CHAPTER 5

COMPRESSED DOMAIN H.264 DECODER

5.1. Architecture of Compressed Domain H.264 Decoder

This chapter explains the compressed domain decoder which is the first part of transcoder in detail. The core processes of decoder are H.264 parsing, transform translation, intra prediction, motion compensation, reconstruction with drift management and deblocking. The architecture of compressed domain decoder is shown in Fig. 5.1.

The input to the decoder is H.264 Baseline Profile bitstream which is the compressed bitstream of YUV 4:2:0 video sequence. Here 4:2:0 represents the type of Chroma sub-sampling of YUV sequence. The outputs of decoder are the decoded syntax elements and the compressed domain decoded frame. The decoded frame is sent to compressed domain resizer to change its resolution. The decoded syntax elements are sent to reuse engine to derive new set of syntax elements to be used by encoder.

In any H.264 compliant decoder, the compressed bitstream is decoded using the procedure given by H.264 Standard. During this decoding, the syntax elements and residue coefficients are obtained. The residue coefficients are de-quantized, inverse transformed to get reconstructed residue values. With the help of syntax elements, the predicted values are calculated with reference to previously reconstructed frames through intra prediction and motion compensation. The reconstructed residue values are added with predicted values to form reconstruction frame. The reconstructed frame is filtered by deblocking filter in spatial domain. All the operations involved in prediction, reconstruction and filtering are performed in spatial domain.

In this research work, H.264 compliant decoder is constructed in compressed domain. In this decoder, the spatial domain operations such as intra prediction, motion compensation, reconstruction and filtering are performed in compressed domain. Drift management is performed during reconstruction to match the equivalence between spatial domain frame and compressed domain frame.
Fig. 5.1 Compressed Domain H.264 Baseline Decoder
The input, H.264 Baseline Profile compressed bitstream, is parsed according to H.264 Standard. The syntax elements are parsed by Exp-Golomb techniques and residue coefficients are parsed by CAVLD. The parsed syntax elements, which are SPS, PPS, SH and SD elements, describe the video properties of sequence, coding types of each frame and coding methods of each block in a frame. The parsed syntax elements are used in intra prediction and motion compensation in the decoder to perform prediction. The parsed residue coefficients are constructed to quantized residue values. These quantized residue coefficients are converted to the compressed domain reconstructed residue coefficients through a special convertor, called transform translator which is shown in Fig. 5.2.

![Fig. 5.2 Transform Translator in H.264 Baseline Decoder](image)

This involves de-quantization and conversion to compressed domain coefficients. In the transform translator, the de-quantization was performed as clearly defined by H.264 Standard. The inverse H.264 transform and the forward integer transform are combined together as single operation. This is applied on the de-quantized residue coefficients to get converted to compressed domain reconstructed residue coefficients. These reconstructed residue coefficients are directly used in compressed domain reconstruction.

Now syntax elements are used to find predicted coefficients in compressed domain. There are two types of calculating predicted coefficients, namely (i) intra prediction and
(ii) motion compensation. Either of these two types is selected based on syntax elements. If the parsed syntax elements indicate I-MB, then intra prediction is performed. Motion compensation is performed for P-MB, which is indicated by syntax elements.

Intra prediction is the process of calculating the predicted coefficients guided by modes with reference to the present compressed domain reconstructed frame. In Baseline Profile compression, there are three intra prediction types for Luma and Chroma components, called Intra 4x4, Intra 16x16 and Intra Chroma. The syntax elements indicate the type of intra prediction for each component and its modes too as shown in Fig. 5.3.

```
Parsed Syntax Elements
Under Intra prediction

Luma

Intra 4x4 Modes
1. Vertical
2. Horizontal
3. DC
4. Diagonal Down Left
5. Diagonal Down Right
6. Vertical Right
7. Horizontal Down
8. Vertical Left
9. Horizontal Up

Intra 16x16 Modes
1. Vertical
2. Horizontal
3. DC
4. Plane

Intra Chroma Modes
1. DC
2. Horizontal
3. Vertical
4. Plane

Chroma
```

Fig. 5.3 Different types of intra prediction and its modes

For each I-MB, the Intra Luma modes and Intra Chroma mode are derived from the incoming H.264 bitstream. Based on the mode, the intra prediction is performed in compressed domain that gives the predicted coefficients of the I-MB.

Motion Compensation is the process of calculating compressed domain predicted coefficients from previously reconstructed frame (i.e., reference frame) with the help of MVs and block size. The compensation addresses all block types like P16x16, P16x8, P8x16, P8x8, P8x4, P4x8 and P4x4, including PSKIP.
For each P-MB, the MVDs and block types are derived from the incoming H.264 bitstream. Block type defines block sizes too. The motion compensation is performed for Luma and Chroma components as shown in Fig. 5.4 below. For each blocks of P-MB, MV is calculated with the help of the MV prediction by adding MVD from parsed syntax elements. The motion compensation involves two steps, namely, integer-pel motion compensation and sub-pel motion compensation. A 6-tap filter and bilinear filter are used to perform the motion compensation. The motion compensation process calculates the predicted coefficients of the P-MB.

![Fig. 5.4 Overview of Motion Compensation](image)

The reconstruction block adds the predicted coefficients and reconstructed residue coefficients and results the compressed domain reconstructed coefficients. There is drift error which occurs across blocks and frames, because of pixel overflow or underflow when the compressed domain reconstructed coefficients are inversed transformed into spatial domain. These errors can be detected in spatial domain only, but the entire prediction and reconstruction processes are done in compressed domain. This problem is solved by drift manager which controls the overflow and underflow of pixel values in compressed domain itself. The research work defined the threshold values to check the occurrence of pixel overflow in compressed domain. The reconstructed coefficients are stored in compressed domain reconstructed frame for future reference.
After constructing all the MBs, there could be blockiness in the reconstructed frame because the input H.264 bitstream is generated by processing the input raw video, block-by-block. There are three varieties of deblinking filters which remove this blockiness as mentioned by H.264 Standard. Based on the strength of blockiness between each 4x4 blocks horizontally and vertically in the frame, different deblinking filter is selected and applied. In this process, the reconstructed coefficients and the syntax elements define the strength of the blockiness. The procedure defined by H.264 Standard to find the strength for blockiness is followed here as it is. But different filters on compressed domain reconstructed coefficients are re-designed to suit with the compressed domain and applied in the research work.

The compressed domain reconstructed frame is stored and used as reference. The H.264 Standard defines that the number of reference frames supported by Baseline Profile is 16. Because the motion estimation process is same on any frame, this research work restricts the number of reference frames to one.

The organization of Compressed Domain Decoder is described as follows.
1. H.264 Baseline Parser (Section 5.2)
2. I-Macroblock Decoding (Section 5.3)
3. P-Macroblock Decoding (Section 5.4)
4. Compressed Domain Reconstruction (Section 5.5)
5. Drift Manager (Section 5.6), and
6. Deblinking Filter (Section 5.7)

5.2. H.264 Baseline Parser

The H.264 Baseline Parser parses the input H.264 bitstream and extracts the syntax elements first by Exp-Golomb Decoding and the residue coefficients for each MB by CAVLD technique.

5.2.1. Parsing process of Syntax Elements by Exp-Golomb Decoding

The parsing process of Syntax Elements by Exp-Golomb decoding is explained in Section 9.1 of H.264 Standard. After parsing the bitstream, the SPS, PPS, SH and SD syntax elements are obtained. The descriptions of SPS, PPS, SH and SD syntax elements are defined by H.264 Standard.
The SPS defines about the properties of video sequence decoded, which is shown in Fig. 5.5 such as the resolution of the video frame, the Chroma subsampling type, number of frames to be decoded, number of previously reconstructed frames to be referred. The PPS explains the properties of each picture or frame which is shown in Fig. 5.5, such as the type of encoding technique, the QP used for compression, deblocking filter parameters and Chroma QP offset.

![Fig. 5.5 SPS and PPS of Foreman H.264 Baseline bitstream](image)

A frame may have multiple slices. So all the slices per frame are explained one-by-one according to H.264 Standard by SH. In Baseline Profile, single slice per frame is considered. The properties such as slice type, number of picture, delta QP, Instantaneous Decoder Refresh frame and deblocking parameters of the slice were understood from SH set. There are only two slice types in Baseline Profile, namely I-slice and P-slice.

For each MB in a slice, the slice data parameters are extracted from bitstream. The SD gives information such as motion or mode information and residual coefficients of each
MB one-by-one in raster scanning order. Fig. 5.6 shows the raster scanning order of MBs arrangement in 176x144 (QCIF) frame.

![Fig. 5.6 The raster scanning order of Macroblocks](image)

5.2.2. CAVLD parsing process for transform coefficient levels

The H.264 Baseline Profile bitstream is parsed to get parsed residue coefficients of each MB after getting syntax elements by CAVLD. The coefficients are extracted for each 4x4 sMB in the order shown in Fig. 5.7 based on coded block pattern.

![Fig. 5.7 Order of sMBs in a Macroblock](image)

The coded block pattern is an 8-bit format which is shown in Fig. 5.8. The last four bits (4 – 7) are called coded block pattern Luma and the two bits (2 – 3) are called coded block pattern Chroma.

\[ Coded \ Block \ Pattern \ Luma = Coded \ Block \ Pattern \mod 16 \]
The order of 4x4 blocks in a Macroblock (Fig. 5.7)

Based on the coded block pattern, the residue coefficients of Luma component and DC and AC coefficients of Chroma components are parsed from bitstream by H.264 Baseline parser. If the 4th bit is set, the residue coefficients of 0th to 3rd blocks are extracted. If the 5th bit is set, the residue coefficients of 4th to 7th blocks are extracted. If the 6th bit is set, the residue coefficients of 8th to 11th blocks are extracted. If the 7th bit is set, the residue coefficients of 12th to 15th blocks are extracted. Otherwise, they are assumed to zero. If 2nd and 3rd bits are ‘01’, then Chroma DC coefficients are extracted. Chroma AC coefficients are assumed as zero. If 2nd and 3rd bits are ‘10’, the Chroma DC and AC coefficients are extracted. If the 2nd and 3rd bits are ‘00’, both the Chroma DC and AC coefficients are considered as zero.

Then the MB QP is calculated as follows.

\[
Q_{P_t} = Q_{P_{t-1}} + \Delta QP 
\]

where,

\[
Q_{P_t} = QP \ of \ present \ macroblock \\
Q_{P_{t-1}} = QP \ of \ previous \ macroblock \\
Q_P{0} = QP \ of \ first \ macroblock = \ Slice \ QP + \Delta QP \\
\Delta QP = delta \ QP
\]
The extracted residue coefficients may belong to any of the four types namely (i) Intra 16x16 MB, (ii) Intra 4x4 blocks (iii) Chroma 8x8 blocks and (iv) Inter blocks. These parsed residue coefficients are arranged in 4x4 matrix format as quantized residue coefficients in residue coefficients arrangement process. The quantized residue coefficients are sent to transform translator.

In Intra 16x16 MB, there are sixteen DC and sixteen sets of 15 AC residue coefficients. They are parsed and arranged as one MB as shown in Fig. 5.9.

![Fig. 5.9 Intra 16x16 Macroblock Parsing and Arranging Process](image)

For Intra 4x4 MB or Inter MB, sixteen coefficients are extracted for each 4x4 block in the order shown in Fig. 5.7 based on the coded block pattern Luma. The coded block pattern Luma is a four-bit pattern. If the bit is one, then sixteen coefficients of all four 4x4 blocks in that 8x8 block are extracted. If the bit is zero, the residue coefficients of that 8x8 block are assumed to zero. These quantized residue coefficients are arranged to 4x4 matrix format in residue coefficients arrangement process. This extraction and arranging process is repeated until all the coefficients of a MB are obtained.
For Chroma 8x8 MB, based on the coded block pattern Chroma, the DC and AC coefficients are extracted for each Chroma component blocks. While extracting DC coefficients, four DC coefficients for Cb component are extracted first and then another four DC coefficients for Cr component are extracted. The parsed four DC coefficients are arranged back to 2x2 matrix format in residue coefficients arrangement process, for each Chroma component.

Four sets of fifteen AC coefficients for each Chroma components are extracted in the order of blocks shown in Fig. 5.10 when coded block pattern Chroma is ‘2’. Otherwise, the coefficients are assumed as zero only and there is no extraction of coefficients. The parsed fifteen AC coefficients for each 4x4 block are arranged to 4x4 matrix format in residue coefficients arrangement process.

\[
\begin{array}{cc}
0 & 1 \\
2 & 3 \\
\end{array}
\]

Fig. 5.10 Order of sMBs in a Chroma Macroblock (4:2:0)

### 5.2.3. Residue Coefficients Arrangement process

The parsed coefficients are identified as either set of 16 coefficients or 15 coefficients, by syntax elements. Those coefficients are arranged in a 4x4 matrix format in this process. This involves inverse zig-zag scanning process for transform coefficients as shown in Fig. 5.11. The inverse scanning process maps the sequence of entries to the positions in the corresponding matrix.

\[
\begin{array}{cccc}
15 & 14 & 10 & 9 \\
13 & 11 & 8 & 3 \\
12 & 7 & 4 & 2 \\
6 & 5 & 1 & 0 \\
\end{array}
\]

Fig. 5.11 Zig-Zag Scan for a 4x4 sMB
The Table 5.1 provides the mapping from the index \( idx \) of input list of 16 elements to indices \( i \) and \( j \) of the 2D array \( c \).

<table>
<thead>
<tr>
<th>( idx )</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zig-zag</td>
<td>( c_{00} )</td>
<td>( c_{01} )</td>
<td>( c_{10} )</td>
<td>( c_{20} )</td>
<td>( c_{11} )</td>
<td>( c_{02} )</td>
<td>( c_{03} )</td>
<td>( c_{12} )</td>
<td>( c_{21} )</td>
<td>( c_{30} )</td>
<td>( c_{31} )</td>
<td>( c_{22} )</td>
<td>( c_{13} )</td>
<td>( c_{23} )</td>
<td>( c_{32} )</td>
<td>( c_{33} )</td>
</tr>
</tbody>
</table>

5.2.4. Conclusion of Parsing Process

The H.264 bitstream is parsed into syntax elements and residue coefficients. These parsed residue coefficients are converted to quantized residue coefficients. Based on MB type, the parsed syntax elements and quantized residue coefficients are sent to I-MB decoding process or P-MB decoding process. The same parsing process is repeated for each MB one-by-one.

5.3. I-Macroblock Decoding

The MB type is decided based on the mb_type, which is parsed in slice data as defined in the following Table 5.2. This table is derived from Table 7-11 and Table 7-13 of H.264 Standard (pp. 91 & 93). Each I-MB will have either Intra Luma 4x4 or Intra Luma 16x16 and Intra Chroma 8x8 details. Based on MB type, intra prediction is performed.

<table>
<thead>
<tr>
<th>Frame &amp; mb_type</th>
<th>Frame &amp; mb_type</th>
<th>Macrobloc type</th>
<th>Mode</th>
<th>Coded block pattern Chroma</th>
<th>Coded block pattern Luma</th>
</tr>
</thead>
<tbody>
<tr>
<td>- &amp; 0</td>
<td>P &amp; 0</td>
<td>Inter 16x16</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>- &amp; 1</td>
<td>P &amp; 1</td>
<td>Inter 16x8</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>- &amp; 2</td>
<td>P &amp; 2</td>
<td>Inter 8x16</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>- &amp; 3</td>
<td>P &amp; 3</td>
<td>Inter 8x8</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>- &amp; 4</td>
<td>P &amp; 4</td>
<td>Inter 8x8 ref0</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>I &amp; 0</td>
<td>P &amp; 5</td>
<td>Intra 4x4</td>
<td>NA</td>
<td>Fig. 5.8</td>
<td></td>
</tr>
<tr>
<td>I &amp; 1</td>
<td>P &amp; 6</td>
<td>Intra 16x16</td>
<td>Vertical</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I &amp; 2</td>
<td>P &amp; 7</td>
<td>Intra 16x16</td>
<td>Horizontal</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I &amp; 3</td>
<td>P &amp; 8</td>
<td>Intra 16x16</td>
<td>DC</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I &amp; 4</td>
<td>P &amp; 9</td>
<td>Intra 16x16</td>
<td>Plane</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I &amp; 5</td>
<td>P &amp; 10</td>
<td>Intra 16x16</td>
<td>Vertical</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I &amp; 6</td>
<td>P &amp; 11</td>
<td>Intra 16x16</td>
<td>Horizontal</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I &amp; 7</td>
<td>P &amp; 12</td>
<td>Intra 16x16</td>
<td>DC</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I &amp; 8</td>
<td>P &amp; 13</td>
<td>Intra 16x16</td>
<td>Plane</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I &amp; 9</td>
<td>P &amp; 14</td>
<td>Intra 16x16</td>
<td>Vertical</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
5.3.1. Intra Luma 4x4 Macroblock

Here, the MB type is ‘0’. Sixteen Intra 4x4 modes (one mode per 4x4 sMB) are derived at first. One Chroma mode, which is applicable for both Chroma components, is derived next. Then the quantized residue coefficients are translated to compressed domain reconstructed residue coefficients. Then compressed domain intra prediction based on modes is performed. The reconstruction of MB is executed and drift manager is applied to remove the drift error.

5.3.1.1. Identification of modes for Intra Luma 4x4 MB

The parsed syntax elements give Intra Luma 4x4 mode differences and Intra Chroma 8x8 mode. The Intra Luma modes for each 4x4 in a MB are calculated with the help of most probable modes. The mode of each 4x4 block (0 – Vertical, 1 – Horizontal, 2 – DC, 3 – Diagonal Down Left, 4 – Diagonal Down Right, 5 – Vertical Right, 6 – Horizontal Down, 7 – Vertical Left, and 8 – Horizontal Left) in the order shown in Fig. 5.7 is calculated as follows. The most probable mode is the minimum mode among the intra 4x4 modes of top and left sMBs. If the mode of neighbour block is not available or not the type, then it is considered as ‘2’ (DC mode). If the mode difference value is ‘0’, then the mode is equal to most probable mode. Else if the mode difference value is ‘1’ followed by three bits, then the last three bits are converted to decimal value. The range of decimal

<table>
<thead>
<tr>
<th>Frame &amp; mb_type</th>
<th>Frame &amp; mb_type</th>
<th>Macroblock type</th>
<th>Mode</th>
<th>Coded block pattern Chroma</th>
<th>Coded block pattern Luma</th>
</tr>
</thead>
<tbody>
<tr>
<td>I &amp; 10</td>
<td>P &amp; 15</td>
<td>Intra 16x16</td>
<td>Horizontal</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>I &amp; 11</td>
<td>P &amp; 16</td>
<td>Intra 16x16</td>
<td>DC</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>I &amp; 12</td>
<td>P &amp; 17</td>
<td>Intra 16x16</td>
<td>Plane</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>I &amp; 13</td>
<td>P &amp; 18</td>
<td>Intra 16x16</td>
<td>Vertical</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 14</td>
<td>P &amp; 19</td>
<td>Intra 16x16</td>
<td>Horizontal</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 15</td>
<td>P &amp; 20</td>
<td>Intra 16x16</td>
<td>DC</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 16</td>
<td>P &amp; 21</td>
<td>Intra 16x16</td>
<td>Plane</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 17</td>
<td>P &amp; 22</td>
<td>Intra 16x16</td>
<td>Vertical</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 18</td>
<td>P &amp; 23</td>
<td>Intra 16x16</td>
<td>Horizontal</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 19</td>
<td>P &amp; 24</td>
<td>Intra 16x16</td>
<td>DC</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 20</td>
<td>P &amp; 25</td>
<td>Intra 16x16</td>
<td>Plane</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 21</td>
<td>P &amp; 26</td>
<td>Intra 16x16</td>
<td>Vertical</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 22</td>
<td>P &amp; 27</td>
<td>Intra 16x16</td>
<td>Horizontal</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 23</td>
<td>P &amp; 28</td>
<td>Intra 16x16</td>
<td>DC</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 24</td>
<td>P &amp; 29</td>
<td>Intra 16x16</td>
<td>Plane</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>I &amp; 25</td>
<td>P &amp; 30</td>
<td>IPCM</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

*NA – Not Applicable*
values is ‘0’ to ‘7’. If the decimal value is less than most probable mode, then the mode is the decimal value only. Otherwise the mode is decimal value + 1. Now the quantized residue coefficients are translated to compressed domain reconstructed residue coefficients.

5.3.1.2. Transform Translator for Intra Luma 4x4 MB

The H.264 quantized residue coefficients are converted to compressed domain reconstructed residue coefficients. These reconstructed residue coefficients are independent of QP. Each block of quantized residue coefficients are converted to block of compressed domain reconstructed residue coefficients in transform translator, by de-quantization with $QP_i$, and the combination of H.264 inverse transform and core forward transform as shown in Fig. 5.2.

The de-quantization process as defined in H.264 Standard is followed.

$$DQX(y, x) = \text{sign}(QX(y, x)) \cdot \left| QX(y, x) \times 2^\left\lfloor \frac{QP}{6} \right\rfloor \right| \cdot Vi(y, x), \ y = 0 ... 3, x = 0 ... 3 \ldots \ldots \ldots (5-4)$$

where,

- $QX$ is the quantized values and $DQX$ is the dequantized values
- $Vi$ is the inverse scaling matrix merged with de-quantization. It has three unique values which are spread in a 4x4 matrix, $Vi = \begin{pmatrix} a & b & a & b \\ b & c & b & c \\ a & b & a & b \\ b & c & b & c \end{pmatrix}$

where,

- $a, b, \text{ and } c$ are derived from Table 5.3 based on $\text{mod}(QP, 6)$.

<table>
<thead>
<tr>
<th>Table 5.3 Inverse Scaling Matrix values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{mod}(QP, 6)$</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>$a$</td>
</tr>
<tr>
<td>$b$</td>
</tr>
<tr>
<td>$c$</td>
</tr>
</tbody>
</table>

The de-quantized coefficients are inverse transformed as compliant with H.264 Standard to get spatial domain values as follows.

$$res = \frac{(Ci \times DQX \times Ci^T)}{2^6} \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots (5-5)$$

where,

- $res$ is spatial domain 4x4 matrix and $DQX$ is dequantized 4x4 matrix
The spatial domain coefficients are converted to compressed domain values by core forward integer transform as follows.

\[ RES = Cf \times res \times Cf^T \] 

where,

- \( res \) is spatial domain 4x4 matrix and \( RES \) is compressed domain 4x4 matrix
- \( Cf = \begin{pmatrix} 1 & 1/2 & -1/2 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} \) and \( Cf^T \) is the transpose of \( Cf \)

At the end of transform translator, the compressed domain reconstructed residue coefficients are obtained for a MB.

5.3.1.3. Compressed domain Intra Luma 4x4 MB Prediction and Reconstruction

Based on the Intra Luma 4x4 mode, the compressed domain predicted coefficients of each 4x4-block are calculated. The compressed domain residue and predicted coefficients are added as reconstructed coefficients and drift manager controlled the compressed domain drift error. This reconstructed block is stored in the Luma reconstruction frame for further reference by prediction engine. The Intra Luma 4x4 mode is updated in the mode table which enables the most probable mode prediction process. At the end of 15th block, compressed domain reconstructed MB will be available at the reconstruction frame.

5.3.2. Intra Chroma 8x8 Macroblock

Intra Chroma mode and Chroma QP are calculated for each Intra Chroma 8x8 MB.

The Intra Chroma mode for both the Chroma components is derived as follows.

\[ Chroma \text{ in}tra \text{ mode} = \begin{cases} \text{DC,} & \text{chroma mode} = 0 \\ \text{Horizontal,} & \text{chroma mode} = 1 \\ \text{Vertical,} & \text{chroma mode} = 2 \\ \text{Plane,} & \text{chroma mode} = 3 \end{cases} \] 

Chroma QP (\( QPC \)) is calculated as follows.

\[ QPC = fn(QPi) + QPC_{offset} \] 

\( fn(QPi) \) is derived from Table 5.4.
**QPCoffset** is the Chroma Quantization Offset parsed as syntax element.

<table>
<thead>
<tr>
<th><strong>QP</strong></th>
<th>QPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 30</td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>33-34</td>
<td>33</td>
</tr>
<tr>
<td>&gt;34</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>QP</strong></th>
<th>36-37</th>
<th>38-39</th>
<th>40-41</th>
<th>42-44</th>
<th>45-47</th>
<th>&gt;47</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn(QP)</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
</tr>
</tbody>
</table>

The Chroma 8x8 MB residual coefficients are having two different types of coefficients, namely DC and AC coefficients. These quantized DC and AC residue coefficients are sent to transform translator to get compressed domain reconstructed residue coefficients.

1. The 2x2 DC and AC Coefficients are de-quantized first using Equation (5.4). The Vi values, in the case of 2x2 DC, are 'a' only.
2. The de-quantized DC coefficients are inverse HADAMARD transformed as follows.
   \[ x = (H_i \times X \times H_i^T) \gg 2 \]
   \[ (5.9) \]
   where,
   - \( x \) is inverse HADAMARD transformed 2x2 matrix,
   - \( X \) is compressed domain 2x2 matrix,
   - \( H_i = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \) and \( H_i^T \) is the transpose of \( H_i \)
3. These DC coefficients are combined with AC coefficients as shown in Fig. 5.12.
4. The combined DC and AC coefficients are inverse transformed to get pixel values. Here the inverse transform is applied on each 4x4 blocks.
5. The compressed domain 8x8 residue coefficients are obtained by performing core forward transform on each 4x4 pixel blocks.

Based on the Intra Chroma mode, the compressed domain predicted coefficients of 8x8 block of Cb component are calculated. The compressed domain residue and predicted coefficients are sent to reconstruction and drift manager to get compressed domain reconstructed block. This block is stored in the Cb reconstruction frame for further reference by compressed domain prediction engine. The same mode is used to predict the compressed domain predicted coefficients of 8x8 block of Cr component. In the same way, the reconstructed coefficients are calculated and stored in Cr reconstruction frame. This Cr reference frame is referred by compressed domain prediction engine.
5.3.3. Intra Luma 16x16 Macroblock

If the MB type is ‘1 – 24’, the MB is Intra Luma 16x16 and Intra Chroma 8x8 block. Only the Intra Chroma 8x8 mode is parsed. Based on the mb_type, the Intra Luma 16x16 mode and the coded block pattern Luma and Chroma are derived from Table 5.2. Intra Chroma 8x8 block is reconstructed as described in 5.3.2. The Intra 16x16 block residual coefficients have two different types of coefficients, namely DC and AC coefficients. Based on coded block pattern, the AC residue coefficients of Luma and Chroma components are extracted. Those quantized DC and AC residue coefficients of Intra Luma 16x16 MB are sent to transform translator to get compressed domain reconstructed residue coefficients. The steps of transform translator are given below.

1. The 4x4 DC and AC Coefficients are de-quantized first using Equation (5.4). The \( V_i \) values, in the case of 4x4 DC, are ‘a’ only.
2. The de-quantized 4x4 DC coefficients are inverse HADAMARD transformed as follows.
   \[
   x = (H_i^T \times X \times H_i) \gg 4 \…………………………………………………………… (5-10)
   \]
   where,
\( x \) is inverse HADAMARD transformed 4x4 matrix

\[ X \] is compressed domain 4x4 matrix

\[ Hi = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \]

and \( Hi^T \) is the transpose of \( Hi \)

3. The resultant DC coefficients are appropriately combined with AC coefficients.

4. The combined DC and AC coefficients are inverse transformed to get pixel values. Here the inverse transform is applied on each 4x4 blocks.

5. The compressed domain 16x16 residue coefficients are obtained by performing core forward transform on each 4x4 pixel blocks.

Based on the mode, the compressed domain predicted coefficients of 16x16 MB are calculated. The compressed domain residue and predicted coefficients are sent to reconstruction and drift manager to get compressed domain reconstructed block. This block is stored in the Luma reconstruction frame for further reference by compressed domain prediction engine.

5.3.4. Conclusion of I-Macroblock Decoding Process

Based on Luma and Intra Chroma modes, the I-MB is decoded and kept in reconstruction frame for further reference. The intra prediction, reconstruction processes in this decoding are done in compressed domain.

5.4. P-Macroblock Decoding

The MB type is decided based on the mb_type, which is parsed in slice data as defined in the following Table 5.2. There are five block types of P-MBs namely, P16x16, P16x8, P8x16, P8x8 and PSKIP. Each P-MB will have either P16x16 or P16x8 or P8x16 or P8x8 or PSKIP details. Here the block size of Chroma is half of the block size. For Eg. P16x8 means the block size of Luma MB is 16x8 and that of Chroma MB is 8x4. Based on MB type, inter prediction is performed. In P-MB type, the MV for Chroma is same as Luma. The block type indicates the block size that followed for a MB. The Fig. 5.13 shows the different block types in a frame in Mobile sequence. The decoding process of P16x16, P16x8, P8x16, P8x8 and PSKIP are explained as follows.
5.4.1. Inter 16x16 Macroblock (P16x16)

If the parsed syntax element, mb_type, is ‘0’, the MB type is P16x16 MB. Only one MVD is parsed from bitstream to represent the MB. The MV is calculated by adding MVD and MVP which is derived from MV prediction engine. Then the residue coefficients are converted to compressed domain reconstructed residue coefficients by transform translator as mentioned in Section 5.3.1.2. With the help of MV, the compressed domain predicted MB for Luma and Chroma components are calculated. The compressed domain predicted coefficients and residue coefficients are sent to reconstruction and drift manager to get compressed domain reconstructed MB. The reconstructed MB coefficients are stored appropriately in reconstructed Luma and Chroma frames. The same process is applicable for other block sizes of P-MBs.

5.4.2. Inter 16x8 Macroblock (P16x8)

If the mb_type is ‘1’, the MB type is P16x8 MB. There are two MVDs which are parsed from bitstream to represent the top 16x8 and bottom 16x8 blocks in a MB as shown in Fig. 5.14. The MV for the top 16x8-block is calculated by MV prediction engine. With the help of top MV, the compressed domain predicted coefficients of top 16x8-block are calculated. The compressed domain predicted and residue coefficients are sent to reconstruction and drift manager to get compressed domain reconstructed MB.
These coefficients are stored in reconstructed frame. Then the MV for the bottom 16x8-block is calculated by MV prediction engine. The compressed domain prediction, reconstruction and drift management of bottom 16x8-block are done. The reconstructed coefficients are stored in reconstructed frame.

5.4.3. Inter 8x16 Macroblock (P8x16)

If the mb_type is ‘2’, the MB type is P8x16 MB. There are two MVDs which are parsed from bitstream to represent the left 8x16 and right 8x16 blocks in a MB as shown in Fig. 5.15. The above-said process is done for left 8x16-block and right 8x16-block. The reconstructed coefficients are stored in reconstructed frame.

5.4.4. Inter 8x8 Macroblock (P8x8)

If the mb_type is ‘3’ or ‘4’, the MB type is P8x8 MB. The mb_type ‘4’ represents that there is only one reference frame. There are four 8x8 sMBs available in a MB shown in Fig. 5.16. For each 8x8 sMB, sMB type (sub_mb_type) is parsed from bitstream. There are atleast four MVDs which are parsed from bitstream to represent each block in a MB.

If the sub_mb_type of an 8x8 sMB is ‘0’, then sMB is P8x8 type that requires only one MV. If the sub_mb_type is ‘1’, then sMB type is P8x4, which requires two MVs. If the sub_mb_type is ‘2’, then sMB type is P4x8, which requires two MVs. If the sub_mb_type is ‘3’, then sMB type is P4x4, which requires four MVs.

The reconstruction is done on each 8x8 sMB one-by-one such as topleft, topright, bottom left and bottom right 8x8 sMB in order. If the sMB type is P8x8, then the MV for the 8x8-block is calculated by MV prediction engine. With the help of MV, the predicted coefficients of that 8x8-block are calculated. The predicted and residue coefficients are
sent to reconstruction and drift manager to get reconstructed sMB. These coefficients are stored in compressed domain reconstructed frame.

If the sMB type is P8x4, then above-said process is done for top 8x4-block and bottom 8x4-block. If the sMB type is P4x8, then the above-said process is performed for left 4x8 sMB and then right 4x8 sMB. If the sMB type is P4x4, then the above-said process is performed for topleft 4x4, topright 4x4, bottomleft 4x4 and bottom right 4x4 blocks. These coefficients are stored in reconstructed frame.

5.4.5. Inter SKIP Macroblock (PSKIP)

If the mb_type is ‘0’, the MB type is Inter 16x16 (P16x16) MB. If the SKIP flag is enabled, then number of MBs to be skipped is parsed from bitstream. For each skipped MBs, the MV is calculated by MV prediction engine. With the help of MV, the predicted MB is calculated. The residue coefficients are assumed to be zero in SKIP MBs. The predicted coefficients are sent to reconstruction and drift manager to get reconstructed MB. These coefficients are stored in compressed domain reconstructed frame.

5.4.6. Conclusion of P-Macroblock Decoding

Based on MB type, sMB types and MVs, the Luma and Chroma components of P-MB are decoded, reconstructed and kept in reconstruction frame for further reference. The motion compensation and reconstruction processes in this decoding are performed in compressed domain.

5.5. Compressed Domain Reconstruction

As the additive property of core forward transform is satisfied, the sum of transform coefficients gives sum of pixel coefficients if inverse transform is performed.

If $rec \xrightarrow{T} REC, pred \xrightarrow{T} PRED$ and $rres \xrightarrow{T} RRES$, then

$$(pred + rres) \xrightarrow{T} (PRED + RRES)$$

(5-11)

The compressed domain reconstruction is the sum of predicted values with reconstructed residual values in compressed domain. This is done for each 4x4 sMB in a MB.

$$REC = PRED + RRES$$

(5-12)
5.6. Drift Manager

After the spatial domain reconstruction, each pixel values are ensured to have positive 8 bits, i.e., if the pixel value exceeds 255, it is brought back to 255 or if the pixel value lowered below 0, it is brought forward to 0. This operation is called ClipY in H.264 Standard. The compressed domain reconstructed values should reflect equivalent to the spatial domain reconstructed values.

In the compressed domain, there is no possibility to get indicated by this excessive pixel value. If the compressed domain allows the excessive value, this creates drift error in reconstructed values, in turn, all other predicted values to the extent of quality degradation at reconstructed frame. This drift error is avoided by drift manager which worked on compressed domain. The steps for finding the possibility of occurrence of excessive pixel values and correcting those of a given compressed domain 4x4 coefficients \((h)\) are given below.

5.6.1. Pre-processing for pseudo values

The block is pre-processed to get pseudo values, as follows.

\[
d = P \times h \times P^T
\]

where,

\[
P = \begin{pmatrix}
64 & 51 & 64 & 25 \\
64 & 25 & -64 & -52 \\
64 & -26 & -64 & 51 \\
64 & -52 & 64 & -26
\end{pmatrix}
\]

i.e., \(P = [2^8 \times C_f^{-1}]\)

This process involves lots of multiplication and summation. So it is simplified as horizontal and vertical transformation flows with addition and shifting operations only in order to implement in hardware.

Horizontal Transformation flow is given below.

\[
g(1,x) = (h(1,x) + h(3,x)) \gg 1, x = 1 \ldots 4
\]
\[
g(2,x) = (h(1,x) - h(3,x)) \gg 1, x = 1 \ldots 4
\]
\[
t(x) = (h(2,x) - 2 \times h(4,x)) \gg 1, x = 1 \ldots 4
\]
\[
g(3,x) = (t(x) \gg 3) + (t(x) \gg 4) + (t(x) \gg 7) + (t(x) \gg 8), x = 1 \ldots 4
\]
\[
t(x) = (2 \times h(2,x) + h(4,x)) \gg 1, x = 1 \ldots 4
\]
\[
g(4,x) = (t(x) \gg 3) + (t(x) \gg 4) + (t(x) \gg 7) + (t(x) \gg 8), x = 1 \ldots 4
\]
\[
f(1,x) = (g(1,x) + g(4,x)) \gg 1, x = 1 \ldots 4
\]
\[
f(2,x) = (g(2,x) + g(3,x)) \gg 1, x = 1 \ldots 4
\]
\[
f(3,x) = (g(2,x) - g(3,x)) \gg 1, x = 1 \ldots 4
\]
\[
f(4,x) = (g(1,x) - g(4,x)) \gg 1, x = 1 \ldots 4
\]
Vertical Transformation flow is given below.

\[ e(y, 1) = (f(y, 1) + f(y, 3)) \gg 1, y = 1 \ldots 4 \]
\[ e(y, 2) = (f(y, 1) - f(y, 3)) \gg 1, y = 1 \ldots 4 \]
\[ t(y) = (f(y, 2) - 2 \times f(y, 4)) \gg 1, y = 1 \ldots 4 \]
\[ e(y, 3) = (t(y) \gg 3) + (t(y) \gg 4) + (t(y) \gg 7) + (t(y) \gg 8), y = 1 \ldots 4 \]
\[ t(y) = (2 \times f(y, 2) + f(y, 4)) \gg 1, y = 1 \ldots 4 \]
\[ e(y, 4) = (t(y) \gg 3) + (t(y) \gg 4) + (t(y) \gg 7) + (t(y) \gg 8), y = 1 \ldots 4 \]
\[ d(y, 1) = (e(y, 1) + e(y, 4)) \gg 1, y = 1 \ldots 4 \]
\[ d(y, 2) = (e(y, 2) + e(y, 3)) \gg 1, y = 1 \ldots 4 \]
\[ d(y, 3) = (e(y, 2) - e(y, 3)) \gg 1, y = 1 \ldots 4 \]
\[ d(y, 4) = (e(y, 1) - e(y, 4)) \gg 1, y = 1 \ldots 4 \]

5.6.2. Identification of Pixel excessive in compressed domain

If any of the values in the resultant matrix \(d\) is greater than 251 or less than 3, there is a possibility of pixel excessive. The above-said horizontal and vertical transformations are approximately the core inverse transform, but not the same. The core inverse transform involved scalar multiplications. But here, there is no scalar multiplication. Only shifting operations are used for deriving a matrix in which, the possibility can be identified.

5.6.3. Clipping

When the pixel excessive in compressed domain is identified, the compressed domain coefficients are corrected by doing

1. Core Inverse Transform
2. Clipping and
3. Core Forward Transform

Core Inverse Transform is performed on 4x4 sMB as the block size of forward transform is 4x4. The inverse transform is applied on 4x4 sMB of compressed domain values to get spatial domain values.

The core inverse integer transform is given below.

\[ x = (Ci \times X \times Ci^T) \gg 32 \]

where,

\[ x \text{ is spatial domain } 4x4 \text{ matrix} \]
\[ X \text{ is compressed domain } 4x4 \text{ matrix} \]
\[ Ci = \begin{pmatrix} 16384 & 13107 & 16384 & 6554 \\ 16384 & 6554 & -16384 & -13107 \\ 16384 & -6554 & -16384 & 13107 \\ 16384 & -13107 & 16384 & -6554 \end{pmatrix} \] and \( Ci^T \) is the transpose of \( Ci \)

Core Inverse Transform is modified to suit the hardware in two steps, horizontal and vertical transformations, which are explained below.

**Horizontal Transformation** is performed as given below.

\[
\begin{align*}
g(1, x) &= (X(1, x) + X(3, x)) \gg 1, \quad x = 1 \ldots 4 \\
g(2, x) &= (X(1, x) - X(3, x)) \gg 1, \quad x = 1 \ldots 4 \\
g(3, x) &= (13107 \times (X(2, x) - 2 \times X(4, x)) + 2^{15}) \gg 16, \quad x = 1 \ldots 4 \\
g(4, x) &= (13107 \times (2 \times X(2, x) + X(4, x)) + 2^{15}) \gg 16, \quad x = 1 \ldots 4
\end{align*}
\]

\[
\begin{align*}
f(1, x) &= (g(1, x) + g(4, x)) \gg 1, \quad x = 1 \ldots 4 \\
f(2, x) &= (g(2, x) + g(3, x)) \gg 1, \quad x = 1 \ldots 4 \\
f(3, x) &= (g(2, x) - g(3, x)) \gg 1, \quad x = 1 \ldots 4 \\
f(4, x) &= (g(1, x) - g(4, x)) \gg 1, \quad x = 1 \ldots 4
\end{align*}
\]

**Vertical Transformation** is performed as given below.

\[
\begin{align*}
e(y, 1) &= (f(y, 1) + f(y, 3)) \gg 1, \quad y = 1 \ldots 4 \\
e(y, 2) &= (f(y, 1) - f(y, 4)) \gg 1, \quad y = 1 \ldots 4 \\
e(y, 3) &= (13107 \times (f(y, 2) - 2 \times f(y, 4)) + 2^{15}) \gg 16, \quad y = 1 \ldots 4 \\
e(y, 4) &= (13107 \times (2 \times f(y, 2) + f(y, 4)) + 2^{15}) \gg 16, \quad y = 1 \ldots 4
\end{align*}
\]

\[
\begin{align*}
x(y, 1) &= (e(y, 1) + e(y, 4)) \gg 1, \quad y = 1 \ldots 4 \\
x(y, 2) &= (e(y, 2) + e(y, 3)) \gg 1, \quad y = 1 \ldots 4 \\
x(y, 3) &= (e(y, 2) - e(y, 3)) \gg 1, \quad y = 1 \ldots 4 \\
x(y, 4) &= (e(y, 1) - e(y, 4)) \gg 1, \quad y = 1 \ldots 4
\end{align*}
\]

After the spatial domain reconstruction, each pixel values are ensured to have positive 8 bits, i.e., the pixel value is clipped to 0 or 255 when it exceeds. If the clipping process is taken place, then core forward transform is applied on the clipped 4x4-block coefficients. The core forward transform is the same explained in the transform translator.

\[
X = Cf \times x \times Cf^T
\]  \hspace{1cm} \text{.................................................................................................................. (5-15)}

where,

\[
x \text{ is spatial domain 4x4 matrix and } X \text{ is compressed domain 4x4 matrix}
\]

\[
Cf = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} \text{ and } Cf^T \text{ is the transpose of } Cf.
\]

Core forward transform is modified to suit the hardware in two steps, horizontal and vertical transformations, which are explained below.
Horizontal transformation is performed row by row in a $4 \times 4$ sMB.

$$e(i, 1) = x(i, 1) + x(i, 4), i = 1 \ldots 4$$
$$e(i, 2) = x(i, 2) + x(i, 3), i = 1 \ldots 4$$
$$e(i, 3) = x(i, 2) - x(i, 3), i = 1 \ldots 4$$
$$e(i, 4) = x(i, 1) - x(i, 4), i = 1 \ldots 4$$

$$f(i, 1) = e(i, 1) + e(i, 2), i = 1 \ldots 4$$
$$f(i, 2) = 2 \times e(i, 4) + e(i, 3), i = 1 \ldots 4$$
$$f(i, 3) = e(i, 1) - e(i, 2), i = 1 \ldots 4$$
$$f(i, 4) = e(i, 4) - 2 \times e(i, 3), i = 1 \ldots 4$$

Vertical transformation is performed column by column.

$$g(1, j) = f(1, j) + f(4, j), j = 1 \ldots 4$$
$$g(2, j) = f(2, j) + f(3, j), j = 1 \ldots 4$$
$$g(3, j) = f(2, j) - f(3, j), j = 1 \ldots 4$$
$$g(4, j) = f(1, j) - f(4, j), j = 1 \ldots 4$$

$$X(1, j) = g(1, j) + g(2, j), j = 1 \ldots 4$$
$$X(2, j) = g(3, j) + 2 \times g(4, j), j = 1 \ldots 4$$
$$X(3, j) = g(1, j) - g(2, j), j = 1 \ldots 4$$
$$X(4, j) = g(4, j) - 2 \times g(3, j), j = 1 \ldots 4$$

### 5.6.4. Conclusion of Drift Manager

At the end of Drift Manager, the spatial domain excess values are clipped to 8-bit values. The pseudo values indicate the possibility of excess values, so actions can be taken. The probing of identifying the pixel excess is done in compressed domain. But this operation is performed in spatial domain values only. The compressed domain values will not reflect the pixel excess values.
5.7. Deblocking Filter

5.7.1. Introduction

Quantization error in the residuals of the intra-coded blocks and inter-coded blocks can induce discontinuities at the block boundaries. The quantization in H.264 Standard creates blockiness in the boundaries of each block at higher compression. Block may be 4x4 to 16x16. The blockiness is very visible in the video frame that affects the quality of the video. These discontinuities result in two undesirable consequences: (1) The discontinuities can show up as visible ‘blocky’ artifacts in the decompressed video frames, especially in smoothly textured regions and (2) The quality of the reconstructed frame as a predictor for future frames is reduced. To mitigate these effects, the H.264 uses an in-loop deblocking filter to attempt to remove the block-boundary discontinuities. Peter, L. et al. (2003) described the adaptive deblocking filter used in H.264.

The filtering is performed on the reconstructed frame prior to its use as a reference frame for the subsequent frame(s). Therefore, the encoder and decoder must perform the same filtering operation. After constructing the reconstruction frame (the entire frame), the deblocking filter is applied to reduce the blockiness. This process is adhered as per the H.264 Standard. This blockiness is removed or suppressed by deblocking filter which is well defined by H.264 Standard.

The adaptive deblocking filter described by Peter, L. et al. (2003) achieved substantial objective and subjective quality improvements with a reasonably simple algorithm. The good performance is based on reliable detection of real and artificially created edges and efficient filtering of the latter ones. Bitrate savings exceeding 9% are observed with equal PSNR levels together with significantly improved visual quality.

5.7.2. Deblocking process

The deblocking filter is applied only when the QP is greater than 26. This filter is applied on compressed domain reconstructed frames of Luma, Cb and Cr components. After applying the deblocking filter, the reconstructed frame is used as reference frame for predicting blocks for the next frame.

All the steps in deblocking filtering process except the pixel values (as per H.264 Standard) are followed here. Instead of pixel values, the compressed domain blocks are
used. The spatial domain values are obtained by inverse transform. Over the spatial domain values, the deblocking filter is applied. The deblocking filtered blocks are again transformed to compressed domain values and stored back in the reconstruction frame.

Filtering process will be applied on internal edges of a MB (called 4x4 blocks) and edges of a MB (shown in Fig. 5.17) depends on filter boundary strength calculated for each edges. The filter will be applied horizontally and vertically to smoothen the blocks.

![Fig. 5.17 Boundaries in a Macroblock to be filtered (Fig. 8.10 of H.264 Standard)](image)

For each MB, the filter is applied on the edges in following ways.
1. Left vertical MB edge and Left vertical internal edges
2. Top horizontal MB edge and Top horizontal internal edges

For each MB, the filter is applied on Luma components first, and then Chroma components. For each block, the boundary strength is calculated. The boundary strength is calculated based on the QP, modes, MVs and block type. If the boundary strength is greater than zero, then corresponding filter coefficients are derived as per the H.264 Standard.

Filtering is done on pixel values of the block; so compressed domain block is converted to spatial domain block and then filtering is applied on those pixel values. If the boundary strength is 4, then three pixels on both the sides with respect to boundary are filtered. Otherwise, two pixels are filtered. After filtering operation, the spatial domain values are transformed to compressed domain values and are stored back to reconstructed frame.
5.8. Conclusion of Compressed Domain H.264 Decoder

The input H.264 bitstream is parsed into syntax elements and residue coefficients. The reconstructed residue coefficients and the predicted MB are calculated based on syntax elements. The reconstructed MB is constructed by adding predicted and reconstructed residue coefficients. The pixel excessiveness is checked and controlled. So the drift error is avoided. The reconstructed frame is sent to resizer to change its screen size. The syntax elements are preserved for the reference of reuse engine.

In this research work, the compressed domain decoder is designed with intra prediction, motion compensation, and reconstruction, drift manager and transform translator. The entire flow of decoder is coded in MATLAB.

The compressed domain decoder coded in MATLAB can decode any H.264 Baseline Profile bitstream. Verification process of this decoder is shown in Fig. 5.18. H.264 bitstream was given as input of decoder. The compressed domain decoding operation was performed. The output of decoder was converted to spatial domain YUV sequences. The same bitstream was decoded by JM Decoder that generated YUV sequences. Those two YUV sequences were checked and found same for many sequences with different screen sizes and QPs. This experiment proved that the compressed domain decoder developed and coded in MATLAB is in compliance to H.264 Standard.

![Flowchart](image-url)  
**Fig. 5.18 Verifying the functionality of Compressed Domain Decoder**