CHAPTER 2

VIDEO DECOMPRESSSION USING H.264/
ADVANCED VIDEO CODING

2.1 SCOPE

Of-late H.264 is one of the promising audio-video standards which are gaining importance in many audio-video applications including mobile. The main goals of H.264 standard are to enhance compression performance and to provide a network friendly representation addressing video telephony and for storage, broadcast or streaming applications. The H.264 decoder standard is ported on TI DSP for the different set of H.264 encoded frames. This chapter briefs about the analysis of the computational complexity of a software-based H.264/AVC baseline profile decoder suitable for mobile applications. The analysis is based on determining the number of CPU cycles required by the H.264 baseline decoder to perform the key decoding sub-functions. To estimate the computational complexity of an H.264/AVC baseline decoder implementation, it is important to understand its two major components: time complexity and space (or storage) complexity. Time complexity is measured by the number of CPU cycles required to execute a specific implementation of an algorithm. Storage complexity is measured by
the approximate amount of memory required to implement an algorithm. The cost of hardware for a given application is a function of both time and storage complexities as both affect the size of the underlying blocks. The encoded frames such as Akiyo, Foreman, Mobile, News and Suzie are considered for experimentation. The experiments are conducted using H.264 decoder standard on a DSP which is the core processor of most of the handheld devices with and without deblocking filter.

### 2.2 INTRODUCTION TO ADVANCED VIDEO CODING

The H.264/AVC Baseline profile decoder is compatible designed specifically to meet the requirements of multimedia streaming and TV applications in mobile handsets. Baseline profile decoder supports playback resolutions of QCIF up to 15 frames per second and also delivers the required performance and power savings for the emergent applications such as mobile TV.
The figure 2.1 shows the block diagram of H.264 decoder. The entropy decoding unit parses the incoming bit stream extracting the block residual information as well as other syntax elements. The block residual information is inverse quantized and transformed before being added to the Prediction data. Such prediction data is generated by either the intra or inter prediction unit, according to the decoded syntax elements. The intra prediction unit is capable of generating a prediction for all intra frames. The inter prediction unit can generate a prediction using a single motion vector down to ¼ pixel. The sub pixels are generated according to the interpolation filters specified in the ITU-T H.264 Baseline specification. Each macro block is optionally filtered according to the information extracted from the bit stream, before being output. A reconstructed macro block is also stored in the external memory in order to be subsequently used by the inter prediction unit.

2.3 METHODOLOGY USED FOR DECODING

Figure 2.2: Steps for Decoding of H.264 encoded frames
The critical routines of the H.264 decoder are investigated and computation complexity of the routines is measured. For the performance analysis of H.264 Baseline profile decoder, the profiler is used which is a tool to measure the proportion of time or processing cycles spent in each subroutine. The profiler provides an estimate of actual
cycles spent for implementing particular function/routine and memory requirements. Hence the profiler provides an estimate on how much of the execution time can be attributed to each grain selected. This provides a guideline for assembly level optimization. Figure 2.3 shows the flow chart of decoding H.264 encoded frames.

The software modules with their flow graphs to understand the software architecture of the proposed H.264 decoder is shown in figure 2.3 and figure 2.4 depicts the representation of the video sequence in terms of blocks. The program execution starts with the main function. The initialization routine initializes the 128KB of Cache memory and input configuration open the encoded bit streams in read mode. The H.264Create Decoder routine will create a decoder instance and returns a pointer to the decoder structure in addition it sets the default values to all the necessary variables. The Load bit stream function fills the buffer with required amount of input data from the input file and it returns a pointer to the bit stream structure. It copies the data from file to the input buffer. The function H.264 Decode is the heart of the decoder program which decodes a slice of the picture. It initially reads a slice of data from the buffer and then decodes each slice repeatedly after which it is passed through deblocking filter and stores a picture and flushes the frames from the input buffer. The H.264 delete decoder function deletes the
decoder instance and frees all of the allocated memory. After all the bit streams are read from the input file (till the end of the file), the streams are decoded and closes the input function.

2.4 IMPLEMENTATION DETAILS

The test streams, stored in a file, are read at a picture basis and written in a stream buffer allocated in external memory. The decoder reads the stream from this memory decodes a picture and writes it in a picture buffer and also in a file. A PC running Code Composer Studio v3.1 has been used to carry out the simulation profiles. Results are given in average clock cycles per frame for the full decoder and for its main functional blocks.

The H.264 decoder is implemented on TI DSP TMS320 DM642 operating at 600 MHz, processor. The different video inputs are considered and the experimentation is done with and without deblocking filter. The sample result is displayed for further discussion. The first stage in any optimization process is to identify the critical routines and measure their current performance. The sections of code which utilize maximum of processor time out of the total time taken by the application to complete one execution is identified.

Input Parameter list

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input file name</td>
<td>Suzie</td>
</tr>
<tr>
<td>Resolution</td>
<td>176x144</td>
</tr>
<tr>
<td>Number of frames</td>
<td>150</td>
</tr>
<tr>
<td>Color space</td>
<td>YUV 4:2:0</td>
</tr>
<tr>
<td>Frames per Second</td>
<td>30</td>
</tr>
<tr>
<td>Source</td>
<td>Uncompressed progressive</td>
</tr>
<tr>
<td>Compressed by</td>
<td>27:1</td>
</tr>
</tbody>
</table>

Figure 2.5: Frame of the Suzie sequence
The other inputs considered for the experimentation are Akiyo, Foreman, Mobile and News. The details of the parameters for the input considered are shown in figures 2.6 to 2.9.

**Input file name** : Akiyo
**Resolution** : 176x144
**Number of frames** : 150
**Color space** : YUV 4:2:0
**Frames per Second** : 30
**Source** : Uncompressed progressive

Figure 2.6: Frame of the Akiyo sequence

**Input file name** : Foreman
**Resolution** : 176x144
**Number of frames** : 150
**Color space** : YUV 4:2:0
**Frames per Second** : 30
**Source** : Uncompressed progressive

Figure 2.7: Frame of the Foreman sequence

**Input file name** : Mobile
**Resolution** : 176x144
**Number of frames** : 150
**Color space** : YUV 4:2:0
**Frames per Second** : 30
**Source** : Uncompressed progressive

Figure 2.8: Frame of the Mobile sequence
2.5 EXPERIMENTAL RESULTS AND DISCUSSIONS

The experimental result obtained for Suzie input is discussed in detail to estimate the complexity involved in various blocks of the H.264 decoder. The computational complexity and the memory requirement of each block are discussed in detail. The detailed analysis is required since the analysis gives an insight for optimization of the code used in various blocks.

Table 2.1: Profiling result for the input Suzie ported on TIDSP to decode 15 frames

<table>
<thead>
<tr>
<th>Sl no</th>
<th>Function</th>
<th>Memory (Bytes)</th>
<th>Time (ms)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>h264dec_common.c</td>
<td>32</td>
<td>0</td>
<td>Contains common data types, macros &amp; functions</td>
</tr>
<tr>
<td>2</td>
<td>h264dec_file_ip.c</td>
<td>612</td>
<td>5.376</td>
<td>Input file configuration</td>
</tr>
<tr>
<td>3</td>
<td>h264dec_file_op.c</td>
<td>968</td>
<td>0.022</td>
<td>Output file configuration</td>
</tr>
<tr>
<td></td>
<td>File Name</td>
<td>Lines</td>
<td>Time</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>-------------------------------</td>
<td>-------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>4</td>
<td>h264dec_decoder.c</td>
<td>1384</td>
<td>261.97</td>
<td>Main function</td>
</tr>
<tr>
<td>5</td>
<td>h264dec_sliceheader.c</td>
<td>5620</td>
<td>0.5985</td>
<td>Define structure for slice headers and parses them</td>
</tr>
<tr>
<td>6</td>
<td>h264dec_block.c</td>
<td>1704</td>
<td>384</td>
<td>Handles coefficient transformation and block entering at a high level</td>
</tr>
<tr>
<td>7</td>
<td>h264dec_nalu.c</td>
<td>3360</td>
<td>52.246</td>
<td>Contains a simple NAL unit parser for H.264 elementary streams.</td>
</tr>
<tr>
<td>8</td>
<td>h264dec_paramset.c</td>
<td>3146</td>
<td>0.163</td>
<td>Decode a sequence parameter set (SPS), Decode a picture parameter set (PPS), use sequence &amp; picture parameter set</td>
</tr>
<tr>
<td>9</td>
<td>h264dec_dpb.c</td>
<td>8720</td>
<td>1.076</td>
<td>Memory management</td>
</tr>
<tr>
<td>10</td>
<td>h264dec_init.c</td>
<td>4908</td>
<td>1.492</td>
<td>Initialize decoder</td>
</tr>
<tr>
<td>11</td>
<td>h264dec_vlc.c</td>
<td>8632</td>
<td>440.4</td>
<td>Function for parsing Exp-Golomb and arbitrary codes</td>
</tr>
<tr>
<td>12</td>
<td>h264dec_interprediction.c</td>
<td>11848</td>
<td>1461.72</td>
<td>Get &amp; interpolation of full, 1/2 and 1/4 sub pixel, calculate predictor motion vector, initializes list depending on current picture type &amp; reorder.</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td>Memory Usage</td>
<td>Computation Time</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>---------------------------</td>
<td>--------------</td>
<td>------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>13</td>
<td>h264dec_loopfilter.c</td>
<td>3460</td>
<td>1.66</td>
<td>Debloack a picture, deblock a macro block, Calculates 16 strength values for one stripe in a macro block, Filters one edge of 16 (luma) or 8 (chroma) pel.</td>
</tr>
<tr>
<td>14</td>
<td>h264dec_fmo.c</td>
<td>3268</td>
<td>2.171</td>
<td>Flexible macro block ordering</td>
</tr>
<tr>
<td>15</td>
<td>h264dec_intraprediction.c</td>
<td>11496</td>
<td>44.76</td>
<td>Implements the various intra prediction modes</td>
</tr>
<tr>
<td>16</td>
<td>h264dec_image.c</td>
<td>14160</td>
<td>706.7</td>
<td>Decodes a slice. This is where most of the work is done</td>
</tr>
</tbody>
</table>

The table 2.1 shows the functions used along with the memory usage and computation time for the input Suzie. From the table 2.1, it is possible to identify the critical routines in the decoder block and also the time spent in each function block. The total storage space required for the decoder is 83.318 KB. The result gives the estimation of the memory requirement and computation time in terms of clock cycles for the various important blocks. Stall cycles spent in execution can also be estimated from the result. The results shown in the table 2.1 also indicate that the most of the time is consumed in motion compensation, deblocking filter and 4 x 4 Inverse Discrete Cosine Transform (IDCT). Therefore, these modules can be selected for optimization and optimization results in decoding more frames per second. The deblocking filter consumes more than 30% of the total time required for decoding a frame. Since deblocking filter is highly adaptive and accounts for one third of the computational complexity of an H.264 decoder, implementation of the decoder without deblocking filter in considered for the mobile applications.
Figure 2.10: The decoded frames (Frames 1 to 12) using H.264 decoder
The figure 2.10 shows the decoded frame for the encoded video Suzie. The various parameters such as Mean Square Error (MSE), Peak Signal to Noise ratio (PSNR), structural Similarity Index Measure (SSIM), Mean Sum Absolute Difference (MSAD) and Video Quality Metric (VQM) are considered for the comparison of the decoder performance with and without deblocking filter. The definition of the various parameters considered and their significance with and without deblocking filter is discussed as follows.

2.5.1 Mean Square Error (MSE)

The Mean Square Error measures the difference between the frames which is usually applied to Human Visual System. It is based on pixel-pixel comparison of the image frames. Minimizing the MSE is equivalent to least-squares optimization in a minimum energy sense, for which many mathematical tools are available. MSE \(d(X,Y)\) is given by

\[
d(X,Y) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} (X_{i,j} - Y_{i,j})^2}{mn}
\]  

(2.1)

Where \(X_{i,j}\) is the input frame and \(Y_{i,j}\) is the decompressed frame.
Figure 2.11: MSE plot with deblocking filter on TI processor

Figure 2.12: MSE plot without deblocking filter on TI processor
The figures 2.11 and 2.12 shows the MSE plot for the decoder implemented with and without deblocking filter for the frames considered considering the different inputs. It is seen from the figure that the value of MSE is less for the decoder with deblocking filter than without deblocking filter.

2.5.2 Peak Signal to Noise Ratio (PSNR)

PSNR is measured on a logarithmic scale and depends on the MSE of original and an impaired image or video frame, relative to \((2^n - 1)^2\) (the square of the highest-possible signal value in the image, where \(n\) is the number of bits per image sample). For a given image or image sequence, high PSNR usually indicates high quality and low PSNR usually indicates low quality.

(2.2)

Figure 2.13: PSNR plot with deblocking filter on TI processor
The figures 2.13 and 2.14 show the PSNR plot for the decoder implemented with and without deblocking filter for the frames considered considering the different inputs. It is seen from the figure that the value of PSNR is more for the decoder with deblocking filter than without deblocking filter.

### 2.5.3 Mean Sum Absolute Difference (MSAD)

The value of this metric is the mean sum of the absolute difference of the color components in the correspondent points of image. This metric is used for testing codec’s and filters performance.
Figure 2.15: MSAD plot with deblocking filter on TI processor

Figure 2.16: MSAD plot without deblocking filter on TI processor
The figures 2.15 and 2.16 shows the MSAD plot for the decoder implemented with and without deblending filter for the frames considered considering the different inputs. It is seen from the figure that the value of MSAD is less for the decoder with deblending filter than without deblending filter.

2.5.4 Structural Similarity Index Measure (SSIM)

The SSIM metric is based on the evaluation of three different measures, the luminance, contrast, and structure comparison measures which are computed as

\[
I(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.4}
\]

\[
C(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.5}
\]

\[
s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{2.6}
\]

Where \(X\) and \(Y\) correspond to two different frames which is required to match, i.e. two different frames such as input and decompressed frame. \(\mu_X\), \(\sigma_X^2\), and \(\sigma_{XY}\) the mean of \(X\), the variance of \(X\), and the covariance of \(X\) and \(Y\) respectively, while \(C_1\), \(C_2\), and \(C_3\) are constants given by \(C_1 = (K_1 L)^2\), \(C_1 = (K_2 L)^2\) and \(C_3 = \frac{C_2}{2}\). \(L\) is the dynamic range for the sample data, i.e. \(L=255\) for 8 bit content and \(K_1<<1\) and \(K_2<<1\) are two scalar constants. Given the above measures the structural similarity can be computed as

\[
SSIM(x, y) = [I(x, y)]^\alpha . [C(x, y)]^\beta . [s(x, y)]^\gamma \tag{2.7}
\]
Figure 2.17: SSIM plot with deblocking filter on TI processor

Figure 2.18: SSIM plot without deblocking filter on TI processor
The figures 2.17 and 2.18 shows the SSIM plot for the decoder implemented with and without deblocking filter for the frames considered considering the different inputs. It is seen from the figure that the value of SSIM is more for the decoder with deblocking filter than without deblocking filter. Higher value of SSIM indicates that the decompressed frame is almost same as the input frame.

2.5.5 Video Quality Metrics (VQM)

VQM provides an objective measurement for perceived video quality. It measures the perceptual effects of video impairments including blurring, jerky/unnatural motion, global noise, blocks distortion and color distortion, and combines them into a single metric. VQM takes the original video and the decompressed video as inputs and it is computed using Video quality measurement tool. VQM is calculated using the following steps.

Step1: This step calibrates the sampled video in preparation for feature extraction. It estimates and corrects the spatial and temporal shift as well as the contrast and brightness offset of the processed video sequence with respect to the original video sequence.

Step2: This step extracts a set of quality features that characterizes perceptual changes in the spatial, temporal, and chrominance properties from spatial-temporal sub-regions of video streams using a mathematical function.

Step3: This step computes a set of quality parameters that describe perceptual changes in video quality by comparing features extracted from the processed video with those extracted from the original video.

VQM is computed using a linear combination of parameters calculated from previous steps.
Figure 2.19: VQM plot with deblocking filter on TI processor

Figure 2.20: VQM plot without deblocking filter on TI processor
The figures 2.19 and 2.20 shows the VQM plot for the decoder implemented with and without deblocking filter for the frames considered considering the different inputs. It is seen from the figure that the value of VQM is less for the decoder with deblocking filter than without deblocking filter. Lower value of VQM indicates that the decompressed frame is almost same as the input frame.

Table 2.2: Performance of the decoder with and without deblocking filter for the input Suzie

<table>
<thead>
<tr>
<th>Decoder block</th>
<th>MSE</th>
<th>PSNR</th>
<th>MSAD</th>
<th>SSIM</th>
<th>VQM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without deblocking Filter</td>
<td>26.13</td>
<td>33.98</td>
<td>3.684</td>
<td>0.898</td>
<td>1.526</td>
</tr>
<tr>
<td>With deblocking filter</td>
<td>12.76</td>
<td>37.10</td>
<td>2.54</td>
<td>0.948</td>
<td>1.056</td>
</tr>
</tbody>
</table>

From the figures 2.11 to 2.20 it is clear that the deblocking filter is essential if the quality of the video is very important for the given applications. If the quality of the video is not very important for the applications such as mobile applications, the decoder can be implemented without deblocking filter which reduces the computational complexity. The table 2.2 shows the significance of the deblocking filter. With deblocking filter the decoder achieves an average PSNR of 37.1dB for the given input stream in comparison with 33.98 dB without deblocking filter.

2.6 SUMMARY & CONCLUSIONS

The un-optimized code can decode approximately less number of frames per second. Hence the experimental result gives an indication of the critical routines which consumes more time for computation. The experimental result identifies the critical routines of the decoder block and critical routines are deblocking filter, inter-prediction, variable length coding and IDCT which consumes more time for computation. The
experiments are conducted without deblocking filter and the results are compared with deblocking filter. The quality of the video is satisfactory for the mobile applications even if the deblocking filter is removed which is substantiated in the results provided. Also, the power consumption can be minimized by removing deblocking filter which consumes some memory in the decoder block. One of the reasons for lesser number of frames decoded could be that the data is read from the memory and stored back in memory which increases the computation time for decoding. It is possible to decode more frames by displaying the video on a display device without storing the video in a memory. The critical routines identified can be selected for optimization. The optimized code will decode more frames per second. Similar experiments are conducted for various other encoded formats of different compression ratios representing different scenarios and features.

In applications where only few frames are to be considered such as highlights of an event, segmentation and summarization is one of the options. To play the video frame in a short duration, only selected frames need to be considered and the next section discusses about video segmentation and summarization.