Winter 2007

Power-Aware Design Methodologies for FPGA-Based Implementation of Video Processing Systems

Hau Trung Ngo
Old Dominion University

POWER-AWARE DESIGN METHODOLOGIES FOR FPGA-BASED IMPLEMENTATION OF VIDEO PROCESSING SYSTEMS

By

Hau Trung Ngo
B. S. May 2001, Old Dominion University
M. S. May 2003, Old Dominion University

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

ELECTRICAL AND COMPUTER ENGINEERING

OLD DOMINION UNIVERSITY
December 2007

Approved by:

Vijayan K. Asari (Director)

Shirshak K. Dhali (Member)

Min Song (Member)

Ravi Mukkamala (Member)

ABSTRACT

POWER-AWARE DESIGN METHODOLOGIES FOR FPGA-BASED IMPLEMENTATION OF VIDEO PROCESSING SYSTEMS

Hau Trung Ngo
Old Dominion University
Director: Dr. Vijayan Asari

The increasing capacity and capabilities of FPGA devices in recent years provide an attractive option for performance-hungry applications in the image and video processing domain. FPGA devices are often used as implementation platforms for image and video processing algorithms for real-time applications due to their programmable structure that can exploit inherent spatial and temporal parallelism. While performance and area remain as two main design criteria, power consumption has become an important design goal especially for mobile devices. Reduction in power consumption can be achieved by reducing the supply voltage, capacitances, clock frequency and switching activities in a circuit. Switching activities can be reduced by architectural optimization of the processing cores such as adders, multipliers, multiply and accumulators (MACs), etc. This dissertation research focuses on reducing the switching activities in digital circuits by considering data dependencies in bit level, word level and block level neighborhoods in a video frame.

The bit level data neighborhood dependency consideration for power reduction is illustrated in the design of pipelined array, Booth and log-based multipliers. For an array multiplier, operands of the multipliers are partitioned into higher and lower parts so that the probability of the higher order parts being zero or one increases. The gating technique for the pipelined approach deactivates part(s) of the multiplier when the above special values are detected. For the Booth multiplier, the partitioning and gating technique is integrated into the Booth recoding scheme. In addition, a delay correction strategy is developed for the Booth multiplier to reduce the switching activities of the sign extension part in the partial products. A novel architecture design for the computation of log and

inverse-log functions for the reduction of power consumption in arithmetic circuits is also presented. This also utilizes the proposed partitioning and gating technique for further dynamic power reduction by reducing the switching activities.

The word level and block level data dependencies for reducing the dynamic power consumption are illustrated by presenting the design of a 2-D convolution architecture. Here the similarities of the neighboring pixels in window-based operations of image and video processing algorithms are considered for reduced switching activities. A partitioning and detection mechanism is developed to deactivate the parallel architecture for window-based operations if higher order parts of the pixel values are the same. A neighborhood dependent approach (NDA) is incorporated with different window buffering schemes. Consideration of the symmetry property in filter kernels is also applied with the NDA method for further reduction of switching activities.

The proposed design methodologies are implemented and evaluated in a FPGA environment. It is observed that the dynamic power consumption in FPGA-based circuit implementations is significantly reduced in bit level, data level and block level architectures when compared to state-of-the-art design techniques. A specific application for the design of a real-time video processing system incorporating the proposed design methodologies for low power consumption is also presented. An image enhancement application is considered and the proposed partitioning and gating, and NDA methods are utilized in the design of the enhancement system. Experimental results show that the proposed multi-level power aware methodology achieves considerable power reduction. Research work is progressing in utilizing the data dependencies in subsequent frames in a video stream for the reduction of circuit switching activities and thereby the dynamic power consumption.

ACKNOWLEDGEMENTS

My deepest gratitude is to my advisor, Dr. Vijayan K. Asari for his advice and support during my study at Old Dominion University. He has spent countless hours patiently guiding me throughout these years. His encouragement has helped me overcome many challenges during my research work.

I am sincerely grateful to members of my dissertation committee, Dr. Shirshak K. Dhali, Professor and Chair of the Department of Electrical and Computer Engineering; Dr. Min Song, Associate Professor of the Department of Electrical and Computer Engineering; and Dr. Ravi Mukkamala, Professor of the Department of Computer Science. Their invaluable comments and suggestions are greatly appreciated.

I would also like to thank my family for their encouragement and support. They are the greatest source of happiness.

TABLE OF CONTENTS

Chapter **Page**

Chapter **Page**

LIST OF FIGURES

Figure **Page**

Figure **Page**

LIST OF TABLES

Chapter I

INTRODUCTION

As the capacity of VLSI chips keeps growing, more processing units are being integrated in[1]to a single chip. It is possible to fit an entire signal processing system in a single chip in order to significantly improve the performance. However, these signal processing systems also consume a considerable amount of energy. While performance and area are still very important considerations for system designers, power consumption has become an increasingly critical concern. Operating frequency and chip capacity have grown steadily as technology continues to improve. With increased capacity and higher operating frequency, more capabilities are added to portable electronic devices that enable these devices to integrate more computationally intensive applications such as multimedia and image and video processing applications. Since portable devices are operated with a battery, it is particularly important to incorporate low power design methodologies in order to prolong battery life.

Most existing low power design methodologies for video processing systems focus on the design techniques for functional and computational units. Many researchers have presented low power design techniques for multiplication which is one of the fundamental and frequently used operations in signal processing applications. Since power consumption is directly related to switching activities of each computational module in a system, most low power design techniques for functional units consider the input data switching patterns to reduce the switching rate. Image and video frame pixels have a very high spatial redundancy such that the higher bits in the binary representations of the pixels in a block are usually the same (only the lower bits are different). In addition, the difference in the pixel values in the same neighborhood is usually small. Window-based operations, such as two-dimensional convolution (2-D convolution) and image filtering, are the most common operations in image and video processing applications, which have high redundancy in magnitude variations of neighboring pixels.

The reference model of this work is *IEEE Transactions on Circuits and Systems for Video Technology*

Computations for video processing algorithms require a large number of complex operations on every pixel in an image. Although processing speed in general purpose computers has increased significantly in recent years, these systems have processing power only to support real-time processing of very small size video frames. For larger image size, real-time processing in general purpose computers is not possible because these systems can only process data sequentially with significant memory overhead. Digital Signal Processors (DSPs) can be used to support real-time video processing applications. The use of DSPs for a video processing application provides some improvement compared to software means in general purpose computers, by employing 'limited' parallelization in the core processor and utilizing the optimized DSP library for some complex operations. Still, the approach using DSPs does not take full advantage of the inherent parallelism in the video processing algorithms. Hence, processing of 25 to 30 large size video frames (such as 1024×1024 frames) per second in DSPs is still not possible.

The intensive computations in many video processing applications require massive parallel processing capabilities to support real-time processing of a large size video stream. Field Programmable Gate Array (FPGA) provides an attractive solution to this problem because of its high density, high performance and re-configurability to support specific applications. An FPGA is a semiconductor device with programmable logic components and programmable interconnect networks. Logic components within an FPGA can be programmed to perform standard logic gates, such as AND, XOR, etc., or more complex combinational logic components such as decoders, adders, etc. An FPGA offers a good combination of the flexibility of a general purpose computer and hardware-based high speed operation that is comparable to an Application Specific Integrated Circuit (ASICs). In recent years, a new generation of FPGA with embedded DSP modules has become a preferred choice for professional video processing systems. An architecture design for FPGA technology can fully exploit the data and I/O parallelism in video processing applications. Furthermore, FPGA technology allows developers to upgrade designs promptly and easily to satisfy new standards and requirements.

This dissertation addresses a multi-level, power-aware design methodology for video processing applications. The multi-level methodology refers to techniques for design of low power computational units, such as a multiplier in the logic/arithmetic level, and the design of low power functional modules such as a 2-D convolution module in the architecture level with neighborhood data dependency considerations. The main hypothesis of this research is that the proposed multi-level power-aware design approaches produce an effective and more power-efficient video processing system.

1.1. Low Power Design Approach

Digital CMOS circuits have two major types of power dissipation -- dynamic and static power. Dynamic power is related to the switching activities or the logic changes of the circuits. Static power is mostly related to the fabrication technology parameters. The total power consumption is described in the following equations:

$$P_{total} = P_{dynamic} + P_{static} \tag{1.1}$$

$$P_{dynamic} = P_{cap} + P_{sckt} \tag{1.2}$$

$$P_{cap} = \alpha \cdot f_{clk} \cdot C_L \cdot V_{DD}^2 \tag{1.3}$$

$$P_{sckt} = \alpha \cdot f_{clk} \cdot I_{peak} \cdot \left(\frac{t_r - t_f}{2} \right) \cdot V_{DD} \tag{1.4}$$

$$P_{static} = I_{static} \cdot V_{DD} \tag{1.5}$$

where P_{cap} is the dynamic power due to capacitance charging and discharging, P_{sckt} is the average power due to short circuit current, C_L is the capacitance of the load, f_{clk} is the clock frequency, α is the switching activity probability, V_{DD} is the supply voltage, I_{peak} is the peak current, I_{static} is the static current, and t_r and t_f are the rise time and fall time of the short circuit current, respectively.

In general, static power consumption in CMOS circuits is much lower than dynamic power consumption. Typically, dynamic power consumption is the dominant variable in

CMOS circuits when they are in active mode during operations. The leakage power occurs mostly when the CMOS circuits are in idle mode so that very little or no switching activities are present. Since static power is mainly dependent on the fabrication technology, the research focus in this dissertation is on the reduction of dynamic power consumption. Low power design techniques must be developed to reduce one or more variables in equation (1.3) while maintaining functionality, without sacrificing the speed, performance of the circuits. The objective for the design of low dynamic power consumption circuits is now primarily the task of minimizing switching activity α, loading capacitance C_L, clock frequency f_{clk} or supply voltage V_{DD}. Reducing the loading capacitance (C_L) can be done by low level (transistor level) design. The capacitance can be reduced by reducing the transistor sizes and wire lengths, but this can degrade the performance. Reduction of the operating clock frequency when the computational load does not require full capacity of the processor is a common approach to reduce dynamic power. This approach will also degrade the performance and it is difficult to implement in custom architectures. With fabrication technology supporting multiple supply voltages, a lower voltage can be used in non-critical computational modules. This approach is used commonly in modern, low-power microprocessors. New generations of FPGA devices also begin to offer multiple-voltage operating modes for low power applications. Of all the variables in equation (1.3), architecture designers have the most control over the switching activity, α. There have been extensive research works in developing techniques to reduce switching activities. Some of the most common approaches include clock and data gating control to disable data-path units when outputs are not used, re-timing and pipelining to reduce glitches on large data buses, and recoding techniques to reduce the amount of combinational logic blocks in multipliers.

All of these low-power design techniques consider the immediate input data patterns to fine tune the design for the reduction of dynamic power consumption. Characteristics of the data stream to the input of the computational modules as well as the dependency of pixels in the neighboring window within the image are considered in this dissertation research for the reduction of dynamic power consumption.

1.2. Specific Objectives

The research works in this dissertation address power-aware design methodologies by reducing switching activities in computational modules at various design levels. The proposed power-aware design technique exploits the special characteristics of high correlation in the bit presentations, the repeated values, zero values, and insignificant bits to reduce switching activities.

The characteristics in the bit representation of the input data stream to computational modules such as adders, multipliers, dividers, etc. are considered for the design of power-aware arithmetic units. At this logical/arithmetic level, special bit patterns in the inputs are detected, and decisions are made to disable computational units and data buses. A partitioning and gating technique is developed and applied to the design of pipelined array and Booth multipliers, and their performance and power consumption are analyzed and compared with those of conventional design architectures. Approximation techniques for logarithm and inverse-logarithm of base two are developed and implemented in an FPGA environment. Error analysis and performance evaluations are carried out for the approximation methods. The log-based multiplier incorporating the partitioning and gating technique is also designed, and its performance and power consumption are analyzed and compared with those with conventional designs.

Secondly, the power-aware technique based on a neighborhood dependent approach is considered at the architecture level where window-based operations are employed. The architectural level approach refers to the overall computation of the window-based operation where the block of pixels in the neighborhood is examined for possible redundant computations. A technique to analyze pixels in a neighboring window is developed to detect higher bit redundancy based on on-chip buffering schemes. Furthermore, symmetry property in filtering operations is considered to further reduce the number of operations thereby reducing the switching activities. Standard 2-D convolutions with common kernels are used to evaluate the performance and overhead and dynamic power consumption of the proposed approach.

The proposed design methodologies are implemented with VHDL descriptions utilizing commercial FPGA development software environments from Altera. The VHDL modules are mapped onto newer generations of FPGA devices such as Cyclone II and Stratix III from Altera, and the power consumption of the implementation is evaluated using power analyzer tools such as PowerPlay from Altera. Several test images are used as input vectors for accurate estimation of power consumption. Experimental results are analyzed, and comparisons with different techniques are presented.

1.3. Organization

This dissertation is organized as follows. Chapter II presents a literature review of current and existing low power design methodologies for arithmetic modules and multimedia systems. Existing techniques in physical, logic, architectural and system levels are studied and discussed in this chapter. A detailed discussion of design methodologies to reduce dynamic power consumption by minimizing switching activities in computational logic blocks is also presented.

Chapter III presents the proposed low power design for arithmetic modules, which includes the design of a pipelined array multiplier and radix-4 multiplier. This chapter also presents a comparison of the performance and power consumption of the proposed multipliers with respective conventional designs. Logarithmic domain computations for power-aware design are presented in chapter IV. The approximation methods for logarithm and inverse-logarithm of base two are discussed and analyzed. Design of log-based multipliers and their implementations in the FPGA environment and performance comparison with state-of-the art techniques are also presented in this chapter.

The neighborhood dependent approach for the design of a low power window-based operation is addressed in chapter V. The proposed design technique considers the neighboring pixels in the processing windows to detect and eliminate redundant or unnecessary computations for power reduction. A novel on-chip detection technique is

developed for this approach. Data partitioning methodology is employed in the procedure to eliminate unnecessary operations. Experimental results and comparisons are also presented in this chapter.

Chapter VI discusses the power-aware design methodology in an application-specific architecture. A nonlinear video enhancement application is used to apply the neighborhood-dependent approach with the low power arithmetic units proposed in this dissertation research. Discussion of hardware algorithm development and implementation is provided in this chapter. Experimental results are provided for analysis in terms of performance, area and power dissipation.

Chapter VII concludes this dissertation with a summary of various contributions and suggestions for future work. Proposed methodologies for power-ware design of FPGA-based implementation of video processing systems are summarized. The description of the ongoing research activities on power reduction methodologies based on data dependencies in subsequent frames of a video stream is also presented in this chapter.

Chapter II

LITERATURE REVIEW

Many design techniques have been developed for power dissipation minimization at various design levels ranging from high-level software programming to technological innovations. All methods aim to reduce one of the variables in equation (1.3). Common approaches include methods to scale voltage or clock frequency in processors, compress instructions to reduce logic toggle in buses, reduce load capacitance by balancing layout, and minimize switching activities within a clock cycle in circuits. In general, reducing clock frequency, voltage or load capacitance requires careful analysis of the target applications because it might degrade performance of the overall system. A general survey of various techniques that have been developed to reduce power dissipation in different levels of the design cycle is presented in this chapter.

2.1. System-Level Power-Aware Design Techniques

At the system level, designers generally address the power-aware design issue by employing programming techniques to minimize external memory accesses. This relies on compiler and power management software to scale voltage and clock frequency and to pack multiple instructions in an instruction package to reduce logic toggle on the instruction bus and instruction cache.

In [1], the authors provide a survey of some general methods for system-level power-aware design in real-time systems. These methods primarily apply to microprocessor-based designs. The survey covers techniques that deal with communication between caches and buses, instruction set utilization, voltage and frequency scaling, network and operating system model. One of the most common approaches is to scale voltage and clock frequency. This approach is known as dynamic voltage scaling (DVS). Many commercial processors, including Pentium M, mobile Pentium 4, AMD's Athlon and Transmeta's Crusoe and Efficieon processors, use this approach. These processors allow software to monitor the activities and adjust the clock frequency and voltage settings

accordingly. For example, Transmeta's Crusue and Efficieon processors use a software layer known as the Code Morphing Software and a power management unit (LongRun) to dynamically change voltage and frequency to provide the performance needed by the application at any moment [2-3]. These are Very Long Instruction Word (VLIW) processors that require Code Morphing Software to translate multiple x86 instruction words into a long instruction that maps to multiple functional units. Krishna et al. [4] proposed a scheduling technique for optimizing voltage and clock frequency by a two-phase procedure to reduce energy consumption. The offline phase of the procedure computes the estimated worst case execution time for the tasks, and the online phase monitors. It adjusts the voltage level to guarantee the duration of all tasks that will not exceed the worst case time. Another DVS approach is presented in [5]. In this method, the authors proposed double-sample pipeline latches to adjust clock frequency in the pipelined processor. An error detection mechanism is needed to decide if error occurs and a fall-back procedure is needed. Most of the techniques following the DVS approach require direct control of the power distribution network and clock frequency management. This might degrade performance.

Many techniques consider manipulation of instruction sequences in the instruction cache to reduce toggle activities. One popular approach is to compress the instructions with minimal hardware overhead on the existing Instruction Set Architecture (ISA). Benini et al. [6] proposed a design of a compressor/de-compressor that is inserted between the memory and the CPU to pack most frequently used instructions into a shorter format. This approach saves power by issuing fewer fetch commands to the instruction memory. A similar method that compresses shorter instructions into 32-bit words is presented in [7]. The authors used a Finite-State-Machine (FSM) that relies on the Markov model to make transitions to approximate the compressed code. Each state in the Markov model is assigned to a specific bit in the instruction word and probability of the transition can be computed. Since compressing instructions can cause increased toggles on the instruction bus, the authors proposed a mechanism to invert bits to reduce bit-toggling between consecutive instruction fetches. Stan and Burleson [8] proposed a similar approach that deals the bit-transitions on the address bus with a bit-inversion mechanism. In this

method, the number of bit transitions with respect to the previous data on the bus is computed in advance, and the decision is made whether to invert the bits. If the number of transitions is greater than half the bus width, data is inverted. An additional bit is attached to indicate an inversion condition. Mamidipaka et al. [9] realized that logic transitions in large buses are a significant source of power dissipation in embedded systems. They proposed an address encoding scheme that exploits temporal and spatial locality in the applications. The encoding scheme employs self-organizing lists to maintain frequently used data. Considerable power savings were observed for both data and address buses.

Other approaches in the high level design phase consider modifying program behaviors to better utilize instructions and memory accesses. One of the methods is presented in [10] where a heuristic model is used to determine the best design for the data path of the algorithm. This technique reduces power by three main methods: (*i*) reusing operands of the arithmetic modules by rearranging the execution codes in the algorithm, (*ii*) unrolling codes in loops and (*iii*) maximizing parallel execution with common operands. Lee et al. [11] proposed a method to keep loops compact so that instructions can be maintained in a small loop cache. This technique utilizes a special class of branch instructions, called backward branch instructions, to support the small cache design.

2.2. Architecture-Level Power-Aware Design Techniques

Techniques to achieve low power consumption in the architectural design level refer to methods that incorporate novel mechanisms to the existing architectures to reduce switching activities in the system. Similar to approaches that minimize bit toggles on the instruction cache and bus, two approaches that deal with switching activities on data buses are presented in [12-13]. In [12], the authors proposed a methodology to disable control signals to registers, multiplexers and tri-states to stop data from large data buses. This technique is based on the Boolean equations of all control signals in the data-path. The complexity of the control units increases as the logic depth of the data-path increases. This is a good approach given that a careful analysis is performed to balance

the logic depth and the complexity of the control unit. In [13], the authors proposed a memory-interface architecture that reduces the read-write request commands for video processing applications. The method exploits the regular memory-accessing patterns to develop an array address-translation algorithm for microprocessor-based processing. The first step is to determine the size of the loops in the programs so that data can be arranged in the memory for more efficient accessing. The pixel values are assigned to locations in different banks of the memory based on a set of conditions.

Other researchers studied the characteristics of input data and proposed computational units that can exploit those characteristics to reduce switching activities. Gandhi and Mahapatra [14] proposed integer functional units that operate on operand sub-words. This approach exploits the frequently occurring operand sub-words to reduce the switching activities in the microprocessor. The functional units perform three basic integer operations: addition, multiplication and multiplexing. The controlling unit detects sub-words for a special value and generates signals to bypass data within the functional units when necessary. Benini et al. [15] presented a methodology to replace combinational logic block with equivalent cells which can be "frozen" by asserting a controlling signal. This "frozen" unit can eliminate glitches in the combinational blocks and the "frozen" cell is implemented at the layout level.

A considerable amount of research work has been performed on design methodologies for low power components of DSP processors [16-22]. DSP applications are computationally intensive and require massive parallel processing capabilities to support real-time applications. The issue is that more parallel processing capability means more power dissipation. In [16], the authors presented an implementation of an FFT processor core that operates on order-based coefficients. The processing scheme is based on the minimization of the Hamming distance between successive coefficients that are fed to the core. The proposed implementation achieves good reduction rate when a smaller number of coefficients are used. The reduction rate decreases significantly when a large number of coefficients (512 points) are considered. Wu et al. [17] presented a data switching scheme based on the dynamic range of input data to achieve a low power design for a

Multiply-and-Accumulator (MAC). The switching activities are reduced for both Booth multiplier and adder by selecting an operand with a smaller dynamic range for the Booth encoding algorithm. The proposed design achieves a reduction rate of 17% to 21% of the power dissipation.

In [18-20], the authors presented an analysis of the Finite Response Filter (FIR) design that utilizes the differential coefficients method in the computational procedure. This approach requires more on-chip storage, but it reduces computations. A larger data set might have a negative impact on the performance; therefore, a tradeoff study needs to be considered before implementation. Yu et al. [21] proposed another method to analyze input data patterns for a possible reduction of dynamic power dissipation in FIR filters. The main contribution of this work is the reduced representation of the 2's complement number to avoid sign extension bits. This method has more impact when more data changes sign between consecutive operations. Many considered the design of arithmetic units within FIR filters to reduce power dissipation [22-24]. In [22], the authors proposed a VLSI architecture design of a configurable adder/subtractor and a configurable shifter be used in the FIR filter instead of multipliers. Alternatively, a segmented scheme for coefficient decomposition for an FIR core is presented in [23]. The coefficients are segmented with fixed bit widths to be fed to arithmetic units. This technique replaces a large multiplication with smaller multiplication and shifter operations. In addition, more data are packed into a block for simultaneous processing to reduce switching activities. In [24], the authors perform an architecture analysis to evaluate different standard types of multipliers to be used in the MAC units for DSP applications. The goal is to study the trade-offs between power consumption and performance. The authors conclude that the Booth multiplier is the best candidate for high speed applications with decent power consumption.

2.3. Logic/Arithmetic-Level Power-Aware Design Techniques

Most of the research work to reduce switching activities is done in the logic and arithmetic modules. Many researchers have proposed circuit designs to reduce switching

activities in computational modules such as multipliers, adder/subtractor units, etc. In [25], the authors proposed a method to reduce switching activities in combinational logic design using k-maps to modify logic equations. This technique introduces a large overhead to spread the terms in the sum-of-products expression into smaller and simpler expressions. In [26], several different types of adders are analyzed and compared for their performances and power consumption. In this work, the authors indicated that the adders proposed in [27-28] have good performances in terms of delay and power consumption. Design of low-power carry skip adder is presented in [29]. In this method, the authors divide the adder into variable-size blocks to achieve balance in the delay which helps to reduce glitches in the adders. In [30], a multiplexer-based full adder design is presented. The proposed design reduces the number of transistors needed to implement a full adder. In addition, there is no direct connection to a power-supply which helps to minimize short-current power consumption.

One of the common approaches in designing low-power multipliers is to partition the input operands into sub-words so that some of these sub-words can be ignored in the computational procedure [14, 31-32]. The research works presented in [31-32] focus exclusively on multiplication operations. These approaches also partition the input operands into smaller sub-words to possibly reduce the number of switching activities. In [31], the main idea presented is to utilize the identity $(-1)^2=1$ and the sign bits of the operands of two consecutive multiplications and conditionally exchange the sub-words. The exchange methodology is not trivial in this approach. In [32], the authors proposed a decomposition methodology of the operands to utilize shifting operations. Both of these methodologies reduce switching activities but increase circuit complexity and additional delay.

Inserting pipelining registers to reduce glitches in computational and arithmetic modules is also a common approach [33-35]. In [33], the authors study the effect of inserting pipelined registers in the FPGA-based multipliers, and it is proven that the pipelining technique is a very effective way to reduce glitches. Furthermore, the authors presented additional pipelining at the bit level for digit-serial multipliers. The effect of a digit-serial

multiplier is that it allows a faster clock rate, but throughput is reduced as each bit is considered serially. The reduction in glitches is modest. The approaches in [34-35] are similar to [33] where both of these methods consider inserting pipelining stages for combinational logic blocks. The effect of these methods is the reduction of glitches at the cost of increased area consumption. Analysis has to be performed to determine the trade-offs of area and power consumption. An extensive study of the impact of pipelining on the power consumption in FPGA is presented in [36]. The authors concluded that pipelining can reduce the amount of energy per operation in an algorithm in the range of 40% to 90%.

Approximation methods for logical and arithmetic units are also a very common approach for reducing switching activities. Approximation methods for multiplications through truncation of the input operands are presented in [37-39]. In these techniques, truncation of the insignificant portion of operands is considered to reduce the number of bits in the multiplication process. Because parts of the operands are ignored in the multiplication, errors exist in the results. The common idea is to ignore the least significant bits (LSBs) of the operands because they have low impact on the precision of the computational results. These methods can reduce as much as 60% of the switching activities. Another truncation approach for a multiplier is presented in [40]. In this approach, a right shift operation is performed on the operands in conjunction with truncation to reduce word-length. This technique requires a sign extension unit to fill in the most significant bits after the right shifting operation, which might present problems for large multipliers. Another approximation method is presented in [41] where the author proposed a subtraction of two's complement data via variable truncation of the most significant bits (MSBs). The idea is to detect a block of sign bits that can be truncated in the subtraction operation of two's complement data. This, in effect, will reduce the unnecessary switching activities in the subtractor unit which is mostly due to the two's complement transformation and carry bit propagation. The method is designed and implemented efficiently at the transistor level. Some more designs of low power adders at the transistor level are presented in [42]. These methods are effective but not

very useful for FPGA-based designs because FPGA-based designs are realized through the use of look-up-tables (LUTs) and flip-flops (FFs).

Transforming input data to a power-efficient format for multipliers is also a common approach. Fujino et al. [43] proposed a technique to detect a sign change from consecutive operation in a multiplier. When the condition is detected, data is transformed to a two's complement form. The condition in the proposed scheme is determined by the number of bit changes in consecutive data. This approach requires significant overhead for a relatively small multiplier. Another approach that manipulates input data before computation is presented in [44]. In this method, input data is complemented if the Hamming distance is greater than a predefined threshold. The combination of all input data and its decision is stored in a look up table. Therefore, this approach is not very suitable for the design of large multipliers.

The use of registers and latches in the design of multipliers is well studied. For example, Liu et al. [45] proposed a design of an asynchronous multiplier for low power consumption [45]. In this approach, the authors presented a split register technique that partitions the register as master and slave registers to reduce load capacitance for control wires. Furthermore, asynchronous control signals are utilized in the radix-2 multiplier for a reduction in the number of glitches. The use of pipeline registers for low power design of multipliers is presented in [46-47]. In [46], the authors presented the design of a multiplier with programmable pipeline stages that can be configured for 2, 4, or 8 cycles. The longer latency is used for non-critical operations in the array multiplier. A similar method is proposed in [47] where a 2-D pipeline gating technique is used to stop data from propagating through the pipeline stage when zero bits are detected.

Many researchers have proposed low power techniques for high performance multipliers. One of the most commonly implemented types of multiplier is the carry-save-adder (CSA) multiplier. The CSA multiplier is sometimes preferred over other types because of its regular structure which makes it easier to realize in the prototype implementation. Many have proposed techniques to reduce the power dissipation of the CSA multiplier

[48-51]. In [48], the authors presented a column bypass method in the adder array to disable blocks of logic when a '0' bit is detected in the operand. A similar approach is proposed in [49] where each functional unit is integrated with a multiplexer to provide a bypass path through the column of the array. Both approaches provide improvement with insignificant overhead to the regular structure of the CSA array. Other researchers proposed ways to arrange the fast carry save adders to achieve low switching activities in the multipliers [50-51]. The authors of [50] have studied different arrangements of the adders in the hybrid multiplier using both carry-save adders and ripple carry adders. In [51], the researchers proposed a method to integrate the final carry-save adder into an array structure. All these design methodologies are applied to radix-2 operations.

One of the most widely adopted implementations of the high performance multiplier is the implementation of the well-known modified Booth algorithm (or radix-4 Booth algorithm). This algorithm is commonly used because it reduces the number of partial products; therefore, it can perform the accumulation faster with less switching activities. Some researchers have proposed simple ways to improve the modified Booth multiplier including Yu et al [52] and Khoo et al [53]. In [52], a simple modification to the Booth encoding logic is proposed to increase the probability of a zero in the partial products. Yu et al. [53] proposed to arrange the carry-save adder array in the most significant bits first to utilize the sign extension zero-encoding to reduce the switching activities in the array. Input data manipulation is also very common in the low power design techniques presented in the literature [54-55]. In these methods, operands are dynamically interchanged to increase the probability of the zero-encoding property in the modified Booth algorithm. In [54], the authors proposed methods to evaluate sign changes in consecutive operations to determine if operands should be interchanged. Chen et al. [55] proposed a method to compare the two operands for each operation by determining the effective dynamic range of each operand and deciding to switch operands based on the comparison result. By using the multiplier operand with smaller dynamic range for Booth encoding procedure, the partial products have a greater chance to be zero; hence, it reduces the switching activities. The authors presented an implementation of a parallel architecture for generating the partial products. The research presented in [56-57] has a

similar approach to [55]; however, the input data is partitioned into smaller parts to increase the chances of data interchanging.

2.4. FPGA-Based Power-Aware Design Techniques

The new generation of FPGA devices with more dedicated DSP resources, such as multipliers and embedded configurable processors is attracting more interest in the image and video processing market. The increasing capacity and capability of FPGA devices provide an attractive option for performance-hungry applications in the image and video processing domain. FPGAs are often used as an implementation platform for image processing algorithms for real-time applications because of their programmable structure which can be used to exploit data I/O, spatial and temporal parallelism. Some of the most current research works that employ FPGA for general image processing algorithms are presented in [58-60]. Draper et al. [58] studied different architecture approaches with various constraints for image processing algorithms on FPGAs. The authors analyzed the available compilation technology that can map point, window and globally process in a high level language to dedicated architecture design for FPGAs. Similarly, research work presented in [59-60] deal with a mapping technology that can translate an algorithm level design to a parallel architecture design of real-time applications in the image processing domain. Results shown in these research works show many difficulties in task realization of high level image processing algorithms in specific real-time parallel architecture. Furthermore, general compilation techniques might not fully exploit the inherent parallelism in specific algorithms.

With increasing capacity, all computational modules can be fitted in an FPGA device that forms a complete processing system on chip. The integration of more components on a chip also leads to the issue of heat and power dissipation in FPGA-based designs. In [61-62], the authors presented different algorithmic-level optimizing techniques to achieve energy-efficient designs for signal processing applications. These techniques include the appropriate selection of types of architecture to support parallel processing, methods to disable computational units when necessary and algorithmic implementation

methodologies. Two typical operations, which are FFT and matrix multiplication, in signal processing applications are used to illustrate the design techniques and to evaluate the performance. Estimations and low level simulations are used to compute energy consumption for evaluation. Results reported in [61-62] show significant reduction in energy consumption when these algorithmic-level optimizing techniques are employed. Another optimization technique for FPGA-based design is presented in [63]. In this technique, the author proposed a reprogramming technique to re-map FPGA's look-up-tables (LUTs) to balance delay and to reduce glitches. The method analyzes the local LUTs within a neighborhood and generates Boolean functions for reprogramming the LUTs without changing the layout of the design.

A comparative study of the impact on logic depth on power consumption in FPGA and standard CMOS cells is presented in [64]. Pipelined registers are inserted in the design of multiplication using Xilinx, Altera FPGA and standard CMOS cells. While the use of pipeline stages helps to reduce power consumption in all devices, the relative higher off-chip power consumption for CMOS devices makes the improvement in FPGA more dominant. The study of the dynamic power consumption and some basic optimizing techniques in a Xilinx Virtex FPGA family is presented in [65-66]. Similarly, techniques to reduce dynamic power consumption and low power design options in a new generation of Altera FPGA are presented in [67-68]. For FPGA devices, all basic optimizing techniques for CMOS design such as clock gating, pipelining, retiming, voltage and frequency scaling are applicable.

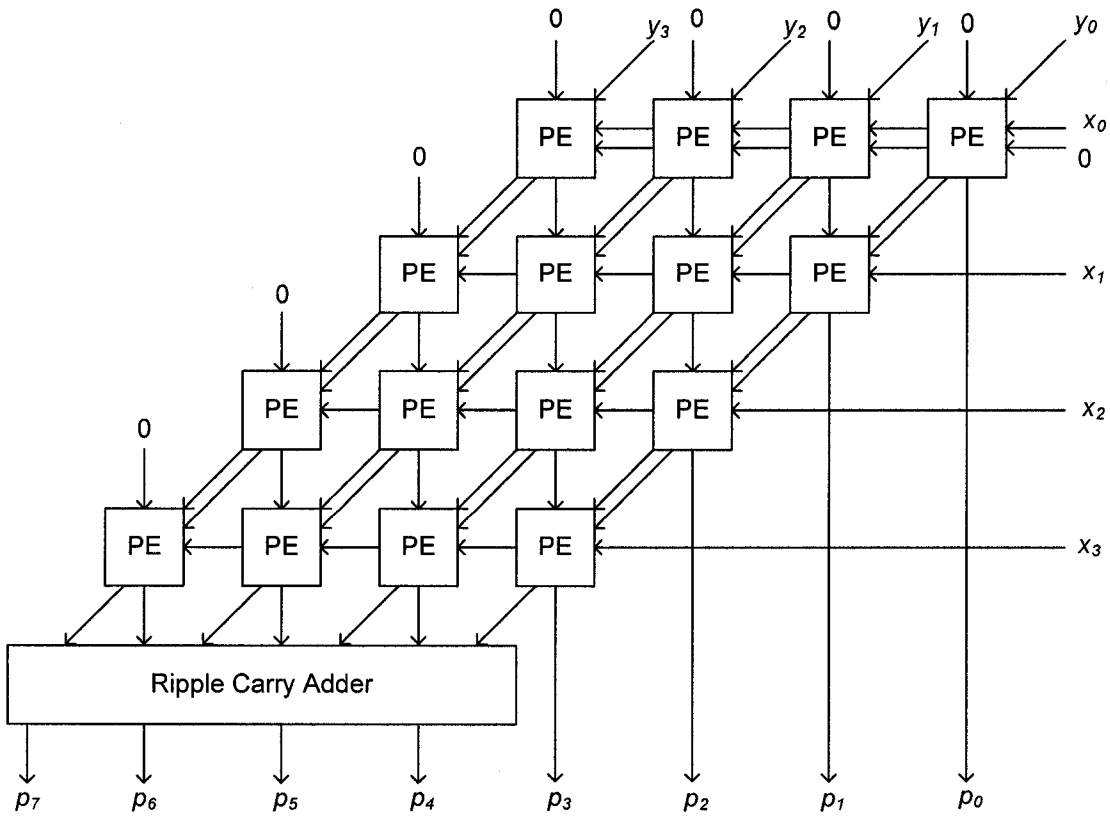
Chapter III

POWER-AWARE DESIGN TECHNIQUES FOR MULTIPLIERS IN VIDEO PROCESSING APPLICATIONS

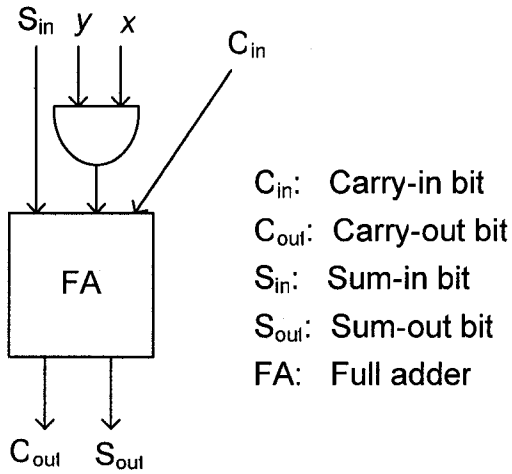
Previous research works for low power design of arithmetic units were mainly carried out in the circuit and logic level [25-33, 37-57]. Such research does not consider the data characteristics in the operands in the design. Since dynamic power dissipation is directly related to the switching activities of the circuits, it is desirable to consider the characteristics of the data in the design to reduce power consumption more effectively. Multiplication is one of the fundamental and most widely used operations in image and video processing applications. A partitioning and data gating technique for the design of pipelined multipliers is proposed to reduce switching activities exploiting characteristics of the input data in video processing applications. FPGA-based implementation and simulation results of the proposed multiplier design technique are presented in this chapter.

3.1. Array Multiplier

An array multiplier refers to an efficient layout of interconnected topology for combinational processing modules to perform vector multiplication. One of the most well-known array multipliers is the carry-save-adder (CSA) array multiplier. Carry save adder is one of the widely used adder designs for fast arithmetic, which is desirable for video processing applications. The layout of the CSA array multiplier makes it a preferred choice for implementation in FPGA due to its regular structure. The drawback of this design is that the carry bits are propagated through all stages of the array multiplier. Hence, the propagated carry bits generate more switching activities and thereby more power is dissipated. Architecture of a 4×4 CSA array multiplier is shown in Figure 3.1 where each PE consists a full adder (FA) and an AND gate. Other array multipliers include ripple carry, Braun, and Baugh-Wooley array multipliers [69]. Pipeline stages are inserted in the array multipliers to reduce switching activities due to long propagation delays in the array layout.



(a) 4×4 CSA multiplier architecture.



(b) Processing element (PE) in CSA multiplier.

Figure 3.1. CSA array multiplier.

Some of the previous research to reduce power consumption in array multipliers includes data bypassing and signal gating techniques such as those presented in [48-49, 70]. In data bypassing techniques, the common approach is to provide a bypassing route to the next level in the array through a multiplexer or a latch when a '0' bit is detected in the operand. An example of such a bypassing concept is illustrated in Figure 3.2. This approach in general requires more overhead since a bypassing mechanism has to be included in each processing module.

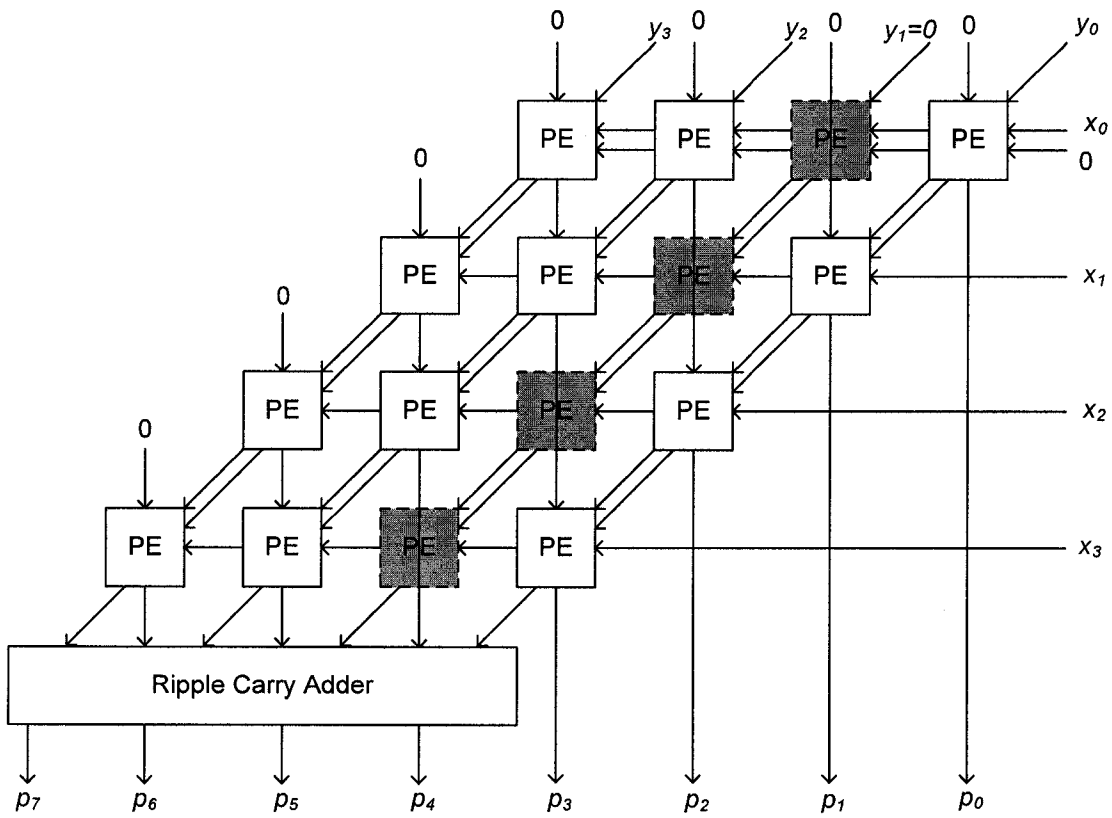


Figure 3.2. Bypassing concept for a 4×4 CSA multiplier.

For the signal/data gating approach, when a gating signal is active, the output port of the logic block maintains the current data; hence, no transitions occur. The gating technique is generally used as a mechanism to stop unnecessary switching in the sign extension part of the two's complement data [70]. The gating technique can also be applied to pipeline registers to stop unwanted data from propagating through stages. One such approach is

proposed in [47]. A block diagram for illustrating the concept of gating pipeline stages 4.3 and that for the 2-D gating of individual registers is shown in Figure 3.4.

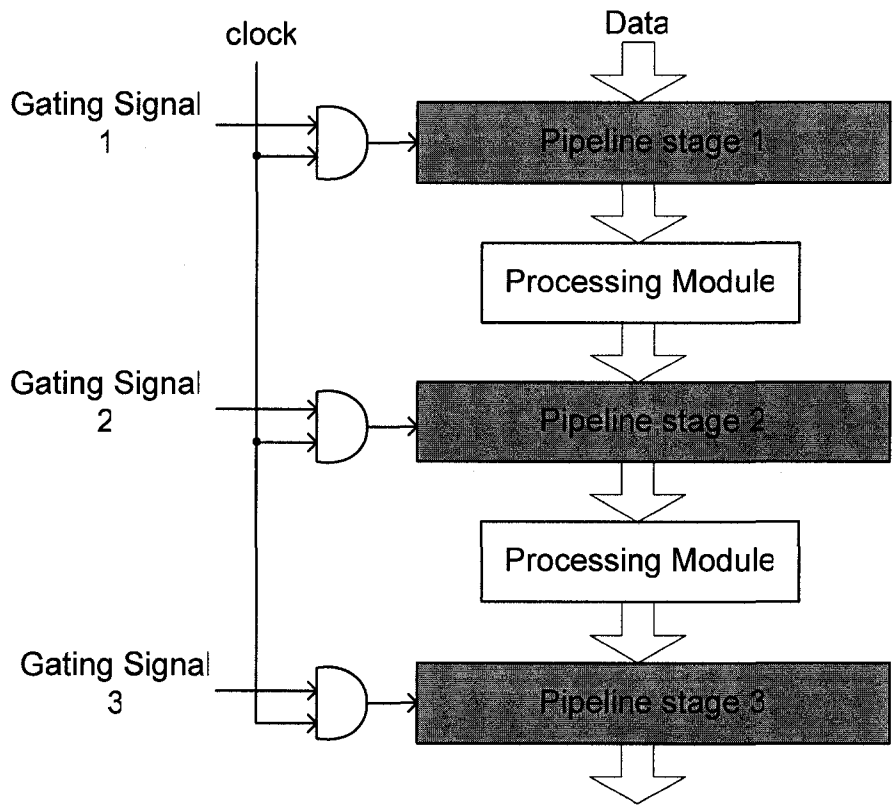


Figure 3.3. Block diagram of 1-D gating technique for a pipelined multiplier.

Inserting pipelining registers in parallel array multipliers is a widely used technique to reduce glitches due to imbalance in layout of the interconnected topology. The proposed multiplier design uses this approach as a baseline design for gating technique based on characteristics of the input data patterns. Pipelining is particularly beneficial for FPGA-based design since logic elements of FPGA devices have embedded flip-flops which can be used without high overhead penalty. Furthermore, inserting pipeline stages in the large array multiplier circuits reduces propagation delays which results in increased operating clock frequency.

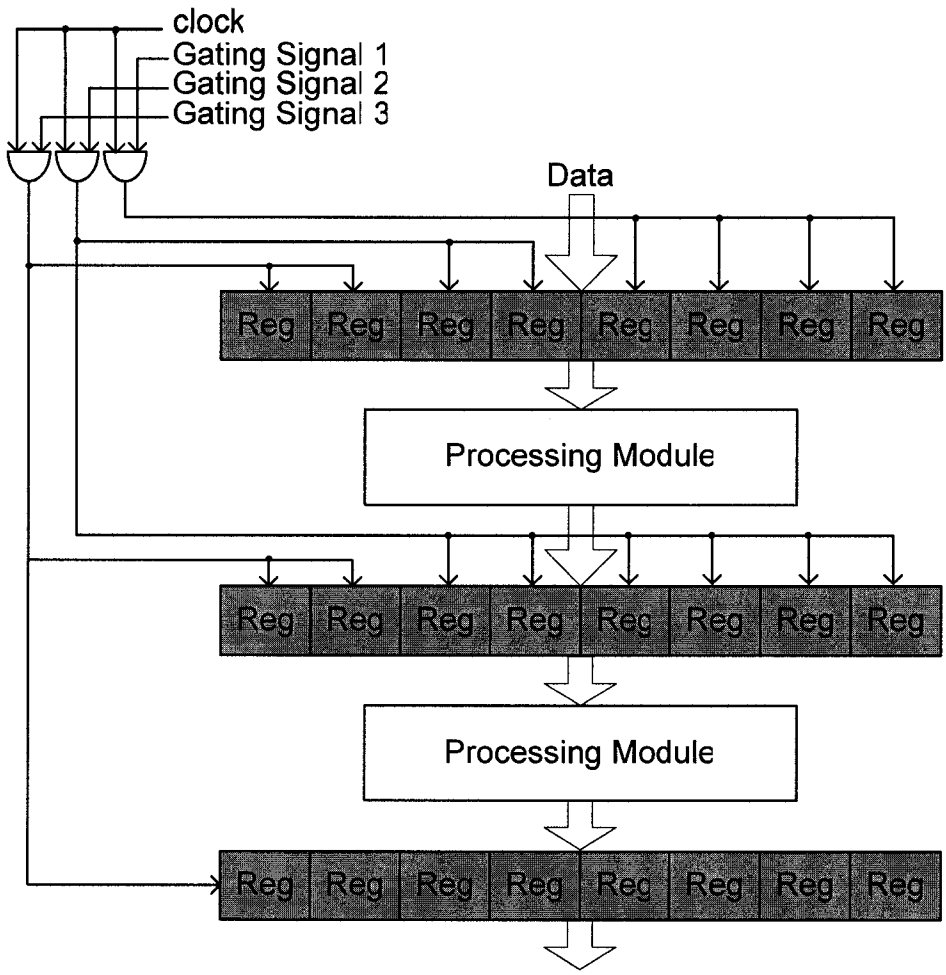


Figure 3.4. Block diagram of 2-D gating technique for a pipelined multiplier.

3.2. Recoding for Parallel Multiplier

Recoding a multiplication operand is a popular approach for high performance because that reduces the number of partial products (PPs). The most common recoding technique is the Booth algorithm, which does not generate PP for a group of consecutive 0's or 1's. The basic radix-2 Booth multiplication algorithm evaluates two bits at a time and generates a PP that is one digit of the set {-1, 0, 1} multiplying with the multiplicand. For example, let P be the result of the multiplication of a multiplier X and the multiplicand Y where X and Y are represented in two n-bit two's complement numbers as:

$$X = -x_{n-1}2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i \tag{3.1}$$

$$Y = -y_{n-1}2^{n-1} + \sum_{i=0}^{n-2} y_i 2^i \tag{3.2}$$

The recoding scheme for radix-2 Booth algorithm is shown in Table 3.1.

Table 3.1. Recoding scheme for radix-2 Booth algorithm.

x_i	x_{i-1}	**PP**
0	0	0
0	1	Y
1	0	$-Y$
1	1	0

where $i = 0,1,2,.....,n$.

The radix-2 Booth multiplication algorithm works quite well if multiplier X has a group of consecutive 0's or 1's. For instance, if X="00111110", then the Booth algorithm will only generate two PPs instead of five as in conventional multiplication. However, if the multiplier X has isolated 1's, the algorithm becomes inefficient. For example, if X="01010101", then eight PPs are generated instead of four as in the conventional approach. The problem can be addressed with radix-4 Booth multiplication where three bits are used for the recoding scheme. The basic radix-4 Booth multiplication algorithm evaluates three bits at a time and generates a PP that is one digit of the set {-2, -1, 0, 1, 2} multiplying with the multiplicand. Radix-4 recoding scheme can reduce up to half of the PPs compared to the radix-2 method. The recoding scheme for radix-4 Booth multiplication algorithm is shown in Table 3.2. For discussion, an n-bit multiplier is considered, and n is an even number.

Table 3.2. Recoding scheme for radix-4 Booth algorithm.

x_{2i+1}	x_{2i}	x_{2i-1}	**PP**
0	0	0	0
0	0	1	Y
0	1	0	Y
0	1	1	$2Y$
1	0	0	$-2Y$
1	0	1	$-Y$
1	1	0	$-Y$
1	1	1	0

where $i = 0,1,2,.....,\frac{n}{2}-1$.

Many techniques have been proposed to modify the Booth multiplier design to achieve low-power dissipation [43, 52, 56, 57]. These techniques include new recoding schemes, bit inversion, dynamic range detection and operand switching and efficient layout of parallel adders. One of the effective methods is the one that computes the dynamic ranges of the operands and uses the operand with smaller dynamic range for the multiplier. This effectively increases the chance for more PPs to be zero. When a PP is zero, an appropriate controlling sequence is activated to reduce switching activities in the multiplier circuit. The block diagram for this approach with a dynamic range detection (DRD) unit presented in [57] is shown in Figure 3.5. In this approach, two register stages (master and slave) are used before the Booth encoding step to support data switching. A sign extension unit is used to align the sum of all the partial products to match the number precision used in the systems.

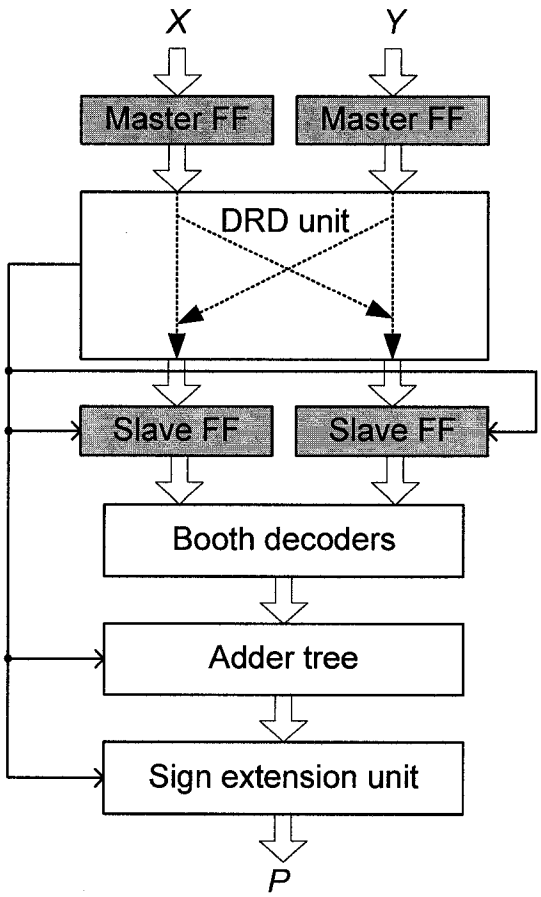


Figure 3.5. Multiplier with DRD unit proposed in [57].

3.3. Partitioning and Gating Technique for Multiplier Design for Video Processing Applications

While existing design techniques usually focus on the bit-level of the multiplier structure or interconnect topology, most of these techniques do not take into account the data characteristics of the application data. Since power consumption is directly related to the switching activities of the processing systems, it is intuitively logical to consider data characteristics in the multiplier design for reduced switching activities. While data in image processing applications may have large dynamic ranges, one operand of the multiplication frequently has small magnitudes in data representations. This characteristic occurs fairly commonly in window-based operations such as digital filtering operations.

These are some of the most frequently used operations in image and video processing algorithms. The data characteristics of these operations are that magnitudes of the filter coefficients are usually small and dynamic ranges of the pixel values within a neighborhood block are small. To illustrate the exploitation of this characteristic, a multiplier design that partitions the operands into smaller parts by which the computations can be performed with multiple smaller multiplier units is proposed. With this approach, one or more smaller multipliers, adders and supported modules can be deactivated when the special conditions such as a zero or a one in input data is detected.

Let's consider the multiplication of 2 n-bit numbers X and Y, the partitioning process of the operands in the multiplication $P=X\times Y$ into m-bit higher and $(n-m)$-bit lower parts is described as:

$$P = X \times Y \tag{3.3}$$

$$P = \left(X_H \times 2^m + X_L\right)\left(Y_H \times 2^m + Y_L\right) \tag{3.4}$$

$$P = \left(X_H \times Y_H \times 2^{2m}\right) + \left(X_H \times Y_L + X_L \times Y_H\right)2^m + X_L \times Y_L \tag{3.5}$$

where,

$$X = \left(x_{n-1}, x_{n-2}, x_{n-3}, \ldots, x_0\right)$$

$$Y = \left(y_{n-1}, y_{n-2}, y_{n-3}, \ldots, y_0\right)$$

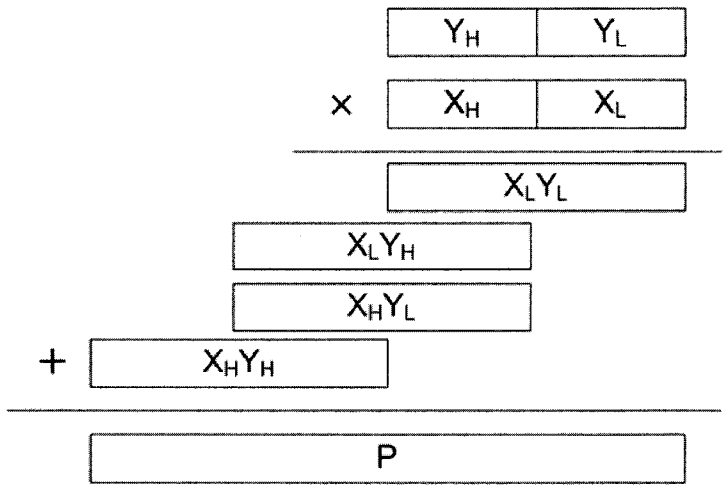
$$X_H = \left(x_{n-1}, x_{n-2}, x_{n-3}, \ldots, x_m\right)$$

$$X_L = \left(x_{m-1}, x_{m-2}, x_{m-3}, \ldots, x_0\right)$$

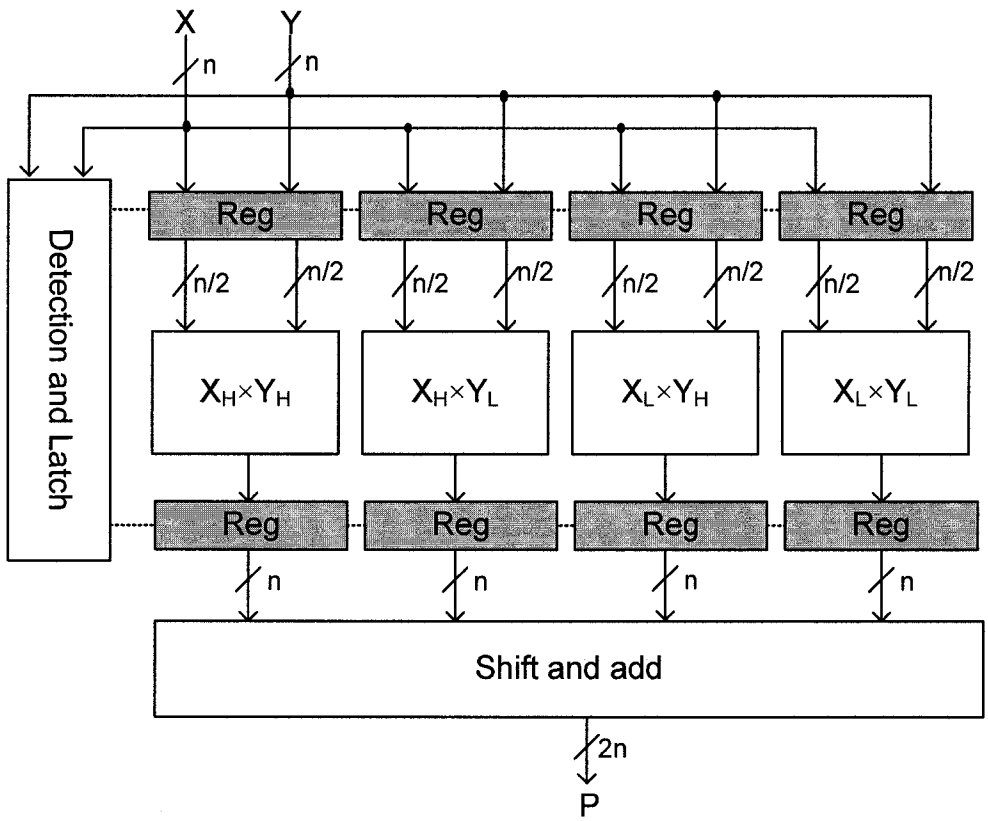
$$Y_H = \left(y_{n-1}, y_{n-2}, y_{n-3}, \ldots, y_m\right)$$

$$Y_L = \left(y_{m-1}, y_{m-2}, y_{m-3}, \ldots, y_0\right)$$

The main approach is to detect a zero in each of these partitioned data parts, namely Y_H, Y_L, X_H and X_L, and to disable the appropriate multiplier units. In [71], the authors proposed an architecture design for partitioning data into halves and employing a clock gating technique to disable input and output registers that are connected to the multipliers. This method is illustrated in Figure 3.6.



(a) Multiplication with partitioned data.



(b) Gating technique for individual smaller multipliers.

Figure 3.6. Partitioning method for multiplier design.

In the proposed design technique, in addition to partitioning the data into smaller parts to increase the chance of obtaining smaller operands to be zero, the data characteristics of the filter coefficient operands with smaller magnitudes is also utilized in the design. In addition, the gating technique is also applied to the subsequent adder stages in the pipelined multiplier design. While the partitioning method can be employed with different word-lengths, operands are partitioned into halves for discussion here. That is $m=n/2$. One of the design goals is to provide a standard I/O interface such that the multiplier can be connected to any system without pre-conditions. So, the full-width input/output data interfaces are used. The partitioning mechanism is considered as part of the multiplier design. Considering multiplications in image filtering operations, let's consider *X's* to be the pixel values and *Y*'s to be the filter coefficients. Initially, zero conditions are detected in both X_H and X_L of the pixel values. The gating signals z_{XH} and z_{XL} are set to logic '1' if X_H and X_L are zero respectively. The signals z_{XH} and z_{XL} are generated based on:

$$z_{XH} = x'_{n-1} \cdot x'_{n-2} \cdot x'_{n-3} \cdot \ldots \cdot x'_{m} \tag{3.6}$$

$$z_{XL} = x'_{m-1} \cdot x'_{m-2} \cdot x'_{m-3} \cdot \ldots \cdot x'_{0} \tag{3.7}$$

Since most of the coefficients in different filter kernels have small magnitudes, detection of zero and one conditions in the higher half of the coefficient *Y* is performed. Conditions of one are not considered for *X* because the pixel values vary from different images captured under different environments. Two conditions in the Y_H operands are considered in the gating technique. These conditions set the gating control bits to logic '1' when an input is zero or one is detected. The gating signals z_{YH} and o_{YH} are generated as:

$$tmp = y'_{n-1} \cdot y'_{n-2} \cdot y'_{n-3} \cdot \ldots \cdot y'_{m+1} \tag{3.8}$$

$$z_{YH} = tmp \cdot y'_{m} \tag{3.9}$$

$$o_{YH} = tmp \cdot y_{m} \tag{3.10}$$

There are many ways to implement gating logic that prevents data from propagating through the depth of the data paths. The simplest way is to use the basic AND gate, and set the control signal to '0' when data is not needed. This approach is simple, but the switching to '0' still has the effect on the subsequent computational modules that might toggle unnecessarily due to this condition. Another way to implement gating logic is to insert a latch or a register to hold the data if a control signal to turn it off is given. This approach prevents unnecessary switching activities to the downstream modules. The disadvantage of this method is the overhead and additional latency. Other common approaches to implement the gating technique include the use of a tri-state buffer and a transmission gate instead of a latch or a register. For these two approaches, the main concern is the charge leakage and large current that may occur. Since FPGA devices provide embedded flip-flops within logic elements, the latch and register implementation is used in this research work.

The circuits for detection of z_{XH}, z_{XL}, z_{YH} and o_{YH} are shown in Figure. 3.7. Figure 3.8. shows the architecture design for a pipelined multiplier which utilizes the partitioning and gating technique based on the characteristics of the multiplier operands. The smaller multipliers can be implemented with conventional multipliers such as CSA array multiplier or Booth multiplier. The gating signals l_{XHYH}, l_{XHYL}, l_{XLYH} and l_{XLYL} shown in Figure 3.8. are determined based on the input data patterns and the expressions for generating these signals are:

$$l_{XHYH} = z'_{XH} \cdot z'_{YH} \cdot o'_{YH} \tag{3.11}$$

$$l_{XHYL} = z'_{XH} \tag{3.12}$$

$$l_{XLYH} = z'_{XL} \cdot z'_{YH} \cdot o'_{YH} \tag{3.13}$$

$$l_{XLYL} = z'_{XL} \tag{3.14}$$

Each of the smaller multiplier units in Figure 3.7 are implemented as CSA array multiplier or Booth multiplier in the experiments. Since previous research has shown that multiplications in radix-4 consume less power than multiplication in radix-2, a radix-4 Booth multiplier design is considered in this section. The proposed design is based on the

standard three signal recoding scheme and gating signals are generated based on the conditions shown in Table 3.3.

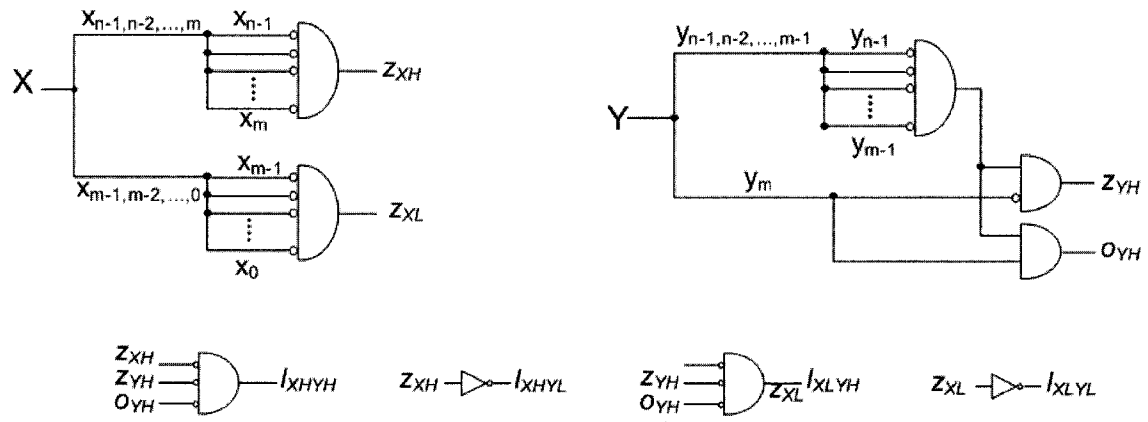


Figure 3.7. Circuits for generating gating signals.

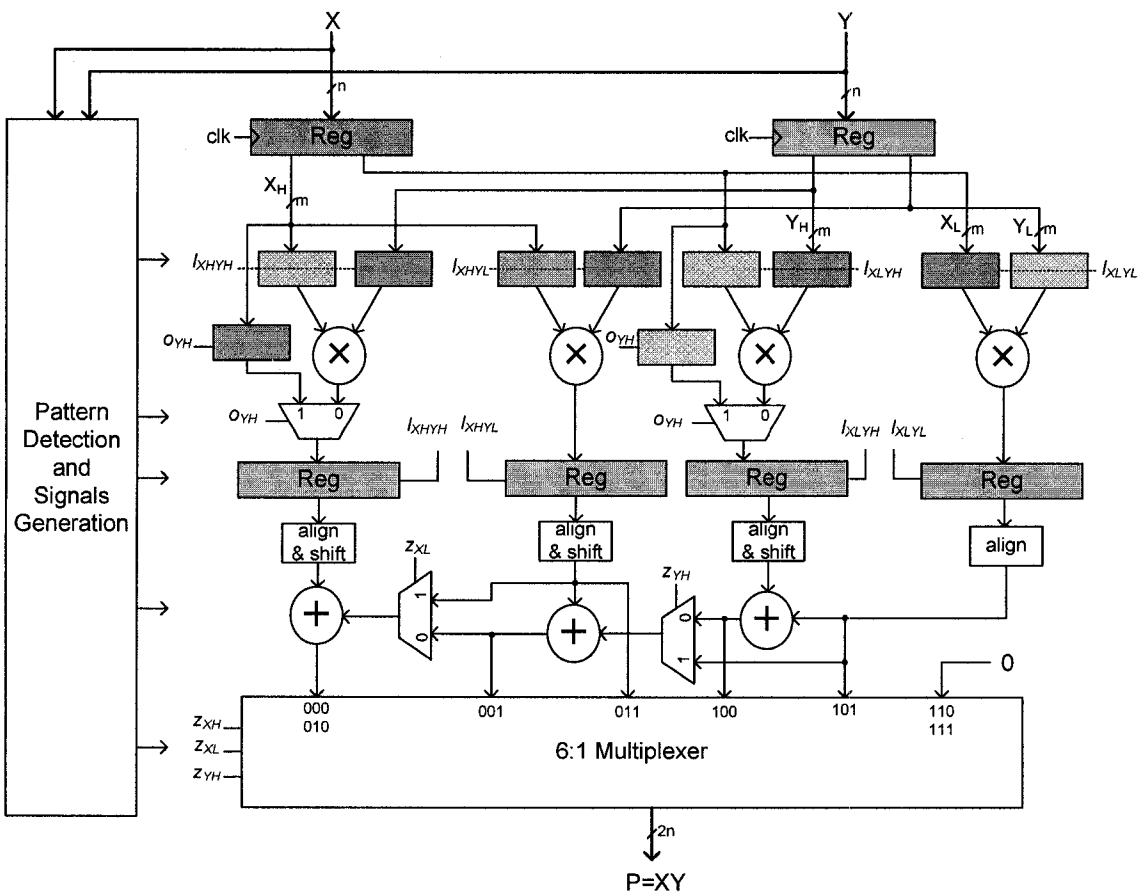


Figure 3.8. Proposed design of the pipelined multiplier.

Table 3.3. Recoding scheme for the proposed *n-bit* radix-4 Booth multiplier.

x_{2i+1}	x_{2i}	x_{2i-1}	PP	*zero*	*two*	*neg*
0	0	0	NC	1	0	0
0	0	1	*Y*	0	0	0
0	1	0	*Y*	0	0	0
0	1	1	*2Y*	0	1	0
1	0	0	*-2Y*	0	1	1
1	0	1	*-Y*	0	0	1
1	1	0	*-Y*	0	0	1
1	1	1	NC	1	0	0

NC: no change

The expressions for the three signals *zero*, *two* and *neg* are:

$$zero_i = \left(x'_{2i+1} \cdot x'_{2i} \cdot x'_{2i-1}\right) + \left(x_{2i+1} \cdot x_{2i} \cdot x_{2i-1}\right) \tag{3.15}$$

$$two_i = \left(x_{2i} \oplus x_{2i-1}\right)' \cdot zero' \tag{3.16}$$

$$neg_i = \left(x_{2i} \cdot x_{2i-1}\right)' \cdot x_{2i+1} \tag{3.17}$$

The circuits to generate these signals are shown in Figure 3.9.

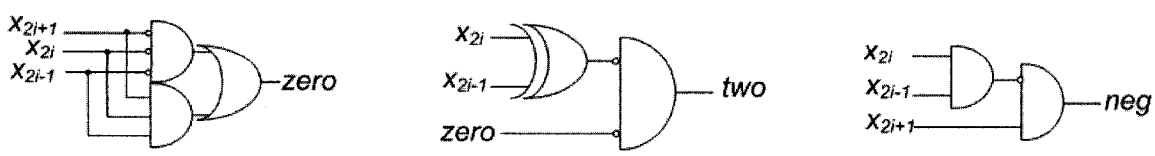


Figure 3.9. Circuits to generate recoding signals.

These signals are used in the gating technique that is incorporated in the proposed Booth multiplier design. The architecture of an 8-bit Booth multiplier design is shown in Figure 3.10. The shifting (<<) and sign extension (ext.) modules are provided for illustration only since these operations can be done by bus shifting. The complement unit (comp) is used to invert all bits in the input data when at least one of the '*neg*' signals is active. The expression for the complement operation of an n-bit input number Y is:

$$comp_i = neg \oplus y_i \qquad \text{for } i = n-1, n-2, n-3, \ldots, 0 \tag{3.18}$$

The circuit for the complement unit is shown in Figure 3.11. The correction unit is used to adjust the negative PPs. When a negative PP is produced, it is generated by the complement unit (comp), and the correcting term is added with the correction unit. The correction terms (***CT***) generated for different possible combinations of *neg* signals are listed in Table 3.4.

Table 3.4. Correction terms used in the proposed Booth multiplier.

neg_3	neg_2	*CT*	neg_1	neg_0	*CT*
0	0	0	0	0	0
0	1	16	0	1	1
1	0	64	1	0	4
1	1	80	1	1	5

While pixel values are always positive numbers, filter coefficients might be negative numbers in many applications. The use of two's complement (2C) numbers is most common in digital systems because addition of signed numbers is a straightforward operation. However, for multiplications, sign bits in the number representations may be dominant parts which would propagate through the multipliers; hence more switching activities are performed. There has been a lot of research that uses sign magnitude (SM) representation to reduce switching activities for multiplications [72, 73]. In general, numbers represented with SM have a lower switching rate than numbers represented in

2C especially for multimedia applications [12]. In the proposed design, the I/O interface is implemented with two's complement representation and multiplication is performed with sign magnitude representation. Thus, the approach is to include a 2C-to-SM converter at the input for the multiplicand Y only because pixel values X will always be positive numbers. Then, the result P is converted back to 2C by a SM-to-2C converter before it is outputted. This procedure is illustrated in Figure 3.12.

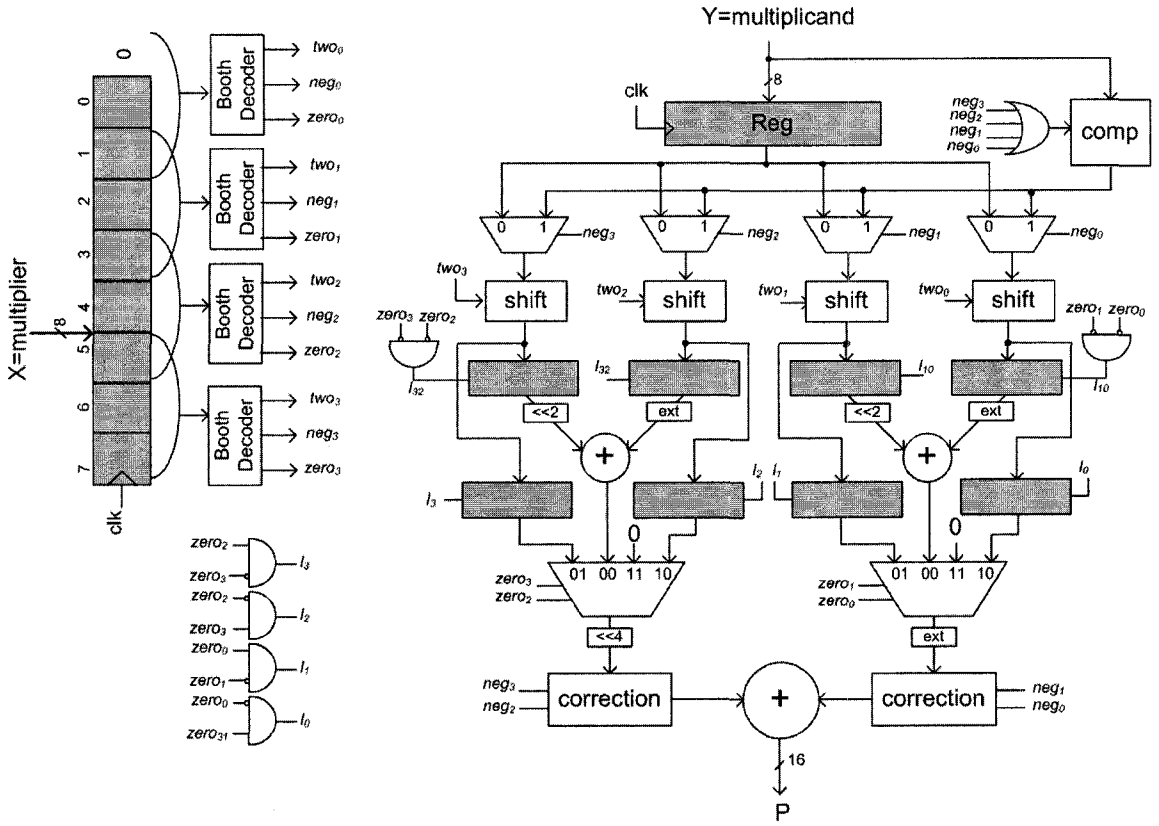


Figure 3.10. Proposed design of the 8-bit Booth multiplier.

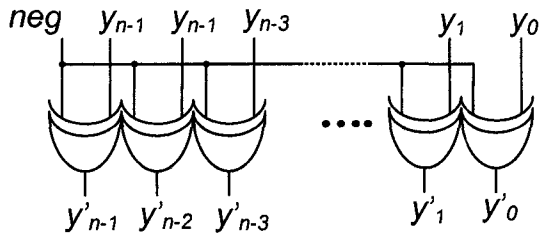


Figure 3.11. Circuit for the complement unit (comp).

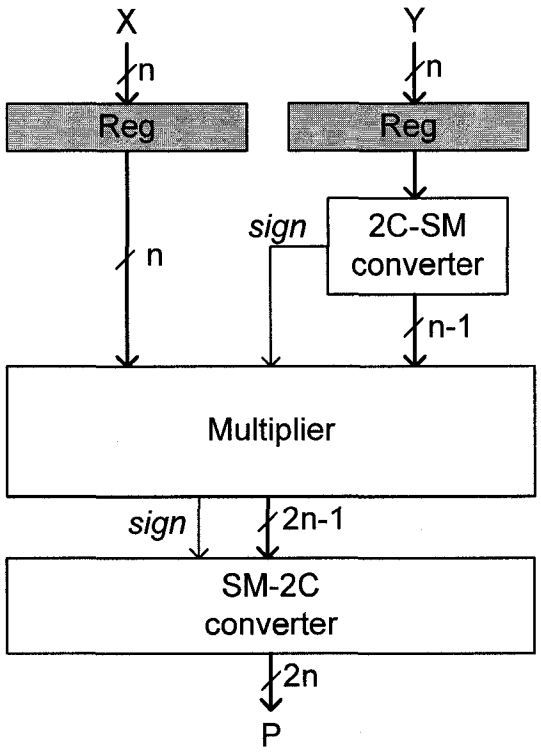


Figure 3.12. Schematic of the multiplication operation with SM representation.

3.4. Experimental Results

The proposed design for a partition and gating technique based on data characteristics is implemented and applied with two standard multiplier designs: (1) pipelined CSA array multiplier (CSA) and (2) radix-4 Booth multiplier (Booth). These two multiplier implementations are used as baseline schemes for evaluation purposes. The proposed CSA array and Booth multiplier designs with consideration of data characteristics are referred in the evaluation as CSA_PG and Booth_PG respectively. In addition, two's complement and sign magnitude representations for the proposed Booth multiplier design are also studied. Booth_PG2C refers to the proposed design which uses two's complement representation while Booth_PGSM refers to the proposed sign-magnitude Booth multiplier. These design schemes are implemented in VHDL using Altera's Quartus II design tool. The designs are fitted and evaluated in a Cyclone II (2C35) FPGA in the Altera's DE2 developmental and educational board. A summary of resource

utilization and maximum speed achieved from the implementation along with latency for each design is provided in Table 3.5.

Table 3.5. Implementation results with various multiplier designs with Altera's FPGA.

Scheme		Logic elements	F_{max} (MHz)	Latency
Array multiplier	CSA	174	257	3
	CSA_PG	193	202	4
Booth multiplier	Booth	243	184	4
	Booth_PG2C	291	153	4
	Booth_PGSM	308	140	5

Two other existing and related Booth multipliers are implemented and evaluated for comparison of power consumption with the proposed multiplier designs. The first related design [55] considers the dynamic range of the input data and switches the multiplier and multiplicand if conditions are met. This design is referred to as Booth_DRD in this evaluation. The second related research work [57] also considers the dynamic range of the data. In addition, the operands are partitioned into 2 sub-words to increase the chance of a zero partial product [56]. This approach is referred to as Booth_DRDPAR. A 12×12 multiplier is designed and implemented for each scheme. Multiplications for two image filters are considered in the evaluations. The two filters are: (1) a low pass filter with a Gaussian kernel and (2) a Laplacian filter for edge detection. Pixel values are extracted from the test images and fed to the multiplier while filter coefficients are connected to the multiplicand. Simulation tools from Altera Quartus II software are used, and logic transitions are recorded to the Signal Activity File (SAF). The SAF file is used in the

PowerPlay tool to compute the dynamic power consumption of the designs. Power consumption for pipelined CSA array multiplier (CSA) and the proposed pipelined CSA array multiplier with a partitioning and gating technique (CSA_PG) for a low pass filter and Laplacian filter is shown in Tables 3.6 and 3.7 respectively. The power consumption in these experiments is measured at 50MHz. The two test images used to extract pixel values for the multipliers are shown in Figure 3.13.

(a) Peppers (128×128 resolution).

(b) Island (160×106 resolution).

Figure 3.13. Test images.

As shown in Tables 3.6 and 3.7, the results for CSA_PG improve the power consumption of the CSA multiplier considerably. The proposed technique achieves more than 17% power reduction in the smoothing operation. For the edge detection operation, the proposed partitioning and gating technique for a CSA multiplier achieves 14% and 12% power reduction for the two test images. Results for CSA_PG in smoothing operations

(Gaussian kernel) are better compared to those in edge detection operations (Laplacian kernel). This is due to sign changes in the kernel of the second application. Sign changes in the coefficients propagate through the entire sign extension part in the binary representation of the data.

Table 3.6. Experimental results for a Gaussian filter for array multiplier.

Test Image / Method	**Peppers** (128×128)		**Island** (160×106)	
	Power (mW)	**Normalized ratio**	**Power** (mW)	**Normalized ratio**
CSA	40.73	1.00	38.56	1.00
CSA_PG	**32.99**	**0.81**	**32.01**	**0.83**

.Table 3.7. Experimental results for a Laplacian filter for array multiplier.

Test Image / Method	**Peppers** (128×128)		**Island** (160×106)	
	Power (mW)	**Normalized ratio**	**Power** (mW)	**Normalized ratio**
CSA	33.47	1.00	35.57	1.00
CSA_PG	**28.78**	**0.86**	**31.27**	**0.88**

The power consumptions for pipelined Booth multiplier (Booth), Booth multiplier with dynamic range determination (Booth_DRD), Booth multiplier with partition and dynamic

range determination (Booth_DRDPAR), the proposed two's complement Booth multiplier with partitioning and gating technique (Booth_PG2C) and the proposed sign magnitude Booth multiplier (Booth_PGSM) with partitioning and gating technique for low pass filter and Laplacian filter are shown in Tables 3.8 and 3.9 respectively. For the smoothing operation, both the proposed Booth_PG2C and Booth_PGSM multipliers achieve significant power reduction that ranges from 16% to 28%. For the edge detection operation, the proposed Booth_PG2C and Booth_PGSM multipliers achieve a power reduction in the range of 10 to 18% compared to the baseline design. For this operation, the design of sign magnitude representation achieves better results than the design with two's complement representation due to the sign changes in the kernel coefficients. In both operations, the proposed multiplier designs have favorable results compared to other existing methods as shown in Tables 3.8 and 3.9.

Table 3.8. Experimental results for a Gaussian filter using Booth multiplier.

Test Image / **Method**	**Peppers** (128×128)		**Island** (160×106)	
	Power (mW)	**Normalized ratio**	**Power** (mW)	**Normalized ratio**
Booth	31.82	1.00	28.42	1.00
BoothDRD	25.71	0.81	24.74	0.87
Booth_DRDPAR	29.76	0.94	29.01	1.02
Booth_PG2C	**23.19**	**0.72**	**21.35**	**0.75**
Booth_PGSM	**24.51**	**0.77**	**23.82**	**0.84**

Table 3.9. Experimental results for a Laplacian filter using Booth multiplier.

Test Image / Method	**Peppers** (128×128)		**Island** (160×106)	
	Power (mW)	**Normalized ratio**	**Power** (mW)	**Normalized ratio**
Booth	27.31	1.00	30.32	1.00
BoothDRD	30.67	1.12	32.72	1.09
Booth_DRDPAR	31.35	1.15	34.00	1.12
Booth_PG2C	**24.64**	**0.90**	**25.80**	**0.85**
Booth_PGSM	**23.09**	**0.85**	**24.85**	**0.82**

3.5. Summary

In this chapter, we have developed, implemented and evaluated a power-aware design technique that integrates the partitioning and gating methods to reduce switching activities in digital circuits. Image and video processing applications such as digital filtering have special characteristics such that the magnitudes of the coefficients in the filters are usually small. The proposed power-aware design technique employs the partitioning of the multiplier operands so that smaller multipliers are used. When the special condition such as the higher order part of one of the operands is zero or one, the corresponding multiplier and its data path are disabled by the gating technique. The proposed partitioning and gating technique is applied to the pipelined CSA array multiplier. The evaluation results show an average 18 % and 12% power reduction for a

smoothing operation and edge detection operation respectively when compared to the baseline CSA array multiplier.

A design of radix-4 Booth multiplier that incorporates the partitioning and gating technique in the recoding scheme has also been presented in this chapter. Design of a sign magnitude Booth multiplier with the proposed partitioning and gating technique has been presented. Simulation results show that both Booth_PG2C and Booth_PGSM achieve good power reduction compared to the baseline designs and other existing techniques. The Booth_PG2C multiplier design achieves better power reduction in smoothing operation while the Booth_PGSM has better results in the edge detection operation. The Booth_PG2C multiplier has an average of 44% power reduction compared to the baseline CSA multiplier and 26% power reduction compared to the baseline Booth multiplier for smoothing operation. The Booth_PGSM has an average 30% power reduction compared to the baseline CSA multiplier and 18% power reduction compared to the baseline Booth multiplier for edge detection operation.

Chapter IV

DESIGN OF LOGARITHMIC DOMAIN ARITHMETIC UNITS FOR LOW POWER CONSUMPTION

In recent years, there has been increased interest in employing logarithmic number systems (LNS) for multimedia applications in handheld devices [74-75]. Multimedia and signal processing applications frequently use complex arithmetic operations such as multiplication, division, powering, etc. LNS offer an alternative to calculate equivalent computations with much simpler arithmetic operations. However, computations in the logarithmic domain suffer some loss of accuracy. For signal processing applications that can tolerate a small amount of error, logarithmic arithmetic can simplify many complex computations. Table 4.1 shows some of the equivalent computations with normal (linear) and logarithmic arithmetic operations.

Table 4.1. Operations in normal (linear) binary and logarithmic domains.

Operations	**Linear domain**	**Logarithmic domain**
Multiplication	$x \times y$	$\log(x)+\log(y)$
Division	$x \div y$	$\log(x)-\log(y)$
Powering	x^y	$y \times \log(x)$

Logarithmic arithmetic modules also have advantages over the modules in linear and fixed-point systems in terms of area, delay and power dissipations. Paliouras et al. [74] have shown that computations in LNS reduce bit activities significantly compared to fixed-point systems. In [75], the authors show that LNS is an attractive alternative for arithmetic modules in handheld devices due to the low utilization of area and power. In

this chapter, an improved approximation method for computation of binary logarithm that can be used to compute complex operations such as multiplication and division with reduced area and power consumption is proposed. An efficient architecture design for the $\log_2$ and inverse- $\log_2$ approximation and a log-based multiplier design are presented. Partitioning and gating technique are also applied to the proposed log-based multiplier design for reduced power consumption.

4.1. Approximation of Binary Logarithm

The logarithm of different bases can be obtained from the logarithm of base two with an additional multiplication operation. The logarithm of base two can be calculated with existing techniques such as look up tables, successive iterations, series expansions and polynomial approximations. However, these algorithms are costly in terms of computational complexity and hardware resources. The algorithm presented in this section utilizes a binary numeric system to logically compute $\log_2$ and inverse-$\log_2$ (power of 2) values in a very efficient procedure. By providing an approximation technique for computing $\log_2$ and inverse $\log_2$, the complicated and computationally intensive tasks such as logarithm, division and powering operations can be closely approximated by simple operations such as additions, subtractions, multiplications and shifting.

The approximation method based on the idea of locating the index of the leading '1' bit in the binary number was first proposed by Mitchell [76]. The index is actually a weighted factor corresponding to a bit position in the polynomial form of a binary number; therefore, it is in decreasing order from left to right. The index of the leading '1' bit in the binary number is interpreted as the integer part of the logarithm result and the remaining bits after the leading '1' bit are considered as the fractional part of the result. Consider a fixed point number N that is in the interval $2^j \le N \le 2^{k+1}$ where $k \ge j$, N can be expressed as:

$$N = \left(n_k, n_{k-1}, ..., n_1, n_0, n_{-1}, ..., n_{j+1}, n_j\right) \tag{4.1}$$

$$N = \sum_{i=j}^{k} 2^i n_i \tag{4.2}$$

where n_i='0' or '1'. If we assume the most significant bit is as a '1', then we can re-write equation (4.2) as:

$$N = 2^k \left(1 + \sum_{i=j}^{k-1} 2^{i-k} n_i\right) \tag{4.3}$$

Let the term $\sum_{i=j}^{k-1} 2^{i-k} n_i = F$, then F is in the interval $0 \le F < 1$ since $k \ge j$ and N is:

$$N = 2^k \left(1 + F\right) \tag{4.4}$$

Then, $\log_2(N)$ is:

$$\log_2\left(N\right) = k + \log_2\left(1 + F\right) \tag{4.5}$$

The approximation method that Mitchell proposed in [76] is expressed as:

$$\log_2\left(N\right)_{Mit} = k + F \tag{4.6}$$

So, the error due to the approximation method can be described as:

$$Err_{Mit} = \log_2\left(1 - F\right) - F \tag{4.7}$$

The maximum error occurs when the fractional part F is 0.5. For example, if the integer number N=6, the binary representation for N would be 110. Using the approximation method described above, the index for the leading '1' bit (most significant '1' bit) would be 2 and the fractional part in binary would be 0.10 or 0.50 in decimal; therefore, the

result $\log_2(N)$ obtained by the estimation method is 2.50. The actual value of $\log_2(6)$ is 2.5850, so the error is about 0.0850. The approximation concept is illustrated in Figure 4.1 where an 8-bit number N is equal to 254. Using the index locating mechanism, the integer part of the $\log_2(N)$ is extracted as 7 and the remaining bits "1111110" are appended to the result as a fractional part as shown in Figure 4.1. For this example, the approximated $\log_2(N)$ is equal to 7.98438 in decimal whereas the actual value of $\log_2(N)$ is 7.98868; hence, the error is approximately 0.00430 as illustrated in Figure 4.1.

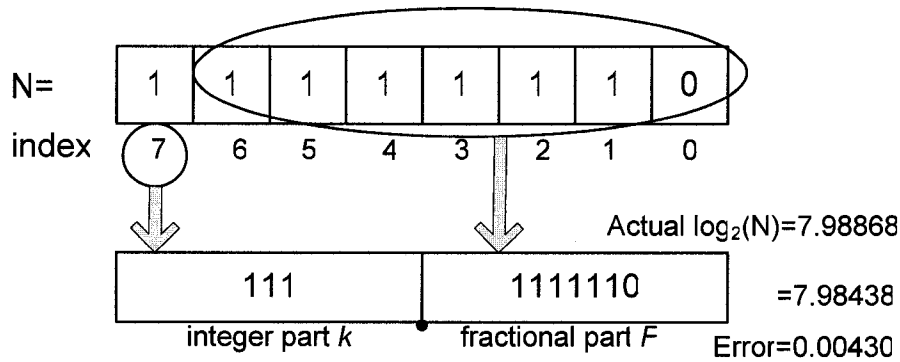


Figure 4.1. An example to illustrate the concept of $\log_2$ approximation method.

Similarly, the inverse $\log_2$ can be found based on the approximation technique. The approximation method for inverse-$\log_2$ implements the reverse-procedure of the $\log_2$ approximation where the integer part of the input number is interpreted as the index for the leading '1' bit in the result and the fractional part is appended to the result after the leading '1' bit. Figure 4.2 shows the approximated curve obtained from the Mitchell's method and the actual curve (obtained by Matlab program) for the $\log_2$ of an integer N. As shown in Figure 4.2, the approximation method has no error when the number N is a perfect power-of-two number such as 128 and 256. The maximum error of the binary logarithm approximation method occurs at the mid-point between the two consecutive power-of-two numbers such as 192. Correction methods for the binary logarithm approximation method are investigated in the next section. The approximated inverse $\log_2$ curve and the actual inverse-$\log_2$ (or 2^N) curve (obtained by Matlab program) input an number N are shown in Figure 4.3.

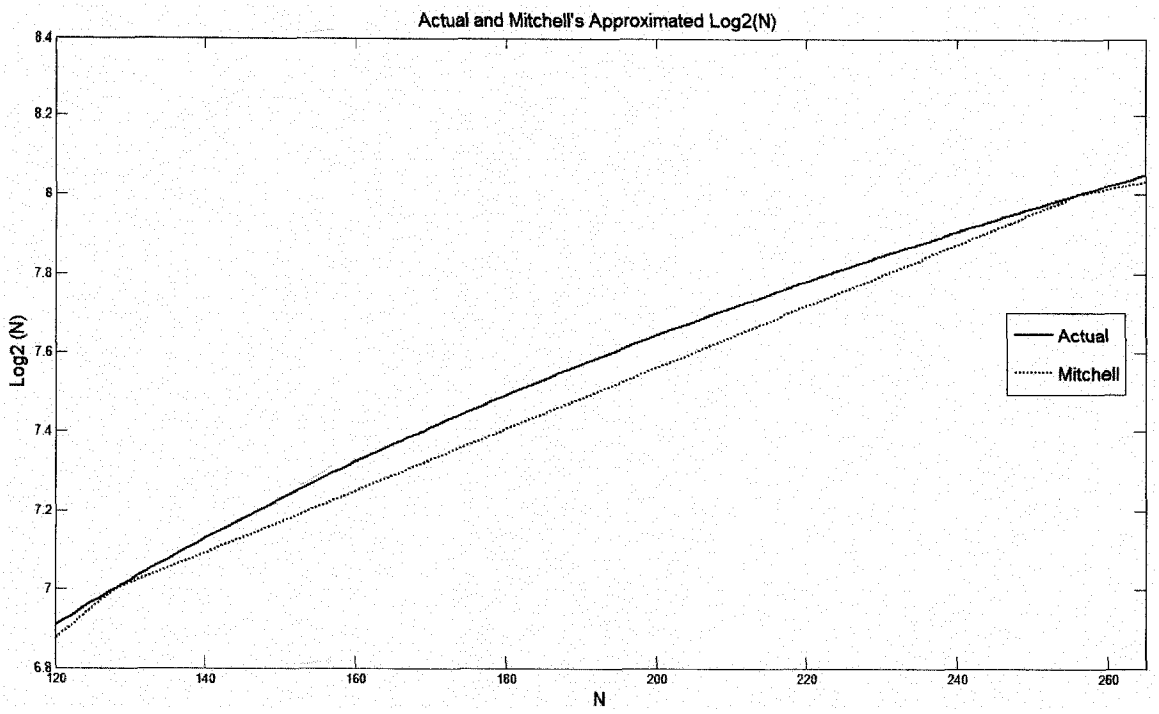


Figure 4.2. Actual values and Mitchell's approximated values of $\log_2(N)$.

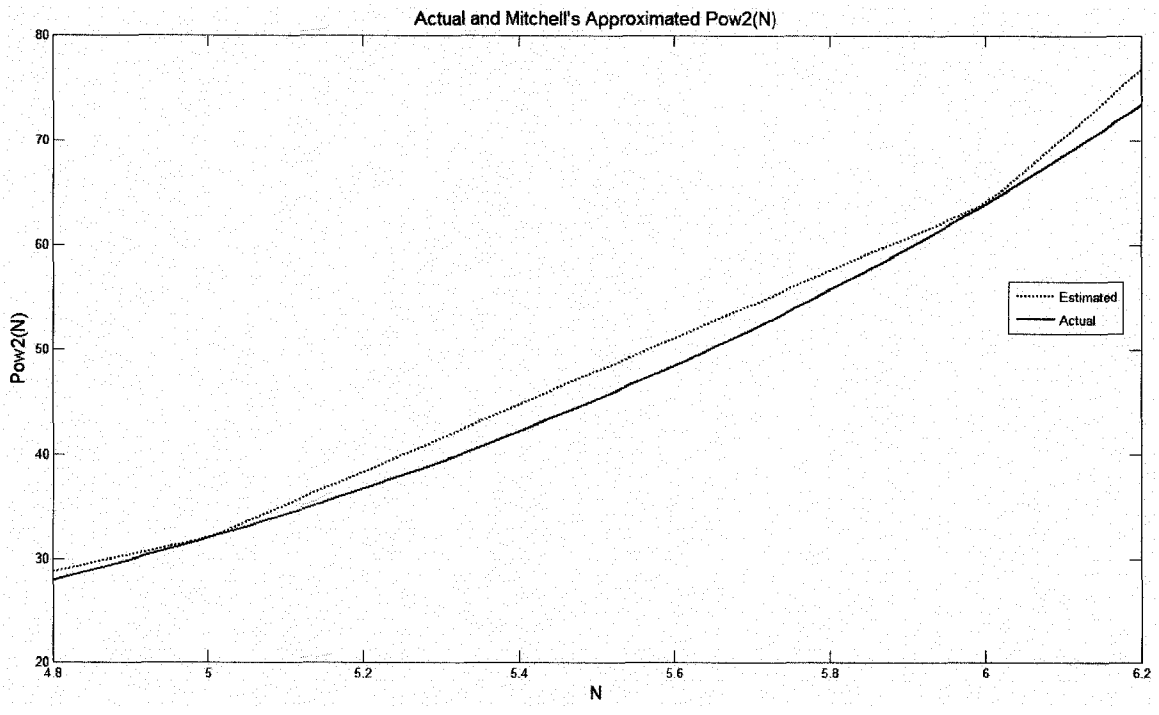


Figure 4.3. Actual values and Mitchell's approximated values of inverse-$\log_2(N)$.

4.2. Error Correction for Binary Logarithm Approximation

While Mitchell's method provides a fast and efficient way to compute $\log_2(N)$, it also introduces a large error range, which is $0 \le Err_{Mit} \le 0.08496$. The error percent $ErrPercent_{Mit}$ is in the range of $0 \le ErrPer_{Mit} \le 5.3605$. Many correction methods have been proposed to improve Mitchell's approximation technique. These methods include operand decomposition and look-up table (LUT) based and region-based approaches [77-81]. Mahalingam et al. [77] proposed an operand decomposition method to improve the accuracy of Mitchell's method in multiplication operations. This approach requires the generation of four intermediate numbers and performs Mitchell's approximation for each of them before summing the results. Two inverse-log operations are required before the final adder for the multiplication result. In addition, to achieve significant error reduction, one of the correction methods is needed to combine with this operand decomposition approach. A LUT-based approach achieves good correction results but the drawback is that it needs large storage overhead [78]. The region based correction approaches are very effective with low overhead [79-81].

In [79], Hall et al. proposed a method to divide the fractional parts into four sub-intervals and use different correction coefficients for each sub-interval. The correction coefficients are obtained from trial and error techniques. The four correction equations for four sub-intervals are:

$$\log_2(1+F)_{Hall} = F + \frac{37}{128}F + \frac{1}{128} \qquad \text{for } F \in [0.00, 0.25) \tag{4.8}$$

$$\log_2(1+F)_{Hall} = F + \frac{3}{64}F + \frac{1}{16} \qquad \text{for } F \in [0.25, 0.50) \tag{4.9}$$

$$\log_2(1+F)_{Hall} = F + \frac{7}{64}F' + \frac{1}{32} \qquad \text{for } F \in [0.50, 0.75) \tag{4.10}$$

$$\log_2(1+F)_{Hall} = F + \frac{29}{128}F' \qquad \text{for } F \in [0.75, 1.00) \tag{4.11}$$

where $F' = (1 - F)$. This approach reduces the error percentage $ErrPercent_{Hall}$ to a fraction within the range of $-0.7812 \leq ErrPer_{Hall} \leq 0.1258$.

SanGregory et al. [80] proposed a method that divides the fraction into two sub-regions that use a straight line with increased slope in the upper half and a straight line with decreased slope in the lower half. The slopes of the lines are adjusted such that the midpoint of each line intersects with the actual $\log_2(N)$ curve forcing the error to be zero at the midpoint. This correction method reduces error percent of Mitchell's approximation method within the range of $-1.5403 \leq ErrPer_{SanGreg} \leq 0.4314$.

Abed and Siferd [81] presented another region-based approach to compensate for the error in Mitchell's approximation method. In this method, the authors proposed correction equations for different sub-intervals. While Hall et al. use all the fractional bits in the correction procedure, Abed and Siferd use three or four most significant bits in their correction equations. The equations for a two-region correction strategy are:

$$\log_2(1+F)_{Abed} = F + \frac{1}{4}F_{3MSB} \qquad \text{for } F \in [0.00, 0.50) \tag{4.12}$$

$$\log_2(1+F)_{Abed} = F + \frac{1}{4}F'_{3MSB} \qquad \text{for } F \in [0.50, 1.00) \tag{4.13}$$

where $F'_{3MSB} = \left(1 - F_{3MSB} - \frac{1}{8}\right)$. This correction method reduces the error percentage of Mitchell's approximation method within the range of $-0.9299 \leq ErrPer_{Abed2} \leq 0.5544$. Similarly, Abed and Siferd proposed another correction method with consideration of three sub-regions. The correction equations for these three sub-regions are:

$$\log_2(1+F)_{Abed} = F + \frac{1}{4}f_{4MSB} \qquad \text{for } F \in [0.00, 0.25) \tag{4.14}$$

$$\log_2(1+F)_{Abed} = F + \frac{1}{4} + \frac{1}{64} \qquad \text{for } F \in [0.25, 0.75) \tag{4.15}$$

$$\log_2(1+F)_{Abed} = F + \frac{1}{4}F'_{4MSB} \qquad \text{for } F \in [0.75, 1.00) \tag{4.16}$$

where $F'_{4MSB} = \left(1 - F_{4MSB} - \frac{1}{16}\right)$. This correction method reduces the error percentage of Mitchell's approximation method within the range of $-0.4314 \le ErrPer_{Abed3} \le 0.2684$.

4.3. Proposed Error Correction for Binary Logarithm Approximation

The proposed method to correct error due to the approximation technique for binary logarithm is similar to the previous region-based approaches. As discussed in the previous section, no error due to Mitchell's approximation occurs for power-of-two numbers such as four, eight etc., and the maximum error occurs at mid point between two consecutive power-of-two numbers. The proposed method considers the maximum error which occurs in Mitchell's approximation technique as the starting correction coefficient. To reduce arithmetic operations, a coefficient is chosen such that the correction procedure can be performed with a small number of inversion, shifting and addition operations. The general expressions for the correction method with consideration of all the bits in the fraction F are:

$$F = \left(f_{-1}, f_{-2}, f_{-3}, \ldots, f_j\right) \tag{4.17}$$

$$\log_2(1+F)_{pro} = F + F_{pro1} \tag{4.18}$$

$$F_{pro1} = \left(\frac{1}{2^4} + \frac{1}{2^6}\right)F'_1 \tag{4.19}$$

$$F'_1 = \left(0, f_{-1} \oplus f_{-2}, f_{-1} \oplus f_{-2}, \ldots, f_{-1} \oplus f_j\right) \tag{4.20}$$

As shown in equations (4.17) to (4.20), this approach requires (*j-1*) XOR operations for bit inversion when the most significant bit (MSB) f_{-1} of the fraction F is '1'. The correction procedure has two shift operations and two add operations. This correction

method reduces the error percentage of Mitchell's approximation method within the range of $0 \leq ErrPer_{pro1} \leq 1.4154$.

Next, bit f_{-2} of the fraction F is considered to further reduce the error in the approximation method. The expressions for this approach which considers two bits in the fraction to generate correction terms are:

$$\log_2(1+F)_{pro} = F + F_{pro1} + F_{pro2} \tag{4.21}$$

$$F_{pro2} = \left(\frac{1}{2^5} + \frac{1}{2^7}\right) F_2' \tag{4.22}$$

$$F_2' = \left(0, 0, f_{-2} \oplus f_{-3}, f_{-2} \oplus f_{-4} \ldots, f_{-2} \oplus f_j\right) \tag{4.23}$$

This method requires the same number of operations as in the previous method when the bit f_{-1} is considered. Furthermore, additional (j-2) XOR operations for bit inversion when bit f_{-2} of the fraction F is '1', two shift operations and two add operations are required. This correction method reduces the error percentage of Mitchell's approximation method within the range of $-0.0441 \leq ErrPer_{pro2} \leq 0.5743$.

The error percentage curves of the Mitchell's approximation method and the two proposed methods are shown in Figure 4.4. The errors of the proposed correction method are generated for 10000 consecutive numbers and the numbers are shown in log scale in Figure 4.4. With the 1-bit consideration for region division, the proposed correction method reduces error range by more than 84% compared to that of the baseline Mitchell's approximation method. Similarly, the correction method that considers two bits for region division achieves more than 88% error reduction compared to the baseline approximation method.

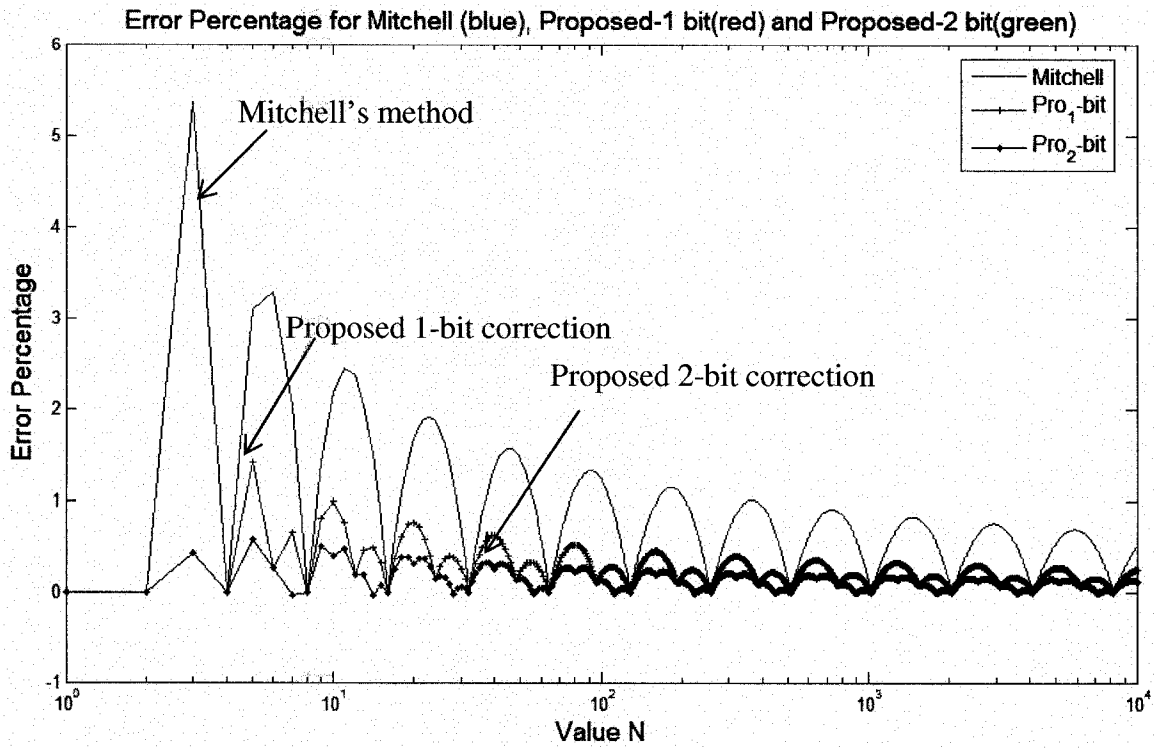


Figure 4.4. Error percent curves of Mitchell's and proposed methods.

Table 4.2 provides a summary of error percentage peak, ranges and mean square error percentage (MSEP) for Mitchell's method and the two proposed correction methods. The numbers in parentheses are the normalized number. The additional operations due to correction procedures for each proposed method are also provided in Table 4.2. For video processing applications that can tolerate small errors, methods with consideration for 1-bit and 2-bits are good candidates for performing log-based computations because of their simplicity and low overhead. Table 4.3 shows the result comparisons of the peak error percentages and error percentage ranges of the proposed methods with other methods by Mitchell [76], SanGregory et al. [80], Hall et al. [79] and Abed et al. [81]. As shown in Table 4.3, for methods that consider 1 bit for dividing the fractional parts into two

regions, the proposed method in this research work has the lowest error percentage range. Similarly, the proposed 2-bits approach also has the smallest error percentage range among all other methods that use 2 bits for region partitioning.

Table 4.2. Error evaluation and overhead of proposed correction methods for binary logarithm approximation.

Method	**Mitchell method**	**Proposed 1-bit method**	**Proposed 2-bit method**
Number of parallel additions	None	2	5
Max. number of bit inversion	None	j-1	$2j$-3
Max. (+) error percentage	5.3605	1.4154	0.5743
Max. (-) error percentage	0	0	-0.0441
Error percentage range	5.3605 (1.00)	1.4154 (0.26)	0.6184 (0.12)
MSEP	0.2848 (1.00)	0.0333 (0.12)	0.0092 (0.03)

Table 4.3. Comparisons with other error correction methods.

Method	**Bits used for regions identification**	**Max. (+) error percentage**	**Max. (–) error percentage**	**Error percentage range**
Mitchell	None	5.3605	0	5.3605
SanGregory et al.	1	0.4314	-1.5403	1.9717
Abed$_2$ et al.	1	0.9299	-0.5544	1.4843
Proposed 1-bit	**1**	**1.4154**	**0**	**1.4154**
Hall et al.	2	0.1258	-0.7182	0.9071
Abed$_3$ et al.	2	0.4314	-2684	0.6998
Proposed 2-bit	**2**	**0.5743**	**-0.0441**	**0.6184**

4.4. Design and Implementation of a Log-Based Multiplier

In this section, a fast and efficient design of a log-based multiplier that can be used for signal processing applications is presented. The bit-level partitioning and gating technique is used to detect a special condition in the higher half of the operand in the window-based operation and deactivate the architecture appropriately to reduce power dissipation. For implementation, Mitchell's approximation algorithm is used as a baseline method to compute binary logarithm. The block diagram for the log-based multiplier ($P=X\times Y$) is shown in Figure. 4.5. Since $\log_2$ of any number less than or equal to 0 is not valid, data adjustment modules such as two's complement to sign magnitude conversion module and zero detection and adjustment module are included to make sure that valid results are produced.

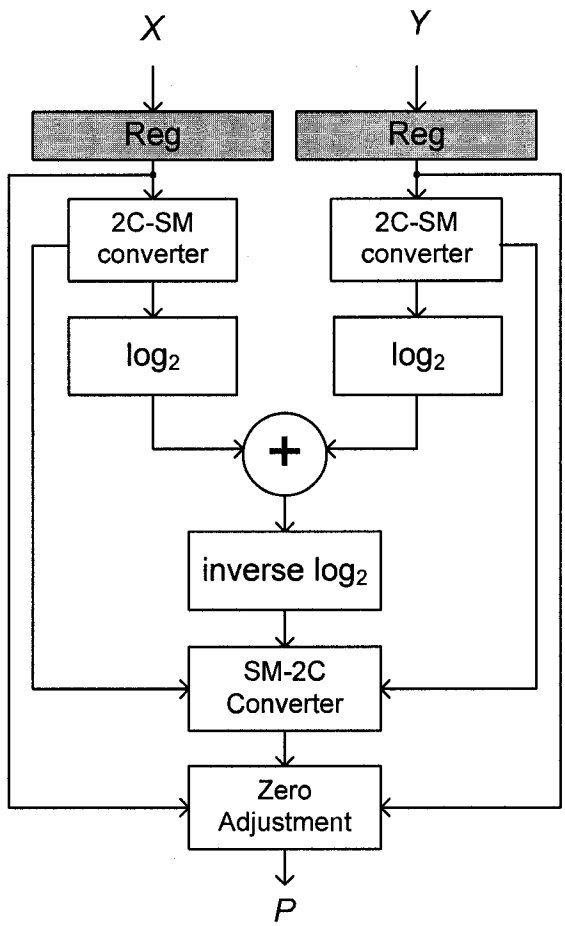
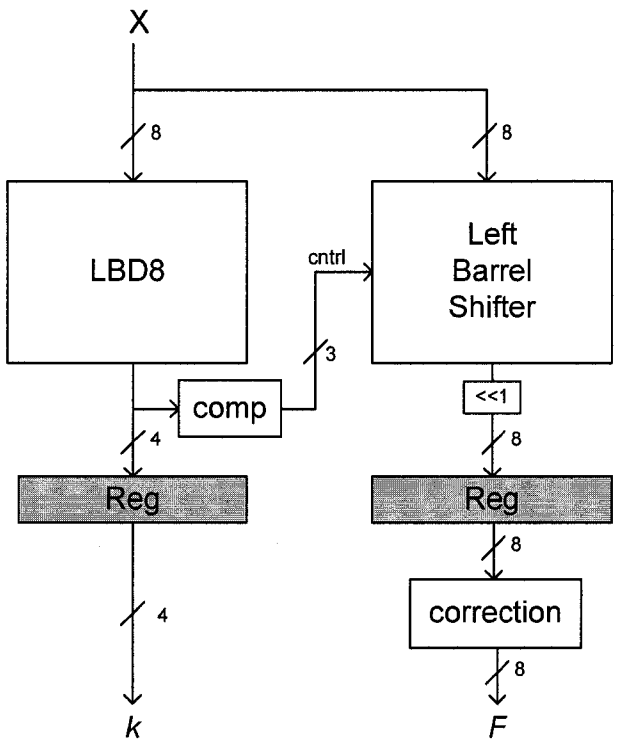
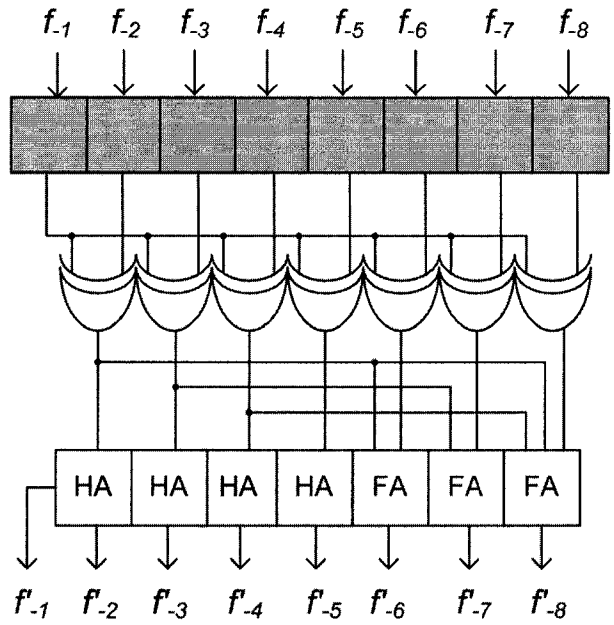


Figure 4.5. Schematic of a log-based multiplier.

The main computational modules in the log-domain multiplier are the binary logarithm and inverse-logarithm modules. The approximation method requires a leading '1' bit detection and a shifting mechanism to compute the integer and fractional parts of the binary logarithm result. The block diagram of a hardware architecture to approximate a binary logarithm of an 8-bit number X is shown in Figure 4.6. The architecture consists of an 8-bit leading bit detection (LBD8) module, a 3-bit complement unit (comp) and an 8-bit barrel shifter. The LBD8 module detects the position of the leading '1' bit and outputs the 4-bit word k that indicates the position of the leading '1' bit. The three least significant bits (LSBs) of k are inverted to generate a 3-bit control bus (*ctrl*) that directs the number of bits to be shifted in the barrel shifter. An additional bus shift operation is performed on the output of the barrel shifter to obtain the fractional part F of the binary logarithm.



(a) Binary logarithm approximation with correction.



(b) Correction circuit for proposed 1-bit approach.

Figure 4.6. Architecture for binary logarithm computation of an 8-bit number.

A barrel shifter is used to generate the fractional part of the binary logarithm as shown in Figure 4.6. For an m-bit input number x, the barrel shifter generates an m-bit output number xs which is the value x shifted by a number of bit position specified by $\log_2(m)$-bit control bus *cntrl.* An architecture design which consists of $\log_2(m)$ stages of m multiplexers is used for the design of an m-bit barrel shifter in this work. Figure 4.7 shows the architecture for an 8-bit left barrel shifter. As the length of the data words become larger, propagation delay of the barrel shifter increase considerably which causes more switching activities in a parallel structure. Pipeline stages can be used to store intermediate results of the multiplexer stages to reduce propagation delay and switching activities. The drawback is that the latency increases as additional register stages are inserted. In this research work, additional pipeline stages are added in the implementation of a barrel shifter if the data length is larger than eight.

4.4.1. Design of the Leading Bit Detection Unit

The leading bit detector used in the binary logarithm must be fast and simple to take advantage of the approximation method. While leading bit detecting circuits are common in floating point arithmetic, they are usually too complex for the approximation method. Mitchell and others used the standard shifter and down counter to determine the leading '1' bit [76]. This is the simplest approach but it requires a maximum of m cycles for an m-bit number. The latency varies depending on the actual position of the leading '1' bit. SanGregory et al. [80] presented a serial evaluating circuit with a look-ahead mechanism to compute the integer part of the binary logarithm. This approach is efficient, but the propagation delay for the entire circuit increases as m becomes larger, which reduces the operating frequency. Abed et al. [81] addresses this propagation delay issue with a modular approach where multiple 4-bit leading '1' detectors can be used in parallel. This approach combines both serial and parallel processing in the circuit. Each individual 4-bit leading '1' detector performs operations serially and multiple 4-bit leading '1' detectors perform in parallel. This approach is one of the most efficient methods suitable for the binary logarithm approximation technique.

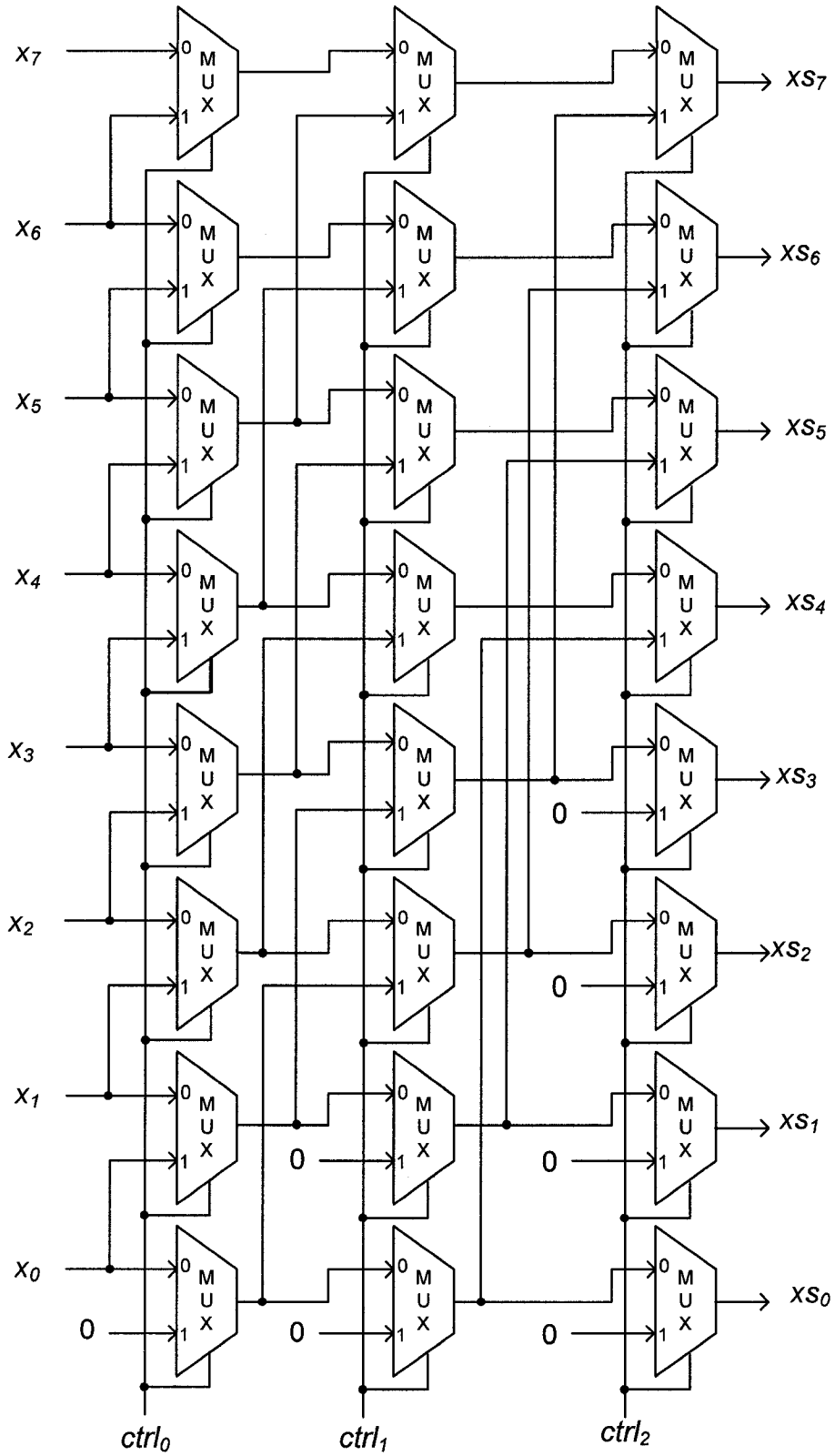
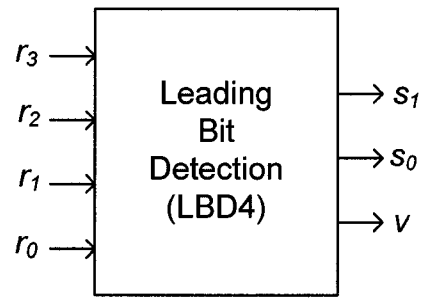
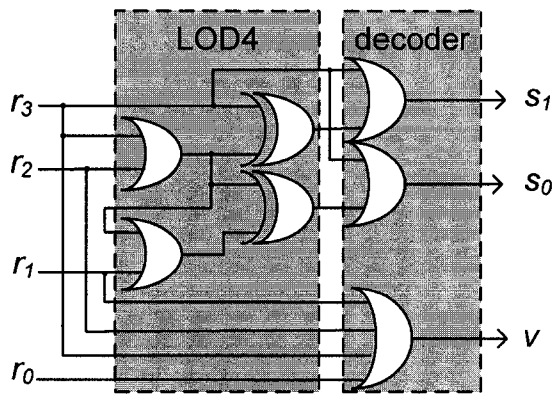


Figure 4.7. An architecture for a 8-bit left barrel shifter.

The proposed design of the leading bit detector in this work is similar to that of Abed et al. [81] in which a modular approach is employed with multiple 4-bit leading bit detectors (LBD4) for larger data words. The LBD4 units perform operations in parallel, and the results are used in subsequent units that generate final results. The operations in each LBD4 are performed serially. While methods developed by Abed et al. and SanGregory et al. require a separate ROM based decoder to generate the integer part k, the decoder design for computing k in this work is incorporated in the LBD4 units with small additional logic gates. With the proposed design, multiple LBD4 units can be used for different length data words without significant modification to the circuit. For custom ROM based decoders such as those used in Abed et al. and SanGregory et al., each data length requires a separate ROM decoder. The circuit design for a 4-bit leading bit detector is shown in Figure 4.8, where the 4-bit input word is r and the 2-bit position of the leading '1' bit is s.



(a) Symbol.



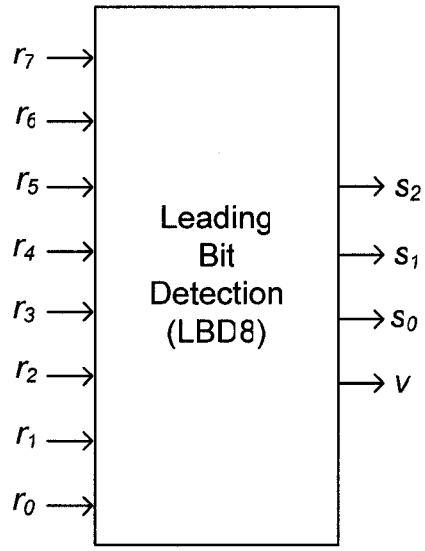
(b) Logic circuit.

Figure 4.8. 4-bit leading bit detector (LBD4).

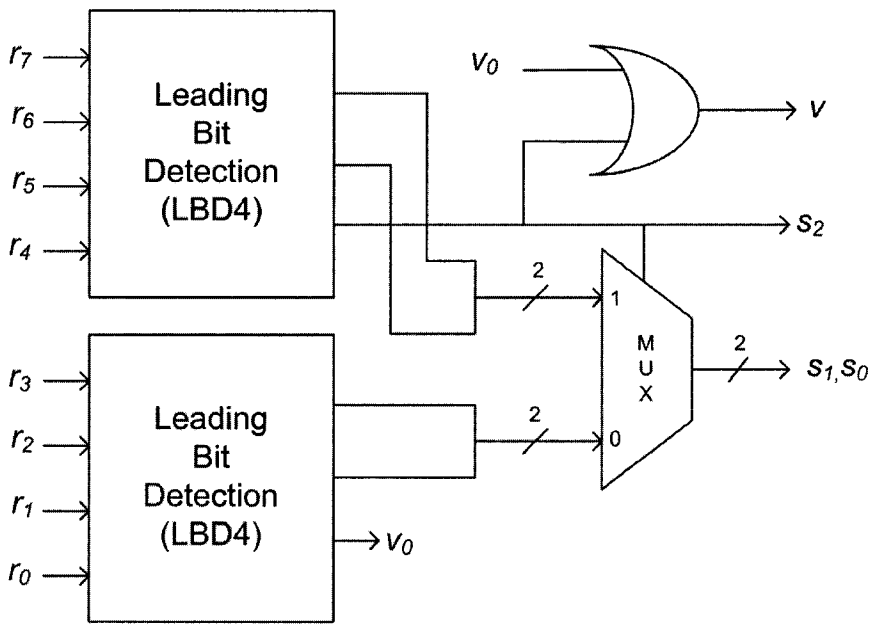
The valid bit v is included to indicate whether all bits in the input word r are '0'. In the circuit design, two logic stages are shown. The first stage is a circuit to determine the leading '1' bit using a LOD4 unit which generates 3-bit results for three MSBs. No more than one bit can be set to '1' at a given time. A logic '1' at any output bit of the LOD4 unit indicates that the leading '1' bit is present at that position. The 3-bit result of the LOD4 unit is used in the second stage, which is a decoder to generate the position s of the leading '1' bit. For example, if a '1' is set for the MSB at the output of the LOD4, the decoder generates a '11' for s. A '1' is set for the middle bit of the output of the LOD4 unit which triggers the decoder to generate a '10' for s. Similarly, a '01' is generated by the decoder if a '1' is detected at the LSB of the output of the LOD4 unit.

For data words with larger bit lengths such as 8, 16 or 32, multiple LBD4 units are used in parallel. The block diagram of an 8-bit leading bit detector (LBD8) is shown in Figure 4.9. In this Figure, two LBD4 units are used for the most significant 4 bits and the least significant 4 bits of the input r respectively. The valid bit v_1 of the higher order LBD4 unit is used to select the results of the higher or lower order LBD4 for the final output via a 2-bit 2:1 multiplexer. In addition, the valid bit v_1 of the higher order LBD4 unit is also used as the MSB of the result s. An OR gate with inputs v_1 and v_0 is used to generate the valid bit for the LBD8 as shown in Figure 4.9. The propagation delay of this LBD8 design is the sum of the propagation delays of a LBD4 and a multiplexer.

Similarly, a 16-bit leading bit detector (LBD16) is designed with a combination of five LBD4 units and a 2-bit 4:1 multiplexer as shown in Figure. 4.10. In the first stage of the LBD16, four LBD4 units process data simultaneously to generate four 2-bit results *ts's* and four valid bits *v's*. The four valid bits are then used to feed the second stage LBD4 unit to determine which block has the leading '1' bit. The result s from the second stage LBD4 unit is then used as select lines for the 4:1 multiplexer. The 2-bit s of the second stage LBD4 unit are also the most significant two bits of the final result s (s_3s_2). The output of the multiplexer contains the least significant two bits of the result s (s_1s_0). The valid bit from the second stage LBD4 unit indicates if the input r is zero (when v is '0').



(a) Symbol.



(b) Logic circuit.

Figure 4.9. 8-bit leading bit detector (LBD8).

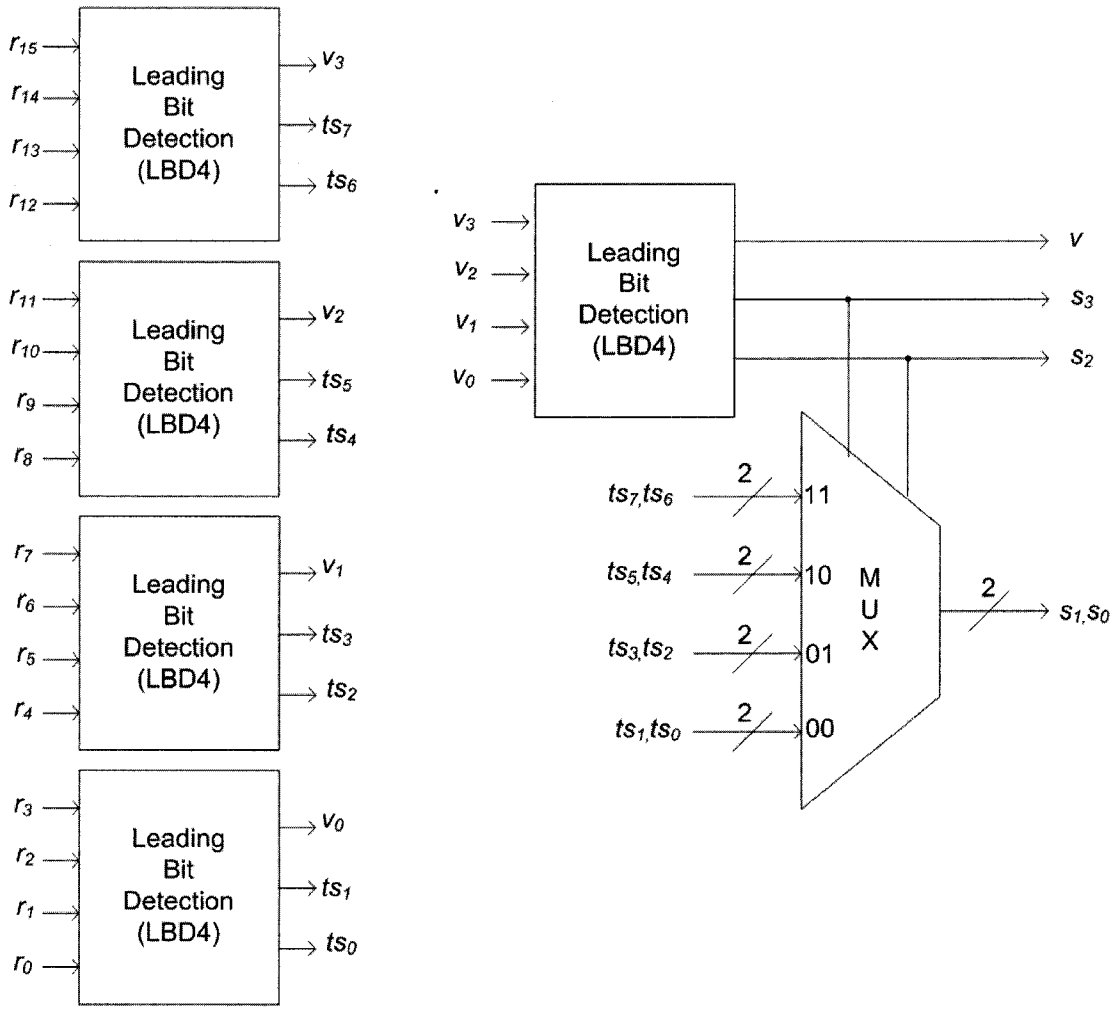


Figure 4.10. 16-bit leading bit detector (LBD16).

Similarly, two LBD16 can be used to detect leading '1' bit for 32-bit word as shown in Figure 4.11. Each LBD16 unit generates a 4-bit result to indicate the position of the leading '1' bit in that unit, and a valid bit to indicate that the input bits are not all zeros. Two valid bits obtained from two LBD16 units are used to generate the valid bit for the LBD32 unit as shown in Figure 4.11. The valid bit obtained from the higher order LBD16 unit is used as a controlling bit for a 4-bit multiplexer which selects the appropriate 4-bit result from one of the LBD16 units as the least significant four bits of

the final results. In addition, the valid bit from the higher order LBD16 is also used as the MSB of the final result.

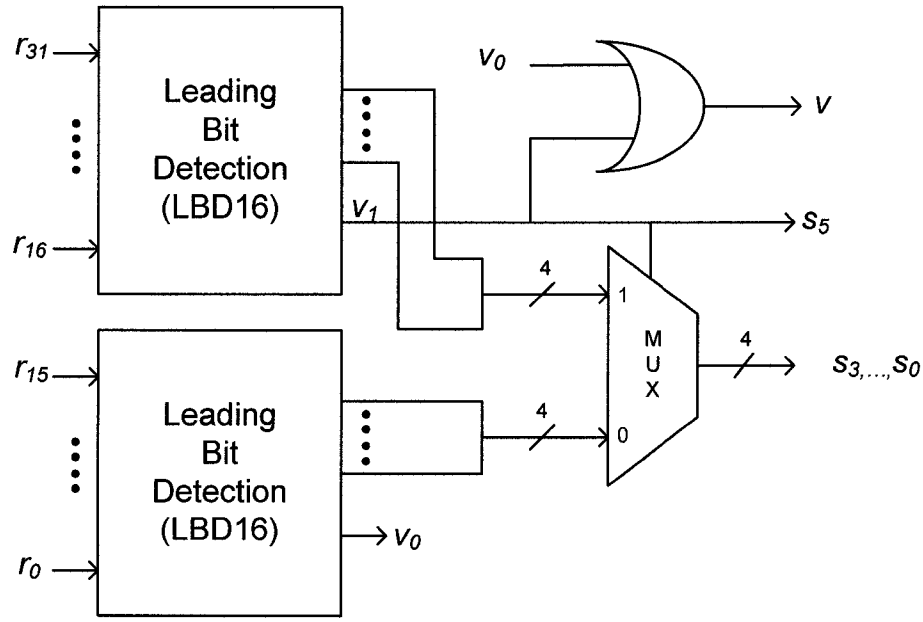


Figure 4.11. 32-bit leading bit detector (LBD32).

4.4.2. Design of a Power-Aware Log-Based Multiplier

Similar to the approach considered in the low power design of a multiplier in the linear domain, the proposed design for the log-based multiplier considers the special characteristic of image and video processing applications. That is, one operand of the multiplication frequently has a small magnitude in data representation. This characteristic occurs regularly in window-based operations such as digital filtering. The data characteristics of these applications are that magnitudes of the filter coefficients are usually small. To take advantage of this characteristic, the proposed log-based multiplier design partitions the coefficient operand into smaller parts that can be computed with multiple smaller computational units. With this approach, the computational modules can be disabled when the special conditions such as input data equals to zero or one is detected.

Let's consider the multiplication of two n-bit numbers X and Y in the multiplier which is used in a digital filtering operation. Assume that Y is the coefficient that is partitioned into smaller part and X is the pixel value. The partitioning procedure for the multiplication of $P = X \times Y$ is described as

$$P = X \times Y \tag{4.27}$$

$$P = \left(Y_H \times 2^m + Y_L\right) \times X \tag{4.28}$$

$$P = \left(X \times Y_H \times 2^m\right) + \left(X \times Y_L\right) \tag{4.29}$$

where,

$$X = \left(x_{n-1}, x_{n-2}, x_{n-3}, \ldots, x_0\right)$$

$$Y = \left(y_{n-1}, y_{n-2}, y_{n-3}, \ldots, y_0\right)$$

$$Y_H = \left(y_{n-1}, y_{n-2}, y_{n-3}, \ldots, y_m\right)$$

$$Y_L = \left(y_{m-1}, y_{m-2}, y_{m-3}, \ldots, y_0\right)$$

Since most of the coefficients in different filter kernels have small magnitude, detection of zero and one conditions in the higher half of the coefficient Y is performed. The gating technique is based on two conditions in the Y_H and Y_L operands. These conditions are set to logic '1' when an input is zero or one is detected and the gating signals z_H, z_H, o_H and o_L are generated as:

$$tmp = y'_{n-1} \cdot y'_{n-2} \cdot y'_{n-3} \cdots \cdot y'_{m+1} \tag{4.30}$$

$$z_H = tmp \cdot y'_m \tag{4.31}$$

$$o_H = tmp \cdot y_m \tag{4.32}$$

$$tmp2 = y'_{m-1} \cdot y'_{m-2} \cdot y'_{m-3} \cdots \cdot y'_1 \tag{4.33}$$

$$z_L = tmp2 \cdot y'_0 \tag{4.34}$$

$$o_L = tmp2 \cdot y_0 \tag{4.35}$$

The circuits for detection of z_H, z_L, o_H and o_L are shown in Figure. 4.12. Figure 4.13 shows the architecture design for a pipelined log-based multiplier which utilizes the partitioning and gating technique based on the characteristics of the multiplier operands. The special value detection (SVD) module in Figure 4.13 is the component to generate z_H, z_L, o_H and o_L signals for the gating technique. Pipelined flip-flops (ff) are included to propagate these control signals to subsequent pipeline stages as shown in Figure 4.13. Figure 4.13 does not show the data conversion and zero adjustment modules since these are needed for all log-based multipliers.

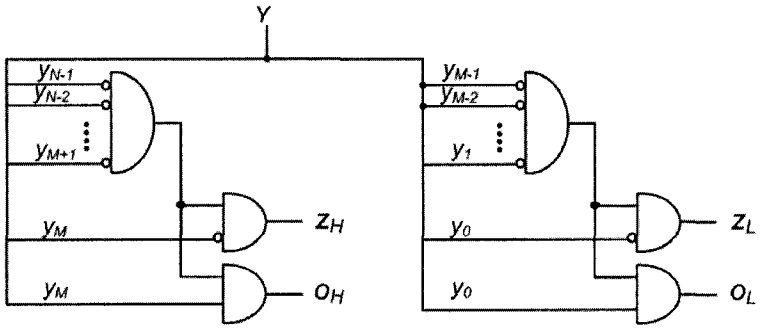


Figure 4.12. Circuits for generating gating signals in log-based multiplier design.

4.5. Experimental Results

The proposed log-based multiplier is implemented with Altera's Quartus II software tool. Mitchell's binary logarithm approximation method is also implemented as a baseline design for comparison. Multiplications in 2-D convolution with a Gaussian kernel for image smoothing and Laplacian kernel for edge detection are used in the simulation. The implementation results are shown in Table 4.4 where Mitchell_REG is the implementation for regular Mitchell's method, and Mitchell_PG is the implementation of the proposed partitioning and gating technique applied to Mitchell's algorithm. Pixel values are extracted from the test images and kernel coefficients are fed to the multiplier through a vector simulation file. The power consumption results for two applications are shown in Tables 4.5 and 4.6 respectively. Results of the simulations of a standard CSA array multiplier and a Booth multiplier are also shown in Tables 4.5 and 4.6 for

comparison between multiplication in the linear and logarithmic domain. The log-based multiplier achieves more than 50% power reduction compared to the conventional CSA array and Booth multipliers. The partitioning and gating techniques have significant impact or power consumption in the edge detection application compared to the baseline log-based multiplier, as shown in Table 4.6. This is due to smaller magnitude values of the kernel coefficients.

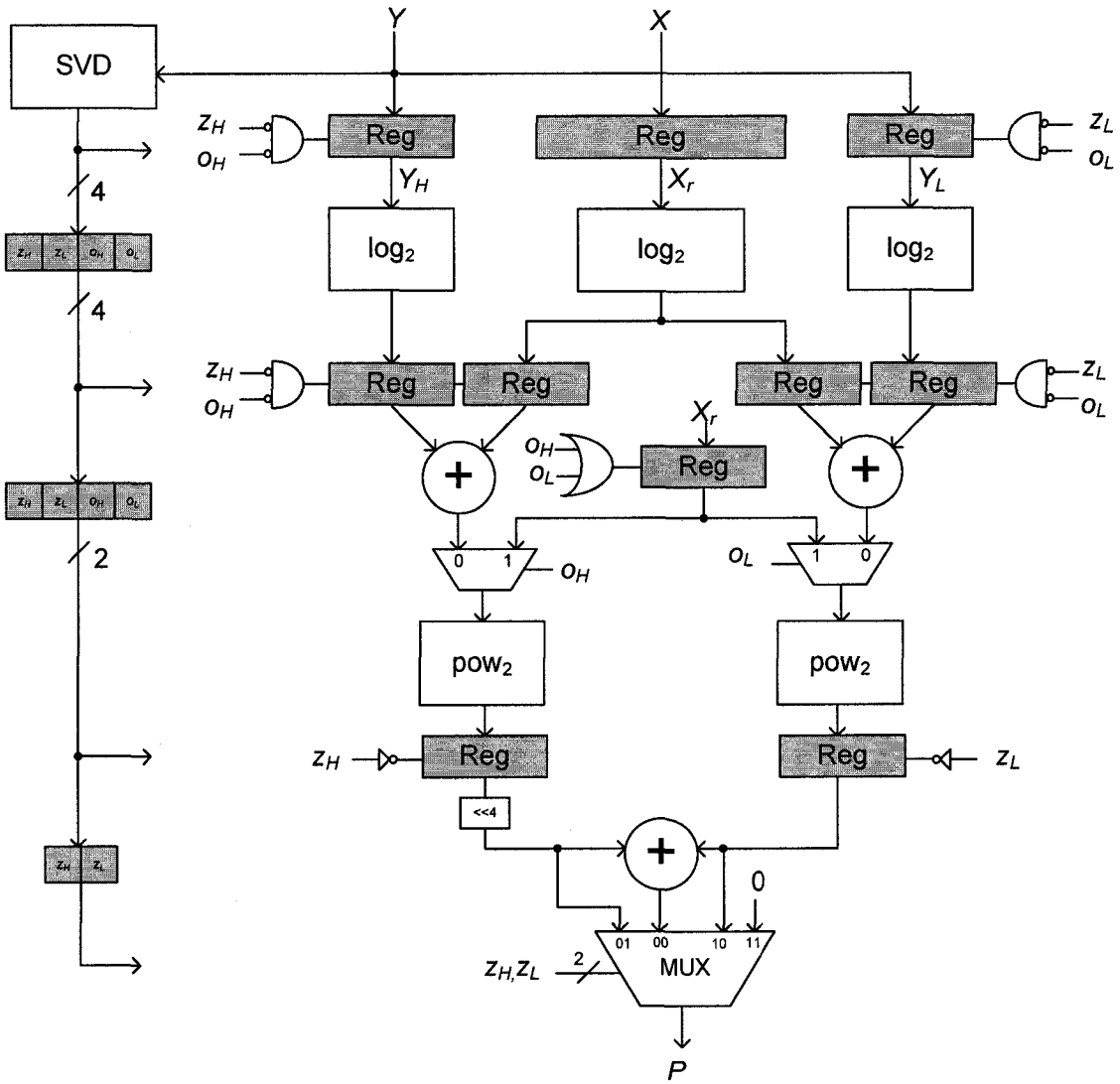


Figure 4.13. Design of a log-based multiplier with partitioning and gating technique.

Table 4.4. Implementation results of log-based multipliers.

Method	**Logic elements**	**F_{max}** (MHz)	**Latency**
Mitchell_REG	244	91	5
Mitchell_PG	434	89	6

Table 4.5. Experimental results of a Gaussian filter.

Test Image / **Method**	**Peppers** (128×128)		**Island** (160×106)	
	Power (mW)	**Normalized ratio**	**Power** (mW)	**Normalized ratio**
CSA	40.73	--	35.56	--
Booth	31.82	--	28.42	--
Mitchell_REG	19.40	1.00	18.35	1
Mitchell_PG	**18.86**	**0.97**	**17.63**	**0.96**

Table 4.6. Simulation results for a Laplacian filter.

Test Image / Method	**Peppers** (128×128)		**Island** (160×106)	
	Power (mW)	**Normalized ratio**	**Power** (mW)	**Normalized ratio**
CSA	35.57	--	33.47	--
Booth	30.32	--	27.31	--
Mitchell_REG	21.32	1	20.63	1
Mitchell_PG	**15.08**	**0.71**	**14.07**	**0.68**

4.6. Summary

In this chapter, a log-based multiplier has been developed, implemented and evaluated for digital filtering operations in video processing algorithms. An approximation method for binary logarithm with fast and efficient operations is used in the architecture design of the proposed log-based multiplier. Various error correction methods to improve the accuracy of the binary logarithm approximation technique have been investigated. A correction technique that uses one or two bits for region division in the correction procedure has been proposed to improve accuracy of the approximation method. Simulation results show that the proposed method performs favorably when compared to other correction methods. A novel design for fast and efficient leading bit detection has been presented and implemented for the binary logarithm approximation. The leading bit detection strategy is based on a modular approach that reduces the serial operations to a four-bit baseline LBD4 block. Multiple LBD4 units are combined to perform the operation on larger data words. The partitioning and gating technique has also been applied to the log-

based multiplier (Mitchell-PG). Simulation results show that the log-based multiplier reduces more than 50% in power consumption compared to the linear pipelined CSA array and Booth multipliers. The proposed Mitchell-PG multiplier achieved more than 29% power reduction compared to the baseline Mitchell-based multiplier in the edge detection operation.

Chapter V

POWER-AWARE DESIGN TECHNIQUE FOR WINDOW-BASED OPERATION IN VIDEO PROCESSING APPLICATIONS

Window-based operations such as two dimensional (2-D) convolution operations are commonly used in image and video processing applications. In this chapter, a new design technique that considers the neighborhood pixels within the window to detect and eliminate redundant or unnecessary computations for power reduction is presented. A novel on-chip detection technique is developed for the proposed neighborhood depended approach (NDA) to reduce computations. Similar to the techniques presented for the multiplier design in the previous chapters, data partitioning methodology is employed in the on chip buffer design to eliminate redundant computations. This NDA method is applied to different window buffering schemes and experimental results are presented.

5.1. Window-Based Operations

Window-based operations require a large number of repetitive computational operations on a fixed window of neighboring pixels centered on a reference pixel (pixel under consideration). A window-based operation is performed when a window with an area of $K \times L$ pixels for a $K \times L$ mask is extracted from an input image and is transformed according to the kernel mask to produce an output pixel. The concept of a window-based operation is illustrated in Figure 5.1, where a 3×3 kernel mask is used to compute the values of the output pixels by performing the required function with values in the kernel masking with the pixel values in the input image. To compute the pixel value of an output pixel, each reference pixel in the window of the input image is extracted and multiplied with the corresponding weight in kernel mask and the results are summed to produce the value of the output pixel. Some of the most frequently used window-based operations include smoothing, edge detection, template matching and morphological operations.

Window-based operation is one of the most computationally intensive operations in image processing applications. These operations require $(K \times L)$ multiplications and $(K \times L$

-1) additions to compute one pixel of the image where the size of the kernel mask is $K \times L$. Due to the repetitive nature in the applications, these window-based operations present a high degree of processing parallelism that can be exploited to achieve higher performance. Maximum utilization of the inherent parallelism in window-based operations is considered to satisfy the demand of real-time processing. Parallel processing is necessary to achieve demand throughput for real-time applications.

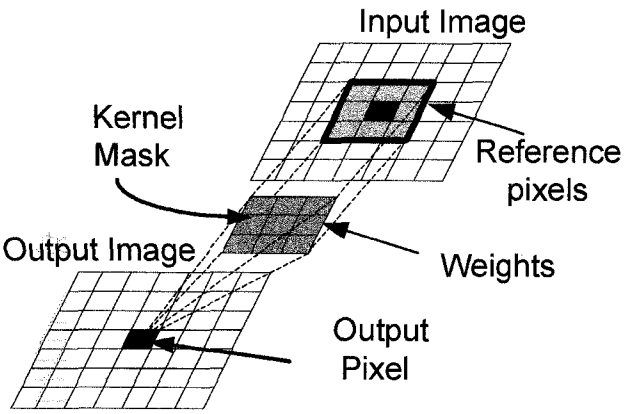


Figure 5.1. Concept of a window-based operation.

Architecture design techniques for parallel processing such as systolic, pipeline or parallel architectures often require a large amount of extra resources to support multiple processing elements (PEs). For example, in a systolic design that consists of a large number of interconnected homogeneous processing elements, considerable resources are needed to implement delay lines for partial results as data traverse through the architecture. This is a particularly important issue for FPGA-based designs in which area resource is a constraint. Some of the latest research approaches address this problem with a configurable option in their design to provide a capability to re-use the PEs for various window-based operations [22, 82-84]. While approaches in [22, 82-83] address the problems in VLSI design for application specific integrated circuits (ASICs), the technique proposed in [84] targets the FPGA environment for quick prototyping. The approaches proposed in [22, 82-83] generally exploit different ways of interconnecting the processing cores such that data is retrieved more efficiently. The main application for

these approaches is computation of the Fast-Fourier-Transform (FFT) to calculate the frequency response of a filter. In [84], the authors proposed an architecture design for processing element cores in a systolic architecture. The core is designed with a configurable capability to support re-using of PEs for larger kernels. In addition, the authors used the unrolling technique to compute the result in favor of overlapped data. This is done for reducing I/O communication with external memory.

A similar technique of unrolling computation is employed in [85-86]. Loop rolling is a procedure to replace iterations with straight-line executions. These approaches focus on general purpose computing with support for image processing algorithms; therefore, these two approaches do not fully exploit the inherent parallelism in the algorithms. Another FPGA-based approach that considers the overlapped data in window-based processing is proposed in [87]. The authors presented a generalized addressing mode that fetches a block of data from memory based on the window-operation accessing sequence. The main goal is to maximize the sharing of data among a sequence of operations. To support parallel processing for real-time applications, efficient on-chip buffering schemes must be employed to reduce I/O bandwidth with external memory.

5.2. On-Chip Window Buffering Schemes

For discussion of the on-chip buffering scheme design, 2-D convolution is considered to be the typical window-based operation. Spatial convolution operations are computational and memory intensive. For a 2-D convolution with a $K \times L$ kernel mask, $(K \times L)$ multiplications, $(K \times L-1)$ additions and $(K \times L)$ memory accesses to pixels in the reference window are required for one output pixel. In order to take advantage of the inherent parallelism in the operations, accesses to the reference pixels must be achieved simultaneously. This demands efficient on-chip buffering schemes to support a parallel architecture.

Generally, 2-D convolution can be computed by dividing the operation into separate 1-D convolutions based on either column-wise or row-wise directions. These 1-D convolution

modules are operated in parallel, and each of these 1-D convolution modules can also be designed as parallel or systolic pipelined architecture. 1-D convolutions are performed for all the rows (or columns) and the intermediate results are summed to obtain the final result for the output pixel. Systolic design can achieve a throughput rate of one output per cycle in raster scan order when the operation reaches steady state; however, this approach needs considerable resources to implement delay elements to synchronize with the input data stream [88]. A full-window buffering (FWB) scheme proposed by Bosi et al. [82] for parallel processing is also capable of producing an output per cycle in steady state. For the column-based 1-D module, with a $K \times L$ kernel and an image size of $R \times S$, this approach requires $(K-1)$ delay lines, each of which is a FIFO buffer with length $(R-L)$. Figure 5.2 shows the block diagram of this scheme. Each of the 1-D convolution modules shown in Figure 5.2 consists of L multipliers and an adder tree to sum all the products. The advantage of this approach is the single input dataflow that requires only one memory access per pixel. The disadvantage of this scheme is the large area overhead of the delay lines.

To eliminate the delay lines in the buffering schemes, a smaller number of pixels is maintained in the FIFO buffer with increased memory bandwidth demand [82, 89-90]. One approach is to maintain K number of FIFO buffers which feed the data to the shift registers for each 1-D convolution module. This approach is known as a single window partial buffering (SWPB) scheme. In each clock cycle, K pixels are shifted from the FIFO buffers to the shift registers. Shift registers in a particular row provide parallel access to all pixels in that row so that parallel computation in a 1-D convolution module can be performed simultaneously. This approach eliminates the need for delay lines, but it requires K memory accesses to maintain the dataflow in the buffer. Figure 5.3 shows the block diagram of this SWPB scheme. K FIFO queues are needed to support parallel loading of K rows in one clock cycle. The FIFO queues are interfaced directly with the external memory for inputting pixel values. While the SWPB scheme utilizes the overlapped pixels in the rows, Zhang et al. [91] proposed a multi-window partial buffering (MWPB) scheme to extend the SWPB approach for utilization of overlapped pixels in columns.

The idea is to utilize the data already in the FIFOs to reduce external memory accesses. The concept is illustrated in Figure 5.4. For each window $win_{x,y}$ to compute an output pixel (x,y), there are two neighboring windows $win_{x,y+1}$ and $win_{x+1,y}$ for output pixel $(x,y+1)$ and $(x+1,y)$ respectively, which have overlapped data (shaded areas) with $win_{x,y}$. The window $win_{x,y+1}$ has $(L\text{-}1)$ columns overlapped with the window $win_{x,y}$ while window $win_{x+1,y}$ has $(K\text{-}1)$ overlapped rows as shown in Figure. 5.4. The approach proposed in [91] has $(2K\text{-}1)$ FIFO buffers to support $(2K\text{-}1)$ row operations. This method requires accesses to a column of $(2K\text{-}1)$ pixels to be fed to the FIFOs, and $(2K\text{-}1)$ values are passed to the shift registers at a time to support parallel processing. In this method, a single datum from shift registers of each row is accessed at a time whereas the SWPB and FWB approaches access all data from a row simultaneously. The required memory bandwidth for this approach is $(2K\text{-}1)/S$ per cycle. The block diagram of this MWPB scheme is shown in Figure 5.5.

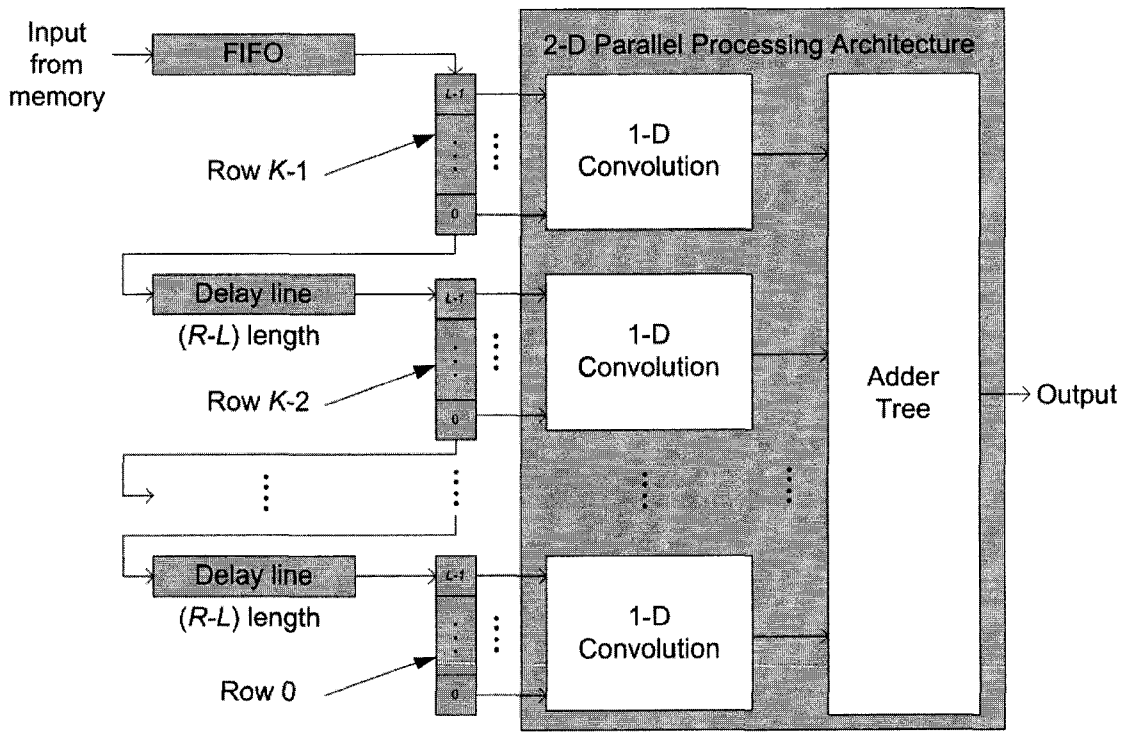


Figure 5.2. Full-window buffering (FWB) scheme for a parallel processing architecture.

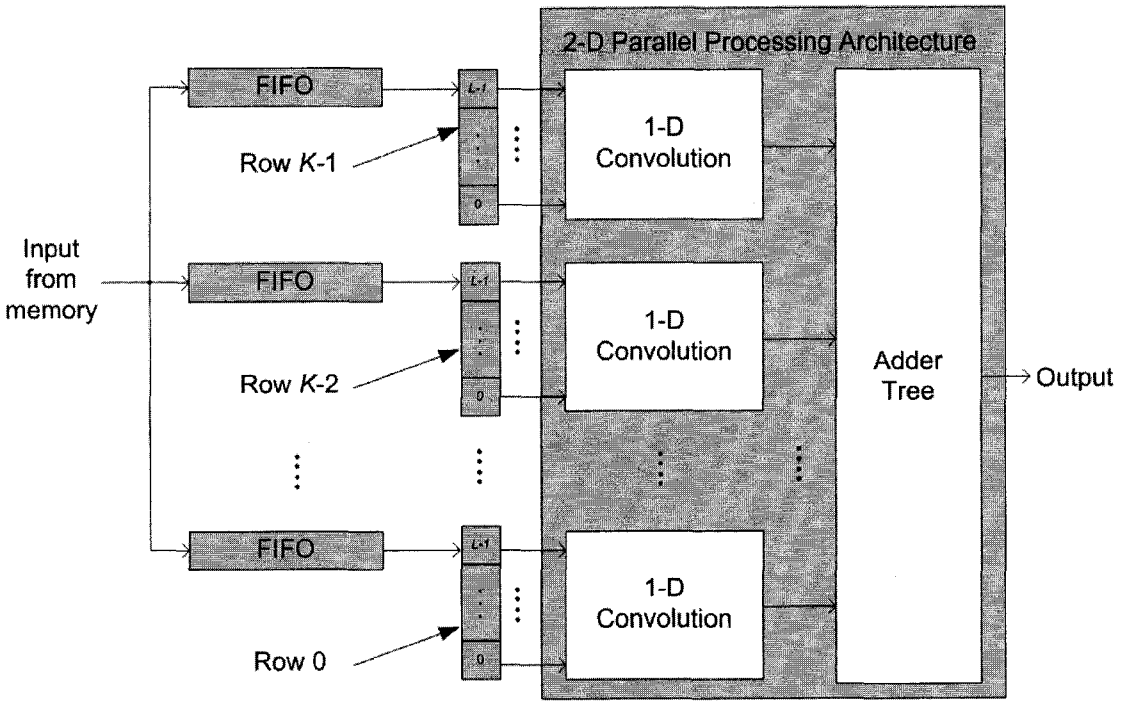


Figure 5.3. Single-window partial buffering (SWPB) scheme.

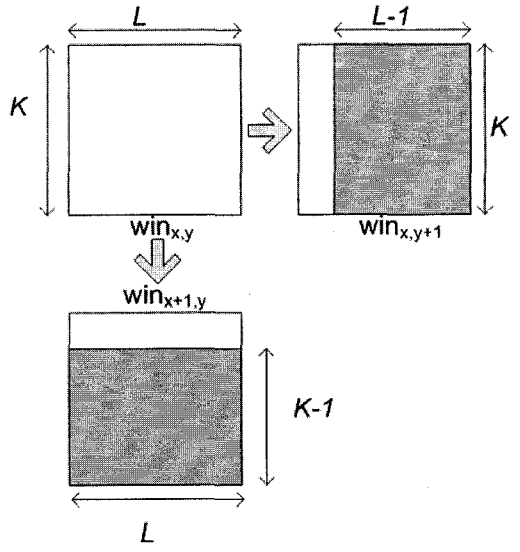


Figure 5.4. Neighboring windows with overlapped pixels.

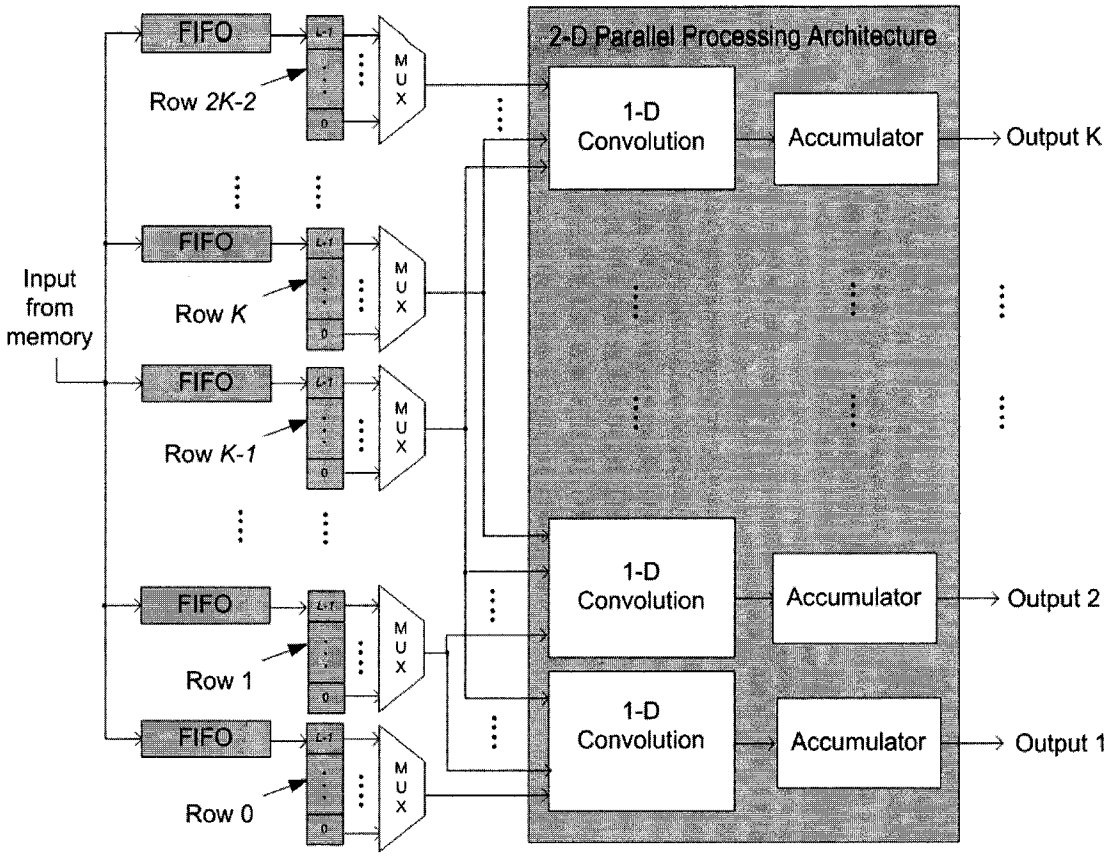


Figure 5.5. Multiple-window partial buffering (MWPB) scheme.

5.3. Symmetry Consideration for Reduced Computations

Some of the most commonly used 2-D convolution operations possess a special characteristic that can be exploited to reduce computations and area resources. This special characteristic is the symmetry property in the computational operations. A kernel with symmetry property has repetitive weight values in the window arranged in a symmetric pattern. The symmetry property of a kernel can be classified as one of three main categories: horizontal symmetry, vertical symmetry or quadrant symmetry. These types of symmetry are shown in Figure 5.6 where same color squares represent same weight values in the kernel. Since the computational operations/procedures for the weights in the kernel are identical, repetitive values in the kernel weights present an

opportunity to reduce computations and area resources by reusing the results or re-arranging the computational procedure to eliminate redundant operations.

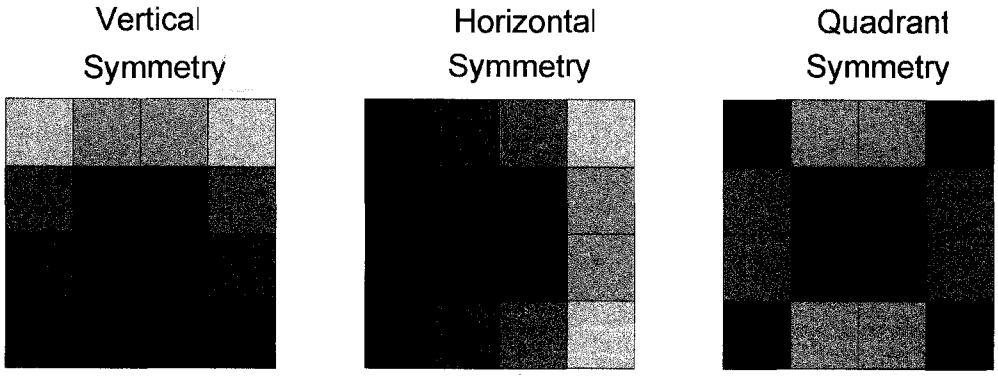


Figure 5.6. Types of symmetry in kernel mask.

An example of an operation with quadrant symmetry property is the smoothing operation by a 2-D convolution with a Gaussian kernel. For example, consider a 4×4 Gaussian kernel in Figure 5.7. It is observed that weights w_{11}, w_{14}, w_{41} and w_{44} have the same value, weights w_{12}, w_{13}, w_{42} and w_{43} have the same value, weights w_{21}, w_{24}, w_{31} and w_{34} have the same value, and weights w_{22}, w_{23}, w_{32} and w_{33} have the same value. Therefore, the weights in the Gaussian kernel can be partitioned into four quadrants as shown in Figure 5.7 and only one quadrant needs to be considered in the computational procedure. This arrangement effectively eliminates 75% of the redundant computations. For example, if we consider the weights in quadrant one (e.g. weights w_{11}, w_{12}, w_{21} and w_{22} in Figure 5.7), we can eliminate three multiplication operations for each weight in quadrant one being processed. This is achieved by reading values of four pixels at the same time and performing one parallel addition operation and then one multiplication operation. One of the on-chip buffering schemes are employed to support parallel reads of four pixel values at a time. In this research work, the SWPB scheme is utilized in the implementation of the 2-D convolution with consideration of the symmetry property in the kernels.

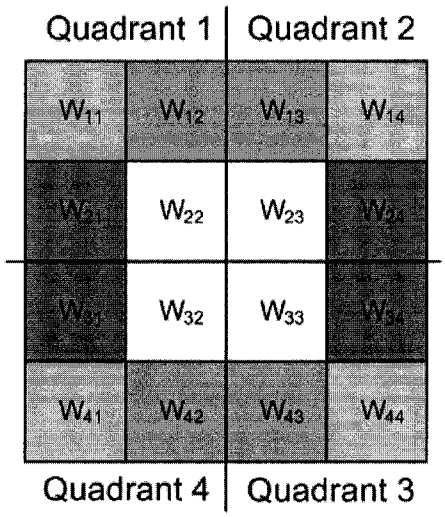


Figure 5.7. Gaussian convolution kernel with symmetry property.

Equations (4.1) and (4.2) define the basic 2-D convolution operation with a $K \times K$ Gaussian kernel where the center pixel of the kernel is overlapped with the center pixel of the image under the window of consideration. Equation (4.1) is used for an even-sized kernel and equation (4.2) is used for an odd-sized kernel.

$$O(x,y) = \sum_{i=0}^{K-1}\sum_{j=0}^{K-1} W(i,j) \cdot I\left(x - i + \frac{K}{2} - 1, y - j + \frac{K}{2} - 1\right) \tag{4.1}$$

$$O(x,y) = \sum_{i=0}^{K-1}\sum_{j=0}^{K-1} W(i,j) \cdot I\left(x - i + \frac{K-1}{2}, y - j + \frac{K-1}{2}\right) \tag{4.2}$$

where O is the output pixel, W is the weight of the kernel, (x, y) is the position of the pixel under consideration, and (K, K) is the size of the kernel. With consideration of the symmetry property in the kernel, summations of the pixels in four quadrants are performed first to reduce three multiplication operations. The general expression for even kernel becomes:

$$O(x,y)=\sum_{i=0}^{\frac{K}{2}-1}\sum_{j=0}^{\frac{K}{2}-1}W(i,j)\cdot\left[I\left(x+i-\frac{K}{2}+1,y+j-\frac{K}{2}+1\right)+I\left(x-i+\frac{K}{2},y+j-\frac{K}{2}+1\right)\right.$$
$$\left.+I\left(x+i-\frac{K}{2}+1,y-j+\frac{K}{2}\right)+I\left(x-i+\frac{K}{2},y-j+\frac{K}{2}\right)\right] \quad (4.3)$$

For an odd kernel, the expression is described in equation (4.4).

$$O(x,y)=\sum_{i=0}^{\frac{K-1}{2}-1}\sum_{j=0}^{\frac{K-1}{2}-1}\left\{W(i,j)\cdot\left[\begin{array}{l}I\left(x+i-\left(\frac{K-1}{2}\right),y+j-\left(\frac{K-1}{2}\right)\right)\\+I\left(x-i+\left(\frac{K-1}{2}\right),y+j-\left(\frac{K-1}{2}\right)\right)\\+I\left(x+i-\left(\frac{K-1}{2}\right),y-j+\left(\frac{K-1}{2}\right)\right)\\+I\left(x-i+\left(\frac{K-1}{2}\right),y-j+\left(\frac{K-1}{2}\right)\right)\end{array}\right]\right\}$$
$$+\sum_{j=0}^{\frac{K-1}{2}-1}\left\{W(x,j)\cdot\left[\begin{array}{l}I\left(x,y+j-\left(\frac{K-1}{2}\right)\right)\\+I\left(x,y-j+\left(\frac{K-1}{2}\right)\right)\end{array}\right]+I\left(x+j-\left(\frac{K-1}{2}\right),y\right)+I\left(x-j+\left(\frac{K-1}{2}\right),y\right)\right\}+W\left(\frac{K-1}{2},\frac{K-1}{2}\right)\cdot I(x,y) \quad (4.4)$$

The block diagrams for the implementations of the symmetry approach are shown in Figures 5.8 and 5.9. In Figure 5.8, an even sized kernel with quadrant symmetry property is considered. In this implementation, pixels at locations in the window that overlap with kernel weights whose values are the same are fed to the parallel adder first and the sum of

this operation is multiplied with the corresponding weight. With this method, a larger multiplier is used in place of the four smaller multipliers in the conventional approach. While the implementation for an even sized kernel is straightforward, implementation for an odd sized kernel requires more resources arranged in a more complicated structure. The block diagram of the parallel architecture for the 2-D convolution with odd size kernel is shown in Figure 5.9. For pixel values that are not in the center row and column, the same structure is used. For the pixels in the center column and row, a different set of computational modules is used. The center pixel of the window requires a dedicated multiplier as shown in Figure 5.9.

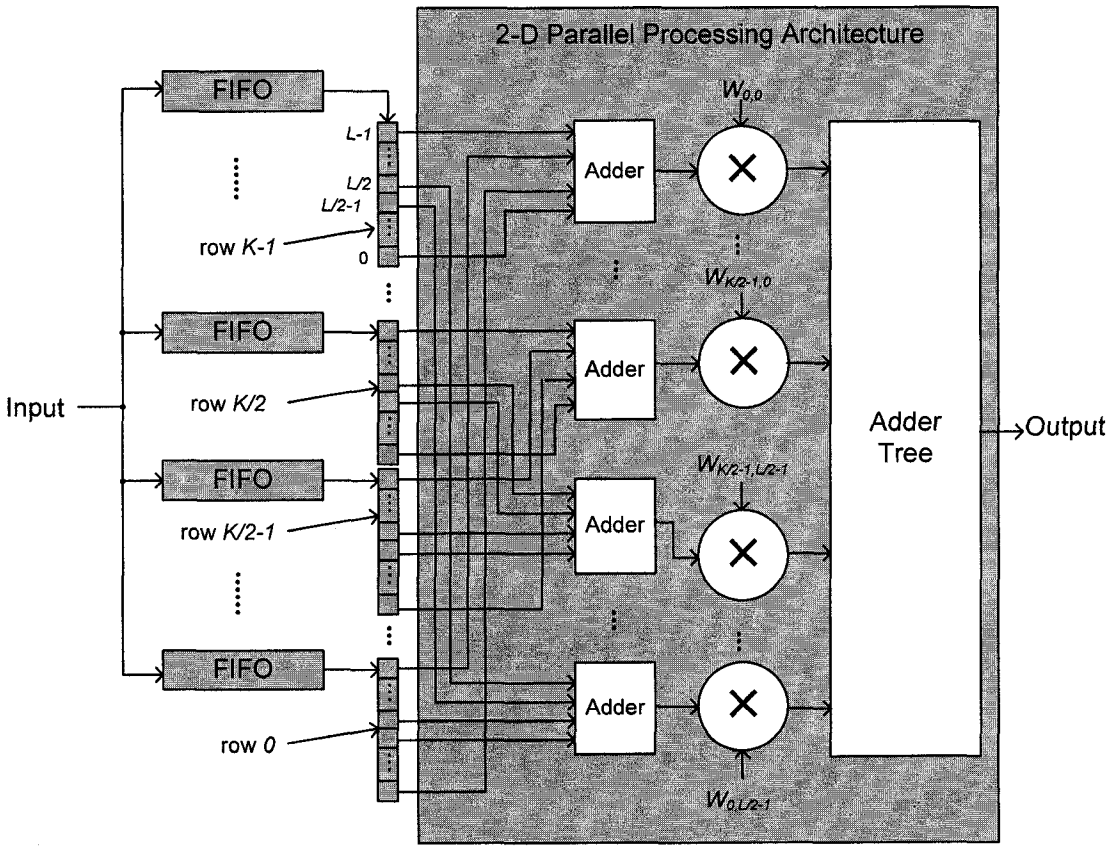


Figure 5.8. Parallel architecture for 2-D convolution with an even symmetry kernel.

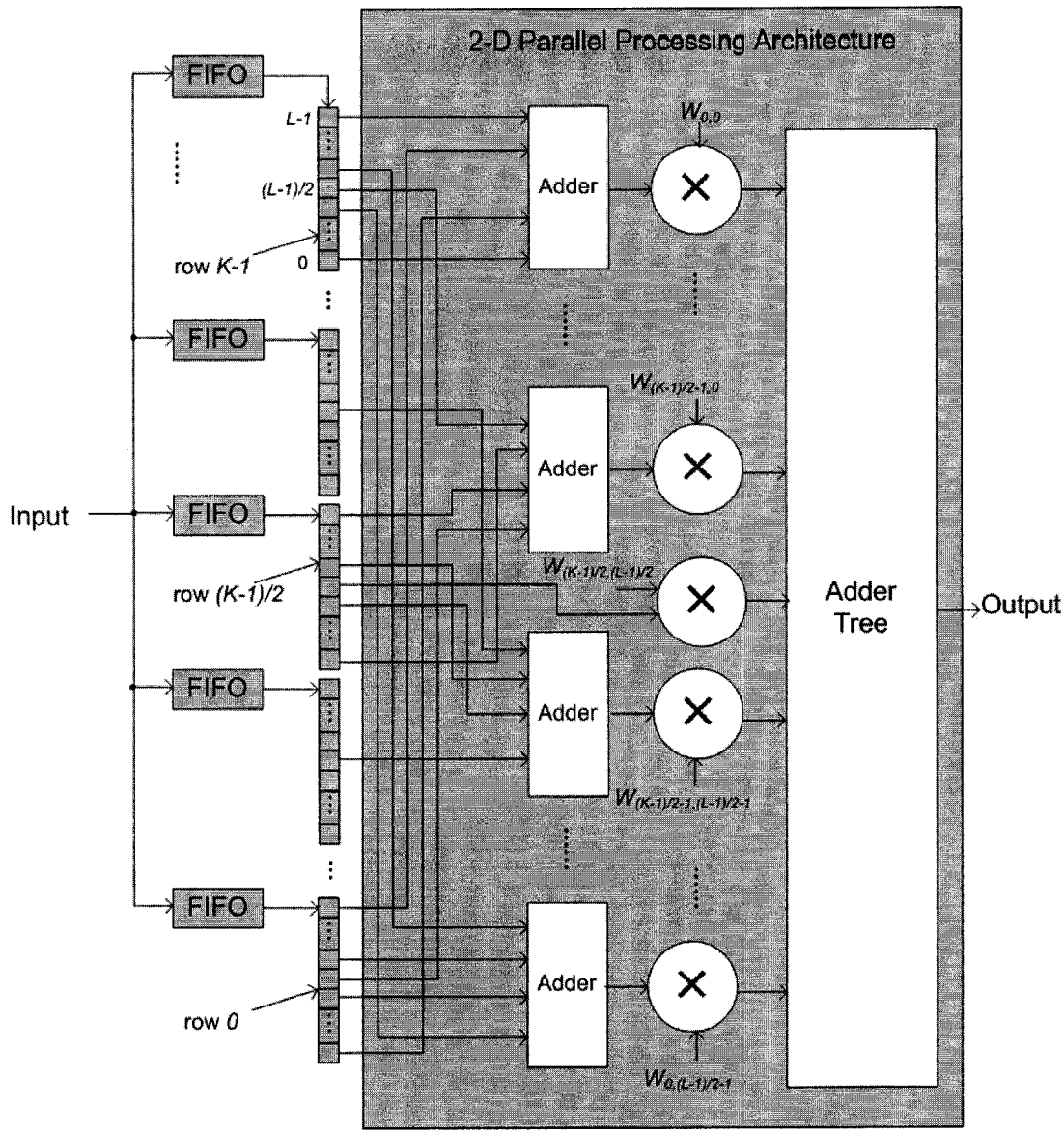


Figure 5.9. Parallel architecture for 2-D convolution with an odd symmetry kernel.

5.4. Neighborhood Dependent Approach (NDA) for Power Reduction in Window-Based Operations

Most of the previous low power design techniques for window-based operations focus on the optimization methods in the processing cores and efficient I/O interfaces with external memory [9, 13-14, 16, 18-24, 43, 55-57, 75]. No existing technique considers the

data characteristic in neighborhood windows to reduce computations and power dissipation. Image and video frame pixels have a very high spatial redundancy such that higher bits in the binary representations of the pixels in the same window are usually the same. The architecture design technique proposed in this chapter exploits the special characteristic of high redundancy in higher bits, the repeated values, and zero values to reduce switching activities. The red, green and blue color band values in a neighborhood (square) of an image are shown in Figure 5.10. As shown in the window of the neighboring pixels in Figure 5.10, many pixels have the same value. The difference in the consecutive pixel values in the same neighborhood is usually small; therefore, only a small number of lower bits will be changed in consecutive operations. This characteristic can be exploited in the design of processing elements for the kernel-based operation to reduce switching activities.

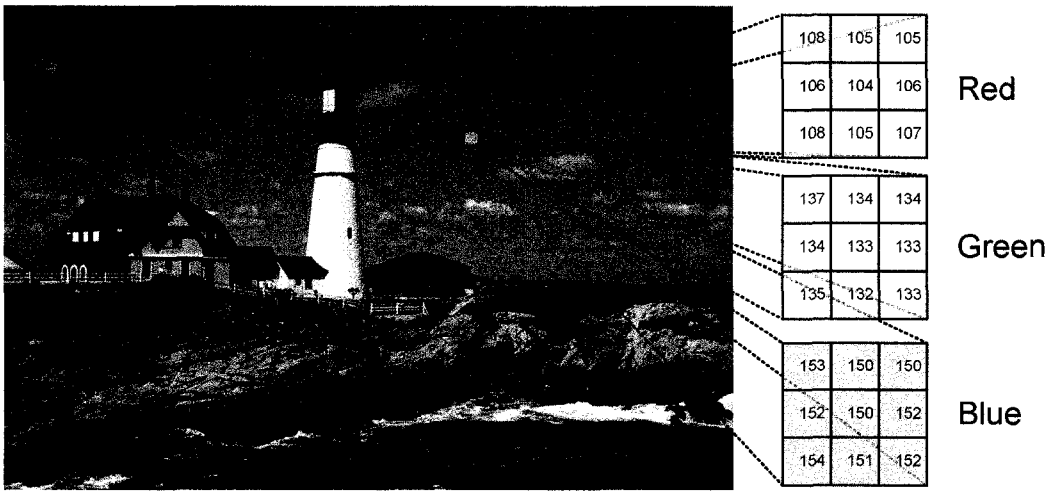


Figure 5.10. A window of neighboring pixels in an image.

To take advantage of the high redundancy in higher bits of the pixel values, a partitioning method similar to the method applied to the multiplier design is employed to partition the pixel buffer to higher and lower parts. For instance, the pixel values in the window-based operation with a $K \times L$ mask can be partitioned into halves as:

$$O = \sum_{i=0}^{K-1}\sum_{j=0}^{L-1} W(i,j) \times I(i,j) \tag{4.5}$$

$$O = \sum_{i=0}^{K-1}\sum_{j=0}^{L-1}\left[\left(W(i,j) \times I_H(i,j) \times 2^{N/2}\right) + \left(W(i,j) \times I_L(i,j)\right)\right] \tag{4.6}$$

where I_H and I_L are the higher and lower halves of pixel values I respectively. Once the pixels are read, partitioned and stored in the appropriate on-chip buffers, a detection scheme is applied to determine if all values in the higher half of the pixels are the same. If this is the case, the 2-D convolution architecture can be disabled and only one multiplication operation is needed to compute the output result. Assume that the pixel values are represented with I_H and I_L, and I_H is an m-bit binary number. For a $K \times L$ mask, the detection scheme proposed in this chapter can be summarized as:

//To determine if all values in a row are the same

for $i = 0,1,....,K-1$

for $j = 0,1,....,L-2$

$$bv_j = \left(I_{Hi,j(0)} \oplus I_{Hi,j+1(0)}\right)' \cdot \left(I_{Hi,j(1)} \oplus I_{Hi,j+1(1)}\right)' \cdot .... \cdot \left(I_{Hi,j(m)} \oplus I_{Hi,j+1(m)}\right)'$$

end

$$br_i = bv_0 \cdot bv_1 \cdot .... \cdot bv_{L-2}$$

end

//To determine if all values in first column are the same

for $i = 0,1,....,K-2$

$$bu_i = \left(I_{Hi,0(0)} \oplus I_{Hi+1,0(0)}\right)' \cdot \left(I_{Hi,0(1)} \oplus I_{Hi+1,0(1)}\right)' \cdot .... \cdot \left(I_{Hi,0(1)} \oplus I_{Hi+1,0(1)}\right)'$$

end

$$bc = bu_0 \cdot bu_1 \cdot .... \cdot bu_{K-2}$$

//To determine if all values in a window are the same

$$bw = br_0 \cdot br_1 \cdot .... \cdot br_{K-1} \cdot bc$$

Each bv_j bit is determined by comparing two consecutive numbers in the same row. This computation is achieved by bit-wise XNOR operations and a parallel AND operation of the XNOR results. So, for each number in the same row, an extra bit is maintained for $bv_j s$. A logic '1' indicates that the j pixel has the same value as the $(j+1)$ pixel in row i. Bit br_i indicates if all numbers in row i has the same value. This bit is generated by a bit-wise AND operation of all $bv_j s$ in the same row. In addition, a comparison of all pixel values in the first column of the window is also performed. This operation is achieved by comparing the first value in a row i with the first value in row $(i+1)$ and the result bu_i is set. Then, $(K-1)$ $bu_i s$ results of the comparisons are ANDed to determine value bc. A logic '1' for bc means that all values in the first row are the same. Similarly, bit bw indicates that the current window has the same value for all elements. Bit bw is computed by a bit-wise AND operation of all $br_i s$ and bc in the window. Bit bw is set to '1' if all numbers in the window have the same value. If this condition is detected the parallel architecture for 2-D convolution is disabled and a multiplication is carried out with the pixel value $I_{H0,0}$ and the sum of the kernel mask which is pre-computed and stored in a register.

This neighborhood dependent approach (NDA) can easily be incorporated with different buffering schemes such as FWB, SWPB and MWPB. Since full window buffering approach requires large on-chip storage for multiple pixel delay lines, the switching activities increase significantly as the pixels propagate through these delay lines without any actual computations. In this research work, the partial buffering schemes are considered for storing the window on-chip to support parallel processing. In the SWPB scheme, values in one window is maintained in the shift registers. As the pixels are fed to the shift registers or each row in the window, the incoming pixels are compared with the pixel values in the last position $(L-1)$ in a row. The result of this comparison is stored with an additional bit at pixel location $(L-2)$ as the new pixel is shifted in. The circuit for an m-bit comparator of two values a and b is shown in Figure 5.11.

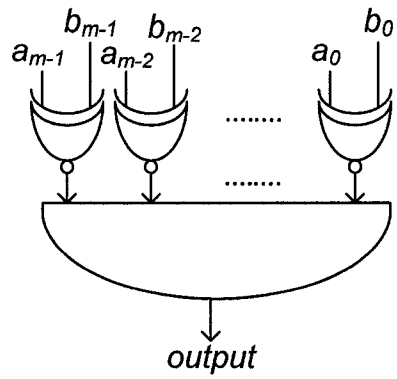


Figure 5.11. Circuit of an *m*-bit comparator.

The block diagram for the parallel 2-D convolution architecture with consideration of the proposed NDA technique and SWPB scheme is shown in Figure 5.12. For each row of *L* values, additional (*L-1*) bits are needed for comparison. An (*L-1*)-bit AND gate is used for each row to compute the bc_j which indicates if pixel values in row *j* are the same. For the first column comparisons, (*K-1*) *m*-bit comparators are needed for generating (*K-1*) $bu_i s$. A (*K-1*)-bit AND gate is used to compute bit *bc*. A *K*-bit AND gate is then used to determine if the entire window consists of the same pixel value. This is achieved by ANDing all (*K-1*) br_i values and *bc*. If this condition is detected (indicated by *bw* which is equal to '1'), then all the (*K-1*) 1-D convolution modules are disabled and the result is computed by a multiplication of the first pixel in the widow with a pre-stored sum value of the kernel weight. An additional 'Delay' module is added to synchronize output data with the pipelined 1-D convolution and adder tree as shown in Figure 5.12. If the condition is not detected, normal operations are performed in the parallel architecture and the additional pipelined multiplier and delay module are disabled. Since all processing modules are designed as pipelined structure, enabling and disabling the processing component are achieved by de-activating the latching signals for the registers in the first pipelined stage.

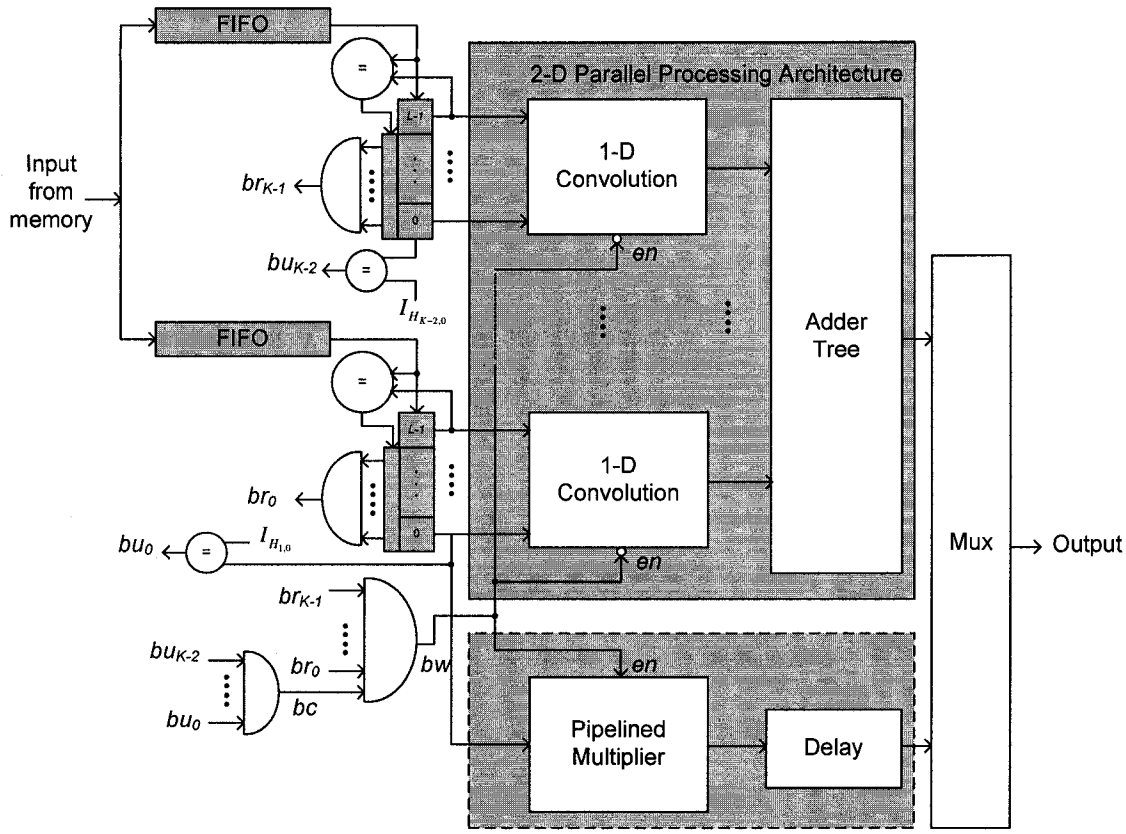


Figure 5.12. NDA is incorporated with SWPB scheme.

The NDA approach is applied to the multiple-window buffering scheme (MWPB) in a similar way as in the SWPB scheme. Pixel values within a row are compared as pixels are fed from the FIFO queue to the shift registers. For each row, (*L-1*) additional bits for comparison results are stored within the (*L-1*) shift registers for the detection scheme. These (*L-1*) bits are ANDed to indicate if values in each row are the same. Since this approach evaluates multiple windows at the same time, it requires multiple detection modules. For processing *K* windows simultaneously, *K* detection units and *K* additional multipliers are needed as shown in Figure 5.13. Each processing path for a window requires a dedicated control line (*en*) as shown in Figure 5.13 because data in each row is processed serially. Once the window is determined to have the same value, the entire processing path for that window is disabled for ($K \times L$) cycles and the pipelined multiplier

is disabled for $(K \times L\text{-}1)$ cycles. This ensures that K results are available at the same time without unnecessary computations.

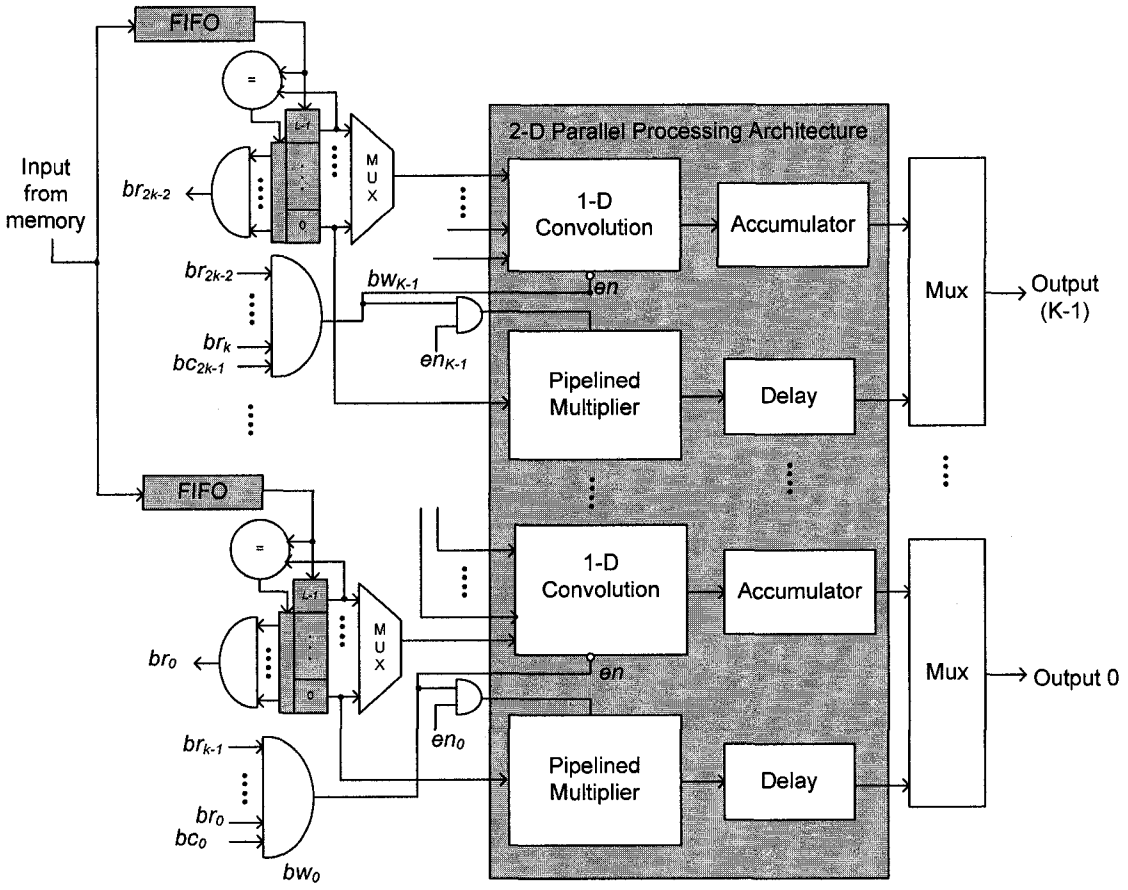


Figure 5.13. NDA is incorporated with MWPB scheme.

The proposed NDA technique can also be applied to the parallel architecture that incorporates the symmetry method to further reduce the computations in the 2-D convolution with symmetry kernel masks. With symmetry approach, either an SWPB or MWPB scheme is suitable. Figure 5.14 shows the block diagram of the parallel architecture for 2-D convolution with consideration of symmetry property and neighborhood dependent approach. The SWPB buffering scheme and even kernel size are used in this implementation. The operating strategy for detection of special conditions

remains the same as in the conventional SWPB approach. When the system is in normal operating mode, the parallel processing architecture is active as in the conventional approach. When the special condition for NDA method is detected the entire parallel architecture is disabled and a single multiplication is performed with the pipelined multiplier.

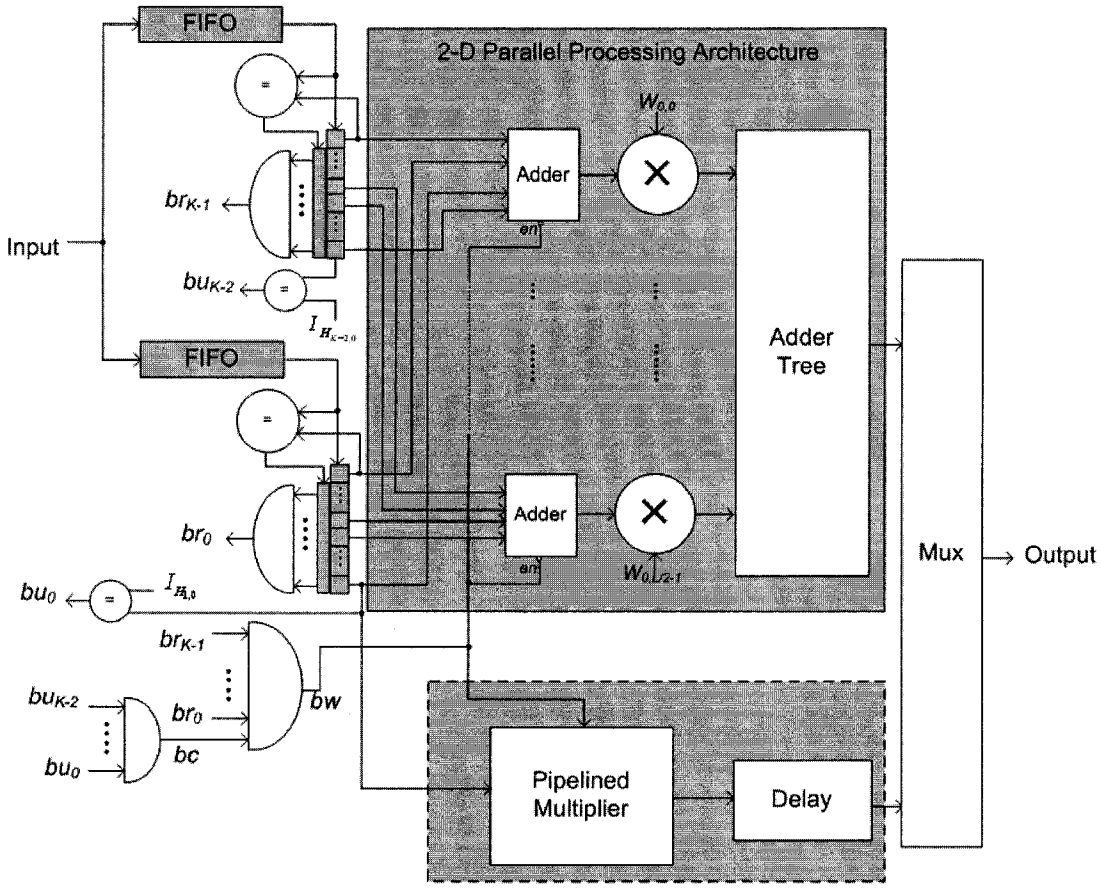


Figure 5.14. NDA is incorporated with SWPB scheme and quadrant symmetry property.

5.5. Experimental Results

The proposed NDA method for window-based operation is applied to the parallel architecture for the 2-D convolution operation with different filter kernels. The evaluation is performed for two on-chip buffering schemes: (1) SWPB and (2) MWPB. The parallel architecture is implemented with a VHDL hardware description language. The designs

are synthesized with Altera's Quartus II software environment for Cyclone II FPGA. Table 5.1 gives a summary of resource utilization, maximum speed achieved for the implementation and latency for six designs: (1) parallel architecture with SWPB scheme (SWPB), (2) parallel architecture with SWPB scheme and the proposed NDA approach (SWPB_NDA), (3) parallel architecture with MWPB scheme (MWPB), (4) parallel architecture with MWPB scheme and the proposed NDA approach (MWPB_NDA), (5) parallel architecture with SWPB scheme and quadrant symmetry property (QUAD) and (6) parallel architecture with SWPB scheme and quadrant symmetry property with the proposed NDA approach (QUAD_NDA). All implementations use 3×3 filtering kernel. A smoothing filter with a Gaussian kernel is used as the application for the simulations. Pixel values are extracted from the test images and fed to the FIFO queues of the parallel architectures with different buffering schemes. The simulation tool of Altera Quartus II software is used and logic transitions are recorded to the Signal Activity File (SAF). The SAF file is used in the PowerPlay tool to compute the dynamic power consumption of the designs. The power consumptions for various design schemes are shown in Table 4.2.

Table 5.1. Implementation results for parallel architecture with various buffering schemes for 2-D convolution with a 3×3 filtering kernel.

Scheme	**Logic elements**	**F_{max} (MHz)**
SWPB	882	198
SWPB_NDA	1026	183
MWPB	956	195
MWPB_NDA	1215	181
QUAD	428	194
QUAD_NDA	524	178

Table 5.2. Simulation results of a Gaussian smoothing filter.

Test Image / Method	**Peppers** (128×128)		**Island** (160×106)	
	Power (mW)	**Normalized ratio**	**Power** (mW)	**Normalized ratio**
SWPB	69.72	1.00	63.22	1.00
SWPB_NDA	**34.59**	**0.50**	**31.35**	**0.50**
MWPB	63.10	1.00	54.55	1.00
MWPB_NDA	**33.17**	**0.53**	**28.45**	**0.52**
QUAD	45.82	1.00	40.17	1.00
QUAD_NDA	**29.12**	**0.64**	**25.26**	**0.63**

As shown in Table 5.2, the proposed NDA method incorporated in the architecture designs with various buffering schemes achieved significant power reduction. For SWPB scheme, the proposed design achieves 50% power reduction when the NDA method is utilized. For the MWPB scheme, more than 47% power reduction is achieved when the NDA method is incorporated. In the implementations of parallel architecture for 2-D convolution with consideration of the symmetry property, the proposed design with the NDA method achieved more than a 36% power reduction for both the test images.

5.6. Summary

In this chapter, a neighborhood dependent approach to reduce power consumption in 2-D convolution in image and video processing applications has been proposed and evaluated.

Pixel values in the same window and neighboring windows have high correlation in higher bits that is utilized to reduce switching activities in the proposed NDA method. Pixel values are partitioned and stored in the higher half and lower half buffers in the proposed NDA method. Various on-chip buffering schemes to support parallel processing in 2-D convolution were used in the implementations. The proposed NDA method was incorporated with SWPB, MWPB buffering schemes for power reduction. The NDA method was also incorporated along with the quadrant symmetry property in 2-D convolution design. Experimental results show that the proposed NDA method can be used to support high performance of the parallel architecture, and the designs achieved more than 36% power reduction in all schemes.

Chapter VI

MULTI LEVEL POWER-AWARE DESIGN TECHNIQUES FOR REAL-TIME VIDEO ENHANCEMENT

In this chapter, a nonlinear image enhancement algorithm developed in ODU Vision Lab is used as an application for which the proposed multi-level power-aware design techniques can be applied to reduce power consumption. The image enhancement algorithm used for this purpose is the Illuminance-Reflectance Model for Enhancement (IRME) [92-93]. The IRME algorithm deals with the issue of mapping high dynamic range images to low dynamic range renditions, which is similar to human eyes observing a scene. The algorithm is based on illuminance perception and processing to achieve dynamic range compression while retaining or even enhancing visually important features. A parallel and pipelined architecture design for real time processing of this image enhancement algorithm is presented. Power-aware techniques utilized for the reduction of power dissipation include the design of the proposed Booth multiplier, log-based divider, log-based powering unit, and NDA approach for 2-D convolution with a Gaussian kernel.

6.1. Image Enhancement Algorithm

The IRME algorithm is composed of three major steps: (1) illuminance estimation and reflectance extraction, (2) dynamic range compression and contrast enhancement on illuminance, and (3) image restoration step which uses illuminance and reflectance to recover the intensity image, color recovery and white balance adjustment for color images.

6.1.1. Illuminance Estimation and Reflectance Extraction

To accurately estimate the illuminance of a scene from an image is a difficult task. In this algorithm, the low-pass result of the intensity image through a Gaussian filter is used as the illuminance estimation. In the spatial domain, this filtering process is a 2-D discrete

convolution with a Gaussian kernel (of size $K \times K$), which can be mathematically expressed as:

$$L(x,y) = \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} I(x-i, y-j) W(i,j) \tag{6.1}$$

where L is the illuminance, I is the intensity image (of size $M \times N$) and W is the 2-D Gaussian function. W is defined as:

$$W(x,y) = P.e^{\left(\frac{-(x^2+y^2)}{c^2}\right)} \tag{6.2}$$

where P is determined by

$$\iint P \cdot e^{\left(\frac{-(x^2+y^2)}{c^2}\right)} dxdy = 1 \tag{6.3}$$

and c is the scale (Gaussian surround space constant), which determines the size of the neighborhood. In this algorithm, c = 1~4 is commonly used. For color images, the intensity images are obtained as:

$$I(x,y) = \max\left[r(x,y), g(x,y), b(x,y)\right] \tag{6.4}$$

where $r(x,y)$, $g(x,y)$ and $b(x,y)$ are the red, green and blue components of color images in the RGB color space. This method is the definition of the value component in the HSV color space. Once the illuminance $L(x,y)$ is found, the reflectance $R_f(x,y)$ is computed using the relationship:

$$I(x,y) = L(x,y) R_f(x,y) \tag{6.5}$$

It is observed from those images that the illuminance is comprised of the mid and low-frequency information of the image, which is an approximated illuminance that contains both the illuminance and the low-frequency part of reflectance. However, the discontinuity of illumination is maintained leading to the elimination of halo artifacts. The visually important features (high frequency reflectance) and a small part of the illuminance information are contained in the reflectance in which the major illumination effect is removed. Therefore, important image features will be kept after the dynamic range compression of illuminance. Based on these observations, reflectance and illuminance are also regarded as details.

6.1.2. Dynamic Range Compression of Illuminance and Contrast Enhancement

Dynamic range compression of illuminance is realized in the algorithm by gamma compression that is described as:

$$L'(x,y) = \left(\frac{L(x,y)}{255}\right)^{\gamma} \cdot 255 \tag{6.6}$$

where $L'(x,y)$ is the illuminance after dynamic range compression. In equation (6.6), γ is generally in the range of 0.3 to 0.7. It has been noted that the illuminance also contains the low frequency information of reflectance which can be degraded during dynamic range compression. For the original image with a low contrast or slow-varying reflectance map, the degradation of mid-frequency features may be obvious in the output images. Therefore, a center-surround type of contrast enhancement method is utilized to compensate this degradation. First, a low-pass result of $L'(x, y)$, denoted as $L''(x, y)$, with a larger scale c (lower cut-off frequency) is computed through the same operations as in equations (6.1) to (6.3). Then, a local contrast enhancement is performed on $L'(x,y)$ based on:

$$L'_E(x,y) = \begin{cases} L'(x,y) + \beta_1 \left(L'(x,y) - L''(x,y)\right) & \text{for} \quad L'(x,y) > L''(x,y) \\ L'(x,y) + \beta_2 \left(L'(x,y) - L''(x,y)\right) & \text{otherwise} \end{cases} \tag{6.7}$$

where $L'_E(x,y)$ is the illuminance after contrast enhancement, β_1 and β_2 are the adjustment factors which determine the strength of the contrast enhancement. Typical value for β_1 and β_2 are 0.3 and 0.6 respectively.

6.1.3. Image Restoration and Adjustment

After the dynamic range compression of illuminance procedure has been completed, illuminance $L'_E(x,y)$ and reflectance $R_f(x,y)$ are combined using equation (6.8) to produce an intensity image $I'(x,y)$ with compressed dynamic range as:

$$I'(x,y) = L'_E(x,y) \cdot R_f(x,y) \tag{6.8}$$

For color images, a color restoration process based on the chromatic information of the original image is applied on the output intensity image to recover the red, green and blue color bands as:

$$R(x,y) = \frac{I'(x,y)}{I(x,y)} r(x,y) \tag{6.9}$$

$$G(x,y) = \frac{I'(x,y)}{I(x,y)} g(x,y) \tag{6.10}$$

$$B(x,y) = \frac{I'(x,y)}{I(x,y)} b(x,y) \tag{6.11}$$

The effect of this step is that the ratio among color components in the original image is preserved in the enhanced image. The output color images produced by the color restoration procedure may have an unnatural appearance with their color hue and saturation. However, their colors can be adjusted through a color saturation and white balance adjustment operation, which is explained as:

$$R'(x,y) = R(x,y) + H_r\left(A(x,y) - R(x,y)\right) \tag{6.12}$$

$$G'(x,y) = G(x,y) + H_g\left(A(x,y) - G(x,y)\right) \tag{6.13}$$

$$B'(x,y) = B(x,y) + H_b\left(A(x,y) - B(x,y)\right) \tag{6.14}$$

where $A(x,y) = 0.33 \times [R(x,y) + G(x,y) + B(x,y)]$. Coefficients H_r, H_g and H_b are used for color saturation adjustment and white balance correction of each color band. According to equations (6.12) to (6.14), color saturation is realized by comparing the intensity of each band and the average color value of the pixel and compensating the value of that intensity in that band if it is less than the average; otherwise, the value of the intensity in that band will be suppressed.

6.2. Architecture Design for the Image Enhancement Algorithm

As in many other image processing applications, the real-time image enhancement application presented in this chapter provides a high level of data parallelism where computations involving each pixel in the image are identical. It is essential to take advantage of the parallelism in the application to achieve maximum throughput. The architecture for IRME image enhancement algorithm is designed to exploit the inherent parallelism in the technique by partitioning computations to various modules in pipelined stages. Pixel values are transformed into the logarithmic domain by utilizing the proposed architectural design of the approximation technique to avoid complex and computationally intensive operations such as division, logarithm and powering. The enhancement architecture consists of three main computational modules: (1) illuminance enhancement module, (2) contrast enhancement module, and (3) color restoration/adjustment module. To achieve efficient computations for the enhancement algorithm, some of the equations in the previous section are modified into a hardware-efficient algorithm to better suit the architecture design. The implementation of the image enhancement algorithm utilizes parallel and pipelined architecture design techniques for real-time processing.

6.2.1. Illuminance Enhancement Module

This module is designed to perform the dynamic range compression of the illuminance operation described in equation (6.6). The fast and efficient approximation method for computing the $\log_2$ operation is utilized to approximate the powering operation. The procedure for the dynamic range compression of the illuminance step is transformed to operate in the logarithmic domain where the powering operation in equation (6.6) is achieved by a multiplication operation. The actual operation for compressing the dynamic range of the illuminance is defined as:

$$L'(x,y) = \log_2^{-1}\left[\gamma \cdot \log_2\left(L_N(x,y)\right)\right] \cdot 255 \tag{6.15}$$

where $L_n(x,y)$ is the normalized illuminance which is computed by $L_n(x,y) = \frac{L(x,y)}{255}$. Division by 255 and multiplication by 255 are approximated very closely by fast arithmetic shift operations. Intensity $I(x,y)$ of the pixel is first passed through the 2-D convolution unit which is implemented using a parallel architecture design with consideration of quadrant symmetry to obtain illuminance $L(x,y)$. The SWPB buffering scheme is utilized in this work. The block diagram of the illuminance enhancement module is shown in Figure 6.1. The parameter γ is latched into the system at initialization. Equation (6.15) is carried out by a $\log_2$ operation, a multiplication, an inverse-$\log_2$ operation, and two arithmetic shift operations. Reflectance $R_f(x,y)$ is also computed in this module by performing operations defined as:

$$R_f(x,y) = \frac{I(x,y)}{L(x,y)} \tag{6.16}$$

The division operation is avoided for higher performance and reduced resource consumption by utilizing the approximation technique to transform a division operation to a subtraction operation in the logarithmic domain as shown in equation (6.17):

$$R_f(x,y) = \log_2^{-1}\left[\log_2\left(I(x,y)\right) - \log_2\left(L(x,y)\right)\right] \tag{6.17}$$

Implementation of the log-based divider using the approximation method significantly reduces the resource utilization compared to the fully pipelined divider generated by a core generator. A typical pipelined divider obtained from the commercial core generator consumes more than 80% of the resources compared to the proposed log-based divider used in this implementation.

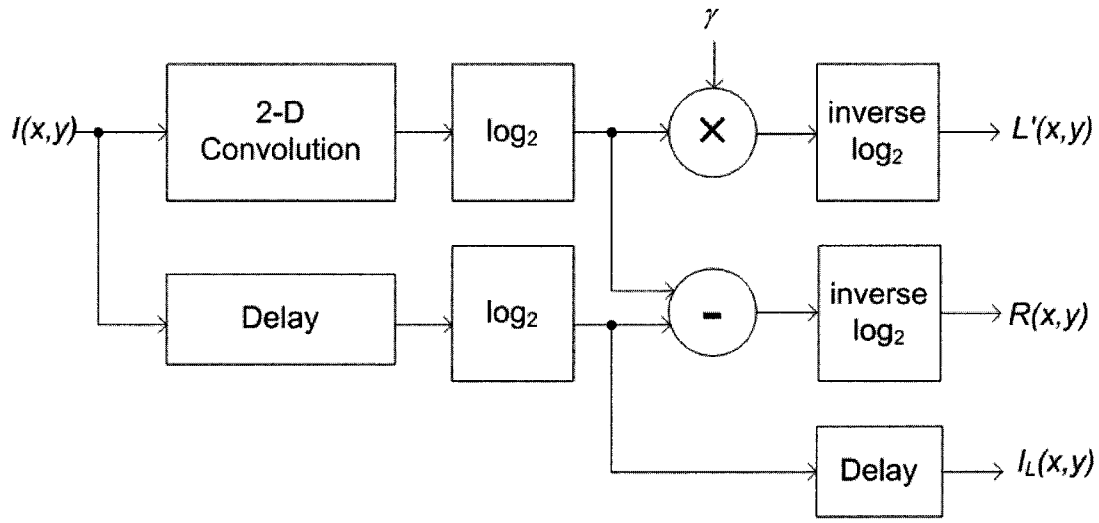


Figure 6.1. Block diagram of the illuminance enhancement module.

The intensity of a pixel in the logarithmic domain is passed along the illuminance enhancement module for future operations in subsequent modules shown in Figure 6.1. Delay elements are inserted in the pipelined stages as shown in Figure 6.1 to synchronize data paths for enhanced illuminance $L'(x,y)$, pixel reflectance value $R_f(x,y)$ and the intensity in the logarithmic domain $I_L(x,y)$ at the output of the module,. Delay elements are configured as FIFO queues with embedded RAM blocks in an FPGA. Parallel design with quadrant symmetry property presented in Chapter 4 is used for the implementation of the 2-D convolution unit.

6.2.2. Contrast Enhancement Module

The contrast enhancement module considers surrounding pixels to adjust the enhanced illuminance obtained from the illuminance enhancement module. First, the enhanced illuminance $L'(x,y)$ is passed through a convolution module to obtain the low-pass result $L''(x,y)$ which contains the local information necessary to increase or decrease the enhanced illuminance as described in equation (6.7). Similar to the parameter γ in the illuminance enhancement module, contrast adjustment factors β_1 and β_2 which determine the strength of contrast enhancement are pre-stored in registers upon system initialization. The architecture design for the contrast enhancement module is very straightforward and the block diagram for this module is shown in Figure 6.2. For this contrast enhancement module, a larger Gaussian kernel is used to consider the information from surrounding pixels.

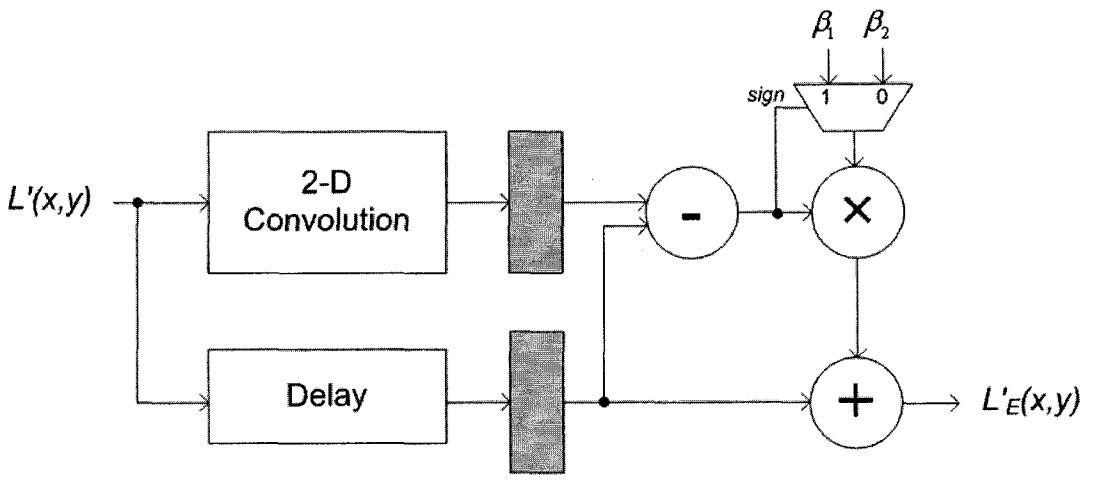


Figure 6.2. Block diagram of the contrast enhancement module.

6.2.3. Color Restoration and Adjustment Module

Enhanced illuminance $L'_E(x,y)$ obtained after performing the illuminance and contrast enhancement is used with the original reflectance value $R_f(x,y)$ to compute the intensity

of the enhanced pixel. This is achieved by a multiplication operation. The color restoration procedure is carried out in this module based on equations (6.9)-(6.11) to obtain three color bands (red, green and blue) for the enhanced pixels. Data in equations (6.9)-(6.11) are transformed to the logarithmic domain to reduce the computational complexity of the division operation. Equations (6.9) to (6.11) are modified as:

$$R(x,y) = \log_2^{-1}\left[\log_2\left(I'(x,y)\right) - \log_2\left(I(x,y)\right)\right] r \tag{6.17}$$

$$G(x,y) = \log_2^{-1}\left[\log_2\left(I'(x,y)\right) - \log_2\left(I(x,y)\right)\right] g \tag{6.18}$$

$$B(x,y) = \log_2^{-1}\left[\log_2\left(I'(x,y)\right) - \log_2\left(I(x,y)\right)\right] b \tag{6.19}$$

Computation of color restoration and the white balance adjustment step are also included in this module. The block diagram of the color restoration and adjustment module is shown in Figure 6.3. Color justification coefficients H_r, H_g and H_b for red, green and blue color bands are latched into registers in this module at the system initialization stage. Enhanced color components R', G' and B' at the outputs of this module are written to an output RAM module for interfacing with the video encoder.

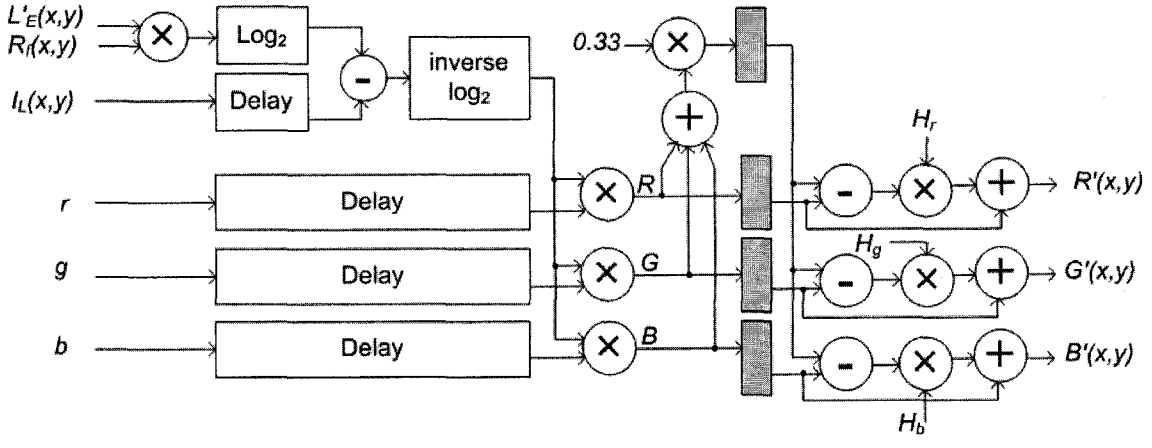


Figure 6.3. Block diagram of the color restoration and adjustment module.

6.3. Experimental Results

The video enhancement system design is implemented with VHDL utilizing Altera's Quartus II software tool. The design is fitted in a Cyclone II FPGA in the Altera's DE2 developmental and educational board. Table 6.1 gives a summary of resource utilization and maximum speed achieved from the implementation. Two SDRAM banks are used as input and output frame buffers.

Table 6.1. Implementation results for the proposed architecture designs of IRME image enhancement algorithm.

Description	**Implementation without power-aware techniques**	**Implementation with power-aware techniques**
Device	Cyclone II 2C35 FPGA	Cyclone II 2C35 FPGA
Logic elements	5317 (~ 17%)	6256 (~ 19%)
SDRAMs	3 (75%)	3 (75%)
Max. clock freq.	65 MHz	62 MHz

A set of test images captured under various lighting environments is used to evaluate the proposed design. Two typical test images shown in Figure 6.4a are used in this chapter to compare the enhanced results of the hardware system and the software program. The first test image is captured in an extremely dark environment and the second test image is captured in a complex lighting condition. Figure 6.4b shows the enhanced images obtained by the simulation using software program written in Matlab that uses double-precision floating point number format. Images in Figure 6.4c show the enhanced results obtained by the hardware architecture that uses an 18-bit fixed point number format. The

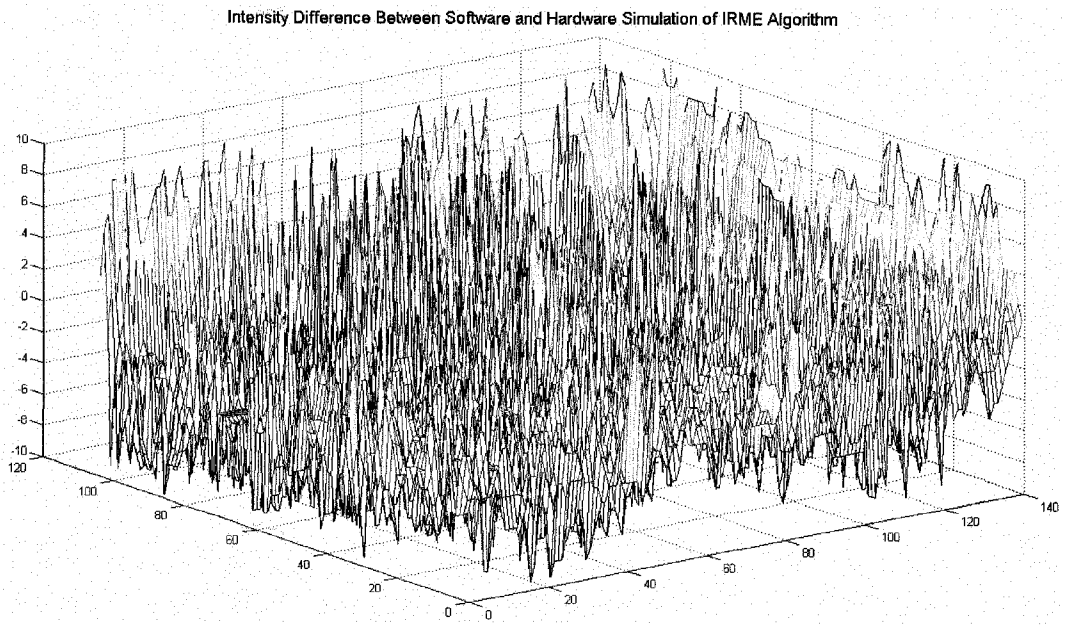
enhanced images obtained from the hardware system have minor degradation due to the approximation procedure and the fixed point arithmetic. The average error for the first test image (uniform and extreme dark image) is approximately 0.1078 in terms of pixel intensity. Similarly, the average error for the second test image (captured in complex and nonlinear lighting condition) is about 0.1278. The intensity difference between Matlab and FPGA implementations for the first test (uniform dark) image is shown in Figure 6.5a and the histogram of the differences is shown Figure 6.5b. Similarly, Figure 6.6 shows the intensity difference and histogram of the differences for the second test image (captured in nonlinear lighting environment). Fixed point representation understandably shows limitation in dynamic range.

(a) Original images.

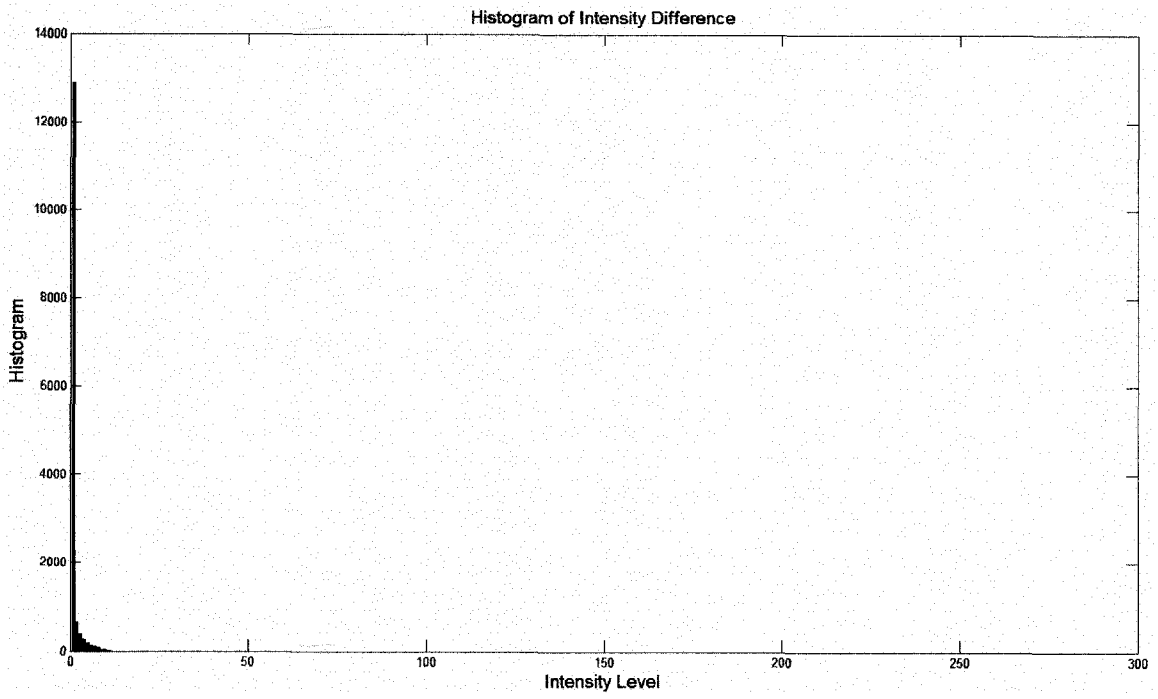
(b) Enhanced images by software mean.

(c) Enhanced image by hardware architecture.

Figure 6.4. Comparison between enhanced images obtained by a software program in Matlab and enhanced images obtained by the hardware architecture.

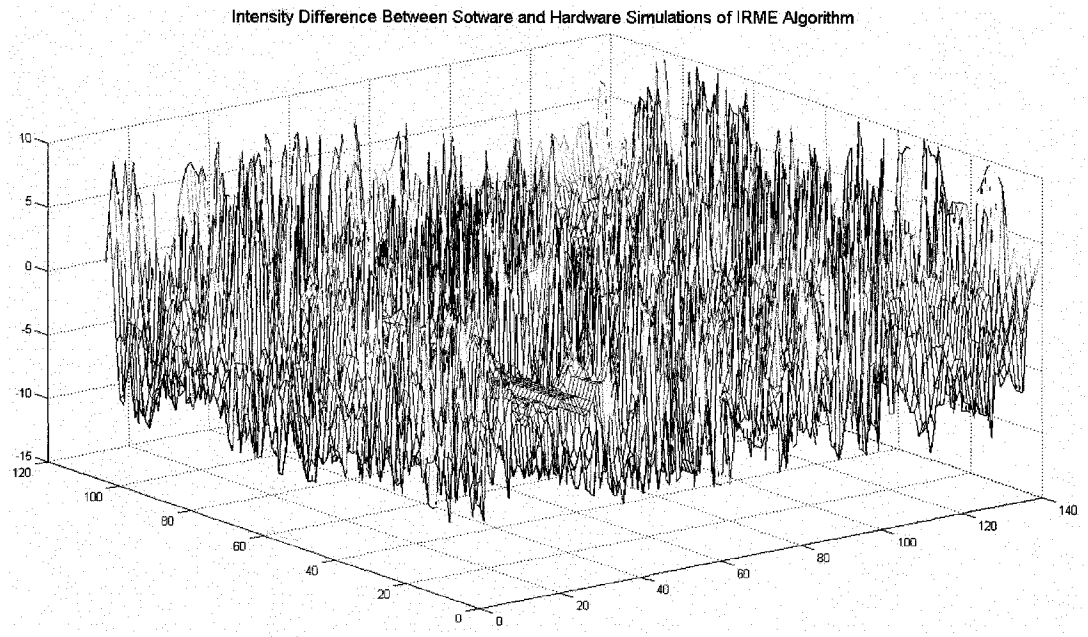


(a) Intensity differences between resulting images.

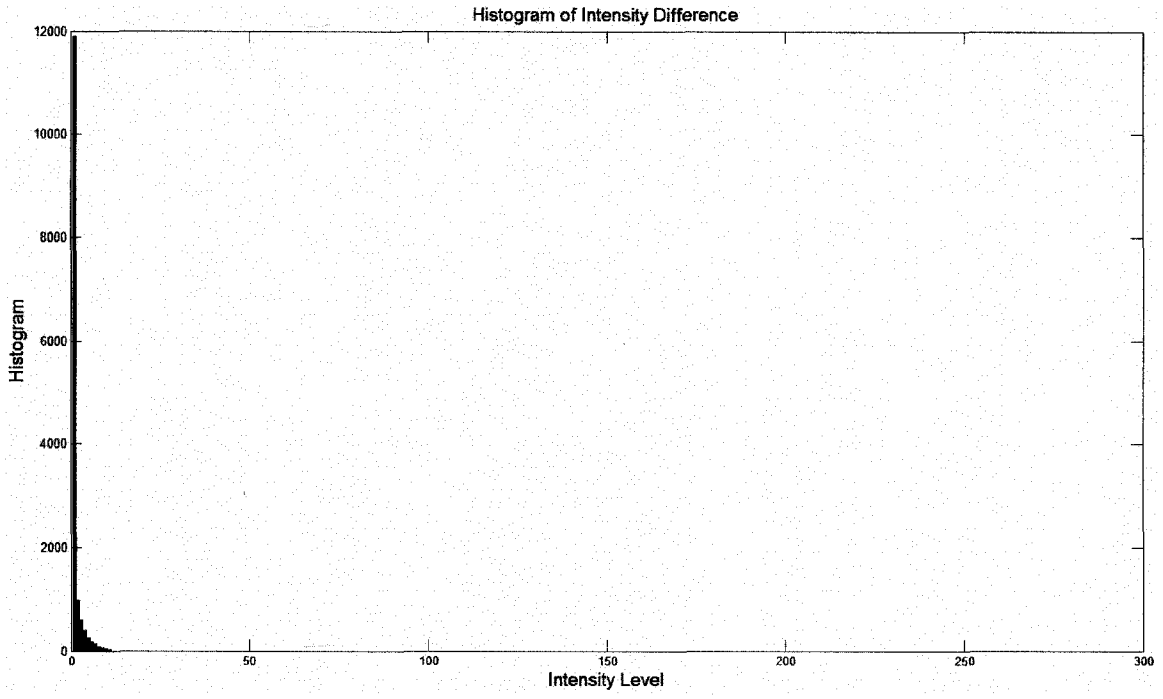


(b) Histogram of intensity differences.

Figure 6.5. Error analysis between the software and hardware implementations for the first test image.



(a) Intensity differences between resulting images.



(b) Histogram of intensity differences.

Figure 6.6. Error analysis between the software and hardware implementations for the second test image.

For performance evaluation, three implementations for real-time video enhancement of the IRME algorithm using DSP processors are estimated. These implementations use Texas Instrument (TI) DSP processors that are based on the advanced Very Long Instruction Word (VLIW) architecture which has eight independent functional units to support parallel processing. In addition, a DSP library with optimized DSP functions such as FFT and IFFT is used to support real-time applications. Table 6.2 lists the configurations and performance of the proposed system for IRME algorithm and other DSP-based implementations of IRME algorithm. As shown in Table 6.2, even though the proposed architecture operates with a much lower clock frequency, it outperforms other implementations by as much as 26 folds. The main reason for the difference in performance is that the proposed system utilizes maximum parallel operations inherent in the enhancement algorithm while DSP processors have a fixed number of functional units which limit parallel processing to a certain bound.

Table 6.2. Performance comparison of the proposed FPGA based enhancement system with three DSP-based enhancement implementations (for 256×256 frames).

Device	TMS3026711 (DSP-based)	TMS3026713 (DSP-based)	TMS302DM642 (DSP-based)	Cyclone II (FPGA-based)
$\mathbf{F_{max}}$	150 MHz	225 MHz	600 MHz	60 MHz
External memory	100 MHz SDRAM	90 MHz SDRAM	133 MHz SDRAM	133 MHz SDRAM
Architecture design	VLIW (8 func. units)	VLIW (8 func. units)	VLIW (8 func. units)	Parallel-pipeline
Number system	32-bit floating point	32-bit floating point	32-bit fixed point	18-bit fixed point
Frame rate	39 fps	59 fps	157 fps	1022 fps

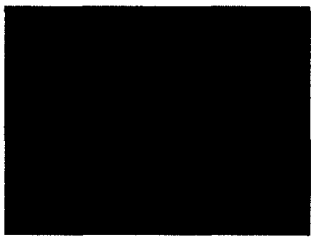
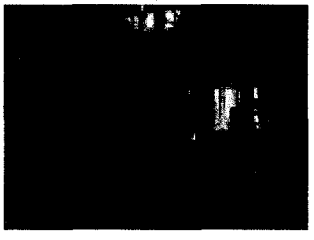
For power consumption analysis, a simulation tool from Altera Quartus II software is used and logic transitions are recorded to the Signal Activity File (SAF). The SAF file is used in the PowerPlay tool to compute the dynamic power consumption of two designs: (1) enhancement system without the proposed power-aware techniques and (2) enhancement system with the proposed power-aware techniques for Booth multiplier, binary logarithm and NDA scheme for 2-D convolution. Table 6.3 shows 10 test images captured under various lighting and background environments, and the respective enhanced images using the proposed system architecture. Table 6.4 provides the dynamic power consumed by the architectures with and without applying the multi-level power design methodologies while enhancing the images shown in Table 6.3. The architecture design without utilizing the multi-level power-aware design methodologies is referred to as "Design without NDA" in Table 6.3. Similarly, the architecture design that incorporates the multi-level power-aware design methodologies is referred to as "Design without NDA" in Table 6.3. It can be observed that the power saving due to the application of the power-aware design methodologies can be reasonably predicted based on the statistical nature of the input image.

6.4. Summary

The multi-level power-aware design methodologies presented in this dissertation have been successfully applied to a real-time architecture design for a practical application of video enhancement. A pipelined and parallel architecture design has been implemented and evaluated for the IRME enhancement algorithm. The architecture consists of three main modules: (1) the illuminace enhancement module, (2) the contrast enhancement module and (3) the color restoration and adjustment module. The NDA method is utilized in the design and implementation of the 2-D convolution unit in the illuminance enhancement and contrast enhancement modules. The proposed Booth multiplier design is used in all three modules. Division and powering operations in the IRME algorithm are transformed to log-based subtraction and multiplication operations in the architecture design. The binary logarithm approximation technique is used for this transformation. Simulation results of two typical test images for the IRME algorithm show minor

difference in pixel intensities between the results obtained by software and hardware means. The proposed design and implementation of the parallel pipelined architecture with utilization of multi-level low power design methodologies achieved more than 29% power reduction compared to the baseline implementation.

Table 6.3. Test images and the respective enhanced images.

Test image	Original image	Enhanced image
1		
2		
3		
4		
5		

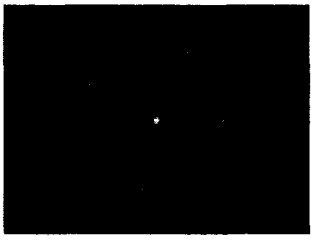
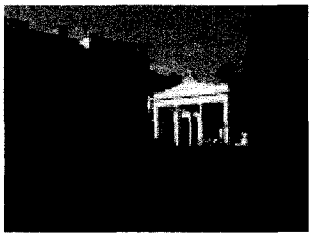
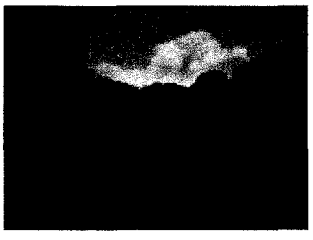
Test image	Original image	Enhanced image
6		
7		
8		
9		
10		

Table 6.4. Experimental results of IRME enhancement algorithm.

Method / Image	Design without NDA		Design With NDA	
	Power (mW)	**Normalized ratio**	**Power** (mW)	**Normalized ratio**
1	122.95	1.00	81.68	0.66
2	150.24	1.00	100.86	0.67
3	160.16	1.00	105.04	0.66
4	154.14	1.00	103.17	0.67
5	160.43	1.00	104.76	0.65
6	144.89	1.00	102.16	0.71
7	121.93	1.00	78.26	0.64
8	187.01	1.00	122.33	0.65
9	158.65	1.00	103.51	0.65
10	145.40	1.00	96.21	0.66

Chapter VII

CONCLUSION AND FUTURE WORKS

In this dissertation, a multi-level approach for power-aware design of real-time video processing systems has been developed, implemented and evaluated. A novel power-aware partitioning and gating technique for pipelined CSA array, Booth and log-based multipliers at the arithmetic and logic level has been presented. At the architectural level, a novel neighborhood dependent approach has been proposed in this dissertation for window-based operations in video processing applications. The power-aware design techniques were utilized in the design of a real-time video enhancement application at the system level. The main contributions of this dissertation can be summarized as follows.

- A partitioning and gating technique to handle special values in higher order parts of the multiplier operands has been proposed to deactivate part(s) of the multipliers for reduced switching activities. Pipelined CSA array multiplier with the new partitioning and gating technique has achieved over 17% power reduction compared to the baseline pipelined CSA array multiplier.

- A radix-4 recoding technique with consideration of the partitioning and gating method has been proposed for radix-4 Booth multiplier. A delayed correction technique for negative partial products has also been presented for the Booth multiplier design to reduce the effect of sign bit extension in the data representation. The new Booth multiplier design achieved over 15% power reduction compared to the baseline Booth multiplier. In addition, the sign magnitude representation of the multiplier operand was incorporated and the proposed Booth_PGSM achieved better results in edge detection compared with the proposed Booth_PG2C.

- Logarithmic domain arithmetic units have been designed and evaluated for power reduction in place of complex operations in the linear domain such as

multiplication, division, powering, etc. A fast and efficient approximation method for binary logarithm and inverse-logarithm was used. An error correction method has been proposed for the binary logarithm approximation technique. The new correction method considers one and two bits to determine regions of interests for correction. The results show significant improvement in accuracy over the baseline Mitchell method, and the results are comparable with other existing methods.

- A new modular design technique for leading bit detection to be used in the binary logarithm approximation method has been proposed. The new detection method integrates a leading one detection unit and a decoder to determine the bit position. The new technique has been used for data words of various lengths by combining multiple functional blocks without significant custom adjustment to the architectural structure. A log-based multiplier with consideration of the partitioning and gating technique has also been presented. The log-based multiplier achieved more than 50% power reduction compared to the linear multipliers (CSA array and Booth). The proposed Mitchell-PG multiplier achieved more than 29% power reduction compared to the Mitchell based multiplier in the edge detection application.

- A novel neighborhood dependent approach has been proposed for low power design and evaluated for window-based operation in video processing applications. The proposed NDA method utilizes the partitioning method to reduce computations for the higher order bits of the pixel values in a window. The NDA method was incorporated in the parallel architecture for 2-D convolution with SWPB and MWPB on-chip buffering schemes. In addition, the proposed NDA method was also applied to the 2-D convolution with quadrant symmetry property. The 2-D convolution with proposed the NDA method achieved over 36% power reduction compared to implementations without NDA method for all implementations.

- A high performance pipelined and parallel architecture for IRME image enhancement algorithm has been presented and implemented. The design utilized the proposed Booth multiplier with partitioning and gating technique in all three main computational modules of the image enhancement system. The NDA method was applied to the parallel architecture design of 2-D convolution modules. In addition, division and powering operations were computed in the logarithmic domain utilizing the proposed binary logarithm approximation technique. The multi-level approach for power-aware design of the image enhancement system achieved more than a 29% power reduction compared to the baseline implementation without consideration of the proposed multi-level power aware design.

Research work in this dissertation considered radix-2 and radix-4 computations for multipliers because they are commonly used in digital systems. As we observed, the radix-4 computations reduce power consumption due to the reduced number of partial products generated. Research work is progressing to consider higher radix arithmetic for power reduction such as radix-8 Booth multiplier design. Another extension of this work is to consider a hybrid system design with fixed point representation and logarithmic representation for multimedia systems. Research work is also progressing in utilizing the data dependencies in subsequent frames in a video stream for the reduction of circuit switching activities and thereby the dynamic power consumption.

REFERENCES

[1] O. Unsal and I. Koren, "Sysetm-level power-aware design techniques in real-time systems," *Proc. of IEEE*, vol. 91, no. 7, pp. 1-15, 2003.

[2] A. Klaiber, "The technology behind crusoe processors," *White Paper - Transmeta Corporation*, pp. 1-18, 2000.

[3] J. Dehnert, B. Grant, J. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson, "The Transmeta code morphing software: using speculation, recovery, and adaptive retranslation to address real-life challenges," *Proc. of Intl. Symp. on Code Generation and Optimization* , pp. 15-24, 2003.

[4] C. M. Krishna and Y. H. Lee, "Voltage-clock-scaling adaptive scheduling techniques," *IEEE Trans. on Computers*, vol. 52, no. 12, pp. 1586-1593, 2003.

[5] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low power pipelined based on circuit level timing speculation," *Proc. of 36th Intl. Symp. on Microarchitecture*, pp. 7-18, 2003.

[6] L. Benini, A. Macii, and M. Poncino, "Selective instruction compression for memory energy reduction in embedded systems," *Proc. of Intl. Symp. on Low Power Electronics and Design*, pp. 206-211, 1999.

[7] H. Lekatsas, J. Henkel, and W. Wolf, "Approximate arithmetic coding for bus transition reduction in low power designs," *IEEE Trans. on VLSI Systems*, vol. 13, no. 6, pp. 696-707, 2005.

[8] M. R. Stan and W. P. Burleson, "Bus-invert coding for low power I/O," *IEEE Trans. on VLSI*, vol. 3, issue 1, pp. 49-58, 1995.

[9] M. Mamidipaka, D. Hirschberg and N. Dutt, "Low power address encoding using self-organizing lists," *Proc. of Intl. Symp. on Low Power Electronics and Design,* pp. 188-193, 2001.

[10] R. Henning and C. Chakrabarti, "An approach to switching activities consideration during high-level, low-power design space exploration," *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 49, no. 5, pp. 339 - 351 2002.

[11] L. H. Lee, B. Moyer and J. Arends, "Instruction fetch energy reduction using loop caches for embedded applications with small tight loops," *Proc. of Intl. Symp. on Low Power Electronics and Design, 1999,* pp. 267-269, 1999.

[12] H. Kapadia, L. Benini and G. D. Mecheli, "Reducing switching activity on datapath buses with control-signal gating," *IEEE Journal of Solid State Circtuis*, vol. 34, no. 3, pp. 405-414, 1999.

[13] H. Kim, I. C. Park, "High-performance and low-power memory interface architecture for video processing applications," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 11, no. 11, pp. 1160-1170, 2001.

[14] K. Gandhi and N. Mahaptra, "Dynamically exploiting frequent operand values for energy efficiency in integer functional units," *Proc. of 18th Intl. Conf. on VLSI Design,* pp. 570-575, 2005.

[15] L. Benini, G. D. Micheli, A. Macii, E. Macii, M. Poncino and R. Scarsi, "Glitch power minimization by selective gate freezing," *IEEE Trans. on VLSI,* vol. 8, no. 3, pp. 287-298, 2000.

[16] M. Hasan and T. Arslan, "Implementation of low-power FFT processor cores using a novel order-based processing scheme," *IEE Proc. of Circuits, Devices and Systems*, vol. 150, no. 3, pp. 149-154, 2003.

[17] Y. -W. Wu, O. T.-C. Chen and R.-L. Ma, "A low power digital signal processor core by minimizing inter-data switching activities," *Proc. of 44th IEEE Midwest Symp. on Circuits and Systems,* vol. 1, pp 172-175, 2001.

[18] N. Sankarayya, K. Roy and D. Bhattacharya, "Algorithms for low power and high speed FIR filter realization using differential coefficients," *IEEE Trans. on Circuits*

and Systems – II: Analog and Digital Signal Processing, vol. 44, no. 6, pp. 488-497, 1997.

[19] A. P. Vinod and I. Setiawan, "A minimal-difference differential coefficients method for low complexity FIR filter realization," *Proc. of 8th Intl. Symp. on Signal Processing and Its Applications*, pp. 155-158, 2005.

[20] C. –Y. Han, H. –J. Park and L. –S. Kim, "A low power array multiplier using separated multiplication technique," *IEEE Trans. on Circuits and Systems – II: Analog and Digital Signal Processing,* vol. 48, no. 9, pp. 866-871, 2001.

[21] Z. Yu, M. –L. Yu, K. Azadet and A. N. Wilson, "A low power FIR filter design technique using dynamic reduced signal representation," *Proc. of Intl. Symp. on VLSI Technology, Systems, and Applications,* pp. 113-116, 2001.

[22] E. F. Stefatos, H. Wei, T. Arslan and R. Thomson, "Low-power reconfigurable VLSI architecture for the implementation of FIR filters," *Proc. of 19th IEEE Intl. Parallel and Distributed Processing Symposium*, pp. 4-7, 2005.

[23] A. T. Erdogan, M. Hasan and T. Arslan, "Algorithmic low-power FIR cores," *IEE Proc. of Circuits, Devices and Systems*, vol. 150, no. 3, pp. 155-160, 2003.

[24] A. T. Erdogan, E. Zwyssig, and T. Arslan, "Architectural trade-offs in the design of low power FIR filtering cores," *IEE Proc. of Circuit Devices and Systems*, vol. 151, no. 1, pp. 10-17, 2004.

[25] V. Menon, S. Chennupati, N. K. Samala, D. Radhakrishman, and B. Izadi, "Switching activity minimization in combinational logic design," *Proc. of the Intl. Conf. on Embedded Systems and Applications*, pp. 47-53, 2004.

[26] M. Vratonjic, B. R. Zeydel, and V. G. Oklobdzija, "Low and ultra-low power arithmetic units: design and comparison," *Proc. of Intl. Conf. on Computer Design*, pp. 249-252, 2005.

[27] K. Chirca and M. Schulte, "A static low power, high performance 32-bit carry skip adder," *Euromicro Symp. on Digital System Design*, pp. 615-619, 2004.

[28] S. Mathew, "A 4GHz 130nm address generation unit with 32-bit sparse-tree adder core," *IEEE JSSCC*, vol. 38, no.5, pp. 689-695, 2003.

[29] M. Schulte, K. Chirca, J. Glossner, H. Wang, S. Mamidi, B. Balzola and S. Vassiliadis, "A low-power carry skip adder with fast saturation," *Proc. of the Intl. Conf. on Application-Specific Systems, Architectures and Processors*, pp. 269-279, 2004.

[30] Y. Jiang, A. A-Sheraidah, Y. Wang, E. Sha and J. –G. Chung, "A novel multiplexer-based low-power full adder," *IEEE Trans. on Circuits and Systems – II: Express Briefs,* vol. 51, no. 7, pp. 345-348, 2004.

[31] P. M. Seidel, "Dynamic operand modification for reduced power multiplication," *Proc. of 36th Asilomar Conf. on Signals, Systems and Computers*, pp. 52-56, 2002.

[32] M. Ito, D. Chinnery, and K. Keutzer, "Low power multiplication algorithm for switching activity reduction through operand decomposition," *Proc. of the 21st Intl. Conf. on Computer Design*, pp. 21-26, 2003.

[33] N. Rollin and M. J. Wirthlin, "Reducing energy in FPGA multipliers through glitch reduction," *Proc. of the 7th Intl. Conf. on Military Applications of Programmable Logic Devices*, pp. 1-10, 2005.

[34] R. Fisher, K. Buchenrieder, and U. Nageldinger, "Reducing the power consumption of FPGAs through retiming," *Proc. of the 12th IEEE Intl. Conf. and Workshops on Engineering of Computer-Based Systems*, pp. 89-94, 2005.

[35] Y. L. Hs and S. J. Wang, "Retiming-based logic synthesis for low power," *Proc. of Intl. Symp. on Low Power Elect. and Design*, pp. 275-278, 2002.

[36] S. Wilton, S. Ang, and W. Luk, "The impact of pipelining on energy per operation in field-programmable gate arrays," *Proc. of 14th Intl. Conf. on Field Programmable Logic and Application*, pp. 719-728, 2004.

[37] E. J. King and E. E. Swartzlander, "Data-dependent truncated scheme for parallel multiplication," *Proc. of Asilomar Conf. Circuits and Systems*, pp. 1178-1182, 1998.

[38] M. J. Schulte , J. E. Stine , and J. G. Jansen, "Reduced power dissipation through truncated multiplication," *Proc. of the IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, p.61-69, 1999.

[39] C. –H. Chang, R. K. Satzoda and S. Sekar, "A novel multiplexer based truncated array multiplier," *Proc. of the IEEE Intl. Symp. on Circuits and Systems,* pp. 85-88, 2005.

[40] K. Han, B. L. Evans and E. E. Swartzlander, "Low-power multiplier with data wordlength reduction," *Proc. of 39th Asilomar Conf. on Signals, Systems and Computers,* pp. 1615 – 1619, 2005.

[41] V. Moshnyaga, "Reducing switching activity of subtraction via variable trucaction of the most-significant bits," *Journal of VSLI Signal Processing*, no. 33, pp. 75-82, 2003.

[42] . Mudassir and Z. Abid, "New parallel multipliers based on low power adders," *Proc. of Canadian Conf. on Electrical and Computer Engineering*, pp 694-697, 2005.

[43] M. Fujino and V. Moshnyaga, "Dynamic operand transformation for low power multiplier-accumulator design," *Proc. of Intl. Symp. on Circuits and Systems,* vol. 5, pp. V-345 - V-348, 2003.

[44] M. Riazati, A. Sobhani, M. M. Dastjerdi, A. A. Kusha and A. Khakifirooz, "Low-power multiplier with static decision for input manipulation," *Proc. of Intl. Symp. on Circuits and Systems,* pp. 2721-2724, 2006.

[45] Y. Liu and S. Furber, "The design of a low power asynchronous multiplier," *Proc. of Intl. Symp. on Low Power Electronic and Design*, pp. 301-306, 2004.

[46] S. Kim and M C. Papaefthymiou, "Reconfigurable low energy multiplier for multimedia system design," *Proc. of IEEE Workshop on VLSI,* pp. 129-134, 2000.

[47] J. Di and J. S. Yuan, "Power-aware pipelined multiplier design based on 2-dimensional pipeline gating," *Proc. of the 13th ACM Great Lakes Symp. on VLSI,* pp. 64-67, 2003.

[48] G. Economakos, K. Anagnostopoulos, "Bit level architectural exploration technique for the design of low power multipliers," *Proc. of IEEE Intl. Symp. on Circuits and Systems*, pp. 1483-1486, 2006.

[49] M. C. Wen, S. J. Wang, and Y. N. Lin, "Low power parallel multiplier with column bypassing," *Electronics Letters*, vol. 41, no. 10, pp. 581 – 583, 2005.

[50] M. Fonseca, E. Costa, S. Bambi and J. Monteiro, "Design of a radix-2 hybrid array multiplier using carry save adder," *Proc. of 18th Symposium on Integrated Circuits and Systems Design*, pp. 172-177, 2005.

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

[52] K. Y. Khoo, Z. Yu and A. N. Wilson Jr., "Improved Booth encoding for low power multipliers," *Proc. of IEEE Intl. Symp. On Circuits and Systems*, pp, I62-I65, 2006.

[53] Z. Yu, L. Wasserman and A. N. Wilson Jr., "A painless way to reduce power dissipation by over 18% in Booth-encoded carry-save array multipliers for DSP," *Proc of IEEE workshop on Signal Processing Systems*, pp. 571-580, 2000.

[54] T. Ahn and K. Choi, "Dynamic operand interchange for low power," *Electronic Letters*, vol.33, no. 25, pp. 2118-2120, 1997.

[55] O. T-C Chen, S. Wang and Y-W Wu, "Minimization of switching activities of partial product for designing low-power multipliers," *IEEE Trans. on Very Large Scale Integration (VLSI) System*, vol. 11, no.3, pp. 418-433, 2003.

[56] J. Park, S. Kim and Y-S Lee, "A low power Booth multiplier using novel data partition method," *Proc. of IEEE Asia-Pacific Conf. on Advanced System Integrated Circuits*, pp. 54-57, 2004.

[57] O. T-C Chen, N. –Y. Shen and C. –C. Shen, "A low-power multiplication accumulation calculation unit for multimedia applications," *Proc. of IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, vol. 2, pp. II-645-II-648, 2003.

[58] K. T. Gribbon, C. T. Johnston, and D. G. Bailey, "A real-time FPGA implementation of a barrel distortion correction algorithm with bilinear interpolation," *Proc. of Image and Vision Computing*, pp. 402-407, 2003.

[59] B. A. Draper, J. R. Beveridge, A. P. W. Bohm, C. Ross, and M. Chawathe, "Accelerate image processing on FPGAs," *IEEE Trans. on Image Processing,* vol. 12, no. 12, pp. 1543 – 1551, 2003.

[60] B. A. Draper, J. R. Beveridge, A. P. W. Bohm, C. Ross, and M. Chawathe, "Implementing image applications on FPGAs," *Proc. of 16th Intl. Conf. on Pattern Recognition,* vol. 3, pp. 265-268, 2002.

[61] S. Choi, R. Scrofano, V. K. Prasanna and J. W. Jang, "Energy-efficient signal processing using FPGAs" *Proc. of the 2003 ACM/SIGDA 11th Intl. Symp. on FPGA*, pp. 225-234, 2003.

[62] V. K. Prasanna, "Energy-efficient computations on FPGAs," *The Journal of Supercomputing,* vol. 32, issue 2, pp. 139-162, 2005.

[63] B. Kumthekar, L. Benini, E. Macii and F. Somenzi, "Power optimization of FPGA-based designs without rewiring," *IEE Proc. of Computers and Digital Techniques*, vol. 147, issue 3, pp. 167-174, 2000.

[64] E. Boemo, S. L. –Buedo, C. S. Perez, J. Jauregui and J. Meneses, "Logic depth and power consumption: a comparative study between standard cells and FPGAs," *Proc. of XIII Design of Circuits and Integrated Systems Conf.*, pp. 1-5, 1998.

[65] J. Becker, M. Huebner and M. Ullmann, "Power estimation and power measurement of Xilinx Virtex FPGAs: trade-offs and limitations," *Proc. of 16th Symp. on Integrated Circuits and Systems Design)*, pp. 283-288, 2003.

[66] L. Shang, A. S. Kaviani and K. Bathala, "Dynamic power consumption in Virtex II FPGA family," *Proc. of 2002 ACM/SIGDA 11th Intl. Symp. on FPGA*, pp. 157-164, 2002.

[67] Altera Inc., "Power optimization," *Quartus II Handbook Volume 2: Design Implementation and Optimization*, chapter 9, pp. 9_1-9-40, 2007.

[68] Altera Inc., "Power optimization in Stratix III FPGAs," *Applications Note 437*, pp. 1-13, 2007.

[69] R. C. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. on Computer*, vol. C-22, pp. 1045-1047, 1973.

[70] Z. Huang and M. Ercegovac, "Two dimensional gating for low power array multiplier design," *Proc. Intl. Symp. on Circuits and Systems*, vol. 1, pp. I-489 - I-492, 2002.

[71] A. A. Fayed and M. A. Bayoumi, "A novel architecture for low-power design of parallel multipliers," *Proc. of IEEE Computer Society Workshop on VLSI*, pp. 149-154, 2001.

[72] S. Ramprasad, N. R. Shanbha and I. N. Hajj, "Analytical estimation of signal transition activity from word-level statistics," *IEEE Trans. on Comp. Aided Design of Integrated Circuits and Systems*, vol. 16, no. 7, pp. 718-733, 1997.

[73] M. Zheng and A. Albicki, "Low power and high speed multiplication design through mixed number representations," *Proc. of 1995 IEEE Intl. Conf. Computer Design*, pp. 566-570, 1995.

[74] V. Paliouras and T. Stouraitis, "Low-power properties of the logarithmic number system," *Proc. of 15th IEEE Symp. on Computer Arithmetic*, pp. 229-236, 2001.

[75] B. –G. Nam, H. Kim and H. –J. Yoo, "A low-power unified arithmetic unit for programmable handheld 3-D graphics systems," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 8, pp. 1767-1778, 2007.

[76] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers,* pp. 512-517, 1962.

[77] V. Mahalingam and N. Ranganathan, "Improving accuracy in Mitchell's logarithmic multiplication using operand decomposition," *IEEE Trans. on Computers*, vol. 55, no. 12, pp. 1523-1535, 2006.

[78] M. J. Duncan, "Improve Mitchell based logarithmic multiplier for low power DSP applications," *Proc. of IEEE Intl. System on a Chip Conf.*, pp. 17-20, 2003.

[79] E. L. Hall, D. D. Lynch and S. J. Dwyer III, "Generation of products and quotients using approximate binary logarithms for digital filtering applications," *IEEE Trans. on Computers*, vol. 19, issue 3, pp. 97-105, 1970.

[80] S.L. SanGregory, C. Brothers, D. Gallagher, and R. Siferd, "A fast, low-power logarithm approximation with CMOS VLSI implementation", *Proc. of Midwest Symp. on Circuits and Systems,* vol. 1, pp. 388–391, 1999.

[81] K. H. Abed and R. Siferd, "CMOS VLSI implementation of a low power logarithmic converter," *IEEE Trans. on Computers*, vol. 52, no. 11, pp. 141-1433, 2003.

[82] B. Bosi, G. Bois, and Y. Savaria, "Reconfigurable pipelined 2-D convolvers for fast digital signal processing," *IEEE Trans. on VSLI System*, vol. 7, no. 3, pp. 299-308, 1999.

[83] S. Hong and S. Chin, "Domain specific reconfigurable processing core architecture for digital filtering applications," *Journal of VLSI Signal Processing*, no. 40, pp. 239-259, 2005.

[84] C. T. Huitzil and M. A. Estrada, "FPGA-based configurable systolic architecture for window-based image processing," *EURASIP Journal on Applied Signal Processing*, vol. 7, pp. 1024-1034, 2005.

[85] R. Managuli, G. York, D. Kim, and Y. Kim, "Mapping of two dimensional convolution on very long instruction word media processor for real-time processing," *Journal of Elect. Imaging*, vol. 7, no. 3, pp. 327-335, 2000.

[86] D. Draper, W. Najjar, and W. Bohm, "Compiling and optimizing image processing algorithms for FPGA's," *proc. Intl. Workshop on Comp. Architecture for Machine Performance*, pp. 240-246, 2000.

[87] M. A. Estrada and C. T. Huitzil, "Real-time field programmable gate array architecture for computer vision," *Journal of Electronic Imaging*, vol. 10, no. 1, pp. 57-59, 2001.

[88] K. Doshi and P. Varman, "A modular systolic architecture for image convolutions," *Proc. of 14th Intl. Symp. on Computer Architecture*, pp. 56-63, 1987.

[89] X. Liang, J. S. N. Jean and K. Tomko, "Data buffering and allocation in mapping generalized template matching on reconfigurable systems," *Journal of Supercomputer*, vol. 19, vol. 1, pp. 77-91, 2001.

[90] F. C. Tormo and P. Molinet, "Area-efficient 2-D shift-variant convolvers for FPGA-based digital image processing," *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 53, no. 2, pp. 105-109, 2006.

[91] H. Zhang, M. Xia and G. Hu, "A Multiwindow partial buffering scheme for FPGA-based 2-D convolvers," *IEEE Trans. on Circutis and Systems II: Express Briefs*, vol. 54, no. 2, pp. 200-204, 2007.

[92] L. Tao and V. K. Asari, "An efficient illuminance-reflectance nonlinear video stream enhancement model," *Proc. of IS&T/SPIE Symp. on Electronic Imaging: Real-Time Image Processing III*, vol. 6063. pp. 60630I-1-12, 2006.

[93] Hau T. Ngo, Ming Z. Zhang, Li Tao and K. Vijayan Asari, "Design of a digital architecture for real-time video enhancement based on illuminance-reflectance model," *Proc. of 49th IEEE Intl. Midwest Symp. on Circuits and Systems*, pp. 3179-1-5, 2006.

VITA

(December 2007)

Hau Trung Ngo
http://www.lions.odu.edu/~hngox001

EDUCATION

MS in Computer Engineering, Old Dominion University, Norfolk VA, 2003.
BS in Computer Engineering, Old Dominion University, Norfolk VA, 2001.
AS in Engineering, Tidewater Community College, Virginia Beach, VA, 1999.

RESEARCH PUBLICATIONS

1. Hau T. Ngo, Vijayan K. Asari, Ming Z. Zhang, and Li Tao "Design of a systolic-pipelined architecture for real-time enhancement of color video stream based on an illuminance-reflectance model," *Journal of VLSI Integration* (accepted for publication).
2. Ming Z. Zhang, Hau T. Ngo, Adam R. Livingston, and Vijayan K. Asari, "A high performance architecture for implementation of 2-D convolution with quadrant symmetric kernels," *International Journal of Computers and Applications* (accepted for publication).
3. Hau T. Ngo, Ming Z. Zhang, Li Tao and Vijayan K. Asari, "An architecture design for real-time video enhancement based on a luma-dependent nonlinear approach," *International Journal of Embedded System, Special Issue on Media and Stream Processing* (in print).
4. Hau T. Ngo and Vijayan K. Asari, "Design of a FPGA-based high performance, power-aware architecture for real-time radial lens distortion correction of video streams," *International Journal of Programmable Devices, Circuits, and Systems – Special Issue on FPGA-Based Digital Signal and Image Processing Systems*, vol. 7, no. 1, pp. 33-41, May 2007.
5. Ming Z. Zhang, Hau T. Ngo and Vijayan K. Asari, "Multiplier-less VLSI architecture for real-time computation of multi-dimensional convolution" *Journal of Microprocessors and Microsystems*, vol. 31, pp. 25-37, February 2007.
6. Rajkiran Gottumukkal, Hau T. Ngo and K. Vijayan Asari, "Multi-lane architecture for eigenface based real-time face recognition" *Journal of Microprocessors and Microsystems*, vol. 30, no. 4, pp. 216-224, June 2006.
7. Hau T. Ngo and K. Vijayan Asari, "A pipelined architecture for real-time correction of barrel distortion in wide-angle camera images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 3, pp. 436-444, March 2005.
8. Ming-Jung Seow, Hau T. Ngo, and K. Vijayan Asari, "Systolic implementation of 2-D block based Hopfield neural network for efficient pattern association" *Journal of Microprocessors and Microsystems*, vol. 27, no. 8, pp. 359-366, August 2003.