Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete

Evaluating and Implementing JPEG XR
Optimized for Video Surveillance

Master thesis performed in Computer Engineering
by

Lang Yu

Report number: LiTH-ISY-EX--10/4300--SE
Linköping 1st March 2010
Evaluating and Implementing JPEG XR
Optimized for Video Surveillance

Master thesis in Computer Engineering
at Linköping Institute of Technology
by

Lang Yu
LiTH-ISY-EX--10/4300--SE

Supervisor: Lars Viklund, Oskar Flordal
Axis Communications AB
Examiner: Dake Liu
ISY/ Datorteknik, Linköpings Universitet
Linköping  1st March 2010
Evaluating and Implementing JPEG XR Optimized for Video Surveillance

Lang Yu

Abstract
This report describes both evaluation and implementation of the new coming image compression standard JPEG XR. The intention is to determine if JPEG XR is an appropriate standard for IP based video surveillance purposes. Video surveillance, especially IP based video surveillance, currently has an increasing role in the security market. To be a good standard for surveillance, the video stream generated by the camera is required to be low bit-rate, low latency on the network and at the same time keep a high dynamic display range. The thesis start with a deep insightful study of JPEG XR encoding standard. Since the standard could have different settings, optimized settings are applied to JPEG XR encoder to fit the requirement of network video surveillance. Then, a comparative evaluation of the JPEG XR versus the JPEG is delivered both in terms of objective and subjective way. Later, part of the JPEG XR encoder is implemented in hardware as an accelerator for further evaluation. SystemVerilog is the coding language. TSMC 40nm process library and Synopsys ASIC tool chain are used for synthesize. The throughput, area, power of the encoder are given and analyzed. Finally, the system integration of the JPEG XR hardware encoder to Axis ARTPEC-X SoC platform is discussed.

Keywords
Master Thesis: Evaluating and Implementing JPEG XR Optimized for Video Surveillance

Lang Yu
Institute of Technology in Linköping University
Reg Nr: LiTH-ISY-EX–10/4300–SE

February 16, 2010
Abstract

This report describes both evaluation and implementation of the new coming image compression standard JPEG XR. The intention is to determine if JPEG XR is an appropriate standard for IP based video surveillance purposes. Video surveillance, especially IP based video surveillance, currently has an increasing role in the security market. To be a good standard for surveillance, the video stream generated by the camera is required to be low bit-rate, low latency on the network and at the same time keep a high dynamic display range. The thesis start with a deep insightful study of JPEG XR encoding standard. Since the standard could have different settings, optimized settings are applied to JPEG XR encoder to fit the requirement of network video surveillance. Then, a comparative evaluation of the JPEG XR versus the JPEG is delivered both in terms of objective and subjective way. Later, part of the JPEG XR encoder is implemented in hardware as an accelerator for further evaluation. SystemVerilog is the coding language. TSMC 40nm process library and Synopsys ASIC tool chain are used for synthesize. The throughput, area, power of the encoder are given and analyzed. Finally, the system integration of the JPEG XR hardware encoder to Axis ARTPEC-X SoC platform is discussed.

Acknowledgements

This thesis is the result of my work as both master student in computer engineering division, LITH in Linköping University and intern student in Axis Communications AB. I would like to express my sincere appreciation to all of the people who supported me during my thesis work.

In particular, I would like to express my appreciation to my examiner Professor Dake Liu in Linköping University, he gave me guidance and many good suggestions; My supervisors Lars Viklund and Oskar Flordal in Axis Communications, they gave me the chance to work with them and supported me in daily Technical details. And also my colleague Niklas Pavlovic in Axis Communications, he encouraged and helped me when I was in tollgate. Furthermore, I would like to thank Ted B. Zimmerman, a retired engineer in Lockheed-Martin, he helped me correct the grammar mistakes in my thesis. Finally, I would like to thank my parents, Tingming Yu and Xing Zhou, for 24 years encouragement and support.
# Contents

## 1 Introduction

1.1 Background ............................................. 1  
1.2 Purpose and Goal ..................................... 1  
1.3 Methodology .......................................... 2  
1.4 Chapter Overview .................................... 2  

## 2 JPEG XR Theory

2.1 Overview ................................................. 3  
2.2 Color Conversion ....................................... 5  
  2.2.1 Color Space ...................................... 5  
  2.2.2 Down Sampling ................................... 6  
2.3 Image Partitioning ..................................... 7  
  2.3.1 Tiles ............................................. 7  
  2.3.2 Macro Blocks .................................... 8  
  2.3.3 Blocks .......................................... 8  
2.4 Lapped Bi-orthogonal Transform ...................... 8  
  2.4.1 Pre-filtering ................................... 8  
  2.4.2 Frequency Transform .............................. 12  
  2.4.3 First Stage Pre-filtering ......................... 15  
  2.4.4 First Stage Frequency Transform ............... 17  
  2.4.5 Second Stage Pre-filtering ....................... 17  
  2.4.6 Second stage Frequency Transform ............. 19  
2.5 Quantization ........................................... 19  
  2.5.1 Quantization Parameters ......................... 19  
2.6 Adaptive Prediction ................................... 20  
  2.6.1 DC Prediction .................................. 21  
  2.6.2 LP Prediction .................................. 22  
  2.6.3 HP Prediction .................................. 22
## CONTENTS

2.7 Adaptive Scan ................................. 23
   2.7.1 DC Scan ................................. 23
   2.7.2 LP Scan ................................. 26
   2.7.3 HP Scan ................................. 26
2.8 Adaptive Entropy Coding .......................... 27
   2.8.1 Coefficients Normalization .................. 28
   2.8.2 Run Level Encoding ....................... 30
   2.8.3 Adaptive VLC Encoding ..................... 31
   2.8.4 Bit Stream Structure ....................... 37

3 Evaluation and Comparison .......................... 41
   3.1 Video Surveillance Requirements ................. 41
   3.2 Performance Comparison ....................... 42
      3.2.1 Methodology ...................... 42
      3.2.2 Tools ............................... 44
      3.2.3 Evaluation ....................... 44
      3.2.4 Subjective Observations ............... 56

4 RTL Model Implementation ........................... 59
   4.1 Hardware Issues ............................. 59
      4.1.1 Software Profiling .................... 59
      4.1.2 Parallelization ....................... 60
      4.1.3 Bitwise Operation ..................... 60
      4.1.4 Memory Access Analysis ............... 60
   4.2 Software/Hardware Partitioning ................. 63
   4.3 Hardware Implementation ...................... 63
      4.3.1 Picture Core Transform ................. 67
      4.3.2 Quantization ......................... 68
      4.3.3 Prediction Unit ....................... 70
   4.4 Simulation and Verification .................... 74
      4.4.1 Test Bench ........................... 74
      4.4.2 Verification ......................... 75
   4.5 Synthesize Result ............................ 76
      4.5.1 Throughput ........................... 76
      4.5.2 Area ................................ 76
      4.5.3 Power ................................ 77
   4.6 System Integration ............................ 78
Chapter 1

Introduction

1.1 Background

In IP based video surveillance, compression is necessary in order to transmit video efficiently over the network. There is a multitude of standard methods available for compression. Axis is currently using the JPEG [1] still-image standard and the H.264 [10] (or MPEG-4 Part 10) moving-picture standard in their cameras.

A newer coming method accepted as a standard is JPEG XR [11] which is an image compression algorithm intended for still-image photos. It is a more elaborate version of the normal JPEG algorithm. It is growing in popularity thanks to industry backing and might reach a greater industry adoption in the future, which makes it interesting for use in motion pictures.

Is it suitable to use a still-image method in video surveillance, though? Will the JPEG XR standard prove adequate for the special needs of video surveillance? Will it beat current still-image methods? How does JPEG XR compare from a hardware point of view? How suitable is the standard for hardware implementation? The intention of this thesis is to determine the answers to these questions.

1.2 Purpose and Goal

The purpose of this thesis is to examine the suitability of JPEG XR in the context of network surveillance, and to profile it to better suit hardware implementation.

There are two goals in this thesis. One is to deliver a comparative evaluation of the motion-JPEG XR versus the motion-JPEG currently in use, both in terms of objective and subjective way. The other is to implement a prototype hardware JPEG XR encoder for further video surveillance purpose research.

The comparative evaluation focuses on the performance improvement mainly in the Image Quality of the motion-JPEG XR versus the motion-JPEG, given a roughly equivalent Compression Ratio. The implementation of hardware JPEG
XR prototype encoder focuses on accelerating the encoding process to a certain throughput level with reasonable silicon area and power consumption.

1.3 Methodology

Several steps are applied to examine the suitability of the JPEG XR standard from several angles.

Initially, a requirement analysis of the video surveillance application area is carried out. At this point, the intention is to find out more about the special needs of surveillance cameras. Like what is the image quality needed, the bit-rate and power needed.

Later on, the JPEG XR standard is carefully investigated and comparatively evaluated. The standard has a lot of setting options, optimized setting for video surveillance purpose is chosen to do the evaluation. Objectively, the Image Quality, in terms of PSNR, is compared between JPEG and JPEG XR given a equal Compression Ratio. Blocking effect, flickering effect are compared subjectively. Some software and surveillance scenario video streams are used in evaluation.

Finally, a prototype JPEG XR hardware encoder is implemented based on the optimized setting used in evaluation phase. An industry demands Encoding Throughput is achieved. The Silicon Area and Power Consumption is measured by an ASIC Synthesis tool. The System Integration issue of the hardware to Axis ARTPEC-X SOC platform is also discussed.

1.4 Chapter Overview

Five chapters are included in this report:

- **Chapter 1** provides general intention of this thesis.
- **Chapter 2** describes the JPEG XR technical details that includes the theory background and how each compression step is done. Since there are a lot of optional configurations, only the optimized part for video surveillance is introduced.
- **Chapter 3** describes how evaluation is done. Both objective and subjective results are given and compared.
- **Chapter 4** describes the implementation of JPEG XR encoder in hardware. Each hardware module architecture is presented. Synthesize results are given and analyzed. System Integration is discussed and performed.
- **Chapter 5** provides a conclusion for this thesis.
Chapter 2

JPEG XR Theory

The JPEG XR compression standard has a number of options. In this report, we only discuss the encoding part of JPEG XR since encoder is more important than decoder in a surveillance camera. Parameters and options discussed in this chapter are optimized to fit the requirement of video surveillance. Each phase of the image data flow will be specified in the following sections.

2.1 Overview

JPEG XR uses some advanced features that compresses image data in an efficient manner which keeps the same compression ratio but has better PSNR compared to JPEG. Several steps are employed to compress an image as shown in Figure 2.1.

- At the beginning, a raw image is input to the color conversion module that converts the original image color to another color space which is better suited for downsampling (Introduced in Section 2.2). In this part, we choose YUV color space in a downsampling ratio of YUV420.

- In addition, each channel of the image is divided into 3 levels of sub-image which is tiles, macroblocks and blocks. The tile size is highly related to the use of memory and compression results and therefore should be carefully selected (Introduced in Section 2.3). A macroblock is a predefined 16×16 pixels matrix while block size is usually 4×4 but it could also be 2×4 or 2×2 for special cases.

- After partitioning, an optional two-stage pre-filtering process called Picture Overlap Transform (POT) is performed to eliminate the artifacts in the boundary between blocks (Introduced in Section 2.4.1). Notice that the two stages filtering are interlaced with the two-stage frequency transform.
Figure 2.1: The data flow of JPEG XR encoding
• Later on, a two-stage frequency transform called Picture Core Transform (PCT) is employed to transform the image data from spatial domain to frequency domain (Introduced in Section 2.4.2). This divides the image into DC, Low Pass, High Pass three bands. DC component is permuted in the most top-left corner of a macroblock while LP and HP are distributed from left to right and top to bottom.

• To remove components with lesser importance\(^1\) to the visible image quality, quantization is introduced to divide the pixel coefficients (Introduced in Section 2.5). Quantization parameters (QP) could give a great impact on the image quality. If set too high, a significant amount of pixel coefficient will be lost which lead to lossy compression.

• Moreover, Adaptive Coefficient Prediction is used to remove quantized coefficients redundancy among macroblocks and blocks that reduce the storage (Introduced in Section 2.6). In most situations, prediction is used when the inter-block redundancy is strong.

• Adaptive scan is used to convert a two-dimensional matrix to a one-dimensional array so that it could be entropy encoded (Introduced in Section 2.7). Other than the traditional Zig-zag fixed scan order in JPEG, the scan order of adaptive scan is quite dynamic. It depends on the prediction direction and a statistical average and is updated in real-time. The most probable non-zero coefficients are scanned first and the probability is counted by numbers of non-zero coefficient.

• Finally, Adaptive entropy encoding is designed to further reduce the storage (Introduced in Section 2.8). Different from JPEG, JPEG XR encode DC, LP, HP frequency bands separately. Not all values are entropy encoded.

2.2 Color Conversion

2.2.1 Color Space

RGB color space is the most common way nowadays to represent color in computer display. One color is decomposed into red, green and blue components and stored separately. The drawback of RGB color space is that the three components have equal weights which is not so good for compression. The alternative triplet composed color space called YCbCr \([16]\) has advantage for compression. \(Y\) represents Luminance\(^2\) which indicates the intensity of a picture while \(CbCr\) represents chrominance\(^3\) which describes how colorful a picture is. \(Cb\) gives the Blueness and \(Cr\) gives

\(^1\)The human eyes are usually less sensitive to high frequency components \([17]\).
\(^2\)Luma for short
\(^3\)Chroma for short
the redness. Research has proven that human eyes are more sensitive to Luma components [16] which gives an unequal importance for these three components. Then we could down-sample Cb and Cr components to reduce storage without visibly decreasing the image quality. JPEG uses a linear transformation from RGB to YCbCr specified as follow:

\[
Y = 0.299R + 0.587G + 0.114B \tag{2.1}
\]

\[
Cb = -0.1687R - 0.3313G + 0.5B + 2^{P_s-1} \tag{2.2}
\]

\[
Cr = 0.5R - 0.4187G - 0.0813B + 2^{P_s-1} \tag{2.3}
\]

In Equations 2.1–2.3, \( P_s \) denotes the precision of sample like \( P_s = 8 \) means that each color component has 8 bits. The transform is a little bit lossy due to roundoff error. JPEG XR specifies a lossless color space conversion, given by:

\[
Y = G + \left\lfloor \frac{R - G + \left\lceil \frac{B - R}{2} \right\rceil}{2} \right\rfloor \tag{2.4}
\]

\[
Cb = -\left[ R - G + \left\lceil \frac{B - R}{2} \right\rceil \right] \tag{2.5}
\]

\[
Cr = B - R \tag{2.6}
\]

JPEG XR is an integer-based standard. Equations 2.4–2.6 do not contain the rounding operation, therefore the inverse operation can be performed in decoder without loss.

### 2.2.2 Down Sampling

Down sampling is the synonym of sub sampling. The reason we use YCbCr color space is that Cb and Cr can be down-sampled without visible quality degrade. JPEG XR support YUV444, YUV422 and YUV420 down sampling ratio\(^4\). It means, for example, YUV444, has a horizontal \times vertical block number of Y: 4 \times 4; U: 4 \times 4, V: 4 \times 4 in each macroblock. In this way, we can have Table 2.1:

<table>
<thead>
<tr>
<th>Down sampling ratio</th>
<th>Y plane size</th>
<th>U plane size</th>
<th>V plane size</th>
<th>YUV ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>YUV444</td>
<td>4 \times 4</td>
<td>4 \times 4</td>
<td>4 \times 4</td>
<td>1:1:1</td>
</tr>
<tr>
<td>YUV422</td>
<td>4 \times 4</td>
<td>2 \times 4</td>
<td>2 \times 4</td>
<td>2:1:1</td>
</tr>
<tr>
<td>YUV420</td>
<td>4 \times 4</td>
<td>2 \times 2</td>
<td>2 \times 2</td>
<td>4:1:1</td>
</tr>
</tbody>
</table>

**Table 2.1:** Down sampling ratio of YUV color space

---

\(^4\)Monochrome, RGB, CMYK, n-component formats are also supported but will not be discussed here.
In video surveillance application, low bit rate is preferable if the image quality is acceptable. Compared to YUV444, YUV420 only has 50% of the data and then is utilized in this report.

2.3 Image Partitioning

JPEG XR standard divides an image into small parts so that they can be processed one by one more easily. As in Figure 2.2, three hierarchies are presented to divide an image. They are tiles, macroblocks and blocks.

2.3.1 Tiles

Tile size has to be decided at the beginning of JPEG XR compression since it is largely related to the hardware architecture [2]. Each tile in an image is processed independently like a small image. Dividing the image into small tiles has the following advantages:

- **Flexibility**: Tiles in the same image could have different process configurations. For example, different quantization parameters, different adaptive scan orders and different entropy coding tables. This is a good feature when people are only interested in certain parts of an image.

- **Memory saving**: In adaptive prediction phase, one row of coefficients have to be saved. Smaller tile means less horizontal length which occupies less memory. It reduces either the memory accessing (if using main memory) or silicon area (if using on-chip memory).
• **Robustness**: Tile gives the system robustness. If one error occurs in one tile, the others will not get affected.

These features make JPEG XR more flexible and suitable for hardware implementation especially when memory size constraint is in high priority.

Nevertheless, using too many tiles will decrease the compression efficiency since redundancy among tiles can not be extracted. The compression ratio decreases when number of tile increases. It is a tradeoff between hardware implementation and compression efficiency. To achieve the best compression ratio, one whole image is set as one tile in this report.

### 2.3.2 Macro Blocks

Macroblock is a basic data unit in JPEG XR. The size of a macroblock is predefined by the standard. In Luma plane, each macroblock consists of \(16 \times 16\) pixels in which 1 is DC component, 15 are LP component, 240 are HP component. In Chroma plane of YUV422 and YUV420, each macroblock is \(8 \times 16\) and \(8 \times 8\) respectively. PCT, POT, quantization, prediction stages are all designed to process one macroblock each time.

### 2.3.3 Blocks

Block is the smallest data unit in JPEG XR. Each block consists of \(4 \times 4\) pixels and one macroblock consists of \(4 \times 4\) blocks.

### 2.4 Lapped Bi-orthogonal Transform

The Lapped Bi-orthogonal Transform (LBT) [18] is a novel manner of image processing combined in JPEG XR. Why LBT is superior will not be discussed in this report, interested readers can easily find material by Googling. LBT consists of two parts: Pre-filtering operation POT introduced in Section 2.4.1 and Frequency transform operation PCT introduced in Section 2.4.2.

#### 2.4.1 Pre-filtering

The pre-filtering scheme used in JPEG XR compression is called Picture Overlap Transform (POT). Frequency transform taking place in blocks can not take advantage in solving redundancy across block boundary. Therefore some artifacts will come out in the image compressing. POT is a new feature in JPEG XR that could somehow ”smooth” the artifacts on block boundaries. POT has three overlap levels: non-overlapping, one-level overlapping and two-level overlapping.
• Non-overlapping does not use any pre-filtering operation and is the fastest way but has low compression ratio.

• One-level overlapping has one stage POT. It has better compression ratio in the price of more complex computation and more processing time.

• Two-level overlapping mode has two-stage POT means highest computation complexity, memory usage and processing time. But at the same time, we get best PSNR under relatively low bit-rate.

Compared to PCT, which is only performed inside each $4 \times 4$ pixels block, POT is more complex and memory consuming since it performs not only inside each macroblock but also between the boundary of adjacent macroblocks. The POT consists of two major operations: one is $4 \times 4$ Pre-filtering module specified in Figure 2.3, the other is $4$-point Pre-filtering module.

• $4 \times 4$ Pre-filtering: The $4 \times 4$ Pre-filtering is applied to all macroblock junction\(^5\) and fully interior areas. We can see the POT insists on 4 sub-operations: $2 \times 2$ Hadamard Transform $T_H$, 2-point Forward Rotation $T_R$, 2-point Forward Scaling $T_S$, $2 \times 2$ Odd odd Transform $T_{oddodd}$. These sub-operations will be introduced later.

• 4-point Pre-filtering: Linear 4-point pre-filters are applied to the edge straddling $2 \times 4$ and $4 \times 2$ boundary areas of the image. It has 2-point Forward Rotation $T_R$, 2-point Forward Scaling $T_S$ and a few micro-operations\(^6\).

\(^5\) $12 \times 4$ in top and bottom boundary, $4 \times 12$ in left and right boundary, $12 \times 12$ in interior area

\(^6\) Like add, subtract, multiplication, shifting
Hadamard Transform

The Hadamard transform is an example of a generalized class of Fourier transforms. It can be regarded as being built out of size-2 discrete Fourier transforms, and is in fact equivalent to a multidimensional DFT of size $2 \times 2 \times 2 \times \ldots \times 2 \times 2$.

$H_m$ is a $2^m \times 2^m$ Hadamard transform matrix. This matrix (scale by a normalization factor), that transforms $2^m$ real numbers $x_n$ into $2^m$ real numbers $X_k$. We define the $1 \times 1$ Hadamard transform $H_0$ by the identity $H_0 = 1$, and then define $H_m$ for $m > 0$ by:

$$H_m = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{pmatrix}$$

(2.7)

where the $\frac{1}{\sqrt{2}}$ is a normalization that is sometimes omitted. Thus, other than this normalization factor, the Hadamard matrices are made up entirely of 1 and -1.

Some examples of the Hadamard matrices are like:

$$H_0 = +1$$  \hspace{1cm} (2.8)

$$H_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^7 \hspace{1cm} (2.9)$$

Equation 2.9 describes a 2-point Hadamard operator. The $2 \times 2$ Hadamard transform is developed by taking the Kronecker product of the 2-point Hadamard operator with itself as shown in Equation 2.10.

$$T_H = H_1 \otimes H_1 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

(2.10)

Then we get operator $T_H$. Symbol $\otimes$ stands for Kronecker product. According to standard document [11], $T_H$ can be described in pseudo-code as shown in Listing 2.1.

Forward Rotation

Forward Rotation is an operation that rotates input vectors in the plane by an angle $\theta$. The related rotation matrix is in Equation 2.11:

$$T_R = \begin{pmatrix} -\cos \theta & \sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

(2.11)

$\theta$ in Equation 2.11 equals to $\frac{\pi}{8}$. Operator $T_R$ can be remapped the to pseudo-code in Listing 2.2 [11]:

\footnote{This $H_1$ is precisely the size-2 DFT. It can also be regarded as the Fourier transform on the two-element additive group of $\mathbb{Z}/(2)$.}
T2x2hEnc(iCoeff[])
{
    iCoeff[0] += iCoeff[3]
    iCoeff[1] -= iCoeff[2]
    valT1 = iCoeff[3]
    valT2 = iCoeff[2]
    iCoeff[2] = ((iCoeff[0] - iCoeff[1]) >> 1) - valT1
    iCoeff[3] = valT2 + (iCoeff[1] >> 1)
    iCoeff[1] += iCoeff[2]
    iCoeff[0] -= (iCoeff[3] * 3 + 4) >> 3
}

Listing 2.1: Pseudo-code of $2 \times 2$ Hadamard transform

FwdRotate(iCoeff[])
{
    iCoeff[1] -= (iCoeff[0] + 1) >> 1
    iCoeff[0] += (iCoeff[1] + 1) >> 1
}

Listing 2.2: Pseudo-code of 2-point Forward Rotation

Forward Scaling

Forward scaling scales the input vector by a factor $S$. This operation can be described in Equation 2.12:

$$T_S = \begin{pmatrix}
S^2 & S^2 & 1 & 1 \\
S^2 & S^2 & 1 & 1 \\
1 & 1 & S^{-2} & S^{-2} \\
1 & 1 & S^{-2} & S^{-2}
\end{pmatrix} \tag{2.12}$$

The operator $T_S$ can be mapped to pseudo-code in Listing 2.3.

Odd odd Transform

$2 \times 2$ Odd odd transform is an operation that developed by taking Kronecker product of a two-point forward rotation of Equation 2.11 with itself. It can be described in

FwdScale(iCoeff[])
{
    iCoeff[1] -= (iCoeff[0] * 3 + 0) >> 4
    iCoeff[1] -= (iCoeff[0] >> 7)
    iCoeff[1] += (iCoeff[0] >> 10)
    iCoeff[0] -= (iCoeff[1] * 3 + 0) >> 3
    iCoeff[1] = (iCoeff[0] >> 1) - iCoeff[1]
    iCoeff[0] -= iCoeff[1]
}

Listing 2.3: Pseudo-code of 2-point Forward Scaling
Lapped Bi-orthogonal Transform

```plaintext
FwdT0dd0dd(iCoeff[ ]) {
  iCoeff[3] += iCoeff[0]
  valT1 = iCoeff[3] >> 1
  valT2 = iCoeff[2] >> 1
  iCoeff[0] -= valT1
  iCoeff[1] += valT2
  iCoeff[0] += (iCoeff[1] * 3 + 4) >> 3
  iCoeff[1] -= (iCoeff[0] * 3 + 2) >> 2
  iCoeff[0] += (iCoeff[1] * 3 + 6) >> 3
  iCoeff[1] -= valT2
  iCoeff[0] += valT1
  iCoeff[2] += iCoeff[1]
  iCoeff[3] -= iCoeff[0]
}
```

**Listing 2.4: Pseudo-code of 2 × 2 Odd odd Transform**

Equation 2.13.

\[
T_{\text{oddodd}} = T_R \otimes T_R = \begin{pmatrix}
-\cos \theta & \sin \theta \\
\sin \theta & \cos \theta
\end{pmatrix} \otimes \begin{pmatrix}
-\cos \theta & \sin \theta \\
\sin \theta & \cos \theta
\end{pmatrix}
\]  

(2.13)

The \( \theta \) is \( \frac{\pi}{8} \). According to the standard document this operation can be mapped to pseudo-code in Listing 2.4 [11].

### 2.4.2 Frequency Transform

The reason why we transform spatial data into frequency domain is: Human eyes are usually less sensitive to high frequency component so that it could be removed to reduce overhead. Frequency transform could extract frequency components which are uniformly distributed in spatial data and put same frequency components together. Once the high frequency component has been put together, it is easy to remove them by using quantization. That is why the quantization parameters in high frequency band is usually much larger than the low frequency and DC bands.

The frequency transform function used in JPEG XR is called Photo Core Transform (PCT). After pre-filtering\(^8\), it transforms an image from spatial domain into frequency domain\(^9\), like the old Discrete Cosine Transform (DCT) from JPEG does. The differences are:

- PCT defines three clear frequency bands DC, Low pass, High pass which DCT does not. The advantage that clearly define three bands is they could be processed separately and is more flexible.

\(^8\)If there is pre-filtering
\(^9\)Different frequency components are put together
JPEG XR usually uses a $4 \times 4$ pixels block as the minimum processing unit which is more fine-grained than the $8 \times 8$ block used in JPEG.

PCT has only one module which is specified in Figure 2.4. It is simpler than POT since the input is always a $4 \times 4$ block. 3 sub-operations $T_H$, $T_{odd}$, $T_{oddodd}$ are employed in PCT.

The $4 \times 4$ PCT module processes one block in 2 phases specified in Figure 2.5. In the first phase, $2 \times 2$ Hadamard transform $T_H$ is applied to the corners, boundaries and interior areas of the block in parallel. In the second phase, $T_H$ is applied to the top-left corner while Odd transform $T_{odd}$ is applied to the top-right and bottom-left. The bottom-right corner is processed by Odd odd Transform $T_{oddodd}$. Later, these sub-operations will be introduced.

**Hadamard Transform in PCT**

$T_H$ has the same theoretic definition as in Pre-filtering, but the process is slightly different in standard document. The pseudo-code is shown in Listing 2.5.
Figure 2.5: Two phases Sub-operations on each block

Listing 2.5: Pseudo-code of $2 \times 2$ Hadamard transform

```plaintext
def T2x2h(iCoeff[], valRound):
    iCoeff[0] += iCoeff[3]
    iCoeff[1] -= iCoeff[2]
    valT1 = ((iCoeff[0] - iCoeff[1] + valRound) >> 1)
    valT2 = iCoeff[2]
    iCoeff[2] = valT1 - iCoeff[3]
    iCoeff[3] = valT1 - valT2
    iCoeff[0] -= iCoeff[3]
    iCoeff[1] += iCoeff[2]
```

Listing 2.5: Pseudo-code of $2 \times 2$ Hadamard transform
Odd Transform

Odd Transform \( T_{\text{odd}} \) specified in Equation 2.14 is a Kronecker product of a 2-point forward rotation operator of Equation 2.11 with a 2-point Hadamard operator of Equation 2.9.

\[
T_{\text{odd}} = T_R \otimes H_1 = \begin{pmatrix}
-\cos \theta & \sin \theta \\
\sin \theta & \cos \theta
\end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix}
1 & 1 \\
1 & -1
\end{pmatrix} 
\quad (2.14)
\]

\( T_{\text{odd}} \) can be mapped into pseudo-code as shown in Listing 2.6.

Odd odd Transform in PCT

2 \times 2 Odd odd Transform \( T_{\text{oddodd}} \) has the same theoretic definition as in Pre-filtering of Equation 2.13, but the process is slightly different in standard document. The pseudo-code is shown in Listing 2.7.

Detailed image processing areas will be specified from Section 2.4.3 to Section 2.4.6.

2.4.3 First Stage Pre-filtering

The first stage Pre-filtering “smooth” the boundary among macroblocks within each tile. The processing area is indicated in Figure 2.6.

The 4 \times 4 Pre-filtering is applied to sub-figure (K), (L), (N) while 4-point pre-filtering is applied to the other parts.

- (A), (B), (C), (G) only take place in the left-most, right-most, top-most, bottom-most edges of a tile respectively.
Figure 2.6: First stage Pre-filtering area
Listing 2.7: Pseudo-code of $2 \times 2$ Odd odd Transform

- (E), (F), (I), (J) only take place in the Top-left-most, Top-right-most, Bottom-left-most, Bottom-right-most corners of a tile respectively.

- (D), (H), (L), (N), (O), (P) take place in the boundaries across macroblocks within a tile. They need extra coefficients from neighbor macroblocks.

- (K) take place in the fully interior $12 \times 12$ area of a macroblock.

2.4.4 First Stage Frequency Transform

After first stage pre-filtering, PCT is applied to extract frequency components in macroblocks as shown in Figure 2.7.

One $16 \times 16$ pixels macroblock is decomposed into 16 $4 \times 4$ pixel blocks. Each of the block is transformed by a $4 \times 4$ PCT module which is specified in Figure 2.4. After transform, the DC component of each block has been put in the most top-left corner of the block.

2.4.5 Second Stage Pre-filtering

The second stage pre-filtering is similar to the first stage but only concentrates on the $4 \times 4$ DC component of each macroblocks within a tile. The processing area is indicated in Figure 2.8.

The 4-point pre-filtering is applied to all the areas indicated in the sub-figures.

- (B), (C), (E), (F) only take place in the top-left-most, top-right-most, bottom-left-most, bottom-right-most corners of a tile respectively.
Figure 2.7: *First stage frequency transform area*

Figure 2.8: *Second stage pre-filtering area.*
• (A), (D), (G), (H), (I) take place in the boundaries across macroblocks within a tile. They need extra coefficients from neighbor macroblocks.

### 2.4.6 Second stage Frequency Transform

Then in the second stage PCT, all of the 16 DC components are taken out to reform a new $4 \times 4$ block\(^{10}\). The same scheme is performed on this block and generates 1 DC and 15 LP coefficients as shown in Figure 2.9. The remaining 250 coefficients are HP components. From now on, the LBT is completed.

### 2.5 Quantization

After frequency transform, the less important high frequency components have been collected together and can be removed by quantization. In quantization, the transform coefficients are divided by a quantization parameter and rounded to an integer value called the quantization value. It is at this stage in the compression algorithm that distortion can be introduced. This happens if the quantization parameter is greater than one. JPEG XR supports both lossless and lossy compression which is decided in this stage. If lossy, the quantization result value can not be reconstructed in the decoder. This happened mostly in HP band.

### 2.5.1 Quantization Parameters

Instead of a quantization table which is used in JPEG, JPEG XR uses Quantization Parameters (QP). The value of QP is ranged from 1 to 255. It could variant across frequency bands, color planes and different tiles. The wide range of QP gives the encoder accuracy and flexibility to control the trade-off between image quality and bit throughput. One thing needed to be taken into account is in the later prediction

\(^{10}\)Not really put all DC together, they are in the original place.
if (iQP < 16) {
    iMan = iQP;
    iExp = iScaledShift;
} else {
    iMan = 16 + (iQP % 16);
    iExp = ((iQP >> 4) - 1) + iScaledShift;
}
quant = iMan << iExp;

Listing 2.8: Pseudo-code for QuantMap

static int quantize_dc(int value, int quant)
{
    int sign = 1;
    if (value < 0) {
        sign = -1;
        value = -value;
    }

    int offset = quant >> 1;
    return sign * (value + offset)/quant;
}

Listing 2.9: Pseudo-code for the DC quantization

stage, prediction can only be performed between two coefficients that used the same QP. Video surveillance camera in most of time prefers simple computation which both lower the power and streaming latency. So in this report, one QP is used for one whole image.

Since different color planes have unequal importance, QP will not be used directly but be remapped as shown in Listing 2.8. The variable “quant” is then used to derive the quantization values. The pseudo-code is in Listing 2.9 and 2.10 for the DC and LPHP bands respectively.

In this case, the quantization algorithm uses division which is very expensive for hardware implementing. To reduce hardware cost, the reciprocal of quant are pre-calculated and stored in a look-up table indexed by the quantization value. Then the division can be replaced by multiplication and shifting. The specific implementation will be discussed in Chapter 4.

2.6 Adaptive Prediction

In most cases, pixels/blocks of small area in an image have relatively equal values. Like a blue sky, every pixel is about the same with a slight difference. We say there
static int quantize_lphp(int value, int quant)
{
    int sign = 1;
    if (value < 0) {
        sign = -1;
        value = -value;
    }

    int offset = (quant*3 + 1) >> 3;

    return sign * (value + offset)/quant;
}

Listing 2.10: Pseudo-code for the LPHP quantization

is large redundancy between blocks in an image because of similar pixels/blocks always being put together. Therefore, it is more efficient to store the differences than to store the actual values. More zeros could be generated which is more fit for entropy coding. The key point of prediction is to use current coefficient value subtracts previous neighbor coefficient value, and then store the residual.

Adaptive prediction is used in JPEG XR. The prediction performs only when the system makes sure two neighbor macroblocks have big enough similarity. It is called “adaptive” because it dynamically changes the predict direction to one of the neighbor macroblocks which has the biggest similarity with the current macroblock. The predict direction could be from left, top or top-left and the prediction scheme is variant across different frequency bands. The detailed DC, LP and HP prediction will be introduced in the following sections.

2.6.1 DC Prediction

The DC prediction is done in between macroblocks. It can happen from top, left and top-left. The DC level of the one deemed most similar is selected and the difference between that one and the present one is calculated. The algorithm that gives the predict direction is shown in Listing 2.11.

Function labs() in Listing 2.11 means get absolute value. We can see the calculation of horweight and vertweight involves all three color planes. Figure 2.10 gives an example how DC prediction happens between macroblocks. Since there is only one DC coefficient in each macroblock, four values represent four macroblocks. In the example, value 54 is the DC value in current macroblock. Comparing among the three directions, value 53 in the left has the biggest similarity and then is chosen as the subtrahend. The result value 1 is finally passed on to the next stage, adaptive scan.
2.6.2 LP Prediction

As DC prediction, LP prediction also happens between macroblocks. But it has only two predict directions which are from left and top. LP predict direction is decided by the DC prediction mode if the quantization parameters are the same for the two macroblocks involved, otherwise it is skipped altogether. If DC predict direction is from top-left, the prediction will not perform in LP. The decision of LP prediction mode uses the same algorithm as in Listing 2.11. Figure 2.11 gives an example how LP prediction works.

2.6.3 HP Prediction

Different from the DC and LP prediction, the HP prediction performs within each macroblock. The intention is to extract and save the difference among blocks inside each macroblock. The HP predict direction could be from left, top or not at all. It is decided by the LP coefficients in current macroblock. The algorithm is described

```c
calculate_dc_mode()
{
  if (downsampling ratio is YUV420)
    scale = 8;

  horweight = labs(topleft_y - left_y)*scale +
             labs(topleft_u - left_u) + labs(topleft_v - left_v);

  vertweight = labs(topleft_y - top_y)*scale +
              labs(topleft_u - top_u) + labs(topleft_v - top_v);

  if ((horweight*4) < vertweight)
    predict from top;

  if ((vertweight*4) < horweight)
    predict from left;

  otherwise
    predict from top_left;
}
```

Listing 2.11: Pseudo-code for DC prediction

Figure 2.10: Example of DC-prediction
in Listing 2.12.

The LP coefficients $L_{P_y}(1)$, $L_{P_y}(2)$, $L_{P_y}(3)$, $L_{P_y}(4)$, $L_{P_y}(8)$, $L_{P_y}(12)$, $L_{P_u}(1)$, $L_{P_u}(2)$, $L_{P_v}(1)$, $L_{P_v}(2)$ correspond to the pixels in Figure 2.12.

Figure 2.13 indicates the way HP prediction performs within a macroblock.

### 2.7 Adaptive Scan

The purpose of scanning is reordering the two-dimensional pattern into a one-dimensional array. The coefficients order needs to be adjusted so that zero coefficients can be put together as much as possible. And then it can be efficiently entropy coded. In JPEG, fixed zigzag scan order is utilized while JPEG XR uses a dynamic scan order. The scan order keeps on changing to adapt the changing distribution of non-zero coefficients so that zero coefficients can be always put together. Note that adaptive scan only performs within blocks, macroblocks are scanned separately in a traditional raster scan way.

#### 2.7.1 DC Scan

Since there is just one single DC component per macroblock it does not need to be adaptively scanned. The DC component uses normal raster scan order.
calculate_hp_mode()
{
    horweight = 0;
    vertweight = 0;

    /* Add up the LP magnitudes along the top edge */
    horweight = LP_y(1) + LP_y(2) + LP_y(3);

    /* Add up the LP magnitudes along the left edge */
    vertweight = LP_y(4) + LP_y(8) + LP_y(12);

    if(downsampling_ratio is YUV420)
    {
        horweight += LP_u(1) + LP_v(1);
        vertweight += LP_u(2) + LP_v(2);
    }

    if (horweight * 4 < vertweight)
        predict from left;
    if (vertweight * 4 < horweight)
        predict from top;
    otherwise
        no prediction;
}

Listing 2.12: Pseudo-code for HP prediction

Figure 2.12: LP block decides the predict mode of HP prediction
Figure 2.13: Example of HP prediction

(a) HP predict from left

(b) HP predict from top
2.7.2 LP Scan

The LP components are scanned adaptively. To achieve this, a scan order and a scan counter are utilized to update the scan order after each macroblock. LP scan order and scan counter have initial values given in Table 2.2.

<table>
<thead>
<tr>
<th>Scan Order</th>
<th>4</th>
<th>1</th>
<th>5</th>
<th>8</th>
<th>2</th>
<th>9</th>
<th>6</th>
<th>12</th>
<th>3</th>
<th>10</th>
<th>13</th>
<th>7</th>
<th>14</th>
<th>11</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan Counter</td>
<td>32</td>
<td>30</td>
<td>28</td>
<td>26</td>
<td>24</td>
<td>22</td>
<td>20</td>
<td>18</td>
<td>16</td>
<td>14</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 2.2: Initial scan order of LP

The algorithm will check each of the 15 positions in an LP block. If a non-zero coefficient is found in a particular position, the corresponding scan counter will be increased by 1. If zero, the scan counter keeps its original value. There is an example. If the initial status is like in Table 2.2, after scanning three LP blocks given in Figure 2.14, the scan state should be changed to Table 2.3.

Figure 2.14: Example: Three LP blocks need to be scanned

Table 2.3 gives the updated scan order. Then in the next LP block, a new scan order will be used. The scan status will be reset to Table 2.2 after scanning each of the 16 LP blocks.

2.7.3 HP Scan

The HP component is also adaptively scanned. Since there are 16 HP blocks in a macroblock, we should at first decide the scan order of blocks in each macroblock. The block scan order is indicated in Figure 2.15.

Then in each HP block, there are two initial scan orders. Which one should be used depends on the HP predict direction introduced in Section 2.6.3. These two
scan orders are given in Table 2.4.

If the HP predict direction is from top, vertical scan order is used. Otherwise horizontal scan order is used. The scan counter performs the same as in LP scan.

### 2.8 Adaptive Entropy Coding

In information theory, the entropy means the amount of information contained in a message. If the entropy is low, we need less space to store the message. Ideally, it would be preferable that the bit distribution in the bit stream is first a “1” followed by as many trailing zeros as possible. What the previous stages have done is to change the distribution of the bit stream and improve the entropy coding efficiency.

JPEG XR uses adaptive entropy coding [12] which has several advanced features to reduce data overhead:

- First of all, not all of the bits need to be encoded. A process called normalization is used to decide which part of the bits should be kept as plain or be

---

11Include no prediction and predict from left
encoded. The plain bits in the DC and LP bands are called refinement while in the HP band they are called flexbits.

- After the bits are separated, the plain bits will be directly written into the bitstream while the rest of the bits need to be encoded by using a combination of both Run Level Encoding (RLE) and the Variable Length Coding (VLC).

Figure 2.16 gives the general architecture of adaptive entropy coding. The process is divided into three frequency bands: DC, LP, and HP. Each time the process is within one macroblock and it can be pipelined with previous stages. Since each macroblock has only one DC component, the DC band does not have Run Level encoding and some parts of the VLC coding. This makes the DC part simpler than the other two. Moreover, two data dependent loops are applied in this architecture. One is the normalization modelbits, the other is the adaptive VLC code table selection. The final bit stream consists of three parts in each frequency band. They are Coded Bit Pattern (CBP), VLC coded bits and plain bits. CBP signals whether or not any non-zero coefficients are present in the corresponding band for this macroblock. It is a bit mask with a bit for each YUV color plane. The actual coefficients are divided into VLC coded bits and plain bits by normalization. The details will be introduced in the following subsections.

### 2.8.1 Coefficients Normalization

To support high dynamic range display, the pixel sample of JPEG XR has a high bit-depth from 8 bits up to 32 bits. In this report, 16 bits/sample is used. It requires a huge VLC code table if all of the long bits need to be encoded. Thus, it is more preferable to efficiently encode certain parts of the bit stream and store the rest as plain bits. In contrast, if all bits are stored as plain bits, the image size will not be reduced even though the VLC code table is small. It is actually a tradeoff between the size of VLC code table and bit stream. In the standard document [11], the number of plain bits is called modelbits. Table 2.5 gives the initial number of modelbits in different frequency bands.

<table>
<thead>
<tr>
<th>Horizontal Scan Order</th>
<th>4</th>
<th>1</th>
<th>5</th>
<th>8</th>
<th>2</th>
<th>9</th>
<th>6</th>
<th>12</th>
<th>3</th>
<th>10</th>
<th>13</th>
<th>7</th>
<th>14</th>
<th>11</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical Scan Order</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>12</td>
<td>15</td>
<td>13</td>
<td>10</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>Scan Counter</td>
<td>32</td>
<td>30</td>
<td>28</td>
<td>26</td>
<td>24</td>
<td>22</td>
<td>20</td>
<td>18</td>
<td>16</td>
<td>14</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 2.4:** Two initial scan orders of HP
Figure 2.16: General Adaptive Entropy Coding Architecture
Table 2.5: Initial Model bits variant among frequency bands

<table>
<thead>
<tr>
<th>Frequency band</th>
<th>Model bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>8</td>
</tr>
<tr>
<td>LP</td>
<td>4</td>
</tr>
<tr>
<td>HP</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2.17 gives an example how VLC coded bits and plain bits are separated. Assume each coefficient has 16 bits. If the number of modelbits equals 8, the 8 LSB of the coefficient will be kept as plain bits and directly written to the bit stream. The remaining 8 MSB will later be encoded using the VLC table.

As shown in Figure 2.16, there is a data dependent loop in normalization. The number of modelbits is not constant when encoding. It changes dynamically according to the feedback from the previous coefficient in order to obtain the coding efficiency. To achieve this, a variable called Lap\_mean is used to measure the variance of the coefficients. The Lap\_mean accumulates when the coefficient VLC coding part bits (MSB) is a non-zero number after normalization. Once Lap\_mean reaches a certain bound, it will be reset to zero and the modelbits will increase or decrease. This feature guarantees the later VLC coding table is a reasonable size.

2.8.2 Run Level Encoding

Run level encoding is a subset of run length encoding. These encodings are now widely used in image compression standard. The basic idea is to encode information about runs of identical numbers. An example of Run Length Encoding is shown in Figure 2.18. The coded bits format is like the number of runs followed by the level. In this example, there is big redundancy in the original bit stream. After coding, 19 bytes have been compressed into 8 bytes.

Run Level Encoding has similar concept as Run Length Encoding but is usually used under a situation that the encoder produces long runs of zeros between non-zero numbers. Here is an example given in Figure 2.19. The number of runs only
Figure 2.18: An example of Run Length Encoding

represents how many zeros appear between non-zero levels. Since there is huge amount of zeros from the previous stage, Run Level Encoding is highly efficient. In JPEG XR, Run Level Encoding processes in each $4 \times 4$ blocks which means 15 coefficients per time.

Figure 2.19: An example of Run Level Encoding

2.8.3 Adaptive VLC Encoding

Adaptive VLC Encoding encodes the bit pattern by using VLC coding tables. As shown in Figure 2.16, except for the DC band, LP and HP bands both have three parts of bits needed to be encoded. They are Index Coding, Level Coding and Run Coding.

Index Coding

Index contains the information which describes some coding events of Run and Level. The information will be used in decoder to reconstruct the image. There are three kinds of Index: First_Index, Index_A and Index_B.
As specified in the standard document [11], First_Index appears at the beginning\textsuperscript{12} of a Run Level Encoding output which contains 15 coefficients. It jointly codes three events shown as follows:

- The binary event of whether the run before the first non-zero coefficient is non-zero or zero as follows:
  - If (First_Index & 1) is equal to 0, this run is non-zero.
  - Otherwise this run is zero.

- The binary event of whether the magnitude of the first non-zero coefficient is equal to 1 or greater than 1 as follows:
  - If (First_Index & 2) is equal to 0, this magnitude is equal to 1.
  - Otherwise, this magnitude is greater than 1.

- The ternary event of whether the first coefficient is the last coefficient in the block, and if there are more non-zero coefficients whether the run before the next non-zero coefficient is zero or non-zero, is as follows:
  - If (First_Index >> 2) is equal to 0, the first coefficient is the last coefficient in the block.
  - Otherwise, if (First_Index >> 2) is equal to 1, the run before the next non-zero coefficient is zero.
  - Otherwise ((First_Index >> 2) is equal to 2), the run before the next non-zero coefficient is non-zero.

These three events have $2 \times 2 \times 3 = 12$ combinations which gives First_Index value ranged from 0 to 11. First_Index has 5 coding tables specified in Table 2.6. How to adaptively selected a table to encode First_Index will be introduced in subsection Adaptive VLC coding table selection.

Besides of First_Index, Index_A appears in the middle\textsuperscript{13} of a run Level encoding output. Index_A jointly codes the following two events [11]:

- The binary event of whether the magnitude of the next non-zero coefficient is equal to 1 or greater than 1, is as follows:
  - If (Index_A & 1) is equal to 0, this magnitude is equal to 1.
  - Otherwise, this magnitude is greater than 1.

\textsuperscript{12}At position 1, the position range is from 1 to 15
\textsuperscript{13}At position 2 to 14
• The ternary of event whether this coefficient is the last coefficient in the block, and if there are more non-zero coefficients, whether the run before the next non-zero coefficient is zero or non-zero, is as follows:

- If \((\text{Index}_A >> 1)\) is equal to 0, this coefficient is the last coefficient in the block.
- Otherwise, if \((\text{Index}_A >> 1)\) is equal to 1, the run before the next non-zero coefficient is zero.
- Otherwise (i.e. when \((\text{Index}_A >> 1)\) is equal to 2), the run before the next non-zero coefficient is non-zero.

These two events have \(2 \times 3 = 6\) combinations which gives \(\text{Index}_A\) value ranged from 0 to 5. \(\text{Index}_A\) has 4 coding tables specified in Table 2.7.

<table>
<thead>
<tr>
<th>Index Value</th>
<th>Table 0</th>
<th>Table 1</th>
<th>Table 2</th>
<th>Table 3</th>
<th>Table 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00001</td>
<td>0010</td>
<td>11</td>
<td>001</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>000001</td>
<td>00010</td>
<td>001</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0000001</td>
<td>00000</td>
<td>0000001</td>
<td>0000001</td>
<td>0000001</td>
</tr>
<tr>
<td>3</td>
<td>001001</td>
<td>0011</td>
<td>0001</td>
<td>0010</td>
<td>00010</td>
</tr>
<tr>
<td>4</td>
<td>010</td>
<td>010</td>
<td>010</td>
<td>011</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>001101</td>
<td>00011</td>
<td>000010</td>
<td>000001</td>
<td>000000</td>
</tr>
<tr>
<td>6</td>
<td>011</td>
<td>11</td>
<td>011</td>
<td>011</td>
<td>011</td>
</tr>
<tr>
<td>7</td>
<td>000110</td>
<td>011</td>
<td>100</td>
<td>00011</td>
<td>000011</td>
</tr>
<tr>
<td>8</td>
<td>001</td>
<td>100</td>
<td>101</td>
<td>100</td>
<td>001</td>
</tr>
<tr>
<td>9</td>
<td>001111</td>
<td>00001</td>
<td>000011</td>
<td>00001</td>
<td>000001</td>
</tr>
<tr>
<td>10</td>
<td>011</td>
<td>101</td>
<td>0001</td>
<td>101</td>
<td>0001</td>
</tr>
<tr>
<td>11</td>
<td>011</td>
<td>101</td>
<td>0001</td>
<td>101</td>
<td>0001</td>
</tr>
</tbody>
</table>

Table 2.6: \(\text{First Index VLC Coding tables}\)

<table>
<thead>
<tr>
<th>Index Value</th>
<th>Table 0</th>
<th>Table 1</th>
<th>Table 2</th>
<th>Table 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>01</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>00000</td>
<td>0000</td>
<td>0001</td>
<td>00001</td>
</tr>
<tr>
<td>2</td>
<td>001</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>00001</td>
<td>0001</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
<td>11</td>
<td>11</td>
<td>0001</td>
</tr>
<tr>
<td>5</td>
<td>0001</td>
<td>001</td>
<td>001</td>
<td>001</td>
</tr>
</tbody>
</table>

Table 2.7: \(\text{Index A VLC Coding tables}\)

Last but not least, \(\text{Index B}\) appears at the end\(^{14}\) of a run level encoding output.

\(^{14}\)At position 15
It jointly codes the following two events [11]:

- The binary event of whether the magnitude of the next non-zero coefficient is equal to 1 or greater than 1, is as follows:
  - If \((\text{Index}_B \& 1)\) is equal to 0, this magnitude is equal to 1.
  - Otherwise, this magnitude is greater than 1.

- The binary event whether this coefficient is the last coefficient in the block or if there are more non-zero coefficients, is as follows:
  - If \((\text{Index}_B >>> 1)\) is equal to 0, this coefficient is the last coefficient in the block.
  - Otherwise, the run before the next non-zero coefficient is zero.

These two events have \(2 \times 2 = 4\) combinations which gives \(\text{Index}_B\) value ranged from 0 to 3. \(\text{Index}_B\) has only one coding table specified in Table 2.8.

<table>
<thead>
<tr>
<th>Index Value</th>
<th>Table 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>111</td>
</tr>
</tbody>
</table>

*Table 2.8: \(\text{Index}_B\) VLC Coding tables*

**Level Coding**

Level Coding encodes the non-zero coefficients from the run level encoding output. It has two parts, \(\text{Abs}\_\text{Level}_\text{Index}\) and \(\text{Abs}\_\text{Level}\).

\(\text{Abs}\_\text{Level}_\text{Index}\) denotes how many bits are enough to represent the level. The value is ranged from 0 to 6. It is encoded using two VLC coding tables as shown in Table 2.9. If the level value needs to be represented using 6 or more bits, an array will give the number of extra bits available to encode the last bit of the value. See [11] for details.

After Encoding \(\text{Abs}\_\text{Level}_\text{Index}\), if \(\text{Abs}\_\text{Level}\) value can be represented in less than 6 bits, \(\text{Abs}\_\text{Level}\) value will be directly written to bitstream. Note, since it is called \(\text{Abs}\_\text{Level}\), Level value is encoded in the format of a sign and a magnitude.
Run Coding

As Level Coding encodes the level value, Run Coding encodes the runs of zero from the run level encoding output. Run Coding has conditions. It changes the code table under different Max_Run values. Max_Run value means the maximum runs of zero could appear in each $4 \times 4$ block. This value is ranged from 1 to 14. There are cases when Max_Run has different values as follows:

- If Max_Run equals 1, do nothing.
- If Max_Run equals 2, encode run value using Table 2.10.
- If Max_Run equals 3, encode run value using Table 2.11.
- If Max_Run equals 4, encode run value using Table 2.12.
- If Max_Run is greater than or equal to 5, encode run value using Run_Index Table 2.13. In this case, Run value is remapped to a smaller Run_Index value which is ranged from 0 to 4. How the remapping works is specified in [11].

Adaptive VLC Table Selection

In the previous sections, some values need to be encoded using several VLC tables. These tables are predefined and designed to suit a wide range of image statistics which provides better compression ratio. The most efficient table is selected based

<table>
<thead>
<tr>
<th>Index Value</th>
<th>Table 0</th>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>001</td>
</tr>
<tr>
<td>3</td>
<td>001</td>
<td>0001</td>
</tr>
<tr>
<td>4</td>
<td>0001</td>
<td>00001</td>
</tr>
<tr>
<td>5</td>
<td>00000</td>
<td>000000</td>
</tr>
<tr>
<td>6</td>
<td>00001</td>
<td>000001</td>
</tr>
</tbody>
</table>

Table 2.9: Abs_Level_Index VLC Coding tables

<table>
<thead>
<tr>
<th>Run Value</th>
<th>Table 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.10: Run value VLC Coding table when Max_Run = 2
on the history of previous encoded symbols. Thus it is a data dependent loop as shown in Figure 2.16.

As specified in [11], assume we have VLC code tables $T_0, T_1, ..., T_n$. The index order 0 to n is predefined based on their relative similarity. That means $T_0$ and $T_1$ shows bigger similarity to each other than the similarity between $T_0$ and $T_2$. Let $T_3$ be the default code table of symbol A. Then $T_2$ and $T_4$ are the two nearest tables which have bigger similarity to $T_3$ than others. For each value of symbol A, a metric Delta is used to estimate whether $T_2$ or $T_4$ is more efficient to encode current value instead of using $T_3$. All of the Delta values are precomputed and stored in Delta Tables. Delta Tables are variant across different symbols\(^{15}\). The Delta value is positive if the current symbol is more efficiently encoded using a new table instead of using the current table. The Delta value is negative if the current symbol is more efficiently encoded using the current table than using a new table. For example, to encode a symbol value, new table needs 3 bits while current table needs 2 bits. Current table is more efficient and the Delta value is $2 - 3 = -1$.

Once the Delta value is obtained from the Delta table, adaptive process will compute the relative advantage of the two nearest tables for encoding the next symbol value. Delta value is added to a register called Discriminant. Discriminant is used to accumulate the statistics regarding code table transition. Once the discriminant register exceeds the predefined upper bound, the VLC coding table index will increase by 1. Otherwise, if discriminant variable is below the predefined lower bound, the VLC coding table index will decrease by 1. This transits the current table to one of the nearest new tables.

After Adaptive entropy coding, the whole JPEG XR encoding is accomplished.

\(^{15}\)For instance, First Index symbol has different Delta table to Index A symbol. All Delta Tables are specified in [11].
2.8.4 Bit Stream Structure

The bit stream structure of JPEG XR is given in Figure 2.20. As we can see, the image header is put at the beginning of the stream. It contains all of the configuration meta data like image size, quantization parameters, and prediction mode which will be used in decoder. After image header, there is table index. All of the code table index information is included in this part. It describes how the VLC bits and plain bits are encoded. Finally, the following is the coded bit stream itself. The bit stream is sorted tile by tile. If the image has only one tile, macroblocks are sorted in a raster scan order.

Each macroblock has stream structure as shown in Figure 2.21. The order is like DC followed by LP and then HP.

The bit stream structures of DC, LP and HP band are given in Figure 2.22, Figure 2.23 and Figure 2.24 respectively. The DC band only needs to encode one Level value in each macroblock. The LP band encodes 15 coefficients in each macroblock. The HP band encodes $15 \times 16 = 240$ coefficients. Index Coding, Level Coding and Run Coding are all needed in the LP and HP bands.
Figure 2.21: Stream structure in each macroblock

Figure 2.22: Stream structure in the DC band

Figure 2.23: Stream structure in the LP band
Figure 2.24: Stream structure in the HP band
Chapter 3

Evaluation and Comparison

In this chapter, several steps are applied to do the evaluation. First, the requirement of video surveillance is analyzed. Based on the analysis, optimized configuration is applied to JPEG XR encoder. And then, the objective performance of optimized JPEG XR in terms of PSNR and compression ratio compared to JPEG is evaluated. Finally, the subjective user experience result is given.

3.1 Video Surveillance Requirements

The intention of this thesis is to determine if JPEG XR is a good standard that fits the requirement of IP based video surveillance camera. An in-depth requirement analysis is carried out and several important requirement options are given as follows:

- **Latency.** Low latency is essential in video surveillance applications where live monitoring takes place. Fast response to movement in the surveillance scenario should be guaranteed especially when PTZ cameras or PTZ dome cameras are used [3].

- **Bit Rate.** Bit rate is also important for video surveillance since most of the monitoring needs to be stored. Lower bit rate can save huge amount of storage. Moreover, in a network environment, low bit rate helps avoid network congestion [3].

- **Power.** Network camera now has the tendency to use Power on Ethernet [3] (PoE) so no extra power cord is needed. Since PoE only supports limited power, low power is preferable in surveillance camera.

The later optimization for JPEG XR encoder is based on these three points.
3.2 Performance Comparison

The comparison is made between JPEG XR and JPEG in both objective evaluation and subjective observation. Several raw video streams regarding surveillance scenario are used in evaluation. The details are described in the following subsections.

3.2.1 Methodology

Video surveillance emphasizes that the image quality should be evaluated in sequences. Therefore our evaluation target is based on raw video stream. As shown in Figure 3.1, each raw video stream contains hundreds of frames. These frames are separately compressed to both JPEG and JPEG XR format. JPEG files could be directly packed and played by Mplayer [8]. Since currently there is no commercial JPEG XR decoder software under Linux\(^1\), to see the JPEG XR compression result on computer screen, raw image can only be encoded to .jxr format and decoded back to .ppm. The following optimizations are applied to JPEG XR encoder to match the requirement of video surveillance mentioned in Section 3.1.

- To reduce the bit rate, down-sampling ratio of color space is set to YUV420. It has only 50% of the data compared to YUV444.
- Also, to obtain the maximum compression and reduce the bit rate, one whole image is set as a tile.
- The POT is complex in computation and will introduce latency. To reduce the latency, no overlapping filter is used.
- Also, to simplify computation, quantization parameter is the same for all three frequency bands and three color planes.

At the same time, JPEG is also given a reasonable setting. As mentioned before, quantization parameter is the main factor that affects the compression ratio\(^2\). We hereby set the quantization parameters from 30 to 80. The QP is set in this range because we can obtain good image quality with reasonable compression ratio. If the QP is set too small like from 1 to 29, the bit rate will be too high. If the QP is set too big like from 81 to 255, the image quality will be unacceptable. Frames are compressed under each parameters and average value of PSNR and compression ratio among frames is obtained. Since JPEG uses a matrix but not a certain number as QP, it has an image quality parameter range from 1 to 100\(^3\) to indicate the compression. We test JPEG use image quality parameters from 10 to 60 to make the comparison. The reason why we choose image quality parameters in this range is similar to the QP range in JPEG XR.

\(^1\)There is one named Windows Photo Gallery under windows.
\(^2\)Higher the QP, better the compression ratio, lower the image quality
\(^3\)Higher image quality means lower compression ratio.
PSNR

PSNR is commonly used for measuring the quality of reconstructed image. Normally, a color image has 3 channels YCbCr. In this evaluation we only calculate PSNR of the Luminance channel since the other two chroma channels have been downsampled in YUV420 mode. That makes the evaluation of the chroma channels meaningless (the PSNR could be very low but does not matter to human eyes). PSNR is described in Equation 3.2. The unit is db. MSE in Equation 3.1 denotes Mean Squared Error for two $m \times n$ grey-scale images $I$ and $K$ in which one of them is a reconstructed noisy image.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \| I(i,j) - K(i,j) \|^2$$ (3.1)

$$PSNR = 10 \log_{10} \left( \frac{MAX_I^2}{MSE} \right)$$ (3.2)

Compression Ratio

Compression Ratio is obtained by dividing the original file size by the compressed file size, as described in Equation 3.3.

---

4Grey scale = Luminance only
\[
\text{Compression Ratio} = \frac{\text{Original Size}}{\text{Compressed Size}}
\]  
(3.3)

3.2.2 Tools

The following tools are used in the evaluation:

- \textit{Mplayer} [8] is used to output the raw video frame and play the compressed video stream.
- JPEG XR reference C code [15] is used to encode/decode the jxr format.
- \textit{Portable Pixel Map} .ppm [7] is selected as the input raw image format.
- \textit{Netpbm} tool box [6] is employed to encode JPEG file.
- PSNR, Compression Ratio calculation and comparison is done by using \textit{Octave} [4].

3.2.3 Evaluation

Seven YUV format raw video streams of different sizes are evaluated. The same compression ratio is applied to the JPEG encoded images and JPEG XR encoded images to perform a fair subjective comparison and the result is shown as follow.

\textbf{Train Station}

Sample raw video stream “Train Station” describes a surveillance scenario in a crowded central station. Distinguishing people’s faces under this condition is critical. It contains 200 \(720 \times 576\) frames. The subjective naked eyes comparison is given in Figure 3.2 (a) \(\sim\) (c) in which the JPEG XR uses quantization parameter 71 while JPEG uses quality parameter 10. Under the same compression ratio, QP 71 in JPEG XR and quality parameter 10 in JPEG gives the most obvious subjective image quality contrast between these two standard. The same reason is applied to the evaluation of following video streams.

Figure 3.2 (d) shows objective quality (measured by PSNR) on y-axis versus compression ratio on x-axis. JPEG XR curve is always on the upper place of JPEG. Its PSNR performance is better than JPEG especially when compression ratio is high (more than 2 db).
Figure 3.2: Sample Video “Train Station”, 720 × 576, YUV420
Sample raw video stream “Intersection” describes a surveillance scenario in a parking exit. It contains 200 $720 \times 576$ frames. Subjective comparison is given in Figure 3.3 (a) $\sim$ (c) in which the JPEG XR uses quantization parameter 59 while JPEG uses quality parameters 19. Figure 3.3 (d) shows objective quality comparison.

**Figure 3.3:** Sample Video “Intersection”, $720 \times 576$, YUV420
Reception

Sample raw video stream “Reception” describes a surveillance scenario in a front-reception counter. This is to evaluate JPEG XR under an indoor situation. It contains 200 $720 \times 576$ frames. Subjective comparison is given in Figure 3.4 (a) ~ (c) in which the JPEG XR uses quantization parameter 68 while JPEG uses quality parameters 10. The light reflection from the floor in JPEG is blurred but it is better in JPEG XR. Figure 3.4 (d) shows objective quality comparison.

![Sample Video “Reception”, 720 x 576, YUV420](image)

**Figure 3.4:** Sample Video “Reception”, 720 $\times$ 576, YUV420
Lundagard

Sample raw video stream “Lundagard” describes a surveillance scenario that overlooks the entrance of a building. It contains 200 $1280 \times 720$ frames. Subjective comparison is given in Figure 3.5 in which the JPEG XR uses quantization parameter 75 while JPEG uses quality parameters 11. Note the artifacts of JPEG is quite obvious in the mid top of the image and JPEG XR is more smooth. Figure 3.6 shows objective quality comparison.

Parking

Sample raw video stream “Parking” gives a surveillance scenario in a Parking field. Different from “Intersection”, it contains larger 200 $1280 \times 720$ frames. Subjective comparison is given in Figure 3.7 in which the JPEG XR uses quantization parameter 70 while JPEG uses quality parameters 12. Note the bottom-right area of the image, the shadow in JPEG is quite rigid like turbulence but it is more natural in JPEG XR. Figure 3.8 shows objective quality comparison.

Retail

Sample raw video stream “Retail” gives a surveillance scenario in the entrance of a supermarket. It contains 200 $1280 \times 720$ frames. Subjective comparison is given in Figure 3.9 in which the JPEG XR uses quantization parameter 71 while JPEG uses quality parameters 11. In this comparison, people can see the characters on the advertise paper (in the mid-bottom of the image) clearly in JPEG XR. But the character in JPEG is obscure. Figure 3.10 shows objective quality comparison.

Mobile

Sample raw video stream “Mobile” is a classic sequence with more movement and lots of colors. Keep a sharp edge between colors after encoding is the key point under this condition. “Mobile” contains 300 $352 \times 288$ frames. Subjective comparison is given in Figure 3.11 (a) ~ (c) in which the JPEG XR uses quantization parameter 76 while JPEG uses quality parameters 17. The color testing proves JPEG XR is better than JPEG. Note the date “1”, “4”, “11”, “15” in the calender is red in the original image. JPEG compression makes the red numbers turn into black which is a huge mistake. The edge between colors in the background is also noised. JPEG XR does not have any critical mistakes in this testing. Figure 3.11 (d) shows objective quality comparison. The compression ratio becomes much smaller than the other sample videos (only 30+ times compare to 150 ~ 200 times) since this sample video stream has smaller size, and the data that could be compressed is less.
Figure 3.5: Sample Video “Lundagard”, 1280 × 720, YUV420
Figure 3.6: PSNR versus Compression Ratio
Figure 3.7: Sample Video “Parking”, 1280 × 720, YUV420
Figure 3.8: PSNR versus Compression Ratio
Figure 3.9: Sample Video “Retail”, 1280 × 720, YUV420
Figure 3.10: PSNR versus Compression Ratio
Figure 3.11: Sample Video “Mobile”, 352 × 288, YUV420
3.2.4 Subjective Observations

When doing a rather subjective evaluation of the JPEG XR standard, here are some things that were observed:

- JPEG XR uses $4 \times 4$ block as the minimum processing unit, and it is more fine-grained than the $8 \times 8$ block used in JPEG. The blocking effects found in JPEG is significantly reduced in JPEG XR. Figure 3.12 of an enhanced scene in the sample video stream “Train station” clearly shows the effect.

- Primarily we note that more details have kept in the JPEG XR images. We can more clearly distinguish details, such as color, faces and text, in the films. This would typically be a good thing in video surveillance.

- Due to JPEG XR not being intended for moving pictures but rather still images, there is an interesting effect visible, and it also happens the same way for JPEG. The compressed video stream is a little bit flickering during play. You have to pay much attention to find this detail. However, This does not effect the PSNR in a bad way and it is hard to say how this effect happens.
It might be related to the noise of the video stream itself or it could also be the problem of the refresh frequency of the display monitor.

Due to the time limitation, the subjective observation is performed among limited people. Some more people are needed to participate to give feedbacks. This could be done in future work.
Chapter 4

RTL Model Implementation

JPEG XR Software encoder is suitable for general purpose computing architecture like a desktop computer. But most of the video surveillance camera nowadays are more performance and low-power oriented. Thus, a faster and low-power hardware based implementation is needed to accelerate the encoding process. In this chapter, the implementation of hardware is discussed and a prototype JPEG XR encoder is built based on the reference C model. The solution on how the accelerator can be integrated into the Axis ARTPEC-X SoC platform is also introduced.

4.1 Hardware Issues

Although the purpose of an accelerator is to speed up the encoding process, hardware complexity should also be taken into account since it is largely related to chip area and power consumptions. In the following subsections, some hardware issues involved in software profiling, parallelization, bit operation and memory access analysis are discussed to indicate which parts of the software are needed to be implemented in hardware.

4.1.1 Software Profiling

To get information regarding which part of the software is mostly CPU consuming, the GNU Profiler [5] is used to profile the reference C code. The profiler statistics the percentage of time each function occupied in run time and counts the number of times a particular function is called. In our case, a $1280 \times 720$ image with no overlap filter and QP 64 is encoded. The detailed profiling report is given in Appendix A. The profiling report shows the number of times each function is called. From the statistic we can determine that the functions related to PCT, quantization, prediction, adaptive scan and entropy coding are executed in most of the time. The profiling result is one of the key criteria to decide which part of the encoder should be implemented in hardware.
4.1.2 Parallelization

Hardware accelerates the encoding process by not only saving a lot of CPU instructions but also parallelizing the process. Most of the reference C code is written sequentially. It is the drawback of the C language itself and a big constraint to speed up the encoding. In contrast, hardware is more parallelism friendly and has big room to improve the performance. Running several independent processes at the same time can greatly accelerate the encoding. So, the parallel analysis of the C code is performed and possible parallelism is extracted as follows:

- In PCT stage1, hadamard transform, Odd transform and Odd Odd transform can be run in parallel.
- In PCT stage2, each color plane of YUV can be processed in parallel.
- Since the HP frequency band does not need to be processed by PCT stage2, it can be parallel quantized when the DC and LP band is processed in PCT stage2. Also, the DC band and LP band can be quantized in parallel.
- The DC, LP and HP prediction can be processed in parallel.

These parallelism are implemented in hardware.

4.1.3 Bitwise Operation

The profile result in Section 4.1.1 indicates that the adaptive entropy coding consumes a lot of CPU resource. Especially for the VLC coding part, a lot of bits need to be written to the bit stream. The most common operation in adaptive entropy coding is bitwise operation. For example: shifting, AND, OR, write bit stream etc. Hardware has the advantage in manipulating bits and it is more expensive to do it in C code\(^1\). So, it is preferable to implement this part of encoder in hardware. But for the time limitation, adaptive entropy coding is not implemented in this thesis. However, it could be done in future work.

4.1.4 Memory Access Analysis

The biggest difference between hardware and software is the memory access. The reference C Code uses cache. It assumes the memory is infinite (Define arrays in any size,) and data can be fetched anywhere anytime in arbitrary size. Addressing overhead is not considered. Allowing for the big memory consumption and frequent memory access of the PCT module and prediction module, it might be reasonable to use a general purpose CPU with big cache, virtual page and page frame. But it is a nightmare for memory limited embedded system. Before hardware implementation,

\(^1\)Could be software or firmware
the memory access is analyzed. Either local scratch memory or shared system main memory can be used to buff intermediate data. The pros and cons of these two memory access solutions are discussed.

Use Local Scratch Memory and Register File

Figure 4.1 gives a very simplified architecture about memory access. The less important intermediate components are not shown. The image frames are stored in the system main memory (DDR) with specified starting address and ending address. The Simple Stream Interface (SSIF) is an on-chip communication protocol which is developed by Axis Communications. It is used to stream the image data. A DDR controller is used to communicate with the DDR memory. The image frames in the DDR are read into the JPEG XR module linearly from starting address to ending address by the DMA. Since scratch memory and register file are applied inside JPEG XR module, there is no main memory access during the data process. After the process is completed, the processed data is linearly written to the system main memory with specified starting address by DMA. Ending address depends on the JPEG XR configuration. So, the memory access for the proposed hardware is straightforward. It is a linear function of the image size.

Use Shared System Main Memory

The alternative way is using shared system main memory. As shown in Figure 4.2, the intermediate data is buffered in system main memory since no local memory
is built in JPEG XR module. To communicate with the system main memory, a memory interface has to be built. Compared to the first solution in Figure 4.1, the biggest advantage of this solution is that without build-in memory, the silicon area is significantly reduced. However, the drawbacks are also obvious. The memory access to the system main memory is increased in this solution. Since there is no iterative loop in JPEG XR module, the memory access is still a linear function of the image size. Besides the increasing of main memory access, more latency is introduced in this solution because of the extra access time for the external memory.2

Comparing these two solutions, some conclusions are made:

- Using local memory or register file can avoid excessive access to main memory and lower the load of system bus.
- Low latency can be achieved by using local memory. No addressing overhead, fetch data immediately. Using shared system main memory will increase the system overhead since memory addressing time is usually longer than the data processing time.
- Local memory and register file is easier to implement. It does not need to access the system bus and schedule the bus access time compared to using shared system main memory.

2Typically more longer than the data processing time
• Local memory and register file occupy more silicon area than using system main memory.

According to the information above, local scratch memory and register file is used in the prototype encoder since it is better fitted to the requirement of video surveillance.

4.2 Software/Hardware Partitioning

The discussions from the subsections above inform us which parts of the encoder should be implemented in hardware. The reasons why software should be implemented in hardware are as follows:

• Application specified hardware saves CPU instructions (or resource).
• Hardware is easier to program in parallel.
• Bitwise operation is more efficient in hardware.
• Hardware can be designed to avoid excessive main memory access.

For these reasons, in Figure 4.3, the whole system has been divided into two parts. The main image data stream will go through hardware while image information like header and meta data will go through software. At last, these two data streams will be packed and stored in the main memory. For the sake of time limitation, adaptive scan and adaptive entropy coding function blocks are not implemented in the prototype.

Finally, the file I/O operations, RGB to YUV420 conversion, adaptive scan, adaptive entropy coding as well as packaging are kept as software. PCT, quantization and adaptive prediction are implemented in the prototype.

4.3 Hardware Implementation

To meet the requirement of IP based Video Surveillance application mentioned in Section 3.1, the following options are chosen for optimization purpose:

• To reduce the bit rate, YUV420 is chosen since it halves the bit rate of YUV444.
• To reduce the latency, no over-lap filter is used. The computation of POT takes too much time.
• A fixed quantization parameter is chosen for quick prototyping.
Figure 4.3: Software/hardware partitioning
• The memory size is designed to process image with a width less than or equal to 1280 pixels\(^3\). It can be extended to process wider image but this would require more silicon area.

• The image width and height should be integer times of 16 pixels.

As in Figure 4.4, the general hardware is implemented by using the “on-the-fly” fashion. Macroblock is the basic data unit needed to be processed in each function module. The sub-modules are pipe-lined to achieve a better performance as shown in Figure 4.5.

\(^3\)Typical 720p HD-ready picture
Figure 4.4: General hardware architecture
4.3.1 Picture Core Transform

First Stage PCT

Figure 4.6 shows the architecture of first stage PCT. It is a $4 \times 4$ PCT operation which contains two phases:

- Phase one consists of one 256 bits buffer, one multiplexer and one $2 \times 2$ hadamard module.

- Phase two consists of one 256 bits buffer, one $2 \times 2$ hadamard module, two $2 \times 2$ odd modules and one $2 \times 2$ oddodd module.

The implementation of the $4 \times 4$ PCT is quite straight forward. The hadamard, odd, oddodd transform modules are all consists of arithmetic operations. The detailed algorithms are specified in Chapter 2. The multiplexer and two buffers are applied to reuse the $2 \times 2$ hadamard module as well as meet the timing constraint.

Since the bit-depth in this paper is 16 bits/pixel, for a throughput of 2 pixels/cycle, the first 256 bits buffer can be fulfilled in $256 \div 32 = 8$ cycles. The $2 \times 2$ hadamard, $2 \times 2$ odd and $2 \times 2$ oddodd operations are designed to finish in one clock cycle, so the first phase takes 4 cycles and the second phase takes one cycle. The first stage PCT can be finished in 5 cycles.

Second Stage PCT

The second stage PCT is shown in Figure 4.4. Only DC and Low Pass bands data are processed in the second stage. High Pass band data are directly bypassed to quantization module. One $4 \times 4$ PCT module and two $2 \times 2$ PCT modules are
applied to Y,U,V color planes respectively. The 4 × 4 PCT module is the same as in the first stage while the 2 × 2 PCT architecture is shown in Figure 4.7. We can see the 2 × 2 PCT module consists of 4 right-shifting operations and one 2 × 2 hadamard module. The process is finished in one clock cycle.

4.3.2 Quantization

As specified in Chapter 2, the cost of division is very expensive in hardware. Therefore, division will not be performed directly but replaced by multiplication and shifting. The idea is saving the reciprocal of the divisors in a look-up table [13]. And this look-up table is indexed (or addressed) by the integer divisors themselves. The look-up table is implemented as a register file in hardware, and all of the reciprocals stored in the register file are pre-calculated. When needed, the reciprocal
if (iQP < 16) {
    iMan = iQP;
    iExp = iScaledShift;
} else {
    iMan = 16 + (iQP % 16);
    iExp = ((iQP >> 4) - 1) + iScaledShift;
}
quant = iMan << iExp;

Listing 4.1: Pseudo-code for QuantMap

can be read out in the same clock cycle. Since in JPEG XR, all of the computation is based on integer, and the reciprocals of the divisor are in most case decimal numbers, up-scaling is needed to transform these decimal numbers to integer. After multiplying with the coefficients, right shifting (or down-scaling) is applied to the result to compensate the up-scaling. The details are given as follows.

According to Listing 4.1 [11], a quantization parameter ‘‘quant’’ can be represented by a mantissa ‘‘iMan’’ and an exponent ‘‘iExp’’. The ‘‘quant’’ is obtained by left-shifting ‘‘iMan’’ for ‘‘iExp’’ bits. From Chapter 2, we know the quantization parameter is ranged from 1 to 255. After remapping, the ‘‘iMan’’ has value ranged from 1 to 31. That means that 31 reciprocals are needed to be stored in the register file. In our case, each reciprocal has a bit-width of 16 bits. As specified in [14], to transform decimal to integer, each reciprocal needs to be up-scaled by $2^{15}$ (or left shifted by 15 bits). ‘‘iScaledShift’’ is a variable ranged from 0 to 1. Its value is variant across frequency bands and color planes, Table 4.1 gives the ‘‘iScaledShift’’ values in all the cases.

<table>
<thead>
<tr>
<th></th>
<th>DC Band</th>
<th>LP Band</th>
<th>HP Band</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y plane</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>U plane</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>V plane</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.1: iScaleShift value

Equations 4.1 to 4.3 show how the quantization works. In Equation 4.2, the fraction $\frac{1}{iMan} \times 2^{15}$ is the reciprocal we want to save in the look-up table. After multiplication, we have to compensate the result by right shift $iExp + 15$ bits as shown in Equation 4.3.

\[
\frac{coeff}{quant} = \frac{coeff}{iMan \times 2^{iExp}} 
\]  

(4.1)

\[
\frac{coeff}{iMan \times 2^{iExp}} = coeff \times \frac{1}{iMan} \times 2^{15} 
\]  

(4.2)
\[
coeff \times \frac{1}{2^{i\text{Exp}}} \times 2^{15} = \left(\coeff \times \frac{1}{i\text{Man}} \times 2^{15}\right) \gg (i\text{Exp} + 15) \tag{4.3}
\]

An example is shown in Equations 4.4 to 4.5. Assume the mantissa “iMan” is 17, 1928 is the integer number needed to be stored in the look-up table. Some rounding error is introduced in this step.

\[
\frac{1}{17} = 0.058823529 \tag{4.4}
\]

\[
0.058823529 \times 2^{15} = 1927.529358272 \approx 1928 \tag{4.5}
\]

The hardware architecture is shown in Figure 4.8. A 31 x 16 bits register is used to store the up-scaled reciprocal.

(Note: For simplicity and quick prototyping reason, a fixed quantization parameter 64 is set to the quantization module.)

### 4.3.3 Prediction Unit

The basic idea of prediction is to save the difference between neighbor blocks and macroblocks once their difference is small.

**High Pass Prediction**

High pass prediction happens within a macroblock so it is quite independent. There are three predict modes: from top, left and no prediction. Since low pass coefficients in current macroblock decides the predict mode and they arrive later than the raster-scanned high pass coefficients, we need to store all of the high pass coefficients in a register file. The actual coefficients involved in the prediction are \((16+4+4) \times 6 \times 16\) bits \(^4\) as shown in Figure 4.9 which are 1, 2, 3, 4, 8, 12.

Although, we cannot bypass the other 9 coefficients since they will be rewritten by next incoming data, we have to save all 15 coefficients. As shown in Figure 4.10, a \(24 \times 15 \times 16 = 5760\) bits register file is employed and once the LP prediction data arrives, HP prediction can be performed. Since all of the necessary data is stored in the register file, the HP prediction can be completed in one clock cycle.

**DC and Low Pass Prediction**

DC and LP prediction is in the same module since they share the same predict mode in this design case. Different from high pass prediction, the DCLP bands

\(^4\)16 blocks for Y channel, 4 blocks for each chroma channel
Figure 4.8: Quantization module hardware architecture

Figure 4.9: High pass coefficients needed in the prediction
predict from neighbor macroblocks which means buffers should be employed to store previous data. These buffers include:

- Since the DCLP coefficients arrive in raster scan order, to predict from top, at least one row of macroblocks has to be saved. The size of scratch memory depends on the width of image. It is a linear monotonous increasing function of the image width in macroblock. According to Section 2.6.2, in each macroblock, only 8 coefficients\(^5\) are related to actual prediction. So the memory size for a 1280 × 720 image should be 80 × 8 × 16 = 10240 bits\(^6\). Equation 4.6 shows the relation between the image width and memory size. A local scratch memory is employed as a FIFO buffer to store one row. In that way, the later row coefficients could predict from top.

\[
\text{memory size} = 8 \times 16 \times \frac{\text{imagewidth}}{16} = 8 \times \text{imagewidth} \quad (4.6)
\]

- To be able to predict from left, according to Section 2.6.2, a 13 × 16 = 128 bits register is used. The data is also used to decide the predict mode.

- To predict from top left\(^7\), a 8 × 16 = 128 bits register is used to store the data from scratch memory.

\(^5\)1 DC coefficient and 3 LP coefficients for Y channel, 1 DC and 1 LP for both chroma channels

\(^6\)1 row has 80 macroblocks; 16 is the bit depth.

\(^7\)Only available for DC
In Figure 4.11, red blocks indicates the macroblocks need to be saved in local scratch memory. New incoming coefficients will always rewrite two buffers:

- Scratch memory, indicated by "1" in Figure 4.11.

- Incoming data buffer, indicated by "2" in Figure 4.11.

Before incoming data rewrites scratch memory, the old coefficients have to be saved in scratch memory buffer indicated by "0" in Figure 4.11. Then, once the predict mode is decided, prediction can be done in one of these three directions.

The actual hardware structure is shown in Figure 4.12.
4.4 Simulation and Verification

4.4.1 Test Bench

The hardware is described in System Verilog code. A testbench is designed to read in the image data and send it to the hardware. The testbench architecture is given in Figure 4.13.

In this testbench, a Bus Functional Model (BFM) is designed to model the Simple Streaming Interface (SSIF) and send the image data to RTL module. The BFM interacts with the Design Under Testing (DUT) by both driving and sampling the DUT signal. It has the ability to connect and interface to higher layers of testbench supporting multiple languages as well as supporting different levels of RTL abstraction. In that way, the C code can easily mix with the System Verilog code for simulation and verification. The BFM is designed as a class like in C++, its member functions describe the SSIF protocol. Synopsys VCS is used to simulate the whole system.
4.4.2 Verification

Verification is done by comparing the pure software C model output data and the hardware output data. Figure 4.14 shows how verification works to make sure the hardware has correct functionality. Several 1280 × 720 raw image frames are used as the input pattern. The output data of each function block from the pure reference C code are stored in hex files. Comparison with the hardware output is made and all of the data are identical.
4.5 Synthesize Result

In this phase, Synopsys ASIC tool chain and the TSMC 40nm low power process library is used to synthesize the hardware. The system is synthesized for running at 100 MHz.

4.5.1 Throughput

The throughput of the input SSIF bus is 2 pixels/cycle running at 100 MHz, i.e. 200 Mpixels/s. A typical 1920 \times 1080 full HD stream with YUV420 down-sampling and 25 frames/second ratio need a throughput of 1920 \times 1080 \times 1.5 \times 25 = 77760000 = 77.76 Mpixels/s. Thus the prototype encoder is fast enough to process full HD stream.

4.5.2 Area

The silicon area of each hardware module is shown in Table 4.2. It is a summary of the area report generated by Synopsys ASIC synthesize tool. Please read the area report in Appendix B for more details.

<table>
<thead>
<tr>
<th>Module</th>
<th>Area((\mu m^2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st pct</td>
<td>13731</td>
</tr>
<tr>
<td>2st pct y</td>
<td>11840</td>
</tr>
<tr>
<td>2st pct u</td>
<td>1251</td>
</tr>
<tr>
<td>2st pct v</td>
<td>1253</td>
</tr>
<tr>
<td>quantization</td>
<td>8743</td>
</tr>
<tr>
<td>hp prediction</td>
<td>45122</td>
</tr>
<tr>
<td>dclp prediction</td>
<td>101148</td>
</tr>
<tr>
<td>jpegxr top</td>
<td>193929</td>
</tr>
</tbody>
</table>

Table 4.2: Silicon area of each hardware module

The area unit in this table is \(\mu m^2\). It shows the total area of the design is 193929 \(\mu m^2\) and the prediction modules occupied the largest area. It is because scratch memory and register file are included in prediction modules. As mentioned in Section 4.3.3, the size of scratch memory in dclp prediction module is 10240 bits and the size of register file in hp prediction module is 5760 bits. The synchronized RAM mem (jpegxr_sync_ram_1rw_h_426_242_793_0) (see Appendix B) is the 10240 bits scratch memory which in this phase implemented in flip-flops\(^8\). Its area is a

---

\(^8\)With wrapper, flip-flops behave like synchronized RAM, they could be replaced by full-customized memory IP.
linear monotonous increasing function of the image width given in Equation 4.6. The 5760 bits register file in hp prediction module is also implemented in flip-flops.

The PCT modules contain a lot of arithmetic units that occupied the second largest area 28075 $\mu m^2$ in the design.

Quantization module occupied 8743 $\mu m^2$.

### 4.5.3 Power

The power consumption of each hardware module is shown in Table 4.3. It is a summary of the power report generated by Synopsys ASIC synthesize tool. Please read the power report in Appendix C for more details.

<table>
<thead>
<tr>
<th>Module</th>
<th>Power(mW)</th>
<th>Percentage(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st pct</td>
<td>0.482</td>
<td>6.3</td>
</tr>
<tr>
<td>2st pct y</td>
<td>0.316</td>
<td>4.1</td>
</tr>
<tr>
<td>2st pct U</td>
<td>0.0341</td>
<td>0.4</td>
</tr>
<tr>
<td>2st pct V</td>
<td>0.0340</td>
<td>0.4</td>
</tr>
<tr>
<td>quantization</td>
<td>0.2175</td>
<td>2.9</td>
</tr>
<tr>
<td>hp prediction</td>
<td>2.189</td>
<td>28.5</td>
</tr>
<tr>
<td>dclp prediction</td>
<td>3.930</td>
<td>51.2</td>
</tr>
<tr>
<td>jpegxr top</td>
<td>7.669</td>
<td>100</td>
</tr>
</tbody>
</table>

**Table 4.3: Power consumption of each hardware module**

The power report is generated under a switching activity that the static switching probability is set to 50% and the toggle rate is set to 50%. The switching activity is highly depends on the application and environment like how large is the size of the image in processing and how long time the camera is in operation. According to [9], usually, the data path logic like multiplexers, adders, AND gates, OR gates can have toggle rates ranged from 6% to 12%. Random logic, such as decoders or encoders can have toggle rates around 50%, while control logic can toggle at high level, close to 100% toggle rate. As an encoder, the toggle rate is set to 50%. The more active the switching is, the larger power consumption will be. Under this switching activity, the predicted power consumption of the design is 7.669 mW. The prediction modules turn out to be the biggest power consumer which consume 79.7% of the power. Like the situation in area, the synchronized RAM mem (jpegxr sync ram lrw h 426 242 793 0) (see Appendix C) in dclp prediction module and register file in hp prediction module play an important role in power consumption. In addition, the PCT modules consume 11.2% of the power. Quantization modules have 2.9% of the power consumption.
4.6 System Integration

In this section, we discussed how the proposed hardware module could be integrated with the Axis ARTPEC-X SoC platform. Two different system integration solutions are analyzed.

4.6.1 Integrate Proposed Module Inside SoC

The ARTPEC-X SoC platform is developed by Axis Communications AB Sweden. An embedded Linux operation system is running on this platform. An image sensor is attached to capture image. Figure 4.15 is a simplified architecture of ARTPEC-X. Only relevant components are shown. The JPEG XR module is directly connected to the system bus and its address is mapped to the main memory table. Then CPU can control and configure the JPEG XR module by setting the register interface. For example, set different quantization parameters. The Image Processing Pipeline (IPP) transforms and down-samples the raw image data into YUV420 format and then sends to the JPEG XR module. The output from JPEG XR module is sent to a DMA in order to reduce CPU load. The DMA manages to store all of the data in main memory, then the software part of encoder running in the embedded Linux environment will finish the rest of compression (adaptive scan, entropy coding, packagizing).
4.6.2 Integrate Proposed Module as Co-processor in a FPGA

In this solution, as shown in Figure 4.16, the prototype encoder is implemented in a FPGA as co-processor outside the ARTPEC-X SoC. The IPP sends image data to JPEG XR module via a simple FPGA link. The simple FPGA link, developed by Axis, is an application specific protocol used to communicate between the ARTPEC-X SoC and FPGA. As a chip-to-chip communication protocol, the simple FPGA link is slower than SSIF which is used for on-chip communications. Since they do not have the same bandwidth, a FIFO buffer is employed to balance the speed. Compared to integrating JPEG XR module inside the ARTPEC-X, implementing JPEG XR prototype into the FPGA is a quick approach to the actual hardware. There is no need of layout, masks or other manufacturing steps which is required for ASIC design flow. But the drawback is also obvious: The communication between the SoC and the FPGA add more latency to the system. Moreover, since FPGA realizes functions by using look-up table, it consumes more power than an ASIC. In addition, the unit cost of FPGA is higher than ASIC for high volume designs.

Generally, ASIC is more performance oriented and good for long term, large volume production. FPGA is better for quick marketing, small volume production. Whether to put JPEG XR module in the SoC or FPGA should depend on marketing, economics etc.
Figure 4.16: JPEG XR module in a FPGA as co-processor with ARTPEC-X
Chapter 5

Conclusion

5.1 Achievement

In this thesis, a deep insightful study of JPEG XR standard is carried out. In the evaluation phase, optimized configuration is applied to JPEG XR encoder to meet the latency, bit-rate and power requirement of video surveillance. According to the data collected in Chapter 3, with optimized configuration, the PSNR performance of JPEG XR is on average 2 to 4 db better than JPEG. The advantage is more obvious especially when the compression ratio is increased. That means JPEG XR keeps better image quality when bit-rate is low. Moreover, its actually subjective user experience is also better than JPEG. More image details (like color, object edge) are kept and the block effect is more fine-grained compared to JPEG.

As the standard is implemented in hardware, we found the architecture of JPEG XR is more complex than JPEG because some dynamic features are introduced in JPEG XR. For example: In adaptive prediction, the predict mode is dynamically changed to find the predict direction with the largest similarity to current block/-macroblock. In adaptive scan, the scan order is kept on changing according to the non-zero and zero coefficients distribution. And in adaptive entropy coding, the VLC coding table is changing to perform the most efficient coding. These dynamic features change encoding strategy to adapt different bit stream and always keep the highest encoding efficiency. Compared to JPEG XR, JPEG is “static”. Its setting does not change during the encoding. These dynamic features are the biggest difference between JPEG and JPEG XR.

Since the original JPEG XR encoder is described in C code, some simplifications and optimizations are made during the prototype hardware encoder implementation.

- Since the size of macroblock is 16 × 16, the encoder can only process image with width and height that are both integer times of 16.

- Image information, meta data, file I/O operations are cut down. They can be done by software.
• To reduce the bit-rate, YUV420 down-sampling rate is chosen because it has only half data of the YUV444.

• Color conversion is not implemented since the ARTPEC-X SoC already provides YUV420 data.

• One whole image is set as one tile to achieve the highest compression.

• The POT is not implemented since it introduces too much latency on the encoding.

• To reduce computation complexity, the quantization parameter is the same for all three frequency bands and three color planes.

• For quick prototyping, a fixed quantization parameter is used in quantization.

• The memory size in prediction module is set to process frames equal to or less than $1280 \times 720$. It can be extended to process arbitrary large image. The memory size is a linear monotonous increasing function of image width.

• The memory and register file in prediction module is implemented in flip-flops (with wrapper, they behave like synchronized RAM). They can be replaced by full customized memory IP.

• Because of time limitation, adaptive scan and adaptive entropy coding are not implemented.

The final prototype encoder has a throughput of 200 Mpixels/s. It is fast enough to process Full-HD stream and larger video streams. In general, from both software and hardware, JPEG XR has satisfied the low bit-rate, low latency and high image quality requirements for IP based video surveillance. It turns out to be a good replacement of old JPEG standard.

5.2 Problems

In evaluation phase, due to lack of experience, sometimes it is difficult to judge the quality of an image subjectively. The conclusion is made from a personal angle. A survey regarding the image quality might be carried out among certain amount of people in the future to enhance the subjective evaluation.

The implementation of hardware is all based on the standard document and the C reference code. Unfortunately, they are both difficult to read and lack of comments which takes me a huge amount of time to investigate. Since the standard is originally written in C, it does not take too much thinking about hardware implementation. The differences between C code and hardware are given as follows:
Most of the C code processes data in a sequential way while in hardware, performance is achieved by utilizing parallel processing.

C code uses cache, it fetches data anywhere anytime in arbitrary size and does not consider the addressing overhead. Embedded hardware has to predefine the memory size and schedule the addressing timing.

C code directly uses division in quantization. In hardware, it is replaced by multiplication and shifting with look-up table filled with pre-calculated parameters.

Moreover, some parts of the C code is not hardware friendly. The POT and adaptive entropy coding are two examples. The POT pre-filter largely reduces the block effect and ringing artifact, but it comes with an expensive price: the neighbor macroblocks are needed to be stored in big memory and extra big latency is introduced. Regarding to adaptive entropy coding, the C code describes it in a very trivial way: Too many control signals are needed which makes people easily confused. Thus, these two parts are not implemented as hardware in this thesis.

5.3 Future Work

As the subjective evaluation is not completed and not all of the JPEG XR encoder is implemented in hardware, to be a really efficient accelerator running with ARTPEC-X SoC, more work needs to be done.

A survey regarding the motion-JPEG XR video stream quality among certain number of people should be carried out to enhance the subjective evaluation.

Adaptive Scan and adaptive entropy coding should be also implemented as hardware. This will significantly reduce the bit-stream inside the SoC.

More configure options need to be given in JPEG XR encoder. For example: the bit-depth, tiles number can be configured. Quantization parameters can be set differently across frequency bands and color places. It gives flexibility to the encoder to satisfy different requirement.

When memory size is large enough, the encoder should process image in arbitrary size.
Bibliography


# Appendix A

## Software Profiling Report

This report is generated by GNU Profiler, a 1280 × 720 raw image is processed.

---

**Flat profile:**

Each sample counts as 0.01 seconds.

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>self</th>
<th>calls</th>
<th>ms/call</th>
<th>total</th>
<th>ms/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>seconds</td>
<td>seconds</td>
<td>ms/call</td>
<td>ms/call</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.22</td>
<td>0.06</td>
<td>0.06</td>
<td>1371600</td>
<td>0.00</td>
<td>0.00</td>
<td>quantize_lphp</td>
</tr>
<tr>
<td>10.81</td>
<td>0.10</td>
<td>0.04</td>
<td>90000</td>
<td>0.00</td>
<td>0.00</td>
<td>_FwdPermute</td>
</tr>
<tr>
<td>10.81</td>
<td>0.14</td>
<td>0.04</td>
<td>45</td>
<td>0.89</td>
<td>2.33</td>
<td>collect_and_scale_up4</td>
</tr>
<tr>
<td>10.81</td>
<td>0.18</td>
<td>0.04</td>
<td>45</td>
<td>0.89</td>
<td>0.89</td>
<td>yuv444_to_yuv422_up4</td>
</tr>
<tr>
<td>8.11</td>
<td>0.21</td>
<td>0.03</td>
<td>86400</td>
<td>0.00</td>
<td>0.00</td>
<td>w_DECODE_BLOCK_ADAPTIVE</td>
</tr>
<tr>
<td>5.41</td>
<td>0.23</td>
<td>0.02</td>
<td>457200</td>
<td>0.00</td>
<td>0.00</td>
<td>_2x2T_h</td>
</tr>
<tr>
<td>5.41</td>
<td>0.25</td>
<td>0.02</td>
<td>3600</td>
<td>0.01</td>
<td>0.01</td>
<td>calculate_hpcbp_up1</td>
</tr>
<tr>
<td>5.41</td>
<td>0.27</td>
<td>0.02</td>
<td>49</td>
<td>0.41</td>
<td>0.41</td>
<td>_jxr_clear_strip_cur</td>
</tr>
<tr>
<td>4.05</td>
<td>0.28</td>
<td>0.01</td>
<td>921600</td>
<td>0.00</td>
<td>0.00</td>
<td>_jxr_cell_div2</td>
</tr>
<tr>
<td>4.05</td>
<td>0.30</td>
<td>0.01</td>
<td>180000</td>
<td>0.00</td>
<td>0.00</td>
<td>_T_odd</td>
</tr>
<tr>
<td>2.70</td>
<td>0.31</td>
<td>0.01</td>
<td>26153</td>
<td>0.00</td>
<td>0.00</td>
<td>w_DECODE_ABS_LEVEL</td>
</tr>
<tr>
<td>2.70</td>
<td>0.32</td>
<td>0.01</td>
<td>10800</td>
<td>0.00</td>
<td>0.00</td>
<td>dclphp_unshuffle</td>
</tr>
<tr>
<td>2.70</td>
<td>0.33</td>
<td>0.01</td>
<td>7200</td>
<td>0.00</td>
<td>0.00</td>
<td>block_shuffle420</td>
</tr>
<tr>
<td>2.70</td>
<td>0.34</td>
<td>0.01</td>
<td>3600</td>
<td>0.00</td>
<td>0.00</td>
<td>block_shuffle444</td>
</tr>
<tr>
<td>2.70</td>
<td>0.35</td>
<td>0.01</td>
<td>45</td>
<td>0.22</td>
<td>0.56</td>
<td>rgb_to_yuv444_up4</td>
</tr>
<tr>
<td>2.70</td>
<td>0.36</td>
<td>0.01</td>
<td>45</td>
<td>0.22</td>
<td>0.22</td>
<td>yuv422_to_yuv420_up3</td>
</tr>
<tr>
<td>1.35</td>
<td>0.36</td>
<td>0.01</td>
<td>21735</td>
<td>0.00</td>
<td>0.00</td>
<td>_jxr_quant_map</td>
</tr>
<tr>
<td>1.35</td>
<td>0.37</td>
<td>0.01</td>
<td>_InvT_odd_odd_POST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>950400</td>
<td>0.00</td>
<td>0.00</td>
<td>_jxr_floor_div2</td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>483399</td>
<td>0.00</td>
<td>0.00</td>
<td>_jxr_wbitstream_uint1</td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>90000</td>
<td>0.00</td>
<td>0.00</td>
<td>_T_odd_odd</td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>90000</td>
<td>0.00</td>
<td>0.00</td>
<td>_jxr_4x4PCT</td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>86400</td>
<td>0.00</td>
<td>0.00</td>
<td>w_BLOCK_FLEXBITS</td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>60446</td>
<td>0.00</td>
<td>0.00</td>
<td>put_byte</td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>45068</td>
<td>0.00</td>
<td>0.00</td>
<td>w_DECODE_INDEX</td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>41894</td>
<td>0.00</td>
<td>0.00</td>
<td>_jxr_wbitstream_uint2</td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>27566</td>
<td>0.00</td>
<td>0.00</td>
<td>RunLengthEncode</td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>27566</td>
<td>0.00</td>
<td>0.00</td>
<td>w_DECODE_BLOCK</td>
</tr>
</tbody>
</table>

...
0.00 0.37 0.00 225 0.00 0.00 _jxr_ResetTotalsAdaptiveScanLP
0.00 0.37 0.00 135 0.00 1.04 PCT_stage1_up2
0.00 0.37 0.00 135 0.00 0.07 PCT_stage2_up1
0.00 0.37 0.00 135 0.00 0.00 w_guess_dc_quant
0.00 0.37 0.00 49 0.00 0.00 dump_all_strips
0.00 0.37 0.00 49 0.00 0.41 _jr_rotate_mb_strip
0.00 0.37 0.00 49 0.00 6.63 wflush_process_strip
0.00 0.37 0.00 45 0.00 7.22 _ jr_wflush_mb_strip
0.00 0.37 0.00 45 0.00 0.67 scale_and_shuffle_up3
0.00 0.37 0.00 23 0.00 0.00 _jxr_wbitstream_uint8
0.00 0.37 0.00 17 0.00 0.00 get_uint8
0.00 0.37 0.00 12 0.00 0.00 InitVLCTable2
0.00 0.37 0.00 8 0.00 0.00 _jxr_InitVLCTable
0.00 0.37 0.00 4 0.00 0.00 _jxr_wbitstream_syncbyte
0.00 0.37 0.00 3 0.00 0.00 _jxr_InitializeModelMB
0.00 0.37 0.00 3 0.00 0.00 put_ch_mode
0.00 0.37 0.00 3 0.00 0.00 read_pnm_number
0.00 0.37 0.00 2 0.00 0.00 _jxr_wbitstream_flush
0.00 0.37 0.00 2 0.00 0.00 _jxr_wbitstream_uint16
0.00 0.37 0.00 2 0.00 0.00 jxr_get_TILING_FLAG
0.00 0.37 0.00 1 0.00 0.00 __make_jxr
0.00 0.37 0.00 1 0.00 0.00 _jxr_InitCBPVLCL
0.00 0.37 0.00 1 0.00 0.00 _jxr_InitHPVLC
0.00 0.37 0.00 1 0.00 0.00 _jxr_InitLPVLC
0.00 0.37 0.00 1 0.00 0.00 _jxr_InitializeAdaptiveScanHP
0.00 0.37 0.00 1 0.00 0.00 _jxr_InitializeAdaptiveScanLP
0.00 0.37 0.00 1 0.00 0.00 _jxr_InitializeCountCBPLP
0.00 0.37 0.00 1 0.00 0.00 _jxr_make_mbstore
0.00 0.37 0.00 1 0.00 0.00 _jr_w_DC_QP
0.00 0.37 0.00 1 0.00 0.00 _jr_w_HP_QP
0.00 0.37 0.00 1 0.00 0.00 _jr_w_LP_QP
0.00 0.37 0.00 1 0.00 365.00 _jxr_w_TILE_SPATIAL
0.00 0.37 0.00 1 0.00 0.00 _jxr_wbitstream_intVLW
0.00 0.37 0.00 1 0.00 0.00 _jxr_wbitstream_mark
0.00 0.37 0.00 1 0.00 0.00 _jxr_wbitstream_uint3
0.00 0.37 0.00 1 0.00 0.00 clear_vlc_tables
0.00 0.37 0.00 1 0.00 0.00 close_file
0.00 0.37 0.00 1 0.00 0.00 emit_ifd
0.00 0.37 0.00 1 0.00 0.00 fill_in_image_defaults
0.00 0.37 0.00 1 0.00 0.00 get_file_parameters
0.00 0.37 0.00 1 0.00 0.00 jxr_create_container
0.00 0.37 0.00 1 0.00 0.00 jxr_create_image
0.00 0.37 0.00 1 0.00 0.00 jxr_destroy
0.00 0.37 0.00 1 0.00 0.00 jxr_destroy_container
0.00 0.37 0.00 1 0.00 0.00 jxr_set_BANDS_PRESENT
0.00 0.37 0.00 1 0.00 0.00 jxr_set_DISABLE_TILE_OVERLAP
0.00 0.37 0.00 1 0.00 0.00 jxr_set_INTERNAL_CLR_FMT
0.00 0.37 0.00 1 0.00 0.00 jxr_set_OUTPUT_BITDEPTH
0.00 0.37 0.00 1 0.00 0.00 jxr_set_OVERLAP_FILTER
0.00 0.37 0.00 1 0.00 0.00 jxr_set_QP_UNIFORM
0.00 0.37 0.00 1 0.00 0.00 jxr_set_TRIM_FLEXBITS
<table>
<thead>
<tr>
<th>Time</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
<th>Value 6</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>jxr_set_block_input</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>jxr_set_user_data</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>365.00</td>
<td>jxr_write_image_bitstream</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>jxrc_begin_ifd_entry</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>jxrc_begin_image_data</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>jxrc_close_container</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>jxrc_set_image_shape</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>jxrc_set_pixel_format</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>jxrc_start_file</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>365.00</td>
<td>main_compress</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>need_trim_flexbits_flag</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>need_windowing_flag</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>open_input_file</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>open_pnm_input_file</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>short_header_ok</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>w_INDEX_TABLE</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>365.00</td>
<td>w_TILE</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>w_TILE_HEADER_DC</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>w_TILE_HEADER_HIGHPASS</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>w_TILE_HEADER_LOWPASS</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>w_image_header</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.37</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>w_image_plane_header</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B

Area Report

The area report is generated by Synopsys ASIC synthesize tool. The unit is $\mu m^2$. Divide ”Total cell area” with 5.09 to get ”gate-count”.

| Number of ports: | 12586 |
| Number of nets: | 15302 |
| Number of cells: | 6640 |
| Number of references: | 74 |
| Combinational area: | 52469.473962 |
| Noncombinational area: | 84367.007589 |
| Net Interconnect area: | undefined (Wire load has zero net area) |
| Total cell area: | 136836.481551 |

<table>
<thead>
<tr>
<th>Cell name</th>
<th>total area</th>
<th>gate area</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1 (jpegxr_hadamard_0)</td>
<td>1244</td>
<td>1244</td>
</tr>
<tr>
<td>h2 (jpegxr_hadamard_4)</td>
<td>1267</td>
<td>1257</td>
</tr>
<tr>
<td>odd1 (jpegxr_odd_transform_0)</td>
<td>2430</td>
<td>2430</td>
</tr>
<tr>
<td>odd2 (jpegxr_odd_transform_3)</td>
<td>2437</td>
<td>2437</td>
</tr>
<tr>
<td>odd_odd1 (jpegxr_odd_odd_transform_0)</td>
<td>2174</td>
<td>2174</td>
</tr>
<tr>
<td>pct_1 (jpegxr_fwd_4x4pct_stage1_0)</td>
<td>13731</td>
<td>13731</td>
</tr>
<tr>
<td>h1 (jpegxr_hadamard_3)</td>
<td>1244</td>
<td>1244</td>
</tr>
<tr>
<td>h2 (jpegxr_hadamard_2)</td>
<td>1245</td>
<td>1245</td>
</tr>
<tr>
<td>odd1 (jpegxr_odd_transform_2)</td>
<td>2439</td>
<td>2439</td>
</tr>
<tr>
<td>odd2 (jpegxr_odd_transform_1)</td>
<td>2430</td>
<td>2430</td>
</tr>
<tr>
<td>odd_odd1 (jpegxr_odd_odd_transform_1)</td>
<td>2189</td>
<td>2189</td>
</tr>
<tr>
<td>pct_2_y (jpegxr_fwd_4x4pct_stage2_0)</td>
<td>11840</td>
<td>11840</td>
</tr>
<tr>
<td>h0 (jpegxr_hadamard_1)</td>
<td>1245</td>
<td>1245</td>
</tr>
<tr>
<td>pct_2_u (jpegxr_fwd_2x2pct_stage2_0)</td>
<td>1251</td>
<td>1251</td>
</tr>
<tr>
<td>h0 (jpegxr_hadamard_5)</td>
<td>1245</td>
<td>1245</td>
</tr>
<tr>
<td>pct_2_v (jpegxr_fwd_2x2pct_stage2_1)</td>
<td>1253</td>
<td>1253</td>
</tr>
<tr>
<td>q_hp (jpegxr_fwd_quant_hp_0)</td>
<td>3066</td>
<td>3066</td>
</tr>
<tr>
<td>q_dc_y (jpegxr_fwd_quant_dclp_N1_0)</td>
<td>201</td>
<td>201</td>
</tr>
</tbody>
</table>
q_lp_y (jpegxr_fwd_quant_dclp_N15_0) 3106 3106
q_dc_u (jpegxr_fwd_quant_dclp_N1_2) 292 292
q_lp_u (jpegxr_fwd_quant_dclp_N3_0) 892 892
q_dc_v (jpegxr_fwd_quant_dclp_N1_1) 292 292
q_lp_v (jpegxr_fwd_quant_dclp_N3_1) 894 894
pred_hp (jpegxr_pred_hp_0) 45122 45122
  mem (jpegxr_sync_ram_1rw_h_426_242_793_0) 96386 96386
  mem1 (jpegxr_jxr_pred_fifo_mem_h_542_242_174_0) 96388 96388
pred_dclp (jpegxr_pred_dclp_I_rst_rstif_mod__0) 101148 101148
jpegxr_top (jpegxr_top) 193929 193929

-------------------------------------------------------------------------------
Appendix C

Power Report

The power report is generated by Synopsys ASIC synthesize tool. The switching probability is set to 50% and the toggle rate is set to 50%.

Global Operating Voltage = 0.99

Power-specific unit information:
- Voltage Units = 1V
- Capacitance Units = 1.000000pf
- Time Units = ns
- Dynamic Power Units = 1mW (derived from V,C,T units)
- Leakage Power Units = 1nW

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Switch Power</th>
<th>Int Power</th>
<th>Leak Power</th>
<th>Total Power</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpegxr_top</td>
<td>3.110</td>
<td>4.557</td>
<td>1.89e+03</td>
<td>7.669</td>
<td>100.0</td>
</tr>
<tr>
<td>pred_dclp</td>
<td>1.618</td>
<td>2.311</td>
<td>961.482</td>
<td>3.930</td>
<td>51.2</td>
</tr>
<tr>
<td>mem1</td>
<td>1.531</td>
<td>2.206</td>
<td>913.579</td>
<td>3.738</td>
<td>48.7</td>
</tr>
<tr>
<td>mem</td>
<td>1.531</td>
<td>2.206</td>
<td>913.540</td>
<td>3.738</td>
<td>48.7</td>
</tr>
<tr>
<td>pred_hp</td>
<td>0.923</td>
<td>1.265</td>
<td>439.140</td>
<td>2.189</td>
<td>28.5</td>
</tr>
<tr>
<td>q_lp_v</td>
<td>9.87e-03</td>
<td>1.56e-02</td>
<td>8.971</td>
<td>2.55e-02</td>
<td>0.3</td>
</tr>
<tr>
<td>q_lp_u</td>
<td>9.95e-03</td>
<td>1.56e-02</td>
<td>9.012</td>
<td>2.56e-02</td>
<td>0.3</td>
</tr>
<tr>
<td>q_dc_v</td>
<td>2.47e-03</td>
<td>5.45e-03</td>
<td>2.874</td>
<td>7.92e-03</td>
<td>0.1</td>
</tr>
<tr>
<td>q_dc_u</td>
<td>2.47e-03</td>
<td>5.45e-03</td>
<td>2.874</td>
<td>7.92e-03</td>
<td>0.1</td>
</tr>
<tr>
<td>q_hp</td>
<td>3.01e-02</td>
<td>4.32e-02</td>
<td>32.971</td>
<td>7.33e-02</td>
<td>1.0</td>
</tr>
<tr>
<td>pct_2_v</td>
<td>9.43e-04</td>
<td>3.27e-03</td>
<td>2.073</td>
<td>4.21e-03</td>
<td>0.1</td>
</tr>
<tr>
<td>h0</td>
<td>2.98e-02</td>
<td>4.32e-02</td>
<td>31.867</td>
<td>7.30e-02</td>
<td>1.0</td>
</tr>
<tr>
<td>pct_2_u</td>
<td>0.127</td>
<td>0.189</td>
<td>127.100</td>
<td>0.316</td>
<td>4.1</td>
</tr>
<tr>
<td>odd_odd1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>CDF</td>
<td>Size</td>
<td>Bitrate</td>
<td>Length</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
<td>--------</td>
<td>---------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>odd2 (jpegxr_odd_transform_1)</td>
<td>1.30e-02</td>
<td>2.06e-02</td>
<td>25.520</td>
<td>3.37e-02</td>
<td>0.4</td>
</tr>
<tr>
<td>odd1 (jpegxr_odd_transform_2)</td>
<td>1.30e-02</td>
<td>2.06e-02</td>
<td>27.673</td>
<td>3.37e-02</td>
<td>0.4</td>
</tr>
<tr>
<td>h2 (jpegxr_hadamard_2)</td>
<td>1.30e-02</td>
<td>2.06e-02</td>
<td>27.822</td>
<td>3.37e-02</td>
<td>0.4</td>
</tr>
<tr>
<td>h1 (jpegxr_hadamard_3)</td>
<td>1.30e-02</td>
<td>2.06e-02</td>
<td>13.597</td>
<td>3.36e-02</td>
<td>0.4</td>
</tr>
<tr>
<td>pct_1 (jpegxr_fwd_4x4pct_stage1_0)</td>
<td>0.197</td>
<td>0.284</td>
<td>142.722</td>
<td>0.482</td>
<td>6.3</td>
</tr>
<tr>
<td>odd_odd1 (jpegxr_odd_odd_transform_0)</td>
<td>1.30e-02</td>
<td>2.08e-02</td>
<td>25.209</td>
<td>3.39e-02</td>
<td>0.4</td>
</tr>
<tr>
<td>odd2 (jpegxr_odd_transform_3)</td>
<td>1.32e-02</td>
<td>2.11e-02</td>
<td>27.992</td>
<td>3.43e-02</td>
<td>0.4</td>
</tr>
<tr>
<td>odd1 (jpegxr_odd_transform_0)</td>
<td>1.30e-02</td>
<td>2.08e-02</td>
<td>27.681</td>
<td>3.39e-02</td>
<td>0.4</td>
</tr>
<tr>
<td>h2 (jpegxr_hadamard_4)</td>
<td>1.31e-02</td>
<td>2.08e-02</td>
<td>13.995</td>
<td>3.39e-02</td>
<td>0.4</td>
</tr>
<tr>
<td>h1 (jpegxr_hadamard_0)</td>
<td>1.56e-02</td>
<td>2.75e-02</td>
<td>13.514</td>
<td>4.32e-02</td>
<td>0.6</td>
</tr>
</tbody>
</table>