LOW POWER H.264 VIDEO COMPRESSION HARDWARE DESIGNS

by

Mustafa Parlak

Submitted to the Graduate School of Engineering and Natural Sciences

in partial fulfillment of
the requirements for the degree of
Doctorate of Philosophy

Sabancı University

February 2009

LOW POWER H.264 VIDEO COMPRESSION HARDWARE DESIGNS

APPROVED BY:

Assist. Prof. Dr. İlker Hamzaoğlu …….
(Thesis Supervisor)

Assist. Prof. Dr. Ayhan Bozkurt …….

Assist. Prof. Dr. Müjdat Çetin …….

Prof. Dr. Günhan Dündar …….

Assist. Prof. Dr. Hakan Erdoğan …….

DATE OF APPROVAL: …….

To my Mother, Father, Brothers and Sisters

To my beloved wife Neslihan and our future children

ACKNOWLEDGEMENT

I would like to thank my supervisor, Dr. İlker Hamzaoğlu for all his guidance, support, and patience throughout my PhD study. It has been a great honor for me to work under his guidance.

I would also like to thank my thesis committee members Dr. Ayhan Bozkurt and Dr. Müjdat Çetin for their valuable comments on the dissertation, and Dr. Günhan Dündar and Dr. Hakan Erdoğan for participating in my thesis jury.

My special thanks to System-on-Chip Design & Test group members, particularly Yusuf Adıbelli and Özgür Taşdizen for their collaboration and help during my PhD study.

My sincere thanks to all my friends and colleagues in Sabancı University including Mehmet Özdemir, Alisher Kholmatov, Ünal Şen and İbrahim İnanç. I appreciate their friendship and help which made my life easier and more pleasant during my PhD study.

I am particularly grateful to my parents and my wife, Neslihan, for their constant support, encouragement, assistance and patience. Without them, this study would never have been possible.

I would like to thank Sabanci University for supporting this research. I would also like to thank Scientific and Technological Research Council of Turkey (TUBITAK) for supporting this research under the contract 106E153.

LOW POWER H.264 VIDEO COMPRESSION HARDWARE DESIGNS

Mustafa Parlak

ABSTRACT

Video compression systems are used in many commercial products such as digital camcorders, cellular phones and video teleconferencing systems. H.264 / MPEG4 Part 10, the recently developed international standard for video compression, offers significantly better video compression efficiency than previous international standards. However, this coding gain comes with an increase in encoding complexity and therefore in power consumption. Since portable devices operate with battery, it is important to reduce power consumption so that the battery life can be increased. In addition, consuming excessive power degrades the performance of integrated circuits, increases packaging and cooling costs, reduces the reliability and may cause device failures. Therefore, power consumption is an important design metric for video compression hardware.

In this thesis, we propose low power hardware designs for Deblocking Filter (DBF), intra prediction and intra mode decision parts of an H.264 video encoder. The proposed hardware architectures are implemented in Verilog HDL and mapped to Xilinx Virtex II FPGA. We performed detailed power consumption analysis of FPGA implementations of these hardware designs using Xilinx XPower tool. We also measured the power consumptions of DBF hardware implementations on a Xilinx Virtex II FPGA and there is a good match between estimated and measured power consumption results.

We then worked on decreasing the power consumption of FPGA implementations of these H.264 video compression hardware designs by reducing switching activity using Register Transfer Level (RTL) low power techniques. We applied several RTL low power techniques such as clock gating and glitch reduction to these designs and quantified their impact on the power consumption of the FPGA implementations of these designs. We proposed novel computational complexity and power reduction techniques which avoid

unnecessary calculations in DBF, intra prediction and intra mode decision parts of an H.264 video encoder. We quantified the computation reductions achieved by the proposed techniques using H.264 Joint Model software encoder. We applied these techniques to proposed hardware designs and quantified their impact on the power consumption of the FPGA implementations of these designs.

DÜŞÜK GÜÇ KULLANIMLI H.264 VİDEO SIKIŞTIRMA DONANIM TASARIMLARI

Mustafa Parlak

ÖZET

Video sıkıştırma sistemleri, dijital kameralar, cep telefonları ve video telekonferans sistemleri gibi bir çok ticari üründe kullanılmaktadır. Yakın tarihte geliştirilmiş uluslararası bir standart olan H.264 / MPEG4 Part 10, kendinden önceki standartlara göre belirgin şekilde daha iyi sıkıştırma verimi sağlamaktadır. Ancak,bu kodlama kazancı hesaplama karmaşıklığı ve güç tüketimi artışını beraberinde getirmektedir. Taşınabilir cihazlar pil ile çalıştığı için, güç tüketimini azaltmak pil ömrünün uzamasını sağlayacaktır. Bunun yanında aşırı güç tüketimi, entegre devrelerin performansını düşürür, paketleme ve soğutma maliyetlerini arttırır, dayanıklılığını azaltır ve bozulmalarına sebep olabilir. Bu nedenle,güç tüketimi, video sıkıştırma donanımları için önemli bir tasarım ölçüsüdür.

Bu tezde, H.264 Blok Giderici Filtre (BGF), çerçeve içi öngörü ve çerçeve içi kip seçimi algoritmaları için düşük güç kullanımlı donanım tasarımları önerildi. Önerilen donanım mimarileri Verilog HDL ile gerçeklendi ve Xilinx Virtex II FPGA ye sentezlendi. Xilinx XPower yazılımı kullanılarak bu donanımların FPGA gerçeklemelerinin detaylı güç tüketim analizleri yapıldı. Ayrıca Xilinx Virtex II FPGA üzerinde çalışan BGF donanımının güç tüketimi ölçüldü ve tahmin edilen güç tüketimi ile ölçülen güç tüketimi arasında yakın sonuçlar elde edildi.

Daha sonra H.264 video sıkıştırma donanım tasarımlarının FPGA gerçeklemelerinin güç tüketimleri saklayıcı aktarma (RTL) seviyesinde düşük güç teknikleri ile anahtarlama aktiviteleri düşürülerek azaltılmaya çalışıldı. Bu donanımlara saat kapılama, küçük sıçramaları azaltma gibi RTL seviyesinde düşük güç teknikleri uygulandı ve bu tekniklerin bu donanımların FPGA içindeki güç tüketimleri üzerindeki etkileri belirlendi. Ayrıca bu tezde H.264 video kodlayıcıda bulunan BGF, çerçeve içi öngörü ve çerçeve içi kip seçimi modüllerindeki gereksiz hesaplamaları engelleyen, özgün sayısal karmaşıklık ve güç

tüketimi azaltıcı teknikler önerildi. Önerilen tekniklerin hesaplama miktarında yaptığı azalma H.264 referans yazılımı (JM) kullanılarak belirlendi. Bu teknikler önerilen donanım tasarımlarına uygulandı ve bu tekniklerin bu donanımların FPGA içindeki güç tüketimleri üzerindeki etkileri belirlendi.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 H.264 Video Compression Standard

Video compression systems are used in many commercial products, from consumer electronic devices such as digital camcorders, cellular phones to video teleconferencing systems. These applications make the video compression hardware devices an inevitable part of many commercial products. To improve the performance of the existing applications and to enable the applicability of video compression to new real-time applications, recently, a new international standard for video compression is developed. This new standard, offering significantly better video compression efficiency than previous video compression standards, is developed with the collaboration of ITU and ISO standardization organizations. Hence it is called with two different names, H.264 and MPEG4 Part 10 [1].

H.264 video coding standard has a much higher coding efficiency (capable of saving up to %50 bit rate at the same level of video quality) than the previous standards [2]. Due to its high coding efficiency and due to its flexibility and robustness to different

communication environments, in the near future, H.264 is expected to be widely used in many applications such as digital TV, DVD, video transmission in wireless networks, and video conferencing over the internet.

The human visual system appears to distinguish scene content in terms of brightness and color information individually, and with greater sensitivity to the details of brightness than color [3]. Same as the previous video compression standards, H.264 is designed to take advantage of this by using YCbCr color space. In YCbCr color space, each pixel is represented with three 8-bit components called Y, Cb, and Cr. Y, the luminance (luma) component, represents brightness. Cb and Cr, chrominance (chroma) components, represent the extent to which the color differs from gray toward blue and red, respectively. Since the human visual system is more sensitive to luma component than chroma components, H.264 standard uses 4:2:0 sampling. In 4:2:0 sampling, for every four luma samples, there are two chroma samples, one Cb and one Cr.

The top-level block diagram of an H.264 video encoder is shown in Figure 1.1. As shown in the figure, the video compression efficiency achieved in H.264 standard is not a result of any single feature but rather a combination of a number of encoding tools such as motion estimation, intra prediction and deblocking filter (DBF). Same as the previous video compression standards, H.264 standard does not specify all the algorithms that will be used in an encoder such as mode decision. Instead, it defines the syntax of the encoded bit stream and functionality of the decoder that can decode this bit stream.

As shown in Figure 1.1, an H.264 encoder has a forward path and a reconstruction path. The forward path is used to encode a video frame and create the bit stream by using intra and inter predictions. The reconstruction path is used to decode the encoded frame and reconstruct the decoded frame. Since a decoder never gets original images, but rather works on the decoded frames, reconstruction path in the encoder ensures that both encoder and decoder use identical reference frames for intra and inter prediction. This avoids possible encoder – decoder mismatches [1,3,4].

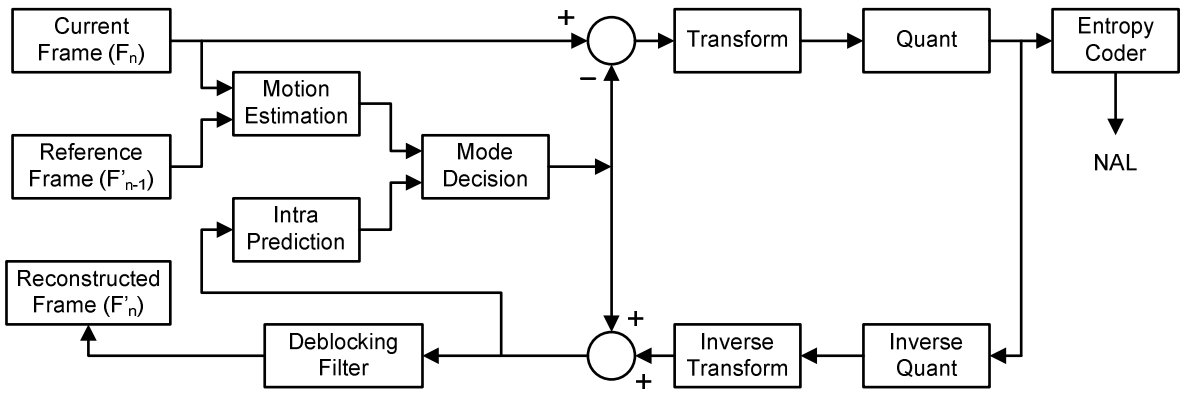

Figure 1.1 H.264 Encoder Block Diagram

Forward path starts with partitioning the input frame into macroblocks (MB). Each MB is encoded in intra or inter mode depending on the mode decision. In both intra and inter modes, the current MB is predicted from the reconstructed frame. Intra mode generates the predicted MB based on spatial redundancy, whereas inter mode, generates the predicted MB based on temporal redundancy. Mode decision compares the required amount of bits to encode a MB and the quality of the decoded MB for both of these modes and chooses the mode with better quality and bit-rate performance. In either case, intra or inter mode, the predicted MB is subtracted from the current MB to generate the residual MB. Residual MB is transformed using 4x4 and 2x2 integer transforms. Transformed residual data is quantized and quantized transform coefficients are re-ordered in a zig-zag scan order. The reordered quantized transform coefficients are entropy coded. The entropy-coded coefficients together with header information, such as MB prediction mode and quantization step size, form the compressed bit stream. The compressed bit stream is passed to network abstraction layer (NAL) for storage or transmission [1,3,4].

Reconstruction path begins with inverse quantization and inverse transform operations. The quantized transform coefficients are inverse quantized and inverse transformed to generate the reconstructed residual data. Since quantization is a lossy process, inverse quantized and inverse transformed coefficients are not identical to the original residual data. The reconstructed residual data are added to the predicted pixels in order to create the reconstructed frame. DBF is, then, applied to reduce the effects of blocking artifacts in the reconstructed frame [1,3,4].

H.264 intra prediction and mode decision algorithms have a very high computational complexity. Because, in order to improve the compression efficiency, H.264 standard uses many intra prediction modes for a MB and selects the best mode for that MB using a mode decision algorithm.

The DBF algorithm used in H.264 standard is more complex than the DBF algorithms used in previous video compression standards. First of all, H.264 DBF algorithm is highly adaptive and applied to each edge of all the 4×4 luma and chroma blocks in a MB. Second, it can update 3 pixels in each direction that the filtering takes place. Third, in order to decide whether the DBF will be applied to an edge, the related pixels in the current and neighboring 4×4 blocks must be read from memory and processed. Because of these complexities, the DBF algorithm can easily account for one-third of the computational complexity of an H.264 video decoder [4,5].

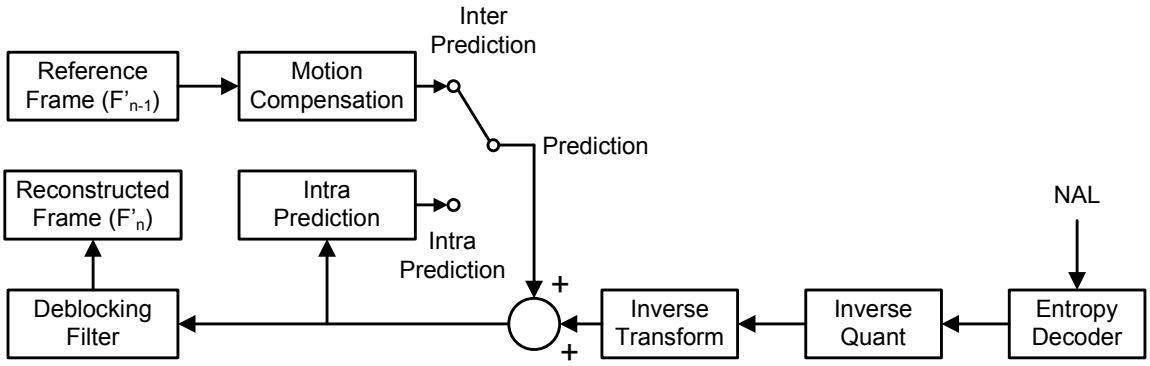

Figure 1.2 H.264 Decoder Block Diagram

H.264 decoder is similar to the reconstruction path of H.264 encoder. It receives a compressed bit stream from the NAL as shown in Figure 1.2. The bit stream is decoded, inverse quantized and inverse transformed to get residual data. Using the header information decoded from the bit stream, the decoder creates a prediction block, identical to prediction block generated in reconstruction path of H.264 encoder. The prediction block is added to the residual block to create the reconstructed block. Blocking artifacts are, then, removed from reconstructed block by applying DBF.

H.264 has three profiles; Baseline, Main, and Extended. Each profile has 14 levels. A profile is a set of algorithmic features and a level shows encoding capability such as picture

size and frame rate. In this thesis, we use Baseline profile level 2.0. Baseline profile has lower latency than main and extended profiles, and it is used for wireless video applications and video conferencing. In Baseline profile level 2.0, video is digitized at CIF (352x288) size YCbCr using 4:2:0 sampling at 30 frames per second, and I slices, P slices and context-adaptive variable length entropy coding are supported [1,3].

1.2 Low Power Hardware Design

Multimedia applications running on portable devices have increased recently and this trend is expected to continue in the future. Since portable devices operate with battery, it is important to reduce power consumption so that battery life can be increased. Therefore, power consumption has become a critical design metric for portable applications.

In addition, consuming excessive power for a long time causes the chips to heat up and degrades the performance, because transistors run faster when they are cool rather than hot. Excessive power consumption also increases packaging and cooling costs. Excessive power consumption also reduces the reliability and may cause device failures [6]. Repeated cycling from hot to cool shortens the life of a chip by inducing mechanical stress that can literally tear a chip apart. Hot metal interconnects on the chip are also more susceptible to disintegration because of a phenomenon called electromigration. Therefore, there is an upper bound for allowed power consumption in integrated circuits (IC).

The maximum allowable power consumption for ICs projected by International Technology Roadmap for Semiconductors in 2005 is given in Figure 1.3 [7]. In the figure, maximum allowable power for three types of applications is presented; high-performance applications for which a heat sink on the package is permitted, cost-performance applications for which economical power management solutions are used and applications with portable battery for which no cooling system is used. In all cases, total power consumption continues to increase, despite the use of a lower supply voltage. The increased power consumption is driven by higher operating frequencies, higher overall interconnect capacitance, and exponentially growing number of scaled transistors [7]. Therefore, power consumption is an important design metric for all applications.

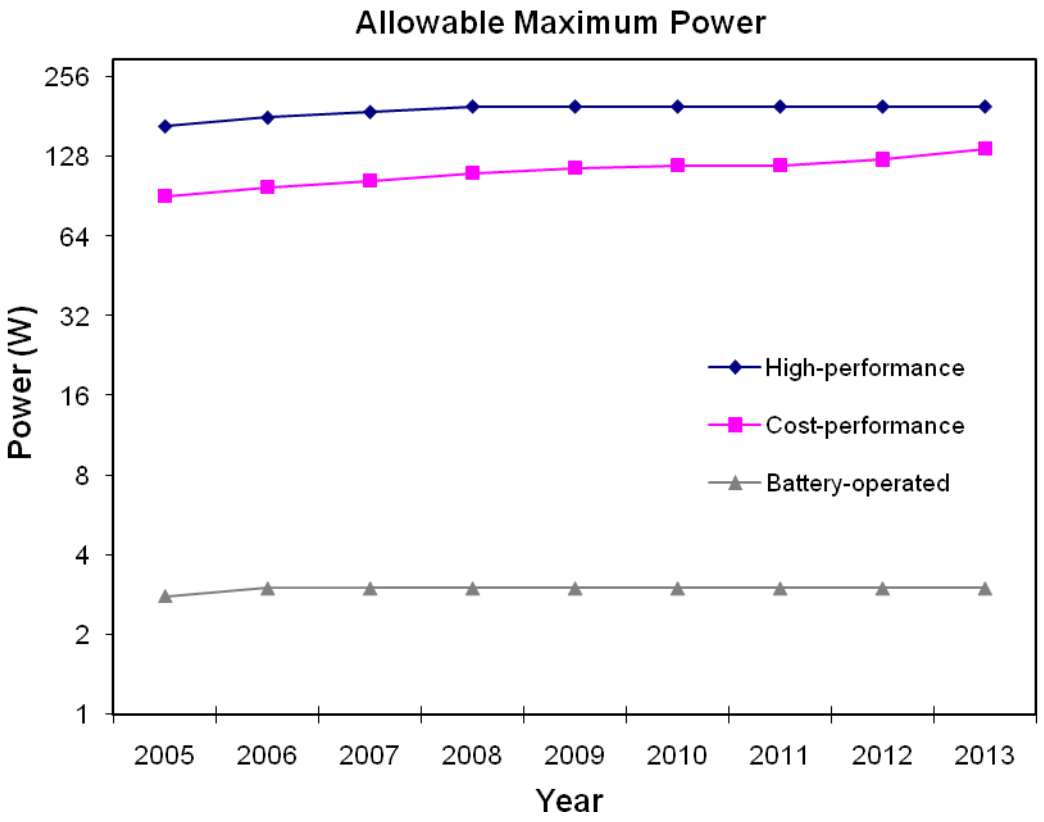

Figure 1.3 ITRS 2005 Projection of Maximum Allowable Power Consumption for ICs

Field Programmable Gate Arrays (FPGA) consume more power than standard cell-based Application Specific Integrated Circuits (ASIC). FPGAs have look-up tables and programmable switches. Look-up table based logic implementation is inefficient in terms of power consumption and programmable switches have high power consumption because of large output capacitances. Therefore, power consumption is an even more important design metric for FPGA implementations.

ICs have static and dynamic power consumption. Static power consumption is a result of leakage currents in an IC. Dynamic power consumption is a result of short circuit currents and charging and discharging of capacitances in an IC. Dynamic power consumption is proportional to the switching activity (α), total capacitance (C_L), supply voltage (V_{DD}), operating frequency (f) and short circuit current (I_{SC}) as shown in the following equation. The power consumption due to charging and discharging of capacitances is the dominant component of dynamic power consumption and it can be reduced either by decreasing switching activity, capacitance, supply voltage or frequency.

$$P_{dyn} \approx \alpha_{0\to1} C_L V_{DD}^2 f + I_{SC} V_{DD} f \quad (1.1)$$

In this thesis, we focused on decreasing the power consumption of FPGA implementations of H.264 video compression hardware by reducing switching activity using Register Transfer Level (RTL) low power techniques such as clock gating [8,9,10], glitch reduction [11,12] and computational complexity reduction [13,14,15].

The power consumption of a digital hardware implementation on a Xilinx FPGA is estimated using Xilinx XPower tool. Since the switching activity is input pattern dependent, in order to estimate the dynamic power consumption, timing simulation of the placed and routed netlist of that hardware implementation is done for several input patterns using Mentor Graphics ModelSim SE 6.1c and the signal activities are stored in a Value Change Dump (VCD) file. This VCD file is used for estimating the power consumption of that hardware using Xilinx XPower tool.

1.3 Thesis Contributions

H.264 standard is expected to be used in many applications in the near future. Therefore, in the last few years, hardware architectures for implementing H.264 encoders and decoders for portable devices, digital TV and DVD recorder applications are started to be developed in both academia and industry. However, since H.264 standard has been recently developed, there are a small number of publications in the literature about designing hardware architectures for H.264 standard [16,17,18,19,20,21]. There are even fewer publications in the literature about low power hardware architectures for H.264 standard [22,23,24].

In this thesis, we proposed low power hardware designs for DBF, intra prediction and intra mode decision parts of an H.264 video encoder and performed detailed power consumption analysis of FPGA implementations of these hardware designs. We also measured the power consumptions of DBF hardware implementations on a Xilinx Virtex II FPGA and there is a good match between estimated and measured power consumption results. We applied several RTL low power techniques such as clock gating and glitch

reduction to these designs and quantified their impact on the power consumption of the FPGA implementations of these designs. We proposed novel computational complexity and power reduction techniques for DBF, intra prediction and intra mode decision parts of an H.264 video encoder. We quantified the computation reductions achieved by the proposed techniques using H.264 Joint Model (JM) software encoder version 14.0. We applied these techniques to proposed hardware designs and quantified their impact on the power consumption of the FPGA implementations of these designs.

We propose two efficient and low power H.264 DBF hardware implementations that can be used as part of an H.264 video encoder or decoder for portable applications [25]. The first implementation (DBF_4x4) starts filtering the available edges as soon as a new 4x4 block is ready by using a novel edge filtering order to overlap the execution of DBF module with other modules in the H.264 encoder/decoder. Overlapping the execution of DBF hardware with the execution of the other modules in the H.264 encoder/decoder improves the performance of the H.264 encoder/decoder. The second implementation (DBF_16x16) starts filtering the available edges after a new 16x16 MB is ready.

Both DBF hardware architectures are implemented in Verilog HDL and both implementations are synthesized to 0.18 μm UMC standard cell library. Both DBF implementations can work at 200 MHz and they can process 30 VGA (640×480) frames per second. DBF_4×4 and DBF_16×16 hardware implementations, excluding on-chip memories, are synthesized to 7.4 K and 5.3 K gates respectively. These gate counts are the lowest among the H.264 DBF hardware implementations presented in the literature. DBF_16x16 has 36% less power consumption than DBF_4x4 on a Xilinx Virtex II FPGA on an Arm Versatile PB926EJ-S development board. Therefore, DBF_4×4 hardware can be used in an H.264 encoder or decoder for which the performance is more important, whereas DBF_16×16 hardware can be used in an H.264 encoder or decoder for which the power consumption is more important.

We propose a novel computational complexity and power reduction technique for H.264 DBF algorithm. This technique avoids unnecessary calculations in DBF algorithm by exploiting spatial redundancy present in the pixels that will be filtered by DBF algorithm and therefore reduces the power consumption of H.264 DBF hardware significantly. If some or all of the pixels that will be filtered are equal, H.264 DBF

equations simplify significantly. Since the proposed technique uses the subtraction operations performed before the filtering process to determine whether the pixels that will be filtered are equal or not, the equality of the pixels are determined with a very small overhead. By exploiting the equality of the pixels, the proposed technique reduces the amount of addition and shift operations performed by H.264 DBF up to 39% and 50% respectively with a small comparison overhead.

We propose a novel technique for reducing the amount of computations performed by H.264 intra prediction algorithm and therefore reducing the power consumption of H.264 intra prediction hardware significantly without any PSNR and bit rate loss. The proposed technique performs a small number of comparisons among neighboring pixels of the current block before the intra prediction process. If the neighboring pixels of the current block are equal, the prediction equations of H.264 intra prediction modes simplify significantly for this block. By exploiting the equality of the neighboring pixels, the proposed technique reduces the amount of computations performed by 4x4 luminance, 16x16 luminance, and 8x8 chrominance prediction modes up to 60%, 28%, and 68% respectively with a small comparison overhead. We also implemented an efficient 4x4 intra prediction hardware including the proposed technique using Verilog HDL. We quantified the impact of the proposed technique on the power consumption of this hardware on a Xilinx Virtex II FPGA using Xilinx XPower, and it reduced the power consumption of this hardware up to 18.6% [26].

We propose a novel computational complexity reduction technique for H.264 intra mode decision. The proposed technique exploits the fixed prediction block patterns of intra prediction modes and the distribution property of Hadamard transform. The proposed technique reduces the computational complexity of Sum of Absolute Transformed Difference (SATD) based intra 4x4, intra 16x16 and intra 8x8 mode decisions by 46%, 64% and 62% respectively without any PSNR loss. In addition, it avoids the calculation of intra 16x16 and intra 8x8 plane prediction modes by slightly modifying SATD criterion used in H.264 reference software (JM) which slightly impacts the coding efficiency. It doesn't affect the PSNR for some videos, it increases the PSNR slightly for some videos and it decreases the PSNR slightly for some videos. Since plane mode is the most computationally intensive 16x16 and 8x8 prediction mode, avoiding plane mode

calculations reduces the computational complexity of 16x16 and 8x8 intra prediction algorithm by 80%.

1.4 Thesis Organization

The rest of the thesis is organized as follows.

Chapter II, first, gives an overview of H.264 DBF algorithm. It, then, presents two efficient low power H.264 DBF hardware implementations and the impact of several RTL low power techniques on these hardware implementations. A novel computational complexity and power reduction technique for H.264 DBF algorithm is also presented in this chapter.

Chapter III, first, gives an overview of H.264 intra prediction algorithm. It, then, presents a novel computational complexity and power reduction technique for H.264 intra prediction. An efficient H.264 intra prediction hardware implementing this technique and its power consumption analysis is also presented in this chapter.

Chapter IV, first, gives an overview of H.264 intra mode decision algorithm. It, then, presents a novel computational complexity and power reduction technique for H.264 intra mode decision.

Chapter V presents conclusions and future work.

CHAPTER II

LOW POWER H.264 DEBLOCKING FILTER HARDWARE DESIGNS

The video compression efficiency achieved in H.264 standard is not a result of any single feature but rather a combination of a number of encoding tools. As it is shown in the top level block diagrams of an H.264 encoder and decoder in Figure 1.1 and 1.2., one of these tools is the adaptive DBF algorithm [1,3,4,27]. DBF is applied to each MB, a 16×16 pixel array, after inverse quantization and inverse transform. DBF improves the visual quality of decoded frames by reducing the visually disturbing blocking artifacts and discontinuities in a frame due to coarse quantization of MBs and motion compensated prediction. Since the filtered frame is used as a reference frame for motion-compensated prediction of future frames, DBF also increases coding efficiency resulting in bit rate savings [27].

The DBF algorithm used in H.264 standard is more complex than the DBF algorithms used in previous video compression standards. First of all, H.264 DBF algorithm is highly adaptive and applied to each edge of all the 4×4 luma and chroma blocks in a MB. Second, it can update 3 pixels in each direction that the filtering takes place. Third, in order to decide whether the DBF will be applied to an edge, the related pixels in the current and neighboring 4×4 blocks must be read from memory and processed.

Because of these complexities, the DBF algorithm can easily account for one-third of the computational complexity of an H.264 video decoder [27].

In this thesis, we propose two efficient and low power H.264 DBF hardware implementations that can be used as part of an H.264 video encoder or decoder for portable applications [28,29]. The first implementation (DBF_4×4) starts filtering the available edges as soon as a new 4x4 block is ready by using a novel edge filtering order. The second implementation (DBF_16×16) starts filtering the available edges after a new 16x16 MB is ready.

The execution of DBF_4×4 hardware can be overlapped with the execution of the other modules in an H.264 encoder/decoder much more than the execution of DBF_16×16 hardware can be overlapped with the execution of the other modules. Overlapping the execution of DBF hardware with the execution of the other modules in the H.264 encoder/decoder improves the performance of the H.264 encoder/decoder. However, because of the nature of the DBF algorithm, control unit and address generation of DBF_16×16 hardware is simpler. Therefore, DBF_16x16 hardware has less area and consumes less power than DBF_4×4 hardware.

Both DBF hardware architectures are implemented in Verilog HDL and both implementations are verified to work correctly in a Xilinx Virtex II FPGA on Arm Versatile PB926EJ-S Development Board. Both hardware implementations can work at 67 MHz on a Xilinx Virtex II FPGA and they can process 30 CIF (352x288) frames per second. Both hardware implementations can work at 200 MHz when synthesized to 0.18 μm UMC standard cell library and they can process 30 VGA (640×480) frames per second. DBF_4×4 and DBF_16×16 hardware implementations, excluding on-chip memories, are synthesized to 7.4 K and 5.3 K gates respectively.

The power consumptions of both DBF hardware implementations on a Xilinx Virtex II FPGA are estimated using Xilinx XPower tool. DBF_16×16 has 36% less power consumption than DBF_4×4. The power consumption of DBF_16×16 is further reduced by 28% by using block SelectRAMs instead of distributed SelectRAMs and by 3.1% by using clock gating. Furthermore, power consumption of DBF datapath is reduced by 13% using clock gating and by 4.7% using glitch reduction technique. The power consumptions of

both implementations on a Xilinx Virtex II FPGA are also measured and the measurement results are consistent with the estimation results.

Therefore, these two H.264 DBF hardware implementations can be used as part of H.264 video encoders or decoders for portable applications with different power-performance requirements. DBF_4×4 hardware can be used in an H.264 encoder or decoder for which the performance is more important, whereas DBF_16×16 hardware can be used in an H.264 encoder or decoder for which the power consumption is more important.

Several hardware architectures for real-time implementation of H.264 DBF algorithm are presented in the literature [16,30,31,32,33]. These architectures achieve high performance at the expense of high hardware cost. The gate counts of our H.264 DBF hardware implementations are the lowest among these H.264 DBF hardware implementations. We could not compare power consumptions of our DBF hardware implementations with these DBF hardware implementations, since the power consumptions of these DBF hardware implementations are not reported.

We also propose a novel computational complexity and power reduction technique for H.264 DBF algorithm. This technique avoids unnecessary calculations in DBF algorithm by exploiting spatial redundancy present in the pixels that will be filtered by DBF algorithm and therefore reduces the power consumption of H.264 DBF hardware significantly. If some or all of the pixels that will be filtered are equal, H.264 DBF equations simplify significantly. Since the proposed technique uses the subtraction operations performed before the filtering process to determine whether the pixels that will be filtered are equal or not, the equality of the pixels are determined with a very small overhead. By exploiting the equality of the pixels, the proposed technique reduces the amount of addition and shift operations performed by H.264 DBF up to 39% and 50% respectively with a small comparison overhead.

2.1 Overview of H.264 Adaptive Deblocking Filter Algorithm

H.264 adaptive DBF algorithm removes visually disturbing block boundaries created by coarse quantization of MBs and motion compensated prediction. MBs in a frame are filtered in raster scan order. Filtering is applied to each edge of all the 4x4 luma and chroma blocks in a MB as shown in Figure 2.1. The vertical 4×4 block edges in a MB are filtered before the horizontal 4×4 block edges in the order shown in Figure 2.2 [1].

DBF algorithm for one row/column of a vertical/horizontal edge is shown in Figure 2.3 [27]. There are several conditions that determine whether a 4×4 block edge will be filtered or not. There are additional conditions that determine the strength of the filtering for the 4x4 block edges that will be filtered. As shown in the Figure 2.3, boundary strength (BS) parameter, α and β threshold values and the values of the pixels in the edge determine the outcomes of these conditions, and the values of up to 3 pixels on both sides of an edge can be changed depending on the outcomes of these conditions. H.264 DBF algorithm can be divided into eight modes based on the outcomes of these conditions as shown in Figure 2.3.

H.264 DBF algorithm is adaptive in three levels; slice level, edge level and sample level [1,27]. Slice level adaptivity is used to adjust the filtering strength in a slice to the characteristics of the slice data. The filtering strength in a slice is adjusted by encoder using the offset-a and offset-b parameters. The α and β threshold values that determine whether a 4x4 block edge will be filtered or not and how strong the filtering will be for an edge are a function of quantization parameter (QP) and these two offset parameters.

Edge level adaptivity is used to adjust the filtering strength for an edge to the characteristics of that edge. The filtering strength for an edge is adjusted using the Boundary Strength (BS) parameter. Every edge is assigned a BS value depending on the coding modes and conditions of the 4x4 blocks. The conditions used for determining the BS value for an edge between two neighboring 4x4 blocks are summarized in Table 2.1 [1,27]. The strength of the filtering done for an edge is proportional to its BS value. No filtering is done for the edges with a BS value of zero, whereas strongest filtering is done for the edges with a BS value of four.

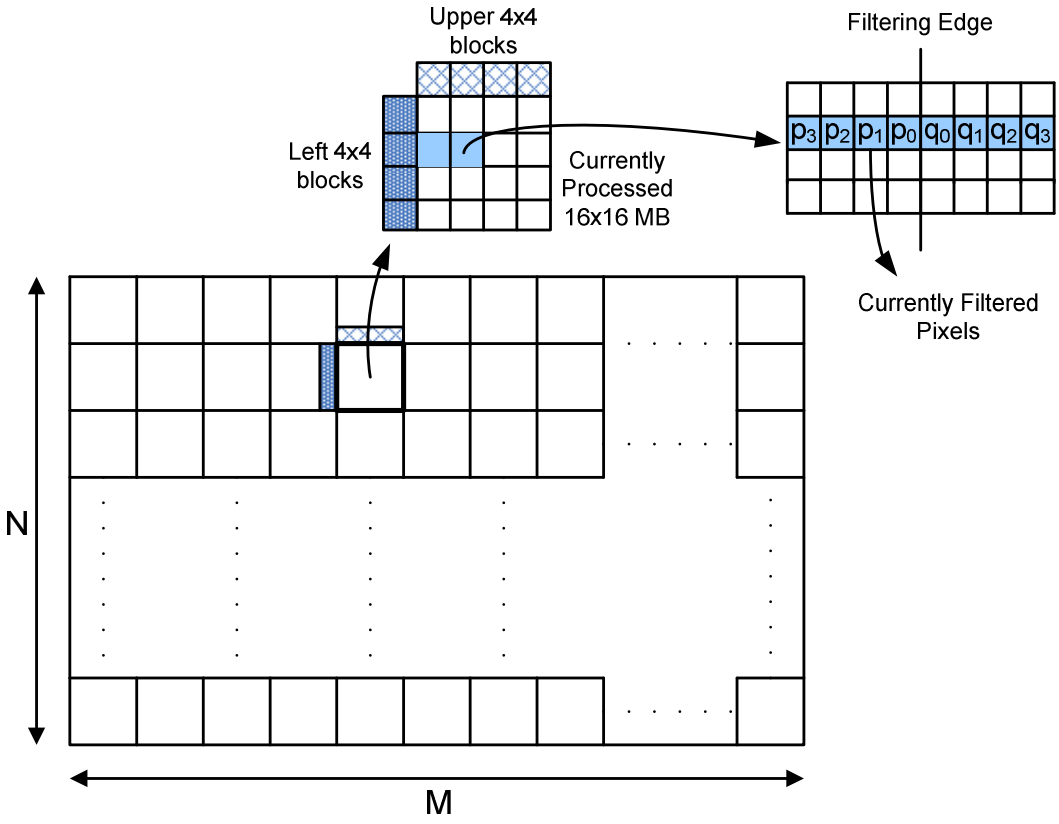

Figure 2.1 Illustration of H.264 DBF Algorithm

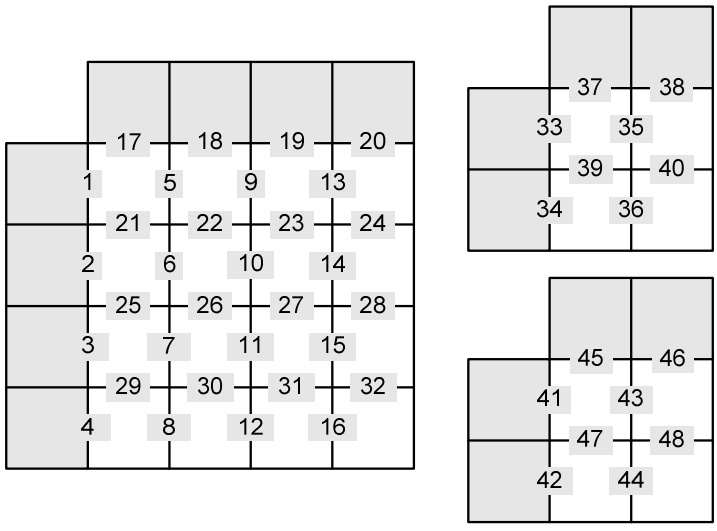

Figure 2.2 Edge Filtering Order in a MB Specified in H.264 Standard

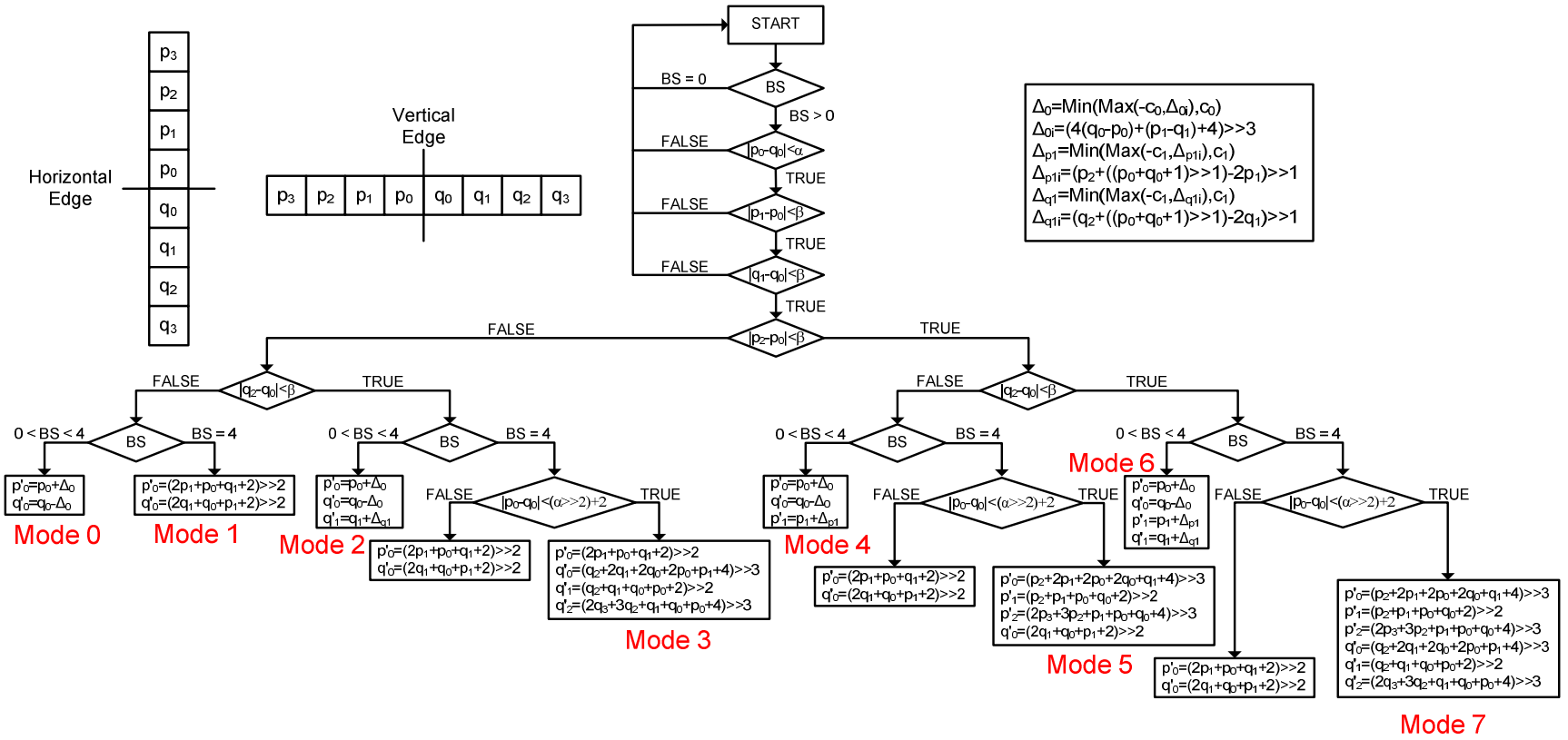

Figure 2.3 H.264 Deblocking Filter Algorithm

Sample level adaptivity is used to adjust the filtering strength for an edge to the characteristics of the pixels in that edge in order to distinguish the true edges from those created by quantization. The filtering strength for an edge is therefore determined by comparing pixel gradients in that edge with α and β threshold values for that edge.

Table 2.1 Conditions that Determine BS

Coding Modes and Conditions	**BS**
One of the blocks is intra and the edge is a macroblock edge	4
One of the blocks is intra	3
One of the blocks has coded residuals	2
Difference of block motion ≥ 1 luma sample distance and Motion compensation from different reference frames	1
Else	0

2.2 Proposed Hardware Architectures

The block diagram of proposed DBF hardware is shown in Figure 2.4. Both DBF hardware, DBF_4x4 and DBF_16x16, include a datapath, a control unit, one 384×8 register file and two dual-port internal SRAMs to store partially filtered pixels.

As it can be seen from Figure 1.1 and 1.2, in an H.264 encoder and decoder, DBF module gets its input, reconstructed MB, from Inverse Transform/Quant (IT/IQ) module. IT/IQ module generates the reconstructed MB, one 4×4 block at a time. A 384×8 input buffer, IBUF, is used between IT/IQ and DBF modules to store one reconstructed MB (256 luminance pixels + 128 chrominance pixels) generated by IT/IQ module.

The datapaths of both DBF hardware implementations are the same. The DBF datapath is implemented as a two stage pipeline to improve the clock frequency and throughput. As shown in Figure 2.5, the first pipeline stage includes one 12-bit adder and two shifters to perform numerical calculations like multiplication and addition. The second pipeline stage includes one 12-bit comparator, several two's complementers and multiplexers to determine conditional branch results.

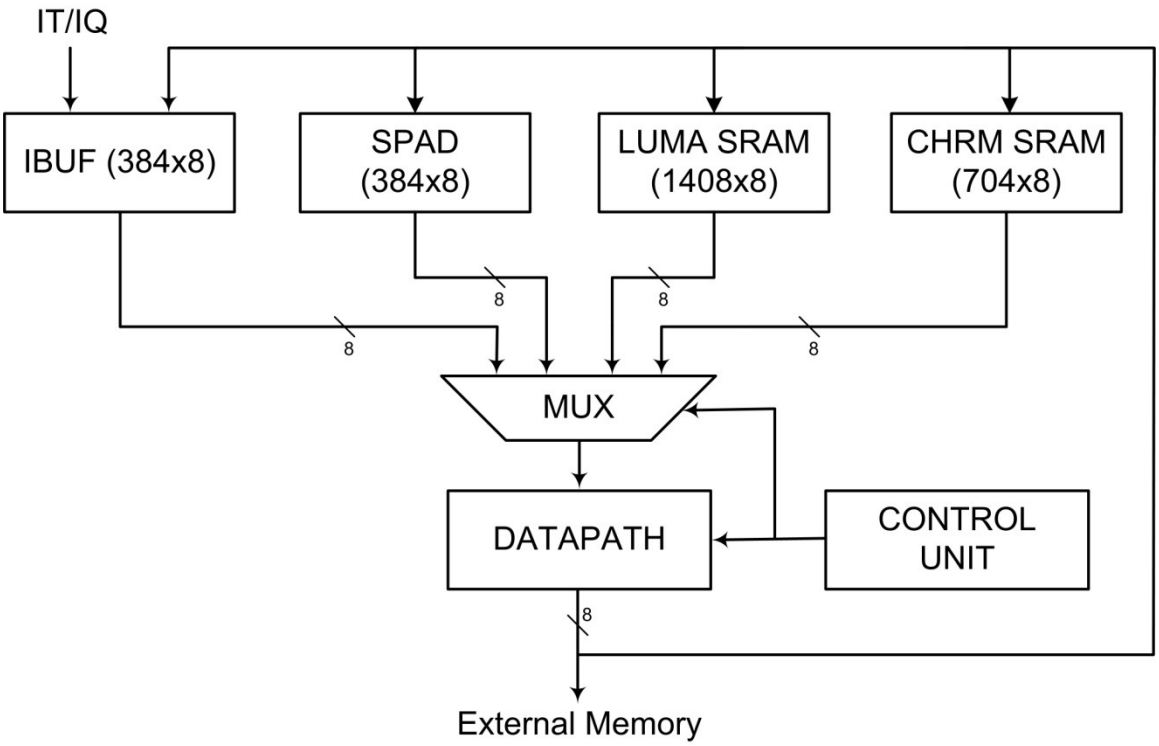

Figure 2.4 Proposed DBF Hardware Block Diagram

DBF_4×4 hardware starts filtering available edges as soon as a new 4x4 block is ready by using a novel edge filtering order we proposed. There are 16 4×4 blocks in a MB and they are processed by IT/IQ module in the order shown in Figure 2.6 [1]. The proposed novel edge filtering order for a MB is shown in Figure 2.7.

The idea behind this novel order is that after a new 4×4 block is ready start filtering the edges that can be filtered without violating the filtering order specified in the H.264 standard [1]. After the first 4×4 block in a MB is processed and loaded into IBUF by IT/IQ module, DBF module can only filter edge 1 without violating the filtering order specified in H.264 standard. After the second 4×4 block is loaded into IBUF, DBF module can filter edge 2 and edge 3, and so on.

The execution of DBF_4×4 hardware can be overlapped with the execution of the other modules in an H.264 encoder/decoder much more than the execution of DBF_16×16 hardware can be overlapped with the execution of the other modules. Overlapping the execution of DBF hardware with the execution of the other modules in the H.264 encoder/decoder improves the performance of the H.264 encoder/decoder.

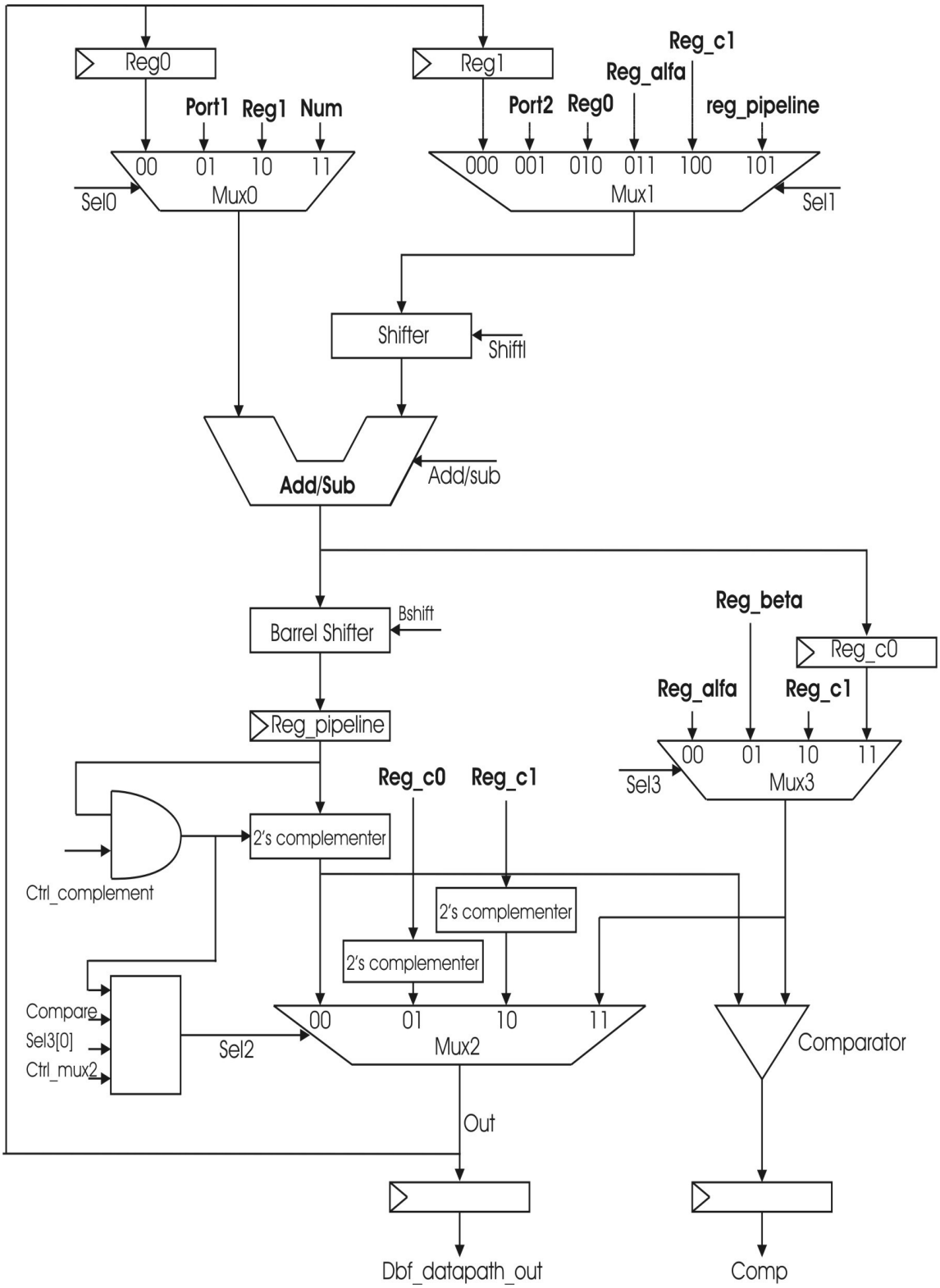

Figure 2.5 Proposed DBF Hardware Datapath

DBF_16x16 hardware uses the same edge filtering order specified in the H.264 standard. Since this edge filtering order has a regular pattern, control unit and address generation of DBF_16×16 hardware is simpler. Therefore, DBF_16x16 hardware has less area and consumes less power than DBF_4×4 hardware.

There are three on-chip memories in both DBF hardware implementations. A 384×8 register file, SPAD, is used to store partially filtered pixels in a 16×16 MB until all the edges in this MB are fully filtered. Since SPAD is the most frequently accessed memory in the DBF hardware, we reduced the number of access to SPAD by adding two registers in datapath to store some of the temporary results.

In the M×N frame shown in Figure 2.8, squares represent 16x16 MBs and each MB has sixteen 4×4 blocks. In order to filter a MB, its upper and left neighboring 4×4 blocks, shown as shaded small squares in Figure 2.8, should be available. Since our DBF hardware gets its input MB from IT/IQ hardware and it does not have access to off-chip frame memory, the upper 4×4 blocks of all MBs in a row of the frame, shown as lightly shaded small squares in Figure 2.8, and the left 4×4 blocks of the current MB, shown as darkly shaded small squares in Figure 2.8, have to be stored in on-chip local memory.

The left 4x4 blocks are stored in SPAD. The upper 4x4 luminance and chrominance blocks are stored in the 1408×8 LUMA SRAM and 704×8 CHROMA SRAM memories shown in Figure 2.4 respectively. For a CIF size video, 4×352×8 = 1408×8 memory is needed for storing upper 4x4 luminance blocks and 4x88x8+4x88x8 = 704×8 memory is needed for storing upper 4x4 chrominance blocks.

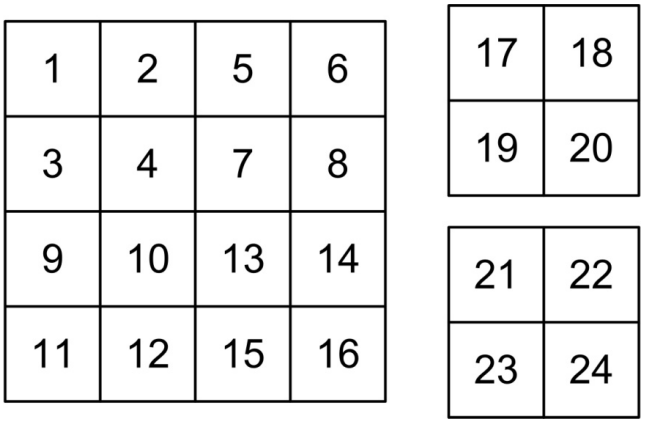

Figure 2.6 Processing Order of 4×4 Blocks by IT/IQ Module

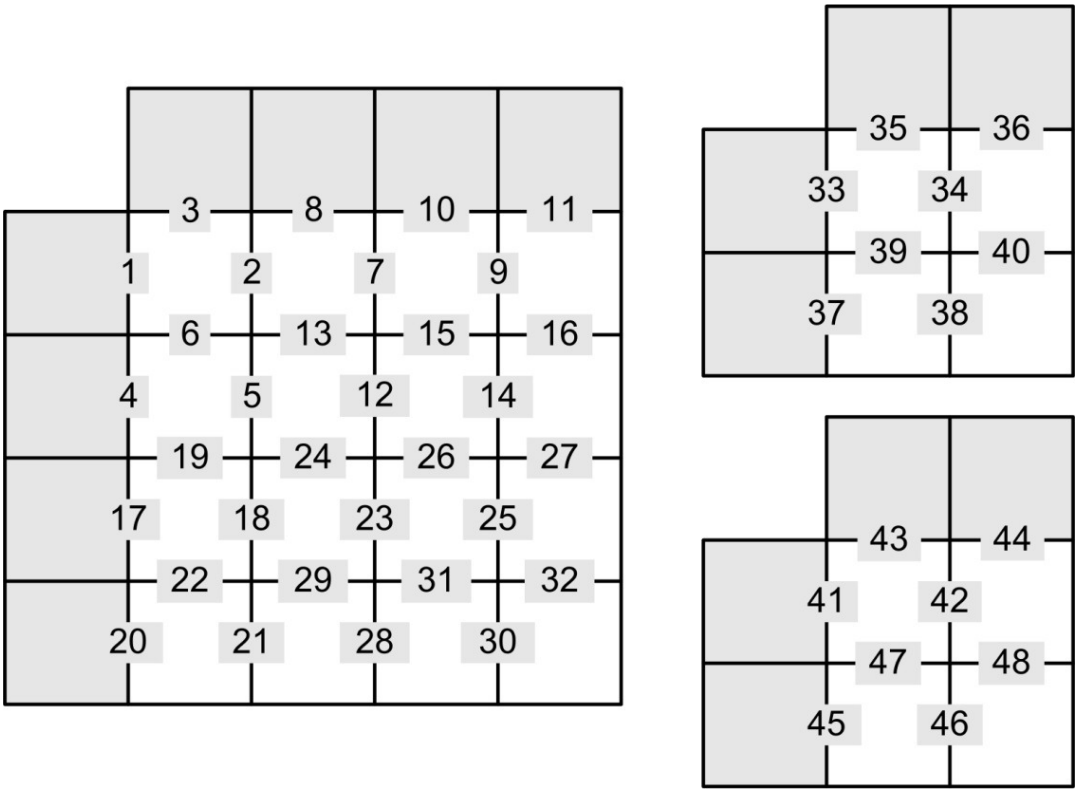

Figure 2.7 Proposed Novel Edge Filtering Order

The DBF hardware implementations in the literature use off-chip memory for storing these neighboring 4×4 blocks [16,30,31,32,33]. Since accessing on-chip SRAMs consumes less power than accessing off-chip memory, using on-chip SRAMs for storing these neighboring 4×4 blocks reduces power consumption of our DBF hardware implementations.

Transpose pixel arrays are used to transpose the horizontal aligned pixels into vertical aligned positions in several DBF hardware implementations in the literature [30,31,32,33]. Since the memories used in our DBF hardware implementations are 8-bit wide, any pixel stored in memory can directly be accessed, therefore, there is no need for transposing one row of eight pixel data into one column of eight pixel data. Not using a transpose pixel array reduces area of our DBF hardware implementations.

The edges 1, 2, 3, 4, 17, 18, 19, 20, 33, 34, 37, 38, 41, 42, 45 and 46 of a MB shown in Figure 2.3 are not filtered if this MB is located in the upper or the left frame boundary. This is not the case for the MBs located inside the frame. This causes an

irregularity and, therefore, increases the complexity of the control unit. In order to avoid this irregularity and therefore simplify the control unit, we have extended the frames at the upper and left frame boundaries for 4 pixels in depth as shown in Figure 2.8. We assigned zero to these pixels and assigned zero to the BS values of these edges in order to avoid filtering these edges without causing an irregularity in the control unit.

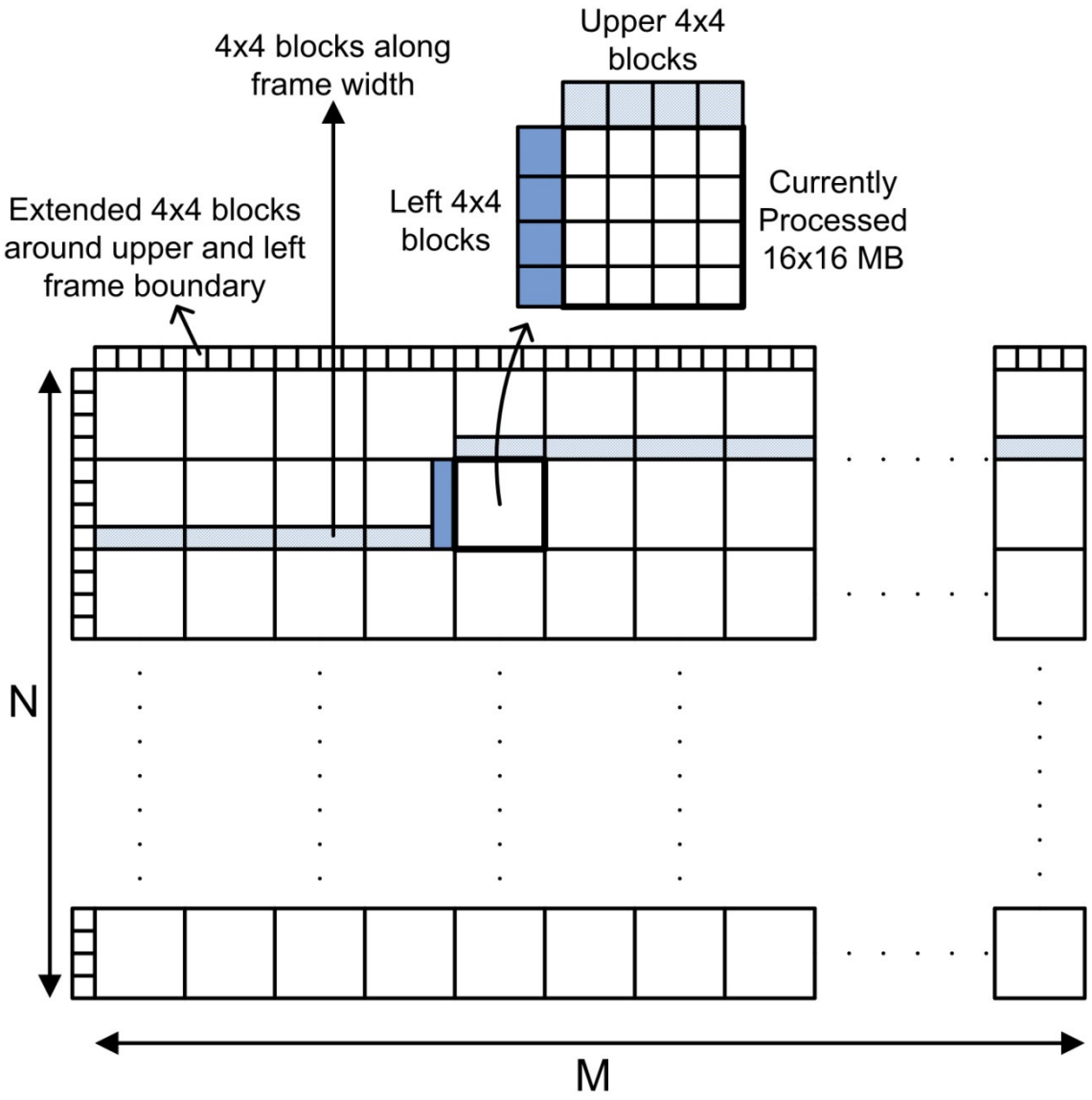

Figure 2.8 4x4 Blocks Stored in LUMA and CHROMA SRAMs

2.3 Implementation Results

The proposed DBF hardware architectures are implemented in Verilog HDL. The implementations are verified with RTL simulations using Mentor Graphics ModelSim SE. RTL simulation results matched the results of a MATLAB model of the H.264 adaptive DBF algorithm.

The Verilog RTL designs are synthesized to a 2V8000ff1157 Xilinx Virtex II FPGA with speed grade 5 using Mentor Graphics Precision RTL 2005b. The resulting netlists are placed and routed to the same FPGA using Xilinx ISE 8.2i.

DBF_4x4 hardware works at 67 MHz and it takes 5248 clock cycles in the worst-case for DBF_4×4 hardware to process a MB. The FPGA implementation can process a CIF (352x288) frame in 30.9 ms (396 MB * 5248 clock cycles per MB * 14.9 ns clock cycle = 30.9 ms). Therefore, it can process 1000/30.9 = 32 CIF frames per second.

DBF_16x16 hardware works at 72 MHz and it takes 5376 clock cycles in the worst-case for DBF_16×16 hardware to process a MB. The FPGA implementation can process a CIF (352x288) frame in 29.6 ms (396 MB * 5376 clock cycles per MB * 13.9 ns clock cycle = 29.6 ms). Therefore, it can process 1000/29.6 = 33 CIF frames per second.

FPGA resource usages of both DBF implementations including on chip memories are shown in Table 2.2. LUMA SRAM and CHROMA SRAM are implemented as dual-port block SelectRAMs. SPAD and IBUF are implemented as dual-port distributed SelectRAMs.

Both DBF hardware implementations are synthesized to 0.18 μm UMC standard cell library. Both hardware implementations can work at 200 MHz and they can process 30 VGA (640x480) frames per second. DBF_4×4 and DBF_16×16 hardware implementations, excluding on-chip memories, are synthesized to 7.4 K and 5.3 K gates respectively.

As shown in Table 2.3, these gate counts are the lowest among the H.264 DBF hardware implementations presented in the literature [16,30,31,32,33]. These hardware implementations achieve high performance at the expense of high hardware cost. We

achieved real-time performance by only using one 12-bit adder, one 12-bit comparator, a few shifters, and a number of multiplexers in our datapath.

Table 2.2 FPGA Resource Usage and Clock Frequency After Place and Route

Resource	DBF_4x4 Hardware	DBF_16x16 Hardware
Function Generators	4074	2537
DFFs	335	306
Block SelectRAMs	2	2
Clock Frequency	67 MHz	72 MHz

Table 2.3 DBF Hardware Comparison

Category	[7]	[10]	[12]	[31]	[32]	DBF_16x16
Gate Count	20.6 K	9.2 K	11.8 K	14.8 K	7.5 K	5.3 K
Technology	0.25 μ Artisan	0.18 μ UMC	0.18 μ UMC	0.18 μ UMC	0.13 μ TSMC	0.18 μ UMC
On-chip Memory Size	160x32	80x32	140x32	160x32	32x32	384x8

2.4 ARM Versatile / PB926EJ-S Development Board Implementation

Both DBF hardware implementations are verified to work correctly in the ARM Versatile PB926EJ-S development environment shown in Figure 2.9. As shown in the figure, the development environment consists of a PC connected to ARM Versatile PB926EJ-S board through ARM Multi-ICE, a logic tile mounted on the Versatile PB926EJ-S baseboard and a color LCD panel [34].

PC is used to create the bit stream that will be loaded into the 8-million-gate Xilinx Virtex II FPGA on the logic tile which can be configured to implement custom-designed logic. ARM Multi-ICE is used for communicating between PC and Arm Versatile board, and AXD Debugger from ARM Developer Suite is used for debugging the system. The Color LCD panel is used to display images for visual verification.

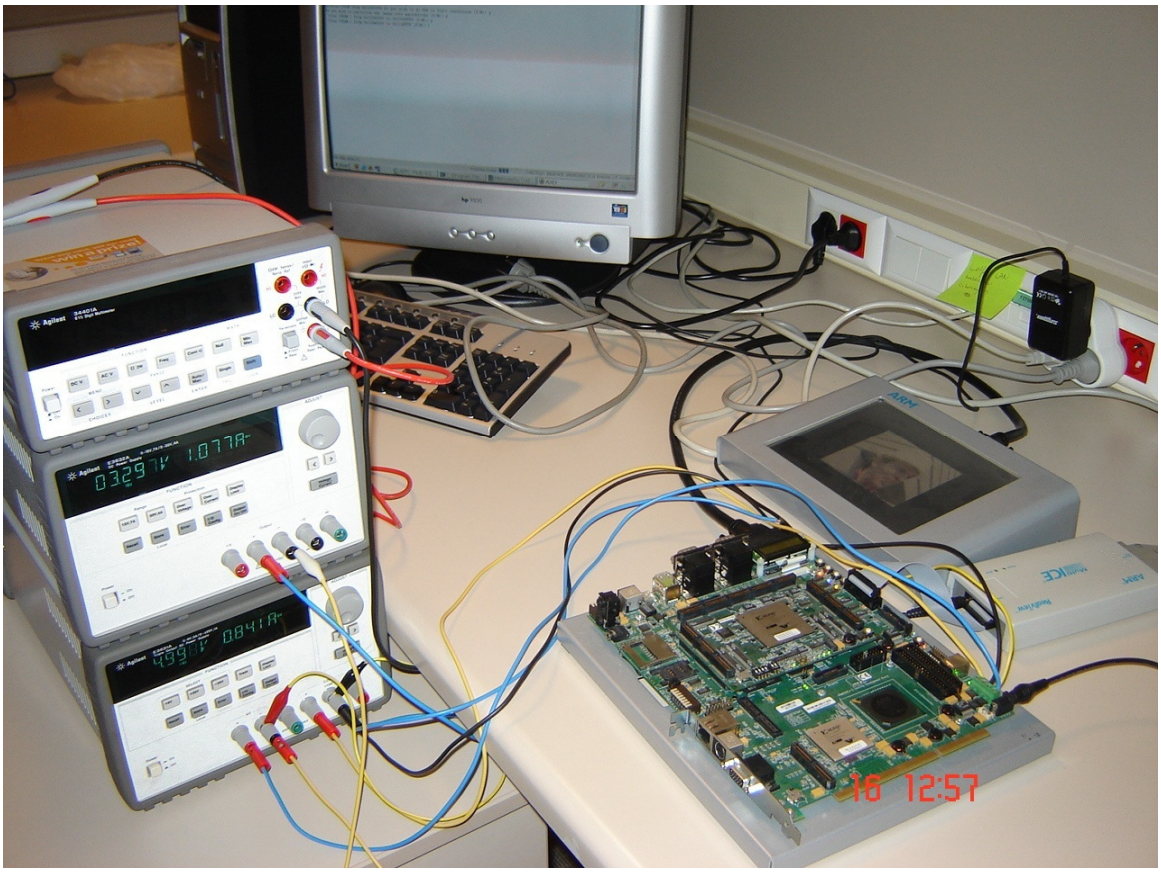

Figure 2.9 ARM Versatile / PB926EJ-S Development Environment and Power Measurement Setup

As shown in Figure 2.10, an AHB bus interface is designed and integrated into DBF hardware in order to communicate with ARM processor and external SRAM through AHB bus, and DBF Hardware is integrated into the FPGA on the logic tile as a master of the AHB S bus.

A video frame is loaded into SRAM located on the board from PC using software. This video frame is used as an input to DBF hardware running on the FPGA. DBF hardware applies the H.264 DBF algorithm to this video frame and writes the resulting frame back to SRAM. The resulting video frame is shown on the color LCD panel.

An unfiltered video frame and the same video frame filtered by H.264 DBF hardware running in the FPGA on the logic tile are shown in Figure 2.11 and Figure 2.12. As it can be seen from the figure, some of the blocking artifacts in the unfiltered video frame are reduced and some of them are totally removed.

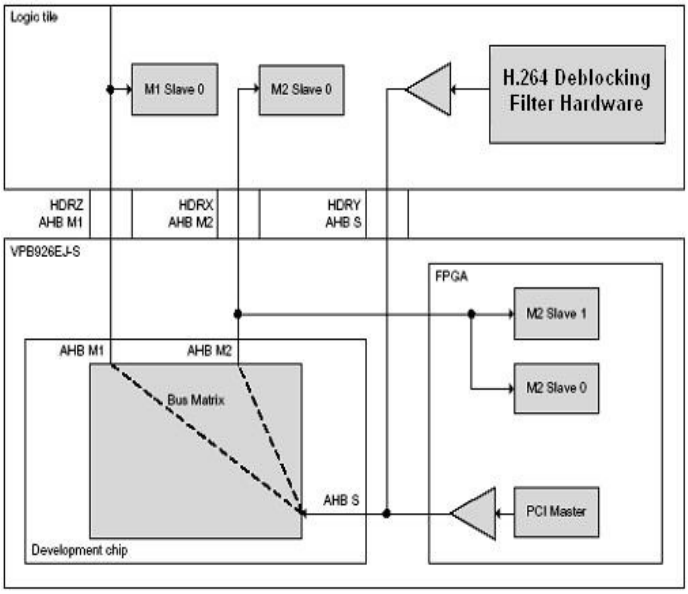

Figure 2.10 Integration of Deblocking Filter Hardware into ARM Versatile Board

2.5 Power Consumption Results

The power consumptions of both DBF hardware implementations on a Xilinx Virtex II FPGA are estimated using Xilinx XPower tool. In order to estimate the power consumption of a DBF hardware implementation, timing simulation of the placed and routed netlist of that DBF hardware implementation is done using Mentor Graphics ModelSim SE for one frame of Foreman video sequence and the signal activities are stored in a VCD file. This VCD file is used for estimating the power consumption of that DBF hardware implementation using Xilinx XPower tool.

The power consumptions of both DBF hardware implementations on a Xilinx Virtex II FPGA at 50 MHz are shown in Table 2.4. Since DBF hardware will be used as part of an H.264 encoder or decoder only internal power consumption is considered and input and output power consumptions are ignored. To make a fair comparison between the power consumptions of the two DBF hardware implementations, we used same number of distributed SelectRAMs and block SelectRAMs for both implementations. As shown in the table, DBF_16x16 hardware has 36% less power consumption than DBF_4x4 hardware.

Figure 2.11 Unfiltered Video Frame

Figure 2.12 The Same Frame Filtered by H.264 Deblocking Filter Algorithm

Table 2.4 Power Consumption of DBF Hardware Implementations at 50 MHz

Category	DBF_4×4	DBF_16×16
Clock	56.37 mW	50.36 mW
Logic	145.65 mW	52.47 mW
Signal	83.56 mW	79.39 mW
Total	285.58 mW	182.22 mW

Table 2.5 Power Consumption Comparison of Block SelectRAM and Distributed SelectRAM

Category	Maximum Switching Activity	Minimum Switching Activity
Block SelectRAM	12.02 mW	3.7 mW
Distributed SelectRAM	67.54 mW	21.67 mW

The power consumption of a DBF hardware implementation can be divided into three main categories; signal power, logic power and clock power. Signal power is the power dissipated in routing tracks between logic blocks. A significant amount of power is dissipated in routing tracks. It accounts for 29% of total power consumption of DBF_4×4 hardware and 43% of total power consumption of DBF_16×16 hardware. Logic power is the amount of power dissipated in the parts where computations take place. Clock power is due to clock tree used in the FPGA. Since there is less number of flip-flops in DBF_16×16 hardware in comparison with the DBF_4×4 hardware, the clock power of DBF_16x16 hardware is less than the clock power of DBF_4x4 hardware.

Xilinx Virtex-II FPGAs have block SelectRAM and distributed SelectRAM memories. In DBF hardware implementations, we used both block SelectRAMs and distributed SelectRAMs as local memories for storing intermediate results. We, therefore, characterized the power consumptions of block SelectRAMs and distributed SelectRAMs using Xilinx XPower tool for the cases when there is maximum switching activity and minimum switching activity in the RAMs, and the results are shown in Table 2.5.

The results show that the power consumption of a distributed SelectRAM is much more than the power consumption of a block SelectRAM. This is because a distributed

SelectRAM is formed by look up tables in Configurable Logic Blocks (CLBs) and this causes the memory to be distributed in the FPGA and have long interconnects. On the other hand, a block SelectRAM is a carefully designed and optimized full-custom SRAM.

Therefore, we decided to use only block SelectRAMs in DBF_16×16 hardware. Using block SelectRAMs instead of distributed SelectRAMs in DBF_16×16 hardware provided additional 28% power reduction and total power consumption of DBF_16x16 hardware is reduced from 182.22 mW to 130.45 mW.

In addition, we applied clock gating and glitch reduction techniques to DBF datapath for reducing its power consumption. DBF datapath is two-stage pipelined. The first stage performs numerical calculations every clock cycle, but the second stage is not active for a considerable amount of clock cycles. Therefore, we turned off the second stage by clock gating when it is inactive. Table 2.6 shows the impact of clock gating on datapath power consumption. The datapath power consumption is reduced by 13% using clock gating.

Table 2.6 Impact of Clock Gating on Datapath Power Consumption

Category	**Datapath**	**Datapath with Clock Gating**
Clock	7.46 mW	6.71 mW
Logic	7.62 mW	5.88 mW
Signal	18.41 mW	16.56 mW
Total	33.49 mW	29.15 mW

Table 2.7 Impact of Glitch Reduction on Datapath Power Consumption

Category	**Datapath**	**Datapath without Glitches**	**Pipelined Datapath**
Clock	7.46 mW	7.37 mW	9,25 mW
Logic	7.62 mW	6.60 mW	6,07 mW
Signal	18.41 mW	16.47 mW	16,59 mW
Total	33.49 mW	30.44 mW	31,91 mW

Glitch is a spurious transition at a node within a single cycle before the node settles to the correct logic value. Unlike ASICs, in which signals can be routed using any

available silicon, FPGAs implement interconnects using fixed metal tracks and programmable switches. The relative scarcity of programmable switches often forces signals to take longer routes than would be seen in an ASIC. As a result, the potential for unequal delays among signals, and hence the creation of glitches, is more likely than that in an ASIC. Thus, reducing glitches by pipelining is an effective power reduction technique for FPGAs [12].

The impact of glitches on DBF datapath power consumption can be seen by simulating the datapath under zero delay model and analyzing its power consumption. The glitch free power consumption of DBF datapath is shown in Table 2.7. The glitch free power consumption shows the maximum power consumption reduction that can be obtained by reducing glitches. Table 2.7 shows the impact of reducing glitches by pipelining on datapath power consumption. We inserted two pipeline registers immediately before the inputs of the adder. This reduced the datapath power consumption by 4.7%. We therefore obtained 50% of maximum possible power reduction that can be obtained by reducing glitches.

We also measured the power consumptions of both DBF hardware implementations on a Xilinx Virtex II FPGA using the setup shown in Figure 2.9. Using this setup, we measured the average current before DBF hardware is running on the FPGA. We, then, measured the average current while DBF hardware is running on the FPGA at 34 MHz in a continuous loop. Since the FPGA on the logic tile is supplied with 3.3 V power supply, the power consumption of DBF hardware is calculated by multiplying the difference in average current with 3.3 V.

The power consumption measurement and estimation results are shown in Table 2.8. DBF_4x4 hardware used for these measurements and estimations has 3 distributed SelectRAMs and 2 block SelectRAMs, however, DBF_16×16 hardware used for these measurements and estimations has 5 block SelectRAMs. The power consumption measurement results are slightly larger than the power consumption estimation results. The difference between measured and estimated results is caused by the power consumed for reading the unfiltered MBs from and writing the filtered MBs to the SRAM on the logic tile through AHB bus which is not included in power consumption estimations.

Table 2.8 Power Consumption Estimations and Measurements of DBF_16×16 and DBF_4×4 Hardware at 34 MHz

DBF Hardware	Average Current without DBF	Average Current with DBF	Estimated Power	Measured Power
DBF_4×4	999 mA	1076 mA	220.6 mW	254.1 mW
DBF_16×16	1119 mA	1152 mA	89.7 mW	108.9 mW

2.6 A Novel Computational Complexity Reduction Technique for H.264 Deblocking Filter

In this section, we propose a novel computational complexity reduction technique which avoids unnecessary calculations in H.264 DBF and therefore reduces the power consumption of H.264 DBF hardware.

H.264 DBF algorithm is highly adaptive and eight different filtering equations are used based on BS, α and β parameters and pixel gradient as shown in Figure 2.3. Eight possible conditions, called modes, and the pixels used in filtering equations used in these modes are listed in Table 2.9. The filtering equations used in each mode are given in Figure 2.3. As it can be seen from filtering equations, H.264 DBF algorithm can be implemented using only addition and shift operations.

If some or all of the pixels used in filtering equations are equal, then the filtering equations either simplify significantly or become unnecessary. The filtering equations used in mode 6 are given in Table 2.10. For mode 6, if the pixels that will be filtered are all equal, Δ0, Δp1 and Δq1 become zero and filtering becomes unnecessary. In addition, if some of the pixels that will be filtered are equal, the filtering equations used in this mode simplify significantly. Table 2.11 shows such a case in which p1=p0=q0=q1=p. As shown in Tables 2.10 and 2.11, some or all of the calculations are avoided if the equality of the pixels that will be filtered is known before the filtering process.

Table 2.9 DBF Modes

BS	|p2-p0|<β	|q2-q0|<β	Pixels used in filtering equations	Mode
BS=4	False	False	p1, p0, q0, q1	1
	False	True	p1, p0, q0, q1, q2, q3	3
	True	False	p3, p2, p1, p0, q0, q1,	5
	True	True	p3, p2, p1, p0, q0, q1, q2, q3	7
4>BS>0	False	False	p1, p0, q0, q1	0
	False	True	p1, p0, q0, q1, q2	2
	True	False	p2, p1, p0, q0, q1	4
	True	True	p2, p1, p0, q0, q1, q2	6

Table 2.10 Equations for Mode 6 and their Simplified Versions when p2=p1=p0=q0=q1=q2

Equations for mode 6	**Simplified equations for mode 6 when p2=p1=p0=q0=q1=q2**
p'0 = p0+Δ0	p'0 = p0
q'0 = q0-Δ0	q'0 = q0
p'1 = p1+Δp1	p'1 = p1
q'1 = q1+Δq1	q'1 = q1
Δ0i = (4(q0-p0)+(p1-q1)+4)>>3	Δ0i = 0
Δ0 = Min(Max(-c0, Δ0i),c0)	Δ0 = 0
Δp1i = (p2+((p0+q0+1)>>1)-2p1)>>1	Δp1i = 0
Δp1 = Min(Max(-c1, Δp1i),c1)	Δp1 = 0
Δq1i = (q2+((p0+q0+1)>>1)-2q1)>>1	Δq1i = 0
Δq1 = Min(Max(-c1, Δq1i),c1)	Δq1 = 0

Table 2.11 Equations for Mode 6 and their Simplified Versions when p1=p0=q0=q1

Equations for mode 6	**Simplified equations for mode 6 when p1=p0=q0=q1**
p'0 = p0+Δ0	p'0 = p0
q'0 = q0-Δ0	q'0 = q0
p'1 = p1+Δp1	p'1 = p1+Δp1
q'1 = q1+Δq1	q'1 = q1+Δq1
Δ0i = (4(q0-p0)+(p1-q1)+4)>>3	Δ0i = 0
Δ0 = Min(Max(-c0, Δ0i),c0)	Δ0 = 0
Δp1i = (p2+((p0+q0+1)>>1)-2p1)>>1	Δp1i = (p2-p) >>1
Δp1 = Min(Max(-c1, Δp1i),c1)	Δp1 = Min(Max(-c1, Δp1i),c1)
Δq1i = (q2+((p0+q0+1)>>1)-2q1)>>1	Δq1i = (q2-p) >>1
Δq1 = Min(Max(-c1, Δq1i),c1)	Δq1 = Min(Max(-c1, Δq1i),c1)

In order to avoid the overhead for determining the equality of the pixels that will be filtered, we propose to use the subtraction operations performed in conditional branches of DBF algorithm as shown in Figure 2.3. These conditional branches include the five subtraction operations shown in (2.1)-(2.5). If two pixels are equal, their difference will be equal to zero. Therefore, by only checking the results of these five subtraction operations, we can determine the equality of the pixels that will be filtered without performing additional comparison operations. For example, if the results of all the equations shown in (2.1)-(2.5) are zero, following pixels become equal p2=p1=p0=q0=q1=q2. If equations (2.1), (2.2), and (2.4) are zero, then following pixels become equal p2=p1=p0=q0.

$$p_0 - q_0 \tag{2.1}$$

$$p_1 - p_0 \tag{2.2}$$

$$q_1 - q_0 \tag{2.3}$$

$$p_2 - p_0 \tag{2.4}$$

$$q_2 - q_0 \tag{2.5}$$

The amount of simplification in the filtering equations depends on the DBF mode and the pixels that are equal. We calculated both the amount of addition and shift operations performed for filtering in each mode, and the amount of addition and shift operations performed for filtering for each mode when different combinations of pixels that will be filtered are equal. The results are shown in Tables 2.12 – 2.19. In these tables, CB in category column denotes the subtraction operations performed before the filtering.

Table 2.12 The Amount of Computation Required by DBF Mode 0 For Different Equal Pixel Combinations

Category	Original		If $p_0=q_0$		If $p_1=p_0=q_0=q_1$	
	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts
CB	5	0	5	0	5	0
Δoi	4	2	2	1	0	0
p'0, q'0	2	0	2	0	0	0
Total	11	2	9	1	5	0

Table 2.13 The Amount of Computation Required by DBF Mode 1 For Different Equal Pixel Combinations

Category	Original		If $p_1=p_0=q_0$ or $p_0=q_0=q_1$		If $p_1=p_0=q_0=q_1$	
	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts
CB	5	0	5	0	5	0
p'0, q'0	6	4	5	5	0	0
Total	11	4	10	5	5	0

Table 2.14 The Amount of Computation Required by DBF Mode 2 For Different Equal Pixel Combinations

Category	Original		If $p_0=q_0$		If $p_0=q_0=q_1$		If $p_1=p_0=q_0=q_1$		If $p_1=p_0=q_0=q_1=q_2$	
	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts
CB	5	0	5	0	5	0	5	0	5	0
Δoi	4	2	2	1	2	1	0	0	0	0
Δq1i	4	3	2	2	1	1	1	1	0	0
p'0, q'0, q'1	3	0	3	0	3	0	1	0	0	0
c0	1	0	1	0	1	0	0	0	0	0
Total	17	5	13	3	12	2	7	1	5	0

Table 2.15 The Amount of Computation Required by DBF Mode 3 For Different Equal Pixel Combinations

Category	Original		If $p_0=q_0$		If $p_0=q_0=q_1$		If $p_1=p_0=q_0=q_1$		If $p_1=p_0=q_0=q_1=q_2$		If $p_1=p_0=q_0=q_1=q_2=q_3$	
	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts
CB	5	0	5	0	5	0	5	0	5	0	5	0
p'0	3	2	3	2	2	3	0	0	0	0	0	0
q'0	2	2	2	2	2	2	2	2	0	0	0	0
q'1	2	1	2	1	2	1	3	2	0	0	0	0
q'2	3	2	3	2	3	2	3	2	3	2	0	0
(α>>2)+2	1	1	0	0	0	0	0	0	0	0	0	0
Total	16	8	15	7	14	8	13	6	8	2	5	0

Table 2.16 The Amount of Computation Required by DBF Mode 5 For Different Equal Pixel Combinations

Category	Original		If $p_0=q_0$		If $p_1=p_0=q_0$		If $p_1=p_0=q_0=q_1$		If $p_2=p_1=p_0=q_0=q_1$		If $p_3=p_2=p_1=p_0=q_0=q_1$	
	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts
CB	5	0	5	0	5	0	5	0	5	0	5	0
p'0	2	2	2	2	2	2	2	2	0	0	0	0
p'1	2	1	2	1	2	1	3	2	0	0	0	0
p'2	3	2	3	2	3	2	3	2	3	2	0	0
q'0	3	2	3	2	2	3	0	0	0	0	0	0
(α>>2)+2	1	1	0	0	0	0	0	0	0	0	0	0
Total	16	8	15	7	14	8	13	6	8	2	5	0

Table 2.17 The Amount of Computation Required by DBF Mode 6 For Different Equal Pixel Combinations

Category	Original		$p_0=q_0$		$p_1=p_0=q_0$		$p_0=q_0=q_1$		$p_1=p_0=q_0=q_1$		$p_2=p_1=p_0=q_0=q_1$		$p_1=p_0=q_0=q_1=q_2$		$p_2=p_1=p_0=q_0=q_1=q_2$	
	A	S	A	S	A	S	A	S	A	S	A	S	A	S	A	S
CB	5	0	5	0	5	0	5	0	5	0	5	0	5	0	5	0
Δoi	4	2	2	1	2	1	2	1	0	0	0	0	0	0	0	0
Δp1i	4	3	2	2	1	1	2	2	1	1	0	0	1	1	0	0
Δq1i	4	3	2	2	2	2	1	1	1	1	1	1	0	0	0	0
p'0, q'0, p'1, q'1	4	0	4	0	4	0	4	0	2	0	1	0	1	0	0	0
c0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
Total	22	8	16	5	15	4	15	4	9	2	7	1	7	1	5	0

Table 2.18 The Amount of Computation Required by DBF Mode 7 For Different Equal Pixel Combinations

Category	Original		$p_0=q_0$		$p_1=p_0=q_0$ / $p_0=q_0=q_1$ / $p_1=p_0=q_0=q_1$		$p_2=p_1=p_0=q_0=q_1$		$p_1=p_0=q_0=q_1=q_2$		$p_2=p_1=p_0=q_0=q_1=q_2$		$p_3=p_2=p_1=p_0=q_0=q_1=q_2$		$p_2=p_1=p_0=q_0=q_1=q_2=q_3$		$p_3=p_2=p_1=p_0=q_0=q_1=q_2=q_3$	
	A	S	A	S	A	S	A	S	A	S	A	S	A	S	A	S	A	S
CB	5	0	5	0	5	0	5	0	5	0	5	0	5	0	5	0	5	0
p'0	2	2	2	2	2	2	0	0	2	2	0	0	0	0	0	0	0	0
p'1	4	1	3	2	3	2	0	0	2	1	0	0	0	0	0	0	0	0
p'2	3	2	3	2	3	2	3	2	3	2	3	2	0	0	3	2	0	0
q'0	2	2	2	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
q'1	2	1	2	1	2	1	2	1	0	0	0	0	0	0	0	0	0	0
q'2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	0	0	0	0
(α>>2)+2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Total	22	11	20	11	20	11	15	7	15	7	11	4	8	2	8	2	5	0

Table 2.19 The Amount of Computation Required by DBF Mode 4 For Different Equal Pixel Combinations

Category	Original		If $p_0=q_0$		If $p_1=p_0=q_0$		If $p_1=p_0=q_0=q_1$		If $p_2=p_1=p_0=q_0=q_1$	
	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts	# of Add.	# of Shifts
CB	5	0	5	0	5	0	5	0	5	0
Δoi	4	2	2	1	2	1	0	0	0	0
Δp1i	4	3	2	2	1	1	1	1	0	0
p'0, q'0, p'1	3	0	3	0	3	0	1	0	0	0
c0	1	0	1	0	1	0	0	0	0	0
Total	17	5	13	3	12	2	7	1	5	0

The number of addition and shift operations required by each DBF mode can be obtained from the filtering equations shown in Figure 2.3. But, since modes 3, 5, and 7 have common parts in filtering equations, the results of one equation can be reused for other equations. Therefore, we organized the filtering equations used in modes 3, 5 and 7 using data reuse in order to reduce the amount of computations. The filtering equations used in mode 7 are shown in (2.6)-(2.11). Table 2.20 shows the organized filtering equations and the number of addition and shift operations required by these equations. The filtering equations used in modes 3 and 5 are also organized using data reuse.

The number of occurrences of different equal pixel combinations for DBF mode 6 obtained by using H.264 JM reference software version 14.0 for several video sequences are shown in Table 2.21. In this table, the column Total shows the number of all occurrences of DBF mode 6. In the simulations, eight CIF size videos are used and 30 frames are encoded for each video. I-frame period is set to 10 frames (IPPPPPPPPPIP…).

By using the subtraction operations shown in (2.1)-(2.5), we can only check the equality of three pixels (p2 - q2) on each side of an edge. However, for modes 3, 5, and 7, when BS = 4, DBF algorithm can access up to four pixels on each side of an edge, and therefore p3 or q3 or both can be used in filtering calculations. Extra comparison operations can be performed to determine the equality of pixels p3/q3 with the other pixels that will be filtered for modes 3, 5, and 7. However, the simulation results obtained

by H.264 JM reference software version 14.0 for several video sequences showed that the number of occurrences of mode 3 with p1=p0=q0=q1=q2=q3 and mode 5 with p3=p2=p1=p0=q0=q1 is very low. The occurrence of mode 7 with p3=p2=p1=p0=q0=q1=q2=q3 is quite high. Therefore, we propose that comparison of p3/q3 with the other pixels that will be filtered is performed only for mode 7.

We determined the computation reduction achieved by the proposed technique for H.264 DBF algorithm using H.264 JM reference software version 14.0 and the information given in Tables 2.12 -2.19. As shown in Table 2.22, the amount of reductions achieved in addition and shift operations ranges from 6% to 39% and 8% to 50% respectively. The proposed technique, on the other hand, has to check if the results of several subtraction operations are equal to zero or not. But this overhead is quite small considering that checking whether a number is zero or not can be efficiently implemented in hardware.

$$p'0=(p2+2p1+2p0+2q0+q1+4)>>3 \quad (2.6)$$

$$p'1=(p2+p1+p0+q0+2)>>2 \quad (2.7)$$

$$p'2=(2p3+3p2+p1+p0+q0+4)>>3 \quad (2.8)$$

$$q'0=(q2+2q1+2q0+2p0+p1+4)>>3 \quad (2.9)$$

$$q'1=(q2+q1+q0+p0+2)>>2 \quad (2.10)$$

$$q'2=(2q3+3q2+q1+q0+p0+4)>>3 \quad (2.11)$$

Table 2.20 The Amount of Computation Required by DBF Mode 7

Equations	**Number of Additions**	**Number of Shifts**
A = p0+q0+2	2	0
p'1=(p2+p1+A)>>2	2	1
p'0=(2p'1-p2+q1)>>3	2	2
p'2=(2(p3+p2)+p'1+2)>>3	3	2
q'1=(q2+q1+A)>>2	2	1
q'0=(2q'1-q2+p1)>>3	2	2
q'2=(2(q3+q2)+q'1+2)>>3	3	2
Total	16	10

Table 2.21 Number of Occurrences of Different Equal Pixel Combinations for DBF Mode 6

Video	QP	p2=p1=p0=q0 =q1=q2		p2=p1=p0=q0=q1 or p1=p0=q0=q1=q2		p1=p0=q0=q1		p1=p0=q0 or p0=q0=q1		p0=q0		Total
		Number	Percent	Number	Percent	Number	Percent	Number	Percent	Number	Percent	
Foreman	28	26123	11.05%	9102	3.85%	1842	0.78%	17531	7.41%	9825	4.15%	236477
	35	36212	17.22%	12222	5.81%	1437	0.68%	13950	6.63%	6364	3.03%	210289
	42	48944	24.01%	12714	6.24%	1026	0.50%	17100	8.39%	5512	2.70%	203824
Akiyo	28	33082	33.97%	7744	7.95%	488	0.50%	6369	6.54%	1530	1.57%	97380
	35	42881	42.40%	9199	9.09%	318	0.31%	5792	5.73%	1390	1.37%	101144
	42	50355	45.88%	10753	9.80%	346	0.32%	10821	9.86%	1179	1.07%	109763
Mother Daughter	28	35466	30.70%	5617	4.86%	945	0.82%	7074	6.12%	2649	2.29%	115516
	35	42439	36.59%	8537	7.36%	535	0.46%	7305	6.30%	2152	1.86%	115988
	42	52986	44.01%	9392	7.80%	432	0.36%	11308	9.39%	1408	1.17%	120392
Football	28	14406	5.26%	10023	3.66%	1015	0.37%	13919	5.08%	7588	2.77%	274053
	35	52003	13.37%	26813	6.90%	1570	0.40%	22974	5.91%	6507	1.67%	388825
	42	101730	22.66%	30105	6.71%	1836	0.41%	36445	8.12%	6927	1.54%	448948
Bus	28	8619	6.00%	3869	2.69%	692	0.48%	6555	4.56%	4899	3.41%	143638
	35	21322	14.47%	8600	5.84%	658	0.45%	7615	5.17%	3087	2.09%	147383
	42	63013	28.80%	15244	6.97%	869	0.40%	19470	8.90%	4011	1.83%	218779
Mobile	28	15357	12.03%	5210	4.08%	978	0.77%	9645	7.56%	5633	4.41%	127646
	35	21678	19.66%	5869	5.32%	678	0.61%	7639	6.93%	3375	3.06%	110274
	42	25517	24.16%	6959	6.59%	424	0.40%	7764	7.35%	2176	2.06%	105620
Soccer	28	11711	6.34%	5509	2.98%	768	0.42%	8677	4.70%	5855	3.17%	184637
	35	39792	18.97%	14674	7.00%	1077	0.51%	13261	6.32%	4659	2.22%	209748
	42	99686	35.69%	18762	6.72%	1405	0.50%	28753	10.30%	6348	2.27%	279284
Ice	28	100016	42.46%	13125	5.57%	1643	0.70%	17351	7.37%	4879	2.07%	235574
	35	102303	48.19%	14089	6.64%	1221	0.58%	15209	7.16%	3620	1.71%	212292
	42	107071	50.45%	12060	5.68%	759	0.36%	16546	7.80%	2823	1.33%	212228

Table 2.23 shows the number of comparisons of p3/q3 with the other pixels that will be filtered, the amount of addition reductions achieved by the proposed technique, and the percentage of comparison overhead to the amount of addition reductions. As it can be seen from the table, the overhead of comparing p3/q3 with the other pixels that will be filtered is much smaller than the amount of addition reductions achieved by the proposed technique.

Table 2.22 Computation Reduction Results

Video	QP	Addition			Shift		
		Total	Reduc.	Percent	Total	Reduc.	Percent
Foreman	28	7504838	903375	12.04%	2706474	453966	16.77%
	35	6341801	1055606	16.65%	2371653	520408	21.94%
	42	6043722	1293325	21.40%	2330406	635053	27.25%
Akiyo	28	3230526	853418	26.42%	1218346	423397	34.75%
	35	3126663	1028670	32.90%	1210204	505048	41.73%
	42	3251718	1220752	37.54%	1275602	600120	47.05%
Mother Daughter	28	3813739	897367	23.53%	1414588	447827	31.66%
	35	3575747	1051759	29.41%	1373804	520664	37.90%
	42	3554944	1279287	35.99%	1395284	633134	45.38%
Football	28	11185206	684795	6.12%	3996661	343872	8.60%
	35	14058053	1688599	12.01%	5276546	827260	15.68%
	42	14587141	2707295	18.56%	5661659	1322879	23.37%
Bus	28	5294142	348248	6.58%	1825543	175631	9.62%
	35	5071890	643220	12.68%	1839661	316397	17.20%
	42	6925371	1601917	23.13%	2658301	782325	29.43%
Mobile	28	4500104	525377	11.67%	1556662	264521	16.99%
	35	3813736	615855	16.15%	1348916	306365	22.71%
	42	3428735	687133	20.04%	1271056	339669	26.72%
Soccer	28	6518356	454296	6.97%	2307204	227567	9.86%
	35	6710928	1137576	16.95%	2523192	557117	22.08%
	42	8589680	2434607	28.34%	3353835	1191758	35.53%
Ice	28	7166804	2374575	33.13%	2714576	1176426	43.34%
	35	6374752	2388316	37.47%	2440427	1180183	48.36%
	42	6305002	2462723	39.06%	2437715	1220809	50.08%

Table 2.23 Comparison Overhead

Video	QP	Comparison Overhead	Addition Reduction	Percent
Foreman	28	8766	903375	0.97%
	35	8846	1055606	0.84%
	42	10040	1293325	0.78%
Akiyo	28	11182	853418	1.31%
	35	11352	1028670	1.10%
	42	12416	1220752	1.02%
Mother Daughter	28	13042	897367	1.45%
	35	13622	1051759	1.30%
	42	15968	1279287	1.25%
Football	28	5240	684795	0.77%
	35	10814	1688599	0.64%
	42	17292	2707295	0.64%
Bus	28	2768	348248	0.79%
	35	5204	643220	0.81%
	42	11314	1601917	0.71%
Mobile	28	5234	525377	1.00%
	35	6774	615855	1.10%
	42	6956	687133	1.01%
Soccer	28	3610	454296	0.79%
	35	8534	1137576	0.75%
	42	19404	2434607	0.80%
Ice	28	31300	2374575	1.32%
	35	31032	2388316	1.30%
	42	34312	2462723	1.39%

CHAPTER III

LOW POWER H.264 INTRA PREDICTION HARDWARE DESIGNS

H.264 intra prediction algorithm has a very high computational complexity. In this chapter, we propose a novel technique for reducing the amount of computations performed by H.264 intra prediction algorithm and therefore reducing the power consumption of H.264 intra prediction hardware significantly without any PSNR and bit rate loss. The proposed technique performs a small number of comparisons among neighboring pixels of the current block before the intra prediction process. If the neighboring pixels of the current block are equal, the prediction equations of H.264 intra prediction modes simplify significantly for this block. By exploiting the equality of the neighboring pixels, the proposed technique reduces the amount of computations performed by 4x4 luminance, 16x16 luminance, and 8x8 chrominance prediction modes up to 60%, 28%, and 68% respectively with a small comparison overhead. We also implemented an efficient 4x4, 16x16 and 8x8 intra prediction hardware architectures including the proposed technique using Verilog HDL. We quantified the impact of the proposed technique on the power consumption of these hardware architectures on a Xilinx Virtex II FPGA using Xilinx XPower, and it reduced the power consumption of 4x4, 16x16 and 8x8 hardware architectures up to 18.6%, 8.3% and 21.5% respectively.

The proposed technique is applicable to H.264 4x4 luminance, 16x16 luminance and 8x8 chrominance prediction modes. The technique performs a small number of comparisons among neighboring pixels of the current block before the intra prediction process. If the neighboring pixels used for calculating the predicted pixels by an intra 4x4 prediction mode are all equal, the predicted pixels by this mode also become equal. Therefore, the prediction equations simplify to a constant value and prediction calculations for this mode become unnecessary. Furthermore, if the neighboring pixels used for calculating the predicted pixels by an intra 16x16 or an intra 8x8 prediction mode are equal, the prediction equations used by this mode simplify significantly.

The simulation results obtained by H.264 reference software, JM 14.0 [35], for several video sequences showed that this technique reduces the amount of computations performed by H.264 intra 4x4, 16x16 and 8x8 prediction modes up to 60%, 28%, and 68% respectively with a small comparison overhead. The proposed technique, for each MB, requires 12 and 24 comparisons for intra 4x4 and intra 8x8 prediction modes respectively. Since intra 4x4 and intra 16x16 prediction modes operate on the same MB, the comparison results for intra 4x4 prediction modes are also used for intra 16x16 prediction modes.

Several techniques are reported in the literature for reducing the computational complexity of H.264 intra prediction algorithm [36,37,38,39]. These techniques reduce the amount of computation for H.264 intra prediction algorithm by trying selected intra prediction modes rather than trying all intra prediction modes. However, these techniques have a drawback of PSNR and bit rate loss and they require significant amount of pre-computation.

We also designed efficient H.264 4x4, 16x16 and 8x8 intra prediction hardware architectures including the proposed technique. The hardware architectures are implemented in Verilog HDL. The Verilog RTL codes are verified to work at 50 MHz in a Xilinx Virtex II FPGA. The impact of this technique on the power consumption of these hardware implementations on a Xilinx Virtex II FPGA is quantified using Xilinx XPower tool. The proposed technique reduced the power consumption of 4x4, 16x16 and 8x8 hardware architectures up to 18.6%, 8.3% and 21.5% respectively.

Several hardware architectures for H.264 4x4 intra prediction algorithm are reported in the literature [40,41,42,43]. However, they do not report their power consumption and they do not implement the proposed technique.

3.1 H.264 Intra Prediction Algorithm Overview

Intra prediction algorithm predicts the pixels in a MB using the pixels in the available neighboring blocks. For the luma component of a MB, a 16x16 predicted luma block is formed by performing intra predictions for each 4x4 luma block in the MB and by performing intra prediction for the 16x16 MB. There are nine prediction modes for each 4x4 luma block and four prediction modes for a 16x16 luma block. A mode decision algorithm is then used to compare the 4x4 and 16x16 predictions and select the best luma prediction mode for the MB. 4x4 prediction modes are generally selected for highly textured regions while 16x16 prediction modes are selected for flat regions.

There are nine 4x4 luma prediction modes designed in a directional manner. A 4x4 luma block consisting of the pixels a to p is shown in Figure 3.1. The pixels A to M belong to the neighboring blocks and are assumed to be already encoded and reconstructed and are therefore available in the encoder and decoder to generate a prediction for the current MB. Each 4x4 luma prediction mode generates 16 predicted pixel values using some or all of the neighboring pixels A to M as shown in Figure 3.2. The examples of each 4x4 luma prediction mode for real images are given in Figure 3.3. The arrows indicate the direction of prediction in each mode. The predicted pixels are calculated by a weighted average of the neighboring pixels A-M for each mode except Vertical, Horizontal and DC modes.

The prediction equations used in each 4x4 luma prediction mode are shown in Figure 3.4 where [x, y] denotes the position of the pixel in a 4x4 block (the top left, top right, bottom left, and bottom right positions of a 4x4 block are denoted as [0, 0], [0, 3], [3, 0], and [3, 3], respectively) and pred[x, y] is the prediction for the pixel in the position [x, y].

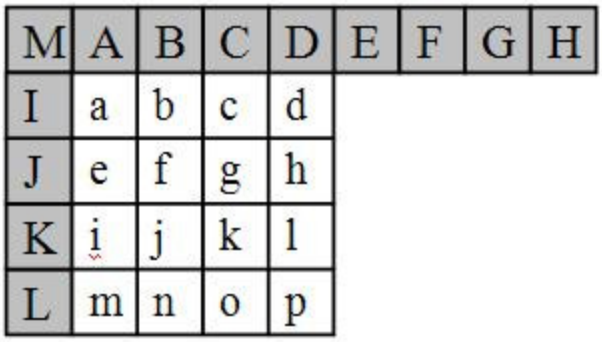

Figure 3.1 A 4x4 Luma Block and Neighboring Pixels

DC mode is always used regardless of the availability of the neighboring pixels. However, it is adopted based on which neighboring pixels A-M are available. If pixels E, F, G and H have not yet been encoded and reconstructed, the value of pixel D is copied to these positions and they are marked as available for DC mode. The other prediction modes can only be used if all of the required neighboring pixels are available [1,3]. Available 4x4 luma prediction modes for a 4x4 luma block depending on the availability of the neighboring 4x4 luma blocks are given in Table 3.1.

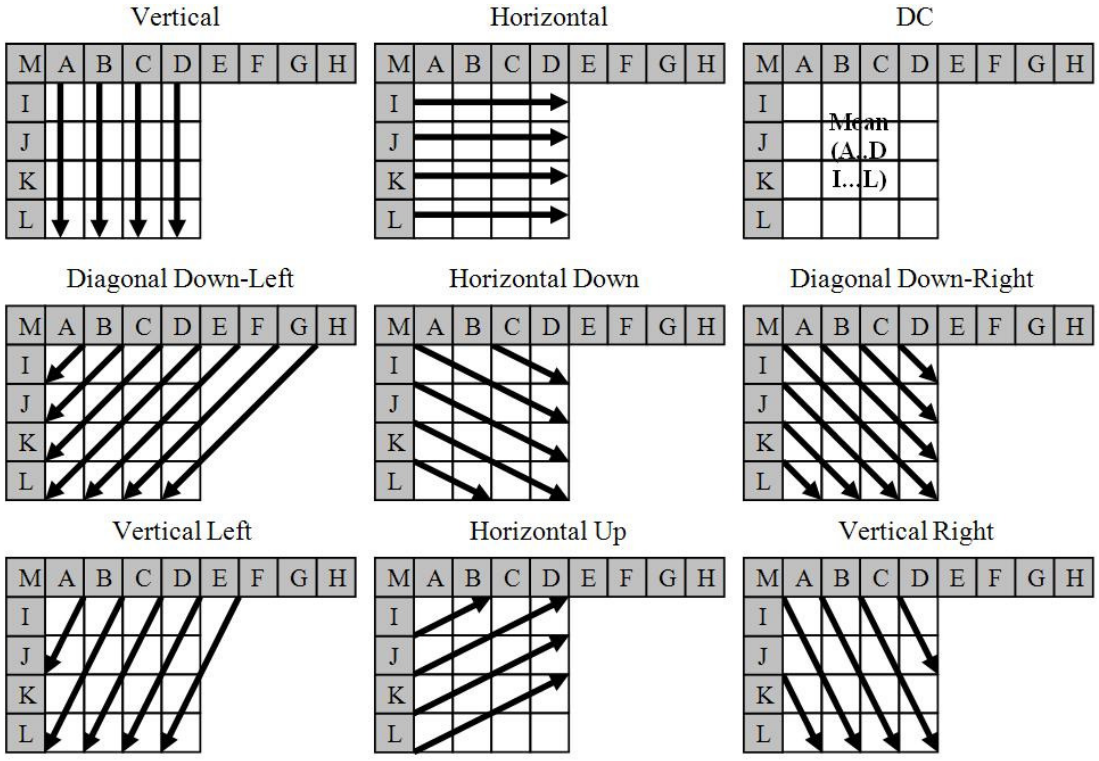

Figure 3.2 4x4 Luma Prediction Modes

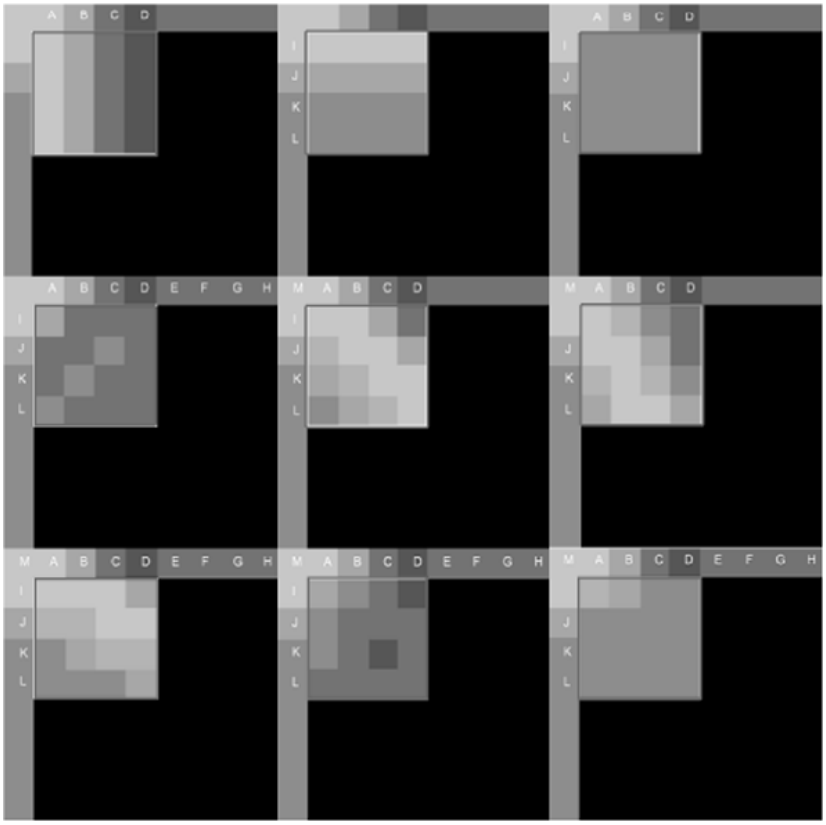

Figure 3.3 Examples of Real Images for 4x4 Luma Prediction Modes

pred[0, 0] = A	pred[0, 0] = I
pred[0, 1] = B	pred[0, 1] = I
pred[0, 2] = C	pred[0, 2] = I
pred[0, 3] = D	pred[0, 3] = I
pred[1, 0] = A	pred[1, 0] = J
pred[1, 1] = B	pred[1, 1] = J
pred[1, 2] = C	pred[1, 2] = J
pred[1, 3] = D	pred[1, 3] = J
pred[2, 0] = A	pred[2, 0] = K
pred[2, 1] = B	pred[2, 1] = K
pred[2, 2] = C	pred[2, 2] = K
pred[2, 3] = D	pred[2, 3] = K
pred[3, 0] = A	pred[3, 0] = L
pred[3, 1] = B	pred[3, 1] = L
pred[3, 2] = C	pred[3, 2] = L
pred[3, 3] = D	pred[3, 3] = L
(a) 4x4 Vertical Mode	(b) 4x4 Horizontal Mode

pred[x, y] = (A + B + C + D + I + J + K + L + 4) >> 3	If the left and the top neighboring pixels are available
pred[x, y] = (I + J + K + L + 2) >> 2	Else If only the left neighboring pixels are available
pred[x, y] = (A + B + C + D + 2) >> 2	Else If only the top neighboring pixels are available
pred[x, y] = 128	Else

(c) 4x4 DC Mode

pred[0, 0] = A + 2B + C + 2 >> 2	pred[0, 0] = A + 2M + I + 2 >> 2
pred[0, 1] = B + 2C + D + 2 >> 2	pred[0, 1] = M + 2A + B + 2 >> 2
pred[0, 2] = C + 2D + E + 2 >> 2	pred[0, 2] = A + 2B + C + 2 >> 2
pred[0, 3] = D + 2E + F + 2 >> 2	pred[0, 3] = B + 2C + D + 2 >> 2
pred[1, 0] = B + 2C + D + 2 >> 2	pred[1, 0] = M + 2I + J + 2 >> 2
pred[1, 1] = C + 2D + E + 2 >> 2	pred[1, 1] = A + 2M + I + 2 >> 2
pred[1, 2] = D + 2E + F + 2 >> 2	pred[1, 2] = M + 2A + B + 2 >> 2
pred[1, 3] = E + 2F + G + 2 >> 2	pred[1, 3] = A + 2B + C + 2 >> 2
pred[2, 0] = C + 2D + E + 2 >> 2	pred[2, 0] = I + 2J + K + 2 >> 2
pred[2, 1] = D + 2E + F + 2 >> 2	pred[2, 1] = M + 2I + J + 2 >> 2
pred[2, 2] = E + 2F + G + 2 >> 2	pred[2, 2] = A + 2M + I + 2 >> 2
pred[2, 3] = F + 2G + H + 2 >> 2	pred[2, 3] = M + 2A + B + 2 >> 2
pred[3, 0] = D + 2E + F + 2 >> 2	pred[3, 0] = J + 2K + L + 2 >> 2
pred[3, 1] = E + 2F + G + 2 >> 2	pred[3, 1] = I + 2J + K + 2 >> 2
pred[3, 2] = F + 2G + H + 2 >> 2	pred[3, 2] = M + 2I + J + 2 >> 2
pred[3, 3] = G + 3H + 2 >> 2	pred[3, 3] = A + 2M + I + 2 >> 2

(d) 4x4 Diagonal Down Left Mode

(e) 4x4 Diagonal Down Right Mode

pred[0, 0] = M + A + 1 >> 1	pred[0, 0] = M + I + 1 >> 1
pred[0, 1] = A + B + 1 >> 1	pred[0, 1] = I + 2M + A + 2 >> 2
pred[0, 2] = B + C + 1 >> 1	pred[0, 2] = B + 2A + M + 2 >> 2
pred[0, 3] = C + D + 1 >> 1	pred[0, 3] = C + 2B + A + 2 >> 2
pred[1, 0] = I + 2M + A + 2 >> 2	pred[1, 0] = I + J + 1 >> 1
pred[1, 1] = M + 2A + B + 2 >> 2	pred[1, 1] = M + 2I + J + 2 >> 2
pred[1, 2] = A + 2B + C + 2 >> 2	pred[1, 2] = M + I + 1 >> 1
pred[1, 3] = B + 2C + D + 2 >> 2	pred[1, 3] = I + 2M + A + 2 >> 2
pred[2, 0] = M + 2I + J + 2 >> 2	pred[2, 0] = J + K + 1 >> 1
pred[2, 1] = M + A + 1 >> 1	pred[2, 1] = I + 2J + K + 2 >> 2
pred[2, 2] = A + B + 1 >> 1	pred[2, 2] = I + J + 1 >> 1
pred[2, 3] = B + C + 1 >> 1	pred[2, 3] = M + 2I + J + 2 >> 2
pred[3, 0] = I + 2J + K + 2 >> 2	pred[3, 0] = K + L + 1 >> 1
pred[3, 1] = I + 2M + A + 2 >> 2	pred[3, 1] = J + 2K + L + 2 >> 2
pred[3, 2] = M + 2A + B + 2 >> 2	pred[3, 2] = J + K + 1 >> 1
pred[3, 3] = A + 2B + C + 2 >> 2	pred[3, 3] = I + 2J + K + 2 >> 2

(f) 4x4 Vertical Right Mode

(g) 4x4 Horizontal Down Mode

pred[0, 0] = A + B + 1 >> 1	pred[0, 0] = I + J + 1 >> 1
pred[0, 1] = B + C + 1 >> 1	pred[0, 1] = I + 2J + K + 2 >> 2
pred[0, 2] = C + D + 1 >> 1	pred[0, 2] = J + K+ 1 >> 1
pred[0, 3] = D + E + 1 >> 1	pred[0, 3] = J + 2K + L + 2 >> 2
pred[1, 0] = A + 2B + C + 2 >> 2	pred[1, 0] = J + K+ 1 >> 1
pred[1, 1] = B + 2C + D + 2 >> 2	pred[1, 1] = J + 2K + L + 2 >> 2
pred[1, 2] = C + 2D + E + 2 >> 2	pred[1, 2] = K + L + 1 >> 1
pred[1, 3] = D + 2E + F + 2 >> 2	pred[1, 3] = K + 3L + 2 >> 2
pred[2, 0] = B + C + 1 >> 1	pred[2, 0] = K + L + 1 >> 1
pred[2, 1] = C + D + 1 >> 1	pred[2, 1] = K + 3L + 2 >> 2
pred[2, 2] = D + E + 1 >> 1	pred[2, 2] = L
pred[2, 3] = E + F + 1 >> 1	pred[2, 3] = L
pred[3, 0] = B + 2C + D + 2 >> 2	pred[3, 0] = L
pred[3, 1] = C + 2D + E + 2 >> 2	pred[3, 1] = L
pred[3, 2] = D + 2E + F + 2 >> 2	pred[3, 2] = L
pred[3, 3] = E + 2F + G + 2 >> 2	pred[3, 3] = L
(h) 4x4 Vertical Left Mode	(i) 4x4 Horizontal Up Mode

Figure 3.4 Prediction Equations for 4x4 Luma Prediction Modes

Table 3.1 Availability of 4x4 Luma Prediction Modes

Availability of Neighboring 4x4 Luma Blocks	**Available 4x4 Luma Prediction Modes**
None available	DC
Left available, Top not available	Horizontal, DC, Horizontal Up
Top available, Left not available	Vertical, DC, Vertical Left, Diagonal Down-Left
Both available	All Modes

There are four 16x16 luma prediction modes designed in a directional manner. Each 16x16 luma prediction mode generates 256 predicted pixel values using some or all of the upper (H) and left-hand (V) neighboring pixels as shown in Figure 3.5. Vertical, Horizontal and DC modes are similar to 4x4 luma prediction modes. Plane mode is an approximation of bilinear transform with only integer arithmetic. The examples of each 16x16 luma prediction mode for real images are given in Figure 3.6. The prediction equations used in 16x16 luma prediction modes are shown in Figure 3.7 where [y, x] denotes the position of the pixel in a MB (the top left, top right, bottom left, and bottom

right positions of a MB are denoted as [0,0], [0,15], [15,0], and [15,15], respectively), p represents the neighboring pixel values and Clip1 is to clip the result between 0 and 255.

DC mode is always used regardless of the availability of the neighboring pixels. However, it is adopted based on which neighboring pixels are available. The other prediction modes can only be used if all of the required neighboring pixels are available [1,3]. Available 16x16 luma prediction modes for a MB depending on the availability of the neighboring MBs are given in Table 3.2.

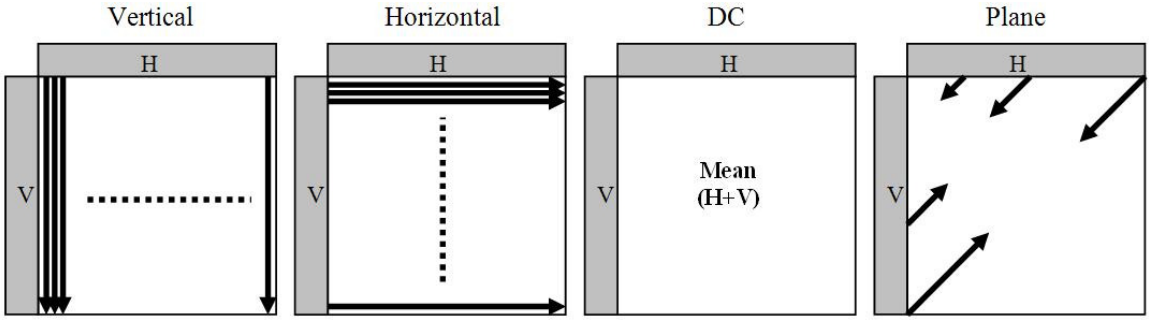

Figure 3.5 16x16 Luma Prediction Modes

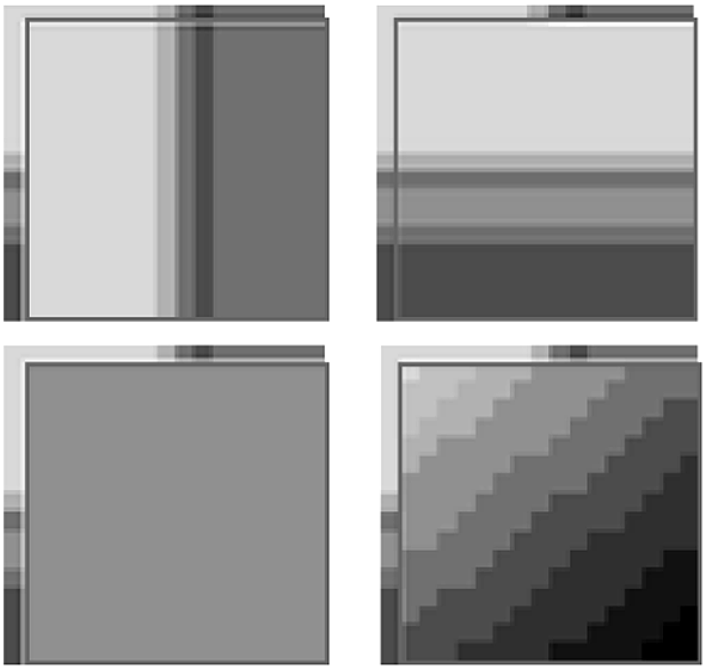

Figure 3.6 Examples of Real Images for 16x16 Luma Prediction Modes

Table 3.2 Availability of 16x16 Luma Prediction Modes

Availability of Neighboring 16x16 Luma Blocks	**Available 16x16 Luma Prediction Modes**
None available	DC
Left available, Top not available	Horizontal, DC
Top available, Left not available	Vertical, DC
Both available	All Modes

pred[x, 0] = p[-1, 0]
pred[x, 1] = p[-1, 1]
pred[x, 2] = p[-1, 2]
pred[x, 3] = p[-1, 3]
pred[x, 4] = p[-1, 4]
pred[x, 5] = p[-1, 5]
pred[x, 6] = p[-1, 6]
pred[x, 7] = p[-1, 7]
pred[x, 8] = p[-1, 8]
pred[x, 9] = p[-1, 9]
pred[x, 10] = p[-1, 10]
pred[x, 11] = p[-1, 11]
pred[x, 12] = p[-1, 12]
pred[x, 13] = p[-1, 13]
pred[x, 14] = p[-1, 14]
pred[x, 15] = p[-1, 15]

(a) 16x16 Vertical Mode

pred[0, y] = p[0, -1]
pred[1, y] = p[1, -1]
pred[2, y] = p[2, -1]
pred[3, y] = p[3, -1]
pred[4, y] = p[4, -1]
pred[5, y] = p[5, -1]
pred[6, y] = p[6, -1]
pred[7, y] = p[7, -1]
pred[8, y] = p[8, -1]
pred[9, y] = p[9, -1]
pred[10,y] = p[10, -1]
pred[11, y] = p[11, -1]
pred[12, y] = p[12, -1]
pred[13, y] = p[13, -1]
pred[14, y] = p[14, -1]
pred[15, y] = p[15, -1]

(b) 16x16 Horizontal Mode

$$pred[x, y] = \left(\sum_{x'=0}^{15} p[x', -1] + \sum_{y'=0}^{15} p[-1, y'] + 16 \right) >> 5$$ If the left and the top neighboring pixels are available

$$pred[x, y] = \left(\sum_{y'=0}^{15} p[-1, y'] + 8 \right) >> 4$$ Else If only the left neighboring pixels are available

$$pred[x, y] = \left(\sum_{x'=0}^{15} p[x', -1] + 8 \right) >> 4$$ Else If only the top neighboring pixels are available

$$pred[x, y] = 128$$ Else //If the left and the upper neighboring pixels are not available

(c) 16x16 DC Mode with x=0..15 and y=0..15

$$pred[x, y] = Clip1((a + b*(x-7) + c*(y-7) + 16) >> 5)$$
$$a = 16*(p[-1,15] + p[15,-1])$$
$$b = (5*H + 32) >> 6$$
$$c = (5*V + 32) >> 6$$
$$H = \sum_{x'=0}^{7} (x' + 1)*(p[8 + x', -1] - p[6 - x', -1])$$
$$V = \sum_{y'=0}^{7} (y' + 1)*(p[-1,8 + y'] - p[-1,6 - y'])$$

(d) 16x16 Plane Mode with x, y = 0..15

Figure 3.7 Prediction Equations for 16x16 Luma Prediction Modes

For the chroma components of a MB, a predicted 8x8 chroma block is formed for each 8x8 chroma component by performing intra prediction for the MB. The chroma component of a MB and its neighboring pixels are shown in Figure 3.8. There are four 8x8 chroma prediction modes which are similar to 16x16 luma prediction modes. A mode decision algorithm is used to compare the 8x8 predictions and select the best chroma prediction mode for each chroma component of the MB. Both chroma components of a MB always use the same prediction mode. The prediction equations used in 8x8 chroma prediction modes are shown in Figure 3.9 where [x, y] denotes the position of the pixel in a MB (the top left, top right, bottom left, and bottom right positions of a MB are denoted as [0,0], [0,7], [7,0], and [7,7], respectively), p represents the neighboring pixel values and Clip1 is to clip the result between 0 and 255.

DC mode is always used regardless of the availability of the neighboring pixels. However, it is adopted based on which neighboring pixels are available. The other prediction modes can only be used if all of the required neighboring pixels are available [1,3]. Available 8x8 chroma prediction modes for a MB depending on the availability of the neighboring MBs are given in Table 3.3.

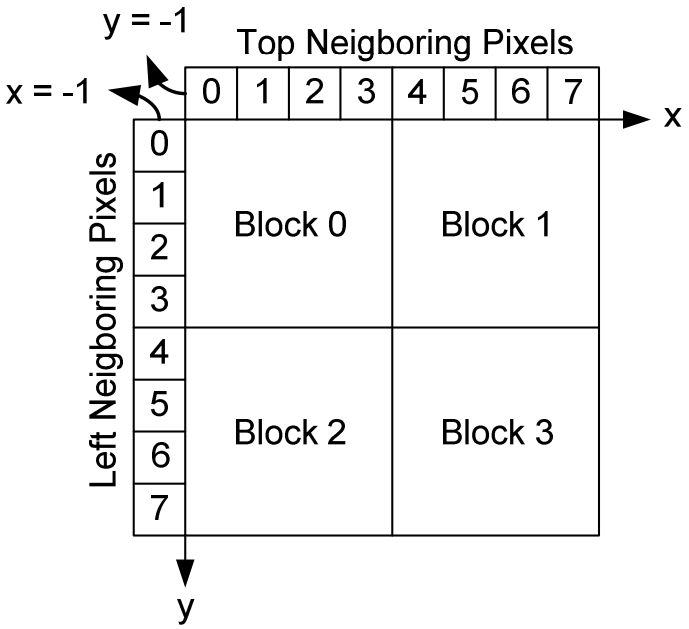

Figure 3.8 Chroma Component of a MB and its Neighboring Pixels

Table 3.3 Availability of 8x8 Luma Prediction Modes

Availability of Neighboring 8x8 Luma Blocks	**Available 8x8 Luma Prediction Modes**
None available	DC
Left available, Top not available	Horizontal, DC
Top available, Left not available	Vertical, DC
Both available	All Modes

predc[x, 0] = p[-1, 0]
predc[x, 1] = p[-1, 1]
predc[x, 2] = p[-1, 2]
predc[x, 3] = p[-1, 3]
predc[x, 4] = p[-1, 4]
predc[x, 5] = p[-1, 5]
predc[x, 6] = p[-1, 6]
predc[x, 7] = p[-1, 7]

(a) 8x8 Vertical Mode

predc[0, y] = p[0, -1]
predc[1, y] = p[1, -1]
predc[2, y] = p[2, -1]
predc[3, y] = p[3, -1]
predc[4, y] = p[4, -1]
predc[5, y] = p[5, -1]
predc[6, y] = p[6, -1]
predc[7, y] = p[7, -1]

(b) 8x8 Horizontal Mode

$$predc[x, y] = \left(\sum_{x'=0}^{3} p[x', -1] + \sum_{y'=0}^{3} p[-1, y'] + 4 \right) >> 3$$

If p[x, –1] with x = 0..3, and p[–1, y] with y = 0..3 are available

$$predc[x, y] = \left(\sum_{y'=0}^{3} p[-1, y'] + 2 \right) >> 2$$

Else If p[–1, y] with y = 0..3 are available and p[x, –1] with x = 0..3 are not available

$$predc[x, y] = \left(\sum_{x'=0}^{3} p[x', -1] + 2 \right) >> 2$$

Else If p[x, –1] with x = 0..3 are available and p[–1, y] with y = 0..3 are not available

$$predc[x, y] = 128$$

Else //If p[x, –1] with x = 0..3, and p[–1, y] with y = 0..3 are not available

(c) 8x8 DC Mode with x=0..3 and y=0..3 (Block 0 in Fig. 3.8)

$$predc[x, y] = \left(\sum_{x'=4}^{7} p[x', -1] + 2 \right) >> 3$$

If p[x, –1] with x = 4..7 are available

$$predc[x, y] = \left(\sum_{y'=0}^{3} p[-1, y'] + 2 \right) >> 2$$

Else If p[–1, y] with y = 0..3 are available

$$predc[x, y] = 128$$

Else //If p[x, –1] with x = 4..7, and p[–1, y] with y = 0..3 are not available

(c) 8x8 DC Mode with x=4..7 and y=0..3 (Block 1 in Fig. 3.8)

$$predc[x, y] = \left(\sum_{y'=4}^{7} p[-1, y'] + 2 \right) >> 2$$

If p[–1, y] with y = 4..7 are available

$$predc[x, y] = \left(\sum_{x'=0}^{3} p[x', -1] + 2 \right) >> 2$$

Else If p[x, –1] with x = 0..3 are available

$$predc[x, y] = 128$$

Else //If p[x, –1] with x = 0..3, and p[–1, y] with y = 4..7 are not available

(c) 8x8 DC Mode with x=0..3 and y=4..7 (Block 2 in Fig. 3.8)

$$predc[x, y] = \left(\sum_{x'=4}^{7} p[x', -1] + \sum_{y'=4}^{7} p[-1, y'] + 4 \right) >> 3$$

If p[x, –1] with x = 4..7, and p[–1, y] with y = 4..7 are available

$$predc[x, y] = \left(\sum_{y'=4}^{7} p[-1, y'] + 2 \right) >> 2$$

Else If p[–1, y] with y = 4..7 are available and p[x, –1] with x = 4..7 are not available

$$predc[x, y] = \left(\sum_{x'=4}^{7} p[x', -1] + 2 \right) >> 2$$

Else If p[x, –1] with x = 4..7 are available and p[–1, y] with y = 4..7 are not available

$$predc[x, y] = 128$$

Else //If p[x, –1] with x = 4..7, and p[–1, y] with y = 4..7 are not available

(c) 8x8 DC Mode with x=4..7 and y=4..7 (Block 3 in Fig. 3.8)

$$pred[x, y] = Clip1((a + b*(x-7) + c*(y-7) + 16) >> 5)$$
$$a = 16*(p[-1,7] + p[7,-1])$$
$$b = (17*H + 16) >> 5$$
$$c = (17*V + 16) >> 5$$
$$H = \sum_{x'=0}^{3} (x'+1)*(p[4+x', -1] - p[2-x', -1])$$
$$V = \sum_{y'=0}^{3} (y'+1)*(p[-1, 4+y'] - p[-1, 2-y'])$$

(d) 8x8 Plane Mode with x, y = 0..7

Figure 3.9 Prediction Equations for 8x8 Chroma Prediction Modes

3.2 Proposed Computational Complexity and Power Reduction Technique

The proposed technique exploits equality of neighboring pixels for simplifying the prediction calculations done by H.264 intra prediction modes. The technique is applied to H.264 4x4 luminance, 16x16 luminance and 8x8 chrominance prediction modes.

Intra 4x4 modes use 13 neighboring pixels for prediction calculations. The proposed technique for intra 4x4 modes is based on the equality of the neighboring pixels A, B, C, D, E, F, G, H, I, J, K, L, M of the currently processed 4x4 block. Each intra 4x4 prediction mode uses some of these neighboring pixels to predict a 4x4 block. H.264 4x4 intra prediction modes and the neighboring pixels they use for prediction calculations are

shown in Table 3.4. The prediction equations of a 4x4 intra prediction mode simplify to a constant value if the neighboring pixels used by this mode are all equal.

Table 3.4 4x4 Intra Modes and Corresponding Neighboring Pixels

4x4 Intra Modes	Neighboring Pixels
Vertical	A, B, C, D
Horizontal	I, J, K, L
DC	A, B, C, D, I, J, K, L
Diagonal Down Left	A, B, C, D, E, F, G, H
Diagonal Down Right	A, B, C, D, I, J, K, L, M
Vertical Right	A, B, C, D, I, J, K, M
Horizontal Down	A, B, C, I, J, K, L, M
Vertical Left	A, B, C, D, E, F, G
Horizontal Up	I, J, K, L

The prediction equation used by DC mode is given in equation (3.1). If neighboring pixels A, B, C, D, I, J, K, L are equal, we can substitute A (one of the neighboring pixels) in place of every neighboring pixel in equation (3.1). Therefore, in this case, the equation (1) simplifies to A as shown in (3.2).

$$pred[y, x] = [(A+B)+(C+D)+(I+J)+(K+L)+4] >> 3 \quad (3.1)$$

$$pred[y, x] = [8A+4] >>3 = A \quad (3.2)$$

This is the case for other prediction modes as well. For example, as shown in Figure 3.4, DDL mode uses A, B, C, D, E, F, G, H neighboring pixels in its prediction equations. The prediction equation for the pixel [0, 0] is given in equation (3.3). If neighboring pixels A, B, C, D, E, F, G, H are all equal, all prediction equations of DDL mode simplifies to a constant value as shown in (3.4).

$$pred[0, 0] = A + 2B + C + 2 >> 2 \quad (3.3)$$

$$pred[0,0] = [4A+2] >>2 = A \quad (3.4)$$

Since, in this case, all predicted pixels by DDL mode will be the same and equal to one of the neighboring pixels, the calculations done by DDL prediction mode become unnecessary. Therefore, during 4x4 intra prediction, the calculations done by DDL mode can be avoided by only comparing a few neighboring pixels at the beginning of the prediction process. During 4x4 intra prediction, the calculations done by the other prediction modes can be avoided in the same way by comparing the neighboring pixels used by the prediction equations of these modes. Therefore, this technique can significantly reduce the computational complexity with a small comparison overhead.

The number of 4x4 intra prediction modes with equal neighboring pixels in a frame varies from frame to frame. We analyzed CIF sized Foreman, Akiyo and Mother&Daughter frames at 28, 35 and 42 QP values using JM 14.0 to determine how many prediction modes have equal neighboring pixels. The percentages of 4x4 modes that have equal neighboring pixels for each frame are given in Table 3.5. The percentage of prediction modes with equal neighboring pixels vary from 14% to 89%. The percentage increases with higher QP values. Vertical, Horizontal, Horizontal up, DC, DDL and Vertical left modes typically have more than 50% equal neighboring pixels. DDR, Horizontal down and Vertical right have relatively lower percentage with a typical value of more than 25%.

Table 3.6 shows the amount of computation performed by the prediction equations of each 4x4 intra mode in terms of number of addition and shift operations. Vertical and Horizontal modes require no computation. The prediction equations of the other modes include only addition and shift operations. Vertical right, Horizontal down and Vertical left modes have large amount of computation. A total of 884183 addition and 529181 shift operations are performed by the H.264 4x4 intra prediction algorithm for a CIF (352x288) frame.

Based on this information and the information given in Tables 3.5 and 3.6, we calculated the computation reduction achieved by the proposed technique for CIF size Foreman, Akiyo and Mother&Daughter frames. As shown in Table 3.7, the computation reduction ranges from 28% to 60%. The proposed technique, on the other hand, has an overhead of only 74882 comparisons for a CIF (352x288) frame.

Table 3.5 Percentage of 4x4 Intra Prediction Modes with Equal Neighboring Pixels

	4x4 Intra Modes	QP = 28	QP = 35	QP = 42
Foreman	VERT	50.17%	68.75%	84.31%
	HORZ/HORZ_UP	47.76%	65.74%	79.51%
	DC	29.34%	48.93%	68.77%
	DDL	40.94%	61.10%	80.26%
	DDR	14.08%	21.26%	24.61%
	VERT_RIGHT	14.55%	21.61%	25.02%
	HORZ_DOWN	14.47%	21.89%	24.78%
	VERT_LEFT	41.56%	61.58%	80.51%
Akiyo	VERT	65.01%	75.14%	85.89%
	HORZ/HORZ_UP	66.19%	78.82%	87.06%
	DC	48.94%	62.52%	76.69%
	DDL	56.66%	67.52%	81.06%
	DDR	28.54%	34.00%	35.05%
	VERT_RIGHT	28.88%	34.20%	35.31%
	HORZ_DOWN	29.25%	34.44%	35.50%
	VERT_LEFT	57.20%	67.93%	81.66%
Mother Daughter	VERT	57.58%	74.23%	87.58%
	HORZ/HORZ_UP	62.06%	77.90%	89.13%
	DC	43.62%	60.24%	78.31%
	DDL	48.33%	65.75%	82.04%
	DDR	29.20%	37.03%	37.50%
	VERT_RIGHT	29.34%	37.34%	37.86%
	HORZ_DOWN	30.59%	38.01%	38.19%
	VERT_LEFT	48.82%	66.16%	82.51%

Table 3.6 Computation Amount of 4x4 Intra Modes

Modes	Number of Addition	Number of Shift
DDL	21	14
DDR	21	14
VERT_RIGHT	26	16
HORZ_DOWN	26	16
VERT_LEFT	25	15
HORZ_UP	15	9
DC (Left Avail.)	4	1
DC (Top Avail.)	4	1
DC (Both Avail.)	8	1

Table 3.7 Intra 4x4 Modes Computation Reduction Results

	QP	Addition Reduction		Shift Reduction	
		Number	Percent	Number	Percent
Foreman	28	246939	27.93%	146816	27.74%
	35	365863	41.38%	216263	40.87%
	42	459269	51.94%	269710	50.97%
Akiyo	28	386890	43.76%	229707	43.41%
	35	461728	52.22%	273099	51.61%
	42	521463	58.98%	306887	57.99%
Mother Daughter	28	359883	40.70%	214067	40.45%
	35	469840	53.14%	278673	52.66%
	42	539033	60.96%	317345	59.97%

Intra 8x8 chrominance modes use 17 neighboring pixels for prediction calculations and intra 16x16 luminance modes use 33 neighboring pixels for prediction calculations. Therefore, the probability of all the neighboring pixels of an intra 8x8 or an intra 16x16 mode being equal is much smaller than that of an intra 4x4 mode. Therefore, as shown in Figure 3.10, we divide the neighboring pixels of intra 16x16 and intra 8x8 modes into four pixel groups (H1, H2, H3, H4, V1, V2, V3, V4) and check the equality of the neighboring pixels in each group separately.

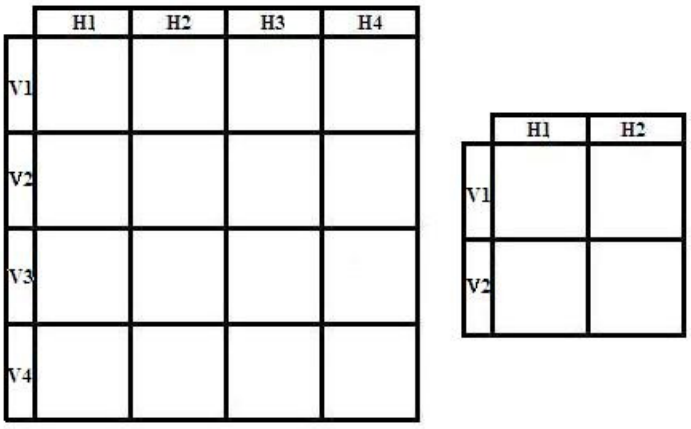

Figure 3.10 Four Pixel Groups of Neighboring Pixels of a MB

We analyzed CIF sized Foreman, Akiyo and Mother Daughter frames at 28, 35 and 42 QP values respectively using JM 14.0 to determine how many 16x16 luminance and 8x8 chrominance four pixel groups have equal pixels. The percentages of 16x16 luminance and 8x8 chrominance four pixel groups that have equal pixels for each frame are given in Tables 3.8 and 3.9 respectively. There are 396 MBs in CIF sized frame, but 378 MBs have horizontal groups H1, H2, H3, H4 and 374 MBs have vertical groups V1, V2, V3, V4. For intra 16x16 luminance modes, the percentage of four pixel groups with equal pixels ranges from 43% to 77% and it is typically greater than 50%. For intra 8x8 chrominance modes, the percentage ranges from 73% to 90% and it is typically more than 80%.

Table 3.10 shows the amount of computation performed by the prediction equations of each 16x16 and 8x8 intra mode in terms of number of addition and shift operations. Vertical and Horizontal modes require no computation. The prediction equations of the DC mode include only addition and shift operations. Plane mode have large amount of computation, and as shown in Figure 3.7, it uses multiplication in the prediction equations. But the multiplication operation can be replaced with addition and shift operations [40,43]. Therefore, a total of 121631 addition and 106067 shift operations are performed by the H.264 16x16 intra prediction algorithm for a CIF (352x288) frame, and a total of 30778 addition and 106067 shift operations are performed by the H.264 8x8 intra prediction algorithm for a CIF (352x288) frame.

The proposed technique simplifies both 16x16 and 8x8 DC and plane mode prediction equations significantly. As shown in (3.5), 16x16 DC mode prediction equations add the upper and left neighboring pixels with a constant value and divide the result by 32. The part of the prediction equation using the neighboring pixels in H1 group is shown in equation (3.6). If the neighboring pixels in H1 group are equal, in this part of the prediction equation, instead of adding the four neighboring pixels in the H1 group, one of the neighboring pixels can be shifted by 2 as shown in (3.6). In this way, three addition operations are replaced with one shift operation. This is the case for the other four neighboring pixel groups as well. Whenever the four pixels in a group are equal, three addition operations are avoided by doing one shift operation. A similar computation reduction is achieved for 16x16 plane mode as well.

Table 3.8 Percentage of 16x16 Intra Prediction Modes with Equal Neighboring Pixels

		QP = 28	QP = 35	QP = 42
Foreman	H1	45.96%	61.36%	68.43%
	H2	46.21%	65.15%	72.22%
	H3	47.22%	62.12%	71.97%
	H4	43.94%	62.12%	71.46%
	V1	46.97%	58.59%	65.15%
	V2	45.20%	58.33%	65.91%
	V3	45.71%	56.06%	67.17%
	V4	43.69%	56.57%	62.37%
Akiyo	H1	61.36%	63.38%	70.45%
	H2	58.33%	64.65%	71.46%
	H3	58.59%	66.67%	71.21%
	H4	57.83%	60.86%	70.71%
	V1	61.87%	65.91%	70.96%
	V2	58.59%	67.93%	71.46%
	V3	58.84%	67.93%	70.20%
	V4	61.11%	69.70%	71.21%
Mother Daughter	H1	48.99%	65.15%	74.49%
	H2	52.53%	67.93%	71.46%
	H3	49.24%	65.91%	74.49%
	H4	47.47%	64.39%	68.18%
	V1	51.77%	65.66%	74.49%
	V2	58.59%	71.97%	76.52%
	V3	57.32%	70.45%	76.77%
	V4	60.10%	74.24%	77.53%

$$\text{pred}[y,x] = (\sum (p[x',-1]+p[-1, x'])+16) >> 5, \text{ with } x' = 0, 1, \ldots,15$$
$$= (p[0,-1]+ p[1,-1]+\ldots+p[15,-1] + p[-1,0]+ p[-1,1]+\ldots+p[-1,15]+16)>>5 \quad (3.5)$$

$$p[0,-1]+p[1,-1]+p[2,-1]+p[3,-1] = 4*p[0,-1] = p[0,-1]<<2 \quad (3.6)$$

Table 3.9 Percentage of 8x8 Intra Prediction Modes (Chroma CB, CR) with Equal Neighboring Pixels

			QP = 28	QP = 35	QP = 42
Foreman	Cb	H1	78.03%	84.85%	87.37%
		H2	78.79%	85.35%	87.12%
		V1	79.04%	85.86%	86.11%
		V2	78.54%	84.85%	86.87%
	Cr	H1	83.33%	85.35%	84.60%
		H2	84.60%	85.35%	87.12%
		V1	86.62%	85.10%	85.10%
		V2	83.59%	85.35%	85.10%
Akiyo	Cb	H1	73.23%	79.55%	82.07%
		H2	74.75%	81.31%	84.34%
		V1	75.51%	78.79%	84.09%
		V2	77.53%	79.55%	83.33%
	Cr	H1	78.03%	83.08%	85.10%
		H2	80.81%	83.59%	86.87%
		V1	80.05%	83.08%	87.37%
		V2	79.80%	81.57%	86.11%
Mother Daughter	Cb	H1	84.85%	84.09%	80.30%
		H2	80.05%	82.32%	84.09%
		V1	81.57%	86.36%	87.37%
		V2	83.59%	87.37%	87.63%
	Cr	H1	82.83%	85.10%	86.36%
		H2	85.10%	86.62%	88.38%
		V1	82.83%	86.62%	89.14%
		V2	85.61%	86.62%	89.90%

Table 3.10 Computation Amount of Intra 16x16 and Intra 8x8 Modes

MODES	**Intra 16x16**		**Intra 8x8**	
	Number of Addition	**Number of Shift**	**Number of Addition**	**Number of Shift**
PLANE	307	296	89	82
DC (Left Available)	16	1	8	2
DC (Top Available)	16	1	8	2
DC (Both Available)	32	1	24	4

Plane mode prediction equations, however, are more complex than DC mode prediction equations. Plane mode has two calculation steps as shown in Figure 3.7. The first step calculates a, b, c parameters from the neighboring pixels of the current MB, and only 16% of the total plane mode calculations are performed in the first step. The second step calculates the predicted pixels from a, b, c parameters and 84% of the total plane mode calculations are performed in the second step. The predicted pixel values by the plane mode are the weighted sum of a, b and c parameters. If b or c or both are equal to zero, the plane mode prediction equations simplify significantly. Therefore, the proposed technique also checks whether b and c parameters are equal to zero or not before the second step, and in this way, it avoids many additional unnecessary calculations with an additional small comparison overhead.

Based on the information given in Tables 3.8, 3.9, and 3.10, we calculated the computation reduction achieved by the proposed technique for intra 16x16 and intra 8x8 prediction modes for CIF-sized Foreman, Akiyo and Mother Daughter frames. As shown in Tables 3.11 and 3.12, the computation reduction ranges from 28% to 68%.

H.264 intra 4x4 prediction equations and intra 16x16 prediction equations use the same neighboring pixels at MB boundaries. Since the proposed technique checks the equality of these neighboring pixels for intra 4x4 modes, these equality results are re-used for checking the equality of four neighboring pixel groups for intra 16x16 modes, and therefore, 3008 1-bit comparisons are performed for intra 16x16 DC and plane modes. In addition, 714 comparisons are performed for checking the equality of parameters b and c to zero for 16x16 plane mode. The proposed technique, on the other hand, requires 3x4x2=24 comparison operations for checking the equality of four neighboring pixel groups for intra 8x8 prediction calculations of the current Cb and Cr chrominance blocks. Therefore, it has an overhead of 5226 comparisons for intra 8x8 prediction modes.

Table 3.11 Intra 16x16 Computation Reduction Results

	QP	Addition Reduction		Shift Reduction	
		Number	Percent	Number	Percent
Foreman	28	16183	13.30%	9403	8.87%
	35	19662	16.17%	10764	10.15%
	42	21831	17.95%	11786	11.11%
Akiyo	28	28865	23.73%	20182	19.03%
	35	30204	24.83%	20608	19.43%
	42	30950	25.45%	20604	19.43%
Mother Daughter	28	25660	21.10%	17911	16.89%
	35	34543	28.40%	24566	23.16%
	42	33779	27.77%	22875	21.57%

Table 3.12 Intra 8x8 (Chroma CB, CR) Computation Reduction Results

		QP	Addition Reduction		Shift Reduction	
			Number	Percent	Number	Percent
Foreman	Cb	28	19347	47.60%	11293	36.69%
		35	24624	60.58%	15847	51.49%
		42	26018	64.01%	17055	55.41%
	Cr	28	23379	57.52%	14688	47.72%
		35	26925	66.24%	18113	58.85%
		42	27771	68.33%	18885	61.36%
Akiyo	Cb	28	19712	48.50%	12035	39.10%
		35	21863	53.79%	13650	44.35%
		42	24893	61.24%	16277	52.89%
	Cr	28	21746	53.50%	13557	44.05%
		35	23199	57.08%	14678	47.69%
		42	25498	62.73%	16591	53.91%
Mother Daughter	Cb	28	20844	51.28%	12318	40.02%
		35	23833	58.64%	15046	48.89%
		42	25751	63.36%	17022	55.31%
	Cr	28	21327	52.47%	12706	41.28%
		35	25386	62.46%	16496	53.60%
		42	27618	67.95%	18482	60.05%

3.3 Proposed Intra Prediction Hardware Architectures

The proposed hardware architecture for implementing H.264 4x4 intra prediction algorithm and the proposed technique is shown in Figure 3.11. Three local neighboring buffers, top neighboring buffer, left neighboring buffer and reconstructed pixel neighboring buffer are used to store the neighboring pixels in the previously coded and reconstructed neighboring 4x4 luma blocks in the current MB. After a 4x4 luma block in the current MB is coded and reconstructed, the neighboring pixels in this block are stored in the corresponding local buffers. Nine parallel datapaths are used to calculate the predicted pixels. Each datapath is used to calculate the predicted pixels by a different 4x4 intra prediction mode and they are optimized for performance and power consumption.

Thirteen registers are used to store the neighboring pixels (A, B, C, D, E, F, G, H, I, J, K, L, M) for the current 4x4 block. When a new 4x4 block comes, neighboring pixel registers are loaded with the current neighboring pixels (A, B, C, D, E, F, G, H, I, J, K, L, M) in four cycles. 12 8-bit comparators are used to check for the equality of the neighboring pixels. During register load, comparisons among neighboring pixels are performed and prediction modes with equal neighboring pixels are determined. Based on the comparison results, a disable signal is generated and sent to the datapaths used for implementing the prediction modes with equal neighboring pixels.

Nine 4x32 register files are used to store the predicted pixels for nine 4x4 intra prediction modes. When a datapath implementing a prediction mode is disabled, the corresponding predicted pixel register file is loaded with one of the neighboring pixels.

The proposed hardware architecture for implementing H.264 16x16 intra prediction algorithm and the proposed technique is shown in Figure 3.12. Two local neighboring buffers are used to store previously coded and reconstructed neighboring pixels same as the ones used in intra 4x4 hardware. After loading the next MB, the neighboring pixels are loaded to their corresponding locations. Horizontal and vertical prediction register files are loaded with corresponding neighboring pixels without any prediction calculation. After the neighboring pixels are loaded, DC and plane mode modules start calculating the pixel predictions in parallel. Since intra 4x4 blocks located on upper and

left boundaries of a MB uses the same neighboring pixels with 16x16 intra modes, the comparison results obtained for neighboring pixels of intra 4x4 blocks can be reused for intra 16x16 DC and plane modes. Therefore, 16x16 intra prediction hardware architecture does not include a comparison hardware.

The proposed hardware architecture for implementing H.264 8x8 intra prediction algorithm and the proposed technique is shown in Figure 3.13. 8x8 intra prediction hardware architecture includes a comparison hardware for checking the equality of the neighboring pixels in four pixel groups before starting the prediction calculations. When a new MB arrives, the equality check of the neighboring pixels is done while the neighboring pixels are loaded to the corresponding neighboring buffers. After the neighboring pixels are loaded, 8x8 intra prediction modules start calculating the pixel predictions.

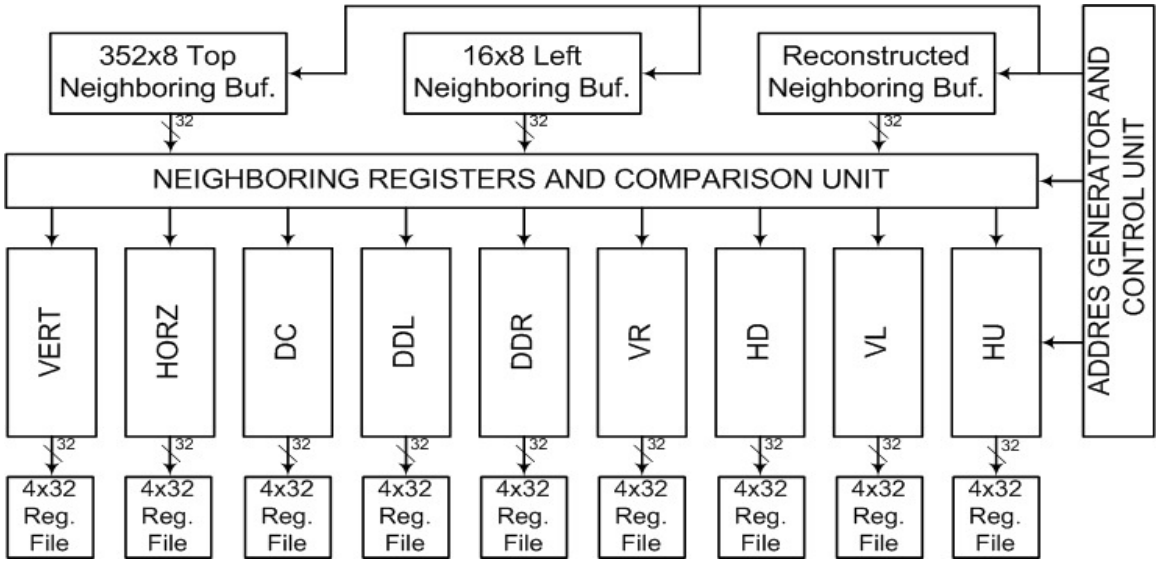

Figure 3.11 4x4 Intra Prediction Hardware Architecture

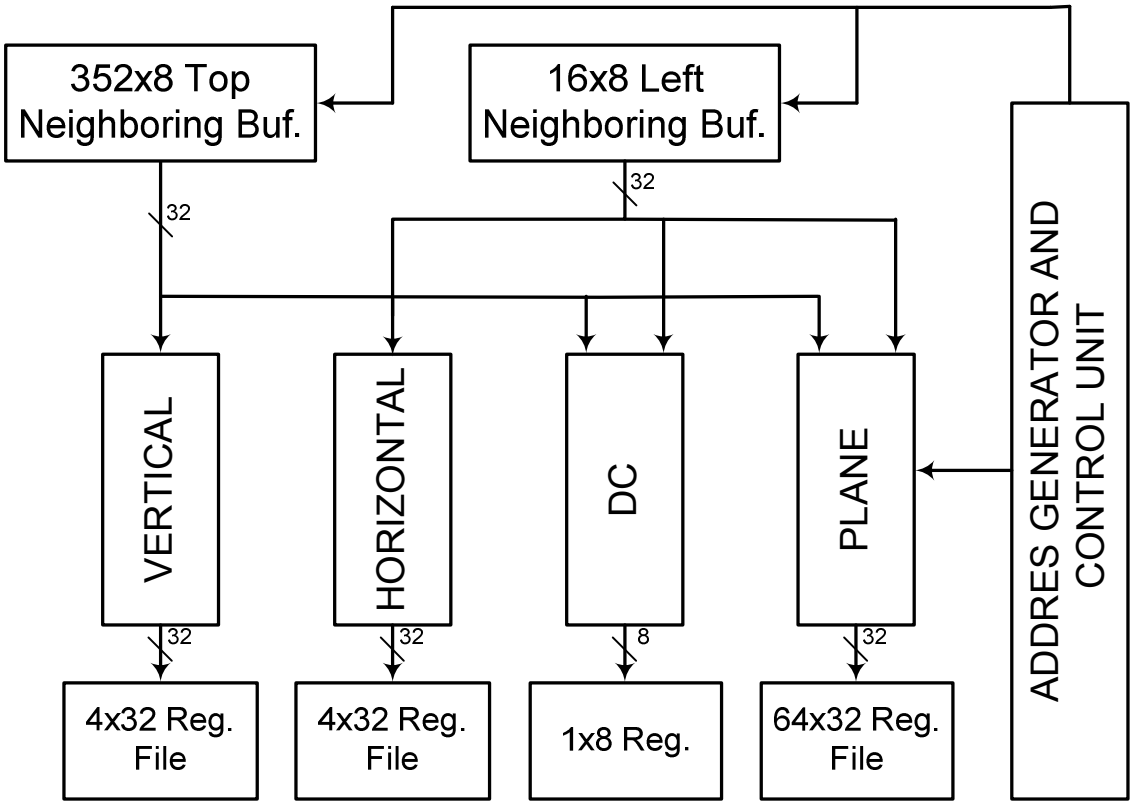

Figure 3.12 16x16 Intra Prediction Hardware Architecture

All of the intra prediction hardware architectures are synthesized to a 2V8000ff1157 Xilinx Virtex II FPGA with speed grade 5 using Mentor Graphics Precision RTL 2005b. The resulting netlists are placed and routed to the same FPGA at 50 MHz using Xilinx ISE 8.2i. Table 3.13 shows FPGA resource usages of H.264 intra 4x4, 16x16, and 8x8 hardware architectures. The columns labeled I show the resource usages of original H.264 intra prediction hardware architectures and the columns labeled II show the resource usages of H.264 intra prediction hardware architectures including the proposed technique.

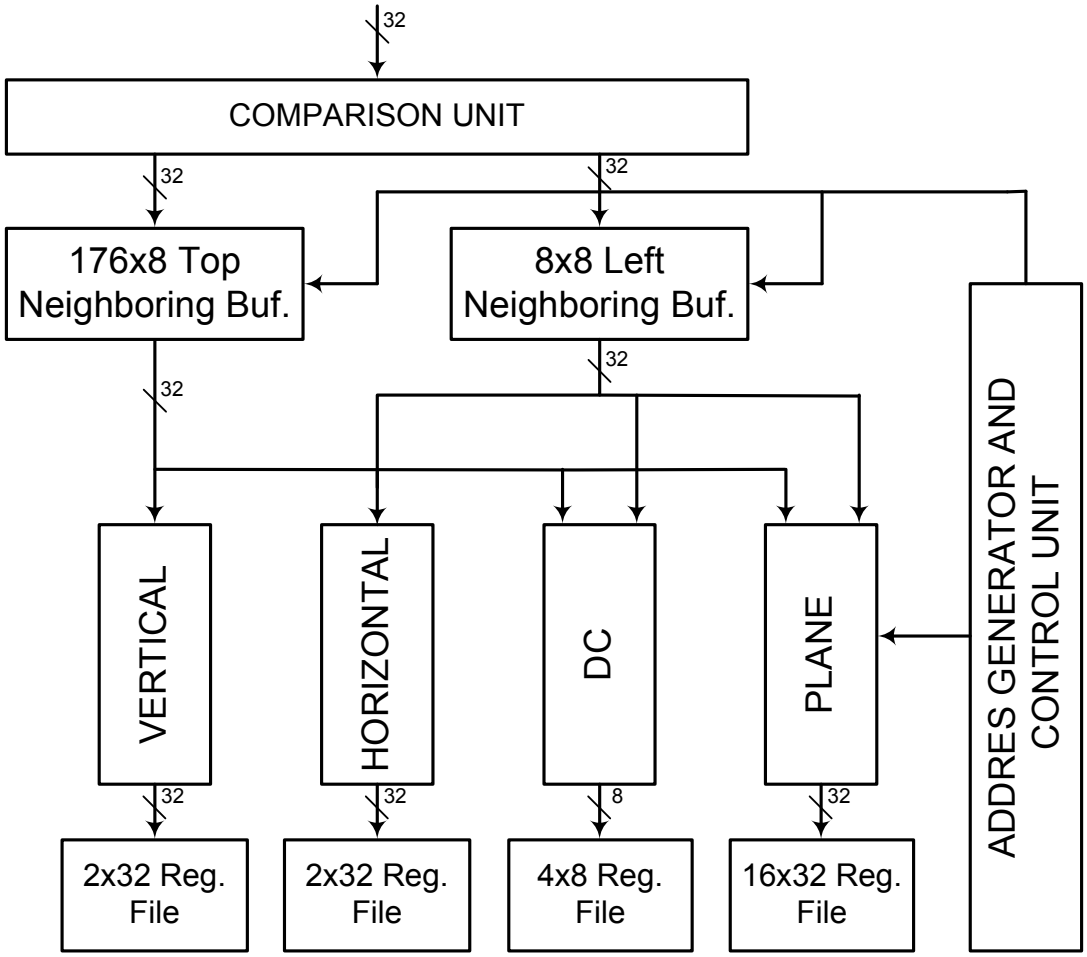

Figure 3.13 8x8 Intra Prediction Hardware Architecture

Table 3.13 FPGA Resource Usages of Original Intra Prediction Hardware and Intra Prediction Hardware with Proposed Technique

Resource	Intra 4x4		Intra 16x16		Intra 8x8	
	I	II	I	II	I	II
Function Generators	2448	2514	1330	1423	958	994
DFFs	359	361	179	187	119	126
Block SelectRAMs	2	2	2	2	2	2
Clock Frequency	50 MHz	50 MHz	50 MHz	50 MHz	50 MHz	50 MHz

3.4 Power Consumption Analysis

The power consumption of intra prediction hardware on a Xilinx Virtex II FPGA is estimated using Xilinx XPower tool. In order to estimate its power consumption, timing simulation of the placed and routed netlist of intra prediction hardware is done using Mentor Graphics ModelSim SE. Foreman, Akiyo and Mother&Daughter frames are used as inputs for timing simulations and the signal activities are stored in VCD files. These VCD files are used for estimating the power consumption of intra prediction hardware using Xilinx XPower tool.

The power consumptions of the proposed hardware implementations on a Xilinx Virtex II FPGA at 25 MHz are shown in Tables 3.14 - 3.22 for different QP values and video frames. As shown in the tables, the proposed power reduction technique reduces the power consumption of the intra 4x4, 16x16 and 8x8 prediction hardware up to 18.6%, 8.3%, 21.5% respectively.

Since intra prediction hardware will be used as part of an H.264 video encoder, only internal power consumption is considered and input and output power consumptions are ignored. Therefore, the power consumption of an intra prediction hardware can be divided into three main categories; signal power, logic power and clock power. Signal power is the power dissipated in routing tracks between logic blocks. Logic power is the amount of power dissipated in the parts where computations take place. Clock power is due to clock tree used in the FPGA.

There are several reasons for the differences between the computation reduction percentages shown in Tables 3.7, 3.11, 3.12 and the power reduction percentages shown in Tables 3.14 – 3.22. The first reason is the power consumption overhead for the comparisons performed before the prediction process. For intra 4x4 prediction hardware, there are at most 12 comparisons among 13 neighboring pixels. For intra 8x8 prediction hardware there are 12 comparisons among 16 neighboring pixels. These comparison operations add some power consumption overhead to 4x4 and 8x8 intra prediction hardware architectures. On the other hand, intra 16x16 prediction hardware does not have comparison overhead. In addition, intra 16x16 and intra 8x8 hardware have extra logic

overhead to handle the division of the neighboring pixels for 16x16 and 8x8 prediction modes into four pixel groups for equality checking and this additional logic results in a power consumption overhead.

The second reason is the clock power. Since we did not do clock gating in the FPGA for the disabled datapaths, the datapaths for all prediction modes are supplied with clock regardless of the equality of the neighboring pixels. Therefore, as shown in Tables 3.14 – 3.22, this implementation of the power reduction technique does not reduce the clock power.

The third reason is that even if the datapath of a prediction mode is disabled, address generator, control unit, neighboring registers and local neighboring buffers consume power for writing the predicted pixels for that prediction mode into the corresponding register file.

Table 3.14 Power Consumption Reduction of Intra 4x4 Prediction Hardware (QP=28)

Frames	Category	Power (mW)		
		4x4 Intra Prediction Hardware	**4x4 Intra Pred. Hardware with Power Red. Tech.**	**Reduction Percent.**
Foreman	Clock	35	33	5.71%
	Logic	29.32	26.62	9.21%
	Signal	55.15	46.90	14.95%
	Total	119.47	106.52	10.84%
Akiyo	Clock	35	33	5.71%
	Logic	28.55	25.18	11.82%
	Signal	50.98	39.89	21.74%
	Total	114.53	98.07	14.37%
Mother Daughter	Clock	35	33	5.71%
	Logic	28.37	25.16	11.31%
	Signal	50.58	40.17	20.57%
	Total	113.95	98.33	13.70%

Table 3.15 Power Consumption Reduction of Intra 4x4 Prediction Hardware (Q=35)

Frames	Category	Power (mW)		
		4x4 Intra Prediction Hardware	**4x4 Intra Pred. Hardware with Power Red. Tech.**	**Reduction Percent.**
Foreman	Clock	35	33	5.71%
	Logic	28.80	25.56	11.24%
	Signal	53.58	42.34	20.98%
	Total	117.37	100.89	14.04%
Akiyo	Clock	35	33	5.71%
	Logic	28.32	24.51	13.47%
	Signal	50.23	37.12	26.11%
	Total	113.55	94.62	16.67%
Mother Daughter	Clock	35	33	5.71%
	Logic	28.10	24.16	11.95%
	Signal	49.53	36.06	24.58%
	Total	112.63	93.22	15.77%

Table 3.16 Power Consumption Reduction of Intra 4x4 Prediction Hardware (Q=42)

Frames	Category	Power (mW)		
		4x4 Intra Prediction Hardware	**4x4 Intra Pred. Hardware with Power Red. Tech.**	**Reduction Percent.**
Foreman	Clock	35	33	5.71%
	Logic	28.50	24.84	12.85%
	Signal	52.27	39.32	24.78%
	Total	115.77	97.16	16.08%
Akiyo	Clock	35	33	5.71%
	Logic	28.07	23.96	14.63%
	Signal	49.11	34.83	29.09%
	Total	112.18	91.79	18.18%
Mother Daughter	Clock	35	33	5.71%
	Logic	27.77	23.62	14.95%
	Signal	48.60	34.00	30.04%
	Total	111.37	90.62	18.63%

Table 3.17 Power Consumption Reduction of Intra 16x16 Prediction Hardware (QP=28)

Frames	Category	Power (mW)		
		16x16 Intra Prediction Hardware	16x16 Intra Pred. Hardware with Power Red. Tech.	Reduction Percent.
Foreman	Clock	18.3	19.5	-6.56%
	Logic	19.48	18.27	6.21%
	Signal	38.76	36.16	6.71%
	Total	76.54	73.93	3.41%
Akiyo	Clock	18.3	19.5	-6.56%
	Logic	18.82	17.27	8.24%
	Signal	36.69	34.02	7.28%
	Total	73.81	70.79	4.09%
Mother Daughter	Clock	18.3	19.5	-6.56%
	Logic	17.54	15.92	9.24%
	Signal	35.13	32.36	7.88%
	Total	70.97	67.78	4.49%

Table 3.18 Power Consumption Reduction of Intra 16x16 Prediction Hardware (QP= 35)

Frames	Category	Power (mW)		
		16x16 Intra Prediction Hardware	16x16 Intra Pred. Hardware with Power Red. Tech.	Reduction Percent.
Foreman	Clock	18.3	19.5	-6.56%
	Logic	17.68	16.17	8.54%
	Signal	36.89	34.16	7.40%
	Total	72.87	69.83	4.17%
Akiyo	Clock	18.3	19.5	-6.56%
	Logic	18.82	17.27	8.24%
	Signal	34.89	31.53	9.63%
	Total	72.01	68.3	5.15%
Mother Daughter	Clock	18.3	19.5	-6.56%
	Logic	16.54	14.72	11.00%
	Signal	34.51	31.16	9.71%
	Total	69.35	65.38	5.72%

Table 3.19 Power Consumption Reduction of Intra 16x16 Prediction Hardware (QP= 42)

Frames	Category	Power (mW)		
		16x16 Intra Prediction Hardware	16x16 Intra Pred. Hardware with Power Red. Tech.	Reduction Percent.
Foreman	Clock	18.3	19.5	-6.56%
	Logic	16.13	14.22	11.84%
	Signal	35.89	33.16	7.61%
	Total	70.32	66.88	4.89%
Akiyo	Clock	18.3	19.5	-6.56%
	Logic	15.93	14.37	9.79%
	Signal	33.65	29.55	12.18%
	Total	67.88	63.42	6.57%
Mother Daughter	Clock	18.3	19.5	-6.56%
	Logic	14.88	12.62	15.19%
	Signal	32.91	28.52	13.34%
	Total	66.09	60.64	8.25%

Table 3.20 Power Consumption Reduction of Intra 8x8 Prediction Hardware (QP=28)

Frames	Category	Power (mW)		
		8x8 Intra Prediction Hardware	8x8 Intra Pred. Hardware with Power Red. Tech.	Reduction Percent.
Foreman	Clock	9.2	10.8	-17.39%
	Logic	14.86	12.86	13.46%
	Signal	30.86	25.96	15.88%
	Total	54.92	49.62	9.65%
Akiyo	Clock	9.2	10.8	-17.39%
	Logic	13.88	12.01	13.47%
	Signal	29.33	23.79	18.89%
	Total	52.41	46.6	11.09%
Mother Daughter	Clock	9.2	10.8	-17.39%
	Logic	13.54	11.87	12.33%
	Signal	28.36	22.16	21.86%
	Total	51.1	44.83	12.27%

Table 3.21 Power Consumption Reduction of Intra 8x8 Prediction Hardware (QP=35)

Frames	Category	Power (mW)		
		8x8 Intra Prediction Hardware	8x8 Intra Pred. Hardware with Power Red. Tech.	Reduction Percent.
Foreman	Clock	9.2	10.8	-17.39%
	Logic	13.65	11.21	17.88%
	Signal	29.86	23.13	22.54%
	Total	52.71	45.14	14.36%
Akiyo	Clock	9.2	10.8	-17.39%
	Logic	12.69	10.86	14.42%
	Signal	27.92	19.09	31.63%
	Total	49.81	40.75	18.19%
Mother Daughter	Clock	9.2	10.8	-17.39%
	Logic	12.21	10.15	16.87%
	Signal	26.87	18.11	32.60%
	Total	48.28	39.06	19.10%

Table 3.22 Power Consumption Reduction of Intra 8x8 Prediction Hardware (QP=42)

Frames	Category	Power (mW)		
		8x8 Intra Prediction Hardware	8x8 Intra Pred. Hardware with Power Red. Tech.	Reduction Percent.
Foreman	Clock	9.2	10.8	-17.39%
	Logic	12.83	10.31	19.64%
	Signal	28.67	21.32	25.64%
	Total	50.7	42.43	16.31%
Akiyo	Clock	9.2	10.8	-17.39%
	Logic	11.69	10.03	14.20%
	Signal	26.11	17.23	34.01%
	Total	47	38.06	19.02%
Mother Daughter	Clock	9.2	10.8	-17.39%
	Logic	11.34	9.56	15.70%
	Signal	25.78	15.99	37.98%
	Total	46.32	36.35	21.52%

CHAPTER IV

LOW POWER H.264 INTRA MODE DECISION HARDWARE DESIGNS

H.264 intra prediction algorithm generates a prediction for a MB based on spatial redundancy [1,4]. H.264 intra prediction algorithm achieves better coding results than the intra prediction algorithms used in previous video compression standards. However, this coding gain comes with a significant increase in computational complexity. H.264 intra prediction algorithm uses 9 4x4 luma, 4 16x16 luma, and 4 8x8 chroma modes. The luma component of each MB in a frame has 16 4x4 blocks and each 4x4 block can be coded with one of 9 different 4x4 prediction modes. The same MB can also be coded with one of 4 different 16x16 prediction modes. Therefore, in order to choose the best mode for the luma component of a MB, intra predictions for 148 different prediction modes are calculated.

H.264 Joint Model (JM) reference software encoder implements two different intra mode decision algorithms; Lagrangian Rate Distortion Optimization (RDO) based mode decision and Sum of Absolute Transformed Difference (SATD) based mode decision [35]. Lagrangian RDO based mode decision algorithm selects the prediction mode that minimizes the Lagrangian cost function shown in (4.1). Distortion (D) and rate (R) for each prediction mode are determined by encoding the current block using this prediction mode and calculating the distortion and rate. λ is calculated based on Quantization Parameter (QP). This technique has extremely high computational complexity.

$$J = D + \lambda R \tag{4.1}$$

SATD based intra mode decision algorithm also selects the prediction mode that minimizes the Lagrangian cost function shown in (4.1). However, it estimates distortion as SATD and rate as the number of bits used for encoding the prediction mode. This SATD based cost function is defined as

$$J_{SATD} = SATD + 4\lambda R \tag{4.2}$$

SATD based intra 16x16 mode decision algorithm used in JM software calculates the cost of each intra 16x16 mode and selects the mode with minimum cost. For each intra 16x16 mode, the SATD value for a MB is calculated as follows: For each 4x4 block in a MB, denoted as (0,…,15) in Figure 4.1, find residue block by subtracting predicted block from current block, and apply Hadamard Transform (HT) to each 4x4 residue block as illustrated in Figure 4.2. Form a 4x4 DC block, as shown in Figure 4.1, by extracting DC coefficients (upper leftmost coefficient, shown as gray in Figure 4.1) of each transformed 4x4 block and dividing them by 2, and apply HT to this 4x4 DC block. Add the absolute values of all AC coefficients and Hadamard transformed and scaled DC coefficients.

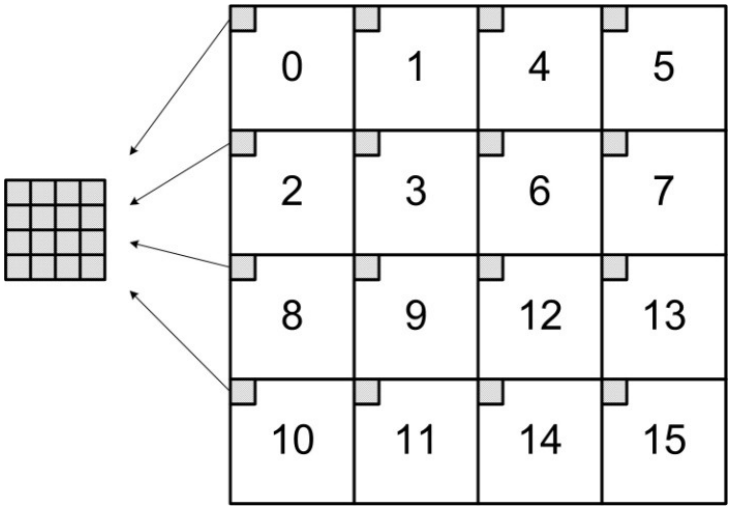

Figure 4.1 Formation of DC Block for Intra 16x16 Prediction Modes

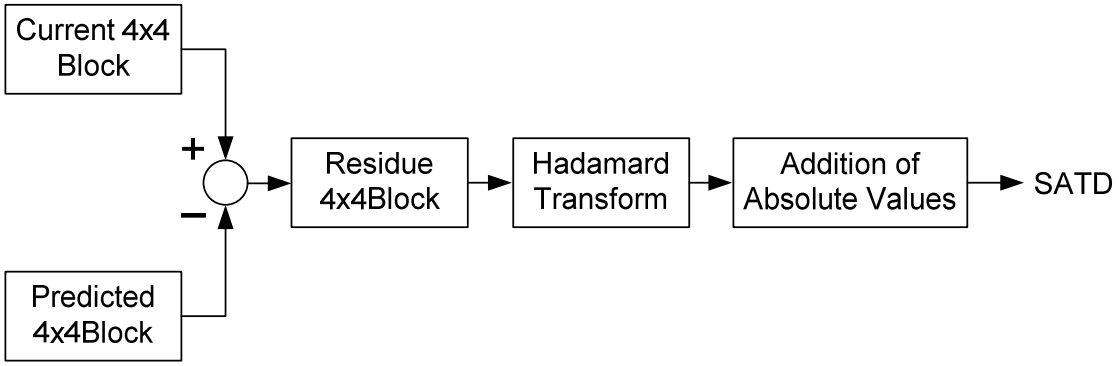

Figure 4.2 SATD Calculation for Each 4x4 Block

Intra 4x4 mode decision algorithm calculates the cost of each 4x4 mode for each 4x4 block denoted as 0,…,15 in Figure 4.1 and chooses the mode with the minimum cost for each 4x4 block. After the best modes are selected for all 4x4 blocks, the costs of the best modes for all 4x4 blocks are added to determine the total cost of the current MB. This cost is compared with the cost of the best 16x16 mode to decide the intra mode for the luma component of this MB. The Intra 8x8 mode decision algorithm is very similar to intra 16x16 mode decision algorithm except that a 4x4 DC block is not formed.

The computational complexity of the SATD based mode decision algorithm is also high. As it is shown in Figure 4.3, only 11% of all the addition operations performed for intra search are performed for intra prediction and 89% are performed for intra mode decision. Intra prediction shown in Figure 4.3 is implemented using only addition and shift operations as explained in [40,43,44]. Intra mode decision shown in Figure 4.3 includes residue operations (subtractions are counted as additions), HT operations, and addition of absolute values.

In this thesis, we propose a novel computational complexity and power reduction technique for H.264 intra mode decision. The proposed technique reduces the number of additions and shifts performed by 16x16 and 8x8 intra prediction algorithms by 80% and it reduces the number of additions performed by SATD based 4x4, 16x16 and 8x8 intra mode decision algorithms used in H.264 JM reference software encoder by 46%, 64% and 62% respectively for a CIF size frame by using fixed predicted block patterns of intra modes and distribution property of HT and by slightly modifying intra 16x16 and 8x8 plane mode prediction equations used for cost calculation by SATD based 16x16 and 8x8 intra mode decision algorithms.

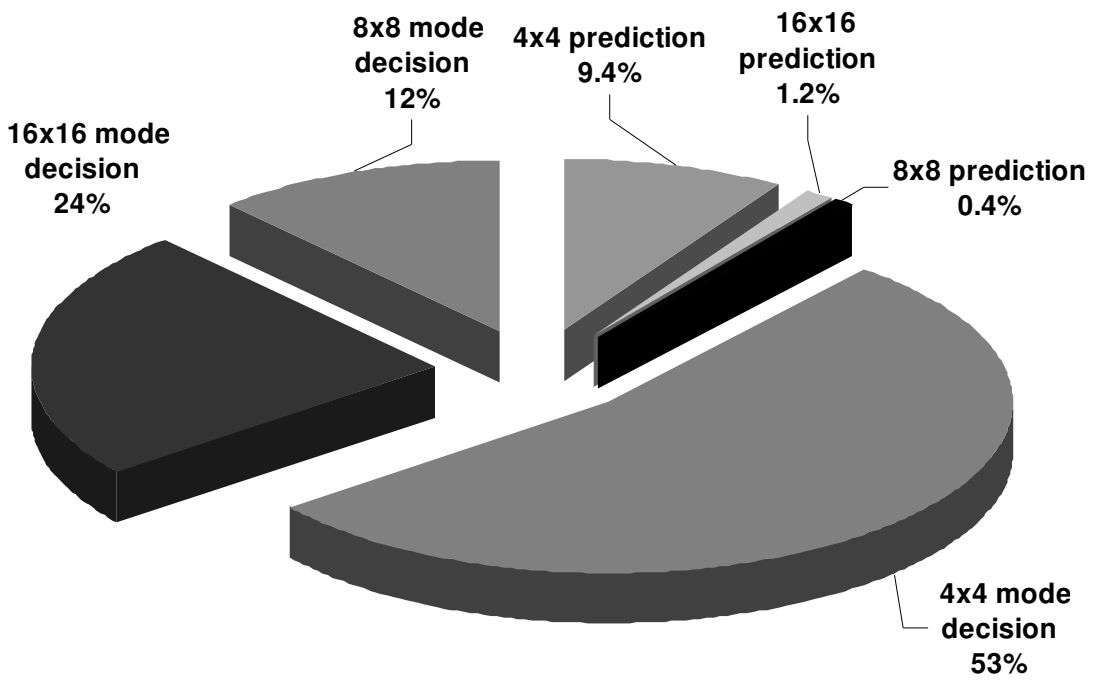

Figure 4.3 Addition Operations Performed by Intra Prediction and Mode Decision

Several techniques are proposed in the literature to reduce the computational complexity of H.264 intra mode decision. In [45], a new cost function and rate predictor, and a technique similar to the technique proposed in this thesis are proposed only for intra 4x4 mode decision. Selective intra mode decision techniques proposed in [36,37,38,40,46,47] calculate only the cost of the intra modes likely to be selected by the mode decision and select one of these intra modes at the expense of PSNR loss. The proposed technique can be used together with these selective mode decision techniques for further reducing computational complexity of H.264 intra mode decision.

4.1 Hadamard Transform

HT is a linear transform and HT of a 4x4 block Z is defined as:

$$T = H * Z * H \tag{4.3}$$

where

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{4.4}$$

If we write block Z explicitly then, Equation (4.3) becomes:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} z_0 & z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 & z_7 \\ z_8 & z_9 & z_{10} & z_{11} \\ z_{12} & z_{13} & z_{14} & z_{15} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{4.5}$$

Only binary shift and integer addition/subtraction operations are used in HT. HT defined in (4.5) can be implemented with 64 additions using the fast HT algorithm shown in Figure 4.4 [48].

As part of H.264 intra mode decision hardware, we designed a high speed HT hardware based on this fast HT algorithm. The designed hardware is two-stage pipelined to improve clock frequency and has 16 adders/subtractors. It finishes HT operations of a 4x4 block in 4 clock cycles.

4.2 Proposed Computational Complexity Reduction Technique

HT is a linear operation and it can be applied before subtraction operation as shown in (4.6). H, C, P are the Hadamard matrix, current 4x4 block, predicted 4x4 block respectively and the Hadamard matrix is shown in (4.4). In this way, two HTs are performed instead of one. However, this decreases the computational complexity of SATD based H.264 intra mode decision. Since the predicted blocks have regular patterns, HTs of the predicted blocks (H*P*H) can be calculated with a small amount of computation. In addition, since, HT of the current block (H*C*H) is common to all intra modes, once HT of the current block is found it can be used for all intra prediction modes.

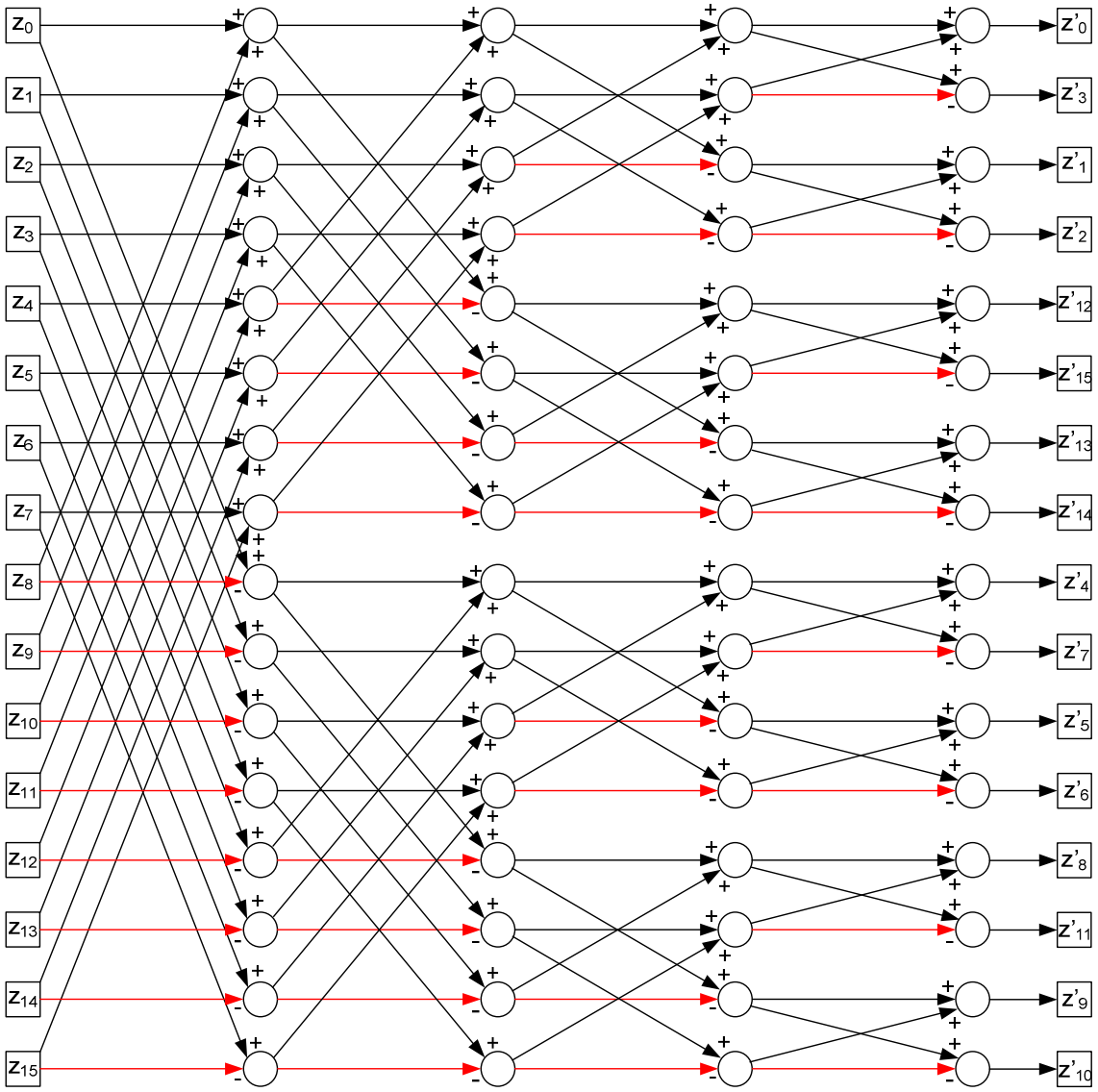

Figure 4.4 Fast HT Algorithm for a 4x4 Block

$$T = H*(C-P)*H = (H*C*H) - (H*P*H) \tag{4.6}$$

A similar technique is proposed only for intra 4x4 mode decision in [45,48]. In this thesis, we generalized this technique for the mode decision of all intra prediction modes, we showed that this technique reduces the number of residue calculations required for intra mode decision as well and we applied this technique to 16x16 and 8x8 plane modes by proposing a small modification in the prediction equations used for calculating the cost of the 16x16 and 8x8 plane modes for intra mode decision.

4.2.1 HT of Predicted Blocks by Intra 4x4 Modes

The predicted block patterns of horizontal, vertical and DC prediction modes and the result of performing HT for these predicted block patterns are shown in Figure 4.5. HT of a 4x4 block can be calculated with 64 addition operations [45]. On the other hand, as shown in Figure 4.5, HT of a 4x4 block predicted by vertical or horizontal modes can be calculated with 8 addition and 4 shift operations and HT of a 4x4 block predicted by DC mode can be calculated with only 1 shift operation.

Vertical

$$\begin{bmatrix} k & m & n & p \\ k & m & n & p \\ k & m & n & p \\ k & m & n & p \end{bmatrix} \xrightarrow{HT} \begin{bmatrix} 4(k+m+n+p) & 4(k+m-n-p) & 4(k-m-n+p) & 4(k-m+n-p) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Horizontal

$$\begin{bmatrix} k & k & k & k \\ m & m & m & m \\ n & n & n & n \\ p & p & p & p \end{bmatrix} \xrightarrow{HT} \begin{bmatrix} 4(k+m+n+p) & 0 & 0 & 0 \\ 4(k+m-n-p) & 0 & 0 & 0 \\ 4(k-m-n+p) & 0 & 0 & 0 \\ 4(k-m+n-p) & 0 & 0 & 0 \end{bmatrix}$$

DC

$$\begin{bmatrix} p & p & p & p \\ p & p & p & p \\ p & p & p & p \\ p & p & p & p \end{bmatrix} \xrightarrow{HT} \begin{bmatrix} 16p & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.5 Hadamard Transform of Vertical, Horizontal and DC Modes

The predicted block pattern of DDL mode is shown in (4.7) where k-s are defined in [1]. HT of this predicted block, shown as TDDL in (4.8), can be efficiently calculated if equations in Table 4.1 are pre-calculated. TDDL can be calculated by using pre-calculated values as shown in Table 4.2. The predicted block pattern of DDR mode is shown in (4.9) where k-s are defined in [1]. HT of this predicted block, shown as TDDR in (4.10), can be efficiently calculated if equations in Table 4.3 are pre-calculated. TDDR

be calculated by using pre-calculated values as shown in Table 4.4.The predicted block pattern of VR mode is shown in (4.11) where k-s are defined in [1]. HT of this predicted block, shown as TVR in (4.12), can be efficiently calculated if equations in Table 4.5 are pre-calculated. TVR can be calculated by using pre-calculated values as shown in Table 4.6.

The predicted block pattern of HD mode is shown in (4.13). HT of this predicted block is given in (4.14). Since this mode is similar to VR, we do not give explicit equations. The predicted block pattern of VL mode is shown in (4.15). HT of this predicted block is given in (4.16). Since this mode is similar to VR, we do not give explicit equations. The predicted block pattern of HUP mode is shown in (4.17) where k-s are defined in [1]. HT of this predicted block, shown as THUP in (4.18), can be efficiently calculated if equations in Table 4.7 are pre-calculated. THUP can be calculated by using pre-calculated values as shown in Table 4.8.

$$DDL = \begin{bmatrix} k & m & n & p \\ m & n & p & q \\ n & p & q & r \\ p & q & r & s \end{bmatrix} \tag{4.7}$$

$$TDDL = \begin{bmatrix} k+2m+3n+4p+3q+2r+s & k+2m+n-q-2r-s & k-n-q+s & k+n-q-s \\ k+2m+n-q-2r-s & k+2m-n-4p-q+2r+s & k-3n+3q-s & k-n-q+s \\ k-n-q+s & k-3n+3q-s & k-2m-n+4p-q-2r+s & k-2m+n-q+2r-s \\ k+n-q-s & k-n-q+s & k-2m+n-q+2r-s & k-2m+3n-4p+3q-2r+s \end{bmatrix} \tag{4.8}$$

$$DDR = \begin{bmatrix} k & m & n & p \\ q & k & m & n \\ r & q & k & m \\ s & r & q & k \end{bmatrix} \tag{4.9}$$

$$TDDR = \begin{bmatrix} 4k+3q+2r+s+3m+2n+p & q+2r+s-m-2n-p & -q+s-m+p & q+s-m-p \\ -q-2r-s+m+2n+p & 4k+q-2r-s+m-2n-p & 3q-s-3m+p & q-s+m-p \\ -q+s-m+p & -3q+s+3m-p & 4k-q-2r+s-m-2n+p & q-2r+s-m+2n-p \\ -q-s+m+p & q-s+m-p & -q+2r-s+m-2n+p & 4k-3q+2r-s-3m+2n-p \end{bmatrix} \tag{4.10}$$

$$VR = \begin{bmatrix} k & m & n & p \\ q & r & s & t \\ x & k & m & n \\ y & q & r & s \end{bmatrix} \tag{4.11}$$

$$TVR = \begin{bmatrix} 2k+2q+x+y+2m+2r+2n+2s+p+t & 2k+2q+x+y-2n-2s-p-t & x+y-2m-2r+p+t & x+y-p-t \\ -x-y+p+t & -x-y+2m+2r-p-t & 2k+2q-x-y-2n-2s+p+t & 2k+2q-x-y-2m-2r+2n+2s-p-t \\ -x+y+p-t & -x+y+2m-2r-p+t & 2k-2q-x+y-2n+2s+p-t & 2k-2q-x+y-2m+2r+2n-2s-p+t \\ 2k-2q+x-y+2m-2r+2n-2s+p-t & 2k-2q+x-y-2n+2s-p+t & x-y-2m+2r+p-t & x-y-p+t \end{bmatrix} \tag{4.12}$$

$$HD = \begin{bmatrix} k & m & n & p \\ q & r & k & m \\ s & t & q & r \\ x & y & s & t \end{bmatrix} \tag{4.13}$$

$$THD = \begin{bmatrix} 2k+2q+2s+x+2m+2r+2t+y+n+p & x+y-n-p & x-y-n+p & 2k+2q+2s+x-2m-2r-2t-y+n-p \\ 2k-2s-x+2m-2t-y+n+p & 2q-x+2r-y-n-p & 2q-x-2r+y-n+p & 2k-2s-x-2m+2t+y+n-p \\ -2q+x-2r+y+n+p & 2k-2s+x+2m-2t+y-n-p & 2k-2s+x-2m+2t-y-n+p & -2q+x+2r-y+n-p \\ -x-y+n+p & 2k-2q+2s-x+2m-2r+2t-y-n-p & 2k-2q+2s-x-2m+2r-2t+y-n+p & -x+y+n-p \end{bmatrix} \tag{4.14}$$

$$VL = \begin{bmatrix} k & m & n & p \\ q & r & s & t \\ m & n & p & x \\ r & s & t & y \end{bmatrix} \tag{4.15}$$

$$TVL = \begin{bmatrix} 2m + k + q + 2r + 2n + 2s + 2p + 2t + x + y & 2m + k + q + 2r - 2p - 2t - x - y & k + q - 2n - 2s + x + y & k + q - x - y \\ k + q - x - y & k + q - 2n - 2s + x + y & - 2m + k + q - 2r + 2p + 2t - x - y & k - 2m + q - 2r + 2n + 2s - 2p - 2t + x + y \\ k - q - x + y & k - q - 2n + 2s + x - y & - 2m + k - q + 2r + 2p - 2t - x + y & k - 2m - q + 2r + 2n - 2s - 2p + 2t + x - y \\ 2m + k - q - 2r + 2n - 2s + 2p - 2t + x - y & 2m + k - q - 2r - 2p + 2t - x + y & k - q - 2n + 2s + x - y & k - q - x + y \end{bmatrix} \tag{4.16}$$

$$HUP = \begin{bmatrix} k & m & n & p \\ n & p & q & r \\ q & r & s & s \\ s & s & s & s \end{bmatrix} \tag{4.17}$$

$$THUP = \begin{bmatrix} 2n + k + 2q + 6s + m + 2p + 2r & k - 2s + m & k - m & 2n + k + 2q - m - 2p - 2r \\ k + 2n - 6s + m + 2p & k - 2q + 2s + m - 2r & k - 2q - m + 2r & k + 2n - m - 2p \\ k - 2q + 2s + m - 2r & k - 2n + 2s + m - 2p & k - 2n - m + 2p & k - 2q - m + 2r \\ k - 2s + m & k - 2n + 2q - 2s + m - 2p + 2r & k - 2n + 2q - m + 2p - 2r & k - m \end{bmatrix} \tag{4.18}$$

Table 4.1 Pre-calculated Values for DDL Prediction Mode

Equations	Number of Addition/Subtractions	Number of Shift
$a(1) = 4p$	0	1
$a(2) = q + n$	1	0
$a(3) = q - n$	1	0
$a(4) = 2(q+n) + (q+n) = 3(q+n)$	1	1
$a(5) = 2(q-n) + (q-n) = 3(q-n)$	1	1
$a(6) = 2(m+r)$	1	1
$a(7) = 2(m-r)$	1	1
$a(8) = k + s$	1	0
$a(9) = k - s$	1	0
Total	8	5

Table 4.2 DDL Mode Prediction Calculations Using Pre-calculated Values

Equations	Number of Addition/Subtractions	Number of Shift
$TDDL[0,0] = a(1) + a(4) + a(6) + a(8)$	3	0
$TDDL[0,1] = a(7) + a(9) - a(3)$	2	0
$TDDL[0,2] = a(8) - a(2)$	1	0
$TDDL[0,3] = a(9) - a(3)$	1	0
$TDDL[1,0] = TDDL[0,1]$	0	0
$TDDL[1,1] = a(6) + a(8) - a(1) - a(2)$	3	0
$TDDL[1,2] = a(5) + a(9)$	1	0
$TDDL[1,3] = a(8) - a(2)$	1	0
$TDDL[2,0] = TDDL[0,2]$	0	0
$TDDL[2,1] = TDDL[1,2]$	0	0
$TDDL[2,2] = a(1) - a(2) - a(6) + a(8)$	3	0
$TDDL[2,3] = a(9) - a(3) - a(7)$	2	0
$TDDL[3,0] = TDDL[0,3]$	0	0
$TDDL[3,1] = TDDL[1,3]$	0	0
$TDDL[3,2] = TDDL[2,3]$	0	0
$TDDL[3,3] = a(8) - a(6) + a(4) - a(1)$	3	0
Total	20	0

Table 4.3 Pre-calculated Values for DDR Prediction Mode

Equations	Number of Additions/Subtractions	Number of Shifts
$a(1) = 4k$	0	1
$a(2) = q + m$	1	0
$a(3) = q - m$	1	0
$a(4) = 2(q + m) + (q + m) = 3(q + m)$	1	1
$a(5) = 2(q - m) + (q - m) = 3(q - m)$	1	1
$a(6) = 2(n + r)$	1	1
$a(7) = 2(n - r)$	1	1
$a(8) = p + s$	1	0
$a(9) = p - s$	1	0
Total	8	5

Table 4.4 DDR Mode Prediction Calculations Using Pre-calculated Values

Equations	Number of Additions/Subtractions	Number of Shifts
$TDDR[0,0] = a(1) + a(4) + a(6) + a(8)$	3	0
$TDDR[0,1] = a(3) - a(7) - a(9)$	2	0
$TDDR[0,2] = a(8) - a(2)$	1	0
$TDDR[0,3] = a(3) - a(9)$	1	0
$TDDR[1,0] = -TDDR[0,1]$	1	0
$TDDR[1,1] = a(1) + a(2) - a(6) - a(8)$	3	0
$TDDR[1,2] = a(5) + a(9)$	1	0
$TDDR[1,3] = a(2) - a(8)$	1	0
$TDDR[2,0] = TDDR[0,2]$	0	0
$TDRR[2,1] = -TDDR[1,2]$	1	0
$TDDR[2,2] = a(1) - a(2) - a(6) + a(8)$	3	0
$TDDR[2,3] = a(3) + a(7) - a(9)$	2	0
$TDDR[3,0] = -TDDR[0,3]$	1	0
$TDDR[3,1] = TDDR[1,3]$	0	0
$TDDR[3,2] = -TDDR[2,3]$	1	0
$TDDR[3,3] = a(1) - a(4) + a(6) - a(8)$	3	0
Total	24	0

Table 4.5 Pre-calculated Values for VR Prediction Mode

Equations	Number of Additions/Subtractions	Number of Shifts
$a(1) = y + x$	1	0
$a(2) = y - x$	1	0
$a(3) = p + t$	1	0
$a(4) = p - t$	1	0
$a(5) = a(1) + a(3)$	1	0
$a(6) = a(2) + a(4)$	1	0
$a(7) = a(1) - a(3)$	1	0
$a(8) = a(4) - a(2)$	1	0
$a(9) = 2(k + n)$	1	1
$a(10) = 2(k - n)$	1	1
$a(11) = 2(q + s)$	1	1
$a(12) = 2(q - s)$	1	1
$a(13) = a(9) - a(11)$	1	0
$a(14) = a(9) + a(11)$	1	0
$a(15) = a(10) + a(12)$	1	0
$a(16) = a(10) - a(12)$	1	0
$a(17) = 2(m + r)$	1	1
$a(18) = 2(m - r)$	1	1
Total	18	6

Table 4.6 VR Mode Prediction Calculations Using Pre-calculated Values

Equations	Number of Additions/Subtractions	Number of Shifts
$TVR[0,0] = a(14) + a(17) + a(5)$	2	0
$TVR[0,1] = a(15) + a(7)$	1	0
$TVR[0,2] = a(5) - a(17)$	1	0
$TVR[0,3] = a(7)$	0	0
$TVR[1,0] = -a(7)$	1	0
$TVR[1,1] = a(17) - a(5)$	1	0
$TVR[1,2] = a(15) - a(7)$	1	0
$TVR[1,3] = a(14) - a(17) - a(5)$	2	0
$TVR[2,0] = a(6)$	0	0
$TVR[2,1] = a(18) - a(8)$	1	0
$TVR[2,2] = a(6) + a(16)$	1	0
$TVR[2,3] = a(13) - a(18) - a(8)$	2	0
$TVR[3,0] = a(8) + a(13) + a(18)$	2	0
$TVR[3,1] = a(16) - a(6)$	1	0
$TVR[3,2] = a(8) - a(18)$	1	0
$TVR[3,3] = -a(6)$	1	0
Total	18	0

Table 4.7 Pre-calculated Values for HUP Prediction Mode

Equations	Number of Additions/Subtractions	Number of Shifts
$a(1) = k - m$	1	0
$a(2) = k + m$	1	0
$a(3) = 2(n + p)$	1	1
$a(4) = 2(n - p)$	1	1
$a(5) = 2(q + r)$	1	1
$a(6) = 2(q - r)$	1	1
$a(7) = 2s$	0	1
$a(8) = a(7) + 4s = 6s$	1	1
$a(9) = a(2) + a(7)$	1	0
$a(10) = a(2) + a(5)$	1	0
$a(11) = a(1) + a(6)$	1	0
Total	10	6

Table 4.8 HUP Mode Prediction Calculations Using Pre-calculated Values

Equations	Number of Additions/Subtractions	Number of Shifts
$THUP[0,0] = a(10) + a(3) + a(8)$	2	0
$THUP[0,1] = a(2) - a(7)$	1	0
$THUP[0,2] = a(1)$	0	0
$THUP[0,3] = a(11) + a(4)$	1	0
$THUP[1,0] = a(2) + a(3) - a(8)$	2	0
$THUP[1,1] = a(9) - a(5)$	1	0
$THUP[1,2] = a(1) - a(6)$	1	0
$THUP[1,3] = a(1) + a(4)$	1	0
$THUP[2,0] = THUP[1,1]$	0	0
$THUP[2,1] = a(9) - a(3)$	1	0
$THUP[2,2] = a(1) - a(4)$	1	0
$THUP[2,3] = THUP[1,2]$	0	0
$THUP[3,0] = a(2) - a(7)$	1	0
$THUP[3,1] = a(10) - a(3) - a(7)$	2	0
$THUP[3,2] = a(11) - a(4)$	1	0
$THUP[3,3] = a(1)$	0	0
Total	15	0

4.2.2 HT of Predicted Blocks by Intra 16x16 and 8x8 Horizontal, Vertical and DC Modes

In addition to the computation reduction achieved for HT of a 4x4 block, since a MB is partitioned into 4x4 blocks for HT as shown in Figure 4.6, the proposed technique significantly reduces amount of computations required for intra 16x16 and 8x8 mode decisions by data reuse. For intra 16x16 vertical mode, the predicted pixels in the 4x4 predicted blocks 0, 2, 8, and 10 are the same as shown in (4.19) and HT of this block is shown in (4.20). HT of the predicted block 0 can be reused for predicted blocks 2, 8, and 10 as well. The same is true for the other vertical predicted 4x4 blocks in the same column. For intra 16x16 horizontal mode, the predicted pixels in the 4x4 predicted blocks 0, 1, 4, and 5 are the same as shown in (4.21) and HT of this block is shown in (4.22). Therefore, HT of the predicted block 0 can be reused for predicted blocks 1, 4, and 5 as well. The same is true for the other horizontal predicted 4x4 blocks in the same row. For

DC mode, the predicted pixels in all the 4x4 predicted blocks are the same as shown in (4.23) and HT of this block is shown in (4.24). Therefore, HT of the predicted block 0, shown in (4.24), can be reused for all the other 4x4 DC predicted blocks.

$$B0 = B2 = B8 = B10 = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 \\ h_0 & h_1 & h_2 & h_3 \\ h_0 & h_1 & h_2 & h_3 \\ h_0 & h_1 & h_2 & h_3 \end{bmatrix} \tag{4.19}$$

$$\mathrm{HT(B0) = HT(B2) = HT(B8) = HT(B10) =}$$
$$\begin{bmatrix} 4(h_0 + h_1 + h_2 + h_3) & 4(h_0 + h_1 - h_2 - h_3) & 4(h_0 - h_1 - h_2 + h_3) & 4(h_0 - h_1 + h_2 - h_3) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.20}$$

$$B0 = B2 = B8 = B10 = \begin{bmatrix} v_0 & v_0 & v_0 & v_0 \\ v_1 & v_1 & v_1 & v_1 \\ v_2 & v_2 & v_2 & v_2 \\ v_3 & v_3 & v_3 & v_3 \end{bmatrix} \tag{4.21}$$

$$HT(B0) = HT(B1) = HT(B4) = HT(B5) = \begin{bmatrix} 4(v_0 + v_1 + v_2 + v_3) & 0 & 0 & 0 \\ 4(v_0 + v_1 - v_2 - v_3) & 0 & 0 & 0 \\ 4(v_0 - v_1 - v_2 + v_3) & 0 & 0 & 0 \\ 4(v_0 - v_1 + v_2 - v_3) & 0 & 0 & 0 \end{bmatrix} \tag{4.22}$$

	h_0	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}	h_{12}	h_{13}	h_{14}	h_{15}
v_0	0				1				4				5			
v_1																
v_2																
v_3																
v_4	2				3				6				7			
v_5																
v_6																
v_7																
v_8	8				9				12				13			
v_9																
v_{10}																
v_{11}																
v_{12}	10				11				14				15			
v_{13}																
v_{14}																
v_{15}																

Figure 4.6 16x16 MB and its Neighboring Pixels

$$B0 = ... = B15 = \begin{bmatrix} p & p & p & p \\ p & p & p & p \\ p & p & p & p \\ p & p & p & p \end{bmatrix}$$

$$where\ p = \left(\sum_{i=0}^{15} (h_i + v_i) + 16 \right) >> 5 \tag{4.23}$$

$$HT(B0) = ... = HT(B15) \begin{bmatrix} 16p & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.24}$$

Intra 16x16 mode decision algorithm also includes applying HT to 4x4 DC blocks formed by DC coefficients of HT of each 4x4 block shown in Figure 4.6. We propose to apply the same technique to 4x4 DC blocks as well. After HT is applied to current block (C) and predicted block (P), a 4x4 DC block is formed by DC coefficients of HT of C and a 4x4 DC block is formed by DC coefficients of HT of P as shown in (4.6). Then, HT is applied to these 4x4 DC blocks and the results are subtracted. 4x4 DC blocks formed by DC coefficients of HT of predicted blocks by intra modes have the same block patterns as the HT of predicted blocks themselves. For example, 4x4 DC block formed by DC coefficients of HT of predicted block by vertical mode is shown in (4.25) and its HT is shown in (4.26). It has 4 nonzero elements same as the HT of predicted block itself. Horizontal mode is similar to vertical mode. 4x4 DC block formed by DC coefficients of HT of predicted block by horizontal mode is shown in (4.27) and its HT is shown in (4.28). 4x4 DC block formed by DC coefficients of HT of predicted block by DC mode is shown in (4.29) and its HT is shown in (4.30). It has 1 nonzero element same as the HT of predicted block itself. Therefore, HT of 4x4 DC blocks for each intra 16x16 prediction mode can be calculated with small amount of computation by using the proposed technique.

Intra 8x8 Vertical, Horizontal and DC modes are very similar to corresponding intra 16x16 modes except that no 4x4 DC block is formed. Therefore, similar computation reductions are achieved for intra 8x8 Vertical, Horizontal and DC modes.

$$4\times\begin{bmatrix} h_0+h_1+h_2+h_3 & h_4+h_5+h_6+h_7 & h_8+h_9+h_{10}+h_{11} & h_{12}+h_{13}+h_{14}+h_{15} \\ h_0+h_1+h_2+h_3 & h_4+h_5+h_6+h_7 & h_8+h_9+h_{10}+h_{11} & h_{12}+h_{13}+h_{14}+h_{15} \\ h_0+h_1+h_2+h_3 & h_4+h_5+h_6+h_7 & h_8+h_9+h_{10}+h_{11} & h_{12}+h_{13}+h_{14}+h_{15} \\ h_0+h_1+h_2+h_3 & h_4+h_5+h_6+h_7 & h_8+h_9+h_{10}+h_{11} & h_{12}+h_{13}+h_{14}+h_{15} \end{bmatrix} \quad (4.25)$$

$$HT(DC)=\begin{bmatrix} X & Y & Z & T \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

where

$$X = 16(h_0 + h_1 + h_2 + h_3 + h_4 + h_5 + h_6 + h_7 + h_8 + h_9 + h_{10} + h_{11} + h_{12} + h_{13} + h_{14} + h_{15})$$
$$Y = 16(h_0 + h_1 + h_2 + h_3 + h_4 + h_5 + h_6 + h_7 - h_8 - h_9 - h_{10} - h_{11} - h_{12} - h_{13} - h_{14} - h_{15})$$
$$Z = 16(h_0 + h_1 + h_2 + h_3 - h_4 - h_5 - h_6 - h_7 - h_8 - h_9 - h_{10} - h_{11} + h_{12} + h_{13} + h_{14} + h_{15})$$
$$T = 16(h_0 + h_1 + h_2 + h_3 - h_4 - h_5 - h_6 - h_7 + h_8 + h_9 + h_{10} + h_{11} - h_{12} - h_{13} - h_{14} - h_{15})$$

(4.26)

$$4\times\begin{bmatrix} v_0 + v_1 + v_2 + v_3 & v_0 + v_1 + v_2 + v_3 & v_0 + v_1 + v_2 + v_3 & v_0 + v_1 + v_2 + v_3 \\ v_4 + v_5 + v_6 + v_7 & v_0 + v_1 + v_2 + v_3 & v_0 + v_1 + v_2 + v_3 & v_0 + v_1 + v_2 + v_3 \\ v_8 + v_9 + v_{10} + v_{11} & v_8 + v_9 + v_{10} + v_{11} & v_8 + v_9 + v_{10} + v_{11} & v_8 + v_9 + v_{10} + v_{11} \\ v_{12} + v_{13} + v_{14} + v_{15} & v_{12} + v_{13} + v_{14} + v_{15} & v_{12} + v_{13} + v_{14} + v_{15} & v_{12} + v_{13} + v_{14} + v_{15} \end{bmatrix}($$

4.27)

$$HT(DC)=$$
$$16\times\begin{bmatrix} v_0 + v_1 + v_2 + v_3 + v_4 + v_5 + v_6 + v_7 + v_8 + v_9 + v_{10} + v_{11} + v_{12} + v_{13} + v_{14} + v_{15} & 0 & 0 & 0 \\ v_0 + v_1 + v_2 + v_3 + v_4 + v_5 + v_6 + v_7 - v_8 - v_9 - v_{10} - v_{11} - v_{12} - v_{13} - v_{14} - v_{15} & 0 & 0 & 0 \\ v_0 + v_1 + v_2 + v_3 - v_4 - v_5 - v_6 - v_7 - v_8 - v_9 - v_{10} - v_{11} + v_{12} + v_{13} + v_{14} + v_{15} & 0 & 0 & 0 \\ v_0 + v_1 + v_2 + v_3 - v_4 - v_5 - v_6 - v_7 + v_8 + v_9 + v_{10} + v_{11} - v_{12} - v_{13} - v_{14} - v_{15} & 0 & 0 & 0 \end{bmatrix}$$

(4.28)

$$DC=\begin{bmatrix} 16p & 16p & 16p & 16p \\ 16p & 16p & 16p & 16p \\ 16p & 16p & 16p & 16p \\ 16p & 16p & 16p & 16p \end{bmatrix}$$

(4.29)

$$HT(DC)=\begin{bmatrix} 256p & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(4.30)

4.2.3 HT of Predicted Blocks by Intra 16x16 and 8x8 Plane Mode

Plane mode is the most complex prediction mode and it constitutes almost 90% of addition and 100% of shift operations performed by 16x16 and 8x8 intra predictions. Plane mode first calculates a, b, c parameters from the neighboring pixels of the current MB. It then calculates the predicted pixels using a, b, c as shown in (2.7.d). If the following two small modifications are made in plane mode equations, HT of a block predicted by plane mode can be calculated with a very small amount of computation; Clip1 in (4.31) is removed (Clip1 is a function which clips the predicted pixel value between 0 and 255) and right shift by 5 in (4.31) is changed to divide by 32. The new plane mode equation shown in (4.32) is only used for calculating the cost of 16x16 and 8x8 plane modes for intra mode decision. If plane mode is selected by mode decision, the actual predicted pixels will be calculated using a, b, and c.

$$pred[x, y] = Clip1((a + b*(x-7) + c*(y-7) + 16) >> 5) \tag{4.31}$$

$$pred[x, y] = (a + b*(x-7) + c*(y-7) + 16)/32 \tag{4.32}$$

Modified plane mode equation shown in (4.32) simplifies HT of 16x16 plane mode significantly. Equation (4.33) shows HT of 16x16 plane mode.

Using the modified equation given in (4.32), we can calculate the cost of the plane mode by only using a, b, c parameters without calculating actual predicted pixels. Therefore, the number of additions and shifts performed by 16x16 and 8x8 intra prediction algorithms for intra mode decision is reduced by approximately 80%. As shown in (4.33) for modified plane mode, HT of predicted blocks 1,…,15 are exactly the same as HT of predicted block 0 except DC coefficient. Therefore, HT of predicted block 0 can be reused for all other predicted 4x4 blocks.

In addition, proposed technique can be applied to 4x4 DC block formed by DC coefficients of HT of predicted block by plane mode as well. 4x4 DC block formed by DC coefficients of HT of predicted block by plane mode is shown in (4.34) and its HT is shown in (4.35). As shown in (4.34) and (4.35), HT of plane mode as well as HT of its DC block can be found easily once a, b, c parameters are calculated.

$$\frac{1}{32}\times\begin{bmatrix}
16a-88b-88c+256 & -16b & 0 & -8b & 16a-24b-88c+256 & -16b & 0 & -8b & 16a+40b-88c+256 & -16b & 0 & -8b & 16a+104b-88c+256 & -16b & 0 & -8b \\
-16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-8c & 0 & 0 & 0 & -8c & 0 & 0 & 0 & -8c & 0 & 0 & 0 & -8c & 0 & 0 & 0 \\
16a-88b-24c+256 & -16b & 0 & -8b & 16a-24b-24c+256 & -16b & 0 & -8b & 16a+40b-24c+256 & -16b & 0 & -8b & 16a+104b-24c+256 & -16b & 0 & -8b \\
-16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-8c & 0 & 0 & 0 & -8c & 0 & 0 & 0 & -8c & 0 & 0 & 0 & -8c & 0 & 0 & 0 \\
16a-88b+40c+256 & -16b & 0 & -8b & 16a-24b+40c+256 & -16b & 0 & -8b & 16a+40b+40c+256 & -16b & 0 & -8b & 16a+104b+40c+256 & -16b & 0 & -8b \\
-16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-8c & 0 & 0 & 0 & -8c & 0 & 0 & 0 & -8c & 0 & 0 & 0 & -8c & 0 & 0 & 0 \\
16a-88b+104c+256 & -16b & 0 & -8b & 16a-24b+104c+256 & -16b & 0 & -8b & 16a+40b+104c+256 & -16b & 0 & -8b & 16a+104b+104c+256 & -16b & 0 & -8b \\
-16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 & -16c & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-8c & 0 & 0 & 0 & -8c & 0 & 0 & 0 & -8c & 0 & 0 & 0 & -8c & 0 & 0 & 0
\end{bmatrix} \tag{4.33}$$

$$\frac{1}{32}\times\begin{bmatrix}
16a-88b-88c+256 & 16a-24b-88c+256 & 16a+40b-88c+256 & 16a+104b-88c+256 \\
16a-88b-24c+256 & 16a-24b-24c+256 & 16a+40b-24c+256 & 16a+104b-24c+256 \\
16a-88b+40c+256 & 16a-24b+40c+256 & 16a+40b+40c+256 & 16a+104b+40c+256 \\
16a-88b+104c+256 & 16a-24b+104c+256 & 16a+40b+104c+256 & 16a+104b+104c+256
\end{bmatrix} \tag{4.34}$$

$$\begin{bmatrix}
8a+4b+4c+128 & -32b & 0 & -16b \\
-32c & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
-16c & 0 & 0 & 0
\end{bmatrix} \tag{4.35}$$

4.3 Computation Reduction for Residue Calculations

Residue calculations require more subtraction operations than the addition operations required by intra prediction. The proposed technique also significantly reduces the number of residue calculations in intra mode decision algorithm. The residue calculation is only needed for the nonzero elements in the HT of a predicted block. As shown in Figure 4.5, since HT of a 4x4 block predicted by DC prediction mode has only 1 nonzero element, only 1 residue calculation is needed and 15 subtraction operations are avoided for this 4x4 block. Similarly, since HT of a 4x4 block predicted by vertical and horizontal prediction modes have only 4 nonzero elements, 12 subtraction operations are avoided for each 4x4 block for vertical and horizontal prediction modes. In addition, as shown in (4.33), since HT of a 4x4 block predicted by modified plane mode has 5 nonzero elements, 11 subtraction operations are avoided during residue calculations for each 4x4 block.

4.4 Computation Reduction Results

We quantified the computation reductions achieved by the proposed technique for the SATD based intra mode decision algorithm used in H.264 JM software encoder version 14.0 [35]. For 4x4 modes the computation amounts for a 4x4 block and for 16x16 and 8x8 modes the computation amounts for a 16x16 MB are shown in Table 4.9. The columns labeled I show the amount of computation performed by the original SATD mode decision and the columns labeled II show the amount of computation performed by the SATD mode decision using the proposed technique. Since current block HT is common for both intra 16x16 and 4x4 mode decision, the results of the current block HT for intra 16x16 mode decision are reused for intra 4x4 mode decision. The results show that the proposed technique significantly reduces the computational complexity of SATD based intra 4x4, 16x16 and 8x8 mode decision algorithms.

Table 4.9 Computation Reductions for Intra Prediction Modes

	Prediction Modes	Hadamard Transform				Residue	
		Addition		Shift		Subtraction	
		I	II	I	II	I	II
Intra 4x4	Vertical	64	8	0	4	16	4
	Horizontal	64	8	0	4	16	4
	DC	64	0	0	1	16	1
	Diagonal down left	64	28	0	5	16	16
	Diagonal down right	64	32	0	5	16	16
	Vertical right	64	36	0	6	16	16
	Horizontal down	64	36	0	6	16	16
	Vertical left	64	36	0	6	16	16
	Horizontal up	64	25	0	6	16	16
	Total	576	209	0	43	144	105
Intra 16x16	Vertical	1088	40	16	36	256	52
	Horizontal	1088	40	16	36	256	52
	DC	1088	0	16	17	256	1
	Plane	1088	3	16	26	256	69
	Current block HT	0	1088	0	0	0	0
	Total	4352	1171	64	115	1024	174
Intra 8x8	Vertical	256	16	0	8	64	16
	Horizontal	256	16	0	8	64	16
	DC	256	0	0	4	64	4
	Plane	256	7	0	9	64	20
	Current block HT	0	256	0	0	0	0
	Total	1024	295	0	29	256	56

We also quantified the impact of the proposed modifications for the 16x16 and 8x8 plane mode equations on the rate-distortion performance of the SATD based intra mode decision algorithm used in H.264 JM reference software encoder version 14.0. Rate distortion curves and average PSNR comparison of the original SATD mode decision and the SATD mode decision using modified plane mode equations for several CIF size benchmark video frames are shown in Figures 4.7 and 4.8, and Table 4.10. The average

PSNR values shown in Table 4.10 are calculated using the technique described in [49]. The proposed plane mode equation modifications don't affect the PSNR for Football, they increase the PSNR slightly for Foreman and Mother&Daughter, and they decrease the PSNR slightly for other video frames shown in Table 4.10.

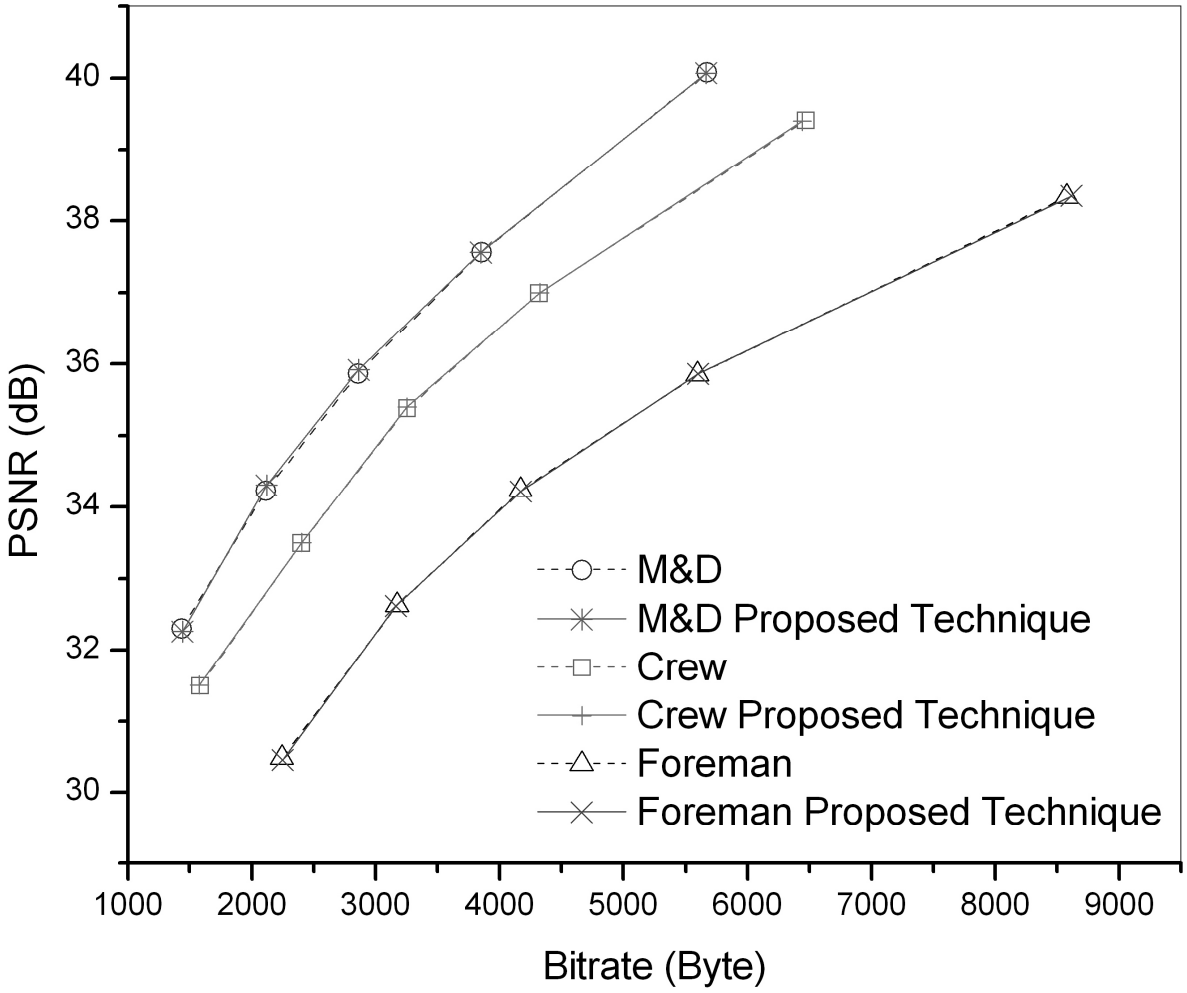

Figure 4.7 Rate Distortion Curves of the Original SATD Mode Decision and SATD Mode Decision with Proposed Technique for Mother&Daughter (M&D), Crew and Foreman

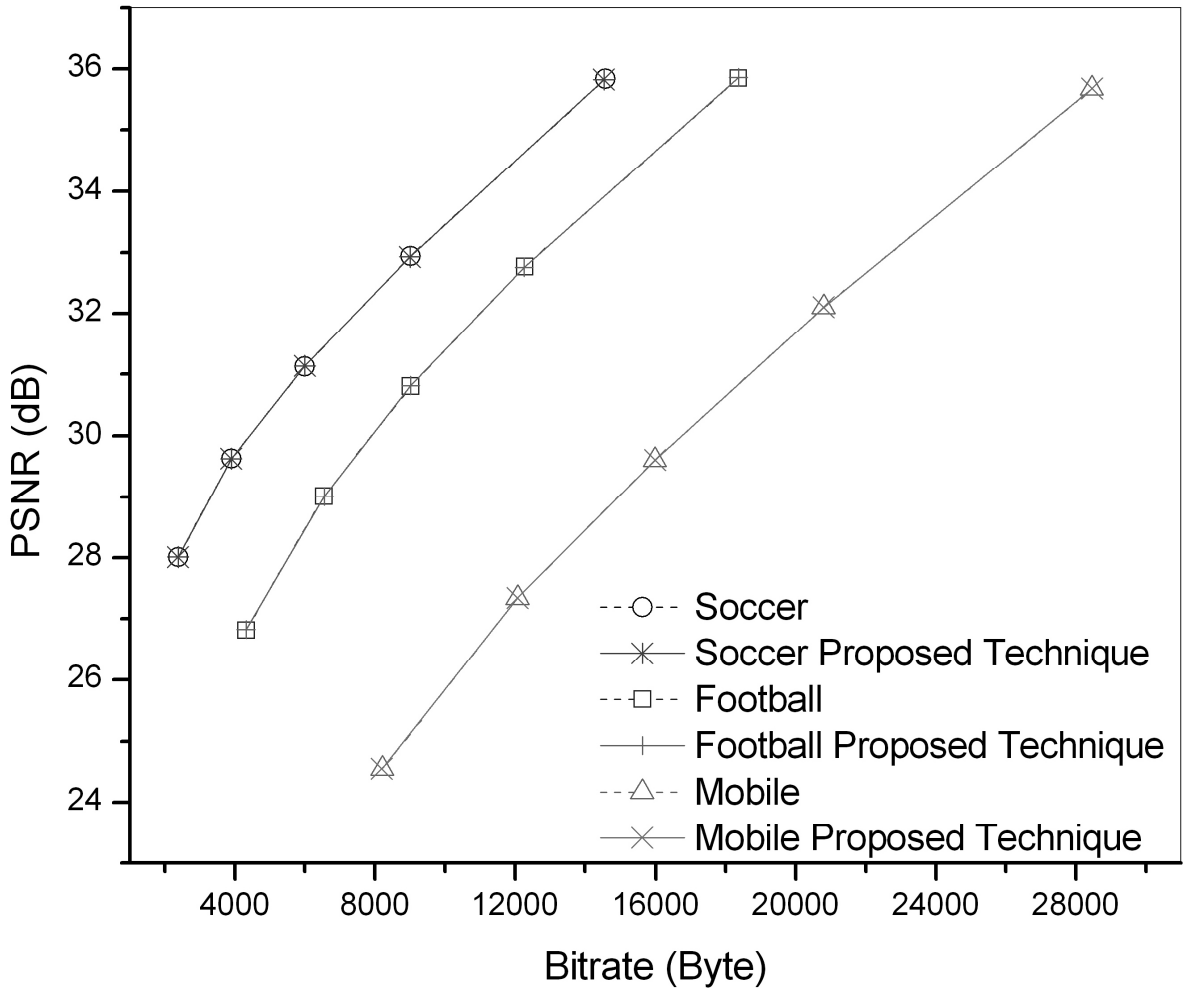

Figure 4.8 Rate Distortion Curves of the Original SATD Mode Decision and SATD Mode Decision with Proposed Technique for Soccer, Football and Mobile

Table 4.10 Average PSNR Comparison of the Original SATD Mode Decision with Proposed Technique

VIDEO FRAME	Original (dB)	Proposed Technique (dB)	Difference (dB)
MOBILE	30.638	30.634	-0.004
MOTHER&DAUGHTER	36.804	36.815	0.011
FOREMAN	35.279	35.303	0.024
FOOTBALL	31.980	31.980	0
CREW	36.157	36.122	-0.035
HARBOUR	31.580	31.579	-0.001
SHORTCIF	31.828	31.822	-0.006
SOCCER	32.461	32.448	-0.013
AKIYO	37.253	37.226	-0.027

4.5 Proposed 16x16 Intra Mode Decision Hardware Architectures

We designed two different hardware architectures for H.264 16x16 intra mode decision. The first hardware architecture, shown in Figure 4.9, implements the original SATD intra mode decision algorithm used in H.264 JM software encoder. The second hardware architecture, shown in Figure 4.10, includes the proposed computational complexity and power reduction technique.

H.264 16x16 intra mode decision hardware consists of two parts; the first part generates predicted blocks by each prediction mode in parallel and the second part calculates SATD cost for each prediction mode using the predicted blocks. The main differences between two architectures are the residue operation and simplification of HT because of fixed prediction block pattern of each intra mode. The first hardware architecture first performs the residue operation and then performs HT. The second hardware architecture, on the other hand, first performs HT and then performs residue operation.

As shown in Figure 4.9, three local buffers are used to store the inputs to intra mode decision hardware; 352x8 top neighboring buffer, 16x8 left neighboring buffer and 256x8 current block buffer. Horizontal predicted block (16x8), vertical predicted block (16x8), DC predicted block (1x8), and plane predicted block (256x8) are used to store the predicted blocks by the corresponding intra prediction modes. Residue block (256x8) is used to store the difference between the current MB and the predicted MB. Top neighboring buffer, plane predicted block, current block, and residue block are implemented as Block SelectRAMs, and other buffers are implemented as Distributed SelectRAMs.

Whenever a new MB arrives, the intra prediction module starts to calculate prediction values for each mode in parallel. After intra prediction is finished, the mode decision hardware starts to process each mode by subtracting predicted block from current block. HT is applied to residue block and absolute values of resulting AC coefficients are added. Then, HT is applied to the 4x4 DC block formed by DC coefficients and the resulting coefficients are added. HT module in Figure 4.9 implements the fast HT algorithm described in section 4.1.

As shown in Figure 4.10, the proposed hardware for SATD intra mode decision with proposed computational complexity and power reduction technique has the same intra prediction search hardware except that the size of the buffer used for storing the predicted block by plane mode is reduced from 256x8 to 3x8. Since this hardware calculates the SATD cost using only a, b, c parameters, it stores only a, b, c parameters.

After intra prediction, HT is applied to predicted blocks by each prediction mode. Since HT of horizontal, vertical, DC and plane prediction modes simplify significantly using the proposed technique, HT module in Figure 4.10 is much simpler than HT module in Figure 4.9. After HT, AC coefficients of predicted blocks by each prediction mode are subtracted from corresponding AC coefficients of transformed current block. HT is applied to DC coefficients of both current block and predicted blocks by each prediction mode again. Then, DC coefficients of predicted blocks by each prediction mode are subtracted from corresponding DC coefficients of current block. Finally, absolute values of AC and DC coefficient differences are added to find SATD cost.

4.6 Power Consumption Analysis

The power consumption of 16x16 intra mode decision hardware including proposed technique on a Xilinx Virtex II FPGA is estimated using Xilinx XPower tool. In order to estimate its power consumption, timing simulation of the placed and routed netlist of 16x16 intra mode decision hardware is done using Mentor Graphics ModelSim SE. Foreman, Akiyo and Mother&Daughter frames are used as inputs for timing simulations and the signal activities are stored in VCD files. These VCD files are used for estimating the power consumption of 16x16 intra mode decision hardware using Xilinx XPower tool.

The power consumption of 16x16 intra mode decision hardware implementation on a Xilinx Virtex II FPGA at 25 MHz is shown in Table 4.11 for different video frames at QP = 42. As shown in the table, the proposed power reduction technique reduces the power consumption of 16x16 intra mode decision hardware up to 39.2%.

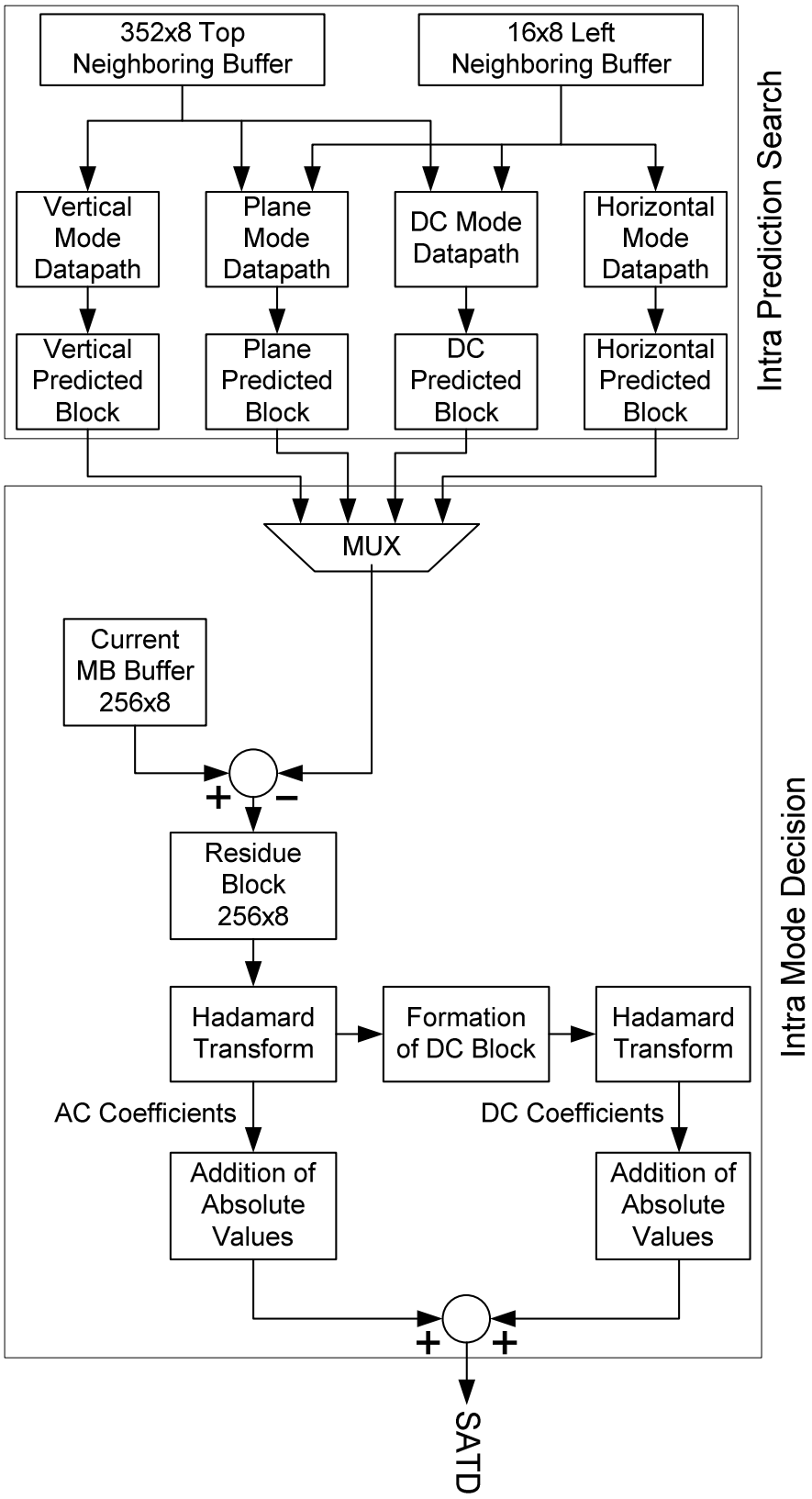

Figure 4.9 Proposed Hardware for Original Intra 16x16 Mode Decision

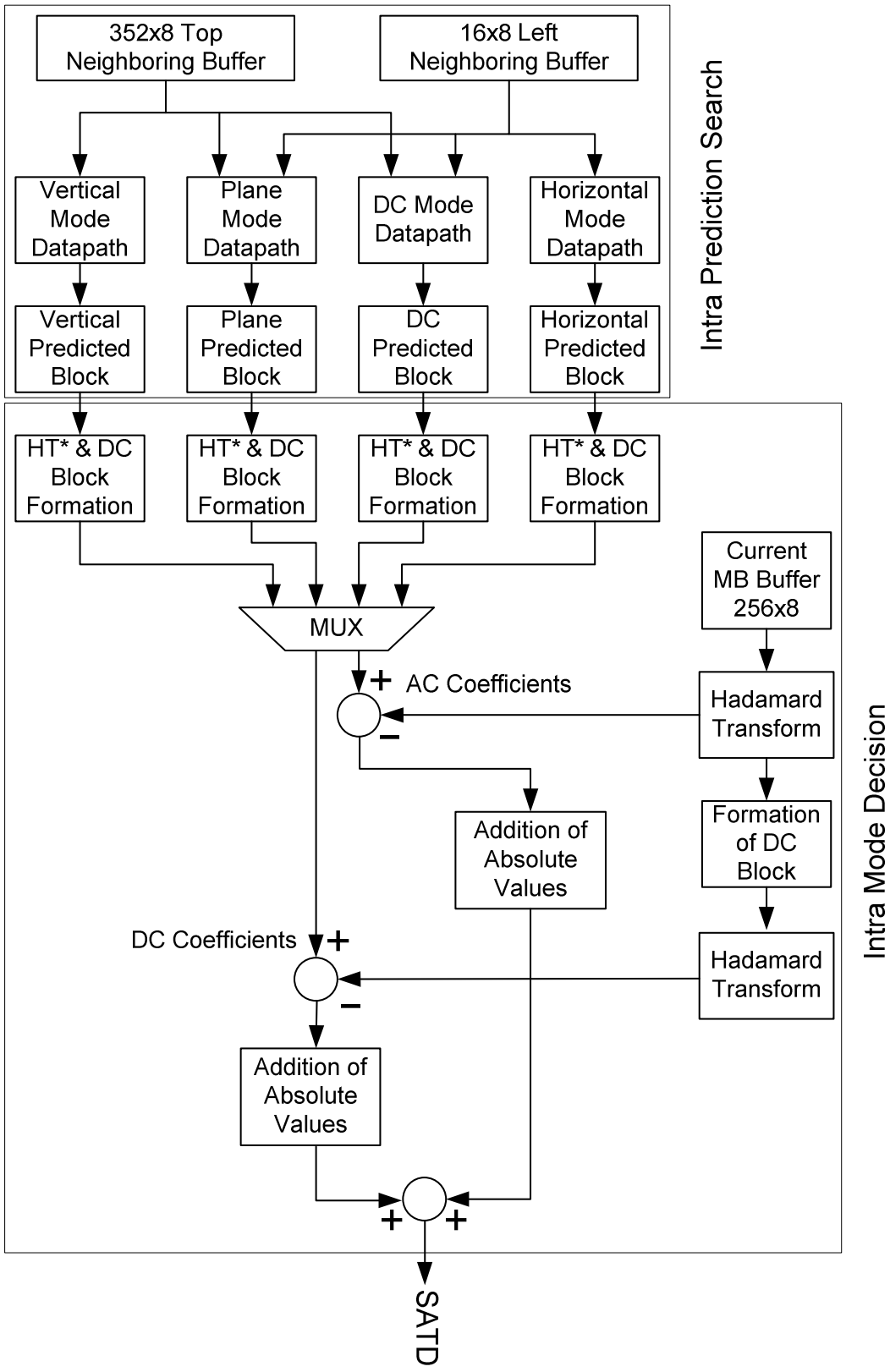

Figure 4.10 Proposed Hardware for Intra 16x16 Mode Decision with Proposed Technique

Table 4.11 Power Consumption Reduction of Intra 16x16 Mode Decision Hardware (QP=42)

Frames	Category	Power (mW)		
		Intra 16x16 Mode Decision Hardware	**Intra 16x16 Mode Decision Hardware with Power Red. Tech.**	**Reduction Percent.**
Foreman	Clock	51.6	41.1	20.35%
	Logic	38.19	23.46	38.57%
	Signal	83.65	44.52	46.78%
	Total	173.44	109.08	37.11%
Akiyo	Clock	51.6	41.1	20.35%
	Logic	35.43	21.66	38.87%
	Signal	81.83	41.98	48.70%
	Total	168.86	104.74	37.97%
Mother Daughter	Clock	51.6	41.1	20.35%
	Logic	34.73	19.85	42.84%
	Signal	78.69	38.92	50.54%
	Total	165.02	99.87	39.48%

CHAPTER V

CONCLUSIONS AND FUTURE WORK

In this thesis, we proposed low power hardware designs for DBF, intra prediction and intra mode decision parts of an H.264 video encoder. The proposed hardware architectures are implemented in Verilog HDL and mapped to Xilinx Virtex II FPGA. We performed detailed power consumption analysis of FPGA implementations of these hardware designs using Xilinx XPower tool. We also measured the power consumptions of DBF hardware implementations on a Xilinx Virtex II FPGA by measuring the average currents before DBF hardware is running on the FPGA and while DBF hardware is running on the FPGA. There is a good match between estimated and measured power consumption results which shows that power estimation results obtained by using Xilinx XPower tool are very accurate.

We worked on decreasing the power consumption of FPGA implementations of these H.264 video compression hardware designs by reducing switching activity using Register Transfer Level (RTL) low power techniques. We applied several RTL low power techniques such as clock gating and glitch reduction to these designs and quantified their impact on the power consumption of the FPGA implementations of these designs.

We proposed novel computational complexity and power reduction techniques which avoid unnecessary calculations in DBF, intra prediction and intra mode decision parts of an

H.264 video encoder. We quantified the computation reductions achieved by the proposed techniques using H.264 JM software encoder version 14.0. We applied these techniques to these hardware designs and quantified their impact on the power consumption of the FPGA implementations of these designs.

As a future work, the impact of the computational complexity and power reduction technique proposed for H.264 DBF on the power consumption of FPGA implementations of H.264 DBF hardware designs can be quantified. The computational complexity and power reduction technique proposed for H.264 intra prediction can be extended for the cases where some of the neighboring pixels are equal or there is a small difference between the neighboring pixels. The neighboring pixels used in intra prediction calculations can be divided into small groups in order to increase the likelihood that the neighboring pixels in a group are equal. The equivalence check between neighboring pixels in a group can be performed between most significant bits of these pixels in order to reduce the computational complexity even further at the expense of PSNR loss.

The proposed computational complexity and power reduction techniques can be applied to other parts of an H.264 video encoder such as motion estimation. Power consumption analysis of ASIC implementations of the proposed hardware designs can be performed and the impact of the proposed computational complexity and power reduction techniques on the power consumption of the ASIC implementations of the proposed designs can be quantified using Synopsys PrimePower tool.

BIBLIOGRAPHY

[1] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, "Draft ITU-T Recommendation and Final Draft InternationalStandard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC," May 2003.

[2] Thomas Wiegand and Heiko Schwarz Ralf Schäfer, "The emerging H.264/AVC standard," Heinrich Hertz Institute, Berlin, EBU Technical Review 2003.

[3] I. Richardson, *H.264 and MPEG-4 Video Compression*.: Wiley, 2003.

[4] G. J. Sullivan, G. Bjøntegaard, and A. Luthra T. Wiegand, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[5] Anthony Joch, Faouzi Kossentini, Antti Hallapuro Michael Horowitz, "H.264/AVC Baseline Profile Decoder Complexity Analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 704-716, July 2003.

[6] P.E. Ross, "Beat the heat," *IEEE Spectrum*, vol. 41, no. 5, p. 38–43, May 2004.

[7] The International Technology Roadmap for Semiconductors. (2005 Edition) [Online]. http://www.itrs.net/Links/2005ITRS/Home2005.htm

[8] C.A. Papachristou, M. Nourani and M. Spining, "A Multiple Clocking Scheme for Low-power RTL Design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 2, pp. 266 – 276, June 1999.

[9] G.E. Tellez, A. Farrahi and M. Sarrafzadeh, "Activity-driven Clock Design for Low Power Circuits," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 62-65, November 1995.

[10] L. Benini, P. Siegel and G. De Micheli, "Saving Power by Synthesizing Gated Clocks for Sequential Circuits," *IEEE Design & Test of Computers*, vol. 11, no. 4, pp. 32 – 41, 1994.

[11] A. Raghunathan, S. Dey and N.K. Jha, "Register Transfer Level Power Optimization with Emphasis on Glitch Analysis and Reduction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 8, pp. 1114 – 1131, August 1999.

[12] Seteven J. E. Wilton, Su-Shin Ang and Wayne Luk, "The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays," *International Conference on Field-Programmable Logic and Applications*, pp. 719-728, August 2004.

[13] L. Benini, G. De Micheli and E. Macii, "Designing Low-power Circuits: Practical Recipes," *IEEE Circuits and Systems Magazine*, vol. 1, no. 1, pp. 6 – 25, 2001.

[14] K.J.R. Liu , An-Yeu Wu, A. Raghupathy and Jie Chen, "Algorithm-based Low-power and High-performance Multimedia Signal Processing," *Proceedings of the IEEE Volume 86*, pp. 1155 – 1202, June 1998.

[15] A. Acquaviva, L. Benini and B. Ricco, "An Adaptive Algorithm for Low-power Streaming Multimedia Processing," *Design, Automation and Test in Europe*, pp. 273 – 279, March 2001.

[16] Yu-Wen Huang, To-Wei Chen, Bing-Yu Hsieh, Tu-Chih Wang, Te-Hao Chang, Liang-Gee Chen, "Architecture Design for Deblocking Filter in H.264/JVT/AVC," in *International Conference on Multimedia and Expo*, pp. I - 693-6 vol.1, July 2003.

[17] T. C. Wang, Y. W. Huang, H. C. Fang and L. G. Chen, "Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC / H.264," *IEEE ISCAS*, 2003.

[18] Tu-Chih Wang, Yu-Wen Huang, Hung-Chi Fang, Liang-Gee Chen, "Performance Analysis of Hardware Oriented Algorithm Modifications in H.264," *Multimedia and Expo ICME*, pp. 601-604, July 2003.

[19] W.T. Staehler, E.A. Berriel, A.A. Susin and S. Bampi, "Architecture of an HDTV Intraframe Predictor for a H.264 Decoder," *14th Int. Conference on VLSI-SoC*, pp. 228 –

233, October 2006.

[20] Y. W. Huang, B. Y. Hsieh, T. C. Chen and L. G. Chen, "Hardware Architecture Design for H.264/AVC Intra Frame Coder," *IEEE ISCAS*, pp. 269-272, April 2004.

[21] Y. W. Huang, T. C. Chen, C. H. Tsai, C. Y. Chen, T. W. Chen, C. S. Chen, C. F. Shen, S. Y. Ma, T. C. Wang, B. Y. Hsieh, H. C. Fang, L. G. Chen, "A 1.3TOPS H.264/AVC Single-Chip Encoder for HDTV Applications," *IEEE ISSCC*, pp. 586 – 588, February 2005.

[22] T. A. Lin, T. M. Liu and C. Y. Lee, "A Low Power H.264/AVC Decoder," *Int. Symposium on VLSI Technology, System and Applications*, April 2005.

[23] T. M. Liu, T. A. Lin, S. Z. Wang, W. P. Lee, K. C. Hou, J. Y. Yang and C. Y. Lee, "A 125-μW, Fully Scalable MPEG-2 and H.264/AVC Video Decoder for Mobile Applications," *IEEE Journal of Solid-State Circuits*, pp. 161 – 169, February 2006.

[24] T. M. Liu and C. Y. Lee, "Memory-Hierarchy-Based Power Reduction for H.264/AVC Video Decoder," *IEEE Int. Symposium on VLSI Design, Automation and Test* ,pp. 1 – 4, April 2006.

[25] Mustafa Parlak and Ilker Hamzaoglu, "Low Power H.264 Deblocking Filter Hardware Implementations," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 2, pp. 808 - 816, May 2008.

[26] Mustafa Parlak, Yusuf Adibelli and Ilker Hamzaoglu, "A Novel Computational Complexity and Power Reduction Technique for H.264 Intra Prediction," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 4, pp. 2006-2014, November 2008.

[27] Peter List, Anthony Joch, Jani Lainema, Gisle Bjøntegaard and Marta Karczewicz "Adaptive Deblocking Filter," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 614-619, July 2003.

[28] Mustafa Parlak and Ilker Hamzaoglu, "An Efficient Hardware Architecture for H.264 Adaptive Deblocking Filter Algorithm," *NASA/ESA Conference on Adaptive Hardware and Systems,* pp. 381-385, June 2006.

[29] Mustafa Parlak and Ilker Hamzaoglu, "A Low Power Implementation of H.264 Adaptive Deblocking Filter Algorithm," *NASA/ESA Conference on Adaptive Hardware and Systems,* pp. 127-133, August 2007.

[30] Shih-Chien Chang, Wen-Hsiao Peng, Shih-Hao Wang and Tihao Chiang "A Platform Based Bus-interleaved Architecture for De-blocking Filter in H.264/MPEG-4 AVC," *IEEE Transactions on Consumer Electronics*, vol. 51, no. 1, pp. 249-255, February 2005.

[31] Chao-Chung Cheng and Tian-Sheuan Chang "An Hardware Efficient Deblocking Filter for H.264/AVC," *International Conference on Consumer Electronics,*pp. 235 – 236, January 2005.

[32] Heng-Yao Lin, Jwu-Jin Yang, Bin-Da Liu and Jar-Ferr Yang, "Efficient Deblocking Filter Architecture for H.264 Video Coders," *IEEE International Symposium on Circuits and Systems,* pp. 2617-2620, May 2006.

[33] G. Khurana, A.A. Kassim, Tien Ping Chua, M. B. Mi, "A Pipelined Hardware Implementation of in-loop Deblocking Filter in H.264/AVC," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 2, pp. 536-540, May 2006.

[34] Versatile Platform Baseboard for ARM926EJ-S User Guide, May 2004. [Online]. http://www.arm.com

[35] Joint Video Team of ITU-T VCEG and ISO/IEC MPEG. Joint Model Reference Software, Version 14.0. [Online]. http://iphome.hhi.de/suehring/tml

[36] Feng Pan, Xiao Lin, S. Rahardja, K.P. Lim, Z.G. Li, Dajun Wu, Si Wu, "Fast Mode Decision Algorithm for Intraprediction in H.264/AVC Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 7, p. 813 – 822, July 2005.

[37] I. Choi, J. Lee and B. Jeon, "Fast Coding Mode Selection With Rate-Distortion Optimization for MPEG-4 Part-10 AVC/H.264," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 12, p. 1557 – 1561, December 2006.

[38] An-Chao Tsai, A. Paul, Jai-Ching Wang and Jhing-Fa Wang, , "Efficient Intra Prediction in H.264 Based on Intensity Gradient Approach," *ISCAS 2007*, pp. 3952 – 3955, May 2007.

[39] Ling-Jiao Pan and Yo-Sung Ho, "A Fast Mode Decision Algorithm for H.264/AVC Intra Prediction," *IEEE Workshop on Signal Processing Systems*, pp. 704 – 709, October 2007.

[40] Yu-Wen Huang, B. Y. Hsieh, T. C. Chen and L. G. Chen, "Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 15, no. 3, pp 378 – 401, March 2005.

[41] Genhua Jin, Jin-Su Jung and Hyuk-Jae Lee, "An Efficient Pipelined Architecture for H.264/AVC Intra Frame Processing," *IEEE ISCAS*, pp. 1605 – 1608, May 2007.

[42] Ilker Hamzaoglu, Ozgur Tasdizen and Esra Sahin, "An Efficient H.264 Intra Frame Coder System Design," *IFIP/IEEE International Conference on VLSI-SoC*, pp. 200 – 205, October 2007.

[43] Esra Sahin and Ilker Hamzaoglu, "An Efficient Hardware Architecture for H.264 Intra Prediction Algorithm," *DATE*, April 2007.

[44] Ilker Hamzaoglu, Ozgur Tasdizen and Esra Sahin, "An Efficient H.264 Intra Frame Coder System," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 4, pp. 1903 - 1911, November 2008.

[45] C. H. Tseng, H. M. Wang and J. F. Yang, "Enhanced Intra-4x4 Mode Decision for H.264/AVC Coders," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 8, pp. 1027-1032, August 2006.

[46] Changsung Kim and C. C. J. Kuo, "Feature-Based Intra-/InterCoding Mode Selection for H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 4, pp. 441 - 453, April 2007.

[47] Jia-Ching Wang, Jhing-Fa Wang, Jar-Ferr Yang and Jang-Ting Chen, "A Fast Mode Decision Algorithm and Its VLSI Design for H.264/AVC Intra-Prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 10, pp. 1414 - 1422 , October 2007.

[48] H.-M. Wang, C.-H. Tseng, and J.-F. Yang, "Computation Reduction for Intra 4x4 Mode Decision with SATD Criterion in H.264/AVC," *IET Signal Processing*, vol. 1, no. 2, pp. 121 - 127, September 2007.

[49] G. Bjontegaard, "Calculation of average PSNR differences between RD curves," 13th Video Coding Experts Group Meeting, VCEG-M33 2001.