Efficient H.264 Encoder with Main Profile for High Frame Rate Video Streams

Submitted by:

Fariaa Faheem	2019-EE-001
Marwa Waseem	2019-EE-007
Hamza Akhtar	2019-EE-012
Filza Shahid	2019-EE-151

Supervised by: Dr. Muhammad Tahir

Department of Electrical Engineering

University of Engineering and Technology Lahore

Efficient H.264 Encoder with Main Profile for High Frame Rate Video Streams

Submitted to the faculty of Electrical Engineering
of the University of Engineering and Technology Lahore
in partial fulfillment of the requirements for the Degree of

Bachelor of Science

in

Electrical Engineering.

Internal Examiner	External Examiner

Director
Undergraduate Studies

Department of Electrical Engineering

University of Engineering and Technology Lahore

Declaration

I declare that the work contained in this thesis is my own, except where explicitly stated otherwise. In addition this work has not been submitted to obtain another degree or professional qualification.

Signed: _____

Date: _____

Acknowledgments

We would like to express our sincere gratitude to our supervisor Dr. Muhammad Tahir and co-supervisor Sir Umer Shahid, for their unwavering support and guidance throughout this project. We would like to thank Ms. Shehzeen Malik for her advice, constructive criticism and assistance in keeping our progress on schedule. We would like to acknowledge the contribution of our friends and families for their support during the development of project.

Dedicated to our families.

Contribution to Sustainable Development Goals

Our project is a part of a long term project of **Open Eyes**,the idea is to develop a main RISC-V based processor, a video processor, ISP and integrate them on ChipYard to fabricate an IC.This project goes beyond sustainability by tackling the issue of data storage and security in the camera industry. Security camera manufacturers often retain the data collected by their cameras on the servers and then analyze it. This practice, however is alarming as it raises concerns about the data privacy, potential misuse of data and breach susceptibility.

Every thing related to camera is being developed from scratch under this project, this eliminates the risk of data misuse by any third part data storage and substantially enhances data security. Our strategy will ensure that the data stays in the hands of the camera owners. This will empower the camera industry and the audience of it by putting privacy, security, and data sovereignty as top priorities that contribute to sustainable practices.

Our project will also empower the organizations and individuals to maintain a high level of trust in the camera technology they use and to make informed decisions about their data.

Contents

Acknowledgments iii

Contribution to Sustainable Development Goals v

List of Figures viii

List of Tables ix

Abbreviations x

Abstract xi

1 Introduction 1
- 1.1 H264 Profiles . . . 1
  - 1.1.1 Baseline Profile . . . 1
  - 1.1.2 Main Profile . . . 2
  - 1.1.3 High Profile . . . 2
- 1.2 H264 Process . . . 2
  - 1.2.1 Forward Path . . . 3
  - 1.2.2 Reconstruction Path . . . 3
- 1.3 H264 Major Components . . . 3
  - 1.3.1 Prediction . . . 3
  - 1.3.2 Intra-Prediction . . . 4
  - 1.3.3 Inter-Prediction . . . 4
    - 1.3.3.1 Motion Vector Prediction . . . 4
  - 1.3.4 Transform . . . 5
  - 1.3.5 Quantization . . . 5
  - 1.3.6 Entropy Encoding . . . 5
- 1.4 H264 syntax . . . 6

2 Motivations and Problem Statement 8
- 2.1 Motivations . . . 8
- 2.2 Why H264 is more efficient? . . . 8
- 2.3 Project Continuation . . . 8
- 2.4 Need for further Improvements . . . 9

3 Prediction, Transformation, Quantization and Encoding 10
- 3.1 Predicton . . . 10
- 3.2 Prediction . . . 10

3.2.1 Intra-Prediction . . . 10
3.2.1.1 4x4 Luma Prediction . . . 11
3.2.1.2 8x8 Chroma Prediction . . . 11
3.2.2 Inter-Prediction . . . 12
3.2.2.1 Block Matching Technique . . . 13
3.3 Transform Coding . . . 13
3.3.1 DC Transform . . . 13
3.3.2 Core Transform . . . 14
3.4 Quantization . . . 15
3.5 Encoding . . . 16
3.5.1 Context Adaptive Variable Length Coding,CAVLC . . . 16

4 Hardware Architecture for Inter-Prediction 18
4.1 Full Search ME Algorithm . . . 18
4.2 256 PE VBS FS ME Hardware Architecture . . . 19
4.2.1 PI-PPSAD Architecture for SAD Calculation . . . 21
4.2.2 Zig-Zag Search Pattern . . . 22
4.2.3 Data Flow in 265 PEs . . . 22

5 Results and Performance 25
5.1 Performance Evaluation of Baseline Profile . . . 25
5.1.1 Effect of Different Values for QP . . . 25
5.2 Performance Evaluation of Inter-Prediction Module . . . 25
5.2.1 Resource Utilization . . . 26
5.2.2 Simulation Results . . . 26

6 Conclusions and Future Directions 28

References 29

List of Figures

1.1 Top Level Diagram for H.264 Encoder . . . 3
1.2 Intra-Prediction . . . 4
1.3 Inter-Prediction . . . 4
1.4 Motion Vector Prediction . . . 5
1.5 Overall syntax for h264 encoder . . . 7

3.1 Reference samples for 4x4 Luma . . . 11
3.2 Major Modes for 4x4 Luma Prediction . . . 12
3.3 Major Modes for 8x8 Chroma Prediction . . . 12
3.4 Motion Estimation using Block Matching . . . 13

4.1 265 PE VBS ME Hardware Architecture . . . 19
4.2 16 x 16 PE Matrix . . . 20
4.3 Working of an Adder Tree . . . 20
4.4 PI-PPSAD Architecture . . . 21
4.5 Zig-Zag Search Pattern . . . 22

5.1 Schematic . . . 26
5.2 Simulation waveform of inter-prediction module for one search range. . . . 27

List of Tables

4.1 Data Flow of PEs in 256PE VBS ME Hardware (a) 23
4.2 Data Flow of PEs in 256PE VBS ME Hardware (b) 24

5.1 Size and quality of encoded video for different QP values. 25
5.2 Resource utilization of inter-prediction module for Artix-7. 26

Abbreviations

AVC

MB

VBS

PE

ME

FBS

MV

DCT

QP

FS

CAVLC

¡¡¡¡¡¡¡ Updated upstream ======= **CABAC**

¿¿¿¿¿¿¿ Stashed changes **RAM**

NAL

VCL

CBP

FPGA

¡¡¡¡¡¡¡ Updated upstream ======= **FDCT**

¿¿¿¿¿¿¿ Stashed changes **DFT**

DST

QP

¡¡¡¡¡¡¡ Updated upstream ======= **VLC**

¿¿¿¿¿¿¿ Stashed changes

Abstract

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Chapter 1

Introduction

Digital video refers to a sequence of images displayed on a screen at a predetermined rate. The process of transferring video content from one medium to another depends on both the duration of the video and the size of the encoded bits. **Video compression** techniques are commonly utilized to reduce the overall size of the video, which consequently leads to a decrease in the amount of data required for transmission and storage of digital video signals.

The most current video coding standard is known as **H.264/AVC or MPEG-4 Part-10**. This standard was jointly developed by the **ITU-T Video Coding Experts Group** and **ISO/IEC JTC 1 Moving Picture Experts Group**. Compared to typical video coding standards, H.264/AVC provides significantly higher efficiency, capable of reducing bit rate requirements by up to 50% while maintaining the same level of video quality. It is designed to cover a wide range of video resolutions, from QCIF to HDTV [7].

1.1 H264 Profiles

The H.264 family of standards includes various capabilities. These profiles are mainly used to reduce the frame count by implementing motion prediction and temporal compression [9]. The most common ones are:

- Baseline Profile
- Main Profile
- High Profile

1.1.1 Baseline Profile

In the realm of video encoding, baseline profiles are commonly employed in applications that require low-power consumption and cost-efficiency. These profiles are capable of achieving an impressive compression ratio of 1000:1, resulting in a streamlet of 1

Gbps being compressed down to approximately 1 Mbps. Baseline profiles utilize a 4:2:0 chrominance sampling method, whereby color information is sampled at half the vertical and horizontal resolution of the black-and-white information. This technique enables the reduction of data without significantly impacting the overall quality of the video.

Furthermore, Universal Variable Length Coding (UVLC) and Context Adaptive Variable Length Coding (CAVLC) are employed as the primary entropy encoding techniques within this profile. Such encoding methods contribute significantly to the efficient compression of video data, while ensuring that the encoded video stream is in compliance with relevant standards.

1.1.2 Main Profile

Significant enhancements were made to the Baseline Profile through the introduction of advanced frame prediction algorithms, resulting in the development of the Main Profile. This updated profile is primarily utilized for standard-definition digital TV broadcasts in MPEG-4 format.

However, it should be noted that the Main Profile is not employed in high-definition broadcasts. Rather, alternate profiles are used in such scenarios to ensure optimal video quality and compatibility with relevant standards.

1.1.3 High Profile

Introduced in the year 2004, the High Profile is considered to be the most efficient and powerful profile within the H.264 family. It is primarily utilized in high-definition television applications such as Blu-ray Disc storage, Digital Video Broadcasting (DVB) and HDTV broadcast services. This profile is capable of achieving an exceptional compression ratio of 2000:1, which is a significant improvement over previous encoding standards. It utilizes an adaptive transform method that allows for the selection of either 4x4 or 8x8 pixel blocks. This enables preservation of video quality while reducing network bandwidth consumption by up to 50%.

Furthermore, the application of this compression technique facilitates the compression of a 1 Gbps stream to approximately 512 Kbps, further emphasizing the impressive capabilities of the High Profile.

The overall procedure of H.264 includes various components. The top level block diagram of an H.264 Encoder is shown in Figure 1.1.

1.2 H264 Process

An H264 encoder has a **forward path** and a **reconstruction path** [1]. The forward path uses **intra** and **inter predictions** to encode a video frame to create a bit stream. The reconstruction path is used to decode the encoded frame and to reconstruct the decoded frame. Reconstruction path in encoder ensures that both encoder and decoder make use of similar reference frames for inter and intra prediction. This is to avoid

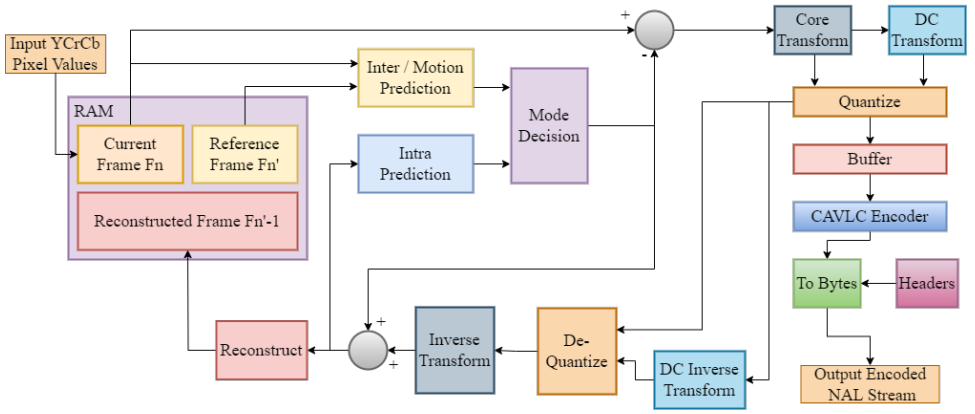

FIGURE 1.1: Top Level Diagram for H.264 Encoder

encoder-decoder mismatches.

1.2.1 Forward Path

The input frame is partitioned into **Macro-Blocks (MB)**. These MB are then encoded in intra or inter mode. This depends on mode decision. The current MB is predicted from reconstructed frame. This predicted MB is generated by intra prediction based on **spatial redundancy**, and by inter prediction based on **temporal redundancy**. The mode is chosen based on better quality and bit rate performance of these 2 modes. The Predicted MB is subtracted from current MB to create a **Residual MB**. Residual data is **transformed** (4x4 integer transform), then **quantized**. The obtained coefficient are re-ordered in a **zig-zag order** which are regarded as **entropy encoded**. These coefficients along with header information form the **compressed bit stream**. This stream is forwarded to NAL for storage or transmission.

1.2.2 Reconstruction Path

This path takes quantized transform coefficients and performs inverse quantization and inverse transform. In this way, reconstructed residual data is generated, but they are not identical to original residual data as quantization is a lossy process. In order to create the reconstructed frame, the reconstructed residual data are added to predicted pixels.

1.3 H264 Major Components

1.3.1 Prediction

In order to guarantee a high compression ratio in H.264 encoders, prediction is a technique utilized [10]. In prediction, a 16x16 pixel block known as a macroblock from a previous video frame or the present frame is utilized to forecast macroblocks in the current frame. Detailed information on predictions and its types are given in chapter 3.

There are basically 2 modes for prediction:

1.3.2 Intra-Prediction

Intra prediction is performed without referring to any data outside the current slice i.e prediction from previously coded data in the same slice. It reduces spatial redundancies by exploiting spatial correlation between adjacent blocks in a given picture. There are 3 choices of block size for luma component i.e. 16x16, 8x8 or 4x4. Whereas for chroma component, a single prediction block is generated. Once, the prediction has been made, it is subtracted from current block to make a residual. The overview of the process is shown in figure 1.2.

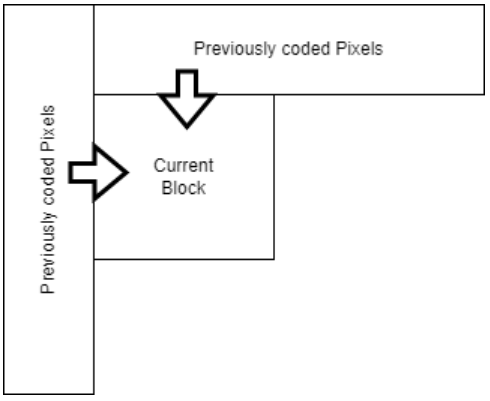

FIGURE 1.2: Intra-Prediction

1.3.3 Inter-Prediction

Inter prediction is the process of predicting a block of luma and chroma samples from a picture that has been previously coded and transmitted i.e reference picture. It uses temporal sampling technique. For this, a prediction region is selected, then a prediction block is generated. After that the prediction block is subtracted from original block of samples to form a residual. The overview of the process is shown in figure 1.3.

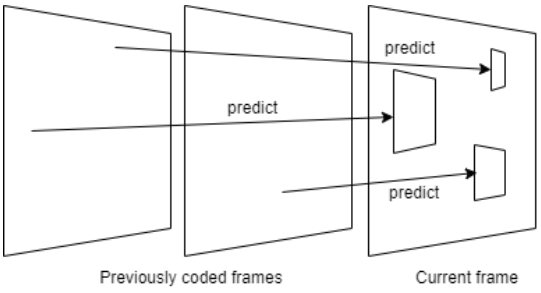

FIGURE 1.3: Inter-Prediction

1.3.3.1 Motion Vector Prediction

In video coding, motion vectors for neighboring partitions are typically closely related, and so each motion vector can be predicted using previously coded vectors from nearby partitions. A predicted vector is created based on these previous motion vectors, and the

difference between the current vector and the predicted vector is encoded and transmitted. The way in which predicted vector is predicted depends on the size of the motion compensation partition and whether there are nearby vectors available for reference. For further details related to its algorithm and hardware development, refer to chapter 4.

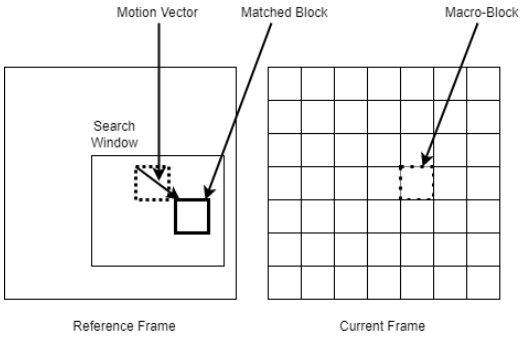

FIGURE 1.4: Motion Vector Prediction

1.3.4 Transform

Once the residual data, which essentially comprises a block of residual coefficients, is obtained, it is subjected to the core transform process. This transform is an integer-based 4x4 or 8x8 transform that provides a scaled approximation to the Discrete Cosine Transform (DCT) [10]. In certain cases, a portion of the output from this integer transform is subject to further transformation through a DC Transform, referred to as the Hadamard transform [10]. The same residual data can be reconstructed using the DC inverse transform, which is carried out prior to rescaling. Finally, the rescaled coefficients are inverse transformed using a 4x4 or 8x8 inverse integer transform. To gain more insight of this process, refer to chapter 3.

1.3.5 Quantization

The transformed coefficients are subjected to quantization using a non-uniform quantizer. In this process, each coefficient is divided by an integer value, which reduces the precision of the coefficient values as determined by the Quantization Parameter (QP). The use of a non-uniform quantizer results in a smaller number of bits being used to represent each coefficient value, which in turn reduces the amount of data required to represent the video. The quantized transform coefficients of a block are typically scanned in a zig-zag pattern, which is a common approach for video coding standards. For more details, refer to chapter 3.

1.3.6 Entropy Encoding

In H.264 stream or file encoding, the symbols are coded in a series. The quantized transform coefficients are efficiently transmitted using the Context-Adaptive Variable Length Coding (CAVLC) method. The statistical distribution of the quantized transform coefficients typically shows larger values for low-frequency components that decrease gradually

towards the high-frequency part. As a result, the number of nonzero quantized coefficients (N) and their size and position are coded separately. This enables the receiver to reconstruct the original signal more accurately. The coefficients are scanned in a zig-zag pattern, and then quantized to reduce their precision using the quantization parameter (QP). The efficient transmission of quantized transform coefficients in H.264 helps in reducing the size of the compressed video file while maintaining the quality of the video. The final Network Abstraction Layer can be seen in figure 1.5. Further details of this implementation are mentioned in chapter 3.

1.4 H264 syntax

H.264 consists of 2 layers: the **Network Abstraction Layer (NAL)** and the **Video Coding Layer (VCL)** [10]. The NAL consists of a series of NAL Units, with **Sequence Parameter Sets (SPS)** and **Picture Parameter Sets (PPS)** being the most common units that signal certain control parameters to the decoder. In the VCL, coded video data is communicated in the form of slices.An **access unit**, which can be a coded frame or field, is made up of one or more slices. Each slice consists of a Slice Header and Slice Data, with the latter being a series of coded **macro blocks (MB)** and skip macro block indicators signaling that certain macro block positions contain no data.

- **MB type:** I/intra coded, P/inter coded from one reference frame
- **Prediction information:** prediction mode for I macro block, choice reference frame and motion vectors for P macro block
- **Coded Block Pattern CBP:** indicates which luma and chroma blocks contain non zero residual co-efficient
- **Quantization Parameter QP:** for macro blocks with CBP not 0
- **Residual Data:** for blocks containing non-zero residual coefficients

The basic H264 syntax can be seen in figure 1.5

In our project, we also successfully implemented the Baseline profile on FPGA board. The synthesis and the results obtained can be accessed in detail in chapter 5. The rest if the thesis is organized as follows:

Chapter 2 gives an overview about the purpose of this project and mentions the problem statment of our project. Chapter 3 explains Intra 4x4 luma and 8x8 Chroma Prediction. It also gives an overview of Inter prediction and its Block matching technique. Further the Transform coding, Quantization and Entropy Encoding and explained in detail. Chapter 4 describes the Hardware Architecture for Inter Prediction in detail which is the main aim of our project. It explains the algorithm for carrying out the prediction of pixels. After that it explains the 256 PE VBS FS ME hardware architecture and

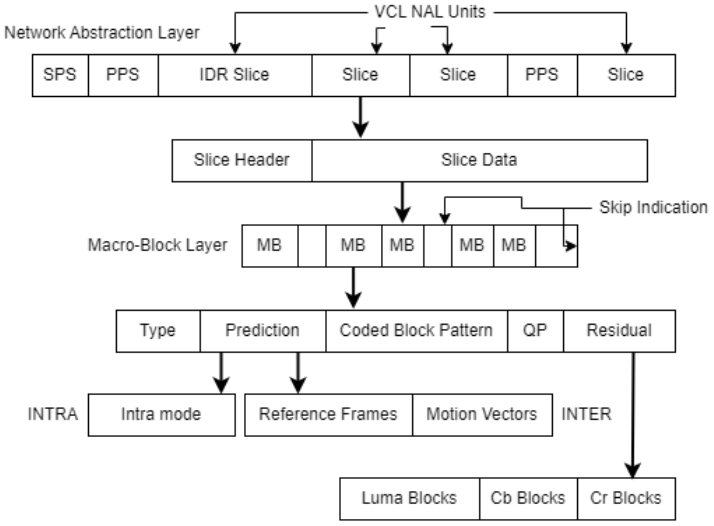

FIGURE 1.5: Overall syntax for h264 encoder

its implementation in detail. Chapter 5 follows with the simulation, results and the performance of the Baseline Profile and also the independent model of Inter Prediction. It also exhibits the implementation of Baseline Profile model of H.264 video encode on FPGA board and associated results. At last, the conclusion and future work are mentioned in Chapter 6.

Chapter 2

Motivations and Problem Statement

2.1 Motivations

With the rapid increase in the development of products and services offering full-motion digital video, digital video coding is currently gaining importance and has a considerable monetary impact on computer, imaging and telecommunications technology.There have been a lot of efforts to achieve an efficient video encoding format, which offers a better remote viewing quality at a lower bit-rate and uses less bandwidth than the previous standards (i.e., MPEG-2, H.263, or MPEG-4 Part 2). H.264 or MPEG- Part 10 has all these new, better compression features that makes it more emphatic.The primary features that set H.264 apart from competing standards include network friendliness and good video quality at both high and low bit rates. [2]

2.2 Why H264 is more efficient?

It is proposed that H264 can deliver two times better performance than the previous MPEG-2 coding standard, both in terms of compression efficiency and picture eminence. Moreover, previous H.63 and MPEG-4 implement block-based motion estimation to reduce temporal redundancy between frames. But in H.264, block matching efficiency is improved by some innovative features such as variable block size motion estimation (VBS_ME) and motion vector prediction [6]. The computational complexity of this model is increased by the factor of four due to these features.

2.3 Project Continuation

This is a **long-term project**, in collaboration with **10x Engineers** and team from **NUST**. A team from 10x worked on the ISP part of it whereas our group worked on the encoder part of the camera. This work is unique in Pakistan as very few engineers here work on SoC designs. This makes our project valuable as it can lead to new opportunities

for local research and it can make a significant impact on the video compression industry and contribute to the development of new technologies in Pakistan and beyond.

2.4 Need for further Improvements

In terms of encoding efficiency and computational complexity, there is still a need for further improvements in the Baseline profile of H.264 especially if we are planning to use this standard for real time applications. Our contribution is a significant step towards enhancing the over-all performance of the CODEC i.e. by optimizing the inter-prediction algorithm to reduce the bit-rate and improve the visual quality of the video.

Chapter 3

Prediction, Transformation, Quantization and Encoding

¡¡¡¡¡¡¡ Updated upstream

3.1 Predicton

=======

3.2 Prediction

¿¿¿¿¿¿¿ Stashed changes Prediction is defined as duplication of the information contained in a macro-block using previously coded data. This duplicated information is subtracted from the macro-block to form a residual. There are 2 types of prediction.

- Intra-Prediction
- Inter-Prediction

3.2.1 Intra-Prediction

Intra-prediction utilizes the space dependency to compress the video. The frames which are intra coded using intra-prediction are called I-frames. Following are the possible prediction modes:

- **4x4 luma:** having 9 directional modes and is suitable for macro blocks that has lot of details
- **8x8 luma:** having 9 directional modes and is for high profiles only.
- **16x16 luma:** having 4 directional modes that is suitable for macro block with smoother area
- **8x8 chroma:** 4 possible prediction modes and used for chrominance components

In our model, 4x4 luma prediction and 8x8 chroma prediction is being implemented.

3.2.1.1 4x4 Luma Prediction

For this type of prediction, each macro block that is of **16x16** (256 pixels each of which is 8 bit wide) is divided into **4x4** block (16 pixels) [3]. Figure 3.1 shows the reference samples for 4x4 luma prediction. 4 pixels **A,B,C,D** (adjacent to current block) of block **a** on top of current block, pixels **E,F,G,H** of block **b** on top right corner, **I,J,K,L** of block **c** at adjacent left of current block and 1 pixel **M** of **d** block on top left corner are used for prediction of 16 pixels in the current block.

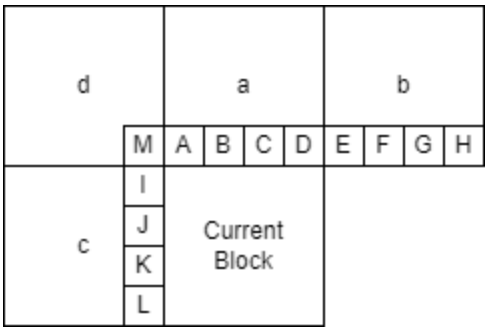

FIGURE 3.1: Reference samples for 4x4 Luma

There are total 9 prediction modes that are supported in this prediction. Major modes that are implemented in our model are as follows:

- **Mode 0 (Vertical):** The predicted block is constructed by using upper samples A,B,C,D of block 'a' as shown in figure 3.2. They are extrapolated vertically. It is suitable to predict vertical edges in the block.
- **Mode 1 (Horizontal):** In this mode, left samples I,J,K,L of block c are used. They are extrapolated horizontally and is suitable for horizontal edges. It can be seen in figure 3.2.
- **Mode 2 (DC):** It utilizes average of all adjacent samples (A to D and I to L) to form the prediction of current block. It is suitable for smooth areas. Its process is shown in figure 3.2.

For the details of remaining modes refer to [10]. Figure 3.2 display the above 3 prediction modes. To create a predict sample, every color stands for a particular formula. The encoder determines each prediction direction's cost by finishing processing for all of the prediction directions, then outputs the one with the lowest cost.

3.2.1.2 8x8 Chroma Prediction

This type of prediction applies on chrominance components. It is similar to 16x16 luma prediction which can be referred in [10] except the block size is 8x8 and there is different order of mode number which are:

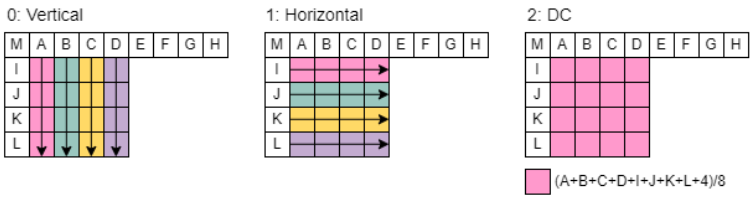

FIGURE 3.2: Major Modes for 4x4 Luma Prediction

- Mode 0: DC
- Mode 1: Horizontal
- Mode 2: Vertical
- Mode 3: Plane

The working of first 3 modes in similar to mode 2,1,0 of 4x4 luma prediction respectively. For details of Plane mode refer to [10]. The Implemented mode in our model is DC. These modes are shown in the figure 3.3.

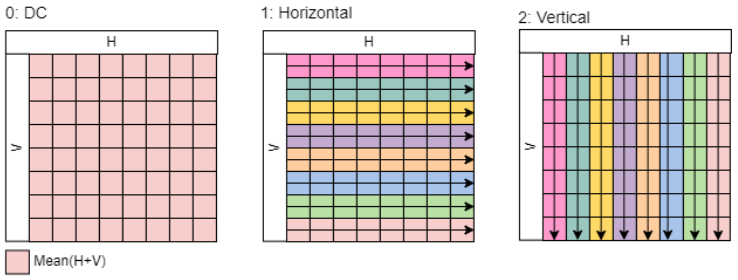

FIGURE 3.3: Major Modes for 8x8 Chroma Prediction

3.2.2 Inter-Prediction

The process of predicting a block of luma and chroma samples from a reference picture that has been previously been coded and transmitted i.e. exploits temporal redundancy between successive frames. For this a prediction region is selected, a prediction block is generated and then it is subtracted from original block of samples to form a residual. This is then coded and transmitted. Reference pictures are stored in Decoded Picture Buffer. The offset between position of current block and search region in the reference picture is called motion vector. This prediction is also known as **Motion Estimation** [4]. It has the capability of extracting true motion information thus enhancing the quality of displayed images in video enhancement systems. The preferred technique for motion estimation is the **Block Matching (BM)** Technique.

3.2.2.1 Block Matching Technique

This method divides the current frame into non-overlapping NxN macro-blocks and seeks out the block from the reference frame that most closely resembles the current block within a specified search range. The Sum of Absolute Difference (SAD), which is appropriate for hardware implementations, is the recommended block matching criterion.

In figure 3.4, **(x,y)** represents the location of the current frame. The search window in the reference frame is in **[-r,r]** region in both x and y directions. Both current and reference block lies within the range of search window. The SAD value is calculated by accumulating absolute differences of corresponding pixels in both current and reference blocks. A motion vector is the relative motion of current block in reference frame, they are specified in relative coordinates. Thus if **(x+a,y+b)** is the location of best matching block in reference frame, then **(a,b)** represents the motion vector. Motion Estimation is performed on luma component and resulting motion vectors are also used for chroma components.

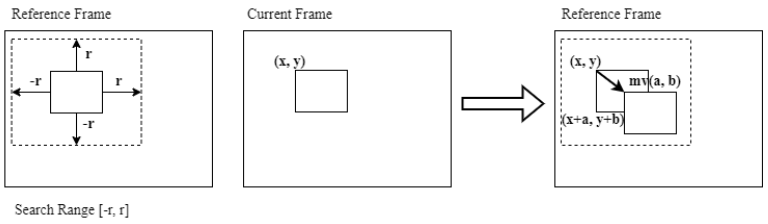

FIGURE 3.4: Motion Estimation using Block Matching

There are several algorithms for Block Matching. Among them mostly used is **Full Search (FS)** algorithm. The SAD values for each search position within a specific search range are calculated by this approach to determine the reference block that most closely resembles the present block. It has the best performance related to other algorithms as it searches all the search locations in a given search range. But its computational complexity is high and its hardware consume a lot of power. For further improvement, instead of **fixed block size (FBS) FS ME** algorithm, **variable block size (VBS) FS ME** algorithm is incorporated. For details of FBS ME algorithm, refer to [that ppr]. VBS FS ME algorithm will be further explained in detail in chapter 4.

3.3 Transform Coding

3.3.1 DC Transform

The DC transform is a type of discrete cosine transform (DCT) that only operates on the DC component (i.e., the average pixel value) of a 2x2 input block, and it is used to reduce the amount of spatial redundancy in the video data.
This DC transform is a rather simple operation, it calculates the average of all the pixels in the input block, and then subtracts that average value from each pixel. The resulting

values are then encoded and transmitted. For a 2x2 input block X, the sum of all the pixels is calculated as Given a 2x2 input block X, compute the sum of all the pixels in the block as:

$$SUM(X) = X(0,0) + X(0,1) + X(1,0) + X(1,1) \tag{3.1}$$

The average value of the input block is computed as:

$$AVG(X) = SUM(X)/4 \tag{3.2}$$

then the average value is subtracted from each pixel in the input block to obtain the transformed block Y, where:

$$Y(i,j) = X(i,j) - AVG(X) \tag{3.3}$$

In our code this dc transform is implemented in a pipe-lined manner, i.e. by using 4 stages. In each stage the input block is first divided into two sub-blocks, the transform is applied to each sub-block then these transformed sub-blocks are recombined at the end to form the output block. This pipe-lining is done to ensure the reduced latency and higher throughput of transform.

3.3.2 Core Transform

Core transform or Forward Discrete Cosine Transform (**FDCT**) is a linear transformation that transforms the given input data from spatial domain to frequency domain. This transform is **computationally less complex** and has **high energy compaction** as compared to **DFT**, **DST**, **WHT** and **DWT**. Therefore, it is highly preferred in video compression. FDCT uses A i.e. a transform matrix and its transpose to convert the input matrix or a martix of samples X into a result matrix Y i.e. an NxN block of coefficients, as shown in the equation below:

$$Y = AXA^T \tag{3.4}$$

Here, A in an NxN transform matrix, the equation for A_{ij} is given by:

$$A_{ij} = C_i \cos\left(\frac{(2j+1)i\pi}{2N}\right)$$

where

$$C_i = \begin{cases} \frac{1}{\sqrt{N}} & \text{if } i = 0 \\ \frac{\sqrt{2}}{\sqrt{N}} & \text{if } i > 0 \end{cases}$$

It can be simplified as a matrix, The matrix A is given by:

$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}$$

where

$$a = \frac{1}{2}, \quad b = \frac{1}{2}\cos\left(\frac{\pi}{8}\right), \quad c = \frac{1}{2}\cos\left(\frac{3\pi}{8}\right)$$

In our code, we have used 36-bit input, XXIN, which is a 4x9-bit matrix in little-endian order for our implementation of core transform. The transform generates a 14-bit output, YNOUT, in reverse zigzag order.
The output is only considered valid when the VALID signal is high. This VALID signal is controlled by the ENABLE signal. A READY signal that is used to indicate when the module is ready to accept input.
The output of core tranform is a set of coefficients in frequency domain, these coefficients, concentrate most of the signal energy in a few low frequency components. This property gives us efficient compression as the low magnitude coefficients are further discarded through quantization.

3.4 Quantization

The process of mapping the input values from a large set(typically a continuous set) into smaller often finite sized set is called Quantization. This helps in representing the data signal with fewer bits than the original one as the range of possible values in the quantized set in smaller.
We have used a **Scalar Quantizer** in our implementation, here each element on the matrix is quantized independently without considering the correlation between elements. For better understanding it can be considered as a process of rounding a fractional number to the nearest integer.
This process is irreversible and lossy since it is not possible to determine the exact value of the original fractional number from the rounded integer.
The formula used for quantization in our implementation is:

$$X_q = \text{round}\left(\frac{X}{2 \cdot QP} \cdot \text{scale}\right)$$

Here, **X** is the input value to be quantized. **QP** is the Quantization Parameter. The division by (2*QP) scales the input value based on the QP. **Scale** is the scaling factor and **round** function is used to round the scaled value to nearest integer, this scaled and rounded value is then multiplied by the scaled factor.

¡¡¡¡¡¡¡ Updated upstream =======

3.5 Encoding

Variable Length Encoding is an encoding technique typically used in H.264 main profile, such encoder maps the input data into a series of code words, called variable length codes (VLCs). Each symbol corresponds to a codeword, which can vary in length but always has to have a fixed amount of bits. Short VLCs are used to represent frequently occurring symbols, while lengthy VLCs are used to represent less often occurring symbols. This results in data compression when there are enough encoded symbols in a given amount of data.

3.5.1 Context Adaptive Variable Length Coding,CAVLC

Context-Adaptive Variable Length Coding also known as CAVLC is a method coding transformed and quantized coefficients,it is an important component in video compression, particularly in the H.264 video coding standard. This encoding technique uses context adaptation to select various sets of variable-length codes based on the statistics of recently-coded coefficients.
High compression efficiency is attained by CAVLC by utilising the statistical characteristics of video data. It improves compression over fixed-length coding methods by adjusting the coding scheme based on the local context of the coefficients.
CAVLC effectively represents the quantized coefficients using variable length codes, lowering the bitrate of the compressed video stream. As a result, the video data is represented in more concise ways.
Compared to other entropy coding methods like CABAC (Context-Adaptive Binary Arithmetic Coding), CAVLC encoding and decoding processes are computationally simpler. As a result, real-time video encoding applications are better suited for it.
After transformation and quantization, blocks usually contain a lot of zeros, CAVLC uses run-level coding to represent this string of zeros compactly. A block of coefficients is converted into a series of variable length codes (VLCs) using zigzag scan (or field scan). The highest non-zero coefficients are often the sequence of +1 or -1 (called high frequency coefficients) and these are signaled by CAVLC as 'Trailing 1s' (T1s) in a compact way.
The number of non-zero coefficients in neighbouring blocks is correlated and this number is encoded with the help of a look up table. CAVLC encoding proceeds as follows:

- First the lookup table encodes both number of non-zero coefficients and the number of Trailing 1s (T1s).
- Each T1 is signaled by lookup table, and the sign in encoded by a single bit i.e. 0 for '+' and 1 for '-', this is done in a reverse order starting from highest frequency T1.
- The sign and magnitude remaining non-zero coefficients is encoded in reverse order (from highest frequency back towards DC coefficients).

- Sum of all zeros before highest non-zero coefficient in the reverse ordered array is coded with a VLC.
- The number of zeros preceding each non-zero coefficient (run before) is encoded in reverse order.

¿¿¿¿¿¿¿ Stashed changes

Chapter 4

Hardware Architecture for Inter-Prediction

There are many new technologies such as intra prediction, in-loop deblocking filter and context based arithmetic coding introduced in the latest H.264/AVC standard. Among all of these amazing technologies, **Variable Block Size Motion Estimation (VBSME)** is one of the most powerful techniques. In comparison with the previous Fixed Block Size Motion Estimation (FBSME), VBSME divides one MB into smaller blocks to fit different motion directions. In this way, the coding performance is proved.

4.1 Full Search ME Algorithm

The Full Search Algorithm works in the following way:

- At first, both the search window and current block positions are fixed at the certain point i.e top left corner of frame. The current block starts the loading of the pixels from the top left corner. Absolute difference of each pixel of current block with the corresponding pixel of the search window is calculated.

$$Diff(m,n,k,l\ = |S(m+k,n+l) - C(k,l)| \tag{4.1}$$

- After that the Sum of all Absolute Differences (SAD) for that particular position of current block is calculated.

$$SAD(m,n) = \sum_{k=0}^{W-1}\sum_{l=0}^{H-1} Diff(m,n,k,l) \tag{4.2}$$

  Then the current block is shifted by 1 row or column of pixels and again SAD is calculated. In this way, several SADs are calculated in a single search window.

- Each SAD when calculated is compared with the previous SAD value and the smaller SAD value is taken. In this way at end, minimum SAD value and the

corresponding motion vectors values are depicted for one search window.

$$SAD_{min} = min(SAD(m, n)) \tag{4.3}$$

In the above equation (4.1), (4.2) and (4.3) The domain of m and n is $m \in [0, M-1]$ and $m \in [0, N-1]$ respectively.M and N are width and height of search window respectively. H and W are height and width of current block respectively. C(k,l) represents the pixel value of the current block and S(m+k, n+l) is the pixel value from the search window of the reference frame.[6]

4.2 256 PE VBS FS ME Hardware Architecture

We designed a parallel **256 PE VBS FSME Hardware Architecture**. [4] This hardware is implemented in System Verilog. First of all, the pixels in the current MB are stored in a Block Ram **(c_BRAM)**. The pixels of the search window are also stored in a block RAM **(s_BRAM)**. The architecture is shown in the figure 4.1

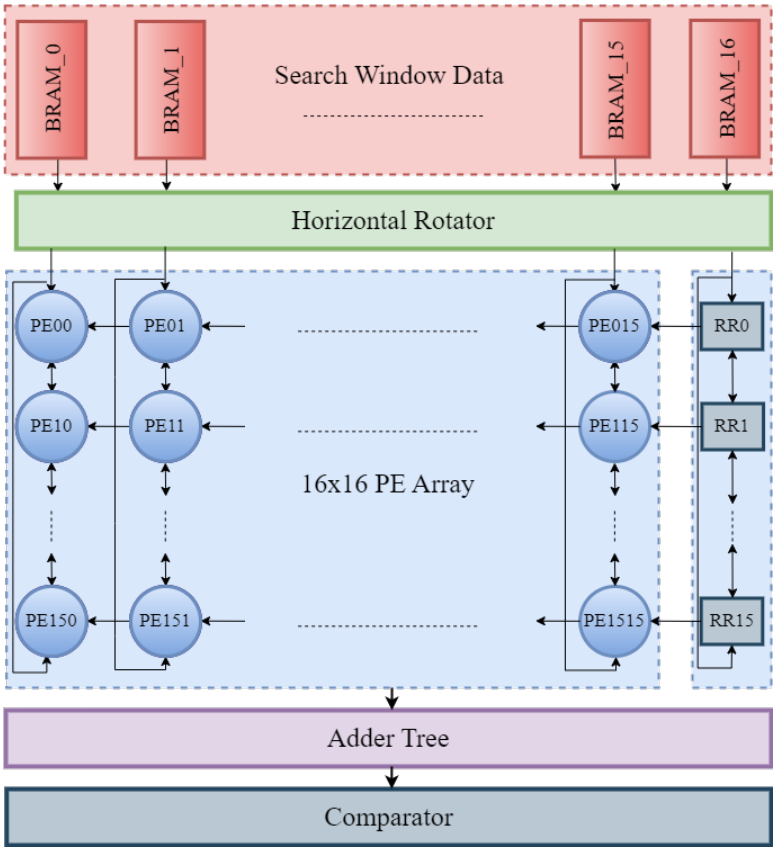

FIGURE 4.1: 265 PE VBS ME Hardware Architecture

In this design, a **2-D systolic PE array** is used. The use of this type of array introduced parallel computing and pipelinability in the structure. There are **16x16 = 256 PEs** (16 rows and 16 columns) interconnected with each other in a form of matrix. All of

them are made capable of shifting data down, up and left. It means that a 16x16 current block can move around the 48x48 search window in down, up and right direction. For a **16x16 MB**, a Motion Vector MV is found in one cycle in a search range of **[-16, 15]** pixels. Pixels are defined as 8 bit positive integers.

As soon as the control signals for loading current and search pixels are enabled, both current block and search window pixels starts loading into PEs. In 1 clock cycle, 16 pixels are loaded for both of them. (1st PE is filled in each of 16 columns). Thus, the whole PE matrix is filled in 16 cycles. As depicted in figure 4.2, **pixel_cpr_in** and **pixel_spr_in** are input array of 16 8-bit pixels from current block and search window of reference block respectively. The **pe**_matrix module concatenates the output signals of each column of processing elements to form the final output signal **ad** which is an array of 8*(16**2) bits.

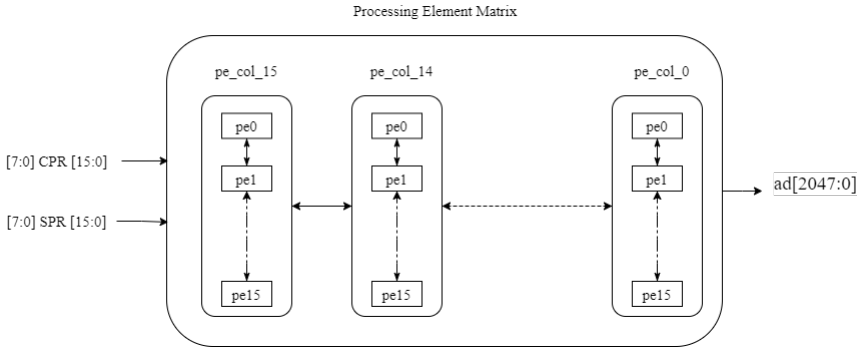

FIGURE 4.2: 16 x 16 PE Matrix

Each PE calculates the absolute difference between current MB pixel and the search window pixel. In this way 16x16 = 256 absolute differences are obtained in one location and they all are calculated simultaneously with the loading of data in PEs as the design is combinational.

After absolute differences are all calculated, The SAD for that search location is determined by adding the absolute differences calculated by the above PEs as shown in figure. All these absolute differences are added up together using Adder Trees. The working of a simple adder tree is shown in figure 4.3. It can seen clearly in the figure that there is 2 cycle latency for an adder tree.

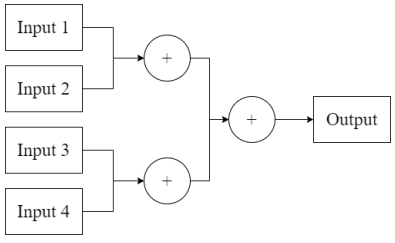

FIGURE 4.3: Working of an Adder Tree

4.2.1 PI-PPSAD Architecture for SAD Calculation

For SAD calculation of a 16x16 current block, we consider **PI-PPSAD** structure. The current pixels adn reference pixels are broadcasted into the respective PE arrays. This is shown by the vertical dash lines in the figure 4.4. Each row of PE determines the absolute difference with one row in macro block. One row SAD gets accumulated with partial SAD propagated in, which belongs to the same search position. After that, the result is propagated to the next stage in vertical.

In out Pi-PPSAD architecture, when the output of seventh PE row is obtained, inside one macro-Block, the SADs of the upper half partitions which include S4x4_YX (Y: 0-1 X: 0-3), S8x4_YX (Y: 0-1 X: 0-1), S8x8_0X (X: 0-1), they are depicted through one adder tree. In order to calculate S8x16_0, S8x16_1 and S16x16, the SADs of S8x8_00 and S8x8_01 are continue to be transmitted by making the use of shift registers. At the last stage of pipeline, they are summed with SADs of S8x8_10 and S8x8_11 to compute S8x16_0, S8x16_1 and the final SAD value depicted as S16x16 for that specific position of current and search block. This process is shown in figure 4.4.

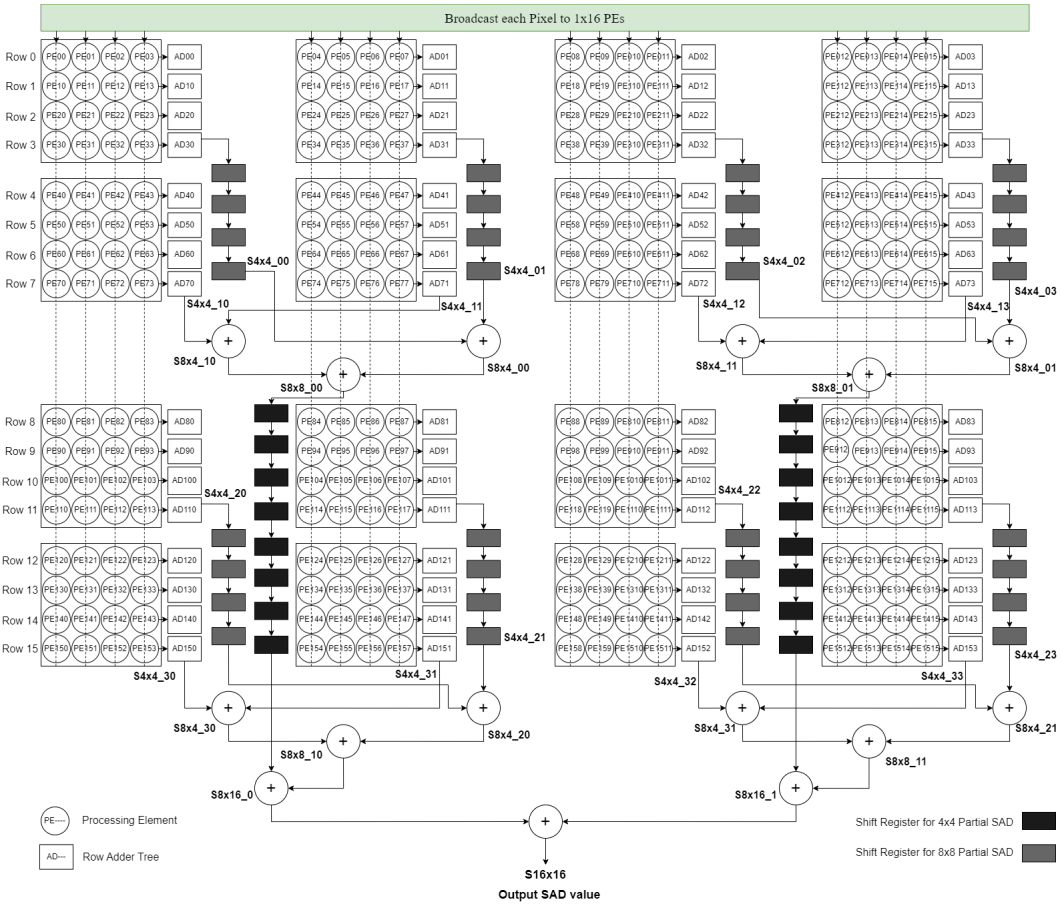

FIGURE 4.4: PI-PPSAD Architecture

Initially, pipelining causes latency of 16 cycles in the SAD calculation. But, once the pipeline is filled, only one cycle is needed per SAD value. Current block data is loaded

only once when the current block is changed while search are pixels are fed in every cycle to the PEs. Their data flow is further explained in table ?? and ??. As, there are 48 columns of reference pixels search range window [-16,15]. Thus, per 16x16 macro block, the approximate number of cycles is 48*32 = 1536 cycles.

4.2.2 Zig-Zag Search Pattern

The proposed hardware searches the search locations in a 48x48 search window in a zig-zag pattern as shown in figure 4.5. Usually most of the proposed ME architectures with 256 PEs make use of vertical search flow i.e start from top left corner and when end of the column is reached, the search location at top of the next column is searched. Thus, we have to either broadcast multiple pixels into the PEs or delay all the PEs until they are filled every time when we reach the end of the column and move to the top of the next [8].

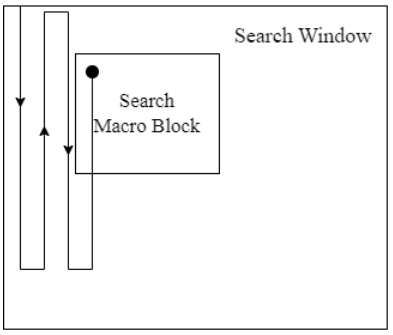

FIGURE 4.5: Zig-Zag Search Pattern

The zig-zag architecture used in our implementation can referred from [5]. But, this structure either use both row and column aligned memories or use data duplication. The implemented architecture overcomes this problem by introducing pipeline of 16 8-bit temporary registers. The search starts from top left corner of the search window and continues downwards until the last search location of this column is searched. After that, current block shifts to one right column in a way that pixel columns are shifted one left. Thus the search proceeds with the last search location of this next column and continues upwards until the first search location is reached. Only 16 new search window pixels are required by the current block PE array in each cycle to calculate the SAD of the next search location not keeping in fact its position in the search window.

4.2.3 Data Flow in 265 PEs

Data flow of PEs for fixed location of a current marco block pixels while searching all the locations in search window is shown in table ?? and ?? where sw(x,y) are the search window pixels.

When iterating the search locations in the 1^{st} column of search window, vertical up shift is performed in the PE array in each cycle. All PEs are provided the search window pixels from their neighboring PEs except for those PEs in the last row. These last row

PEs read 16 new search window pixels from 16 BRAMs in each cycle. The 17^{th} BRAM help in performing left shift after all search locations in a column has been searched. It is connected to temporary registers. When there is the need for left shift, the pixels need for the right most PEs become ready in those temporary registers. When we are done with the searches in the first 16 columns, a left shift is performed in the PE array and the PEs in the 16^{th} column receive search window pixels from temporary registers.

Clock	**1^{st} Column**				**2^{nd} Column**				
	PE(0,15)	**PE(0,14)**	...	**PE(0,0)**	**PE(1,15)**	**PE(1,14)**	...	**PE(1,0)**	...
0	sw(0,0)	none	...	none	sw(1,0)	none	...	none	...
1	sw(0,1)	sw(0,0)	...	none	sw(1,1)	sw(1,0)	...	none	...
...	...	...	...	...	...	...	...	...	...
14	sw(0,14)	sw(0,13)	...	none	sw(1,14)	sw(1,13)	...	none	...
15	sw(0,15)	sw(0,14)	...	sw(0,0)	sw(1,15)	sw(1,14)	...	sw(1,0)	...
16	sw(0,16)	sw(0,15)	...	sw(0,1)	sw(1,16)	sw(1,15)	...	sw(1,1)	...
17	sw(0,17)	sw(0,16)	...	sw(0,2)	sw(1,17)	sw(1,16)	...	sw(1,2)	...
...	...	...	...	...	...	...	...	...	...
45	sw(0,45)	sw(0,44)	...	sw(0,30)	sw(1,45)	sw(1,44)	...	sw(1,30)	...
46	sw(0,46)	sw(0,45)	...	sw(0,31)	sw(1,46)	sw(1,45)	...	sw(1,31)	...
47	sw(1,46)	sw(1,45)	...	sw(1,31)	sw(2,46)	sw(2,45)	...	sw(2,31)	...
48	sw(1,45)	sw(1,44)	...	sw(1,30)	sw(2,45)	sw(2,44)	...	sw(2,30)	...
...	...	...	...	...	...	...	...	...	...
77	sw(1,16)	sw(1,15)	...	sw(1,1)	sw(2,16)	sw(2,15)	...	sw(2,1)	...
78	sw(1,15)	sw(1,14)	...	sw(1,0)	sw(2,15)	sw(2,14)	...	sw(2,0)	...
1007	sw(31,46)	sw(31,45)	...	sw(31,31)	sw(32,46)	sw(32,45)	...	sw(32,31)	...
1008	sw(31,45)	sw(31,44)	...	sw(31,30)	sw(32,45)	sw(32,44)	...	sw(32,30)	...
...	...	...	...	...	...	...	...	...	...
1037	sw(31,16)	sw(31,15)	...	sw(31,1)	sw(32,16)	sw(32,15)	...	sw(32,1)	...
1038	sw(31,15)	sw(31,14)	...	sw(31,0)	sw(32,15)	sw(32,14)	...	sw(32,0)	...

TABLE 4.1: Data Flow of PEs in 256PE VBS ME Hardware (a)

Once left shift is performed, in each cycle, vertical down shift is performed and all the PEs are provided search window pixels from their neighbouring PEs except the PEs in the 1^{st} row. These 1^{st} row PEs read 16 new search window pixels from 16 BRAMs. This search pattern of data flow is also depicted in figure 4.5. There total 17 BRAMs and each BRAM store pixels in every 17^{th} column of the search window. The 1^{st} BRAM stores pixels in 1^{st}, 18^{th} and 35^{th} olumns, 2^{nd} BRAM fills in 2^{nd}, 19^{th} and 36^{th} columns and so the rest of BRAMs. The search window pixels which are read from the BRAMs have static order. But, the search window pixels that are required by PE array and temporary registers, the order for them varies depending on the columns being processed. Horizontal rotator hardware solve this problem by reordering the 16+1 pixels in a search macro block. Out of all the SADs that are being calculated during this searching, the minimum value and its corresponding motion vectors are depicted as output. This task is performed by comparator which compares the upcoming SAD value with the previously stored minimum value. After comparison, minimum one is depicted as output This type of Architecture requires large area and a lot of hardware

Clock		**16^{th} Column**				**Temp Column**			
	...	**PE(15,15)**	**PE(15,14)**	...	**PE(15,0)**	**reg 15**	**Reg 14**	...	**Reg 0**
0	...	sw(15,0)	none	...	none	sw(16,0)	none	...	none
1	...	sw(15,1)	sw(15,0)	...	none	sw(16,1)	sw(16,0)	...	none
...	...	...	...	...	...	...	...	...	...
14	...	sw(15,14)	sw(15,13)	...	none	sw(16,14)	sw(16,13)	...	none
15	...	sw(15,15)	sw(15,14)	...	sw(15,0)	sw(16,15)	sw(16,14)	...	sw(16,0)
16	...	sw(15,16)	sw(15,15)	...	sw(15,1)	sw(16,16)	sw(16,15)	...	sw(16,1)
17	...	sw(15,17)	sw(15,16)	...	sw(15,2)	sw(16,17)	sw(16,16)	...	sw(16,2)
...	...	...	...	...	...	...	...	...	...
45	...	sw(15,45)	sw(15,44)	...	sw(15,30)	sw(16,45)	sw(16,44)	...	sw(16,30)
46	...	sw(15,46)	sw(15,45)	...	sw(15,31)	sw(16,46)	sw(16,45)	...	sw(16,31)
47	...	sw(16,46)	sw(16,45)	...	sw(16,31)	sw(17,46)	sw(17,45)	...	sw(17,31)
48	...	sw(16,45)	sw(16,44)	...	sw(16,30)	sw(17,45)	sw(17,44)	...	sw(17,30)
...	...	...	...	...	...	...	...	...	...
77	...	sw(16,16)	sw(16,15)	...	sw(16,1)	sw(17,16)	sw(17,15)	...	sw(17,1)
78	...	sw(16,15)	sw(16,14)	...	sw(16,0)	sw(17,15)	sw(17,14)	...	sw(17,0)
1007	...	sw(46,46)	sw(46,45)	...	sw(46,31)	none	none	...	none
1008	...	sw(46,45)	sw(46,44)	...	sw(46,30)	none	none	...	none
...	...	...	...	...	...	...	...	...	...
1037	...	sw(46,16)	sw(46,15)	...	sw(46,1)	none	none	...	none
1038	...	sw(46,15)	sw(46,14)	...	sw(46,0)	none	none	...	none

TABLE 4.2: Data Flow of PEs in 256PE VBS ME Hardware (b)

implementation but the performance rate is high due to number of parallel computation and use of pipeline structure.

Chapter 5

Results and Performance

¡¡¡¡¡¡¡ Updated upstream

5.1 Performance Evaluation of Baseline Profile

5.1.1 Effect of Different Values for QP

In order to check the effect of quantization parameter on encoded video, the baseline profile is simulated using various values of QP. These results have been tabulated using a raw video of size **43.5 MB**.

QP	Encoded Video Size (MB)	SNR (dB)		
		Y	U	V
20	6.79	42.26	46.04	46.15
28	3.46	36.24	41.50	41.83
35	1.93	31.47	37.25	37.00

TABLE 5.1: Size and quality of encoded video for different QP values.

As discussed earlier in Chapter 3, the **Quantization Parameter** in quantization module determines to what extent the video should be compressed. A higher value of QP implies more compression, hence the size of encoded video is reduced but at the cost of poor video quality. The loss of information can be seen by reduced SNR values in the table above 5.1. On the contrary, a lower value of QP enhances the quality of video but at the expense of greater video size. Therefore, a moderate value for QP should be selected between 20-30 so that none of the metrics (size and quality) is compromised.

5.2 Performance Evaluation of Inter-Prediction Module

As stated in Chapter 4, the inter-prediction module developed for this project employs a 256 PE VBS ME hardware architecture. Thie module uses a datapath-controller design which is shown in the figure 5.1. This schematic is generated using Vivado.

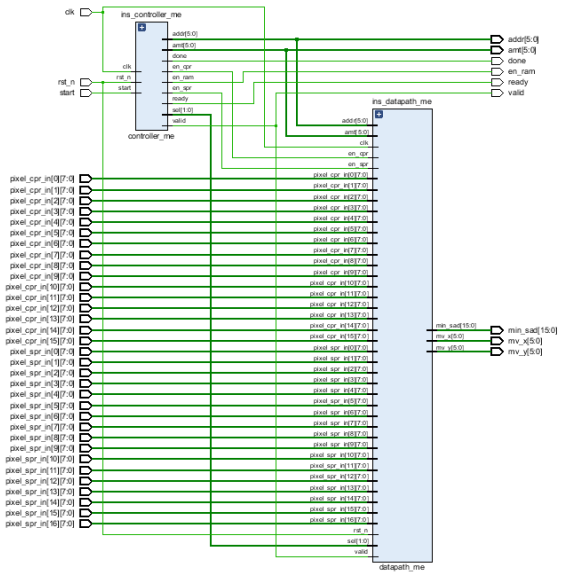

FIGURE 5.1: Schematic

5.2.1 Resource Utilization

The module for inter-prediction is analyzed and synthesized using Vivado for Xilinx Artix-7. The resource utilization for this module is shown below.

Resource	Utilization	Available	Utilization %
Look Up Tables	7376	63400	11.63
Registers	7266	126800	5.73
Input/Output	311	210	148.10

TABLE 5.2: Resource utilization of inter-prediction module for Artix-7.

It can be seen from the results above that the utilization of look up tables and registers is within limits but the input/output has exceeded from the available resources. This issue is due to the fact that inter-prediction block has been synthesized separately. Once this module is integrated within the H.264 top module, the input/output ports will be reduced sufficiently.

5.2.2 Simulation Results

In order to verify the working of inter-prediction module, it is simulated using a MB of size 16x16 from the current frame and a search range of size 48x48 from the reference frame. Once the MB has traversed the search range, it asserts a signal done and the final values for motion vectors in x and y direction, corresponding to the minimum SAD value are obtained.

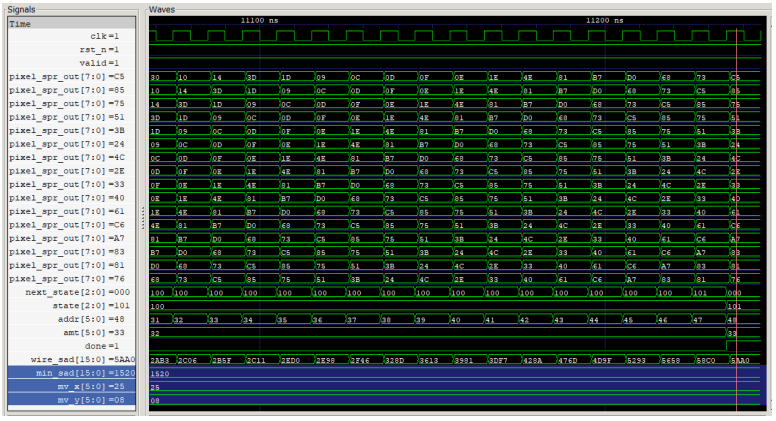

FIGURE 5.2: Simulation waveform of inter-prediction module for one search range.

=======

¿¿¿¿¿¿¿ Stashed changes

Chapter 6

Conclusions and Future Directions

References

[1] Esra Şahin. An efficient h.264 intra frame coder hardware design. Master thesis, Sabanci University, Electrical and Electronics Engineering Department (EECS), Istanbul, Turkey, 2006. URL https://suurcan.sabanciuniv.edu/fedora/objects/thesis:395/datastreams/SAHIN_ESRA_thesis_05042007_1217_02/pdf?origin=browser.

[2] Ihab Amer, Wael Badawy, and Graham Jullien. A high-performance hardware implementation of the h. 264 simplified 8× 8 transformation and quantization. In *Advanced Information Processing Systems*, pages 84–89. Springer, 2005.

[3] Anil Kumar C., Pradeep Kumar B.P., Venu K.N., and Lavanya Vaishnavi D.A. Intra prediction algorithm for video frames of h.264. *Nat. Volatiles & Essent. Oils*, 8(5):11357–11367, 2021.

[4] Caglar Kalaycioglu, Onur Can Ulusel, and Ilker Hamzaoglu. *Low power techniques for motion estimation hardware*. 34956 Tuzla, Istanbul, Turkey, 2011.

[5] M. Kim, I. Hwang, and S.-I. Chae. A fast vlsi architecture for full-search variable block size motion estimation in mpeg-4 avc/h.264. In *ASP DAC*, pages 631–634, January 2005.

[6] Brian MH Li and Philip HW Leong. Serial and parallel fpga-based variable block size motion estimation processors. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):624–635, 2003.

[7] Chia-Chi Lin, Wei-Cheng Chen, Ming-Hsiang Tsai, and Cheng-Wei Tsai. Parallel improved hdtv720p targeted propagate partial sad architecture for variable block size motion estimation in h.264/avc. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E91-A(4):958–966, 2008.

[8] Author's Name. Hardware architecture design for variable block-size motion estimation in mpeg-4 avc/jvt/itu-t h.264. In *IEEE Int. Symp. on Circuits Syst.*, pages 796–799, 2003.

[9] RGB. H.264 profiles. https://www.rgb.com/h264-profiles, Accessed on May 12, 2023.

[10] Iain E. Richardson. *The H.264 Advanced Video Compression Standard*. Wiley, UK, 2nd edition, 2010.