A FINE GRAINED MANY-CORE H.264 VIDEO ENCODER

By

STEPHEN THE UY LE
B.S. (Oregon State University) March, 2007

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:

Chair, Dr. Bevan M. Baas

Member, Dr. Venkatesh Akella

Member, Dr. Soheil Ghiasi

Committee in charge
2010

Abstract

Video encoding has become an integral part for everyday computing from televisions and computers to portable devices such as cell phones. Achieving high quality resolution over limited bandwidth has lead to the development of the H.264 video standard providing greater encoding performance. In this work an H.264 baseline video encoder is presented on a fine grained 167-core programmable processor allowing for greater flexibility and parallelization. The encoder presented is capable of encoding QCIF-resolution video at 1.00 GHz while dissipating an average of 438 mW, and CIF-resolution at 1.20 GHz while an average of 787 mW. The Asynchronous Array of Simple Processors (AsAP) platforms provides a new method of coding over a large number of simple processors allowing for a higher level of parallelization than digital signal processors (DSP) while avoiding the complexity of a fully application specific integrated circuit (ASIC).

Acknowledgments

I would like to take this chance to thank to everyone that made this work possible. First I would like to thank Professor Baas for his time, guidance, and support throughout my years at Davis. The lessons that I learned here will help me for the rest of my career. I would like to thank Professor Akella and Professor Ghiasi for their valuable support and time in reviewing my thesis.

Thank you to the University of California Davis for providing this educational institution for learning and enrichment. Graduate school has opened many more doors for me in the future.

I would also like to thank Zhibin Xiao and Gouri Landge for their previous work on video encoding and Dean Truong for helping me work on the AsAP chip, without your help it would have taken me much longer to complete this project. I would also like to thank Henna Huang, Layne Miao and all of the members of the VCL lab for their help and support throughout my project making it not only a great learning experience but a great place to work.

Lastly I would like to thank my family. To my wife and daughter who put up with me during these long months, thank you for your support and understanding on those 70+ hours of work a week. Thank you to my parents and siblings for their encouragement and support throughout this time.

This work was supported by ST Microelectronics; IntellaSys; SRC GRC Grant 1598 and CSR Grant 1659; UC Micro; NSF Grant 0430090, CAREER Award 0546907 and Grant 0903549; Intel; and SEM. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily relect the views of the National Science Foundation or any of the sponsors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Goals of Parallel Video Encoding

Demand for high quality video has become increasingly important in today's society from standard applications such as television broadcasting to streaming videos via cell phones. Video is often stored or transmitted prior to use, because of bandwidth limitations however video must be encoded for efficient transmission/storage. The computational complexity of this process has led to many different solutions with application specific processors having great success. Programmable solutions though flexible are not able to handle the computational load required and have focused on smaller applications. To achieve high quality video encoding on a small programmable chip, task and data level parallelism must be exploited at a fine grained level. The goal of this project is to develop a real-time H.264 video encoder with performance comparable to application specific processors and the flexibility of programmable processors.

1.2 Project Contributions

Research contributions of this project include:

- An MPI-C Baseline H.264 video encoder
- A real-time H.264 baseline video encoder on a asynchronous array of simple processors
- Thorough performance analysis for parallelization of a H.264 encoder

- Thorough processor analysis for a large fine grained application

1.3 Organization

The remainder of this paper is divided as follows. Chapter 2 provides a basic overview of video encoding and the H.264 video standard. Chapter 3 deals with processing platforms used for video encoding specifically the asynchronous array of simple processors proposed for this implementation. Chapter 4 discusses some tools, methodologies, and pitfalls of parallel programming. Chapter 5 presents the proposed implementation. Results, analysis, and comparisons with other encoders are given in chapter 6. Chapter 7 concludes the paper with possibilities for future work.

Chapter 2

Overview of Video Encoding and the H.264 Standard

Encoding a video sequence has always been a challenge due to the complexity of compressing then reproducing an exact copy of the original file again. As the picture size increases the problem becomes even greater, requiring more data to be compressed in the same amount of time. The H.264 standard [2] provides new encoding techniques yielding greater compression and higher quality. The standard itself does not present an encoder or decoder but a syntax that must be met to ensure that an encoded video stream can be decoded on a H.264 compliant decoder. Hence the syntax given in the standard is defined for the decoding process and an encoder must essentially produce that same syntax.

The H.264 Standard is a joint development of the Moving Picture Experts Group (MPEG) and the Video Coding Experts Group (VCEG) released by the International Telecommunication Union (ITU) Telecommunication Standardization Sector (ITU-T) as H.264 and Part 10 of MPEG-4.

2.1 General Video Encoding Concepts

Video sequences are a series of still pictures (referred to as frames from this point forward) that are flashed at a high rate to give the impression that objects in the pictures are moving. In most applications this requires approximately 25 frames per second (fps). Because of the high sampling rate, the difference between each successive frame is relatively small, hence if only the difference

Frames Per Second (frame rate)	Seconds Between Frames
20	0.05
25	0.04
30	0.03

Table 2.1: Time interval for various frame rates

Video Length (WxH in pixels)	1080p (1920x1080)	720p (1280x720)	SDTV (720x480)	CIF (352x288)
1sec	0.19	0.08	0.03	0.01
1 min	11.20	4.98	1.87	0.82
1 hour	671.85	298.60	111.97	49.27

Table 2.2: Size of uncompressed video in gigabytes @ 30fps and 24-bit pixel depth

between each frame is encoded the transmitted data is fairly small in comparison to the original video. Increasing the frame rate (number of frames per second) can provide an illusion of nearly continuous motion but would requires greater encoder/decoder performance to provide real time video.

2.1.1 Digital Video

Video is stored digitally then converted for viewing, however storing raw video would require an enormous amount of space as shown in Table 2.2. To store a one hour segment of standard definition quality video would require 47 regular DVDs! To compress videos, three parameters are often looked at:

- temporal sampling: frame rate - how often is a frame sampled from a video sequence?
- spatial sampling: pixels - what is the number of pixels used to represent each frame?
- pixel depth: bits - how many bits are used to represent each pixel?

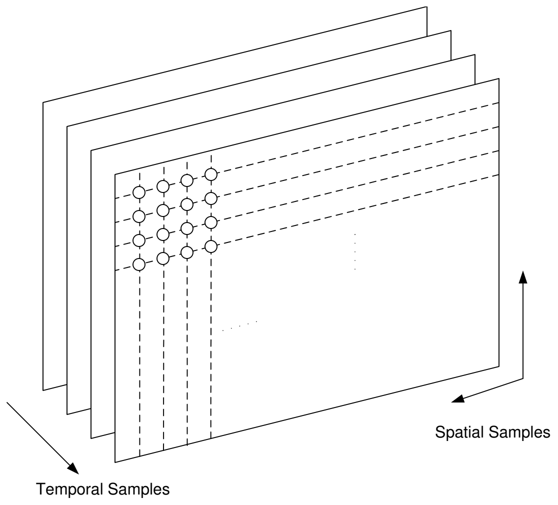


Figure 2.1: Temporal and spatial sampling

Format	Horizontal x Vertical Resolution	Pixels per Frame (4:2:0 Format)	Ratio of Pixels per Frame Compared to SDTV
Sub-QCIF	128x96	12288	.02 : 1.0
Quarter CIF	176x144	38016	.06 : 1.0
CIF	352x288	152064	.25 : 1.0
4CIF (SDTV)	704x576	608256	1.0 : 1.0
720p	1280x720	1382400	2.3 : 1.0
1080p	1920x1080	2073600	3.4 : 1.0

Table 2.3: Sample of various frame sizes

2.1.2 Video Format

Various video formats are used depending on the quality of the video needed, smaller formats are more compact and require less space to store but do not provide very high quality resolution; larger formats allow for more detail but require more storage space, and consequentially more computational power to encode. The Common Intermediate Format (CIF) and smaller ones are commonly used for streaming type applications, such as mobile devices. 4CIF is often used for standard definition televisions (SDTV) and DVD-videos, and 720p (720 lines of progressively scanned data) or 1080p is commonly used for High Definition (HD) quality video. Table 2.3 and Fig. 2.2 give a comparison of some common video formats, larger ones are of course possible simply by increasing the horizontal and vertical resolution of each frame.

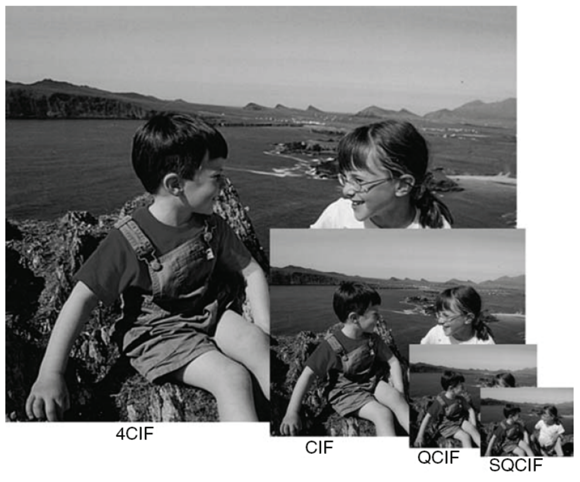


Figure 2.2: Different frame sizes for digital video [1]

RGB

The pictures in Fig. 2.2 are shown in black and white where each spatial sample is represented by one value giving the brightness of the pixel. To represent color images at least three values are required per pixel. One of the common methods of representing this color space is the RGB format where Red, Blue, and Green are each represented by one number giving the brightness desired, when combined together these three primary colors can create any other color.

YCbCr (YUV)

Another common format YCbCr(commonly referred to as YUV) takes advantage of the fact that the human eye is more sensitive to brightness than color, that is we can notice slight changes in light and dark easier than different shades of a color. In RGB format each color is represented equally, to get every pixel in color requires 3 values per pixel. In the YUV format the brightness/luminance (luma) is separated from the color (chroma) so that each can be given a different weight. One variable (Y) determines the luminance component, and two variables U and V give the chrominance of each pixel. The conversion between RGB and YUV format can be done using the simplified equations (2.1) recommended by the ITU-R.

$$\begin{aligned} R &= Y + 1.402Cr \\ G &= Y - 0.344Cb - 0.714Cr \\ B &= Y + 1.772Cb \end{aligned} \tag{2.1}$$

Sampling Formats

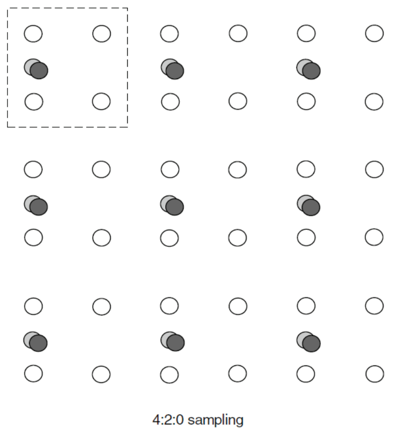
Various sampling format for YUV give different weights to the luma and chroma components. Full sampling (referred to as 4:4:4) gives equal weight to all three, similar to RGB, this format requires 3 values to represent each pixel. The 4:2:2 format gives the chroma components half the weight of the luma components. For each 4x4 block of luma pixels, the chroma component is represented with the same weight in the vertical direction but half the weight in the horizontal position, that is every other column is represented with both luma and chroma components, and the intermediate columns are only represented by luma components. The 4:4:4 and 4:2:2 formats are generally used for high quality color videos. The more popular 4:2:0 format used in this paper gives chroma one quarter the resolution of the luma component. As shown in Fig. 2.3(a), for each 4x4 block of luma pixels there is only one U and V component. Also note that the numbering scheme for 4:2:0 does not necessarily correspond to representations and directions.

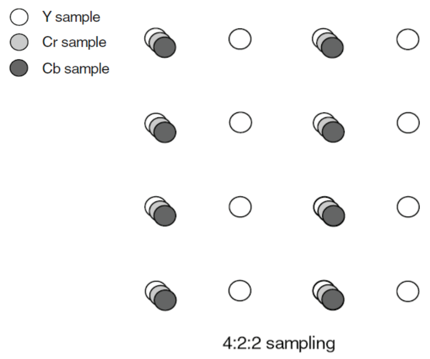
2.1.3 Macroblock Partitioning

In sampling video frames, pixels are grouped into blocks of 16x16 to form a macroblock. Figure 2.6 shows the partition of macroblocks on a CIF video frame. Encoding is done on a macroblock basis with macroblock 0 in the top left corner and the last macroblock in the bottom right. For the 4:2:0 format, each 16x16 (256 pixels) luma macroblock corresponds to two 8x8 (64 pixels) chroma block. Within each macroblock the pixels are ordered starting with index 0 in the top left corner and 255 in the bottom right corner.

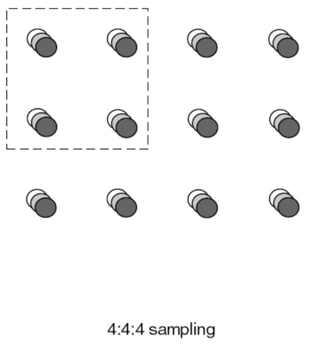
2.1.4 Encoding Motion

Between each successive frame there is relatively little differences in motion. Generally it is from either the camera panning/moving or from some object/person moving, Fig. 2.7 shows





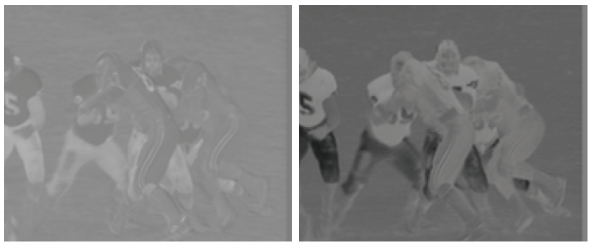
(a) 4:2:0 format (b) 4:2:2 format



(c) 4:4:4 format

Figure 2.3: Various video sampling formats [1]

Figure 2.4: Y component of YUV picture



(a) U Component in YUV (b) V Component in YUV

Figure 2.5: U and V Components of YUV picture

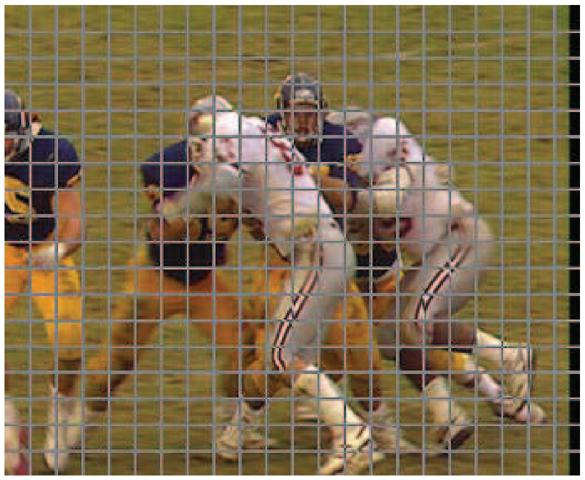
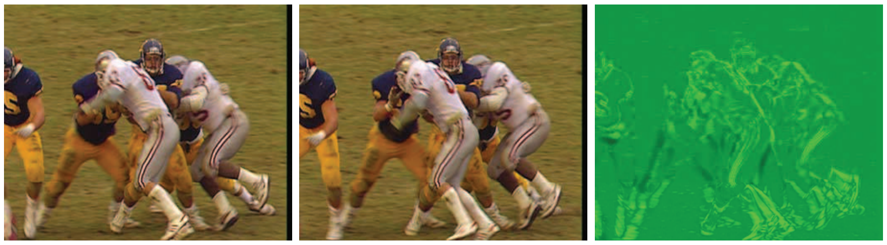
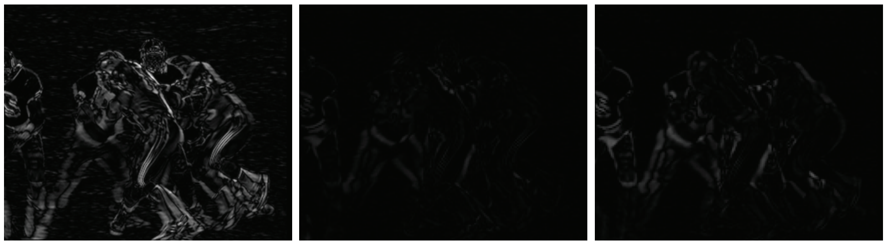


Figure 2.6: Macroblocks partition of 16x16 pixels in CIF frame



(a) frame 0 (b) frame 1 (c) frame 1 - frame 0



(d) Y component of difference (e) U component of difference (f) V component of difference

Figure 2.7: Difference between two frames

two successive frames with the bright spots in Fig. 2.7(c) showing the difference. To determine the difference between each frame, each macroblock (16x16 square of pixels) is compared to a similar area in a previous frame to find the closest match. One method is to overlay the two regions then do direct subtraction to find the sum of absolute differences (SAD) for that position, the current macroblock is then moved one pixel in any direction and the SAD is recalculated. Once this is done a certain number of times, the position with the minimum SAD is chosen to be encoded. Since there is little motion between frames due to the high frame rate, the search area can generally be limited to a small area. If two frames are uncorrelated (have no similarities) the SAD will be much greater, this will only generate more data to be encoded but will not affect the accuracy/quality of the decoded picture.

2.2 Overview of H.264

The H.264 standard defines a syntax for decoding a compressed video, in this work an encoder perspective will be taken. The encoding process should match the decoding process as close as possible to ensure quality video compression and decompression. For a detailed explanation of

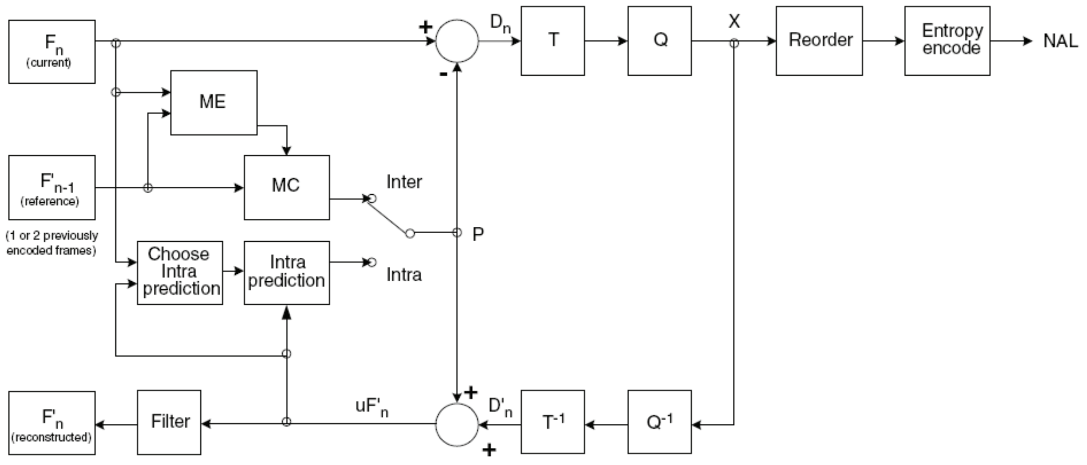


Figure 2.8: General H.264 encoder path [1]

the decoder please refer to the standard [2].

2.2.1 Encoding Path

Figure 2.8 shows the main functional blocks that are generally included in a H.264 compliant encoder. The encoder provides multiple paths for encoding depending on which mode is chosen, for the baseline encoder only two paths are available.

The top blocks in figure 2.8 make up the forward/encoding path, while the bottom blocks represent the reconstruction path for reference frames. The input (Fn) contains the frame to be encoded, macro blocks to be encoded are sent to the intra or inter prediction modules. After intra or inter prediction, a predicted macroblock (P) is formed and subtracted from the original input (Fn) to form (Dn) the difference macroblock. The difference macroblock is then integer transformed and quantize to produce a set of transform coefficients (X). The transform coefficients are then entropy encoded and sent out.

The reconstruction path is formed from the transform coefficients (X) prior to being entropy encoded. They are scaled and inverse transformed to produce reconstructed difference macroblock (Dn’) which are then added to back to the prediction macroblock (P) to from the reconstructed macroblock (uFn’). For display purposes, the reconstructed macroblock is also goes through a filter to remove any differences introduced in the quantization and scaling process.

2.2.2 Profiles

The H.264 standard supports three types of profiles: baseline, main, and extended. In the baseline profile intra (I) slices and inter (P) slices, entropy encoding via context-adaptive variable-length coding (CAVLC), slice groups, and redundant slices are supported. Intra frame are produced from data in the current frame only, inter frames are produced from previously encoded frames. The main profile also supports B frames/slices which are predicted from both previous and future frames as well as other methods for optimizing coding. The extend profiles supports switching I and P frames for more efficient switching between frames as well as other optimization methods. Figure 2.9 shows a visual representation of the differences and similarities between the three types of profiles. A new commonly used set of parameters is referred to as the constrained baseline profile which includes only the options that are overlapped between all three profiles, specifically, I slices, P slices, and CAVLC, this allows for a simpler encoder/decoder. This work deals with the baseline profile, specifically the constrained baseline options and will be discussed in the section below, for a more detailed description of the main and extended please refer to the standard [2] and Richardson [1].

2.2.3 Intra Prediction

In intra prediction the current macroblock is encoded using only the previously encoded macroblocks in the same frame. The first frame for every video sequence must be intra coded since there are no previous frames that can be used as reference. Intra prediction is also commonly used when switching video sequences where there is little to no correlation between the frames.

Since encoding is done in order from the top left corner to the bottom right corner, the macroblocks directly above and to the left are generally available for comparison. Because of the high correlation between neighboring macroblocks in a frame, a fairly accurate prediction can be made to predict the current macroblock. Intra prediction supports 16x16 macroblock partition and 4x4 macroblock partition for the luma components, the 16x16 modes are generally used for homogeneous areas where there is relatively little difference such as a background, the 4x4 block mode are used in areas of greater detail such as facial features. Both 16x16 and 4x4 prediction are computed then compared, the best prediction (minimum SAD) mode is then sent off to be

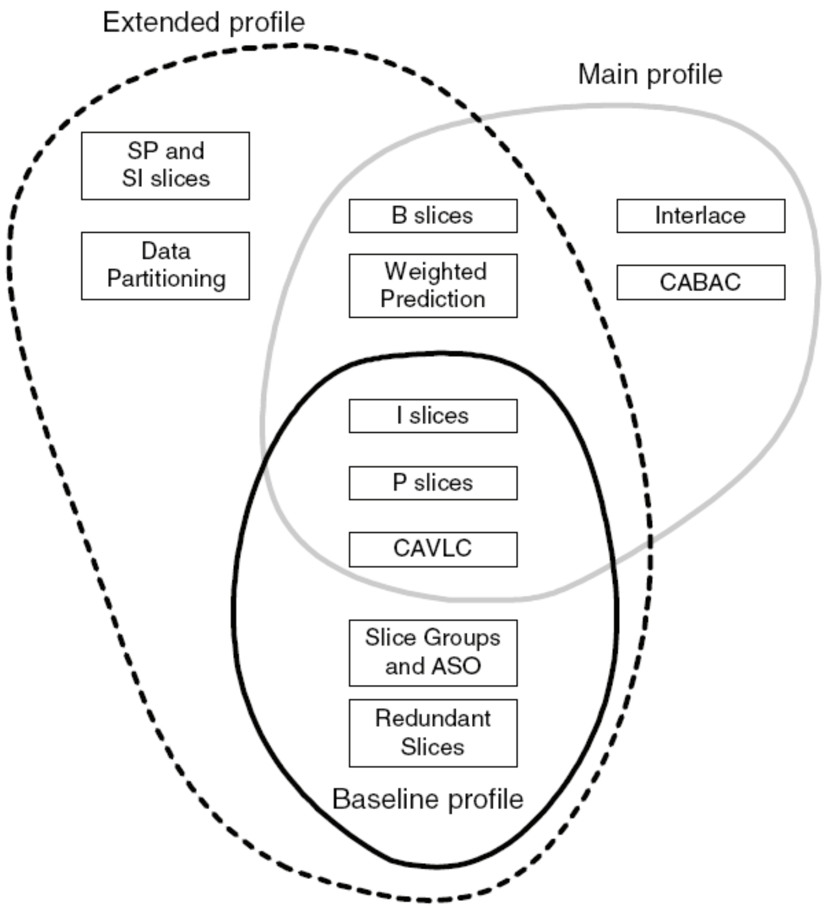


Figure 2.9: H.264 profiles [1]

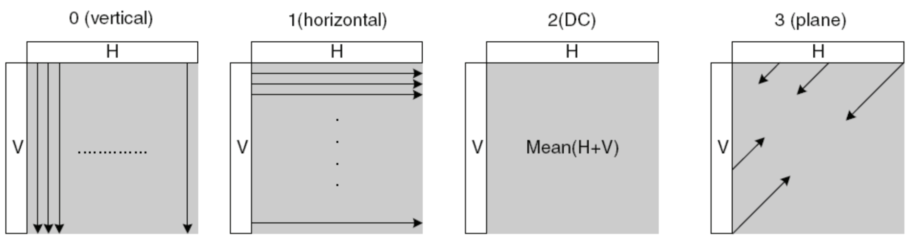


Figure 2.10: Prediction modes for intra 16x16 macroblock partition [1]

encoded. A predicted value for each pixel is first determined depending on the mode, this value is then subtracted from the current pixel to get the residue which will be encoded.

16x16 Prediction

In intra 16x16 luma prediction, four modes are possible, vertical, horizontal, DC, and plane. All four modes are computed to determine the closes match (least amount of residue - minimum SAD), the best mode is then chosen for comparison with intra 4x4 luma prediction. In vertical prediction, the predicted value is taken from the last row of the above macroblock (H), shown in Fig. 2.10, hence for every pixel in that column the same predicted value is used. In horizontal prediction, the predicted value is taken from the right most column of the left neighboring macroblock in the same row (V), as with vertical prediction, the same predicted value is used for the entire row, and is subtracted from the current pixel. In DC prediction, the average of last row of the above macroblock and the right most column of the left macroblock values (H+V)are taken to be the predicted value, if either of above or left data is unavailable, the predicted value for that side is take to be 2 $\hat{}$(bit_depth-1) (for a bit depth of 8, this is 128), hence this mode can always be used for prediction. In DC prediction mode, the same predicted number is used for the entire block. In plane mode a linear plane function is generated from the upper and left samples (H, V) with different predicted values depending on the location.

4x4 Prediction

In intra 4x4 luma prediction the four modes from intra 16x16 prediction are available as well as five additional modes. The 16x16 macroblock is divided into four rows and four columns

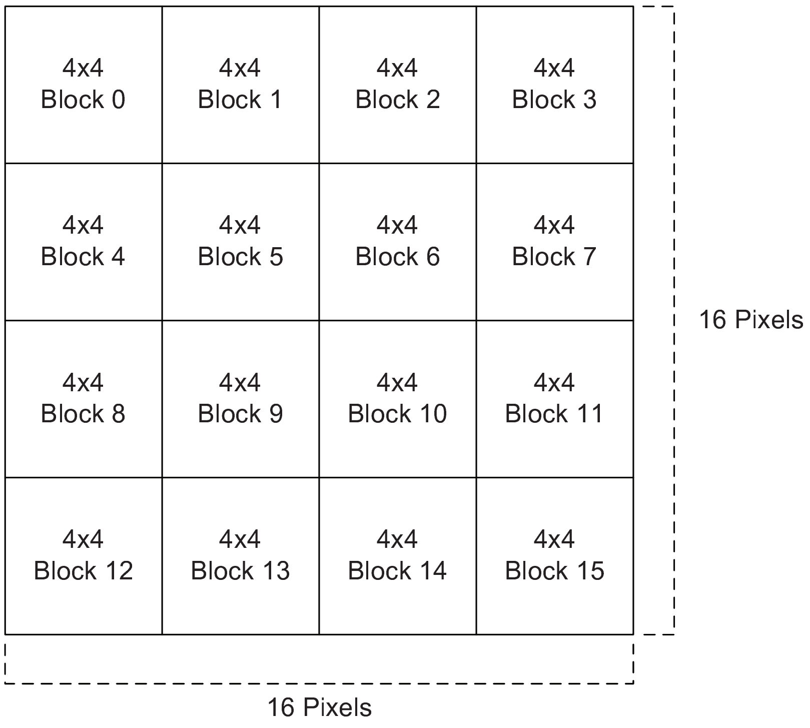


Figure 2.11: Partition for intra 4x4 macroblocks

producing 16 blocks 4 pixels tall by 4 pixels wide as shown in Fig. 2.11. The best mode for each 4x4 block is first determined, once this is done the total SAD for all 16 blocks are compared with the 16x16 prediction mode to determine the best one. Values used for prediction are computed from either the above and left macroblock if the current 4x4 block lies on an edge (blocks 0-3, 4, 8, or 12), and within the same macroblock for the other 4x4 blocks. The modes available for prediction are shown in Fig. 2.12. Vertical, horizontal, and DC modes are computed in a similar manner as intra 16x16 mode. For the remaining modes the predicted value is extrapolated from the top, left, and top-right pixels. For a more detailed description of these modes please refer to the H.264 Standard.

Chroma Prediction

Intra chroma prediction is done independent of the luma prediction but with similar modes to intra 16x16 prediction numbered differently. In DC prediction, the chroma block is divided into a total of four 4x4 blocks and predicted in a similar manner to intra 4x4 blocks but with different constraints on which side (H) or (V) is chosen for the prediction value. Please refer to standard for

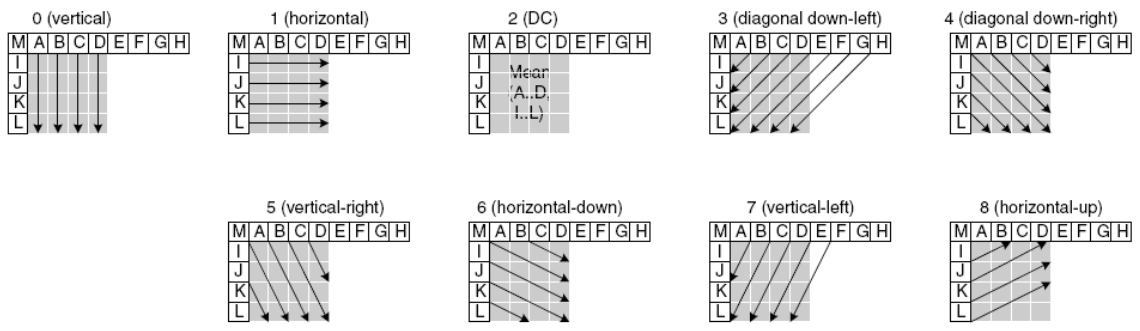


Figure 2.12: Prediction modes for intra 4x4 macroblock partitions [1]

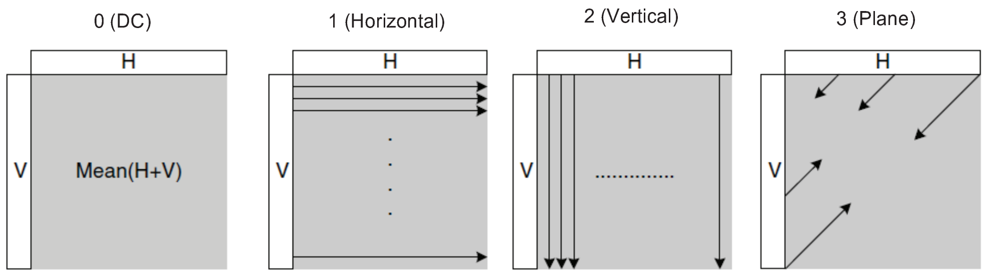


Figure 2.13: Intra prediction chroma modes [1]

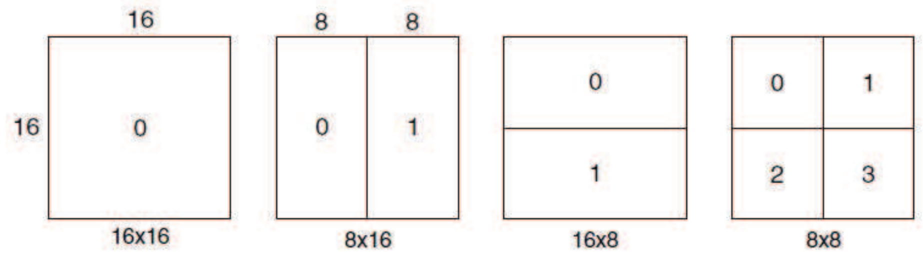
a more detailed description of how the predicted value is determined.

Encoding Prediction Modes

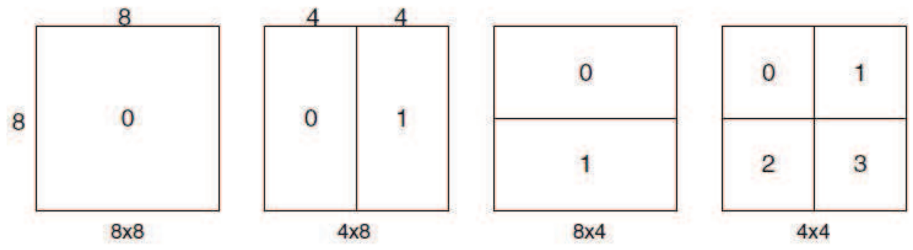
The prediction mode for each macroblock and 4x4 block if used must also be sent to the decoder in order to reproduce the current frame. In intra 4x4 prediction, there is a high probability that the prediction modes for adjacent (above and left) blocks are similar to the current one. Based on this the encoder and decoder can calculate the most probable mode of coding. If the predicted mode for coding is the same as the one used, a flag is asserted and only one bit is needed to signify this, if a different mode is used the flag is de-asserted and the encoding mode used must be sent.

2.2.4 Inter Prediction

In inter prediction macroblocks are predicted from previously encoded frame, using the reconstructed frame data from the encoder rather than the original data to provide the closest match



(a) ME Block Partition



(b) ME Sub-Block Partition

Figure 2.14: Macroblock partitions for motion estimation [1]

to the data used by the decoder. Four macroblock partitions are available for inter prediction given in Fig. 2.14(a), if an 8x8 block mode is chosen, each 8x8 block can be furthered partitioned as shown in Fig. 2.14(b). An example of how different partitions may be chosen for a frame are shown in Fig. 2.15.

Motion Estimation

In motion estimation a search window of 3x3 macroblocks (48x48 pixels) is often used, the current block is moved within the search window to find the best match by calculating the resulting residue. Once the best matching block is found a set of motion vectors for that block is calculated by taking the difference in position of the current block in the frame with the position of the chosen block in the reference frame, this provides directions for the decoder to find the best matching position. The search is repeated for the other block modes and the best one is chosen.

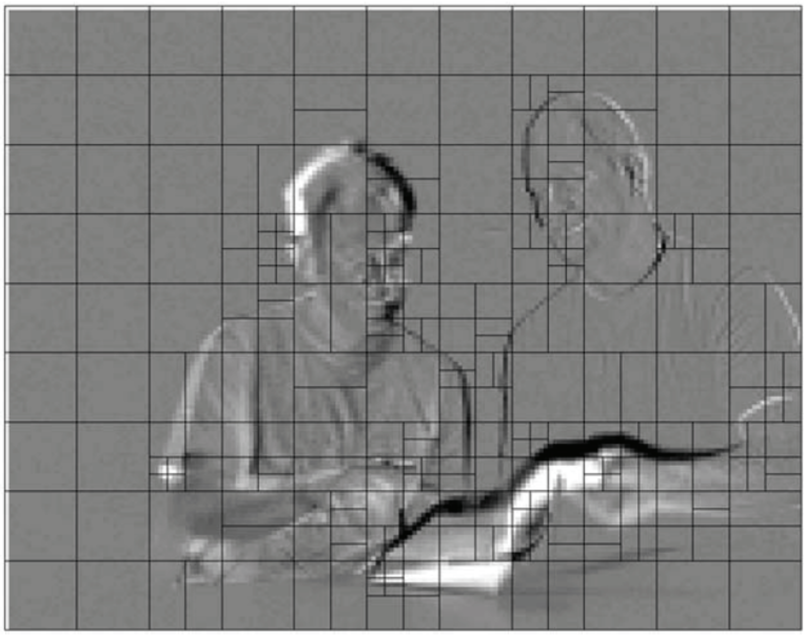


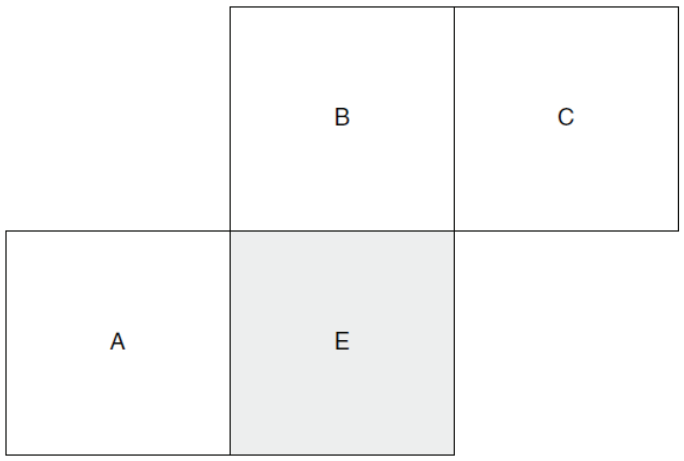
Figure 2.15: Sample macroblock partition for ME [1]

Encoding Motion Vectors

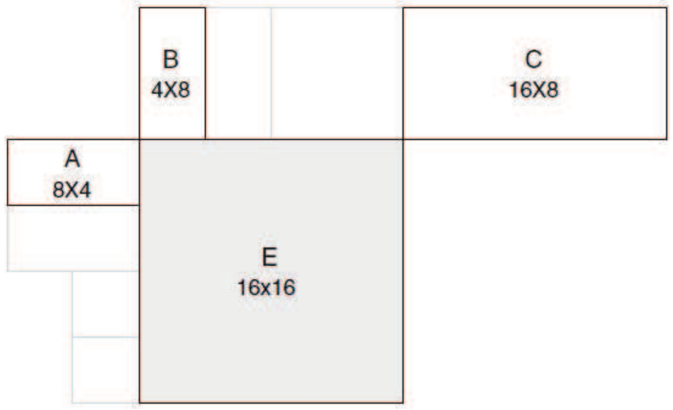
Motion vectors are encoded similarly to prediction modes in intra 4x4 prediction. There is a high correlation between neighboring blocks since objects that move generally consists of groups/block of pixels, hence motion vectors are predicted from the above, left, and above-right set of motion vectors. If the blocks chosen are of similar size, Fig. 2.16(a) shows which ones are used for prediction. If the neighboring blocks are of different sizes, Fig. 2.16(b) shows which blocks are used for prediction under certain conditions, please refer to the standard [2] as to when which blocks are used. The neighboring motion vectors are used to generate a predicted motion vector which is used to subtract the current motion vector from to produce a motion vector difference that is encoded and sent.

2.2.5 Integer Transform and Quantization

To further reduce the number of bits required for representation, the difference macroblock is integer transformed and quantize. Prior to the integer transform each macroblock is reordered and data is sent according to Fig. 2.17 regardless of what prediction mode is used. If intra



(a) ME Same Size Neighbor Blocks



(b) ME Different Size Neighbor Blocks

Figure 2.16: MV prediction from neighboring blocks [1]

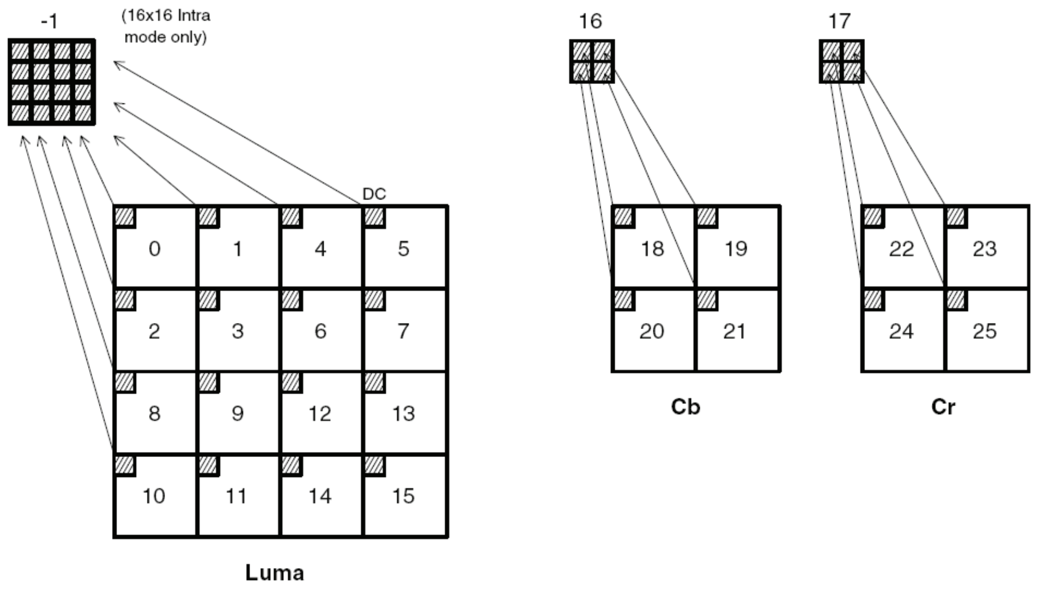


Figure 2.17: Block order for integer transform and CAVLC [**?**]

16x16 prediction is used, an additional block is sent containing the DC coefficients (first pixel of each 4x4 block) prior to the rest of the data, hence 25 blocks are sent rather than 24 in the normal case. In H.264, three types of transforms are used, a Hadamard for the DC coefficients used in intra 16x16 prediction, a Hadamard transform for the DC chroma coefficients in any macroblock, and DCT based integer transform for the remaining residue data.

4x4 Residual Transform & Quantization

Blocks 0-15 and 18-25 first undergo an integer transform that uses only integer arithmetic (additions and shifts) which allows the decoder and reconstruction path to reproduce the data with 100% accuracy. The quantization/scaling process however is not loss-less and introduces minor errors in the reconstruction of data. The reconstruction path however is built to mirror the decoding path, thus the same error is introduced in both sides and will yield the same prediction for the following blocks. The DC coefficients for the chroma components, blocks 16 and 17 are taken after this transform and quantization process. The DC coefficients for intra 16x16 mode are bypassed directly to the Hadamard transform though.

The forward integer transform is done using equation (2.2) where X represents the 4x4 input block to be transformed and Y is the resulting transform. For a full derivation of this equation please refer to Richardson [1].

$$\begin{bmatrix} & & \\ & Y & \\ & & \\ & & \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} & & \\ & X & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \tag{2.2}$$

After the integer transformed, the coefficients are quantize to further reduce the energy needed to send/store the residual data by rounding. The rounding is performed by equation (2.3) where Y_{ij} is the input after the integer transformed, Z_{ij} is the resulting output, the MF (multiplication factor) is derived in tables within the H.264 standard and f is defined by equation (2.5) for inter prediction and equation (2.6) for intra prediction. The quantization parameter (QP) in equation (2.4) is in the range of 0-51 and the floor function is the matlab rounding function. Varying the QP varies the resolution of the encoded values and the amount of work needed to encode the remaining data.

$$|Z_{ij}| = (|Y_{ij}| * MF_{ij} + f) >> qbits$$
$$sign(Z_{ij}) = sign(Y_{ij}) \tag{2.3}$$

$$qbits = 15 + floor(QP/6) \tag{2.4}$$

$$f = 2^{qbits}/6 \text{ :inter prediction} \tag{2.5}$$

$$f = 2^{qbits}/3 \text{ :intra prediction} \tag{2.6}$$

4x4 DC Transform & Quantization

If the selected encoding mode is intra 16x16 the luma DC coefficients (first coefficients for every 4x4 luma block) are sent directly to a Hadamard transform given by equation (2.7) and quantize by using equation (2.8). The input X_DC is from the original 4x4 input, Y_DC is the transformed output, and Z_DC is the quantize output. Qbits and f are defined as before and $MF_{0,0}$ is the MF coefficient in the (0,0) position from the tables used in equation (2.3).

$$\begin{bmatrix} & & \\ & Y_DC & \\ & & \\ & & \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} & & \\ & X_DC & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{2.7}$$

$$\begin{aligned} |Z_{DC(ij)}| &= (|Y_{DC(ij)}| * MF_{0,0} + 2f) >> (qbits + 1) \\ sign(Z_{D(ij)}) &= sign(Y_{D(ij)}) \end{aligned} \tag{2.8}$$

2x2 Chroma DC Transform & Quantization

The chroma DC coefficients are first transformed using equation (2.2) then sent to another Hadamard transform for further transform and quantization. The four coefficients for Cb and Cr are transformed separately using equation (2.9) and are quantize using equation (2.10). Constants $MF_{0,0}$, f, and qbits are defined as before.

$$\begin{bmatrix} & Y_C & \\ & & \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} & X_C & \\ & & \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.9}$$

$$\begin{aligned} |Z_{C(ij)}| &= (|Y_{C(ij)}| * MF_{0,0} + 2f) >> (qbits + 1) \\ sign(Z_{C(ij)}) &= sign(Y_{C(ij)}) \end{aligned} \tag{2.10}$$

2.2.6 Reference Frame Reconstruction

After the forward encoding process data is passed to both the entropy encoder and reference frame reconstruction block. The chroma DC coefficients and intra 16x16 DC coefficients are first inverse transformed and re-scaled then re-inserted back into their respective blocks before being inverse transformed again. For details and the equations used for reconstruction please refer to Richardson [1]. The reconstructed data is then stored and used for both intra and inter prediction.

2.2.7 Entropy Coding

After undergoing integer transform and quantization, the residue data can be further encoded using a context-adaptive variable-length coding method which takes advantage that there are mostly zeros, the number of non zero coefficients for neighboring blocks are correlated, most of the non-zero data is either positive or negative one, and that coefficients closer to the DC value (closer to the beginning) are generally higher. The prediction modes (intra prediction) and motion vectors (inter prediction) however do not have these properties and are coded using Exp-Golomb coding.

Exp-Golomb Coding

Exp-Golomb coding is used for encoding the prediction modes in intra prediction, motion vectors in inter prediction and the block patterns for intra and inter prediction. The following references tables are partially shown in this text, complete tables can be found in the H.264 standard section 9.1.1. Intra and inter prediction modes are predicted using unsigned Exp-Golomb codes in Table 2.5. Motion vectors are first mapped to code numbers using Table 2.4 then the code words are encoded using Table 2.6.

In encoding macroblocks, some blocks after the transform and quantization process contain only data with value zero, these blocks do not need to be entropy encoded using the CAVLC but can be signaled using the coded block pattern. The coded block pattern is a 6-bit field with the first four bits used to represent an 2x2 region of the macroblock (4 8x8 blocks). If all the data within that block is zero then the corresponding bit is set to zero else it is set to one. The last two bits are used to show the three possibilities for both chroma blocks this is shown in Table 2.7, once the codeNum is obtained it is coded using Table 2.8 which correspond to the bit strings in Table 2.5.

codeNum	syntax element value
0	0
1	1
2	-1
3	2
4	-2
...	...

Table 2.4: Signed Exp_Golomb code table

Bit string	codeNum	inter mode
1	0	16x16
010	1	16x8
011	2	8x16
00100	3	8x8 w/ sub partition
00101	4	8x8 w/o sub partition
...	...	...

Table 2.5: Explicit Exp-Golomb code

Bit string form	Range of codeNum
1	0
0 1 X_0	1-2
0 0 1 X_1 X_0	3-6
0 0 0 1 X_2 X_1 X_0	7-14
...	...

Table 2.6: Prefix and suffix for codeNum

bit field	description
5:4	00: All Chroma Data 0 01: DC = 0, AC != 0 10: DC 0, AC!= 0
3 2 1 0	0: Block = 0 1: Block != 0

Table 2.7: Coded block pattern for intra4x4 modes

codeNum	coded block pattern	
	Intra4x4, Intra8x8	Inter
0	47	0
1	31	16
2	15	1
3	0	2
...	...	...

Table 2.8: CodeNum for block patterns

CAVLC Components	Description
coeff_toden	Encodes the number of non-zero and trailing one coefficients per block
trailing_one_sign_flag	Encodes the sign of the trailing ones per block
level	Encodes the magnitude and sign of the remaining non-zero coefficients
total_zero	Encodes the number of zero coefficients after the first non-zero number
run_before	Encodes the number of zeros before each non-zero coefficients

Table 2.9: CAVLC components

Context Adaptive Variable Length Coding (CAVLC)

The CAVLC process does not take data in raster scan order (top left to bottom right) but in zig zag order as shown in Fig. 2.18, data is however encoded in the reverse order, starting at the end and working back up. First the coeff_token (non-zero coefficients and trailing ones) for the 4x4 block is determined and encoded using tables. The sign of each trailing one (positive or negative ones that occur at the end of the zig zag scan before any other non-zero coefficients) is then encoded and sent. The level (sign and magnitude) of the remaining non-zero coefficients are then encoded. Finally the number of zeros before the first non-zero coefficient and the number of zeros between each non-zero coefficient is encoded and sent. For a more detailed description of the CAVLC along with the tables please refer to Zhibin's [3] research work.

Encoding coeff_token

The coeff_token encodes the number of total non-zero coefficients in each 4x4 block, this can range from 0-16. The number of trailing ones is limited to three per 4x4 block, all other trailing ones after that are considered as normal non-zero coefficients. The encoded value is based

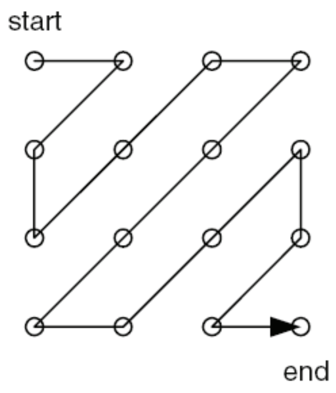


Figure 2.18: Zig zag scan order for CAVLC [1]

on 4 look up tables depending on the number of non-zero values in the above and left blocks. Because the decoder cannot know the number of non-zero coefficients in the current block this value is predicted from surrounding previously encoded blocks, as the number of non-zero coefficients increases different tables are selected.

Encoding trailing_one_sign_flag

After integer transform and quantization, many of the remaining coefficients are +/- 1 and 0's. Trailing ones are counted in reverse zig-zag scan order and are the +/- 1's before any other non-zero coefficients are encountered. Up to 3 +/- 1's are counted as trailing ones and only the sign needs to be encoded and can be done with just one bit, 0 - negative, 1 - positive.

Encoding levels

The level for each remaining non-zero coefficient (in reverse zig-zag order) is encoded in two parts, the suffix (0-6 bits) and a prefix. The values for the prefix and suffix vary depending on the previously encoded values for the current block. Please refer to subsection 9.2.2.1 of the H.264 standard for a detailed description of the level parsing process.

Encoding total_zero

The total number of zero coefficients before the first non-zero coefficient (normal zig-zag order) is also encoded using a table, by doing this any zeros preceding the first non-zero coefficient

NAL Unit Octet (bit)	Description
7:3	NAL Unit Type
2:1	NAL Reference Id (NRI)
0	Forbidden Bit, always 0

Table 2.10: Bit field for NALU

will not need to be encoded.

Encoding run_before

The number of zeros before each non-zero value is also encoded in order to reconstruct the residual data block. Starting in reverse zig-zag scan order, the number of zeros is encoded. The encoded value changes based on the number of zeros remaining and the last run of zeros before the last coefficient does not need to be encoded since we already know the total number of zeros.

2.2.8 Network Abstraction Layer

Once the residue data, prediction modes, and motion vectors have been encoded they are added to the network abstraction layer to be sent the encoder. The network abstraction layer (NAL) contains vital information for the decoder to decode and reproduce the original video sequence. Each video sequence is started with a picture parameter set (PPS) that contains information for the rest of sequence. This is followed by the sequence parameter set (SPS) which contains data on variables specific the frames in this sequence, and finally before each frame is sent, a slice header (SH) is sent with relevant data for the current frame. Each PPS, SPS, and SH is signaled in the bit sequence by a NAL unit which contains a start code, block type, and data. The bit fields for the NALU is given Table 2.10, for a complete listing of the values and function please refer to table 7-1 of the H.264 standard.

Picture Parameter Set
profile_idc
constrained_set0_flag
constrained_set1_flag
constrained_set2_flag
constrained_set3_flag
reserved_zero_4bits
level_idc
seq_parameter_set_id
log2_max_frame_num_minus4
pic_order_cnt_type
log2_max_pic_order_cnt_lsb_minus4
num_ref_frames
gaps_in_frame_num_value_allowed_flag
pic_width_in_mbs_minus1
pic_height_in_map_units_minus1
frame_mbs_only_flag
direct_8x8_inference_flag
frame_cropping_flag
vui_parameters_present_flag

Table 2.11: PPS fields

Sequence Parameter Set
pic_parameter_set_id
seq_parameter_set_id
entropy_coding_mode_flag
pic_order_present_flag
num_slice_groups_minus1
num_ref_idx_10_active_minus1
num_ref_idx_11_active_minus1
weighted_pred_flag
weighted_bipred_idc
pic_init_qp_minus26
pic_init_qs_minus26
chroma_qp_index_offset
deblocking_filter_control_present_flag
constrained_intra_pred_flag
redundant_pic_cnt_present_flag

Table 2.12: SPS fields

Slice Header
first_mb_in_slice
slice_type
pic_parameter_set_id
frame_num
idr_pic_id
pic_order_cnt_lsb
no_output_of_prior_flag
long_term_reference_flag
slice_qp_delta

Table 2.13: SH fields

Chapter 3

Processing Platforms Used for Video Encoding

Various platforms have been used for video processing with varying results. The platform spectrum ranges from general purpose computers to chips built specifically for video encoding, depending on the application performance needed, different platforms are chosen. This chapter looks at some the various platforms used for video encoding and presents the proposed platform.

3.1 Related Work in H.264 Processing

Previous implementations of H.264 video encoders have been done on nearly all levels. Encoders on general purpose (GP) processors have been developed as the golden model for comparison and development. General purpose processors however are not able to meet the constraints of real-time video encoding and is used primarily as a reference point. Multimedia co-processors have also been used for video encoding, these however have focused on smaller frame sizes, generally CIF and below with high power consumptions making them not as desirable in portable applications. ASIC have managed to provide low power real time video encoding, however since these processors are application specific, they cannot be easily modified for future implementations or other configurations. Section 3.1.1 presents implementations on GP processors, section 3.1.2 presents solution on DSP chips, and section 3.1.3 presents ASIC implementations.

GP Processing Platform	
Processor Type	Intel Xeon w/ HT
Number of Processors	4
Threads	8
Speed	2.8GHz
L2 Cache	256 KB
L3 Cache	2 MB
Performance	4.6 frames/s (CIF)

Table 3.1: Performance of H.264 Video encoder on Intel Quad Core Processor [4]

DSP Processing Platform	
Processor	Intel PXA27x
Speed	624 MHz
Accelerator	Intel MMX (64-bit SIMD)
Memory	
Performance	49 frames/s (QCIF Average)

Table 3.2: Performance of video encoder on Intel DSP platform [5]

3.1.1 Video Encoding on General Purpose (GP) Processors

The H.264 video encoder is implemented in software for general purpose processors, Chen and others [4] looks specifically at optimizing an encoder on Intel's Pentium 4 processor. Because of the complexity of the encoding process their implementation was done using processors with multi-threading capabilities using software optimization to speed up the process by 4.6x over traditional SIMD implementations which provide approximately 1 frame/s for CIF-resolution sequences.

3.1.2 Video Encoding on Digital Signal Processors (DSP)

Digital signal processors have provided more promising results with faster throughput. Wei and others [5] has implemented a real time H.264 video encoder on an Intel PXA27x processor used on the HP IPAQ hx4700 PDA which includes Intel's Wireless MMX technology for multimedia acceleration yielding an average of 49 frames/s for CIF-resolution sequences using a QP value of 28.

ASIC Processing Platform	
Technology	.180 um CMOS
Voltage	1.8V
Core Area	7.68 x 4.13 mm^2
SRAM	34.72 KB
Performance (SD 720x480)	81 MHz 581 mW
Performance (720p 1280x720)	108 MHz 785 mW

Table 3.3: Encoder performance on ASIC Platform [6]

3.1.3 Video Encoding on Application Specific Integrated Circuits

ASIC have by far provided the best performance compared to other implementations, Huang and others [6] have implemented a chip consisting of five major blocks (Integer/Fractional ME, Intra Prediction, Entropy Encoding, and Deblock Filtering), to perform real-time 720p encoding.

3.2 Proposed H.264 Video Encoder Platform

The proposed architecture for this video encoder is an asynchronous array of simple processors (AsAP) [7] specifically the second generation AsAP2 developed by the VLSI computation lab at the University of California Davis. This platform allows for a highly parallel implementation over a small area, using little power with performance scalability.

3.2.1 General Overview of AsAP2 Architecture

The AsAP array has 167-processors, with 3 dedicated processors for FFT computation, Viterbi coding, and motion estimation. There are also 3 shared memory banks at the bottom of the array each containing 16KB of memory. Some key features of the AsAP processors are:

- each processor is small, containing only 128 word of instruction and 128 words of data memory

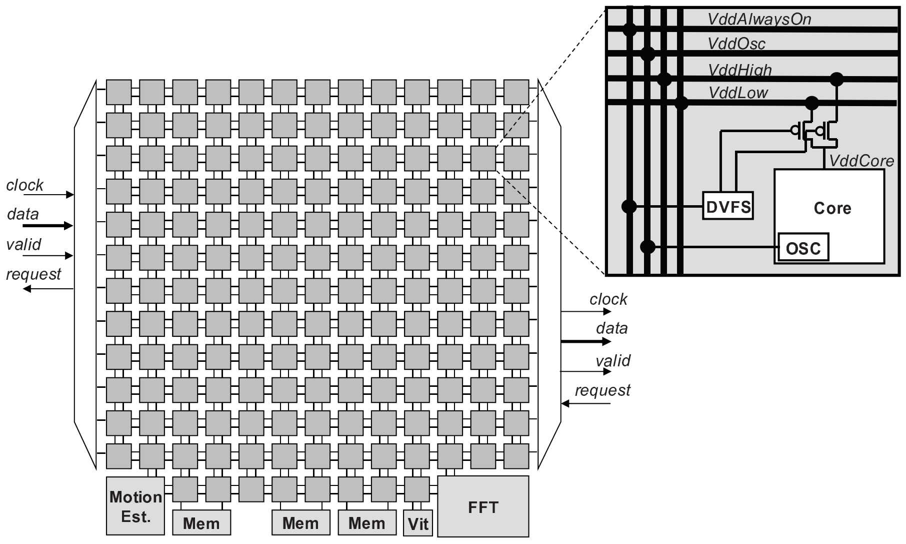


Figure 3.1: General layout of AsAP array

- each processor has 4 links in each direction for nearest neighbor and long distance communication
- each processor has 2 inputs with dual-clock FIFO's [8]
- each processor's voltage and frequency can by dynamically scaled for optimization

3.2.2 Dynamic Voltage and Frequency Scaling (DVFS)

Each processor in the AsAP array can be independently frequency scaled from 10MHz to 1.2GHz as well as choose from either of two voltage rails (Vdd_{high} and Vdd_{low}) [9] allowing for performance optimization and reduced energy consumption for computationally less intensive workloads. The ability to have each processor configured independently allows for higher performance in bottle-necks areas in the encoding path and less wasted energy on idle modules. Because of the instruction and data memory constraints of AsAP many processors are used for routing or long distance connections only, these processors can be set to lower frequencies and voltages to minimize power consumption.

Frequency	Voltage	Power
1.2 GHz	1.3 V	62 mW
1.07 GHz	1.2 V	47.5 mW
66 MHz	.675 V	.608 mW

Table 3.4: AsAP power measurements for various voltage and frequency configurations

3.2.3 Memory Architecture

Each processor has 128 words of instruction memory for programming and 128 word data memory for storage, a 27 word dynamic configuration memory is also available for use as pointers and setting input output configurations. The data memory (DMem) is a single ported SRAM 16-bits wide by 128 words. The memory space is replicated allowing for memory access for two operands in a single cycle. The three shared memory (16 KB each) [10] at the bottom of the processor array can be accessed by two processor each for additional storage space. Each contains a single ported 16-bit x 8KWord SRAM with single cycle read and writes.

3.2.4 Processor Interconnect

Each processor has 8 links as shown in Figure 3.2, the links are independent of the core so they can configured for long distance communication (bypassing the core). Two dual clock FIFOs can connect any of the 8 links to the core for input data. The dual clock FIFO's allow each core to run at a separate frequency and still be able to communicate with each other. Each FIFO is 64 words deep with a stall signal to tell the communicating processor when it is full. The link directions are signified by north, east, south, west, and up, down, left, right. For long distance communication, compass direction outputs can talk to any other compass direction, and any direction output can be connected to another (north and connect to south but not down). The input direction to each processor is statically configured during programming, while the output direction can by dynamically switched during run-time.

3.2.5 Motion Estimation Accelerator

Motion estimation is a highly computational task and has been implemented as a hardware processor on AsAP. The motion estimation accelerator (ME_ACC) [11] allows for communication

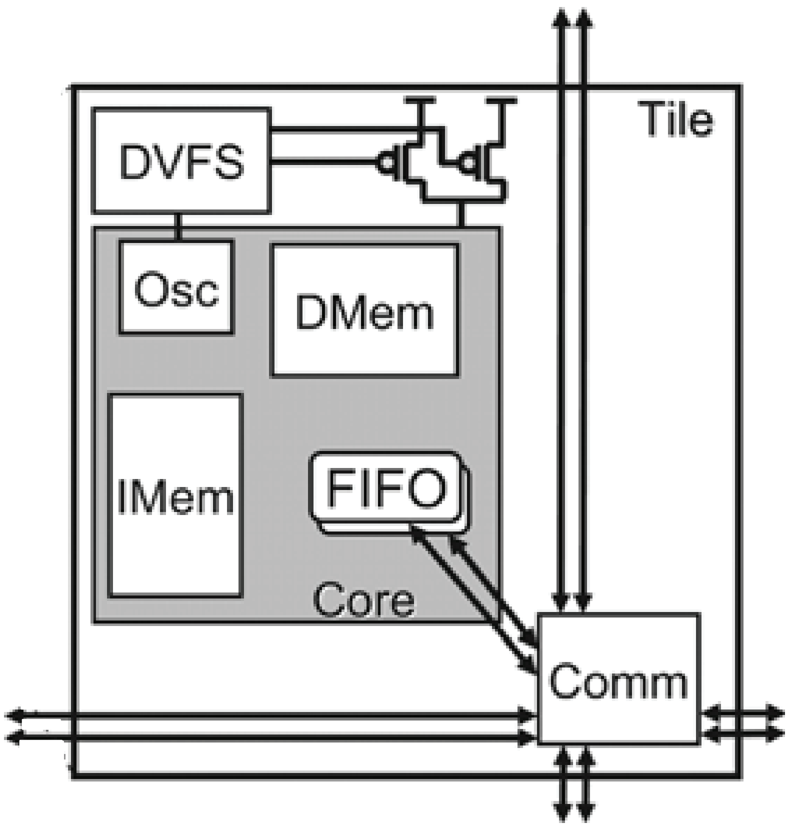


Figure 3.2: Major blocks in AsAP processor

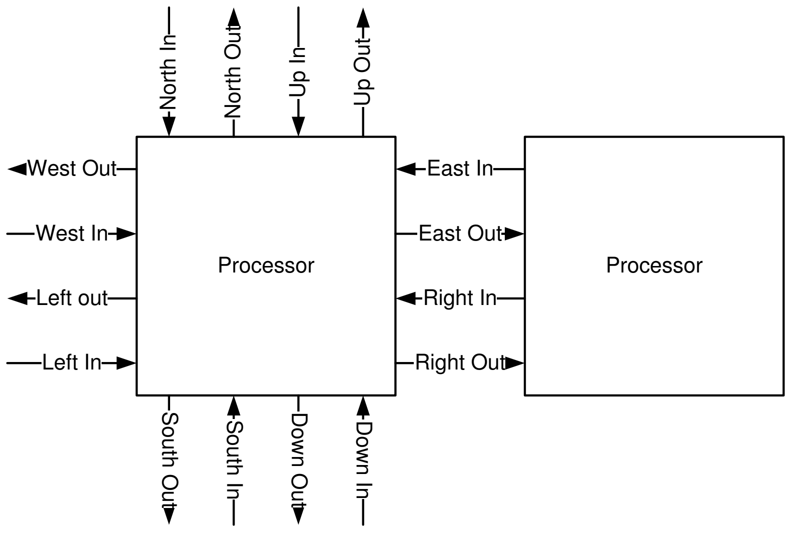


Figure 3.3: Nearest neighbor communication links for AsAP processors

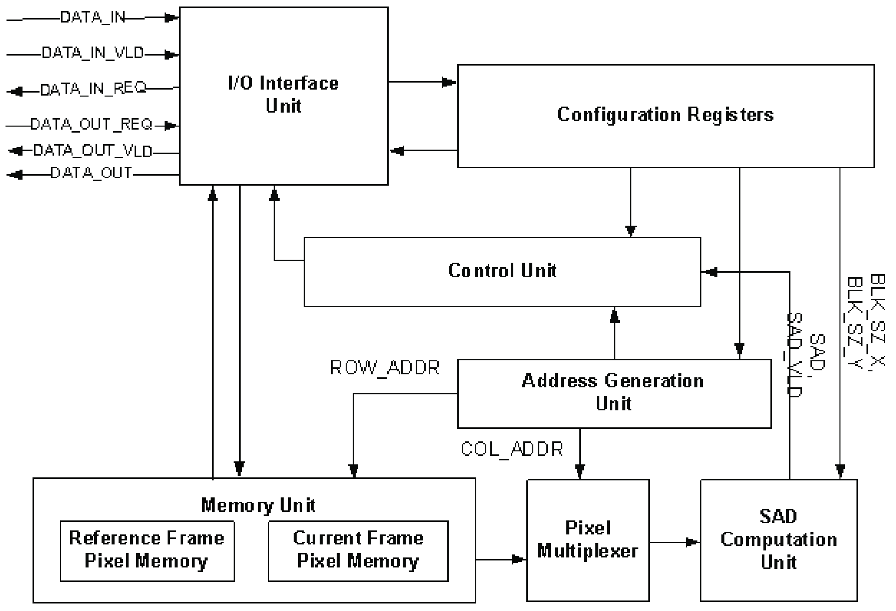


Figure 3.4: block diagram ME accelerator on AsAP2

with two neighboring processors for control. The block diagram for the ME ACC is given in Figure 3.4. The accelerator has its own memory for storing the current macroblock and search window allowing for faster computation.

General Architecture

After the ME_ACC loads the required current and reference data, a set of search patterns is loaded along with the block size used for prediction and the start signal. The ME_ACC will then return the SAD value for that search position to the neighboring processor, this can be continued by sending a continue signal or abort signal once the neighboring processor determines that a sufficient match has been found.

Dedicated Memory

The ME_ACC has two dedicated memory banks one for storing the current macroblock and one for storing the reference search window. The current macroblock memory consist of 2 banks, each 8 8-bit words wide, and with 16 rows, allowing for one 16x16 macroblock. The refer-

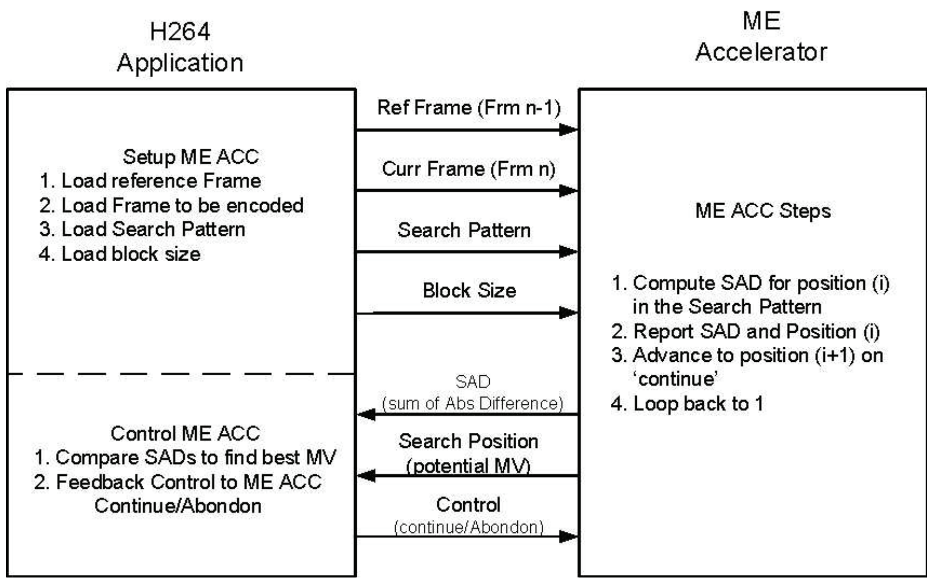


Figure 3.5: Steps for running ME accelerator

ence memory contains 8 banks each 8 8-bit words wide with 64 rows allowing for 16 macroblocks (4x4 macroblocks). The extra row and column in the reference memory can be used for pre-loading of memory to hide latency. For a more detailed description of the memory architecture please refer to Gouri's thesis [11].

Programmable Search Algorithm

The search area and pattern in the ME_ACC are user defined allowing for different types of searches such as full search, 4-step, diamond as well as any custom ones. The accelerator currently allows for 4 sets of search patterns, each with 64 different programmable search locations. During run time the desired search pattern is selected by writing to a register, the indexes in the search pattern are then incremented by writing to the ME_CONT register.

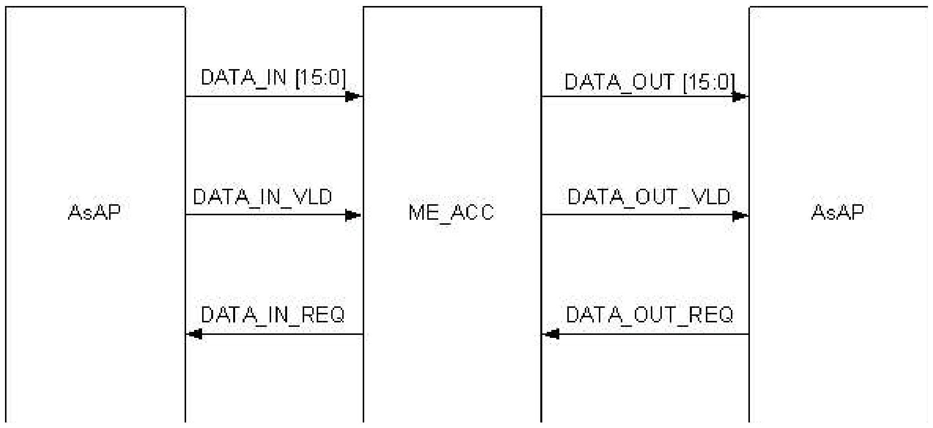


Figure 3.6: Interface for ME accelerator on AsAP2

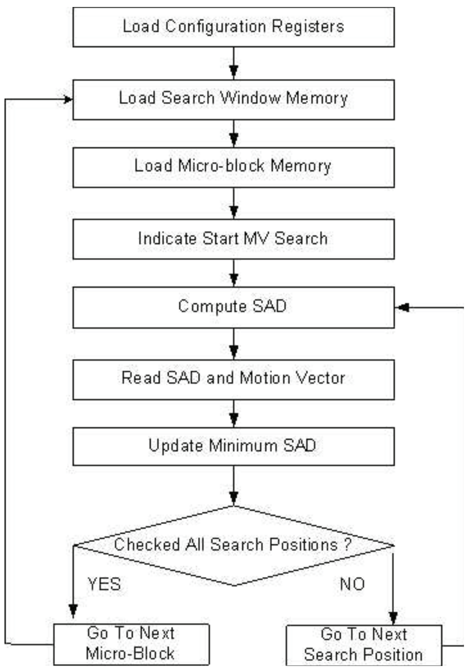

Load Configuration Registers
Load Search Window Memory
Load Micro-block Memory
Indicate Start MV Search
Compute SAD
Read SAD and Motion Vector
Update Minimum SAD
Checked All Search Positions ?
YES
NO
Go To Next Micro-Block
Go To Next Search Position

Figure 3.7: Flow diagram for ME algorithm

Chapter 4

Parallel Programming Tools

A high level C H.264 video encoder was first developed as a reference model for the AsAP implementation using the message passing interface (MPI). Due to the complexity of parallel programming a MPI wrapper was used and is discussed in the following section. The next section deals with issues in converting the MPI/C program to assembly for AsAP.

4.1 Message Passing Interface (MPI)

The message passing interface is a specification for an application programming interface (API) allowing multiple computers to commute with each other while running a single program. However, programming with the MPI syntax can be confusing and time consuming since many of the commands required are simply for communication protocols. Two tools have been developed by Paul and Eric [12] to make this process easier. The first is a MPI wrapper allowing the programmer to essentially use C, and the second is a mapping tool for a visual connection of parallel programs.

4.1.1 Parallel C/MPI Wrapper

The MPI wrapper allows the programmer to write in C with a few key words for designating different nodes such as begin, end, ibuf, and obuf. Once the C code is written a script goes through and add the necessary instruction for making the code compatible for an MPI simulator.

4.1.2 AsAP Arbitrary Mapping Tool

The arbitrary mapping tool allows the programmer to visually see the connections for each parallel program block and connect them making the communication between blocks easier. The tool also follows the AsAP model and can propose a mapping algorithm for mapping the programs onto an AsAP chip. Here since the programs are written in C and not assembly, the primary use of the tool was for a visual mapping of the C blocks.

Figure 4.1 shows the encoder modules in C and Fig. 4.2 shows the low level processors that model those same processors in AsAP assembly. An example mapping of these processors to the AsAP chip is shown in Fig. 4.3, in this implementation however a hand mapping is used for simplicity. Because the encoder is programmed in parts, using the mapping tool, every time a new processors is added, would require the programmer to go back and and change all the processor coordinates by hand. A comparison of the hand mapped encoder to the proposed mapping from the tool is given in the analysis section.

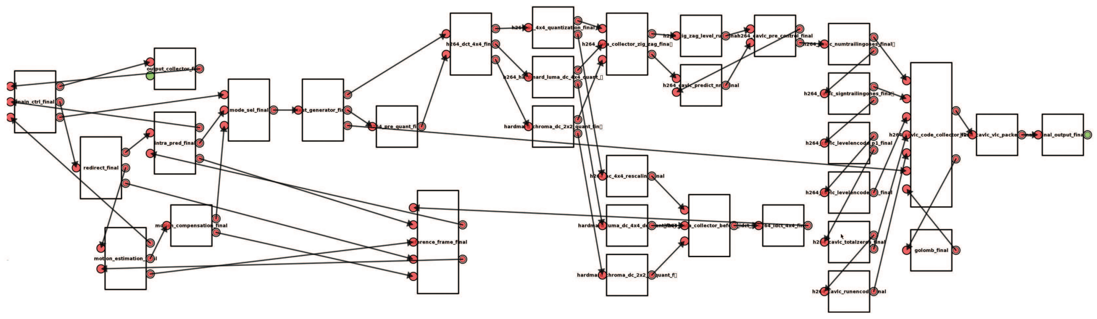


Figure 4.1: Communication links for reference C model in AsAP arbitrary mapping tool

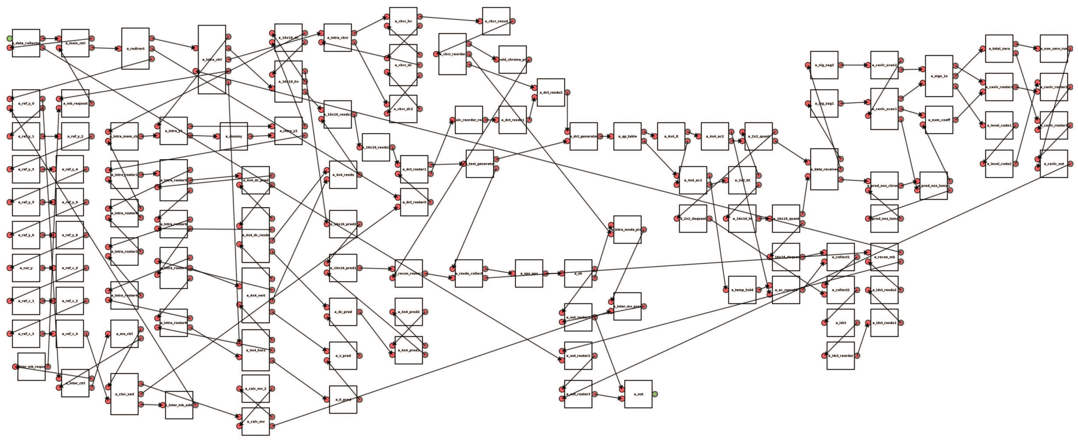


Figure 4.2: Communication links for AsAP processors in AsAP arbitrary mapping tool

4.2 Parallel Programming

Because programming on AsAP is considerably different than C or traditional assembly this section talks about the methodologies used in programming, what had to be changed and altered, what was harder, what was improved, and what are some of the common problems/pitfalls that were encountered in programming this video encoder.

4.2.1 Methodology

Two main differences in programming AsAP vs. other chips or using MPI is the size of the instruction/data memory available and the input limits per processor.

Limited Data Memory - Processors

Video encoding is a highly memory intensive process, from the sheer size of throughput needed for standard quality video to the memory reference needed for prediction, the 128 data memory posed as a great challenge. Because there is only 128 16-bit words of memory, even if the macro block data is packed, it would not fit onto a single processor and would have to be split into at least two, with the luma data packed (two pixels per word) into one processor and the chroma data into another processor. Even so that leaves no memory left in the luma processors for variables using in calculations, hence the memory processors would have to be separate from the computational processors. Data is accessed by using the dynamic configuration memory (DCMem) to determine where in the processor the data resides and passes it along. When more data space is needed (but significantly less than that provided by the big memories) multiple processors can be connected in a loop to form a FIFO like buffer.

Limited Instruction Memory

The small instruction memory available for each processor is fairly adequate for simple tasks, however to perform more computationally intensive task, the programs had to be split up into smaller blocks. This creates more parallelism if the program can be broken up in such a manner that both blocks can be executed at the same time. The challenge is to find good breaking points in the program where branching off to another processor would require little overhead because certain

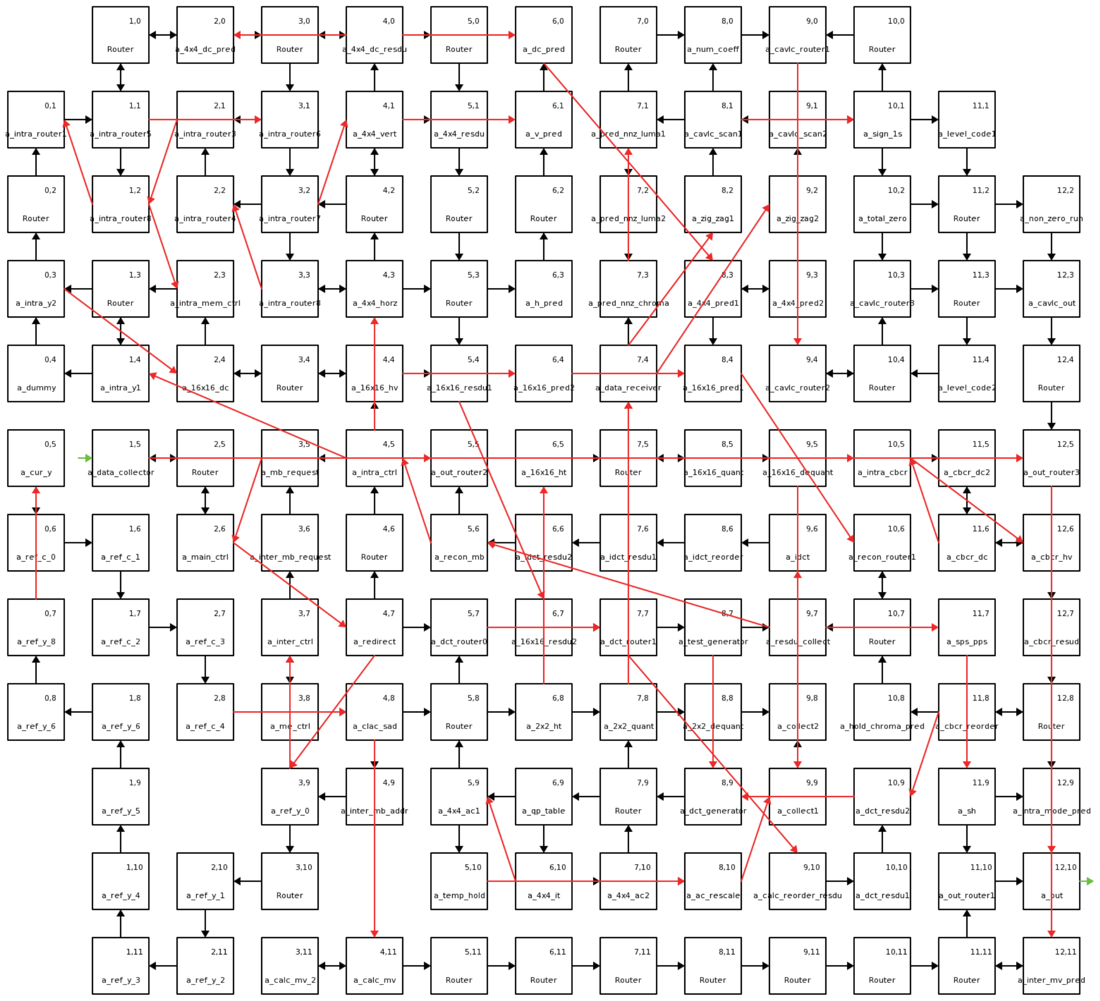


Figure 4.3: Proposed mapping of processors from AsAP arbitrary mapping tool.

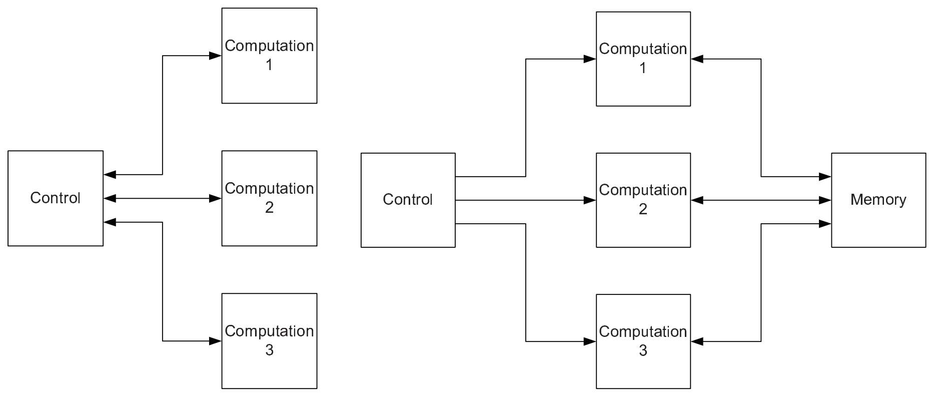
control information and data would be needed by both/multiple processors. Generally it is safe to go to a different processor once you have exited all conditional loops including if statements because there would generally be less data overlap and only a few control values need to be passed along with what was recently computed.

Limited Inputs

Perhaps the biggest difference in programming in AsAP is the limited number of inputs to both the chip and each individual processors. The AsAP2 chip has only one external input and output for off chip communication thus not suitable to control flow operations where the input depends on the output. Because of the limited memory on board the current and reference frames cannot be stored on chip and must be stored off chip in the FPGA, when a processor needed a macroblock, it would send a request signal. Since there is only one output, the request signal and encoded video output must both share this, requiring that control bits be sent to the FPGA for determining where each output should be routed.

Having only two inputs per processor core posed an even greater challenge. Because of the limited instruction memory, many of the modules had to be broken up to smaller parts, and at some later point combined again to re-construct the data as shown in Fig. 4.4(a). A similar problem is where each processor now also requires inputs from multiple sources as shown in Fig. 4.4(b). The biggest challenge however is when there is data and control dependencies between the processors as shown in Fig. 4.5, each processor not only needs to communicate with the control and memory processors, but also pass along data amongst themselves.

In general, three different solutions were used depending which problem type was encountered and what conditions/constraints were needed. For problem 1 the solution in Fig. 4.6(a) is simple and sufficient, using additional processors for routing, adding the inputs together two at a time until all of them could be combines, similar to building a multiple input AND gate from only two input AND gates. The only constraint is that the data flow must be know, which input would data appear at first and how many data points should be taken from each input to ensure proper data flow. Solution 2 can also be used to solve either problem 1, 2, and 3. However for solving problem 3, this method requires all the above processors to stall while waiting for data to return for the requesting processor below it. Interrupts cannot be used since there are only two inputs so it



(a) Multiple Input Problem 1 (b) Multiple Input Problem 2

Figure 4.4: Problems with more than 3 inputs per processor

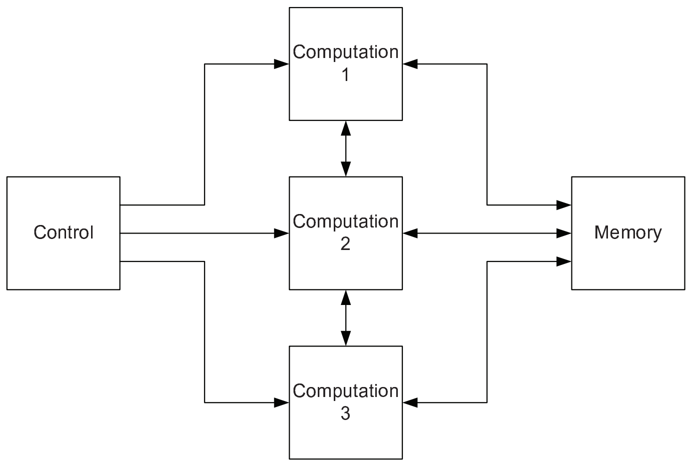


Figure 4.5: Problems with 3 or more inputs per processor and using a feedback path

cannot be determined if the data was requested by the current processor or one below it. Adding in the extra word for control is possible, but now generates the problem of extra instructions used for checking the condition every time a new data is received. Since AsAP does not support traditional interrupts, branch on empty FIFO conditions are used to avoid stalling. These branch instructions must be added often throughout the program to avoid stalling other processors, this would however require additional instruction memory which is not readily available. Solution 2 is most useful when a program is split between multiple processors and one must stall while waiting for data to be computed by another processor that requires additional data that cannot be stored locally. An example is when computing SAD values and the current macroblock data is stored on another processors.

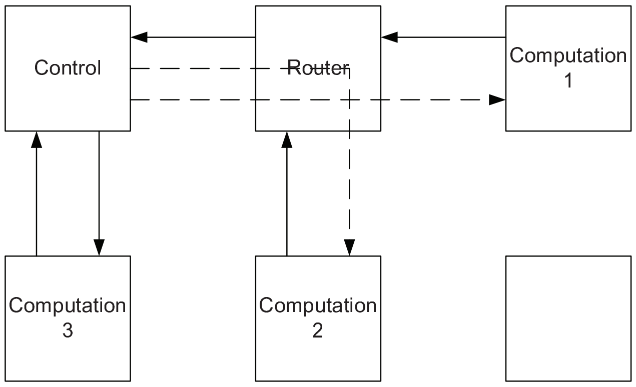
The third solution provides a method for avoiding the latency of solution 2 but requires 3 times the number of processors. In Fig. 4.7 the data path up the computational processor are for passing control information along as well as conditions or complete signals. Routers 1-3 are used for sending request for data, router 1 is the problem, requiring 3 additional routers. Since router 1 already has two inputs from the requesting processor it cannot also receive data from the memory processor, thus requiring routers 4-6 to send back the requested data. Routers 4-6 can also be used to passing information from computational processors 1, 2, and 3 because direct links are not available as in solution 2. Since it is not known when a processor might request data, tags are also assigned to requested data to determine which processor was the original requester.

4.2.2 Pitfalls

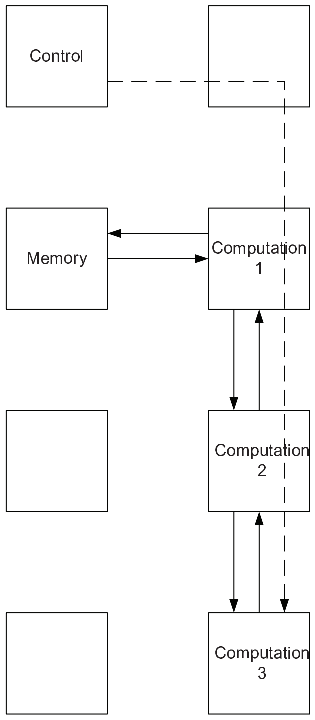
The bulk of debugging time in programming AsAP has been centered around several key issues that were discussed above, primarily the program splitting and processor intercommunication.

I/O Mis-Match

Because of sheer size of the program, making sure that data is processed in the proper order has been extremely difficult. The main processing block in video encoding is the 16x16 macroblock, when data is passed between modules they must generally be sent together, and must all be present before continuing, however because each macroblock contains 384 words they cannot be stored on any single processor, to reduce the number of processors used for simply storing data,



(a) Multiple input solution 1



(b) Multiple input solution 2

Figure 4.6: Solutions for more than 2 inputs per processor

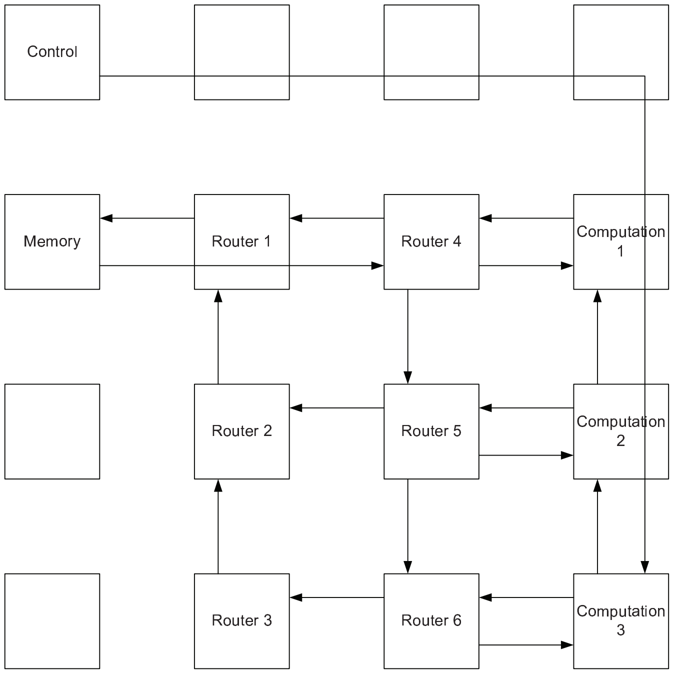


Figure 4.7: Solution for 3 or more inputs per processor with feedback

when data is to be compared such as in comparing SADs for different modes, the comparing processor must wait for all other to complete before continuing, this may cause some processors to stall in the process. To ensure that data is passed along correctly the simplest methods are often used rather than the most efficient algorithms.

Program Splitting Overhead

As mentioned above splitting programs to blocks that would fit inside each processor requires a certain amount of overhead. It is often tempting to try and optimize the code and try to fit it to exactly 128 words, this however leaves no room for changes or improvements. If anything needs to be altered, the entire code must be re-written. The simplest solution is often better, breaking up the program while requiring additional overhead and added complexity to processor communication allows for the most room for further improvements and additions and program scalability

Error Tracing

Error tracing in AsAP has proved to be rather difficult, because it is not often possible to tell directly which processor is causing the problem. Errors were generally due to I/O mismatches rather than computational problems. To solve this the program counter must be looked at for each processor then matched to the instruction to determine the trace-back path for the error.

Chapter 5

Implementation

Because of the complexity of video encoding and the H.264 standard, not all aspects of the standard are implemented in this work. Some key differences from the standard that are not implemented in both the C model and AsAP version are sub-pixel interpolation in inter prediction, inter prediction is limited to 1 reference frame, intra prediction is limited to 3 modes only, and no de-block filtering is implemented. Using a programmable platform however allows for these additional function to be added at a later time. This chapter goes into more detail of the MPI model and AsAP implementation.

5.1 Parallel C Implementation

The parallel C implementation was developed following the guidelines from the standard [2] and the reference software JM version 12.1. In the C programs, code is kept as simple as possible using only standard functions and basic data elements such as integers and arrays for an easy transition to assembly. Since there are no memory constraints in C, all current/reference frame data is stored locally. The .h264 bitstream output of this implementation is compatible with the reference JM decoder software and can be viewed on third party YUV viewing software.

5.1.1 General Overview

The main difference between the MPI model and the standard encoder shown in Chapter 2 is that there is no de-block filtering. Because this version is used as a reference model for developing

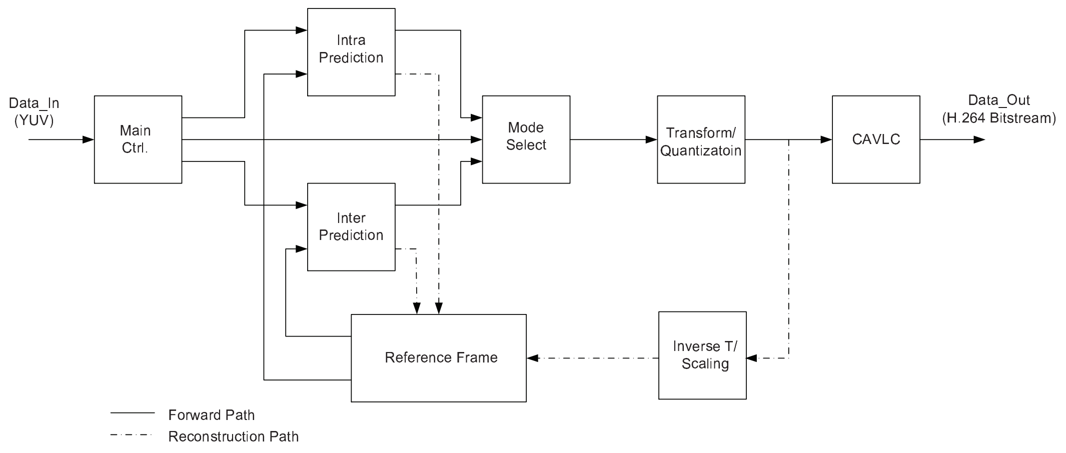


Figure 5.1: H.264 encoder path for parallel C/MPI implementation

the AsAP implementation, it has not been optimized to exploit the parallel capabilities on the MPI simulator, rather the code has been keep in an efficient and easy to debug/alter manner.

5.1.2 Intra Prediction

For simplicity, only vertical, horizontal, and DC modes are used for both luma and chroma blocks, please refer to section 2.2.3 for a more detailed description of these prediction modes. The best mode is chosen directly after computing the SAD for each block and prior to integer transform. Prediction modes are computed as described before in section 2.2.3 and sent directly to the output. Because there are no memory constraints all data is stored locally in this module and passed along to neighboring blocks when complete. Latency is not a factor in this version so SADs for each prediction mode are computed sequentially for easy coding.

5.1.3 Inter Prediction

Motion estimation on AsAP is done via an accelerator, since this is not modeled in C, a simple full search algorithm is used with the smallest partition being 8x8 with no sub-partitions. Every position within the search window is calculated and compared with the previous position to determine the best match for that block. The MV is then saved along with its SAD for comparison with other partition sizes. Inter prediction is done in two blocks where the first block computes the the SAD and motion vectors for each block partition and the second block, motion compensation computes the residue for each macroblock. In the motion estimation stage the residue data is not saved due to the large amount of space needed to store every possible search position when using the full search algorithm and is recalculated in the motion compensation stage. Prediction modes for inter prediction are encoded at the output prior to bit-packing using a table for Exp-Golomb coding since only a four modes are used.

Motion Vector Prediction

Once motion vectors have been computed for the current macroblock, a predicted motion vector is computed as described in section 2.2.4 to get a difference motion vector to encode and transmit. Motion vector prediction in this MPI model supports the 4 main block partitions used in prediction only. Once the difference motion vector is computed, it is encoded using Exp-Golomb coding as described in section 2.2.7 with the range of code numbers from 0-126 allowing for motion vectors in the range of +/- 63, large enough to support any motion vector for a 48x48 (3x3 macroblock) search window.

5.1.4 Integer Transform & Quantization & Entropy Coding

The integer transform, quantization, and entropy coding process is implemented as described in section 2.2.7. Because they must be the mirror image of what is used in the decoder little flexibility is provided in encoding options. The transform and quantization process is a direct process where each block must wait till the preceding block is complete before continuing and is not parallelized in this implementation.

5.1.5 Network Abstraction Layer (NAL)

The NAL for the MPI-C implementation has been mainly hard coded since only a baseline encoder is implemented and many of the extra functions are not used. The SPS and PPS have been entirely hardcoded with the exception of the picture width and height in the SPS which are computed for every new sequence. The slice header has also been hard coded with the exception of the slice type, frame number, and QP value and is resent for every frame/slice. The NAL module also packs the output data into eight bit format to be stored in the .h264 bitstream file. Data is read bitwise into a large buffer, whenever the buffer contains more than 8-bits of data, the first 8-bits to be stored are sent out one word.

5.1.6 Reference Frame Reconstruction

Because our MPI simulator does not have the ability to read and write to an open file, all inputs must be previously stored in an data file. The reconstructed macroblocks are stored within a module and sent to the intra/inter prediction modules when requested.

Inverse Transform & Quantization

The inverse transform and quantization process is done after the forward transform and quantization, once complete they are re-ordered to match the original macroblock mapping and sent to the reconstruction model. During the prediction process, the modes and residue values of the predicted macroblock have been previously sent to the reconstruction module and are added with the reconstructed residue values giving, a nearly duplicate copy of the original input, and exact match the output of a decoder.

5.2 AsAP Implementation

Using the AsAP multi-core layout, the encoder can use implemented using a fine grained sub-macroblock partition to exploit the greatest amount of parallelism. Using the MPI code as a reference module, the modules are further partitioned to individual processes.

5.2.1 General Overview

For testing purposes and memory storage, the AsAP chip is attached to a Virtex-5 board kit, which programs the AsAP chip upon power on and serves as main memory. All computation is performed on AsAP however and the FPGA board is used mainly for I/O. Figure 5.3 shows a high level diagram of the main blocks in the H.264 encoder, these are similar blocks to the MPI implementation with the exception of the inter prediction module which consists of a motion estimation accelerator and a motion compensation module.

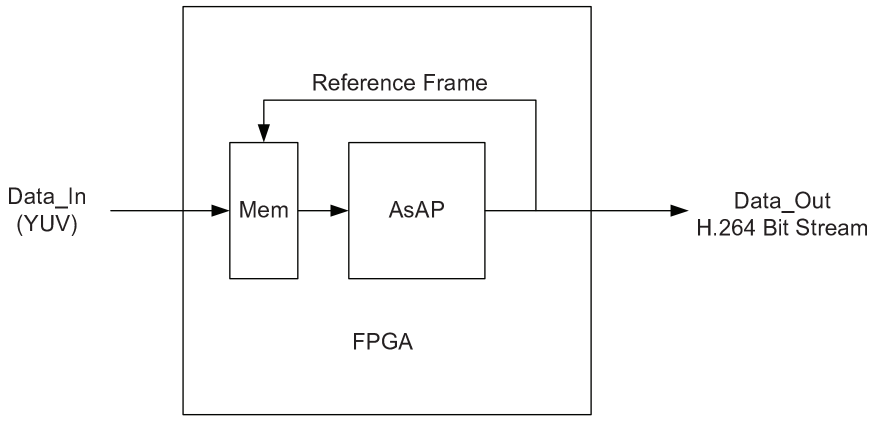


Figure 5.2: Block diagram of FPGA and AsAP connections

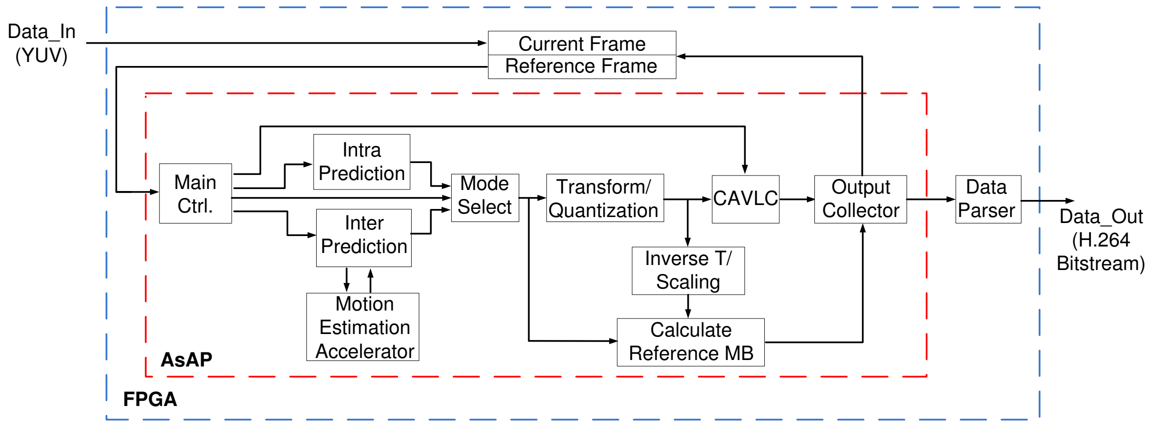


Figure 5.3: H.264 video encoder path for AsAP implementation

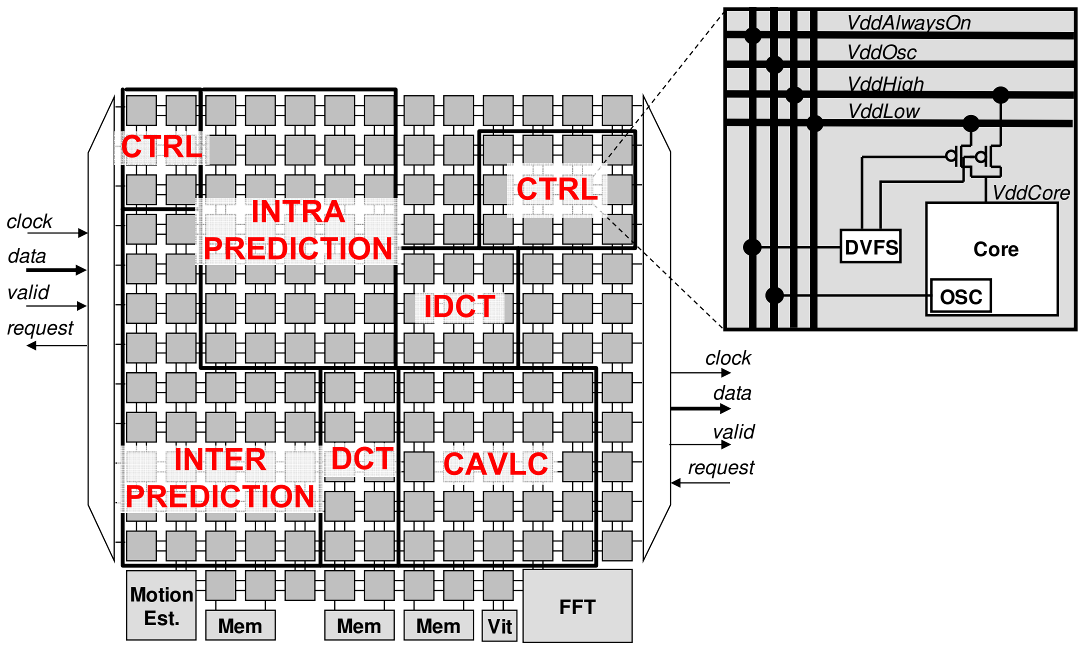


Figure 5.4: Partition of H.264 modules on AsAP chip

5.2.2 Memory Organization

Three memory intensive task in the H.264 encoding process are the current/reference frame management, motion vector management, and non-zero coefficient management. They arise from the fact that encoding is based not only on the current macroblock but also previously encoded ones.

Current/Reference Frame Management

Each frame being encoded must be buffered to ensure that no data is lost, because of the size of a picture frame, this data is stored off chip in FPGA memory, this also allows for future work on larger than 1080p resolution pictures without any changes to the AsAP programs. After each frame is encoded, it must also be saved for motion vector prediction in the next frame. The FPGA memory is divided into three banks as shown in Fig. 5.7. Bank 0 holds the current frame, bank 1 holds the reconstructed current frame, and bank 2 holds the previously reconstructed frame. After each frame the bank pointers are incremented so that the reconstructed frame that was in bank 1 now becomes the reference frame, data is read into bank 2, and the reconstructed data is in bank 0.

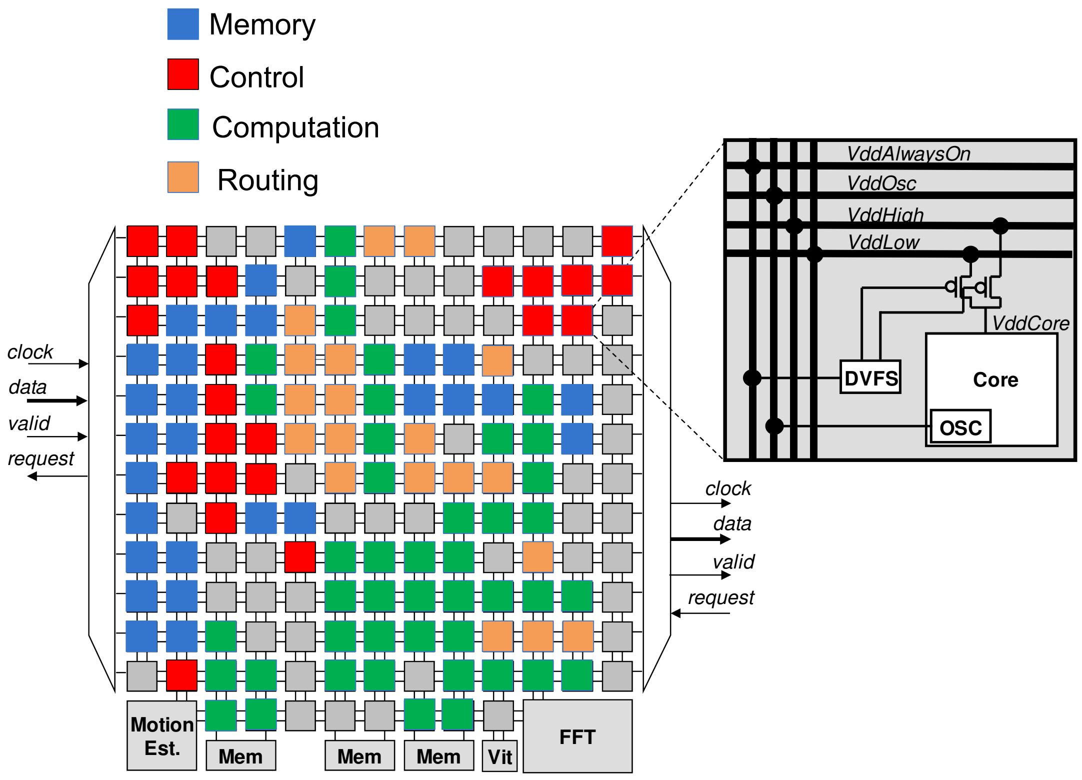


Figure 5.5: Partition of processors type on AsAP

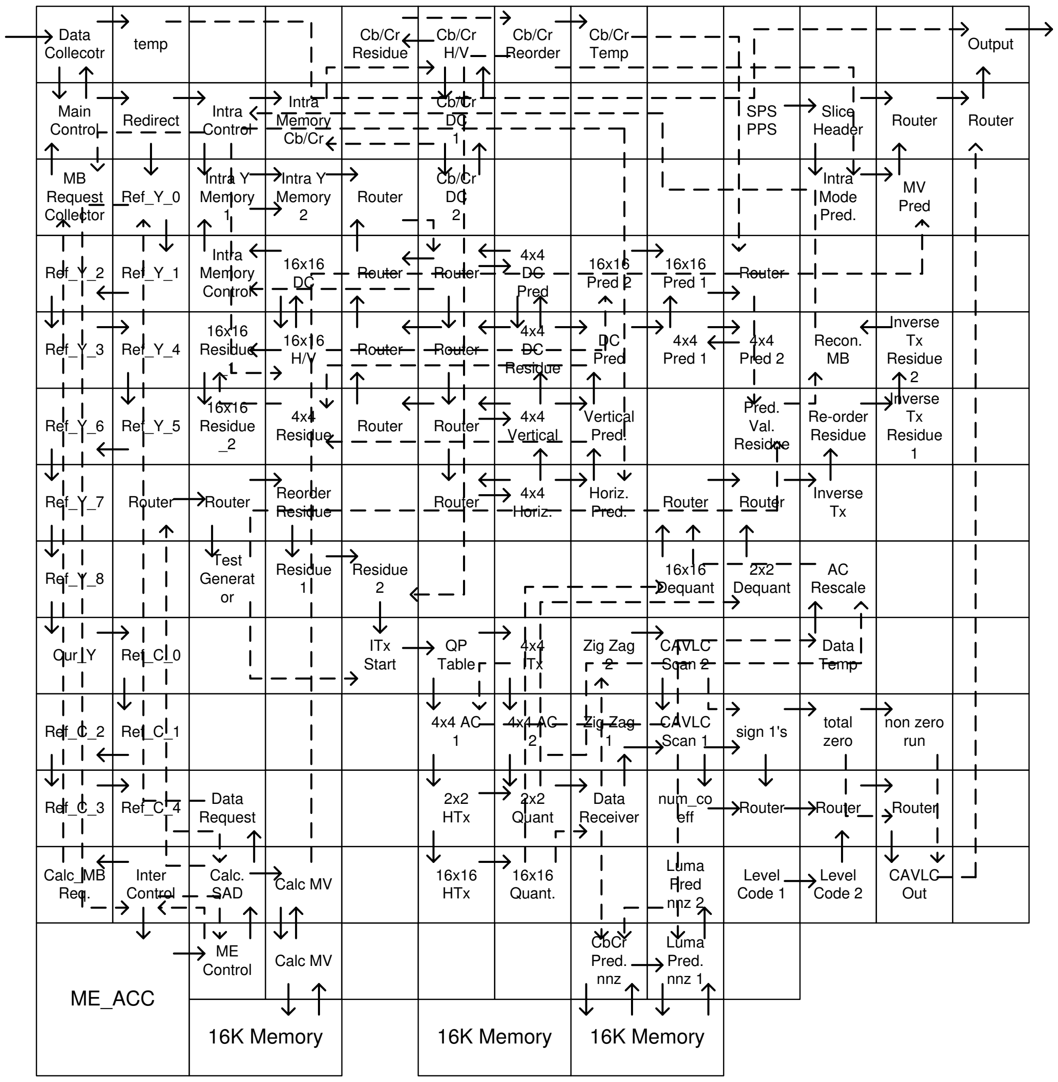


Figure 5.6: Links for communication between AsAP processors

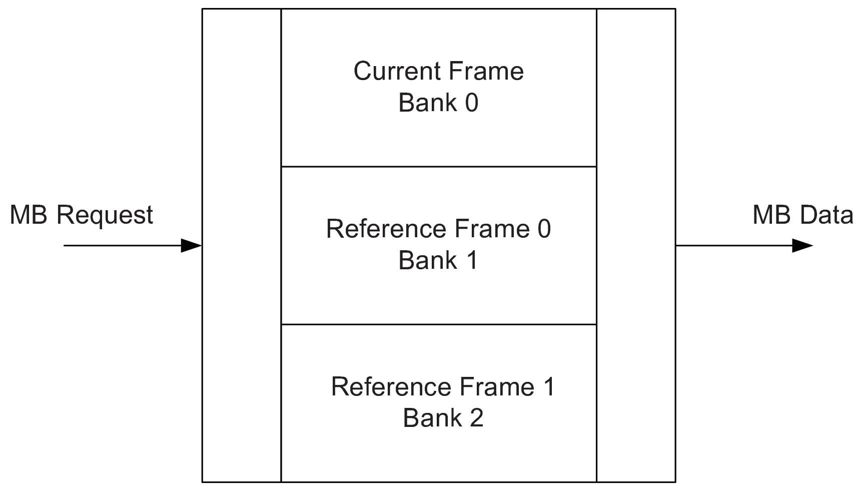


Figure 5.7: Current/reference frame managment of AsAP implementation

Macroblock Management

Within the FPGA memory, data is stored according to macroblocks as shown in Fig. 5.8 with luma data first followed by chroma data. When a data request is make all 384 bytes of luma and chroma data are sent at once. The only logic in the FPGA is a small memory controller for where data should be stored. This can be implemented by using simple counter to keep track of the start address for each macroblock.

Motion Vector Management

The H.264 standard supports sub-partitions of blocks for inter prediction, with two motion vectors per block this becomes a possible maximum of 32 motion vectors when using the smallest partition size(16 4x4 blocks). For motion vector prediction the preceding row of macroblock motion vectors must be saved. For 1080p-resolution, this would be a maximum of 3840 words. For simplicity all motion vectors for a macroblock are save even though only motion vectors that are on the bottom of a macroblock are needed, hence the mode for each macroblock must also be stored adding an additional 120 word for 1080p support. The block mode and motion vectors are stored in BigMem with the first 500 address allocated for block mode and the remaining address for motion vector data. The motion vector data is addressed in increment of 32 words, hence the MV for the first macroblock is stored starting at address 500 and the MV for macroblock 2 is stored starting at

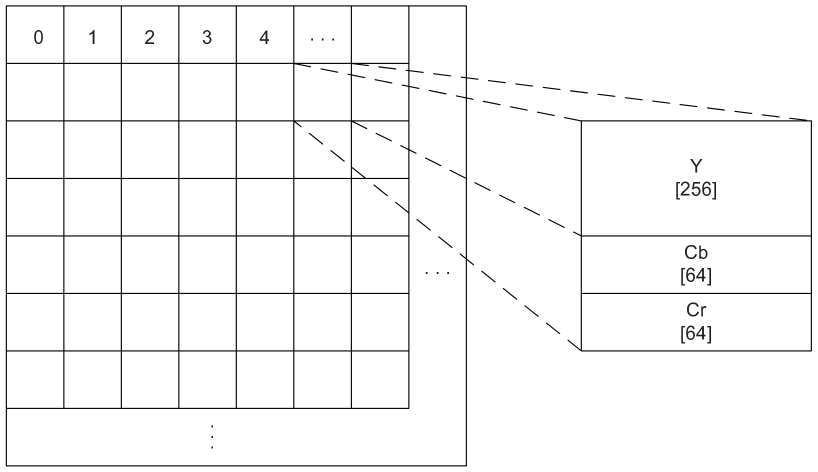


Figure 5.8: Orginization for macro block storage in memory

address 532 and so on.

Non-Zero Coefficient Management

Similarly to motion vectors, the number of non-zero coefficients must be predicted in the CAVLC using the above and left previously encoded data. Because the CAVLC process is performed on 4x4 blocks, at least 4x120 memory addresses must be reserved for a frame of 1080p resolution requiring the use of BigMem. Because the needed address space is much smaller than the 16 KB available, all the number of non-zero coefficients for each macroblock in a frame and stored for easy addressing.

5.2.3 Control Logic

One of the greatest challenges of partitioning a program over such a large area is controlling the flow of data between processors. Ensuring that data is present when needed, and buffered when un-used is vital in preventing dead lock. Two main areas for control logic are macroblock control and I/O operations for AsAP.

Macroblock Control

Video encoding is done on a macroblock basis, however to ensure that the decoding path produces the same data as the encoding path, prediction can only be done with previously encoded blocks. For intra prediction this requires each macroblock to go through the intra prediction process, integer transform, quantization, scaling, inverse transform, and reconstruction before the next macroblock can be predicted. At each step proper control information must be present to ensure accuracy. The chroma prediction process is much faster than luma prediction and the predicted value used must be buffered prior to being sent to the reconstruction blocks to prevent a dead lock situation at the integer transform. Because input FIFOs to each processor are only 64 words deep and chroma data is 128 words for both Cb and Cr components, the chroma prediction processors will halt and not send data to the integer transform while waiting the reconstruction processor to complete. However the reconstruction processor must wait for inverse transformed data before reconstructing the reference frame thus causing a dead lock situation.

Basic macroblock and frame information is also sent along at each stage to ensure accuracy and increase code reuse. Parameters such as frame width, frame height, macro block width (frame width /16), macro block number, encoding mode (intra/inter), block mode, and macNo % macWidth (macro block number mod macro blocks width) are used at nearly every stage and transmitted to save limited IMEM having to used to recompute these values. Many processors can start some initial computation without all of the current data being present, this however requires that the control information be broadcasted to many processors via long distance interconnects creating an additional mapping issue.

External Memory Request & NAL

Because AsAP has only one input and one output link for communication with the FPGA, both macroblock requests and the encoded .h264 bitstream must come from the same output. For this the protocol in Table 5.1 is used. On the FPGA side a small block must also be present to parse the data depending on the control bits, this however is rather simple and can be done in a few lines of code. The protocol given will work with frame sizes up to 1080p, beyond this however each request will need to consist of multiple 16-bit words in order to provide addressing for a greater

Bit Field	Field Name	Description
15:14	Control Bits	00: Write to output, payload contains H.264 output 01: Read from memory bank 0 payload contains macroblock number requested 10: Read from memory bank 1 payload contains macroblock number requested 11: Write to memory bank 2, payload contains pixel data
13:0	Payload	For control bits 00 and 11: only bits 7:0 are used For control bits 01 and 10 13:0 are used to request up to macroblock number 1023

Table 5.1: Output format for AsAP H.264 video encoder

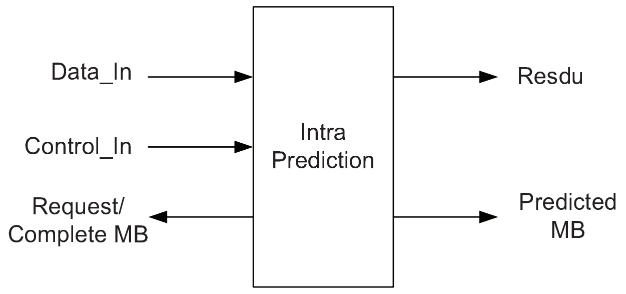


Figure 5.9: High level diagram of intra prediction in AsAP

number of macroblocks.

5.2.4 Intra Prediction

The intra prediction process although generally used only once per sequence constitutes a rather large amount of computation. Figure 5.9 shows a high level block diagram for the intra prediction module. Data_in and control_in contain information for the current macroblock being predicted, the request_MB signal is for requesting neighboring macroblock used for prediction. The residue output goes to a re-ordering processor for the integer transform process and the predicted macroblock goes to the reconstruction processor to be added to the reconstructed residue data.

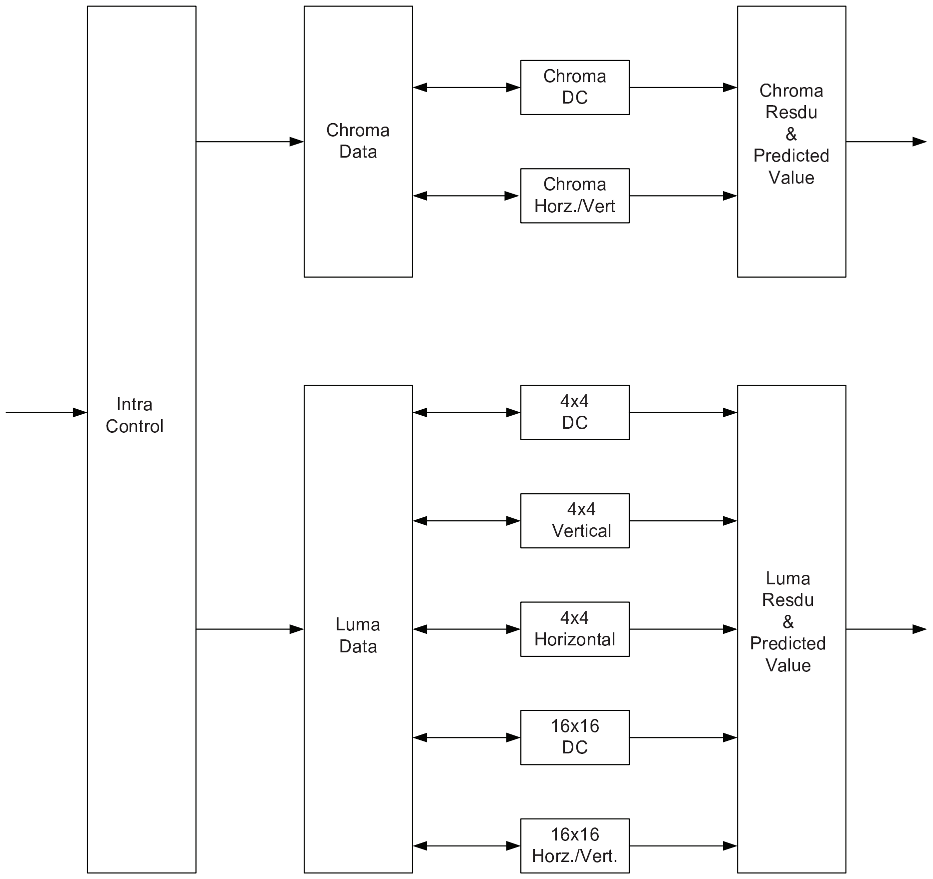


Figure 5.10: Block diagram of Intra prediction in AsAP

Parallelized Luma Prediction

Luma prediction can be done for both 16x16 and 4x4 blocks modes at the same time, furthermore each prediction mode can also be done in parallel. In this implementation a processor holds the current data to be predicted which is read by all processors doing prediction. Figure 5.10 shows how these blocks can be divided into parallel task. Each processor has a local copy of the above and left macroblock and accesses the current data through a shared network.

When mapped to individual processors, half the processors are used for routing as shown in Fig. 5.11. As explained in chapter four these routing processors are required to over come the input limitation of each processor and to limit the number of processors being used for memory storage. Here two processors are used to store luma data (256 8-bit words), this can be done in one processor if data is compacted to 16-bit words, however two processors are used to reduce addressing complexity.

Parallelized Chroma Prediction

Similar to luma prediction chroma prediction can also be parallelized as shown in Fig. 5.10. Because there are only 64 Cb and 64 Cr data only one processor is needed for storage while 3 are used for computation. To reduce the number of routing processors needed, data is automatically sent to the DC mode computation processors for computing the SAD for each mode and requested individually on the second pass for computing residue. In the first pass only Cb data is used for computing the prediction mode. In the second pass both Cb and Cr are used for residue calculation. Since chroma prediction is much smaller and faster than luma prediction the extra time used by the chroma prediction processors to compute the SAD first then residue the second time is acceptable.

5.2.5 Inter Prediction

Inter prediction is a computationally intensive task that is done on the majority of frames/slices. To provide higher throughput AsAP uses a programmable motion estimation accelerator for this. A high level diagram of the ME_ACC interface with neighboring processors is shown in Fig. 5.13.

Search Window Management

The ME_ACC is capable of holding a 4x4 macroblock region for the search window. To speed up the prediction process however only a 3x3 search window is used. The fourth column can be used to pre-load data for the next prediction cycle, however the addressing scheme for this becomes rather complex and is not easily done in a single processor. Here a new search window is loaded from main memory for every macroblock. Although this is time consuming, it allows for much simpler code. Each macroblock when read into the ME_ACC is also stored locally on one of 11 processors so that only 3 new macroblocks need to be read in from main memory.

Diamond Search Algorithm

A modified diamond search algorithm is use for all block sizes. The modified algorithm uses only 5 search points as opposed to the nine points generally tested, and is repeated 4 times

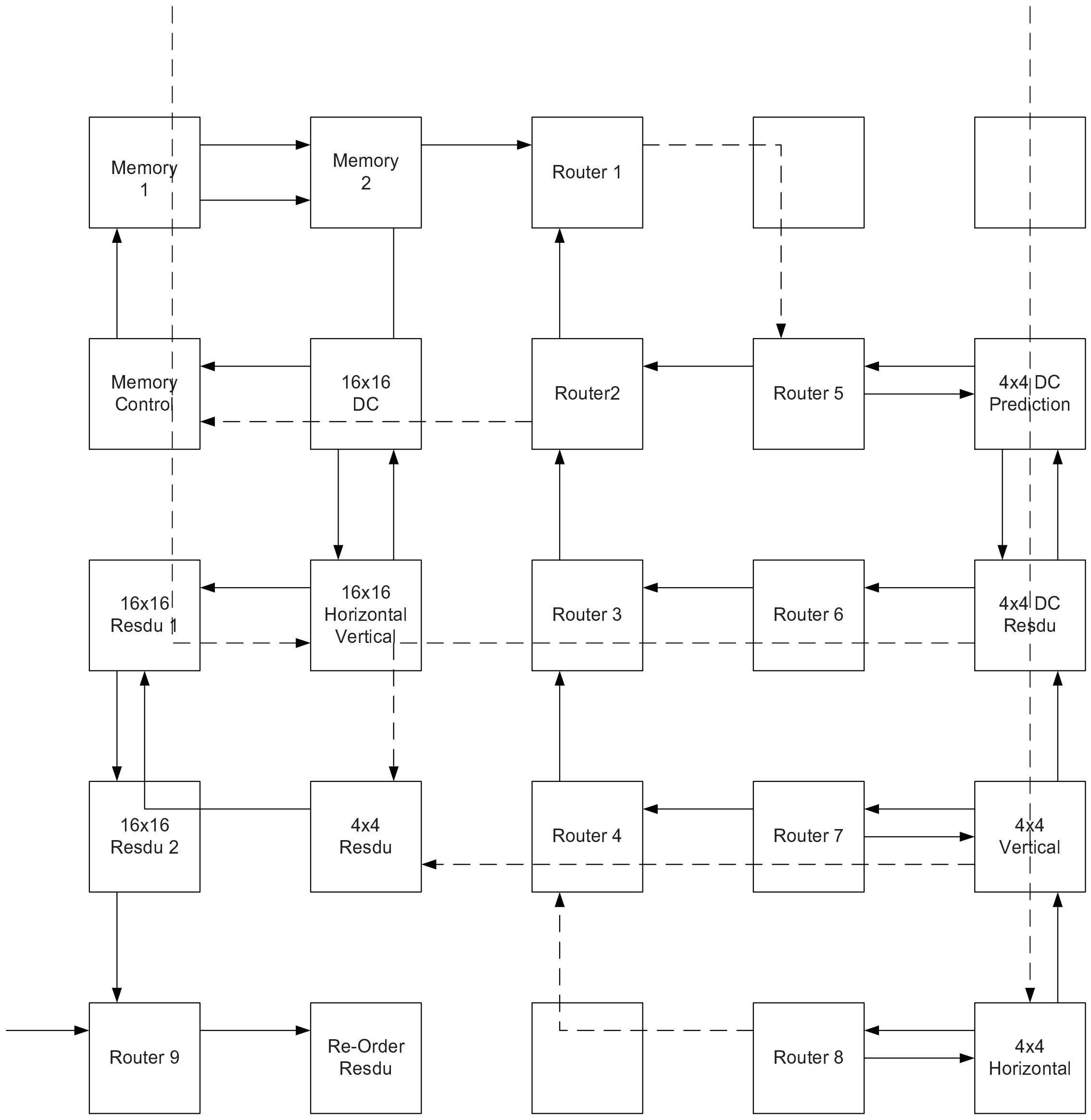


Figure 5.11: Processor mapping for intra prediction in AsAP

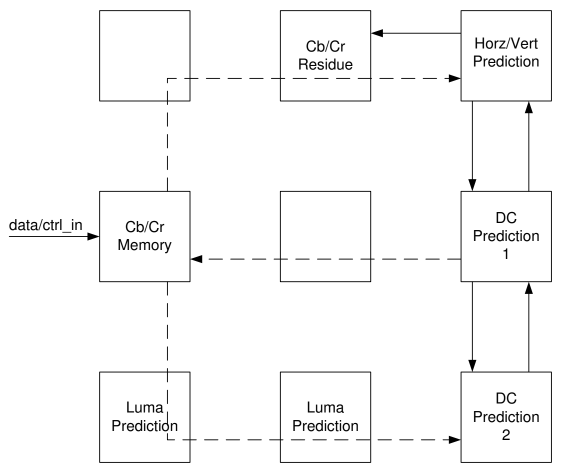


Figure 5.12: Layout of intra chroma prediction processors on AsAP

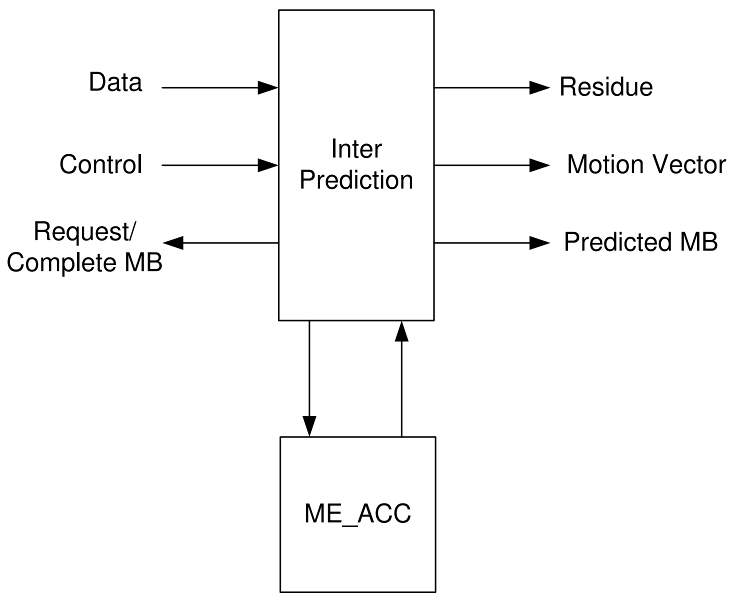


Figure 5.13: High level diagram of inter prediction in AsAP

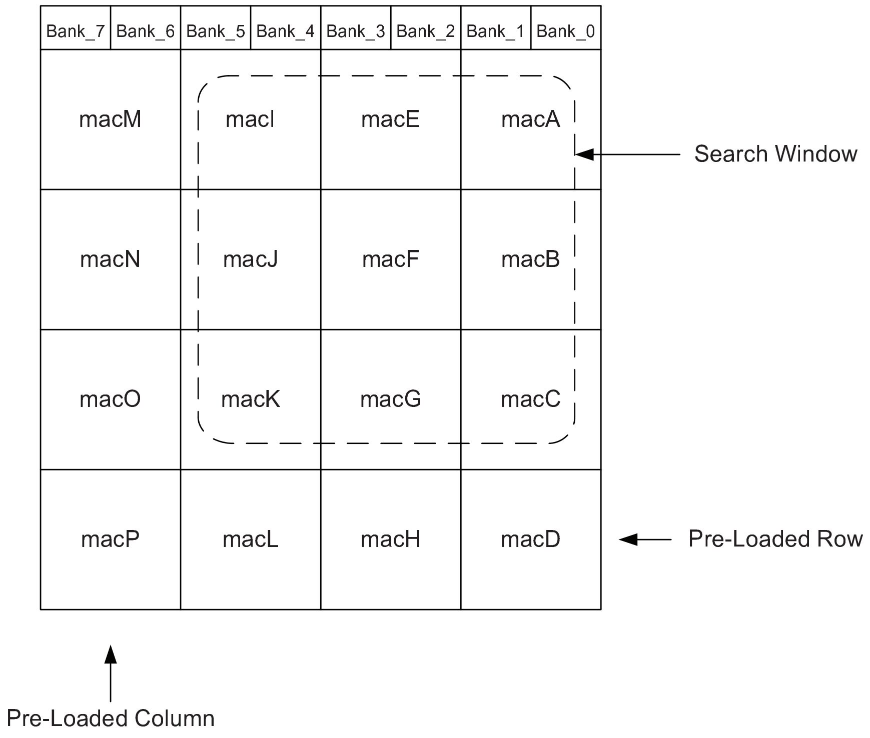


Figure 5.14: Reference frame macroblock allocation in ME_ACC

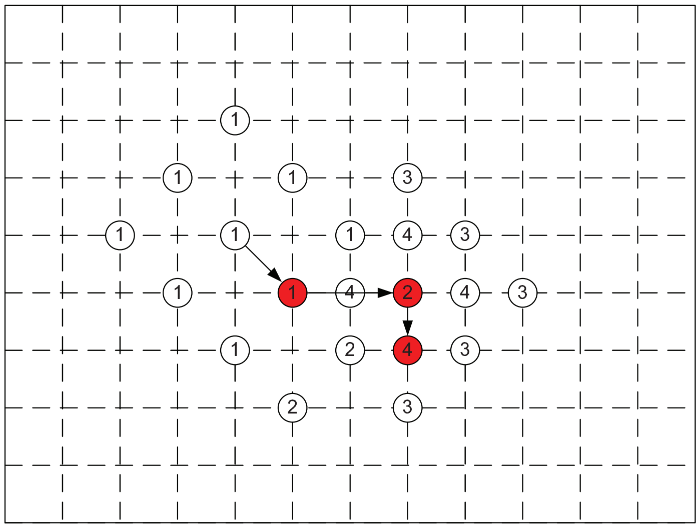


Figure 5.15: Diamond search pattern for motion estimation

to find the best match. Although this process is not as accurate as a full diamond search the only drawback would be slightly higher entropy values to be encoded. This is generally not significant unless there is a great difference in motion between frames, generally the majority of a frame is constant or near constant.

Residue Calculations

Once the best set of motion vectors are computed, they are sent to a residue calculation processor. The data used for this prediction is read from the 11 processors that hold a mirror copy of the ME_ACC memory. A block diagram of this can be seen in Fig. 5.16 along with the AsAP layout is shown in Fig. 5.17.

5.2.6 Integer Transform & Quantization & Entropy Coding

Integer transform and quantization are done in similar fashion to the MPI-C model with the exception that the integer transform path is replicated once with each path taking half of the blocks to be encoded. Figure 5.18 shows the block diagram for this and Fig. 5.19 shows the processor layout.

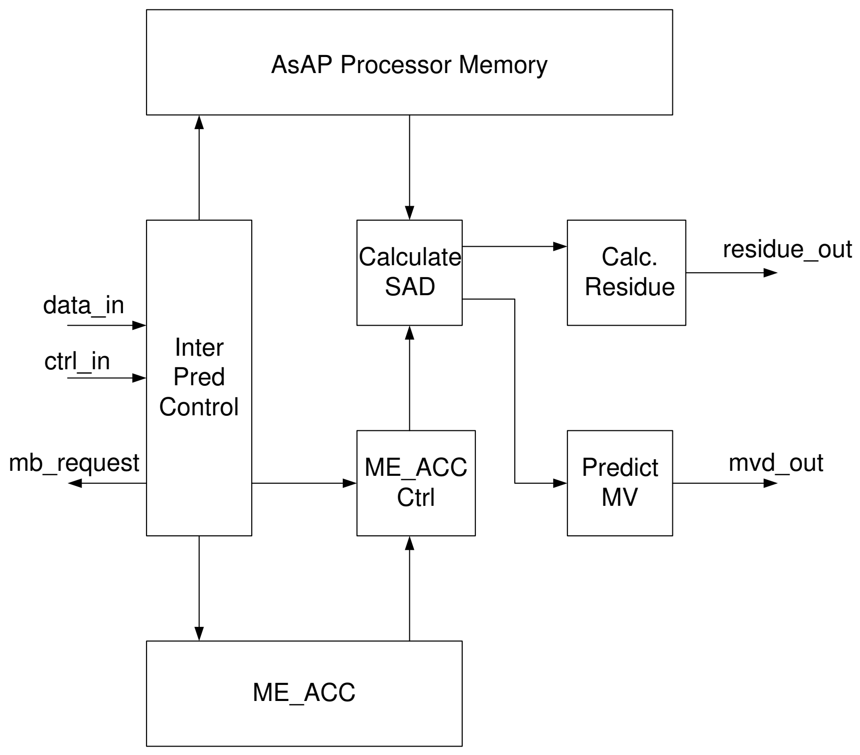


Figure 5.16: Block diagram of inter prediction module in AsAP

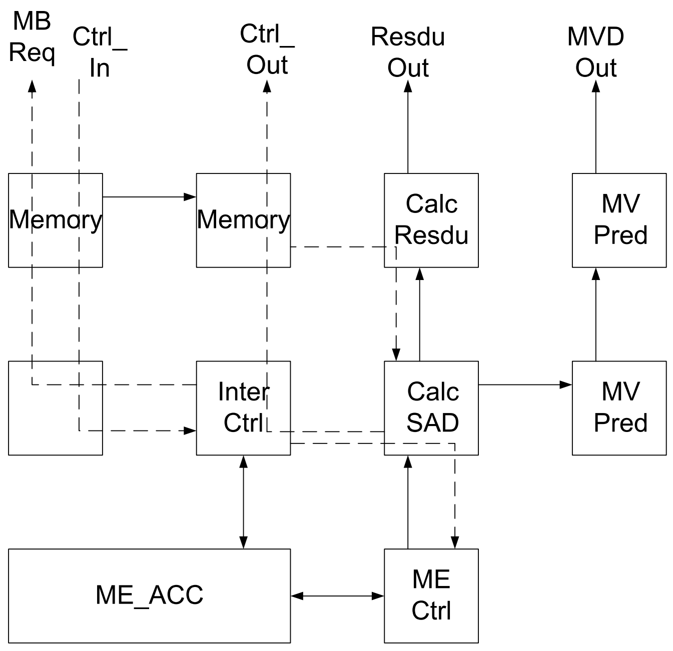


Figure 5.17: Layout of inter prediction module in AsAP

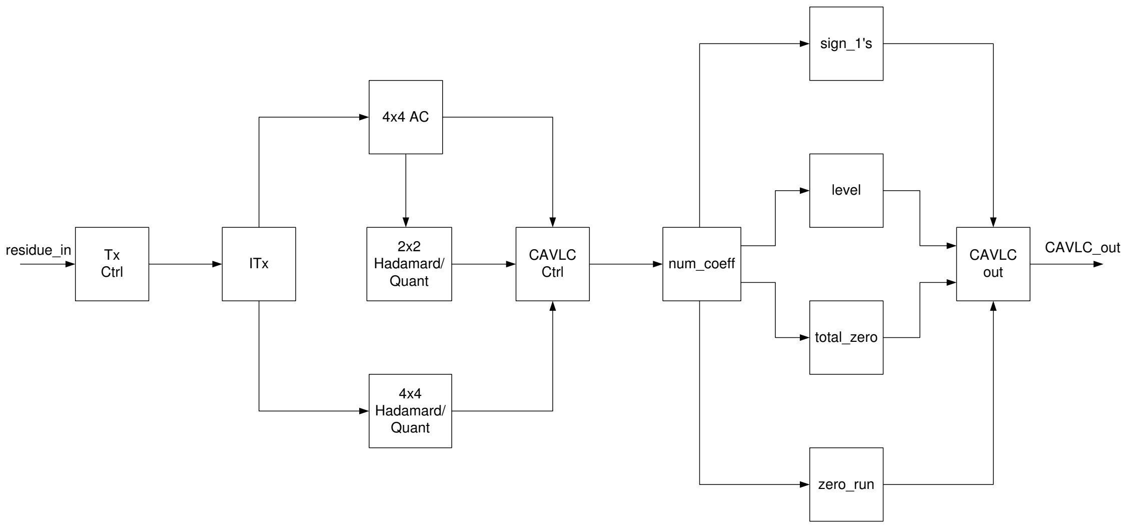


Figure 5.18: Block diagram of integer transform and CAVLC

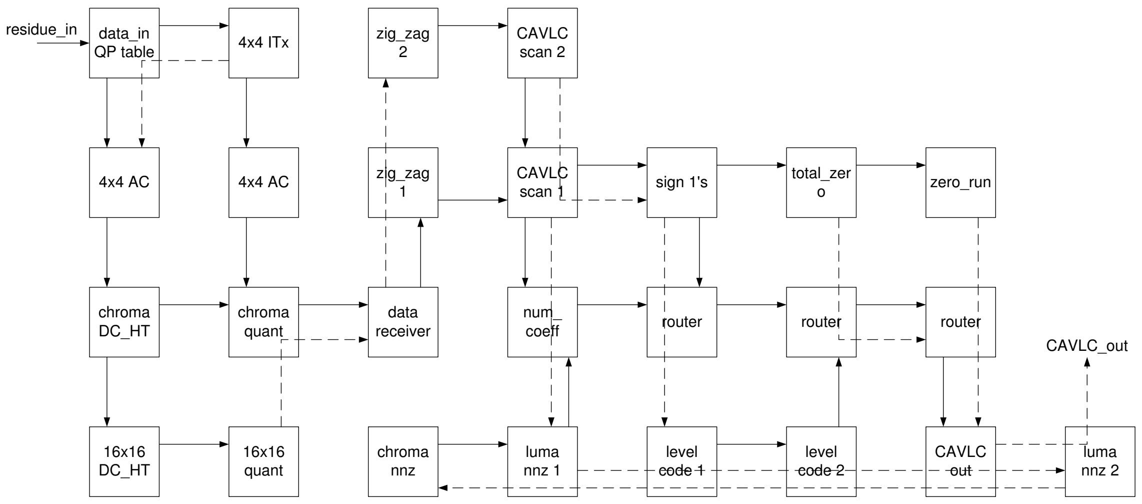


Figure 5.19: Layout of integer transform and CAVLC modules on AsAP

5.2.7 Network Abstraction Layer (NAL)

The NAL layer is handled by the processor network in Fig. 5.20. The algorithm for each is again similar to the MPI-C model with the exception of the final processor which must also handle macroblock request signals.

5.2.8 Reference Frame Reconstruction

Reference frame reconstruction is done similarly to the reference MPI-C model, predicted data is added to re-scaled transformed data and sent to main memory for storage. Figure 5.21 shows the high level block diagram where Fig. 5.22 shows the AsAP layout.

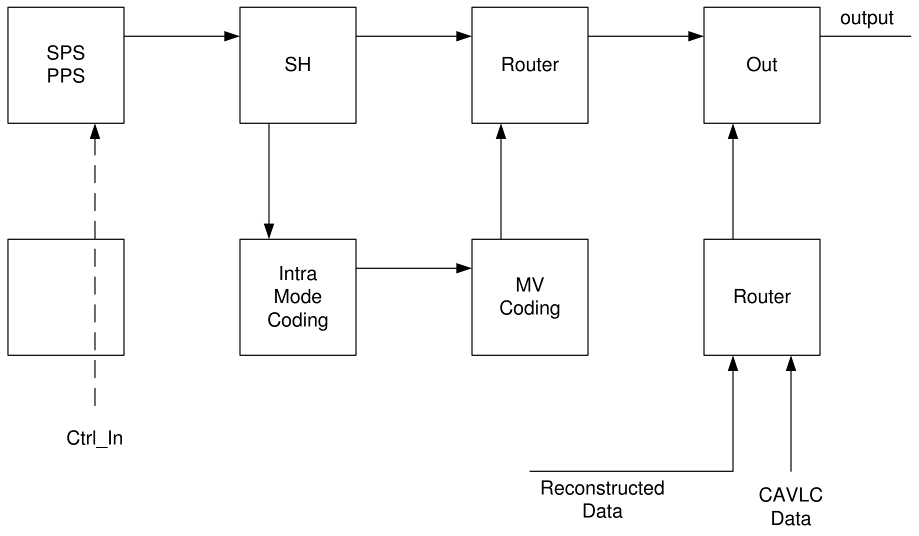


Figure 5.20: Layout of header module on AsAP

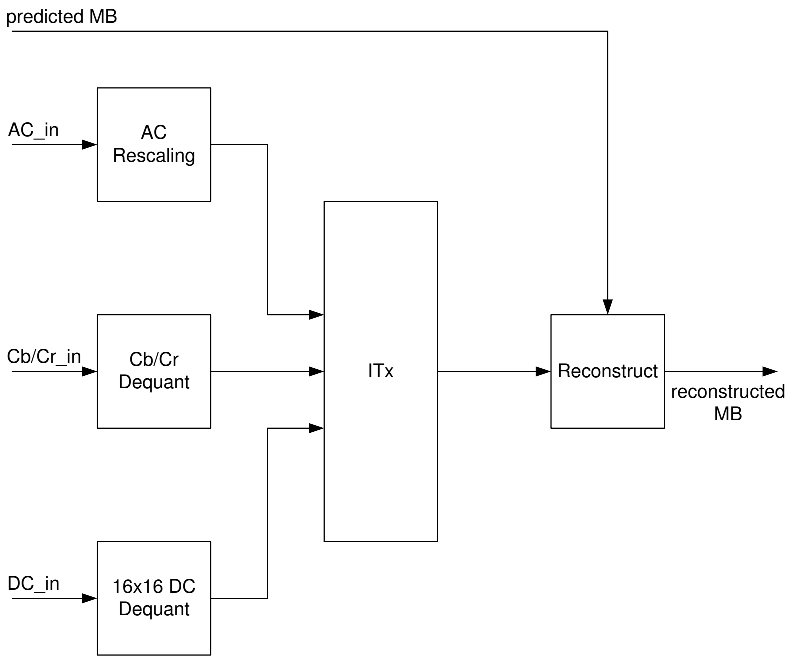


Figure 5.21: Block diagram of reconstruction module

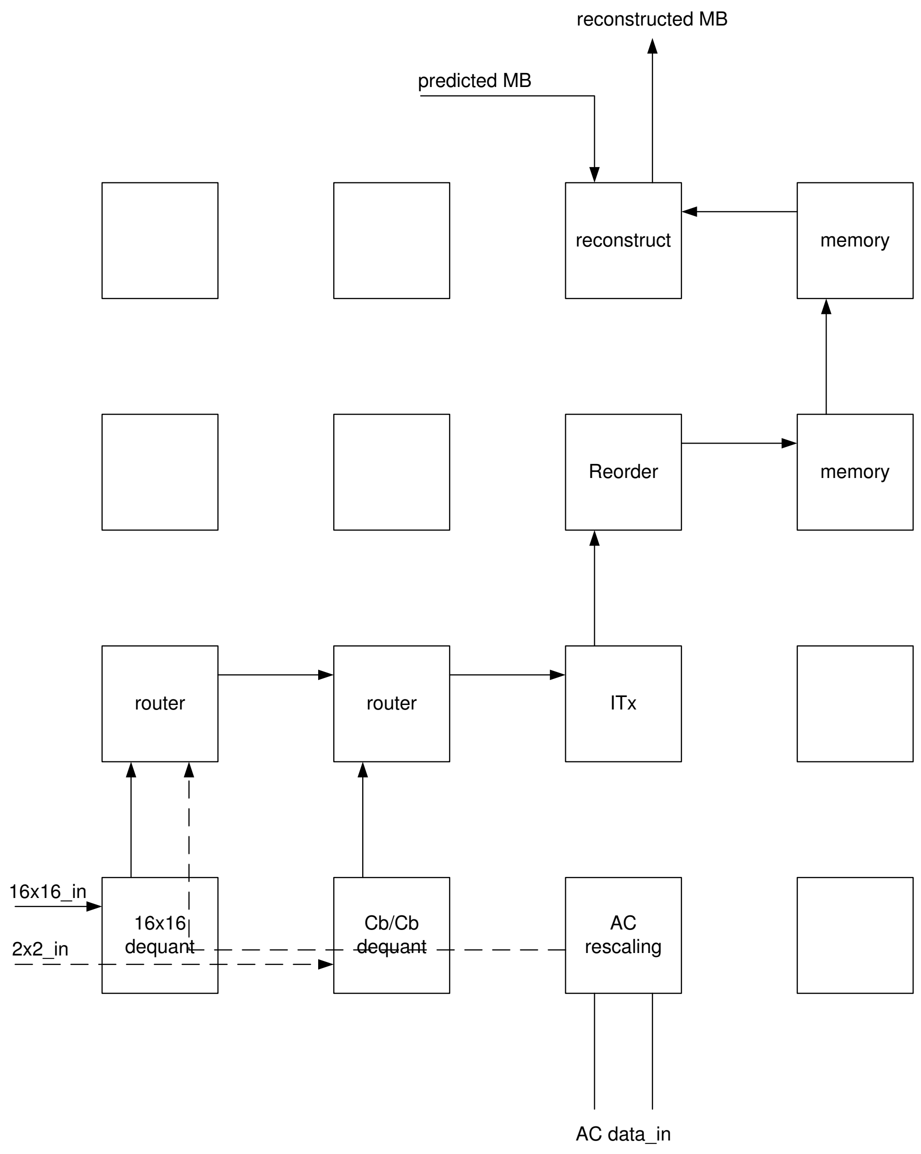


Figure 5.22: Layout of reconstruction module on AsAP

Chapter 6

Results and Analysis

Initial results for the implemented H.264 encoder are gathered from simulation of the AsAP chip in SimVision and NCVerilog. Results and data for analysis are gathered using assembler functions that were able to measure the energy, traffic, and activity of each processor. Section 6.1 presents the metrics used for testing and analysis, section 6.2 gives a comparison of the encoder with other implementations, and section 6.3 gives an analysis of the encoder on the AsAP chip.

6.1 Metrics for Testing and Analysis

For an accurate analysis of the encoder on the AsAP chip, the following metrics have been considered:

- Program Size: The program size (instruction memory) and data memory of each computational and routing processor has been collected for analysis. Processors which are solely used for long distance interconnects have been ignored.
- Energy: The energy of each processors is measured. The energy measurements consists of the average power per cycle of the MAC, ALU, Branch, FIFO read/write, and NOP/stall as measured in lab. In simulation, as each instruction is processed, the corresponding average energy is added to the total. The activity of each processor is also recorded, and used for an estimate of the energy consumption.
- Speed: The speed of each processors is initially set to the maximum frequency to get the

highest performance. For minimum energy consumption, the energy of each processor can be scaled later.

- Communication: Links between every processors will be analyzed for their communication distance and average throughput.

Accuracy

The encoded data is compared to the reference output of the MPI-C output for accuracy. For smaller frame sizes the .h264 output bitstream can also be decoded using the reference software JM, and compared using Elecard YUV viewer. Because of the encoding algorithm implemented, the encoder is nearly data independent. In intra prediction regardless of the input data, all processors doing prediction will execute the same set of instructions except for the processor calculating the chosen prediction will repeat its calculation again. Although a different mode may be chosen every time, the number of instruction repeated in any of the processors is very similar and will yield the same approximate power. In inter prediction, the ME_ACC will undergo the same number of iterations every time, the only variance will be in the motion vector prediction. In the motion vector prediction depending on the mode, different conditions are taken but the number of instructions per condition is approximately equal and will have little effect on the overall accuracy. Simulation setup time is not a factor for intra prediction, at the start of every macroblock the pipeline will be empty because prediction cannot start until the previous prediction is complete, essentially starting over every time. In inter prediction, the only overlap of computation between macroblocks is in the integer transform and CAVLC phases, both of which are much shorter and less power consuming than the prediction process thus will not have a significant effect on results. To account for this though a much longer simulation time is used so that this small factor can be averaged out.

Video Sequences (Size and Length)

Because of the limitation of the simulator, representative macroblocks are chosen and the results are interpolated for the entire frame. The average time for a macroblock to be encoded is calculated from the time it is requested to the time the reference frame is completed in intra prediction and from the time requested to when the ME_ACC completes the search pattern for inter prediction. This is then multiplied by the number of macroblocks per frame to determining the number of frames per second the encoder is capable of for the given frequency and voltage.

6.2 Performance Comparisons

Performance comparison is broken down into two categories, intra frames and inter frames, in each situation the simulated results of a few macroblock are used to interpolate the data for each frame. Two different comparison are given depending on the frame sizes, general purpose and programmable processors are compared for smaller frame sizes only since the performance of these types of platforms do not allow for encoding of larger frame sizes. Since data is only presented for a few frame sizes, the number of frames per second has been interpolated for various other frame sizes for comparison based on the number of macroblocks that can be encoded per second. The operation frequency has been kept the same as in the original implementation. Since ASICs are built specifically for encoding and have been optimized only, know data is presented. The reported memory size is for internal memory only, depending on the platform, the encoders may also use additional off-chip memory, please refer for the references for a more detailed description of each. Power is also only reported for larger frame sizes since no power/energy data is given in other implementations for small frame sizes for comparison.

The encoder has also been implemented on the AsAP chip with bit accurate values as compared to AsAP simulation and MPI simulation. Table 6.1 gives the maximum frequency and encoded frames per second achieved with each given voltage and the corresponding power values. Results of the encoder implementation on-chip is consistent with per-processor performance of the AsAP2 chip as presented in the IEEE Journal of Solid-State Circuits with a few minor exceptions [7]. The single processor speed when running at .675 V was 66 MHz and 1.2 GHz at 1.3 V. The performance increase is not linear though, at speeds beyond 1 GHz, the performance increase begins to level off, possibly due to power grid noise and velocity saturation in the device. The max speed achieved in the H.264 encoder at 1.3 V was about 800 MHz, approximately a 30% reduction in speed but with linear increases in performance as voltage is increased. Various factors may be contributing to the lower performance, most notably the stress on the power grid from running 115 processors simultaneously as opposed to a single processor. The encoder performance and behavior though are

Voltage (V)	Max Frequncy (MHz)	Intra fps (QCIF)	Inter fps (QCIF)	Power Intra (mW)	Power Inter (mW)
0.8	172	19	95	108.8	365.1
0.9	295	33	160	213.6	452.6
1.0	410	49	233	419.0	662.3
1.1	539	66	324	696.3	908.4
1.2	651	82	427	802.7	1059
1.3	798	96	478	947.5	1189

Table 6.1: Performance of H.264 video encoder on AsAP2 chip

expected and consistent with single processor measurements. The maximum frequency achieved increases linearly with increases in voltage as seen in the single processor test for frequencies below 1 GHz. The power measurements for the encoder are also within 30% of the expected power when the measured per-processor power is multiplied by the number of active processors in the encoder.

Platform	Intel Xeon [4] (4 cores)	Intel PXA27x [5]	ADSP-BF561 [13] (2 cores)	ARM [14] 1136J-S	ASIC [15]	ASIC [6]	AsAP intra	AsAP inter
Technology (CMOS)	90nm	-	-	-	180nm	180nm	65nm	65nm
Area (mm^2)	-	26.78	-	-	31.72	27.1	18.87	19.2
Internal (KB) Memory	256 KB (L2) 2MB (L3)	-	328	-	34.7	64000	16	32
QCIF (fps)	18.4 *	49	218 *	15	-	-	41	216
Frequency	2.8 GHz	624 MHz	600 MHz	69 MHz	-	-	1.2 GHz	1.2 GHz
CIF (fps)	4.6	12.25 *	54 *	3.75 *	-	-	10	54
Frequency	2.8 GHz	624 MHz	600 MHz	69 MHz	-	-	1.2 GHz	1.2 GHz
4CIF (fps)	-	-	13.6 *	-	30	-	2	13.5
Frequency	-	-	600 MHz	-	81 MHz	-	1.2 GHz	1.2 GHz
Power (mW)	-	-	-	-	581	-	702	955
720p (fps)	-	-	6 *	-	30	-	-	5.94
Frequency	-	-	600 MHz	-	108 MHz	-	-	1.2 GHz
Power (mW)	-	-	-	-	785	-	-	955
1080p (fps)	-	-	2.7 *	-	-	25	-	2.64
Frequency	-	-	600 MHz	-	-	200 MHz	-	1.2 GHz
Power (mW)	-	-	-	-	-	1219	-	955

Table 6.2: Comparison of H.264 Encoders * These value are interpolated from given data based on the number of macroblocks that can be encoded per second

	Custom Layout	Mapping Tool
Number of Processors	115	147
Number of Memory Proc.	33	33
Number of Routers	21	53
Computational Proc.	61	61
Long Distance Links	48	52

Table 6.3: Comparison of custom layout and proposed mapping from AsAP arbitrary mapping tool

6.3 Analysis

6.3.1 Chip Utilization

The current implementation uses 115 processors, 2 shared memories, and the motion estimation accelerator. The 115 processors do not include those used only for long distance communication. Table 6.3 gives a comparison of overall processor numbers, routers, and long distance communication links between the current implementation and the proposed mapping using the arbitrary mapping tool. Memory processors are counted as those that are only used for storing data, the only computational instructions within them are for address decoding only, this number is the same for both implementations and listed here for reference only.

6.3.2 Processor Energy

This section breaks down the energy used by the main blocks in the H.264 encoder in the intra and inter prediction process: prediction, integer transform, reference macroblock reconstruction, and CAVLC. The power numbers reported is the average energy per cycle for each processors. Based on this information it can be determined which modules should be focused on for optimization. Power estimates are based on the activity and average power per processor when running at full speed as found in the AsAP2 JSSC paper [7]. The average power for each instruction is measure in lab, in simulation, at every cycle the average power for instruction in the pipeline are added up and reported. The measurements are taken from the start of simulation after configuration time till the end simulation.

The majority of power consumed during the encoding process is from the prediction process, constituting approximately 60% of the total power.

Operation	Power (mW)
FIFO Read/Write	4.6
DMEM Read	13.0
DMEM Write	9.1
Nearest Neighbor Comm.	5.9
Long-distance comm.	12.1
Branch	27.2
MAC	53.3
ALU	48.5
NOP/stall	31.0

Table 6.4: Power consumption of various AsAP instructions as measured in lab used for power calculations in simulation

In intra prediction the majority of power is consumed in the routing processors due to the use of branch on empty FIFO instructions. During periods where no data is present, the program counter continuously jumps between to instructions keeping the oscillator running. In other processors, when idle and waiting for data, the oscillator is halted after a certain number of cycles and energy is saved.

In inter prediction the power is consumed in the memory processors and motion estimation unit. Data is directly passed to the ME_ACC for SAD computation but is also stored locally in processors for residue calculation. Each processors holds data that is packed to 16-bits. In the chain of memory processors, the first processors stores the first 256 luma data and passes everything else along, the second processors stores the second set of luma data and passes everything else along. Because of this, the earlier processors experience a heavier workload and thus more power. Power for memory processors are generally constant across different macroblocks because the same number of writes and reads must be done every time. The residue calculation processors consume relatively little power as compared to the memory processors because they must only compute the index for where data is stored and do simple addition, however depending on the data and mode chosen the activity and power consumption may be different. These processors are also idle most of the time since the majority of time is spent in loading of the ME_ACC and the clocks can be halted.

Intra prediction has been parallelized for performance resulting in 9 processors for the 6 prediction modes computed. Since all luma prediction processors share a common set of data, request are constantly made to the memory processors keeping the routers and memory control

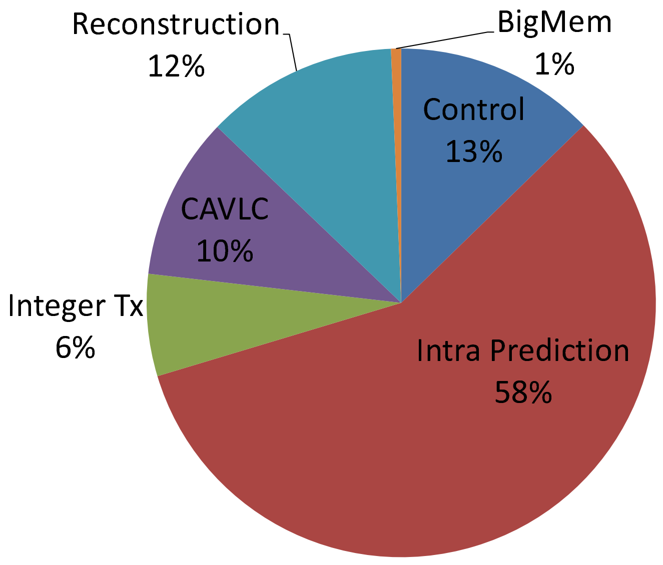


Figure 6.1: Power distribution of major blocks in intra prediction

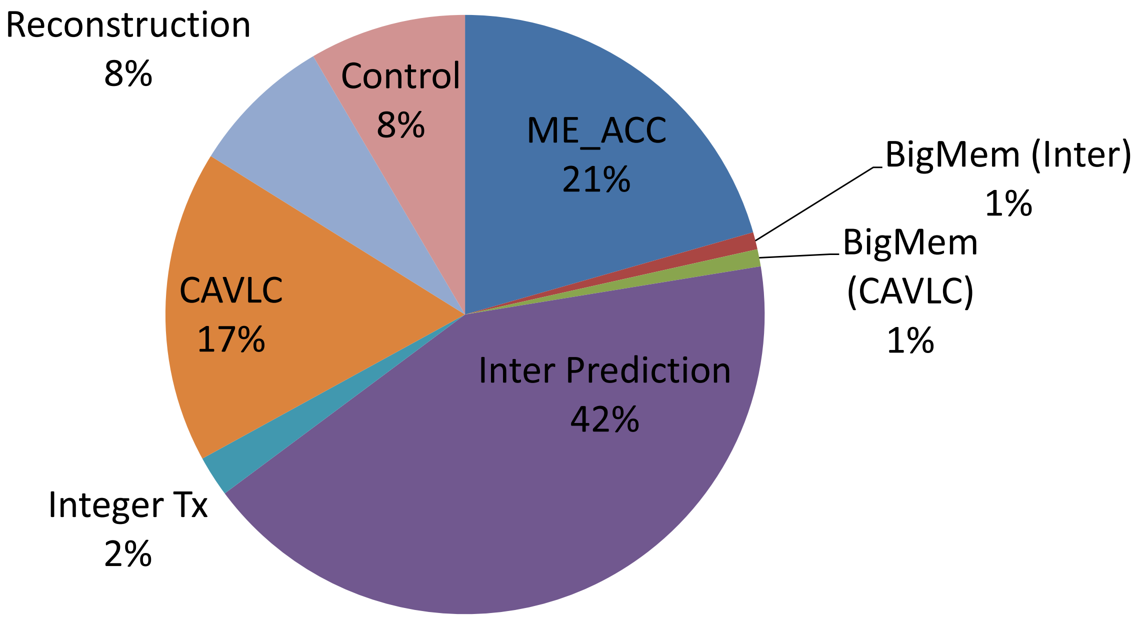
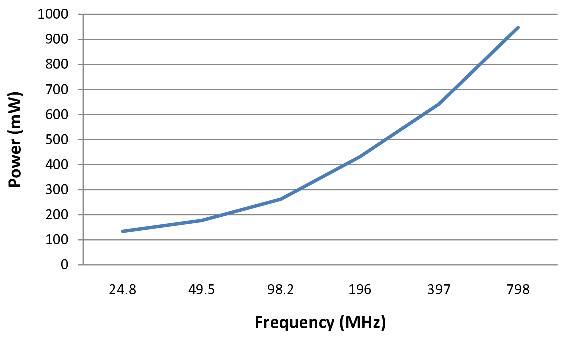
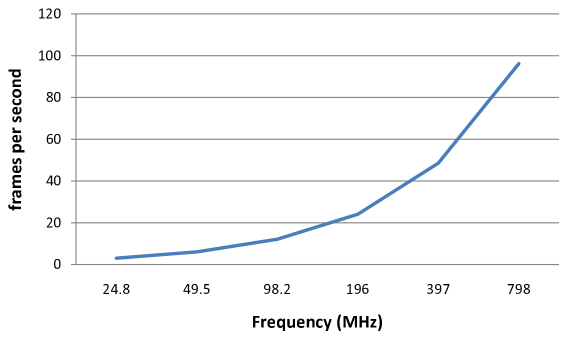


Figure 6.2: Power distribution for major blocks in inter prediction



(a) Average power vs. frequency

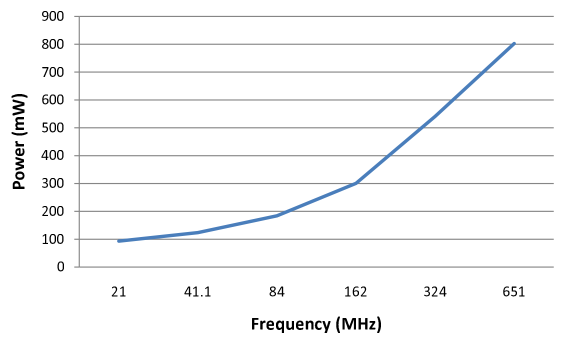


(b) Encoded frames per second vs. frequency

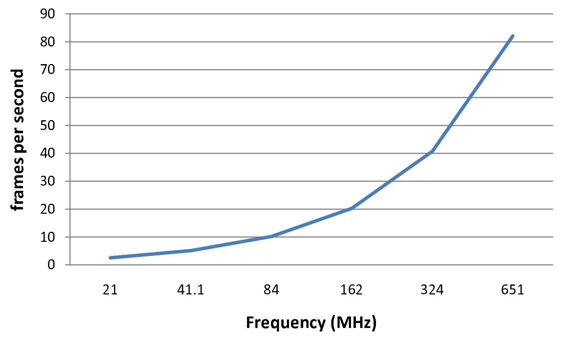
Figure 6.3: Average power and number of encoded frames per second vs. frequency at 1.3V on AsAP2 chip

processor router active most of the time. Data is requested twice for the 16x16 and 4x4 modes, once to determine the SAD for each mode than again to compute the actual SAD. This was done because the data memory for each processors is only 128 word and cannot store both the current data, variables needed for prediction, and the predicted data. It is not know which block/mode will be chosen till all blocks/mode are computed, saving the predicted value for each would consume too much memory space.

Figure 6.10 shows the average power for processors in the reconstruction module for both intra and inter prediction, one of the processors is shown as using no power because it is only used for intra 16x16 prediction (the test case here uses intra 4x4 prediction, hence this block is not used and only passes data which consumes very little power compared to other processors). In inter prediction, because the prediction can continue once the SAD for each macroblock is computed, the reconstruction module is used more often, however because data comes from one source only, few

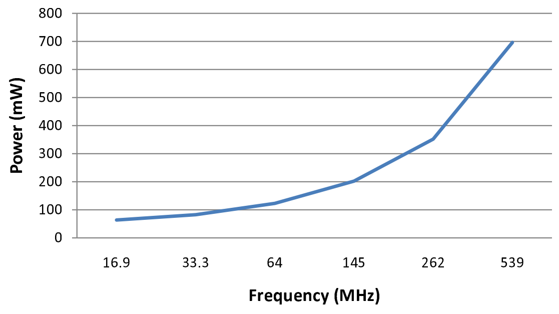


(a) Average power vs. frequency

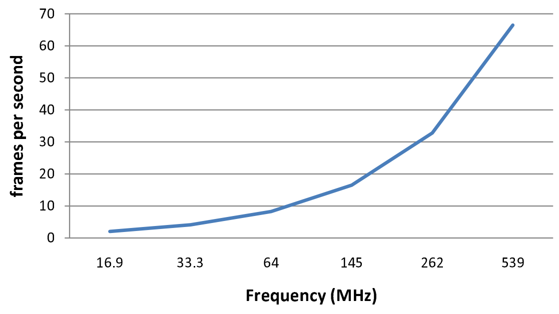


(b) Encoded frames per second vs. frequency

Figure 6.4: Average power and number of encoded frames per second vs. frequency at 1.2V on AsAP2 chip

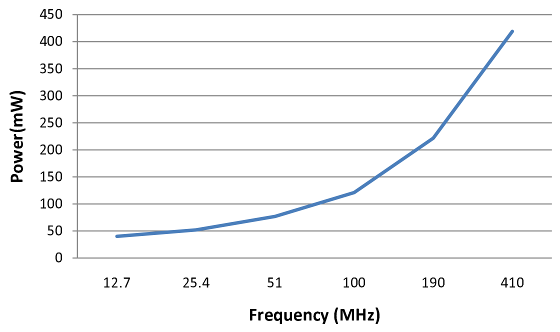


(a) Average power vs. frequency

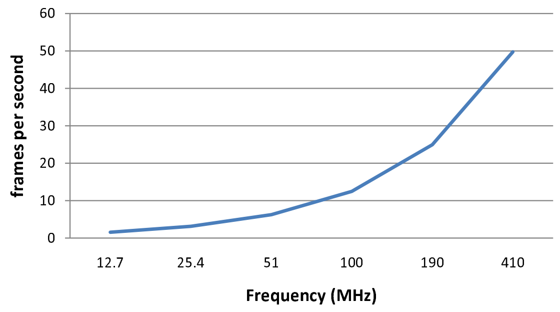


(b) Encoded frames per second vs. frequency

Figure 6.5: Average power and number of encoded frames per second vs. frequency at 1.1V on AsAP2 chip

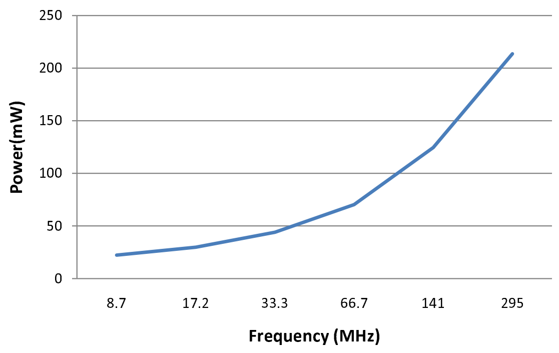


(a) Average power vs. frequency

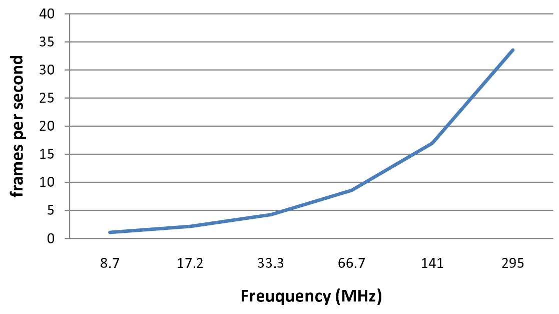


(b) Encoded frames per second vs. frequency

Figure 6.6: Average power and number of encoded frames per second vs. frequency at 1.0V on AsAP2 chip

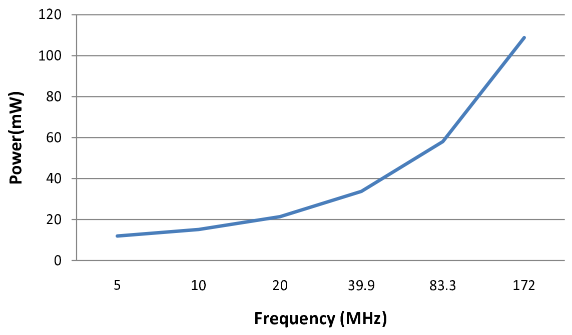


(a) Average power vs. frequency

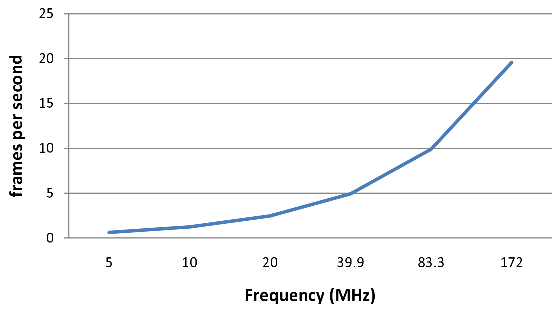


(b) Encoded frames per second vs. frequency

Figure 6.7: Average power and number of encoded frames per second vs. frequency at 0.9V on AsAP2 chip

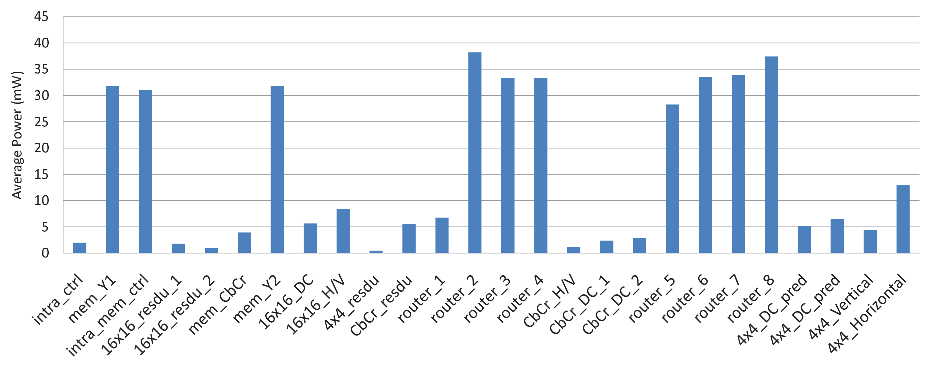


(a) Average power vs. frequency

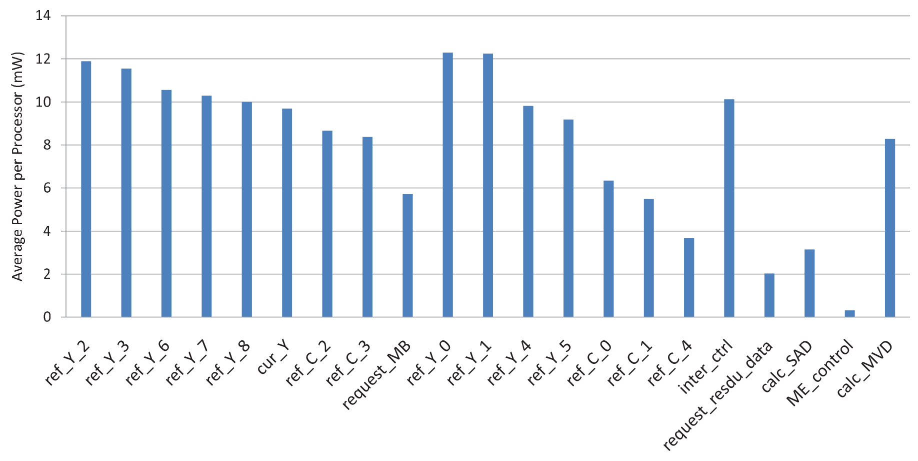


(b) Encoded frames per second vs. frequency

Figure 6.8: Average power and number of encoded frames per second vs. frequency at 0.8V on AsAP2 chip



(a) Average power of intra prediction processors



(b) Average power of inter prediction processors

Figure 6.9: Average power of prediction processors

routing processors are used which the primary power consumers in the intra prediction process.

The majority of power used for integer transform is in routing for the intra prediction mode, inputs are constantly scanned for which type of intra prediction mode is used. The actual transform processors have been optimized by Zhibin [3] therefore consuming very little power. The amount of time that these processors are active is also very small when compared to the entire encoding process hence they are idle most of the time.

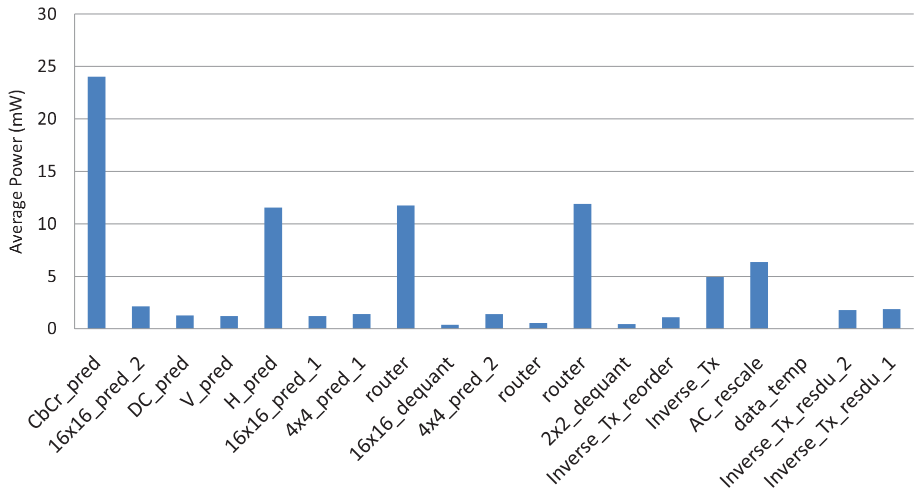
The CAVLC used in this work has also been previously optimized by Zhibin [3], some changes were make to the prediction processes for non zero coefficients resulting in the much larger power consumption. The power for each processor is data dependent however, so for another input sequence the power difference may also be different. These processors can be optimized by either code scheduling to reduce the number of no-ops used or DVFS.

Energy Overhead of Switching I/P Frames

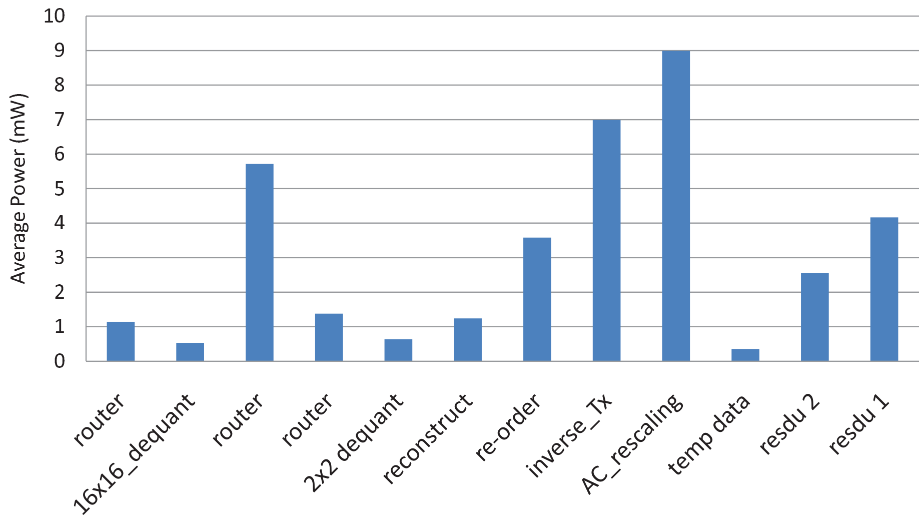
The energy overhead associated with switching frame types is negligible since the only difference would be that the SPS and PPS need to be resent which results in a average power of 129 uW for that processor. The only control change would be an additional 8 lines of code inside the main control processor which is negligible. This low switching cost is possible because the header information is hardcoded and do not need to be recomputed.

6.3.3 Processor Utilization

Processor utilization is presented as the percentage of time the processors is active, stalling (when oscillator is on or off combined) on input, stalling on output, and stalling on no-ops. The activity tracks closely with those of the power plots. The stall on no-ops can be eliminated through code scheduling to hide latencies. By analyzing the stall on inputs/outputs, DVFS values can be calculated for maximum utilization. The majority of stalls occur on reads where data from a previous processor is needed, this arises from the fact that during intra prediction each macroblock must be predicted, computed, transformed, and reconstructed before the next macroblock can begin. These long periods of stalling generally do not consume too much power because the processor oscillators are halted after a certain period, the exception again are routing processors which continuously

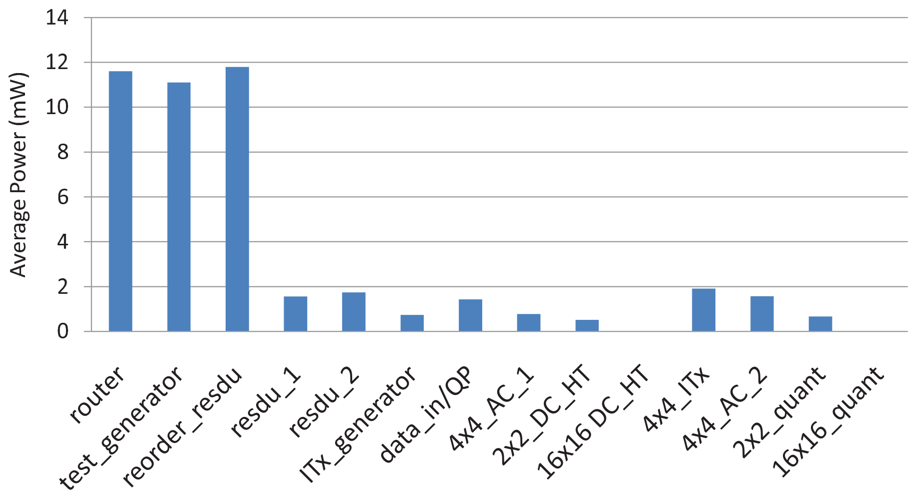


(a) Average power of reconstruction processors in intra prediction

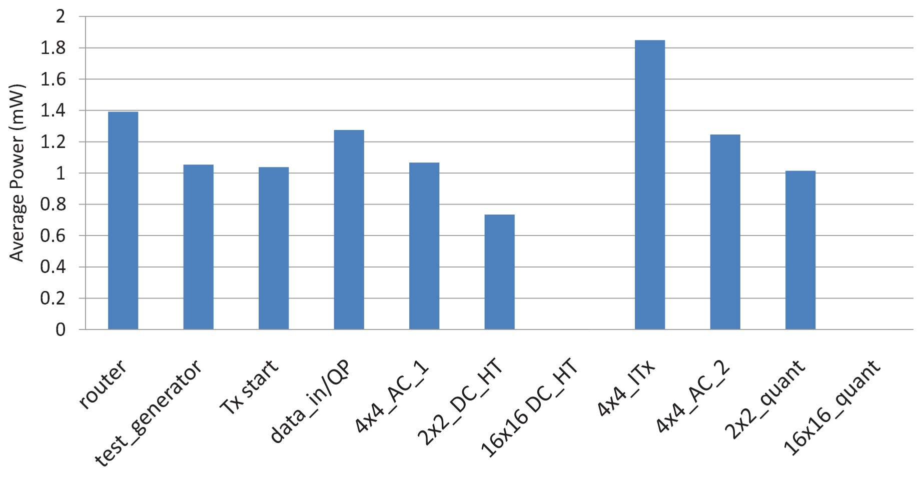


(b) Average power of reconstruction processors in inter prediction

Figure 6.10: Average power of reconstruction processors

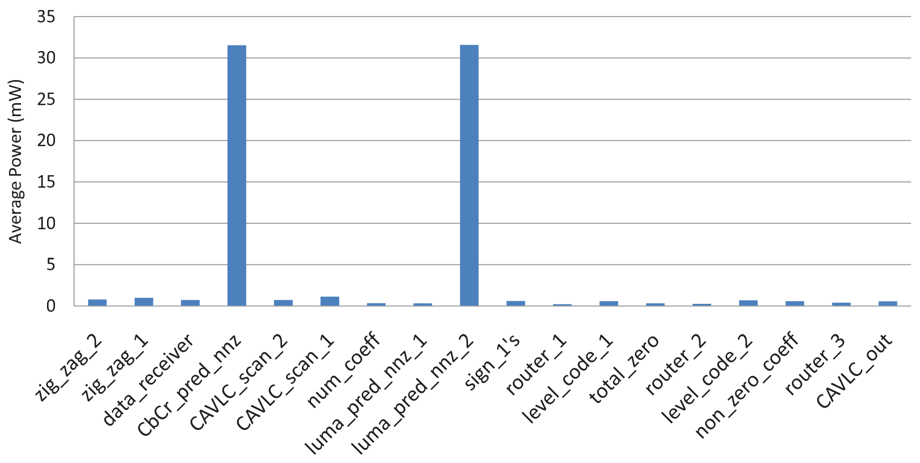


(a) Average power of processors for integer transform in intra prediction

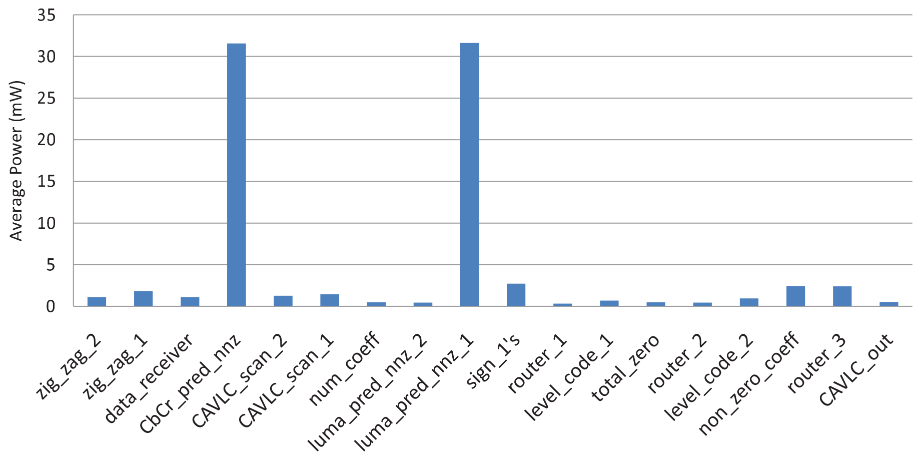


(b) Average power of processors for integer transform in inter prediction

Figure 6.11: Average power of integer transform processors



(a) Average power of CAVLC processors in intra prediction



(b) Average power of CAVLC processors in inter prediction

Figure 6.12: Average power for CAVLC processors

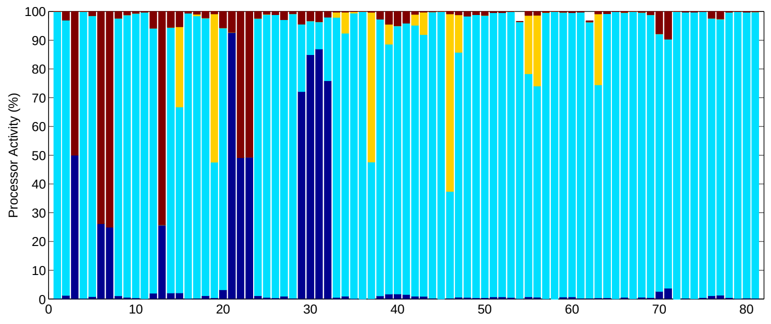


Figure 6.13: Average activity of processors used in intra prediction

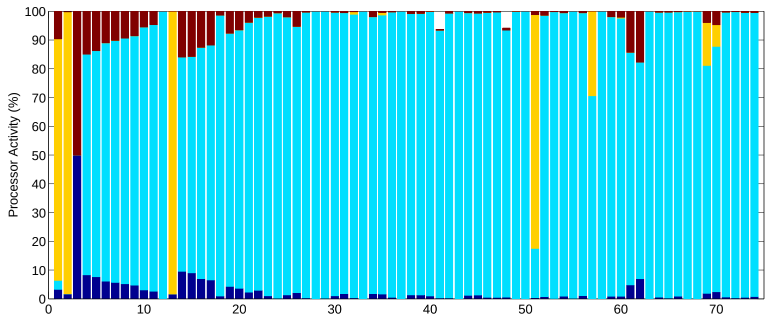
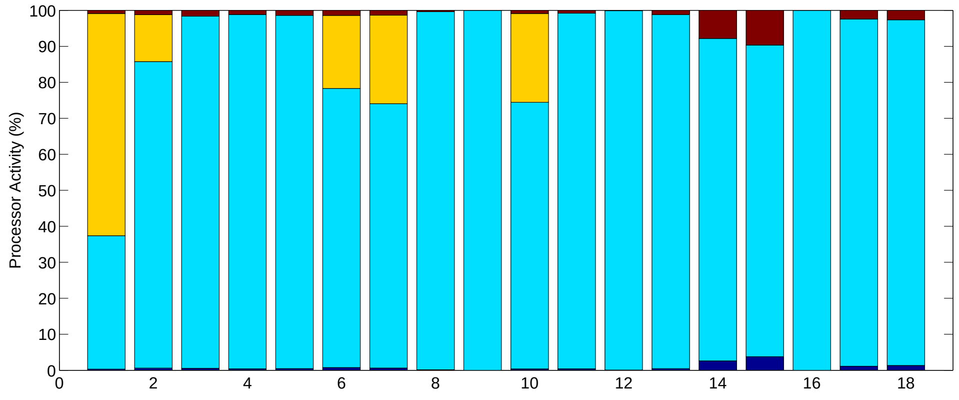


Figure 6.14: Average Activity of processors used in inter prediction

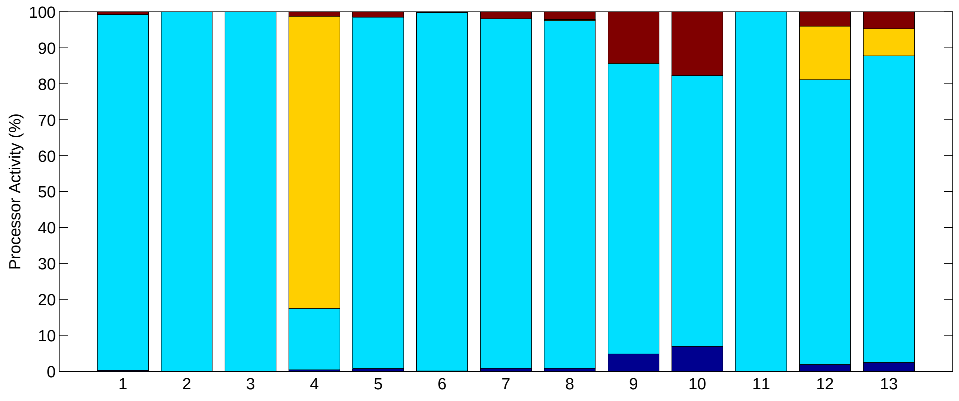
branch between a few no-ops and branch on FIFO instructions. The indexes for the x-axis for these plots correspond to the same functions as those shown for the power plots.

During the reconstruction phase, although the inverse transform process is also done on a 4x4 block basis, they must be reordered prior to being transformed. As shown in Fig. 6.15, the first few processors hold the DC data and must wait for the forward integer transform process to complete before starting, showing the stall on input. Once this is complete, the inverse transform process is fairly simple consisting of mainly adds and shifts resulting in the majority of time waiting for more inputs.

Because the integer transform and CAVLC process are quick compared to the prediction process, the majority of time is spent idle waiting for more data to encode. For maximum optimization these blocks are important, but the focus should be on the prediction blocks first because they

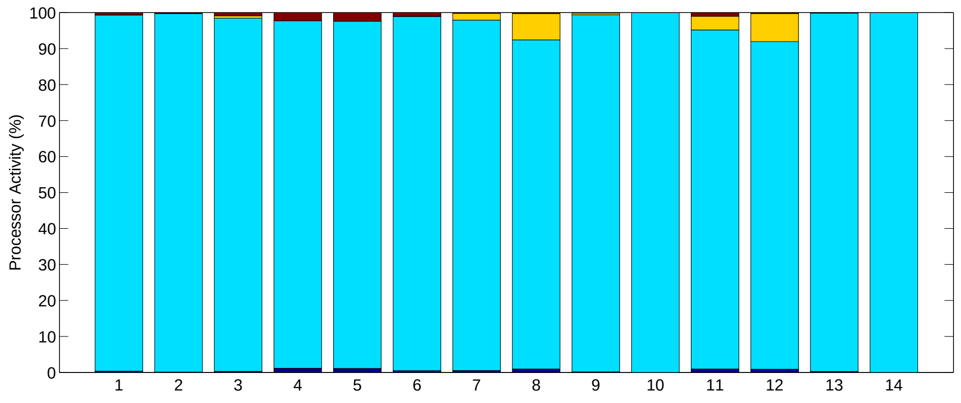


(a) Average activity of reconstruction processors for intra prediction

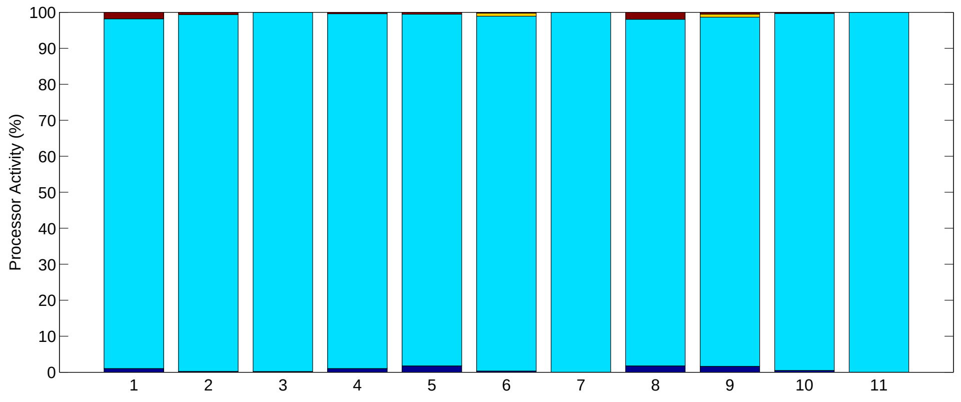


(b) Average activity of reconstruction processors for inter prediction

Figure 6.15: Average activity of reconstruction processors



(a) Average activity of integer transform processors for intra prediction



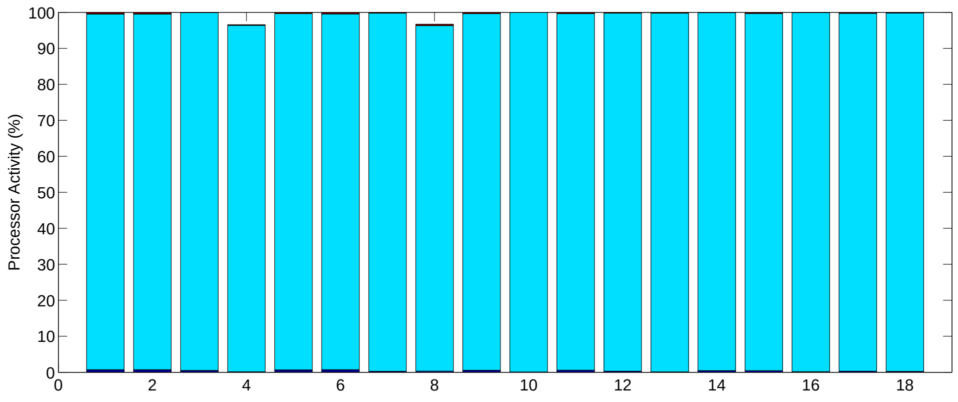
(b) Average activity of integer transform processors for inter prediction

Figure 6.16: Average activity of integer transorm processors

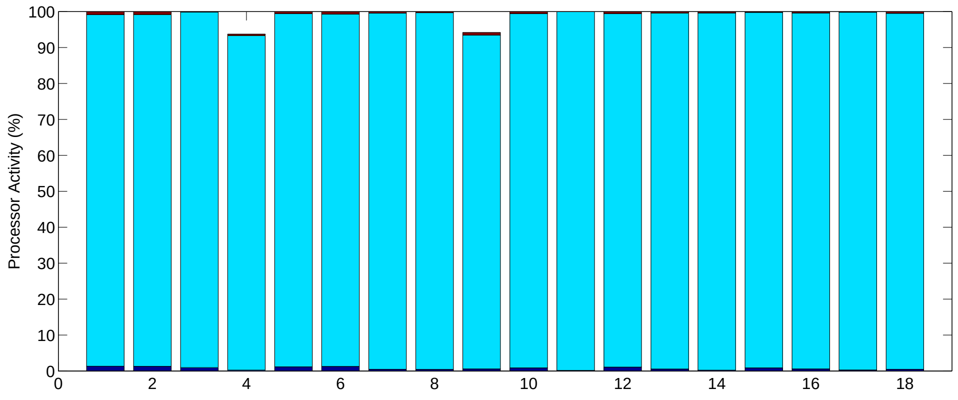
are bottle necks using the majority of encoding time.

Optimization

Because of the low activity of the processors, the majority of power is wasted on cycling on empty cycles. For the power optimization the voltage and frequency of each processor can be scaled for efficiency. Original measurements are taken with the chip running at full power at 1.3V and 1.2 GHz, using this activity number, the frequency is scaled for each processors for greater utilization. Reducing the frequency by a factor of 2 increases the activity of that processors by a factor of 2. To achieve maximum efficiency the frequency can be scaled by any multiple, in this

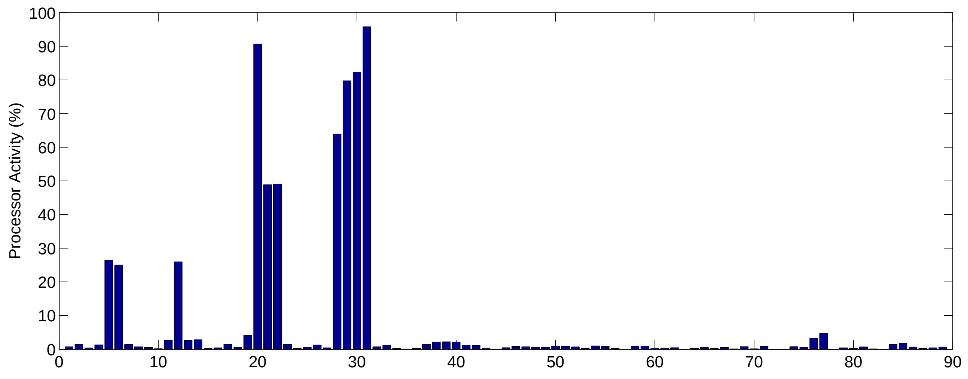


(a) Average activity of CAVLC processors for intra prediction

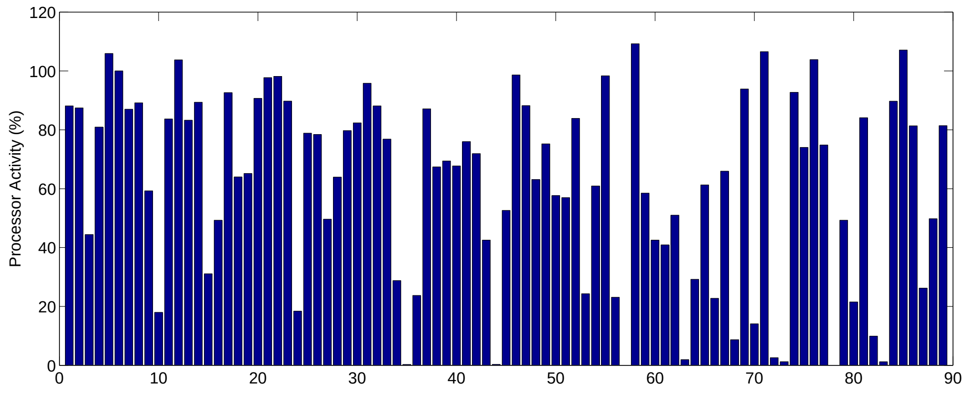


(b) Average activity of CAVLC processors for inter prediction

Figure 6.17: Average activity of CAVLC processors



(a) Average activity of intra prediction processors prior to frequency scaling



(b) Average activity of intra prediction processors post frequency scaling

Figure 6.18: Comparison of average activty before and after frequency scaling for intra prediction

example though only factors of two are used. Each processors is frequency is scaled until the new activity is greater than 110%.

Using the new activity and frequency for each processors, the voltage is scaled. Because only two voltage rails are available, operating frequency is used to determine which voltage the processors should used. The comparison frequency is determined from optimum frequency at that particular voltage. Power is then calculated using the voltage, frequency, and new activity number, as shown in Fig. 6.20, power is reduce by over 50%.

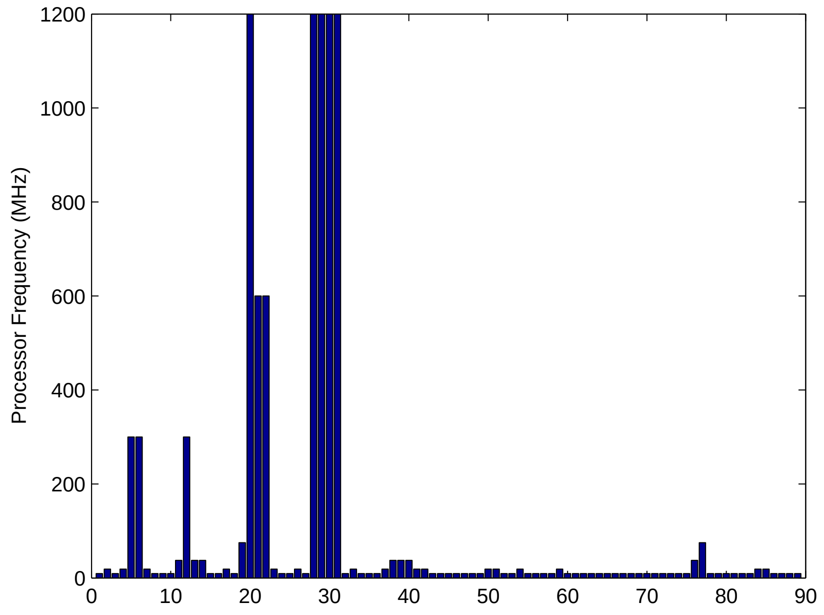


Figure 6.19: Frequency of intra prediction processors after frequency scaling

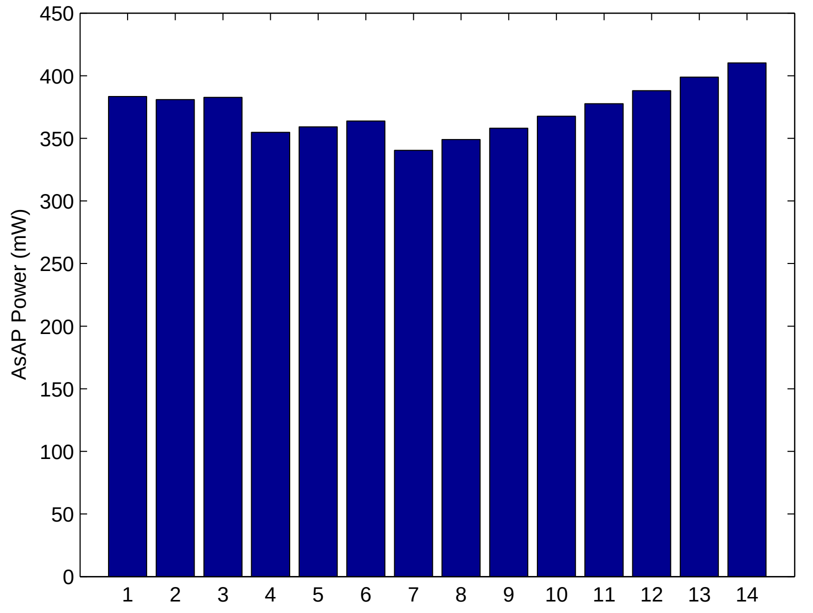
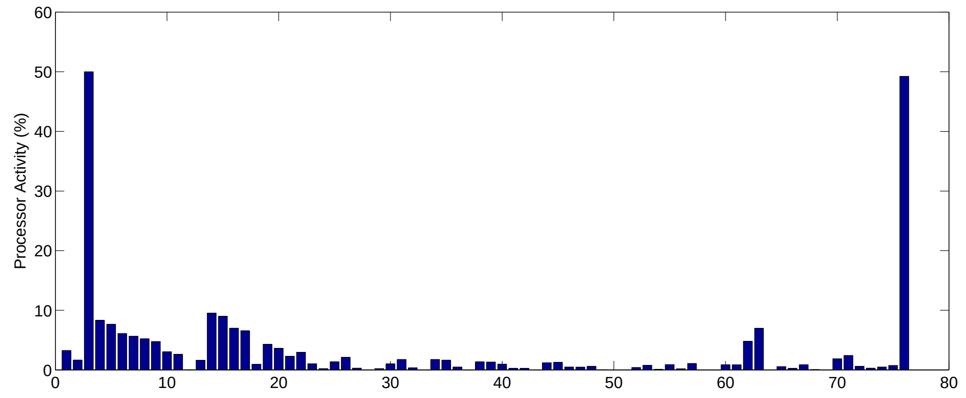
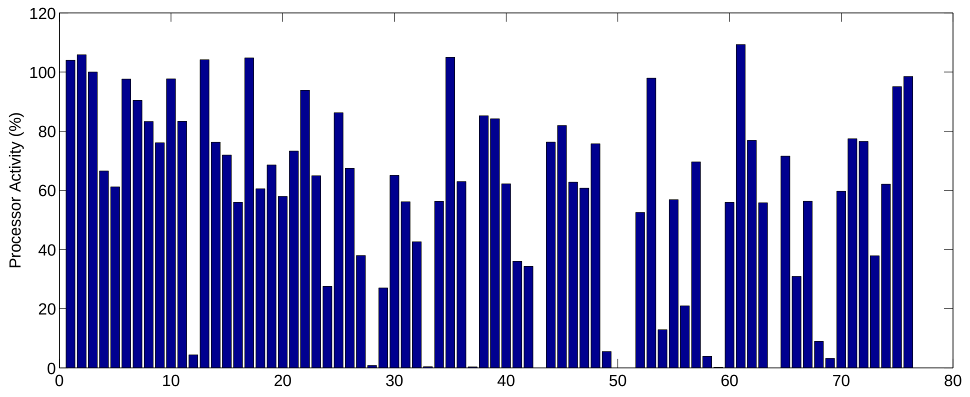


Figure 6.20: Average power of intra prediction processors after frequency and voltage scaling



(a) Average activity of inter prediction processors prior to frequency scaling



(b) Average activity of inter prediction processors post frequency scaling

Figure 6.21: Comparison of average activity before and after frequency scaling for inter prediction

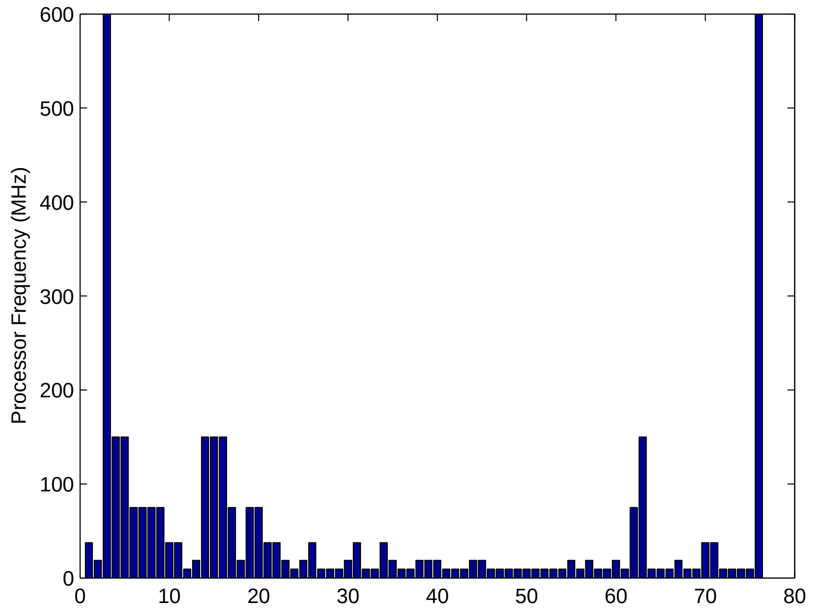


Figure 6.22: Frequency of inter prediction processors after frequency scaling

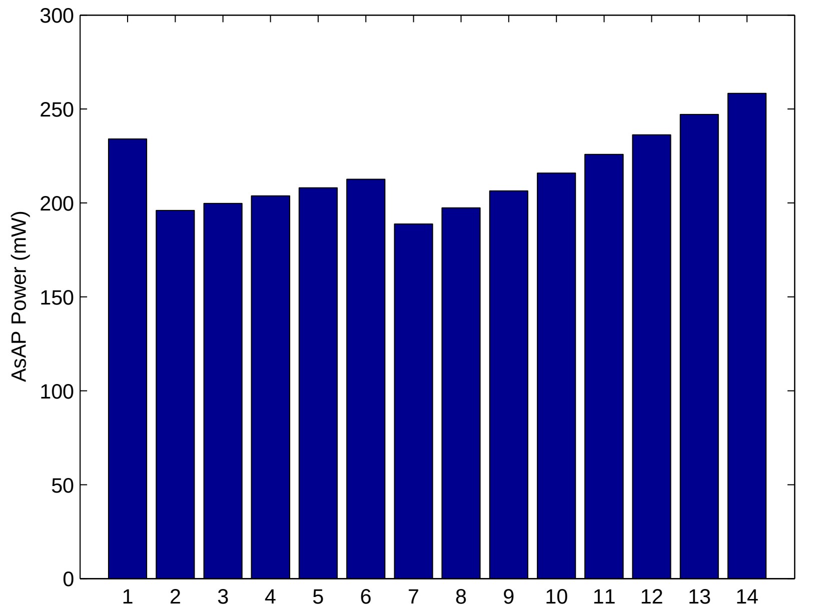


Figure 6.23: Average power of inter prediction processors after frequency and voltage scaling

6.3.4 Processor Memory Usage

This section presents the instruction data memory usage of the all the processors. The majority of the processors average around 45 instructions and the main computation processors averaging around 100 instructions. A greater instruction memory block would reduce the number of processors required and increase the percentage of time that each computation processor is active, however much of this space might not be used because nearly half the processors are used for routing and memory purposes. Increasing the amount of code in each processors may also decrease throughput due to less parallelization, but would also reduce the overhead of code splitting. Many of the processors in intra prediction if further divided would not provide any additional speed up through parallelization because of data dependencies. The dynamically configurable memory (DCMem) count is only given for memory processors where they are used for computation, DCMem registers that are used for setting output directions and long distance communication are not used.

The instruction count spread for integer transform and CAVLC are fairly constant because the majority of processors are used for computation and few routing processors are needed. These processes are also done on a 4x4 block basis only so the amount of control can also be reduced because data can be stored locally and passed along at every stage. In the CAVLC even less variables are needed for the actual computation but the majority of the instruction memory is used for storing look-up tables and checking previous encoding levels.

6.3.5 Communication

This section looks at the communication links used the AsAP processors. Each link is assigned a number for reference only, this does not correlate with any modules or functions. Link length is measured by the number of processors between the beginning and end minus 1, so a link between two neighboring processors would have a length of one. The average number of cycles between writes is taken from beginning of simulation (after configuration) till the end of simulation and is used to calculate the average throughput. These throughput values are calculated based on a simulation running at 1.3V and 1.22 GHz. Max throughput is calculated from the average number of cycles between writes when stalling time is not included. These max values are much greater than the average throughput giving the approximate throughput when active. Two tables are given,

Processor	Imem	Dmem	DCMem
data collector	46	6	-
main control	82	9	-
redirect	38	5	-
MB request	17	0	-
intra control	95	7	-
intra Cb/Cr mem	60	128	7
intra Y mem 1	42	128	5
intra Y mem 2	47	128	5
intra mem control	13	1	-
16x16 DC	117	78	-
16x16 resdu	70	128	7
16x16 resdu 2	97	128	18
router	14	3	-
router	37	5	-
reorder resdu	49	28	-
intra resdu	31	128	8
intra resdu 2	37	128	7
test generator	113	89	-
transform generator	37	9	-
16x16 H/V	127	51	-
4x4 DC pred	118	47	-
4x4 DC resdu	102	49	-
4x4 vertical	116	55	-
4x4 horizontal	103	55	-
router	15	3	-
MB reconstruction	14	3	-
16x16 pred 1	40	128	5
16x16 pred 2	22	128	2
4x4 pred 1	111	128	4
4x4 pred 2	92	128	4
pred. collector	34	5	-
DC pred	24	18	-
vertical pred	22	18	-
horizontal pred	17	18	-
4x4 resdu	19	6	-
Cb/Cr resdu	27	128	5
router	21	1	-
router	26	2	-
router	14	0	-
router	17	0	-
router	18	1	-
router	26	1	-
router	18	1	-
router	15	0	-
router	16	3	-
chroma pred	41	128	2
Cb/Cr H/V	127	59	-
Cb/Cr DC	112	62	-
Cb/Cr DC 2	121	54	-
16x16 dequant	77	50	-
2x2 dequant	58	24	-
AC rescale	94	65	-
router	23	9	-
router	19	2	-
inverse transform	79	44	-
reorder	46	28	-
reorder resdu 1	36	128	8
reorder resdu 2	35	128	7
QP table	46	118	-
4x4 transform	117	32	-
4x4 AC 1	44	75	-
4x4 AC 2	62	74	-
2x2 DC transform	85	70	-
2x2 quantization	118	58	-
16x16 DC transform	78	40	-
16x16 quantization	31	51	-
zig zag 2	83	40	-
temp hold	6	128	3
CAVLC scan 2	69	50	-
data receiver	103	10	-
luma nnz 2	72	24	-
luma nnz 1	127	48	-
Cb/Cr nnz	110	44	-
zig zag 1	79	35	-
CAVLC scan 1	98	50	-
num coeff	75	119	-
sign trailing 1's	115	89	-
router	21	10	-
level code 1	71	50	-
level code 2	121	44	-
total zero	56	124	-
router	13	3	-
non zero run	46	62	-
router	25	5	-
CAVLC out	44	14	-
SPS/PPS	50	25	-
SH	126	10	-
intra pred mode	63	45	-
inter mv mode	90	45	-
router	11	5	-
output	81	22	-
inter control	115	25	-
ME_ACC control	106	40	-
calc SAD	58	15	-
calc MV	121	24	-
calc MV mem control	38	20	-
ref Y 0	45	128	5
ref Y 1	46	128	5
ref Y 2	46	128	5
ref Y 3	46	128	5
ref Y 4	46	128	5
ref Y 5	46	128	5
ref Y 6	46	128	5
ref Y 7	46	128	5
ref Y 8	46	128	5
cur Y	46	128	5
ref C 0	41	128	5
ref C 1	41	128	5
ref C 2	41	128	5
ref C 3	41	128	5
ref C 4	35	128	5
calc inter MB	72	12	-
inter resdu calc	124	10	-

Figure 6.24: Number of instruction and data memory words used per processors. Dynamic Memory (DC Mem) is only listed for memory processors where they are used for data storage and computation.

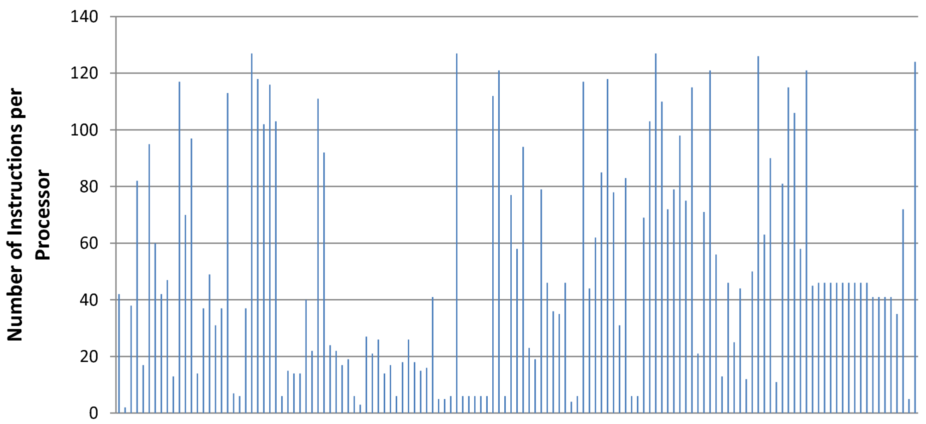
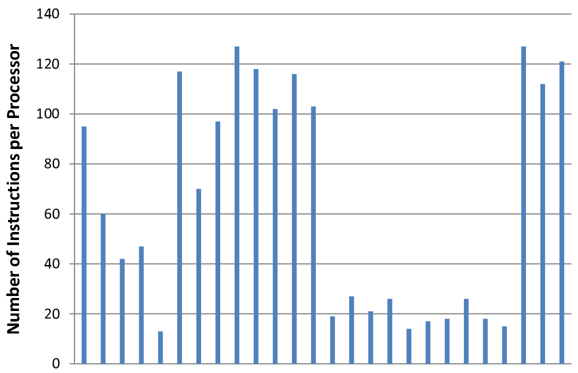
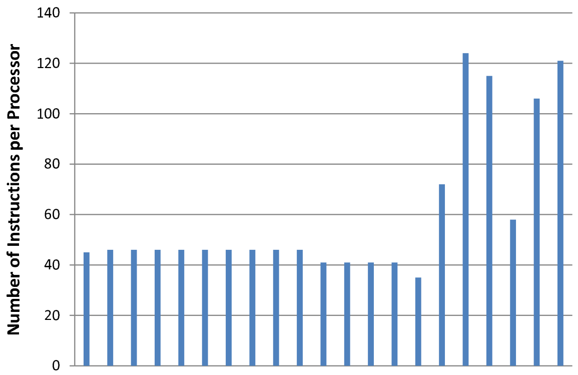


Figure 6.25: Number of instruction memory words used per processor

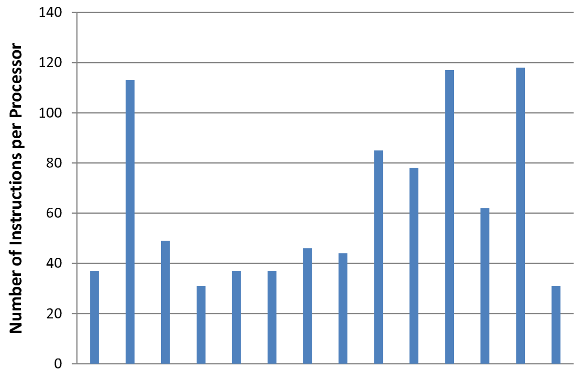


(a) Number of instruction memory words for intra prediction processors

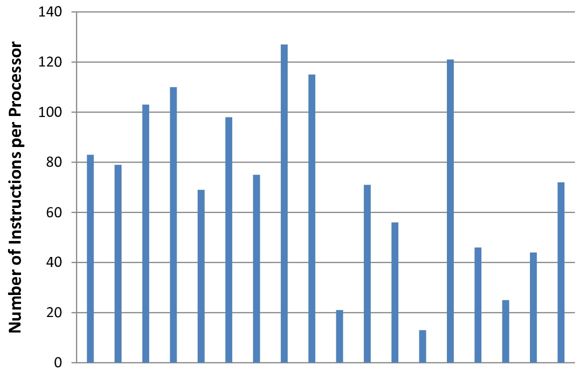


(b) Number of instruction memory words for inter prediction processors

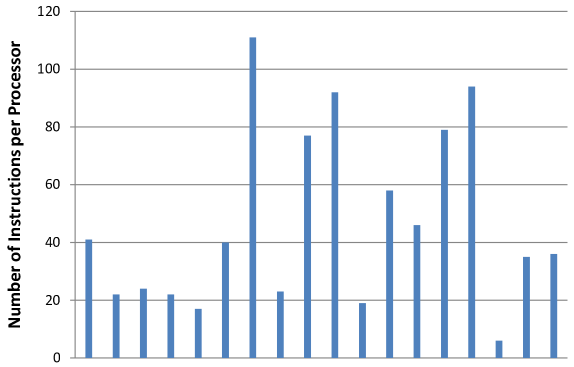
Figure 6.26: Instruction memory usage for prediction processors



(a) Number of instruction memory words used for integer transform processors



(b) Number of instruction memory words used for CAVLC processors



(c) Number of instruction memory words used for reconstruction processors

Figure 6.27: Number of instruction memory words used for integer transform, CAVLC, and reconstruction processors

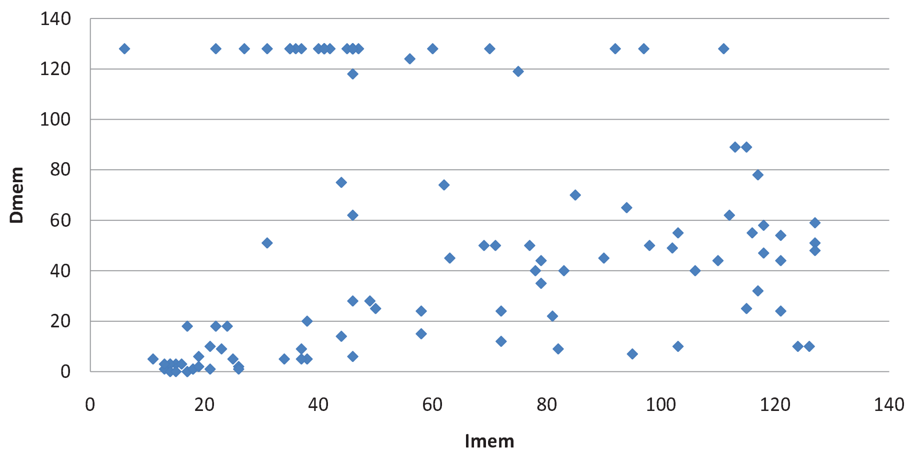
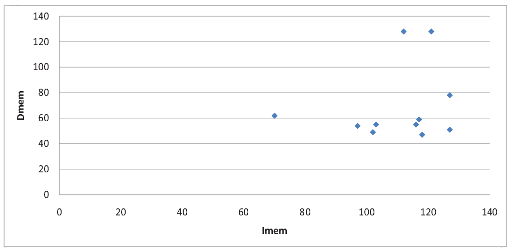
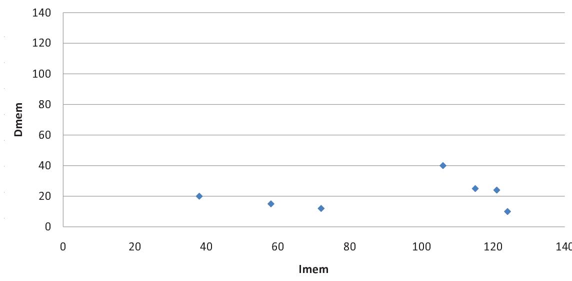


Figure 6.28: Scatter plot of instruction memory vs. data memory for all processor.

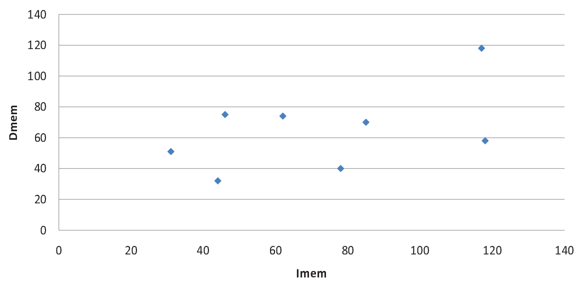


(a) Scatter plot of instruction and data memory for intra prediction processors.

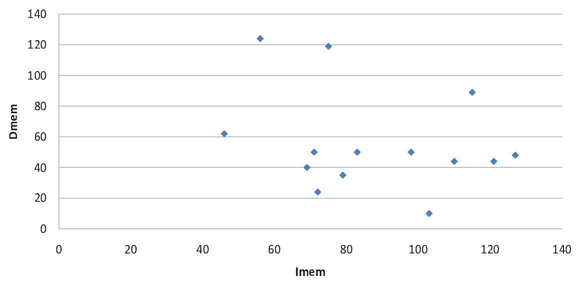


(b) Scatter plot of instruction and data memory for inter prediction processors.

Figure 6.29: Scatter plot of instruction and data memory for prediction processors



(a) Scatter plot of instruction and data memory for integer prediction processors.



(b) Scatter plot of instruction and data memory for CAVLC processors.

Figure 6.30: Scatter plot of instruction and data memory for integer prediction and CAVLC processors

Processor	Length	Average Throughput	Max Throughtput	Processor	Length	Average Throughput	Max Throughtput
data collector	1	0.241	7.6	zig zag 2	1	12	505
main control	1	33.2	809.3	zig zag 1	1	12	510
main control	1	0.103	1.9	CAVLC data receiver	1	13.1	1200
MB request	1	0.0685	0.0695	CAVLC data receiver	1	0.37	52.1
redirect	1	64.3	186700	CAVLC data receiver	2	12.8	1200
intra control	1	6.6	105.1	Cb/Cr pred nnz	1	0.705	100
intra control	4	813	813	CAVLC scan 2	1	3.8	180
intra control	9	2.2	37.9	CAVLC scan 2	2	6.5	314
intra control	3	0.103	2.2	CAVLC scan 1	1	7.4	316
intra control	1	2.2	37.9	CAVLC scan 1	3	2.1	88.9
intra mem Y 1	1	813	154900	CAVLC scan 1	1	3.9	165.2
intra mem Y 2	1	98.2	98.6	num coeff	1	4.5	430.6
test generator	9	0.236	47.3	luma pred nnz 2	2	5.1	544
16x16 DC	1	14.8	109.1	luma pred nnz 1	2	0.994	65
16x16 DC	1	0.45	4.7	luma pred nnz 1	1	0.994	65
16x16 H/V	1	2	16.1	SPS/PPS	1	1.2	1200
Tx start	1	24.5	98600	sign 1's	1	9	449
Cb/Cr DC	1	8.3	117.6	sign 1's	2	7.4	368
QP table	1	24.3	1600	sign 1's	1	3.1	151
QP table	1	0.236	24.9	router	1	7.5	7000
4x4 AC 1	6	12.4	472.8	level code 1	1	8.5	393
4x4 AC 1	1	13	510	SH	1	2000	5200
2x2 DC HT	1	12.1	2200	intra mode pred	1	143.7	2300
2x2 DC HT	1	0.236	62.9	Inverse Tx reorder	1	8.2	443
reorder Cb/Cr	9	5.1	1600	AC rescale	3	0.0337	0.207
4x4 DC pred	1	0.74	11.6	total zero	1	7.4	699
4x4 DC pred	1	2.4	37.1	total zero	2	3.1	250
4x4 horizontal	1	4.7	32	router	1	8.1	1200
4x4 horizontal	1	0.461	3.3	level code 2	1	6.7	484
4x4 Tx	1	11.8	274.1	router	1	4.6	1300
4x4 Tx	2	12.3	284	MV pred	1	6200	17100
4x4 AC 2	6	11.6	485	non zero run	2	7.4	420
4x4 AC 2	1	11.5	482	router	1	15.7	3900
2x2 quant	1	23.2	3500	CAVLC out	11	18.3	1700
2x2 quant	5	0.498	67.4	output	1	13.4	1400

Figure 6.31: Average and max throughput of major intra prediction communication links. Link lenght is determined by the number of intersected processros minus 1.

one for intra and one for inter prediction.

Processor	Length	Average Throughput	Max Throughtput	Processor	Length	Average Throughput	Max Throughtput
data collector	1	1.1	6.4	2x2 DC HT	1	0.193	35.8
main control	1	216	8900	4x4 Tx	1	10.7	276.3
main control	1	0.534	23.1	4x4 Tx	2	10.4	272.6
MB request	1	1.4	1.4	4x4 AC 2	1	10.3	465
ref Y 2	1	13.7	62.3	4x4 AC 2	6	10.3	465
ref Y 3	1	13.7	66.2	2x2 quant	1	20.7	2600
ref Y 6	1	13.7	81.7	2x2 quant	6	0.455	63.7
ref Y 7	1	13.7	88.6	zig zag 2	1	10.8	482.3
ref Y 8	1	13.7	96.7	zig zag 1	1	10.7	486.6
cur Y	1	13.7	106.6	data receiver	2	11.4	1000
ref C 2	1	13.7	153.4	data receiver	1	0.386	37.8
ref C 3	1	13.7	179.6	data receiver	1	11.4	1000
request MB	9	1.4	2500	Cb/Cr pred nnz	1	0.649	99.6
redirect	1	215.3	9300	CAVLC scan 2	1	3.3	185.4
ref Y 0	1	13.7	59.1	CAVLC scan 2	2	4.6	257.5
ref Y 0	1	18.5	61.4	CAVLC	1	5.4	261.7
ref Y 1	1	13.7	59.8	CAVLC	3	1.9	93.6
ref Y 4	1	13.7	70.7	CAVLC scan 1	1	3.5	172.3
ref Y 5	1	13.7	75.8	num coeff	1	4.1	415.1
router	1	0.338	15.1	luma pred nnz 2	2	4.3	511.8
ref C 0	1	13.7	118.6	luma pred nnz 1	2	0.853	63.6
ref C 1	1	13.7	133.8	luma pred nnz 1	1	0.853	63.6
ref C 4	1	13.7	211.6	SPS/PPS	1	1.1	719
inter control	2	0.241	4.3	sign 1's	1	6.8	444.3
inter control	1	0.265	4.8	sign 1's	2	5.4	352.1
inter control	1	133	3000	sign 1's	1	2.8	183
router	1	0.289	10.8	router	1	6.8	4100
test generator	8	0.434	54.1	level code 1	1	6.3	369.1
test generator	3	0.048	6	SH	1	2.7	4200
calc. resdu	9	13.7	389.1	resdu reorder	2	8.5	306
calc SAD	1	0.193	2.7	AC rescale	3	0.048	0.207
calc SAD	6	0.241	3.4	total zero	1	5.4	531
ME control	2	3.6	391.7	total zero	2	2.8	275.4
ME control	1	0.526	58	router	1	6.4	987.2
calc MV	17	0.193	1600	level code 2	1	2.1	440.6
Integer Start	1	20.5	8800	router	1	4.5	1100
QP table	1	20.6	1400	inter mv pred	1	0.338	399
QP table	1	0.338	24.9	non zero run	2	4.5	438
4x4 AC	6	11	455.8	router	1	13.1	2700
4x4 AC	1	11.8	488.6	CAVLC out	11	13.5	1300
2x2 DC HT	1	10.7	1800	output	1	15.8	1300

Figure 6.32: Average and max throughput of major inter prediction communication links. Link length is determined by the number of intersected processors minus 1.

Chapter 7

Future Work and Conclusion

7.1 Architecture Enhancements for Parallel Programming

This implementation of the H.264 encoder has been highly parallelized for maximum performance, however many of the techniques used are due to limitation in the processing platform, some enhancements are suggested for improving the parallel programming process and increasing performance.

7.1.1 Multiple I/O Chips

The small and simple processor architecture of AsAP allows for the easy replication of many cores and low power advantages. Larger programs will be bound by memory issues however and having multiple I/O ports for the chip will give much needed bandwidth and reduce extra control logic that was added to support dual output functions as well as long distance communication. Multiple I/O ports would also allow for multiple applications to run simultaneously.

7.1.2 Multiple Input Processors

As shown in the analysis, the majority of power is consumed through routing processors, this was necessary because only two inputs were available per processor and could not be dynamically configured. Increasing the number of inputs per processor would greatly reduce the number of processors and power needed.

7.1.3 Local Shared Memory

Many of the processors are used as shared memory for other processors, these however only provide small amounts of extra memory and when many are needed the access time is greatly increased because the data must trickle through all the routing processors. Having small local shared memories weather dedicated or processor configured would allow for faster throughput and free up more processors for other tasks.

7.2 Tool Enhancements for Parallel Programming

The majority of time spent in implementing this encoder was in the debug phase, because of the number of processors used (131) and the simulation time, this proved to be the most difficult task.

7.2.1 Arbitrary Mapping Tool For AsAP2

The mapping for this encoder has been hardcoded, using a tool similar to the arbitrary mapping tool available for MPI-C and AsAP1 would greatly speed up the process and reduce communication connection problems. This would also allow for greater ease in modifying this application at a later point.

7.2.2 Analysis of I/O Traffic

One of the most useful debugging tools was to look at the program counter and stall signals for the input FIFOs and output in ModelSim. A tool that would allow for quick easy viewing as well as flagging which processor caused the final stall signal (trace back) to occur would greatly reduce debugging time. Another feature that would be helpful would be to also have a few instructions in the processor causing the stall available to quick viewing. Often times the error would be a simple typo or branch error but would require a lot debugging time to manually do trace back.

7.2.3 Enhanced I/O File Operations

The H.264 encoder has a feedback loop for storing the reconstructed data, since this is too large to fit on AsAP, it must be stored off chip. In simulation however this is not possible because the input file used for reading cannot also be written to in the same instance. A tool that can provide this function would improve the accuracy of testing and debugging.

7.3 Additional Encoding Functions on AsAP

The encoder presented has been simplified reducing the capabilities and advantages of the H.264 standard. Future work in these areas would improve the compression and quality of encoded videos.

Intra Prediction

In intra prediction, only 3 modes are currently used, by using all available modes in the standard the entropy values of the encoded bitstream can be reduced. Each prediction mode has been implemented on either one or two processors, adding additional modes on separate processors should be fairly simple, the only problems would be with the shared memory which can be replicated and mode decision which may need to be done on a separate processor.

Inter Prediction

One the advantages of H.264 over previous standards is the ability to use interpolated samples. Because of the complexity of the motion estimation process only integer samples were used in the ME_ACC. Fully supporting interpolated samples in the future will probably also need to be done via an accelerator. A possible work around for this would be use a smaller search window and interpolate the samples outside of the ME_ACC and send them in as if they were part of the window used by the accelerator. This process though would like increase the latency of the encoder significantly.

Encoding With Slices

The H.264 standard also supports encoding using slices. The advantage of this is that each slice of the picture/frame can be encoded independently (using data for previous matching slices only). On AsAP these can be done in parallel increasing the throughput, the only limitation would then be in hardware where each module would need to be replicated.

Support for HD-resolution

The algorithm presented can be optimized to support up to 720p real time or nearly 1080p-resolution (20fps) when using inter prediction. Initial calculations for this are based off the number of macroblocks that can be loaded for the ME_ACC within a second. The current implementation loads 10 macroblocks every time, this can be reduced to 4 macroblocks by re-using the previously loaded ones however the control logic becomes more complicated since the reference blocks must be stored in processors for residue calculation and do not provide for easy indexing.

7.4 Conclusion

Here an H.264 compliant baseline encoder has been implemented in both MPI and on a programmable parallel processor. The AsAP implementation is shown to be comparable in performance to other DSP while achieving the low power capabilities of ASICs. An analysis of performance and chip usage is also given to provide a stepping stone for future programs and architectures in parallel processing. In applications such as this, the primary source of power consumption are from the ME_ACC and the shared memories, optimizing code for the computation processors will not provide any significant reduction in power will only provide more speed. A more practical approach towards power might be to not use the ME_ACC but implement the process using individual processors, although this is much more complex, there is the possibility of 90% power reduction.

Bibliography

[1] Iain E. G. Richardson. *H.264 adn MPEG-4 Video Compression Video Coding for Next generation Multimedia*. John Wiley Sons Ltd, 2003.

[2] International Standard Organization and Information Technology-Coding of Audio-Visual Objects. *Part10-Advanced Video Coding*. ISO/IEC 14496-10.

[3] Z. Xiao and B. M. Baas. A high-performance parallel CAVLC encoder on a fine-grained many-core system. In *IEEE International Conference of Computer Design (ICCD)*, October 2008.

[4] Yen-Kuang Chen, Eric Q. Li, Xiaosong Zhou, and Steven Ge. Implementation of h.264 encoder and decoder on personal computers. *Journal of Visual Communication and Image Representation*, 2006.

[5] Zhenyu Wei, Kai Lam Tang, and King N. Ngan. Implementation of h.264 on mobile devices. In *IEEE Transactions on Consumer Electronics*, 2007.

[6] Yu-Wen Huang et al. A 1.3 tops h.264/avc single-chip encoder for hdtv applications. In *ISSCC Conference on Multimedia Processing*, 2005.

[7] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, T. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, P. Mejia, A. Tran, J. Webb, E. Work, Z. Xiao, and B. Baas. A 167-processor 65 nm computational platform with per-processor dynamic supply voltage and dynamic clock frequency scaling. In *Symposium on VLSI Circuits*, pages 22–23, June 2008.

[8] R. W. Apperson, Z. Yu, M. J. Meeuwsen, T. Mohsenin, and B. M. Baas. A scalable dual-clock FIFO for data transfers between arbitrary and haltable clock domains. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(10):1125–1134, October 2007.

[9] W. H. Cheng and B. M. Baas. Dynamic voltage and frequency scaling circuits with two supply voltages. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1236–1239, May 2008.

[10] Michael Meeuwsen, Zhiyi Yu, and Bevan M. Baas. A shared memory module for asynchronous arrays of processors. 2007.

[11] Gouri Landge. A configurable motion estimation accelerator for video compression. Master's thesis, University of California Davis, 2009.

[12] Eric W. Work. Algorithms and software tools for mapping arbitrarily connected tasks onto an asynchronous array of simple processors. Master's thesis, University of California, Davis, CA, USA, September 2007. `http://www.ece.ucdavis.edu/vcl/pubs/theses/2007-4`.

[13] 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*, pages 265–269, 2009.

[14] G Nageswara Rao, D Jaya Chandra Prasad RSV, and Srividya Narayanan. Real-time software implementation of h.264 baseline profile video encoder for mobile and handheld devices. In *IEEE International Conference on Acoustics, Speech and Signal Processing, 2006*, 2006.

[15] Tung-Chien Chen et al. Analysis and architecture design of an hdtv720p 30 frames/s h.264/avc encoder. *IEEE Transactions on Circuits and Systems for Video Technology*, 2006.