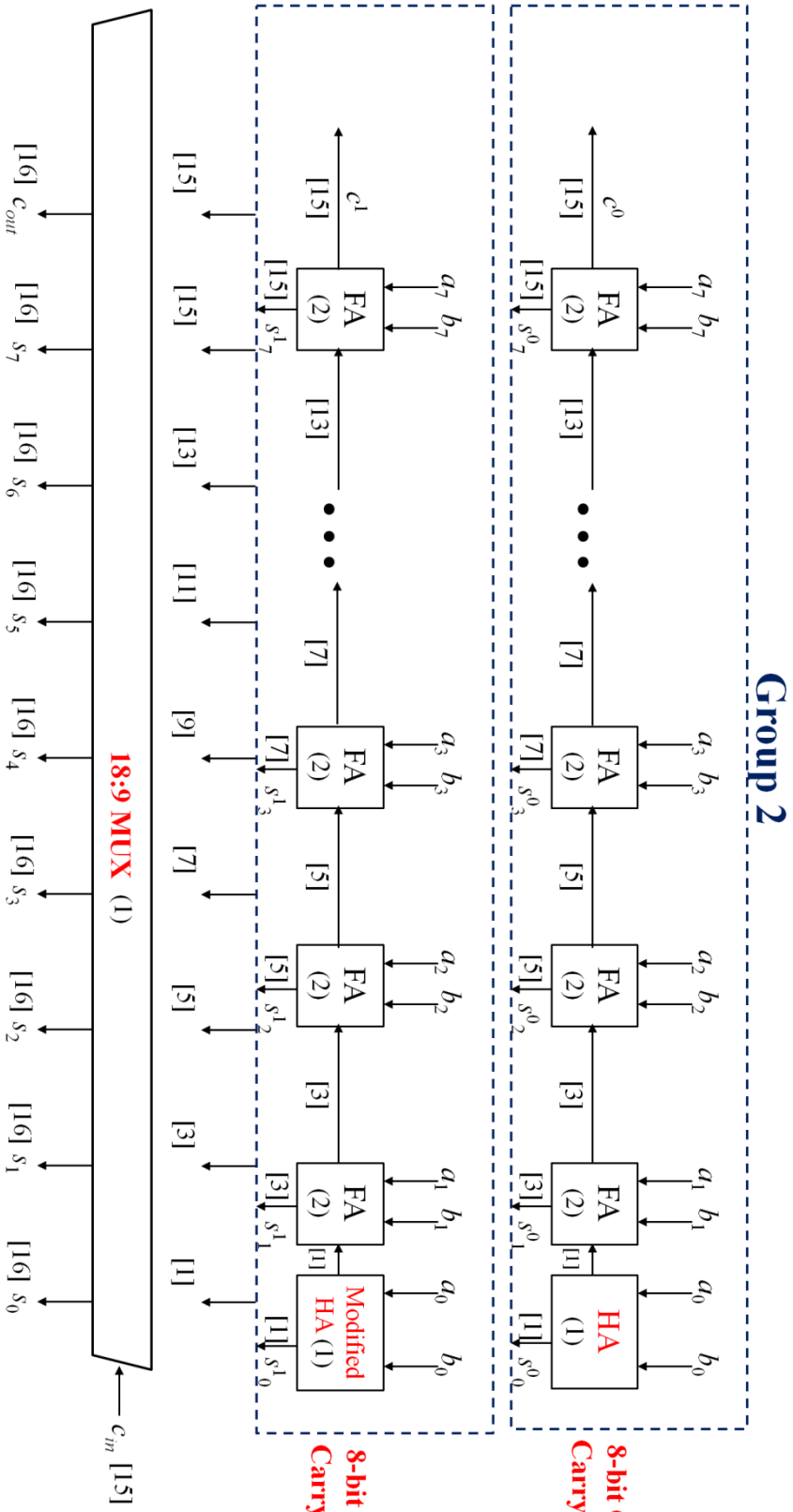


FIGURE 3.7: DELAY EVALUATION OF CONVENTIONAL CSELA (GROUP2)

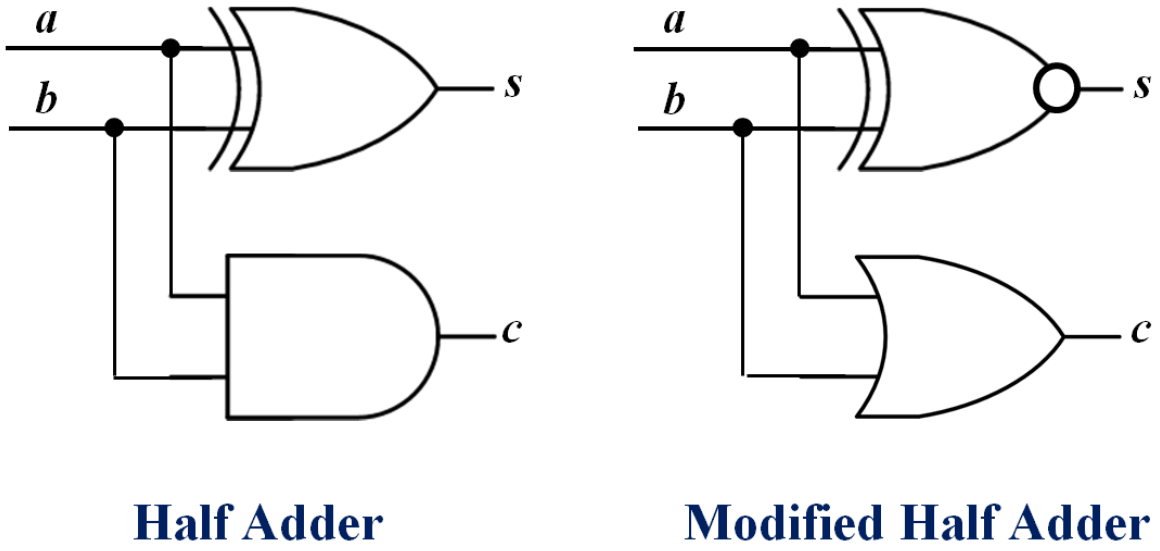


FIGURE 3.8: A HALF ADDER AND A MODIFIED HALF ADDER

3.4 The Proposed Design

In this section, we show how to construct the final adder for uniform signal arrival profiles derived from the proposed array structure.

3.4.1 Modified Schemes and Architecture of the CSELA

To reduce area and power dissipation of a conventional CSELA, the n + 1-bit add-one circuit is required to replace the n-bit adder with C_{in} = 1. The Boolean expression of a 4-bit add-one circuit is shown as below.

$$s^1_0 = \sim s^0_0$$
$$s^1_1 = s^0_1 \oplus s^0_0$$
$$s^1_2 = s^0_2 \oplus (s^0_1 \bullet s^0_0)$$
$$c^1 = c^0 \oplus (s^0_2 \bullet s^0_1 \bullet s^0_0)$$

As shown in Figure 3.9, a 4-bit add-one circuit can be implemented directly from Boolean expression.

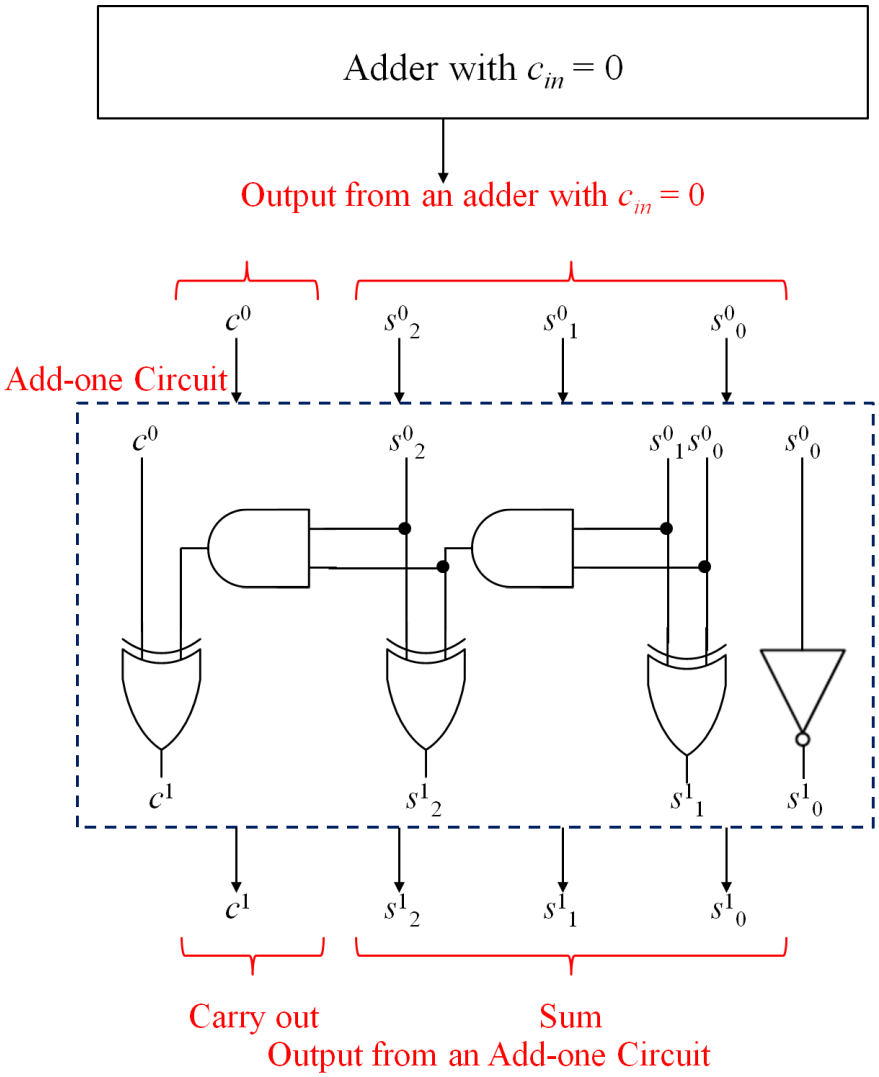


FIGURE 3.9: THE IMPLEMENTATION OF A 4-BIT ADD-ONE CIRCUIT (ADAPTED FROM [83])

Figure 3.10 illustrates CSELA operation using an add-one circuit. This module produces two partial results and the MUX is used to select either an adder with $c_{in} = 0$ or an add-one circuit output according to the control signal, c_{in}. In case of the multiplexer the output of the *n*-bit adder with $c_{in} = 0$ is chosen when 0 is asserted at c_{in} and the output of an add-one circuit, which is the same as the output of an adder with $c_{in} = 1$ is chosen, with the assertion of 1 at c_{in}.

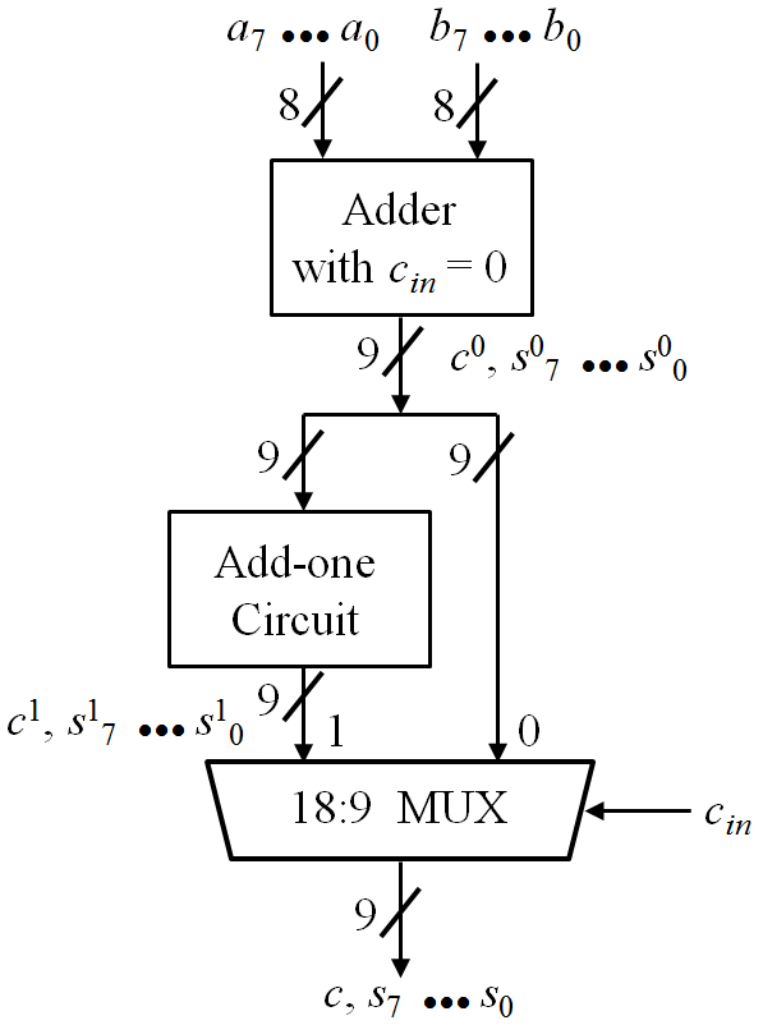


FIGURE 3.10: BLOCK DIAGRAM OF THE COMPONENTS WITHIN THE PROPOSED CSELA REQUIRED FOR MUX OPERATION

The structure of the modified CSELA using a CRA and an add-one circuit is shown in Figure 3.11. This adder has also 8 groups of different size of a pair of CRAs and add-one circuits. The delay evaluation of group2 is shown in Figure 3.12. The one set of an 8-bit CRA has 7 FAs and a 1 HA for no carry input, and the other set of a 9-bit add-one circuit has 9 × XOR2, 8 × AND2 and 1 × INV gates for carry input. The arrival time of data outputs from an add-one circuit is 1.5 × T_{XOR} greater than the arrival time of data output from a CRA with $c_{in} = 0$.

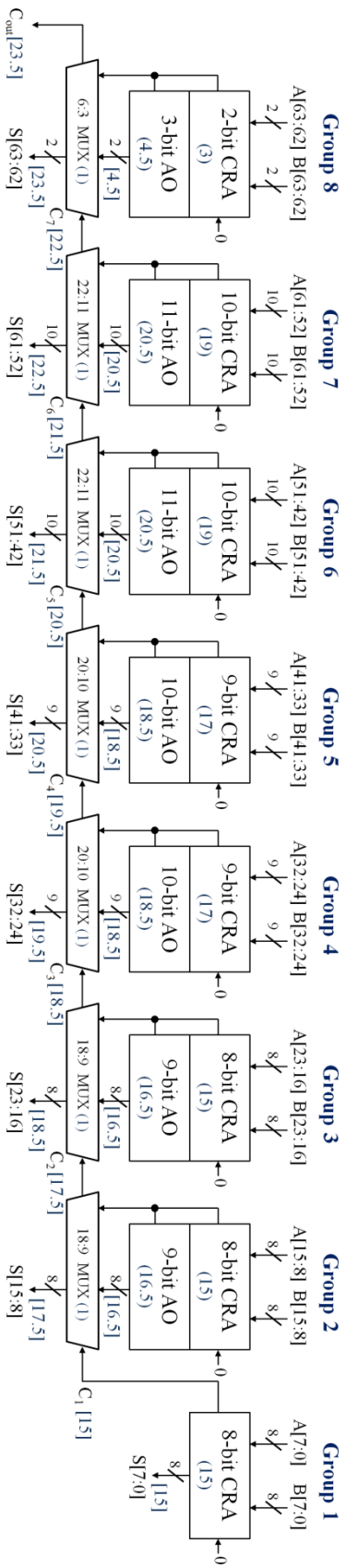


FIGURE 3.11: THE MODIFIED CSELA USING CRA AND AN ADD-ONE CIRCUITWITH VARIABLE BLOCK SIZE (BLOCK SIZES OF 2−10−10−9−9−8−8−8)

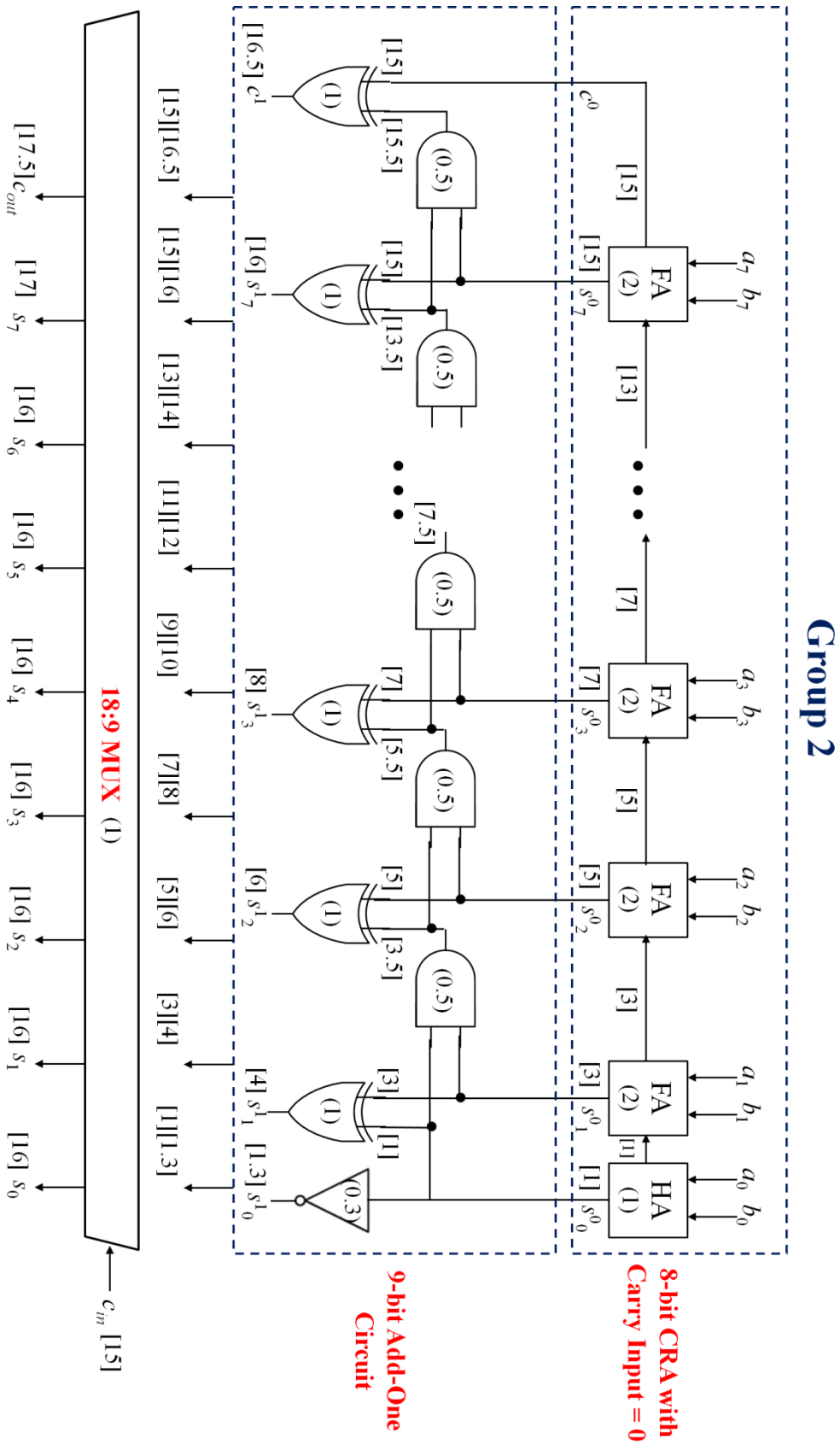


FIGURE 3.12: DELAY EVALUATION OF THE MODIFIED CSELA USING CRA AND AN ADD-ONE CIRCUIT (GROUP2)

The estimated delay of this adder is evaluated. The arrival time of data output ($s^1_0(t) = 1.3 \times T_{XOR} \sim s^1_6(t) = 14 \times T_{XOR}$) from an add-one circuit is earlier than the arrival time of selection input ($c_{in}(t) = 15 \times T_{XOR}$) of a MUX. Thus, the final sum output from s_0 to s_6 in group2 is summation ($s_0(t) \sim s_6(t) = 16$) of the arrival time of MUX selection input ($c_{in}(t) = 15 \times T_{XOR}$) and MUX gate delay ($T_{MUX} = 1 \times T_{XOR}$). However, for the highest sum output (s_7) and carry out (c_{out}) in group2, the arrival time of data outputs from an add-one circuit ($s^1_7(t) = 16 \times T_{XOR}$, $c^1(t) = 16.5 \times T_{XOR}$) is later than the arrival time of MUX selection input ($c_{in}(t) = 15 \times T_{XOR}$). Thus, the delays are the sum ($s_7(t) = 17 \times T_{XOR}$, $c_{out}(t) = 17.5 \times T_{XOR}$) of the arrival time of data outputs from an add-one circuit ($s^1_7(t) = 16 \times T_{XOR}$, $c^1(t) = 16.5 \times T_{XOR}$) and MUX gate delay ($T_{MUX} = 1 \times T_{XOR}$). For the remaining groups, the arrival time of MUX selection input is always greater than the arrival time of data inputs from an add-one circuit. Thus, the delay depends on the arrival time of MUX selection input and its gate delay.

Based on the gate count of Table 3.1, the total number of gates in group2 is determined as follows:

Group 2 → (8-bit CRA with C_{in}= 0) + (9-bit add-one circuit) + (18:9 MUX)= 43 × A_{XOR2}

8-bit CRA with C_{in} = 0 → 7 × FA + HA = 22.3 × A_{XOR2}

9-bit add-one circuit → 9 × XOR2 + 8 × AND2 + INV = 11.7 × A_{XOR}

18:9 MUX → 9 × 2:1 MUX = 9 × A_{XOR2}

Similarly, the delay and area of the other groups are estimated.

As shown in Figure 3.13, in the structure of the modified CSELA using a CCA and an add-one circuit, the arrival time of data output ($s^1_0(t) = 1.3 \times T_{XOR} \sim s^1_2(t) = 3.5 \times T_{XOR}$) from an add-one circuit is earlier than the arrival time of selection input ($c_{in}(t) = 4.3 \times T_{XOR}$) of a MUX. Thus, the final sum output from s_0 to s_2 in group2 is summation ($s_0(t) \sim s_2(t) = 5.3$) of the arrival time of MUX selection input ($c_{in}(t) = 4.3 \times T_{XOR}$) and MUX gate delay ($T_{MUX} = 1 \times T_{XOR}$). However, for the higher sum output (from s_3 to s_7) and carry out (c_{out}) in group2, the arrival time of data outputs from an add-one circuit ($s^1_3(t) = 4.5 \times T_{XOR}, \cdots, s^1_7(t) = 7.5 \times T_{XOR}$, $c^1(t) = 8 \times T_{XOR}$) is later than the arrival time of MUX selection input ($c_{in}(t) = 4.3 \times T_{XOR}$). Thus, the delays are the sum ($s_7(t) = 8.5 \times T_{XOR}$, $c_{out}(t) = 9 \times T_{XOR}$) of the arrival time of data outputs from an add-one circuit ($s^1_7(t) = 7.5 \times T_{XOR}$, $c^1(t) = 8 \times T_{XOR}$) and MUX gate delay ($T_{MUX} = 1 \times T_{XOR}$). For the remaining groups, the arrival time of MUX selection input is always greater than the arrival time of data inputs from an add-one circuit. Thus, the delay depends on the arrival time of MUX selection input and its gate delay.

Based on the gate count of Table 3.1, the total number of gates in group 2 is determined as follows:

Group 2 → (8-bit CCA with $c_{in} = 0$) + (9-bit add-one circuit) + (18:9 MUX)= $51.6 \times A_{XOR2}$

8-bit CCA with $c_{in} = 0$ → = $30.9 \times A_{XOR2}$

9-bit add-one circuit → 9 × XOR2 + 8 × AND2 + INV = $11.7 \times A_{XOR}$

18:9 MUX → 9 × 2:1 MUX = $9 \times A_{XOR2}$

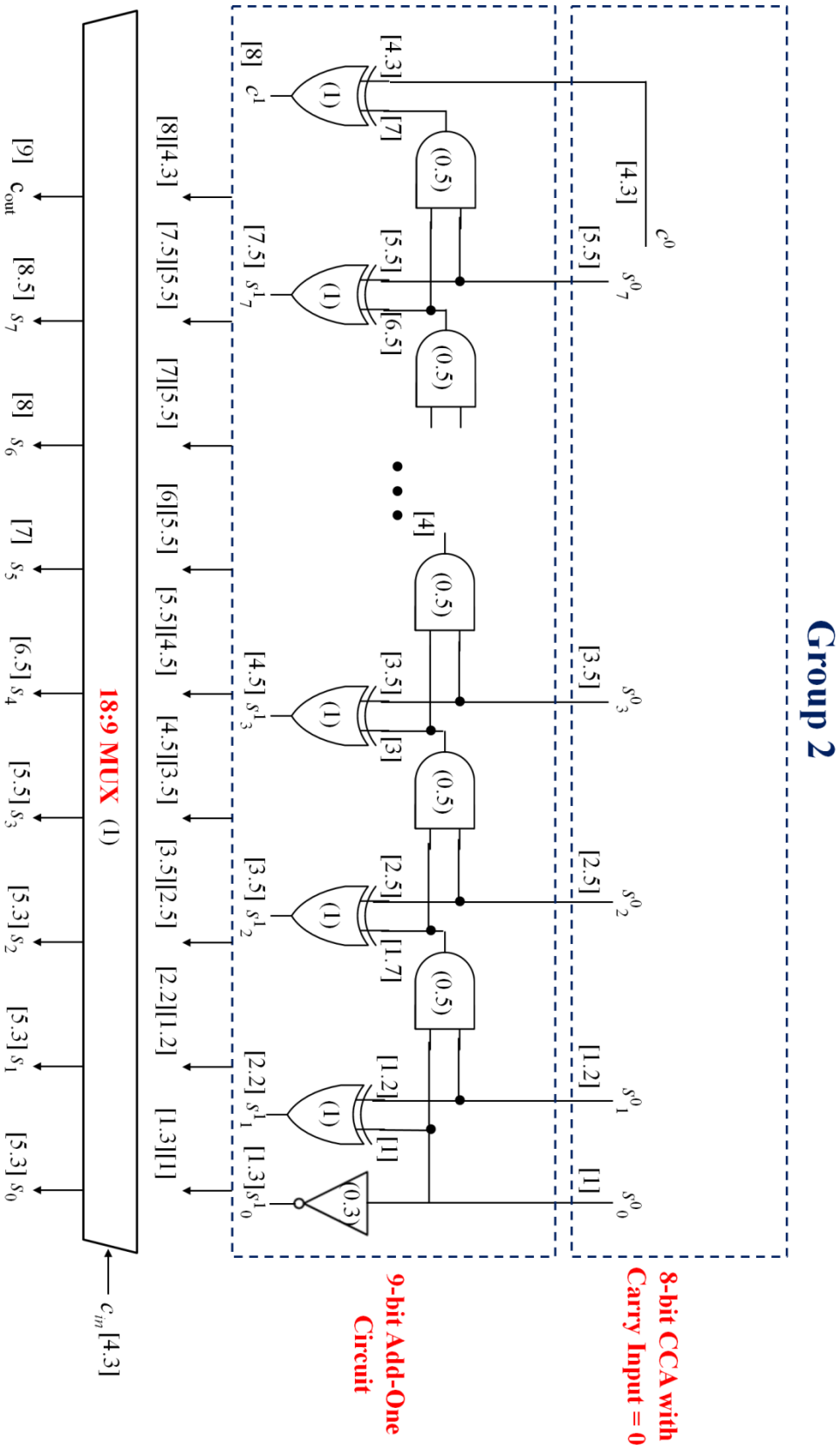


FIGURE 3.13: DELAY EVALUATION OF THE MODIFIED CSELA USING CCA AND AN ADD-ONE CIRCUIT (GROUP2)

The estimated delay and area of the all groups of the conventional and modified CSELAs are evaluated and listed in Table 3.2. The delay and area numbers are accumulated value - e.g., the area of group 2 is summation of group 1 and 2 areas. Theoretically, as to delay, the Modified CSELA based on a CLA is the best followed by the Modified CSELA based on a CCA. In terms of area, the Modified CSELA based on a CRA is the best followed by the Modified CSELA based on a CLA. It is clear that the modified CSELA based on a CRA has less area than a conventional one. The power dissipation can be also reduced, as it is proportional to the amount of hardware used to implement the design. The modified CSELA based on a CLA has less area and delay and the modified CSELA based on CCA has less delay with slight delay increase.

Table 3.2: Delay and Area of Conventional and Modified CSELAs

Group	Conventional CSELA				Modified CSELA (CRA)			
	Delay		Area		Delay		Area	
Group 1	15	1.00	22.3	1.00	15	1.00	22.3	1.00
Group 2	16	1.00	75.9	1.00	17.5	1.10	65.3	0.86
Group 3	17	1.00	127.8	1.00	18.5	1.09	108.3	0.85
Group 4	18	1.00	188.4	1.00	19.5	1.08	156.6	0.83
Group 5	19	1.00	249.0	1.00	20.5	1.08	204.9	0.82
Group 6	20	1.00	316.6	1.00	21.5	1.08	258.5	0.82
Group 7	21	1.00	384.2	1.00	22.5	1.07	312.1	0.81
Group 8	22	1.00	395.8	1.00	23.5	1.06	323.3	0.81

Group	Modified CSELA (CLA)				Modified CSELA (CCA)			
	Delay		Area		Delay		Area	
Group 1	4.2	0.28	25.7	1.15	5.5	0.37	30.9	1.39
Group 2	6.7	0.40	73.9	0.97	9	0.56	91.5	1.21
Group 3	7.7	0.45	125.6	0.98	10	0.59	164.1	1.28
Group 4	8.7	0.48	180.8	0.96	11	0.61	232.2	1.23
Group 5	9.7	0.51	239.5	0.96	12	0.63	305.8	1.23
Group 6	10.7	0.54	301.7	0.95	13	0.65	384.9	1.22
Group 7	11.7	0.56	353.4	0.92	14	0.67	455.5	1.19

3.4.2 Optimal Group Distribution

Another problem is to find the optimal block sizes which minimize the total worst case delay of an adder for a corresponding reduction array. Total delay can be reduced by dividing the adder into variable sized blocks that balance the delay of inputs to the carry chain. In a 4-level UL LR split structure, delay is increased from LSB through 7-bit. Considering the structure of CLG and CCA, 2^n-bit size is efficient to design an 64-bit adder, thus we use an 8-bit adder as a basic block in group 1. Considering additional MUX delay, the adder with optimal variable size can be created when the input delay through the adder with $c_{in} = 0$ is less than or equal to the delay of the previous stage carry because the final adder need not the wait for input from PPR array. Using this delay calculation, two types of adders with variable block sizes can be created because the size of the last block is less than one of basic block size. If we combine the last two groups into one single group, it would make the computation time slower because the size of the last module is longer than those of other group. The CSELA based on CRA with block size of 2−10−10−9−9−8−8−8 is called MCSELA_10_2_CRA and block size of 12−10−9−9−8−8−8 is called MCSELA_12_CRA.

3.4.3 The Structure Optimization

In modified CSELA based on CRA, a CRA is slow because each FA must wait for the carry bit to be calculated by the previous FA. For example, the 12-bit last group of the MCSELA_12_CRA can only start operation until the output from the 12-bit CRA is ready because the arrival time of data output ($s^1(t) = 24.5 \times T_{XOR}$) from CRAs is greater than the arrival time of selection input ($c_{in}(t) = 21.5 \times T_{XOR}$) of a MUX. Therefore, we use a fast CLA instead of a slow CRA as the last group of the MCSELA_12_CRA. The CLA can avoid the wait time to calculate the final result. The proposed final adder is comprised of one group of CLA and six

groups of CRA. We consider here only a simple one-level CLA, and divide the CLA into several small groups. Thus, for less complex implementation the 12-bit input vectors are divided into 3 groups of 4-bit or 6 groups of 2-bit and the groups are connected as in a CRA. These schemes are called MCSELA_12_CLA4 and MCSELA_12_CLA2, respectively.

Based on the consideration of delay values of Table 3.1, the arrival time of data output (4-bit: $s^1_0(t) = 1.3 \times T_{XOR} \sim c_{out}(t) = 8.9 \times T_{XOR,}$ 2-bit: $s^1_0(t) = 1.3 \times T_{XOR} \sim c_{out}(t) = 9.5 \times T_{XOR}$) from two CLAs based on 4-bit and 2-bit is earlier than the arrival time of selection input ($c_{in}(t) = 21.5 \times T_{XOR}$) of a MUX. Thus, the final sum output in group7 is summation ($s_0(t) \sim s_6(t) = 22.5 \times T_{XOR}$) of the arrival time of MUX selection input ($c_{in}(t) = 21.5 \times T_{XOR}$) and MUX gate delay ($T_{MUX} = 1 \times T_{XOR}$), and two adders have the same delay. Comparing the proposed adders, these adders have approximately 12% and 4% faster than the MCSELA_12_CRA and the MCSELA_10-2_CRA, respectively.

Based on the gate count of Table 3.1, the total number of gates in group7 is determined as follows:

Group 7 → (12-bit one-level CLA with each 4-bit group) + (13-bit add-one circuit) + (26:13 MUX) = 61.4 × A_{XOR2}

12-bit one-level CLA with each 4-bit group→ 3 × (8 × AND2 + 5 × OR2 + 3 × AND3 + OR3 + 2 × AND4 + 1 × OR4 + 4 × XOR2) = 31.5 × A_{XOR2}

13-bit add-one circuit → 13 × XOR2 + 12 × AND2 + INV = 16.9 × A_{XOR2}

26:13 MUX → 13 × A_{XOR2}

Group 7 → (12-bit one-level CLA with each 2-bit group) + (13-bit add-one circuit) + (26:13 MUX) = 54.5 × A_{XOR2}

12-bit one-level CLA with each 2-bit group→ 6 × (4 × AND2 + 3 × OR2 + 2 × XOR2) = 24.6 × A_{XOR}

13-bit add-one circuit → 13 × XOR2 + 12 × AND2 + INV = 16.9 × A_{XOR2}

26:13 MUX → 13 × A_{XOR2}

Two adders have the same delay while the CLA based on 2-bit group is smaller than 4-bit group because it is derived based on a simpler arithmetic expression. MCSELA_CLA4 and MCSELA_CCA have internal fast block, and thus it is not necessary to replace with fast adders. Table 3.3 shows delay and area estimates. MCSELA_CLA4 is the fastest followed by MCSELA_CCA. However, they are not optimized for non-uniform LS region, and thus would probably increase delay. In terms of area, the MCSELA_12_CLA2 is the best because this adder uses less XOR2, which are relatively large gates, than the other adders. MCSELA_CCA is the largest because this adder uses many MUX21 gates, which are relatively larger in size. Because smaller area usually leads to less switching capacitance, the results could provide a rough estimation of relative power consumptions in different schemes. Thus the MCSELA_12_CLA2 would consume the smallest power, due to the lowest gate count.

TABLE 3.3: DELAY AND AREA COMPARISONS OF MODIFIED CSELASWITH VARIABLE BLOCK SIZES FOR A 4-LEVEL UL LR STRUCTURE

Adder	Delay (ns)		Area (NAND2)	
Conventional CSELA (2−10−10−9−9−8−8−8)	23.0 × T_{XOR}	1.00	409.4	1.00
MCSELA_10_2_CRA (2−10−10−9−9−8−8−8_CRA)	23.5 × T_{XOR}	1.02	323.3	0.79
MCSELA_12_CRA (12−10−9−9−8−8−8_CRA)	25.5 × T_{XOR}	1.11	323.7	0.79
MCSELA_12_CRA_CLA4 (12−10−9−9−8−8−8_CLA4)	22.5 × T_{XOR}	0.98	323.9	0.79
MCSELA_12_ CRA_CLA2 (12−10−9−9−8−8−8_CLA2)	22.5 × T_{XOR}	0.98	314.0	0.76
MCSELA_CLA4 (10−11−10−9−8−8−8_CLA4)	11.7 × T_{XOR}	0.51	353.4	0.86
MCSELA_CCA (10−11−10−9−8−8−8_CCA)	14 × T_{XOR}	0.61	455.5	1.11

Similarly, the delay and area of the adder for corresponding array structure with voltage islands are estimated. The delay and area estimates are shown in Table 3.4. In terms of delay, the MCSELA_CLA4 and MCSELA_17_CLA2 are the fastest, and the MCSELA_17_CLA2 is the best candidate only when small area is the main goal.

TABLE 3.4: DELAY AND AREA COMPARISONS OF MODIFIED CSELA WITH VARIABLE BLOCK SIZES FOR A PPR ARRAY USING VOLTAGE ISLANDS

Adder	Delay (ns)		Area (NAND2)	
Conventional CSELA (6−11−10−10−9−9−9)	24.0 × T_{XOR}	1.00	406.1	1.00
MCSELA_11_6_CRA (6−11−10−10−9−9−9_CRA)	24.5 × T_{XOR}	1.02	320.4	0.79
MCSELA_17_CRA (17−10−10−9−9−9_CRA)	35.5 × T_{XOR}	1.48	319.8	0.79
MCSELA_17_CLA4 (17−10−10−9−9−9_ CLA4)	23.5 × T_{XOR}	0.98	318.1	0.78
MCSELA_17_CLA2 (17−10−10−9−9−9_ CLA2)	23.5 × T_{XOR}	0.98	304.9	0.75
MCSELA_CLA (10−11−10−9−8−8−8_CLA)	12.7 × T_{XOR}	0.53	350.5	0.86
MCSELA_CCA (10−11−10−9−8−8−8_CCA)	15 × T_{XOR}	0.63	452.6	1.11

Figure 3.14 and Figure 3.16 show the modified CSELA using a CRA and an add-one circuit with variable block size. Figure 3.15 and Figure 3.17 show delay evaluation of the modified CSELA using a CRA and an add-one circuit with fixed 2-bit block (group 7).

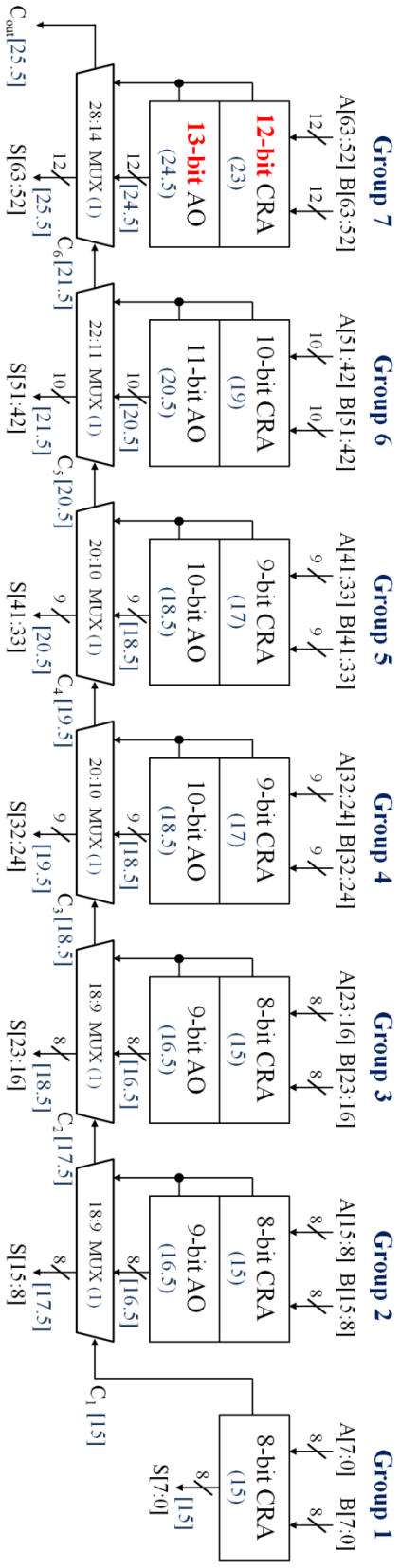


FIGURE 3.14: THE MODIFIED CSELA USING A CRA AND AN ADD-ONE CIRCUIT WITH VARIABLE BLOCK SIZE (BLOCK SIZES OF 12−10−9−9−8−8−8)

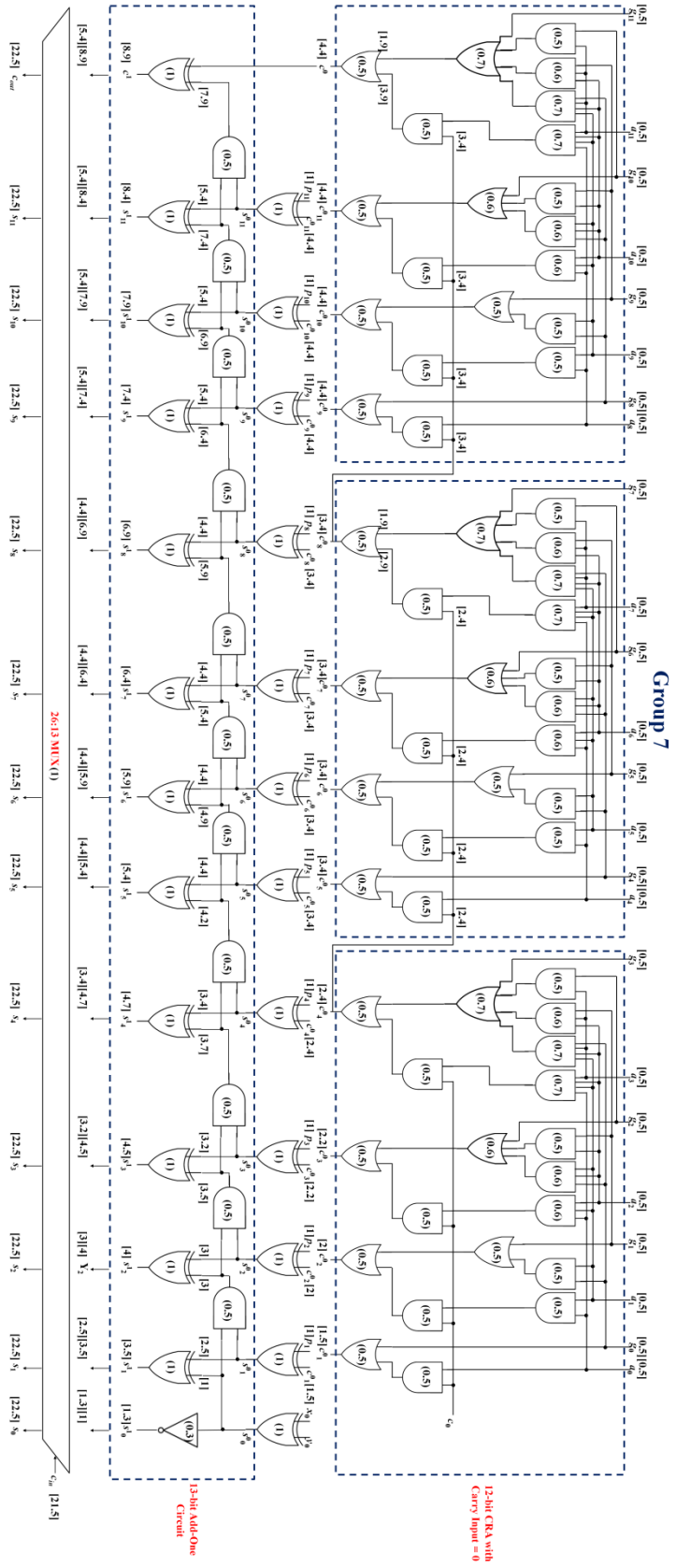


FIGURE 3.15: DELAY EVALUATION OF THE MODIFIED CSELA USING CLA4 AND AN ADD-ONE CIRCUIT (GROUP7)

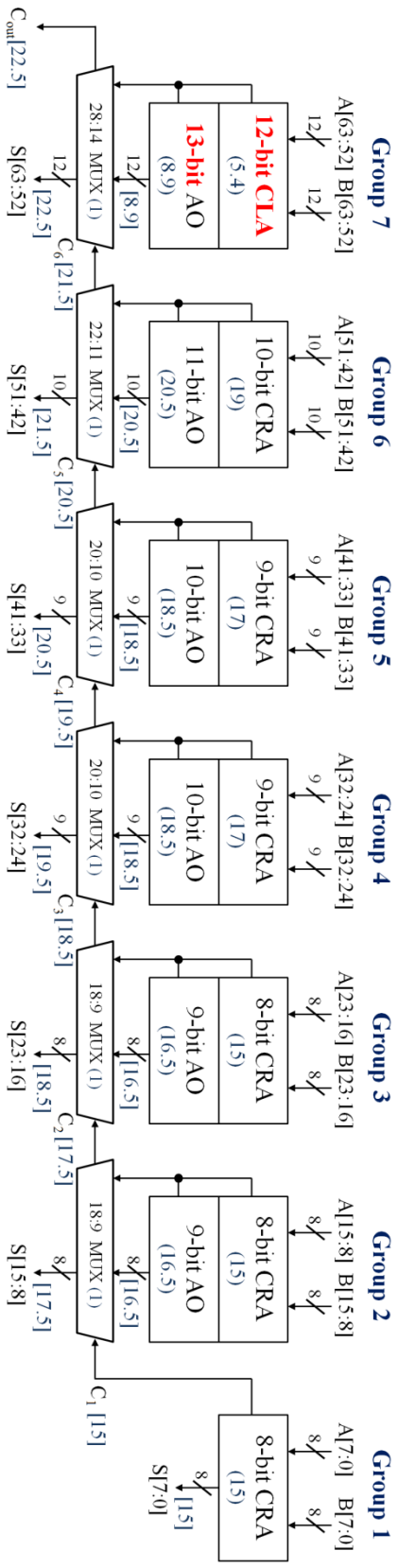


FIGURE 3.16: THE MODIFIED CSELA USING A CRA, A CLA AND AN ADD-ONE CIRCUIT WITH VARIABLE BLOCK SIZE (BLOCK SIZES OF 12−10−9−9−8−8−8)

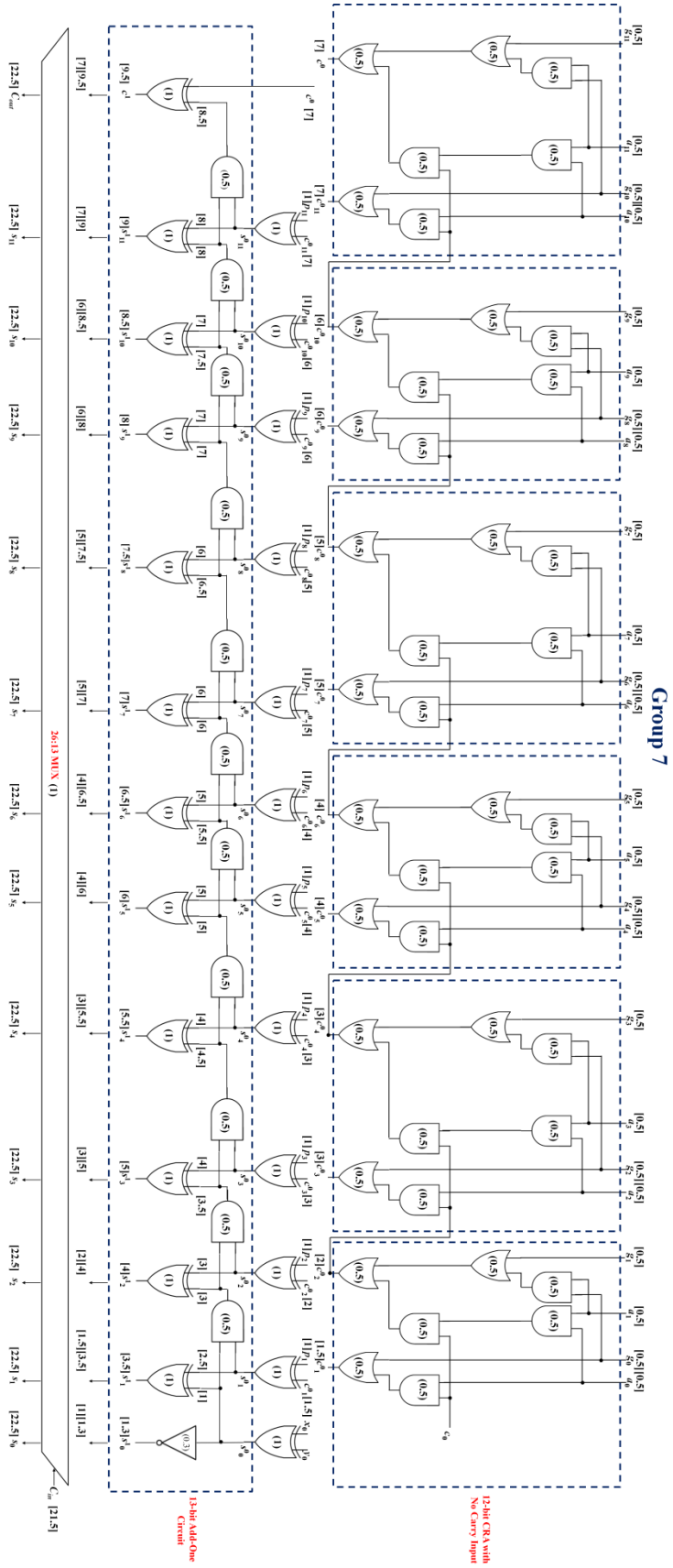


FIGURE 3.17: DELAY EVALUATION OF THE MODIFIED CSELA USING CLA2 AND AN ADD-ONE CIRCUIT (GROUP7)

3.5 Experimental Evaluation

We have implemented 64-bit adders with different structure optimizations. The design and simulation methodologies are described in Appendix A. As our major focus is on the final addition step, the array structures proposed in Chapter 2 are reused for the PPR arrays in our designs. Fortunately, as array structures are already optimized, they would not significantly affect the relative difference of actual measured values for the final CPA although they may affect absolute values. We only provide the combined results of reduction array and the CPA to consider the input signal profiles to the final CPA.

3.5.1 Results for Split Array Multipliers

Seven schemes with different structure optimization techniques are implemented. Table 3.5 summarizes the results for the proposed and conventional adders with a 4-level UL LR array structure. The smallest value of each characteristic is highlighted in boldface. The baseline structure is a conventional CSELA. Compared to a 4-level UL LR array structure with a conventional CSELA, a 4-level UL LR array structure with the modified CSELA based on CRA achieves between 12% and 16% less power, between 14% and 17% less area with between 1% and 10% delay overhead. The modified CSELA approach increases delay because the extra 1-bit add-one circuit should be executed after the carry output of a CRA with $C_{in} = 0$. The area and power reduction is because it is implemented by using a single CRA and a multiplexer-based add-one circuit instead of two CRAs. The interesting results are that all adders have similar delay, a 4-level UL LR array structure with MCSELA_CLA and MCSELA_CCA have slight delay decrease, but it is not enough large than we expect because this adder need to wait for higher input bit in LS region. They have slight decrease power

reduction (MCSELA_CLA) and power increase (MCSELA_CCA). One reason is they have more switching activities due to internal complex structure compared to CRA structure. A 4-level UL LR array structure with MCSELA_10_2_CRA dissipates 9% less delay compared to a MCSELA_12_CRA because the MS 2-bit is executed with the other group simultaneously, but the MS 2-bit of a MCSELA_12_CRA must wait for the carry bit to be calculated from the lower bit. The power and area of a 4-level UL LR array structure with MCSELA_10_2_CRA are close to those of a MCSELA_12_CRA. The good results are from a 4-level UL LR array structure with a MCSELA_12_CRA_CLA2 which reduce the power and area by approximately 15% without delay overhead. Because the area of CLAs is much larger than that of CRAs, the area and power overhead for CLAs are more evident. However, our theoretical and experimental conclusions here conflict with the previous work where only CRAs were used. Specifically a MCSELA_12_CRA_CLA2 slightly reduce the power, area as well as delay compared to a MCSELA_12_CRA_CLA4. The reason is that the number of gates required for the group size of 2-bit is smaller than those of 4-bit for implementation. In terms of area and power-delay product, a 4-level UL LR array structure with MCSELA_12_CRA_CLA2 is the best. AMCSELA_CLA presents the fast followed by a MCSELA_CCA. Table 3.6 shows delay and power comparisons of MCSELA_12_ CRA_CLA2 with two supply voltages. MCSELA_12_ CRA_CLA2 is the fastest when supply voltage is increased.

TABLE 3.5: POWER, DELAY AND AREA COMPARISONS OF ADDERS FOR A 4-LEVEL UL LR STRUCTURE

Adder	Power (μW)		Delay (ns)		Area (μm^2)		Power-Delay Product (pJ)	
4-level UL LR array structure with a conventional CSELA	5172	1.00	6.88	1.00	15779	1.00	35.53	1.00
4-level UL LR array structure with MCSELA_10_2_CRA	4396	0.85	6.95	1.01	13571	0.86	30.55	0.86
4-level UL LR array structure with MCSELA_12_CRA	**4345**	**0.84**	7.47	1.10	13096	0.85	32.46	0.92
4-level UL LR array structure with MCSELA_12_CRA_CLA4	4551	0.88	6.91	1.02	13503	0.85	31.45	0.89
4-level UL LR array structure with MCSELA_12_ CRA_CLA2	4348	0.84	6.88	1.00	**13536**	**0.83**	**29.87**	**0.84**
4-level UL LR array structure with MCSELA_CLA	4914	0.95	**6.54**	**0.95**	14517	0.92	30.42	0.86
4-level UL LR array structure with MCSELA_CCA	6102	1.18	6.61	0.96	17672	1.12	40.33	1.14

TABLE 3.6: POWER AND DELAY COMPARISONS OF THE MCSELA_12_CRA_2 WITH DIFFERENT SUPPLY VOLTAGE

Adder	Power (μW)		Delay (ns)		Power-Delay Product (pJ)	
4-level UL LR array structure with MCSELA_12_ CRA_CLA2 at 1.08V [1]	**4348**	**1.00**	6.88	1.00	**29.87**	**1.00**
4-level UL LR array structure with MCSELA_12_ CRA_CLA2 at 1.32V [2]	5044	1.16	**5.99**	**0.87**	30.21	1.01

[1] Array structure: 1.08V, CPA: 1.08V

[2] Array structure: 1.08V, CPA: 1.32V

3.6 Summary

In Chapter 2, we have proposed the structure optimization of PPR arrays. In this chapter, we have studied the signal arrival profiles of the corresponding structures and have presented several delay reduction techniques. The problem is uniform signal input profile to the CPA, but it is not perfectly flat. Detailed experimental results are given to compare the power, delay and area characteristics of each final CPA. Among different optimization techniques for the final CPA, the 4-level UL LR array structure with MCSELA_12_CRAis a primary choice if power is the critical concern. Compared to a 4-level UL LR array structure with a conventional CSELA, this structure reduces the power by approximately 16% with 10% delay overhead. When a small delay is the main goal, the 4-level UL LR array structure with MCSELA_CLA is the best candidate. Compared to a 4-level UL LR array structure with a conventional CSELA, this achieves 5% less power and 5% less delay. In terms of power-delay product, a 4-level UL LR array structure with MCSELA_12_CRA_CLA2 is the best.

Chapter 4 Low-Power Sum-of-Products Unit for Signal Processing Applications

Power dissipation is a critical aspect in today's mobile environment, while high throughput remains a major design goal. To satisfy both requirements, parallelism in the organization of arithmetic units has been employed. Parallel organization can reduce execution time and run at a lower supply voltage, which can reduce power consumption for dynamic power compared to a single multiplier solution.

The previous two chapters presented optimization techniques for the multiplier array and the CPA, respectively. In this chapter, we propose a new design for a sum-of-products unit suitable for signal processing application and present an approach to reducing power dissipation in the design of a sum-of-products operation by utilizing two optimized multipliers while maintaining high throughput. We show that our design outperforms schemes using a single multiplier.

4.1 Introduction

With an increasing complexity of circuits used in mobile devices and increased demand for digital signal processing applications, minimizing power consumption in digital CMOS circuits has become of great importance while performance and area remain the other two major design goals. Most DPSs and GPUs use an existing multiplier or a MAC unit to perform various arithmetic operation [84][85][86]. The multiplier and MAC unit are frequently used but power-demanding components of the DSPs and GPUs. However, traditional DSPs and GPUs require many clock cycles for signal processing applications even when they include high-performance parallel multipliers and/or a MAC unit. This is the critical problem that we

are facing in the recent signal processing applications which require intensive multiplications. Therefore, research on a new arithmetic design is needed to satisfy low-power and high-throughput requirements for signal processing applications in mobile systems.

The proposed sum-of-products designs have advantages over a single multiplier in that they use two multiplications to be performed in parallel and thus reduce execution time. We discuss several key advantages of the proposed arithmetic unit, compare it to other competing arithmetic units and demonstrate its superiority over other arithmetic units. Of course, our design increases the area and power compared to a single multiplier. However, the proposed design can reduce the execution time significantly. We discuss how the energy is reduced (shorter execution time) in typical signal processing applications. Our goal is to reduce the power consumption without increasing the latency and the complexities of arithmetic units.

In this chapter, we focus on describing the overall organization of sum-of-products design and showing competitive advantage. This chapter does not consider how to further optimize the core components for a sum-of-products design: PPR arrays and the final CPA. The previous two chapters presented the optimization of these components. In Chapter 2, we presented several power and delay reduction techniques of PPR arrays. In Chapter 3, we combined these optimization techniques with the structure optimization techniques of the final CPA. This chapter is organized as follows. Section 4.2 addresses the problem of conventional arithmetic units and proposes the sum-of-products unit. Section 4.3 shows the experimental results. We provide energy and execution time estimates for the sum-of-products design and compare them to the estimates for a conventional ARM7 multiplier and the proposed LR array multipliers [51][88]. In Section 4.4, we discuss current problems in our designs, and give a summary. The designs presented in this chapter assume 32-bit integer operands, but they can easily be extended to longer fixed-point operands.

4.2 Sum-of-Products Design

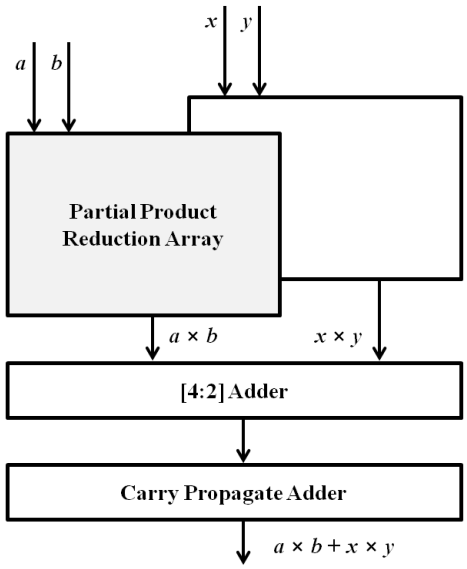
Several power optimization techniques have been proposed in the literature for multipliers and adders. It was obvious that the use of these power optimization techniques in multipliers and adders has been beneficial. However, it is not known if these techniques are suitable for sum-of-products units. Detailed studies are desirable to explore the potential power saving of a sum-of-products unit. In this section, we present the problems of the current arithmetic design and describe a new arithmetic design for solving these problems.

4.2.1 The Proposed Design

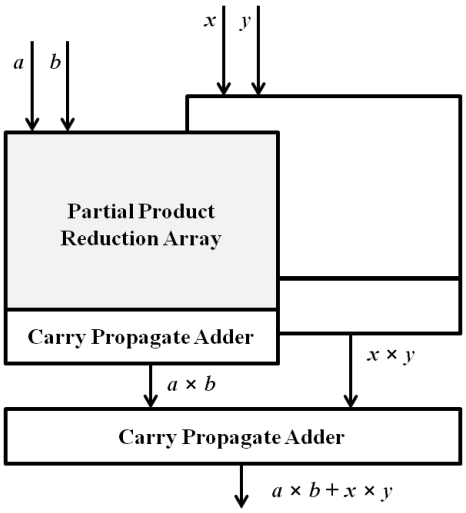
The sum-of-products baseline model consists of two multipliers and a single adder. There are two types of structures. One way to design the sum-of-products is to use two PPR arrays and [4:2] adders followed by a single final CPA. The other way is to use two complete multipliers and then add two products to produce the final result. This structure has two PPR arrays and two CPAs followed by a single CPA. The first structure would be a better solution because it has one less carry-propagate addition; thus, the power dissipation and delay are better than those of its counterpart. The sum-of-products can be extended to the inner-product operation. Inner-products consists of two PPR arrays, [6:2] adders and latches for accumulation and a single CPA. The [6:2] adders accumulate four inputs with the previous partial sums and carries. Figure 4.1 shows the baseline models of a sum-of-products, and Figure 4.2 shows the structure of an inner-product arithmetic unit [1].

Our goal is to reduce the power consumption without a significant increase in delay. The PPR arrays and the final CPA are two major components of a sum-of-products unit. Thus, the two main components should be optimized to minimize power and delay from an overall structure prospective. The power dissipation introduced by the PPR array is relatively large

compared to the final CPA because of a large number of gates implemented. We focus on developing low-power PPR arrays based on the LR array multipliers discussed in [51][88], and have proposed new power optimization techniques for LR array structures. These optimization techniques include the 4-level UL LR splitting. We have described detailed optimization techniques of reduction structures for arrays in Chapter 2. On the other hand, the carry propagation path is the critical path in sum-of-products design, and thus the final CPA requires the fastest adder. Following the direction from Chapter 2, we have combined this structure optimization technique with the final adder with signal flow optimization. In Chapter 3, we have discussed a design strategy specific to input arrival time generated by the proposed arrays, and proposed the high-performance, low-power CPA where the input arrival times are not the same as those of conventional multipliers.



(A) TWO PPR ARRAYS, [4:2] ADDER AND A SINGLE CPA



(B) TWO COMPLETE MULTIPLIERS AND A SINGLE CPA (TWO PPR ARRAYSAND THREE CPAS)

FIGURE 4.1: SUM-OF-PRODUCTS UNIT DESIGN

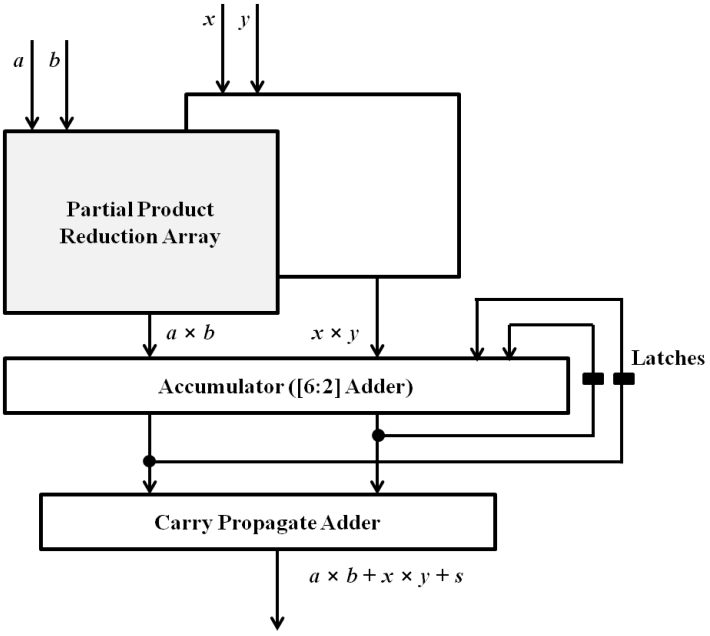


FIGURE 4.2: INNER-PRODUCT UNIT DESIGN

4.3 Experimental Results

The objective of these experiments is to compare the number of cycles and execution time between a parallel organization and a solution with a single multiplier. In these experiments, we use two types of multipliers and sum-of-products with different structure optimization techniques: ARM7 multiplier and a 4-level UL LR split array multiplier with modified CSELA. As our major focus is on overall structure, we use the optimized modules for our experiments. A detailed description of experimental methodologies is given in Appendix B. We also consider the approach to avoid the overflow. Finite length implementation implies maximum representable number. Whenever the results exceed this value, overflow occurs. We assume the input operands are 32-bit integers. Input random variables less than or equal to 32-bit were automatically generated by Cadence tool, so all the input operands avoid the overflow. However, it is possible that the result overflow can occur. To avoid result overflow, we need to increase the number of bits to the output. We can prevent overflow by increasing 1 bit in result data. This prevents a possible overflow.

To compare our results, we select four benchmark programs: a FIR filter, a high pass filter, a matrix multiplication and an Euclidean distance. They are representative signal processing applications using sum-of-products operations. Their mathematical expressions are as below.

FIR filter (We consider 4-tap in this experiment.)

$$y[n] = \sum_{k=-\infty}^{\infty} c[k] \times x[n-k]$$

High pass filter

$$y[n] = \sum_{i=1}^{n} a \times y[n-1] + a \times (x[n] - x[n-1])$$

Matrix multiplication

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nm} \end{bmatrix}, B = \begin{bmatrix} B_{11} & \cdots & B_{1p} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mp} \end{bmatrix} A \cdot B = \begin{bmatrix} AB_{11} & \cdots & AB_{1p} \\ \vdots & \ddots & \vdots \\ AB_{n1} & \cdots & AB_{np} \end{bmatrix}$$

Euclidean distance

$$\mathrm{d}(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

4.3.1 ARM Multiplier Results

The effective approach to comparing relative power dissipation (or energy) and performance is to use the power-delay (or energy-delay) product. For our experiment, it might be more appropriate to use energy because power is the rate at which energy is consumed, while energy is the amount of power consumed. To calculate energy, the total execution time of programs is needed. The execution time required for a program can be written as [90]

Execution time for a program
= Clock cycles for a program × Clock cycle time (4-1)
= Instructions for a program × Clock cycles per instruction × Clock cycle time

We need to limit our simulations to the specific compiler, Instruction Set Architecture (ISA) and micro architecture for accurate results. We consider the ARM's architecture because it has been dominant in mobile devices. All ARM processors have included hardware support for integer multiplication and used two styles of multiplier [91]. The ARM with an M in its name (for example the ARM7DM) has a high-performance multiplier and supports the 64-bit results by using multiply and MAC instructions. This multiplier employs a radix-4 algorithm to produce 2-bit PPs. The carry save array has four layers of adders, each handling two multiplier bits, so the array can multiply 8-bit per clock cycle. The array is cycled up to four times, and the partial sums and carries are combined 32-bit at a time and written back into the register. We consider only high-performance multipliers of the ARM processor in this work because the multiplication performance is critical in signal processing applications. Our experimental design is the ARM7TDMI-S processor which includes an enhanced 32 × 8 single multiplier with a radix-4 algorithm. It is a synthesizable version of the ARM7TDMI core and thus can

provide an accurate and efficient method when trying to measure the cycle counts for an application executed on the ARM multiplier with the cycle-level simulator. The ARM7 processor supports two different ISAs: the 32-bit ARM and the 16-bit Thumb with a T in its name. The Thumb ISA allows for code to be smaller, and can potentially be faster if cache memory to store code cannot be accessed fast but MAC operations are not supported. Therefore, we only consider the ARM ISA when exploring our design. Unfortunately, the ARM7TDMI-S does not include a sum-of-products hardware, but an enhanced single multiplier, and thus, cannot support a single-cycle sum-of-products instruction. The ARM compiler usually avoids generating the sum-of-products instructions, and hence we cannot directly measure the total clock cycles with sum-of-products using cycle-level simulation with compiled assembly code. We have to regenerate the ARM assembly code including sum-of-products manually after analyzing the compiled original ARM assembly code. Suppose we have the modified implementation of ARM7TDMI-S ISA. We note that two consecutive multiply operations are replaced with a sum-of-products operation. Actually, the sum-of-products instruction can execute two multiplications simultaneously, and then two products are added in a CPA to obtain their sum. The ARM7 multiplication finishes in at most 4 cycles and thus a sum-of-products operation takes up to 5 clock cycles due to an additional single cycle final addition. To recreate the modified ARM assembly code, we use the ARM technical reference manual after compiling the original C code [92]. The reference manual defines all instructions and their cycle counts.

The clock cycles of the ARM multiplier for benchmark programs can be measured by running a cycle-level simulation tool using the ARM7TDMI-S Verilog code and the compiled ARM assembly code. The Mentor Graphics hardware/software co-simulation tool such as Questa Codelink profiles clock cycles for programs. The comparison results of clock cycle estimates are shown in Table 4.1. We address the benchmark simulation results from three aspects. First, the cycle reduction is smaller than we expected. Based on an analysis of clock

cycles, the clock cycles of sum-of-products are between 13% and 48% less than those of multiplication only for benchmark programs. In an ideal situation, the sum-of-products would be expected to have 50% reduction in total clock cycles for benchmark programs. Unfortunately, however, the sum-of-products requires more clock cycles to execute the initialization code, and thus this ideal situation does not occur in practice. When the ARM assembly code is generated, the execution environment such as entry point and register initialization should be set up before the main function starts. All the registers have to be initialized to zero, and the counter is also loaded before the execution of the program enters the main function. Unfortunately, the sum-of-products spends a lot of time initializing code compared to a multiplier because it needs more register initialization. Thus, the total number of cycles in sum-of-products is over 50% of those in multiplier, even though the main function of sum-of-products takes approximately half the number of clock cycles. Second, the ratio of the total number of clock cycles for multiplier to sum-of-products would coverage to 50% when the number of iterations increases because the number of cycles in initial functions is fixed regardless of the number of loop iterations, while those for the main code depend upon the number of iterations. When the number of iteration increases, the portion of initialization code becomes smaller. Thus, the initialization portion of sum-of-products becomes relatively smaller. Third, the ratio of the total number of clock cycles for multiplier to sum-of-products in matrix multiplication (67% ~ 87%) and Euclidean distance (62% ~ 76%) are relatively higher than that in other benchmarks (FIR filter: 58% ~ 63%, high pass filter: 52% ~ 55%). This is because the main code of matrix multiplication using sum-of-products operations includes many more three-cycle load instructions than that in other benchmark programs. Also the Euclidean Distance includes additional square root instruction. The ARM7TDMI-S has no square root instructions, so this instruction is generated by the combination of ADD, SUB and MOV instructions. Thus, the square root instruction is handled in approximate 100 clock cycles to complete. It would increase the total number of clock cycles, and lead to longer execution time. They might affect larger energy-delay product.

Consequently, the matrix multiplication and Euclidean distance take many more clock cycles compared to other benchmark programs.

TABLE 4.1: CLOCK CYCLES FOR BENCHMARK PROGRAMS

Clock cycles	FIR Filter (length = 10)		High Pass Filter (length = 10)		Matrix Multiplication (2 × 2)		Euclidean Distance (length = 10)	
Multiplication	155	1.00	177	1.00	168	1.00	236	1.00
Sum-of-products	97	0.63	98	0.55	146	0.87	179	0.76

Clock cycles	FIR Filter (length = 100)		High Pass Filter (length = 100)		Matrix Multiplication (5 × 5)		Euclidean Distance (length = 100)	
Multiplication	1415	1.00	1617	1.00	2528	1.00	1586	1.00
Sum-of-products	817	0.58	845	0.52	1686	0.67	989	0.62

We can directly measure the power and latency of the ARM multiplier using the Synopsys Design Compiler because ARM7TDMI-S is a synthesizable core. Then we can estimate those of the sum-of-products hardware. We assume the sum-of-products hardware consists of two identical ARM7TDMI-S multipliers and Arithmetic Logic Unit (ALU). Table 4.2 shows the power, delay and area of a multiplier and a sum-of-products hardware.

TABLE 4.2: POWER, DELAY AND AREA OF THE ARM7TDMI-S MULTIPLIER AND A SUM-OF-PRODUCTS HARDWARE

Supply Voltage	Hardware	Power (mW)	Delay (ns)	Area (μm^2)
1.32 V	Multiplier*	1.68	0.99	1384
	Sum-of-products**	3.46	1.02	2941
1.2 V	Multiplier*	1.25	1.15	1316
	Sum-of-products**	2.58	1.19	2788
1.08 V	Multiplier*	0.94	1.42	1364
	Sum-of-products**	1.94	1.48	2896

* measured value ** estimated value

The amount of energy used depends on the power dissipation and the time for which it is used, and can be written as

$$\text{Energy (Joules)} = \text{Power (Watts)} \times \text{Time (Seconds)} \quad (4\text{-}2)$$

The easiest and most accurate way to calculate the execution time for benchmark programs is to use the equation (4-9) with measured clock cycles for programs and clock rate. We calculate the energy based on the execution time calculated from the equation (4-9) and the amount of power dissipation measured in Table 4.2. Table 4.3 summarizes the energy, execution time and energy-delay product. The power-delay product and energy-delay product are commonly used

to compare the superiority of designs [93]. In a sense, this is a misnomer as power × delay = (energy / delay) × delay = energy [9]. Instead, the energy-delay product should be used because it involves two independent measures of circuit. The sum-of-products unit dissipates 23% more energy than a single multiplier with a 40% decrease in execution time for a FIR filter program and 12% more energy with a 46% decrease in execution time for a high pass filter. Also, the sum-of-products unit dissipates 42% and 33% more energy than a single multiplier with a 31% and 35% decrease in execution time for a matrix multiplication and Euclidean distance, respectively. The sum-of-products units are better than multipliers only in terms of energy-delay product in the considered benchmarks.

If a single multiplier which operates at higher supply voltage is replaced by a sum-of-products which operates at low supply voltage, the energy can be reduced. This is because the clock cycles per program with a sum-of-products are reduced by approximately half compared to those with the multiplier while reducing supply voltage increases the clock cycle time slightly. For example, if we replace the ARM multiplier at 1.32V with the sum-of-products at 1.08V for a high pass filter program, 22% in execution time and 10% in energy can be decreased. For a FIR filter program, sum-of-products has 14% less execution time while keeping the same energy. As a result, the total energy demanded by the design can be reduced if we use parallel hardware at lower supply voltage.

The multiplier and sum-of-products are characterized in the execution time ratio versus energy ratio in Figure 4.3. Solid lines indicate the measured value and dashed lines indicate the expected value based on trend lines. Energy ratio is decreased as execution time ratio is increased. The sum-of-products unit consumes more energy as the difference of execution time between a sum-of-products and a multiplier is increased, but the energy ratio is expected to be less than 1 if their execution time is the same. This means the sum-of-products unit consumes less energy than a multiplier only when the execution time is the same. The FIR filter, high pass

filter and Euclidean distance programs would be energy-efficient when a single multiplier is replaced with a sum-of-products hardware, but the matrix multiplication program would have the same energy. This is because the difference of clock cycles between sum-of-products and a multiplier only is relatively small. Figure 4.4 shows the energy-delay product comparison between the ARM7TDMI-S multiplier and sum-of-products unit. The experiment shows that the sum-of-products design is better than a single multiplier approach in terms of energy-delay product in all benchmarks.

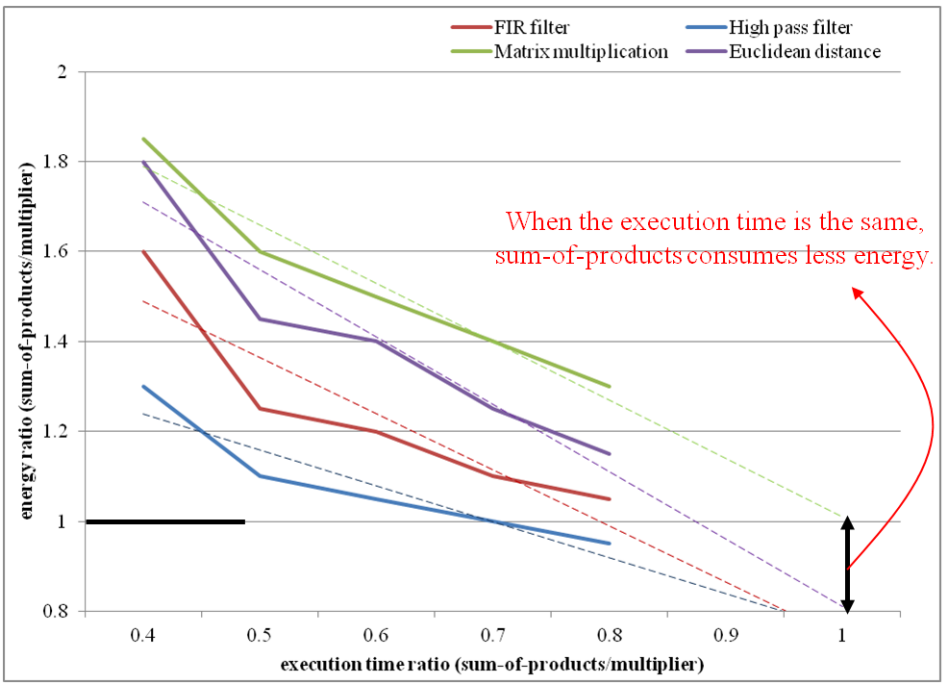


FIGURE 4.3: COMPARISON OF ENERGY RATIO WITH EXECUTION TIME RATIO IN BENCHMARKS

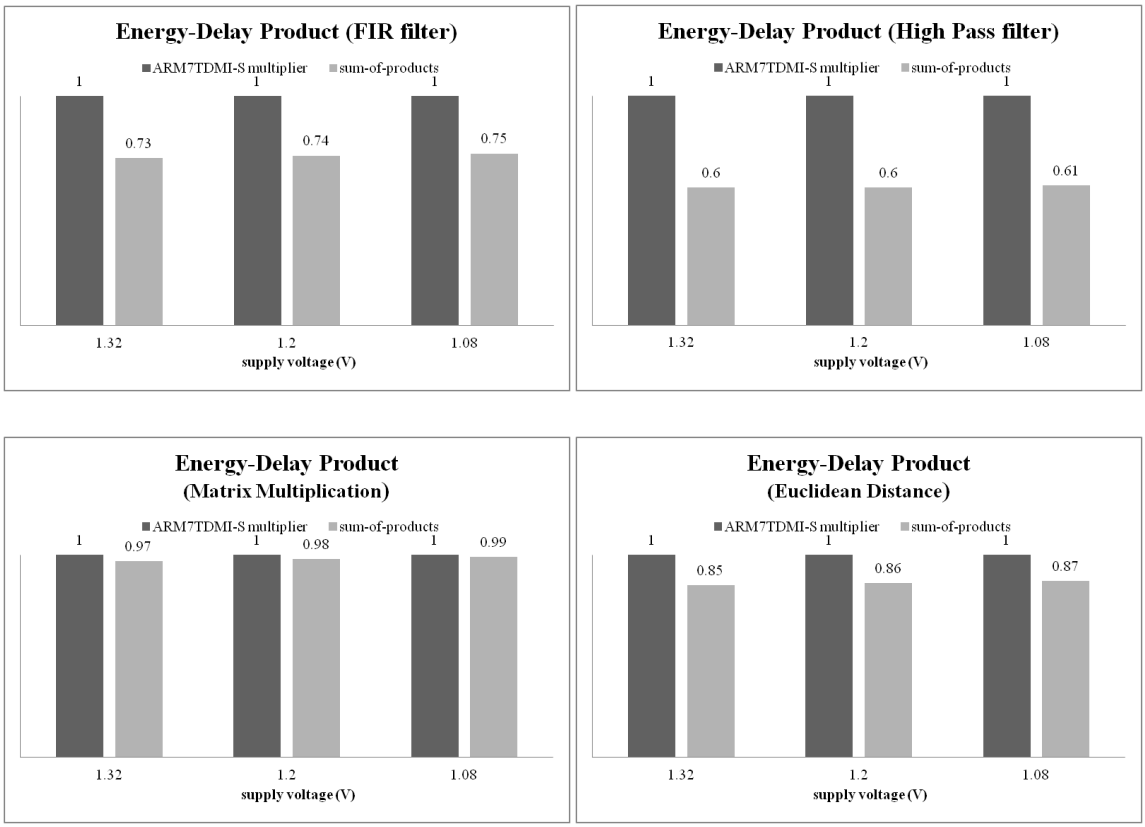


FIGURE 4.4: ENERGY-DELAY PRODUCT COMPARISON BETWEEN THE ARM7TDMI-S MULTIPLIER AND A SUM-OF-PRODUCTS UNIT IN BENCHMARKS

TABLE 4.3: EXECUTION TIME, ENERGY AND ENERGY-DELAY PRODUCT OF THE ARM7TDMI-S MULTIPLIER AND A SUM-OF-PRODUCTS HARDWARE FOR BENCHMARKS

Benchmark Programs	Supply Voltage	Hardware	Execution Time (µs)		Energy (µJ)		Energy-Delay Product	
FIR Filter (length = 100)	1.32 V	ARM7TDMI-S Multiplier*	1.40	1.00	2.35	1.00	3.29	1.00
		Sum-of-products**	0.83	0.59	2.88	1.23	2.40	0.73
	1.2 V	ARM7TDMI-S Multiplier	1.63	1.00	2.03	1.00	3.30	1.00
		Sum-of-products	0.97	0.60	2.50	1.23	2.43	0.74
	1.08 V	ARM7TDMI-S Multiplier	2.01	1.00	1.89	1.00	3.80	1.00
		Sum-of-products	1.21	0.60	2.35	1.24	2.84	0.75
High Pass Filter (length = 100)	1.32 V	ARM7TDMI-S Multiplier	1.60	1.00	2.69	1.00	4.30	1.00
		Sum-of-products	0.86	0.54	2.98	1.11	2.56	0.60
	1.2 V	ARM7TDMI-S Multiplier	1.86	1.00	2.32	1.00	4.31	1.00
		Sum-of-products	1.00	0.54	2.59	1.12	2.60	0.60
	1.08 V	ARM7TDMI-S Multiplier	2.30	1.00	2.16	1.00	4.96	1.00
		Sum-of-products	1.25	0.55	2.43	1.12	3.04	0.61
Matrix Multiplication (5 × 5)	1.32 V	ARM7TDMI-S Multiplier*	2.50	1.00	4.20	1.00	10.52	1.00
		Sum-of-products**	1.72	0.68	5.95	1.42	10.23	0.97
	1.2 V	ARM7TDMI-S Multiplier	2.91	1.00	3.63	1.00	10.56	1.00
		Sum-of-products	2.01	0.69	5.18	1.42	10.39	0.98
	1.08 V	ARM7TDMI-S Multiplier	3.59	1.00	3.37	1.00	12.11	1.00
		Sum-of-products	2.50	0.70	4.84	1.43	12.08	0.99
Euclidean Distance (length = 100)	1.32 V	ARM7TDMI-S Multiplier	1.57	1.00	2.64	1.00	4.14	1.00
		Sum-of-products	1.00	0.64	3.49	1.32	3.52	0.85
	1.2 V	ARM7TDMI-S Multiplier	1.82	1.00	2.28	1.00	4.16	1.00
		Sum-of-products	1.18	0.65	3.04	1.33	3.57	0.86
	1.08 V	ARM7TDMI-S Multiplier	2.25	1.00	2.12	1.00	4.77	1.00
		Sum-of-products	1.46	0.65	2.84	1.34	4.16	0.87

4.3.2 The Design Characteristics of the Proposed Sum-of-Products Units

To verify our results and to determine the implications of the proposed organization, we conducted another experiment. In this experiment, we use the proposed multipliers and sum-of-products instead of ARM7 multipliers. We assume the proposed multipliers and sum-of-products units are replaced with embedded ARM7 multipliers, and the clock cycles for benchmark programs are the same as those in ARM7TDMI-S test environments. We implement the proposed sum-of-products unit using Verilog and a top-down methodology. The proposed designs are synthesized with three supply voltages 1.08V, 1.20V and 1.32V supported by technology. Five schemes with different split array structures and CPA optimization techniques, discussed in detail in Chapter 2 and Chapter 3,are implemented and evaluated: 1) a non-split LR array structure using a [3:2] adder with a conventional CSELA, 2) a 2-level UL LR array structure using a [4:2] adder with a conventional CSELA, 3) a 4-level UL LR array structure using a [4:2] adder with a conventional CSELA, 4) a 4-level UL LR array structure using a [4:2] adder with MCSELA_12_CLA2, which was proposed in Chapter 3. Chapter 2 and Chapter 3 gave a detailed description of organization, design and implementation. The comparison results of power, delay and area estimates are shown in Table 4.4. The smallest value of each characteristic is highlighted in boldface. The baseline structure is a non-split LR array structure using a [3:2] adder with a conventional CSELA. Compared to a non-split array structure, a 2-level structure achieves 24% less power and 45% less delay, while a 4-level structure achieves between 34% and 42% less power and between 58% and 59% less delay with negligible area increase. This result implies more power savings can be achieved if each part is split further. Among all schemes, a 4-level UL LR array structure using a [4:2] adder with MCSELA_12_CLA2 presents the lowest power, the smallest delay and the lowest power-delay

product results in this experiment. These results indicate that a 4-level split structure with a modified CSELA is a useful power- and delay-saving technique in sum-of-products design.

TABLE 4.4: POWER, DELAY AND AREA COMPARISON FOR LR ARRAY MULTIPLIERS UTILIZING SPLIT STRUCTURE AND MODIFIED CPA (1.32 V)

Sum-of-Products	Power (mW)		Delay (ns)		Area (μm^2)		Power-Delay Product (pJ)	
Non-split LR array structure using a [3:2] adder with a conventional CSELA	6.98	1.00	9.48	1.00	11221	1.00	66.17	1.00
2-level UL LR array structure using a [4:2] adder with a conventional CSELA	5.31	0.76	5.21	0.55	**11109**	**0.99**	27.66	0.42
4-level UL LR array structure using a [4:2] adder with a conventional CSELA	4.61	0.66	3.98	0.42	11671	1.04	18.34	0.28
4-level UL LR array structure using a [4:2] adder with MCSELA_12_CLA2*	**4.05**	**0.58**	**3.89**	**0.41**	11445	1.02	**15.74**	**0.24**

*MCSELA_12_CLA2: Modified CSELA with variable block size (block size of 8-8-8-9-10-12)

In this experiment, we consider only split LR array multipliers with the modified CPA. Table 4.5 shows power, delay and area estimates for the proposed sum-of-products design. Synthesis results indicate that the PPR arrays have the most power and area of the sum-of-products design, but the final CPA has a longer latency than PPR arrays because PPR arrays have a 4-level split structure using a [4:2] adder. The delay of a 4-level split structure is 50% less than that of a non-split structure.

TABLE 4.5: POWER, DELAY AND AREA FOR SUM-OF-PRODUCTS (1.32 V)

Hardware	Power (mW)		Delay (ns)		Area (μm^2)	
Sum-of-products	8.49	1.00	6.85	1.00	25850	1.00
LR_4ULS_42*	4.14	0.48	2.10	0.31	12314	0.48
[4:2] adder + MCSELA_12_CLA2	0.21	0.04	4.75	0.69	1222	0.04

* LR_4ULS_42: 4-Level UL Split LR Multiplier using a [4:2] adder

Table 4.6 shows the power, delay and area of the proposed multiplier and sum-of-products hardware, and Table 4.7 summarizes the energy and execution time. The multiplier delays between two experiments are different. The proposed multipliers dissipate more power, delay and area than ARM7 multipliers. The ARM7TDMI-S processor has a single 32 × 8 tree multiplier, while the proposed multiplier is a 4-level split array multiplier. The proposed multiplier has an additional two summation stage using a [4:2] adder, and so has a longer delay and larger area compared to the ARM multiplier. Furthermore, our proposed designs were not optimized with regard to the power, delay and area compared to ARM's design.

The sum-of-products unit dissipates 18% ~ 42% and 7% ~ 29% more energy than a single multiplier while 29% ~ 39% and 36% ~ 45% decrease in execution time and 1% ~ 28% and 19% ~ 41% less in energy-delay product for a FIR filter and high pass filter programs, respectively. The results are from the matrix multiplication program which reduces the execution time by 23% while increasing the energy by 51% and the energy-delay product by 17%. This is because the ratio of the execution time for multiplier to sum-of-products in this program (77%) is much larger than that in other benchmark programs (FIR filter: between 61% and 71%, high pass filter: between 55% and 64%). The sum-of-products is better than the multiplier only solution in terms of energy-delay product with 1.08V supply voltage for Euclidean distance, but with 1.32V and 1.20V supply voltage, the sum-of-products hardware has a slightly larger energy-delay product than the multiplier. This relative difference in

energy-delay product between the proposed multiplier and the sum-of-products hardware is negligible.

TABLE 4.6: POWER, DELAY AND AREA OF THE PROPOSED MULTIPLIER AND SUM-OF-PRODUCTS UNIT

Supply Voltage	Hardware	Power (mW)	Delay (ns)	Area (μm^2)
1.32 V	LR_4ULS_42	4.35	5.90	13536
	Sum-of-products	8.49	6.85	25850
1.2 V	LR_4ULS_42	3.32	6.58	12458
	Sum-of-products	6.48	7.68	23991
1.08 V	LR_4ULS_42	2.40	7.78	12590
	Sum-of-products	4.69	9.04	24143

Figure 4.5 shows the comparison of energy ratio with execution time ratio in benchmarks. When the execution time is the same, sum-of-products consumes less energy than a single multiplier. However, the matrix multiplication program would be not energy-efficient when a multiplier is replaced with a sum-of-products hardware. Figure 4.6 shows the energy-delay product comparison. The experiment shows that the sum-of-products design is better than a single multiplier in terms of energy-delay product in most benchmark programs.

TABLE 4.7: EXECUTION TIME, ENERGY AND ENERGY-DELAY PRODUCT OF THE PROPOSED MULTIPLIER AND SUM-OF-PRODUCTS HARDWARE FOR BENCHMARKS

Benchmark Programs	Supply Voltage	Hardware	Execution Time (µs)		Energy (µJ)		Energy-Delay Product	
FIR Filter (length = 100)	1.32 V	LR_4ULS_42	8.07	1.00	33.39	1.00	269.32	1.00
		Sum-of-products	5.60	0.69	47.51	1.42	265.91	0.99
	1.2 V	LR_4ULS_42	8.89	1.00	29.50	1.00	262.16	1.00
		Sum-of-products	6.27	0.71	40.66	1.38	255.12	0.97
	1.08 V	LR_4ULS_42	12.20	1.00	29.27	1.00	357.06	1.00
		Sum-of-products	7.39	0.61	34.64	1.18	255.83	0.72
High Pass Filter (length = 100)	1.32 V	LR_4ULS_42	9.22	1.00	38.16	1.00	351.70	1.00
		Sum-of-products	5.79	0.63	49.14	1.29	284.45	0.81
	1.2 V	LR_4ULS_42	10.15	1.00	33.71	1.00	342.36	1.00
		Sum-of-products	6.49	0.64	42.05	1.25	272.90	0.80
	1.08 V	LR_4ULS_42	13.94	1.00	33.45	1.00	466.28	1.00
		Sum-of-products	7.64	0.55	35.83	1.07	273.67	0.59
Matrix Multiplication (5 × 5)	1.32 V	LR_4ULS_42	14.92	1.00	64.88	1.00	967.71	1.00
		Sum-of-products	11.55	0.77	98.05	1.51	1132.41	**1.17**
	1.2 V	LR_4ULS_42	16.63	1.00	55.23	1.00	986.37	1.00
		Sum-of-products	12.95	0.78	83.91	1.52	1086.45	**1.18**
	1.08 V	LR_4ULS_42	19.67	1.00	47.20	1.00	928.38	1.00
		Sum-of-products	15.24	0.77	71.48	1.51	1089.50	**1.17**
Euclidean Distance (length = 100)	1.32 V	LR_4ULS_42	9.04	1.00	37.43	1.00	338.34	1.00
		Sum-of-products	6.77	0.75	57.52	1.54	389.66	**1.15**
	1.2 V	LR_4ULS_42	9.96	1.00	33.07	1.00	329.35	1.00
		Sum-of-products	7.60	0.76	49.22	1.49	373.84	**1.14**
	1.08 V	LR_4ULS_42	13.67	1.00	32.81	1.00	448.57	1.00
		Sum-of-products	8.94	0.65	41.93	1.28	374.88	0.84

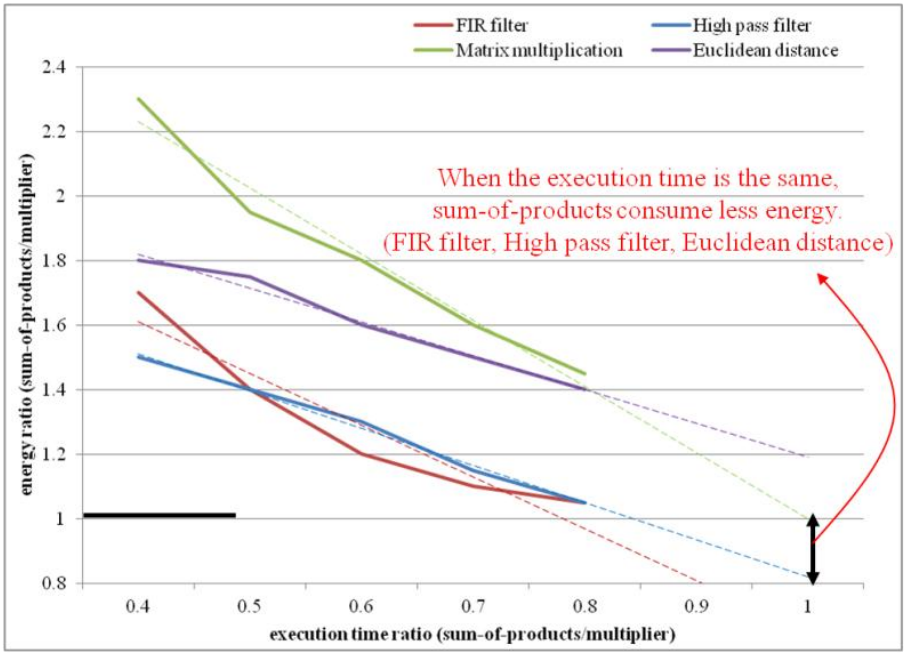


FIGURE 4.5: COMPARISON OF ENERGY RATIO WITH EXECUTION TIME RATIO IN BENCHMARKS

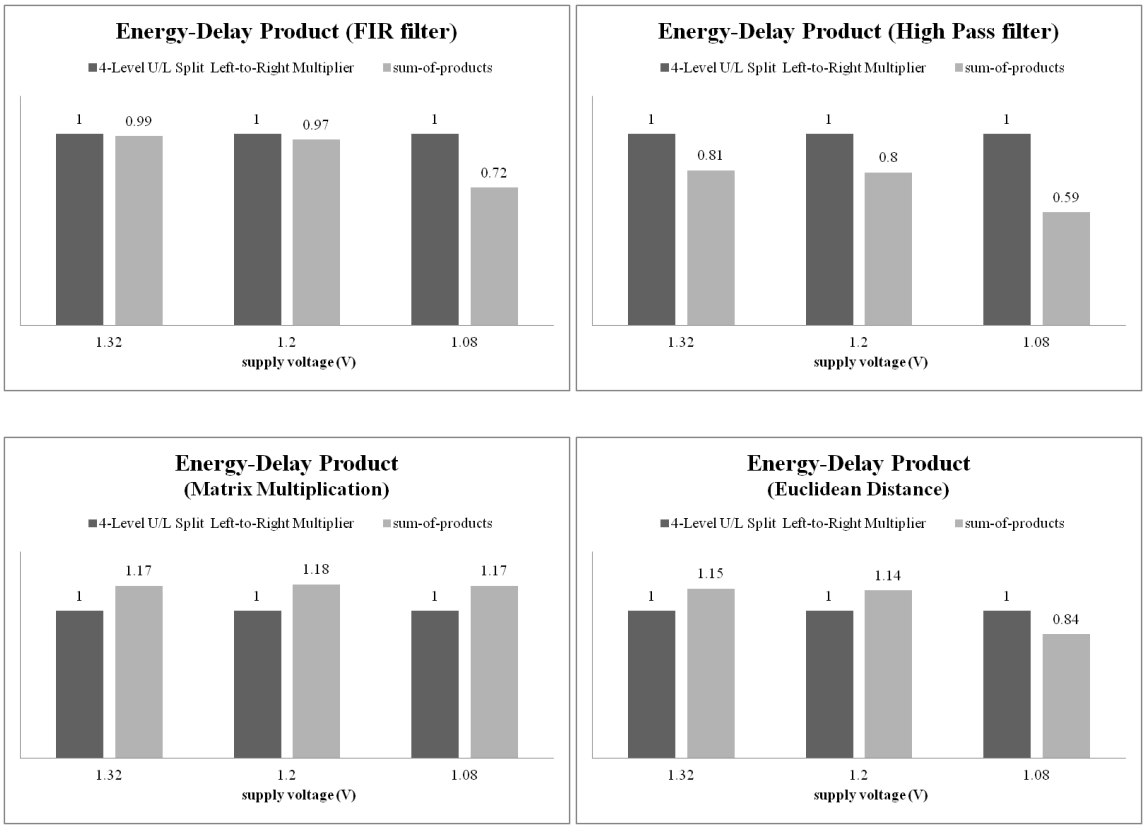


FIGURE 4.6: ENERGY-DELAY PRODUCT COMPARISON BETWEEN 4-LEVEL UL SPLIT LR MULTIPLIERS AND SUM-OF-PRODUCTS UNITS

4.4 Summary

In this chapter, we discussed a new sum-of-products arithmetic unit and presented experimental results about its performance and power. We have utilized parallelism in organization of multipliers. Compared to the ARM7TDMI-S multiplier, the sum-of-products can reduce execution time by approximately 40% with 25% of energy increase in benchmark applications. The proposed designs were synthesized with three supply voltages 1.08, 1.20 and 1.32V supported by Samsung 65nm process technology, and then we measured power and delay of a multiplier and a sum-of-products unit. The clock cycles for benchmark programs were measured by running cycle-level simulation tool using the ARM7TDMI-S Verilog code and compiled ARM assembly code. We calculated energy and execution time in four benchmark programs: FIR filter, high pass filter and Euclidean distance programs. The sum-of-products unit would consume less energy than a single multiplier if the execution time of a sum-of-products unit is the same as that of a single multiplier. Parallel organization can reduce execution time and run at a lower supply voltage, which can reduce power consumption for dynamic power compared to a single solution.

We have demonstrated that the proposed sum-of-products design reduces energy compared with a single multiplier when computing sum-of-products. The point of this design is to compare execution time and energy between parallelism in organization of two multipliers and a single solution with a multiplier. However, we have only compared arithmetic components, not complete processors. Using a sum-of-products arithmetic unit instead of a single multiplier would require register files with four output ports (or five output ports in case of a solution with an accumulator). This would increase the fan-out of the register files and add delays in the registers address decoding if four (or five) operands instead of two are fetched at each clock cycle. The proposed arithmetic components would probably slow down the complete processor and all other parts of the applications. The analysis of complete processor architecture would

provide more accurate results. The implementation and analysis of complete processor architecture are left to future work. Also, four signal processing applications might not be enough to justify the advantage of sum-of-products designs. We have considered a few common applications such as filters (FIR and high-pass), matrix multiplication and Euclidean distance. It is likely that other applications, such as Fast Fourier transform (FFT), polynomial evaluation and Lower Upper (LU) decomposition would benefit from the proposed sum-of-products unit.

Chapter 5 Multi-functional Arithmetic Unit based on Sum-of-Products

Recent digital signal processing applications require many arithmetic operations; as a result modern DSPs and GPUs include separate arithmetic units for supporting each arithmetic operation. This implementation leads to large power and area overhead. Thus, we need to develop a sum-of-products unit capable of supporting several arithmetic operations using essentially the same hardware with input controls.

This chapter presents designs for a Multi-functional Arithmetic Unit based on Sum-of-Products (MAU-SoP) that implements a variety of arithmetic operations. The MAU-SoP can perform a sum-of-products, a multiplication, a multiply-add, a square, a sum-of-squares or an add-multiply computation based on an input control signal. Compared to a conventional sum-of-products unit, the proposed unit has a modest increase in area and delay, due to a modest amount of additional control logic, but allows multiplication-related arithmetic operations to be performed efficiently. The experimental results indicate that a MAU-SoP for 32-bit two's complement operands is implemented with approximately 8% more power, 6% more area and nearly the same worst case delay as the sum-of-products unit proposed in Chapter 4.

5.1 Introduction

Multiplication and related arithmetic operations are found in many digital signal processing applications including filtering, pattern recognition and vector computation. Previous studies on arithmetic operations have mainly focused on designs for dedicated arithmetic hardware, which compute only a single arithmetic operation. These dedicated arithmetic units work well for

nonprogrammable DSP designs in which the relative frequency of each arithmetic operation is known in advance. However they are less suitable for recent programmable DSP, in which the frequency of arithmetic operations is application dependent. For applications that do not frequently perform the specific arithmetic operations, the extra arithmetic units needed for dedicated arithmetic operations goes unused if separate arithmetic units are implemented for supporting each arithmetic operation. Therefore, we need to design a multi-functional arithmetic unit to provide flexibility.

In this chapter, we design a sum-of-products unit capable of supporting several arithmetic operations using essentially the same hardware. Specifically, the use of a sum-of-products operation can allow a general operation to more easily transform the other operations by changing parameters. The remainder of this paper is organized as follows. Section 5.2 presents the overall structure of the MAU-SoP. Section 5.3 describes designs for a MAU-SoP that executes each operation based on a sum-of-products unit. Section 5.4 provides power, delay and area results for the MAU-SoP and compares them to estimates for a conventional sum-of-products unit. Section 5.5 gives conclusions. The designs presented in this chapter are based on the sum-of-products unit proposed in Chapter 4. We assume the input operands are 32-bit integer operands, but they can easily be extended to other types of fixed-point operands.

5.2 MAU-SoP Structure

We describe here how to support several arithmetic operations using the same hardware. Note that the sum-of-products operation $s = a \times b + x \times y$ is a baseline operation. This corresponds to several different arithmetic operations by setting variables accordingly. Table 5.1 summarizes arithmetic operations with corresponding conditions.

TABLE 5.1: OPERATION MODE

Opcode	Operation Mode	Expression	Condition
SOP	Sum-of-products	$a \times b + x \times y$	Baseline
M	Multiplication	$a \times b$	$(x = 0)$ or $(y = 0)$
MA	Multiply-add	$a \times b + x$	$y = 1$
SS	Sum-of-squares	$a^2 + x^2$	$(a = b)$ and $(x = y)$
S	Square	a^2	$(a = b)$ and $((x = 0)$ or $(y = 0))$
AM	Add-multiply	$a \times (b + y)$	$a = x$

5.2.1 The Opcode Decoder

The sum-of-products operation is executed by using two multiplier arrays, [4:2] adders and the final CPA, but the other operations can be executed using less hardware where operations are known in advance. To use less hardware, we need to add an opcode decoder and a MUX, and modify multiplier arrays.

The decoder can detect all operands and then determine the operation mode. Once operation is determined, it can disable blocks unused using signal gating. The gated signals are generated based on the control signals and are combined with an AND gate in parallel. Finally it selects appropriate value for the final result. Figure 5.1 shows the proposed sum-of-products with the opcode decoder and the MUX. All opcodes and control signals are summarized in Table 5.2.

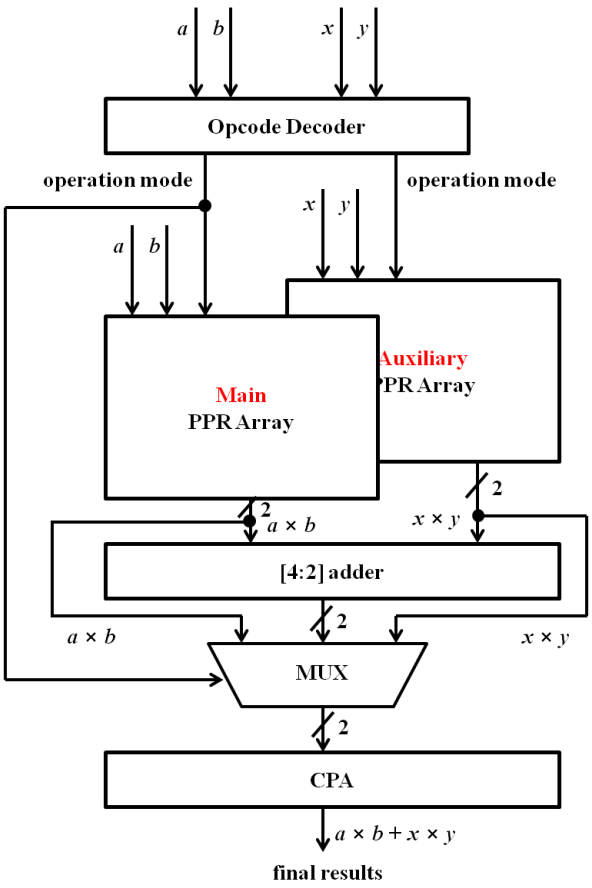


FIGURE 5.1: THE PROPOSED MAU-SOP STRUCTURE

TABLE 5.2: CONTROL SELECTION OF THE DESIGN

Input	Output		
Operation Mode	Turn-on Modules	Turn-off Modules	MUX Selection
Sum-of-products	Two multiplier arrays, [4:2] adder, CPA	None	[4:2] adder
Multiplication	One multiplier array, CPA	One multiplier array, [4:2] adder	Turn-on multiplier array
Multiply-add	One multiplier array, [4:2] adder, CPA	One multiplier array	[4:2] adder
Sum-of-squares	One multiplier array, CPA	One multiplier array, [4:2] adder	Turn-on multiplier array
Square	One multiplier array, CPA	One multiplier array, [4:2] adder	Turn-on multiplier array
Add-multiply	One multiplier array, CPA	One multiplier array, [4:2] adder	Turn-on multiplier array

5.2.2 The Heterogeneous Sum-of-products Unit

Multiplication and multiply-add operations are the most frequently used arithmetic operation; thus, such operations should have higher performance and lower power compared with other operations. The other arithmetic operations are not relatively frequently used, so they should have flexible structures for effective sharing of structure.

We determine the optimal partition, which is a critical problem in low-power design because most operations do not use all modules of a sum-of-products unit. The sum-of-products unit includes two multiplier arrays: a main multiplier array and an auxiliary multiplier array. We use two multiplier arrays independently. Because multiplication and multiply-add operations use only a single multiplier array, we would not add extra gates. This is effective for high speed and low power due to the relatively simpler structure. This is called a main multiplier array. On the other hand, the other operations can be executed by using the modified array. The extra gates are inserted for effective sharing of structure. This modified multiplier array is called an auxiliary multiplier array. This structure is relatively slower, but square, sum-of-squares and add-multiply operations can be executed using less hardware. Compared to the delay of the sum-of-products unit proposed in, Chapter 4, the delay of the proposed sum-of-products operation will be increased because the delay of an auxiliary multiplier array is increased. Table 5.3 describes operating units depending on arithmetic operation.

TABLE 5.3: OPERATING UNITS BASED ON ARITHMETIC OPERATIONS

Operation	Components used in operation
Sum-of-products	Main multiplier array, Auxiliary multiplier array, [4:2] adder, CPA
Multiplication	Main multiplier array, CPA
Multiply-add	Main multiplier array, [4:2] adder, CPA
Sum-of-squares	Auxiliary multiplier array, CPA
Square	Auxiliary multiplier array, CPA
Add-multiply	Auxiliary multiplier array, CPA

5.3 Arithmetic Operations

In this section, we present each arithmetic operation. All designs are theoretically analyzed. For theoretical analysis, the delay of a 2-input XOR gate, T_{XOR2}, is used as the base unit delay. We assume all estimates do not include buffers. The area of inverting logic is not also included as we assume logic polarities are optimized in the actual implementations. Multiplier arrays analyzed here assume the radix-2 non-split LR array. We use area and delay estimates of this structure which have been already analyzed in Chapter 2.

5.3.1 Sum-of-products

When the opcode is SOP (sum-of-products), a sum-of-products is executed by using two multiplier arrays, [4:2] adders and the CPA, as shown in Figure 5.2.

5.3.2 Multiplication

When the opcode is M (multiplication), a multiplication is performed by using a main multiplier array and the CPA, as shown in Figure 5.3. The auxiliary multiplier array and [4:2] adders are turned off using signal gating techniques. The final results are selected by using multiplexers controlled by the opcode decoder.

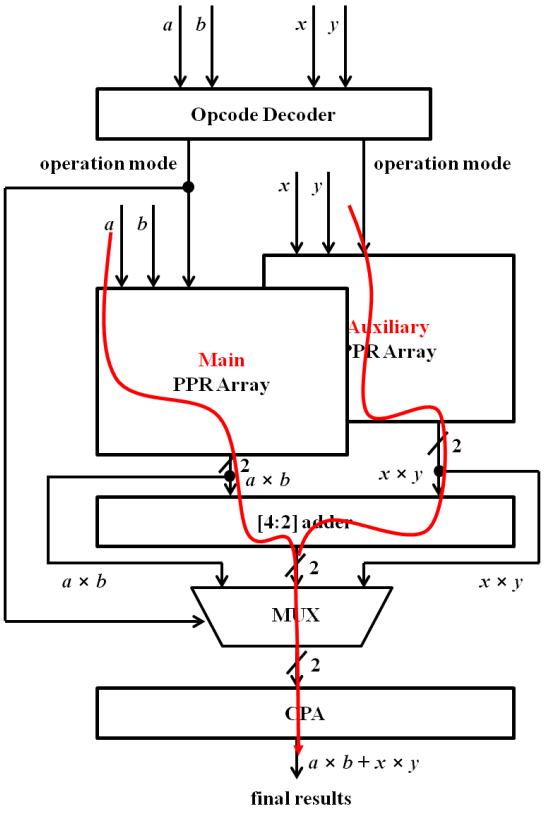


FIGURE 5.2: SUM-OF-PRODUCTS OPERATION

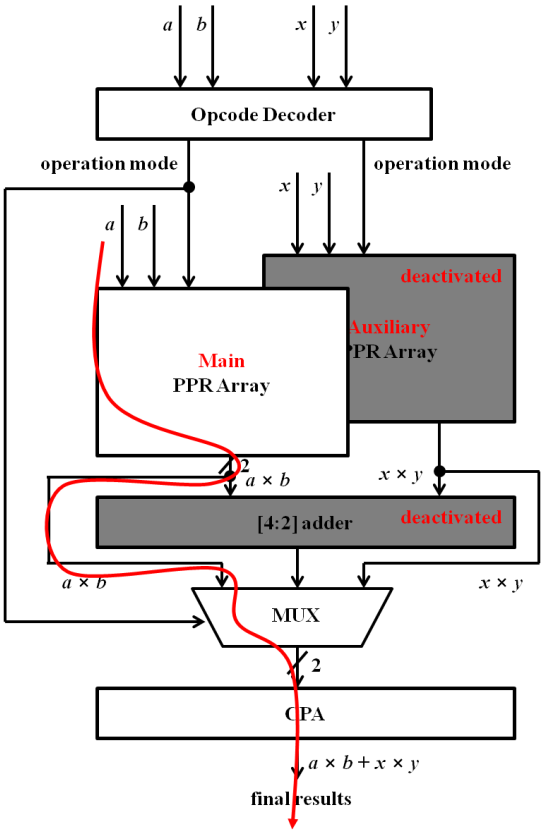


FIGURE 5.3: MULTIPLICATION OPERATION

5.3.3 Multiply-add

When the opcode is MA (multiply-add operation), a main multiplier array, [4:2] adders and the final adder are used while an auxiliary multiplier is deactivated. Two carry-save redundant outputs from one multiplier and one input bypassed are summed using [4:2] adder and values are selected by using multiplexers controlled by the opcode decoder. Finally, the CPA generates the final product. This multiply-add operation has less power dissipation than the baseline sum-of-products because it avoids the execution of one multiplication. One example is multiply-add operation as shown in Figure 5.4.

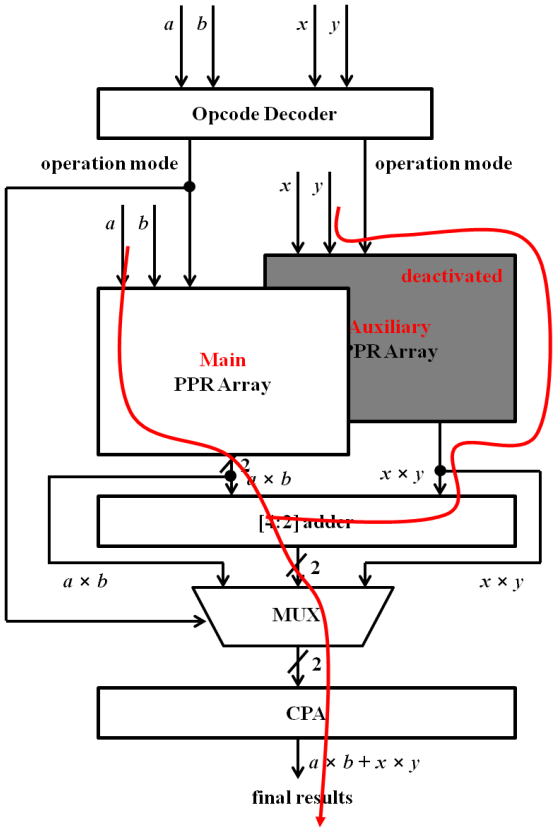


FIGURE 5.4: MULTIPLY-ADD OPERATION

5.3.4 Sum-of-squares

Fundamentally, a sum-of-squares operation is executed using two multiplier arrays, [4:2] adders and the final CPA. However, this design is not efficient in power, area and delay. An efficient way to compute a sum-of-squares is to use the CMSSU technique [94]. A sum-of-squares operation can be executed using a single modified multiplier array and a CPA. First, we examine the difference between sum-of-squares and multiplication matrices. As shown in Figure 5.5, the sum-of-squares matrix has an additional row with 32 × XOR2 gates. It is also shifted left by one bit position relative to the multiplication matrix. Each PP bit a_ib_j above the anti-diagonal in multiplication matrix is replaced by a_ia_j in the sum-of-squares matrix. Each PP bit a_jb_i below the anti-diagonal in the multiplication matrix is replaced by b_jb_i in the sum-of-squares matrix. Multiplication and sum-of-squares matrices can be combined based on these observations. To merge the multiplication and sum-of-squares array, several extra gates are inserted into the original multiplier array. The area and delay increase of extra gates is estimated as

$$\text{Area} = 64 \times A_{MUX21} + 32 \times (A_{XOR2} + A_{AND2} + A_{FA}) + A_{MUX6432}$$

$$\text{Delay} = T_{XOR2}$$

Compared to a single original multiplier matrix, the combined structure has a slight increase in area and delay. However, compared to the fundamental schemes using two multiplier arrays, [4:2] adders and the final CPA, the combined structure can achieve significant power and delay savings. Considering unused modules, the area and delay savings are estimated as

$$\text{Area} = A_{PPG} + 960 \times A_{FA} + 480 \times A_{[4:2]ADDER} - 64 \times A_{MUX21} - 32 \times (A_{XOR2} + A_{AND2} + A_{FA}) - A_{MUX6432}$$

$$\text{Delay} = T_{XOR2}$$

The combined unit performs either multiplication or sum-of-squares operations using essentially the same hardware, a single modified multiplier array, based on input control signals. One example is the sum-of-squares operation as shown in Figure 5.10.

5.3.5 Square

The standard multiplier can be used for computing square operation. However it is not an efficient way to compute a square operation because all the hardware resources are used for the square operation. A different method is to use a dedicated implementation for square [40][41]. A matrix for square consists of the diagonal with entries $a_i a_i = a_i$ and two regions: *A* below the diagonal and *B* above the diagonal, as shown in Figure 5.6(c). This matrix can be considerably simplified before performing multi-operand addition. We can obtain this matrix for square using the matrix for sum-of-squares. Only one region (*A* below the diagonal or *B* above the diagonal) of the matrix for sum-of-squares is used to compute square, while the other region is deactivated. The sum-of-squares matrix requires additional XOR gates and FAs. For square computation, the XOR gates should be removed; then multiplicand bits (a_0, a_1, a_2, a_3, .., a_{31}) are directly connected to the inputs of FAs. '1's in the MSB and 16-bit are shifted right by one bit position relative to the sum-of-squares matrix. The area and delay increase are very small because of the shared structure. Compared to the sum-of-squares matrix, the square matrix has an approximately 50% decrease in delay and area used because it uses only one region of the two regions that make up a sum-of-squares matrix. To merge the multiplication and square

array, several extra gates are inserted into the original multiplier array. The area increase of extra gates is estimated as

$$\text{Area} = 64 \times A_{MUX21} + 32 \times (A_{AND2} + A_{FA}) + A_{MUX6432}$$

Compared to a single original multiplier matrix, the combined structure has a slight increase in area. However, compared to the fundamental schemes using two multiplier arrays, [4:2] adders and the final CPA, the combined structure can achieve significant power and delay savings. Considering unused modules, the area saving is estimated as

$$\text{Area} = A_{PPG} + 720 \times A_{FA} + 480 \times A_{[4:2]ADDER} - 64 \times A_{MUX21} - 32 \times (A_{AND2} + A_{FA}) - A_{MUX6432}$$

$$
\begin{array}{cccccccccccccc}
1 & a_7b_7 & (a_8b_7)' & (a_5b_7)' & (a_4b_7)' & (a_3b_7)' & (a_2b_7)' & (a_1b_7)' & (a_0b_7)' & & & & & \\
 & (a_7b_6)' & a_6b_6 & a_5b_6 & a_4b_6 & a_3b_6 & a_2b_6 & a_1b_6 & a_0b_6 & & & & & \\
 & & (a_7b_5)' & a_6b_5 & a_5b_5 & a_4b_5 & a_3b_5 & a_2b_5 & a_1b_5 & a_0b_5 & & & & \\
 & & & (a_7b_4)' & a_6b_4 & a_5b_4 & a_4b_4 & a_3b_4 & a_2b_4 & a_1b_4 & a_0b_4 & & & \\
 & & & & (a_7b_3)' & a_6b_3 & a_5b_3 & a_4b_3 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & \\
 & & & & & (a_7b_2)' & a_6b_2 & a_5b_2 & a_4b_2 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & \\
 & & & & & & (a_7b_1)' & a_6b_1 & a_5b_1 & a_4b_1 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 \\
 & & & & & & & 1 & (a_7b_0)' & a_6b_0 & a_5b_0 & a_4b_0 & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0
\end{array}
$$

(A) MULTIPLICATION MATRIX

(B) MATRIX FOR SUM-OF-SQUARES

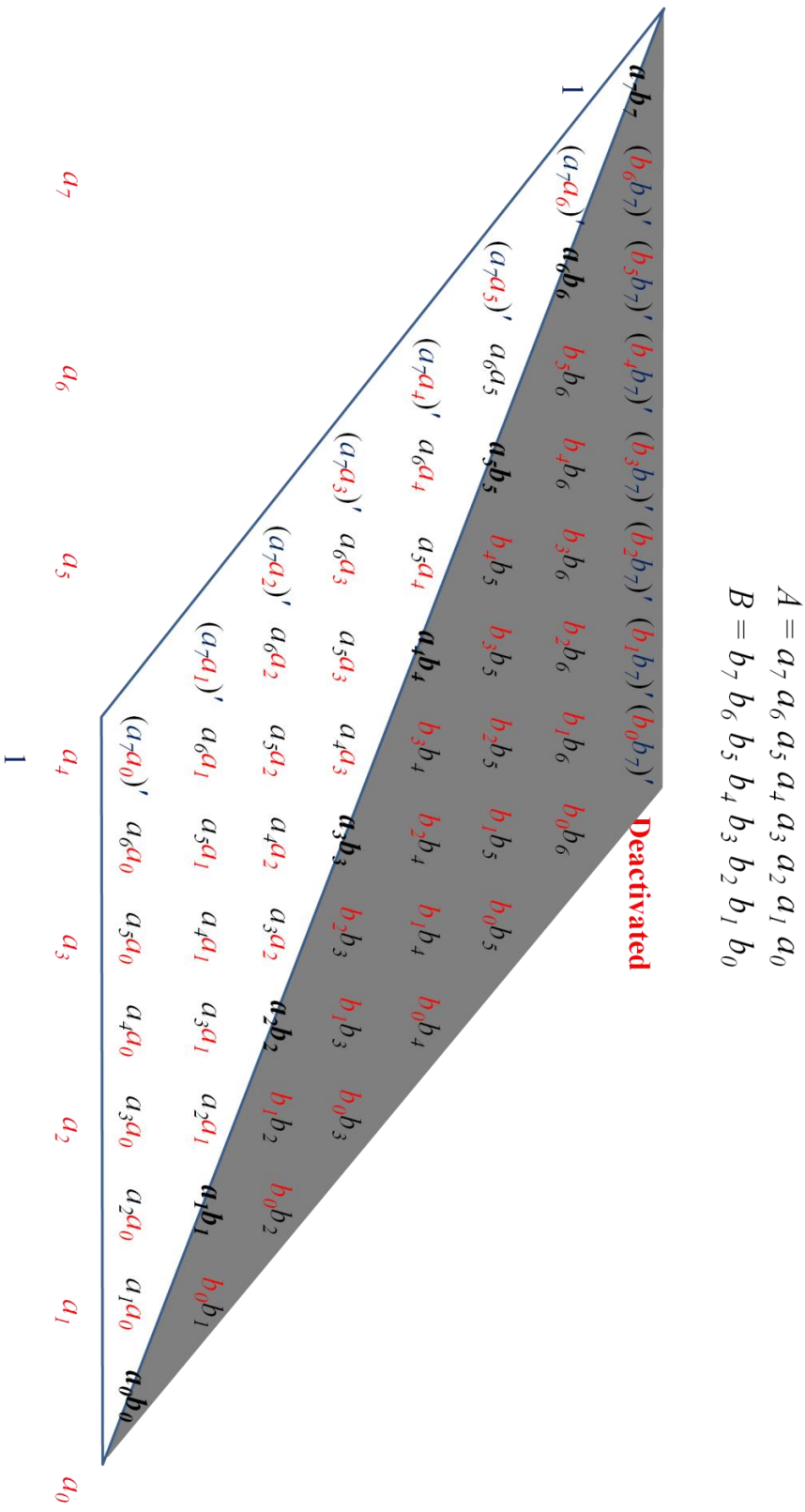
FIGURE 5.5: 8-BIT TWO'S COMPLEMENT SIGNED MATRIX EXAMPLE FOR MULTIPLICATION AND SUM-OF-SQUARES (ADAPTED FROM [94])

$$
\begin{array}{cccccccccccccccccc}
1 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & & & & & & & & & \\
 & (a_7a_6) & (a_7a_5) & (a_7a_4) & (a_7a_3) & (a_7a_2) & (a_7a_1) & (a_7a_0) & a_6a_0 & a_5a_0 & a_4a_0 & a_3a_0 & a_2a_0 & a_1a_0 & a_0 & & & \\
 & & & & & & & & a_6a_1 & a_5a_1 & a_4a_1 & a_3a_1 & a_2a_1 & & & & & \\
 & & & & & & & & a_6a_2 & a_5a_2 & a_4a_2 & a_3a_2 & & & & & & \\
 & & & & & & & & a_6a_3 & a_5a_3 & a_4a_3 & & & & & & & \\
 & & & & & & & & a_6a_4 & a_5a_4 & & & & & & & & \\
 & & & & & & & & a_6a_5 & & & & & & & & &
\end{array}
$$

(A) MATRIX FOR SQUARE (A^2)

$$
\begin{array}{cccccccccccccc}
1 & b_7 & 0 & b_6 & 0 & b_5 & 0 & b_4 & 0 & b_3 & 0 & b_2 & 0 & b_1 & 0 & b_0 \\
 & (b_7b_6)' & (b_7b_5)' & (b_7b_4)' & (b_7b_3)' & (b_7b_2)' & (b_7b_1)' & (b_7b_0)' & b_6b_0 & b_5b_0 & b_4b_0 & b_3b_0 & b_2b_0 & b_1b_0 & & \\
 & & & & & & & b_6b_5 & b_6b_4 & b_6b_3 & b_6b_2 & b_6b_1 & b_5b_1 & b_4b_1 & b_3b_1 & b_2b_1 \\
 & & & & & & & & b_5b_4 & b_5b_3 & b_5b_2 & b_4b_2 & b_3b_2 & & & \\
 & & & & & & & & & b_4b_3 & & & & & &
\end{array}
$$

(B) MATRIX FOR SQUARE (B^2)



(C) MATRIX FOR SQUARE BASED ON SUM-OF-SQUARES ($A^2 + B^2$)

FIGURE 5.6: 8-BIT TWO'S COMPLEMENT SIGNED MATRIX EXAMPLE FOR SQUARE OPERATION

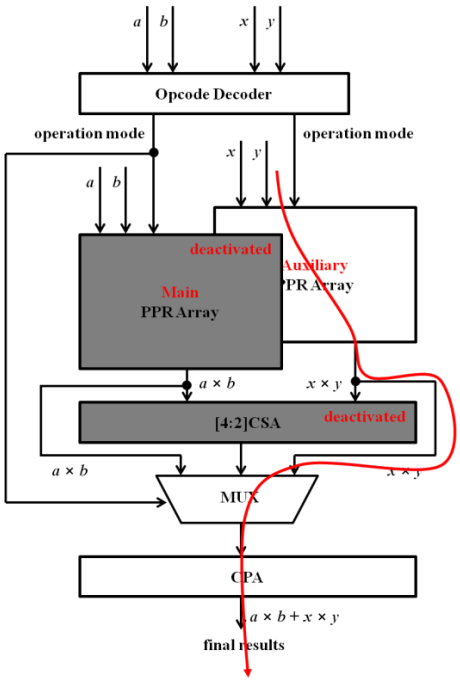


FIGURE 5.7: SUM-OF-SQUARES OPERATION

5.3.6 Add-multiply

Fundamentally, an add-multiply operation is executed using two multiplier arrays, [4:2] adders and the final CPA. However, it can be executed by a single modified multiplier array and the final CPA if an extra adder is inserted into a sum-of-products unit. First, we examine the mathematical property. The sum-of-products of the forms $s = a \times b + x \times y$ can be transformed to add-multiply of the form $s = a \times (b + y)$, where $a = x$. Basically PPG modules have two-input AND gates for representing the PP bit. If we perform arithmetic addition before generating PP bits, an add-multiply operation can be executed by using a single modified multiplier array and the final CPA instead of using all the hardware resources of a sum-of-products unit. This adder is called a PPG adder. When an add-multiply operation is selected, the PPG adder is activated, and then the output of the PPG adder is connected with the inputs of the AND gates in PPG modules. In this case, one PPG and one multiplier array unused are deactivated. When other operations are selected, the PPG adder is deactivated, and remaining modules are activated.

To minimize the delay of an add-multiply operation, we assume that the fastest adder is implemented for the PPG adder. Here we use a CLA as the PPG adder. To avoid a large number of gates with a large number of inputs, we select a one-level CLA. The area and delay increase of a 32-bit one-level CLA based on a 4-bit adder block is estimated as

$$\text{Area} = 8 \times (8 \times A_{AND2} + 5 \times A_{OR2} + 3 \times A_{AND3} + 2 \times A_{AND4} + 2 \times A_{OR4})$$
$$\text{Delay} = T_{AND2} + 8 \times (T_{AND2} + T_{OR2}) + T_{XOR2}$$

We simplify these estimates using Table 3.1

$$\text{Area} = 61.2 \times A_{XOR2}$$
$$\text{Delay} = 9.5 \times T_{XOR2}$$

Compared to the original multiplier array, the proposed structure has a slight increase in delay and area. However, compared to the basic add-multiply operation using two multiplier arrays, [4:2] adders and the final CPA, the proposed operation can achieve significant area saving with slight delay penalty. Considering all modules unused and inserted, the area saving and delay increase are estimated as

$$\text{Area} = A_{PPG} + 1088 \times A_{FA} - 61.2 \times A_{XOR2} \text{ (decrease)}$$
$$\text{Delay} = 6.5 \times t_{XOR2} \text{ (increase)}$$

The total area used is decreased because the area of one PPG module, one array and [4:2] adders unused is larger than that of a PPG adder inserted. The total delay increase is because

the delay increase of the PPG adder is larger than the delay reduction of [4:2] adders. However, the PPG adder would probably affect the power dissipation in the subsequent array and CPA. It is at the first stage on the long path in a sum-of-products unit; hence, it would probably introduce extra unbalanced signal transitions. Additional buffers would also be needed to handle the large fan-out for practical designs.

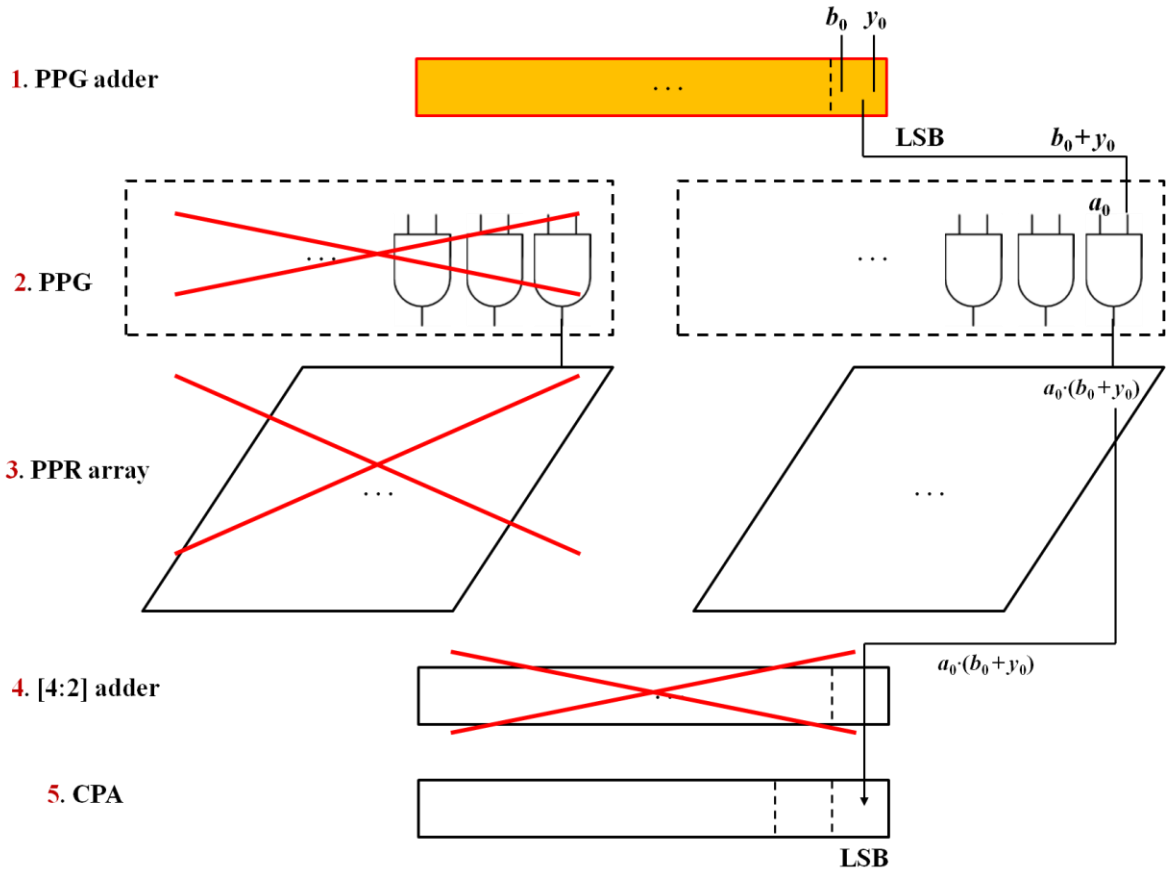


FIGURE 5.8: ADD-MULTIPLY OPERATION

5.3.7 Overall Execution

The combined structure has the original sum-of-products unit with additional gates. The area and delay increases to be estimated are shown in Table 5.4. Based on these observations, the critical path delay would increase by approximately 10 unit delays and the additional

buffering delay. Gates inserted would probably increase the total wire delay because routing becomes more complex.

TABLE 5.4: AREA AND DELAY INCREASE OF ADDED MODULES

Operation	Area	Delay
Sum-of-squares	$64 \times A_{MUX21} + 32 \times (A_{XOR2} + A_{AND21} + A_{FA}) + A_{MUX6432}$	T_{XOR2}
Square	$64 \times A_{MUX21} + 32 \times (A_{AND21} + A_{FA}) + A_{MUX6432}$	0
Add-multiply	$61.2 \times A_{XOR2}$	$9.5 \times T_{XOR2}$
Total	$128 \times A_{MUX21} + 93.2 \times A_{XOR2} + 64 \times (A_{AND21} + A_{FA}) + 2 \times A_{MUX6432}$	$10.5 \times T_{XOR2}$

In a conventional structure, all operations are executed by using all resources of a sum-of-products hardware, because no operands are detected. However, the proposed unit can deactivate several modules when operands meet specific conditions. An area and logic delay comparison between a conventional structure and the proposed one to be estimated is shown in Table 5.5.

TABLE 5.5: AREA AND DELAY IN EACH OPERATION

Operation	The original operations	
	Area	Delay
All operations	$2 \times A_{PPG} + 2048 \times A_{FA} + A_{CPA}$	$T_{AND2} + 48 \times T_{XOR2} + T_{CPA}$

Operation	The proposed operations	
	Area	Delay
Multiply	$A_{DCT} + A_{PPG} + 960 \times A_{FA} + 2 \times A_{MUX6432} + A_{CPA}$	$T_{DCT} + T_{AND2}$ $+ 45 \times T_{XOR2} + T_{MUX21} + T_{CPA}$
Multiply-add	$A_{DCT} + A_{PPG} + 1088 \times A_{FA} + 2 \times A_{MUX6432} + A_{CPA}$	$T_{DCT} + T_{AND2} + 48 \times T_{XOR2} +$ $T_{MUX21} + T_{CPA}$
Sum-of-squares	$A_{DCT} + A_{PPG} + 960 \times A_{FA} + 64 \times A_{MUX21}$ $+ 32 \times (A_{XOR21} + A_{AND21} + A_{FA}) + 3 \times A_{MUX6432} + A_{CPA}$	$T_{DCT} + T_{AND2}$ $+ 46 \times T_{XOR2} + T_{MUX21} + T_{CPA}$
Square	$A_{DCT} + A_{PPG} + 480 \times A_{FA} + 64 \times A_{MUX21}$ $+ 32 \times (A_{AND21} + A_{FA}) + 3 \times A_{MUX6432} + A_{CPA}$	$T_{DCT} + T_{AND2}$ $+ 45 \times T_{XOR2} + T_{MUX21} + T_{CPA}$
Add-multiply	$A_{DCT} + A_{PPG} + 61.2 \times A_{XOR2} + 960 \times A_{FA}$ $+ 2 \times A_{MUX6432} + A_{CPA}$	$T_{DCT} + T_{AND2} + 54.5 \times T_{XOR2}$ $+ T_{MUX21} + T_{CPA}$
Sum-of-products	$A_{DCT} + 2 \times A_{PPG} + 2048 \times A_{FA} + 2 \times A_{MUX6432} + A_{CPA}$	$T_{DCT} + T_{AND2} + 48 \times T_{XOR2}$ $+ T_{MUX21} + T_{CPA}$

Array: radix-2 non-split LR array

1. Multiply: (original operation) two PPGs + two multiplier arrays + CPA

   (proposed operation) a detector + a single PPG + a main multiplier array + MUX + CPA

2. Multiply-add: (original operation) two PPGs + two multiplier arrays + CPA

   (proposed operation) a detector + a single PPG + a main multiplier array + [4:2] adder + MUX + CPA

3. Square: (original operation) two PPGs + two multiplier arrays + CPA

   (proposed operation) a detector + a single PPG + an auxiliary multiplier array + MUX + CPA

4. Sum-of-squares: (original operation) two PPGs + two multiplier arrays + CPA

   (proposed operation) a detector + a single PPG + an auxiliary multiplier array + MUX + CPA

5. Add-multiply: (original operation) two PPGs + two multiplier arrays + CPA

   (proposed operation) a detector + a single PPG + the first adder + a main multiplier array+ MUX + CPA

6. Sum-of-products: (original operation) two PPGs + two multiplier arrays + CPA

   (proposed operation) a detector + two PPGs + two multiplier arrays + MUX + CPA

The cell areas are measured in the NAND2 gate based on the Samsung standard cell library. We assume a Detector (DCT) consists of 4 × 63 (= 32 + 16 + 8 + 4 + 2 + 1) XOR2 gates, a PPG consists of 32 × 32 AND2 gates and MUX6432 consists of 32 × MUX21 gates. The delay of DCT, PPG and MUX6432 is estimated roughly as equivalent to 4 × T_{XOR2}, T_{AND2} and T_{MUX21}, respectively. We use the delay and area estimates of the final CPA that have already been analyzed in Chapter 3. The delay of the final CPA is estimated roughly as equivalent 23 × T_{XOR2}. A comparison of the delay and area estimates is shown in Table 5.6 and Figure 5.9. Area indicates the amount of gate used. The multiplication operation has 45% less area used with 7% more delay, the multiply-add operation has 40% less area used with 12% more delay, and the sum-of-squares operation has 47% less area used with 9% more delay than the baseline sum-of-products structure. The square operation has 60% less area used with 7% more delay. The delay increase of an add-multiply operation is the largest, due to the delay of a PPG adder. The sum-of-products operation with the proposed structure increase area and delay because an opcode decoder and MUXs are inserted into the original sum-of-products unit. It is not easy to estimate the current power reduction of the proposed structure. However, the proposed design can eliminate significant switching activities because several modules are deactivated. All things considered, the proposed structure would consume less power.

TABLE 5.6: AREA AND DELAY COMPARISON BETWEEN THE ORIGINAL AND THE PROPOSED OPERATIONS

Operation	The original operations				The proposed operations			
	Used Area (NAND2)		Delay (T_{XOR})		Used Area (NAND2)		Delay (T_{XOR})	
Multiply	11870	1.00	68.3	1.00	6529	0.55	73.3	1.07
Multiply-add					7122	0.60	76.3	1.12
Sum-of-squares					6292	0.53	74.3	1.09
Square					4820	0.40	73.3	1.07
Add-multiply					6528	0.55	82.8	1.21
Sum-of-products					12344	1.04	76.3	1.12

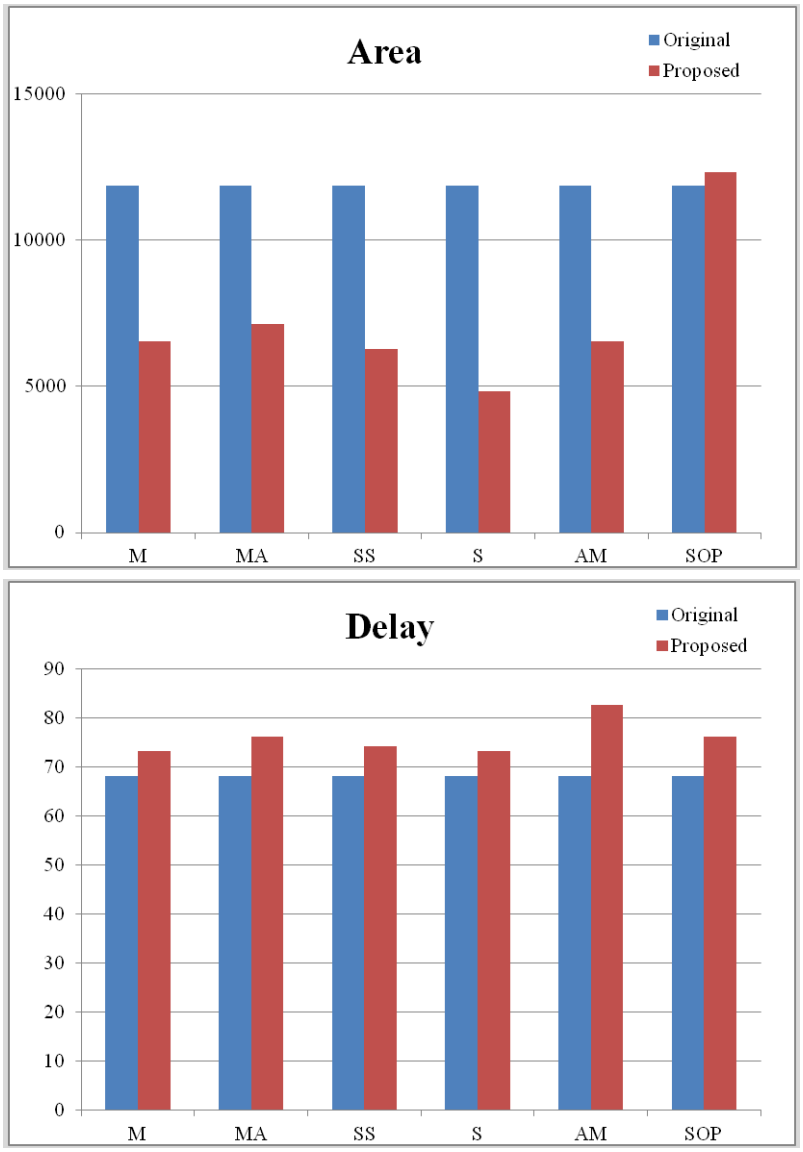


FIGURE 5.9: AREA AND DELAY COMPARISON BETWEEN THE ORIGINAL AND THE PROPOSED OPERATIONS

5.4 Experimental Evaluation

To obtain power, area and delay results, we have implemented structural Verilog models for MAU-SoP, given an operand size of 32-bit, and operand type of two's complement. When comparing the relative benefits of different designs, the structural model reduces the changes made by the synthesis tool. The detailed design and simulation methodologies are described in Appendix A. As our major focus is on the overall structure, we have reused the detailed modules

proposed in Chapter 2 and Chapter 3, and they have not been optimized for different low-power techniques. They have been synthesized using the Synopsys Design Compiler with Samsung 65nm CMOS standard cell low-power library. The designs have been optimized for area with a maximum fan-out of four. The simulation results used an operating voltage of 1.08 volts and a temperature of 125 degrees centigrade. Table 5.7 shows the comparison of power, delay and area estimates for two sum-of-products units. One is proposed in Chapter 4. This does not include the opcode decoder, MUX and extra gates for supporting multi-function. The other is proposed in this chapter. To facilitate comparisons, the percent increase and decrease in power, delay and area between the original and the proposed designs for a given operand size are also shown. Based on these estimates, the delay of the proposed structure is close to that of the original one. This result implies that adding extra gates for multi-functional arithmetic operations doesn't seem to have much effect on delay. Based on a theoretical analysis of the worst case delay paths, we expect the worst case delay of the proposed sum-of-products unit to be about 12% more than that of the original one. The experimental results have 9% less delay than we expected because the ability of the Synopsys synthesis tool makes tradeoffs between area and delay automatically. However, the proposed design has 9% power and 6% area increase. The power increase is slightly larger than the area increase due to more glitches from the PPG adder.

TABLE 5.7: POWER, DELAY AND AREA FOR THE ORIGINAL AND THE PROPOSED SUM-OF-PRODUCTS UNITS

Multiplier	Power (mW)		Delay (ns)		Area (μm^2)	
Original sum-of-products	12.07	1.00	13.36	1.00	23415	1.00
Proposed sum-of-products	13.12	1.09	13.78	1.03	24754	1.06

In Table 5.8, detailed experimental results are given to compare the power, area, and delay characteristics of each component. Because an auxiliary multiplier array includes extra gates, it has more power, delay and area than a main multiplier array.

TABLE 5.8: POWER, DELAY AND AREA FOR THE PROPOSED SUM-OF-PRODUCTS UNIT

Components	Power (mW)		Delay (ns)		Area (μm^2)	
Sum-of-Products (MAU-SoP)	13.12	1.00	13.78	1.00	24754	1.00
Opcode Decoder	0.43	0.03	0.36	0.03	3348	0.13
First Adder	0.21	0.02	1.08	0.08	870	0.04
Main Multiplier	5.78	0.44	7.33	0.53	9308	0.38
Auxiliary Multiplier	6.12	0.47	7.62	0.55	9982	0.40
MUX	0.23	0.02	0.25	0.02	192	0.01
[4:2] adder, CPA	0.35	0.03	4.47	0.32	1054	0.04

Table 5.9 shows power, delay and area estimates for each operation. The multiplication has 37% less power and 31% less power-delay with 9% delay increase. The multiply-add operation has 32% less power, and 22% less power-delay product with 15% delay increase. The sum-of-squares operation has 42% power and 34% power-delay product reduction with 14% delay increase. The square operation has 52% power and 46% power-delay product reduction with 13% delay increase, and the add-multiply operation has 35% power and 15% power-delay decrease with 30% delay increase. Compared to the original structure, the sum-of-products operations have 10% power and 18% delay increase. Compared to the theoretical models, the experimental results demonstrated a greater area increase because theoretical results do not include buffering and conditionally inverting some bits.

TABLE 5.9: POWER, DELAY AND AREA FOR EACH OPERATION

Operation		Power (mW)		Delay (ns)		Power-Delay Product (nJ)	
Original operation		12.07	1.00	13.56	1.00	163.67	1.00
Proposed operation	Multiply	7.60	0.63	14.78	1.09	112.39	0.69
	Multiply-add	8.21	0.68	15.60	1.15	127.99	0.78
	Sum-of-squares	7.00	0.58	15.46	1.14	108.22	0.66
	Square	5.79	0.48	15.32	1.13	88.72	0.54
	Add-multiply	7.85	0.65	17.63	1.30	138.30	0.85
	Sum-of-products	13.28	1.10	16.01	1.18	212.44	1.30

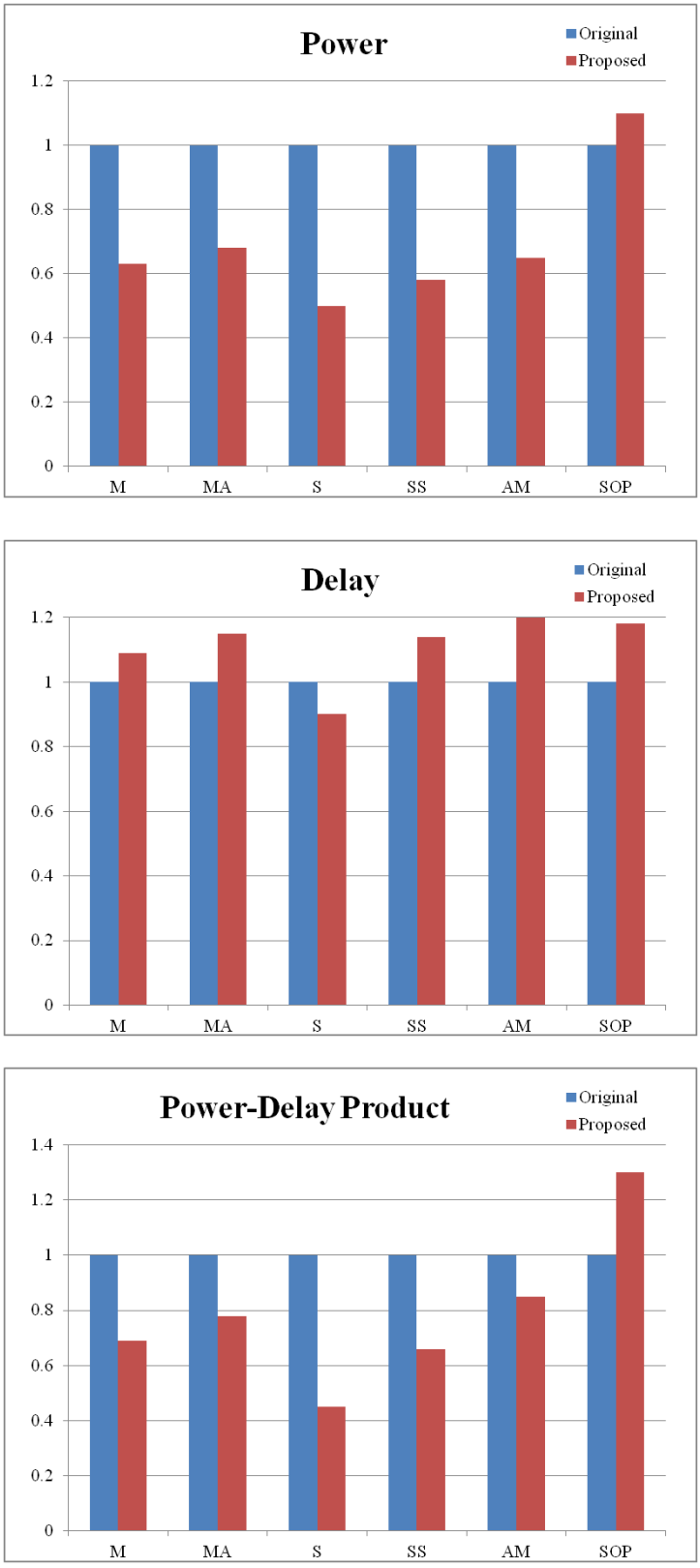


FIGURE 5.10: POWER, DELAY AND POWER-DELAY PRODUCT COMPARISON BETWEEN THE ORIGINAL AND THE PROPOSED OPERATIONS

5.5 Summary

In this chapter, we presented a sum-of-products unit capable of supporting several arithmetic operations using essentially the same hardware. The proposed arithmetic unit is useful in digital signal processing and multimedia applications, since they allow several multiplication-related operations to be performed on shared hardware, which has roughly the same delay as the original design. To achieve higher power savings, we have the control of the turn on/off mechanisms using signal gating techniques. Detailed experimental estimates have been given to compare the power, delay and area characteristics of each operation.

As the number of bit in the input operands increases, the increase rate in delay and area between the original and the proposed sum-of-products structure will be constant because the amount of additional gates is proportional to the operand size. Thus, our proposed structure will be efficient when applied to high-precision.

Chapter 6 SIMD and Approximate Arithmetic Unit based on Sum-of-Products

This chapter presents a fixed-point sum-of-products unit capable of supporting SIMD and approximate arithmetic operations with multiple-precisions. The proposed arithmetic unit can perform these operations using essentially the same hardware. Our goal is to reduce the power consumption without significantly increasing the delay, area and error rate of final results. To reduce the overall power dissipation, we use the SIMD for low-precision input data and approximate operations for high-precision input data. To execute these operations, the mode-dependent multiplexing is inserted into the PPG modules and the mode-dependent kills in the carry chain are inserted into the PPR array and the final CPA. The SIMD technique enables us to reduce a power by executing two 16 × 16-bit operations in parallel. It requires additional INVs, MUXs, AND gates and adders. The SIMD operation has approximately 45% less power, 15% less delay and 50% less execution time. This result indicates that the SIMD operation has almost double throughput increase compared to the standard operation. The approximate operation is to use 1) only a single multiplication when one result is significantly larger than the other result or 2) 16 × 16-bit multipliers instead of 32 × 32-bit multipliers. The approximate operation achieves between 40% and 73% power savings and between 42% delay decrease and 2% increase with between 0.6% and 2.8% mean error.

6.1 Introduction

We are currently facing problems brought about by the change of data characteristics for recent signal processing applications. In the existing signal processing applications, real data is generally limited to small range in most cases, and the case of maximum range rarely occurs [95][96]. However, recent signal processing applications are characterized by wide range with

high-precision [97]. Previous research of low-power design utilizing low-precision cannot be used for recent signal processing applications. Therefore, research on a new low-power approach is needed to satisfy low-power and high-throughput requirements for high-precision data.

A low-power design is highly desirable for recent signal processing, while high performance remains a major design goal. Generally, the requirements of low power and high performance are conflicting because increased performance can typically be achieved with a corresponding increase in power consumption due to increased frequency, increased hardware resources, or a combination of these two factors. To satisfy both requirements, the SIMD and approximate operations have been employed. The SIMD operation can reduce execution time using data-level parallelism, and run at a lower supply voltage, which can reduce dynamic power consumption compared to a standard operation. Likewise, minimizing power consumption with approximate results has become an area of great importance. Although the mobile devices with limited screen size can tolerate a reasonable amount of computation errors because a more sophisticated image could be designed, this might be difficult given the limited screen space and resolution.

In this chapter, we propose a SIMD and Approximate Arithmetic Unit based on Sum-of-Products (SAAU-SoP) operation. This arithmetic unit uses essentially the same hardware. We use a SIMD operation for throughput increase and power decrease using low-precision data and use an approximate operation for power savings using high-precision data. This chapter is organized as follows. First, we introduce problems and related works in Section 6.2. In Section 6.3, the basic structure of the sum-of-products will be presented, and then the details of each part of the sum-of-products will be described along with the required modifications necessary to support the SIMD and approximate arithmetic operations. This is followed by a description of how the SIMD and approximate techniques can be applied to each

module in a sum-of-products unit in Section 6.4. The experimental setup and the results will be presented in Section 6.5. Finally, the chapter will be concluded in Section 6.6.

6.2 Related Work

A SIMD describes multiple processing units that perform the same operation on multiple data simultaneously. This is usually used for signal processing applications [98][99][100]. Recent DSPs and GPUs are implementing wide SIMD, which allows arithmetic operations on 128 or 256-bit at the same time. They allow for the easy parallelization commonly involved in signal processing data. For example, operations to change the brightness of an image can be performed efficiently with the use of a SIMD. Each pixel of an image consists of three values for the brightness of red, green and blue. To change the brightness, all three values are read from memory, a specific value is added or subtracted from them, and the resulting values are written back out to memory. SIMD can maximize the throughput of such types of operations by using data-level parallelism. A recent advancement by mobile Application Processors (APs) is the production of an SIMD processor [101][102]. Specifically, ARM NEON technology is a 128-bit SIMD extension for APs that provides powerful performance acceleration for multimedia applications.

In order to reduce power dissipation, we can utilize imperfect human characteristics. For instance, we cannot recognize the slight difference of the final image and voice, even though the device processes data less accurately. Specifically, mobile devices with limited screen size can tolerate a reasonable amount of computation errors because more sophisticated image could be designed but this might be difficult given the limited screen space and resolution. Furthermore the eye is easily fooled, especially when the image is moving. Thus signal processing application in mobile systems can process their data less accurately. Recently, approximate

operations have become popular [103]. Arithmetic circuits that returned approximate results would require much less power dissipation than those in conventional arithmetic units. Previous studies which trade power dissipation for quality are typically at the algorithm level, where the parameters of quantized levels and the precision of coefficients are traded for the quality of the final solution. Applying sufficient statistical analysis, we can use for the average case rather than the worst case, and achieve significant power and delay savings. They include algorithmic noise tolerance [104][105][106], significance driven computation [107][108][109] and voltage over scaling [110]. All these techniques are based on the voltage over scaling with extra correction modules or limitation of the final results. Several studies on approximate arithmetic have also been conducted. An error-tolerant adder has been proposed in [111]. It operates by splitting the operands into accurate and approximate parts. The approach for logic complexity reduction in adders has been proposed in [112]. Imprecise but simple the mirror adders have been developed. They reduce power dissipation over conventional adder design. Truncated multipliers have been introduced [113][114]. These multipliers keep only the n MS bits of the final result and dispose of the n LS bits after perfuming rounding. Constructing a part of the multiplication matrix would reduce the complexity, but this might incur potentially large errors. Inaccurate multipliers using a 2×2 multiplier block resulting from logic complexity reduction have been proposed in [115][116]. Most previous studies have focused on the separate arithmetic unit for SIMD and approximate operations, and studies of composite arithmetic have not been conducted. In this chapter, we consider how to design and further optimize the sum-of-products structure covering these problems.

6.3 The Proposed Arithmetic Unit

Our goal is to reduce power consumption without a significant increase in the complexities of modules and interconnects. In this chapter, we consider the structure optimization techniques for SIMD and approximate operation. Some of these techniques have been used in multipliers, and they are investigated to describe how they perform in a sum-of-products unit.

6.3.1 The SAAU-SoP Structure

We have considered only the sum-of-products unit based on LR array structures proposed in Chapter 4. We here add a main controller and a dynamic range detector into the proposed structure, as shown in Figure 6.1. The designs presented in this chapter assume they are limited to a 32-bit operand size, but they can easily scale to a wide range of fixed-point operand sizes.

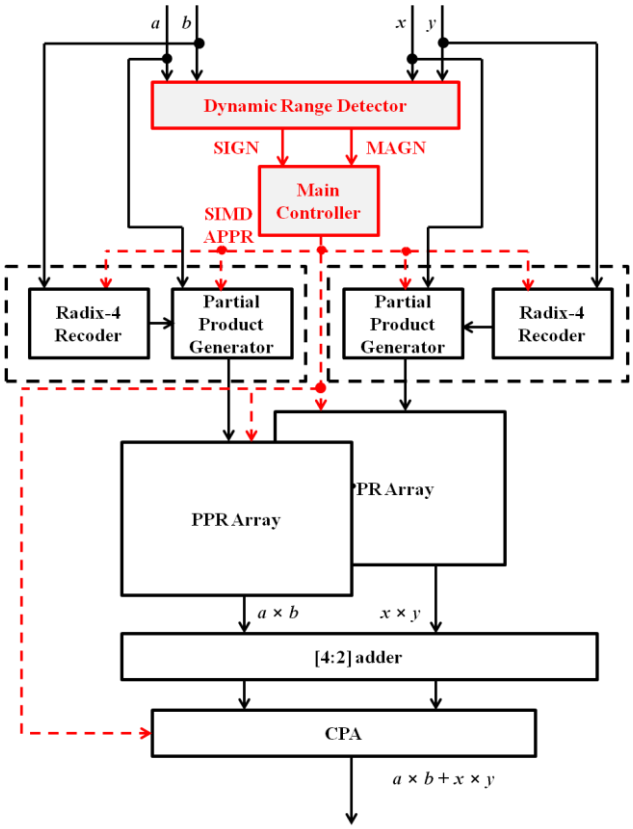


FIGURE 6.1: THE PROPOSED SAAU-SOP STRUCTURE

6.3.1.1 Operation Strategy

We can perform diverse operations based on dynamic input precision. If all input operands are smaller than 16 bits, a SIMD arithmetic operation can be performed. If at least one input operand is wider than 16 bits, a SIMD operation cannot be performed, but approximate operations can be performed by utilizing an approximation. The approximate sum-of-products computes the approximate results based on 64-bit results generated from the result of two 32 × 32-bit multiplications.

Based on these operations, we propose a new control strategy. The dynamic range detector can detect operand precision and generate the control signals to support effective SIMD and approximate operations. These control signals are based on the types of applications (or users) setting information and input operand precision. This information stored in the built-in memory of the system is read out. The detector can be aware of the range of input operands. The detector generates two types of signals: SIGN and MAGN. A 4-bit SIGN signal indicates four operands have positive or negative values. A 4-bit MAGN signal indicates the precision of the input operands. These two signals are then forwarded to a main controller that generates two control signals: SIMD and APPR. A 1-bit SIMD signal controls SIMD operation, and a 1-bit APPR signal controls the approximate operation. These control signals determine which modules are executed for the corresponding operations.

6.3.1.2 Signal Gating

As power dissipation is directly related to switching activities, reducing the switching activities would lead to lower power consumption. The proposed design can deactivate signals in unused parts using a signal gating technique; thus, switching activities occurring in the unused parts can be minimized. The general behavior of signal gating is as follows. The gated

signals are generated based on two control signals: SIMD and APPR. These two control signals are combined with AND gates. The signal gating logics are initially inserted at predetermined positions to identify the input control signals. If the SIMD or the APPR signal is set to 1, the gating logic is activated. Depending on these control signals, the PPGs adjust the number of PPs generated, and select appropriate PPR arrays and CPAs for a given operation. The details of the implementation will be covered in the following sections.

6.3.2 The Proposed Operations

We describe here how to support SIMD and approximate operations. We propose to use an ensemble of several small modules. The ensemble of small modules is selected so as to cover the required operation. Figure 6.2 shows an ensemble of four designs with low-precision: $A[3:0] \times B[3:0]$ shown in yellow (region A), $A[7:4] \times B[3:0]$ shown in violet (region B), $A[3:0] \times B[7:4]$ shown in blue (region C) and $A[7:4] \times B[7:4]$ shown in green (region D).

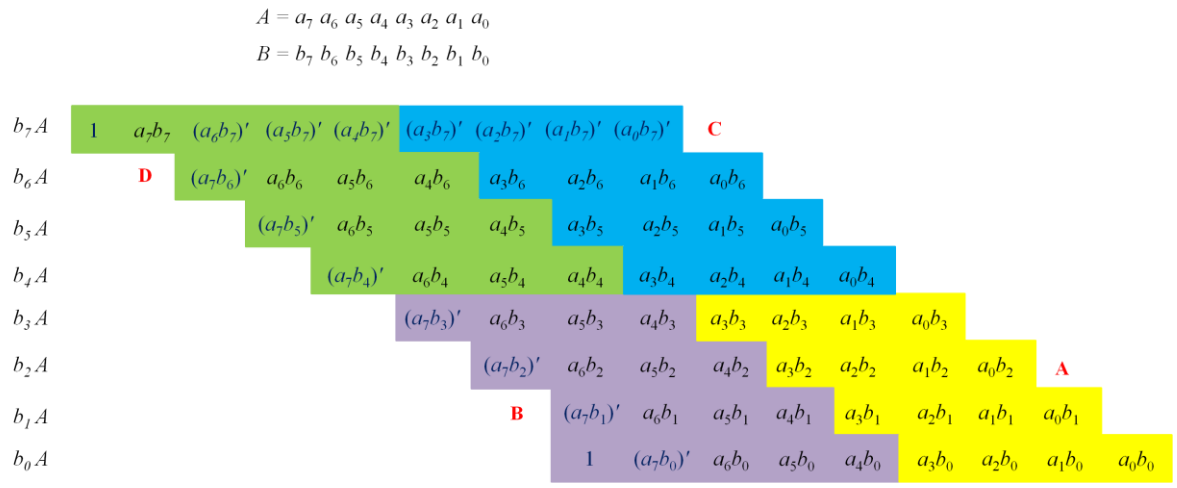


FIGURE 6.2: THE EXAMPLE OF A SIGNED RADIX-2 8-BIT LR MULTIPLIER BIT MATRIX FOR 4-POINT ENSEMBLE

6.3.2.1 SIMD Operation

We first present a SIMD technique. This technique enables power reduction by executing two operations in parallel. Clearly, by including a SIMD operation, one expects that fewer cycles are needed.

Consider another example: matrix multiplication.

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nm} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & \cdots & B_{1p} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mp} \end{bmatrix} A \cdot B = \begin{bmatrix} AB_{11} & \cdots & AB_{1p} \\ \vdots & \ddots & \vdots \\ AB_{n1} & \cdots & AB_{np} \end{bmatrix} \tag{6-1}$$

where the number of columns in A equals the number of rows in B.

We assume that all input data have value not larger than 2^{16} - 1. This matrix multiplication can be implemented in two ways by using a single 32 × 32 or two 16 × 16 SIMD sum-of-products operations. One way to use a standard sum-of-products operation is as follows:

```
for (i = 0; i < n; i++) {
        for (j = 0; j < p; j++) {
                for (k = 0; k < m; k+=2) {
                        AB[i][j] += A[i][k] × B[k][j] + A[i][k+1] × B[k+1][j]        (6-2)
                }
        }
}
```

The expression (6-2) corresponds to a single sum-of-products operation which could be executed as a single instruction. The last line corresponds to an accumulated sum-of-products operation. Matrix multiplication can be also implemented as

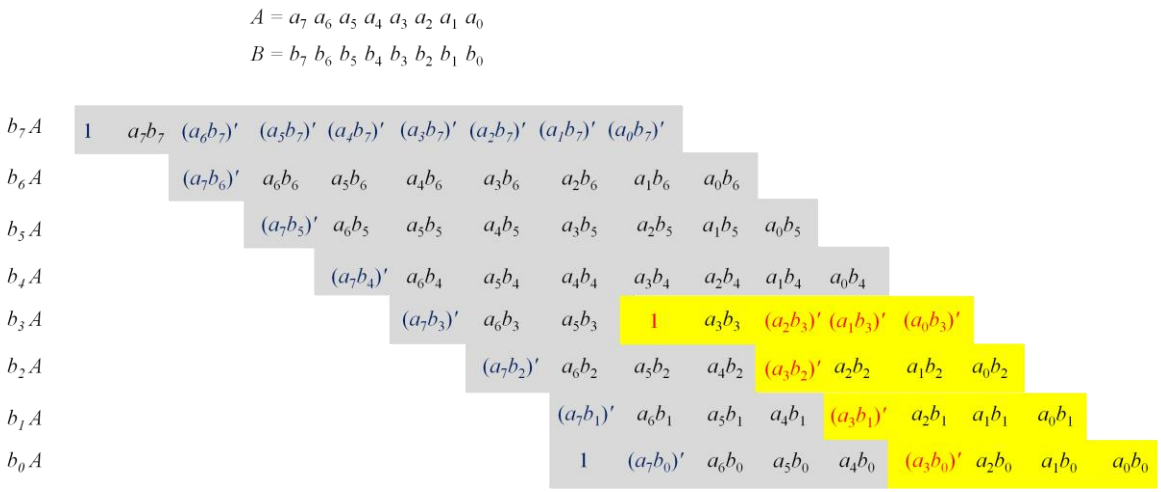
```
for (i = 0; i < n; i++) {
        for (j = 0; j < p; j++) {
                for (k = 0; k < m; k+=4) {
                        AB[i][j] += A[i][k] × B[k][j] + A[i][k+1] × B[k+1][j]
                                + A[i][k+2] × B[k+2][j] + A[i][k+3] × B[k+3][j]}
         }
}
```

(6-3)

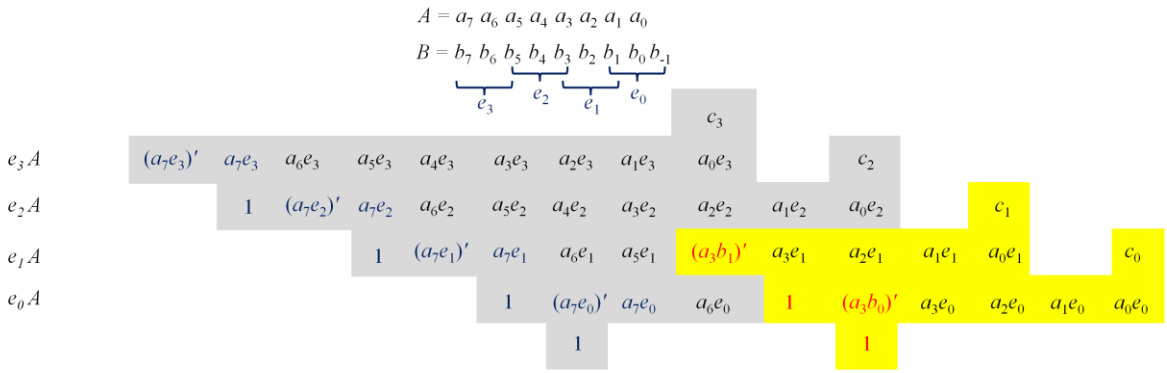
The expression (6-3) corresponds to 4-Dimensional (4D) sum-of-products operations. The last line corresponds to two sum-of-products operations that can be executed simultaneously. Ideally, the SIMD operation requires only half the number of clock cycles compared to using a standard sum-of-products operation. The SIMD operation can reduce the clock cycles, which has an impact on reducing the execution time. The SIMD structure can run at a lower supply voltage, which can reduce power consumption for dynamic power compared to a single solution.

Figure 6.3 shows the example of an 8-bit LR multiplier bit matrix using 4-bit low-precision operation. Used bit of operands is shown in yellow, and unused bit is shown in gray. When the precision of the operand is smaller than the multiplier we intend to use, a number of PPs contain a sign extension bit. Furthermore, the summation of the MS part of a PP array also contains a meaningless sign extension bit. In order to take advantage of low-precision data, signal gating can be applied to deactivate the unused parts of the PPR array to match data precision, thereby

avoiding unnecessary switching activities in the operation. One observes that the positions between b_4A and b_7A in Figure 6.3(a) and between e_2A and e_3A in Figure 6.3(b) are not used; thus, these parts are available for another multiplication. Figure 6.4 shows the example of a signed 8-bit multiplication, where the first 4-bit multiplication, shown in yellow, is computed in parallel with the second 4-bit multiplication, shown in green. Compared to the original matrix, several modifications are needed to compute the 4-bit multiplications. In Figure 6.4, red text indicates the modified bits. In Figure 6.4(a), the uppermost low (b_3A) except for the MSB (a_3b_3) is the opposite in the LS part; thus, all PPs except for the MSB need to be negated. Also, the MSBs (a_3b_0, a_3b_1, a_3b_2) of each row need to be negated to get the correct result. Finally the sign bit (1) is needed in the uppermost (b_3A) low. This means there is a need for extra inputs, which requires several extra adders and MUXs. In the LS part of radix-4 multiplier bit matrix, the MSBs (a_3e_0, a_3e_1) of each row need to be negated, and 1s are added. In the MS part, two compensation bits (c_2, c_3) and 1 are needed to prevent the sign extension. The carry chain in the adder at the boundary position should be disconnected; hence, the power dissipation can be reduced by removing unnecessary switching activities of unused gates, shown in gray, compared to normal operation.

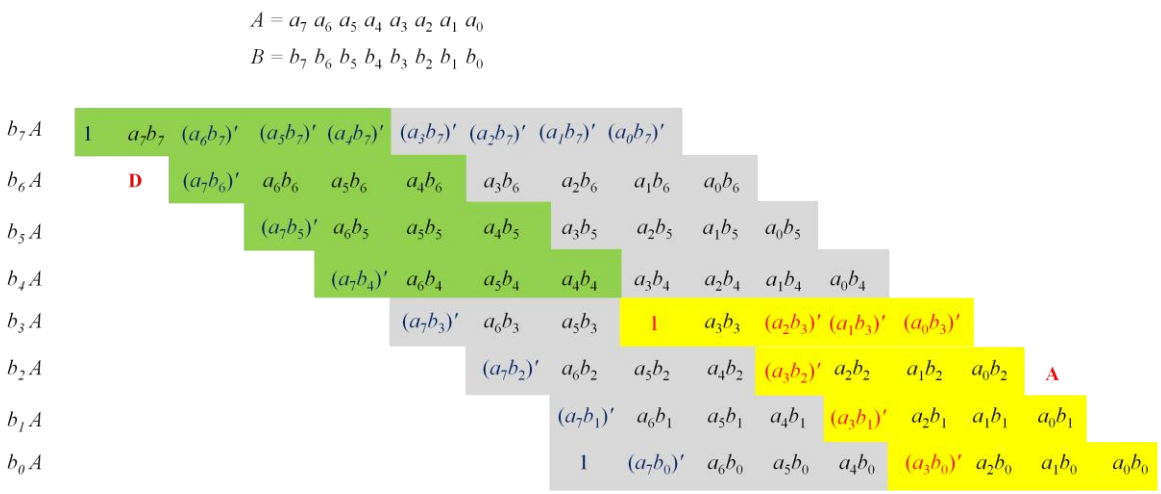


(A) A SIGNED RADIX-2 8-BIT LR MULTIPLIER BIT MATRIX

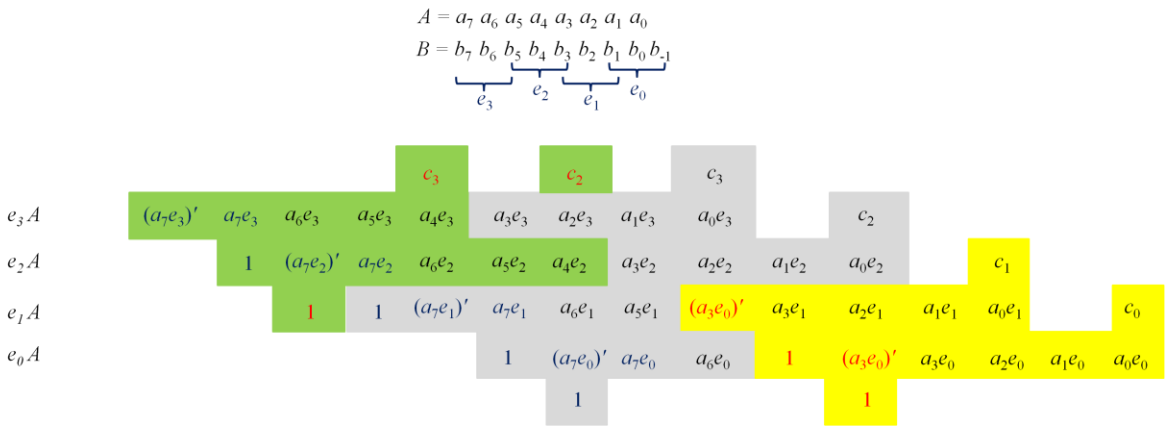


(B) A SIGNED RADIX-4 8-BIT LR MULTIPLIER BIT MATRIX

FIGURE 6.3: THE EXAMPLE OF AN 8-BIT LR MULTIPLIER BIT MATRIX USING 4-BIT LOW-PRECISION OPERATION



(A) A SIGNED RADIX-2 8-BIT LR MULTIPLIER BIT MATRIX



(B) A SIGNED RADIX-4 8-BIT LR MULTIPLIER BIT MATRIX

FIGURE 6.4: THE EXAMPLE OF AN 8-BIT LR MULTIPLIER BIT MATRIX USING 4-BIT SIMD OPERATION

Also, we can apply this SIMD technique to a radix-4 multiplication. In 16 × 16 SIMD multiplication, the radix-4 recoder with digit set {-2, -1, 0, 1,2} is the same as that used for the 16 × 16 standard multiplication except that multiplier operand (*B*) changes. In 16 × 16 standard multiplication, the first PP is generated by assuming *B*[-1] = 0. In SIMD multiplication, 0 needs to be inserted at the element boundary for the first PP of the MS part. These are the extra 0 bit. For example, in the 16 × 16 SIMD multiplication, the ninth PP uses the bit triplet {*B*[17], *B*[16],

0} instead of {*B*[17], *B*[16], *B*[15]} as used in the 16 × 16 normal multiplication, as shown in Figure 6.5.

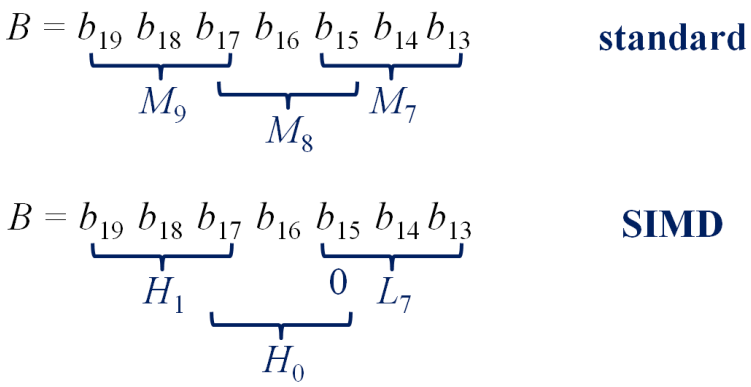


FIGURE 6.5: THE RADIX-4 RECODING (STANDARD AND SIMD MULTIPLICATIONS)

We need to modify the final CPA to support SIMD operation. The ensemble of small adders is selected so as to cover the required operations. The carry input bit from each adder is set to zero, and the carry out bit from each adder cannot be passed into the carry input of the next adder. The MUXs are also added to correctly select the value. To design the final CPA that performs either the standard or SIMD computation, an input control signal *s* is introduced, where *s* is set to 1 for SIMD and 0 for standard computation. Consequently, the implementation of the signed SIMD multiplication does not make significant changes to PPGs and PPR arrays, and it is possible to execute two 16 × 16-bit sum-of-products operations or four 16 × 16-bit multiplications simultaneously if all input operands are smaller than 16-bit.

6.3.2.2 Approximate Operation

Previous works have focused on developing approximate multipliers and have not considered composite arithmetic operations, such as sum-of-products, in reducing power consumption. We propose to investigate power optimization of a sum-of-products unit for approximate operation. If the applications do not have strict quality constraints, we can use the

approximate sum-of-products operation. More power saving can be achieved at the risk of losing more data by choosing a higher level of approximation.

There are two approaches for approximation. One approach is to use only a single multiplication when one result is significantly larger than the other result. This is an easy way of executing the approximate sum-of-products operation by omitting a single multiplication. We use a static analysis based on the prior knowledge for application-specific precision. This method determines reduced precisions with guaranteed error bounds. There are two methods. One way is to use the significant difference between two results, $a \times b$ and $x \times y$. If both operands a, b have more than 16 bits, and both operands x, y have less than 16 bits, it is obvious that one result, $a \times b$, would be much larger than the other result, $x \times y$. In this case, the smaller multiplication, $x \times y$, can be omitted. This scheme is called 32 × 32_low_error. In other cases, two multiplication results are maintained. The other method is to achieve more power saving with higher error. If both operands a, b have more than 16 bits, and either operands x or y has less than 16 bits, the result, $a \times b$, would be larger than the other result, $x \times y$. In these cases, the smaller multiplication, $x \times y$, can be omitted. This scheme is called 32 × 32_high_error. This operation is summarized in Table 6.1. These 32 × 32_low_error and 32 × 32_high_error operations have a small and easily computable error probability of 12.5% and 62.5% with a max error magnitude of 50%, respectively. The results in Table 6.1 show that the max-possible error magnitude remains constant at 50% (maximum error occurs when two multipliers have the maximum values, but one multiplication is omitted), while the min-possible error magnitude remains constant at 0% (minimum error occurs when one multiplier has the minimum values (0), then this multiplication is omitted). The error models assume a uniform distribution of input vectors; hence, the results have the continuous uniform error distribution. The mean error has less than 0.0015% because the LS 16-bit cannot affect the final results. We can use the proposed approach for certain types of signal processing applications which are

inherently capable of absorbing error in arithmetic operations. Specifically, mobile devices with small screen size can tolerate a reasonable amount of computation errors because more sophisticated image cannot be difficult given the physical size. Figure 6.6 shows the architecture level schematic of a sum-of-products unit. The second approach for approximation is to use only 16 × 16-bit multipliers instead of 32 × 32-bit multipliers. This scheme is called 16 × 16_error. By using the small multiplier, even higher power savings will be possible. The difference between power dissipation of two multipliers with different size provides significant power savings. Table 6.2 shows power, delay and area comparison between 16 × 16-bit and 32 × 32-bit multipliers. Based on these estimates, the 16 × 16-bit multiplier has 81% less power, 41% less delay, and 73% less area than the 32 × 32-bit counterparts. The basic idea is to detect the precision of the input operands using a leading zeroes and ones detection circuit and then route them to suitable multiplier. By moving the window, we can obtain more accurate results. The 16-bit window for calculating multiplication is moved into from the leading one bit for positive numbers (zero bit for negative numbers) through the next 16-bit. A leading one/zero detector is provided for using the input operands to detect the leading one bit position for positive numbers (zero bit position for negative numbers). All 32 bits of the input operand is routed into 32 inputs of MUXs. A leading one bit for a positive value (zero bit for a negative value) is mapped to the MSB of a 16-bit window, the next bit is mapped to the second bit position, and the next 16-bit is mapped to the LSB of a 16-bit window. With the MUX, the MSB of the 32-bit operand is chosen when 31 is asserted at selection signal and the LSB of the 32-bit adder is chosen with the assertion of 0 at the selection signal. The 16 bits of the 32-bit input operand for calculating multiplication are then selected with the multiplexer once the position of the leading one (zero) bit is known. Finally, the 16-bit window takes 16-bit from the leading one (zero) bit through the next 16-bit of the operand. We assume the MUX321 gate consists of five stages of 31 × MUX21 (= 16 + 8 + 4 + 2 + 1). MUX321 gates

enable a reduced latency by executing MUX gates in the same stage in parallel. The estimated delay and area of additional MUX are estimated roughly as below

Area: 2 × 16 × MUX321 = 2 × 16 × 31 × MUX21 = 992 × MUX21

Delay: MUX321 = 5 × MUX21

TABLE 6.1: USED PPR MODULES FOR A 32 × 32_LOW_ERROR AND A 32 × 32_HIGH_ERROR METHODS BASED ON INPUT OPERAND PRECISION

Operand Precision				Used Modules				
a	*b*	*X*	*Y*	Accurate Mode	Approximate Mode (32 × 32_low_error)		Approximate Mode (32 × 32_high_error)	
< 16	< 16	< 16	< 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$a \times b + x \times y$	accurate
< 16	< 16	< 16	> 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$x \times y$	Appr.
< 16	< 16	> 16	< 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$x \times y$	Appr.
< 16	< 16	> 16	> 16	$a \times b + x \times y$	$x \times y$	Appr.	$x \times y$	Appr.
< 16	> 16	< 16	< 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$x \times y$	Appr.
< 16	> 16	< 16	> 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$a \times b + x \times y$	accurate
< 16	> 16	> 16	< 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$a \times b + x \times y$	accurate
< 16	> 16	> 16	> 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$x \times y$	Appr.
> 16	< 16	< 16	< 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$a \times b$	Appr.
> 16	< 16	< 16	> 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$a \times b + x \times y$	accurate
> 16	< 16	> 16	< 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$a \times b + x \times y$	accurate
> 16	< 16	> 16	> 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$a \times b$	Appr.
> 16	> 16	< 16	< 16	$a \times b + x \times y$	$a \times b$	Appr.	$a \times b$	Appr.
> 16	> 16	< 16	> 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$a \times b$	Appr.
> 16	> 16	> 16	< 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$a \times b$	Appr.
> 16	> 16	> 16	> 16	$a \times b + x \times y$	$a \times b + x \times y$	Accurate	$a \times b + x \times y$	accurate

Operation Mode	Max. Error (%)	Min. Error (%)	Error Probability (%)
32 × 32_low_error	50.00	0	12.50
32 × 32_high_error	50.00	0	62.50

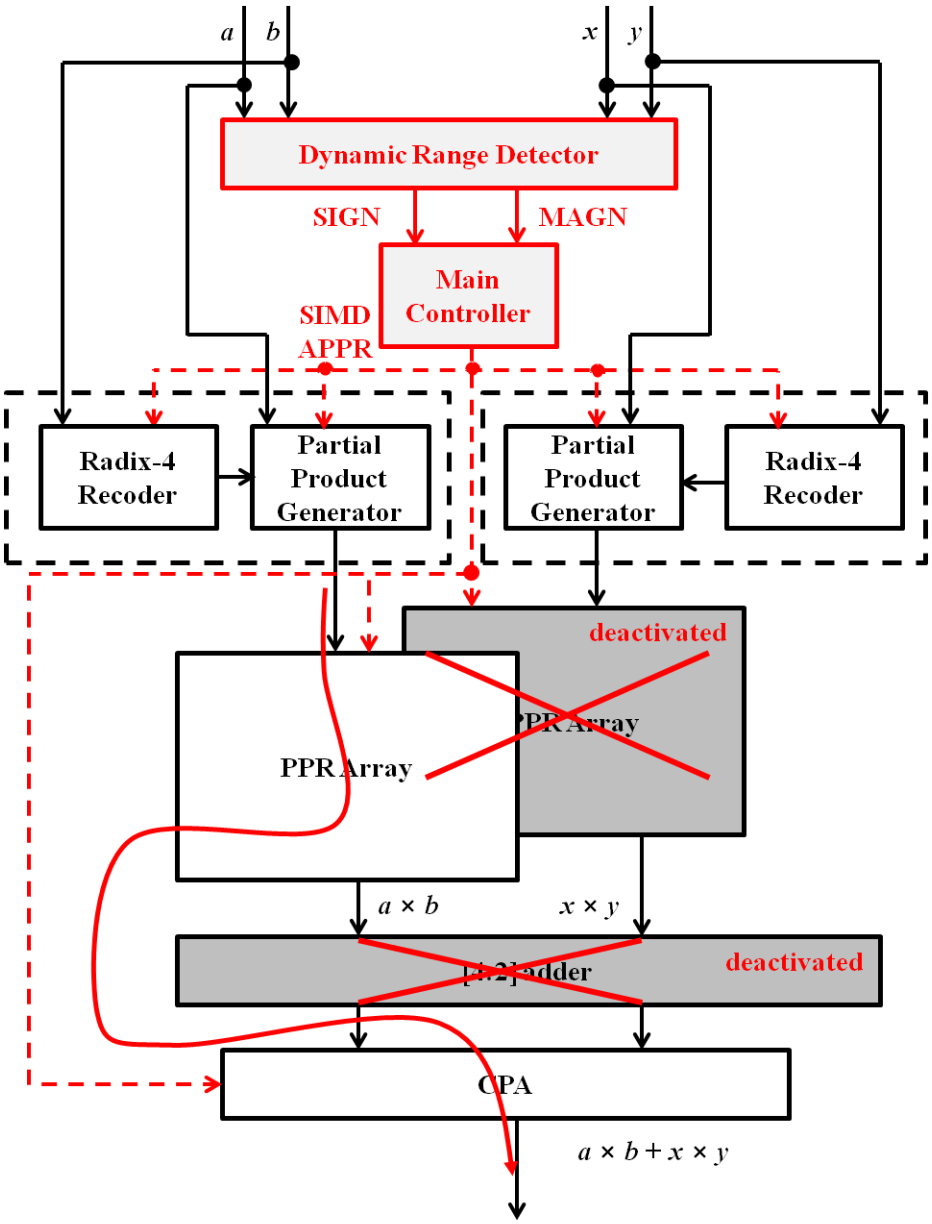
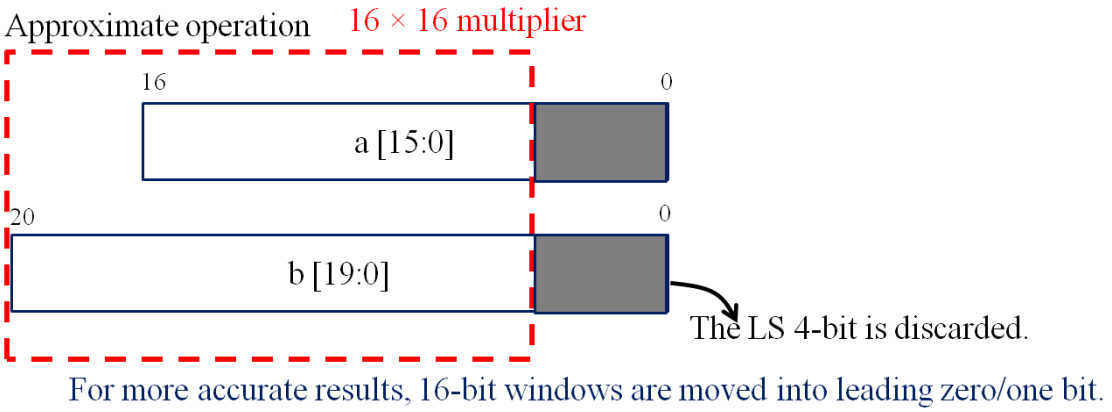


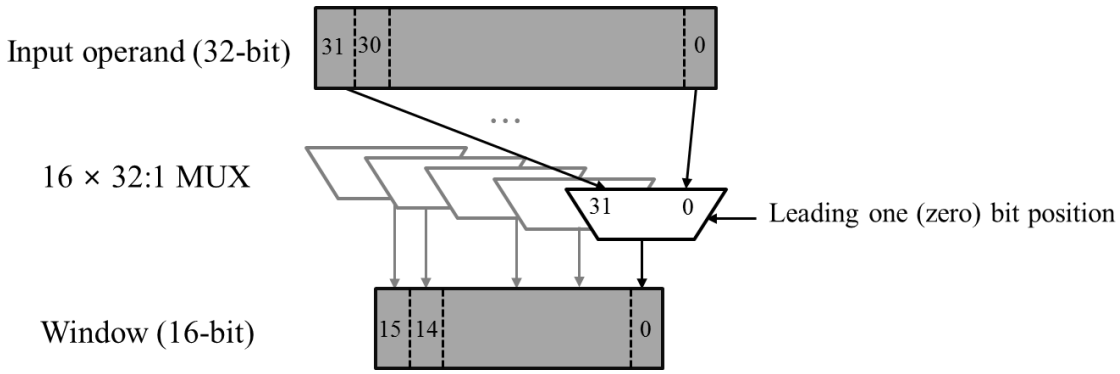
FIGURE 6.6: APPROXIMATE SUM-OF-PRODUCTS OPERATION

TABLE 6.2: POWER, DELAY AND AREA COMPARISON BETWEEN 16 × 16-BIT AND 32 × 32-BIT MULTIPLIERS

	Power (μW)		Delay (ns)		Area (μm^2)	
16 × 16-bit LR multiplier	1127	0.19	5.42	0.59	3265	0.26
32 × 32-bit LR multiplier	5932	1.00	9.18	1.00	12559	1.00



(A) APPROXIMATE OPERATION USING 16 × 16-BIT MULTIPLICATION



(B) MAPPING USING 16 × 16-BIT WINDOWS

FIGURE 6.7: APPROXIMATE SUM-OF-PRODUCTS OPERATION FOR 16 × 16_ERROR METHOD

6.4 Basic Components

In this section, we consider how to design the approximate sum-of-products and how to further optimize the structure.

6.4.1 Dynamic Range Detector and Main Controller

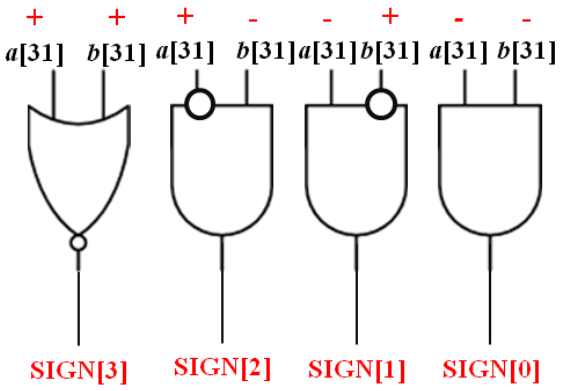
The dynamic range detector detects the signs and effective dynamic ranges of all input data, and then generates the control signals: SIGN and MAGN. These control signals are used to

disconnect carry chains and deactivate unused parts of the PPR array and the final CPA. The detector consists of several OR and MUX gates. Table 6.3 shows the function table, and Figure 6.8 shows the functional blocks of the dynamic range detector. The sign detector consists of simple NOR2, AND2 and INV gates. In the magnitude detector, the OR15 gate is used to detect positive numbers and the NAND15 gate is used to detect negative numbers. The OR15 gate asserts the output to be high if any of the inputs is 1, and the NAND15 gate asserts the output to be high (1) if any of the inputs is 0. Using the combination of simple gates, we can detect the sign and magnitude of operands. Both the 32-bit multiplier and multiplicand operands are divided into two parts, where the detection is completed for 32 and 16-bit ranges. The output of a dynamic range detector are grouped into 4-bit SIGN and 4-bit MAGN signals, where the MS bit of the SIGN signal is set to high (1) if both inputs are positive numbers, the second bit is set to high (1) if input *A* is positive, but input *B* is negative, the third bit is set to high (1) if input *A* is negative, but input *B* is positive number, and the LS bit is set to high (1) if both inputs are negative. The MS bit of the MAGN signal is set to high (1) if the magnitudes of both inputs are larger than 16-bit, the second bit is set to high (1) if the magnitudes of input *A* is larger than 16-bit, but the magnitudes of input *B* is not larger than 16-bit, the third bit is set to high (1) if the magnitudes of input *A* is not larger than 16-bit, but the magnitudes of input *B* is larger than 16-bit, and the LS bit is set to high (1) if the magnitudes of both inputs are not larger than 16-bit. The main controller generates the SIMD signal indicating the SIMD operation and the APPR signal indicating error-tolerant operation based on the users (or application) information and operand sizes. The enable signals for carry-kill, which are inputs of signal gating modules, are generated from the main controller.

TABLE 6.3: FUNCTION TABLE OF DYNAMIC RANGE DETECTOR

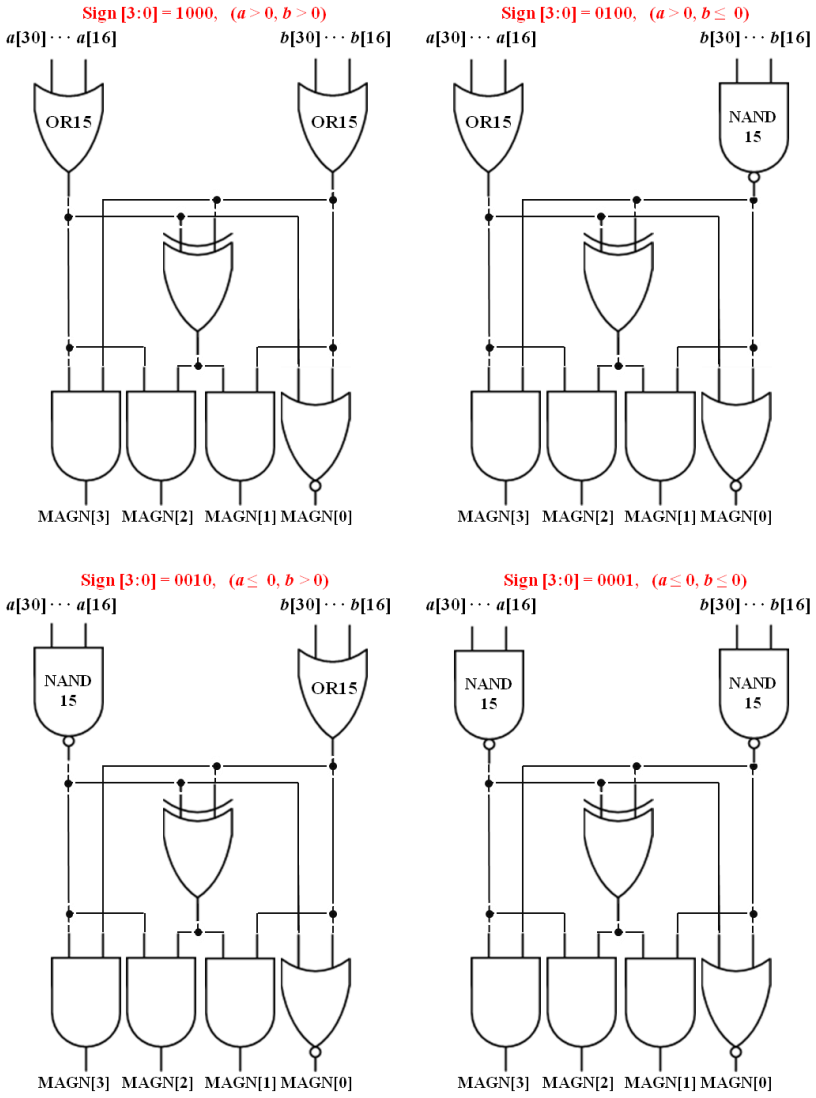
Operand		Output			
a (*x*)	*b* (*y*)	SIGN [3]	SIGN [2]	SIGN [1]	SIGN [0]
+	+	1	0	0	0
+	-	0	1	0	0
-	+	0	0	1	0
-	-	0	0	0	1

Operand Precision		Output			
a (*x*)	*b* (*y*)	MAGN [3]	MAGN [2]	MAGN [1]	MAGN [0]
> 16	> 16	1	0	0	0
> 16	≤ 16	0	1	0	0
≤ 16	> 16	0	0	1	0
≤ 16	≤ 16	0	0	0	1



Sign	SIGN[3:0]
$a > 0$ $b > 0$	1000
$a > 0$ $b < 0$	0100
$a < 0$ $b > 0$	0010
$a < 0$ $b < 0$	0001

(A) A SIGN DETECTOR



(B) A MAGNITUDE DETECTOR

FIGURE 6.8: FUNCTIONAL BLOCKS OF THE DYNAMIC RANGE DETECTOR

6.4.2 The Radix-4 Recoder and the PPG

The PPGs use radix-4 recoding which reduces the number of PPs from n to $\lceil n/2 \rceil$ for an $n \times n$-bit multiplication [42]. The block diagram for the radix-4 recoder and PPG modules are shown in Figure 6.9. This is identical to standard PPG modules except for additional AND2 gates for masking. The PPGs are able to generate the appropriate PPs for each of the supported operation modes. A 32 × 16-bit or 16 × 16-bit multiplication (MAGN[3:0] = 0100 or 0001) requires only the first eight PPs; thus, the first eight PPGs are activated while the next eight PPGs are deactivated. However, the SIMD 16 × 16-bit multiplication requires all sixteen PPs, so all sixteen PPGs are activated. If the magnitude of the multiplier is larger than 16-bit (MAGN[3:0] = 1000 or 0010), all sixteen radix-4 recoders are activated. Because all radix-4 recoders are activated, the control signals for deactivating radix-4 recoders are not necessary. The radix-4 recoder can generate five possible values of -2, -1, 0, 1, and 2 times the input data. The control signals are generated depending on the 3-bit recording scheme. The radix-4 recoder is used to generate three control signals: SHIFT, COMP and ZERO, which are used in the PPG. The SHIFT signal is used to shift the multiplicand operand left by one-bit, the COMP signal inverts the input multiplicand operands, and the ZERO signal is used to output zeros as output of that PP.

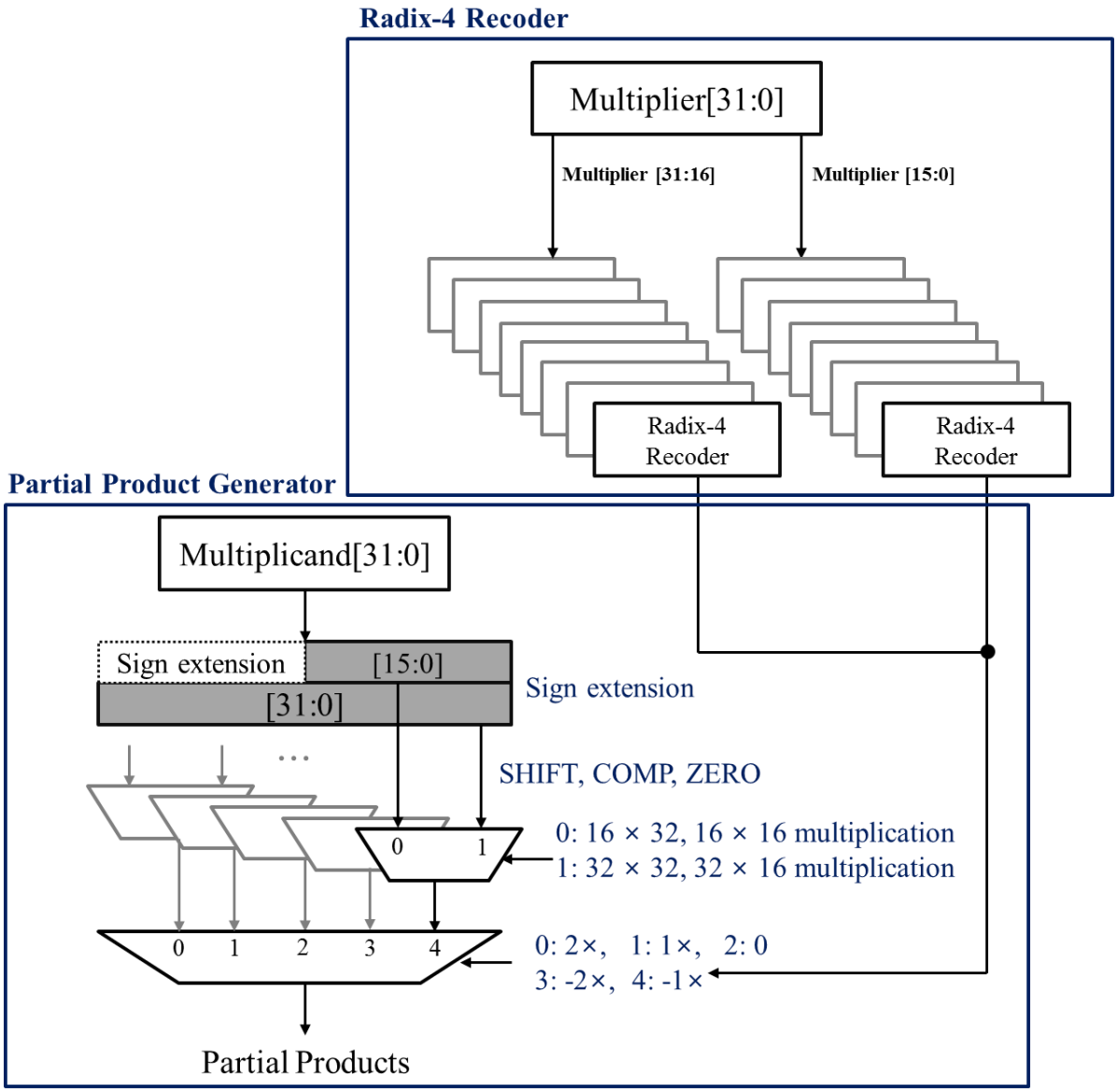


FIGURE 6.9: FUNCTIONAL BLOCKS OF THE RADIX-4 RECODER AND PPGS

6.4.3 The PPR Array

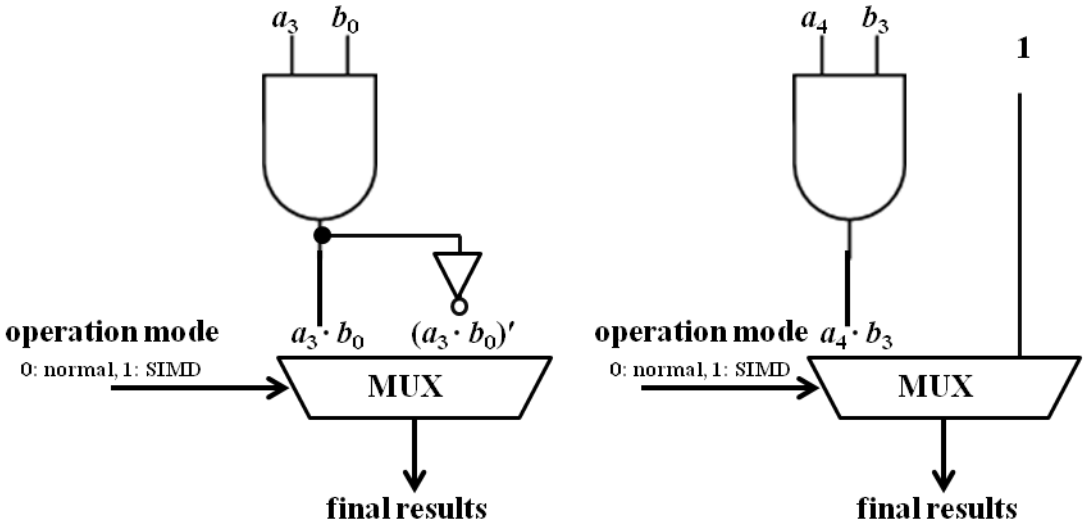
As mentioned in Section6.3.2, the standard PPR array and SIMD PPR array reveal minor differences between two multiplication bit matrices. To unify two PPR arrays, it is necessary to insert several gates at each mode element boundary. Figure 6.10 shows the added modules for SIMD operation. We define an input control signal based on the operation mode, which is set to high (1) for a SIMD computation, and low (0) for a standard operation. In order to support the

SIMD operation, we have to disconnect the carry chain in the adder at the boundary position. The bit positions of the boundaries depend on the operation mode. The killing of carries can be achieved by inserting a 2-input AND gate at each element boundary to mask the carry-in input of each corresponding FA. The carry-kill AND gates can be incorporated into the existing FA design such that they will not significantly increase delay through the array as shown in Figure 6.11. The Boolean equations with the carry-in kill term are given in the following equation.

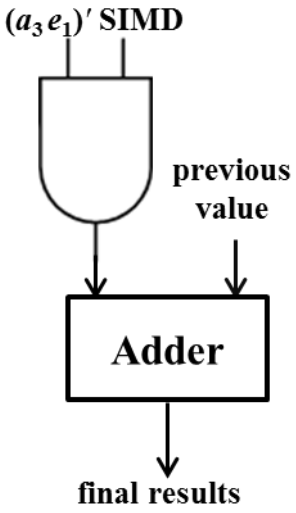
$$\begin{aligned}
&\textit{KILL_carry_v_SIMD}=(\text{MAGN[1]}\quad \text{OR}\quad \text{MAGN[0]})\quad \text{AND}\quad \text{SIMD}\\
&\textit{KILL_sum_v_SIMD}=(\text{MAGN[1]}\quad \text{OR}\quad \text{MAGN[0]})\quad \text{AND}\quad \text{SIMD}\\
&\textit{KILL_carry_h_SIMD}=(\text{MAGN[2]}\quad \text{OR}\quad \text{MAGN[0]})\quad \text{AND}\quad \text{SIMD}\\
&\textit{KILL_sum_h_SIMD}=(\text{MAGN[2]}\quad \text{OR}\quad \text{MAGN[0]})\quad \text{AND}\quad \text{SIMD}
\end{aligned} \tag{6-4}$$

Our gating approach provides gating lines for SIMD operations, as shown in Figure 6.12. This method is efficient because it does not add significant delay to the critical path and does not require much extra hardware.

Compared to a 32-bit radix-2 standard PPR array, a 32-bit radix-2 PPR array for SIMD operation requires an additional 31 × INV (to compute inverted PPs at the uppermost low ($b_{15}A$) and at the MSB at the all row except for the uppermost low ($a_{15}b_0$, $a_{15}b_1$, ⋯, $a_{15}b_{13}$, $a_{15}b_{14}$) and 1 in the part), and 31 × MUX21 (to select one of two results). Compared to a 32-bit radix-4 standard PPR array, a 32-bit radix-4 PPR array for SIMD operation requires an additional 14 × HA and 1 × FA (to add inverted PPs and 1s in the lower adder), and 9 × HA (to add compensation bits and 1s in the higher adder). SIMD operation also requires a 32-bit signal gating with the carry-kill signals used as the enable signals for the signal gating circuit. It requires additional 65 × AND2 gates to mask carry signals.



(A) FOR RADIX-2



(B) FOR RADIX-4

FIGURE 6.10: THE ADDED MODULES FOR SIMD OPERATION

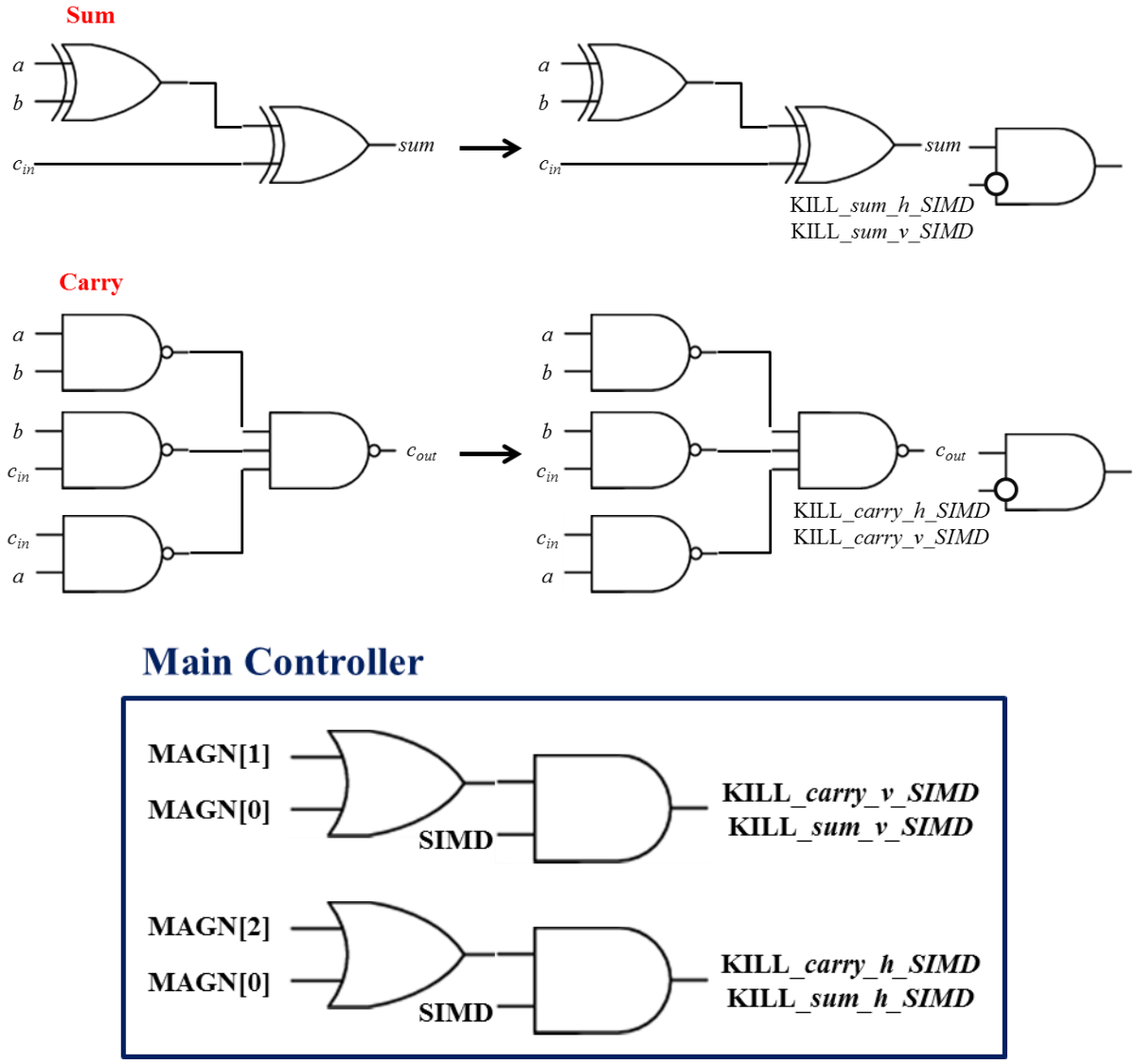


FIGURE 6.11: THE MODIFIED MODULES FOR SIGNAL GATING IN PPR ARRAY

FIGURE 6.12: GATING LINES FOR SIMD OPERATION

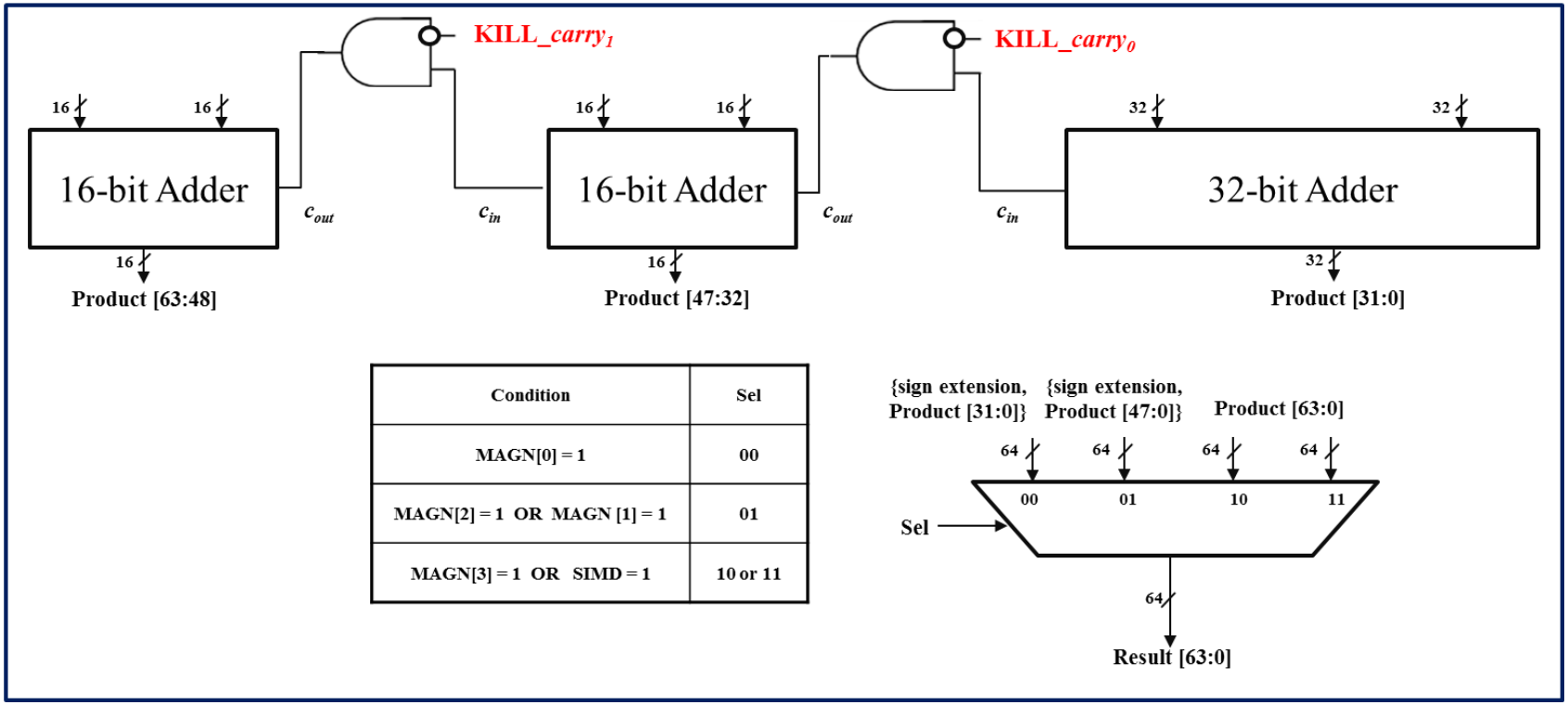
6.4.4 The Final CPA

The final CPA consists of two 16-bit and one 32-bit adders, where the "MAGN" signals indicating the multiplication size determines the selection order of these structures for final addition. In order to minimize the switching activities, we design the final CPA using signal gating with carry control signals, which enable the c_{out} from the lower adder to be passed into the c_{in} of the next adders, as shown in Figure 6.13. The unused blocks of the final CPA are disabled to avoid switching activities. If a 16 × 16-bit SIMD operation is selected, the c_{out} from the lower 32-bit adder cannot be passed into the c_{in} of the next adder; thus, the MS 32-bit adder accepts the carry input as zero. The 48-bit output of the 32 × 16-bit (or 16 × 32-bit) multiplication is distributed between the two adder structures. The c_{out} from the LS 32-bit adder is passed into the c_{in} of the middle 16-bit adder. The c_{out} from the middle 16-bit adder cannot be passed into the c_{in} of next 16-bit adder. In the 32 × 32-bit operation, the c_{out} from the LS 32-bit adder is passed into the c_{in} of the middle 16-bit CLA and then passed into the c_{in} of the MS 16-bit adder. The MUX of the final CPA selects the 64-, lower 48-, 32-bit for accurate operation or higher 32-bit for approximate operation to yield the final result. The gating method involves killing the carries which cross the element boundaries that are determined by the operation mode selected and operand size. This is similar to the method used in the reduction array. The extra 2-input AND gate can be combined into the adder blocks without significantly additional delay. The Boolean equations with the carry-in kill term are

$$KILL\_c_{in0} = (\text{SIMD AND MAGN}[0])$$
$$KILL\_c_{in1} = (\text{SIMD OR MAGN}[3])' \quad (6\text{-}5)$$

Main Controller

Final CPA



Condition	Sel
MAGN[0] = 1	00
MAGN[2] = 1 OR MAGN [1] = 1	01
MAGN[3] = 1 OR SIMD = 1	10 or 11

FIGURE 6.13: FUNCTIONAL BLOCKS OF THE FINAL CPA

6.4.5 Overall Execution

In order to combine the conventional sum-of-products unit with all the proposed schemes, several modules are inserted into the original multiplier matrix. The area and delay increases to be estimated are shown in Table 6.4. We estimate the area and delay based on the detailed design in the previous section and the Samsung library and we estimate dynamic power dissipation based on the proposed structure and equation (1-10). The extra carry-kill AND signal gating operation can be executed in parallel with inversion terms; thus, additional inverters will not increase the total delay. Based on these observations, the critical path delay would increase by approximately 40 × AND2 delay (20 × unit delay) and the additional buffering delay. Adding gates would also increase the total wire delay due to more complex routing. Our proposed arithmetic unit can deactivate several modules unused when SIMD and approximate operations are performed. The area and logic delay comparison to be estimated is shown in Table 6.5 .The cell areas are measured in the NAND2 gate based on the Samsung standard cell library. The delays and areas of the PPR array and the CPA have been analyzed in Chapter 2 and Chapter 3and thus we reuse them. Compared to a 32 × 32-bit standard sum-of-products operation, a 16 × 16-bit SIMD operation has 42% less used area and 35% less delay, and an APPR operation has 39% less used area with a 2% delay increase. The delay decrease of a SIMD operation is because two small operations can be executed in parallel, and the delay increase of APPR operation arises from the extra gates to mask carry signals. The used area decrease of SIMD and APPR operations is due to unused segmentations in the PPR array. Compared to a sum-of-products operation using two 32 × 32-bit multiplications, the proposed operation using only one 32 × 32-bit multiplication with one neglected 32 × 32-bit multiplication has approximately 40% less used area, and one using two 16 × 16-bit multiplication has approximately 65% less used area and 35% less delay. The decrease in area and delay is because one multiplication is neglected and small multiplication is performed. It

is not easy to estimate the current power reduction of the proposed structure. Based on the equation for dynamic power dissipation, reduced power can be achieved by decreasing these factors: capacitance, supply voltage, frequency, and switching activities. The proposed design can eliminate significant switching activities because several modules are deactivated; thus, the proposed structure would consume less power.

TABLE 6.4: DELAY AND AREA INCREASE OF ADDED MODULES (RADIX-2)

(A) DYNAMIC RANGE DETECTOR AND MAIN CONTROLLER

Module	Delay	Area
Dynamic Range Detector	SIGN $T_{AND2} + T_{INV}$ MAGN $T_{AND15} + T_{XOR2} + T_{AND2}$	SIGN $3 \times A_{AND2} + 2 \times A_{INV} + A_{OR2}$ MAGN $A_{OR15} + A_{AND15} + 4 \times (A_{XOR2} + A_{OR2}) + 12 \times A_{AND2}$
Main Controller	SIMD PPG: $T_{OR2} + T_{AND2}$ Common Array: $T_{OR2} + T_{AND2}$ CPA: $2 \times T_{OR3} + T_{AND2}$	SIMD PPG: $A_{OR2} + A_{AND2} + A_{INV}$ Array: $2 \times (A_{OR2} + A_{AND2})$ APPR Array: $A_{OR3} + A_{AND2}$ Common CPA: $A_{OR2} + 2 \times (A_{OR3} + A_{AND2})$
Total	$T_{AND15} + T_{XOR2} + T_{INV} + 5 \times T_{AND2} + 2 \times (T_{OR2} + T_{OR3})$	$A_{OR15} + A_{AND15} + 4 \times A_{XOR2} + 9 \times (A_{OR2} + A_{AND2}) + 3 \times (A_{INV} + A_{OR3})$

(B) SIGNAL GATING MODULE

Module	Delay	Area
PPG	SIMD $T_{AND2} + T_{INV}$	SIMD $15 \times (A_{AND2} + A_{INV})$
Array	SIMD $32 \times T_{AND2}$ APPR $31 \times T_{AND2}$	SIMD $65 \times (A_{AND2} + A_{INV})$ APPR $31 \times (A_{AND2} + A_{INV})$
CPA	Common $2 \times T_{AND2}$	Common $2 \times (A_{AND2} + A_{INV})$
Total	$35 \times A_{AND2} + T_{INV}$	$113 \times (A_{AND2} + A_{INV})$

TABLE 6.5: DELAY AND AREA COMPARISON BETWEEN THE ORIGINAL AND THE PROPOSED OPERATIONS

Operation	The original operations (32 × 32-bit)				The proposed operations (32 × 32-bit)			
	Delay (T_{XOR})		Used Area (NAND2)		Delay (T_{XOR})		Used Area (NAND2)	
SIMD	68.3	11870	11870	6884	45.8	0.67	45.8	0.67

Operation	The original operations (two 32 × 32-bit)				The proposed operations (one 32 × 32-bit)			
	Delay (T_{XOR})		Used Area (NAND2)		Delay (T_{XOR})		Used Area (NAND2)	
APPR	113.8	1.00	28488	1.00	115.0	1.01	17093	0.60

Operation	The original operations (two 32 × 32-bit)				The proposed operations (two 16 × 16-bit)			
	Delay (T_{XOR})		Used Area (NAND2)		Delay (T_{XOR})		Used Area (NAND2)	
APPR	113.8	1.00	28488	1.00	74.4	0.65	10108	0.35

6.5 Experimental Evaluation

The sum-of-products presented in this chapter was implemented in Verilog at a structural level. Because our main concern is power savings at the architecture level, we have reused the detailed designs proposed in the previous chapter, and they have not been further optimized for different low-power techniques. Several modules such as AND2 gates for supporting SIMD operation and masking carry signals have been inserted to a sum-of-products unit. We used Synopsys Design Compiler for synthesis and IC Compiler for the placement and routing. We used PrimeTime to measure delay, and Samsung CubicWare to measure power dissipation. The designs were optimized for area with a maximum fan-out of four. The simulation results used an operating voltage of 1.08 volts and a temperature of 125 degrees centigrade. The detailed description of simulation methodologies are described in Appendix A. Table 6.6 shows the comparison of power, delay and area estimates for the original and the proposed

sum-of-products unit. To facilitate comparisons of these estimates, the relative change (percent increase and decrease) in power, delay and area between the original and proposed units is also shown. Based on these estimates, the power, delay and area of the proposed structure is similar to that of the original one. This result implies that adding gates for SIMD and approximate operations does not seem to have a significant effect on power, delay and area.

TABLE 6.6: POWER, DELAY AND AREA FOR THE ORIGINAL AND THE PROPOSED SUM-OF-PRODUCTS UNITS

Multiplier	Power (mW)		Delay (ns)		Area (μm^2)	
Original sum-of-products	12.07	1.00	13.36	1.00	23415	1.00
Proposed sum-of-products	12.55	1.04	14.02	1.05	24117	1.03

Table 6.7 shows power, delay and power-delay estimates for each operation. The SIMD operation (two 16 × 16-bit sum-of-products) has 47% less power, 15% less delay and 55% less power-delay.

TABLE 6.7: POWER, DELAY AND AREA FOR EACH OPERATION IN ONE MULTIPLIER

Operation	Power (mW)		Delay (ns)		Power-Delay Product (nJ)	
Original operation (32 × 32 sum-of-products)	12.07	1.00	13.56	1.00	163.67	1.00
SIMD operation (two 16 × 16 sum-of-products)	6.40	0.53	11.49	0.85	73.54	0.45

In order to compare relative throughput of the original and the SIMD operation, the most effective approach is to use execution time for a program to compare relative power dissipation (or energy) and performance. The execution time required for a program can be written as

Execution time for a program

= Clock cycles for a program × Clock cycle time

= Instructions for a program × Clock cycles per instruction × Clock cycle time (6-6)

[90]

We reused the ARM's architecture, benchmark programs and simulation tools presented in Chapter 4, and we assumed that all input data have value smaller than 16-bit. The comparison results of clock cycle estimates are shown in Table 6.8. In an ideal situation, the SIMD operation would be expected to have a 50% reduction in total clock cycles for benchmark programs. Unfortunately, however, the original and the SIMD operation have the same clock cycles to execute the initialization code; hence, the SIMD operation has between 30% and 47% reduction.

TABLE 6.8: CLOCK CYCLES FOR BENCHMARK PROGRAMS

Clock cycles	FIR Filter (length = 10)		High Pass Filter (length = 10)		Matrix Multiplication (2 × 2)		Euclidean Distance (length = 10)	
Sum-of-products (Original)	97	1.00	98	1.00	146	1.00	179	1.00
Sum-of-products (SIMD)	54	0.56	52	0.53	102	0.70	108	0.60

We can measure the execution time based on Table 6.7 and Table 6.8. Compared to the original operation, the SIMD operation has between 41% and 55% execution time decrease. Because the SIMD operation can be executed simultaneously and the delay of a unit for supporting a SIMD operation is smaller than that for supporting a standard operation, the SIMD operation has a significant reduction in execution time. Throughput is measured in jobs / time unit, so the SIMD operation has an almost double throughput increase.

TABLE 6.9: EXECUTION TIME FOR BENCHMARK PROGRAMS

Execution time (μs)	FIR Filter (length = 10)		High Pass Filter (length = 10)		Matrix Multiplication (2 × 2)		Euclidean Distance (length = 10)	
Sum-of-products (Original)	1.32	1.00	1.32	1.00	1.98	1.00	2.43	1.00
Sum-of-products (SIMD)	0.62	0.47	0.60	0.45	1.17	0.59	1.24	0.51

To measure accuracy, the program was written in Verilog, but our program generates a small number of values less than 16-bit, so we cannot efficiently measure error rate. We also extract values from real image, Lena, which is a representative image for testing, but all data are less than 16-bit. Finally, we created a list of random numbers in a spread sheet program to compute the error probabilities and mean error, and then measure values for 10 iterations. Table 6.10 shows power, delay and error rate for each operation. The possible error probability is close to 100% because errors do not occur when one multiplication results are equal to 0 for 32 × 32_low_error and 32 × 32_high_error or two operands have more than 16 bits for 16 × 16_error. We define mean error is calculated by the average value of (correct value – approximate value) / correct value. The mean error has less than 3% because the smaller value

and LS 16-bit cannot affect the final results under our strategy. The error probability introduced by 16 × 16_error is slightly smaller than other methods. This method provides the largest power savings with the largest mean error among all approximation methods.

TABLE 6.10: POWER, DELAY AND AREA FOR APPROXIMATE SUM-OF-PRODUCTS OPERATIONS

Operation		Power (mW)		Delay (ns)		Error Probability (%)	Mean Error (%)
Original operation (two 32 × 32)		12.07	1.00	13.56	1.00	–	–
Proposed operation	32 × 32_low_error	7.24	0.60	13.84	1.02	100	0.6
	32 × 32_high_error	7.24	0.60	13.84	1.02	100	1.1
	16 × 16_error	3.26	0.27	7.86	0.58	98	2.8

6.6 Summary

In this chapter, we presented a sum-of-products unit capable of supporting SIMD and approximate operations using essentially the same hardware. The SIMD operation can be used to improve the performance of compact input data, and then trade frequency for supply voltage reduction. The arithmetic units that return approximate results consume less power with inaccurate results. The proposed arithmetic unit allows these operations to be performed as an input control based on input operand size and application setting information, which has roughly the same delay as the original structure. Thus the proposed arithmetic unit is useful in digital signal processing applications having diverse input patterns. Detailed experimental results were given to compare the power, delay and area characteristics of each operation.

In this chapter, we considered only the static precision method. This method can be easily used for the error tolerant design, but it is impossible to determine a safe reduced operating

precision for the error tolerant design without prior knowledge. On the other hand, the dynamic precision selection method ensures that arithmetic operations will give a result whose precision can be determined by the results of tests on the initial samples of applications. This technique provides more accurate results, but requires a first sampling period with the initial input for precision determination. This method has one remaining issue that will not concern us formally in this study but which is of some importance. The problem is how to efficiently design the sampling buffer, which is used for determining the cut-off value. A simple sampling buffer holds several random values, and determines the cut-off size based on maximum, minimum and average values. Although it is simple, this method is relatively inefficient, in that it monitors several random values in fixed amount of space. In some cases, data in some entries have extremely small or large values. They represent magnitude in all input data, but do not reflect all data value similarly. This leads to ineffective truncation when the sampling buffer is eventually accessed. More advanced methods use tables to record history information related to several initial data. This history information read from the table is used for predicting cut-off size.

Although a single multiplier is ignored, the error can be made small enough to be acceptable by appropriately selecting the value of the correction value. A method for selecting the correction value which minimizes the average error will also be left for future work. The proposed sum-of-products unit leads to a simpler implementation, but introduce a large amount of error only because it does not include the correction method. Given certain hardware and error constraints, the appropriate number required for error correction can be readily determined.

Chapter 7 Conclusion and Future Work

In this Chapter we evaluate the research contribution of this dissertation. Also, we provide a critique of the existing research and suggest directions for future research investigations.

7.1 Research Contributions

In this dissertation, we have investigated high-level optimization techniques for a low-power sum-of-products design. We have addressed the low-power design problem from two aspects: internal efforts considering a sum-of-products unit and external efforts considering input data characteristics. For internal efforts, we considered the structure optimization of PPR arrays and the final CPA. For external efforts, we considered SIMD and approximate operations to deactivate some portions of a sum-of-products unit. We have also considered a multi-functional sum-of-products operation using input data characteristics. We performed placement and routing experiments to evaluate all the optimization techniques studied and work toward obtaining more precise results. The main contributions are as follows:

- To reduce power dissipation compared to recent multiplier designs, we proposed several optimization techniques of the reduction structure for the array to utilize the low-power features of a LR array. Experiment results have shown that both power and delay are improved considerably with these techniques. Among different optimization techniques for LR array multipliers, a 4-level UL split structure and a voltage islands technique are the primary choices when power is the main concern.

- The low-power array structure optimization techniques mentioned in Chapter 2 were combined with the high-performance CPA optimization techniques. The high-performance CPA is necessary to improve the performance of a sum-of-products unit while maintaining the power dissipation of the array structure. We presented the problem of adding four carry-save vectors (each two carry-save vectors of two arrays) from two PPR arrays and a design strategy specific to arrival time profiles generated by two PPR arrays. We have shown that the proposed design reduces the power dissipation as well as overall latency.

- The sum-of-products operation is frequently used for many digital signal processing applications. However, many DSPs and GPUs use an existing multiplier or a MAC unit to execute sum-of-products operations, and thus take more clock cycles. To reduce the number of clock cycles, we proposed a sum-of-products unit. This sum-of-products unit is designed based on two parallel PPR arrays, [4:2] adders and the final CPAs. We compared with an existing high-performance parallel multiplier and the ARM multiplier. The sum-of-products unit has 46% less execution time with about 12% energy penalty compared to the ARM7TDMI-S multipliers in four different benchmark programs. Also, it consumes less power than a multiplier only when the execution time is the same.

- Most processors include separate dedicated arithmetic units to support each arithmetic operation. This design is less suitable for current DPSs and GPUs because the frequency of arithmetic operations is application-dependent. We have proposed a sum-of-products unit capable of supporting multiple arithmetic operations: multiplication, multiply-add, square, sum-of-squares, and add-multiply computations using essentially the same hardware based on input control signals. Compared to a conventional sum-of-products unit, the MAU-SoP increases power, area and delay slightly of the proposed sum-of-products unit, but allows

several multiplication-related arithmetic computations to be performed as a single operation. Combining several similar operations to execute on the same hardware reduces the area and power compared to separate implementations.

- To further reduce power consumption in a sum-of-products unit with large-dynamic-range input data, we proposed a sum-of-products unit capable of supporting SIMD and approximate operation. The proposed arithmetic unit can perform multiple-precision sum-of-products for accurate and inaccurate results. The proposed techniques have not changed the basic structure of a sum-of-products unit. Instead, the fundamental components are partitioned and ancillary logic gates have been inserted along the gating boundaries. For input data with a large dynamic range, significant power reduction has been shown to be much better than that of the baseline sum-of-products unit in the experiments.

7.2 Future Works

As an attempt to develop several optimization techniques for a low-power sum-of-products unit, the research presented in this dissertation has achieved good experimental results and demonstrated the efficiency of optimization techniques. However, there are some limitations in our research and several future research directions are possible. In this section, we discuss the remaining design issues.

7.2.1 Other Composite Arithmetic Operations

For potential applications which meet the increasing demand for pervasive secure and multimedia information, one direction is to develop other composite arithmetic operations: polynomials and four-dimensional sum-of-products.

The use of mobile computing technology has changed the way people communicate with each other. However, modern mobile devices are vulnerable to various attacks, even though most personal computers are surely protected by antivirus software [117]. Therefore, mobile processors guarantee confidentiality and integrity of data, as well as ensuring reliable data transfer using cryptography hardware [118]. However, cryptography hardware makes intensive use of arithmetic operations; and thus consumes an enormous amount of power. Currently, most cryptography algorithms have no consideration for low-power dissipation even though they are frequently used in mobile devices. Therefore, cryptography issues should be explored in conjunction with low-power solutions. In particular, polynomials, which are mathematical expressions consisting of a sum of terms, each term including constants, variables or exponents, are used for many cryptography applications such as Elliptic Curve Cryptography (ECC). Thus the design of low-power composite arithmetic operations for polynomials is necessary for low-power cryptography hardware design.

The other potential application is 3-Dimensional (3D) graphics, which have become the most popular application dominated by arithmetic operations. Recent GPUs have integrated shaders, which are programmable cores, for their numerical calculations [119][120]. Shaders are used for vertex and pixel processing. Most 3D graphics hardware has 4-way SIMD units to execute an operation of the 4D attribute simultaneously [121][122][123][124][125] because vertex and pixel attributes such as color, vertex, and texture coordinates are represented with 4D vectors in a 3D graphics processor. Thus the 4D sum-of-products operation is the most

frequently used in 3D graphics applications. We need to develop algorithms and implementations of arithmetic units for a 4D sum-of-products, that computes $Z = A \times B + C \times D + E \times F + G \times H$. It will likely reduce the overall power dissipation of the entire 3D graphics system by various techniques to merge the proposed techniques. We need to develop additional composite arithmetic units for applications that frequently use these operations, and compare them with discrete arithmetic units containing normal multipliers and adders.

7.2.2 64-bit Floating-Point Arithmetic Operations

Recently introduced ARMv8 architectures add support for 64-bit floating-point arithmetic operations [126]. In this dissertation, we have considered only a 32 × 32-bit fixed-point sum-of-products unit for simplicity. We have assumed that the proposed methods will be all easily extended to deal with other floating-point representations of operands. Thus the next step of our research is to develop a 64 × 64-bit sum-of-products unit and to determine which techniques presented can be applied for a 64 × 64-bit hardware design. For example, our studies have mainly focused on developing 4-level split array designs, and the length of each array varies significantly with the input data size. It is the perfect candidate to be easily converted into a 64-bit hardware. Specifically, the lower rows consume much more power than the upper rows, and there would be significant power savings if the length could be further reduced. Considering a 32 × 32-bit, an 8-level splitting was not a good example because each part has only four PPs; thus, we avoided an 8-level splitting, due to this short length. However, in the case of a 64 × 64-bit, it would be a good choice because each part has eight PPs. The voltage islands techniques would also be a valuable technique for a 64-bit hardware. We have proposed to exploit the non-uniform arrival time profiles to achieve power savings with minimal performance degradation, and we applied the voltage islands technique to the regions of

non-uniform input generated by the 32-bit array. Since a 64-bit array has similar non-uniform input arrival time, the voltage islands technique can be applied for a 64-bit array.

The 64-bit operation could be more appropriate to be performed using a floating-point arithmetic. Floating-point arithmetic operations are widely used for advanced applications such as 3D graphics and complex signal processing applications, which require a wide dynamic range of arithmetic operation. However, they require additional complex processes such as alignment, normalization and rounding, and thus suffer from more complex implementation, which significantly increase the area, power consumption and latency. We plan to develop low-power schemes for a floating-point sum-of-products unit.

Appendix A Design and Experimental Methodology

In this Appendix we explain the design and simulation methodologies that have been used to obtain the experimental results in this dissertation. There are many choices for logic styles, design descriptions, and simulation methods. Our choices and the reasons are described below.

A.1 Logic Style

There is several logic styles used today. The CMOS and Transistor Transistor Logic (TTL) technologies are attractive for fabricating VLSI circuits. CMOS is a classification of digital circuits that uses Field Effect Transistors (FET) in the design and TTL is also another classification of digital circuits built from Bipolar Junction Transistors (BJT) and resistors. CMOS is the primary technology choice in the semiconductor industry because of its many good features such as small area, low power, relatively simple fabrication process [13]. The advantage of CMOS to TTL is in the greater density of logic gates within the same material. A single logic gate in a CMOS can consist of as little as two FETs while a logic gate in a TTL can consist of a substantial number of parts as extra components are needed. Also, CMOS logic style is robust with respect to voltage scaling and transistor sizing. It has the advantages of generality and ease-of-use as standard cell based technology libraries and logic synthesis techniques are well developed. CMOS logic style is a good choice in most cases if low power and small power-delay product are of major concern, because CMOS circuits do not draw as much power as TTL circuits [127]. For these reasons, we have chosen CMOS standard cell logic style.

A.2 Library

The CMOS standard cell library we have used is Samsung 65nm standard low-power library. As our primary goal is low-power design, we choose low-power library as the underlying technology. The area, delay and power characteristics of basic cells are simplified and shown in Table 3.1. The values in this table are listed only for high-level estimation purpose. Samsung standard cell library also provides some synthesis optimized arithmetic cells, such as a 1-bit FA, but we have not used the arithmetic cells provided because the optimization of the structures of basic cells is one of our research objectives.

A.3 Design and Verification

We have chosen Cadence verification environment including NC-Verilog, Synopsys design environment including Design Compiler, Power Compiler, PrimeTime, IC Compiler, Mentor Graphics hardware/software co-simulation tool, Questa Codelink and Samsung in-house tool, CubicWare. Design Compiler analyzes HDL designs, optimizes and maps HDL designs into netlist using Samsung standard cell library. With given test data, NC-Verilog collects switching activity information by dynamic timing simulation. In our experiments, we have used CubicWare as a power estimation tool based on switching and capacitance information. Detailed evaluation of designs consists of the following steps.

A.3.1 Algorithm Development and Verification using Software

The first step is to create a functional software version of algorithms implemented in hardware. Algorithm verification provides us with the following.

- Analysis of arithmetic algorithms: By writing the pseudo code and seeing how it runs beforehand, we gain full understanding of every algorithm. This allows better insight into the overall algorithm, helps identify bottlenecks, and also is helpful in designing the actual hardware.
- Algorithm verification: Before designing arithmetic units and writing HDL code, it is a simpler and faster way to verify the correctness of the algorithms using software. Note is that not all parts of the algorithm need to be verified because some parts already covered in previous work are fully verified and debugged.

A.3.2 Hardware Design and Verification using RTL

Once the overall algorithm is fully developed and verified, we develop hardware models. Hardware models refer to the identification of physical components and their interrelationships. This description allows us to understand how their components communicate with each other.

Register Transfer Level (RTL) is a design abstraction which models a digital circuit in terms of the flow of signals between hardware registers, and the logical operations performed on those signals. RTL abstraction is used in HDL to create high level representations of digital circuits, from which actual wiring and registers can be derived. We have implemented all proposed models in technology-independent structural Verilog descriptions. To obtain power, delay and area, we have implemented Verilog models, an operand type, two's complement, and an operand size, 32-bit.

After designing RTL modules using HDL codes, the designs have been verified using RTL verification tool. Verilog simulation is conducted to verify the correctness of each design. Verilog codes are usable for the cycle-based simulation phase. Cycle-based simulation evaluates logic functions across clock cycle boundaries. The purpose of this simulation is to

evaluate input stimuli as rapidly as possible. We have verified logic functions and timing. The verification is performed with random data and some special boundary data tests. The random data set used in the proposed arithmetic unit might not hit corner cases. Thus, we have checked whether test cases include corner cases or not after the tool generates test cases. Cadence tool internally uses the Tcl script language to run simulations on benchmark files, and therefore by writing custom scripts file we have automated the simulation process.

A.3.3 Cycle-level Simulation

The clock cycles of the proposed arithmetic units for benchmark programs could be measured by running cycle-level simulation tool with Verilog and compiled ARM assembly code. The Mentor Graphics hardware/software co-simulation tool such as Questa Codelink profiles clock cycles for programs.

A.3.4 RTL Synthesis and Analysis

All proposed arithmetic units are fully synthesizable. The HDL description could be directly translated and optimized to an equivalent netlist for ASIC implementation. Because synthesized results vary according to the constraints, we have used the same constraints to reduce the effects of Synopsys Design Compiler. Because RTL designs determined the delay range, aggressive delay minimization probably leads to much larger area and power. The synthesis mapping objective has been set to minimize area because smaller area generally helps power saving is for a given design. Buffers have been inserted automatically by Synopsys Design Compiler. There are two main ways to estimate interconnect effects in power, delay and area characteristics. One is to extract interconnect information from the actual placement and routing tool for accurate estimation. The other is to use wire load models, which are provided based on statistical information specific to their processes. The use of wire load models in estimation provides a fast

process with less accuracy. To obtain more precise delay and power calculation, we have performed placement and routing processes. Interconnect parameters have been extracted and back-annotated into Synopsys tool. Delays have been obtained from Synopsys PrimeTime, and powers have been obtained from Samsung in-house power estimation tool, CubicWare. Areas have been obtained from Synopsys Design Compiler and IC Compiler.

A.4 Power, Delay and Area Estimation

A.4.1 Data Set

Power dissipation is directly related to input data characteristics. One scheme may consume less power for certain data patterns but consume more power for other data patterns. Therefore, we have prepared random data sets in order to capture power features in recent applications for independent variables in a random environment. Test data set consists of 32-bit pseudo-random data. The static probability of each bit being 1 in random is 0.5 and the toggle rate of each bit is 0.25.

A.4.2 Estimation and Comparisons

The best way to compare different schemes is to fabricate each design and measure the power, delay and area characteristics of actual chips. However, the fabrication is a rather expensive and time consuming process, which makes it impractical to explore many design alternatives. With the advancement of CAD tools, power, delay and area estimations with back-annotated information can achieve the accuracy. Because the primary concern is the relative difference, we have compared different schemes under the same experimental setting and the absolute errors probably tend to go in the same direction and thus have little effect on

the relative comparison. Experimental results with comparisons of the other schemes have been finally presented. For area analysis, we often found that there are not less consistent results obtained from placement and routing tool because there is no consistency in size of white space, which is automatically allocated by tool. The results obtained from synthesis rather than those from placement and routing often have better relations with the power because of the effects of placement and routing tool. In this case, we have provided the synthesis results. We have used a power-delay curve commonly used to assess the merits of designs in digital CMOS design. The feedback from these actual experiments has been used to further refine and modify the solutions to make them achieve much lower power and higher performance in signal processing applications.

Appendix B Abbreviations

ALU	Arithmetic Logic Unit
AP	Application Processor
BJT	Bipolar Junction Transistors
CCA	Conditional-Carry Adder
CLA	Carry-Lookahead Adder
CLG	Carry-Lookahead Generator
CMOS	Complementary Metal Oxide Semiconductor
CPA	Carry-Propagate Adder
CRA	Carry-Ripple Adder
CSELA	Carry-Select Adder
CSK	Carry-Skip Adder
DCT	Detector
DSP	Digital Signal Processor
ECC	Elliptic Curve Cryptography
EO	Even/Odd
FA	Full Adder
FET	Field Effect Transistors
FFT	Fast Fourier Transform
FIR	Finite Impulse Response

HA	Half Adder
ISA	Instruction Set Architecture
ITRS	International Technology Roadmap for Semiconductors
GPU	Graphics Processing Unit
LR	Left-to-Right
LS	Least Significant
LSB	Least Significant Bit
LU decomposition	Lower Upper decomposition
MAC	Multiply-and-Accumulate
MS	Most Significant
MSB	Most Significant Bit
MSG	Modified Sign Generate
PC	Personal Computer
PP	Partial Product
PPG	Partial Products Generator
PPR	Partial Products Reduction
PPRT	Partial Products Reduction Tree
RL	Right-to-Left
RTL	Register Transfer Level
SIMD	Single Instruction Multiple Data
TTL	Transistor Transistor Logic

UL	Upper/Lower
VLSI	Very Large Scale Integration
2D	2-Dimensional
3D	3-Dimensional
4D	4-Dimensional

LR_32	Radix-2 LR using the default [3:2] adder
LR_42	Radix-2 LR using [4:2] adder
LR_42_Split2	Radix-2 2-level UL split array multiplier using [4:2] adder
LR_42_Split4	Radix-2 4-level UL split array multiplier using [4:2] adder
LR_42_Split2_ Radix4	Radix-4 2-level UL split array multiplier using [4:2] adder

MCSELA_11_6_CRA

Modified CSELA made up of CRAs with block sizes of 9−9−9−10−10−11−6

MCSELA_17_CRA

Modified CSELA made up of CRAs with block sizes of 9−9−9−10−10−17

MCSELA_17_CLA4

Modified CSELA comprised of the last group of a one-level 17-bit CLA divided into 4 groups of 4- and 5-bit and remaining five groups of CRAs with block sizes of 9−9−9−10−10−17

MCSELA_17_CLA2

Modified CSELA comprised of the last group of a one-level 17-bit CLA divided into 8 groups of 2- and 3-bit and remaining five groups of CRAs with block sizes of 9−9−9−10−10−17

MCSELA_10_2_CRA

Modified CSELA made up of CRAs with block sizes of 8−8−8−9−9−10−10−2

MCSELA_12_CRA

Modified CSELA made up of CRAs with block sizes of 8−8−8−9−9−10−12

MCSELA_12_CLA4

Modified CSELA comprised of the last group of a one-level 12-bit CLA divided into 3 groups of 4-bit and six groups of CRAs with block sizes of 8−8−8−9−9−10−12

MCSELA_12_CLA2

Modified CSELA comprised of the last group of a one-level 12-bit CLA divided into 6 groups of 2-bit and six groups of CRAs with block sizes of 8−8−8−9−9−10−12

MAU-SoP	Multi-functional arithmetic unit based on sum-of-products
SAAU-SoP	SIMD and approximate arithmetic unit based on sum-of-products

References

[1] S. W. Heo, S. J. Huh and M. D. Ercegovac, "Power optimization of sum-of-products design for signal processing applications, in *Proc. ASAP*, Jun. 2013, pp. 192–197.

[2] A. Abnoufs and J. Rabaey, "Ultra-low-power domain-specific multimedia processors," in *Proc. IEEE Workshop on VLSI Signal Processing, IX*, Oct.–Nov. 1996, pp. 461–470.

[3] F. Catthoor, *Unified low-power design flow for data-dominated multimedia and telecom applications*, Kluwer Academic Publishers, 2000.

[4] C. Chien, *Digital radio systems on a chip: a systems approach*, Kluwer Academic Publishers, 2001.

[5] ITRS, "International technology roadmap for semiconductors," 2012.

[Online] http://www.itrs.net/Links/2012ITRS/2012Chapters/2012Overview.pdf

[6] J. M. Rabaey, *Low power design essentials*, Springer, 2009.

[7] W. Suntiamorntut, *Energy efficient functional unit for a parallel asynchronous DSP*, Ph.D. dissertation, University of Manchester, 2005.

[8] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel computer architecture: a hardware/software approach*, Morgan Kaufmann Publishers, 1998.

[9] P. C. H. Meier, *Analysis and design of low power digital multipliers*, Ph.D. dissertation, Carnegie Mellon University, 1999.

[10] G. K. Yeap, *Practical low power digital VLSI design*, Kluwer Academic Publishers, 1997.

[11] A. P. Chandrakasan and R.W. Brodersen, "Minimizing power consumption in digital CMOS circuits," in *Proc. IEEE*, vol. 83, no. 4, pp. 498–523, Apr. 1995.

[12] A. P. Chandrakasan, S. Sheng and R. W. Brodersen, "Low power CMOS digital design," *IEEE J.Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[13] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, *Digital integrated* circuits*:a designperspective*, 2nd ed., Prentice Hall, 2003.

[14] J. Kao, S. Narendra, and A. Chandrakasan, "Subthreshold leakage modeling and reduction techniques," in *Proc. ICCAD*, Nov. 2002, pp.141–148.

[15] A. Bellaouar and M. Elmasry, *Low-power digital VLSI design: circuits and systems*, Kluwer Academic Publishers, 2012

[16] J. Frenkil, "A multi-level approach to low-power IC design,"*IEEE Spectrum Magazine*, pp.54–60, Feb. 1998.

[17] E. de Angel, *Low power digital multiplication*, Ph.D. dissertation, University of Texas at Austin, 1996.

[18] P. C. H. Meier, R. Rutenbar, and L. Carley, "Exploring multiplier architecture and layout for low power," in *Proc. CICC*, May 1996, pp.513–516

[19] B. Chen and I. Nedelchev, "Power compiler: a gate-level power optimization and synthesis system," in *Proc. ICCD*, Oct. 1997, pp.74–79.

[20] *Power Compiler Reference Manual*, Synopsys Inc., Nov. 2012.

[21] T. Lang, E. Musoll, and J. Cortadella, "Individual flip-flops with gated clocks for low power datapaths,"*IEEE Trans. Circuits and Systems – II: Analog and Digital Signal Processing*, vol. 44, no. 6, pp. 507–516, Jun. 1997.

[22] V. Tiwari, S. Malik, and P. Ashar, "Guarded evaluation: pushing power management to logic synthesis/design,"*IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp.1051–1060, Oct. 1998.

[23] A.G.M. Strollo, E. Napoli, and D. De Caro, "New clock-gating techniques for low-power flip-flops," in *Proc. ISLPED*, Jul. 2000, pp.114–119.

[24] T. Sakuta, W. Lee, and P.T. Balsara, "Delay balanced multipliers for low power/low voltage DSP core," in *Proc. ISLPED*, Oct. 1995, pp.36–37.

[25] C.A. Fabian and M.D. Ercegovac, "Input synchronization in low power CMOS arithmetic circuit design," in *Proc. ACSSC*, Nov. 1997, pp.172–176.

[26] M. K. Gowan, L.L. Biro, and D.B. Jackson, "Power considerations in the design of the Alpha 21264 microprocessor," in *Proc. DAC*, Jun. 1998, pp.726–731.

[27] T. Xanthopoulos and A.P. Chandrakasan, "A low-power IDCT macrocell for MPEG-2 MP@ML exploiting data distribution properties for minimal activity,"*IEEE J. Solid-State Circuits*, vol. 34, No. 5, pp. 693–703, May 1999.

[28] Wu Ye and M.J. Irwin, "Power analysis of gated pipeline registers," in *Proc. IEEE Int. ASIC/SOC Conf.*, Sep. 1999, pp.281–285.

[29] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proc. ICCAD*, Nov. 1993, pp.398–402.

[30] M. J. G. Lewis, *Low power asynchronous digital signal processing*, Ph.D. dissertation, University of Manchester, Oct. 2000.

[31] K. Kim, P.A. Beerel, and Y. Hong, "An asynchronous matrix-vector multiplier for discrete cosine transform," in *Proc. ISLPED*, Jul. 2000, pp.256–261.

[32] C. Park, B. Choi, S. Kim, E. Jung and D. Lee, "Asynchronous array multiplier with an asymmetric parallel array structure," in *Proc. Conf.on Advanced Research in VLSI*, Mar. 2001, pp.202–212.

[33] G. Gerosa, S. Gary, C. Dietz, D. Pham, K. Hoover, J. Alvarez, H. Sanchez, P. Ippolito, T. Ngo, S. Litch, J. Eno, J. Golab, N. Vanderschaaf and J. Kahle, "A 2.2W, 80MHz

superscalar RISC microprocessor,"*IEEE J. Solid-State Circuits*, vol.29, no.12, pp.1440–1454, Dec. 1994.

[34] R. Goering, "Low power Design,"*Special technology report, SCD source*, Sep. 2008.

[35] D. E. Lackey, P. S. Znchowski, T. R. Eednar, D. W. Stout, S. W. Gould, and J. M. Cobn, "Managing power and performance for system-on-chip designs using voltage islands, " in *Proc. ICCAD*, Nov. 2002, pp. 195–202.

[36] R. W. Brodersen, M. A. Horowitz, D. Markovic, B. Nikolic, and V. Stojanovic, "Methods for true power minimization," in *Proc. ICCAD*, Nov. 2002, pp.35–42.

[37] D. Markovic, V. Stojanovic, B. Nikolic, M. A. Horowitz, and R. W. Brodersen, "Methods for true energy-performance optimization," *IEEE J.Solid-State Circuits*, vol. 39, no. 8, pp. 1282–1293, Aug. 2004.

[38] U. Ko, P. T. Balsara, and W. Lee, "A self-timed method to minimize spurious transitions in low power CMOS circuits," in *Proc. ISLPED*, Oct. 1994, pp. 62–63.

[39] I. Koren, *Computer arithmetic algorithms*, 2nd ed., A. K. Peters, Natick, MA, 2001.

[40] B. Parhami, *Computer arithmetic: algorithms and hardware designs*, 2nd ed., Oxford University Press, USA, 2009.

[41] M. D. Ercegovac and T. Lang, *Digital arithmetic*, Morgan Kaufmann Publishers, Elsevier Science Ltd., 2003.

[42] A. D. Booth, "A signed binary multiplication technique,"*The Quarterly J. of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[43] E. de Angel and E. E. Swartzlander, Jr., "Low power parallel multipliers," in *Proc. IEEE Workshop on VLSI Signal Processing, IX*, Oct.-Nov. 1996, pp. 199–208.

[44] R. Fried, "Minimizing energy dissipation in high-speed multipliers," in *Proc. ISLPED*, Aug. 1997, pp. 214–219.

[45] A. P. Chandrakasan, R. Allmon, A. Stratakos, and R. W. Brodersen, "Design of portable systems," in *Proc. CICC*, May 1994, pp. 259–266.

[46] Z. Huang and M. D. Ercegovac, "Number representation optimization for low-power multiplier design," in *Proc. SPIE on Advanced Signal Processing Algorithms, Architectures, and Implementations*, vol. 4791, Jul. 2002, pp. 345–356.

[47] D. Crookes and M. Jiang, "Using signed digit arithmetic for low-power multiplication,"*Electron. Lett.*, vol. 43, no. 11, pp. 613–614, May 2007.

[48] C. S. Wallace, "A suggestion for a fast multiplier", *IEEE Trans. Comput.*, vol. 13, no. 2, pp. 14-17, Feb. 1964.

[49] L. Dadda, "Some schemes for parallel multipliers", *Alta Frequenza*, vol. 34, pp. 349-356, Mar. 1965.

[50] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm,"*IEEE Trans. Comput*., vol. C-22, no. 2, pp. 1045–1047, Dec. 1973.

[51] Z. Huang, *High-level optimization techniques for low power multiplier design*, Ph.D. dissertation, University of California at Los Angeles, 2004.

[52] J. Di and J. S. Yuan, "Power-aware pipelined multiplier design based on 2-dimensional pipeline gating," in *Proc. GLSVLSI*, Apr. 2003, pp. 64–67.

[53] S. Hong, S. Kim, M. C. Papaefthymiou, and W. E. Stark, "Low power parallel multiplier design for circuit techniques for CMOS low-power high-performance multipliers DSP applications through coefficient optimization," in *Proc. IEEE Int. ASIC/SOC Conf.*, Sep. 1999, pp. 286–290.

[54] A. A. Fayed and M. A. Bayoumi, "A novel architecture for low-power design on parallel multipliers," in *Proc. IEEE Comput. Soc. Workshop on VLSI*, Apr. 2001, pp. 149–154.

[55] M. Bhardwaj, R. Min, and A. P. Chandrakasan, "Power-aware systems," in *Proc. ACSSC*, vol. 2, Oct. 2000, pp. 1695–1701.

[56] N.-Y. Shen and O. T.-C.Chen, "Low-power multipliers by minimizing switching activities of partial products," in *Proc. ISCAS*, vol. 4, May 2002, pp.93–96.

[57] O. T.-C. Chen, S. Wang, and Y.-W. Wu, "Minimization of switching activities of partial products for designing low-power multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 3, pp. 418–433, Jun. 2003.

[58] S.-R. Kuang and J.-P. Wang, "Design of power-efficient configurable booth multiplier," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 3, pp. 568–580, Mar. 2010.

[59] J. Choi, J. Jeon, and K. Choi, "Power minimization of functional units by partially guarded computation," in *Proc. ISLPED*, Jul. 2000, pp. 131–136.

[60] Z. Huang and M. D. Ercegovac, "Two-dimensional signal gating for low-power array multiplier design," in *Proc. ISCAS*, vol. 1, Aug. 2002, pp. 489–492.

[61] G. Economakos and K. Anagnostopoulos, "Bit level architectural exploration technique for the design of low power multipliers," in *Proc. ISCAS*, May 2006, pp. 1483–1486.

[62] J.-N. Ohban, V. G. Moshnyaga, and K. Inoue, "Multiplier energy reduction through bypassing of partial products," in *Proc. APCCAS*, vol. 2, Oct. 2002, pp. 13–17.

[63] M.-C. Wen, S.-J. Wang, and Y.-N. Lin, "Low power parallel multiplier with column bypassing," in *Proc. ISCAS*, May 2005, pp. 1638–1641.

[64] C. Lemonds and S. S. Mahant-Shetti, "A low power 16 by 16 multiplier using transition reduction circuitry, " in *Proc. Int. Workshop on Low Power Design*, Apr. 1994, pp. 139–140.

[65] S. S. Mahant-Shetti, P. T. Balsara, and C. Lemonds, "High performance low power array multiplier using temporal tiling," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst*., vol. 7, no. 1, pp. 121–124, Mar. 1999.

[66] Q. L. Macsorley, "High Speed Arithmetic in Binary Computers", in *Proc. IRE*, vol. 49, pp. 67-91, Jan. 1961.

[67] T. K. Callaway and E. E. Swartzlander Jr., "Power-delay characteristics of CMOS multipliers," in *Proc. ARITH*, Jul. 1997, pp. 26–32.

[68] Z. Yu, L. Wasserman, and A.N. Willson, Jr. "A painless way to reduce power dissipation by over 18% in Booth-encoded carry-save array multipliers for DSP," in *Proc. IEEE Workshop on Signal Processing Syst.*, Oct. 2000, pp.571–580.

[69] A. Goldovsky, B. Patel, M. J. Schulte, R.Kolagotla, H. Srinivas, and G. Burns, "Design and implementation of a 16 by 16 low-power two's complement multiplier," in *Proc. ISCAS*, vol.5, May 2000, pp.345–348.

[70] L. Ciminiera and P. Montuschi, "Carry-save multiplication schemes without final addition,"*IEEE Trans.Comput.*, vol.45, no.9, pp.1050–1055, Sep. 1996.

[71] M. D. Ercegovac and T. Lang, "Fast multiplication without carry-propagate addition," *IEEE Trans. Comput.*, vol. 39, no. 11, pp. 1385–1390, Nov. 1990.

[72] Z. Huang and M.D. Ercegovac, "Low power array multiplier design by topology optimization," in *Proc. SPIE Advanced Signal Processing Algorithms, Architectures, and Implementations XII*, vol. 4791, Jul.2002, pp. 424–435.

[73] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: a view from Berkeley," *Technical report, no. UCB/EECS-2006-183*, EECS, U.C. Berkeley.

[74] J. Iwamura, S. Kazuoa, K. Minorua, and S. Taguchi, "A CMOS/SOS multiplier," in *Proc. ISSCC*, Feb. 1984, pp. 92–93.

[75] P. F. Stelling and V. G. Oklobdzija, "Design strategies for optimal hybrid final adders in a parallel multiplier", *Journal of VLSI Signal Processing*, vol. 14, no. 3, pp.321–31, Dec. 1996.

[76] P. F. Stelling and V. G. Oklobdzija, "Design strategies for the final adder in a parallel multiplier", in *Proc. Asilomar Conf. Signals, Syst. and Comput.*, vol. 1, Oct–Nov. 1995, pp. 591–595.

[77] S. W. Heo, S. J. Huh, and M. D. Ercegovac, "Power optimization ina parallel multiplier using voltage islands," in *Proc. ISCAS*,May 2013., pp. 345–348.

[78] J. Lo, "A fast binary adder with conditional carry generation", *IEEE Trans. Comput.*, vol. 46, no. 2, pp. 248-253, Feb. 1997.

[79] S. Cheng, "64-bit pipeline conditional carry adder with MTCMOS TSPC logic", in *Proc.MWSCAS*,Aug. 2007, pp. 879–882.

[80] V. G. Oklobdzija, "Design and analysis of fast carry-propagate adder under non-equal input signal arrival profile", in *Proc. Asilomar Conf. Signals, Syst. and Comput.*, vol. 1, Nov. 1994, pp. 1398–1401.

[81] T. Y. Chang and M. J. Hsiao, "Carry-select adder using single ripple-carry adder,"*Electron. Lett.*, vol. 34, no. 22, pp. 2101–2103, Oct. 1998.

[82] Y. Kim and L. Kim, "64-bit carry-select adder with reduced area,"*Electron. Lett.*, vol. 37, no. 10, pp. 614–615, May 2001.

[83] B. Ramkumar and H. M. Kittur, "Low-power and area-efficient carry select adder"*IEEE Trans.VLSI Syst.*, vol. 20, no. 2, pp. 371–375, Feb. 2012.

[84] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSPprocessor fundamentals: architectures and features*, IEEE Press, 1997.

[85] Analog Devices, *DSP-218x DSP Instruction Set Reference*, Feb. 2001

[86] Texas Instruments, *TMS320C55x DSP Mnemonic Instruction Set Reference Guide*, Oct. 2002.

[87] Z. Huang and M. D. Ercegovac, "High-performance low-power left-to-right array multiplier design," *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 272–283, Mar. 2005.

[88] Z. Huang and M. D. Ercegovac, "High-performance left-to-right array multiplier design", in *Proc. ARITH*, Jun. 2003, pp. 4–11.

[89] J. Leijten, J. van Meerbergen, and Jochen Jess, "Analysis and reduction of glitches in synchronous networks," in *Proc. EDTC*, Mar. 1995, pp. 398-403.

[90] J. L. Hennessey and D. A. Patterson,*Computer organization and design: the hardware/software interface*, Morgan Kaufman, 2005.

[91] S. Furber, *ARM system architecture*, Addison-Wesley, 1996.

[92] ARM, *ARM7TDMI™ technical reference manual*.

[93] R. Gonzales and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE J. Solid-State Circuits*, vol. 31, no. 9, Sep. 1996

[94] M. J. Schulte, L. Marquette, S. Krithivasan, E. G. Walters, and J. Glossner, "Combined multiplication and sum-of-squares units," in *Proc. ASAP*, Jun. 2003, pp. 204-214.

[95] R. Hegde and N. R. Shanbhag, "Soft digital signal processing," *IEEE Trans. VLSI Syst*., vol. 9, no. 6, pp. 813–823, Dec. 2001.

[96] D. Brooks and M. Martonosi, "Value-based clock gating and operation packing: dynamic strategies for improving processor power and performance," *ACM Trans. Comput. Syst*., vol. 18, no.2, pp.89–126, May 2000.

[97] Khronos Group, OpenGL-ES 2.0. [Online] http://www.khronos.org

[98] P. Ranganathan, S. Adve, and N. Jouppi, "Performance of image and video processing with general-purpose processors and media ISA extensions," in *Proc. ISCA*, vol. 27, May 1999, pp. 124-135.

[99] S.K. Raman, V. Pentkovski, and J. Keshava, "Implementing streaming SIMD extensions on the pentium III processor,"*IEEE Micro*, vol. 20, pp. 47-57, Jul. 2000.

[100] H. Nguyen and L.K. John, "Exploiting SIMD parallelism in DSP and multimedia algorithms using the AltiVec technology,"in *Proc. ICS*, pp. 11-20, Jun. 1999.

[101] ZiiLabs Corporate Website. [Online]. Available:

https://secure.ziilabs.com/products/processors/zms05.aspx

[102] ARM Corperate Website. [Online]. Available:

http://www.arm.com/products/processors/technologies/dsp-simd.php

[103] MIT Websiet. The surprising usefulness of sloppy arithmetic, MIT news, January 3, 2011

[Online]. Available: http://web.mit.edu/newsoffice/2010/fuzzy-logic-0103.html

[104] R. Hegde and N. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *Proc. ISLPED*, Aug. 1999, pp. 30–35.

[105] B. Shim, S. Sridhara, and N. Shanbhag, "Reliable low-power digital signal processing via reduced precision redundancy" *IEEE Trans. VLSI Syst.*, vol. 12, no. 5, pp. 497 – 510, May 2004.

[106] G. Varatkar and N. Shanbhag, "Energy-efficient motion estimation using error-tolerance," in *Proc.ISLPED*, Oct. 2006, pp. 113–118.

[107] D. Mohapatra, G. Karakonstantis, and K. Roy, "Significance driven computation: A voltage-scalable, variation-aware, quality-tuning motion estimator," in *Proc. ISLPED*, Aug. 2009, pp. 195–200.

[108] N. Banerjee, G. Karakonstantis, and K. Roy, "Process variation tolerant low power dct architecture," in *Proc. DATE*, Apr. 2007, pp. 1 – 6.

[109] G. Karakonstantis, D. Mohapatra, and K. Roy, "System level dsp synthesis using voltage overscaling, unequal error protection and adaptive quality tuning," in *Proc. IEEE Workshop on Signal Processing Systems*, Oct. 2009, pp. 133 – 138.

[110] L. N. Chakrapani, K. K. Muntimadugu, L. Avinash, J. George, and K. V. Palem, "Highly energy and performance efficient embedded computing through approximately correct arithmetic: a mathematical foundation and preliminary experimental validation," in *Proc. CASES*, Aug. 2008, pp. 187–196.

[111] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *Proc. ISIC*, Dec. 2009, pp. 69 –72.

[112] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan and K. Roy, "IMPACT: IMPrecise adders for low-power Approximate Computing," in *Proc. ISLPED*, Aug. 2011, pp. 409–414.

[113] M. J. Schulte and E. E. Swartzlander, Jr., "Truncated multiplication with correction constant," in *Proc. VLSI Signal Processing, VI*, pp. 388–396, Oct. 1993.

[114] M. J. Schulte, J. E. Stine, and J. G. Jansen, "Reduced power dissipation through truncated multiplication," in *Proc. IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, Mar. 1999, pp. 61–69.

[115] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. VLSID*, Jan. 2011, pp. 346 –351.

[116] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power in a multiplier architecture," *J. Low Power Electron.*, vol. 7, pp. 482–489, Dec. 2011.

[117] N. J. Rubenkin, "You need a smartphone security suite,"*PC Mag.*, Sep. 2010.

[118] P. Kornerup, P. Montuschi, J.-M. Muller, and E. Schwarz, "Guest editor's introduction: Special section on computer arithmetic,"*IEEE Trans. Comput.*, vol. 58, no. 2, pp. 145–147, Feb. 2009.

[119] E. Lindholm, M. J. Kilgard, and H. Moreton, "A user-programmable vertex engine," in *Proc. SIGGRAPH*, Aug. 2001, pp. 149–158.

[120] D. Blythe, "The direct3D 10 system,"*ACM Trans. Graphics*, vol. 25, no. 3, pp. 724–734, Jul. 2006.

[121] D. Kim, K. Chung, C.-H. Yu, C.-H. Kim, I. Lee, J. Bae, Y.-J. Kim, J.-H. Park, S. Kim, Y.-H. Park, N.-H. Seong, J.-A. Lee, J. Park, S. Oh, S.-W. Jeong, and L.-S. Kim, "An

SoC with 1.3 Gtexels/sec 3-D graphics full pipeline engine for consumer applications,"*IEEE J. Solid-State Circuits*, vol 41, no. 1, pp. 71–84, Jan. 2006.

[122] C. -H. Yu, K. Chung, D. Kim, and L. -S. Kim, "A 120Mvertices/s multi-threaded VLIW vertex processor for mobile multimedia applications," in *Proc. ISSCC*, Feb. 2006, pp. 1606–1615.

[123] C. -H. Yu, K. Chung, D. Kim, and L. -S. Kim, "A 186Mvertices/s 161mW floating-Point vertex processor for mobile graphics systems," in *Proc. CICC*, Sep. 2007, pp. 579–582.

[124] S. -H. Kim, J. -S. Yoon, C. -H. Yu, D. Kim, K. Chung, H. S. Lim, H. -W. Park, and L. -S. Kim, "A 36 fps SXGA 3D display processor with a programmable 3D graphics rendering engine,"*IEEE J. Solid-State Circuits*, vol. 43, no. 5, pp. 1247–1259, May 2008.

[125] S. M. Mueller, C. Jacobi, H. -J. Oh, K. D. Tran, S. R. Cottier, B. W. Michael, H. Nishikawa, Y. Totsuka, T. Namatame, N. Yano, T. Machida, and S.H. Dhong, "The vector floating-point unit in a synergistic processor element of a Cell processor," in *Proc. ARITH*, Jun. 2005, pp. 59–67.

[126] ARM, ARM discloses technical details of the next version of the ARM architecture, Oct. 2011.

http://www.arm.com/about/newsroom/arm-discloses-technical-details-of-the-next-version-of-the-arm-architecture.php

[127] R. Zimmermann and W. Fichtner, "Low-power logic styles: CMOS versuspass -transistor logic,"*IEEE J. Solid-State Circuits*, vol.32, no.7, pp.1079-1090, Jul. 1997.