UC Davis
UC Davis Electronic Theses and Dissertations

Title
H.264 Codec Implementation on a Many-Core Processor Array

Permalink
https://escholarship.org/uc/item/3nz466h2

Author
Callahan, Aidan

Publication Date
2022

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

H.264 Codec Implementation on a Many-Core Processor Array

By

AIDAN THOMAS CALLAHAN

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

____

Bevan M. Baas, Chair

____

Soheil Ghiasi

____

Hussain Al-Asaad

Committee in charge
2022

© Copyright by Aidan Thomas Callahan 2022
All Rights Reserved

Abstract

Due to the rise of higher resolution video over limited transmission bandwidths, video compression algorithms have revolutionized the way we view a digital video today. Video compression has developed into an integral computational block for devices such as computers, televisions, and phones. H.264, also known as Advanced Video Compression (AVC), is a popular standard for the compression of video content. H.264/AVC offers excellent compression performance due to a collection of algorithmic improvements over its predecessors.

The H.264/AVC standard algorithm requires a high level of computational complexity with the opportunity to compute many subtasks in parallel. Consequently, a fine-grained many-core platform is a promising solution for the H.264/AVC algorithm. In this work a baseline H.264/AVC encoder and decoder (codec) is designed and simulated on the KiloCore II chip. KiloCore II contains 697 independently programmable processors, 16 64kB shared SRAM memories, and an efficient 2-D mesh topology for inter-core communication - a promising chip for computationally intensive tasks such as H.264.

In this work, a full H.264 baseline codec is implemented on the KiloCore II platform. The H.264 bitstream syntax is modified for compliance at the macroblock level - the most intricate and computationally taxing elements of H.264/AVC. The proposed codec operates with a 4:2:0 sampled, yuv color space video sequence. All nine intra prediction modes are supported. Additionally, a 2-D logarithmic search algorithm is utilized for integer motion estimation. The full codec is tested and verified through QCIF format video test samples.

Both the encoder and decoder are clocked at 1,780 MHz. The encoder processes 27,239 macroblocks-per-second at 449 mW without any algorithm specific hardware. With the introduction of a motion estimation accelerator, the encoder is able to process 73,010 macroblocks-per-second at 635 mW. The decoder, on the other hand, processes 24,347 macroblocks-per-second at 482 mW. KiloCore II is a competitive platform for video compression achieving a $1.8\times - 49.1\times$ and $1.4\times - 8.1\times$ higher throughput relative to compared encoder and decoder designs, respectively. All in all, KiloCore II outperformed every H.264/AVC baseline codec compared in this work in throughput performance.

Acknowledgments

The research presented in this work would not have been possible without the support and guidance of many people. First and foremost, I would like to thank Professor Bevan Baas for pushing me to take on this project. His patience and meticulous attention to detail challenged me in the best way possible. Through his insight I have become a better problem solver and critical thinker. The skills I learned the past year will last me a lifetime.

I would also like to thank both Professor Soheil Ghiasi and Professor Hussain Al-Asaad for serving as my committee and taking the time to review my work.

Next, I would like to extend my gratitude to members of the VLSI Computational Laboratory (VCL). In particular, I would like to thank Dr. Satyabrata Sarangi, Ziyuan Dong, and Tony Tsoi for providing me with valuable advice during my time as a graduate student.

Finally, I would like to thank my mother (Wendy), father (Stephen), and sister (Erin). As the youngest child in my family, I learned the benefits of hard work, perseverance, and an eagerness to learn through observation. Without this paradigm or my family's support, I would not be where I am today.

Contents

Abstract ii

Acknowledgments iii

List of Figures vii

List of Tables ix

1 Introduction 1
- 1.1 Motivation 1
- 1.2 Thesis Organization 2

2 H.264/AVC Standard Overview 3
- 2.1 Overview 3
- 2.2 Slices and Macroblocks 4
- 2.3 Picture Formats and Pixel Sampling 4
- 2.4 H.264/AVC Bitstream Syntax 7
- 2.5 Video Frame Prediction Methods 8
  - 2.5.1 Luma Intra Prediction (I Frames) 8
  - 2.5.2 Chroma Intra Prediction (I Frames) 11
  - 2.5.3 Luma Inter Prediction (P Frames) 11
  - 2.5.4 Chroma Inter Prediction (P Frames) 13
- 2.6 Motion Estimation 14
- 2.7 Transform, Quantization, and Inverse Transform 14
  - 2.7.1 Transform and Quantization in H.264/AVC 16
  - 2.7.2 Quantization Parameter (QP) 18
  - 2.7.3 Inverse Transform in H.264/AVC 18
- 2.8 Coding Methods 18
  - 2.8.1 Exp-Golomb Coding 19
  - 2.8.2 Context-Adaptive Variable-Length Coding 19

3 KiloCore Architecture 21
- 3.1 Processors 21
- 3.2 Memory 22
- 3.3 Inter-Core Communication 22
- 3.4 Design Flow and Mapping 23
- 3.5 Prior Video Codec Works on the KiloCore Platform 24

4 RTL Implementation **27**
4.1 Design Overview . . . 27
4.2 Encoder Design . . . 28
4.2.1 Input Handling . . . 28
4.2.2 Datapath . . . 30
4.2.3 Intra Prediction . . . 31
4.2.4 Inter Prediction . . . 32
4.2.5 Encoder Overview . . . 36
4.3 Decoder Design . . . 40
4.3.1 Input Handling . . . 40
4.3.2 Datapath . . . 41
4.3.3 Intra Prediction . . . 41
4.3.4 Inter Prediction . . . 43
4.3.5 Decoder Overview . . . 43
4.4 Results and Analysis . . . 46

5 Fine-Grained CPU Array Implementation **47**
5.1 Design Overview . . . 47
5.2 Shared Memories . . . 47
5.3 Encoder . . . 49
5.3.1 Intra Prediction . . . 51
5.3.2 Inter Prediction . . . 52
5.3.3 Motion Estimation Control . . . 53
5.3.4 Motion Estimation Cost Engine . . . 53
5.3.5 Mode Cost Engine . . . 55
5.3.6 Critical Path Optimization . . . 55
5.3.7 Forward Transform . . . 56
5.3.8 CAVLC . . . 57
5.3.9 Reconstruction . . . 57
5.4 Decoder . . . 59
5.4.1 Bitstream Handling . . . 59
5.4.2 Prediction Decoding . . . 59
5.4.3 Inverse CAVLC . . . 61
5.4.4 Inverse Transform . . . 61

6 Verification Methods **63**
6.1 JM Software . . . 63
6.2 Matlab Golden Reference Verification . . . 65
6.2.1 JM Verification . . . 65
6.2.2 Characteristic Verification . . . 68
6.3 RTL and KiloCore Verification . . . 70

7 Results and Comparisons **72**
7.1 Core Utilization . . . 72
7.1.1 Encoder . . . 72
7.1.2 Decoder . . . 73
7.2 Throughput by Stage . . . 76
7.2.1 Encoder . . . 76
7.2.2 Decoder . . . 76

7.3 Total Energy . . . 79
7.3.1 Encoder . . . 79
7.3.2 Decoder . . . 79
7.4 Scaling with Voltage . . . 82
7.5 Mapping . . . 87
7.5.1 Encoder . . . 87
7.5.2 Decoder . . . 87
7.6 Comparisons . . . 90
7.6.1 KiloCore II Performance . . . 90
7.6.2 KiloCore II Performance with Motion Estimation Accelerator . . . 92
7.6.3 Comparisons . . . 93

8 Summary and Future Work **98**
8.1 Summary . . . 98
8.2 Future Work . . . 98

Bibliography **101**

List of Figures

2.1 Scope of the H.264 Standard (highlighted in blue) 4
2.2 Raster Scan Order 5
2.3 Interlaced Fields (Left) and Progressive Frame (Right) Sampling [1] 5
2.4 H.264 Sampling Formats [2] 6
2.5 H.264 Syntax Format 7
2.6 Example of macroblock types and prediction sources [2] 9
2.7 4x4 intra prediction modes [2] 10
2.8 Chroma Intra prediction modes [2] 11
2.9 Macroblock and sub-macroblock partitions 12
2.10 Macroblock and sub-macroblock partition coding scheme 13
2.11 2-D Logarithmic Search including iteration 1 (red), iteration 2 (yellow), and iteration 3 (green) 15
2.12 CAVLC 4x4 pixel scan order [2] 20

3.1 KiloCore pipeline [3] 22
3.2 (a) 2-D mesh interprocessor communication and (b) the generalized communication routing architecture [4] 23
3.3 KiloCore simulation scheme including processor coding, the task script (circuit link configuration), and the test script (cycle-accurate simulation results) 24

4.1 Test bench (blue) and core Verilog (gray) interface 28
4.2 Encoder top finite state machine 30
4.3 Encoder Intra prediction block diagram 32
4.4 Input (green) and memory (blue) required for intra (a) and inter (b) prediction . . . 33
4.5 Input block memory 34
4.6 Reconstructed block memory 34
4.7 Core motion estimation block 35
4.8 Top motion estimation block 36
4.9 Encoder prediction and residual generation datapath 37
4.10 Encoder forward transform and quantization datapath 38
4.11 Encoder reconstruction and CAVLC datapath 39
4.12 Decoder state machine 41
4.13 Decoder Intra prediction 42
4.14 Decoder Inter prediction 43
4.15 Decoder prediction generation and inverse CAVLC 44
4.16 Decoder inverse transform and reconstruction 45

5.1 Number of coefficients memory 48

5.2 Memory architecture design for reconstructed chroma elements 49
5.3 Inter prediction memory switch 50
5.4 KiloCore II luma intra prediction decision 51
5.5 KiloCore II chroma intra prediction decision 52
5.6 KiloCore II Inter prediction top-level block diagram 54
5.7 KiloCore II cost engine 55
5.8 KiloCore II mode cost engine 56
5.9 Motion estimation critical path optimization 56
5.10 KiloCore II forward transform 57
5.11 KiloCore II CAVLC 58
5.12 KiloCore II reconstruction 58
5.13 KiloCore II bitstream handler 60
5.14 KiloCore II prediction decoding 60
5.15 KiloCore II inverse CAVLC 61
5.16 KiloCore II inverse transform 62

6.1 JM trace function output 64
6.2 Transform golden reference verification (gray) using verified parameters (green) . . . 67
6.3 Sample JM CAVLC 4x4 pixel encoding 67
6.4 H.264 golden reference compression ratio 69
6.5 H.264 golden reference PSNR 69
6.6 Golden reference frame comparison for frame 1 of “suzie.qcif” (QP of 15) with original frame (left) and reconstructed frame (right) 70

7.1 KiloCore II encoder core distribution by function 74
7.2 KiloCore II decoder core distribution by function 75
7.3 Encoder throughput, in macroblocks-per-second, by stage 77
7.4 Decoder throughput, in macroblocks-per-second, by stage 78
7.5 Encoder energy distribution by function 80
7.6 Decoder energy distribution by function 81
7.7 Encoder throughput with respect to voltage (intra) 83
7.8 Encoder energy with respect to voltage (intra) 83
7.9 Encoder power with respect to voltage (intra) 83
7.10 Encoder throughput with respect to voltage (inter) 84
7.11 Encoder energy with respect to voltage (inter) 84
7.12 Encoder power with respect to voltage (inter) 84
7.13 Decoder throughput with respect to voltage (intra) 85
7.14 Decoder energy with respect to voltage (intra) 85
7.15 Decoder power with respect to voltage (intra) 85
7.16 Decoder throughput with respect to voltage (inter) 86
7.17 Decoder energy with respect to voltage (inter) 86
7.18 Decoder power with respect to voltage (inter) 86
7.19 KiloCore II encoder place and routed map of cores 88
7.20 KiloCore II decoder place and routed map of cores 89

8.1 Parallel architecture for motion estimation 99
8.2 Memory optimization scheme for local memory (green) and SRAM (orange) 100

List of Tables

2.1 Macroblock level bit syntax proposed in this work . . . 8

3.1 Encoder comparisons between Le [5] and this work . . . 25

4.1 Verilog encoder inputs and outputs . . . 29
4.2 Verilog decoder inputs and outputs . . . 40
4.3 The proposed RTL encoder and decoder performance . . . 46

5.1 KiloCore II shared memory utilization . . . 48
5.2 KiloCore II encoder inputs and outputs . . . 50
5.3 KiloCore COST_ENGINE iterations . . . 53

6.1 JM reference software verified algorithmic block . . . 65
6.2 JM reference total coefficients and T1s . . . 68
6.3 JM reference total zeros and run of zeros . . . 68
6.4 JM reference reconstructed 4x4 pixel block . . . 68
6.5 Encoder RTL and KiloCore test dataset . . . 71
6.6 Decoder RTL and KiloCore test dataset (PSNR and compression performance labeled in figure 6.5) . . . 71

7.1 KiloCore encoder core usage . . . 74
7.2 KiloCore decoder core usage . . . 75
7.3 Encoder throughput, in macroblocks-per-second, by stage . . . 77
7.4 Decoder throughput, in macroblocks-per-second, by stage . . . 78
7.5 KiloCore encoder power distribution by task . . . 80
7.6 KiloCore decoder power distribution by task . . . 81
7.7 KiloCore decoder power distribution by task . . . 82
7.8 KiloCore II encoder performance . . . 91
7.9 KiloCore II decoder performance . . . 91
7.10 KiloCore II final weighted performance metrics . . . 91
7.11 AsAP 2 motion estimation accelerator throughput and power performance for a single QCIF frame . . . 92
7.12 KiloCore II encoder performance with motion estimation accelerator . . . 92
7.13 Comparison of H.264 programmable encoders . . . 94
7.14 Comparison of H.264 special purpose encoders . . . 95
7.15 Comparison of H.264 programmable decoders . . . 96
7.16 Comparison of H.264 special purpose decoders . . . 97

Chapter 1

Introduction

1.1 Motivation

Today most video is streamed from some device, whether it's a computer or a cell phone. Video compression algorithms are critical for both the transmission and storage of high-quality videos in such devices. Modern network bandwidths provide a hard limit to the transmission rates of raw video data and memory costs make storing uncompressed video frames unfeasible. Consequently, video compression has emerged as a lucrative and essential industry today.

While many innovative compression techniques have been proposed and researched, commercial video compression requires standardization to interface across all platforms. Additionally, the necessity to abstract tradeoffs between complexity and performance for different computational needs, such as a cellular device and a high-definition television, imply the need for compression standards. H.264, the compression standard discussed in this work, remains the most used video encoder/decoder (codec) utilized by 83% of industry developers as of 2022 [6]. The success of the AVC compression standard is attributed to a collection of many algorithmic and architectural improvements over its predecessors.

Given the wide acceptance of the H.264 standard, many attempts to optimize throughput and area efficiency are made. To offer an effective H.264/AVC solution, a programmable fine-grained processor design is explored in this work. The KiloCore II platform, the computational unit utilized in this work, contains 697 independently programmable processors. Due to the large opportunity of task and data level parallelism within H.264/AVC, KiloCore II is a promising platform for this

compression standard. The goal of this project is to design a fine-grained codec and show that it is competitive with related works.

1.2 Thesis Organization

The remainder of this work is divided as follows.

Chapter 2 provides an overview of the H.264/AVC standard. Aspects of the standard specific to this work are highlighted. Additionally, any design choice affected by the proposed hardware design is noted.

Chapter 3 provides an explanation of the KiloCore platform. Specific details related to the processor array's independent processors, shared memories, communication layer, and simulation platform are explored.

Chapter 4 provides a hardware overview of the register transfer level (RTL) implementation for both the H.264/AVC encoder and decoder. This model is fully functional and synthesizable. Each computational block and high-level control algorithms are discussed.

Chapter 5 provides a hardware overview of the fine-grained many-core processor array implementation (KiloCore II) of the H.264 codec. Any variance between the KiloCore II implementation and RTL implementation are highlighted.

Chapter 6 discusses how the proposed golden reference software model is designed and verified. Additionally, the RTL and KiloCore II model verification methods are also explained.

Chapter 7 provides results and performance comparisons between the KiloCore II implementation and related works. Throughput, energy consumption, and area are used as benchmarks for comparison.

Chapter 8 outlines a summary of this work and provides details regarding future improvements.

Chapter 2

H.264/AVC Standard Overview

2.1 Overview

H.264/AVC is an industry standard that defines a format for compressed video data. Known as the Joint Movie Team (JVT), both the ITU Video Coding Experts Group (VCEG) and ISO/IEC JTC1 Moving Picture Experts Group (MPEG) designed, and currently manage, the standard. The standard allows flexibility in terms of the choice of compression tools used for an application. Consequently, H.264/AVC is suitable for a wide range of use cases, such as low-latency mobile video streaming and high definition consumer TV. Relative to the performance of prior standards, AVC provides up to a 50% bit rate savings when these techniques are used together effectively [7].

The H.264 standard document contains a comprehensive collection of semantic information, essential instructions, and optional design guidance [8]. Most importantly, the standard document describes: picture formats and scanning processes (Chapter 6); specific bitstream syntax elements and their allowed values (Chapter 7); details regarding the methods of decoding a video frame (Chapter 8); and the parsing process used to extract syntax elements from a valid bitstream (Chapter 9). This dissertation primarily focuses on the stages required to decode a video frame as outlined in Chapter 8 of the H.264/AVC standard.

Another pertinent characteristic of the H.264/AVC standard is that it only specifies the process for decoding a bitstream. An H.264 compliant encoder is assumed to mirror the basic semantics and format of a H.264 decoder, as illustrated in figure 2.1. With this in mind, the standard

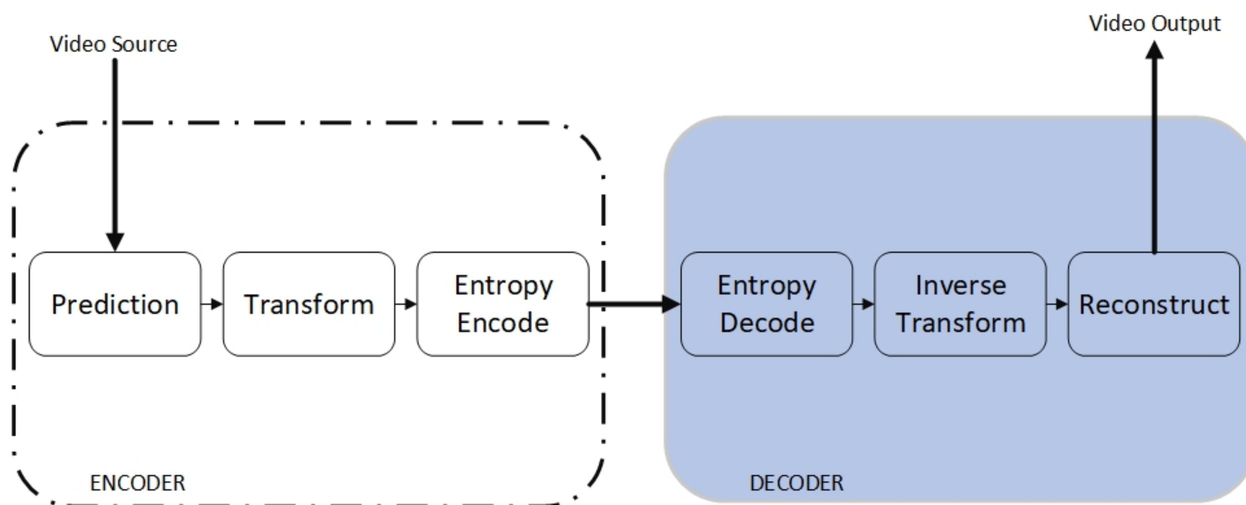

Figure 2.1: Scope of the H.264 Standard (highlighted in blue)

intentionally presents the opportunity for the designer to specify the complexity of approach for the encoder, particularly for the predictive algorithm. An overview of the H.264/AVC standard and the design choices used for the codec in this dissertation are outlined in the following sections.

2.2 Slices and Macroblocks

The fundamental building block of a frame in H.264 is a macroblock (MB), defined as a 16x16 pixel region within the frame. Every video frame consists of an integral number of MBs. For example, a QCIF frame (176 x 144 pixels) has a total of 99 macroblocks in one frame. Macroblocks are processed in raster scan order, which is illustrated below in figure 2.2.

In H.264, frames may be further classified into slice groups. Every video frame consists of one or more individual slices, each with a slice header and integral number of macroblocks. Each slice is decoded independently in a frame, which allows the code to prevent error propagation across the frame. In this work, frames are coded and decoded as one slice group.

2.3 Picture Formats and Pixel Sampling

Chapter 6 of the H.264/AVC standard defines video picture formats. According to the standard, a video sequence is a collection of frames or fields collectively referred to as a picture [8]. A video signal sampled as a field only transmits either the even or odd lines of an image at a time,

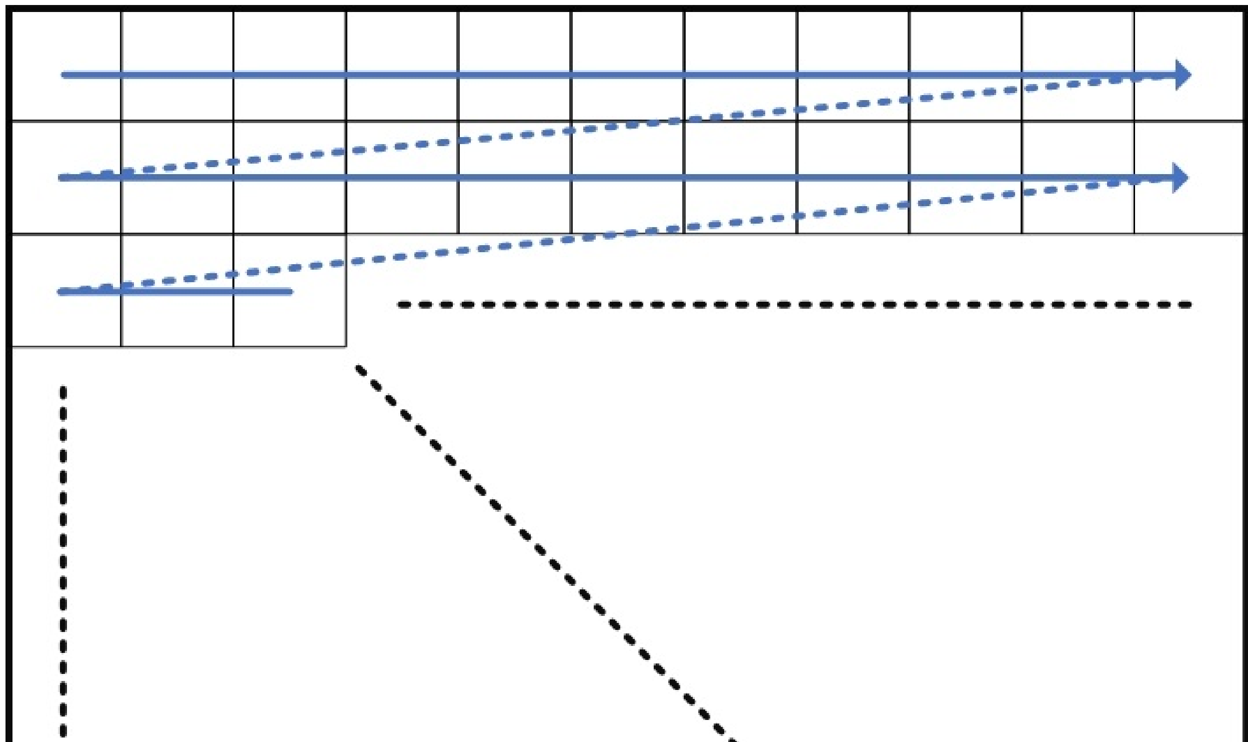

Figure 2.2: Raster Scan Order

known as interlaced scans. In contrast, a signal sampled as a frame transmits the entire image of a video picture, known as progressive scans. Since interlaced scans transmit half the data as progressive scans, twice as many fields per second are transmitted which gives the perception of smoother motion. While the AVC standard supports both interlaced and progressive scanning, only progressive scans are implemented in the scope of this dissertation.

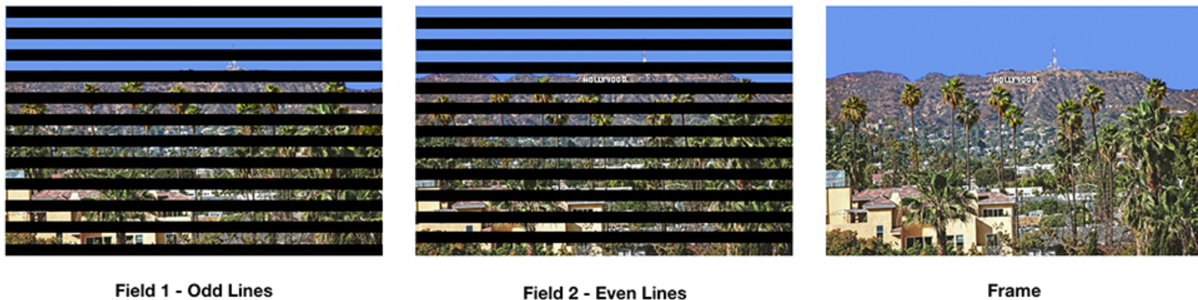

Figure 2.3: Interlaced Fields (Left) and Progressive Frame (Right) Sampling [1]

Chapter 6 of the H.264/AVC standard also covers valid pixel formats supported in the standard. H.264 supports any monochrome or tri-stimulus color sampling. Modern video compression

algorithms commonly convert video frames into one luma and two chroma samples, referred to as the YUV color space. Since the human visual system is less sensitive to color than luminance, the luma component of the YUV color space is often represented with higher resolution than the chroma samples [2]. The H.264/AVC standard supports three methods for sampling chroma elements: 4:2:0, 4:2:2, 4:4:4 sampling. 4:4:4 sampling preserves the same number of chroma components with respect to luma components. In 4:2:2 sampling, the chroma components have the same vertical resolution as luma components but half the horizontal resolution. The sampling scheme followed in this work, 4:2:0 sampling, transmits half the vertical and horizontal chroma resolution with respect to the luma resolution. An illustration of the three sampling schemes are illustrated in figure 2.4.

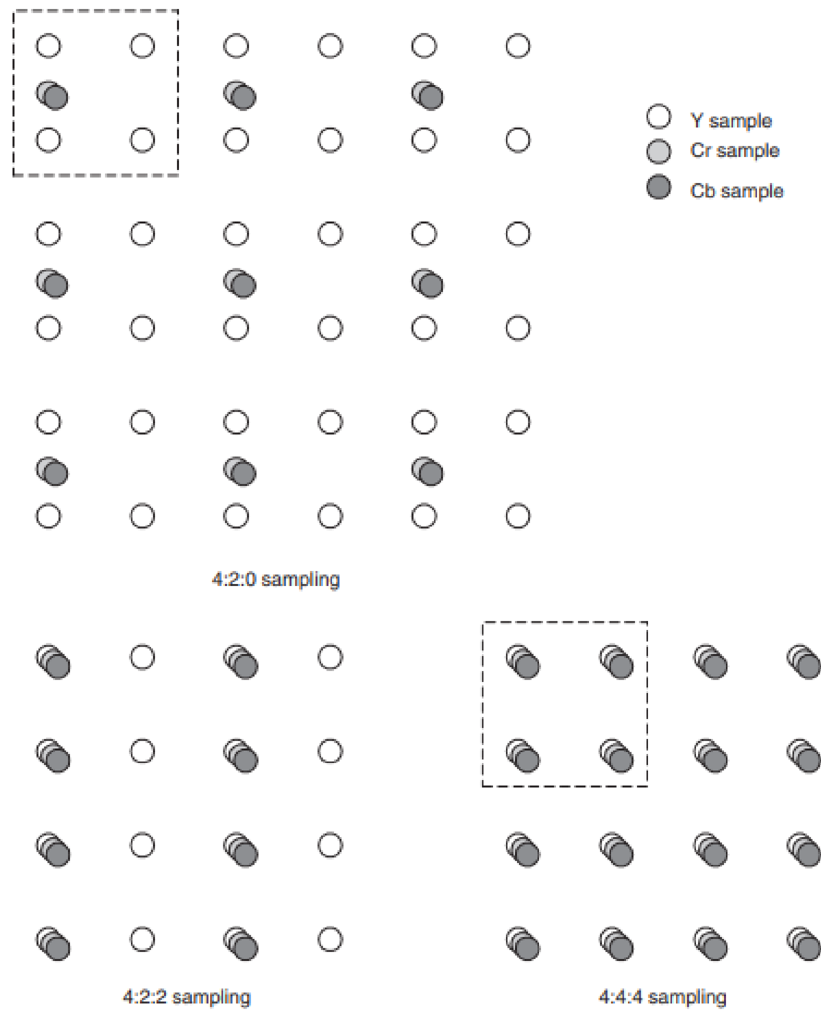

Figure 2.4: H.264 Sampling Formats [2]

2.4 H.264/AVC Bitstream Syntax

Chapter 7 of the H.264/AVC standard outlines the video sequence syntax. The video syntax serves as a hierarchical guideline for the order and form of transmitted data. Such syntax specifications are imperative for encoders and decoders to effectively communicate across platforms. The general syntax hierarchy for H.264 is illustrated in figure 2.5.

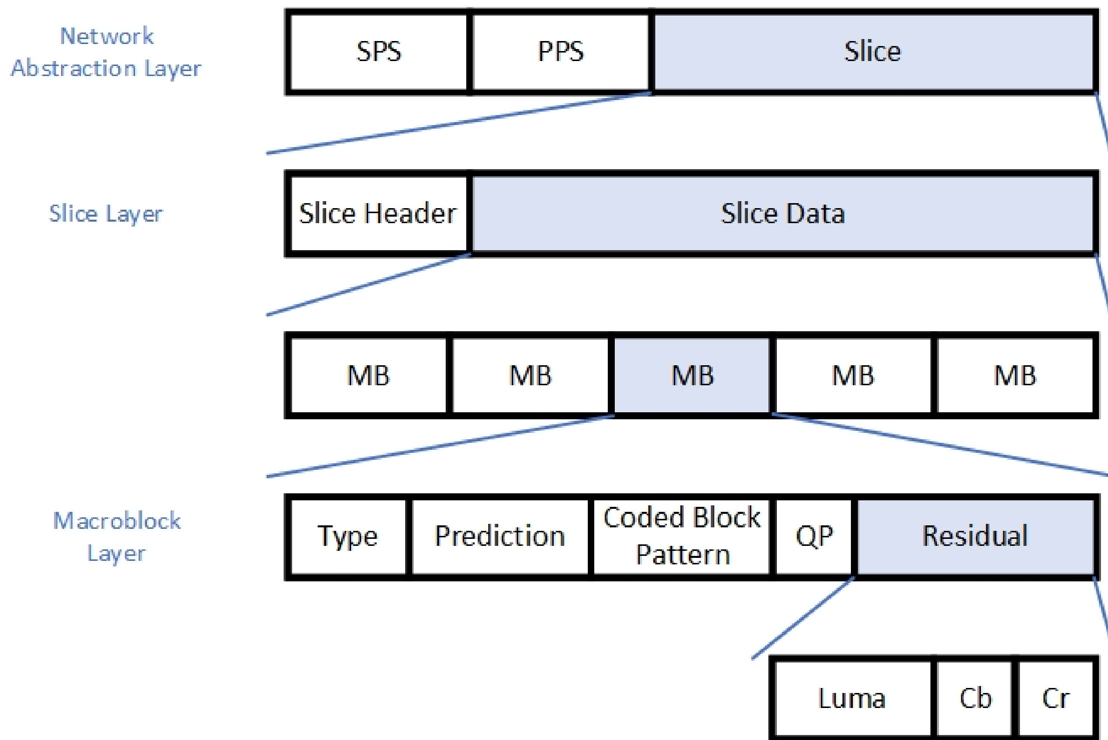

Figure 2.5: H.264 Syntax Format

The Natural Abstract Layer (NAL), the highest syntax level in AVC, consists of Natural Abstraction Layer Units (NALUs) such as Sequence Parameter Sets (SPS), Picture Parameter Sets (PPS), control information, or coded video data. SPS contain global parameters that are common to an entire video, video frame size for example. PPS contain local parameters that apply to a subset of coded frames.

Below the NAL the H.264 standard specifies the slice layer. This syntax layer consists of a slice header followed by the entire slice bitstream. Within the slice header, specific slice information is established, such as the slice prediction mode, frame number, and various control flags. The slice data partition contains all the coded macroblocks within the frame.

The lowest level syntax layer is the macroblock layer. This syntax layer contains all the information necessary to decode a single MB, including the macroblock type, prediction parameters, coded block pattern, quantization parameter, and encoded residual.

Since the scope of this work is to design the fundamental building blocks of an H.264/AVC codec in hardware, a full industry compliant bit stream syntax is not implemented. Instead this work implements a macroblock level compliant bitstream, which is fully outlined sequentially in table 2.1. A top level video sequence header is provided at the beginning of the bitstream to provide the proposed algorithm with frame attributes, such as frame height and width.

Bitstream Information	**Description**
Frame Mode	"1" for intra prediction frame; "0" for inter prediction frame
Luma prediction header	Intra prediction mode or motion vectors
Chroma prediction header	Intra prediction mode or motion vectors
Coded block pattern (CBP)	Indication of whether certain areas of the macroblock contain a residual of zero
QP	Frame quantization parameter
Luma coding	CAVLC coded luma residuals
Chroma coding	CAVLC coded chroma residuals

Table 2.1: Macroblock level bit syntax proposed in this work

2.5 Video Frame Prediction Methods

In the H.264/AVC standard, macroblock prediction is integral to compression performance. At the fundamental level, video prediction is the ability to guess the data values within a MB using previously decoded data. In the realm of video compression we have two prediction tools at our disposal – temporal and spatial data. Spatial predictions use previously coded pixels from the current frame while temporal predictions use coded data from a past or future frame. In H.264, spatially predicted frames and temporally predicted frames are respectively called intra and inter predicted frames. In figure 2.6, data sample origins for intra (I MB) and inter (P and B MBs) predictions are illustrated. This work implements both I and P frame prediction capabilities.

2.5.1 Luma Intra Prediction (I Frames)

In the H.264/AVC standard, either 4x4 pixel or 16x16 pixel blocks are used for spatial prediction (intra prediction). In this dissertation, macroblocks are only predicted in 4x4 sets of

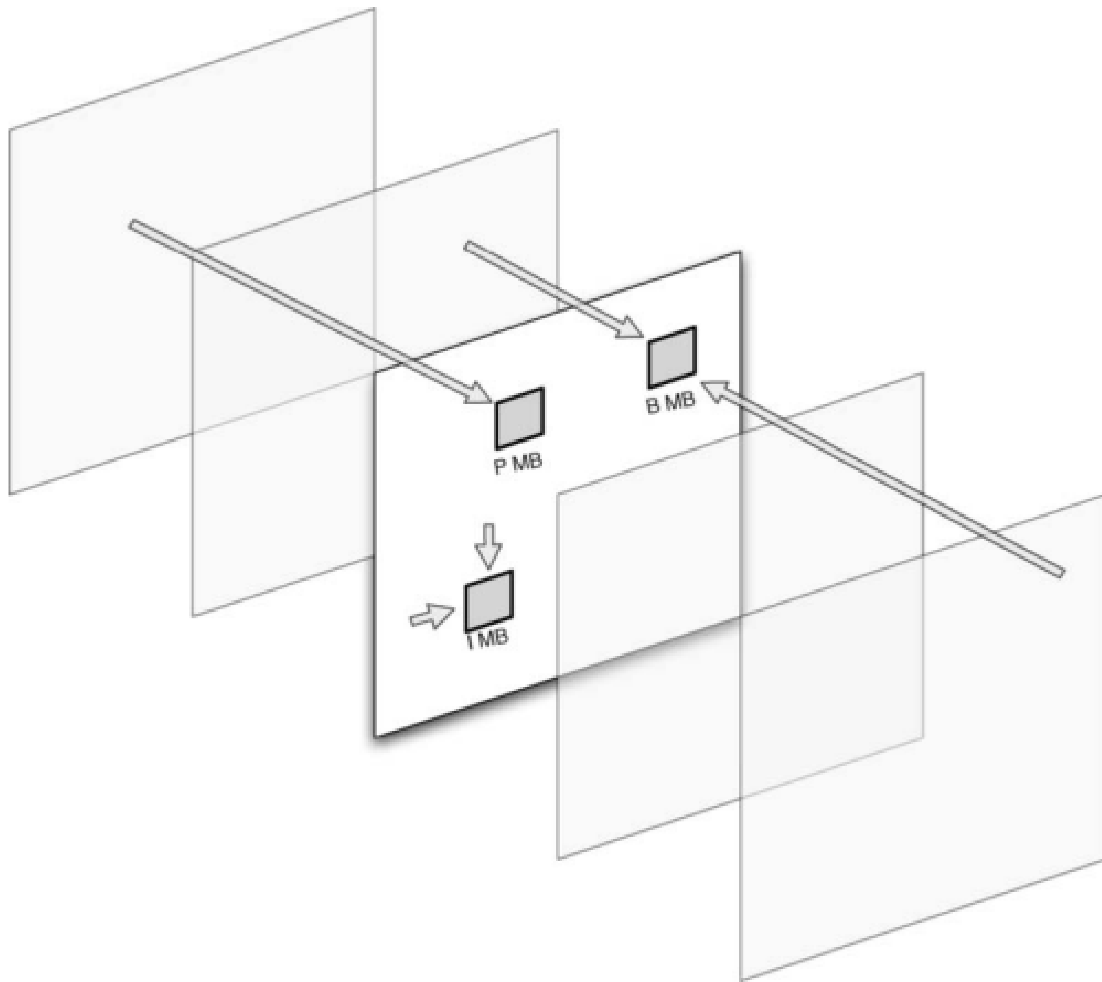

Figure 2.6: Example of macroblock types and prediction sources [2]

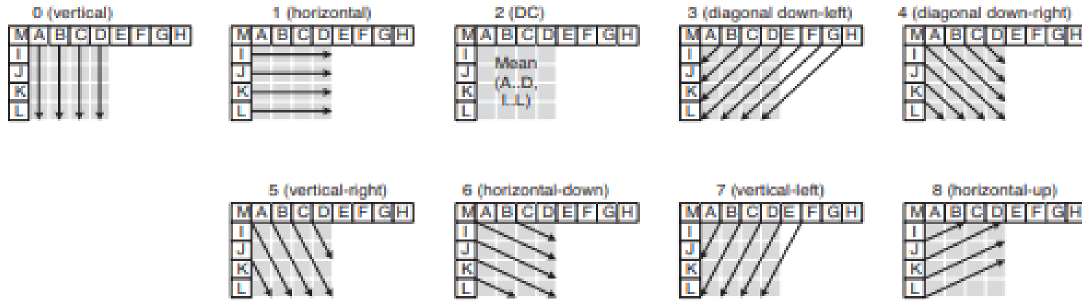

Figure 2.7: 4x4 intra prediction modes [2]

pixels for intra prediction. The AVC standard supports a total of nine prediction methods that utilize previously coded, neighboring pixel samples. In the event that a pixel block does not have access to neighboring pixels, DC prediction, a special method of intra prediction that does not need a full set of previously decoded spatial samples, is utilized. Figure 2.7 illustrates all nine intra prediction modes available in H.264.

On the encoder side, the designer must specify a decision algorithm used to choose which intra prediction mode is used from transmitting the residual. As previously mentioned, the H.264/AVC standard does not define the structure of the encoder. In this work, all intra prediction modes are executed (when neighboring pixel samples are available) in parallel and the mode with the smallest sum of absolute difference (SAE) is used.

An H.264/AVC encoder must transmit an intra prediction mode header within the bitstream so that the decoder knows how to create a prediction for macroblock reconstruction. The transmitted intra prediction mode is context-adaptive, meaning that the coded form depends on characteristics of previously decoded samples. Using the prediction modes of the upper and left previously decoded 4 x 4 pixel blocks, the lowest prediction mode number is established as the most probable prediction mode. Using this model, the prediction mode is decoded from the bitstream as illustrated in algorithm 1. When a prediction mode is correctly predicted, only one bit is added to the bitstream; whereas, four bits are required to transmit an unpredicted intra prediction mode.

Algorithm 1 Decoding Intra Prediction Mode

$x = Bitstream(bp)$
$bp = bp + 1$
$n = min(Intra4x4PredModeTop, Intra4x4PredModeLeft)$
if $x == 1$ **then**
  $PredOut = n;$
else
  $y = Bitstream(bp : bp + 2)$
  $bp = bp + 3$
  if $y < n$ **then**
    $PredOut = y$
  else
    $PredOut = y + 1$
  end if
end if

2.5.2 Chroma Intra Prediction (I Frames)

The chroma samples are encoded in a similar fashion to the luma samples with a few slight nuances. Chroma samples are processed 8x8 pixel blocks at a time - corresponding to a full luma macroblock when chroma elements are 4:2:0 sub-sampled. Consequently, chroma elements have a total of four special 8x8 pixel block prediction modes illustrated in figure 2.8.

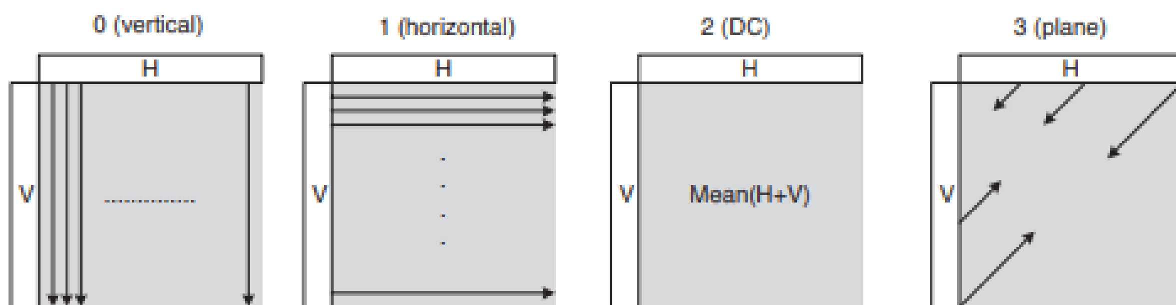

Figure 2.8: Chroma Intra prediction modes [2]

As with the luma intra prediction modes, all chroma intra prediction modes are executed, and the mode with the smallest SAE is transmitted. The selected mode is encoded using an unsigned exp-golomb coding scheme which is discussed in section 2.8.1.

2.5.3 Luma Inter Prediction (P Frames)

The second prediction mode utilized within the scope of this project is inter prediction

based on the previously decoded frame. At sufficiently high frame rates, the difference between two successive frames is small. Consequently, inter prediction searches within a region of the previously decoded frame and finds the smallest error to bit compression ratio.

The H.264/AVC standard offers flexibility to split macroblocks into partitions, each with their own residuals and motion vectors. Increased partitions decrease the reconstructed MBs error at the expense of worse compression compared to MBs with fewer partitions. The four partitions (full, vertical, horizontal, and quad) are first executed from a 16 x 16 macroblock. If a quad partition is the selected mode of transmission, the complete inter prediction process is repeated on each 8 x 8 pixel block. All four macroblock and sub-macroblock partitions are illustrated in figure 2.9.

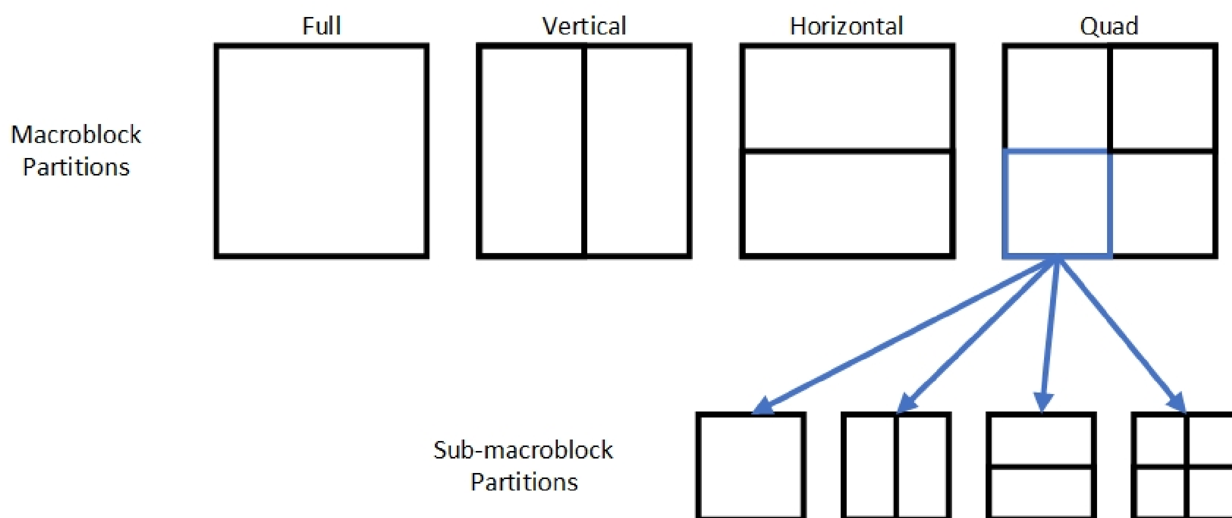

Figure 2.9: Macroblock and sub-macroblock partitions

Along with the macroblock residual, the selected inter prediction mode and corresponding motion vectors are transmitted in the output bitstream. Motion vectors are simply an offset pair (delta X and delta Y) corresponding to each partition's predicted location within the reconstructed frame with respect to the location in the current frame. In this work, the mode and all motion vectors for a macroblock are encoded using a signed exp-golomb encoding [9]. The mode values and motion vectors are transmitted according to the syntax illustrated in figure 2.10.

Like the intra prediction process, a decision algorithm for the selected partition must be established for the encoder. In this thesis, rate-distortion theory is utilized to balance the trade-off between video distortion and bit compression performance [10]. The general form of the Langrangian cost function utilized in H.264/AVC is presented below. Here, $SSD(s, c, MODE)$, $R(s, c, MODE)$,

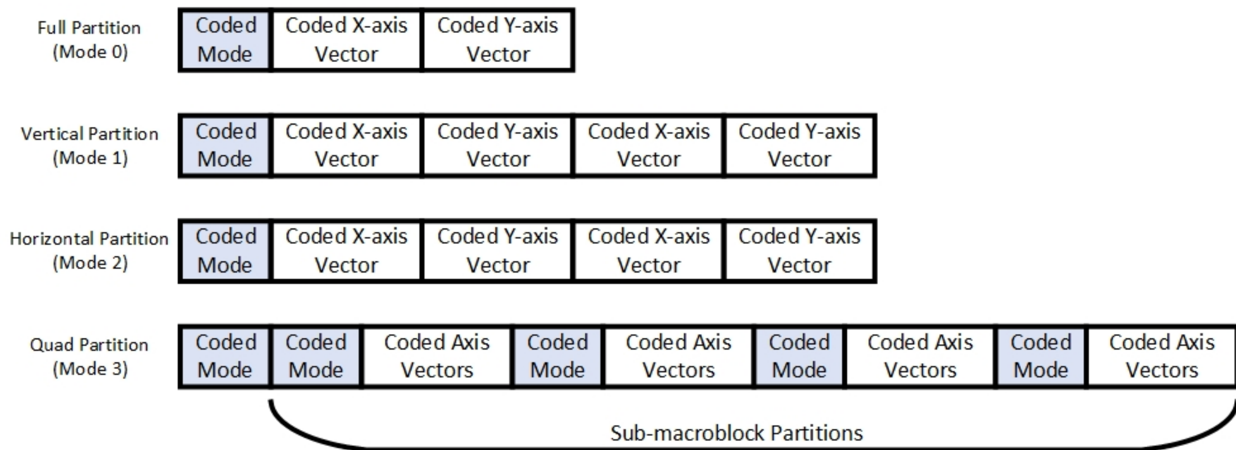

Figure 2.10: Macroblock and sub-macroblock partition coding scheme

and λ represent the video distortion (sum of squared difference), bit compression rate (output bit length), and Lagrange multiplier respectively. In the proposed H.264/AVC encoder, two Lagrangian cost functions are utilized, one for the mode partition selection ($Cost_{mode}$) and the other for motion estimation selections ($Cost_{me}$). In this dissertation, the motion estimation Langrangian uses the sum of absolute difference with a square root scaled Langrange multiplier for the distortion and multiplier terms respectively. This design choice reduces the number of multipliers needed for the computationally heavy motion estimation block.

The Lagrange multiplier equations for both mode and motion estimation predictions are presented below. The multiplier is a function of the quantization parameter (QP) [11], which is discussed in more detail in section 2.7.2.

$$Cost_{mode} = SSD(s, c, MODE) + \lambda_{mode} * R(s, c, MODE)$$
$$\text{where, } \lambda_{mode} = 0.85 * 2^{(QP-12)/3}$$
$$Cost_{me} = SAD(s, c, ME) + \lambda_{me} * R(s, c, ME)$$
$$\text{where, } \lambda_{me} = \sqrt{\lambda_{mode}}$$

2.5.4 Chroma Inter Prediction (P Frames)

In order to keep the encoder design lightweight, a motion estimation scheme is not explored for chroma elements in this work. Consequently, the motion vector for all chroma pixel blocks is simply (0,0). This ensures that the chroma prediction is always the reconstructed pixel block from the previous frame at the same index of the current pixel block.

2.6 Motion Estimation

Motion estimation is a critical block in the H.264/AVC standard for utilizing temporal redundancy to improve compression performance. Motion estimation defines the process for searching a region of a previous frame to find a similar M x N region. Motion compensation, on the other hand, refers to the process of subtracting the candidate region with the current block to form a residual. Due to its computational complexity, many researchers have devoted effort to efficient search algorithms. In this dissertation, a three-step 2-D logarithmic search is used as the encoder's motion estimation search algorithm [12].

In this work, three iterations of the logarithmic search are performed, unless any iteration selects the center location as the smallest error. In each iteration 5 regions are compared – center, top, right, bottom, and left. The minimum motion estimation rate-distortion is selected as the center position for the next iteration of the logarithmic search. The pixel difference between the bordering search regions and the center pixel are dictated by the current iteration. The search regions are separated by four, two, and one pixels for the first, second, and third iterations respectively. Figure 2.11 illustrates a possible scenario for this search algorithm.

The H.264/AVC standard supports sub-pixel interpolation in order to increase motion vector precision. Adjacent luma and chroma samples are interpolated up to quarter-pel resolution using a combination of a Finite Impulse Response filter (FIR) and linear interpolation. In this thesis, sub-pixel interpolation is not supported in order to keep all motion vectors in integer format for simplicity.

2.7 Transform, Quantization, and Inverse Transform

The improved compression and error performance for H.264/AVC compared to prior industry standards is attributed to improvements in several blocks of the codec. Innovations in the transform, quantization, and inverse transform are examples of the development of more efficient algorithmic blocks in H.264/AVC. The main function of the transform stage is to convert the residual from the prediction stage into a different mathematical domain that is compressed more efficiently. Since H.264/AVC is lossy, the quantization of video samples is grouped together in the forward transform stage.

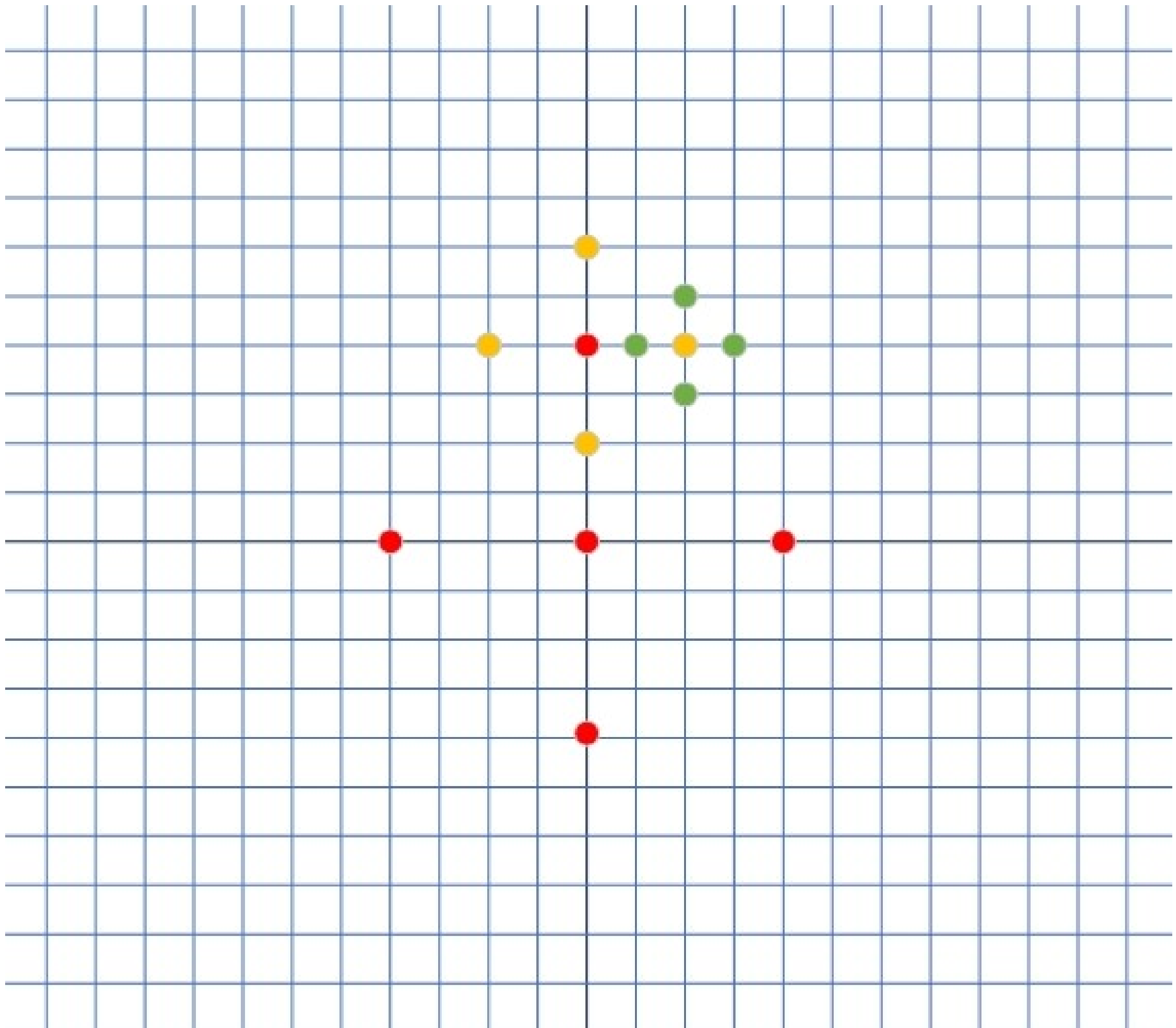

Figure 2.11: 2-D Logarithmic Search including iteration 1 (red), iteration 2 (yellow), and iteration 3 (green)

The three important criteria for the choice of transform are: most of the energy of the transform should be concentrated in a small number of values, the transform should be reversible, and the transform should be computationally inexpensive [2]. Predecessor standards to H.264, such as JPEG, MPEG-2 Video, and MPEG-4 Visual, utilized a two-dimensional Discrete Cosine Transform (DCT) presented in the equation below. A fundamental problem with the DCT is that it requires irrational multiplication which produces mismatch between the forward and inverse transform due to precision errors in hardware. Additionally, the need to operate with floating point numbers introduces unwanted hardware and performance degradation.

$$X_{ij} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} C_x C_y Y_{xy} \cos \frac{(2j+1)y\pi}{2N} \cos \frac{(2i+1)x\pi}{2N}$$

2.7.1 Transform and Quantization in H.264/AVC

The H.264/AVC standard specifies a 4x4 pixel Integer Discrete Cosine Transform (IDCT) that requires only 16-bit arithmetic (no multiplication) with scaling factors that embed the quantization process within forward transform. This change significantly reduces transform and inverse transform complexity with a peak signal-to-noise ratio (PSNR) degredation of less than 0.02 dB [13].

The general construction of the Integer Discrete Cosine Transform for forward transform in H.264/AVC is described below systematically.

1. The traditional DCT (A) is multiplied by 2.5 and rounded to the nearest integer. The new scaled matrix is referred to as the core matrix (C_{f4}). It is relevant to note that multiplication by the core matrix only requires binary shifting and negation within hardware.

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

where $a = \frac{1}{2}, b = \sqrt{\frac{1}{2} \cos \frac{\pi}{8}}, c = \sqrt{\frac{1}{2} \cos \frac{3\pi}{8}}$

$$C_{f4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

2. A factoring matrix (R_{f4}) must be multiplied (Hadamard product) to the core matrix in order to make the transform matrix orthonormal.

$$A_1 = C_{f4} \bullet R_{f4}$$
$$\text{where } R_{f4} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{\sqrt{10}} & \frac{1}{\sqrt{10}} & \frac{1}{\sqrt{10}} & \frac{1}{\sqrt{10}} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{\sqrt{10}} & \frac{1}{\sqrt{10}} & \frac{1}{\sqrt{10}} & \frac{1}{\sqrt{10}} \end{bmatrix}$$

3. The Integer Discrete Cosine Transform formula without quantization is presented below.

$$Y = A_1 \cdot X \cdot A_1^T = [C_{f4} \bullet R_{f4}] \cdot X \cdot [C_{f4}^T \bullet R_{f4}^T]$$
$$Y = [C_{f4} \cdot X \cdot C_{f4}^T] \bullet [R_{f4} \bullet R_{f4}^T]$$
$$S_{f4} = R_{f4} \bullet R_{f4}^T = \begin{bmatrix} \frac{1}{4} & \frac{1}{2\sqrt{10}} & \frac{1}{4} & \frac{1}{2\sqrt{10}} \\ \frac{1}{2\sqrt{10}} & \frac{1}{10} & \frac{1}{2\sqrt{10}} & \frac{1}{10} \\ \frac{1}{4} & \frac{1}{2\sqrt{10}} & \frac{1}{4} & \frac{1}{2\sqrt{10}} \\ \frac{1}{2\sqrt{10}} & \frac{1}{10} & \frac{1}{2\sqrt{10}} & \frac{1}{10} \end{bmatrix}$$

4. Lastly, the scale factor (S_{f4}) is combined with the quantization factor (M_{f4}). The formula for M_{f4} is presented below, but the complete derivation is not provided for simplicity. In total there are six defined M_{f4} matrices, chosen based on the value of the quantization parameter (QP) modulus 6. A binary shift is included in the final forward transform to compensate for the use of only six quantization matrices.

$$M_{f4} \approx \frac{S_{f4} \cdot 2^{15}}{Q_{step}}$$

The complete forward 4x4 Integer Discrete Cosine Transform supported in H.264/AVC is presented in the equation below. The 2^{15} term cancels out the corresponding constant factor in the Mf4 matrix. This value is introduced in order to provide higher precision while maintaining fixed point arithmetic.

$$Y = round([C_{f4}] \cdot [X] \cdot [C_{f4}^T] \bullet [M_{f4}(QP\%6, n)] \cdot \frac{1}{2^{15+floor(\frac{QP}{6})}})$$

2.7.2 Quantization Parameter (QP)

In H.264 the tradeoff between video quality loss and bit compression performance is controlled by a quantization parameter (QP). QP ranges between 0 and 51, where a value of 51 indicates the highest value of quantization. The value of QP directly controls a scaling matrix for the forward and inverse transform blocks. Additionally, the quantization parameter is typically used in the motion estimation cost function as a weight to establish rate-distortion ratios.

In H.264/AVC the quantization parameter is passed as the header component of the macroblock syntax layer. This implies that QP can be chosen dynamically for each MB in the encoder to improve compression performance. In this work, the quantization parameter is always constant for a given video sequence. This eliminates the need for any quantization parameter estimation hardware within the encoder.

2.7.3 Inverse Transform in H.264/AVC

The inverse Integer Discrete Cosine Transform is derived in a similar method to the forward transform. Since the derivation closely mirrors the forward transform derivation, the inverse transform derivation is not presented. The complete inverse IDCT is presented in the equation below.

$$Z = round([C_{i4}^T] \cdot [Y \bullet v(QP\%6, n) \cdot 2^{floor(\frac{QP}{6})}] \cdot [C_{i4}] \cdot \frac{1}{2^6})$$

2.8 Coding Methods

In this dissertation, two coding algorithms are utilized in the proposed codec. Context-adaptive variable-length coding (CAVLC) and inverse CAVLC are implemented at both the end of the encoder and front of the decoder, respectively. CAVLC is a form of entropy coding that is lossless. It is used to encode the residual output of the transform block on the encoder side. The second coding algorithm utilized in this dissertation is signed exponential-Golomb (Exp-Golomb) coding. This coding method is utilized to encode and decode the motion vectors derived from the inter prediction process outlined in Section 2.5.3.

2.8.1 Exp-Golomb Coding

Signed Exp-Golomb coding is a variable length code with a regular construction. The value to be coded is always converted into an unsigned integer referred to as the $CodeNum$. The prefix of an Exp-Golomb code is a string of zeros equal to the bit width of the encoded CodeNum. The suffix is further encoded as illustrated in algorithm 2. Both the prefix and suffix are separated by a single bit. The following algorithm describes the encoding process for Exp-Golomb code. A similar process is mirrored to decode an Exp-Golomb code word.

Algorithm 2 Exp-Golomb Encoding

if $value > 0$ **then**
  $CodeNum = 2 * value - 1$
else
  $CodeNum = (-2) * value$
end if
$M = floor(\log_2 *(CodeNum + 1))$
$Suffix = CodeNum + 1 - 2^M$
$Code = [Mzeros][1][Suffix]$

2.8.2 Context-Adaptive Variable-Length Coding

CAVLC, the entropy encoder block utilized in this work, encodes and decodes residual data from the transform block. CAVLC is designed to capitalize on several characteristics of quantized coefficient blocks. The CAVLC bitstream is derived from a reordered zig-zag scan (figure 2.12) of the residual 4 x 4 pixel block. The five characteristics utilized for the CAVLC code are discussed below.

Total Number of Coefficients and Trailing Ones

The first variable length code (VLC) used for CAVLC is the number of coefficients (coeff_token) and trailing ones (T1). coeff_token values can range between 0 to 16 for a 4x4 residual. T1 values are limited to between 0 and 3. If there are more than three +/- 1s, only the last three are considered in the T1 code. A combination of the coeff_token and T1 values map directly to a set of four look-up tables defined in the H.264/AVC standard [8]. The utilized look-up table is dictated by the number of coefficients in the upper and left previously coded 4 x 4 residuals, hence the term context adaptive.

Trailing One Signs

Figure 2.12: CAVLC 4x4 pixel scan order [2]

The sign of each T1 is coded in this VLC. A value of +1 corresponds to an output of '0' in the bitstream, and vice versa for a negative T1.

Residual Levels

Each residual value that is not a T1 is coded as a level. Levels are coded in order from highest frequency to lowest frequency coefficient. In short, levels utilize a combination of a prefix and suffix value in their coding scheme. For a complete derivation of the level VLC refer to the standard [8].

Total Number of Zeros

The total_zeros VLC is the sum of all the zeros preceding the last coefficient in the reordered residual array. This VLC is dictated by a look-up table using both the coeff_token and total_zeros values as inputs.

Run of Zeros

A series of run_before VLCs are established to indicate the total number of zeros between each non-zero coefficient. Each run_before VLC is created using a look-up table with the run_before and the total number of uncoded zeros left (zeros_left) in the array as inputs. A run of zeros located before the first non-zero coefficient does not need to be coded.

Chapter 3

KiloCore Architecture

This work presents an H.264/AVC implementation on the KiloCore II platform. KiloCore II is a large array of independent, programmable, single issue, RISC-type processors [3]. Each processor on KiloCore II contains local memory for data and instructions. Additionally, each processor has two dual-clocked first in, first out (FIFO) inputs and the ability to fan out to a total of 8 neighboring processors. On chip, KiloCore II has several large memories capable of interfacing with processors. The chip also adopts a globally asynchronous, locally synchronous clocking scheme (GALS clocking [14]) where a 2-D mesh topology passes data between cores. The presence of many independent processing units make KiloCore II well suited for computationally intensive applications [15]. Although this work was designed and simulated on the fourth-generation KiloCore II platform, the following section describes the salient features of the third-generation KiloCore chip [16].

3.1 Processors

The bulk of the KiloCore platform consists of 1000 independently programmable RISC-like cores. Each core has dual-clock FIFOs to support locally synchronous communication. Cores also contain local oscillators (allowing stall controls for increased power performance), a 128x40-bit instruction memory, and 512 bytes of data memory. Figure 3.1 presents the 7-stage KiloCore pipeline present within each core. KiloCore processors send and receive data in 16-bit words. While the processor pipeline is a fixed 16-bit datapath, other word widths are easily handled through software, such as 32-bit floating point.

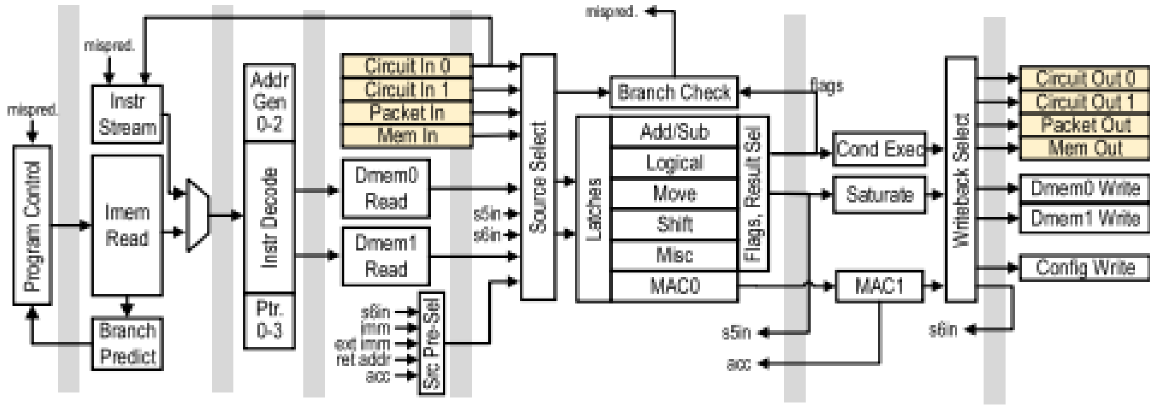

Figure 3.1: KiloCore pipeline [3]

3.2 Memory

The KiloCore chip contains 12 64kB SRAM memories. Each memory is accessible by two neighboring cores. Random and burst reads/writes are supported for these external memory modules. KiloCore memories may also be configured to increase instruction memory for a neighboring core. Each SRAM memory contains two 32x18-bit input buffers, two 32x16-bit output buffers, and one 16x2-bit processor response buffer, and supports 28.4 Gb/s of I/O bandwidth [3].

3.3 Inter-Core Communication

In the KiloCore architecture, data is passed between cores through an efficient 2-D mesh topology. Communication on chip is accomplished through a high-throughput, low-latency circuit-switched network [4] and a very small area packet router [17].

Circuit-switched links are source-synchronous and translate to the destination processor's clock domain. The KiloCore circuit-switched architecture incorporates an asymmetric and output buffered inter-processor communication scheme to obtain a good trade-off between dynamic and static routing architectures [18]. Each processor on the KiloCore chip has two circuit-switched links entering and exiting the processor at each of the four edges. This allows each processor to fan-out to a total of 8 other processors. Figure 3.2 illustrates the circuit-switched linked architecture where only signals related to the west edge are depicted for simplicity.

Each KiloCore processor contains a packet router that takes up 9% of the processor's total

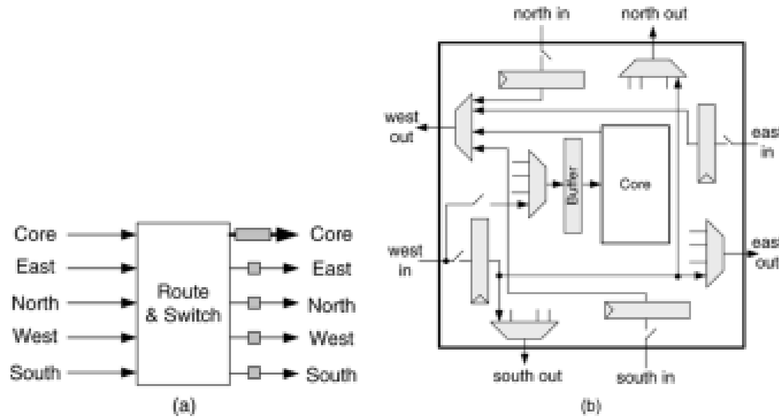

Figure 3.2: (a) 2-D mesh interprocessor communication and (b) the generalized communication routing architecture [4]

area. The packet router is effective for high fan-in communication, high fan-out communication, and administrative messaging. Each packet router supports 45.5 Gb/s of throughput and has its own oscillator, allowing for zero active power when there are no packets to process.

3.4 Design Flow and Mapping

Project Manager is an important tool used for simulating and measuring the accuracy and performance of a design on the KiloCore platform. Project manager utilizes a mix of C++ code, or assembly language, and python scripts to accurately simulate each core and how data is passed between neighboring cores. This section describes the design flow used to design a many-core processor with the VCL Project Manager and mapper tools.

Each core on the KiloCore chip is programmed using C++ code that is compiled into assembly language with a compiler. Cores are represented by a function that takes up to 2 input pointers and up to 8 output pointers. Core functions may also contain a memory port pointer if they are accessing data from one of the shared memories.

The datapath between KiloCore inputs, outputs, cores, and memories are all represented through a python dictionary. Valid elements of the python dictionary utilized in this work are input handlers (received from text files), output handlers (sent to text files), processors (cores), and memory (SRAM memory). Each element mentioned above is linked through linking functions that abstract the functionality of the KiloCore 2-D mesh topology described in the section above.

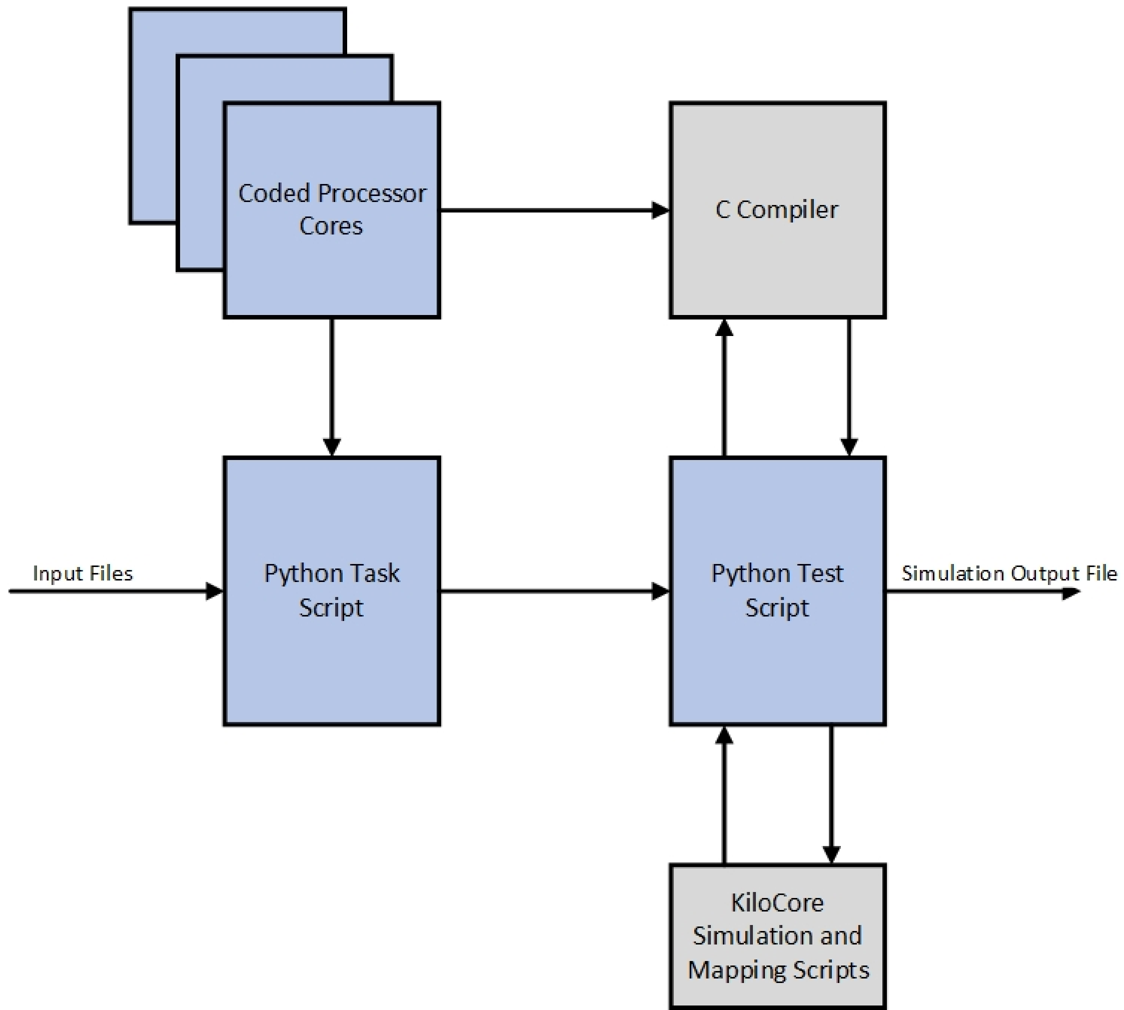

Figure 3.3: KiloCore simulation scheme including processor coding, the task script (circuit link configuration), and the test script (cycle-accurate simulation results)

Linking functions used in this work are circuit links (circuit-switched links), packet links (packet router links), and memory links (SRAM to core links).

Finally, a python test script is utilized to compile any source code, link the source code functions, determine the simulation and mapping parameters, and dump the results to a local folder. The entire Project Manager and mapper workflow is illustrated in figure 3.3.

3.5 Prior Video Codec Works on the KiloCore Platform

In this work, a full H.264 baseline encoder and decoder are designed on the KiloCore platform. Throughout the years, many research efforts were dedicated to efficient video codec

architectures on the KiloCore/AsAP platform at the UC Davis VLSI Computation Lab (VCL). This section highlights the relevant works and their scope in development of H.264/AVC sub-blocks. In turn, this section provides context as to the fundamental differences proposed in this work as it relates to prior KiloCore works.

H.264 Baseline Encoder

There is only one prior H.264 encoder work on a many-core array (AsAP2), implemented by Le [5]. AsAP2, a predecessor to the KiloCore platform outlined previously, has 167 processors, a motion estimation dedicated hardware, and three shared 16kB memories. Consequently, the AsAP2 platform has both a smaller computation power and memory when compared to the KiloCore platform. On the other hand, the AsAP2 platform has a motion estimation accelerator which makes the inter prediction design for the H.264 encoder simpler.

While Le does not present an H.264 decoder, the H.264 encoder is relatively similar to the encoder proposed in this work. Table 3.1 highlights the exact differences between the encoder implemented in Le and this work.

Le [5]	**This Work**
3 Intra prediction modes supported	9 Intra prediction modes supported
Single reference frame for Inter prediction	Single reference frame for Inter prediction
Full search motion estimation algorithm	2D logarithic motion estimation algorithm
Main partitions for motion estimation	Main and sub-block partitions for motion estimation
Forward and inverse transform blocks	Forward and inverse transform blocks
No deblocking filter	No deblocking filter
CAVLC encoder	CAVLC encoder

Table 3.1: Encoder comparisons between Le [5] and this work

H.264 Motion Estimation Accelerators

Braly [19] and Landge [20] devote research efforts to dedicated accelerator hardwares for general video compression motion estimation. For any video compression algorithm, motion estimation is the most computationally intensive sub-block. As a result, dedicated motion estimation hardware can serve as a huge performance boost for a custom H.264 implementation. For this reason, iterations of the AsAP chips contain dedicated motion estimation hardware for video compression applications.

Since KiloCore II does not contain a motion estimation accelerator, the motion estimation

algorithm in this work is implemented on the processor array's programmable cores. With this said, the possible performance improvements of the proposed design with a motion estimation accelerator are estimated later in this work.

H.264 Encoders

Additional researchers in the UC Davis VCL have explored the design of several H.264 compatible entropy encoders. Xiao [21] implements an H.264 CAVLC encoder on the AsAP architecture. In contrast, Kulkarni [22] implements the alternative and more complex H.264 entropy encoder CABAC.

This work implements CAVLC as the entropy encoder for the proposed codec. While the works mentioned above implement entropy encoders, this work implements both a CAVLC encoder and decoder. The CAVLC decoder presents several differences to the encoder design such as handling a bitstream of variable length coded units.

Chapter 4

RTL Implementation

4.1 Design Overview

In order to understand the design requirements of an H.264/AVC codec implementation, a register-transistor level (RTL) solution is explored in this work. This section discusses an implementation of a fully synthesizable RTL monochrome codec that is bit-accurate with respect to the proposed constrained baseline H.264/AVC model.

In order to represent a scalable codec in hardware, large data structures are stored in main memory. Consequently, as the video frame size increases the algorithm will not need to increase the number of instantiated registers. Main memory includes reconstructed video frames, number of coefficients for each neighboring 4x4 block, and previously coded Intra prediction modes. Memory is represented as a series of 2-D arrays within the high level Verilog testbench. A delay penalty is not introduced between the main memory fetch in the Verilog testbench and the input to the core Verilog code. Figure 4.1 illustrates how the main memory modules, inputs, and outputs interface with the core Verilog code.

The following sections discuss the design of the proposed Verilog codec in detail. The terms "test bench" code and "core" code used in this section differentiate the Verilog testbench script and synthesizable Verilog code respectively.

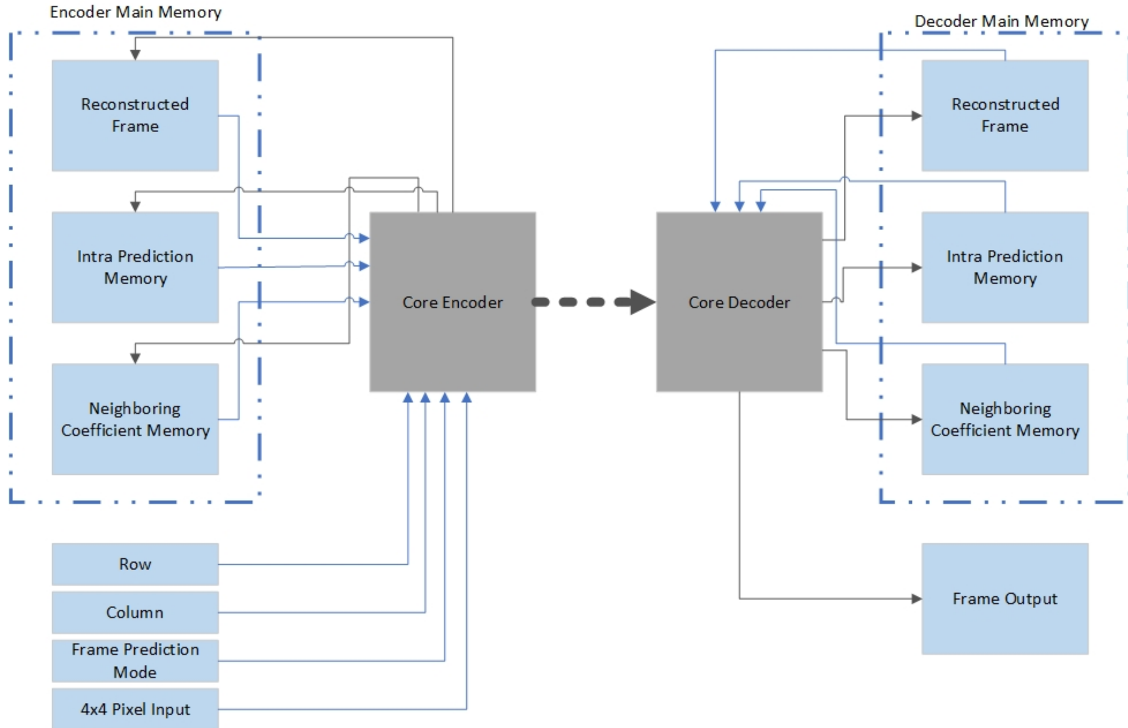

Figure 4.1: Test bench (blue) and core Verilog (gray) interface

4.2 Encoder Design

4.2.1 Input Handling

A protocol is implemented to handle the relay of data between testbench and core code. The testbench parses macroblock frame data from a ".txt" file in raster scan order. Macroblocks are then further parsed into 4x4 pixel blocks, also in raster scan order, and sent to the core Verilog code.

The proposed encoder algorithm operates in units of macroblocks, i.e. the top-level FSM is reset every time a new 16x16 pixel macroblock is processed. Input data into the core Verilog is designed to remain constant until a 4x4 pixel block is completely encoded. A "CAVLC_done flag" is sent high for one clock cycle to update all inputs into the core code to align with the new 4x4 pixel block to be processed. Additionally, the bitstream output of the core code is transmitted to a ".txt" file and the reconstructed frame output is stored in a 2-D memory array within the test bench, available for either the inta or inter prediction algorithms. Table 4.1 lists all inputs and outputs to the core encoder logic.

Encoder Core Inputs	**Description**	**Encoder Core Outputs**	**Description**
go	MB go signal	REC 4x4	Reconstructed 4x4 pixel output
4x4 pixel block	Current pixel block under process	bitstream	MB compressed bit-stream
Row, Col	Top left pixel index for current MB	bitstream pointer	Bitstream length
nU, nL	Top and left neighboring total coefficients	CAVLC_done	Flag to initiate new 4x4 pixel process
predU, predL	Top and left neighboring Intra predictions	MB_Done	Flag to initiate new MB process
REC Pixels	Sequence of reconstructed pixels for predictions	Pred Out	Intra prediction mode output
Pred Mode	Current frame prediction mode	nCoeff Out	Total coefficients output
QP	Frame quantization parameter	-	-
h, w	Frame height and width	-	-

Table 4.1: Verilog encoder inputs and outputs

4.2.2 Datapath

Within the core Verilog logic, a control scheme is implemented using a finite state machine triggered by a series of done signals from specific combinational blocks. The primary benefit of this control scheme is that the top level logic remains unaffected by the number of cycles of lower-level blocks. The top level finite state machine branches two different combinational blocks (inter and intra prediction), but both blocks pass a 4x4 pixel prediction and residual to the same forward transform and CAVLC hardware to conserve area. The proposed encoder finite state machine is illustrated in figure 4.2.

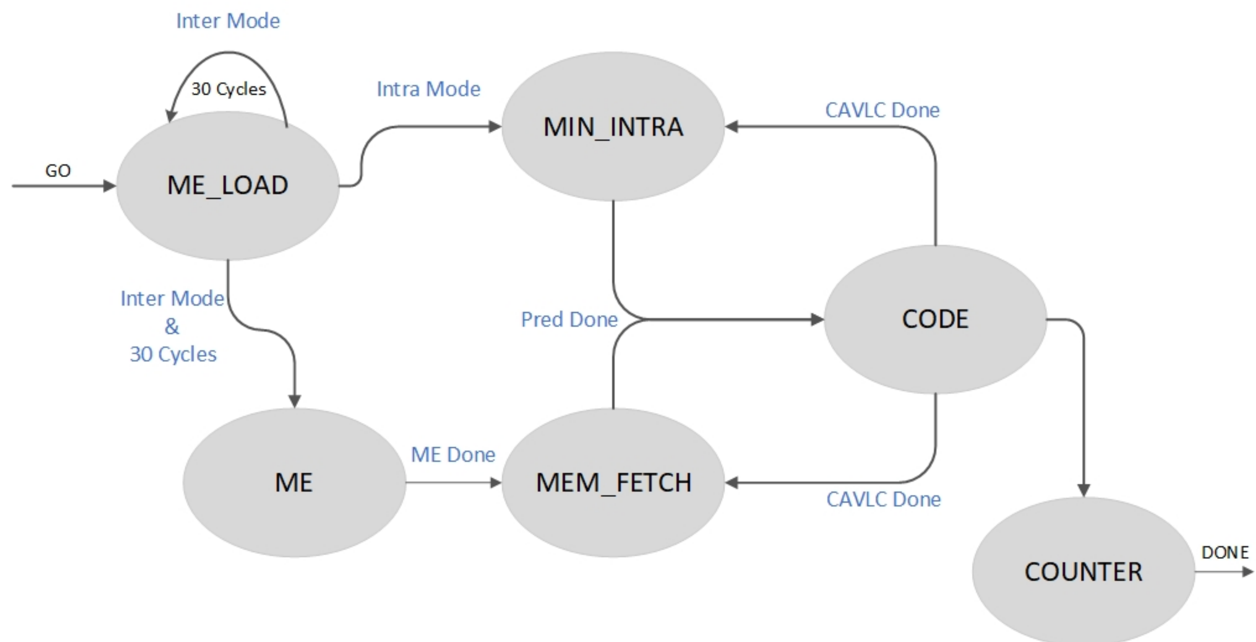

Figure 4.2: Encoder top finite state machine

ME_LOAD

The ME_LOAD state is the initial state for the top finite state machine. A go signal is sent every time a new 16x16 pixel macroblock is processed in the test bench code.

If the prediction mode input indicates an inter prediction frame, this state remains current for 30 cycles. A row of 30 pixels from the reconstructed memory is loaded one-by-one into a core local memory during each clock cycle. This 30x30 pixel block is stored in local memory and is used as the Inter prediction search region. After 30 cycles the state machine passes to the ME state.

If the prediction mode input is Intra prediction, the state machine passes to the MIN_INTRA_PRED

state in the next cycle.

MIN_INTRA_PRED

The MIN_INTRA_PRED state receives a 4x4 pixel input, neighboring reconstructed pixels, and neighboring intra prediction modes from the core Verilog input. The intra prediction algorithm is executed in order to find the lowest error prediction mode, described Section 4.2.3. The outputs of this state are a done signal, 4x4 pixel residual, and a 4x4 pixel prediction. A done signal initiates the CODE state.

ME

The ME state calculates a full 16x16 MB prediction for the currently processed macroblock. Full macroblock and sub-macroblock, if needed, cost estimations are executed in this state. The entire 16x16 prediction is stored in local memory and a done signal initiates the MEM_FETCH state. A high level description of the ME state is discussed in section 4.2.4.

MEM_FETCH

The MEM_FETCH state processes 4x4 pixels from the motion estimation pixels stored in local memory. 4x4 pixels, selected in raster scan order, are passed to the forward transform and encode block and a done flag initiates the CODE state.

CODE

The CODE state processes a 4x4 pixel residual and performs transform, quantization, and CAVLC encoding. This state also reconstructs residuals for the prediction mode in the next frame. When the residual is completely encoded, a done signal initiates either the MIN_INTRA_PRED or MEM_FETCH state (depending on the frame prediction mode) to fetch the next 4x4 residual. If the entire MB is processed (16 4x4 residuals), a done signal is sent from the core Verilog code to indicate the test bench code to pass to the next MB.

4.2.3 Intra Prediction

As previously stated, the Intra prediction decision scheme for this work is to compute all prediction modes and find the smallest sum of absolute error (SAE). Intra prediction modes only require a combination of shifts and additions. Each SAE block finds the absolute value of all residual pixels and sums all the elements. The output of this block is the mode value and residual produced by the Intra prediction mode with the smallest SAE.

The complete MIN_INTRA_PRED datapath is enclosed in one pipeline. Figure 4.3 illustrates the hardware design for the Intra prediction decision process.

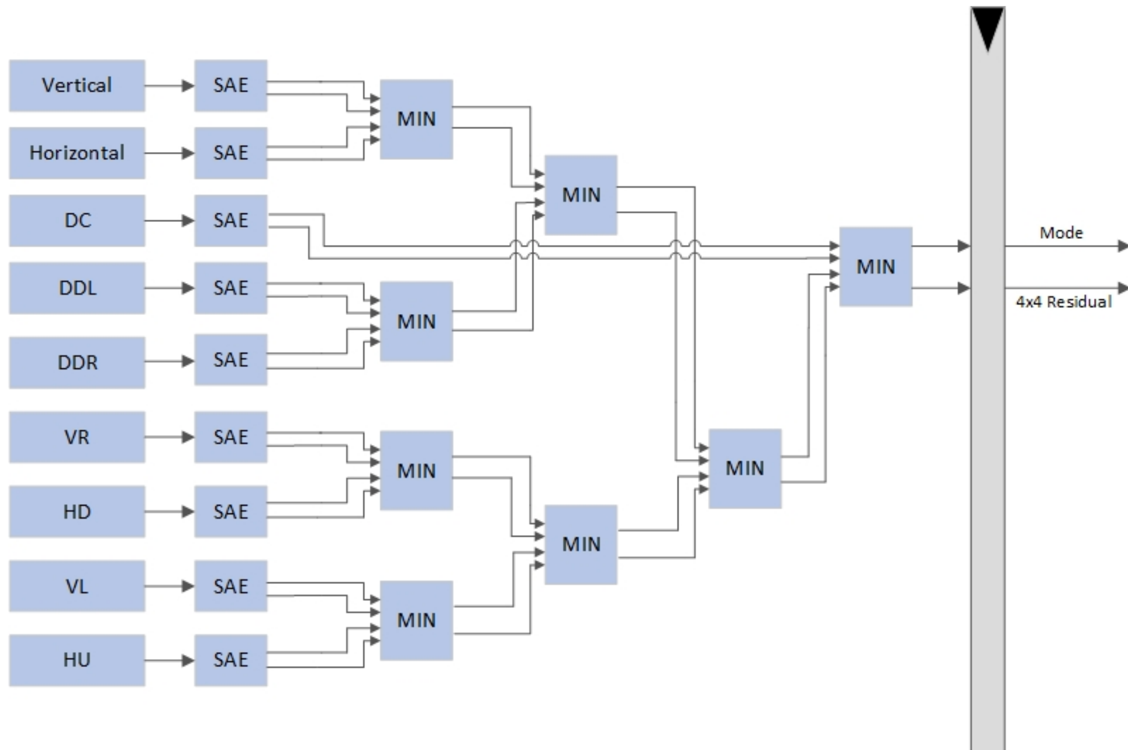

Figure 4.3: Encoder Intra prediction block diagram

4.2.4 Inter Prediction

Motion estimation is the most computationally intensive task for an H.264/AVC codec. Consequently, a great deal of design effort and hardware iterations were required to make this block synthesizable. The following three subsections discuss salient hardware blocks of the proposed inter prediction motion estimation engine.

Local Memory

In this work inter prediction is executed using a 2-D logarithmic search algorithm, as discussed in section 2.6. Unlike intra prediction, inter prediction processes a full MB at a time. Additionally, the inter prediction pixel search region is significantly larger than the neighboring reconstructed pixels required for intra prediction, 900 8-bit pixels compared to 9 8-bit pixels respectively. Figure 4.4 illustrates the discrepancy between pixels processed in the proposed intra

and inter prediction design. Across all partition modes in inter prediction, it is unfeasible to store all reconstructed and input pixels in synthesized memory flip-flops dynamically. Consequently, two separate block memory modules are designed to handle both the reconstructed search window and input pixels.

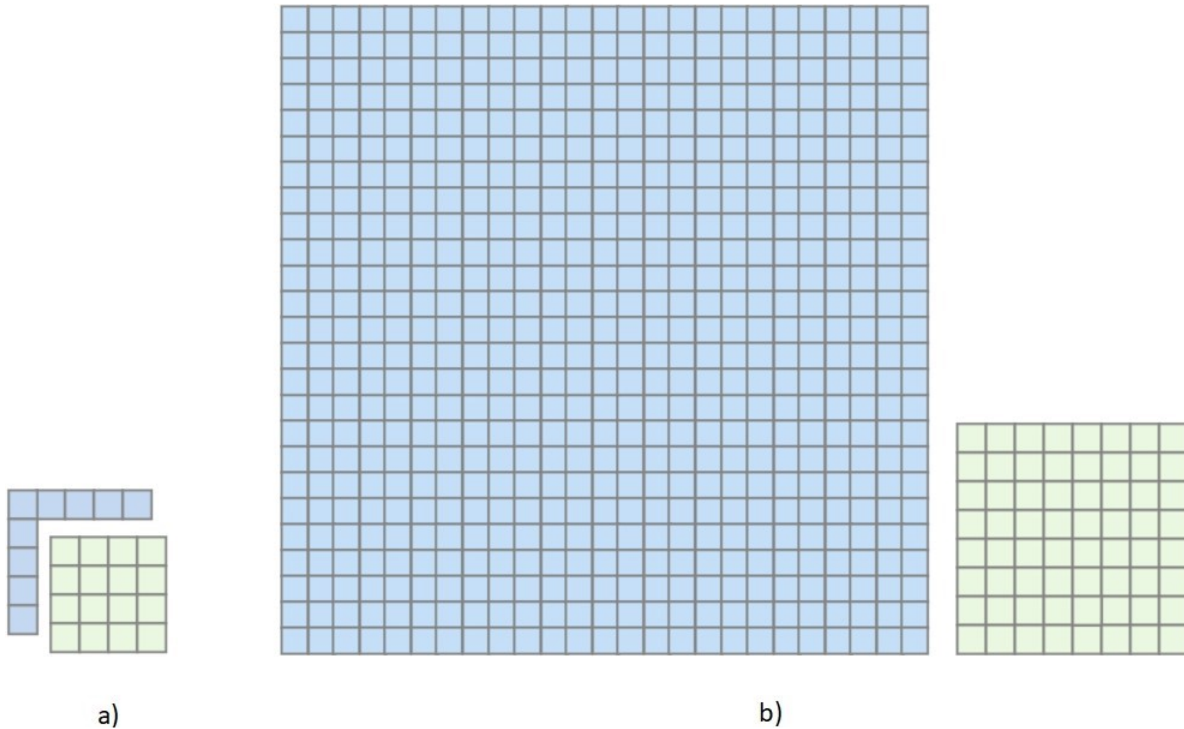

Figure 4.4: Input (green) and memory (blue) required for intra (a) and inter (b) prediction

The input local block memory reads and writes one row of the input MB (16x8-bit) in a single cycle. A read operation is always performed, but memory writing is only achieved if the write "enable_bit" is high.

Like the input block memory, the reconstructed memory writes one row of the reconstructed search window (30x8-bits) in one cycle. In total, the reconstructed memory is capable of reading 20x16x8-bit pixel rows corresponding to every partition configuration in every cardinal direction, including center, of the 2-D logarithmic search algorithm. Consequently, independent row and column read signals are provided to account for every row output. Vertical and quad partitions have twice as many read indexes to allow independent indexing of the first and second half of the row. This memory scheme allows each partition mode to operate completely in parallel in the motion estimation hardware block.

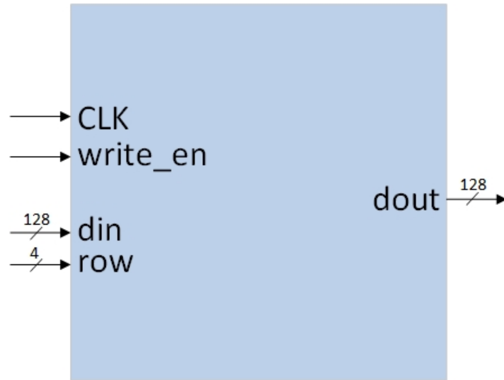

Figure 4.5: Input block memory

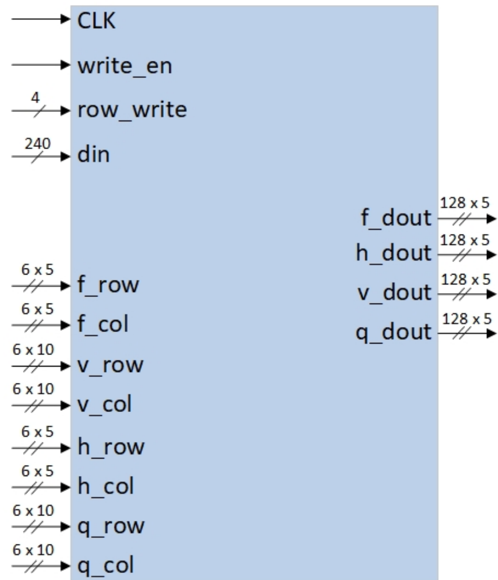

Figure 4.6: Reconstructed block memory

Motion Estimation Core

A key algorithmic block of the motion estimation algorithm is the SAE calculation for each partition. To optimize performance, all partitions are calculated in parallel. For each partition the SAE between the input sequence and reconstructed pixels are calculated one row at a time. Each partition calculation in figure 4.7 calculates the center, top, right, bottom, and left cost respectively. After 16 cycles, or 8 cycles in the case of sub-macroblock partitions, the minimum cost for each direction of a partition is calculated. The indices of the minimum value are passed as the output to be used as the center point for the next iteration of the 2-D logarithmic search.

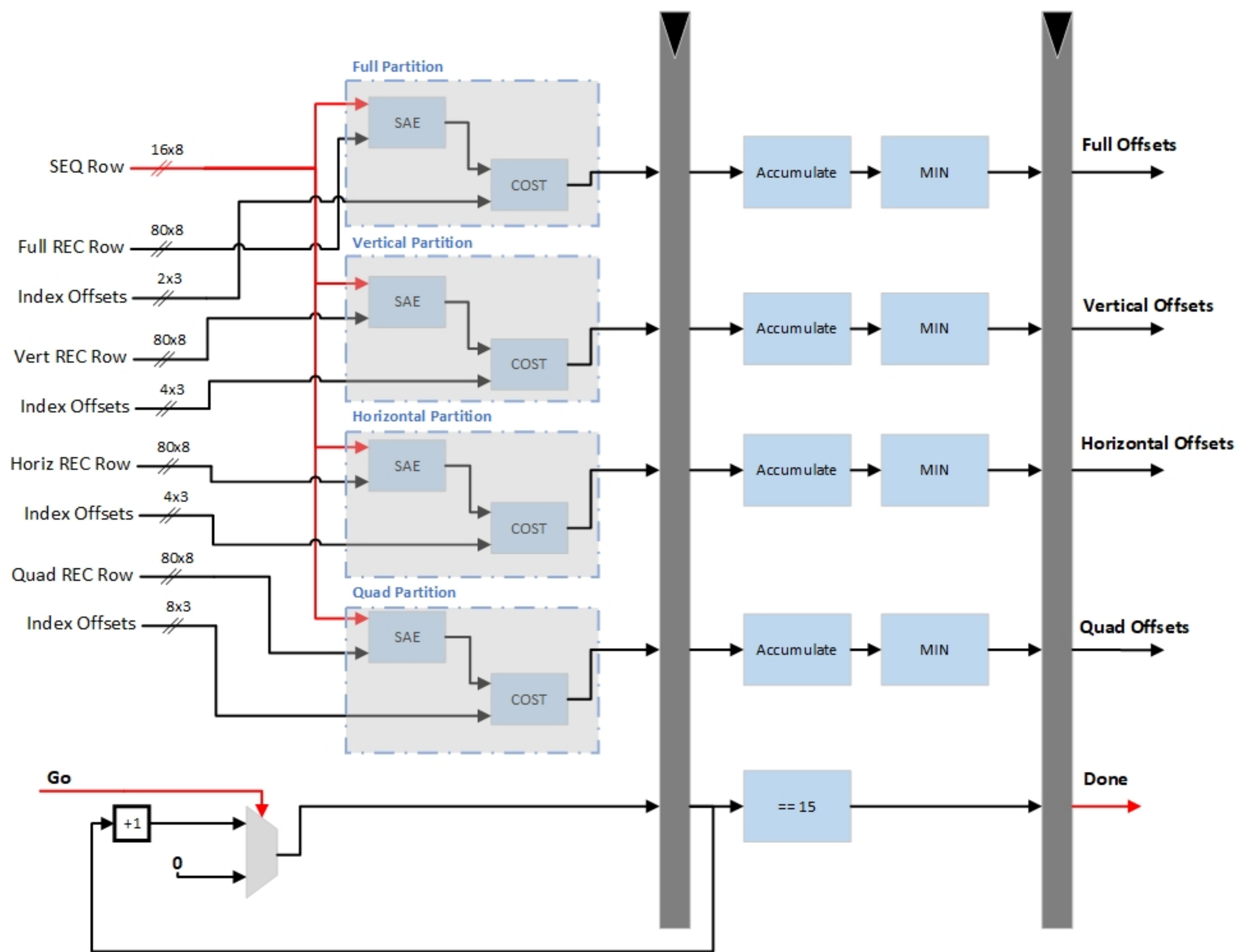

Figure 4.7: Core motion estimation block

Motion Estimation Top

The highest level hardware block of the motion estimation engine combines the functionali-

ties of the local memory and the core motion estimation blocks. The top motion estimation block, illustrated in figure 4.8, manages the memory blocks to fetch a subsequent row for each partition condition at each cycle. Additionally, this top block iterates the core motion estimation block a total of three times. After the last iteration, the index offsets for full, vertical, horizontal, and quad modes are passed to a block that calculates the sum of squared difference cost for each partition. The mode with the smallest cost error is passed to the output of this hardware block.

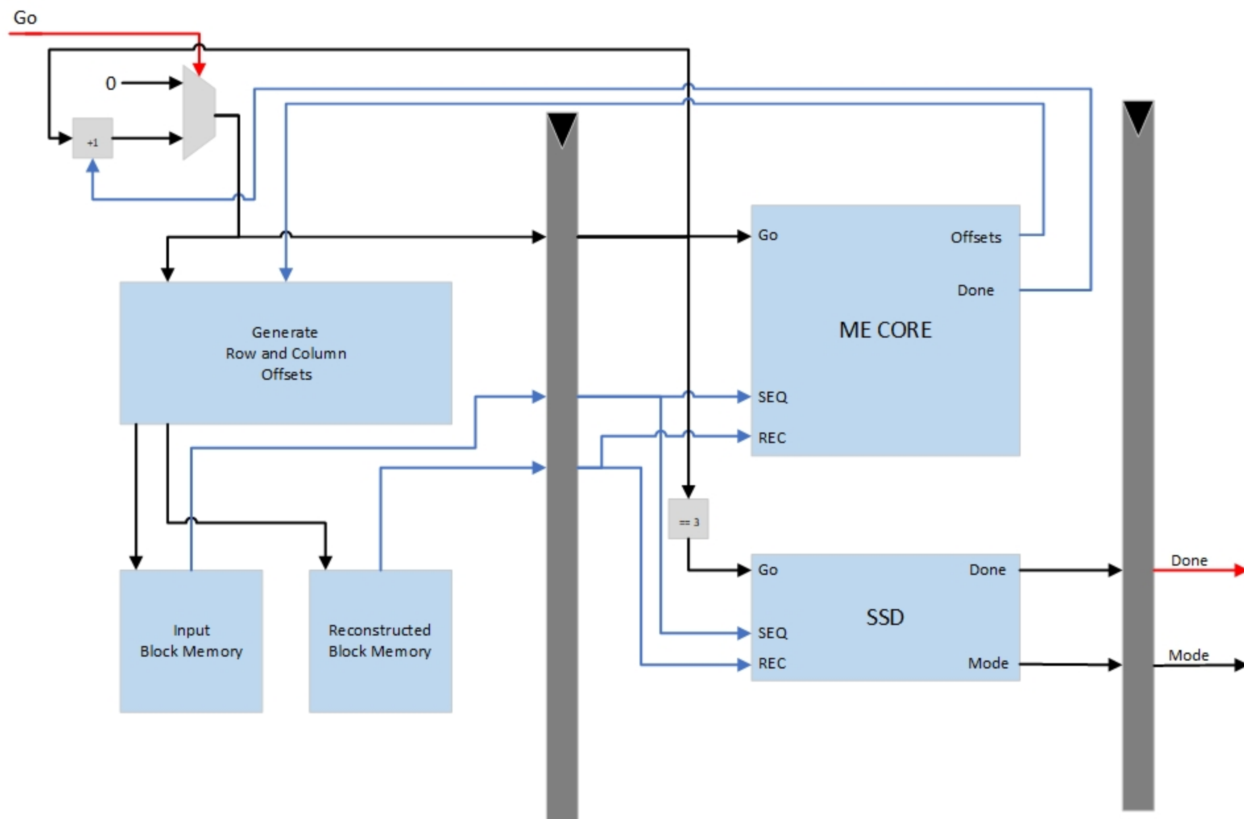

Figure 4.8: Top motion estimation block

4.2.5 Encoder Overview

The proposed H.264/AVC encoder contains a forward transform, inverse transform, and CAVLC block. All three of these blocks are shared by the outputs of Intra and Inter prediction mode blocks. While low-level design details regarding the three encoder blocks are not provided, the top level pipeline for the encoder is provided below.

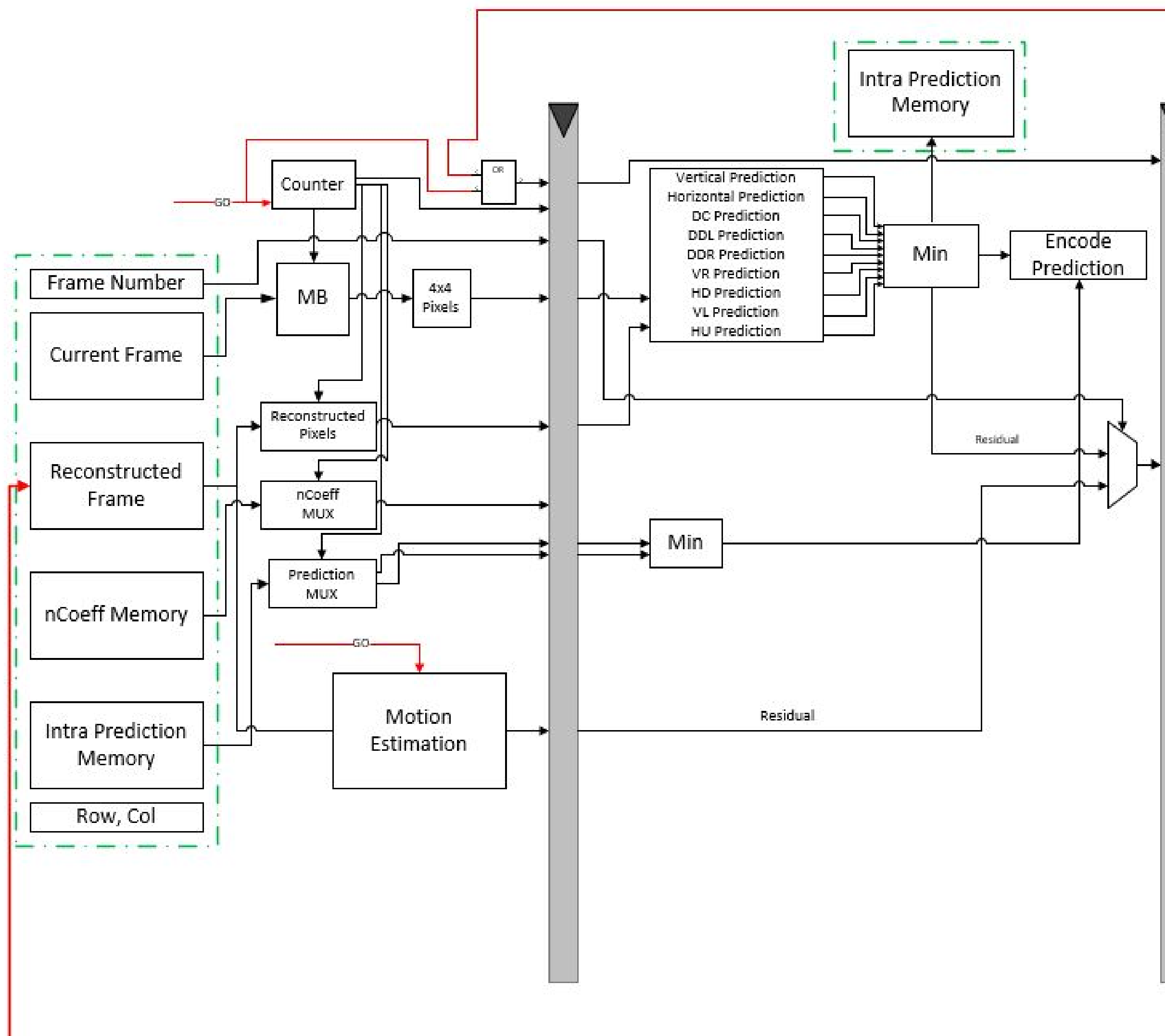

Figure 4.9: Encoder prediction and residual generation datapath

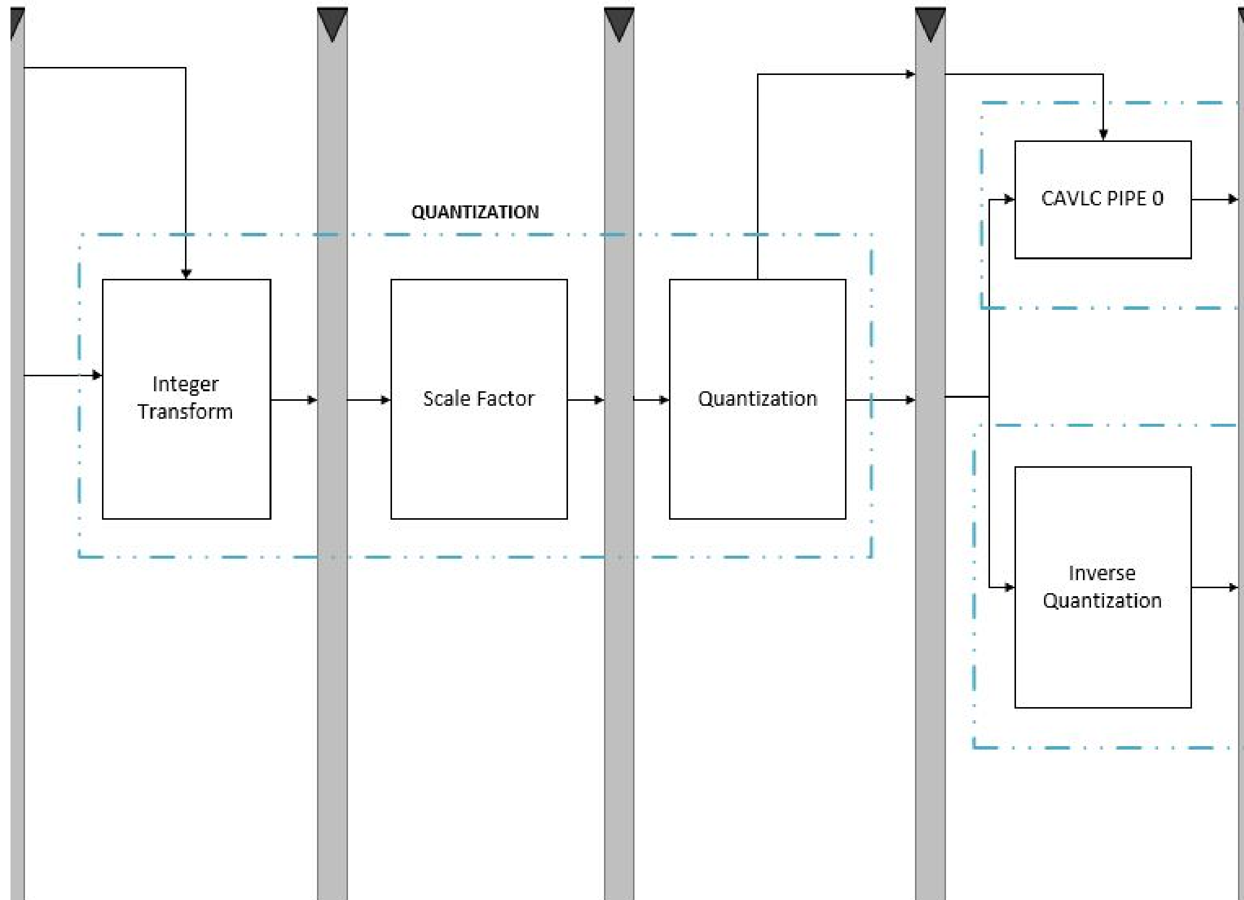

Figure 4.10: Encoder forward transform and quantization datapath

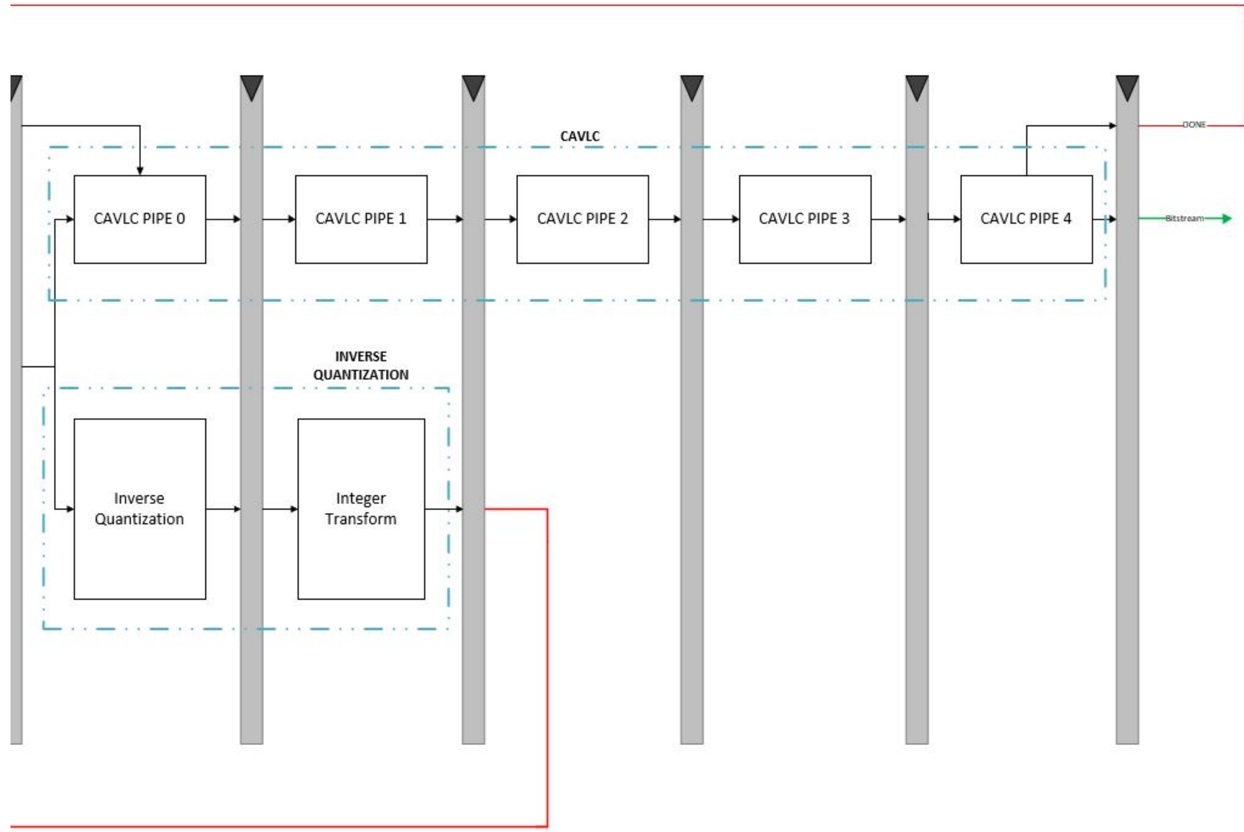

Figure 4.11: Encoder reconstruction and CAVLC datapath

4.3 Decoder Design

4.3.1 Input Handling

Input handling for the H.264/AVC decoder differs greatly compared to the encoder. While the encoder sends a defined width 16 pixel input, the decoder receives a variable length bitstream. Consequently a protocol between the core decoder and Verilog test bench is created to deal with the variable length input.

In the proposed Verilog decoder, the test bench sends the bitstream input incrementally in word widths of 512 bits. Through intensive testing and simulation, a 4x4 output from CAVLC never exceeds this 512 bit limit. Consequently, it is assumed that a complete inverse CAVLC decoding can be executed for each bitstream input. After inverse CAVLC is executed, the core Verilog code sends a CAVLC_done flag to update the bitstream input and bitstream pointer with the most current index within the total bitstream. Table 4.2 lists all inputs and outputs for the core Verilog decoder.

Decoder Core Inputs	Description	Decoder Core Outputs	Description
go	MB go signal	Done	MB completely processed
bitstream	Compressed bitstream	CAVLC_done	Update bitstream input
bitstream pointer	Current index of bitstream	nCoeff Out	Total coefficient for constructed 4x4 pixels
Row, Col	Top left pixel index for current MB	Pred Out	Intra prediction mode for 4x4 pixels
nU, nL	Top and left neighboring total coefficients	REC 4x4	Reconstructed pixel output for 4x4 pixels
predU, predL	Top and left Intra prediction modes	bitstream pointer	Current index for bitstream pointer
REC pixels	Sequence of reconstructed pixels for predictions	-	-
QP	Frame quantization parameter	-	-
h, w	Frame height and width	-	-

Table 4.2: Verilog decoder inputs and outputs

4.3.2 Datapath

As with the H.264 encoder, a top level finite state machine is designed to handle the different combinational blocks of the decoder. Since the decoder does not contain prediction decision hardware, the state machine is simple. Figure 4.12 illustrates the decoder's top level finite state machine.

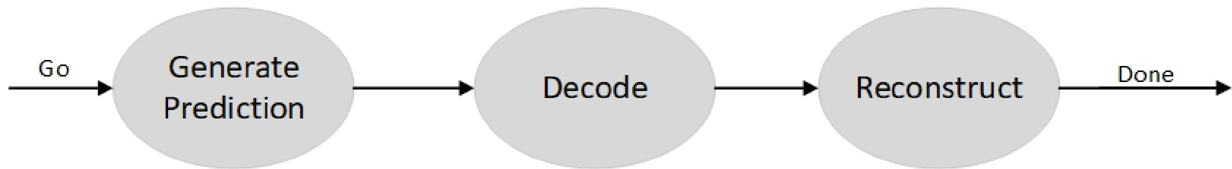

Figure 4.12: Decoder state machine

Prediction Generation

Initially, a macroblock's prediction header is decoded. This includes the prediction mode for the macroblock and relevant prediction generation information. In the case of an I macroblock, the intra prediction mode is decoded. Conversely, P macroblock motion vectors and partition type are decoded.

Decode

This state decodes a video sequence with the inverse CAVLC block and inverse quantization. The output of this state is decoded residual pixels.

Reconstruct

This state simply adds the residual from the decode state and the prediction from the prediction generation state. The output is the reconstructed 4x4 pixel block.

4.3.3 Intra Prediction

The intra prediction mode header is decoded before the residual. Each macroblock has a mode bit indicating whether the following macroblock is encoded using intra or inter prediction. In the case of an intra prediction macroblock, the prediction mode is decoded according to the algorithm in section 2.5.1. The prediction mode value is the control bit to a 9:1 mux. As illustrated in figure 4.13, the output of this mux is the current 4x4 pixel prediction.

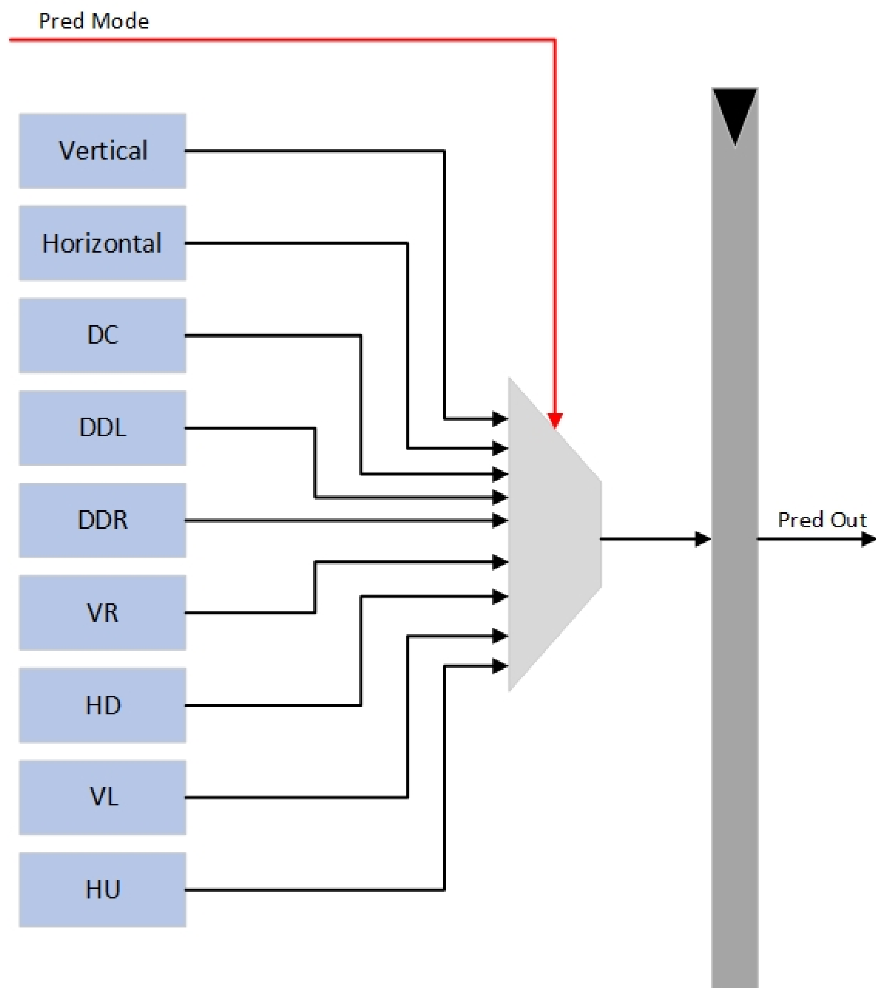

Figure 4.13: Decoder Intra prediction

4.3.4 Inter Prediction

The inter prediction decoder derives partition modes and motion vectors from the bitstream. As seen in figure 4.14, a single set of mode and motion vectors are decoded for macroblock partitions in comparison to four for sub-macroblock partitions. Using the motion vectors, reconstructed pixels are fetched from main memory, in the test bench code, in one cycle. 4x4 pixel blocks are passed from main memory in raster scan order.

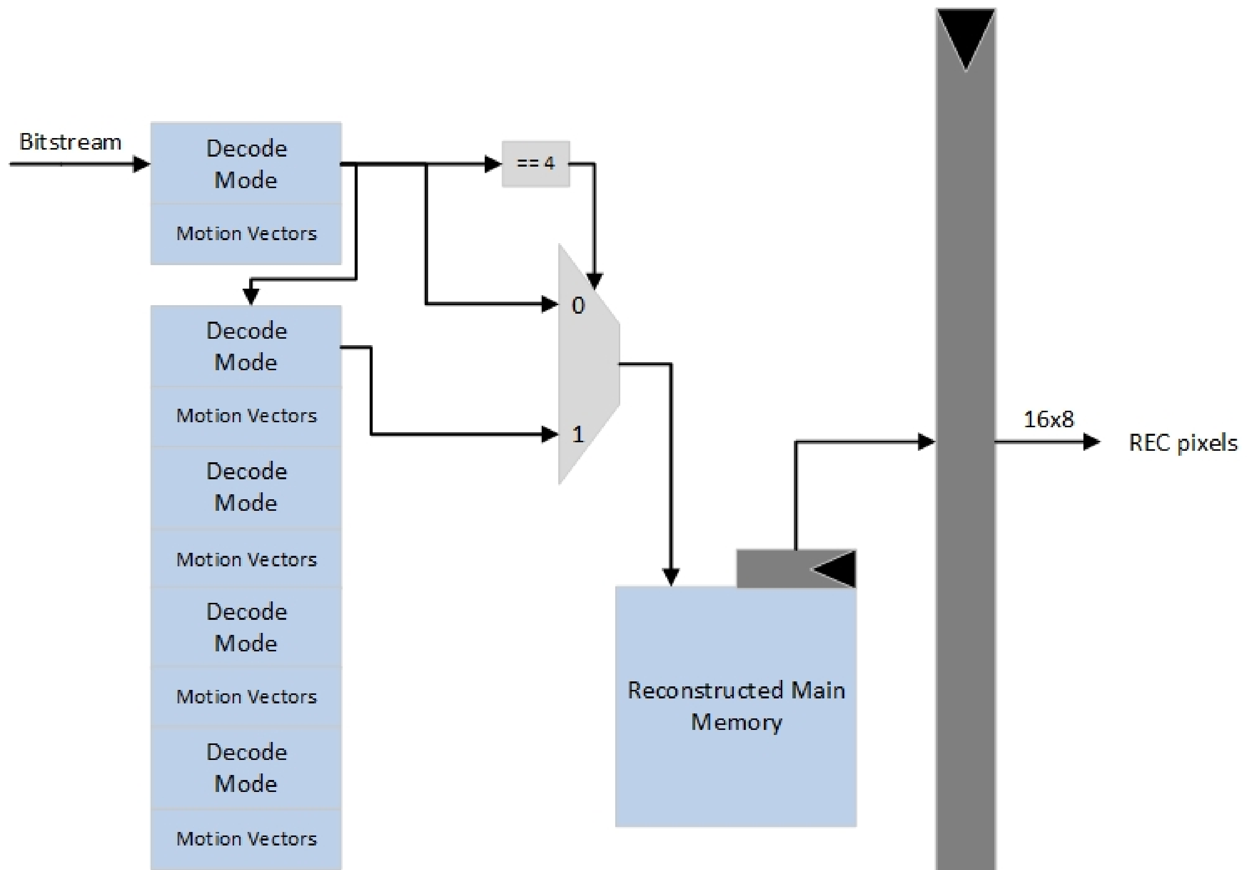

Figure 4.14: Decoder Inter prediction

4.3.5 Decoder Overview

The top level pipeline diagram for the proposed decoder is presented in figures 4.15 and 4.16.

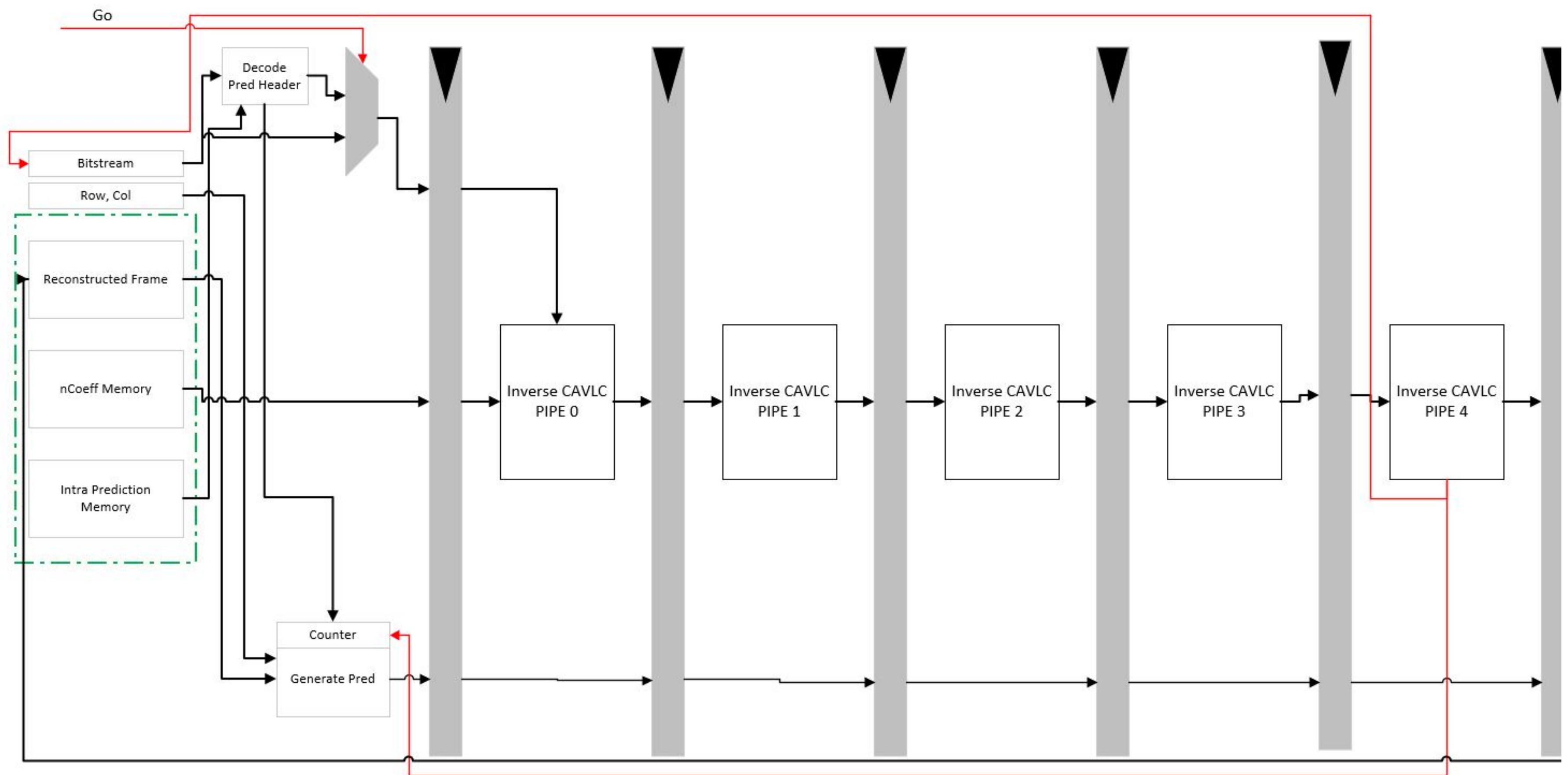

Figure 4.15: Decoder prediction generation and inverse CAVLC

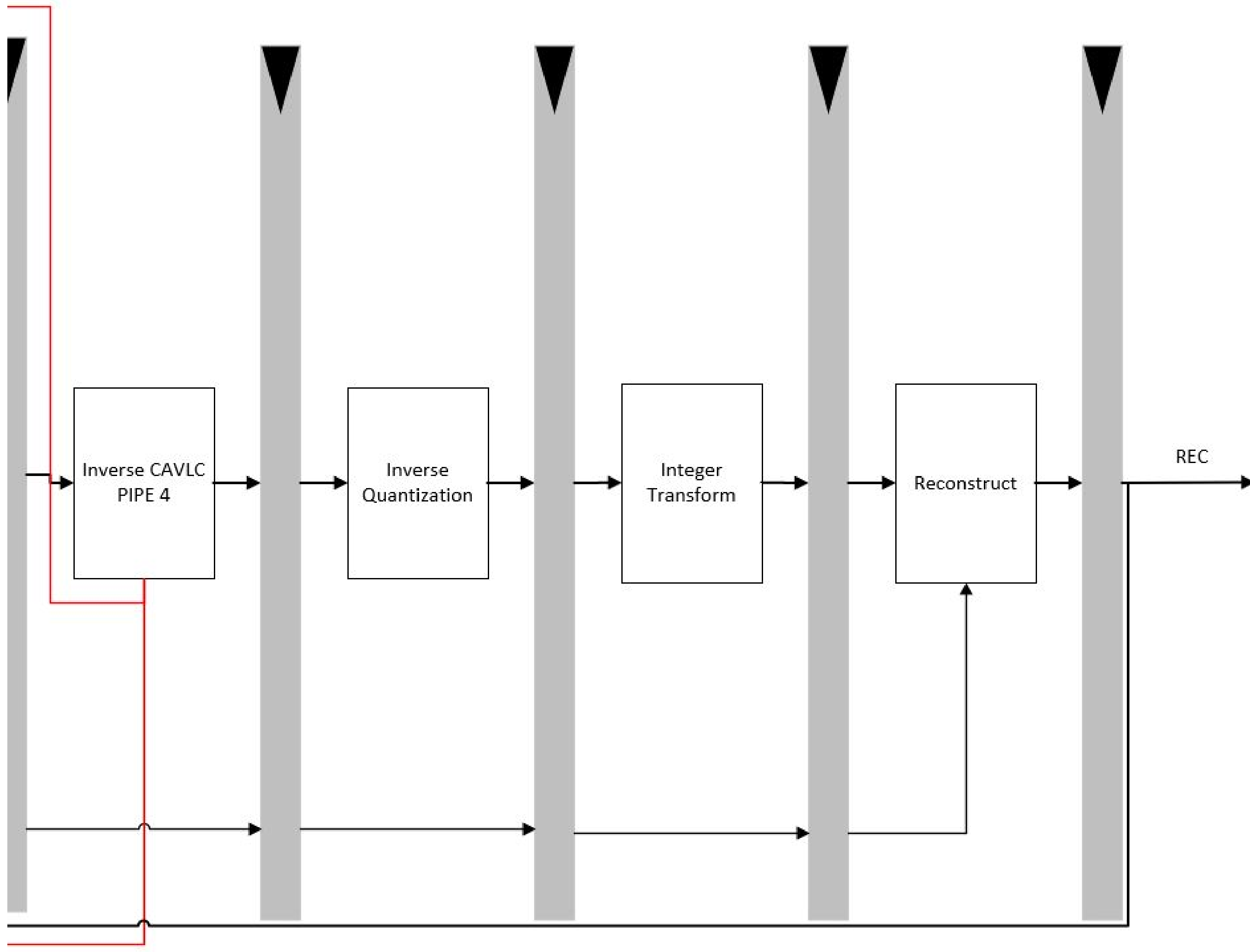

Figure 4.16: Decoder inverse transform and reconstruction

4.4 Results and Analysis

The RTL model proposed in this work is simulated using the 45 nm NanGate FreePDF45 Open Cell Library which is an open-source library developed by NanGate, Inc [23]. Results for the RTL model are synthesized in design compiler, but not implemented on an FPGA. Additionally, the RTL code discussed in this work handles all large memory structures in the testbench logic where a memory fetch penalty is not estimated. Video compression hardware critical paths are typically dictated in large part by memory read and write penalties. Consequently, the following synthesized results for the proposed RTL model are not provided in the final comparisons of this work.

Total cycles are measured in the RTL testbench in order to calculate throughput performance. The results for the RTL model, presented in table 4.3, are a rough indication of FPGA performance for a first-pass design of the proposed codec. Since power results are inaccurate in synthesis, only the throughput and area results are presented

Work	**Technology**	**Area (mm^2)**	**Clock Frequency (MHz)**	**Throughput (mbps*)**	**Scaled Throughput ($mbps/mm^2$)**
RTL Encoder	45 nm	20.4	142	69260	3395
RTL Decoder	45 nm	12.3	102	64539	5247

*Throughput measured in macroblocks-per-second (mbps)

Table 4.3: The proposed RTL encoder and decoder performance

Although the RTL model is not used as a comparison work, the design of video codec hardware in RTL offers several key insights that were considered while designing the fine-grained implementation. Namely, how the area and performance trade-off corresponds to specific motion estimation configurations in hardware. Additionally, the RTL model indicates the need for well designed and tested top-level control algorithms when designing dedicated video compression hardware.

Chapter 5

Fine-Grained CPU Array Implementation

5.1 Design Overview

The KiloCore II platform offers two distinct advantages for designing an H.264/AVC codec. These advantages are exploited in the following fine-grained, many-core video compression design.

The first benefit of KiloCore II is that data is fundamentally handled in a "data in, data out" format. Consequently once data is polled from a processor's FIFO, it manipulates the data according to its programmed function and passes data downstream to the next processor. KiloCore II processors will stall and wait for FIFO inputs if unavailable, eliminating the need for counters as common with RTL code. This offers a significant advantage because complex control structures are not needed during design.

The other advantage the KiloCore II platform offers is a substantial opportunity to parallelize code. The design choices in this section are chosen to exploit this trait.

5.2 Shared Memories

The utilization of the 64 kB shared memories is identical for both the encoder and decoder. The memories utilized in this work are outlined in table 5.1.

The four number of coefficients and intra prediction mode memories read two words (top and left neighboring components) and store one word successively. Consequently both memories are

Memory	Description
Luma_REC0	First luma reconstructed pixel memory
Luma_REC1	Second luma reconstructed pixel memory
Chroma_u_REC	U-component chroma reconstructed pixel memory
Chroma_v_REC	V-component chroma reconstructed pixel memory
nCoeff_luma	Number of coefficients for luma 4x4 blocks
nCoeff_u	Number of coefficients for U-component 4x4 blocks
nCoeff_v	Number of coefficients for V-component 4x4 blocks
IntraPred_mode	Intra prediction modes for 4x4 blocks

Table 5.1: KiloCore II shared memory utilization

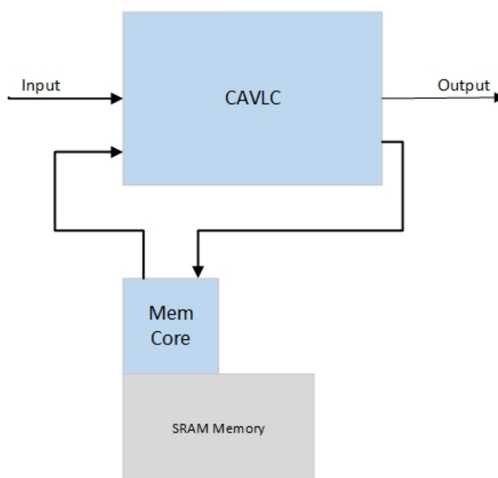

Figure 5.1: Number of coefficients memory

designed to read two words, stall until the output is received, and store the resulting value. This entire algorithm is implemented using one processor linked with a SRAM memory as illustrated in figure 5.1. Since data is stored in memory before the next value is read, compression performance is affected by memory read and write speeds.

Figure 5.2 illustrates the architecture for chroma reconstructed pixels. The intra prediction memory core always reads reconstructed pixel samples for either intra or inter prediction. Since intra prediction accesses reconstructed pixels from the same frame, data is not accessed until the current macroblock is reconstructed and written back into memory. Inter prediction does not have this requirement because the previous frame is used as the predictive reference. Since all motion vectors are 0 for chroma inter prediction, current frame reconstructed pixels just overwrite the previous frame's reconstructed pixels.

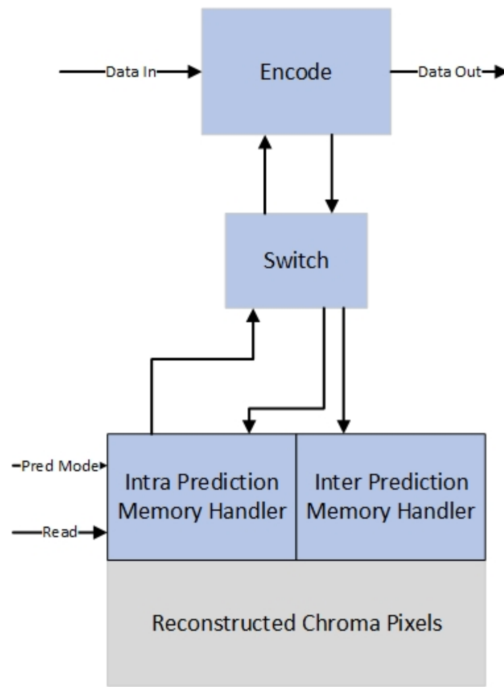

Figure 5.2: Memory architecture design for reconstructed chroma elements

The memory scheme required for luma reconstructed pixels is the most complex in this work. For inter prediction frames, a reconstructed pixel memory is needed for prediction generation and the other is needed for the next frame's prediction generation. For this reason, a control scheme must be established to ensure memory conflicts do not ensue. In the proposed H.264 codec two SRAM memories are alternated for even and odd frames as depicted in figure 5.3.

5.3 Encoder

The KiloCore II encoder is bit accurate to the proposed H.264 encoder software model. The input video sequence passed onto the KiloCore II chip is assumed to be sequenced in raster scan order and the output is structured in the exact same manner as the input. Unlike the RTL encoder, the KiloCore II encoder does not require off-chip memory. Relevant data structures are stored in the SRAM shared memories. The proposed encoder is tested using a QCIF video sequence where it is possible to store an entire video frame in one 64kB memory. Five global inputs are passed into the encoder before the beginning of the video sequence. These inputs include frame height, frame width, quantization parameter, frame start, and frame end. The following sections discuss key hardware blocks for the encoder.

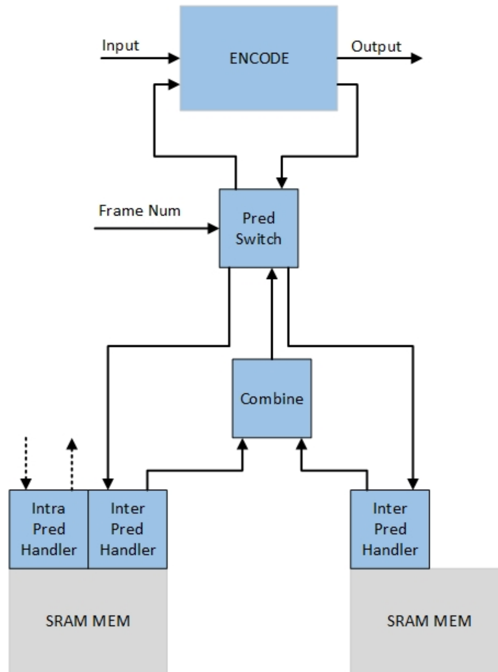

Figure 5.3: Inter prediction memory switch

Inputs	**Description**	**Outputs**	**Description**
SEQ pixels	Video sequence input	bitstream	Compressed bitstream
h, w	Frame height and width	-	-
QP	Video sequence quantization parameter	-	-
f_start, f_stop	Frame start and stop	-	-

Table 5.2: KiloCore II encoder inputs and outputs

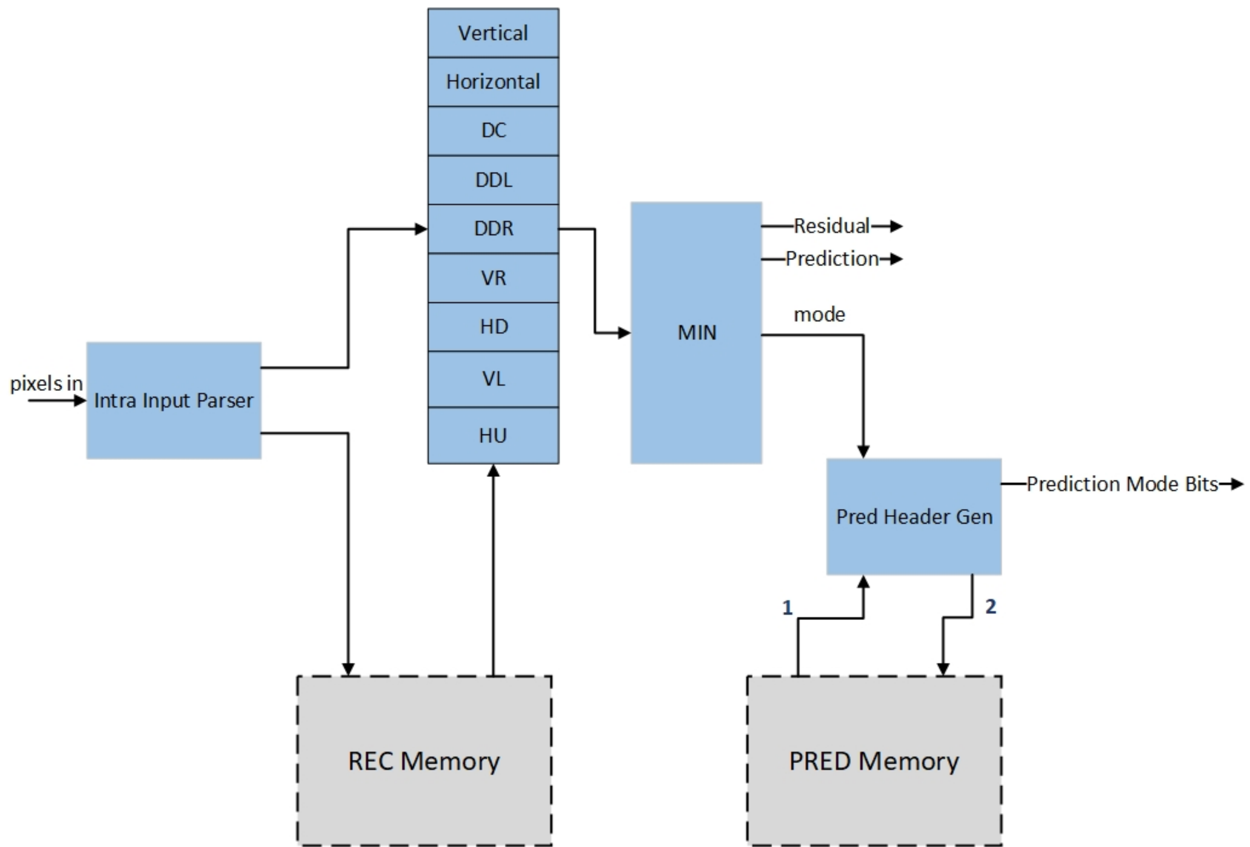

Figure 5.4: KiloCore II luma intra prediction decision

5.3.1 Intra Prediction

Intra prediction on KiloCore II is performed in a similar manner to the RTL implementation. All nine intra prediction modes are executed in parallel in hardware. These prediction modes are calculated once the input sequence pixels and neighboring reconstructed pixels are available. Eight cores are used to find the minimum SAE among all modes and route the desired prediction to a singular core output. The final output is a block of residual pixels for encoding and a block of prediction pixels for reconstruction. Additionally, the selected prediction mode is stored in memory for future intra predictions. The intra prediction mode is also encoded and sent to the bitstream prediction header as depicted in figure 5.5.

Intra prediction for the chroma elements is executed in a similar fashion to the luma elements. The key caveat is that chroma intra prediction is performed on blocks of 8x8 pixels and only has four available prediction modes. In order to handle a larger block size, the intra prediction algorithm for each mode is split into four parallel computations. Next, the residuals and predictions

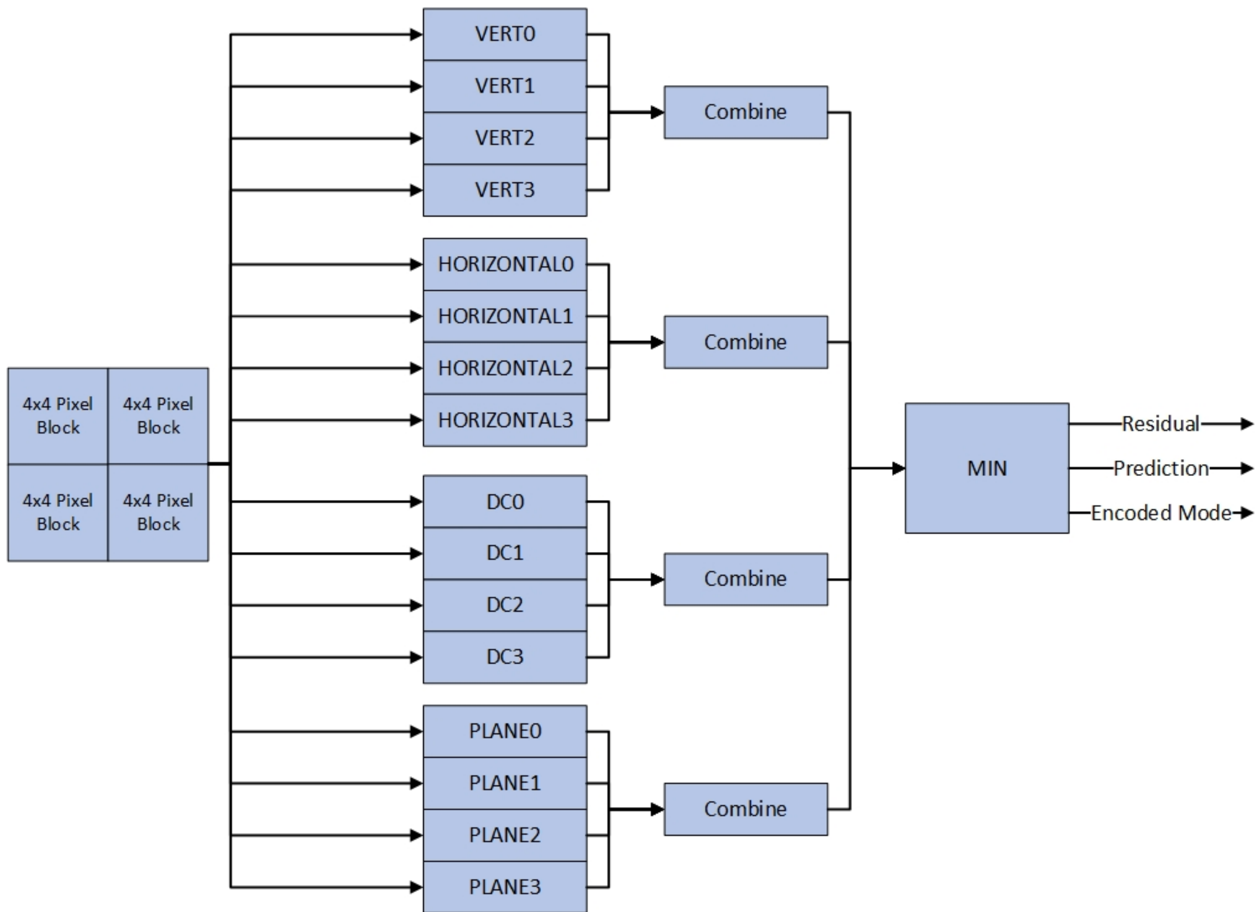

Figure 5.5: KiloCore II chroma intra prediction decision

are recombined and the minimum SAE is found similar to how the luma elements are chosen. The mode for chroma elements is coded using an unsigned exp-golomb scheme. Figure 5.5 illustrates the design choices utilized in this work.

5.3.2 Inter Prediction

The inter prediction motion estimation computational kernel can be broken into three fundamental sections.

- The ME_CNTRL block orchestrates which reconstructed pixels are pulled from memory based on the current macroblock partition. All iterations listed in table 5.3 are executed in one inter prediction MB partition cycle; total iterations are doubled in the scenario that a sub-macroblock partition is selected. There are a range of SAE calculation iterations in this work's motion estimation algorithm (see section 2.6). As a result, the following table lists the

range of possible iterations the algorithm may execute in a single data cycle.

Partition	**SAE Calculations**	**SSD Calculations**	**Total Iterations**
Full	5 - 13	1	6 - 14
Vertical	10 - 26	2	12 - 28
Horizontal	10 -26	2	12 - 28
Quad	20 - 52	4	24 - 56
			126

Table 5.3: KiloCore COST_ENGINE iterations

- The COST_ENGINE block finds the motion estimation for each iteration in the ME_CNTRL block. Error is calculated one row at a time and accumulated until the full height of the partition under test is reached. The block also passes four SSD error values (one for each partition mode) to the next block.

- Finally, the MODE_COST block finds the minimum mode mode error for each partition. If the selected macroblock partition is quad mode, all three bullets are repeated for the sub-macroblock partitions.

Figure 5.6 illustrates the top-level block diagram for the inter prediction motion estimation computational kernel. The 30x30 pixel search region and 16x16 input MB are stored in processor DMEM for faster fetch times. Each key algorithmic block is highlighted in figure 5.6 and discussed further in the following sections. The output of the entire motion estimation block is a 16x16 pixel residual and prediction block. Additionally, an exp-golomb coded prediction mode and motion vector are passed as outputs to the bitstream prediction header.

5.3.3 Motion Estimation Control

The ME_CNTRL algorithm is limited to a single core. This core orchestrates what reconstructed and input pixels are sent to the cost engine module. Once the smallest SAE for a given partition is chosen, the ME_CNTRL core sends a flag to calculate the SSD for the mode. Algorithm 3 outlines the logic for this core.

5.3.4 Motion Estimation Cost Engine

The COST_ENGINE block performs all SAE and SSD calculations for inter prediction.

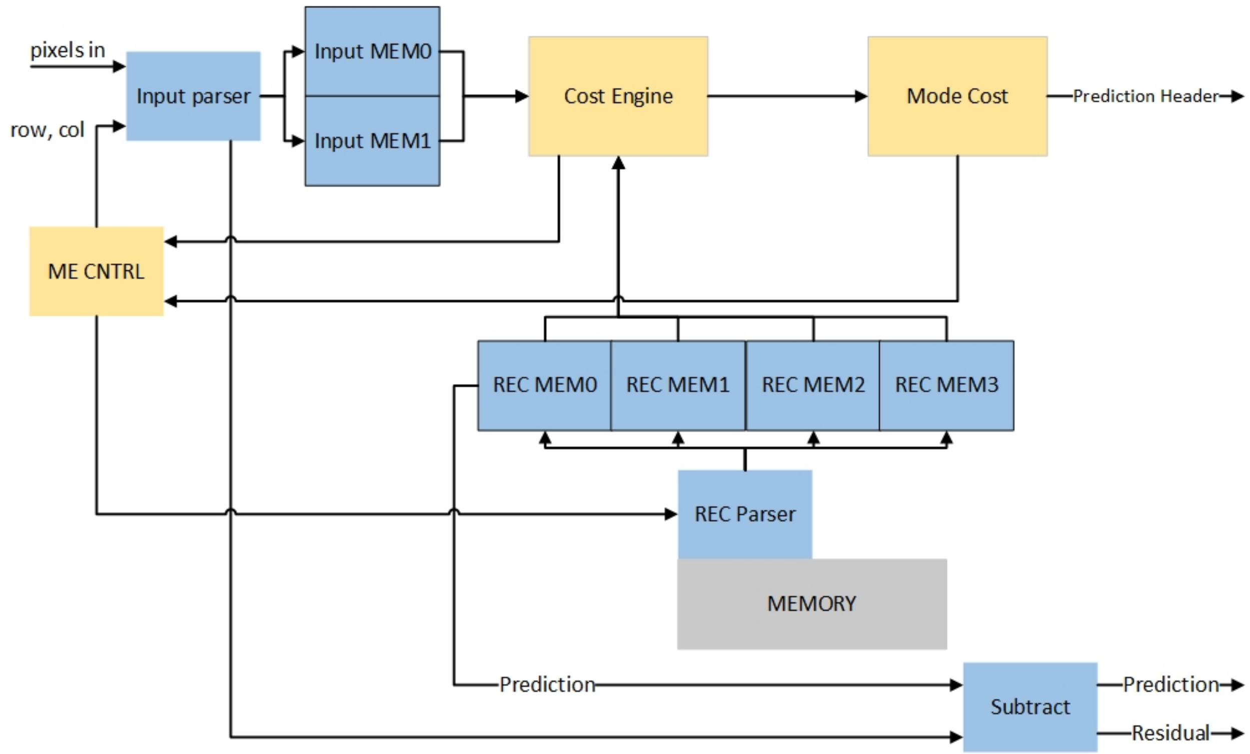

Figure 5.6: KiloCore II Inter prediction top-level block diagram

Algorithm 3 Motion Estimation Control Algorithm

Load DMEM core pixels
$stride = 4$
repeat
  c = Find center SAE
  t = Find top SAE
  r = Find right SAE
  b = Find bottom SAE
  l = Find left SAE
  $idx = min(c, t, r, b, l)$
  $stride = stride >> 1$
until $(idx == c)|(stride == 0)$
ssd = Find SSD(idx)

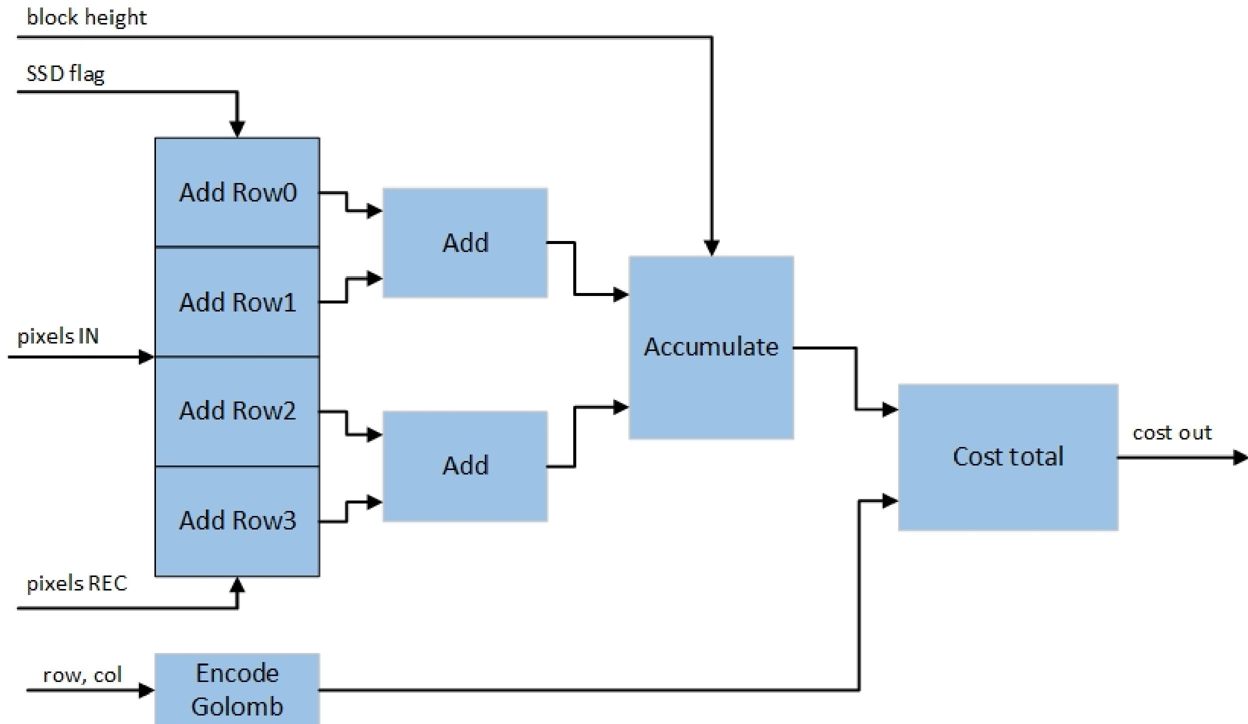

Figure 5.7: KiloCore II cost engine

One cycle of this module includes between 5 and 13 block SAE calculations depending on the outcome of Algorithm 3. This block will always end with a SSD calculation for mode prediction. This module will iterate 9 times and 18 times if a full or sub-macroblock partition is chosen, respectively.

A full reconstructed and pixel block are loaded into the SAE calculation block. SAE is calculated row-by-row and accumulated until the height of the partition is reached. Figure 5.7 illustrates the total algorithm.

5.3.5 Mode Cost Engine

The MODE_COST block accumulates all four SSD error values for each partition mode. The minimum mode cost is calculated and the output is a set of motion vectors for the selected mode and a sub-macroblock partition flag to indicate whether the motion estimation algorithm must be iterated. Figure 5.8 depicts the hardware architecture for this design.

5.3.6 Critical Path Optimization

Due to the computational intensity of the inter predictive algorithm. In this work, the

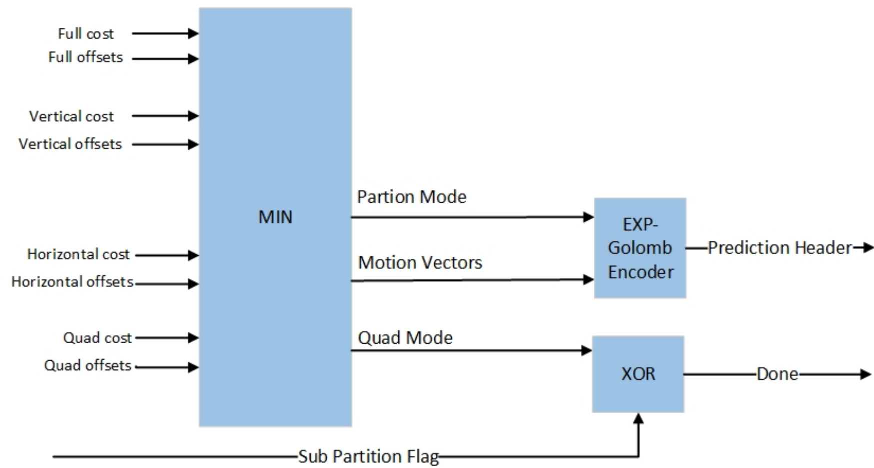

Figure 5.8: KiloCore II mode cost engine

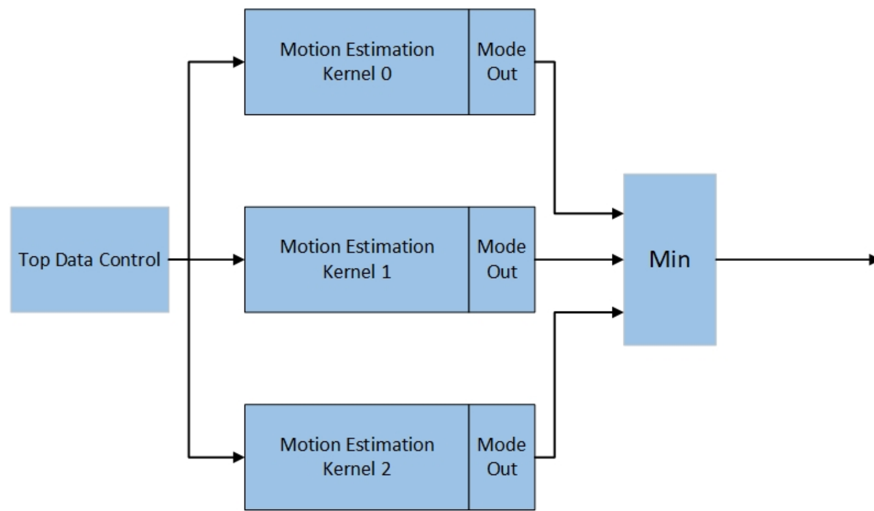

Figure 5.9: Motion estimation critical path optimization

entire motion estimation kernel is instantiated three times to improve the critical path delay. Whereas for one computational kernel 9 iterations are required, the optimized algorithm requires three, as depicted in figure 5.9. While the entire motion estimation algorithm could be parallelized further, three datapaths is a good tradeoff between throughput and area.

5.3.7 Forward Transform

In H.264/AVC forward transform and quantization is a culmination of additions, shifts, and multiplications. On KiloCore this is implemented by executing the four core transform rows on separate processors. Every element of the scale factor calculation, which requires multiplication, is

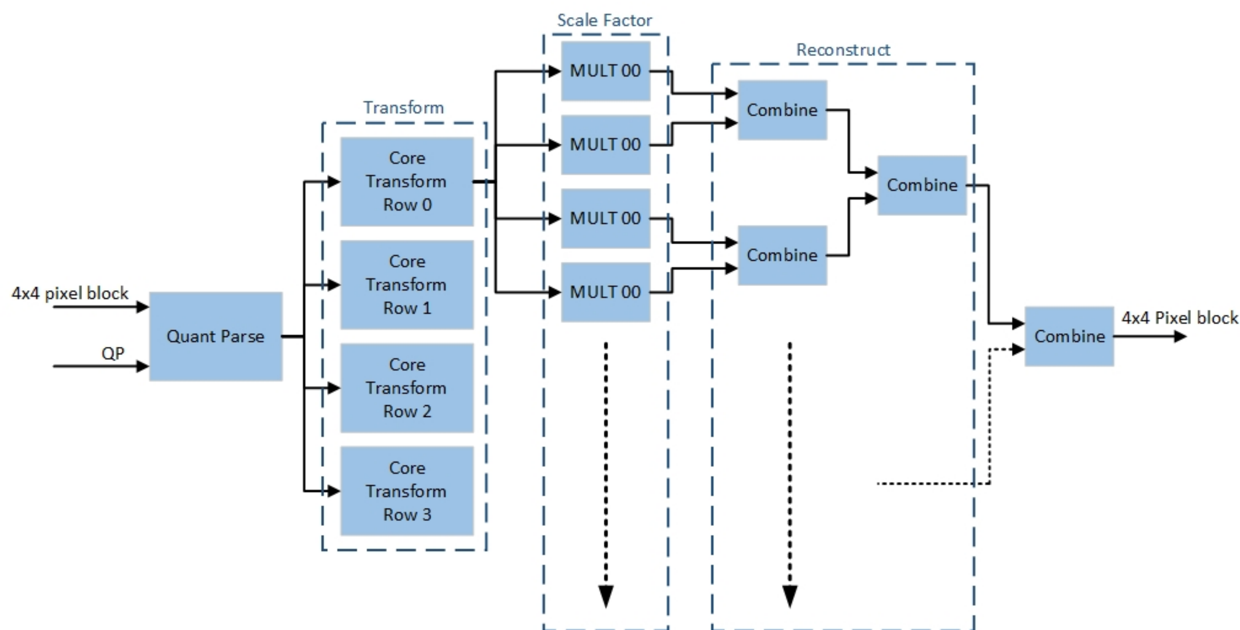

Figure 5.10: KiloCore II forward transform

calculated in parallel (16 total). Overall, this design prioritizes throughput with a highly parallel datapath. Figure 5.10 illustrates the forward transform algorithm.

5.3.8 CAVLC

The CAVLC entropy encoder is depicted in figure 5.11. A single core (CAVLC_in) processes the total coefficients, trailing ones, levels, total zeros, and run of zeros. This processor iterates through the entire 4x4 pixel block. All downstream cores, except for the levels core, generate their respective VLCs through a series of look-up tables. The bitstream is built-up through a series of routing cores as depicted in figure 5.11.

5.3.9 Reconstruction

Reconstruction is an important step for generating valid pixels for future predictions. Figure 5.12 illustrates the reconstruction process of KiloCore II. As depicted, the cores are programmed to handle data successively - a key feature of the KiloCore platform. For any given macroblock, memory is fetched, reconstructed, and stored back in memory in that order. Complex control algorithms are not required.

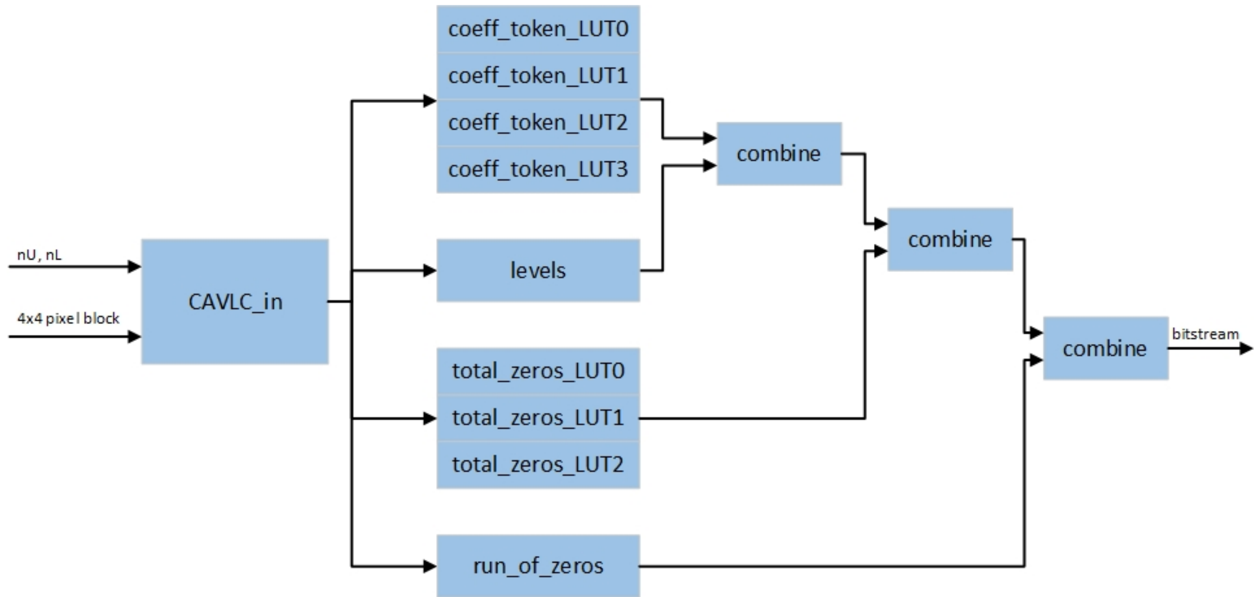

Figure 5.11: KiloCore II CAVLC

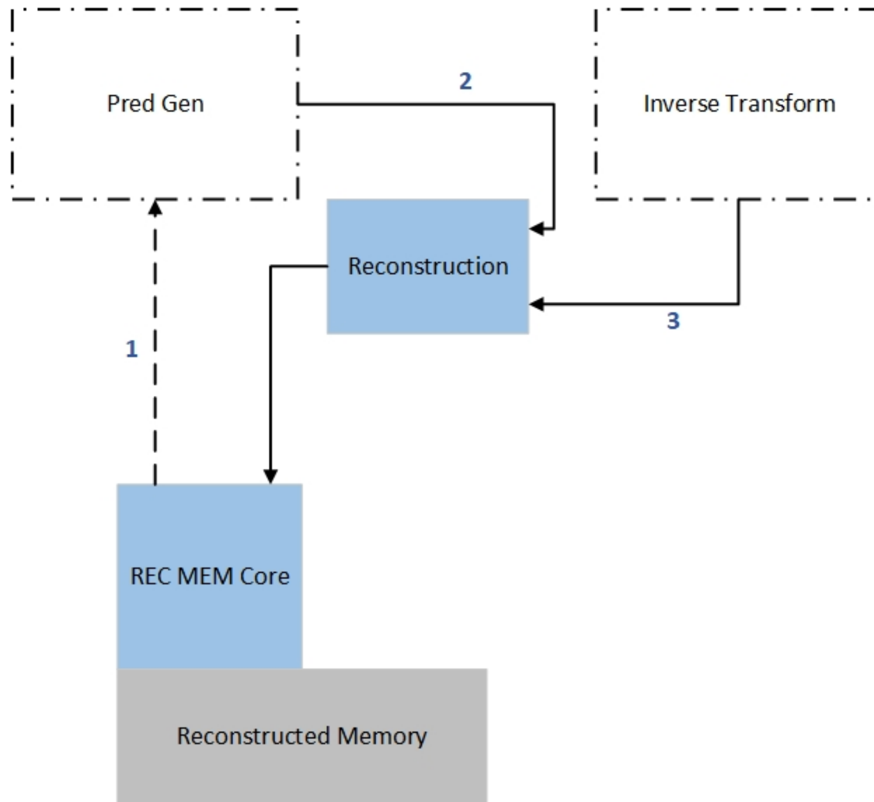

Figure 5.12: KiloCore II reconstruction

5.4 Decoder

The following sections discuss the design of the KiloCore II H.264 decoder. Unlike the encoder, global inputs are not passed as inputs to the decoder because these are decoded in the bitstream header. The sole input to this design is the bitstream from the encoder terminated by an end of file (EOF) flag of -1. The EOF flag allows the decoder to parse the end macroblocks from H.264 bitstream without the possibility of hang-ups due to empty FIFOs.

Algorithm 4 EOF Handler

```
if EOF == 0 then
    in = bitstream(bp)
else
    in = 0
end if
if in = -1 then
    EOF = 1
    in = 0
end if
```

5.4.1 Bitstream Handling

A scheme is designed to handle the collection of variable length codes that make up the H.264/AVC bitstream. Bits from the compressed bitstream are generated into a series of 32-bit packets and passed to and from the inverse CAVLC block. The inverse CAVLC block may send a flag to the packet generation core to pass the current bit from the bitstream. Additionally, the inverse CAVLC block may pass back the 32-bit packet to find the prediction header and update the packet for the next pixel block. Figure 5.13 depicts the bitstream handler scheme.

5.4.2 Prediction Decoding

Decoding the prediction header is a relatively simple task on KiloCore II. An initial header decoding core decodes either a prediction mode value or a series of motion vectors. In either case, values are passed to a router core along with the current prediction mode. A reconstructed memory core will subsequently pass previously reconstructed pixels for intra prediction or motion compensation.

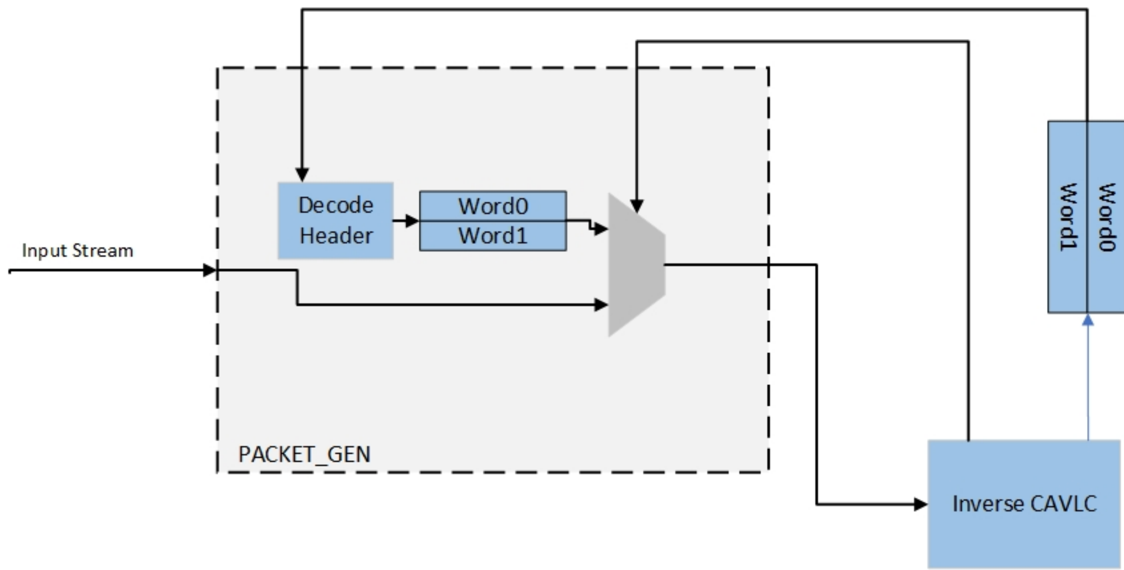

Figure 5.13: KiloCore II bitstream handler

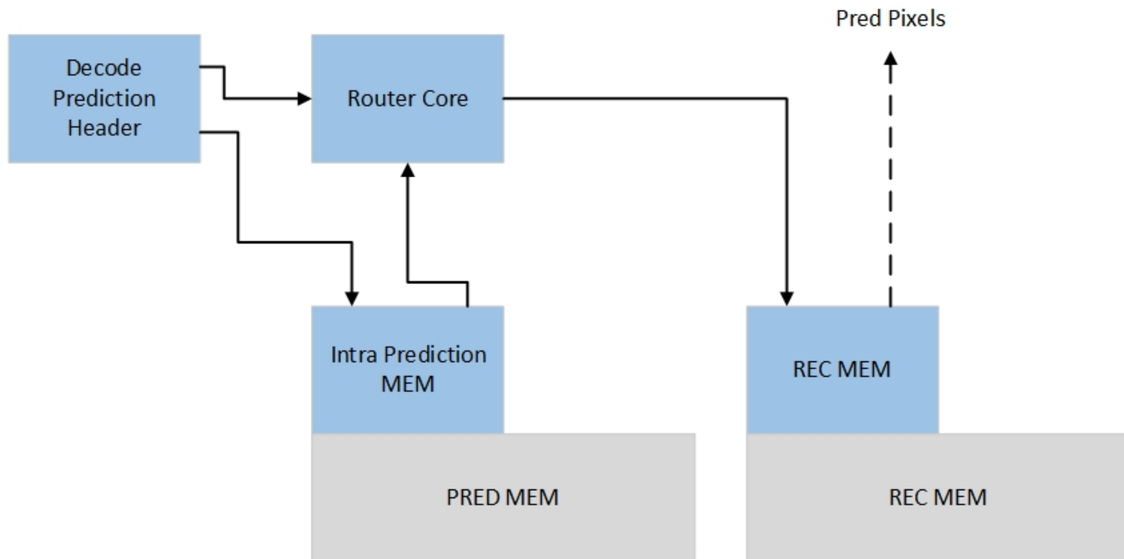

Figure 5.14: KiloCore II prediction decoding

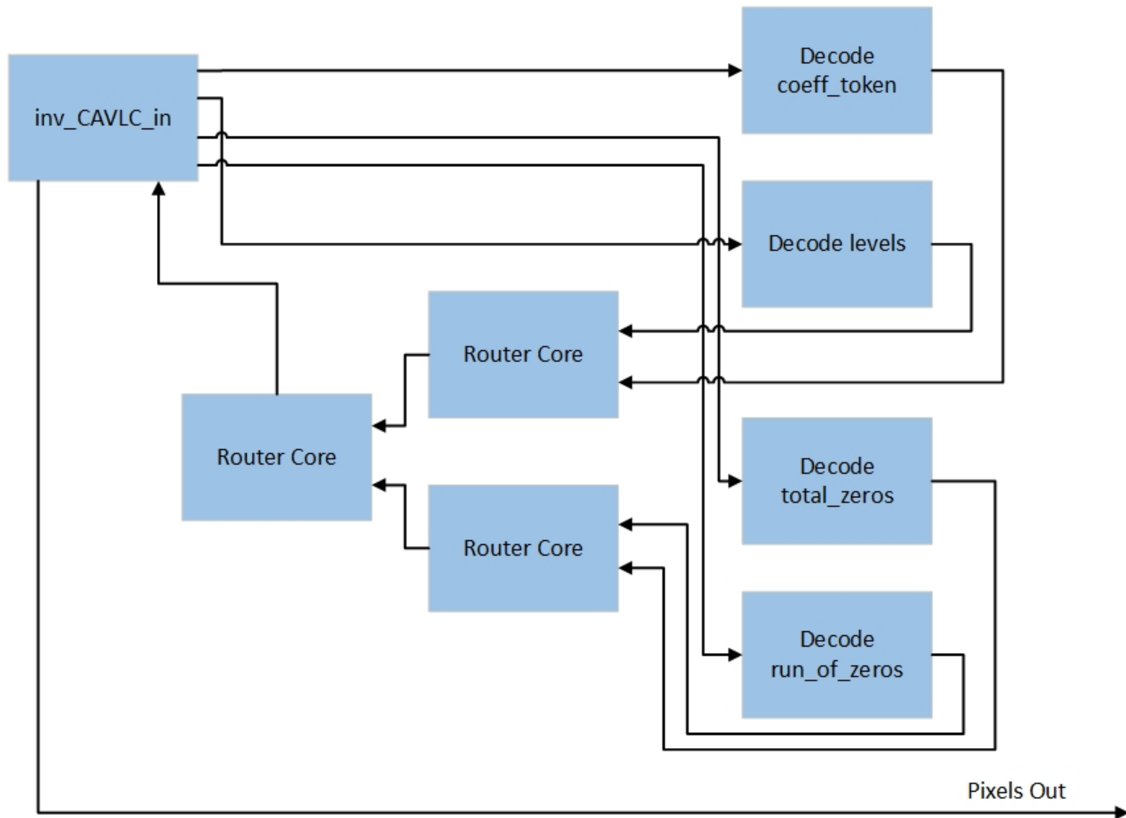

Figure 5.15: KiloCore II inverse CAVLC

5.4.3 Inverse CAVLC

The block diagram for inverse CAVLC is depicted in 5.15. A central processing core, inv_CAVLC_in, offloads decoding tasks for all VLCs in CAVLC. Data is returned back to the inv_CAVLC_in core through a series of routing cores where a 4x4 pixel block is eventually reconstructed.

5.4.4 Inverse Transform

As with the forward transform module on KiloCore II, the inverse transform block is optimized to utilize a high level of parallelization. The scale factor contains 16 independent multiplication cores which are recombined and eventually matrix multiplied. Figure 5.16 presents the inverse transform block diagram for KiloCore II.

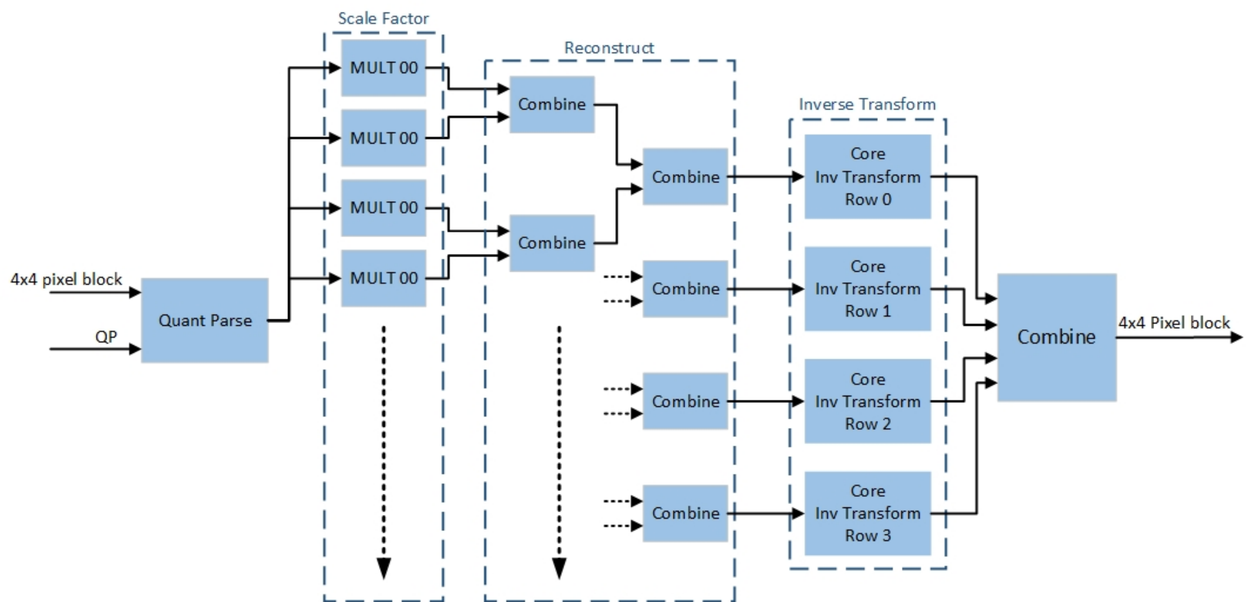

Figure 5.16: KiloCore II inverse transform

Chapter 6

Verification Methods

The proposed KiloCore and RTL implementation are tested and verified using a MATLAB golden reference model that conforms to the proposed H.264/AVC bitstream syntax. The golden reference software used in this design is verified by parsing sections of the H.264 reference software - designed by the standard creators. Additionally, characteristic traits of the proposed codec are observed to conform to characteristics of any generic video compression algorithm. Such traits include the trade-off between compression performance and peak-signal-to-noise ratio, visual inspection of the video frames, and a complete comparison of the reference frames generated independently by the encoder and decoder. In the final section of this chapter, the scope of verification between the software golden reference model and hardware models are discussed.

6.1 JM Software

The JM software is the H.264/AVC reference software developed by the joint team of ISO/IEC MPEG and ITU-T VCEG [24]. In this work, the JM software is used as a tool to verify the functional blocks of the proposed H.264 codec.

The JM software accepts a ".yuv" file for encoding and a ".264" file containing a bitstream for decoding. A trace function is generated which highlights relevant data regarding the generated bitstream. Header parameters and coded macroblock VLCs are all labeled descriptively in the output of the trace function.

Figure 6.1 presents an excerpt from the JM software trace function. In blue, the prediction mode header is highlighted which contains all information to reconstruct a prediction for the luma

and chroma samples. The green highlighted section presents the coded block pattern (CBP), indicating which quadrants of a macroblock have non-zero coefficients. Next, the red section exhibits the quantization parameter value used for this macroblock. Finally, the yellow field describes the entire residual bit stream. This section clearly illustrates each of the five CAVLC VLCs, which are reconstructed into a 4x4 pixel sample.

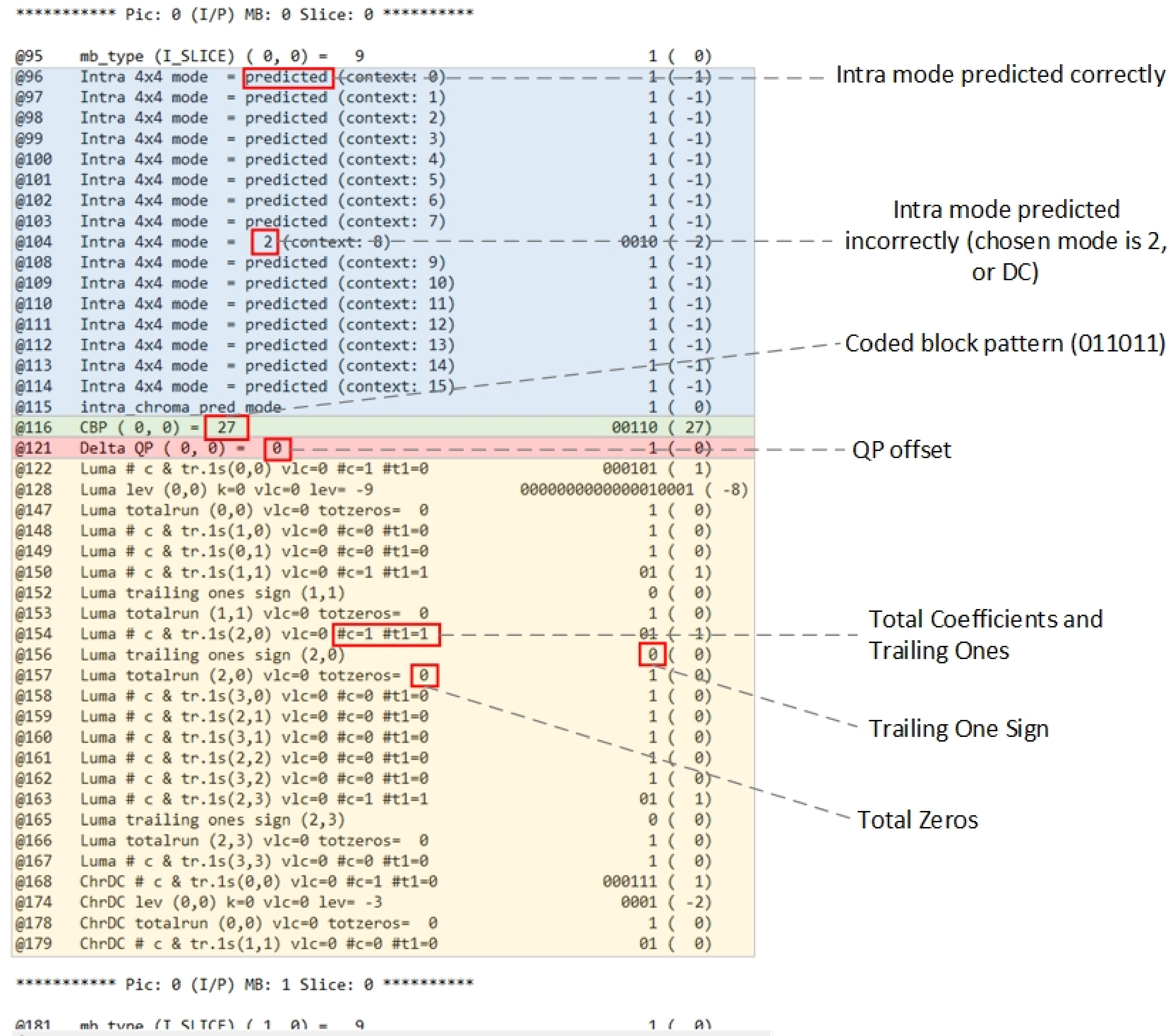

Figure 6.1: JM trace function output

The codec designed in this work is compliant with JM at a macroblock level. While the proposed codec cannot directly interface with the JM software, special care is taken to mimic the syntax of the macroblock layer of the JM reference software. The following sections discuss how the proposed codec was designed for accuracy.

6.2 Matlab Golden Reference Verification

The following two sections describe two methods for testing the general functionality of the proposed codec golden reference model. The first section discusses definitive outputs of the JM software trace function that were verified to bit accuracy. The following section describes key characteristics of an H.264/AVC codec that were verified by observation.

6.2.1 JM Verification

The proposed H.264 codec model is functionally verified and bit accurate to all the features listed in table 6.1. Each element is individually tested using the JM software trace function output for an entire intra prediction frame for two sample QCIF video sequences, "foreman.qcif" and "suzie.qcif". The pure functionality of the motion estimation algorithm is not tested as the standard does not specify motion estimation search algorithms. Additionally, the functionality of exp-Golomb encoding is not explored as this is a trivial block with a structured and uniform output.

Functional Block
Intra Prediction
Forward Transform
CAVLC
Inverse CAVLC
Inverse Transform

Table 6.1: JM reference software verified algorithmic block

The JM software trace file output was parsed with a python script using the textparser library [25]. Pertinent information regarding the proposed CODEC model was extracted from this trace file. Each section below describes how that information was utilized in order to verify accuracy. Forward and inversely related blocks were verified in tandem using the output from the JM software encoder.

Intra Prediction

Intra prediction mode selection is verified by parsing the "Intra 4x4 mode" descriptor from the JM trace function. Each 4x4 pixel block is verified with 100% accuracy to yield the appropriate Intra prediction mode and corresponding variable length code.

As illustrated in figure 6.1, intra prediction modes are described either as “predicted” or through a number representing one of the nine possible modes supported in the standard. If this field is labeled as predicted, the chosen prediction mode is the minimum mode number between the neighboring top and left 4x4 pixel predictions. If the selected intra prediction mode is not the minimum of the neighboring modes, the selected prediction mode is explicitly stated in the trace function output.

Intra prediction verification was achieved by recreating a prediction mode memory from the JM trace function and cross referencing it with the memory created in the golden reference model. An overview of how an intra prediction memory is parsed from the JM software is described in algorithm 5.

Algorithm 5 Decode Intra Prediction Mode

```
if (KEY == "Intra 4x4 mode") then
  predU = predMEM(row,col-1)
  predL = predMEM(row-1,col)
  n = min(predU, predL)
  if (code == "predicted") then
    predMEM(row,col) = n
  else
    predMEM(row,col) = code
  end if
end if
```

Forward Transform and Inverse Transform

A residual data set is used to verify the transform and inverse transform algorithm blocks. The residual data set is generated by subtracting the raw input video sequence with the previously verified prediction pixels. The output of the golden reference transform is cross-verified with the generated transform output of the JM software (the input into the CAVLC encoder). Due to varying rounding schemes, the proposed H.264 CODEC did vary from the forward and inverse transform outputs slightly. Regardless, transform outputs were verified with high accuracy by accounting for rounding errors with a tolerance of two integer values. Figure 6.2 illustrates how the transform block was functionally verified.

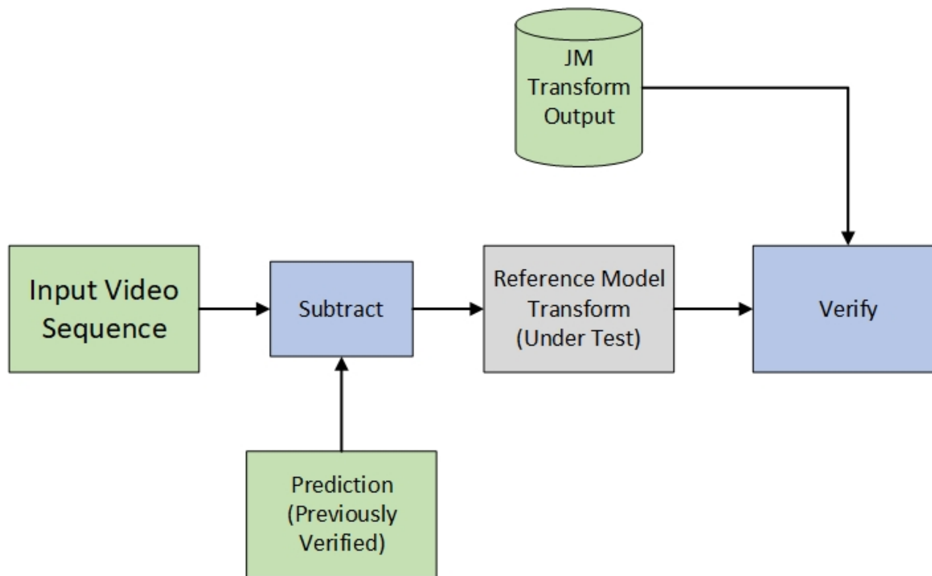

Figure 6.2: Transform golden reference verification (gray) using verified parameters (green)

```
@147   Delta QP ( 0, 0) =    0                                   1 (  0)
@148   Luma # c & tr.1s(0,0) vlc=0 #c=11 #t1=1     000000000001110 ( 11)
@163   Luma trailing ones sign (0,0)                             0 (  0)
@164   Luma lev (0,0) k=9 vlc=1 lev= -2                         11 ( -1)
@166   Luma lev (0,0) k=8 vlc=1 lev= -1                         11 ( -1)
@168   Luma lev (0,0) k=7 vlc=1 lev=  1                         10 (  1)
@170   Luma lev (0,0) k=6 vlc=1 lev= -5                     000011 ( -5)
@176   Luma lev (0,0) k=5 vlc=2 lev=-11                   00000101 (-11)
@184   Luma lev (0,0) k=4 vlc=3 lev= -3                       1101 ( -3)
@188   Luma lev (0,0) k=3 vlc=3 lev=  3                       1100 (  3)
@192   Luma lev (0,0) k=2 vlc=3 lev=  3                       1100 (  3)
@196   Luma lev (0,0) k=1 vlc=3 lev=-12                     001111 (-12)
@202   Luma lev (0,0) k=0 vlc=3 lev=  9                     001000 (  9)
@208   Luma totalrun (0,0) vlc=10 totzeros=  3                 010 (  3)
@211   Luma run (0,0) k=10 vlc=2 run= 1                         10 (  1)
@213   Luma run (0,0) k=9 vlc=1 run= 2                          00 (  2)
@215   Luma # c & tr.1s(1,0) vlc=3 #c=8 #t1=1               011101 (  8)
@221   Luma trailing ones sign (1,0)                             1 (  1)
@222   Luma lev (1,0) k=6 vlc=0 lev= -2                         01 ( -1)
@224   Luma lev (1,0) k=5 vlc=1 lev=  1                         10 (  1)
@226   Luma lev (1,0) k=4 vlc=1 lev=  2                        010 (  2)
@229   Luma lev (1,0) k=3 vlc=1 lev= -1                         11 ( -1)
@231   Luma lev (1,0) k=2 vlc=1 lev=  2                        010 (  2)
@234   Luma lev (1,0) k=1 vlc=1 lev=  2                        010 (  2)
@237   Luma lev (1,0) k=0 vlc=1 lev=  3                       0010 (  3)
@241   Luma totalrun (1,0) vlc=7 totzeros=  6                  010 (  6)
@244   Luma run (1,0) k=7 vlc=5 run= 4                         010 (  4)
@247   Luma run (1,0) k=6 vlc=1 run= 0                           1 (  0)
@248   Luma run (1,0) k=5 vlc=1 run= 0                           1 (  0)
@249   Luma run (1,0) k=4 vlc=1 run= 0                           1 (  0)
@250   Luma run (1,0) k=3 vlc=1 run= 1                          01 (  1)
@252   Luma run (1,0) k=2 vlc=0 run= 1                           0 (  1)
```

Figure 6.3: Sample JM CAVLC 4x4 pixel encoding

CAVLC and Inverse CAVLC

The CALC and inverse CAVLC blocks were verified to have a bit accurate output when compared to the JM reference software. This was confirmed by simply comparing the generated bitstream from the JM trace function to the generated bitstream output of the proposed codec golden reference model. To illustrate the process, the luma block highlighted in figure 6.3 is rebuilt programmatically. In practice the reference data set is generated utilizing a python script. First, an array is generated that contains the ordered level and trailing ones data.

9	-12	3	3	-3	-11	-5	1	-1	-2	1

Table 6.2: JM reference total coefficients and T1s

Next, zero run data is parsed to insert the relevant string of zeros as depicted in table 6.3.

9	-12	3	3	-3	-11	-5	1	-1	0	0	-2	0	1

Table 6.3: JM reference total zeros and run of zeros

Finally, the scanned 1-D array is reverted to its 4x4 pixel form.

9	-12	-11	-5
3	-3	1	0
3	-1	-2	1
0	0	0	0

Table 6.4: JM reference reconstructed 4x4 pixel block

6.2.2 Characteristic Verification

Along with the functional verification from the JM reference software, a few characteristic traits for a general video codec are verified in the proposed golden reference model.

For any functioning codec, it is expected to observe a clear tade-off between the compression performance and peak-signal-to-noise ratio (PSNR). To see this behavior, the quantization parameter for the proposed H.264/AVC golden reference software is varied and the compression ratio and PSNR are monitored. Figure 6.4 and figure 6.5 display the compression ratio and PSNR, respectively, for the same video sequence frame as the quantization parameter increases.

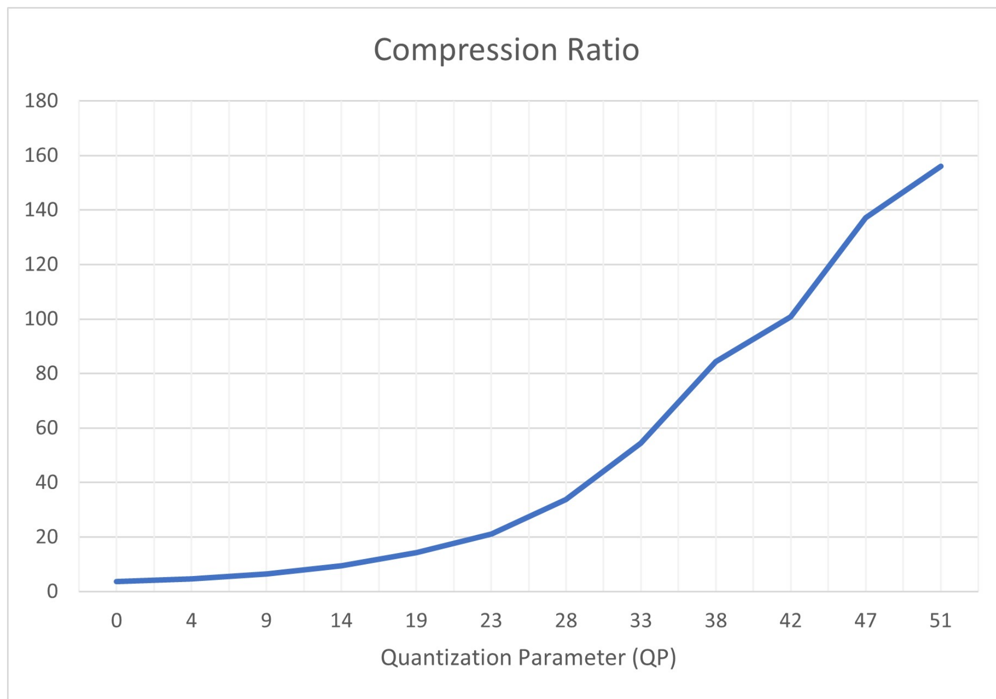

Figure 6.4: H.264 golden reference compression ratio

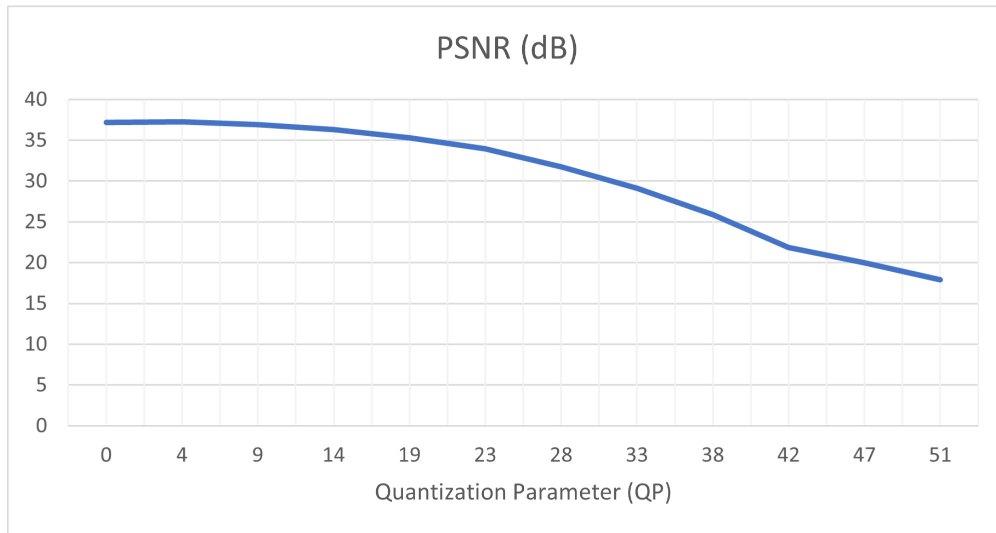

Figure 6.5: H.264 golden reference PSNR

Along with the calculated metrics above, the functionality of the video compression algorithm is also confirmed by inspection. For both the encoder and decoder, side-by-side images for the compressed and uncompressed video frames are displayed in the MATLAB golden reference model. By observation it is easy to note any disconformity beyond expected compression artifacts.

Figure 6.6 displays an original and compressed video frame of “suzie.qcif” using the proposed golden reference model.

Figure 6.6: Golden reference frame comparison for frame 1 of “suzie.qcif” (QP of 15) with original frame (left) and reconstructed frame (right)

6.3 RTL and KiloCore Verification

After a golden reference software was established, both the RTL and KiloCore models were designed and verified. Functionality for the two hardware models were simply verified using a combination of python scripts and a compare plugin on the Notepad++ editor [26]. A complete list of the all the bit accurate test cases verified for the encoder and decoder are listed in table 6.5 and table 6.6, respectively. Test cases are varied across quantization parameters to ensure that all six scaling factors are utilized and tested.

Both datasets utilized three video sequences: “suzie_qcif.yuv”, “claire_qcif.yuv”, and “foreman_qcif.yuv”. All three video sequences are commonly used in evaluating video compression performance and are downloaded from the video trace library [27].

Video Sequence	**Frame Number**	**Prediction Mode**	**QP (mod(QP,6))**	**PSNR**	**Compression Ratio**	**Verified**
suzie_qcif	0	Intra	15 (3)	36.02	10.32	**yes**
suzie_qcif	1	Intra	28 (4)	31.35	33.67	**yes**
suzie_qcif	2	Intra	50 (2)	18.23	151.65	**yes**
claire_qcif	0	Intra	24 (0)	29.15	29.75	**yes**
claire_qcif	1	Intra	19 (1)	28.93	20.54	**yes**
claire_qcif	2	Intra	34 (4)	25.07	64.40	**yes**
foreman_qcif	0	Intra	11 (5)	33.61	6.93	**yes**
foreman_qcif	1	Intra	12 (0)	26.36	7.38	**yes**
foreman_qcif	2	Intra	13 (1)	22.64	7.74	**yes**
suzie_qcif	1	Inter	15 (3)	39.03	25.39	**yes**
suzie_qcif	2	Inter	26 (2)	34.42	66.91	**yes**
claire_qcif	1	Inter	24 (0)	34.63	81.67	**yes**
claire_qcif	2	Inter	25 (1)	35.43	83.37	**yes**
foreman_qcif	1	Inter	28 (4)	32.10	52.53	**yes**
foreman_qcif	2	Inter	35 (5)	27.70	148.68	**yes**

Table 6.5: Encoder RTL and KiloCore test dataset

Video Sequence	**Frame Number**	**Prediction Mode**	**QP (mod(QP,6))**	**Verified**
suzie_qcif	0	Intra	15 (3)	**yes**
suzie_qcif	1	Inter	28 (4)	**yes**
suzie_qcif	2	Inter	50 (2)	**yes**
claire_qcif	0	Intra	24 (0)	**yes**
claire_qcif	1	Inter	19 (1)	**yes**
claire_qcif	2	Inter	35 (6)	**yes**
foreman_qcif	0	Intra	11 (5)	**yes**
foreman_qcif	1	Inter	12 (0)	**yes**
foreman_qcif	2	Inter	13 (1)	**yes**

Table 6.6: Decoder RTL and KiloCore test dataset (PSNR and compression performance labeled in figure 6.5)

Chapter 7

Results and Comparisons

The proposed H.264/AVC encoder and decoder is simulated on the KiloCore II chip. All results discussed in the following section are provided through simulation results from the Project Manager software. Area results are interpolated from the fact that each processor has a total area of 0.055 mm^2 [3].

The following sections analyze how core processors are distributed within the compression algorithm. Additionally, energy and throughput results are discussed followed by a comparison of other CODECs from related work.

The proposed H.264/AVC CODEC in this work was tested utilizing QCIF video sequences. Larger video formats are provided by interpolating the achieved macroblock per second (mbps) throughput. For both the encoder and decoder of this work, throughput and energy analytics are derived from an even weighted average between all test conditions outlined in Table 6.5 and Table 6.6. This ensures that results are well compensated for variance in video sequences and quantization parameters.

7.1 Core Utilization

7.1.1 Encoder

The encoder utilizes a total of 333 cores on the KiloCore II platform. With each core consuming an area of 0.055 mm^2, the total area of the encoder is interpolated as 18.315 mm^2. Table 7.1 outlines how cores are distributed across all algorithmic blocks. The cores used for the

prediction mode data are separated by luma (y components) and chroma (both u and v components) datapaths. The forward transform, inverse transform, CAVLC, and control fields account for all the cores utilized to encode the three color space components.

As depicted in figure 7.1, the luma inter prediction algorithm dominates the area consumption. Inter prediction is the most computationally heavy task in the H.264/AVC standard. Consequently, a high number of parallel cores are utilized in this work to speed up this block's critical path. Additionally, the inter prediction algorithm designates the most cores as local memory (utilizing their DRAM) when compared to other algorithmic blocks. In total, 18 cores alone are used to store pixel data for the motion estimation algorithm in this work.

7.1.2 Decoder

Table 7.2 outlines the total core utilization for the proposed H.264 decoder. This design utilized a total of 131 cores resulting in a total area of 7.205 mm^2. The decoder utilizes a significantly smaller area compared to the encoder primarily due to the lack of an intra and inter predictive algorithm scheme. Additionally, the encoder utilizes both forward and inverse transform blocks while the decoder only requires an inverse transform block.

As seen in figure 7.2, the inverse transform and inverse CAVLC blocks dominate the area consumption for the decoder. The large area consumption required for inverse transform is a result of utilizing a parallel datapath for the luma and chroma reconstructions. In turn, this requires a total of three instantiated inverse transform modules.

While it is possible to parallelize all three color component datapaths in encoding, the bitstream can only be decoded sequentially for the decoder. Consequently, the encoder utilizes three CAVLC modules to optimize throughput while the decoder may only have one inverse CAVLC block. For this reason, the CAVLC algorithm is nearly double the area than its inverse CAVLC counterpart for the decoder.

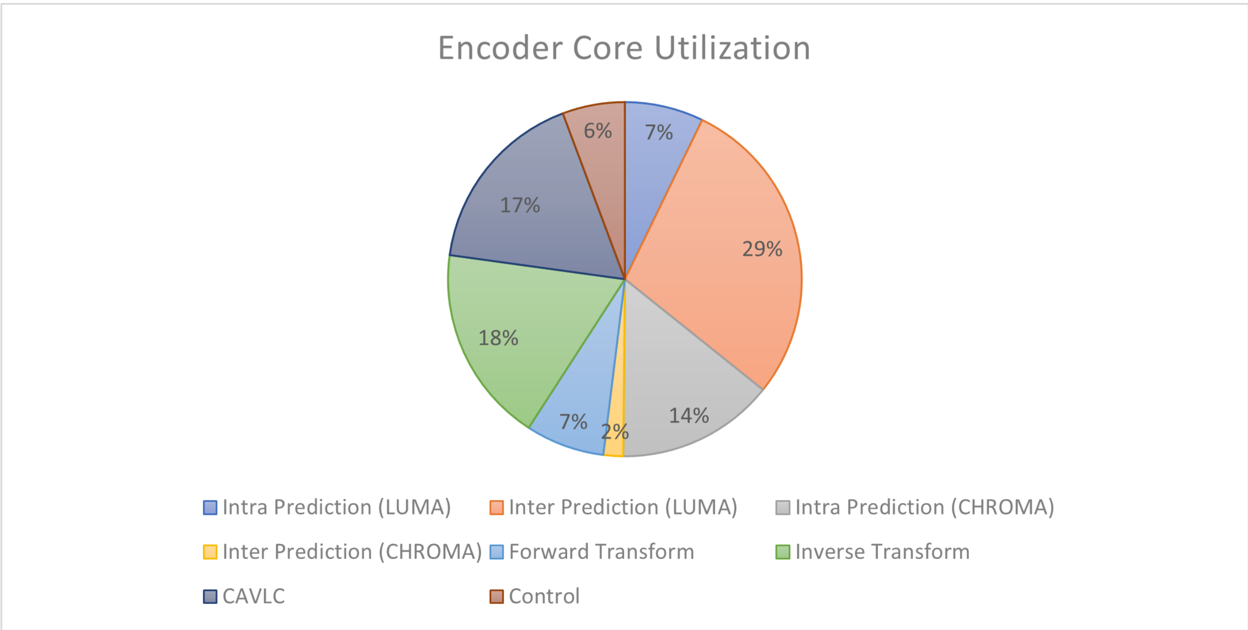

Figure 7.1: KiloCore II encoder core distribution by function

Algorithmic Block	**Number of Cores**	**Area mm²**
Intra Prediction (Luma)	17	0.935
Inter Prediction (Luma)	95	5.225
Intra Prediction (Chroma)	48	2.64
Inter Prediction (Chroma)	6	0.33
Forward Transform	24	1.32
Inverse Transform	60	3.3
CAVLC	57	3.135
Control	19	1.045
Total	333	18.315

Table 7.1: KiloCore encoder core usage

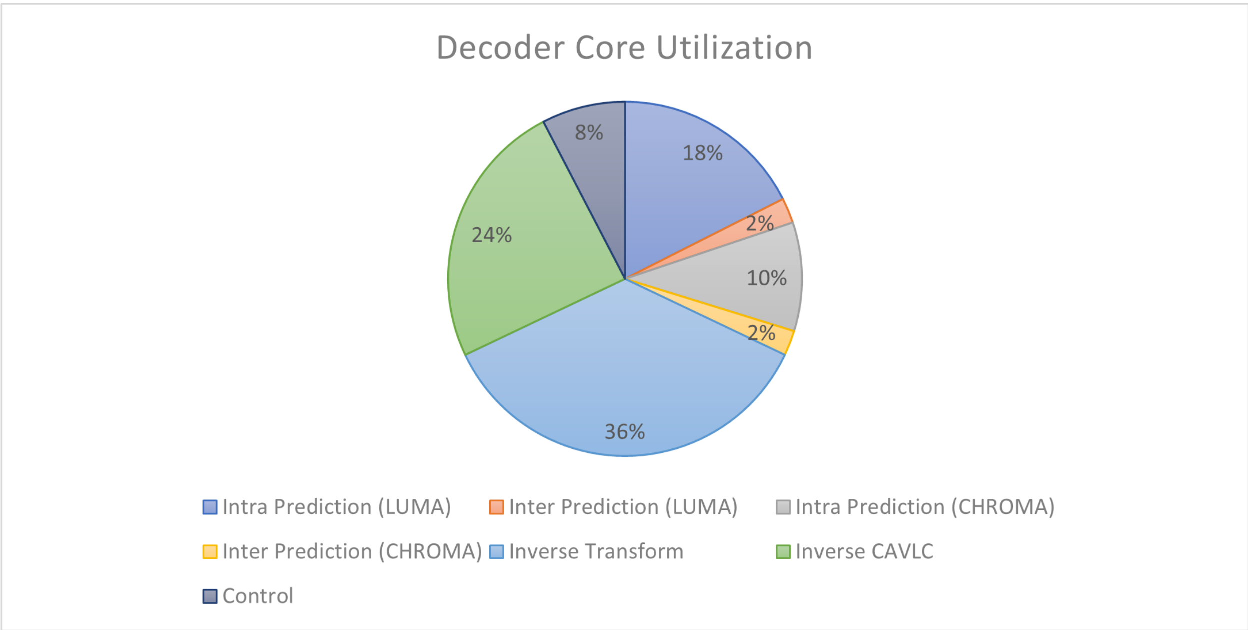

Figure 7.2: KiloCore II decoder core distribution by function

Algorithmic Block	**Number of Cores**	**Area mm^2**
Intra Prediction (Luma)	23	1.265
Inter Prediction (Luma)	3	0.165
Intra Prediction (Chroma)	13	0.715
Inter Prediction (Chroma)	3	0.165
Inverse Transform	47	2.585
Inverse CAVLC	32	1.76
Control	10	0.55
Total	131	7.205

Table 7.2: KiloCore decoder core usage

7.2 Throughput by Stage

7.2.1 Encoder

The following section discusses how throughput varies across all the stages of the encoder. Throughput is measured as the number of macroblocks a stage is able to process in one second (mbps). Table 7.3 outlines each stage's throughput for the encoder.

As depicted in figure 7.3, inter prediction has the lowest throughput by far. This stage requires a significant number of iterative and computationally intensive tasks. Consequently, the critical path of the encoder inter prediction algorithm is dictated by this block's datapath.

The limiting block for the intra prediction mode is more complex than inter prediction. While inter prediction derives its prediction from the previously decoded frame, the intra prediction algorithm derives its prediction from previously decoded pixels of the same frame. Consequently, the next macroblock in intra prediction is not processed until the current macroblock is predicted, transformed, reconstructed, and stored back into memory. As a result, the intra prediction critical path is defined by the prediction, transform, and inverse transform throughputs.

7.2.2 Decoder

Table 7.4 and figure 7.4 depict the throughput by stage for the decoder.

Decoder throughput is severely limited by the inverse CAVLC algorithm. Since H.264/AVC uses a series of variable length codes to optimize compression performance, it is not possible to parallelize bitstream decoding. Consequently, it is only possible to include one inverse CAVLC in hardware. On top of this, the inverse CAVLC algorithm is highly combinational and does not allow room for parallel computation within the algorithm itself.

Besides the inverse CAVLC algorithm, the luma intra prediction and inter prediction blocks provided the next worst throughput performance. These two algorithms require four times more memory reads and writes than the chroma counterparts. For this reason, throughput in this case is dictated by characteristics of the hardware.

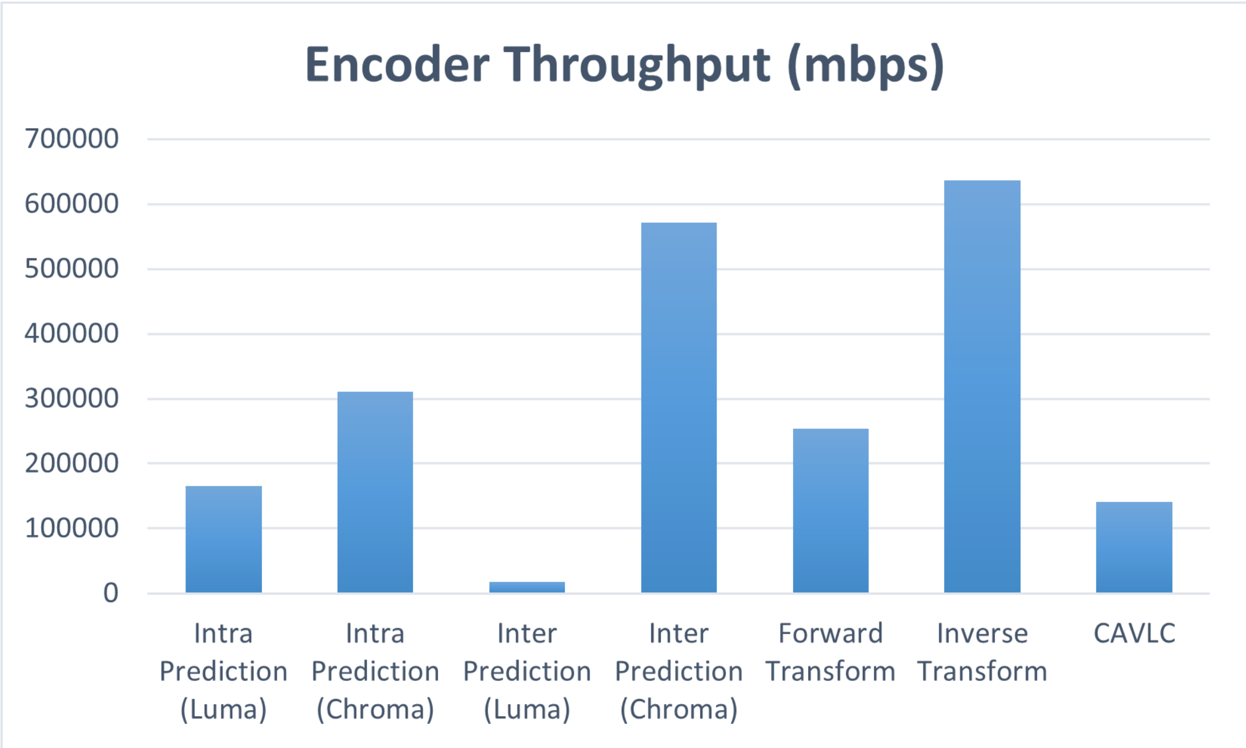

Figure 7.3: Encoder throughput, in macroblocks-per-second, by stage

Stage	Throughput (mbps)
Intra prediction (Luma)	165000
Intra prediction (Chroma)	310896
Inter Prediction (Luma)	17926
Inter Prediction (Chroma)	570685
Forward Transform	253051
Inverse Transform	636168
CAVLC	140284

Table 7.3: Encoder throughput, in macroblocks-per-second, by stage

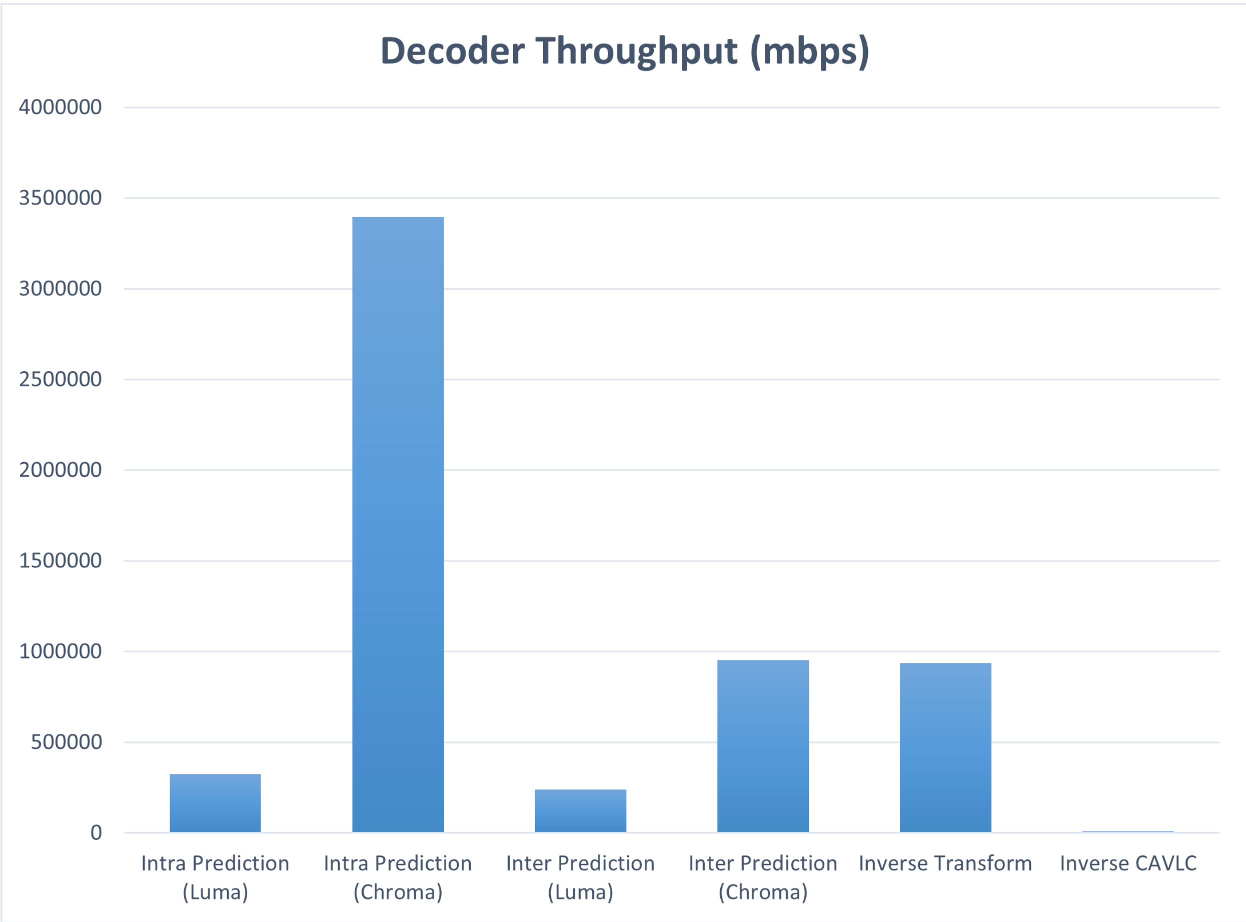

Figure 7.4: Decoder throughput, in macroblocks-per-second, by stage

Stage	**Throughput (mbps)**
Intra prediction (Luma)	324788
Intra prediction (Chroma)	3393898
Inter Prediction (Luma)	239317
Inter Prediction (Chroma)	951998
Inverse Transform	936138
Inverse CAVLC	8684

Table 7.4: Decoder throughput, in macroblocks-per-second, by stage

7.3 Total Energy

Energy consumption results are derived from the cycle-accurate simulator tailored to KiloCore II. Each assembly language operation for a given task has a defined energy consumption [3]. Through a program instruction trace, accurate energy consumption values for the encoder and decoder are derived. The total energy results presented in the following sections are in micro-Joules per QCIF frame.

7.3.1 Encoder

Both figure 7.5 and table 7.5 outline the total energy consumption for the proposed encoder. Total energy consumption has a strong correlation with throughput. Slower and more computationally intensive tasks, such as luma inter prediction, require a large number of operations and consequently a large energy output.

Inter prediction is clearly the dominant algorithmic block in terms of energy consumption for the proposed encoder. Since the choice of motion estimation algorithm is not established in the H.264/AVC standard, metrics such as energy consumption for this module are specific to the application. Regardless, inter prediction still dominates in energy consumption for most high performance compression algorithms.

7.3.2 Decoder

Both figure 7.6 and table 7.7 outline the total energy consumption for the proposed decoder. As with the encoder, the block with the worst throughput performance dominates the total energy consumption.

The inverse CAVLC module consumes the most total energy for the decoder. This algorithmic block requires many loops and branches to decode variable length codes. On KiloCore II an incorrect instruction branch guess produces the largest energy penalty at 41.0 pJ. Additionally, the inverse CAVLC block decodes the luma and both chroma elements without room for parallel optimizations.

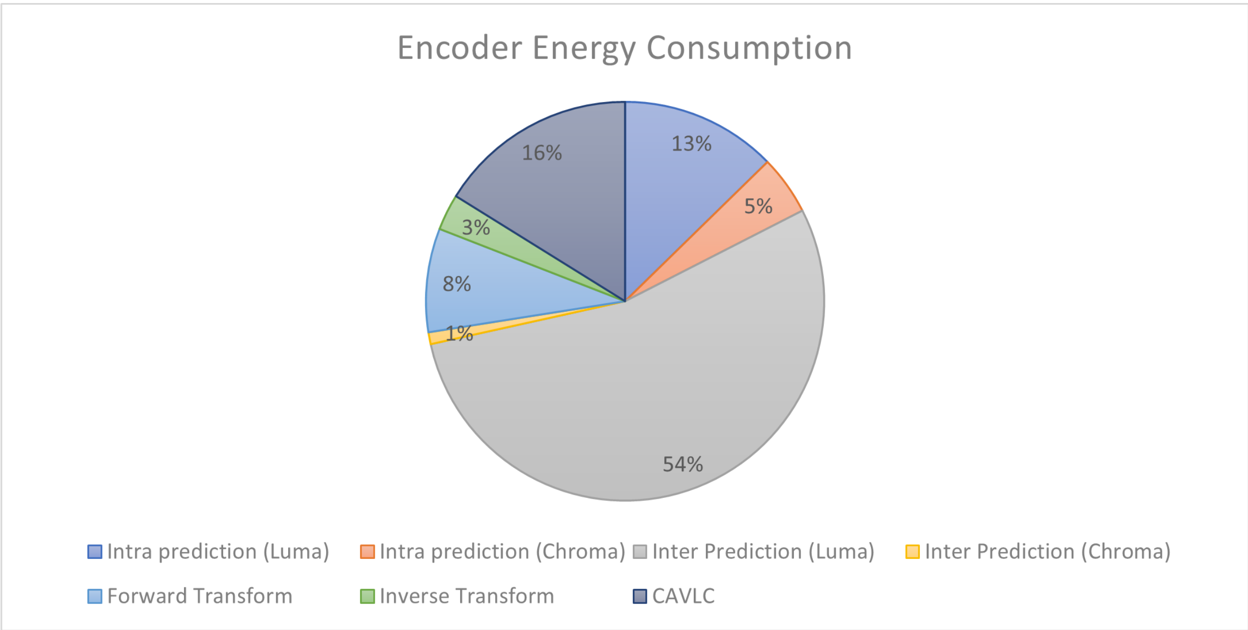

Figure 7.5: Encoder energy distribution by function

Algorithmic Block	**Total Energy (uJ / frame)**	**Percentage (%)**
Intra prediction (Luma)	436.1	12.6
Intra prediction (Chroma)	165	4.8
Inter Prediction (Luma)	1857.8	54
Inter Prediction (Chroma)	33	1.2
Forward Transform	288.8	8.4
Inverse Transform	100.7	2.9
CAVLC	555.4	16.1
Total	3437.1	-

Table 7.5: KiloCore encoder power distribution by task

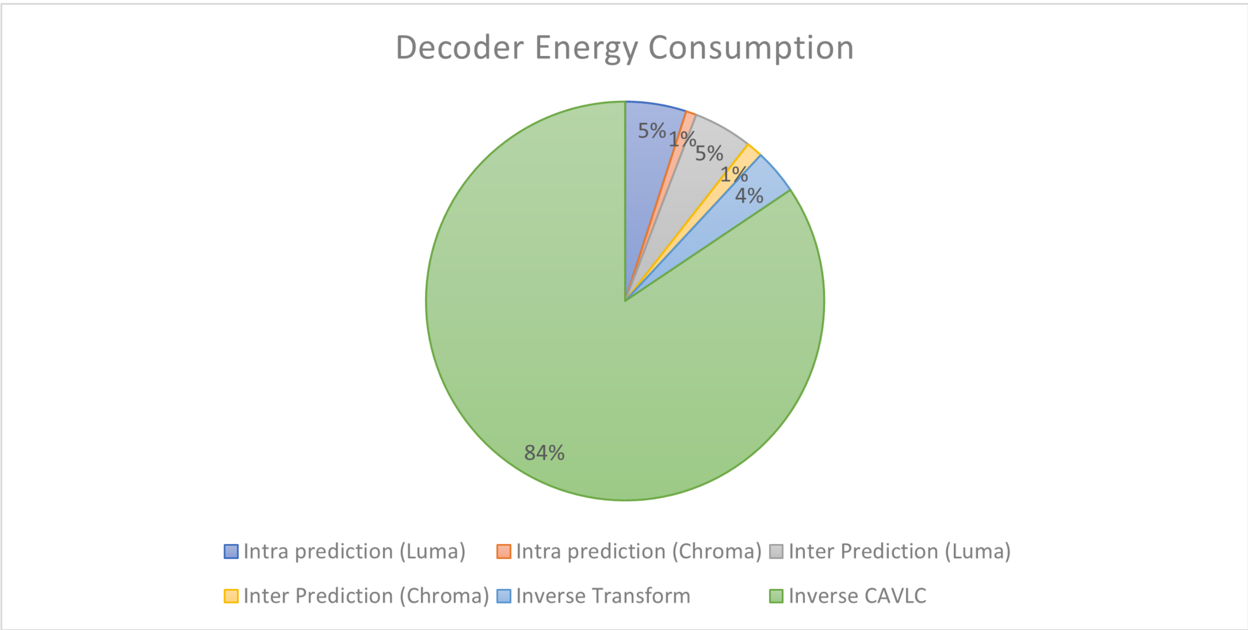

Figure 7.6: Decoder energy distribution by function

Algorithmic Block	**Total Energy (uJ / frame)**	**Percentage (%)**
Intra prediction (Luma)	136.4	5
Intra prediction (Chroma)	23.3	0.8
Inter Prediction (Luma)	131.3	4.8
Inter Prediction (Chroma)	36.4	1.3
Inverse Transform	100.7	3.7
Inverse CAVLC	2317.3	84.4
Total	3437.1	-

Table 7.6: KiloCore decoder power distribution by task

7.4 Scaling with Voltage

A key advantage of the KiloCore II architecture is the ability to scale processor supply voltage. This feature offers extreme versatility when balancing the tradeoffs between throughput and energy consumption for a given application. Table 7.7 lists the results for the proposed codec's throughput, energy, and power performance with respect to varying supply voltages. Results are split between intra and inter prediction frames.

For each defined supply voltage, processor frequency is proportionally scaled. The encoder and decoder algorithms are simulated and the total number of assembly operations are calculated. The measured KiloCore II frequencies and energy scaling constants [28] are used in conjunction with the total operations to calculate throughput, energy, and power across all voltages.

The results presented in figures 7.7 through 7.18 make sense intuitively. As processor supply voltage increases, throughput performance improves along with faster clocking frequencies. The improved throughput performance comes at the expense of higher power consumption.

		Encoder			**Decoder**		
	Voltage (V)	**Throughput (mpbs)**	**Energy (uJ)**	**Power (mW)**	**Throughput (mpbs)**	**Energy (uJ)**	**Power (mW)**
I Frame	0.56	3908.3	70.1	2.7	561.6	201.0	1.1
	0.75	21682.8	119.7	26.2	3116.1	343.0	10.8
	0.9	42040.1	166.9	70.9	6041.8	478.2	29.1
	1.1	60562.3	264.9	162.0	8703.8	758.9	66.7
P Frame	0.56	1156.8	312.1	3.6	861.1	165.2	1.4
	0.75	6417.9	532.7	34.5	4777.3	282.1	13.6
	0.9	12443.5	742.6	93.3	9262.7	393.2	36.7
	1.1	17925.9	1178.5	213.3	13343.7	624.1	84.1

Table 7.7: KiloCore decoder power distribution by task

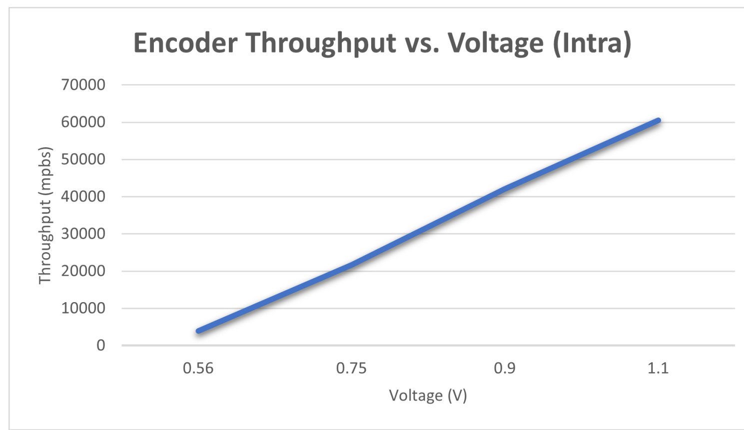

Figure 7.7: Encoder throughput with respect to voltage (intra)

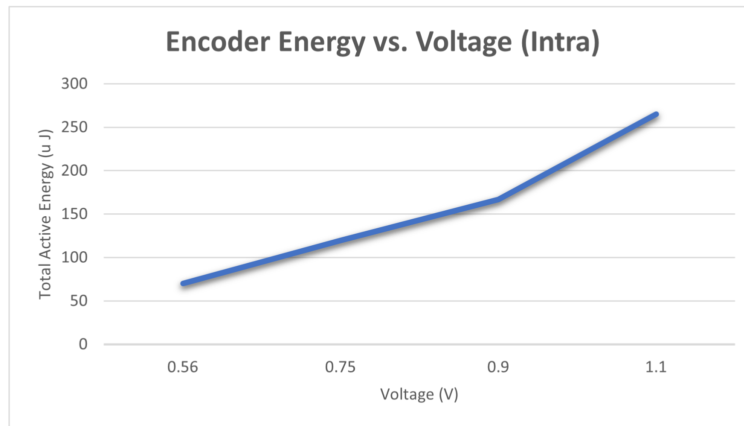

Figure 7.8: Encoder energy with respect to voltage (intra)

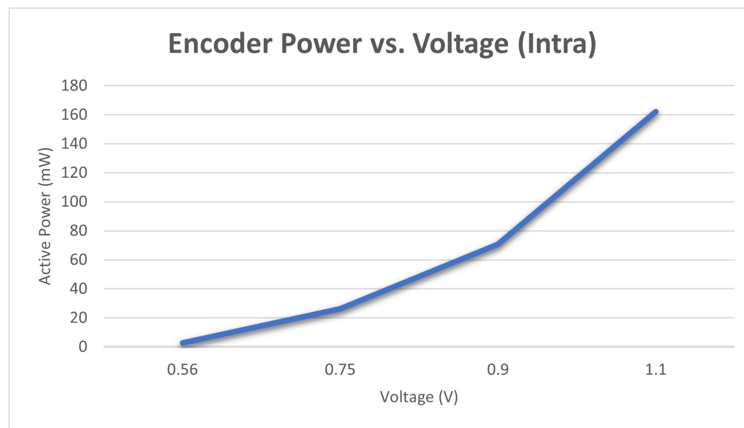

Figure 7.9: Encoder power with respect to voltage (intra)

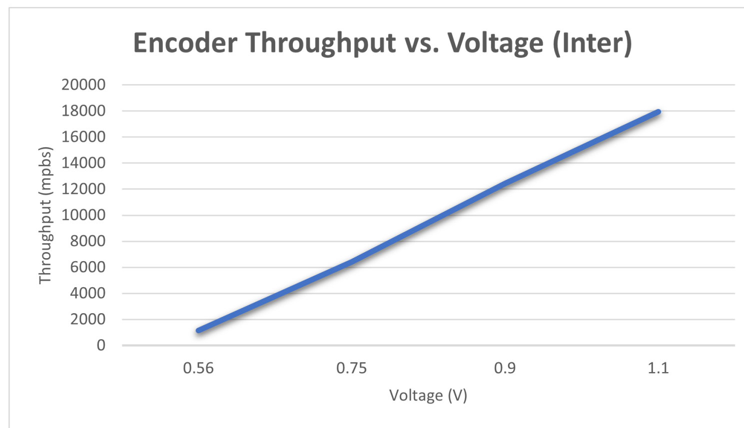

Figure 7.10: Encoder throughput with respect to voltage (inter)

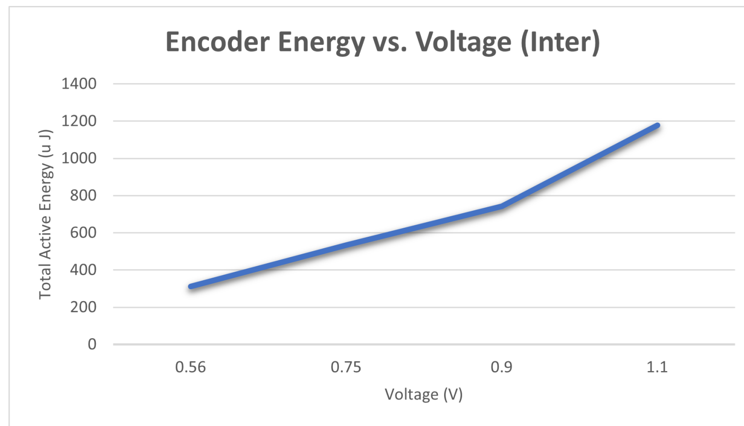

Figure 7.11: Encoder energy with respect to voltage (inter)

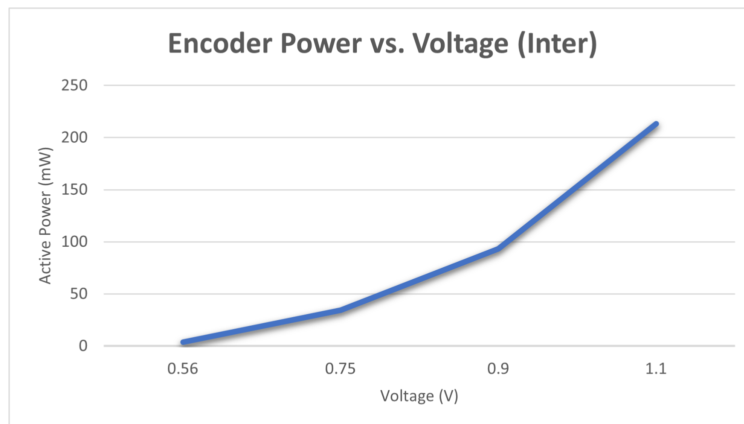

Figure 7.12: Encoder power with respect to voltage (inter)

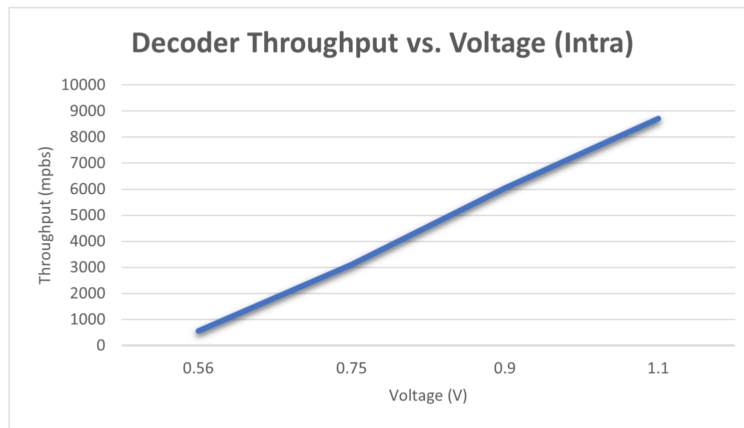

Figure 7.13: Decoder throughput with respect to voltage (intra)

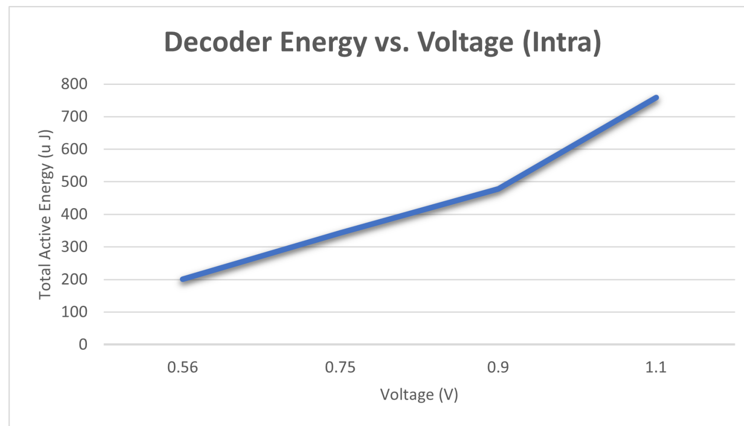

Figure 7.14: Decoder energy with respect to voltage (intra)

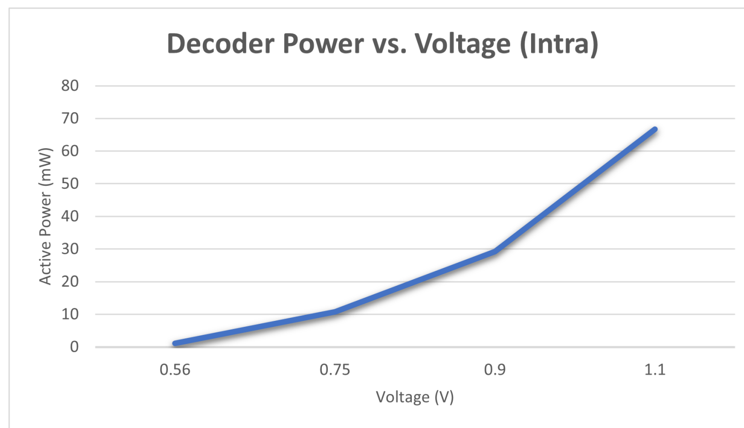

Figure 7.15: Decoder power with respect to voltage (intra)

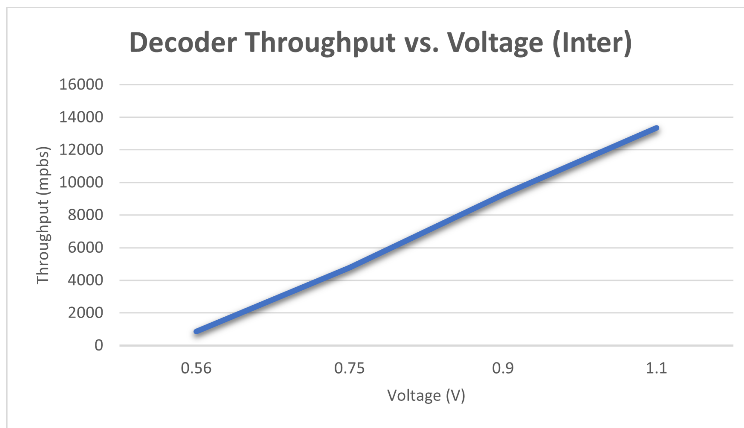

Figure 7.16: Decoder throughput with respect to voltage (inter)

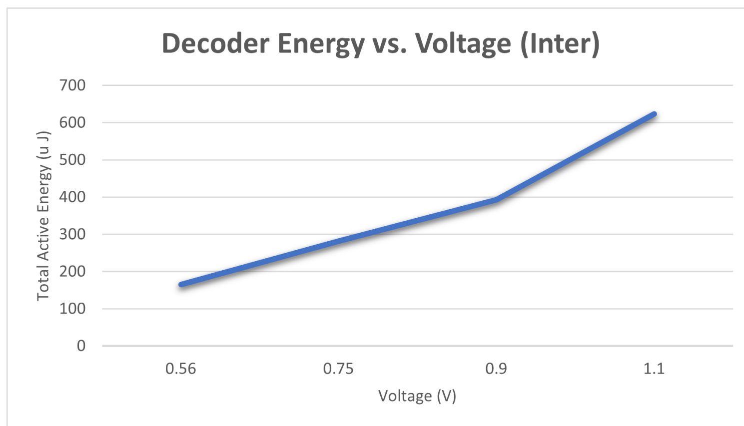

Figure 7.17: Decoder energy with respect to voltage (inter)

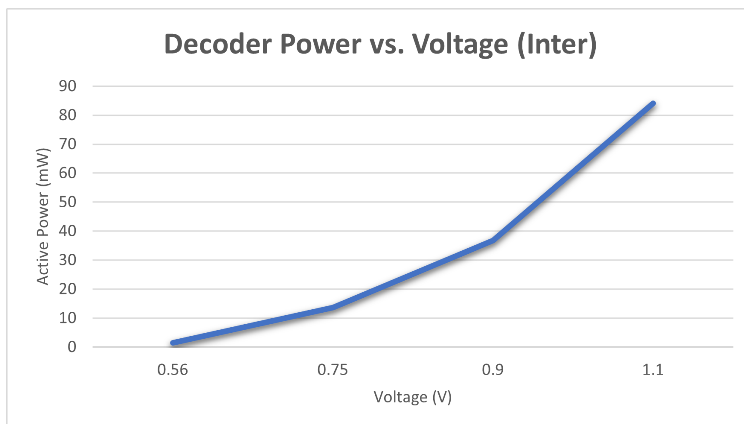

Figure 7.18: Decoder power with respect to voltage (inter)

7.5 Mapping

The proposed H.264/AVC encoder designed on the KiloCore II platform is mapped utilizing the Mapper2 software [29]. The Mapper2 tool is a general place-and-route framework. It is capable of mapping to a N-dimensional architectural model. In this work, it is used to map a 2-D processor array on KiloCore II.

7.5.1 Encoder

Figure 7.19 presents the processor core mapping of the proposed encoder.

All shared SRAM memory modules reside at the bottom of the image. These include the four memories for reconstructed pixels for the y, u, and v components. The other five memories are for previous intra prediction modes and number of coefficients for the y, u, and, v samples.

The left side contains three separate inputs containing pixel inputs for the luma and both chroma inputs. One output on the right side contains the encoded bitstream, while the other two outputs were utilized for debugging.

7.5.2 Decoder

Figure 7.20 presents the processor mapping for the proposed decoder.

The decoder mapping is significantly simpler than the encoder due to the lack of a predictive algorithm scheme. Additionally, the decoder requires two less shared memories due to the fact that the inverse CAVLC algorithm is not parallelized. Consequently, the number of coefficient values for the luma and chroma components are stored in the same SRAM module.

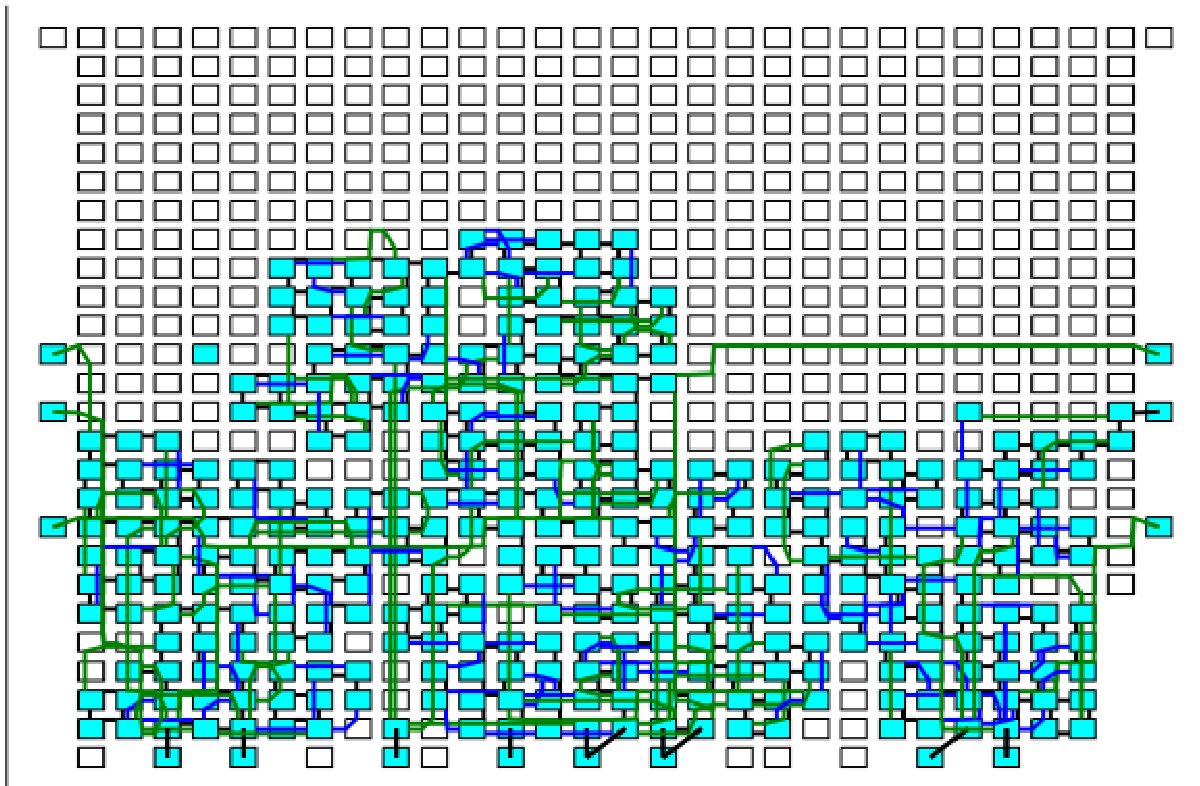

Figure 7.19: KiloCore II encoder place and routed map of cores

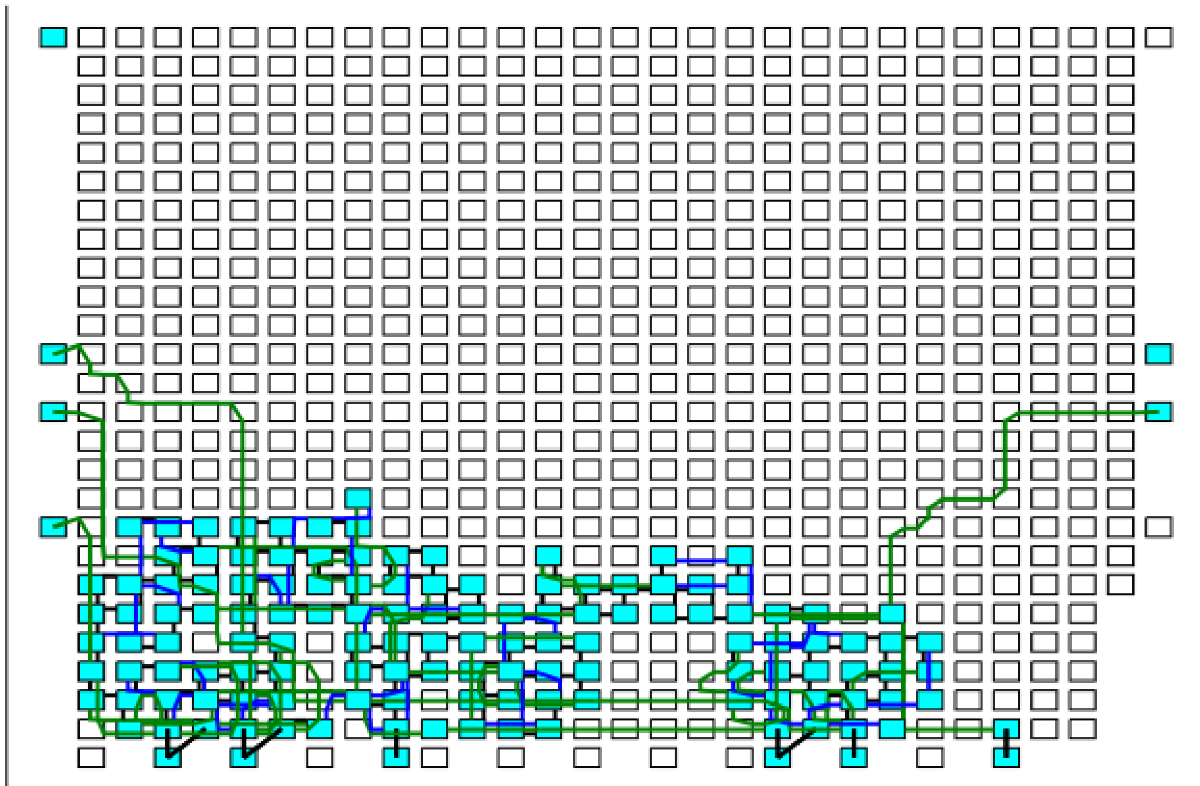

Figure 7.20: KiloCore II decoder place and routed map of cores

7.6 Comparisons

While ASICs typically optimize performance for a given task, they require a large amount of design time and money. FPGAs, on the other hand, compromise design flexibility and computational performance. Digital signal processors (DSP) are special programmable units that accelerate several functions common to video compression, such as multiplication and addition. General purpose CPUs offer the widest range of capabilities, but lack any dedicated hardware to accelerate performance.

KiloCore II is a high performance programmable CPU array aimed to offer both excellent flexibilty and performance [30]. KiloCore II aims to balance the performance and design tradeoffs between ASICs and other programmable platforms. It is generally accepted that the highest level of performance is obtained through ASICs. In the following section, the KiloCore II platform is compared to other implentations of H.264/AVC codecs on programmable platforms. Additionally, the synthesis results for the proposed RTL implementation in this work is provided.

The following sections discuss how performance results are abstracted for KiloCore II. Additionally, the comparison works are introduced briefly. Finally, the compared results are presented in tables 7.13, 7.14, 7.15, and 7.16. The results are split among two tables for both the encoder and decoder, where each respective table compares special purpose and programmable architectures seperately.

7.6.1 KiloCore II Performance

The throughput and energy performance of the H.264 codec in this work varies across quantization parameters and video sequence. In order to provide a reasonable performance indication, the throughput and energy results for all test cases listed in table 7.8 and 7.9 are collected.

The results listed in table 7.8 and 7.9 split intra and inter prediction results due to lengthy simulation times from Project Manager. In order to provide a single metric for both the encoder and decoder, it is assumed that video sequences contain an intra prediction frame every fifth frame. Consequently, a weighted average is used to combine intra and inter prediction frames together. The results of this calculation are provided in table 7.10 and are used to compare with other works.

Video Sequence	Frame Number	Prediction Mode	QP	Throughput (mbps)	Energy (uJ/MB)
suzie_qcif	0	Intra	15	60563	4.9
suzie_qcif	1	Intra	28	60563	3.9
suzie_qcif	2	Intra	50	60563	0.6
claire_qcif	0	Intra	24	60563	2.7
claire_qcif	1	Intra	19	60563	3.4
claire_qcif	2	Intra	34	60563	1.8
foreman_qcif	0	Intra	11	60563	4.9
foreman_qcif	1	Intra	12	60563	4.9
foreman_qcif	2	Intra	13	60563	4.9
suzie_qcif	1	Inter	15	17926	18.7
suzie_qcif	2	Inter	26	18242	18.4
claire_qcif	1	Inter	24	15391	21.8
claire_qcif	2	Inter	25	17235	19.5
foreman_qcif	1	Inter	28	14906	22.5
foreman_qcif	2	Inter	35	14033	23.9
Average	-	**Intra**	-	**60563**	**3.6**
Average	-	**Inter**	-	**16131**	**20.8**

Table 7.8: KiloCore II encoder performance

Video Sequence	Frame Number	Prediction Mode	QP	Throughput (mbps)	Energy (uJ/MB)
suzie_qcif	0	Intra	15	8704	24.1
suzie_qcif	1	Inter	28	13344	26.8
suzie_qcif	2	Inter	50	20124	21.2
claire_qcif	0	Intra	24	15431	13.5
claire_qcif	1	Inter	19	44561	15.1
claire_qcif	2	Inter	35	44963	15.1
foreman_qcif	0	Intra	11	8749	23.9
foreman_qcif	1	Inter	12	18484	22.1
foreman_qcif	2	Inter	13	31389	17.2
Average	-	**Intra**	-	**10961**	**20.5**
Average	-	**Inter**	-	**28810**	**19.6**

Table 7.9: KiloCore II decoder performance

	Throughput	Energy
Encoder	27239	16.5
Decoder	24347	19.8

Table 7.10: KiloCore II final weighted performance metrics

7.6.2 KiloCore II Performance with Motion Estimation Accelerator

Motion estimation is clearly the most computational and area intensive task for a given H.264 encoder. While the KiloCore II platform does not include a dedicated motion estimation accelerator processor, prior generations of KiloCore did contain such hardware [31]. In order to observe the potential of a KiloCore II platform equipped with a motion estimation accelerator, the following section presents an analysis to estimate the power and throughput for this configuration.

The motion estimation accelerator provided on chip for AsAP 2 (the second generation fine-grained many-core processor array) offers several search algorithm configurations with varying compression and computation performance [20]. One such search algorithm is the three step search algorithm which is identical to the search algorithm proposed in this work. The performance for the AsAP 2 motion estimation accelerator with a three step search algorithm is provided in table 7.11.

	Area (mm^2)	**Frequency (MHz)**	**Active Power (mW)**	**Cycles**	**Throughput mbps**	**Energy (uJ)**
AsAP 2 ME_ACC	0.67	938	196	174582	531910	36.4

Table 7.11: AsAP 2 motion estimation accelerator throughput and power performance for a single QCIF frame

The total throughput and power consumption for the encoder with a motion estimation accelerator is estimated through extrapolating the results from the throughput and energy by stage results outlined in section 7.2 and 7.3, respectively. The total energy is simply summed across all algorithmic blocks, including the chroma datapath. The throughput is estimated by summing the total time to complete the critical path. For the inter prediction mode of an encoder the critical path is dictated by the prediction , forward transform, and CAVLC blocks. Finally, a weighted average is calculated in order to find the total encoder performance metrics including intra prediction. Table 7.12 highlights the full results.

	Throughput (mbps)	**Energy (uJ/MB)**	(Area) **(mm^2)**
Intra Prediction	60563	3.6	13.76
Inter Prediction	77159	10.5	13.76
Average	**73010**	**8.7**	**13.76**

Table 7.12: KiloCore II encoder performance with motion estimation accelerator

7.6.3 Comparisons

The following sections compare relevant works to the proposed design in this paper. In all proceeding tables the best performance for a given category is in bold. The implemented H.264/AVC codec achieves up to 49.1× and 8.1× higher throughput than all compared encoders and decoders, respectively. The final column in each table provides a normalized ratio between each respective work and the proposed design (i.e. normalized throughput = compared throughput / lowest throughput). For both the encoder and decoder two tables are presented which are divided into special purpose and fully programmable architectures. The compared works for both the encoder and decoder are introduced briefly below.

Encoder Comparisons

FPGAs offer a good trade-off between performance and flexibility making them a common platform for encoder designs. Hamzaoglu [32] presents a low power H.264 encoder. In Li [33], a SoC FPGA implementation is compared to the performance of CPUs and DSPs. Finally, Lee [34] implements a high throughput FPGA encoder design. A CPU encoder implementation by Rao [35] achieves the lowest throughput of the compared encoders.

An implementation of an H.264 encoder on a DSP is presented in Ouyang [36] and Mohammadnia [37]. DSPs offer dedicated hardware for many arithmetic functions making their throughput competitive.

Huang [38] presents a custom architecture for an H.264 encoder. This design offers a high performance encoder for real-time 720p encoding. In Le [5], the H.264 encoder is implemented on KiloCore II's predecessor KiloCore.

Decoder Comparisons

Both Lee [39] and Peng [40] present a real-time baseline decoder on a SoC FPGA capable of encoding a QCIF video sequence at 20 frame-per-second. The design of an H.264 decoder is explored in Jian [41] and Pescador [42]. Since CPUs do not contain dedicated hardware, the throughputs are comparable. A custom IC by Major [43] achieves a competitive throughput. The proposed platform is a highly reconfigurable fabric of interconnected instruction cells which can be dynamically reconfigured to provide highly parallel FPGA-like representations of typical software operations. Similarly, an implementation of another decoder on a reconfigurable course-grained custom IC (referred to as ADRES) is presented in Mei [44].

Work	**Technology**	**Area (mm^2)**	**Clock Frequency (MHz)**	**Power (mW)**	**Energy (uJ/MB)**	**Throughput (mbps*)**	**Normalized Throughput****
Hamzaoglu [32] FPGA Xilinx Virtex 6	40 nm	-	135	**174**	**7.98**	21,780	14.6
Li [33] SoC FPGA Xilinx Zynq-7000	28 nm	-	190	-	-	26,030	17.5
Lee [34] FPGA Xilinx XC2V6000	150 nm	**8.003**	40	-	-	40,500	27.2
Rao [35] CPU ARM 11	-	-	69	-	-	1,485	1.0
This Work KiloCore II (Without ME_ACC)	32 nm	18.315	**1,780**	449	16.5	27,239	18.3
This Work KiloCore II (With ME_ACC)	32 nm	13.76	**1,780**	635	8.7	**73,010**	**49.1**

*Throughput measured in macroblocks-per-second (mbps)

**$Normalized\,throughput = comparison\,throughput/lowest\,throughput$

Table 7.13: Comparison of H.264 programmable encoders

Work	Technology	Area (mm²)	Clock Frequency (MHz)	Power (mW)	Energy (uJ/MB)	Normalized (mbps*)	Scaled Throughput
Ouyang [36] DSP ADSP-BF561	-	-	600	-	-	21,600	5.3
Mohammadnia [37] DSP DM648	-	-	900	-	-	11,880	2.9
Huang [38] UMC 180 nm Custom IC	180 nm	31.718	108	785	19.3	40,500	9.9
Le [5] Custom IC AsAP 3, Intra	65 nm	18.87	1,200	702	172.9	4,059	1.0
Le [5] Custom IC AsAP 3, Inter	65 nm	19.2	1,200	955	44	21,384	5.2
This Work (Without ME_ACC)	32 nm	18.315	**1,780**	**449**	16.5	27,239	6.7
This Work (With ME_ACC)	32 nm	**13.76**	**1,780**	635	**8.7**	**73,010**	**17.9**

*Throughput measured in macroblocks-per-second (mbps)

**$Normalized\ throughput = comparison\ throughput/lowest\ throughput$

Table 7.14: Comparison of H.264 special purpose encoders

Work	**Technology**	**Area (mm^2)**	**Clock Frequency (MHz)**	**Power (mW)**	**Energy (uJ/MB)**	**Throughput (mbps*)**	**Normalized Throughput**
Lee [39] FPGA Simulation	150 nm	10.6	30	-	-	11,880	3.5
Peng [40] FPGA Simulation	130 nm	-	120	**8**	**0.67**	11,880	3.5
Jian [41] CPU ARM	-	-	800	-	-	5,032	1.4
Pescador [42] CPU 3 GHz dual-core	-	-	-	-	-	3,368	1.0
This Work KiloCore II	32 nm	**7.205**	**1,780**	482	19.8	**24,347**	**7.2**

*Throughput measured in macroblocks-per-second (mbps)

**$Normalized\ throughput = comparison\ throughput/lowest\ throughput$

Table 7.15: Comparison of H.264 programmable decoders

Work	Technology	Area (mm^2)	Clock Frequency (MHz)	Power (mW)	Energy (uJ/MB)	Throughput (mbps*)	Normalized Throughput
Major [43] Custom IC RICA	130 nm	-	-	25.1	1.49	16,848	5.6
Liu [45] Custom IC 180 nm 1P6M CMOS	180 nm	11.289	1.2	**0.865**	**0.29**	2,970	1.0
Mei [44] Custom IC ADRES	-	-	-	-	-	11,880	4.0
Chattopadhyay [46] DSP TMS320	-	-	150	-	-	2,970	1.0
This Work KiloCore II	32 nm	**7.205**	**1,780**	482	19.8	**24,347**	**8.1**

*Throughput measured in macroblocks-per-second (mbps)

**$Normalized\, throughput = comparison\, throughput / lowest\, throughput$

Table 7.16: Comparison of H.264 special purpose decoders

Chapter 8

Summary and Future Work

8.1 Summary

The KiloCore architecture offers both a flexible and powerful platform for video compression algorithms, as found in works [21], [47], and [48]. This work implements the first baseline H.264 CODEC on the KiloCore II platform. The proposed CODEC is macroblock-level compliant and verified using the syntax structure generated from the JM reference software [24]. This work's implementation supports all nine luma element Intra prediction modes and all four chroma element Intra prediction modes. Motion estimation does not use sub-pixel interpolation. Luma motion estimation utilizes a 2-D logarithmic search algorithm while chroma elements use the prior frame's elements at the current index.

The proposed design on KiloCore II utilizes a high degree of parallelization to exploit various tasks in the H.264/AVC algorithm. A fully parallel design is not explored to balance power dissipation with throughput performance.

8.2 Future Work

Several H.264/AVC offers design flexibility with respect to computational complexity and bit compression performance. Consequently, the performance on any given implementation can drastically change with algorithmic tweeks. For example, the H.264 standard does not require that a motion estimation algorithm is used. If the motion estimation algorithm is removed and replaced with uniform motion vector generation, bit compression performance will drop but throughput and

power performance will increase significantly.

In order to fully understand the benefits of the KiloCore II platform, a complete evaluation of computational performance scaled to the consequent compression ratio must be explored for a variety of search algorithms. Without such a test, it is challenging to accurately compare how a KiloCore II architecture matches up with the state of the art. Additionally, testing how KiloCore II performs relative to various search algorithms will highlight the algorithm best fitted for the platform.

Additional improvements to the proposed motion estimation algorithm are also possible. The encoder utilized under half the available cores. This leaves significant room for increased parallelization and improved throughput as a result. In the proposed motion estimation design, the motion estimation kernel is instantiated three times as depicted in Figure 8.1. This had the effect of a three fold improvement in motion estimation when neglecting the introduction of additional routing hardware. For a baseline H.264 encoder there are nine total partitions required for calculation. In future works it is possible to instantiate six more motion estimation kernels and still map to the KiloCore array.

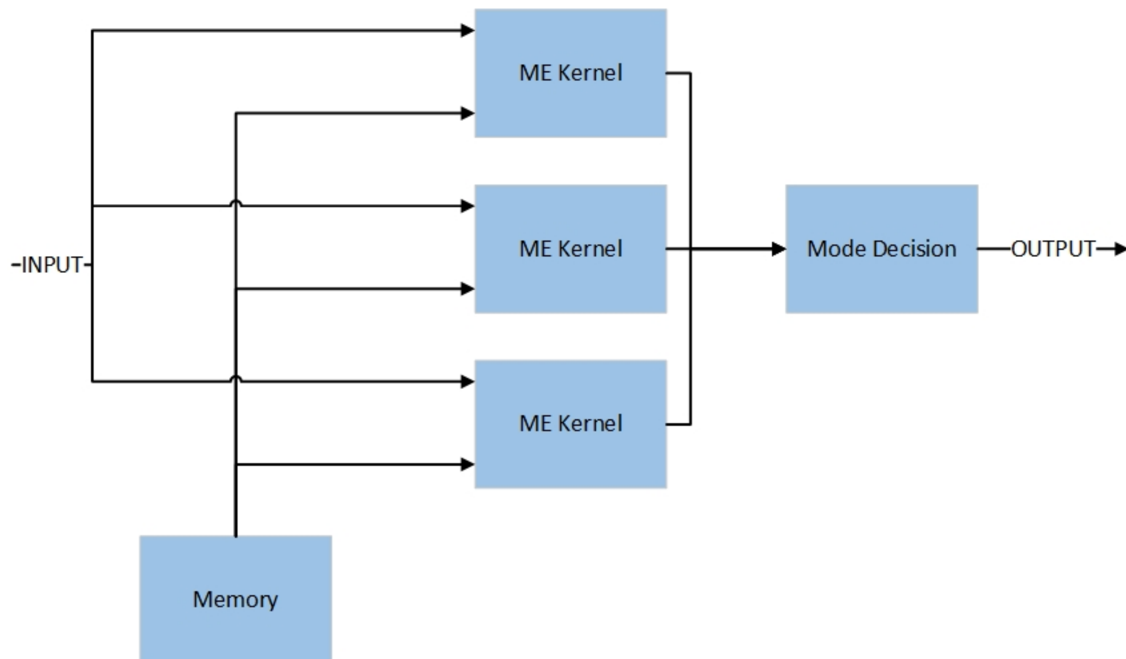

Figure 8.1: Parallel architecture for motion estimation

In the proposed design all context elements (i.e. number of coefficients and Intra prediction

modes) are stored in the shared memory modules. Memory read and writes induce a sizeable throughput penalty compared to arithmetic instructions. Since all macroblocks are processed in raster scan order, elements directly to the left of the current macroblock are processed most recently. This provides the opportunity to store all left context elements in processor DMEM for faster fetch and writes. This requires half the number of memory read and writes utilized by the current implementation.

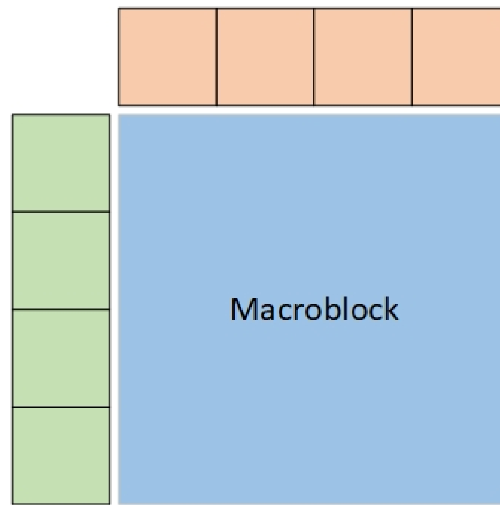

Figure 8.2: Memory optimization scheme for local memory (green) and SRAM (orange)

Bibliography

[1] Vincent Tabora. Progressive vs. interlaced. https://medium.com/hd-pro/progressive-vs-interlaced-e18e2924800e, 2019.

[2] I. Richardson. *The H.264 Advanced Video Compression Standard*. Wiley, United Kingdom, 2010.

[3] Brent Bohnenstiehl, Aaron Stillmaker, Jon J. Pimentel, Timothy Andreas, Bin Liu, Anh T. Tran, Emmanuel Adeagbo, and Bevan M. Baas. Kilocore: A 32-nm 1000-processor computational array. *IEEE Journal of Solid-State Circuits*, 52(4):891–902, 2017.

[4] Zhiyi Yu and Bevan M. Baas. A low-area multi-link interconnect architecture for gals chip multiprocessors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(5):750–762, 2010.

[5] Stephen T. Le. A fine grained many-core h.264 video encoder. Master's thesis, University of California, Davis, CA, USA, March 2010. http://www.ece.ucdavis.edu/vcl/pubs/theses/2010-03.

[6] Bitmovin video developer report. Technical report, BITMOVIN, 2021.

[7] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.

[8] Advanced video coding for generic audiovisual services. Standard, ITU-T, August 2021.

[9] Thaisa Silva, Joao Vortmann, Luciano Agostini, Sergio Bampi, and Altamiro Susin. Fpga based design of cavlc and exp-golomb coders for h.264/avc baseline entropy coding. In *2007 3rd Southern Conference on Programmable Logic*, pages 161–166, 2007.

[10] T. Wiegand, M. Lightstone, D. Mukherjee, T.G. Campbell, and S.K. Mitra. Rate-distortion optimized mode selection for very low bit rate video coding and the emerging h.263 standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(2):182–190, 1996.

[11] Minqiang Jiang and Nam Ling. On lagrange multiplier and quantizer adjustment for h.264 frame-layer video rate control. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(5):663–669, 2006.

[12] J. Jain and A. Jain. Displacement measurement and its application in interframe image coding. *IEEE Transactions on Communications*, 29(12):1799–1808, 1981.

[13] H.S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky. Low-complexity transform and quantization in h.264/avc. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):598–603, 2003.

[14] Zhiyi Yu and Bevan M. Baas. Implementing tile-based chip multiprocessors with gals clocking styles. In *IEEE International Conference of Computer Design (ICCD)*, 2006.

[15] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. Baas. Kilocore: A 32 nm 1000-processor array. In *IEEE HotChips Symposium on High-Performance Chips*, August 2016.

[16] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. Baas. KiloCore: A fine-grained 1,000-processor array for task parallel applications. *IEEE Micro*, 37(2):63–69, March 2017.

[17] Anh T. Tran and Bevan M. Baas. Achieving high-performance on-chip networks with shared-buffer routers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(6):1391–1403, 2014.

[18] Aaron Stillmaker, Brent Bohnenstiehl, and Bevan Baas. The design of the kilocore chip. In *ACM/IEEE Design Automation Conference*, Austin, TX, Jun. 2017.

[19] Michael Braly. A configurable H.265-compatible motion estimation accelerator architecture suitable for realtime 4K video encoding. Master's thesis, University of California, Davis, Davis, CA, USA, December 2015. `http://vcl.ece.ucdavis.edu/pubs/theses/2015-2.braly/`.

[20] Gouri Landge. A configurable motion estimation accelerator for video compression. Master's thesis, University of California, Davis, CA, USA, December 2009. `http://www.ece.ucdavis.edu/vcl/pubs/theses/2009-4`.

[21] Zhibin Xiao and Bevan M. Baas. A high-performance parallel cavlc encoder on a fine-grained many-core system. In *International Conference on Computer Design, (ICCD '08)*, pages 248–254, 2008.

[22] Sharmila Kulkarni. Implementation of context-based adaptive binary arithmetic coding on KiloCore processor arrays. Master's thesis, University of California, Davis, CA, USA, March 2021. `http://vcl.ece.ucdavis.edu/pubs/theses/2021-2.skulkarni/`.

[23] Silvaco. Nangate freepdf45 open cell library. `https://silvaco.com/services/library-design/`, 2022.

[24] Karsten Suehring. H.264/avc reference software. `https://iphome.hhi.de/suehring/tml/download/`, 2015.

[25] eerimoq. Python textparser. `https://pypi.org/project/textparser/`, 2022.

[26] Don Ho. Notepad++. `https://notepad-plus-plus.org/downloads/`, 2022.

[27] P. Seeling and M. Reisslein. Video transport evaluation with H.264 video traces. *IEEE Communications Surveys and Tutorials, in print*, 14(4):1142–1165, 2012. Traces available at `trace.eas.asu.edu`.

[28] Brent Bohnenstiehl, Aaron Stillmaker, Jon Pimentel, Timothy Andreas, Bin Liu, Anh Tran, Emmanuel Adeagbo, and Bevan Baas. A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array. In *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, pages 1–2, 2016.

[29] Mark Hildebrand. Mapper2 project. `https://github.com/hildebrandmw/Mapper2.jl`, 2018.

[30] Bevan M. Baas. A parallel programmable energy-efficient architecture for computationally-intensive DSP systems. In *Signals, Systems and Computers, 2003. Conference Record of the Thirty-Seventh Asilomar Conference on*, November 2003.

[31] Dean N. Truong, Wayne H. Cheng, Tinoosh Mohsenin, Zhiyi Yu, Anthony T. Jacobson, Gouri Landge, Michael J. Meeuwsen, Christine Watnik, Anh T. Tran, Zhibin Xiao, Eric W. Work, Jeremy W. Webb, Paul V. Mejia, and Bevan M. Baas. A 167-processor computational platform in 65 nm cmos. *IEEE Journal of Solid-State Circuits*, 44(4):1130–1144, 2009.

[32] Ilker Hamzaoglu, Aydin Aysu, and Onur Can Ulusel. A low power adaptive h.264 video encoder hardware. In *2014 IEEE Fourth International Conference on Consumer Electronics Berlin (ICCE-Berlin)*, pages 395–399, 2014.

[33] Zhenni Li, Jingjiao Li, Yue Zhao, Chaoqun Rong, and Ji Ma. A soc design and implementation of h.264 video encoding system based on fpga. In *2014 Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics*, volume 2, pages 321–324, 2014.

[34] Sukho Lee, Seongmo Park, Jinho Han, Nakwoong Eum, and Jongwon Park. A 40mhz dedicated hardware h.264/avc video encoder with the reducing memory access scheme. In *2008 IEEE International Symposium on Consumer Electronics*, pages 1–4, 2008.

[35] G.N. Rao, R.S.V. Prasad, D.J. Chandra, and S. Narayanan. Real-time software implementation of h.264 baseline profile video encoder for mobile and handheld devices. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 5, pages V–V, 2006.

[36] Kun Ouyang, Qing Ouyang, and Zhengda Zhou. Optimization and implementation of h.264 encoder on symmetric multi-processor platform. In *2009 WRI World Congress on Computer Science and Information Engineering*, volume 6, pages 265–269, 2009.

[37] Mohammad Reza Mohammadnia, Hasan Taheri, and Seyed Ahmad Motamedi. Implementation and optimization of real-time h.264/avc main profile encoder on dm648 dsp. In *2009 International Conference on Signal Acquisition and Processing*, pages 48–52, 2009.

[38] Yu-Wen Huang, Tung-Chien Chen, Chen-Han Tsai, Ching-Yeh Chen, To-Wei Chen, Chi-Shi Chen, Chun-Fu Shen, Shyh-Yih Ma, Tu-Chih Wang, Bing-Yu Hsieh, Hung-Chi Fang, and Liang-Gee Chen. A 1.3tops h.264/avc single-chip encoder for hdtv applications. In *ISSCC. 2005 IEEE International Digest of Technical Papers. Solid-State Circuits Conference, 2005.*, pages 128–588 Vol. 1, 2005.

[39] Suh Ho Lee, Ji Hwan Park, Seon Wook Kim, and Suki Kim. Implementation of h.264/avc baseline profile decoder for mobile video applications. In *2005 12th IEEE International Conference on Electronics, Circuits and Systems*, pages 1–4, 2005.

[40] Huan-Kai Peng, Chun-Hsin Lee, Jian-Wen Chen, Tzu-Jen Lo, Yung-Hung Chang, Sheng-Tsung Hsu, Yuan-Chun Lin, Ping Chao, Wei-Cheng Hung, and Kai-Yuan Jan. A highly integrated 8mw h.264/avc main profile real-time cif video decoder on a 16mhz soc platform. In *2007 Asia and South Pacific Design Automation Conference*, pages 112–113, 2007.

[41] Guo-An Jian, Ting-Yu Huang, Jui-Chin Chu, and Jiun-In Guo. Optimization of vc-1/h.264/avs video decoders on embedded processors. In *2009 Sixth International Conference on Information Technology: New Generations*, pages 1313–1318, 2009.

[42] F. Pescador, M. Blestel, E. Juarez, M. Raulet, and M. Garrido. H.264/svc decoder performance comparison for dsp-based consumer electronic applications. In *2011 IEEE 15th International Symposium on Consumer Electronics (ISCE)*, pages 171–176, 2011.

[43] Adam Major, Ying Yi, Ioannis Nousias, Mark Milward, Sami Khawam, and Tughrul Arslan. H.264 decoder implementation on a dynamically reconfigurable instruction cell based architecture. In *2006 IEEE International SOC Conference*, pages 49–52, 2006.

[44] Bingfeng Mei, F.-J. Veredas, and B. Masschelein. Mapping an h.264/avc decoder onto the adres reconfigurable architecture. In *International Conference on Field Programmable Logic and Applications, 2005.*, pages 622–625, 2005.

[45] Tsu-ming Liu, Ting-an Lin, Sheng-zen Wang, Wen-ping Lee, Kang-cheng Hou, Jiun-yan Yang, and Chen-yi Lee. An 865-uw h.264/avc video decoder for mobile applications. In *2005 IEEE Asian Solid-State Circuits Conference*, pages 301–304, 2005.

[46] T. Chattopadhyay, S. Banerjee, and A. Pal. Enhancements of h.264 encoder performance for video conferencing and videophone applications in tms320c55x. In *2006 IEEE International Symposium on Consumer Electronics*, pages 1–6, 2006.

[47] Zhibin Xiao, Stephen Le, and Bevan Baas. A fine-grained parallel implementation of a H.264/AVC encoder on a 167-processor computational platform. In *IEEE Asilomar Conference on Signals, Systems and Computers (ACSSC)*, November 2011.

[48] Zhibin Xiao and Bevan M. Baas. A 1080p H.264/AVC baseline residual encoder for a fine-grained many-core system. *Circuits and Systems for Video Technology, IEEE Transactions on*, 21(7):890–902, july 2011.