When slice data partitioning is not used, coded slices are represented by a slice layer without partitioning RBSP that contains a slice header followed by a slice data syntax structure that contains all the syntax elements of categories 2, 3, and 4 (see category column in subclause 7.3) of the macroblock data for the macroblocks of the slice.

When slice data partitioning is used, the macroblock data of a slice is partitioned into one to three partitions contained in separate NAL units. Partition A contains a slice data partition A header, and all syntax elements of category 2. Partition B, when present, contains a slice data partition B header and all syntax elements of category 3. Partition C, when present, contains a slice data partition C header and all syntax elements of category 4.

When slice data partitioning is used, the syntax elements of each category are parsed from a separate NAL unit, which need not be present when no symbols of the respective category exist. The decoding process shall process the slice data partitions of a coded slice in a manner equivalent to processing a corresponding slice layer without partitioning RBSP by extracting each syntax element from the slice data partition in which the syntax element appears depending on the slice data partition assignment in the syntax tables in subclause 7.3.

NOTE 2 – Syntax elements of category 3 are relevant to the decoding of residual data of I and SI macroblock types. Syntax elements of category 4 are relevant to the decoding of residual data of P and B macroblock types. Category 2 encompasses all other syntax elements related to the decoding of macroblocks, and their information is often denoted as header information. The slice data partition A header contains all the syntax elements of the slice header, and additionally a slice_id that are used to associate the slice data partitions B and C with the slice data partition A. The slice data partition B and C headers contain the slice_id syntax element that establishes their association with the slice data partition A of the slice.

8.2.4 Decoding process for reference picture lists construction

This process is invoked at the beginning of decoding of each P, SP, or B slice.

Decoded reference pictures are marked as "used for short-term reference" or "used for long-term reference" as specified by the bitstream and specified in subclause 8.2.5. Short-term reference pictures are identified by the value of frame_num. Long-term reference pictures are assigned a long-term frame index as specified by the bitstream and specified in subclause 8.2.5.

Subclause 8.2.4.1 is invoked to specify

– the assignment of variables FrameNum, FrameNumWrap, and PicNum to each of the short-term reference pictures, and

– the assignment of variable LongTermPicNum to each of the long-term reference pictures.

Reference pictures are addressed through reference indices as specified in subclause 8.4.2.1. A reference index is an index into a reference picture list. When decoding a P or SP slice, there is a single reference picture list RefPicList0. When decoding a B slice, there is a second independent reference picture list RefPicList1 in addition to RefPicList0.

At the beginning of decoding of each slice, reference picture list RefPicList0, and for B slices RefPicList1, are derived as follows.

– An initial reference picture list RefPicList0 and for B slices RefPicList1 are derived as specified in subclause 8.2.4.2.

– The initial reference picture list RefPicList0 and for B slices RefPicList1 are modified as specified in subclause 8.2.4.3.

NOTE – The reordering process for reference picture lists specified in subclause 8.2.4.3 allows the contents of RefPicList0 and for B slices RefPicList1 to be modified in a flexible fashion. In particular, it is possible for a picture that is currently marked "used for reference" to be inserted into RefPicList0 and for B slices RefPicList1 even when the picture is not in the initial reference picture list derived as specified in subclause 8.2.4.2.

The number of entries in the modified reference picture list RefPicList0 is num_ref_idx_l0_active_minus1 + 1, and for B slices the number of entries in the modified reference picture list RefPicList1 is num_ref_idx_l1_active_minus1 + 1. A reference picture may appear at more than one index in the modified reference picture lists RefPicList0 or RefPicList1.

8.2.4.1 Decoding process for picture numbers

This process is invoked when the decoding process for reference picture lists construction specified in subclause 8.2.4 or the decoded reference picture marking process specified in subclause 8.2.5 is invoked.

The variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum are used for the initialisation process for reference picture lists in subclause 8.2.4.2, the modification process for reference picture lists in subclause 8.2.4.3, and for the decoded reference picture marking process in subclause 8.2.5.

To each short-term reference picture the variables FrameNum and FrameNumWrap are assigned as follows. First, FrameNum is set equal to the syntax element frame_num that has been decoded in the slice header(s) of the corresponding short-term reference picture. Then the variable FrameNumWrap is derived as

```
if( FrameNum > frame_num )
    FrameNumWrap = FrameNum – MaxFrameNum                    (8-28)
else
    FrameNumWrap = FrameNum
```

where the value of frame_num used in Equation 8-28 is the frame_num in the slice header(s) for the current picture.

Each long-term reference picture has an associated value of LongTermFrameIdx (that was assigned to it as specified in subclause 8.2.5).

To each short-term reference picture a variable PicNum is assigned, and to each long-term reference picture a variable LongTermPicNum is assigned. The values of these variables depend on the value of field_pic_flag and bottom_field_flag for the current picture and they are set as follows.

– If field_pic_flag is equal to 0, the following applies.
  – For each short-term reference frame or complementary reference field pair:

    PicNum = FrameNumWrap (8-29)

  – For each long-term reference frame or long-term complementary reference field pair:

    LongTermPicNum = LongTermFrameIdx (8-30)

    NOTE – When decoding a frame the value of MbaffFrameFlag has no influence on the derivations in subclauses 8.2.4.2, 8.2.4.3, and 8.2.5.

– Otherwise (field_pic_flag is equal to 1), the following applies.
  – For each short-term reference field the following applies.
    – If the reference field has the same parity as the current field

      PicNum = 2 * FrameNumWrap + 1 (8-31)

    – Otherwise (the reference field has the opposite parity of the current field),

      PicNum = 2 * FrameNumWrap (8-32)

  – For each long-term reference field the following applies.
    – If the reference field has the same parity as the current field

      LongTermPicNum = 2 * LongTermFrameIdx + 1 (8-33)

    – Otherwise (the reference field has the opposite parity of the current field),

      LongTermPicNum = 2 * LongTermFrameIdx (8-34)

8.2.4.2 Initialisation process for reference picture lists

This initialisation process is invoked when decoding a P, SP, or B slice header.

RefPicList0 and RefPicList1 have initial entries as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5.

When the number of entries in the initial RefPicList0 or RefPicList1 produced as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5 is greater than num_ref_idx_l0_active_minus1 + 1 or num_ref_idx_l1_active_minus1 + 1, respectively, the extra entries past position num_ref_idx_l0_active_minus1 or num_ref_idx_l1_active_minus1 are discarded from the initial reference picture list.

When the number of entries in the initial RefPicList0 or RefPicList1 produced as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5 is less than num_ref_idx_l0_active_minus1 + 1 or num_ref_idx_l1_active_minus1 + 1, respectively, the remaining entries in the initial reference picture list are set equal to "no reference picture".

8.2.4.2.1 Initialisation process for the reference picture list for P and SP slices in frames

This initialisation process is invoked when decoding a P or SP slice in a coded frame.

When this process is invoked, there shall be at least one reference frame or complementary reference field pair that is currently marked as "used for short-term reference" or "used for long-term reference".

The reference picture list RefPicList0 is ordered so that short-term reference frames and short-term complementary reference field pairs have lower indices than long-term reference frames and long-term complementary reference field pairs.

The short-term reference frames and complementary reference field pairs are ordered starting with the frame or complementary field pair with the highest PicNum value and proceeding through in descending order to the frame or complementary field pair with the lowest PicNum value.

The long-term reference frames and complementary reference field pairs are ordered starting with the frame or complementary field pair with the lowest LongTermPicNum value and proceeding through in ascending order to the frame or complementary field pair with the highest LongTermPicNum value.

NOTE – A non-paired reference field is not used for inter prediction for decoding a frame, regardless of the value of MbaffFrameFlag.

For example, when three reference frames are marked as "used for short-term reference" with PicNum equal to 300, 302, and 303 and two reference frames are marked as "used for long-term reference" with LongTermPicNum equal to 0 and 3, the initial index order is:

– RefPicList0[0] is set equal to the short-term reference picture with PicNum = 303,

– RefPicList0[1] is set equal to the short-term reference picture with PicNum = 302,

– RefPicList0[2] is set equal to the short-term reference picture with PicNum = 300,

– RefPicList0[3] is set equal to the long-term reference picture with LongTermPicNum = 0, and

– RefPicList0[4] is set equal to the long-term reference picture with LongTermPicNum = 3.

8.2.4.2.2 Initialisation process for the reference picture list for P and SP slices in fields

This initialisation process is invoked when decoding a P or SP slice in a coded field.

Each field included in the reference picture list RefPicList0 has a separate index in the reference picture list RefPicList0.

NOTE – When decoding a field, there are effectively at least twice as many pictures available for referencing as there would be when decoding a frame at the same position in decoding order.

Two ordered lists of reference frames, refFrameList0ShortTerm and refFrameList0LongTerm, are derived as follows. For purposes of the formation of this list of frames, decoded reference frames, complementary reference field pairs, non-paired reference fields and reference frames in which a single field is marked "used for short-term reference" or "used for long-term reference" are all considered reference frames.

– All frames having one or more fields marked "used for short-term reference" are included in the list of short-term reference frames refFrameList0ShortTerm. When the current field is the second field (in decoding order) of a complementary reference field pair and the first field is marked as "used for short-term reference", the first field is included in the list of short-term reference frames refFrameList0ShortTerm. refFrameList0ShortTerm is ordered starting with the reference frame with the highest FrameNumWrap value and proceeding through in descending order to the reference frame with the lowest FrameNumWrap value.

– All frames having one or more fields marked "used for long-term reference" are included in the list of long-term reference frames refFrameList0LongTerm. When the current field is the second field (in decoding order) of a complementary reference field pair and the first field is marked as "used for long-term reference, the first field is included in the list of long-term reference frames refFrameList0LongTerm. refFrameList0LongTerm is ordered starting with the reference frame with the lowest LongTermFrameIdx value and proceeding through in ascending order to the reference frame with the highest LongTermFrameIdx value.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList0ShortTerm and refFrameList0LongTerm given as input and the output is assigned to RefPicList0.

8.2.4.2.3 Initialisation process for reference picture lists for B slices in frames

This initialisation process is invoked when decoding a B slice in a coded frame.

For purposes of the formation of the reference picture lists RefPicList0 and RefPicList1 the term reference entry refers in the following to decoded reference frames or complementary reference field pairs.

When this process is invoked, there shall be at least one reference entry that is currently marked as "used for short-term reference" or "used for long-term reference".

For B slices, the order of short-term reference entries in the reference picture lists RefPicList0 and RefPicList1 depends on output order, as given by PicOrderCnt( ). When pic_order_cnt_type is equal to 0, reference pictures that are marked as "non-existing" as specified in subclause 8.2.5.2 are not included in either RefPicList0 or RefPicList1.

NOTE 1 – When gaps_in_frame_num_value_allowed_flag is equal to 1, encoders should use reference picture list reordering to ensure proper operation of the decoding process (particularly when pic_order_cnt_type is equal to 0, in which case PicOrderCnt( ) is not inferred for "non-existing" frames).

The reference picture list RefPicList0 is ordered such that short-term reference entries have lower indices than long-term reference entries. It is ordered as follows.

– Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having PicOrderCnt(entryShortTerm) less than PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refPicList0 in descending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refPicList0 in ascending order of PicOrderCnt(entryShortTerm).

– The long-term reference entries are ordered starting with the long-term reference entry that has the lowest LongTermPicNum value and proceeding through in ascending order to the long-term reference entry that has the highest LongTermPicNum value.

The reference picture list RefPicList1 is ordered so that short-term reference entries have lower indices than long-term reference entries. It is ordered as follows.

– Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having PicOrderCnt(entryShortTerm) greater than PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refPicList1 in ascending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refPicList1 in descending order of PicOrderCnt(entryShortTerm).

– Long-term reference entries are ordered starting with the long-term reference frame or complementary reference field pair that has the lowest LongTermPicNum value and proceeding through in ascending order to the long-term reference entry that has the highest LongTermPicNum value.

– When the reference picture list RefPicList1 has more than one entry and RefPicList1 is identical to the reference picture list RefPicList0, the first two entries RefPicList1[0] and RefPicList1[1] are switched.

NOTE 2 – A non-paired reference field is not used for inter prediction of frames (independent of the value of MbaffFrameFlag).

8.2.4.2.4 Initialisation process for reference picture lists for B slices in fields

This initialisation process is invoked when decoding a B slice in a coded field.

When decoding a field, each field of a stored reference frame is identified as a separate reference picture with a unique index. The order of short-term reference pictures in the reference picture lists RefPicList0 and RefPicList1 depend on output order, as given by PicOrderCnt( ). When pic_order_cnt_type is equal to 0, reference pictures that are marked as "non-existing" as specified in subclause 8.2.5.2 are not included in either RefPicList0 or RefPicList1.

NOTE 1 – When gaps_in_frame_num_value_allowed_flag is equal to 1, encoders should use reference picture list reordering to ensure proper operation of the decoding process (particularly when pic_order_cnt_type is equal to 0, in which case PicOrderCnt( ) is not inferred for "non-existing" frames).

NOTE 2 – When decoding a field, there are effectively at least twice as many pictures available for referencing as there would be when decoding a frame at the same position in decoding order.

Three ordered lists of reference frames, refFrameList0ShortTerm, refFrameList1ShortTerm and refFrameListLongTerm, are derived as follows. For purposes of the formation of these lists of frames the term reference entry refers in the following to decoded reference frames, complementary reference field pairs, or non-paired reference fields. When pic_order_cnt_type is equal to 0, the term reference entry does not refer to frames that are marked as "non-existing" as specified in subclause 8.2.5.2.

– Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having PicOrderCnt(entryShortTerm) less than or equal to PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refFrameList0ShortTerm in descending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refFrameList0ShortTerm in ascending order of PicOrderCnt(entryShortTerm).

  NOTE 3 – When the current field follows in decoding order a coded field fldPrev with which together it forms a complementary reference field pair, fldPrev is included into the list refFrameList0ShortTerm using PicOrderCnt(fldPrev) and the ordering method described in the previous sentence is applied.

– Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having PicOrderCnt(entryShortTerm) greater than PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refFrameList1ShortTerm in ascending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refFrameList1ShortTerm in descending order of PicOrderCnt(entryShortTerm).

  NOTE 4 – When the current field follows in decoding order a coded field fldPrev with which together it forms a complementary reference field pair, fldPrev is included into the list refFrameList1ShortTerm using PicOrderCnt(fldPrev) and the ordering method described in the previous sentence is applied.

– refFrameListLongTerm is ordered starting with the reference entry having the lowest LongTermFrameIdx value and proceeding through in ascending order to the reference entry having highest LongTermFrameIdx value.

  NOTE 5 – When the complementary field of the current picture is marked "used for long-term reference" it is included into the list refFrameListLongTerm. A reference entry in which only one field is marked as "used for long-term reference" is included into the list refFrameListLongTerm.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList0ShortTerm and refFrameListLongTerm given as input and the output is assigned to RefPicList0.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList1ShortTerm and refFrameListLongTerm given as input and the output is assigned to RefPicList1.

When the reference picture list RefPicList1 has more than one entry and RefPicList1 is identical to the reference picture list RefPicList0, the first two entries RefPicList1[0] and RefPicList1[1] are switched.

8.2.4.2.5 Initialisation process for reference picture lists in fields

Inputs of this process are the reference frame lists refFrameListXShortTerm (with X may be 0 or 1) and refFrameListLongTerm.

The reference picture list RefPicListX is a list ordered such that short-term reference fields have lower indices than long-term reference fields. Given the reference frame lists refFrameListXShortTerm and refFrameListLongTerm, it is derived as follows.

– Short-term reference fields are ordered by selecting reference fields from the ordered list of frames refFrameListXShortTerm by alternating between fields of differing parity, starting with a field that has the same parity as the current field (when present). When one field of a reference frame was not decoded or is not marked as "used for short-term reference", the missing field is ignored and instead the next available stored reference field of the chosen parity from the ordered list of frames refFrameListXShortTerm is inserted into RefPicListX. When there are no more short-term reference fields of the alternate parity in the ordered list of frames refFrameListXShortTerm, the next not yet indexed fields of the available parity are inserted into RefPicListX in the order in which they occur in the ordered list of frames refFrameListXShortTerm.

– Long-term reference fields are ordered by selecting reference fields from the ordered list of frames refFrameListLongTerm by alternating between fields of differing parity, starting with a field that has the same parity as the current field (when present). When one field of a reference frame was not decoded or is not marked as "used for long-term reference", the missing field is ignored and instead the next available stored reference field of the chosen parity from the ordered list of frames refFrameListLongTerm is inserted into RefPicListX. When there are no more long-term reference fields of the alternate parity in the ordered list of frames refFrameListLongTerm, the next not yet indexed fields of the available parity are inserted into RefPicListX in the order in which they occur in the ordered list of frames refFrameListLongTerm.

8.2.4.3 Reordering process for reference picture lists

When ref_pic_list_reordering_flag_l0 is equal to 1, the following applies.

– Let refIdxL0 be an index into the reference picture list RefPicList0. It is initially set equal to 0.

– The corresponding syntax elements reordering_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies.

– If reordering_of_pic_nums_idc is equal to 0 or equal to 1, the process specified in subclause 8.2.4.3.1 is invoked with refIdxL0 as input, and the output is assigned to refIdxL0.

– Otherwise, if reordering_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.4.3.2 is invoked with refIdxL0 as input, and the output is assigned to refIdxL0.

– Otherwise (reordering_of_pic_nums_idc is equal to 3), the reordering process for reference picture list RefPicList0 is finished.

When ref_pic_list_reordering_flag_l1 is equal to 1, the following applies.

– Let refIdxL1 be an index into the reference picture list RefPicList1. It is initially set equal to 0.

– The corresponding syntax elements reordering_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies.

  – If reordering_of_pic_nums_idc is equal to 0 or equal to 1, the process specified in subclause 8.2.4.3.1 is invoked with refIdxL1 as input, and the output is assigned to refIdxL1.

  – Otherwise, if reordering_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.4.3.2 is invoked with refIdxL1 as input, and the output is assigned to refIdxL1.

  – Otherwise (reordering_of_pic_nums_idc is equal to 3), the reordering process for reference picture list RefPicList1 is finished.

8.2.4.3.1 Reordering process of reference picture lists for short-term reference pictures

Input to this process is an index refIdxLX (with X being 0 or 1).

Output of this process is an incremented index refIdxLX.

The variable picNumLXNoWrap is derived as follows.

– If reordering_of_pic_nums_idc is equal to 0

```
if( picNumLXPred – ( abs_diff_pic_num_minus1 + 1 ) < 0 )
    picNumLXNoWrap = picNumLXPred – ( abs_diff_pic_num_minus1 + 1 ) + MaxPicNum      (8-35)
else
    picNumLXNoWrap = picNumLXPred – ( abs_diff_pic_num_minus1 + 1 )
```

– Otherwise (reordering_of_pic_nums_idc is equal to 1),

```
if( picNumLXPred + ( abs_diff_pic_num_minus1 + 1 ) >= MaxPicNum )
    picNumLXNoWrap = picNumLXPred + ( abs_diff_pic_num_minus1 + 1 ) – MaxPicNum      (8-36)
else
    picNumLXNoWrap = picNumLXPred + ( abs_diff_pic_num_minus1 + 1 )
```

picNumLXPred is the prediction value for the variable picNumLXNoWrap. When the process specified in this subclause is invoked the first time for a slice (that is, for the first occurrence of reordering_of_pic_nums_idc equal to 0 or 1 in the ref_pic_list_reordering( ) syntax), picNumL0Pred and picNumL1Pred are initially set equal to CurrPicNum. After each assignment of picNumLXNoWrap, the value of picNumLXNoWrap is assigned to picNumLXPred.

The variable picNumLX is derived as follows

```
if( picNumLXNoWrap > CurrPicNum )
    picNumLX = picNumLXNoWrap – MaxPicNum                                          (8-37)
else
    picNumLX = picNumLXNoWrap
```

picNumLX shall be equal to the PicNum of a reference picture that is marked as “used for short-term reference” and shall not be equal to the PicNum of a short-term reference picture that is marked as "non-existing".

The following procedure is conducted to place the picture with short-term picture number picNumLX into the index position refIdxLX, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxLX.

```
for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx-- )
    RefPicListX[ cIdx ] = RefPicListX[ cIdx – 1]
RefPicListX[ refIdxLX++ ] = short-term reference picture with PicNum equal to picNumLX
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )                    (8-38)
    if( PicNumF( RefPicListX[ cIdx ] ) != picNumLX )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

where the function PicNumF(RefPicListX[cIdx]) is derived as follows:

– If the picture RefPicListX[cIdx] is marked as "used for short-term reference", PicNumF(RefPicListX[cIdx]) is the PicNum of the picture RefPicListX[cIdx].

– Otherwise (the picture RefPicListX[cIdx] is not marked as "used for short-term reference"), PicNumF(RefPicListX[cIdx]) is equal to MaxPicNum.

NOTE 1 – A value of MaxPicNum can never be equal to picNumLX.

NOTE 2 – Within this pseudo-code procedure, the length of the list RefPicListX is temporarily made one element longer than the length needed for the final list. After the execution of this procedure, only elements 0 through num_ref_idx_lX_active_minus1 of the list need to be retained.

8.2.4.3.2 Reordering process of reference picture lists for long-term reference pictures

Input to this process is an index refIdxLX (with X being 0 or 1).

Output of this process is an incremented index refIdxLX.

The following procedure is conducted to place the picture with long-term picture number long_term_pic_num into the index position refIdxLX, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxLX.

```
for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx-- )
    RefPicListX[ cIdx ] = RefPicListX[ cIdx – 1]
RefPicListX[ refIdxLX++ ] = long-term reference picture with LongTermPicNum equal to long_term_pic_num
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )                    (8-39)
    if( LongTermPicNumF( RefPicListX[ cIdx ] ) != long_term_pic_num )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

where the function LongTermPicNumF(RefPicListX[cIdx]) is derived as follows:

– If the picture RefPicListX[cIdx] is marked as "used for long-term reference", LongTermPicNumF(RefPicListX[cIdx]) is the LongTermPicNum of the picture RefPicListX[cIdx].

– Otherwise (the picture RefPicListX[cIdx] is not marked as "used for long-term reference"), LongTermPicNumF(RefPicListX[cIdx]) is equal to 2 * (MaxLongTermFrameIdx + 1).

NOTE 1 – A value of 2 * (MaxLongTermFrameIdx + 1) can never be equal to long_term_pic_num.

NOTE 2 – Within this pseudo-code procedure, the length of the list RefPicListX is temporarily made one element longer than the length needed for the final list. After the execution of this procedure, only elements 0 through num_ref_idx_lX_active_minus1 of the list need to be retained.

8.2.5 Decoded reference picture marking process

This process is invoked for decoded pictures when nal_ref_idc is not equal to 0.

NOTE – The decoding process for gaps in frame_num that is specified in subclause 8.2.5.2 may also be invoked when nal_ref_idc is equal to 0, as specified in clause 8.

A decoded picture with nal_ref_idc not equal to 0, referred to as a reference picture, is marked as “used for short-term reference” or "used for long-term reference". For a decoded reference frame, both of its fields are marked the same as the frame. For a complementary reference field pair, the pair is marked the same as both of its fields. A picture that is marked as "used for short-term reference" is identified by its FrameNum and, when it is a field, by its parity. A picture that is marked as "used for long-term reference" is identified by its LongTermFrameIdx and, when it is a field, by its parity.

Frames or complementary field pairs marked as “used for short-term reference” or as "used for long-term reference" can be used as a reference for inter prediction when decoding a frame until the frame, the complementary field pair, or one of its constituent fields is marked as “unused for reference”. A field marked as “used for short-term reference” or as

"used for long-term reference" can be used as a reference for inter prediction when decoding a field until marked as "unused for reference".

A picture can be marked as "unused for reference" by the sliding window reference picture marking process, a first-in, first-out mechanism specified in subclause 8.2.5.3 or by the adaptive memory control reference picture marking process, a customised adaptive marking operation specified in subclause 8.2.5.4.

A short-term reference picture is identified for use in the decoding process by its variables FrameNum and FrameNumWrap and its picture number PicNum, and a long-term reference picture is identified for use in the decoding process by its long-term picture number LongTermPicNum. When the current picture is not an IDR picture, subclause 8.2.4.1 is invoked to specify the assignment of the variables FrameNum, FrameNumWrap, PicNum and LongTermPicNum.

8.2.5.1 Sequence of operations for decoded reference picture marking process

Decoded reference picture marking proceeds in the following ordered steps.

1. All slices of the current picture are decoded.
2. Depending on whether the current picture is an IDR picture, the following applies.
   - If the current picture is an IDR picture, the following applies.
     - All reference pictures are marked as "unused for reference".
     - Depending on long_term_reference_flag, the following applies.
       - If long_term_reference_flag is equal to 0, the IDR picture is marked as "used for short-term reference" and MaxLongTermFrameIdx is set equal to "no long-term frame indices".
       - Otherwise (long_term_reference_flag is equal to 1), the IDR picture is marked as "used for long-term reference", the LongTermFrameIdx for the IDR picture is set equal to 0, and MaxLongTermFrameIdx is set equal to 0.
   - Otherwise (the current picture is not an IDR picture), the following applies.
     - If adaptive_ref_pic_marking_mode_flag is equal to 0, the process specified in subclause 8.2.5.3 is invoked.
     - Otherwise (adaptive_ref_pic_marking_mode_flag is equal to 1), the process specified in subclause 8.2.5.4 is invoked.
3. When the current picture is not an IDR picture and it was not marked as "used for long-term reference" by memory_management_control_operation equal to 6, it is marked as "used for short-term reference".

After marking the current decoded reference picture, the total number of frames with at least one field marked as "used for reference", plus the number of complementary field pairs with at least one field marked as "used for reference", plus the number of non-paired fields marked as "used for reference" shall not be greater than Max(num_ref_frames, 1).

8.2.5.2 Decoding process for gaps in frame_num

This process is invoked when frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum.

NOTE 1 – Although this process is specified as a subclause within subclause 8.2.5 (which defines a process that is invoked only when nal_ref_idc is not equal to 0), this process may also be invoked when nal_ref_idc is equal to 0 (as specified in clause 8). The reasons for the location of this subclause within the structure of this Recommendation | International Standard are historical.

NOTE 2 – This process can only be invoked for a conforming bitstream when gaps_in_frame_num_value_allowed_flag is equal to 1. When gaps_in_frame_num_value_allowed_flag is equal to 0 and frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process should infer an unintentional loss of pictures.

When this process is invoked, a set of values of frame_num pertaining to "non-existing" pictures is derived as all values taken on by UnusedShortTermFrameNum in Equation 7-22 except the value of frame_num for the current picture.

The decoding process generates and marks a frame for each of the values of frame_num pertaining to "non-existing" pictures, in the order in which the values of UnusedShortTermFrameNum are generated by Equation 7-22, using the "sliding window" picture marking process as specified in subclause 8.2.5.3. The generated frames are also marked as "non-existing" and "used for short-term reference". The sample values of the generated frames may be set to any value. The bitstream shall not contain data that results in a reference to these generated frames which are marked as "non-existing" in the inter prediction process, a reference to these frames in the reordering commands for reference picture lists for short-term reference pictures (subclause 8.2.4.3.1), or a reference to these frames in the assignment process of a LongTermFrameIdx to a short-term reference picture (subclause 8.2.5.4.3).

When pic_order_cnt_type is not equal to 0, TopFieldOrderCnt and BottomFieldOrderCnt are derived for each of the "non-existing" frames by invoking the decoding process for picture order count in subclause 8.2.1. When invoking the process in subclause 8.2.1 for a particular "non-existing" frame, the current picture is considered to be a picture considered having frame_num inferred to be equal to UnusedShortTermFrameNum, nal_ref_idc inferred to be not equal to 0, nal_unit_type inferred to be not equal to 5, IdrPicFlag inferred to be equal to 0, field_pic_flag inferred to be equal to 0, adaptive_ref_pic_marking_mode_flag inferred to be equal to 0, delta_pic_order_cnt[0] (if needed) inferred to be equal to 0, and delta_pic_order_cnt[1] (if needed) inferred to be equal to 0.

NOTE 3 – The decoding process should infer an unintentional picture loss when any of these values of frame_num pertaining to "non-existing" pictures is referred to in the inter prediction process, is referred to in the reordering commands for reference picture lists for short-term reference pictures (subclause 8.2.4.3.1), or is referred to in the assignment process of a LongTermFrameIdx to a short-term reference picture (subclause 8.2.5.4.3). The decoding process should not infer an unintentional picture loss when a memory management control operation not equal to 3 is applied to a frame marked as "non-existing".

8.2.5.3 Sliding window decoded reference picture marking process

This process is invoked when adaptive_ref_pic_marking_mode_flag is equal to 0.

Depending on the properties of the current picture as specified below, the following applies.

– If the current picture is a coded field that is the second field in decoding order of a complementary reference field pair, and the first field has been marked as "used for short-term reference", the current picture is also marked as "used for short-term reference".

– Otherwise, the following applies.

  – Let numShortTerm be the total number of reference frames, complementary reference field pairs and non-paired reference fields for which at least one field is marked as "used for short-term reference". Let numLongTerm be the total number of reference frames, complementary reference field pairs and non-paired reference fields for which at least one field is marked as "used for long-term reference".

  – When numShortTerm + numLongTerm is equal to Max(num_ref_frames, 1), the condition that numShortTerm is greater than 0 shall be fulfilled, and the short-term reference frame, complementary reference field pair or non-paired reference field that has the smallest value of FrameNumWrap is marked as "unused for reference". When it is a frame or a complementary field pair, both of its fields are also marked as "unused for reference".

8.2.5.4 Adaptive memory control decoded reference picture marking process

This process is invoked when adaptive_ref_pic_marking_mode_flag is equal to 1.

The memory_management_control_operation commands with values of 1 to 6 are processed in the order they occur in the bitstream after the current picture has been decoded. For each of these memory_management_control_operation commands, one of the processes specified in subclauses 8.2.5.4.1 to 8.2.5.4.5 is invoked depending on the value of memory_management_control_operation. The memory_management_control_operation command with value of 0 specifies the end of memory_management_control_operation commands.

Memory management control operations are applied to pictures as follows.

– If field_pic_flag is equal to 0, memory_management_control_operation commands are applied to the frames or complementary reference field pairs specified.

– Otherwise (field_pic_flag is equal to 1), memory_management_control_operation commands are applied to the individual reference fields specified.

8.2.5.4.1 Marking process of a short-term reference picture as "unused for reference"

This process is invoked when memory_management_control_operation is equal to 1.

Let picNumX be specified by

$$picNumX = CurrPicNum - (difference\_of\_pic\_nums\_minus1 + 1). \quad (8\text{-}40)$$

Depending on field_pic_flag the value of picNumX is used to mark a short-term reference picture as "unused for reference" as follows.

– If field_pic_flag is equal to 0, the short-term reference frame or short-term complementary reference field pair specified by picNumX and both of its fields are marked as "unused for reference".

– Otherwise (field_pic_flag is equal to 1), the short-term reference field specified by picNumX is marked as "unused for reference". When that reference field is part of a reference frame or a complementary reference field pair, the

frame or complementary field pair is also marked as "unused for reference", but the marking of the other field is not changed.

8.2.5.4.2 Marking process of a long-term reference picture as "unused for reference"

This process is invoked when memory_management_control_operation is equal to 2.

Depending on field_pic_flag the value of LongTermPicNum is used to mark a long-term reference picture as "unused for reference" as follows.

– If field_pic_flag is equal to 0, the long-term reference frame or long-term complementary reference field pair having LongTermPicNum equal to long_term_pic_num and both of its fields are marked as "unused for reference".

– Otherwise (field_pic_flag is equal to 1), the long-term reference field specified by LongTermPicNum equal to long_term_pic_num is marked as "unused for reference". When that reference field is part of a reference frame or a complementary reference field pair, the frame or complementary field pair is also marked as "unused for reference", but the marking of the other field is not changed.

8.2.5.4.3 Assignment process of a LongTermFrameIdx to a short-term reference picture

This process is invoked when memory_management_control_operation is equal to 3.

Given the syntax element difference_of_pic_nums_minus1, the variable picNumX is obtained as specified in subclause 8.2.5.4.1. picNumX shall refer to a frame or complementary reference field pair or non-paired reference field marked as "used for short-term reference" and not marked as "non-existing".

When LongTermFrameIdx equal to long_term_frame_idx is already assigned to a long-term reference frame or a long-term complementary reference field pair, that frame or complementary field pair and both of its fields are marked as "unused for reference". When LongTermFrameIdx is already assigned to a non-paired reference field, and the field is not the complementary field of the picture specified by picNumX, that field is marked as "unused for reference".

Depending on field_pic_flag the value of LongTermFrameIdx is used to mark a picture from "used for short-term reference" to "used for long-term reference" as follows.

– If field_pic_flag is equal to 0, the marking of the short-term reference frame or short-term complementary reference field pair specified by picNumX and both of its fields are changed from "used for short-term reference" to "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

– Otherwise (field_pic_flag is equal to 1), the marking of the short-term reference field specified by picNumX is changed from "used for short-term reference" to "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx. When the field is part of a reference frame or a complementary reference field pair, and the other field of the same reference frame or complementary reference field pair is also marked as "used for long-term reference", the reference frame or complementary reference field pair is also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

8.2.5.4.4 Decoding process for MaxLongTermFrameIdx

This process is invoked when memory_management_control_operation is equal to 4.

All pictures for which LongTermFrameIdx is greater than max_long_term_frame_idx_plus1 – 1 and that are marked as "used for long-term reference" are marked as "unused for reference".

The variable MaxLongTermFrameIdx is derived as follows.

– If max_long_term_frame_idx_plus1 is equal to 0, MaxLongTermFrameIdx is set equal to "no long-term frame indices".

– Otherwise (max_long_term_frame_idx_plus1 is greater than 0), MaxLongTermFrameIdx is set equal to max_long_term_frame_idx_plus1 – 1.

NOTE – The memory_management_control_operation command equal to 4 can be used to mark long-term reference pictures as "unused for reference". The frequency of transmitting max_long_term_frame_idx_plus1 is not specified by this Recommendation | International Standard. However, the encoder should send a memory_management_control_operation command equal to 4 upon receiving an error message, such as an intra refresh request message.

8.2.5.4.4.1 Marking process of all reference pictures as "unused for reference" and setting MaxLongTermFrameIdx to "no long-term frame indices"

This process is invoked when memory_management_control_operation is equal to 5.

All reference pictures are marked as “unused for reference” and the variable MaxLongTermFrameIdx is set equal to “no long-term frame indices”.

8.2.5.4.5 Process for assigning a long-term frame index to the current picture

This process is invoked when memory_management_control_operation is equal to 6.

When a variable LongTermFrameIdx equal to long_term_frame_idx is already assigned to a long-term reference frame or a long-term complementary reference field pair, that frame or complementary field pair and both of its fields are marked as "unused for reference". When LongTermFrameIdx is already assigned to a non-paired reference field, and the field is not the complementary field of the current picture, that field is marked as “unused for reference”.

The current picture is marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

When field_pic_flag is equal to 0, both its fields are also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

When field_pic_flag is equal to 1 and the current picture is the second field (in decoding order) of a complementary reference field pair, and the first field of the complementary reference field pair is also currently marked as "used for long-term reference), the complementary reference field pair is also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

After marking the current decoded reference picture, the total number of frames with at least one field marked as “used for reference”, plus the number of complementary field pairs with at least one field marked as “used for reference”, plus the number of non-paired fields marked as “used for reference” shall not be greater than Max(num_ref_frames, 1).

NOTE – Under some circumstances, the above statement may impose a constraint on the order in which a memory_management_control_operation syntax element equal to 6 can appear in the decoded reference picture marking syntax relative to a memory_management_control_operation syntax element equal to 1, 2, or 4.

8.3 Intra prediction process

This process is invoked for I and SI macroblock types.

Inputs to this process are constructed samples prior to the deblocking filter process and, for Intra_NxN prediction modes (where NxN is equal to 4x4 or 8x8), the values of IntraNxNPredMode from neighbouring macroblocks.

Outputs of this process are specified as follows.

- If the macroblock prediction mode is Intra_4x4 or Intra_8x8, the outputs are constructed luma samples prior to the deblocking filter process and (when ChromaArrayType is not equal to 0) chroma prediction samples of the macroblock $pred_C$, where C is equal to Cb and Cr.
- Otherwise, if mb_type is not equal to I_PCM, the outputs are luma prediction samples of the macroblock $pred_L$ and (when ChromaArrayType is not equal to 0) chroma prediction samples of the macroblock $pred_C$, where C is equal to Cb and Cr.
- Otherwise (mb_type is equal to I_PCM), the outputs are constructed luma and (when ChromaArrayType is not equal to 0) chroma samples prior to the deblocking filter process.

The variable MvCnt is set equal to 0.

Depending on the value of mb_type the following applies.

- If mb_type is equal to I_PCM, the sample construction process for I_PCM macroblocks as specified in subclause 8.3.5 is invoked.
- Otherwise (mb_type is not equal to I_PCM), the following applies.
  - The decoding processes for Intra prediction modes are described for the luma component as follows.
    - If the macroblock prediction mode is equal to Intra_4x4, the Intra_4x4 prediction process for luma samples as specified in subclause 8.3.1 is invoked.
    - Otherwise, if the macroblock prediction mode is equal to Intra_8x8, the Intra_8x8 prediction process as specified in subclause 8.3.2 is invoked.
    - Otherwise (the macroblock prediction mode is equal to Intra_16x16), the Intra_16x16 prediction process as specified in subclause 8.3.3 is invoked with S'_L as the input and the outputs are luma prediction samples of the macroblock $pred_L$.

– When ChromaArrayType is not equal to 0, the Intra prediction process for chroma samples as specified in subclause 8.3.4 is invoked with S'_{Cb}, and S'_{Cr} as the inputs and the outputs are chroma prediction samples of the macroblock $pred_{Cb}$ and $pred_{Cr}$.

Samples used in the Intra prediction process are the sample values prior to alteration by any deblocking filter operation.

8.3.1 Intra_4x4 prediction process for luma samples

This process is invoked when the macroblock prediction mode is equal to Intra_4x4.

Inputs to this process are the values of Intra4x4PredMode (if available) or Intra8x8PredMode (if available) from neighbouring macroblocks or macroblock pairs.

The luma component of a macroblock consists of 16 blocks of 4x4 luma samples. These blocks are inverse scanned using the 4x4 luma block inverse scanning process as specified in subclause 6.4.3.

For all 4x4 luma blocks of the luma component of a macroblock with luma4x4BlkIdx = 0..15, the derivation process for the Intra4x4PredMode as specified in subclause 8.3.1.1 is invoked with luma4x4BlkIdx as well as Intra4x4PredMode and Intra8x8PredMode that are previously (in decoding order) derived for adjacent macroblocks as the input and the variable Intra4x4PredMode[luma4x4BlkIdx] as the output.

For each luma block of 4x4 samples indexed using luma4x4BlkIdx = 0..15,

1. The Intra_4x4 sample prediction process in subclause 8.3.1.2 is invoked with luma4x4BlkIdx and the array S'_L containing constructed luma samples prior to the deblocking filter process from adjacent luma blocks as the inputs and the outputs are the Intra_4x4 luma prediction samples $pred4x4_L[x, y]$ with x, y = 0..3.

2. The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO) and x, y = 0..3.

$$pred_L[xO + x, yO + y] = pred4x4_L[x, y] \quad (8\text{-}41)$$

3. The transform coefficient decoding process and picture construction process prior to deblocking filter process in subclause 8.5 is invoked with $pred_L$ and luma4x4BlkIdx as the input and the constructed samples for the current 4x4 luma block S'_L as the output.

8.3.1.1 Derivation process for the Intra4x4PredMode

Inputs to this process are the index of the 4x4 luma block luma4x4BlkIdx and variable arrays Intra4x4PredMode (if available) and Intra8x8PredMode (if available) that are previously (in decoding order) derived for adjacent macroblocks.

Output of this process is the variable Intra4x4PredMode[luma4x4BlkIdx].

Table 8-2 specifies the values for Intra4x4PredMode[luma4x4BlkIdx] and the associated names.

Table 8-2 – Specification of Intra4x4PredMode[luma4x4BlkIdx] and associated names

Intra4x4PredMode[luma4x4BlkIdx]	Name of Intra4x4PredMode[luma4x4BlkIdx]
0	Intra_4x4_Vertical (prediction mode)
1	Intra_4x4_Horizontal (prediction mode)
2	Intra_4x4_DC (prediction mode)
3	Intra_4x4_Diagonal_Down_Left (prediction mode)
4	Intra_4x4_Diagonal_Down_Right (prediction mode)
5	Intra_4x4_Vertical_Right (prediction mode)
6	Intra_4x4_Horizontal_Down (prediction mode)
7	Intra_4x4_Vertical_Left (prediction mode)
8	Intra_4x4_Horizontal_Up (prediction mode)

Intra4x4PredMode[luma4x4BlkIdx] labelled 0, 1, 3, 4, 5, 6, 7, and 8 represent directions of predictions as illustrated in Figure 8-1.

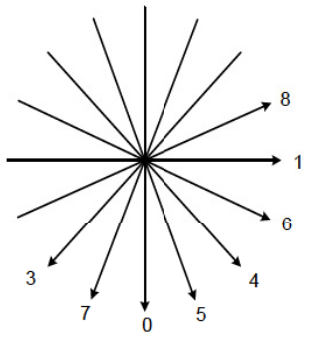


Figure 8-1 – Intra_4x4 prediction mode directions (informative)

Intra4x4PredMode[luma4x4BlkIdx] is derived as follows.

- The process specified in subclause 6.4.10.4 is invoked with luma4x4BlkIdx given as input and the output is assigned to mbAddrA, luma4x4BlkIdxA, mbAddrB, and luma4x4BlkIdxB.
- The variable dcPredModePredictedFlag is derived as follows.
  - If any of the following conditions are true, dcPredModePredictedFlag is set equal to 1
    - the macroblock with address mbAddrA is not available
    - the macroblock with address mbAddrB is not available
    - the macroblock with address mbAddrA is available and coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1
    - the macroblock with address mbAddrB is available and coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1
  - Otherwise, dcPredModePredictedFlag is set equal to 0.
- For N being either replaced by A or B, the variables intraMxMPredModeN are derived as follows.
  - If dcPredModePredictedFlag is equal to 1 or the macroblock with address mbAddrN is not coded in Intra_4x4 or Intra_8x8 macroblock prediction mode, intraMxMPredModeN is set equal to 2 (Intra_4x4_DC prediction mode).
  - Otherwise (dcPredModePredictedFlag is equal to 0 and (the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode or the macroblock with address mbAddrN is coded in Intra_8x8 macroblock prediction mode)), the following applies.
    - If the macroblock with address mbAddrN is coded in Intra_4x4 macroblock mode, intraMxMPredModeN is set equal to Intra4x4PredMode[luma4x4BlkIdxN], where Intra4x4PredMode is the variable array assigned to the macroblock mbAddrN.
    - Otherwise (the macroblock with address mbAddrN is coded in Intra_8x8 macroblock mode), intraMxMPredModeN is set equal to Intra8x8PredMode[luma4x4BlkIdxN >> 2], where Intra8x8PredMode is the variable array assigned to the macroblock mbAddrN.

– Intra4x4PredMode[luma4x4BlkIdx] is derived by applying the following procedure.

```
predIntra4x4PredMode = Min( intraMxMPredModeA, intraMxMPredModeB )
if( prev_intra4x4_pred_mode_flag[ luma4x4BlkIdx ] )
    Intra4x4PredMode[ luma4x4BlkIdx ] = predIntra4x4PredMode
else                                                                                    (8-42)
    if( rem_intra4x4_pred_mode[ luma4x4BlkIdx ] < predIntra4x4PredMode )
        Intra4x4PredMode[ luma4x4BlkIdx ] = rem_intra4x4_pred_mode[ luma4x4BlkIdx ]
    else
        Intra4x4PredMode[ luma4x4BlkIdx ] = rem_intra4x4_pred_mode[ luma4x4BlkIdx ] + 1
```

8.3.1.2 Intra_4x4 sample prediction

This process is invoked for each 4x4 luma block of a macroblock with prediction mode equal to Intra_4x4 followed by the transform decoding process and picture construction process prior to deblocking for each 4x4 luma block.

Inputs to this process are

– the index of a 4x4 luma block luma4x4BlkIdx,

– an ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array cS_L containing constructed luma samples prior to the deblocking filter process of neighbouring macroblocks.

Output of this process are the prediction samples $pred4x4_L$[x, y], with x, y = 0..3 for the 4x4 luma block with index luma4x4BlkIdx.

The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

The 13 neighbouring samples p[x, y] that are constructed luma samples prior to the deblocking filter process, with x = -1, y = -1..3 and x = 0..7, y = -1, are derived as follows.

– The luma location (xN, yN) is specified by

$$xN = xO + x \tag{8-43}$$

$$yN = yO + y \tag{8-44}$$

– The derivation process for neighbouring locations in subclause 6.4.11 is invoked for luma locations with (xN, yN) as input and mbAddrN and (xW, yW) as output.

– Each sample p[x, y] with x = -1, y = -1..3 and x = 0..7, y = -1 is derived as follows.

  – If any of the following conditions is true, the sample p[x, y] is marked as "not available for Intra_4x4 prediction"

    – mbAddrN is not available,

    – the macroblock mbAddrN is coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1,

    – the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1 and the current macroblock does not have mb_type equal to SI,

    – x is greater than 3 and luma4x4BlkIdx is equal to 3 or 11.

  – Otherwise, the sample p[x, y] is marked as "available for Intra_4x4 prediction" and the value of the sample p[x, y] is derived as follows.

    – The location of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as the input and the output is assigned to (xM, yM).

    – Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows.

– If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

$$p[x, y] = cS_L[xM + xW, yM + 2 * yW] \quad (8\text{-}45)$$

– Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

$$p[x, y] = cS_L[xM + xW, yM + yW] \quad (8\text{-}46)$$

When samples p[x, -1], with x = 4..7 are marked as "not available for Intra_4x4 prediction," and the sample p[3, -1] is marked as "available for Intra_4x4 prediction," the sample value of p[3, -1] is substituted for sample values p[x, -1], with x = 4..7 and samples p[x, -1], with x = 4..7 are marked as "available for Intra_4x4 prediction".

NOTE – Each block is assumed to be constructed into a picture array prior to decoding of the next block.

Depending on Intra4x4PredMode[luma4x4BlkIdx], one of the Intra_4x4 prediction modes specified in subclauses 8.3.1.2.1 to 8.3.1.2.9 is invoked.

8.3.1.2.1 Specification of Intra_4x4_Vertical prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 0.

This mode shall be used only when the samples p[x, -1] with x = 0..3 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived by

$$pred4x4_L[x, y] = p[x, -1], \text{ with } x, y = 0..3 \quad (8\text{-}47)$$

8.3.1.2.2 Specification of Intra_4x4_Horizontal prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 1.

This mode shall be used only when the samples p[-1, y], with y = 0..3 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived by

$$pred4x4_L[x, y] = p[-1, y], \text{ with } x,y = 0..3 \quad (8\text{-}48)$$

8.3.1.2.3 Specification of Intra_4x4_DC prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 2.

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows.

– If all samples p[x, -1], with x = 0..3 and p[-1, y], with y = 0..3 are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived by

$$pred4x4_L[x, y] = (p[0, -1] + p[1, -1] + p[2, -1] + p[3, -1] + p[-1, 0] + p[-1, 1] + p[-1, 2] + p[-1, 3] + 4) >> 3 \quad (8\text{-}49)$$

– Otherwise, if any samples p[x, -1], with x = 0..3 are marked as "not available for Intra_4x4 prediction" and all samples p[-1, y], with y = 0..3 are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived by

$$pred4x4_L[x, y] = (p[-1, 0] + p[-1, 1] + p[-1, 2] + p[-1, 3] + 2) >> 2 \quad (8\text{-}50)$$

– Otherwise, if any samples p[-1, y], with y = 0..3 are marked as "not available for Intra_4x4 prediction" and all samples p[x, -1], with x = 0 .. 3 are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0 .. 3 are derived by

$$pred4x4_L[x, y] = (p[0, -1] + p[1, -1] + p[2, -1] + p[3, -1] + 2) >> 2 \quad (8\text{-}51)$$

– Otherwise (some samples p[x, -1], with x = 0..3 and some samples p[-1, y], with y = 0..3 are marked as "not available for Intra_4x4 prediction"), the values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived by

$$pred4x4_L[x, y] = (1 << (BitDepth_Y - 1)) \quad (8\text{-}52)$$

NOTE – A 4x4 luma block can always be predicted using this mode.

8.3.1.2.4 Specification of Intra_4x4_Diagonal_Down_Left prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 3.

This mode shall be used only when the samples p[x, -1] with x = 0..7 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows.

– If x is equal to 3 and y is equal to 3,

$$pred4x4_L[x, y] = (p[6, -1] + 3 * p[7, -1] + 2) >> 2 \quad (8\text{-}53)$$

– Otherwise (x is not equal to 3 or y is not equal to 3),

$$pred4x4_L[x, y] = (p[x + y, -1] + 2 * p[x + y + 1, -1] + p[x + y + 2, -1] + 2) >> 2 \quad (8\text{-}54)$$

8.3.1.2.5 Specification of Intra_4x4_Diagonal_Down_Right prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 4.

This mode shall be used only when the samples p[x, -1] with x = 0..3 and p[-1, y] with y = -1..3 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows.

– If x is greater than y,

$$pred4x4_L[x, y] = (p[x - y - 2, -1] + 2 * p[x - y - 1, -1] + p[x - y, -1] + 2) >> 2 \quad (8\text{-}55)$$

– Otherwise, if x is less than y,

$$pred4x4_L[x, y] = (p[-1, y - x - 2] + 2 * p[-1, y - x - 1] + p[-1, y - x] + 2) >> 2 \quad (8\text{-}56)$$

– Otherwise (x is equal to y),

$$pred4x4_L[x, y] = (p[0, -1] + 2 * p[-1, -1] + p[-1, 0] + 2) >> 2 \quad (8\text{-}57)$$

8.3.1.2.6 Specification of Intra_4x4_Vertical_Right prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 5.

This mode shall be used only when the samples p[x, -1] with x = 0..3 and p[-1, y] with y = -1..3 are marked as "available for Intra_4x4 prediction".

Let the variable zVR be set equal to 2 * x – y.

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows.

– If zVR is equal to 0, 2, 4, or 6,

$$pred4x4_L[x, y] = (p[x - (y >> 1) - 1, -1] + p[x - (y >> 1), -1] + 1) >> 1 \quad (8\text{-}58)$$

– Otherwise, if zVR is equal to 1, 3, or 5,

$$pred4x4_L[x, y] = (p[x - (y >> 1) - 2, -1] + 2 * p[x - (y >> 1) - 1, -1] + p[x - (y >> 1), -1] + 2) >> 2 \quad (8\text{-}59)$$

– Otherwise, if zVR is equal to -1,

$$pred4x4_L[x, y] = (p[-1, 0] + 2 * p[-1, -1] + p[0, -1] + 2) >> 2 \quad (8\text{-}60)$$

– Otherwise (zVR is equal to -2 or -3),

$$pred4x4_L[x, y] = (p[-1, y - 1] + 2 * p[-1, y - 2] + p[-1, y - 3] + 2) >> 2 \quad (8\text{-}61)$$

8.3.1.2.7 Specification of Intra_4x4_Horizontal_Down prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 6.

This mode shall be used only when the samples p[x, -1] with x = 0..3 and p[-1, y] with y = -1..3 are marked as "available for Intra_4x4 prediction".

Let the variable zHD be set equal to 2 * y – x.

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows.

– If zHD is equal to 0, 2, 4, or 6,

$$pred4x4_L[x, y] = (p[-1, y -(x >> 1) - 1] + p[-1, y - (x >> 1)] + 1) >> 1 \quad (8\text{-}62)$$

– Otherwise, if zHD is equal to 1, 3, or 5,

$$pred4x4_L[x, y] = (p[-1, y - (x >> 1) - 2] + 2 * p[-1, y - (x >> 1) - 1] + p[-1, y - (x >> 1)] + 2) >> 2 \quad (8\text{-}63)$$

– Otherwise, if zHD is equal to -1,

$$pred4x4_L[x, y] = (p[-1, 0] + 2 * p[-1, -1] + p[0, -1] + 2) >> 2 \quad (8\text{-}64)$$

– Otherwise (zHD is equal to -2 or -3),

$$pred4x4_L[x, y] = (p[x - 1, -1] + 2 * p[x - 2, -1] + p[x - 3, -1] + 2) >> 2 \quad (8\text{-}65)$$

8.3.1.2.8 Specification of Intra_4x4_Vertical_Left prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 7.

This mode shall be used only when the samples p[x, -1] with x = 0..7 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows.

– If y is equal to 0 or 2,

$$pred4x4_L[x, y] = (p[x + (y >> 1), -1] + p[x + (y >> 1) + 1, -1] + 1) >> 1 \quad (8\text{-}66)$$

– Otherwise (y is equal to 1 or 3),

$$pred4x4_L[x, y] = (p[x + (y >> 1), -1] + 2 * p[x + (y >> 1) + 1, -1] + p[x + (y >> 1) + 2, -1] + 2) >> 2 \quad (8\text{-}67)$$

8.3.1.2.9 Specification of Intra_4x4_Horizontal_Up prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 8.

This mode shall be used only when the samples p[-1, y] with y = 0..3 are marked as "available for Intra_4x4 prediction".

Let the variable zHU be set equal to x + 2 * y.

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows:

– If zHU is equal to 0, 2, or 4

$$pred4x4_L[x, y] = (p[-1, y + (x >> 1)] + p[-1, y + (x >> 1) + 1] + 1) >> 1 \quad (8\text{-}68)$$

– Otherwise, if zHU is equal to 1 or 3

$$pred4x4_L[x, y] = (p[-1, y + (x >> 1)] + 2 * p[-1, y + (x >> 1) + 1] + p[-1, y + (x >> 1) + 2] + 2) >> 2 \quad (8\text{-}69)$$

– Otherwise, if zHU is equal to 5,

$$pred4x4_L[x, y] = (p[-1, 2] + 3 * p[-1, 3] + 2) >> 2 \quad (8\text{-}70)$$

– Otherwise (zHU is greater than 5),

$$pred4x4_L[x, y] = p[-1, 3] \quad (8\text{-}71)$$

8.3.2 Intra_8x8 prediction process for luma samples

This process is invoked when the macroblock prediction mode is equal to Intra_8x8.

Inputs to this process are the values of Intra4x4PredMode (if available) or Intra8x8PredMode (if available) from the neighbouring macroblocks or macroblock pairs.

Outputs of this process are 8x8 luma sample arrays as part of the 16x16 luma array of prediction samples of the macroblock $pred_L$.

The luma component of a macroblock consists of 4 blocks of 8x8 luma samples. These blocks are inverse scanned using the inverse 8x8 luma block scanning process as specified in subclause 6.4.5.

For all 8x8 luma blocks of the luma component of a macroblock with luma8x8BlkIdx = 0..3, the derivation process for Intra8x8PredMode as specified in subclause 8.3.2.1 is invoked with luma8x8BlkIdx as well as Intra4x4PredMode and Intra8x8PredMode that are previously (in decoding order) derived for adjacent macroblocks as the input and the variable Intra8x8PredMode[luma8x8BlkIdx] as the output.

For each luma block of 8x8 samples indexed using luma8x8BlkIdx = 0..3, the following applies.

– The Intra_8x8 sample prediction process in subclause 8.3.2.2 is invoked with luma8x8BlkIdx and the array S'_L containing constructed samples prior to the deblocking filter process from adjacent luma blocks as the input and the output are the Intra_8x8 luma prediction samples $pred8x8_L[x, y]$ with x, y = 0..7.

– The position of the upper-left sample of an 8x8 luma block with index luma8x8BlkIdx inside the current macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.5 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO) and x, y = 0..7.

$$pred_L[xO + x, yO + y] = pred8x8_L[x, y] \quad (8\text{-}72)$$

– The transform coefficient decoding process and picture construction process prior to deblocking filter process in subclause 8.5 is invoked with $pred_r$ and luma8x8BlkIdx as the input and the constructed samples for the current 8x8 luma block S'_L as the output.

8.3.2.1 Derivation process for Intra8x8PredMode

Inputs to this process are the index of the 8x8 luma block luma8x8BlkIdx and variable arrays Intra4x4PredMode (if available) and Intra8x8PredMode (if available) that are previously (in decoding order) derived for adjacent macroblocks.

Output of this process is the variable Intra8x8PredMode[luma8x8BlkIdx].

Table 8-3 specifies the values for Intra8x8PredMode[luma8x8BlkIdx] and the associated mnemonic names.

Table 8-3 – Specification of Intra8x8PredMode[luma8x8BlkIdx] and associated names

Intra8x8PredMode[luma8x8BlkIdx]	Name of Intra8x8PredMode[luma8x8BlkIdx]
0	Intra_8x8_Vertical (prediction mode)
1	Intra_8x8_Horizontal (prediction mode)
2	Intra_8x8_DC (prediction mode)
3	Intra_8x8_Diagonal_Down_Left (prediction mode)
4	Intra_8x8_Diagonal_Down_Right (prediction mode)
5	Intra_8x8_Vertical_Right (prediction mode)
6	Intra_8x8_Horizontal_Down (prediction mode)
7	Intra_8x8_Vertical_Left (prediction mode)
8	Intra_8x8_Horizontal_Up (prediction mode)

Intra8x8PredMode[luma8x8BlkIdx] is derived as follows.

- The process specified in subclause 6.4.10.2 is invoked with luma8x8BlkIdx given as input and the output is assigned to mbAddrA, luma8x8BlkIdxA, mbAddrB, and luma8x8BlkIdxB.
- The variable dcPredModePredictedFlag is derived as follows.
  - If any of the following conditions are true, dcPredModePredictedFlag is set equal to 1
    - the macroblock with address mbAddrA is not available
    - the macroblock with address mbAddrB is not available
    - the macroblock with address mbAddrA is available and coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1
    - the macroblock with address mbAddrB is available and coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1
  - Otherwise, dcPredModePredictedFlag is set equal to 0.
- For N being either replaced by A or B, the variables intraMxMPredModeN are derived as follows.
  - If dcPredModePredictedFlag is equal to 1 or (the macroblock with address mbAddrN is not coded in Intra_4x4 macroblock prediction mode and the macroblock with address mbAddrN is not coded in Intra_8x8 macroblock prediction mode), intraMxMPredModeN is set equal to 2 (Intra_8x8_DC prediction mode).
  - Otherwise (dcPredModePredictedFlag is equal to 0 and (the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode or the macroblock with address mbAddrN is coded in Intra_8x8 macroblock prediction mode)), the following applies.
    - If the macroblock with address mbAddrN is coded in Intra_8x8 macroblock mode, intraMxMPredModeN is set equal to Intra8x8PredMode[luma8x8BlkIdxN], where Intra8x8PredMode is the variable array assigned to the macroblock mbAddrN.
    - Otherwise (the macroblock with address mbAddrN is coded in Intra_4x4 macroblock mode), intraMxMPredModeN is derived by the following procedure, where Intra4x4PredMode is the variable array assigned to the macroblock mbAddrN.

      intraMxMPredModeN = Intra4x4PredMode[luma8x8BlkIdxN * 4 + n] (8-73)

      where the variable n is derived as follows
      - If N is equal to A, depending on the variable MbaffFrameFlag, the variable luma8x8BlkIdx, the current macroblock, and the macroblock mbAddrN, the following applies.

- If MbaffFrameFlag is equal to 1, the current macroblock is a frame coded macroblock, the macroblock mbAddrN is a field coded macroblock, and luma8x8BlkIdx is equal to 2, n is set equal to 3.
- Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a field coded macroblock or the macroblock mbAddrN is a frame coded macroblock or luma8x8BlkIdx is not equal to 2), n is set equal to 1.

- Otherwise (N is equal to B), n is set equal to 2.

– Finally, given intraMxMPredModeA and intraMxMPredModeB, the variable Intra8x8PredMode[luma8x8BlkIdx] is derived by applying the following procedure.

```
predIntra8x8PredMode = Min( intraMxMPredModeA, intraMxMPredModeB )
if( prev_intra8x8_pred_mode_flag[ luma8x8BlkIdx ] )
    Intra8x8PredMode[ luma8x8BlkIdx ] = predIntra8x8PredMode
else                                                                    (8-74)
    if( rem_intra8x8_pred_mode[ luma8x8BlkIdx ] < predIntra8x8PredMode )
        Intra8x8PredMode[ luma8x8BlkIdx ] = rem_intra8x8_pred_mode[ luma8x8BlkIdx ]
    else
        Intra8x8PredMode[ luma8x8BlkIdx ] = rem_intra8x8_pred_mode[ luma8x8BlkIdx ] + 1
```

8.3.2.2 Intra_8x8 sample prediction

This process is invoked for each 8x8 luma block of a macroblock with prediction mode equal to Intra_8x8 followed by the transform decoding process and picture construction process prior to deblocking for each 8x8 luma block.

Inputs to this process are

– the index of an 8x8 luma block luma8x8BlkIdx.

– an (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$) array cS_L containing constructed luma samples prior to the deblocking filter process of neighbouring macroblocks.

Output of this process are the prediction samples pred8x8$_L$[x, y], with x, y = 0..7 for the 8x8 luma block with index luma8x8BlkIdx.

The position of the upper-left sample of an 8x8 luma block with index luma8x8BlkIdx inside the current macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.5 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO).

The 25 neighbouring samples p[x, y] that are constructed luma samples prior to the deblocking filter process, with x = -1, y = -1..7 and x = 0..15, y = -1, are derived as follows.

– The luma location (xN, yN) is specified by

$$xN = xO + x \tag{8-75}$$

$$yN = yO + y \tag{8-76}$$

– The derivation process for neighbouring locations in subclause 6.4.11 is invoked for luma locations with (xN, yN) as input and mbAddrN and (xW, yW) as output.

– Each sample p[x, y] with x = -1, y = -1..7 and x = 0..15, y = -1 is derived as follows.

  – If any of the following conditions is true, the sample p[x, y] is marked as "not available for Intra_8x8 prediction"

    – mbAddrN is not available,

    – the macroblock mbAddrN is coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1,

  – Otherwise, the sample p[x, y] is marked as "available for Intra_8x8 prediction" and the sample value p[x, y] is derived as follows.

    – The location of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as the input and the output is assigned to (xM, yM).

– Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows.

  – If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

$$p[x, y] = cS_L[xM + xW, yM + 2 * yW] \quad (8\text{-}77)$$

  – Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

$$p[x, y] = cS_L[xM + xW, yM + yW] \quad (8\text{-}78)$$

When samples p[x, -1], with x = 8..15 are marked as "not available for Intra_8x8 prediction," and the sample p[7, -1] is marked as "available for Intra_8x8 prediction," the sample value of p[7, -1] is substituted for sample values p[x, -1], with x = 8..15 and samples p[x, -1], with x = 8..15 are marked as "available for Intra_8x8 prediction".

NOTE – Each block is assumed to be constructed into a picture array prior to decoding of the next block.

The reference sample filtering process for Intra_8x8 sample prediction in subclause 8.3.2.2.1 is invoked with the samples p[x, y] with x = -1, y = -1..7 and x = 0..15, y = -1 (if available) as input and p'[x, y] with x = -1, y = -1..7 and x = 0..15, y = -1 as output.

Depending on Intra8x8PredMode[luma8x8BlkIdx], one of the Intra_8x8 prediction modes specified in subclauses 8.3.2.2.2 to 8.3.2.2.10 is invoked.

8.3.2.2.1 Reference sample filtering process for Intra_8x8 sample prediction

Inputs to this process are the reference samples p[x, y] with x = -1, y = -1..7 and x = 0..15, y = -1 (if available) for Intra_8x8 sample prediction.

Outputs of this process are the filtered reference samples p'[x, y] with x = -1, y = -1..7 and x = 0..15, y = -1 for Intra_8x8 sample prediction.

When all samples p[x, -1] with x = 0..7 are marked as "available for Intra_8x8 prediction", the following applies.

– The value of p'[0, -1] is derived as follows.

– If p[-1, -1] is marked as "available for Intra_8x8 prediction", p'[0, -1] is derived by

$$p'[0, -1] = (p[-1, -1] + 2 * p[0, -1] + p[1, -1] + 2) >> 2 \quad (8\text{-}79)$$

– Otherwise (p[-1, -1] is marked as "not available for Intra_8x8 prediction"), p'[0, -1] is derived by

$$p'[0, -1] = (3 * p[0, -1] + p[1, -1] + 2) >> 2 \quad (8\text{-}80)$$

– The values of p'[x, -1], with x = 1..7 are derived by

$$p'[x, -1] = (p[x-1, -1] + 2 * p[x, -1] + p[x+1, -1] + 2) >> 2 \quad (8\text{-}81)$$

When all samples p[x, -1] with x = 7..15 are marked as "available for Intra_8x8 prediction", the following applies.

– The values of p'[x, -1], with x = 8..14 are derived by

$$p'[x, -1] = (p[x-1, -1] + 2 * p[x, -1] + p[x+1, -1] + 2) >> 2 \quad (8\text{-}82)$$

– The value of p'[15, -1] is derived by

$$p'[15, -1] = (p[14, -1] + 3 * p[15, -1] + 2) >> 2 \quad (8\text{-}83)$$

When the sample p[-1, -1] is marked as "available for Intra_8x8 prediction", the value of p'[-1, -1] is derived as follows.

– If the sample p[0, -1] is marked as "not available for Intra_8x8 prediction" or the sample p[-1, 0] is marked as "not available for Intra_8x8 prediction", the following applies.

– If the sample p[0, -1] is marked as "available for Intra_8x8 prediction", p'[-1, -1] is derived by

$$p'[-1, -1] = (3 * p[-1, -1] + p[0, -1] + 2) >> 2 \quad (8\text{-}84)$$

– Otherwise (the sample p[0, -1] is marked as "not available for Intra_8x8 prediction" and the sample p[-1, 0] is marked as "available for Intra_8x8 prediction"), p'[-1, -1] is derived by

$$p'[-1, -1] = (3 * p[-1, -1] + p[-1, 0] + 2) >> 2 \quad (8\text{-}85)$$

– Otherwise (the sample p[0, -1] is marked as "available for Intra_8x8 prediction" and the sample p[-1, 0] is marked as "available for Intra_8x8 prediction"), p'[-1, -1] is derived by

$$p'[-1, -1] = (p[0, -1] + 2 * p[-1, -1] + p[-1, 0] + 2) >> 2 \quad (8\text{-}86)$$

When all samples p[-1, y] with y = 0..7 are marked as "available for Intra_8x8 prediction", the following applies.

– The value of p'[-1, 0] is derived as follows.

– If p[-1, -1] is marked as "available for Intra_8x8 prediction", p'[-1, 0] is derived by

$$p'[-1, 0] = (p[-1, -1] + 2 * p[-1, 0] + p[-1, 1] + 2) >> 2 \quad (8\text{-}87)$$

– Otherwise (if p[-1, -1] is marked as "not available for Intra_8x8 prediction"), p'[-1, 0] is derived by

$$p'[-1, 0] = (3 * p[-1, 0] + p[-1, 1] + 2) >> 2 \quad (8\text{-}88)$$

– The values of p'[-1, y], with y = 1..6 are derived by

$$p'[-1, y] = (p[-1, y-1] + 2 * p[-1, y] + p[-1, y+1] + 2) >> 2 \quad (8\text{-}89)$$

– The value of p'[-1, 7] is derived by

$$p'[-1, 7] = (p[-1, 6] + 3 * p[-1, 7] + 2) >> 2 \quad (8\text{-}90)$$

8.3.2.2.2 Specification of Intra_8x8_Vertical prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 0.

This mode shall be used only when the samples p[x, -1] with x = 0..7 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived by

$$pred8x8_L[x, y] = p'[x, -1], \text{ with } x, y = 0..7 \quad (8\text{-}91)$$

8.3.2.2.3 Specification of Intra_8x8_Horizontal prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 1.

This mode shall be used only when the samples p[-1, y], with y = 0..7 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived by

$$pred8x8_L[x, y] = p'[-1, y], \text{ with } x, y = 0..7 \quad (8\text{-}92)$$

8.3.2.2.4 Specification of Intra_8x8_DC prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 2.

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived as follows.

– If all samples p[x, -1], with x = 0..7 and p[-1, y], with y = 0..7 are marked as "available for Intra_8x8 prediction," the values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived by

$$pred8x8_L[x, y] = (\sum_{x'=0}^{7} p'[x',-1] + \sum_{y'=0}^{7} p'[-1,y'] + 8) >> 4 \quad (8\text{-}93)$$

– Otherwise, if any samples p[x, -1], with x = 0..7 are marked as "not available for Intra_8x8 prediction" and all samples p[-1, y], with y = 0..7 are marked as "available for Intra_8x8 prediction", the values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived by

$$pred8x8_L[x, y] = (\sum_{y'=0}^{7} p'[-1,y'] + 4) >> 3 \quad (8\text{-}94)$$

– Otherwise, if any samples p[-1, y], with y = 0..7 are marked as "not available for Intra_8x8 prediction" and all samples p[x, -1], with x = 0..7 are marked as "available for Intra_8x8 prediction", the values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived by

$$pred8x8_L[x, y] = (\sum_{x'=0}^{7} p'[x',-1] + 4) >> 3 \quad (8\text{-}95)$$

– Otherwise (some samples p[x, -1], with x = 0..7 and some samples p[-1, y], with y = 0..7 are marked as "not available for Intra_8x8 prediction"), the values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived by

$$pred8x8_L[x, y] = (1 << (BitDepth_Y - 1)) \quad (8\text{-}96)$$

NOTE – An 8x8 luma block can always be predicted using this mode.

8.3.2.2.5 Specification of Intra_8x8_Diagonal_Down_Left prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 3.

This mode shall be used only when the samples p[x, -1] with x = 0..15 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived as follows.

– If x is equal to 7 and y is equal to 7,

$$pred8x8_L[x, y] = (p'[14, -1] + 3 * p'[15, -1] + 2) >> 2 \quad (8\text{-}97)$$

– Otherwise (x is not equal to 7 or y is not equal to 7),

$$pred8x8_L[x, y] = (p'[x + y, -1] + 2 * p'[x + y + 1, -1] + p'[x + y + 2, -1] + 2) >> 2 \quad (8\text{-}98)$$

8.3.2.2.6 Specification of Intra_8x8_Diagonal_Down_Right prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 4.

This mode shall be used only when the samples p[x, -1] with x = 0..7 and p[-1, y] with y = -1..7 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived as follows.

– If x is greater than y,

$$pred8x8_L[x, y] = (p'[x - y - 2, -1] + 2 * p'[x - y - 1, -1] + p'[x - y, -1] + 2) >> 2 \quad (8\text{-}99)$$

– Otherwise if x is less than y,

$$pred8x8_L[x, y] = (p'[-1, y - x - 2] + 2 * p'[-1, y - x - 1] + p'[-1, y - x] + 2) >> 2 \quad (8\text{-}100)$$

– Otherwise (x is equal to y),

$$pred8x8_L[x, y] = (p'[0, -1] + 2 * p'[-1, -1] + p'[-1, 0] + 2) >> 2 \quad (8\text{-}101)$$

8.3.2.2.7 Specification of Intra_8x8_Vertical_Right prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 5.

This mode shall be used only when the samples p[x, -1] with x = 0..7 and p[-1, y] with y = -1..7 are marked as "available for Intra_8x8 prediction".

Let the variable zVR be set equal to 2 * x – y.

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived as follows.

– If zVR is equal to 0, 2, 4, 6, 8, 10, 12, or 14

$$pred8x8_L[x, y] = (p'[x - (y >> 1) - 1, -1] + p'[x - (y >> 1), -1] + 1) >> 1 \quad (8\text{-}102)$$

– Otherwise, if zVR is equal to 1, 3, 5, 7, 9, 11, or 13

$$pred8x8_L[x, y] = (p'[x - (y >> 1) - 2, -1] + 2 * p'[x - (y >> 1) - 1, -1] + p'[x - (y >> 1), -1] + 2) >> 2 \quad (8\text{-}103)$$

– Otherwise, if zVR is equal to -1,

$$pred8x8_L[x, y] = (p'[-1, 0] + 2 * p'[-1, -1] + p'[0, -1] + 2) >> 2 \quad (8\text{-}104)$$

– Otherwise (zVR is equal to -2, -3, -4, -5, -6, or -7),

$$pred8x8_L[x, y] = (p'[-1, y - 2*x - 1] + 2 * p'[-1, y - 2*x - 2] + p'[-1, y - 2*x - 3] + 2) >> 2 \quad (8\text{-}105)$$

8.3.2.2.8 Specification of Intra_8x8_Horizontal_Down prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 6.

This mode shall be used only when the samples p[x, -1] with x = 0..7 and p[-1, y] with y = -1..7 are marked as "available for Intra_8x8 prediction".

Let the variable zHD be set equal to 2 * y – x.

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived as follows.

– If zHD is equal to 0, 2, 4, 6, 8, 10, 12, or 14

$$pred8x8_L[x, y] = (p'[-1, y -(x >> 1) - 1] + p'[-1, y - (x >> 1)] + 1) >> 1 \quad (8\text{-}106)$$

– Otherwise, if zHD is equal to 1, 3, 5, 7, 9, 11, or 13

$$pred8x8_L[x, y] = (p'[-1, y - (x >> 1) - 2] + 2 * p'[-1, y - (x >> 1) - 1] + p'[-1, y - (x >> 1)] + 2) >> 2 \quad (8\text{-}107)$$

– Otherwise, if zHD is equal to -1,

$$pred8x8_L[x, y] = (p'[-1, 0] + 2 * p'[-1, -1] + p'[0, -1] + 2) >> 2 \quad (8\text{-}108)$$

– Otherwise (zHD is equal to –2, -3, -4, -5, -6, -7),

$$pred8x8_L[x, y] = (p'[x - 2*y - 1, -1] + 2 * p'[x - 2*y - 2, -1] + p'[x - 2*y - 3, -1] + 2) >> 2 \quad (8\text{-}109)$$

8.3.2.2.9 Specification of Intra_8x8_Vertical_Left prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 7.

This mode shall be used only when the samples p[x, -1] with x = 0..15 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived as follows.

– If y is equal to 0, 2, 4 or 6

$$pred8x8_L[x, y] = (p'[x + (y >> 1), -1] + p'[x + (y >> 1) + 1, -1] + 1) >> 1 \quad (8\text{-}110)$$

– Otherwise (y is equal to 1, 3, 5, 7),

$$pred8x8_L[x, y] = (p'[x + (y >> 1), -1] + 2 * p'[x + (y >> 1) + 1, -1] + p'[x + (y >> 1) + 2, -1] + 2) >> 2 \quad (8\text{-}111)$$

8.3.2.2.10 Specification of Intra_8x8_Horizontal_Up prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 8.

This mode shall be used only when the samples p[-1, y] with y = 0..7 are marked as "available for Intra_8x8 prediction".

Let the variable zHU be set equal to x + 2 * y.

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived as follows:

– If zHU is equal to 0, 2, 4, 6, 8, 10, or 12

$$pred8x8_L[x, y] = (p'[-1, y + (x >> 1)] + p'[-1, y + (x >> 1) + 1] + 1) >> 1 \quad (8\text{-}112)$$

– Otherwise, if zHU is equal to 1, 3, 5, 7, 9, or 11

$$pred8x8_L[x, y] = (p'[-1, y + (x >> 1)] + 2 * p'[-1, y + (x >> 1) + 1] + p'[-1, y + (x >> 1) + 2] + 2) >> 2 \quad (8\text{-}113)$$

– Otherwise, if zHU is equal to 13,

$$pred8x8_L[x, y] = (p'[-1, 6] + 3 * p'[-1, 7] + 2) >> 2 \quad (8\text{-}114)$$

– Otherwise (zHU is greater than 13),

$$pred8x8_L[x, y] = p'[-1, 7] \quad (8\text{-}115)$$

8.3.3 Intra_16x16 prediction process for luma samples

This process is invoked when the macroblock prediction mode is equal to Intra_16x16. It specifies how the Intra prediction luma samples for the current macroblock are derived.

Input to this process is a $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array cS_L containing constructed luma samples prior to the deblocking filter process of neighbouring macroblocks

Outputs of this process are Intra prediction luma samples for the current macroblock $pred_L[x, y]$.

The 33 neighbouring samples p[x, y] that are constructed luma samples prior to the deblocking filter process, with x = -1, y = -1..15 and with x = 0..15, y = -1, are derived as follows.

– The derivation process for neighbouring locations in subclause 6.4.11 is invoked for luma locations with (x, y) assigned to (xN, yN) as input and mbAddrN and (xW, yW) as output.

– Each sample p[x, y] with x = -1, y = -1..15 and with x = 0..15, y = -1 is derived as follows.

  – If any of the following conditions is true, the sample p[x, y] is marked as "not available for Intra_16x16 prediction"

    – mbAddrN is not available,

    – the macroblock mbAddrN is coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1.

    – the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1.

– Otherwise, the sample p[x, y] is marked as "available for Intra_16x16 prediction" and the value of the sample p[x, y] is derived as follows.

  – The location of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as the input and the output is assigned to (xM, yM).

  – Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows.

    – If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

$$p[x, y] = cS_L[xM + xW, yM + 2 * yW] \quad (8\text{-}116)$$

    – Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

$$p[x, y] = cS_L[xM + xW, yM + yW] \quad (8\text{-}117)$$

Let $pred_L[x, y]$ with x, y = 0..15 denote the prediction samples for the 16x16 luma block samples.

Intra_16x16 prediction modes are specified in Table 8-4.

Table 8-4 – Specification of Intra16x16PredMode and associated names

Intra16x16PredMode	Name of Intra16x16PredMode
0	Intra_16x16_Vertical (prediction mode)
1	Intra_16x16_Horizontal (prediction mode)
2	Intra_16x16_DC (prediction mode)
3	Intra_16x16_Plane (prediction mode)

Depending on Intra16x16PredMode, one of the Intra_16x16 prediction modes specified in subclauses 8.3.3.1 to 8.3.3.4 is invoked.

8.3.3.1 Specification of Intra_16x16_Vertical prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[x, -1] with x = 0..15 are marked as "available for Intra_16x16 prediction".

$$pred_L[x, y] = p[x, -1], \text{ with } x, y = 0..15 \quad (8\text{-}118)$$

8.3.3.2 Specification of Intra_16x16_Horizontal prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[-1, y] with y = 0..15 are marked as "available for Intra_16x16 prediction".

$$pred_L[x, y] = p[-1, y], \text{ with } x, y = 0..15 \quad (8\text{-}119)$$

8.3.3.3 Specification of Intra_16x16_DC prediction mode

This Intra_16x16 prediction mode operates, depending on whether the neighbouring samples are marked as "available for Intra_16x16 prediction", as follows.

– If all neighbouring samples p[x, -1], with x = 0..15 and p[-1, y], with y = 0..15 are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[x, y] = \left(\sum_{x'=0}^{15} p[x',-1] + \sum_{y'=0}^{15} p[-1,y'] + 16\right) >> 5, \text{ with } x, y = 0..15 \quad (8\text{-}120)$$

– Otherwise, if any of the neighbouring samples p[x, -1], with x = 0..15 are marked as "not available for Intra_16x16 prediction" and all of the neighbouring samples p[-1, y], with y = 0..15 are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[x, y] = (\sum_{y'=0}^{15} p[-1, y'] + 8) >> 4 \text{, with } x, y = 0..15 \quad (8\text{-}121)$$

– Otherwise, if any of the neighbouring samples p[-1, y], with y = 0..15 are marked as "not available for Intra_16x16 prediction" and all of the neighbouring samples p[x, -1], with x = 0..15 are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[x, y] = (\sum_{x'=0}^{15} p[x', -1] + 8) >> 4 \text{, with } x, y = 0..15 \quad (8\text{-}122)$$

– Otherwise (some of the neighbouring samples p[x, -1], with x = 0..15 and some of the neighbouring samples p[-1, y], with y = 0..15 are marked as "not available for Intra_16x16 prediction"), the prediction for all luma samples in the macroblock is given by:

$$pred_L[x, y] = (1 << (BitDepth_Y - 1)), \text{ with } x, y = 0..15 \quad (8\text{-}123)$$

8.3.3.4 Specification of Intra_16x16_Plane prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[x, -1] with x = -1..15 and p[-1, y] with y = 0..15 are marked as "available for Intra_16x16 prediction".

$$pred_L[x, y] = Clip1_Y((a + b * (x - 7) + c * (y - 7) + 16) >> 5), \text{ with } x, y = 0..15, \quad (8\text{-}124)$$

where:

$$a = 16 * (p[-1, 15] + p[15, -1]) \quad (8\text{-}125)$$

$$b = (5 * H + 32) >> 6 \quad (8\text{-}126)$$

$$c = (5 * V + 32) >> 6 \quad (8\text{-}127)$$

and H and V are specified in Equations 8-128 and 8-129.

$$H = \sum_{x'=0}^{7} (x'+1) * (p[8 + x', -1] - p[6 - x', -1]) \quad (8\text{-}128)$$

$$V = \sum_{y'=0}^{7} (y'+1) * (p[-1, 8 + y'] - p[-1, 6 - y']) \quad (8\text{-}129)$$

8.3.4 Intra prediction process for chroma samples

This process is invoked for I and SI macroblock types. It specifies how the Intra prediction chroma samples for the current macroblock are derived.

Inputs to this process are two $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ arrays cS_{Cb} and cS_{Cr} containing constructed chroma samples prior to the deblocking filter process of neighbouring macroblocks.

Outputs of this process are Intra prediction chroma samples for the current macroblock $pred_{Cb}[x, y]$ and $pred_{Cr}[x, y]$.

Depending on the value of ChromaArrayType, the following applies.

– If ChromaArrayType is equal to 3, the Intra prediction chroma samples for the current macroblock $pred_{Cb}[x, y]$ and $pred_{Cr}[x, y]$ are derived using the Intra prediction process for chroma samples with ChromaArrayType equal to 3 as specified in subclause 8.3.4.5.

– Otherwise (ChromaArrayType is equal to 1 or 2), the following text specifies the Intra prediction chroma samples for the current macroblock $pred_{Cb}[x, y]$ and $pred_{Cr}[x, y]$.

Both chroma blocks (Cb and Cr) of the macroblock use the same prediction mode. The prediction mode is applied to each of the chroma blocks separately. The process specified in this subclause is invoked for each chroma block. In the remainder of this subclause, chroma block refers to one of the two chroma blocks and the subscript C is used as a replacement of the subscript Cb or Cr.

The neighbouring samples p[x, y] that are constructed chroma samples prior to the deblocking filter process, with x = -1, y = -1..MbHeightC - 1 and with x = 0..MbWidthC - 1, y = -1, are derived as follows.

– The derivation process for neighbouring locations in subclause 6.4.11 is invoked for chroma locations with (x, y) assigned to (xN, yN) as input and mbAddrN and (xW, yW) as output.

– Each sample p[x, y] is derived as follows.

  – If any of the following conditions is true, the sample p[x, y] is marked as "not available for Intra chroma prediction"

    – mbAddrN is not available,

    – the macroblock mbAddrN is coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1,

    – the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1 and the current macroblock does not have mb_type equal to SI.

  – Otherwise, the sample p[x, y] is marked as "available for Intra chroma prediction" and the value of the sample p[x, y] is derived as follows.

    – The location of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as the input and the output is assigned to (xL, yL).

    – The location (xM, yM) of the upper-left chroma sample of the macroblock mbAddr is derived by

$$xM = (xL >> 4) * MbWidthC \quad (8\text{-}130)$$
$$yM = ((yL >> 4) * MbHeightC) + (yL \% 2) \quad (8\text{-}131)$$

    – Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows.

      – If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

$$p[x, y] = cS_C[xM + xW, yM + 2 * yW] \quad (8\text{-}132)$$

      – Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

$$p[x, y] = cS_C[xM + xW, yM + yW] \quad (8\text{-}133)$$

Let $pred_C[x, y]$ with x = 0..MbWidthC - 1, y = 0..MbHeightC - 1 denote the prediction samples for the chroma block samples.

Intra chroma prediction modes are specified in Table 8-5.

Table 8-5 – Specification of Intra chroma prediction modes and associated names

intra_chroma_pred_mode	Name of intra_chroma_pred_mode
0	Intra_Chroma_DC (prediction mode)
1	Intra_Chroma_Horizontal (prediction mode)
2	Intra_Chroma_Vertical (prediction mode)
3	Intra_Chroma_Plane (prediction mode)

Depending on intra_chroma_pred_mode, one of the Intra chroma prediction modes specified in subclauses 8.3.4.1 to 8.3.4.4 is invoked.

8.3.4.1 Specification of Intra_Chroma_DC prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 0.

For each chroma block of 4x4 samples indexed by chroma4x4BlkIdx = 0..(1 << (ChromaArrayType + 1)) – 1, the following applies.

– The position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx is derived as

$$xO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 0) \quad (8\text{-}134)$$

$$yO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 1) \quad (8\text{-}135)$$

– Depending on the values of xO and yO, the following applies.

– If (xO, yO) is equal to (0, 0) or xO and yO are greater than 0, the values of the prediction samples $pred_C[x + xO, y + yO]$ with x, y = 0..3 are derived as follows.

– If all samples p[x + xO, -1], with x = 0..3 and p[-1, y +yO], with y = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C[x + xO, y + yO]$, with x, y = 0..3 are derived as

$$pred_C[x + xO, y + yO] = \left(\sum_{x'=0}^{3} p[x'+xO,-1] + \sum_{y'=0}^{3} p[-1, y'+yO] + 4 \right) >> 3 \text{, with x, y = 0..3.} \quad (8\text{-}136)$$

– Otherwise, if any samples p[x + xO, -1], with x = 0..3 are marked as "not available for Intra chroma prediction" and all samples p[-1, y +yO], with y = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C[x + xO, y + yO]$, with x, y = 0..3 are derived as

$$pred_C[x + xO, y + yO] = \left(\sum_{y'=0}^{3} p[-1, y'+yO] + 2 \right) >> 2 \text{, with x, y = 0..3.} \quad (8\text{-}137)$$

– Otherwise, if any samples p[-1, y +yO], with y = 0..3 are marked as "not available for Intra chroma prediction" and all samples p[x + xO, -1], with x = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C[x + xO, y + yO]$, with x, y = 0..3 are derived as

$$pred_C[x + xO, y + yO] = \left(\sum_{x'=0}^{3} p[x'+xO,-1] + 2 \right) >> 2 \text{, with x, y = 0..3.} \quad (8\text{-}138)$$

– Otherwise (some samples p[x + xO, -1], with x = 0..3 and some samples p[-1, y +yO], with y = 0..3 are marked as "not available for Intra chroma prediction"), the values of the prediction samples $pred_C[x + xO, y + yO]$, with x, y = 0..3 are derived as

$$pred_C[x + xO, y + yO] = (1 << (BitDepth_C - 1)), \text{ with x, y = 0..3.} \quad (8\text{-}139)$$

– Otherwise, if xO is greater than 0 and yO is equal to 0, the values of the prediction samples $pred_C[x + xO, y + yO]$ with x, y = 0..3 are derived as follows.

  – If all samples p[x + xO, -1], with x = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C[x + xO, y + yO]$, with x, y = 0..3 are derived as

$$pred_C[x+xO, y+yO] = \left(\sum_{x'=0}^{3} p[x'+xO, -1] + 2 \right) >> 2 \text{, with } x, y = 0..3. \quad (8\text{-}140)$$

  – Otherwise, if all samples p[-1, y +yO], with y = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C[x + xO, y + yO]$, with x, y = 0..3 are derived as

$$pred_C[x+xO, y+yO] = \left(\sum_{y'=0}^{3} p[-1, y'+yO] + 2 \right) >> 2 \text{, with } x, y = 0..3. \quad (8\text{-}141)$$

  – Otherwise (some samples p[x + xO, -1], with x = 0..3 and some samples p[-1, y +yO], with y = 0..3 are marked as "not available for Intra chroma prediction"), the values of the prediction samples $pred_C[x + xO, y + yO]$, with x, y = 0..3 are derived as

$$pred_C[x + xO, y + yO] = (1 << (BitDepth_C - 1)) \text{, with } x, y = 0..3. \quad (8\text{-}142)$$

– Otherwise (xO is equal to 0 and yO is greater than 0), the values of the prediction samples $pred_C[x + xO, y + yO]$ with x, y = 0..3 are derived as follows.

  – If all samples p[-1, y +yO], with y = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C[x + xO, y + yO]$, with x, y = 0..3 are derived as

$$pred_C[x+xO, y+yO] = \left(\sum_{y'=0}^{3} p[-1, y'+yO] + 2 \right) >> 2 \text{, with } x, y = 0..3. \quad (8\text{-}143)$$

  – Otherwise, if all samples p[x + xO, -1], with x = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C[x + xO, y + yO]$, with x, y = 0..3 are derived as

$$pred_C[x+xO, y+yO] = \left(\sum_{x'=0}^{3} p[x'+xO, -1] + 2 \right) >> 2 \text{, with } x, y = 0..3. \quad (8\text{-}144)$$

  – Otherwise (some samples p[x + xO, -1], with x = 0..3 and some samples p[-1, y +yO], with y = 0..3 are marked as "not available for Intra chroma prediction"), the values of the prediction samples $pred_C[x + xO, y + yO]$, with x, y = 0..3 are derived as

$$pred_C[x + xO, y + yO] = (1 << (BitDepth_C - 1)) \text{, with } x, y = 0..3. \quad (8\text{-}145)$$

8.3.4.2 Specification of Intra_Chroma_Horizontal prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 1.

This mode shall be used only when the samples p[-1, y] with y = 0..MbHeightC - 1 are marked as "available for Intra chroma prediction".

The values of the prediction samples $pred_C[x, y]$ are derived as follows.

$$pred_C[x, y] = p[-1, y] \text{, with } x = 0..MbWidthC - 1 \text{ and } y = 0..MbHeightC - 1 \quad (8\text{-}146)$$

8.3.4.3 Specification of Intra_Chroma_Vertical prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 2.

This mode shall be used only when the samples p[x, -1] with x = 0..MbWidthC - 1 are marked as "available for Intra chroma prediction".

The values of the prediction samples $pred_C[x, y]$ are derived as follows.

$$pred_C[x, y] = p[x, -1], \text{ with } x = 0..MbWidthC - 1 \text{ and } y = 0..MbHeightC - 1 \quad (8\text{-}147)$$

8.3.4.4 Specification of Intra_Chroma_Plane prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 3.

This mode shall be used only when the samples p[x, -1], with x = 0..MbWidthC - 1 and p[-1, y], with y = -1..MbHeightC - 1 are marked as "available for Intra chroma prediction".

The values of the prediction samples $pred_C[x, y]$ are derived as follows.

Let the variable xCF be set equal to 4 * (ChromaArrayType == 3) and let the variable yCF be set equal to 4 * (ChromaArrayType != 1).

$$pred_C[x, y] = Clip1_C((a + b * (x - 3 - xCF) + c * (y - 3 - yCF) + 16) >> 5),$$
$$\text{with } x = 0..MbWidthC - 1 \text{ and } y = 0..MbHeightC - 1 \quad (8\text{-}148)$$

where:

$$a = 16 * (p[-1, MbHeightC - 1] + p[MbWidthC - 1, -1]) \quad (8\text{-}149)$$

$$b = ((34 - 29 * (ChromaArrayType == 3)) * H + 32) >> 6 \quad (8\text{-}150)$$

$$c = ((34 - 29 * (ChromaArrayType != 1)) * V + 32) >> 6 \quad (8\text{-}151)$$

and H and V are specified as

$$H = \sum_{x'=0}^{3+xCF} (x'+1) * (p[4 + xCF + x', -1] - p[2 + xCF - x', -1]) \quad (8\text{-}152)$$

$$V = \sum_{y'=0}^{3+yCF} (y'+1) * (p[-1, 4 + yCF + y'] - p[-1, 2 + yCF - y']) \quad (8\text{-}153)$$

8.3.4.5 Intra prediction for chroma samples with ChromaArrayType equal to 3

This process is invoked when ChromaArrayType is equal to 3. This process is invoked for I and SI macroblock types. It specifies how the Intra prediction chroma samples for the current macroblock are derived when ChromaArrayType is equal to 3.

Inputs to this process are constructed samples prior to the deblocking filter process from neighbouring Cb and Cr blocks and for Intra_NxN (where NxN is equal to 4x4 or 8x8) prediction mode, the associated values of IntraNxNPredMode from neighbouring macroblocks.

Outputs of this process are the Intra prediction samples of the Cb and Cr components of the macroblock or in case of the Intra_NxN prediction process, the outputs are NxN Cb sample arrays as part of the 16x16 Cb array of prediction samples of the macroblock, and NxN Cb sample arrays as part of the 16x16 Cb array of prediction samples of the macroblock.

Each Cb, Cr, and luma block with the same block index of the macroblock use the same prediction mode. The prediction mode is applied to each of the Cb and Cr blocks separately. The process specified in this subclause is invoked for each Cb and Cr block.

Depending on the macroblock prediction mode, the following applies.

– If the macroblock prediction mode is equal to Intra_4x4, the following applies.

  – The same process described in subclause 8.3.1 is also applied to Cb or Cr samples, substituting luma with Cb or Cr, substituting luma4x4BlkIdx with cb4x4BlkIdx or cr4x4BlkIdx, substituting $pred4x4_L$ with $pred4x4_{Cb}$ or $pred4x4_{Cr}$, and substituting $BitDepth_Y$ with $BitDepth_C$.

– The output variable Intra4x4PredMode[luma4x4BlkIdx] from the process described in subclause 8.3.1.1 is also used for the 4x4 Cb or 4x4 Cr blocks with index luma4x4BlkIdx equal to index cb4x4BlkIdx or cr4x4BlkIdx.

– The process to derive prediction Cb or Cr samples is identical to the process described in subclause 8.3.1.2 and its subsequent subclauses when substituting luma with Cb or Cr, substituting $pred4x4_L$ with $pred4x4_{Cb}$ or $pred4x4_{Cr}$, and substituting $BitDepth_Y$ with $BitDepth_C$.

– Otherwise, if the macroblock prediction mode is equal to Intra_8x8, the following applies.

– The same process described in subclause 8.3.2 is also applied to Cb or Cr samples, substituting luma with Cb or Cr, substituting luma8x8BlkIdx with cb8x8BlkIdx or cr8x8BlkIdx, substituting $pred8x8_L$ with $pred8x8_{Cb}$ or $pred8x8_{Cr}$, and substituting $BitDepth_Y$ with $BitDepth_C$.

– The output variable Intra8x8PredMode[luma8x8BlkIdx] from the process described in subclause 8.3.2.1 is used for the 8x8 Cb or 8x8 Cr blocks with index luma8x8BlkIdx equal to index cb8x8BlkIdx or cr8x8BlkIdx.

– The process to derive prediction Cb or Cr samples is identical to the process described in subclause 8.3.2.2 and its subsequent subclauses when substituting luma with Cb or Cr, substituting $pred8x8_L$ with $pred8x8_{Cb}$ or $pred8x8_{Cr}$, and substituting $BitDepth_Y$ with $BitDepth_C$.

– Otherwise, if the macroblock prediction mode is equal to Intra_16x16, the following applies.

– The same process described in subclause 8.3.3 and in the subsequent subclause 8.3.3.1 to 8.3.3.4 is also applied to Cb or Cr samples, substituting luma with Cb or Cr, substituting $pred_L$ with $pred_{Cb}$ or $pred_{Cr}$, and substituting $BitDepth_Y$ with $BitDepth_C$.

8.3.5 Sample construction process for I_PCM macroblocks

This process is invoked when mb_type is equal to I_PCM.

The variable dy is derived as follows.

– If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock, dy is set equal to 2.

– Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock), dy is set equal to 1.

The position of the upper-left luma sample of the current macroblock is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with CurrMbAddr as input and the output being assigned to (xP, yP).

The constructed luma samples prior to the deblocking process are generated as specified by:

```
for( i = 0; i < 256; i++ )
   S'L[ xP + ( i % 16 ), yP + dy * ( i / 16 ) ) ] = pcm_sample_luma[ i ]          (8-154)
```

When ChromaArrayType is not equal to 0, the constructed chroma samples prior to the deblocking process are generated as specified by:

```
for( i = 0; i < MbWidthC * MbHeightC; i++ ) {
   S'Cb[ ( xP / SubWidthC ) + ( i % MbWidthC ),
            ( ( yP + SubHeightC – 1 ) / SubHeightC ) + dy * ( i / MbWidthC ) ] =
              pcm_sample_chroma[ i ]                                            (8-155)
   S'Cr[ ( xP / SubWidthC ) + ( i % MbWidthC ),
            ( ( yP + SubHeightC – 1 ) / SubHeightC ) + dy * ( i / MbWidthC ) ] =
              pcm_sample_chroma[ i + MbWidthC * MbHeightC ]
}
```

8.4 Inter prediction process

This process is invoked when decoding P and B macroblock types.

Outputs of this process are Inter prediction samples for the current macroblock that are a 16x16 array $pred_L$ of luma samples and when ChromaArrayType is not equal to 0 two (MbWidthC)x(MbHeightC) arrays $pred_{Cb}$ and $pred_{Cr}$ of chroma samples, one for each of the chroma components Cb and Cr.

The partitioning of a macroblock is specified by mb_type. Each macroblock partition is referred to by mbPartIdx. When the macroblock partitioning consists of partitions that are equal to sub-macroblocks, each sub-macroblock can be further partitioned into sub-macroblock partitions as specified by sub_mb_type. Each sub-macroblock partition is

referred to by subMbPartIdx. When the macroblock partitioning does not consist of sub-macroblocks, subMbPartIdx is set equal to 0.

The following steps are specified for each macroblock partition or for each sub-macroblock partition.

The functions MbPartWidth( ), MbPartHeight( ), SubMbPartWidth( ), and SubMbPartHeight( ) describing the width and height of macroblock partitions and sub-macroblock partitions are specified in Table 7-13, 7-14, 7-17, and 7-18.

The range of the macroblock partition index mbPartIdx is derived as follows.

– If mb_type is equal to B_Skip or B_Direct_16x16, mbPartIdx proceeds over values 0..3.

– Otherwise (mb_type is not equal to B_Skip or B_Direct_16x16), mbPartIdx proceeds over values 0..NumMbPart(mb_type) – 1.

For each value of mbPartIdx, the variables partWidth and partHeight for each macroblock partition or sub-macroblock partition in the macroblock are derived as follows.

– If mb_type is not equal to P_8x8, P_8x8ref0, B_Skip, B_Direct_16x16, or B_8x8, subMbPartIdx is set equal to 0, and partWidth and partHeight are derived as

$$partWidth = MbPartWidth(mb\_type) \quad (8\text{-}156)$$

$$partHeight = MbPartHeight(mb\_type) \quad (8\text{-}157)$$

– Otherwise, if mb_type is equal to P_8x8 or P_8x8ref0, or mb_type is equal to B_8x8 and sub_mb_type[mbPartIdx] is not equal to B_Direct_8x8, subMbPartIdx proceeds over values 0..NumSubMbPart(sub_mb_type) – 1, and partWidth and partHeight are derived as

$$partWidth = SubMbPartWidth(sub\_mb\_type[mbPartIdx]) \quad (8\text{-}158)$$

$$partHeight = SubMbPartHeight(sub\_mb\_type[mbPartIdx]). \quad (8\text{-}159)$$

– Otherwise (mb_type is equal to B_Skip or B_Direct_16x16, or mb_type is equal to B_8x8 and sub_mb_type[mbPartIdx] is equal to B_Direct_8x8), subMbPartIdx proceeds over values 0..3, and partWidth and partHeight are derived as

$$partWidth = 4 \quad (8\text{-}160)$$

$$partHeight = 4 \quad (8\text{-}161)$$

When ChromaArrayType is not equal to 0, the variables partWidthC and partHeightC are derived as

$$partWidthC = partWidth / SubWidthC \quad (8\text{-}162)$$
$$partHeightC = partHeight / SubHeightC \quad (8\text{-}163)$$

Let the variable MvCnt be initially set equal to 0 before any invocation of subclause 8.4.1 for the macroblock.

The Inter prediction process for a macroblock partition mbPartIdx and a sub-macroblock partition subMbPartIdx consists of the following ordered steps

1. Derivation process for motion vector components and reference indices as specified in subclause 8.4.1.

   Inputs to this process are

   – a macroblock partition mbPartIdx,

   – a sub-macroblock partition subMbPartIdx.

   Outputs of this process are

   – luma motion vectors mvL0 and mvL1 and when ChromaArrayType is not equal to 0, the chroma motion vectors mvCL0 and mvCL1,

– reference indices refIdxL0 and refIdxL1,

– prediction list utilization flags predFlagL0 and predFlagL1,

– the sub-macroblock partition motion vector count subMvCnt.

2. The variable MvCnt is incremented by subMvCnt.

3. Derivation process for prediction weights as specified in subclause 8.4.3.

   Inputs to this process are

   – reference indices refIdxL0 and refIdxL1,

   – prediction list utilization flags predFlagL0 and predFlagL1.

   Outputs of this process are

   – variables for weighted prediction $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr.

4. Decoding process for Inter prediction samples as specified in subclause 8.4.2.

   Inputs to this process are

   – a macroblock partition mbPartIdx,

   – a sub-macroblock partition subMbPartIdx,

   – variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available), and partHeightC (if available),

   – luma motion vectors mvL0 and mvL1 and when ChromaArrayType is not equal to 0, the chroma motion vectors mvCL0 and mvCL1,

   – reference indices refIdxL0 and refIdxL1,

   – prediction list utilization flags predFlagL0 and predFlagL1,

   – variables for weighted prediction $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr.

   Outputs of this process are

   – inter prediction samples (pred); which are a (partWidth)x(partHeight) array $predPart_L$ of prediction luma samples and when ChromaArrayType is not equal to 0 two (partWidthC)x(partHeightC) arrays $predPart_{Cr}$, and $predPart_{Cb}$ of prediction chroma samples, one for each of the chroma components Cb and Cr.

For use in derivation processes of variables invoked later in the decoding process, the following assignments are made:

MvL0[mbPartIdx][subMbPartIdx] = mvL0 (8-164)

MvL1[mbPartIdx][subMbPartIdx] = mvL1 (8-165)

RefIdxL0[mbPartIdx] = refIdxL0 (8-166)

RefIdxL1[mbPartIdx] = refIdxL1 (8-167)

PredFlagL0[mbPartIdx] = predFlagL0 (8-168)

PredFlagL1[mbPartIdx] = predFlagL1 (8-169)

The location of the upper-left sample of the partition relative to the upper-left sample of the macroblock is derived by invoking the inverse macroblock partition scanning process as described in subclause 6.4.2.1 with mbPartIdx as the input and (xP, yP) as the output.

The location of the upper-left sample of the macroblock sub-partition relative to the upper-left sample of the macroblock partition is derived by invoking the inverse sub-macroblock partition scanning process as described in subclause 6.4.2.2 with subMbPartIdx as the input and (xS, yS) as the output.

The macroblock prediction is formed by placing the partition or sub-macroblock partition prediction samples in their correct relative positions in the macroblock, as follows.

The variable $pred_L[xP + xS + x, yP + yS + y]$ with x = 0 .. partWidth – 1, y = 0 .. partHeight – 1 is derived by

$$pred_L[xP + xS + x, yP + yS + y] = predPart_L[x, y] \quad (8\text{-}170)$$

When ChromaArrayType is not equal to 0, the variable $pred_C$ with x = 0..partWidthC – 1, y = 0..partHeightC – 1, and C in $pred_C$ and $predPart_C$ being replaced by Cb or Cr is derived by

$$pred_C[xP / SubWidthC + xS / SubWidthC + x, yP / SubHeightC + yS / SubHeightC + y] = predPart_C[x, y] \quad (8\text{-}171)$$

8.4.1 Derivation process for motion vector components and reference indices

Inputs to this process are

- a macroblock partition mbPartIdx,
- a sub-macroblock partition subMbPartIdx.

Outputs of this process are

- luma motion vectors mvL0 and mvL1 and when ChromaArrayType is not equal to 0, the chroma motion vectors mvCL0 and mvCL1,
- reference indices refIdxL0 and refIdxL1,
- prediction list utilization flags predFlagL0 and predFlagL1,
- a sub-partition macroblock motion vector count variable subMvCnt.

For the derivation of the variables mvL0 and mvL1 as well as refIdxL0 and refIdxL1, the following applies.

- If mb_type is equal to P_Skip, the derivation process for luma motion vectors for skipped macroblocks in P and SP slices in subclause 8.4.1.1 is invoked with the output being the luma motion vectors mvL0 and reference indices refIdxL0, and predFlagL0 is set equal to 1. mvL1 and refIdxL1 are marked as not available and predFlagL1 is set equal to 0. The sub-partition motion vector count variable subMvCnt is set equal to 1.
- Otherwise, if mb_type is equal to B_Skip or B_Direct_16x16 or sub_mb_type[mbPartIdx] is equal to B_Direct_8x8, the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in B slices in subclause 8.4.1.2 is invoked with mbPartIdx and subMbPartIdx as the input and the output being the luma motion vectors mvL0, mvL1, the reference indices refIdxL0, refIdxL1, the sub-partition motion vector count subMvCnt, and the prediction utilization flags predFlagL0 and predFlagL1.
- Otherwise, for X being replaced by either 0 or 1 in the variables predFlagLX, mvLX, refIdxLX, and in Pred_LX and in the syntax elements ref_idx_lX and mvd_lX, the following applies.

  1. The variables refIdxLX and predFlagLX are derived as follows.

     – If MbPartPredMode(mb_type, mbPartIdx) or SubMbPredMode(sub_mb_type[mbPartIdx]) is equal to Pred_LX or to BiPred,

     $$refIdxLX = ref\_idx\_lX[mbPartIdx] \quad (8\text{-}172)$$

     $$predFlagLX = 1 \quad (8\text{-}173)$$

     – Otherwise, the variables refIdxLX and predFlagLX are specified by

     $$refIdxLX = -1 \quad (8\text{-}174)$$

     $$predFlagLX = 0 \quad (8\text{-}175)$$

  2. The variable subMvCnt for sub-partition motion vector count is set equal to predFlagL0 + predFlagL1.

3. The variable currSubMbType is derived as follows.

   – If the macroblock type is equal to B_8x8, currSubMbType is set equal to sub_mb_type[mbPartIdx].

   – Otherwise (the macroblock type is not equal to B_8x8), currSubMbType is set equal to "na".

4. When predFlagLX is equal to 1, the derivation process for luma motion vector prediction in subclause 8.4.1.3 is invoked with mbPartIdx subMbPartIdx, refIdxLX, and currSubMbType as the inputs and the output being mvpLX. The luma motion vectors are derived by

$$mvLX[0] = mvpLX[0] + mvd\_lX[mbPartIdx][subMbPartIdx][0] \quad (8\text{-}176)$$

$$mvLX[1] = mvpLX[1] + mvd\_lX[mbPartIdx][subMbPartIdx][1] \quad (8\text{-}177)$$

When ChromaArrayType is not equal to 0 and predFlagLX (with X being either 0 or 1) is equal to 1, the derivation process for chroma motion vectors in subclause 8.4.1.4 is invoked with mvLX and refIdxLX as input and the output being mvCLX.

8.4.1.1 Derivation process for luma motion vectors for skipped macroblocks in P and SP slices

This process is invoked when mb_type is equal to P_Skip.

Outputs of this process are the motion vector mvL0 and the reference index refIdxL0.

The reference index refIdxL0 for a skipped macroblock is derived as follows.

$$refIdxL0 = 0. \quad (8\text{-}178)$$

For the derivation of the motion vector mvL0 of a P_Skip macroblock type, the following applies.

– The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx set equal to 0, subMbPartIdx set equal to 0, currSubMbType set equal to "na", and listSuffixFlag set equal to 0 as input and the output is assigned to mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, and refIdxL0B.

– The variable mvL0 is specified as follows.

  – If any of the following conditions are true, both components of the motion vector mvL0 are set equal to 0.

    – mbAddrA is not available

    – mbAddrB is not available

    – refIdxL0A is equal to 0 and both components of mvL0A are equal to 0

    – refIdxL0B is equal to 0 and both components of mvL0B are equal to 0

  – Otherwise, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxL0, and currSubMbType = "na" as inputs and the output is assigned to mvL0.

NOTE – The output is directly assigned to mvL0, since the predictor is equal to the actual motion vector.

8.4.1.2 Derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8

This process is invoked when mb_type is equal to B_Skip or B_Direct_16x16, or sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx and subMbPartIdx.

Outputs of this process are the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the sub-partition motion vector count subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The derivation process depends on the value of direct_spatial_mv_pred_flag, which is present in the bitstream in the slice header syntax as specified in subclause 7.3.3, and is specified as follows.

– If direct_spatial_mv_pred_flag is equal to 1, the mode in which the outputs of this process are derived is referred to as spatial direct prediction mode.

– Otherwise (direct_spatial_mv_pred_flag is equal to 0), mode in which the outputs of this process are derived is referred to as temporal direct prediction mode.

Both spatial and temporal direct prediction mode use the co-located motion vectors and reference indices as specified in subclause 8.4.1.2.1.

The motion vectors and reference indices are derived as follows.

– If spatial direct prediction mode is used, the direct motion vector and reference index prediction mode specified in subclause 8.4.1.2.2 is used, with subMvCnt being an output.

– Otherwise (temporal direct prediction mode is used), the direct motion vector and reference index prediction mode specified in subclause 8.4.1.2.3 is used and the variable subMvCnt is derived as follows.

  – If subMbPartIdx is equal to 0, subMvCnt is set equal to 2.

  – Otherwise (subMbPartIdx is not equal to 0), subMvCnt is set equal to 0.

8.4.1.2.1 Derivation process for the co-located 4x4 sub-macroblock partitions

Inputs to this process are mbPartIdx and subMbPartIdx.

Outputs of this process are the picture colPic, the co-located macroblock mbAddrCol, the motion vector mvCol, the reference index refIdxCol, and the variable vertMvScale (which can be One_To_One, Frm_To_Fld or Fld_To_Frm).

When RefPicList1[0] is a frame or a complementary field pair, let firstRefPicL1Top and firstRefPicL1Bottom be the top and bottom fields of RefPicList1[0], respectively, and let the following variables be specified as

topAbsDiffPOC = Abs(DiffPicOrderCnt(firstRefPicL1Top, CurrPic)) (8-179)

bottomAbsDiffPOC = Abs(DiffPicOrderCnt(firstRefPicL1Bottom, CurrPic)) (8-180)

The variable colPic specifies the picture that contains the co-located macroblock as specified in Table 8-6.

Table 8-6 – Specification of the variable colPic

field_pic_flag	RefPicList1[0] is …	mb_field_decoding_flag	additional condition	colPic
1	a field of a decoded frame			the frame containing RefPicList1[0]
	a decoded field			RefPicList1[0]
0	a decoded frame			RefPicList1[0]
	a complementary field pair	0	topAbsDiffPOC < bottomAbsDiffPOC	firstRefPicL1Top
			topAbsDiffPOC >= bottomAbsDiffPOC	firstRefPicL1Bottom
		1	(CurrMbAddr & 1) == 0	firstRefPicL1Top
			(CurrMbAddr & 1) != 0	firstRefPicL1Bottom

When direct_8x8_inference_flag is equal to 1, subMbPartIdx is set as follows.

subMbPartIdx = mbPartIdx (8-181)

Let PicCodingStruct(X) be a function with the argument X being either CurrPic or colPic. It is specified in Table 8-7.

Table 8-7 – Specification of PicCodingStruct(X)

X is coded with field_pic_flag equal to …	mb_adaptive_frame_field_flag	PicCodingStruct(X)
1		FLD
0	0	FRM
0	1	AFRM

With luma4x4BlkIdx = mbPartIdx * 4 + subMbPartIdx, the inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx as the input and (x, y) assigned to (xCol, yCol) as the output.

Table 8-8 specifies the co-located macroblock address mbAddrCol, yM, and the variable vertMvScale in two steps:

1. Specification of a macroblock address mbAddrX depending on PicCodingStruct(CurrPic), and PicCodingStruct(colPic).

   NOTE – It is not possible for CurrPic and colPic picture coding types to be either (FRM, AFRM) or (AFRM, FRM) because these picture coding types must be separated by an IDR picture.

2. Specification of mbAddrCol, yM, and vertMvScale depending on mb_field_decoding_flag and the variable fieldDecodingFlagX, which is derived as follows.

   – If the macroblock mbAddrX in the picture colPic is a field macroblock, fieldDecodingFlagX is set equal to 1

   – Otherwise (the macroblock mbAddrX in the picture colPic is a frame macroblock), fieldDecodingFlagX is set equal to 0.

Unspecified values in Table 8-8 indicate that the value of the corresponding variable is not relevant for the current table row.

mbAddrCol is set equal to CurrMbAddr or to one of the following values.

$$mbAddrCol1 = 2 * PicWidthInMbs * (CurrMbAddr / PicWidthInMbs) + (CurrMbAddr \% PicWidthInMbs) + PicWidthInMbs * (yCol / 8) \quad (8\text{-}182)$$

$$mbAddrCol2 = 2 * CurrMbAddr + (yCol / 8) \quad (8\text{-}183)$$

$$mbAddrCol3 = 2 * CurrMbAddr + bottom\_field\_flag \quad (8\text{-}184)$$

$$mbAddrCol4 = PicWidthInMbs * (CurrMbAddr / (2 * PicWidthInMbs)) + (CurrMbAddr \% PicWidthInMbs) \quad (8\text{-}185)$$

$$mbAddrCol5 = CurrMbAddr / 2 \quad (8\text{-}186)$$

$$mbAddrCol6 = 2 * (CurrMbAddr / 2) + ((topAbsDiffPOC < bottomAbsDiffPOC) ? 0 : 1) \quad (8\text{-}187)$$

$$mbAddrCol7 = 2 * (CurrMbAddr / 2) + (yCol / 8) \quad (8\text{-}188)$$

Table 8-8 – Specification of mbAddrCol, yM, and vertMvScale

PicCodingStruct(CurrPic)	PicCodingStruct(colPic)	mbAddrX	mb_field_decoding_flag	fieldDecodingFlagX	mbAddrCol	yM	vertMvScale
FLD	FLD				CurrMbAddr	yCol	One_To_One
	FRM				mbAddrCol1	(2 * yCol) % 16	Frm_To_Fld
	AFRM	2*CurrMbAddr		0	mbAddrCol2	(2 * yCol) % 16	Frm_To_Fld
				1	mbAddrCol3	yCol	One_To_One
FRM	FLD				mbAddrCol4	8 * ((CurrMbAddr / PicWidthInMbs) % 2) + 4 * (yCol / 8)	Fld_To_Frm
	FRM				CurrMbAddr	yCol	One_To_One
AFRM	FLD		0		mbAddrCol5	8 * (CurrMbAddr % 2) +4 * (yCol / 8)	Fld_To_Frm
			1		mbAddrCol5	yCol	One_To_One
	AFRM	CurrMbAddr	0	0	CurrMbAddr	yCol	One_To_One
				1	mbAddrCol6	8 * (CurrMbAddr % 2) + 4 * (yCol / 8)	Fld_To_Frm
		CurrMbAddr	1	0	mbAddrCol7	(2 * yCol) % 16	Frm_To_Fld
				1	CurrMbAddr	yCol	One_To_One

Let mbPartIdxCol be the macroblock partition index of the co-located partition and subMbPartIdxCol the sub-macroblock partition index of the co-located sub-macroblock partition. The partition in the macroblock mbAddrCol inside the picture colPic covering the sample (xCol, yM) is assigned to mbPartIdxCol and the sub-macroblock partition inside the partition mbPartIdxCol covering the sample (xCol, yM) in the macroblock mbAddrCol inside the picture colPic is assigned to subMbPartIdxCol.

The prediction utilization flags predFlagL0Col and predFlagL1Col are set equal to PredFlagL0[mbPartIdxCol] and PredFlagL1[mbPartIdxCol], respectively, which are the prediction utilization flags that have been assigned to the macroblock partition mbAddrCol\mbPartIdxCol inside the picture colPic.

The motion vector mvCol and the reference index refIdxCol are derived as follows.

– If the macroblock mbAddrCol is coded in Intra macroblock prediction mode or both prediction utilization flags, predFlagL0Col and predFlagL1Col are equal to 0, both components of mvCol are set equal to 0 and refIdxCol is set equal to –1.

– Otherwise, the following applies.

  – If predFlagL0Col is equal to 1, the motion vector mvCol and the reference index refIdxCol are set equal to MvL0[mbPartIdxCol][subMbPartIdxCol] and RefIdxL0[mbPartIdxCol], respectively, which are the motion vector mvL0 and the reference index refIdxL0 that have been assigned to the (sub-)macroblock partition mbAddrCol\mbPartIdxCol\subMbPartIdxCol inside the picture colPic.

  – Otherwise (predFlagL0Col is equal to 0 and predFlagL1Col is equal to 1), the motion vector mvCol and the reference index refIdxCol are set equal to MvL1[mbPartIdxCol][subMbPartIdxCol] and RefIdxL1[mbPartIdxCol], respectively, which are the motion vector mvL1 and the reference index refIdxL1

that have been assigned to the (sub-)macroblock partition mbAddrCol\mbPartIdxCol\subMbPartIdxCol inside the picture colPic.

8.4.1.2.2 Derivation process for spatial direct luma motion vector and reference index prediction mode

This process is invoked when direct_spatial_mv_pred_flag is equal to 1 and any of the following conditions is true.

- mb_type is equal to B_Skip
- mb_type is equal to B_Direct_16x16
- sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx, subMbPartIdx.

Outputs of this process are the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the sub-partition motion vector count subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The reference indices refIdxL0 and refIdxL1 and the variable directZeroPredictionFlag are derived by applying the following ordered steps.

1. Let the variable currSubMbType be set equal to sub_mb_type[mbPartIdx].
2. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx = 0, subMbPartIdx = 0, currSubMbType, and listSuffixFlag = 0 as inputs and the output is assigned to the motion vectors mvL0N and the reference indices refIdxL0N with N being replaced by A, B, or C.
3. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx = 0, subMbPartIdx = 0, currSubMbType, and listSuffixFlag = 1 as inputs and the output is assigned to the motion vectors mvL1N and the reference indices refIdxL1N with N being replaced by A, B, or C.

   NOTE 1 – The motion vectors mvL0N, mvL1N and the reference indices refIdxL0N, refIdxL1N are identical for all 4x4 sub-macroblock partitions of a macroblock.

4. The reference indices refIdxL0, refIdxL1, and directZeroPredictionFlag are derived by

$$refIdxL0 = MinPositive(refIdxL0A, MinPositive(refIdxL0B, refIdxL0C)) \quad (8\text{-}189)$$
$$refIdxL1 = MinPositive(refIdxL1A, MinPositive(refIdxL1B, refIdxL1C)) \quad (8\text{-}190)$$
$$directZeroPredictionFlag = 0 \quad (8\text{-}191)$$

   where

$$MinPositive(x, y) = \begin{cases} Min(x, y) & \text{if } x >= 0 \text{ and } y >= 0 \\ Max(x, y) & \text{otherwise} \end{cases} \quad (8\text{-}192)$$

5. When both reference indices refIdxL0 and refIdxL1 are less than 0,

$$refIdxL0 = 0 \quad (8\text{-}193)$$
$$refIdxL1 = 0 \quad (8\text{-}194)$$
$$directZeroPredictionFlag = 1 \quad (8\text{-}195)$$

The process specified in subclause 8.4.1.2.1 is invoked with mbPartIdx, subMbPartIdx given as input and the output is assigned to refIdxCol and mvCol.

The variable colZeroFlag is derived as follows.

- If all of the following conditions are true, colZeroFlag is set equal to 1.
  - RefPicList1[0] is currently marked as "used for short-term reference",
  - refIdxCol is equal to 0,
  - both motion vector components mvCol[0] and mvCol[1] lie in the range of -1 to 1 in units specified as follows.
    - If the co-located macroblock is a frame macroblock, the units of mvCol[0] and mvCol[1] are units of quarter luma frame samples.

– Otherwise (the co-located macroblock is a field macroblock), the units of mvCol[0] and mvCol[1] are units of quarter luma field samples.

NOTE 2 – For purposes of determining the condition above, the value mvCol[1] is not scaled to use the units of a motion vector for the current macroblock in cases when the current macroblock is a frame macroblock and the co-located macroblock is a field macroblock or when the current macroblock is a field macroblock and the co-located macroblock is a frame macroblock. This aspect differs from the use of mvCol[1] in the temporal direct mode as specified in subclause 8.4.1.2.3, which applies scaling to the motion vector of the co-located macroblock to use the same units as the units of a motion vector for the current macroblock, using Equation 8-198 or Equation 8-199 in these cases.

– Otherwise, colZeroFlag is set equal to 0.

The motion vectors mvLX (with X being 0 or 1) are derived as follows.

– If any of the following conditions is true, both components of the motion vector mvLX are set equal to 0.
  – directZeroPredictionFlag is equal to 1
  – refIdxLX is less than 0
  – refIdxLX is equal to 0 and colZeroFlag is equal to 1
– Otherwise, the process specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxLX, and currSubMbType as inputs and the output is assigned to mvLX.

NOTE 3 – The motion vector mvLX returned from subclause 8.4.1.3 is identical for all 4x4 sub-macroblock partitions of a macroblock for which the process is invoked.

The prediction utilization flags predFlagL0 and predFlagL1 are derived as specified using Table 8-9.

Table 8-9 – Assignment of prediction utilization flags

refIdxL0	refIdxL1	predFlagL0	predFlagL1
>= 0	>= 0	1	1
>= 0	< 0	1	0
< 0	>= 0	0	1

The variable subMvCnt is derived as follows.

– If subMbPartIdx is not equal to 0, subMvCnt is set equal to 0.
– Otherwise (subMbPartIdx is equal to 0), subMvCnt is set equal to predFlagL0 + predFLagL1.

8.4.1.2.3 Derivation process for temporal direct luma motion vector and reference index prediction mode

This process is invoked when direct_spatial_mv_pred_flag is equal to 0 and any of the following conditions is true.

– mb_type is equal to B_Skip
– mb_type is equal to B_Direct_16x16
– sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx and subMbPartIdx.

Outputs of this process are the motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The process specified in subclause 8.4.1.2.1 is invoked with mbPartIdx, subMbPartIdx given as input and the output is assigned to colPic, mbAddrCol, mvCol, refIdxCol, and vertMvScale.

The reference indices refIdxL0 and refIdxL1 are derived as follows.

refIdxL0 = ((refIdxCol < 0) ? 0 : MapColToList0(refIdxCol)) (8-196)

refIdxL1 = 0 (8-197)

NOTE 1 – If the current macroblock is a field macroblock, refIdxL0 and refIdxL1 index a list of fields; otherwise (the current macroblock is a frame macroblock), refIdxL0 and refIdxL1 index a list of frames or complementary reference field pairs.

Let refPicCol be a frame, a field, or a complementary field pair that was referred by the reference index refIdxCol when decoding the co-located macroblock mbAddrCol inside the picture colPic. The function MapColToList0(refIdxCol) is specified as follows.

– If vertMvScale is equal to One_To_One, the following applies.
  – If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies.
    – Let refIdxL0Frm be the lowest valued reference index in the current reference picture list RefPicList0 that references the frame or complementary field pair that contains the field refPicCol. RefPicList0 shall contain a frame or complementary field pair that contains the field refPicCol. The return value of MapColToList0( ) is specified as follows.
      – If the field referred to by refIdxCol has the same parity as the current macroblock, MapColToList0(refIdxCol) returns the reference index (refIdxL0Frm << 1).
      – Otherwise (the field referred by refIdxCol has the opposite parity of the current macroblock), MapColToList0(refIdxCol) returns the reference index ((refIdxL0Frm << 1) + 1).
  – Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list RefPicList0 that references refPicCol. RefPicList0 shall contain refPicCol.
– Otherwise, if vertMvScale is equal to Frm_To_Fld, the following applies.
  – If field_pic_flag is equal to 0, let refIdxL0Frm be the lowest valued reference index in the current reference picture list RefPicList0 that references refPicCol. MapColToList0(refIdxCol) returns the reference index (refIdxL0Frm << 1). RefPicList0 shall contain refPicCol.
  – Otherwise (field_pic_flag is equal to 1), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list RefPicList0 that references the field of refPicCol with the same parity as the current picture CurrPic. RefPicList0 shall contain the field of refPicCol with the same parity as the current picture CurrPic.
– Otherwise (vertMvScale is equal to Fld_To_Frm), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list RefPicList0 that references the frame or complementary field pair that contains refPicCol. RefPicList0 shall contain a frame or complementary field pair that contains the field refPicCol.

NOTE 2 – A decoded reference picture that was marked as "used for short-term reference" when it was referenced in the decoding process of the picture containing the co-located macroblock may have been modified to be marked as "used for long-term reference" before being used for reference for inter prediction using the direct prediction mode for the current macroblock.

Depending on the value of vertMvScale the vertical component of mvCol is modified as follows.

– If vertMvScale is equal to Frm_To_Fld

$$mvCol[\,1\,] = mvCol[\,1\,] / 2 \tag{8-198}$$

– Otherwise, if vertMvScale is equal to Fld_To_Frm

$$mvCol[\,1\,] = mvCol[\,1\,] * 2 \tag{8-199}$$

– Otherwise (vertMvScale is equal to One_To_One), mvCol[1] remains unchanged.

The variables currPicOrField, pic0, and pic1, are derived as follows.

– If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies.
  – currPicOrField is the field of the current picture CurrPic that has the same parity as the current macroblock.
  – pic1 is the field of RefPicList1[0] that has the same parity as the current macroblock.
  – The variable pic0 is derived as follows.
    – If refIdxL0 % 2 is equal to 0, pic0 is the field of RefPicList0[refIdxL0 / 2] that has the same parity as the current macroblock.
    – Otherwise (refIdxL0 % 2 is not equal to 0), pic0 is the field of RefPicList0[refIdxL0 / 2] that has the opposite parity of the current macroblock.

– Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), currPicOrField is the current picture CurrPic, pic1 is the decoded reference picture RefPicList1[0], and pic0 is the decoded reference picture RefPicList0[refIdxL0].

The two motion vectors mvL0 and mvL1 for each 4x4 sub-macroblock partition of the current macroblock are derived as follows:

NOTE 3 – It is often the case that many of the 4x4 sub-macroblock partitions share the same motion vectors and reference pictures. In these cases, temporal direct mode motion compensation can calculate the inter prediction sample values in larger units than 4x4 luma sample blocks. For example, when direct_8x8_inference_flag is equal to 1, at least each 8x8 luma sample quadrant of the macroblock shares the same motion vectors and reference pictures.

– If the reference index refIdxL0 refers to a long-term reference picture, or DiffPicOrderCnt(pic1, pic0) is equal to 0, the motion vectors mvL0, mvL1 for the direct mode partition are derived by

$$mvL0 = mvCol \tag{8-200}$$

$$mvL1 = 0 \tag{8-201}$$

– Otherwise, the motion vectors mvL0, mvL1 are derived as scaled versions of the motion vector mvCol of the co-located sub-macroblock partition as specified below (see Figure 8-2)

$$tx = (16\,384 + Abs(td / 2)) / td \tag{8-202}$$

$$DistScaleFactor = Clip3(-1024, 1023, (tb * tx + 32) >> 6) \tag{8-203}$$

$$mvL0 = (DistScaleFactor * mvCol + 128) >> 8 \tag{8-204}$$

$$mvL1 = mvL0 - mvCol \tag{8-205}$$

where tb and td are derived as follows.

$$tb = Clip3(-128, 127, DiffPicOrderCnt(currPicOrField, pic0)) \tag{8-206}$$

$$td = Clip3(-128, 127, DiffPicOrderCnt(pic1, pic0)) \tag{8-207}$$

NOTE 4 – mvL0 and mvL1 cannot exceed the ranges specified in Annex A.

The prediction utilization flags predFlagL0 and predFlagL1 are both set equal to 1.

Figure 8-2 illustrates the temporal direct-mode motion vector inference when the current picture is temporally between the reference picture from reference picture list 0 and the reference picture from reference picture list 1.

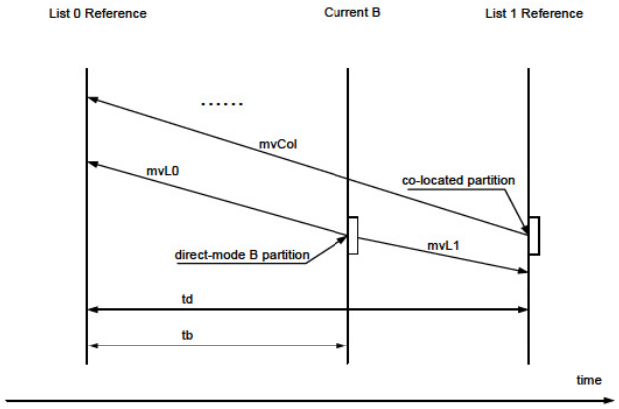


Figure 8-2 – Example for temporal direct-mode motion vector inference (informative)

8.4.1.3 Derivation process for luma motion vector prediction

Inputs to this process are

– the macroblock partition index mbPartIdx,

– the sub-macroblock partition index subMbPartIdx,

– the reference index of the current partition refIdxLX (with X being 0 or 1),

– the variable currSubMbType.

Output of this process is the prediction mvpLX of the motion vector mvLX (with X being 0 or 1).

The derivation process for the neighbouring blocks for motion data in subclause 8.4.1.3.2 is invoked with mbPartIdx, subMbPartIdx, currSubMbType, and listSuffixFlag = X (with X being 0 or 1 for refIdxLX being refIdxL0 or refIdxL1, respectively) as the input and with mbAddrN\mbPartIdxN\subMbPartIdxN, reference indices refIdxLXN and the motion vectors mvLXN with N being replaced by A, B, or C as the output.

The derivation process for median luma motion vector prediction in subclause 8.4.1.3.1 is invoked with mbAddrN\mbPartIdxN\subMbPartIdxN, mvLXN, refIdxLXN with N being replaced by A, B, or C and refIdxLX as the input and mvpLX as the output, unless one of the following is true.

– MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 0, and refIdxLXB is equal to refIdxLX,

$$mvpLX = mvLXB \tag{8-208}$$

– MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 1, and refIdxLXA is equal to refIdxLX,

$$mvpLX = mvLXA \tag{8-209}$$

– MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 0, and refIdxLXA is equal to refIdxLX,

$$mvpLX = mvLXA \tag{8-210}$$

– MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 1, and refIdxLXC is equal to refIdxLX,

$$mvpLX = mvLXC \tag{8-211}$$

Figure 8-3 illustrates the non-median prediction as specified in the four bulleted items above.

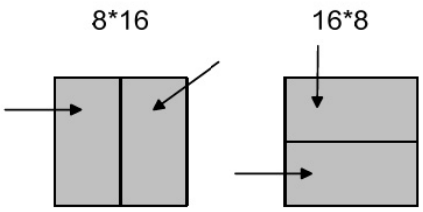


Figure 8-3 – Directional segmentation prediction (informative)

8.4.1.3.1 Derivation process for median luma motion vector prediction

Inputs to this process are

– the neighbouring partitions mbAddrN\mbPartIdxN\subMbPartIdxN (with N being replaced by A, B, or C),

– the motion vectors mvLXN (with N being replaced by A, B, or C) of the neighbouring partitions,

– the reference indices refIdxLXN (with N being replaced by A, B, or C) of the neighbouring partitions, and

– the reference index refIdxLX of the current partition.

Output of this process is the motion vector prediction mvpLX.

The variable mvpLX is derived as follows:

– When both partitions mbAddrB\mbPartIdxB\subMbPartIdxB and mbAddrC\mbPartIdxC\subMbPartIdxC are not available and mbAddrA\mbPartIdxA\subMbPartIdxA is available,

$$mvLXB = mvLXA \tag{8-212}$$

$$mvLXC = mvLXA \tag{8-213}$$

$$refIdxLXB = refIdxLXA \tag{8-214}$$

$$refIdxLXC = refIdxLXA \tag{8-215}$$

– Depending on reference indices refIdxLXA, refIdxLXB, or refIdxLXC, the following applies.

  – If one and only one of the reference indices refIdxLXA, refIdxLXB, or refIdxLXC is equal to the reference index refIdxLX of the current partition, the following applies. Let refIdxLXN be the reference index that is equal to refIdxLX, the motion vector mvLXN is assigned to the motion vector prediction mvpLX:

$$mvpLX = mvLXN \tag{8-216}$$

  – Otherwise, each component of the motion vector prediction mvpLX is given by the median of the corresponding vector components of the motion vector mvLXA, mvLXB, and mvLXC:

$$mvpLX[\,0\,] = Median(\,mvLXA[\,0\,], mvLXB[\,0\,], mvLXC[\,0\,]\,) \tag{8-217}$$

$$mvpLX[\,1\,] = Median(\,mvLXA[\,1\,], mvLXB[\,1\,], mvLXC[\,1\,]\,) \tag{8-218}$$

8.4.1.3.2 Derivation process for motion data of neighbouring partitions

Inputs to this process are

- the macroblock partition index mbPartIdx,
- the sub-macroblock partition index subMbPartIdx,
- the current sub-macroblock type currSubMbType,
- the list suffix flag listSuffixFlag.

Outputs of this process are (with N being replaced by A, B, or C)

- mbAddrN\mbPartIdxN\subMbPartIdxN specifying neighbouring partitions,
- the motion vectors mvLXN of the neighbouring partitions, and
- the reference indices refIdxLXN of the neighbouring partitions.

Variable names that include the string "LX" are interpreted with the X being equal to listSuffixFlag.

The partitions mbAddrN\mbPartIdxN\subMbPartIdxN with N being either A, B, or C are derived in the following ordered steps.

1. Let mbAddrD\mbPartIdxD\subMbPartIdxD be variables specifying an additional neighbouring partition.

- The process in subclause 6.4.10.7 is invoked with mbPartIdx, currSubMbType, and subMbPartIdx as input and the output is assigned to mbAddrN\mbPartIdxN\subMbPartIdxN with N being replaced by A, B, C, or D.

2. When the partition mbAddrC\mbPartIdxC\subMbPartIdxC is not available, the following applies

$$mbAddrC = mbAddrD \tag{8-219}$$

$$mbPartIdxC = mbPartIdxD \tag{8-220}$$

$$subMbPartIdxC = subMbPartIdxD \tag{8-221}$$

The motion vectors mvLXN and reference indices refIdxLXN (with N being A, B, or C) are derived as follows.

- If the macroblock partition or sub-macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN is not available or mbAddrN is coded in Intra prediction mode or predFlagLX of mbAddrN\mbPartIdxN\subMbPartIdxN is equal to 0, both components of mvLXN are set equal to 0 and refIdxLXN is set equal to –1.
- Otherwise, the following applies.
  - The motion vector mvLXN and reference index refIdxLXN are set equal to MvLX[mbPartIdxN][subMbPartIdxN] and RefIdxLX[mbPartIdxN], respectively, which are the motion vector mvLX and reference index refIdxLX that have been assigned to the (sub-)macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN.
  - The variables mvLXN[1] and refIdxLXN are further processed as follows.
    - If the current macroblock is a field macroblock and the macroblock mbAddrN is a frame macroblock

$$mvLXN[1] = mvLXN[1] / 2 \tag{8-222}$$

$$refIdxLXN = refIdxLXN * 2 \tag{8-223}$$

    - Otherwise, if the current macroblock is a frame macroblock and the macroblock mbAddrN is a field macroblock

$$mvLXN[1] = mvLXN[1] * 2 \tag{8-224}$$

$$refIdxLXN = refIdxLXN / 2 \tag{8-225}$$

– Otherwise, the vertical motion vector component mvLXN[1] and the reference index refIdxLXN remain unchanged.

8.4.1.4 Derivation process for chroma motion vectors

This process is only invoked when ChromaArrayType is not equal to 0.

Inputs to this process are a luma motion vector mvLX and a reference index refIdxLX.

Output of this process is a chroma motion vector mvCLX.

A chroma motion vector is derived from the corresponding luma motion vector.

The precision of the chroma motion vector components is $1 \div (4 * \text{SubWidthC})$ horizontally and $1 \div (4 * \text{SubHeightC})$ vertically.

NOTE – For example, when using the 4:2:0 chroma format, since the units of luma motion vectors are one-quarter luma sample units and chroma has half horizontal and vertical resolution compared to luma, the units of chroma motion vectors are one-eighth chroma sample units, i.e., a value of 1 for the chroma motion vector refers to a one-eighth chroma sample displacement. For example, when the luma vector applies to 8x16 luma samples, the corresponding chroma vector in 4:2:0 chroma format applies to 4x8 chroma samples and when the luma vector applies to 4x4 luma samples, the corresponding chroma vector in 4:2:0 chroma format applies to 2x2 chroma samples.

For the derivation of the motion vector mvCLX, the following applies.

– If ChromaArrayType is not equal to 1 or the current macroblock is a frame macroblock, the horizontal and vertical components of the chroma motion vector mvCLX are derived as

$$mvCLX[0] = mvLX[0] \quad (8\text{-}226)$$
$$mvCLX[1] = mvLX[1] \quad (8\text{-}227)$$

– Otherwise (ChromaArrayType is equal to 1 and the current macroblock is a field macroblock), only the horizontal component of the chroma motion vector mvCLX[0] is derived using Equation 8-226. The vertical component of the chroma motion vector mvCLX[1] is dependent on the parity of the current field or the current macroblock and the reference picture, which is referred by the reference index refIdxLX. mvCLX[1] is derived from mvLX[1] according to Table 8-10.

Table 8-10 – Derivation of the vertical component of the chroma vector in field coding mode

Parity conditions		mvCLX[1]
Reference picture (refIdxLX)	Current field (picture/macroblock)	
Top field	Bottom field	mvLX[1] + 2
Bottom field	Top field	mvLX[1] – 2
Otherwise		mvLX[1]

8.4.2 Decoding process for Inter prediction samples

Inputs to this process are

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx.

– variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available) and partHeightC (if available)

– luma motion vectors mvL0 and mvL1 and when ChromaArrayType is not equal to 0 chroma motion vectors mvCL0 and mvCL1

– reference indices refIdxL0 and refIdxL1

– prediction list utilization flags, predFlagL0 and predFlagL1,

– variables for weighted prediction $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr.

Outputs of this process are

– the Inter prediction samples predPart, which are a (partWidth)x(partHeight) array $predPart_L$ of prediction luma samples, and when ChromaArrayType is not equal to 0 two (partWidthC)x(partHeightC) arrays $predPart_{Cb}$, $predPart_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr.

Let $predPartL0_L$ and $predPartL1_L$ be (partWidth)x(partHeight) arrays of predicted luma sample values and when ChromaArrayType is not equal to 0 $predPartL0_{Cb}$, $predPartL1_{Cb}$, $predPartL0_{Cr}$, and $predPartL1_{Cr}$ be (partWidthC)x(partHeightC) arrays of predicted chroma sample values.

For LX being replaced by either L0 or L1 in the variables predFlagLX, RefPicListX, refIdxLX, refPicLX, predPartLX, the following is specified.

When predFlagLX is equal to 1, the following applies.

– The reference picture consisting of an ordered two-dimensional array $refPicLX_L$ of luma samples and when ChromaArrayType is not equal to 0 two ordered two-dimensional arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$ of chroma samples is derived by invoking the process specified in subclause 8.4.2.1 with refIdxLX and RefPicListX given as input.

– The array $predPartLX_L$ and when ChromaArrayType is not equal to 0 the arrays $predPartLX_{Cb}$ and $predPartLX_{Cr}$ are derived by invoking the process specified in subclause 8.4.2.2 with the current partition specified by mbPartIdx\subMbPartIdx, the motion vectors mvLX, mvCLX (if available), and the reference arrays with $refPicLX_L$, $refPicLX_{Cb}$ (if available), and $refPicLX_{Cr}$ (if available) given as input.

For C being replaced by L, Cb (if available), or Cr (if available), the array $predPart_C$ of the prediction samples of component C is derived by invoking the process specified in subclause 8.4.2.3 with the current partition specified by mbPartIdx and subMbPartIdx, the prediction utilization flags predFlagL0 and predFlagL1, the arrays $predPartL0_C$ and $predPartL1_C$, and the variables for weighted prediction $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} given as input.

8.4.2.1 Reference picture selection process

Input to this process is a reference index refIdxLX.

Output of this process is a reference picture consisting of a two-dimensional array of luma samples $refPicLX_L$ and, when ChromaArrayType is not equal to 0, two two-dimensional arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

Depending on field_pic_flag, the reference picture list RefPicListX (which has been derived as specified in subclause 8.2.4) consists of the following.

– If field_pic_flag is equal to 1, each entry of RefPicListX is a reference field or a field of a reference frame.

– Otherwise (field_pic_flag is equal to 0), each entry of RefPicListX is a reference frame or a complementary reference field pair.

For the derivation of the reference picture, the following applies.

– If field_pic_flag is equal to 1, the reference field or field of a reference frame RefPicListX[refIdxLX] is the output. The output reference field or field of a reference frame consists of a $(PicWidthInSamples_L)$x$(PicHeightInSamples_L)$ array of luma samples $refPicLX_L$ and, when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)$x$(PicHeightInSamples_C)$ arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

– Otherwise (field_pic_flag is equal to 0), the following applies.

  – If the current macroblock is a frame macroblock, the reference frame or complementary reference field pair RefPicListX[refIdxLX] is the output. The output reference frame or complementary reference field pair consists of a $(PicWidthInSamples_L)$x$(PicHeightInSamples_L)$ array of luma samples $refPicLX_L$ and, when ChromaArrayType is not equal to 0, two $(PicWidthInSamples_C)$x$(PicHeightInSamples_C)$ arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

  – Otherwise (the current macroblock is a field macroblock), the following applies.

    – Let refFrame be the reference frame or complementary reference field pair RefPicListX[refIdxLX / 2].

    – The field of refFrame is selected as follows.

      – If refIdxLX % 2 is equal to 0, the field of refFrame that has the same parity as the current macroblock is the output.

– Otherwise (refIdxLX % 2 is equal to 1), the field of refFrame that has the opposite parity as the current macroblock is the output.

– The output reference field or field of a reference frame consists of a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$ / 2) array of luma samples $refPicLX_L$ and, when ChromaArrayType is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$ / 2) arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

Depending on separate_colour_plane_flag, the following applies.

– If separate_colour_plane_flag is equal to 0, the reference picture sample arrays $refPicLX_L$, $refPicLX_{Cb}$ (if available), and $refPicLX_{Cr}$ (if available) correspond to decoded sample arrays S_L, S_{Cb} (if available), S_{Cr} (if available) derived in subclause 8.7 for a previously-decoded reference field or reference frame or complementary reference field pair or field of a reference frame.

– Otherwise (separate_colour_plane_flag is equal to 1), the following applies.

  – If colour_plane_id is equal to 0, the reference picture sample array $refPicLX_L$ corresponds to the decoded sample array S_L derived in subclause 8.7 for a previously-decoded reference field or reference frame or complementary reference field pair or field of a reference frame.

  – Otherwise, if colour_plane_id is equal to 1, the reference picture sample array $refPicLX_L$ corresponds to the decoded sample array S_{Cb} derived in subclause 8.7 for a previously-decoded reference field or reference frame or complementary reference field pair or field of a reference frame.

  – Otherwise (colour_plane_id is equal to 2), the reference picture sample array $refPicLX_L$ corresponds to the decoded sample array S_{Cr} derived in subclause 8.7 for a previously-decoded reference field or reference frame or complementary reference field pair or field of a reference frame.

8.4.2.2 Fractional sample interpolation process

Inputs to this process are

– the current partition given by its partition index mbPartIdx and its sub-macroblock partition index subMbPartIdx,

– the width and height partWidth, partHeight of this partition in luma-sample units,

– a luma motion vector mvLX given in quarter-luma-sample units,

– when ChromaArrayType is not equal to 0, a chroma motion vector mvCLX with a precision of one-(4*SubWidthC)-th chroma-sample units horizontally and one-(4*SubHeightC)-th chroma-sample units vertically, and

– the selected reference picture sample arrays $refPicLX_L$, and when ChromaArrayType is not equal to 0, $refPicLX_{Cb}$, and $refPicLX_{Cr}$.

Outputs of this process are

– a (partWidth)x(partHeight) array $predPartLX_L$ of prediction luma sample values and

– when ChromaArrayType is not equal to 0, two (partWidthC)x(partHeightC) arrays $predPartLX_{Cb}$, and $predPartLX_{Cr}$ of prediction chroma sample values.

Let (xA_L, yA_L) be the location given in full-sample units of the upper-left luma sample of the current partition given by mbPartIdx\subMbPartIdx relative to the upper-left luma sample location of the given two-dimensional array of luma samples.

Let ($xInt_L$, $yInt_L$) be a luma location given in full-sample units and ($xFrac_L$, $yFrac_L$) be an offset given in quarter-sample units. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $refPicLX_L$, $refPicLX_{Cb}$ (if available), and $refPicLX_{Cr}$ (if available).

For each luma sample location ($0 <= x_L < partWidth$, $0 <= y_L < partHeight$) inside the prediction luma sample array $predPartLX_L$, the corresponding prediction luma sample value $predPartLX_L[x_L, y_L]$ is derived as follows:

– The variables $xInt_L$, $yInt_L$, $xFrac_L$, and $yFrac_L$ are derived by

$$xInt_L = xA_L + (mvLX[0] >> 2) + x_L \quad (8\text{-}228)$$
$$yInt_L = yA_L + (mvLX[1] >> 2) + y_L \quad (8\text{-}229)$$

$xFrac_L = mvLX[\ 0\ ] \ \&\ 3$ (8-230)
$yFrac_L = mvLX[\ 1\ ] \ \&\ 3$ (8-231)

– The prediction luma sample value $predPartLX_L[\ x_L, y_L\ ]$ is derived by invoking the process specified in subclause 8.4.2.2.1 with $(\ xInt_L, yInt_L\ )$, $(\ xFrac_L, yFrac_L\ )$ and $refPicLX_L$ given as input.

When ChromaArrayType is not equal to 0, the following applies.

Let $(\ xInt_C, yInt_C\ )$ be a chroma location given in full-sample units and $(\ xFrac_C, yFrac_C\ )$ be an offset given in one-(4*SubWidthC)-th chroma-sample units horizontally and one-(4*SubHeightC)-th chroma-sample units vertically. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $refPicLX_{Cb}$, and $refPicLX_{Cr}$.

For each chroma sample location ($0 <= x_C < partWidthC$, $0 <= y_C < partHeightC$) inside the prediction chroma sample arrays $predPartLX_{Cb}$ and $predPartLX_{Cr}$, the corresponding prediction chroma sample values $predPartLX_{Cb}[\ x_C, y_C\ ]$ and $predPartLX_{Cr}[\ x_C, y_C\ ]$ are derived as follows:

– Depending on ChromaArrayType, the variables $xInt_C$, $yInt_C$, $xFrac_C$, and $yFrac_C$ are derived as follows.

  – If ChromaArrayType is equal to 1,

$xInt_C = (\ xA_L / SubWidthC\ ) + (\ mvCLX[\ 0\ ] >> 3\ ) + x_C$ (8-232)
$yInt_C = (\ yA_L / SubHeightC\ ) + (\ mvCLX[\ 1\ ] >> 3\ ) + y_C$ (8-233)

$xFrac_C = mvCLX[\ 0\ ] \ \&\ 7$ (8-234)
$yFrac_C = mvCLX[\ 1\ ] \ \&\ 7$ (8-235)

  – Otherwise, if ChromaArrayType is equal to 2,

$xInt_C = (\ xA_L / SubWidthC\ ) + (\ mvCLX[\ 0\ ] >> 3\ ) + x_C$ (8-236)
$yInt_C = (\ yA_L / SubHeightC\ ) + (\ mvCLX[\ 1\ ] >> 2\ ) + y_C$ (8-237)

$xFrac_C = mvCLX[\ 0\ ] \ \&\ 7$ (8-238)
$yFrac_C = (\ mvCLX[\ 1\ ] \ \&\ 3\ ) << 1$ (8-239)

  – Otherwise (ChromaArrayType is equal to 3),

$xInt_C = xA_L + (\ mvLX[\ 0\ ] >> 2\ ) + x_C$ (8-240)
$yInt_C = yA_L + (\ mvLX[\ 1\ ] >> 2\ ) + y_C$ (8-241)

$xFrac_C = (\ mvCX[\ 0\ ] \ \&\ 3\ )$ (8-242)
$yFrac_C = (\ mvCX[\ 1\ ] \ \&\ 3\ )$ (8-243)

– Depending on ChromaArrayType, the following applies.

  – If ChromaArrayType is not equal to 3, the following applies.

    – The prediction sample value $predPartLX_{Cb}[\ x_C, y_C\ ]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with $(\ xInt_C, yInt_C\ )$, $(\ xFrac_C, yFrac_C\ )$ and $refPicLX_{Cb}$ given as input.

    – The prediction sample value $predPartLX_{Cr}[\ x_C, y_C\ ]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with $(\ xInt_C, yInt_C\ )$, $(\ xFrac_C, yFrac_C\ )$ and $refPicLX_{Cr}$ given as input.

  – Otherwise (ChromaArrayType is equal to 3), the following applies.

    – The prediction sample value $predPartLX_{Cb}[\ x_C, y_C\ ]$ is derived by invoking the process specified in subclause 8.4.2.2.1 with $(\ xInt_C, yInt_C\ )$, $(\ xFrac_C, yFrac_C\ )$ and $refPicLX_{Cb}$ given as input.

    – The prediction sample value $predPartLX_{Cr}[\ x_C, y_C\ ]$ is derived by invoking the process specified in subclause 8.4.2.2.1 with $(\ xInt_C, yInt_C\ )$, $(\ xFrac_C, yFrac_C\ )$ and $refPicLX_{Cr}$ given as input.

8.4.2.2.1 Luma sample interpolation process

Inputs to this process are

– a luma location in full-sample units ($xInt_L$, $yInt_L$),

– a luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$), and

– the luma sample array of the selected reference picture $refPicLX_L$.

Output of this process is a predicted luma sample value $predPartLX_L[x_L, y_L]$.

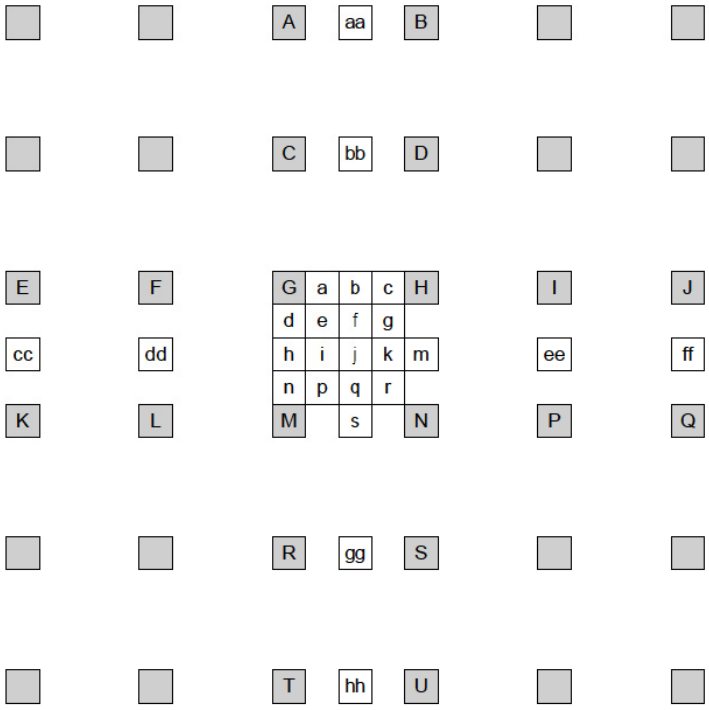


Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation

The variable $refPicHeightEffective_L$, which is the height of the effective reference picture luma array, is derived as follows.

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, $refPicHeightEffective_L$ is set equal to $PicHeightInSamples_L$.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), $refPicHeightEffective_L$ is set equal to $PicHeightInSamples_L$ / 2.

In Figure 8-4, the positions labelled with upper-case letters within shaded blocks represent luma samples at full-sample locations inside the given two-dimensional array $refPicLX_L$ of luma samples. These samples may be used for generating the predicted luma sample value $predPartLX_L[x_L, y_L]$. The locations (xZ_L, yZ_L) for each of the corresponding luma samples Z, where Z may be A, B, C, D, E, F, G, H, I, J, K, L, M, N, P, Q, R, S, T, or U, inside the given array $refPicLX_L$ of luma samples are derived as follows:

$$xZ_L = Clip3(0, PicWidthInSamples_L - 1, xInt_L + xDZ_L)$$
$$yZ_L = Clip3(0, refPicHeightEffective_L - 1, yInt_L + yDZ_L) \quad (8\text{-}244)$$

Table 8-11 specifies (xDZ_L, yDZ_L) for different replacements of Z.

Table 8-11 – Differential full-sample luma locations

Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P	Q	R	S	T	U
xDZ_L	0	1	0	1	-2	-1	0	1	2	3	-2	-1	0	1	2	3	0	1	0	1
yDZ_L	-2	-2	-1	-1	0	0	0	0	0	0	1	1	1	1	1	1	2	2	3	3

Given the luma samples 'A' to 'U' at full-sample locations (xA_L, yA_L) to (xU_L, yU_L), the luma samples 'a' to 's' at fractional sample positions are derived by the following rules. The luma prediction values at half sample positions are derived by applying a 6-tap filter with tap values (1, -5, 20, 20, -5, 1). The luma prediction values at quarter sample positions are derived by averaging samples at full and half sample positions. The process for each fractional position is described below.

– The samples at half sample positions labelled b are derived by first calculating intermediate values denoted as b_1 by applying the 6-tap filter to the nearest integer position samples in the horizontal direction. The samples at half sample positions labelled h are derived by first calculating intermediate values denoted as h_1 by applying the 6-tap filter to the nearest integer position samples in the vertical direction:

$$b_1 = (E - 5 * F + 20 * G + 20 * H - 5 * I + J) \quad (8\text{-}245)$$
$$h_1 = (A - 5 * C + 20 * G + 20 * M - 5 * R + T) \quad (8\text{-}246)$$

The final prediction values b and h are derived using:

$$b = Clip1_Y((b_1 + 16) >> 5) \quad (8\text{-}247)$$
$$h = Clip1_Y((h_1 + 16) >> 5) \quad (8\text{-}248)$$

– The samples at half sample position labelled as j are derived by first calculating intermediate value denoted as j_1 by applying the 6-tap filter to the intermediate values of the closest half sample positions in either the horizontal or vertical direction because these yield an equal result.

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff, \text{ or} \quad (8\text{-}249)$$
$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (8\text{-}250)$$

where intermediate values denoted as aa, bb, gg, s_1 and hh are derived by applying the 6-tap filter horizontally in the same manner as the derivation of b_1 and intermediate values denoted as cc, dd, ee, m_1 and ff are derived by applying the 6-tap filter vertically in the same manner as the derivation of h_1. The final prediction value j are derived using:

$$j = Clip1_Y((j_1 + 512) >> 10) \quad (8\text{-}251)$$

– The final prediction values s and m are derived from s_1 and m_1 in the same manner as the derivation of b and h, as given by:

$$s = Clip1_Y((s_1 + 16) >> 5) \quad (8\text{-}252)$$
$$m = Clip1_Y((m_1 + 16) >> 5) \quad (8\text{-}253)$$

– The samples at quarter sample positions labelled as a, c, d, n, f, i, k, and q are derived by averaging with upward rounding of the two nearest samples at integer and half sample positions using:

$$a = (G + b + 1) >> 1 \quad (8\text{-}254)$$
$$c = (H + b + 1) >> 1 \quad (8\text{-}255)$$
$$d = (G + h + 1) >> 1 \quad (8\text{-}256)$$
$$n = (M + h + 1) >> 1 \quad (8\text{-}257)$$
$$f = (b + j + 1) >> 1 \quad (8\text{-}258)$$
$$i = (h + j + 1) >> 1 \quad (8\text{-}259)$$
$$k = (j + m + 1) >> 1 \quad (8\text{-}260)$$
$$q = (j + s + 1) >> 1 \quad (8\text{-}261)$$

– The samples at quarter sample positions labelled as e, g, p, and r are derived by averaging with upward rounding of the two nearest samples at half sample positions in the diagonal direction using:

$$e = (b + h + 1) >> 1 \quad (8\text{-}262)$$
$$g = (b + m + 1) >> 1 \quad (8\text{-}263)$$
$$p = (h + s + 1) >> 1 \quad (8\text{-}264)$$
$$r = (m + s + 1) >> 1 \quad (8\text{-}265)$$

The luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$) specifies which of the generated luma samples at full-sample and fractional-sample locations is assigned to the predicted luma sample value $predPartLX_L[x_L, y_L]$. This assignment is done according to Table 8-12. The value of $predPartLX_L[x_L, y_L]$ is the output.

Table 8-12 – Assignment of the luma prediction sample $predPartLX_L[x_L, y_L]$

$xFrac_L$	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
$yFrac_L$	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
$predPartLX_L[x_L, y_L]$	G	d	h	n	a	e	i	p	b	f	j	q	c	g	k	r

8.4.2.2.2 Chroma sample interpolation process

This process is only invoked when ChromaArrayType is equal to 1 or 2.

Inputs to this process are

– a chroma location in full-sample units ($xInt_C$, $yInt_C$),

– a chroma location offset in fractional-sample units ($xFrac_C$, $yFrac_C$), and

– chroma component samples from the selected reference picture $refPicLX_C$.

Output of this process is a predicted chroma sample value $predPartLX_C[x_C, y_C]$.

In Figure 8-5, the positions labelled with A, B, C, and D represent chroma samples at full-sample locations inside the given two-dimensional array $refPicLX_C$ of chroma samples.

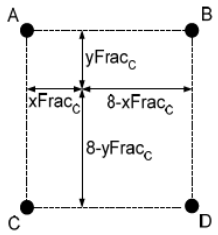


Figure 8-5 – Fractional sample position dependent variables in chroma interpolation and surrounding integer position samples A, B, C, and D

The variable $refPicHeightEffective_C$, which is the height of the effective reference picture chroma array, is derived as follows.

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, $refPicHeightEffective_C$ is set equal to $PicHeightInSamples_C$.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), $refPicHeightEffective_C$ is set equal to $PicHeightInSamples_C$ / 2.

The sample coordinates specified in Equations 8-266 through 8-273 are used for generating the predicted chroma sample value $predPartLX_C[\, x_C, y_C \,]$.

$$xA_C = Clip3(\, 0, PicWidthInSamples_C - 1, xInt_C \,) \quad (8\text{-}266)$$
$$xB_C = Clip3(\, 0, PicWidthInSamples_C - 1, xInt_C + 1 \,) \quad (8\text{-}267)$$
$$xC_C = Clip3(\, 0, PicWidthInSamples_C - 1, xInt_C \,) \quad (8\text{-}268)$$
$$xD_C = Clip3(\, 0, PicWidthInSamples_C - 1, xInt_C + 1 \,) \quad (8\text{-}269)$$

$$yA_C = Clip3(\, 0, refPicHeightEffective_C - 1, yInt_C \,) \quad (8\text{-}270)$$
$$yB_C = Clip3(\, 0, refPicHeightEffective_C - 1, yInt_C \,) \quad (8\text{-}271)$$
$$yC_C = Clip3(\, 0, refPicHeightEffective_C - 1, yInt_C + 1 \,) \quad (8\text{-}272)$$
$$yD_C = Clip3(\, 0, refPicHeightEffective_C - 1, yInt_C + 1 \,) \quad (8\text{-}273)$$

Given the chroma samples A, B, C, and D at full-sample locations specified in Equations 8-266 through 8-273, the predicted chroma sample value $predPartLX_C[\, x_C, y_C \,]$ is derived as follows:

$$predPartLX_C[\, x_C, y_C \,] = (\, (\, 8 - xFrac_C \,) * (\, 8 - yFrac_C \,) * A + xFrac_C * (\, 8 - yFrac_C \,) * B + (\, 8 - xFrac_C \,) * yFrac_C * C + xFrac_C * yFrac_C * D + 32 \,) >> 6 \quad (8\text{-}274)$$

8.4.2.3 Weighted sample prediction process

Inputs to this process are

- mbPartIdx: the current partition given by the partition index
- subMbPartIdx: the sub-macroblock partition index
- predFlagL0 and predFlagL1: prediction list utilization flags
- $predPartLX_L$: a (partWidth)x(partHeight) array of prediction luma samples (with LX being replaced by L0 or L1 depending on predFlagL0 and predFlagL1)
- when ChromaArrayType is not equal to 0, $predPartLX_{Cb}$ and $predPartLX_{Cr}$: (partWidthC)x(partHeightC) arrays of prediction chroma samples, one for each of the chroma components Cb and Cr (with LX being replaced by L0 or L1 depending on predFlagL0 and predFlagL1)
- variables for weighted prediction $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr

Outputs of this process are

- $predPart_L$: a (partWidth)x(partHeight) array of prediction luma samples and
- when ChromaArrayType is not equal to 0, $predPart_{Cb}$, and $predPart_{Cr}$: (partWidthC)x(partHeightC) arrays of prediction chroma samples, one for each of the chroma components Cb and Cr.

For macroblocks or partitions with predFlagL0 equal to 1 in P and SP slices, the following applies.

- If weighted_pred_flag is equal to 0, the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.
- Otherwise (weighted_pred_flag is equal to 1), the explicit weighted prediction process as described in subclause 8.4.2.3.2 is invoked with the same inputs and outputs as the process described in this subclause.

For macroblocks or partitions with predFlagL0 or predFlagL1 equal to 1 in B slices, the following applies.

- If weighted_bipred_idc is equal to 0, the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.
- Otherwise, if weighted_bipred_idc is equal to 1, the explicit weighted sample prediction process as described in subclause 8.4.2.3.2, for macroblocks or partitions with predFlagL0 or predFlagL1 equal to 1 with the same inputs and outputs as the process described in this subclause.
- Otherwise (weighted_bipred_idc is equal to 2), the following applies.
  - If predFlagL0 is equal to 1 and predFlagL1 is equal to 1, the implicit weighted sample prediction as described in subclause 8.4.2.3.2 is invoked with the same inputs and outputs as the process described in this subclause.

– Otherwise (predFlagL0 or predFlagL1 are equal to 1 but not both), the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.

8.4.2.3.1 Default weighted sample prediction process

Input to this process are the same as specified in subclause 8.4.2.3.

Output of this process are the same as specified in subclause 8.4.2.3.

Depending on the available component for which the prediction block is derived, the following applies.

– If the luma sample prediction values $predPart_L[x, y]$ are derived, the following applies with C set equal to L, x set equal to 0 .. partWidth - 1, and y set equal to 0 .. partHeight - 1.

– Otherwise, if the chroma Cb component sample prediction values $predPart_{Cb}[x, y]$ are derived, the following applies with C set equal to Cb, x set equal to 0 .. partWidthC - 1, and y set equal to 0 .. partHeightC - 1.

– Otherwise (the chroma Cr component sample prediction values $predPart_{Cr}[x, y]$ are derived), the following applies with C set equal to Cr, x set equal to 0 .. partWidthC - 1, and y set equal to 0 .. partHeightC - 1.

The prediction sample values are derived as follows.

– If predFlagL0 is equal to 1 and predFlagL1 is equal to 0

$$predPart_C[x, y] = predPartL0_C[x, y] \tag{8-275}$$

– Otherwise, if predFlagL0 is equal to 0 and predFlagL1 is equal to 1

$$predPart_C[x, y] = predPartL1_C[x, y] \tag{8-276}$$

– Otherwise (predFlagL0 and predFlagL1 are equal to 1),

$$predPart_C[x, y] = (predPartL0_C[x, y] + predPartL1_C[x, y] + 1) >> 1. \tag{8-277}$$

8.4.2.3.2 Weighted sample prediction process

Input to this process are the same as specified in subclause 8.4.2.3.

Output of this process are the same as specified in subclause 8.4.2.3.

Depending on the available component for which the prediction block is derived, the following applies.

– If the luma sample prediction values $predPart_L[x, y]$ are derived, the following applies with C set equal to L, x set equal to 0 .. partWidth - 1, and y set equal to 0 .. partHeight - 1.

– Otherwise, if the chroma Cb component sample prediction values $predPart_{Cb}[x, y]$ are derived, the following applies with C set equal to Cb, x set equal to 0 .. partWidthC - 1, and y set equal to 0 .. partHeightC - 1.

– Otherwise (the chroma Cr component sample prediction values $predPart_{Cr}[x, y]$ are derived), the following applies with C set equal to Cr, x set equal to 0 .. partWidthC - 1, and y set equal to 0 .. partHeightC - 1.

The prediction sample values are derived as follows.

– If the predFlagL0 is equal to 1 and predFlagL1 is equal to 0, the final predicted sample values $predPart_C[x, y]$ are derived by

$$\begin{aligned} &\text{if(} logWD_C >= 1 \text{)} \\ &\quad predPart_C[x, y] = Clip1_C(((predPartL0_C[x, y] * w_{0C} + 2^{logWD_C - 1}) >> logWD_C) + o_{0C}) \\ &\text{else} \\ &\quad predPart_C[x, y] = Clip1_C(predPartL0_C[x, y] * w_{0C} + o_{0C}) \end{aligned} \tag{8-278}$$

– Otherwise, if the predFlagL0 is equal to 0 and predFlagL1 is equal to 1, the final predicted sample values $predPart_C[x, y]$ are derived by

$$\begin{aligned} &\text{if}(logWD_C >= 1) \\ &\quad predPart_C[x, y] = Clip1_C(((predPartL1_C[x, y] * w_{1C} + 2^{logWD_C - 1}) >> logWD_C) + o_{1C}) \\ &\text{else} \\ &\quad predPart_C[x, y] = Clip1_C(predPartL1_C[x, y] * w_{1C} + o_{1C}) \end{aligned} \tag{8-279}$$

– Otherwise (both predFlagL0 and predFlagL1 are equal to 1), the final predicted sample values $predPart_C[x, y]$ are derived by

$$\begin{aligned} predPart_C[x, y] = Clip1_C(((predPartL0_C[x, y] * w_{0C} + predPartL1_C[x, y] * w_{1C} + 2^{logWD_C}) >> \\ (logWD_C + 1)) + ((o_{0C} + o_{1C} + 1) >> 1)) \end{aligned} \tag{8-280}$$

8.4.3 Derivation process for prediction weights

Inputs to this process are

– the reference indices refIdxL0 and refIdxL1

– the prediction utilization flags predFlagL0 and predFlagL1

Outputs of this process are

– variables for weighted prediction $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} with C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr

For C being replaced by L and, when ChromaArrayType is not equal to 0, Cb and Cr, the variables $logWD_C$, w_{0C}, w_{1C}, o_{0C}, o_{1C} are derived as follows.

– If weighted_bipred_idc is equal to 2 and the (slice_type % 5) is equal to 1, implicit mode weighted prediction is used as follows.

$$logWD_C = 5 \tag{8-281}$$

$$o_{0C} = 0 \tag{8-282}$$

$$o_{1C} = 0 \tag{8-283}$$

and w_{0C} and w_{1C} are derived as follows.

  – The variables currPicOrField, pic0, and pic1 are derived as follows:
    – If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies.
      – currPicOrField is the field of the current picture CurrPic that has the same parity as the current macroblock.
      – The variable pic0 is derived as follows.
        – If refIdxL0 % 2 is equal to 0, pic0 is the field of RefPicList0[refIdxL0 / 2] that has the same parity as the current macroblock.
        – Otherwise (refIdxL0 % 2 is not equal to 0), pic0 is the field of RefPicList0[refIdxL0 / 2] that has the opposite parity of the current macroblock.
      – The variable pic1 is derived as follows.
        – If refIdxL1 % 2 is equal to 0, pic1 is the field of RefPicList1[refIdxL1 / 2] that has the same parity as the current macroblock.
        – Otherwise (refIdxL1 % 2 is not equal to 0), pic1 is the field of RefPicList1[refIdxL1 / 2] that has the opposite parity of the current macroblock.
    – Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), currPicOrField is the current picture CurrPic, pic1 is RefPicList1[refIdxL1], and pic0 is RefPicList0[refIdxL0].

– The variables tb, td, tx, and DistScaleFactor are derived from the values of currPicOrField, pic0, pic1 using Equations 8-206, 8-207, 8-202, and 8-203, respectively.

– If DiffPicOrderCnt(pic1, pic0) is equal to 0 or one or both of pic1 and pic0 is marked as "used for long-term reference" or (DistScaleFactor >> 2) < -64 or (DistScaleFactor >> 2) > 128, w_{0C} and w_{1C} are derived as

$$w_{0C} = 32 \tag{8-284}$$

$$w_{1C} = 32 \tag{8-285}$$

- Otherwise,

$$w_{0C} = 64 - (\text{DistScaleFactor} >> 2) \tag{8-286}$$

$$w_{1C} = \text{DistScaleFactor} >> 2 \tag{8-287}$$

– Otherwise (weighted_pred_flag is equal to 1 in slices with (slice_type % 5) equal to 0 or 3 or weighted_bipred_idc equal to 1 in slices with (slice_type % 5) equal to 1), explicit mode weighted prediction is used as follows.

– The variables refIdxL0WP and refIdxL1WP are derived as follows.

– If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

$$\text{refIdxL0WP} = \text{refIdxL0} >> 1 \tag{8-288}$$

$$\text{refIdxL1WP} = \text{refIdxL1} >> 1 \tag{8-289}$$

– Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

$$\text{refIdxL0WP} = \text{refIdxL0} \tag{8-290}$$

$$\text{refIdxL1WP} = \text{refIdxL1} \tag{8-291}$$

– The variables logWD_C, w_{0C}, w_{1C}, o_{0C}, and o_{1C} are derived as follows.

– If C is equal to L for luma samples

$$\text{logWD}_C = \text{luma\_log2\_weight\_denom} \tag{8-292}$$

$$w_{0C} = \text{luma\_weight\_l0[refIdxL0WP]} \tag{8-293}$$

$$w_{1C} = \text{luma\_weight\_l1[refIdxL1WP]} \tag{8-294}$$

$$o_{0C} = \text{luma\_offset\_l0[refIdxL0WP]} * (1 << (\text{BitDepth}_Y - 8)) \tag{8-295}$$

$$o_{1C} = \text{luma\_offset\_l1[refIdxL1WP]} * (1 << (\text{BitDepth}_Y - 8)) \tag{8-296}$$

– Otherwise (C is equal to Cb or Cr for chroma samples, with iCbCr = 0 for Cb, iCbCr = 1 for Cr),

$$\text{logWD}_C = \text{chroma\_log2\_weight\_denom} \tag{8-297}$$

$$w_{0C} = \text{chroma\_weight\_l0[refIdxL0WP][iCbCr]} \tag{8-298}$$

$$w_{1C} = \text{chroma\_weight\_l1[refIdxL1WP][iCbCr]} \quad (8\text{-}299)$$

$$o_{0C} = \text{chroma\_offset\_l0[refIdxL0WP][iCbCr]} * (1 << (\text{BitDepth}_C - 8)) \quad (8\text{-}300)$$

$$o_{1C} = \text{chroma\_offset\_l1[refIdxL1WP][iCbCr]} * (1 << (\text{BitDepth}_C - 8)) \quad (8\text{-}301)$$

When explicit mode weighted prediction is used and predFlagL0 and predFlagL1 are equal to 1, the following constraint shall be obeyed for C equal to L and, when ChromaArrayType is not equal to 0, Cb and Cr

$$-128 <= w_{0C} + w_{1C} <= ((\text{logWD}_C == 7) ? 127 : 128) \quad (8\text{-}302)$$

NOTE – For implicit mode weighted prediction, weights w_{0C} and w_{1C} are each guaranteed to be in the range of -64..128 and the constraint expressed in Equation 8-302, although not explicitly imposed, will always be met. For explicit mode weighted prediction with logWD_C equal to 7, when one of the two weights w_{0C} or w_{1C} is inferred to be equal to 128 (as a consequence of luma_weight_l0_flag, luma_weight_l1_flag, chroma_weight_l0_flag, or chroma_weight_l1_flag equal to 0), the other weight (w_{1C} or w_{0C}) must have a negative value in order for the constraint expressed in Equation 8-302 to hold (and therefore the other flag luma_weight_l0_flag, luma_weight_l1_flag, chroma_weight_l0_flag, or chroma_weight_l1_flag must be equal to 1).

8.5 Transform coefficient decoding process and picture construction process prior to deblocking filter process

Inputs to this process are Intra16x16DCLevel (if available), Intra16x16ACLevel (if available), CbIntra16x16DCLevel (if available), CbIntra16x16ACLevel (if available), CrIntra16x16DCLevel (if available), CrIntra16x16ACLevel (if available), LumaLevel (if available), LumaLevel8x8 (if available), ChromaDCLevel (if available), ChromaACLevel (if available), CbLevel (if available), CrLevel (if available), CbLevel8x8 (if available), CrLevel8x8 (if available), and available Inter or Intra prediction sample arrays for the current macroblock for the applicable components pred_L, pred_{Cb}, or pred_{Cr}.

NOTE 1 – When decoding a macroblock in Intra_4x4 (or Intra_8x8) prediction mode, the luma component of the macroblock prediction array may not be complete, since for each 4x4 (or 8x8) luma block, the Intra_4x4 (or Intra_8x8) prediction process for luma samples as specified in subclause 8.3.1 (or 8.3.2) and the process specified in this subclause are iterated. When ChromaArrayType is equal to 3, the Cb and Cr component of the macroblock prediction array may not be complete for the same reason.

Outputs of this process are the constructed sample arrays prior to the deblocking filter process for the applicable components S'_L, S'_{Cb}, or S'_{Cr}.

NOTE 2 – When decoding a macroblock in Intra_4x4 (or Intra_8x8) prediction mode, the luma component of the macroblock constructed sample arrays prior to the deblocking filter process may not be complete, since for each 4x4 (or 8x8) luma block, the Intra_4x4 (or Intra_8x8) prediction process for luma samples as specified in subclause 8.3.1 (or 8.3.2) and the process specified in this subclause are iterated. When ChromaArrayType is equal to 3, the Cb and Cr component of the macroblock constructed sample arrays prior to the deblocking filter process may not be complete for the same reason.

This subclause specifies transform coefficient decoding and picture construction prior to the deblocking filter process.

When the current macroblock is coded as P_Skip or B_Skip, all values of LumaLevel, LumaLevel8x8, CbLevel, CbLevel8x8, CrLevel, CrLevel8x8, ChromaDCLevel, ChromaACLevel are set equal to 0 for the current macroblock.

8.5.1 Specification of transform decoding process for 4x4 luma residual blocks

This specification applies when transform_size_8x8_flag is equal to 0.

When the current macroblock prediction mode is not equal to Intra_16x16, the variable LumaLevel contains the levels for the luma transform coefficients. For a 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following ordered steps are specified.

1. The inverse transform coefficient scanning process as described in subclause 8.5.6 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the two-dimensional array c as the output.
2. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.11 is invoked with c as the input and r as the output.
3. When qpprime_y_zero_transform_bypass_flag is equal to 1, QP'_Y is equal to 0, the macroblock prediction mode is equal to Intra_4x4, and Intra4x4PredMode[luma4x4BlkIdx] is equal to 0 or 1, the intra residual transform-bypass decoding process as specified in subclause 8.5.14 is invoked with nW set equal to 4, nH set equal to 4, horPredFlag set equal to Intra4x4PredMode[luma4x4BlkIdx], and the 4x4 array r as the inputs, and the output is a modified version of the 4x4 array r.

4. The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

5. The 4x4 array u with elements u_{ij} for i, j = 0..3 is derived as

$$u_{ij} = \text{Clip1}_Y(\text{pred}_L[xO + j, yO + i] + r_{ij}) \quad (8\text{-}303)$$

When qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the bitstream shall not contain data that result in a value of u_{ij} as computed by Equation 8-303 that is not equal to $\text{pred}_L[xO + j, yO + i] + r_{ij}$.

6. The picture construction process prior to deblocking filter process in subclause 8.5.13 is invoked with u and luma4x4BlkIdx as the inputs.

8.5.2 Specification of transform decoding process for luma samples of Intra_16x16 macroblock prediction mode

When the current macroblock prediction mode is equal to Intra_16x16, the variables Intra16x16DCLevel and Intra16x16ACLevel contain the levels for the luma transform coefficients. The transform coefficient decoding proceeds in the following ordered steps:

1. The 4x4 luma DC transform coefficients of all 4x4 luma blocks of the macroblock are decoded.
   a. The inverse transform coefficient scanning process as described in subclause 8.5.6 is invoked with Intra16x16DCLevel as the input and the two-dimensional array c as the output.
   b. The scaling and transformation process for luma DC transform coefficients for Intra_16x16 macroblock type as specified in subclause 8.5.9 is invoked with BitDepth_Y, QP'_Y, and c as the input and dcY as the output.
2. The 16x16 array rMb is derived by processing the 4x4 luma blocks indexed by luma4x4BlkIdx = 0..15, and for each 4x4 luma block, the following ordered steps are specified.
   a. The variable lumaList, which is a list of 16 entries, is derived. The first entry of lumaList is the corresponding value from the array dcY. Figure 8-6 shows the assignment of the indices of the array dcY to the luma4x4BlkIdx. The two numbers in the small squares refer to indices i and j in dcY_{ij}, and the numbers in large squares refer to luma4x4BlkIdx.

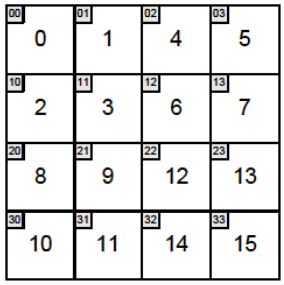


Figure 8-6 – Assignment of the indices of dcY to luma4x4BlkIdx

The elements in lumaList with index k = 1..15 are specified as

$$\text{lumaList}[k] = \text{Intra16x16ACLevel}[\text{luma4x4BlkIdx}][k - 1] \quad (8\text{-}304)$$

   b. The inverse transform coefficient scanning process as described in subclause 8.5.6 is invoked with lumaList as the input and the two-dimensional array c as the output.
   c. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.11 is invoked with c as the input and r as the output.

d. The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

e. The elements rMb[x, y] of the 16x16 array rMb with x = xO..xO + 3 and y = yO..yO + 3 are derived by

$$rMb[xO + j, yO + i] = r_{ij} \quad (8\text{-}305)$$

3. When qpprime_y_zero_transform_bypass_flag is equal to 1, QP'_Y is equal to 0, and Intra16x16PredMode is equal to 0 or 1, the intra residual transform-bypass decoding process as specified in subclause 8.5.14 is invoked with nW set equal to 16, nH set equal to 16, horPredFlag set equal to Intra16x16PredMode, and the 16x16 array rMb as the inputs, and the output is a modified version of the 16x16 array rMb.

4. The 16x16 array u with elements u_{ij} for i, j = 0..15 is derived as

$$u_{ij} = Clip1_Y(pred_L[j, i] + rMb[j, i]) \quad (8\text{-}306)$$

When qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the bitstream shall not contain data that result in a value of u_{ij} as computed by Equation 8-306 that is not equal to $pred_L[j, i] + rMb[j, i]$.

5. The picture construction process prior to deblocking filter process in subclause 8.5.13 is invoked with u as the input.

8.5.3 Specification of transform decoding process for 8x8 luma residual blocks

This specification applies when transform_size_8x8_flag is equal to 1.

The variable LumaLevel8x8[luma8x8BlkIdx] with luma8x8BlkIdx = 0..3 contains the levels for the luma transform coefficients for the luma 8x8 block with index luma8x8BlkIdx.

For an 8x8 luma block indexed by luma8x8BlkIdx = 0..3, the following ordered steps are specified.

1. The inverse scanning process for 8x8 luma transform coefficients as described in subclause 8.5.7 is invoked with LumaLevel8x8[luma8x8BlkIdx] as the input and the two-dimensional array c as the output.

2. The scaling and transformation process for residual 8x8 blocks as specified in subclause 8.5.12 is invoked with c as the input and r as the output.

3. When qpprime_y_zero_transform_bypass_flag is equal to 1, QP'_Y is equal to 0, the macroblock prediction mode is equal to Intra_8x8, and Intra8x8PredMode[luma8x8BlkIdx] is equal to 0 or 1, the intra residual transform-bypass decoding process as specified in subclause 8.5.14 is invoked with nW set equal to 8, nH set equal to 8, horPredFlag set equal to Intra8x8PredMode[luma8x8BlkIdx], and the 8x8 array r as the inputs, and the output is a modified version of the 8x8 array r.

4. The position of the upper-left sample of an 8x8 luma block with index luma8x8BlkIdx inside the macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.5 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO).

5. The 8x8 array u with elements u_{ij} for i, j = 0..7 is derived as

$$u_{ij} = Clip1_Y(pred_L[xO + j, yO + i] + r_{ij}) \quad (8\text{-}307)$$

When qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the bitstream shall not contain data that result in a value of u_{ij} as computed by Equation 8-307 that is not equal to $pred_L[xO + j, yO + i] + r_{ij}$.

6. The picture construction process prior to deblocking filter process in subclause 8.5.13 is invoked with u and luma8x8BlkIdx as the inputs.

8.5.4 Specification of transform decoding process for chroma samples

This process is invoked for each chroma component Cb and Cr separately when ChromaArrayType is not equal to 0.

Depending on ChromaArrayType, the following applies.

– If ChromaArrayType is equal to 3, the transform decoding process for chroma samples with ChromaArrayType equal to 3 as specified in subclause 8.5.5 is invoked.

– Otherwise (ChromaArrayType is not equal to 3), the following text specifies the transform decoding process for chroma samples.

For each chroma component, the variables ChromaDCLevel[iCbCr] and ChromaACLevel[iCbCr], with iCbCr set equal to 0 for Cb and iCbCr set equal to 1 for Cr, contain the levels for both components of the chroma transform coefficients.

Let the variable numChroma4x4Blks be set equal to (MbWidthC / 4) * (MbHeightC / 4).

For each chroma component, the transform decoding proceeds separately in the following ordered steps:

1. The numChroma4x4Blks chroma DC transform coefficients of the 4x4 chroma blocks of the component indexed by iCbCr of the macroblock are decoded.

   a. Depending on the variable ChromaArrayType, the following applies.

   – If ChromaArrayType is equal to 1, the 2x2 array c is derived using the inverse raster scanning process applied to ChromaDCLevel as follows

$$c = \begin{bmatrix} \text{ChromaDCLevel}[\text{iCbCr}][0] & \text{ChromaDCLevel}[\text{iCbCr}][1] \\ \text{ChromaDCLevel}[\text{iCbCr}][2] & \text{ChromaDCLevel}[\text{iCbCr}][3] \end{bmatrix} \tag{8-308}$$

   – Otherwise (ChromaArrayType is equal to 2), the 2x4 array c is derived using the inverse raster scanning process applied to ChromaDCLevel as follows

$$c = \begin{bmatrix} \text{ChromaDCLevel}[\text{iCbCr}][0] & \text{ChromaDCLevel}[\text{iCbCr}][2] \\ \text{ChromaDCLevel}[\text{iCbCr}][1] & \text{ChromaDCLevel}[\text{iCbCr}][5] \\ \text{ChromaDCLevel}[\text{iCbCr}][3] & \text{ChromaDCLevel}[\text{iCbCr}][6] \\ \text{ChromaDCLevel}[\text{iCbCr}][4] & \text{ChromaDCLevel}[\text{iCbCr}][7] \end{bmatrix} \tag{8-309}$$

   b. The scaling and transformation process for chroma DC transform coefficients as specified in subclause 8.5.10 is invoked with c as the input and dcC as the output.

2. The (MbWidthC)x(MbHeightC) array rMb is derived by processing the 4x4 chroma blocks indexed by chroma4x4BlkIdx = 0..numChroma4x4Blks – 1 of the component indexed by iCbCr, and for each 4x4 chroma block, the following ordered steps are specified.

   a. The variable chromaList, which is a list of 16 entries, is derived. The first entry of chromaList is the corresponding value from the array dcC. Figure 8-7 shows the assignment of the indices of the array dcC to the chroma4x4BlkIdx. The two numbers in the small squares refer to indices i and j in dcC_{ij}, and the numbers in large squares refer to chroma4x4BlkIdx.

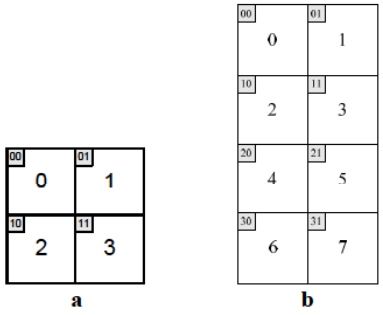


Figure 8-7 – Assignment of the indices of dcC to chroma4x4BlkIdx: (a) ChromaArrayType equal to 1, (b) ChromaArrayType equal to 2

The elements in chromaList with index k = 1..15 are specified as

$$chromaList[k] = ChromaACLevel[chroma4x4BlkIdx][k - 1] \quad (8\text{-}310)$$

b. The inverse scanning process for transform coefficients as specified in subclause 8.5.10 is invoked with chromaList as the input and the two-dimensional array c as the output.

c. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.11 is invoked with c as the input and r as the output.

d. The position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the macroblock is derived by

$$xO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 0) \quad (8\text{-}311)$$

$$yO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 1) \quad (8\text{-}312)$$

e. The elements rMb[x, y] of the (MbWidthC)x(MbHeightC) array rMb with x = xO..xO + 3 and y = yO..yO + 3 are derived by

$$rMb[xO + j, yO + i] = r_{ij} \quad (8\text{-}313)$$

3. When qpprime_y_zero_transform_bypass_flag is equal to 1, QP'_Y is equal to 0, the macroblock prediction mode is equal to Intra_4x4, Intra_8x8, or Intra_16x16, and intra_chroma_pred_mode is equal to 1 or 2, the intra residual transform-bypass decoding process as specified in subclause 8.5.14 is invoked with nW set equal to MbWidthC, nH set equal to MbHeightC, horPredFlag set equal to (intra_chroma_pred_mode – 1), and the (MbWidthC)x(MbHeightC) array rMb as the inputs, and the output is a modified version of the (MbWidthC)x(MbHeightC) array rMb.

4. The (MbWidthC)x(MbHeightC) array u with elements u_{ij} for i = 0..MbHeight – 1 and j = 0..MbWidth – 1 is derived as

$$u_{ij} = Clip1_C(pred_C[j, i] + rMb[j, i]) \quad (8\text{-}314)$$

When qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the bitstream shall not contain data that result in a value of u_{ij} as computed by Equation 8-314 that is not equal to $pred_C[j, i] + rMb[j, i]$.

5. The picture construction process prior to deblocking filter process in subclause 8.5.13 is invoked with u as the input.

8.5.5 Specification of transform decoding process for chroma samples with ChromaArrayType equal to 3

This process is invoked for each chroma component Cb and Cr separately when ChromaArrayType is equal to 3.

Depending on the macroblock prediction mode and transform_size_8x8_flag, the following applies.

– If the macroblock prediction mode is equal to Intra_16x16, the transform decoding process for Cb or Cr residual blocks shall be identical to the process described in subclause 8.5.2 when substituting luma with Cb or Cr, substituting Intra16x16DCLevel with CbIntra16x16DCLevel or CrIntra16x16DCLevel, substituting Intra16x16ACLevel with CbIntra16x16ACLevel or CrIntra16x16ACLevel, and substituting $pred_L$ with $pred_{Cb}$ or $pred_{Cr}$, substituting luma4x4BlkIdx with cb4x4BlkIdx or cr4x4BlkIdx, substituting lumaList with CbList or CrList, substituting $BitDepth_Y$ with $BitDepth_C$, substituting QP'_Y with QP'_C, and substituting $Clip1_Y$ with $Clip1_C$.

– Otherwise, if transform_size_8x8_flag is equal to 1, the transform decoding process for 8x8 Cb or 8x8 Cr residual blocks shall be identical to the process described in subclause 8.5.3 when substituting luma with Cb or Cr, substituting LumaLevel8x8 with CbLevel8x8 or CrLevel8x8, substituting $pred_L$ with $pred_{Cb}$ or $pred_{Cr}$, substituting luma8x8BlkIdx with cb8x8BlkIdx or cr8x8BlkIdx, and substituting $Clip1_Y$ with $Clip1_C$.

– Otherwise (the macroblock prediction mode is not equal to Intra_16x16 and transform_size_8x8_flag is equal to 0), the transform decoding process for 4x4 Cb or 4x4 Cr residual blocks shall be identical to the process described in subclause 8.5.1 when substituting luma with Cb or Cr, substituting LumaLevel with CbLevel or CrLevel, substituting $pred_L$ with $pred_{Cb}$ or $pred_{Cr}$, substituting luma4x4BlkIdx with cb4x4BlkIdx or cr4x4BlkIdx, and substituting $Clip1_Y$ with $Clip1_C$. During the scaling of 4x4 block transform coefficient levels that is specified within subclause 8.5.11, which is invoked as part of the process specified in subclause 8.5.1, the input 4x4 array c is

treated as not relating to a luma residual block coded using Intra_16x16 prediction mode and not relating to a chroma residual block.

8.5.6 Inverse scanning process for transform coefficients

Input to this process is a list of 16 values.

Output of this process is a variable c containing a two-dimensional array of 4x4 values. In the case of transform coefficients, these 4x4 values represent levels assigned to locations in the transform block. In the case of applying the inverse scanning process to a scaling list, the output variable c contains a two-dimensional array representing a 4x4 scaling matrix.

The inverse scanning process for transform coefficients maps the sequence of transform coefficient levels to the transform coefficient level positions. Table 8-13 specifies the two mappings: inverse zig-zag scan and inverse field scan. The inverse zig-zag scan is used for transform coefficients in frame macroblocks and the inverse field scan is used for transform coefficients in field macroblocks.

The inverse scanning process for scaling lists maps the sequence of scaling list entries to the positions in the corresponding scaling matrix. For this mapping, the inverse zig-zag scan is used.

Figure 8-8 illustrates the scans.

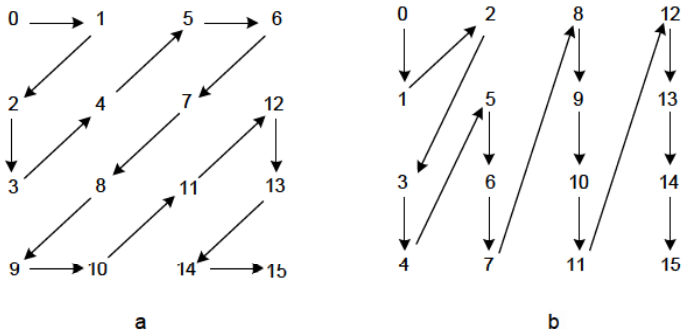


Figure 8-8 – 4x4 block scans. (a) Zig-zag scan. (b) Field scan (informative)

Table 8-13 provides the mapping from the index idx of input list of 16 elements to indices i and j of the two-dimensional array c.

Table 8-13 – Specification of mapping of idx to c_{ij} for zig-zag and field scan

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
zig-zag	c_{00}	c_{01}	c_{10}	c_{20}	c_{11}	c_{02}	c_{03}	c_{12}	c_{21}	c_{30}	c_{31}	c_{22}	c_{13}	c_{23}	c_{32}	c_{33}
field	c_{00}	c_{10}	c_{01}	c_{20}	c_{30}	c_{11}	c_{21}	c_{31}	c_{02}	c_{12}	c_{22}	c_{32}	c_{03}	c_{13}	c_{23}	c_{33}

8.5.7 Inverse scanning process for 8x8 transform coefficients

Input to this process is a list of 64 values.

Output of this process is a variable c containing a two-dimensional array of 8x8 values. In the case of transform coefficients, these 8x8 values represent levels assigned to locations in the transform block. In the case of applying the inverse scanning process to a scaling list, the output variable c contains a two-dimensional array representing an 8x8 scaling matrix.

The inverse scanning process for transform coefficients maps the sequence of transform coefficient levels to the transform coefficient level positions. Table 8-14 specifies the two mappings: inverse 8x8 zig-zag scan and inverse 8x8

field scan. The inverse 8x8 zig-zag scan is used for transform coefficients in frame macroblocks and the inverse 8x8 field scan is used for transform coefficients in field macroblocks.

The inverse scanning process for scaling lists maps the sequence of scaling list entries to the positions in the corresponding scaling matrix. For this mapping, the inverse zig-zag scan is used.

Figure 8-9 illustrates the scans.

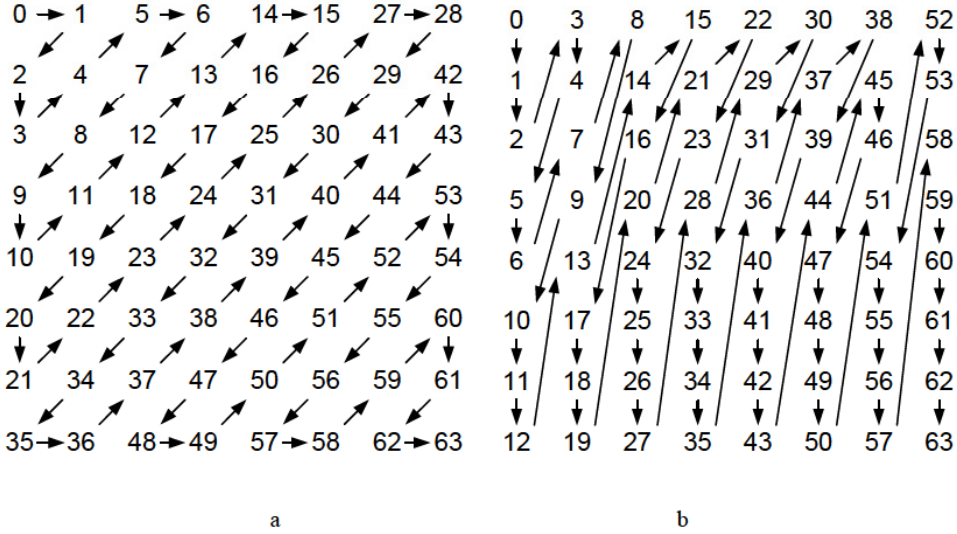


Figure 8-9 8x8 block scans. (a) 8x8 zig-zag scan. (b) 8x8 field scan (informative)

Table 8-14 provides the mapping from the index idx of the input list of 64 elements to indices i and j of the two-dimensional array c.

Table 8-14 – Specification of mapping of idx to c_{ij} for 8x8 zig-zag and 8x8 field scan

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
zig-zag	c_{00}	c_{01}	c_{10}	c_{20}	c_{11}	c_{02}	c_{03}	c_{12}	c_{21}	c_{30}	c_{40}	c_{31}	c_{22}	c_{13}	c_{04}	c_{05}
field	c_{00}	c_{10}	c_{20}	c_{01}	c_{11}	c_{30}	c_{40}	c_{21}	c_{02}	c_{31}	c_{50}	c_{60}	c_{70}	c_{41}	c_{12}	c_{03}

Table 8-14 (continued) – Specification of mapping of idx to cij for 8x8 zig-zag and 8x8 field scan

idx	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
zig-zag	c_{14}	c_{23}	c_{32}	c_{41}	c_{50}	c_{60}	c_{51}	c_{42}	c_{33}	c_{24}	c_{15}	c_{06}	c_{07}	c_{16}	c_{25}	c_{34}
field	c_{22}	c_{51}	c_{61}	c_{71}	c_{32}	c_{13}	c_{04}	c_{23}	c_{42}	c_{52}	c_{62}	c_{72}	c_{33}	c_{14}	c_{05}	c_{24}

Table 8-14 (continued) – Specification of mapping of idx to cij for 8x8 zig-zag and 8x8 field scan

idx	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
zig-zag	c_{43}	c_{52}	c_{61}	c_{70}	c_{71}	c_{62}	c_{53}	c_{44}	c_{35}	c_{26}	c_{17}	c_{27}	c_{36}	c_{45}	c_{54}	c_{63}
field	c_{43}	c_{53}	c_{63}	c_{73}	c_{34}	c_{15}	c_{06}	c_{25}	c_{44}	c_{54}	c_{64}	c_{74}	c_{35}	c_{16}	c_{26}	c_{45}

Table 8-14 (concluded) – Specification of mapping of idx to cij for 8x8 zig-zag and 8x8 field scan

idx	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
zig-zag	c_{72}	c_{73}	c_{64}	c_{55}	c_{46}	c_{37}	c_{47}	c_{56}	c_{65}	c_{74}	c_{75}	c_{66}	c_{57}	c_{67}	c_{76}	c_{77}
field	c_{55}	c_{65}	c_{75}	c_{36}	c_{07}	c_{17}	c_{46}	c_{56}	c_{66}	c_{76}	c_{27}	c_{37}	c_{47}	c_{57}	c_{67}	c_{77}

8.5.8 Derivation process for the chroma quantisation parameters and scaling function

Outputs of this process are:

– QP_C: the chroma quantisation parameter for each chroma component Cb and Cr

– QS_C: the additional chroma quantisation parameter for each chroma component Cb and Cr required for decoding SP and SI slices (if applicable)

NOTE 1 – QP quantisation parameter values QP_Y and QS_Y are always in the range of $-QpBdOffset_Y$ to 51, inclusive. QP quantisation parameter values QP_C and QS_C are always in the range of $-QpBdOffset_C$ to 51, inclusive.

The value of QP_C for a chroma component is determined from the current value of QP_Y and the value of chroma_qp_index_offset (for Cb) or second_chroma_qp_index_offset (for Cr).

NOTE 2 – The scaling equations are specified such that the equivalent transform coefficient level scaling factor doubles for every increment of 6 in QP_Y. Thus, there is an increase in the factor used for scaling of approximately 12 % for each increase of 1 in the value of QP_Y.

The value of QP_C for each chroma component is determined as specified in Table 8-15 based on the index denoted as qP_I.

The variable qP_{Offset} for each chroma component is derived as follows.

– If the chroma component is the Cb component, qP_{Offset} is specified as

$$qP_{Offset} = \text{chroma\_qp\_index\_offset} \tag{8-315}$$

– Otherwise (the chroma component is the Cr component), qP_{Offset} is specified as

$$qP_{Offset} = \text{second\_chroma\_qp\_index\_offset} \tag{8-316}$$

The value of qP_I for each chroma component is derived as

$$qP_I = \text{Clip3}(-QpBdOffset_C, 51, QP_Y + qP_{Offset}) \tag{8-317}$$

The value of QP'_C for the chroma components is derived as

$$QP'_C = QP_C + QpBdOffset_C \tag{8-318}$$

Table 8-15 – Specification of QP_C as a function of qP_I

qP_I	<30	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
QP_C	$=qP_I$	29	30	31	32	32	33	34	34	35	35	36	36	37	37	37	38	38	38	39	39	39	39

When the current slice is an SP or SI slice, QS_C is derived using the above process, substituting QP_Y with QS_Y and QP_C with QS_C.

The 4x4 matrix weightScale(i, j) is specified as follows.

– The variable mbIsInterFlag is derived as follows.

  – If the current macroblock is coded using Inter macroblock prediction modes, mbIsInterFlag is set equal to 1.

  – Otherwise (the current macroblock is coded using Intra macroblock prediction modes), mbIsInterFlag is set equal to 0.

– The variable iYCbCr derived as follows.
  – If separate_colour_plane_flag is equal to 1, iYCbCr is set equal to colour_plane_id.
  – Otherwise (separate_colour_plane_flag is equal to 0), the following applies.
    – If the input array c relates to a luma residual block, iYCbCr is set equal to 0.
    – Otherwise, if the input array c relates to a chroma residual block and the chroma component is equal to Cb, iYCbCr is set equal to 1.
    – Otherwise (the input array c relates to a chroma residual block and the chroma component is equal to Cr), iYCbCr is set equal to 2.
– The inverse scanning process for transform coefficients as specified in subclause 8.5.6 is invoked with ScalingList4x4[iYCbCr + ((mbIsInterFlag == 1) ? 3 : 0)] as the input and the output is assigned to the 4x4 matrix weightScale.

LevelScale(m, i, j) is specified by

$$\text{LevelScale}(m, i, j) = \text{weightScale}(i, j) * \text{normAdjust}(m, i, j) \tag{8-319}$$

where

$$\text{normAdjust}(m,i,j) = \begin{cases} v_{m0} & \text{for } (i\,\%\,2, j\,\%\,2) \text{ equal to } (0,0), \\ v_{m1} & \text{for } (i\,\%\,2, j\,\%\,2) \text{ equal to } (1,1), \\ v_{m2} & \text{otherwise;} \end{cases} \tag{8-320}$$

where the first and second subscripts of v are row and column indices, respectively, of the matrix specified as:

$$v = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}. \tag{8-321}$$

The 8x8 matrix weightScale8x8(i, j) is specified as follows.

– The variable mbIsInterFlag is derived as follows.
  – If the current macroblock is coded using Inter macroblock prediction modes, mbIsInterFlag is set equal to 1.
  – Otherwise (the current macroblock is coded using Intra macroblock prediction modes), mbIsInterFlag is set equal to 0.
– The variable iYCbCr derived as follows.
  – If separate_colour_plane_flag is equal to 1, iYCbCr is set equal to colour_plane_id.
  – Otherwise (separate_colour_plane_flag is equal to 0), the following applies.
    – If the input array c relates to a luma residual block, iYCbCr is set equal to 0.
    – Otherwise, if the input array c relates to a chroma residual block and the chroma component is equal to Cb, iYCbCr is set equal to 1.
    – Otherwise (the input array c relates to a chroma residual block and the chroma component is equal to Cr), iYCbCr is set equal to 2.
– The inverse scanning process for 8x8 transform coefficients as specified in subclause 8.5.7 is invoked with ScalingList8x8[mbIsInterFlag] as the input and the output is assigned to the 8x8 matrix weightScale8x8.

LevelScale(m, i, j) is specified by

$$\text{LevelScale8x8}(m, i, j) = \text{weightScale8x8}(i, j) * \text{normAdjust8x8}(m, i, j) \tag{8-322}$$

where

$$\text{normAdjust8x8}(m,i,j) = \begin{cases} v_{m0} & \text{for } (i\,\%\,4, j\,\%\,4) \text{ equal to } (0,0), \\ v_{m1} & \text{for } (i\,\%\,2, j\,\%\,2) \text{ equal to } (1,1), \\ v_{m2} & \text{for } (i\,\%\,4, j\,\%\,4) \text{ equal to } (2,2), \\ v_{m3} & \text{for } (i\,\%\,4, j\,\%\,2) \text{ equal to } (0,1) \text{ or } (i\,\%\,2, j\,\%\,4) \text{ equal to } (1,0), \\ v_{m4} & \text{for } (i\,\%\,4, j\,\%\,4) \text{ equal to } (0,2) \text{ or } (i\,\%\,4, j\,\%\,4) \text{ equal to } (2,0), \\ v_{m5} & \text{otherwise;} \end{cases} \tag{8-323}$$

where the first and second subscripts of v are row and column indices, respectively, of the matrix specified as:

$$v = \begin{bmatrix} 20 & 18 & 32 & 19 & 25 & 24 \\ 22 & 19 & 35 & 21 & 28 & 26 \\ 26 & 23 & 42 & 24 & 33 & 31 \\ 28 & 25 & 45 & 26 & 35 & 33 \\ 32 & 28 & 51 & 30 & 40 & 38 \\ 36 & 32 & 58 & 34 & 46 & 43 \end{bmatrix}. \tag{8-324}$$

8.5.9 Scaling and transformation process for DC transform coefficients for Intra_16x16 macroblock type

Inputs to this process are

– the variables bitDepth and qP,

– transform coefficient level values for DC transform coefficients of Intra_16x16 macroblocks as a 4x4 array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are 16 scaled DC values for 4x4 blocks of Intra_16x16 macroblocks as a 4x4 array dcY with elements dcY_{ij}.

Depending on the values of qpprime_y_zero_transform_bypass_flag and QP'_Y, the following applies.

– If qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the output dcY is derived as

$$dcY_{ij} = c_{ij} \quad \text{with} \quad i, j = 0..3 \tag{8-325}$$

– Otherwise (qpprime_y_zero_transform_bypass_flag is equal to 0 or QP'_Y is not equal to 0), the following text of this process specifies the output.

The inverse transform for the 4x4 luma DC transform coefficients is specified by:

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}. \tag{8-326}$$

The bitstream shall not contain data that results in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7+\text{bitDepth})}$ to $2^{(7+\text{bitDepth})}-1$, inclusive.

After the inverse transform, the scaling is performed as follows.

– If qP is greater than or equal to 36, the scaled result is derived as

$$dcY_{ij} = (f_{ij} * \text{LevelScale}(qP \,\%\, 6, 0, 0)) << (qP / 6 - 6), \quad \text{with} \quad i, j = 0...3 \tag{8-327}$$

– Otherwise (qP is less than 36), the scaled result is derived as

$$dcY_{ij} = (f_{ij} * \text{LevelScale}(qP \,\%\, 6, 0, 0) + (1 << (5 - qP / 6))) >> (6 - qP / 6), \quad \text{with} \quad i, j = 0...3 \tag{8-328}$$

The bitstream shall not contain data that results in any element dcY_{ij} of dcY with i, j = 0..3 that exceeds the range of integer values from $-2^{(7+bitDepth)}$ to $2^{(7+bitDepth)}-1$, inclusive.

NOTE 1 – When entropy_coding_mode_flag is equal to 0 and qP is less than 10 and profile_idc is equal to 66, 77, or 88, the range of values that can be represented for the elements c_{ij} of c is not sufficient to represent the full range of values of the elements dcY_{ij} of dcY that could be necessary to form a close approximation of the content of any possible source picture by use of the Intra_16x16 macroblock type.

NOTE 2 – Since the range limit imposed on the elements dcY_{ij} of dcY is imposed after the right shift in Equation 8-328, a larger range of values must be supported in the decoder prior to the right shift.

8.5.10 Scaling and transformation process for chroma DC transform coefficients

This process is only invoked when ChromaArrayType is equal to 1 or 2.

Inputs to this process are transform coefficient level values for chroma DC transform coefficients of one chroma component of the macroblock as an (MbWidthC / 4)x(MbIIeightC / 4) array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are the scaled DC values as an (MbWidthC / 4)x(MbHeightC / 4) array dcC with elements dcC_{ij}.

The variables bitDepth and qP are set equal to $BitDepth_C$ and QP'_C, respectively.

Depending on the values of qpprime_y_zero_transform_bypass_flag and QP'_Y, the following applies.

– If qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the output dcC is derived as

$$dcC_{ij} = c_{ij} \text{ with } i = 0..(MbWidthC / 4) - 1 \text{ and } j = 0..(MbHeightC / 4) - 1. \tag{8-329}$$

– Otherwise (qpprime_y_zero_transform_bypass_flag is equal to 0 or QP'_Y is not equal to 0), the following applies.

  – The transformation process for chroma DC transform coefficients as specified in subclause 8.5.10.1 is invoked with bitDepth and c as the inputs and the output is assigned to the (MbWidthC / 4)x(MbHeightC / 4) array f of chroma DC values with elements f_{ij}.

  – The scaling process for chroma DC transform coefficients as specified in subclause 8.5.10.2 is invoked with bitDepth, qP, and f as the inputs and the output is assigned to the (MbWidthC / 4)x(MbHeightC / 4) array dcY of scaled chroma DC values with elements dcY_{ij}.

8.5.10.1 Transformation process for chroma DC transform coefficients

Inputs of this process are transform coefficient level values for chroma DC transform coefficients of one chroma component of the macroblock as an (MbWidthC / 4)x(MbHeightC / 4) array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are the DC values as an (MbWidthC / 4)x(MbHeightC / 4) array f with elements f_{ij}.

Depending on the variable ChromaArrayType, the inverse transform is specified as follows.

– If ChromaArrayType is equal to 1, the inverse transform for the 2x2 chroma DC transform coefficients is specified as

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{8-330}$$

– Otherwise, (ChromaArrayType is equal to 2), the inverse transform for the 2x4 chroma DC transform coefficients is specified as

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \\ c_{20} & c_{21} \\ c_{30} & c_{31} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{8-331}$$

8.5.10.2 Scaling process for chroma DC transform coefficients

Inputs of this process are

– the variables bitDepth and qP

– DC values as an (MbWidthC / 4)x(MbHeightC / 4) array f with elements f_{ij}

Outputs of this process are scaled DC values as an (MbWidthC / 4)x(MbHeightC / 4) array dcC with elements dcC_{ij}.

The bitstream shall not contain data that results in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7+\text{bitDepth})}$ to $2^{(7+\text{bitDepth})}-1$, inclusive.

Scaling is performed depending on the variable ChromaArrayType as follows.

– If ChromaArrayType is equal to 1, the scaled result is derived as

$$dcC_{ij} = ((f_{ij} * \text{LevelScale}(qP \% 6, 0, 0)) << (qP/6)) >> 5, \quad with \quad i, j = 0, 1 \tag{8-332}$$

– Otherwise (ChromaArrayType is equal to 2), the following applies.

– The variable qP_{DC} is derived as

$$qP_{DC} = qP + 3 \tag{8-333}$$

– Depending on the value of qP_{DC}, the following applies.

– If qP_{DC} is greater than or equal to 36, the scaled result is derived as

$$dcC_{ij} = (f_{ij} * \text{LevelScale}(qP_{DC} \% 6, 0, 0)) << (qP_{DC} / 6 - 6), \quad \text{with } i = 0..3, j = 0, 1 \tag{8-334}$$

– Otherwise (qP_{DC} is less than 36), the scaled result is derived as

$$dcC_{ij} = (f_{ij} * \text{LevelScale}(qP_{DC} \% 6, 0, 0) + 2^{5-qP_{DC}/6}) >> (6 - qP_{DC} / 6), \quad \text{with } i = 0..3, j = 0,1 \tag{8-335}$$

The bitstream shall not contain data that results in any element dcC_{ij} of dcC with i, j = 0..3 that exceeds the range of integer values from $-2^{(7+\text{bitDepth})}$ to $2^{(7+\text{bitDepth})}-1$, inclusive.

NOTE 1 – When entropy_coding_mode_flag is equal to 0 and qP is less than 4 and profile_idc is equal to 66, 77, or 88, the range of values that can be represented for the elements c_{ij} of c in subclause 8.5.10.1 may not be sufficient to represent the full range of values of the elements dcC_{ij} of dcC that could be necessary to form a close approximation of the content of any possible source picture.

NOTE 2 – Since the range limit imposed on the elements dcC_{ij} of dcC is imposed after the right shift in Equation 8-332 or 8-335, a larger range of values must be supported in the decoder prior to the right shift.

8.5.11 Scaling and transformation process for residual 4x4 blocks

Input to this process is a 4x4 array c with elements c_{ij} which is either an array relating to a residual block of the luma component or an array relating to a residual block of a chroma component.

Outputs of this process are residual sample values as 4x4 array r with elements r_{ij}.

The variable bitDepth is derived as follows.

– If the input array c relates to a luma residual block, bitDepth is set equal to BitDepth_Y.

– Otherwise (the input array c relates to a chroma residual block), bitDepth is set equal to BitDepth_C.

The variable sMbFlag is derived as follows.

- If mb_type is equal to SI or the macroblock prediction mode is equal to Inter in an SP slice, sMbFlag is set equal to 1,
- Otherwise (mb_type not equal to SI and the macroblock prediction mode is not equal to Inter in an SP slice), sMbFlag is set equal to 0.

The variable qP is derived as follows.

– If the input array c relates to a luma residual block and sMbFlag is equal to 0

$$qP = QP'_Y \tag{8-336}$$

– Otherwise, if the input array c relates to a luma residual block and sMbFlag is equal to 1

$$qP = QS_Y \tag{8-337}$$

– Otherwise, if the input array c relates to a chroma residual block and sMbFlag is equal to 0

$$qP = QP'_C \tag{8-338}$$

– Otherwise (the input array c relates to a chroma residual block and sMbFlag is equal to 1),

$$qP = QS_C \tag{8-339}$$

Depending on the values of qpprime_y_zero_transform_bypass_flag and QP'_Y, the following applies.

– If qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the output r is derived as

$$r_{ij} = c_{ij} \text{ with } i, j = 0..3 \tag{8-340}$$

– Otherwise (qpprime_y_zero_transform_bypass_flag is equal to 0 or QP'_Y is not equal to 0), the following applies.

  – The scaling process for residual 4x4 blocks as specified in subclause 8.5.11.1 is invoked with bitDepth, qP, and c as the inputs and the output is assigned to the 4x4 array d of scaled transform coefficients with elements d_{ij}.

  – The transformation process for residual 4x4 blocks as specified in subclause 8.5.11.2 is invoked with bitDepth and d as the inputs and the output is assigned to the 4x4 array r of residual sample values with elements r_{ij}.

8.5.11.1 Scaling process for residual 4x4 blocks

Inputs of this process are

– the variables bitDepth and qP

– a 4x4 array c with elements c_{ij} which is either an array relating to a residual block of luma component or an array relating to a residual block of a chroma component

Output of this process is a 4x4 array of scaled transform coefficients d with elements d_{ij}.

The bitstream shall not contain data that results in any element c_{ij} of c with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})}-1$, inclusive.

Scaling of 4x4 block transform coefficient levels c_{ij} proceeds as follows.

– If all of the following conditions are true

  – i is equal to 0

  – j is equal to 0

  – c relates to a luma residual block coded using Intra_16x16 prediction mode or c relates to a chroma residual block

  the variable d_{00} is derived by

$$d_{00} = c_{00} \tag{8-341}$$

  – Otherwise, the following applies.

    If qP is greater than or equal to 24, the scaled result is derived as follows

$$d_{ij} = (c_{ij} * \text{LevelScale}(qP \% 6, i, j)) << (qP / 6 - 4), \text{ with } i,j = 0..3 \text{ except as noted above} \tag{8-342}$$

  – Otherwise (qP is less than 24), the scaled result is derived as follows

$$d_{ij} = (c_{ij} * \text{LevelScale}(qP \% 6, i, j) + 2^{3-qP/6}) >> (4 - qP / 6), \text{ with } i, j = 0..3 \text{ except as noted above} \tag{8-343}$$

The bitstream shall not contain data that results in any element d_{ij} of d with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

8.5.11.2 Transformation process for residual 4x4 blocks

Inputs of this process are

– the variable bitDepth

– a 4x4 array of scaled transform coefficients d with elements d_{ij}

Outputs of this process are residual sample values as 4x4 array r with elements r_{ij}.

The bitstream shall not contain data that results in any element d_{ij} of d with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)} - 1$, inclusive.

The transform process shall convert the block of scaled transform coefficients to a block of output samples in a manner mathematically equivalent to the following.

First, each (horizontal) row of scaled transform coefficients is transformed using a one-dimensional inverse transform as follows.

A set of intermediate values is computed as follows.

$$e_{i0} = d_{i0} + d_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-344}$$

$$e_{i1} = d_{i0} - d_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-345}$$

$$e_{i2} = (d_{i1} >> 1) - d_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-346}$$

$$e_{i3} = d_{i1} + (d_{i3} >> 1), \quad \text{with} \quad i = 0..3 \tag{8-347}$$

The bitstream shall not contain data that results in any element e_{ij} of e with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)} - 1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows.

$$f_{i0} = e_{i0} + e_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-348}$$

$$f_{i1} = e_{i1} + e_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-349}$$

$$f_{i2} = e_{i1} - e_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-350}$$

$$f_{i3} = e_{i0} - e_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-351}$$

The bitstream shall not contain data that results in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)} - 1$, inclusive.

Then, each (vertical) column of the resulting matrix is transformed using the same one-dimensional inverse transform as follows.

A set of intermediate values is computed as follows.

$$g_{0j} = f_{0j} + f_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-352}$$

$$g_{1j} = f_{0j} - f_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-353}$$

$$g_{2j} = (f_{1j} >> 1) - f_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-354}$$

$$g_{3j} = f_{1j} + (f_{3j} >> 1), \quad \text{with} \quad j = 0..3 \tag{8-355}$$

The bitstream shall not contain data that results in any element g_{ij} of g with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)} - 1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows.

$$h_{0j} = g_{0j} + g_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-356}$$

$$h_{1j} = g_{1j} + g_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-357}$$

$$h_{2j} = g_{1j} - g_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-358}$$

$$h_{3j} = g_{0j} - g_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-359}$$

The bitstream shall not contain data that results in any element h_{ij} of h with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)} - 33$, inclusive.

After performing both the one-dimensional horizontal and the one-dimensional vertical inverse transforms to produce an array of transformed samples, the final constructed residual sample values is derived as

$$r_{ij} = (h_{ij} + 2^5) >> 6 \quad \text{with} \quad i, j = 0..3 \tag{8-360}$$

8.5.12 Scaling and transformation process for residual 8x8 blocks

Input to this process is an 8x8 array c with elements c_{ij} which is either an array relating to an 8x8 residual block of the luma component or, when ChromaArrayType is equal to 3, an array relating to an 8x8 residual block of a chroma component.

NOTE 1 – When separate_colour_plane_flag is equal to 1, all residual blocks are considered to be associated with the luma component for purposes of the decoding process of each coded picture (prior to the final assignment of the decoded picture to a particular luma or chroma picture array according to the value of colour_plane_id).

Outputs of this process are residual sample values as 8x8 array r with elements r_{ij}.

The variables bitDepth and qP are derived as follows.

- If the input array c relates to a luma residual block, bitDepth is set equal to $BitDepth_Y$ and QP is set equal to QP'_Y.
- Otherwise (the input array c relates to a chroma residual block), bitDepth is set equal to $BitDepth_C$ and QP is set equal to QP'_C.

NOTE 2 – When separate_colour_plane_flag is equal to 1, all residual blocks are considered to be associated with the luma component for purposes of the decoding process of each colour component of a picture.

Depending on the values of qpprime_y_zero_transform_bypass_flag and QP'_Y, the following applies.

- If qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the output r is derived as

$$r_{ij} = c_{ij} \quad \text{with} \quad i, j = 0..7 \tag{8-361}$$

- Otherwise (qpprime_y_zero_transform_bypass_flag is equal to 0 or QP'Y is not equal to 0), the following applies.
  - The scaling process for residual 8x8 blocks as specified in subclause 8.5.12.1 is invoked with bitDepth, qP, and c as the inputs and the output is assigned to the 8x8 array d of scaled transform coefficients with elements d_{ij}.
  - The transformation process for residual 8x8 blocks as specified in subclause 8.5.12.2 is invoked with bitDepth and d as the inputs and the output is assigned to the 8x8 array r of residual sample values with elements r_{ij}.

8.5.12.1 Scaling process for residual 8x8 blocks

Inputs of this process are

- the variables bitDepth and qP
- an 8x8 array c with elements c_{ij} which is either an array relating to a residual block of luma component or an array relating to a residual block of a chroma component

Output of this process is an 8x8 array of scaled transform coefficients d with elements d_{ij}.

The bitstream shall not contain data that results in any element c_{ij} of c with i, j = 0..7 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)}-1$, inclusive.

The scaling process for 8x8 block transform coefficient levels c_{ij} proceeds as follows.

– If qP is greater than or equal to 36, the scaled result is derived as

$$d_{ij} = (c_{ij} * LevelScale8x8(qP \% 6, i, j)) << (qP / 6 - 6), \text{ with } i,j = 0..7 \quad (8\text{-}362)$$

– Otherwise (qP is less than 36), the scaled result is derived as

$$d_{ij} = (c_{ij} * LevelScale8x8(qP \% 6, i, j)) + 2^{5-QP'_Y/6}) >> (6 - qP /6), \text{ with } i,j = 0..7 \quad (8\text{-}363)$$

The bitstream shall not contain data that results in any element d_{ij} of d with i, j = 0..7 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)}-1$, inclusive.

8.5.12.2 Transformation process for residual 8x8 blocks

Inputs of this process are

– the variable bitDepth

– an 8x8 array of scaled transform coefficients d with elements d_{ij}

Outputs of this process are residual sample values as 8x8 array r with elements r_{ij}.

The bitstream shall not contain data that results in any element d_{ij} of d with i, j = 0..7 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)} - 1$, inclusive.

The transform process shall convert the block of scaled transform coefficients to a block of output samples in a manner mathematically equivalent to the following.

First, each (horizontal) row of scaled transform coefficients is transformed using a one-dimensional inverse transform as follows.

– A set of intermediate values e_{ij} is derived by

$$e_{i0} = d_{i0} + d_{i4}, \text{ with } i = 0..7 \quad (8\text{-}364)$$

$$e_{i1} = - d_{i3} + d_{i5} - d_{i7} - (d_{i7} >> 1), \text{ with } i = 0..7 \quad (8\text{-}365)$$

$$e_{i2} = d_{i0} - d_{i4}, \text{ with } i = 0..7 \quad (8\text{-}366)$$

$$e_{i3} = d_{i1} + d_{i7} - d_{i3} - (d_{i3} >> 1), \text{ with } i = 0..7 \quad (8\text{-}367)$$

$$e_{i4} = (d_{i2} >> 1) - d_{i6}, \text{ with } i = 0..7 \quad (8\text{-}368)$$

$$e_{i5} = - d_{i1} + d_{i7} + d_{i5} + (d_{i5} >> 1), \text{ with } i = 0..7 \quad (8\text{-}369)$$

$$e_{i6} = d_{i2} + (d_{i6} >> 1), \text{ with } i = 0..7 \quad (8\text{-}370)$$

$$e_{i7} = d_{i3} + d_{i5} + d_{i1} + (d_{i1} >> 1), \text{ with } i = 0..7 \quad (8\text{-}371)$$

– A second set of intermediate results f_{ij} is computed from the intermediate values e_{ij} as

$$f_{i0} = e_{i0} + e_{i6}, \text{ with } i = 0..7 \quad (8\text{-}372)$$

$$f_{i1} = e_{i1} + (e_{i7} >> 2), \text{ with } i = 0..7 \quad (8\text{-}373)$$

$$f_{i2} = e_{i2} + e_{i4}, \text{ with } i = 0..7 \tag{8-374}$$

$$f_{i3} = e_{i3} + (e_{i5} >> 2), \text{ with } i = 0..7 \tag{8-375}$$

$$f_{i4} = e_{i2} - e_{i4}, \text{ with } i = 0..7 \tag{8-376}$$

$$f_{i5} = (e_{i3} >> 2) - e_{i5}, \text{ with } i = 0..7 \tag{8-377}$$

$$f_{i6} = e_{i0} - e_{i6}, \text{ with } i = 0..7 \tag{8-378}$$

$$f_{i7} = e_{i7} - (e_{i1} >> 2), \text{ with } i = 0..7 \tag{8-379}$$

– Then, the transformed result g_{ij} is computed from these intermediate values f_{ij} as

$$g_{i0} = f_{i0} + f_{i7}, \text{ with } i = 0..7 \tag{8-380}$$

$$g_{i1} = f_{i2} + f_{i5}, \text{ with } i = 0..7 \tag{8-381}$$

$$g_{i2} = f_{i4} + f_{i3}, \text{ with } i = 0..7 \tag{8-382}$$

$$g_{i3} = f_{i6} + f_{i1}, \text{ with } i = 0..7 \tag{8-383}$$

$$g_{i4} = f_{i6} - f_{i1}, \text{ with } i = 0..7 \tag{8-384}$$

$$g_{i5} = f_{i4} - f_{i3}, \text{ with } i = 0..7 \tag{8-385}$$

$$g_{i6} = f_{i2} - f_{i5}, \text{ with } i = 0..7 \tag{8-386}$$

$$g_{i7} = f_{i0} - f_{i7}, \text{ with } i = 0..7 \tag{8-387}$$

Then, each (vertical) column of the resulting matrix is transformed using the same one-dimensional inverse transform as follows.

– A set of intermediate values h_{ij} is computed from the horizontally transformed value g_{ij} as

$$h_{0j} = g_{0j} + g_{4j}, \text{ with } j = 0..7 \tag{8-388}$$

$$h_{1j} = -g_{3j} + g_{5j} - g_{7j} - (g_{7j} >> 1), \text{ with } j = 0..7 \tag{8-389}$$

$$h_{2j} = g_{0j} - g_{4j}, \text{ with } j = 0..7 \tag{8-390}$$

$$h_{3j} = g_{1j} + g_{7j} - g_{3j} - (g_{3j} >> 1), \text{ with } j = 0..7 \tag{8-391}$$

$$h_{4j} = (g_{2j} >> 1) - g_{6j}, \text{ with } j = 0..7 \tag{8-392}$$

$$h_{5j} = -g_{1j} + g_{7j} + g_{5j} + (g_{5j} >> 1), \text{ with } j = 0..7 \tag{8-393}$$

$$h_{6j} = g_{2j} + (g_{6j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}394)$$

$$h_{7j} = g_{3j} + g_{5j} + g_{1j} + (g_{1j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}395)$$

– A second set of intermediate results k_{ij} is computed from the intermediate values h_{ij} as

$$k_{0j} = h_{0j} + h_{6j}, \text{ with } j = 0..7 \quad (8\text{-}396)$$

$$k_{1j} = h_{1j} + (h_{7j} >> 2), \text{ with } j = 0..7 \quad (8\text{-}397)$$

$$k_{2j} = h_{2j} + h_{4j}, \text{ with } j = 0..7 \quad (8\text{-}398)$$

$$k_{3j} = h_{3j} + (h_{5j} >> 2), \text{ with } j = 0..7 \quad (8\text{-}399)$$

$$k_{4j} = h_{2j} - h_{4j}, \text{ with } j = 0..7 \quad (8\text{-}400)$$

$$k_{5j} = (h_{3j} >> 2) - h_{5j}, \text{ with } j = 0..7 \quad (8\text{-}401)$$

$$k_{6j} = h_{0j} - h_{6j}, \text{ with } j = 0..7 \quad (8\text{-}402)$$

$$k_{7j} = h_{7j} - (h_{1j} >> 2), \text{ with } j = 0..7 \quad (8\text{-}403)$$

– Then, the transformed result m_{ij} is computed from these intermediate values k_{ij} as

$$m_{0j} = k_{0j} + k_{7j}, \text{ with } j = 0..7 \quad (8\text{-}404)$$

$$m_{1j} = k_{2j} + k_{5j}, \text{ with } j = 0..7 \quad (8\text{-}405)$$

$$m_{2j} = k_{4j} + k_{3j}, \text{ with } j = 0..7 \quad (8\text{-}406)$$

$$m_{3j} = k_{6j} + k_{1j}, \text{ with } j = 0..7 \quad (8\text{-}407)$$

$$m_{4j} = k_{6j} - k_{1j}, \text{ with } j = 0..7 \quad (8\text{-}408)$$

$$m_{5j} = k_{4j} - k_{3j}, \text{ with } j = 0..7 \quad (8\text{-}409)$$

$$m_{6j} = k_{2j} - k_{5j}, \text{ with } j = 0..7 \quad (8\text{-}410)$$

$$m_{7j} = k_{0j} - k_{7j}, \text{ with } j = 0..7 \quad (8\text{-}411)$$

The bitstream shall not contain data that results in any element e_{ij}, f_{ij}, g_{ij}, h_{ij}, or k_{ij} for i and j in the range of 0..7, inclusive, that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

The bitstream shall not contain data that results in any element m_{ij} for i and j in the range of 0..7, inclusive, that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 33$, inclusive.

After performing both the one-dimensional horizontal and the one-dimensional vertical inverse transforms to produce an array of transformed samples, the final constructed residual sample values are derived as

$$r_{ij} = (m_{ij} + 2^5) >> 6 \quad \text{with} \quad i, j = 0..7 \tag{8-412}$$

8.5.13 Picture construction process prior to deblocking filter process

Inputs to this process are

- a sample array u with elements u_{ij} which is a 16x16 luma block or an (MbWidthC)x(MbHeightC) chroma block or a 4x4 luma block or a 4x4 chroma block or an 8x8 luma block or, when ChromaArrayType is equal to 3, an 8x8 chroma block
- when u is not a 16x16 luma block or an (MbWidthC)x(MbHeightC) chroma block, a block index luma4x4BlkIdx or chroma4x4BlkIdx or luma8x8BlkIdx or cb4x4BlkIdx or cr4x4BlkIdx or cb8x8BlkIdx or cr8x8BlkIdx

The position of the upper-left luma sample of the current macroblock is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with CurrMbAddr as input and the output being assigned to (xP, yP).

When u is a luma block, for each sample u_{ij} of the luma block, the following applies.

- Depending on the size of the block u, the following applies.
  - If u is a 16x16 luma block, the position (xO, yO) of the upper-left sample of the 16x16 luma block inside the macroblock is set equal to (0, 0) and the variable nE is set equal to 16.
  - Otherwise, if u is an 4x4 luma block, the position of the upper-left sample of the 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO), and the variable nE is set equal to 4.
  - Otherwise (u is an 8x8 luma block), the position of the upper-left sample of the 8x8 luma block with index luma8x8BlkIdx inside the macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.5 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO), and the variable nE is set equal to 8.
- Depending on the variable MbaffFrameFlag and the current macroblock, the following applies.
  - If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

$$S'_L[xP + xO + j, yP + 2 * (yO + i)] = u_{ij} \quad \text{with } i, j = 0..nE - 1 \tag{8-413}$$

  - Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

$$S'_L[xP + xO + j, yP + yO + i] = u_{ij} \quad \text{with } i, j = 0..nE - 1 \tag{8-414}$$

When u is a chroma block, for each sample u_{ij} of the chroma block, the following applies.

- The subscript C in the variable S'_C is replaced with Cb for the Cb chroma component and with Cr for the Cr chroma component.
- Depending on the size of the block u, the following applies.
  - If u is an (MbWidthC)x(MbHeightC) Cb or Cr block, the variable nW is set equal to MbWidthC, the variable nH is set equal to MbHeightC, and the position (xO, yO) of the upper-left sample of the (nW)x(nH) Cb or Cr block inside the macroblock is set equal to (0, 0).
  - Otherwise, if u is a 4x4 Cb or Cr block, the variables nW and nH are set equal to 4 and, depending on the variable ChromaArrayType, the position of the upper-left sample of a 4x4 Cb or Cr block with index chroma4x4BlkIdx inside the macroblock is derived as follows.
    - If ChromaArrayType is equal to 1 or 2, the following applies.

$$xO = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 0) \tag{8-415}$$

$$yO = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 1) \tag{8-416}$$

– Otherwise (ChromaArrayType is equal to 3), the position of the upper-left sample of the 4x4 Cb block with index cb4x4BlkIdx or the 4x4 Cr block with index cr4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 Cb or Cr block scanning process in subclause 6.4.4 with cb4x4BlkIdx or cr4x4BlkIdx as the input and the output being assigned to (xO, yO).

– Otherwise (u is an 8x8 Cb or Cr block when ChromaArrayType is equal to 3), the variables nW and nH are set equal to 8 and the position of the upper-left sample of the 8x8 Cb block with index cb8x8BlkIdx or the Cr block with index cr8x8BlkIdx inside the macroblock is derived by invoking the inverse 8x8 Cb or Cr block scanning process in subclause 6.4.6 with cb8x8BlkIdx or cr8x8BlkIdx as the input and the output being assigned to (xO, yO).

– Depending on the variable MbaffFrameFlag and the current macroblock, the following applies.

  – If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

$$S'_C[\ (\ xP / subWidthC\ ) + xO + j,\ (\ (\ yP + SubHeightC - 1\ ) / SubHeightC\ ) + 2 * (\ yO + i\ )\ ] = u_{ij} \quad \text{with } i = 0..nH - 1 \text{ and } j = 0..nW - 1 \tag{8-417}$$

  – Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

$$S'_C[\ (\ xP/ subWidthC\ ) + xO + j,\ (\ yP / SubHeightC\ ) + yO + i\ ] = u_{ij} \quad \text{with } i = 0..nH - 1 \text{ and } j = 0..nW - 1 \tag{8-418}$$

8.5.14 Intra residual transform-bypass decoding process

This process is invoked when qpprime_y_zero_transform_bypass_flag is equal to 1, QP'_Y is equal to 0, the macroblock prediction mode is equal to Intra_4x4, Intra_8x8, or Intra_16x16, and the applicable intra prediction mode is equal to the vertical or horizontal mode. The process for the Cb and Cr components is applied in the same way as for the luma (L or Y) component.

Inputs to this process are

– two variables nW and nH

– a variable horPredFlag

– an (nW)x(nH) array r with elements r_{ij} which is either an array relating to a residual transform-bypass block of the luma component or an array relating to a residual transform-bypass block of the Cb and Cr component.

Output of this process is a modified version of the (nW)x(nH) array r with elements r_{ij} containing the result of the intra residual transform-bypass decoding process.

Let f be a temporary (nW)x(nH) array with elements f_{ij}, which are derived by

$$f_{ij} = r_{ij} \quad \text{with } i = 0..nH - 1 \text{ and } j = 0..nW - 1 \tag{8-419}$$

Depending on horPredFlag, the following applies.

– If horPredFlag is equal to 0, the modified array r is derived by.

$$r_{ij} = \sum_{k=0}^{i} f_{kj} \quad \text{with } i = 0..nH - 1 \text{ and } j = 0..nW - 1 \tag{8-420}$$

– Otherwise (horPredFlag is equal to 1), the modified array r is derived by

$$r_{ij} = \sum_{k=0}^{j} f_{ik} \quad \text{with } i = 0..nH - 1 \text{ and } j = 0..nW - 1 \tag{8-421}$$

8.6 Decoding process for P macroblocks in SP slices or SI macroblocks

This process is invoked when decoding P macroblock types in an SP slice type or an SI macroblock type in SI slices.

Inputs to this process are the prediction residual transform coefficient levels and the predicted samples for the current macroblock.

Outputs of this process are the decoded samples of the current macroblock prior to the deblocking filter process.

This subclause specifies the transform coefficient decoding process and picture construction process for P macroblock types in SP slices and SI macroblock type in SI slices.

NOTE – SP slices make use of Inter predictive coding to exploit temporal redundancy in the sequence, in a similar manner to P slice coding. Unlike P slice coding, however, SP slice coding allows identical reconstruction of a slice even when different reference pictures are being used. SI slices make use of spatial prediction, in a similar manner to I slices. SI slice coding allows identical reconstruction to a corresponding SP slice. The properties of SP and SI slices aid in providing functionalities for bitstream switching, splicing, random access, fast-forward, fast reverse, and error resilience/recovery.

An SP slice consists of macroblocks coded either as I macroblock types or P macroblock types.

An SI slice consists of macroblocks coded either as I macroblock types or SI macroblock type.

The transform coefficient decoding process and picture construction process prior to deblocking filter process for I macroblock types in SI slices is invoked as specified in subclause 8.5. SI macroblock type is decoded as described below.

When the current macroblock is coded as P_Skip, all values of LumaLevel, ChromaDCLevel, ChromaACLevel are set equal to 0 for the current macroblock.

8.6.1 SP decoding process for non-switching pictures

This process is invoked, when decoding P macroblock types in SP slices in which sp_for_switch_flag is equal to 0.

Inputs to this process are Inter prediction samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels.

Outputs of this process are the decoded samples of the current macroblock prior to the deblocking filter process.

This subclause applies to all macroblocks in SP slices in which sp_for_switch_flag is equal to 0, except those with macroblock prediction mode equal to Intra_4x4 or Intra_16x16. It does not apply to SI slices.

8.6.1.1 Luma transform coefficient decoding process

Inputs to this process are Inter prediction luma samples for the current macroblock $pred_L$ from subclause 8.4 and the prediction residual transform coefficient levels, LumaLevel, and the index of the 4x4 luma block luma4x4BlkIdx.

The position of the upper-left sample of the 4x4 luma block with index luma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (x, y).

Let the variable p be a 4x4 array of prediction samples with element p_{ij} being derived as follows.

$$p_{ij} = pred_L[\, x + j, y + i \,] \quad \text{with } i, j = 0..3 \tag{8-422}$$

The variable p is transformed producing transform coefficients c^p according to:

$$c^p = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \\ p_{30} & p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \tag{8-423}$$

The inverse transform coefficient scanning process as described in subclause 8.5.6 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the two-dimensional array c^r with elements c_{ij}^r as the output.

The prediction residual transform coefficients c^r are scaled using quantisation parameter QP_Y, and added to the transform coefficients of the prediction block c^p with i, j = 0..3 as follows.

$$c_{ij}^s = c_{ij}^p + (((c_{ij}^r * LevelScale(QP_Y \,\%\, 6, i, j) * A_{ij}) << (QP_Y / 6)) >> 10) \tag{8-424}$$

where LevelScale(m, i, j) is specified in Equation 8-319, and where A_{ij} is specified as:

$$A_{ij} = \begin{cases} 16 & \text{for} \quad (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\}, \\ 25 & \text{for} \quad (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\}, \\ 20 & \text{otherwise;} \end{cases} \tag{8-425}$$

The function LevelScale2(m, i, j), used in the formulas below, is specified as:

$$\text{LevelScale2}(m,i,j) = \begin{cases} w_{m0} & \text{for} \quad (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\}, \\ w_{m1} & \text{for} \quad (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\}, \\ w_{m2} & \text{otherwise;} \end{cases} \tag{8-426}$$

where the first and second subscripts of w are row and column indices, respectively, of the matrix specified as:

$$w = \begin{bmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{bmatrix} \tag{8-427}$$

The resulting sum, c^s, is quantised with a quantisation parameter QS_Y and with i, j = 0..3 as follows.

$$c_{ij} = \text{Sign}(c_{ij}^s) * ((\text{Abs}(c_{ij}^s) * \text{LevelScale2}(QS_Y \% 6, i, j) + (1 << (14 + QS_Y / 6))) >> (15 + QS_Y / 6)) \tag{8-428}$$

The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.11 is invoked with c as the input and r as the output.

The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}_Y(r_{ij}) \text{ with } i, j = 0..3 \tag{8-429}$$

The picture construction process prior to deblocking filter process in subclause 8.5.13 is invoked with luma4x4BlkIdx and u as the inputs.

8.6.1.2 Chroma transform coefficient decoding process

Inputs to this process are Inter prediction chroma samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels, ChromaDCLevel and ChromaACLevel.

This process is invoked twice: once for the Cb component and once for the Cr component. The component is referred to by replacing C with Cb for the Cb component and C with Cr for the Cr component. Let iCbCr select the current chroma component.

For each 4x4 block of the current chroma component indexed using chroma4x4BlkIdx with chroma4x4BlkIdx equal to 0..3, the following applies.

– The position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the macroblock is derived as follows.

$$x = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 0) \tag{8-430}$$

$$y = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 1) \tag{8-431}$$

– Let p be a 4x4 array of prediction samples with elements p_{ij} being derived as follows.

$$p_{ij} = pred_C[\, x + j, y + i \,] \quad \text{with } i, j = 0..3 \tag{8-432}$$

– The 4x4 array p is transformed producing transform coefficients c^p(chroma4x4BlkIdx) using Equation 8-423.

– The variable chromaList, which is a list of 16 entries, is derived. chromaList[0] is set equal to 0. chromaList[k] with index k = 1..15 are specified as follows.

$$\text{chromaList}[\, k \,] = \text{ChromaACLevel}[\, \text{iCbCr} \,][\, \text{chroma4x4BlkIdx} \,][\, k - 1 \,] \tag{8-433}$$

– The inverse transform coefficient scanning process as described in subclause 8.5.6 is invoked with chromaList as the input and the 4x4 array c^r as the output.

– The prediction residual transform coefficients c^r are scaled using quantisation parameter QPC, and added to the transform coefficients of the prediction block c^p with i, j = 0..3 except for the combination i = 0, j = 0 as follows.

$$c_{ij}^{\,s} = c_{ij}^{\,p}(\, \text{chroma4x4BlkIdx} \,) + (((c_{ij}^{\,r} * \text{LevelScale}(QP_C \,\%\, 6, i, j) * A_{ij}) << (QP_C / 6)) >> 10) \tag{8-434}$$

– The resulting sum, c^s, is quantised with a quantisation parameter QS_C and with i, j = 0..3 except for the combination i = 0, j = 0 as follows. The derivation of c_{00}(chroma4x4BlkIdx) is described below in this subclause.

$$\begin{aligned} c_{ij}(\, \text{chroma4x4BlkIdx} \,) = (\text{Sign}(c_{ij}^{\,s}) * (\text{Abs}(c_{ij}^{\,s}) * \text{LevelScale2}(QS_C \,\%\, 6, i, j) + \\ (1 << (14 + QS_C / 6)))) >> (15 + QS_C / 6) \end{aligned} \tag{8-435}$$

– The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.11 is invoked with c(chroma4x4BlkIdx) as the input and r as the output.

– The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}_C(\, r_{ij} \,) \text{ with } i, j = 0..3 \tag{8-436}$$

– The picture construction process prior to deblocking filter process in subclause 8.5.13 is invoked with chroma4x4BlkIdx and u as the inputs.

The derivation of the DC transform coefficient level c_{00}(chroma4x4BlkIdx) is specified as follows. The DC transform coefficients of the 4 prediction chroma 4x4 blocks of the current component of the macroblock are assembled into a 2x2 matrix with elements c_{00}^p(chroma4x4BlkIdx) and a 2x2 transform is applied to the DC transform coefficients as follows

$$dc^p = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00}^p(0) & c_{00}^p(1) \\ c_{00}^p(2) & c_{00}^p(3) \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{8-437}$$

The chroma DC prediction residual transform coefficient levels, ChromaDCLevel[iCbCr][k] with k = 0..3 are scaled using quantisation parameter QP, and added to the prediction DC transform coefficients as follows.

$$\begin{aligned} dc_{ij}^{\,s} = dc_{ij}^{\,p} + (((\text{ChromaDCLevel}[\, \text{iCbCr} \,][\, j * 2 + i \,] * \text{LevelScale}(QP_C \,\%\, 6, 0, 0) * A_{00}) << (QP_C / 6)) >> 9) \\ \text{with } i, j = 0, 1 \end{aligned} \tag{8-438}$$

The 2x2 array dc^s, is quantised using the quantisation parameter QS_C as follows.

$$\begin{aligned} dc_{ij}^{\,r} = (\text{Sign}(dc_{ij}^{\,s}) * (\text{Abs}(dc_{ij}^{\,s}) * \text{LevelScale2}(QS_C \,\%\, 6, 0, 0) + (1 << (15 + QS_C / 6)))) >> (16 + QS_C / 6) \\ \text{with } i, j = 0, 1 \end{aligned} \tag{8-439}$$

The 2x2 array f with elements f_{ij} and i, j = 0..1 is derived as follows.

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} dc_{00}^r & dc_{01}^r \\ dc_{10}^r & dc_{11}^r \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{8-440}$$

Scaling of the elements f_{ij} of f is performed as follows.

$$c_{00}(j * 2 + i) = ((f_{ij} * \text{LevelScale}(QS_C \% 6, 0, 0)) << (QS_C / 6)) >> 5 \quad \text{with i, j = 0, 1} \tag{8-441}$$

8.6.2 SP and SI slice decoding process for switching pictures

This process is invoked, when decoding P macroblock types in SP slices in which sp_for_switch_flag is equal to 1 and when decoding SI macroblock type in SI slices.

Inputs to this process are the prediction residual transform coefficient levels and the prediction sample arrays $pred_L$, $pred_{Cb}$ and $pred_{Cr}$ for the current macroblock.

8.6.2.1 Luma transform coefficient decoding process

Inputs to this process are prediction luma samples $pred_L$ and the luma prediction residual transform coefficient levels, LumaLevel.

The 4x4 array p with elements p_{ij} with i, j = 0..3 is derived as in subclause 8.6.1.1, is transformed according to Equation 8-423 to produce transform coefficients c^p. These transform coefficients are then quantised with the quantisation parameter QS_Y, as follows:

$$c_{ij}^s = \text{Sign}(c_{ij}^p) * ((\text{Abs}(c_{ij}^p) * \text{LevelScale2}(QS_Y \% 6, i, j) + (1 << (14 + QS_Y / 6))) >> (15 + QS_Y / 6)) \quad \text{with i, j = 0..3} \tag{8-442}$$

The inverse transform coefficient scanning process as described in subclause 8.5.6 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the two-dimensional array c^r with elements c_{ij}^r as the output.

The 4x4 array c with elements c_{ij} with i, j = 0..3 is derived as follows.

$$c_{ij} = c_{ij}^r + c_{ij}^s \quad \text{with i, j = 0..3} \tag{8-443}$$

The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.11 is invoked with c as the input and r as the output.

The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}_Y(r_{ij}) \text{ with i, j = 0..3} \tag{8-444}$$

The picture construction process prior to deblocking filter process in subclause 8.5.13 is invoked with luma4x4BlkIdx and u as the inputs.

8.6.2.2 Chroma transform coefficient decoding process

Inputs to this process are predicted chroma samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels, ChromaDCLevel and ChromaACLevel.

This process is invoked twice: once for the Cb component and once for the Cr component. The component is referred to by replacing C with Cb for the Cb component and C with Cr for the Cr component. Let iCbCr select the current chroma component.

For each 4x4 block of the current chroma component indexed using chroma4x4BlkIdx with chroma4x4BlkIdx equal to 0..3, the following applies.

1. The 4x4 array p with elements p_{ij} with i, j = 0..3 is derived as in subclause 8.6.1.2, is transformed according to Equation 8-423 to produce transform coefficients c^p(chroma4x4BlkIdx). These transform coefficients are then quantised with the quantisation parameter QS_C, with i, j = 0..3 except for the combination i = 0, j = 0 as follows. The processing of c_{00}^p(chroma4x4BlkIdx) is described below in this subclause.

$$c_{ij}^s = (\text{Sign}(c_{ij}^p(\text{chroma4x4BlkIdx})) * (\text{Abs}(c_{ij}^p(\text{chroma4x4BlkIdx})) * \text{LevelScale2}(QS_C \% 6, i, j) + (1 << (14 + QS_C / 6)))) >> (15 + QS_C / 6) \tag{8-445}$$

– The variable chromaList, which is a list of 16 entries, is derived. chromaList[0] is set equal to 0. chromaList[k] with index k = 1..15 are specified as follows.

$$\text{chromaList}[k] = \text{ChromaACLevel}[\text{iCbCr}][\text{chroma4x4BlkIdx}][k - 1] \tag{8-446}$$

– The inverse transform coefficient scanning process as described in subclause 8.5.6 is invoked with chromaList as the input and the two-dimensional array c^r(chroma4x4BlkIdx) with elements c_{ij}^r(chroma4x4BlkIdx) as the output.

– The 4x4 array c(chroma4x4BlkIdx) with elements c_{ij}(chroma4x4BlkIdx) with i, j = 0..3 except for the combination i = 0, j = 0 is derived as follows. The derivation of c_{00}(chroma4x4BlkIdx) is described below.

$$c_{ij}(\text{chroma4x4BlkIdx}) = c_{ij}^r(\text{chroma4x4BlkIdx}) + c_{ij}^s \tag{8-447}$$

– The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.11 is invoked with c(chroma4x4BlkIdx) as the input and r as the output.

– The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}_C(r_{ij}) \text{ with } i, j = 0..3 \tag{8-448}$$

– The picture construction process prior to deblocking filter process in subclause 8.5.13 is invoked with chroma4x4BlkIdx and u as the inputs.

The derivation of the DC transform coefficient level c_{00}(chroma4x4BlkIdx) is specified as follows. The DC transform coefficients of the 4 prediction 4x4 chroma blocks of the current component of the macroblock, c_{00}^P(chroma4x4BlkIdx), are assembled into a 2x2 matrix, and a 2x2 transform is applied to the DC transform coefficients of these blocks according to Equation 8-437 resulting in DC transform coefficients dc_{ij}^P.

These DC transform coefficients are then quantised with the quantisation parameter QS_C, as given by:

$$dc_{ij}^s = (\text{Sign}(dc_{ij}^P) * (\text{Abs}(dc_{ij}^P) * \text{LevelScale2}(QS_C \% 6, 0, 0) + (1 << (15 + QS_C / 6)))) >> (16 + QS_C / 6) \quad \text{with } i, j = 0, 1 \tag{8-449}$$

The parsed chroma DC prediction residual transform coefficients, ChromaDCLevel[iCbCr][k] with k = 0..3 are added to these quantised DC transform coefficients of the prediction block, as given by:

$$dc_{ij}^r = dc_{ij}^s + \text{ChromaDCLevel}[\text{iCbCr}][j * 2 + i] \text{ with } i, j = 0, 1 \tag{8-450}$$

The 2x2 array f with elements f_{ij} and i, j = 0..1 is derived using Equation 8-440.

The 2x2 array f with elements f_{ij} and i, j = 0..1 is copied as follows.

$$c_{00}(j * 2 + i) = f_{ij} \text{ with } i, j = 0, 1 \tag{8-451}$$

8.7 Deblocking filter process

A conditional filtering process is specified in this subclause that is an integral part of the decoding process which shall be applied by decoders conforming to the Baseline, Extended, Main, High, High 10, High 4:2:2, and High 4:4:4 Predictive profiles. For decoders conforming to the High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profiles, the filtering process specified in this subclause, or one similar to it, should be applied but is not required.

The conditional filtering process is applied to all NxN (where N = 4 or N = 8 for luma, N = 4 for chroma when ChromaArrayType is equal to 1 or 2, and N = 4 or N = 8 for chroma when ChromaArrayType is equal to 3) block edges of a picture, except edges at the boundary of the picture and any edges for which the deblocking filter process is disabled by disable_deblocking_filter_idc, as specified below. This filtering process is performed on a macroblock basis after the completion of the picture construction process prior to deblocking filter process (as specified in subclauses 8.5 and 8.6) for the entire decoded picture, with all macroblocks in a picture processed in order of increasing macroblock addresses.

NOTE 1 – Prior to the operation of the deblocking filter process for each macroblock, the deblocked samples of the macroblock or macroblock pair above (if any) and the macroblock or macroblock pair to the left (if any) of the current macroblock are always available because the deblocking filter process is performed after the completion of the picture construction process prior to deblocking filter process for the entire decoded picture. However, for purposes of determining which edges are to be filtered when disable_deblocking_filter_idc is equal to 2, macroblocks in different slices are considered not available during specified steps of the operation of the deblocking filter process.

The deblocking filter process is invoked for the luma and chroma components separately. For each macroblock and each component, vertical edges are filtered first, starting with the edge on the left-hand side of the macroblock proceeding through the edges towards the right-hand side of the macroblock in their geometrical order, and then horizontal edges are filtered, starting with the edge on the top of the macroblock proceeding through the edges towards the bottom of the macroblock in their geometrical order. Figure 8-10 shows edges of a macroblock which can be interpreted as luma or chroma edges.

When interpreting the edges in Figure 8-10 as luma edges, depending on the transform_size_8x8_flag, the following applies.

– If transform_size_8x8_flag is equal to 0, both types, the solid bold and dashed bold luma edges are filtered.

– Otherwise (transform_size_8x8_flag is equal to 1), only the solid bold luma edges are filtered.

When interpreting the edges in Figure 8-10 as chroma edges, depending on ChromaArrayType, the following applies.

– If ChromaArrayType is equal to 1 (4:2:0 format), only the solid bold chroma edges are filtered.

– Otherwise, if ChromaArrayType is equal to 2 (4:2:2 format), the solid bold vertical chroma edges are filtered and both types, the solid bold and dashed bold horizontal chroma edges are filtered.

– Otherwise, if ChromaArrayType is equal to 3 (4:4:4 format), the following applies.

  – If transform_size_8x8_flag is equal to 0, both types, the solid bold and dashed bold chroma edges are filtered.

  – Otherwise (transform_size_8x8_flag is equal to 1), only the solid bold chroma edges are filtered.

– Otherwise (ChromaArrayType is equal to 0), no chroma edges are filtered.

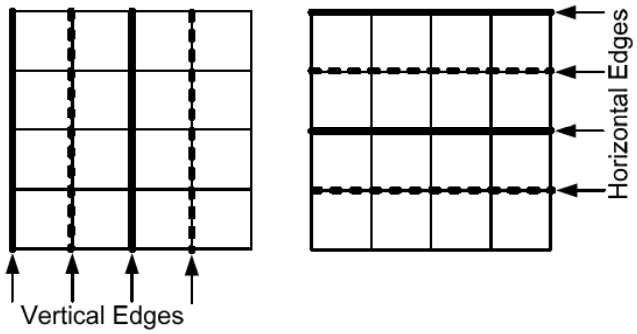


Figure 8-10 – Boundaries in a macroblock to be filtered

For the current macroblock address CurrMbAddr proceeding over values 0..PicSizeInMbs – 1, the following applies.

1. The derivation process for neighbouring macroblocks specified in subclause 6.4.10.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

2. The variables fieldModeMbFlag, filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag are derived as follows.

   – The variable fieldModeMbFlag is derived as follows.

     – If any of the following conditions is true, fieldModeMbFlag is set equal to 1.

       – field_pic_flag is equal to 1.

       – MbaffFrameFlag is equal to 1 and the macroblock CurrMbAddr is a field macroblock.

     – Otherwise, fieldModeMbFlag is set equal to 0.

– The variable filterInternalEdgesFlag is derived as follows.
  – If disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1, the variable filterInternalEdgesFlag is set equal to 0;
  – Otherwise (disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is not equal to 1), the variable filterInternalEdgesFlag is set equal to 1.
– The variable filterLeftMbEdgeFlag is derived as follows.
  – If any of the following conditions is true, the variable filterLeftMbEdgeFlag is set equal to 0.
    – MbaffFrameFlag is equal to 0 and CurrMbAddr % PicWidthInMbs is equal to 0.
    – MbaffFrameFlag is equal to 1 and (CurrMbAddr >> 1) % PicWidthInMbs is equal to 0.
    – disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1.
    – disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 2 and the macroblock mbAddrA is not available.
  – Otherwise, the variable filterLeftMbEdgeFlag is set equal to 1.
– The variable filterTopMbEdgeFlag is derived as follows.
  – If any of the following conditions is true, the variable filterTopMbEdgeFlag is set equal to 0.
    – MbaffFrameFlag is equal to 0 and CurrMbAddr is less than PicWidthInMbs.
    – MbaffFrameFlag is equal to 1, (CurrMbAddr >> 1) is less than PicWidthInMbs, and the macroblock CurrMbAddr is a field macroblock.
    – MbaffFrameFlag is equal to 1, (CurrMbAddr >> 1) is less than PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and CurrMbAddr % 2 is equal to 0.
    – disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1.
    – disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 2 and the macroblock mbAddrB is not available.
  – Otherwise, the variable filterTopMbEdgeFlag is set equal to 1.

3. Given the variables fieldModeMbFlag, filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag the deblocking filtering is controlled as follows.
– When filterLeftMbEdgeFlag is equal to 1, the filtering of the left vertical luma edge is specified as follows.
  – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (0, k)$ with k = 0..15 as input and S'_L as output.
– When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical luma edges is specified as follows.
  – When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (4, k)$ with k = 0..15 as input and S'_L as output.
  – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (8, k)$ with k = 0..15 as input and S'_L as output.
  – When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (12, k)$ with k = 0..15 as input and S'_L as output.
– When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal luma edge is specified as follows.
  – If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to 2 * PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and the macroblock (CurrMbAddr - 2 * PicWidthInMbs + 1) is a field macroblock, the following applies.

- The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, and $(xE_k, yE_k) = (k, 0)$ with k = 0..15 as input and S'_L as output.
- The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, and $(xE_k, yE_k) = (k, 1)$ with k = 0..15 as input and S'_L as output.

– Otherwise, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (k, 0)$ with k = 0..15 as input and S'_L as output.

– When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal luma edges is specified as follows.
  - When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (k, 4)$ with k = 0..15 as input and S'_L as output.
  - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (k, 8)$ with k = 0..15 as input and S'_L as output.
  - When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (k, 12)$ with k = 0..15 as input and S'_L as output.

– When ChromaArrayType is not equal to 0, for the filtering of both chroma components, with iCbCr = 0 for Cb and iCbCr = 1 for Cr, the following applies.
  - When filterLeftMbEdgeFlag is equal to 1, the filtering of the left vertical chroma edge is specified as follows.
    - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (0, k)$ with k = 0..MbHeightC - 1 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.
  - When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical chroma edge is specified as follows.
    - When ChromaArrayType is not equal to 3 or transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (4, k)$ with k = 0..MbHeightC - 1 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.
    - When ChromaArrayType is equal to 3, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (8, k)$ with k = 0..MbHeightC - 1 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.
    - When ChromaArrayType is equal to 3 or transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (12, k)$ with k = 0..MbHeightC - 1 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.
  - When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal chroma edge is specified as follows.
    - If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to 2 * PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and the macroblock (CurrMbAddr – 2 * PicWidthInMbs + 1) is a field macroblock, the following applies.
      - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, and $(xE_k, yE_k) = (k, 0)$ with k = 0..MbWidthC - 1 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.

– The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, and $(xE_k, yE_k) = (k, 1)$ with k = 0..MbWidthC - 1 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.

– Otherwise, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (k, 0)$ with k = 0..MbWidthC - 1 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.

– When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal chroma edge is specified as follows.

– When ChromaArrayType is not equal to 3 or transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (k, 4)$ with k = 0..MbWidthC - 1 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.

– When ChromaArrayType is not equal to 1, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (k, 8)$ with k = 0..MbWidthC - 1 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.

– When ChromaArrayType is equal to 2, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (k, 12)$ with k = 0..MbWidthC - 1 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.

– When ChromaArrayType is equal to 3 or transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, and $(xE_k, yE_k) = (k, 12)$ with k = 0..MbWidthC – 1 as input and S'_C with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1 as output.

NOTE 2 – When field mode filtering (fieldModeFilteringFlag is equal to 1) is applied across the top horizontal edges of a frame macroblock, this vertical filtering across the top or bottom macroblock boundary may involve some samples that extend across an internal block edge that is also filtered internally in frame mode.

NOTE 3 – For example, in 4:2:0 chroma format when transform_size_8x8_flag is equal to 0, the following applies. 3 horizontal luma edges, 1 horizontal chroma edge for Cb, and 1 horizontal chroma edge for Cr are filtered that are internal to a macroblock. When field mode filtering (fieldModeFilteringFlag is equal to 1) is applied to the top edges of a frame macroblock, 2 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr between the frame macroblock and the above macroblock pair are filtered using field mode filtering, for a total of up to 5 horizontal luma edges, 3 horizontal chroma edges for Cb, and 3 horizontal chroma edges for Cr filtered that are considered to be controlled by the frame macroblock. In all other cases, at most 4 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr are filtered that are considered to be controlled by a particular macroblock.

Depending on separate_colour_plane_flag the following applies.

– If separate_colour_plane_flag is equal to 0, the arrays S'_L, S'_{Cb}, S'_{Cr} are assigned to the arrays S_L, S_{Cb}, S_{Cr} (which represent the decoded picture), respectively.

– Otherwise (separate_colour_plane_flag is equal to 1), the following applies.

– If colour_plane_id is equal to 0, the arrays S'_L is assigned to the array S_L (which represent the luma component of the decoded picture).

– Otherwise, if colour_plane_id is equal to 1, the arrays S'_L is assigned to the array S_{Cb} (which represents the Cb component of the decoded picture).

– Otherwise (colour_plane_id is equal to 2), the arrays S'_L is assigned to the array S_{Cr} (which represents the Cr component of the decoded picture).

8.7.1 Filtering process for block edges

Inputs to this process are chromaEdgeFlag, the chroma component index iCbCr (when chromaEdgeFlag is equal to 1), verticalEdgeFlag, fieldModeFilteringFlag, and a set of nE sample locations (xE_k, yE_k), with k = 0..nE - 1, expressed relative to the upper left corner of the macroblock CurrMbAddr. The set of sample locations (xE_k, yE_k) represent the

sample locations immediately to the right of a vertical edge (when verticalEdgeFlag is equal to 1) or immediately below a horizontal edge (when verticalEdgeFlag is equal to 0).

The variable nE is derived as follows.

- If chromaEdgeFlag is equal to 0, nE is set equal to 16.
- Otherwise (chromaEdgeFlag is equal to 1), nE is set equal to (verticalEdgeFlag == 1) ? MbHeightC : MbWidthC.

Let s' be a variable specifying a luma or chroma sample array, be derived as follows.

- If chromaEdgeFlag is equal to 0, s' represents the luma sample array S'_L of the current picture.
- Otherwise, if chromaEdgeFlag is equal to 1 and iCbCr is equal to 0, s' represents the chroma sample array S'_{Cb} of the chroma component Cb of the current picture.
- Otherwise (chromaEdgeFlag is equal to 1 and iCbCr is equal to 1), s' represents the chroma sample array S'_{Cr} of the chroma component Cr of the current picture.

The variable dy is derived as follows.

- If fieldModeFilteringFlag is equal to 1 and MbaffFrameFlag is equal to 1, dy is set equal to 2.
- Otherwise (fieldModeFilteringFlag is equal to 0 or MbaffFrameFlag is equal to 0), dy is set equal to 1.

The position of the upper-left luma sample of the macroblock CurrMbAddr is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddr = CurrMbAddr as input and the output being assigned to (xI, yI).

The variables xP and yP are derived as follows.

- If chromaEdgeFlag is equal to 0, xP is set equal to xI and yP is set equal to yI.
- Otherwise (chromaEdgeFlag is equal to 1), xP is set equal to xI / SubWidthC and yP is set equal to (yI + SubHeightC – 1) / SubHeightC.

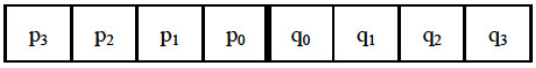


Figure 8-11 – Convention for describing samples across a 4x4 block horizontal or vertical boundary

For each sample location (xE_k, yE_k), k = 0 .. nE - 1, the following applies.

- The filtering process is applied to a set of eight samples across a 4x4 block horizontal or vertical edge denoted as p_i and q_i with i = 0..3 as shown in Figure 8-11 with the edge lying between p_0 and q_0. p_i and q_i with i = 0..3 are specified as follows.
  - If verticalEdgeFlag is equal to 1,

$$q_i = s'[\ xP + xE_k + i,\ yP + dy * yE_k\ ] \quad (8\text{-}452)$$

$$p_i = s'[\ xP + xE_k - i - 1,\ yP + dy * yE_k\ ] \quad (8\text{-}453)$$

  - Otherwise (verticalEdgeFlag is equal to 0),

$$q_i = s'[\ xP + xE_k,\ yP + dy * (\ yE_k + i\ ) - (yE_k \% 2\ )\ ] \quad (8\text{-}454)$$

$$p_i = s'[\ xP + xE_k,\ yP + dy * (\ yE_k - i - 1\ ) - (yE_k \% 2\ )\ ] \quad (8\text{-}455)$$

– The process specified in subclause 8.7.2 is invoked with the sample values p_i and q_i (i = 0..3), chromaEdgeFlag, verticalEdgeFlag, and fieldModeFilteringFlag as input, and the output is assigned to the filtered result sample values p'_i and q'_i with i = 0..2.

– The input sample values p_i and q_i with i = 0..2 are replaced by the corresponding filtered result sample values p'_i and q'_i with i = 0..2 inside the sample array s' as follows.

  – If verticalEdgeFlag is equal to 1,

$$s'[\, xP + xE_k + i,\ yP + dy * yE_k \,] = q'_i \tag{8-456}$$

$$s'[\, xP + xE_k - i - 1,\ yP + dy * yE_k \,] = p'_i \tag{8-457}$$

  – Otherwise (verticalEdgeFlag is equal to 0),

$$s'[\, xP + xE_k,\ yP + dy * (yE_k + i) - (yE_k \% 2) \,] = q'_i \tag{8-458}$$

$$s'[\, xP + xE_k,\ yP + dy * (yE_k - i - 1) - (yE_k \% 2) \,] = p'_i \tag{8-459}$$

8.7.2 Filtering process for a set of samples across a horizontal or vertical block edge

Inputs to this process are the input sample values p_i and q_i with i in the range of 0..3 of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, verticalEdgeFlag, and fieldModeFilteringFlag.

Outputs of this process are the filtered result sample values p'_i and q'_i with i in the range of 0..2.

The content dependent boundary filtering strength variable bS is derived as follows.

– If chromaEdgeFlag is equal to 0, the derivation process for the content dependent boundary filtering strength specified in subclause 8.7.2.1 is invoked with p_0, q_0, and verticalEdgeFlag as input, and the output is assigned to bS.

– Otherwise (chromaEdgeFlag is equal to 1), the bS used for filtering a set of samples of a horizontal or vertical chroma edge is set equal to the value of bS for filtering the set of samples of a horizontal or vertical luma edge, respectively, that contains the luma sample at location (SubWidthC * x, SubHeightC * y) inside the luma array of the same field, where (x, y) is the location of the chroma sample q_0 inside the chroma array for that field.

Let filterOffsetA and filterOffsetB be the values of FilterOffsetA and FilterOffsetB as specified in subclause 7.4.3 for the slice that contains the macroblock containing sample q_0.

Let qP_p and qP_q be variables specifying quantisation parameter values for the macroblocks containing the samples p_0 and q_0, respectively. The variables qP_z (with z being replaced by p or q) are derived as follows.

– If chromaEdgeFlag is equal to 0, the following applies.

  – If the macroblock containing the sample z_0 is an I_PCM macroblock, qP_z is set to 0.

  – Otherwise (the macroblock containing the sample z_0 is not an I_PCM macroblock), qP_z is set to the value of QP_Y of the macroblock containing the sample z_0.

– Otherwise (chromaEdgeFlag is equal to 1), the following applies.

  – If the macroblock containing the sample z_0 is an I_PCM macroblock, qP_z is equal set to the value of QP_C that corresponds to a value of 0 for QP_Y as specified in subclause 8.5.8.

  – Otherwise (the macroblock containing the sample z_0 is not an I_PCM macroblock), qP_z is set equal to the value of QP_C that corresponds to the value QP_Y of the macroblock containing the sample z_0 as specified in subclause 8.5.8.

The process specified in subclause 8.7.2.2 is invoked with p_0, q_0, p_1, q_1, chromaEdgeFlag, bS, filterOffsetA, filterOffsetB, qP_p, and qP_q as inputs, and the outputs are assigned to filterSamplesFlag, indexA, α, and β.

The variable chromaStyleFiltering is set by

$$\text{chromaStyleFiltering} = \text{chromaEdgeFlag} \;\&\&\; (\text{ChromaArrayType} \;!=\; 3) \tag{8-460}$$

Depending on the variable filterSamplesFlag, the following applies.

– If filterSamplesFlag is equal to 1, the following applies.

  – If bS is less than 4, the process specified in subclause 8.7.2.3 is invoked with p_i and q_i (i = 0..2), chromaEdgeFlag, chromaStyleFiltering, bS, β, and indexA given as input, and the output is assigned to p'_i and q'_i (i = 0..2).

  – Otherwise (bS is equal to 4), the process specified in subclause 8.7.2.4 is invoked with p_i and q_i (i = 0..3), chromaEdgeFlag, chromaStyleFiltering, α, and β given as input, and the output is assigned to p'_i and q'_i (i = 0..2).

– Otherwise (filterSamplesFlag is equal to 0), the filtered result samples p'_i and q'_i (i = 0..2) are replaced by the corresponding input samples p_i and q_i:

$$\text{for } i = 0..2, \quad p'_i = p_i \tag{8-461}$$

$$\text{for } i = 0..2, \quad q'_i = q_i \tag{8-462}$$

8.7.2.1 Derivation process for the luma content dependent boundary filtering strength

Inputs to this process are the input sample values p_0 and q_0 of a single set of samples across an edge that is to be filtered and verticalEdgeFlag.

Output of this process is the variable bS.

Let the variable mixedModeEdgeFlag be derived as follows.

– If MbaffFrameFlag is equal to 1 and the samples p_0 and q_0 are in different macroblock pairs, one of which is a field macroblock pair and the other is a frame macroblock pair, mixedModeEdgeFlag is set equal to 1

– Otherwise, mixedModeEdgeFlag is set equal to 0.

The variable bS is derived as follows.

– If the block edge is also a macroblock edge and any of the following conditions are true, a value of bS equal to 4 is the output:

  – the samples p_0 and q_0 are both in frame macroblocks and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode.

  – the samples p_0 and q_0 are both in frame macroblocks and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI.

  – MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode.

  – MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI.

– Otherwise, if any of the following conditions are true, a value of bS equal to 3 is the output:

  – mixedModeEdgeFlag is equal to 0 and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode.

  – mixedModeEdgeFlag is equal to 0 and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI.

  – mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode.

  – mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI.

– Otherwise, if the following condition is true, a value of bS equal to 2 is the output:

  – the luma block containing sample p_0 or the luma block containing sample q_0 contains non-zero transform coefficient levels.

– Otherwise, if any of the following conditions are true, a value of bS equal to 1 is the output:

  – mixedModeEdgeFlag is equal to 1.

– mixedModeEdgeFlag is equal to 0 and for the prediction of the macroblock/sub-macroblock partition containing the sample p_0 different reference pictures or a different number of motion vectors are used than for the prediction of the macroblock/sub-macroblock partition containing the sample q_0.

NOTE 1 – The determination of whether the reference pictures used for the two macroblock/sub-macroblock partitions are the same or different is based only on which pictures are referenced, without regard to whether a prediction is formed using an index into reference picture list 0 or an index into reference picture list 1, and also without regard to whether the index position within a reference picture list is different.

– mixedModeEdgeFlag is equal to 0 and one motion vector is used to predict the macroblock/sub-macroblock partition containing the sample p_0 and one motion vector is used to predict the macroblock/sub-macroblock partition containing the sample q_0 and the absolute difference between the horizontal or vertical component of the motion vectors used is greater than or equal to 4 in units of quarter luma frame samples.

– mixedModeEdgeFlag is equal to 0 and two motion vectors and two different reference pictures are used to predict the macroblock/sub-macroblock partition containing the sample p_0 and two motion vectors for the same two reference pictures are used to predict the macroblock/sub-macroblock partition containing the sample q_0 and the absolute difference between the horizontal or vertical component of the two motion vectors used in the prediction of the two macroblock/sub-macroblock partitions for the same reference picture is greater than or equal to 4 in units of quarter luma frame samples.

– mixedModeEdgeFlag is equal to 0 and two motion vectors for the same reference picture are used to predict the macroblock/sub-macroblock partition containing the sample p_0 and two motion vectors for the same reference picture are used to predict the macroblock/sub-macroblock partition containing the sample q_0 and both of the following conditions are true:

  – The absolute difference between the horizontal or vertical component of list 0 motion vectors used in the prediction of the two macroblock/sub-macroblock partitions is greater than or equal to 4 in quarter luma frame samples or the absolute difference between the horizontal or vertical component of the list 1 motion vectors used in the prediction of the two macroblock/sub-macroblock partitions is greater than or equal to 4 in units of quarter luma frame samples.

  – The absolute difference between the horizontal or vertical component of list 0 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample p_0 and the list 1 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample q_0 is greater than or equal to 4 in units of quarter luma frame samples or the absolute difference between the horizontal or vertical component of the list 1 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample p_0 and list 0 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample q_0 is greater than or equal to 4 in units of quarter luma frame samples.

    NOTE 2 – A vertical difference of 4 in units of quarter luma frame samples is a difference of 2 in units of quarter luma field samples

– Otherwise, a value of bS equal to 0 is the output.

8.7.2.2 Derivation process for the thresholds for each block edge

Inputs to this process are

– the input sample values p_0, q_0, p_1 and q_1 of a single set of samples across an edge that is to be filtered,

– the variables chromaEdgeFlag and bS, for the set of input samples, as specified in 8.7.2,

– the variables filterOffsetA, filterOffsetB, qP_p, and qP_q.

Outputs of this process are the variable filterSamplesFlag, which indicates whether the input samples are filtered, the value of indexA, and the values of the threshold variables α and β.

Let qP_{av} be a variable specifying an average quantisation parameter. It is derived as follows.

$$qP_{av} = (qP_p + qP_q + 1) >> 1 \quad (8\text{-}463)$$

NOTE – In SP and SI slices, qP_{av} is derived in the same way as in other slice types. QS_Y from Equation 7-29 is not used in the deblocking filter.

Let indexA be a variable that is used to access the α table (Table 8-16) as well as the t_{C0} table (Table 8-17), which is used in filtering of edges with bS less than 4 as specified in subclause 8.7.2.3, and let indexB be a variable that is used to access the β table (Table 8-16). The variables indexA and indexB are derived as follows.

$$indexA = Clip3(0, 51, qP_{av} + filterOffsetA) \quad (8\text{-}464)$$

$$indexB = Clip3(0, 51, qP_{av} + filterOffsetB) \quad (8\text{-}465)$$

The variables α' and β' depending on the values of indexA and indexB are specified in Table 8-16. Depending on chromaEdgeFlag, the corresponding threshold variables α and β are derived as follows.

– If chromaEdgeFlag is equal to 0,

$$\alpha = \alpha' * (1 << (BitDepth_Y - 8)) \quad (8\text{-}466)$$

$$\beta = \beta' * (1 << (BitDepth_Y - 8)) \quad (8\text{-}467)$$

– Otherwise (chromaEdgeFlag is equal to 1),

$$\alpha = \alpha' * (1 << (BitDepth_C - 8)) \quad (8\text{-}468)$$

$$\beta = \beta' * (1 << (BitDepth_C - 8)) \quad (8\text{-}469)$$

The variable filterSamplesFlag is derived by

$$filterSamplesFlag = (bS != 0 \ \&\& \ Abs(p_0 - q_0) < \alpha \ \&\& \ Abs(p_1 - p_0) < \beta \ \&\& \ Abs(q_1 - q_0) < \beta) \quad (8\text{-}470)$$

Table 8-16 – Derivation of offset dependent threshold variables α' and β' from indexA and indexB

	indexA (for α') or indexB (for β')																									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
α'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	5	6	7	8	9	10	12	13
β'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	3	3	3	3	4	4	4

Table 8-16 (concluded) – Derivation of indexA and indexB from offset dependent threshold variables α' and β'

	indexA (for α') or indexB (for β')																									
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
α'	15	17	20	22	25	28	32	36	40	45	50	56	63	71	80	90	101	113	127	144	162	182	203	226	255	255
β'	6	6	7	7	8	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	18

8.7.2.3 Filtering process for edges with bS less than 4

Inputs to this process are the input sample values p_i and q_i ($i = 0..2$) of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, chromaStyleFiltering, bS, β, and indexA, for the set of input samples, as specified in 8.7.2.

Outputs of this process are the filtered result sample values p'_i and q'_i ($i = 0..2$) for the set of input sample values.

Let Clip1( ) be a function that is replaced by $Clip1_Y( )$ when chromaStyleFiltering is equal to 0 and by $Clip1_C( )$ when chromaStyleFIltering is equal to 1.

The filtered result samples p'_0 and q'_0 are derived by

$$\Delta = Clip3(-t_C, t_C, ((((q_0 - p_0) << 2) + (p_1 - q_1) + 4) >> 3)) \quad (8\text{-}471)$$

$$p'_0 = Clip1(p_0 + \Delta) \quad (8\text{-}472)$$

$$q'_0 = \text{Clip1}(q_0 - \Delta) \tag{8-473}$$

where the threshold t_C is determined as follows.

– If chromaStyleFiltering is equal to 0,

$$t_C = t_{C0} + ((a_p < \beta) \ ? \ 1 : 0) + ((a_q < \beta) \ ? \ 1 : 0) \tag{8-474}$$

– Otherwise (chromaStyleFiltering is equal to 1),

$$t_C = t_{C0} + 1 \tag{8-475}$$

Depending on the values of indexA and bS the variable t'_{C0} is specified in Table 8-17. Depending on chromaEdgeFlag, the corresponding threshold variable t_{C0} is derived as follows.

– If chromaEdgeFlag is equal to 0,

$$t_{C0} = t'_{C0} * (1 << (\text{BitDepth}_Y - 8)) \tag{8-476}$$

– Otherwise (chromaEdgeFlag is equal to 1),

$$t_{C0} = t'_{C0} * (1 << (\text{BitDepth}_C - 8)) \tag{8-477}$$

Let a_p and a_q be two threshold variables specified by

$$a_p = \text{Abs}(p_2 - p_0) \tag{8-478}$$

$$a_q = \text{Abs}(q_2 - q_0) \tag{8-479}$$

The filtered result sample p'_1 is derived as follows

– If chromaStyleFiltering is equal to 0 and a_p is less than β,

$$p'_1 = p_1 + \text{Clip3}(-t_{C0}, t_{C0}, (p_2 + ((p_0 + q_0 + 1) >> 1) - (p_1 << 1)) >> 1) \tag{8-480}$$

– Otherwise (chromaStyleFiltering is equal to 1 or a_p is greater than or equal to β),

$$p'_1 = p_1 \tag{8-481}$$

The filtered result sample q'_1 is derived as follows

– If chromaStyleFiltering is equal to 0 and a_q is less than β,

$$q'_1 = q_1 + \text{Clip3}(-t_{C0}, t_{C0}, (q_2 + ((p_0 + q_0 + 1) >> 1) - (q_1 << 1)) >> 1) \tag{8-482}$$

– Otherwise (chromaStyleFiltering is equal to 1 or a_q is greater than or equal to β),

$$q'_1 = q_1 \tag{8-483}$$

The filtered result samples p'_2 and q'_2 are always set equal to the input samples p_2 and q_2:

$$p'_2 = p_2 \tag{8-484}$$

$$q'_2 = q_2 \tag{8-485}$$

Table 8-17 – Value of variable t'_{C0} as a function of indexA and bS

	indexA																								
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
bS = 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
bS = 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
bS = 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

Table 8-17 (concluded) – Value of variable t'_{C0} as a function of indexA and bS

	indexA																								
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
bS = 1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13
bS = 2	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5	5	6	7	8	8	10	11	12	13	15	17
bS = 3	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13	14	16	18	20	23	25

8.7.2.4 Filtering process for edges for bS equal to 4

Inputs to this process are the input sample values p_i and q_i ($i = 0..3$) of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, chromaStyleFiltering, and the values of the threshold variables α and β for the set of samples, as specified in subclause 8.7.2.

Outputs of this process are the filtered result sample values p'_i and q'_i ($i = 0..2$) for the set of input sample values.

Let a_p and a_q be two threshold variables as specified in Equations 8-478 and 8-479, respectively, in subclause 8.7.2.3.

The filtered result samples p'_i ($i = 0..2$) are derived as follows.

– If chromaStyleFiltering is equal to 0 and the following condition holds,

$$a_p < \beta \;\&\&\; Abs(p_0 - q_0) < ((\alpha >> 2) + 2) \tag{8-486}$$

then the variables p'_0, p'_1, and p'_2 are derived by

$$p'_0 = (p_2 + 2*p_1 + 2*p_0 + 2*q_0 + q_1 + 4) >> 3 \tag{8-487}$$

$$p'_1 = (p_2 + p_1 + p_0 + q_0 + 2) >> 2 \tag{8-488}$$

$$p'_2 = (2*p_3 + 3*p_2 + p_1 + p_0 + q_0 + 4) >> 3 \tag{8-489}$$

– Otherwise (chromaStyleFiltering is equal to 1 or the condition in Equation 8-486 does not hold), the variables p'_0, p'_1, and p'_2 are derived by

$$p'_0 = (2*p_1 + p_0 + q_1 + 2) >> 2 \tag{8-490}$$

$$p'_1 = p_1 \tag{8-491}$$

$$p'_2 = p_2 \tag{8-492}$$

The filtered result samples q'_i (i = 0..2) are derived as follows.

– If chromaStyleFiltering is equal to 0 and the following condition holds,

$$a_q < \beta \;\&\&\; Abs(p_0 - q_0) < ((\alpha >> 2) + 2) \quad (8\text{-}493)$$

then the variables q'_0, q'_1, and q'_2 are derived by

$$q'_0 = (p_1 + 2*p_0 + 2*q_0 + 2*q_1 + q_2 + 4) >> 3 \quad (8\text{-}494)$$

$$q'_1 = (p_0 + q_0 + q_1 + q_2 + 2) >> 2 \quad (8\text{-}495)$$

$$q'_2 = (2*q_3 + 3*q_2 + q_1 + q_0 + p_0 + 4) >> 3 \quad (8\text{-}496)$$

– Otherwise (chromaStyleFiltering is equal to 1 or the condition in Equation 8-493 does not hold), the variables q'_0, q'_1, and q'_2 are derived by

$$q'_0 = (2*q_1 + q_0 + p_1 + 2) >> 2 \quad (8\text{-}497)$$

$$q'_1 = q_1 \quad (8\text{-}498)$$

$$q'_2 = q_2 \quad (8\text{-}499)$$

9 Parsing process

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ue(v), me(v), se(v), te(v) (see subclause 9.1), ce(v) (see subclause 9.2), or ae(v) (see subclause 9.3).

9.1 Parsing process for Exp-Golomb codes

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ue(v), me(v), se(v), or te(v). For syntax elements in subclauses 7.3.4 and 7.3.5, this process is invoked only when entropy_coding_mode_flag is equal to 0.

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

Syntax elements coded as ue(v), me(v), or se(v) are Exp-Golomb-coded. Syntax elements coded as te(v) are truncated Exp-Golomb-coded. The parsing process for these syntax elements begins with reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows:

```
leadingZeroBits = -1;
for( b = 0; !b; leadingZeroBits++ )
    b = read_bits( 1 )
```

The variable codeNum is then assigned as follows:

$$codeNum = 2^{leadingZeroBits} - 1 + read\_bits(leadingZeroBits)$$

where the value returned from read_bits(leadingZeroBits) is interpreted as a binary representation of an unsigned integer with most significant bit written first.

Table 9-1 illustrates the structure of the Exp-Golomb code by separating the bit string into “prefix” and “suffix” bits. The “prefix” bits are those bits that are parsed in the above pseudo-code for the computation of leadingZeroBits, and are shown as either 0 or 1 in the bit string column of Table 9-1. The “suffix” bits are those bits that are parsed in the

computation of codeNum and are shown as x_i in Table 9-1, with i being in the range 0 to leadingZeroBits - 1, inclusive. Each x_i can take on values 0 or 1.

Table 9-1 – Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges (informative)

Bit string form	Range of codeNum
1	0
0 1 x_0	1-2
0 0 1 x_1 x_0	3-6
0 0 0 1 x_2 x_1 x_0	7-14
0 0 0 0 1 x_3 x_2 x_1 x_0	15-30
0 0 0 0 0 1 x_4 x_3 x_2 x_1 x_0	31-62
...	...

Table 9-2 illustrates explicitly the assignment of bit strings to codeNum values.

Table 9-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative)

Bit string	codeNum
1	0
0 1 0	1
0 1 1	2
0 0 1 0 0	3
0 0 1 0 1	4
0 0 1 1 0	5
0 0 1 1 1	6
0 0 0 1 0 0 0	7
0 0 0 1 0 0 1	8
0 0 0 1 0 1 0	9
...	...

Depending on the descriptor, the value of a syntax element is derived as follows.

If the syntax element is coded as ue(v), the value of the syntax element is equal to codeNum.

- Otherwise, if the syntax element is coded as se(v), the value of the syntax element is derived by invoking the mapping process for signed Exp-Golomb codes as specified in subclause 9.1.1 with codeNum as the input.
- Otherwise, if the syntax element is coded as me(v), the value of the syntax element is derived by invoking the mapping process for coded block pattern as specified in subclause 9.1.2 with codeNum as the input.
- Otherwise (the syntax element is coded as te(v)), the range of possible values for the syntax element is determined first. The range of this syntax element may be between 0 and x, with x being greater than or equal to 1 and the range is used in the derivation of the value of the syntax element value as follows
  - If x is greater than 1, codeNum and the value of the syntax element is derived in the same way as for syntax elements coded as ue(v)

– Otherwise (x is equal to 1), the parsing process for codeNum which is equal to the value of the syntax element is given by a process equivalent to:

```
b = read_bits( 1 )
codeNum = !b
```

9.1.1 Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in subclause 9.1.

Output of this process is a value of a syntax element coded as se(v).

The syntax element is assigned to the codeNum by ordering the syntax element by its absolute value in increasing order and representing the positive value for a given absolute value with the lower codeNum. Table 9-3 provides the assignment rule.

Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v)

codeNum	syntax element value
0	0
1	1
2	–1
3	2
4	–2
5	3
6	–3
k	$(-1)^{k+1}$ Ceil(k÷2)

9.1.2 Mapping process for coded block pattern

Input to this process is codeNum as specified in subclause 9.1.

Output of this process is a value of the syntax element coded_block_pattern coded as me(v).

Table 9-4 shows the assignment of coded_block_pattern to codeNum depending on whether the macroblock prediction mode is equal to Intra_4x4, Intra_8x8 or Inter.

Table 9-4 – Assignment of codeNum to values of coded_block_pattern for macroblock prediction modes

(a) ChromaArrayType is equal to 1 or 2

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
0	47	0
1	31	16
2	15	1
3	0	2
4	23	4
5	27	8
6	29	32

(a) ChromaArrayType is equal to 1 or 2

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
7	30	3
8	7	5
9	11	10
10	13	12
11	14	15
12	39	47
13	43	7
14	45	11
15	46	13
16	16	14
17	3	6
18	5	9
19	10	31
20	12	35
21	19	37
22	21	42
23	26	44
24	28	33
25	35	34
26	37	36
27	42	40
28	44	39
29	1	43
30	2	45
31	4	46
32	8	17
33	17	18
34	18	20
35	20	24
36	24	19
37	6	21
38	9	26

(a) ChromaArrayType is equal to 1 or 2

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
39	22	28
40	25	23
41	32	27
42	33	29
43	34	30
44	36	22
45	40	25
46	38	38
47	41	41

(b) ChromaArrayType is equal to 0 or 3

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
0	15	0
1	0	1
2	7	2
3	11	4
4	13	8
5	14	3
6	3	5
7	5	10
8	10	12
9	12	15
10	1	7
11	2	11
12	4	13
13	8	14
14	6	6
15	9	9

9.2 CAVLC parsing process for transform coefficient levels

This process is invoked when parsing syntax elements with descriptor equal to ce(v) in subclause 7.3.5.3.2 and when entropy_coding_mode_flag is equal to 0.

Inputs to this process are bits from slice data, a maximum number of non-zero transform coefficient levels maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx, cb4x4BlkIdx or cr4x4BlkIdx of the current block of transform coefficient levels.

Output of this process is the list coeffLevel containing transform coefficient levels of the luma block with block index luma4x4BlkIdx or the chroma block with block index chroma4x4BlkIdx, cb4x4BlkIdx or cr4x4BlkIdx.

The process is specified in the following ordered steps:

1. All transform coefficient levels, with indices from 0 to maxNumCoeff - 1, in the list coeffLevel are set equal to 0.
2. The total number of non-zero transform coefficient levels TotalCoeff(coeff_token) and the number of trailing one transform coefficient levels TrailingOnes(coeff_token) are derived by parsing coeff_token (see subclause 9.2.1) as follows.
   - If the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is equal to 0, the list coeffLevel containing 0 values is returned and no further step is carried out.
   - Otherwise, the following steps are carried out.
     a. The non-zero transform coefficient levels are derived by parsing trailing_ones_sign_flag, level_prefix, and level_suffix (see subclause 9.2.2).
     b. The runs of zero transform coefficient levels before each non-zero transform coefficient level are derived by parsing total_zeros and run_before (see subclause 9.2.3).
     c. The level and run information are combined into the list coeffLevel (see subclause 9.2.4).

9.2.1 Parsing process for total number of transform coefficient levels and trailing ones

Inputs to this process are bits from slice data, a maximum number of non-zero transform coefficient levels maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx, cb4x4BlkIdx or cr4x4BlkIdx of the current block of transform coefficient levels.

Outputs of this process are TotalCoeff(coeff_token), TrailingOnes(coeff_token), and the variable nC.

The syntax element coeff_token is decoded using one of the six VLCs specified in the six right-most columns of Table 9-5. Each VLC specifies both TotalCoeff(coeff_token) and TrailingOnes(coeff_token) for a given codeword coeff_token. VLC selection is dependent upon a variable nC that is derived as follows.

– If the CAVLC parsing process is invoked for ChromaDCLevel, nC is derived as follows.
  – If ChromaArrayType is equal to 1, nC is set equal to -1,
  – Otherwise (ChromaArrayType is equal to 2), nC is set equal to -2.
– Otherwise, the following applies.
  – When the CAVLC parsing process is invoked for Intra16x16DCLevel, luma4x4BlkIdx is set equal to 0.
  – When the CAVLC parsing process is invoked for CbIntra16x16DCLevel, cb4x4BlkIdx is set equal to 0.
  – When the CAVLC parsing process is invoked for CrIntra16x16DCLevel, cr4x4BlkIdx is set equal to 0.
  – The variables blkA and blkB are derived as follows.
    – If the CAVLC parsing process is invoked for Intra16x16DCLevel, Intra16x16ACLevel, or LumaLevel, the process specified in subclause 6.4.10.4 is invoked with luma4x4BlkIdx as the input, and the output is assigned to mbAddrA, mbAddrB, luma4x4BlkIdxA, and luma4x4BlkIdxB. The 4x4 luma block specified by mbAddrA\luma4x4BlkIdxA is assigned to blkA, and the 4x4 luma block specified by mbAddrB\luma4x4BlkIdxB is assigned to blkB.
    – Otherwise, if the CAVLC parsing process is invoked for CbIntra16x16DCLevel, CbIntra16x16ACLevel, or CbLevel, the process specified in subclause 6.4.10.6 is invoked with cb4x4BlkIdx as the input, and the output is assigned to mbAddrA, mbAddrB, cb4x4BlkIdxA, and cb4x4BlkIdxB. The 4x4 Cb block specified by mbAddrA\cb4x4BlkIdxA is assigned to blkA, and the 4x4 Cb block specified by mbAddrB\cb4x4BlkIdxB is assigned to blkB.

– Otherwise, if the CAVLC parsing process is invoked for CrIntra16x16DCLevel, CrIntra16x16ACLevel, or CrLevel, the process specified in subclause 6.4.10.6 is invoked with cr4x4BlkIdx as the input, and the output is assigned to mbAddrA, mbAddrB, cr4x4BlkIdxA, and cr4x4BlkIdxB. The 4x4 Cr block specified by mbAddrA\cr4x4BlkIdxA is assigned to blkA, and the 4x4 Cr block specified by mbAddrB\cr4x4BlkIdxB is assigned to blkB.

– Otherwise (the CAVLC parsing process is invoked for ChromaACLevel), the process specified in subclause 6.4.10.5 is invoked with chroma4x4BlkIdx as input, and the output is assigned to mbAddrA, mbAddrB, chroma4x4BlkIdxA, and chroma4x4BlkIdxB. The 4x4 chroma block specified by mbAddrA\iCbCr\chroma4x4BlkIdxA is assigned to blkA, and the 4x4 chroma block specified by mbAddrB\iCbCr\chroma4x4BlkIdxB is assigned to blkB.

– Let nA and nB be the number of non-zero transform coefficient levels (given by TotalCoeff(coeff_token)) in the block of transform coefficient levels blkA located to the left of the current block and the block of transform coefficient levels blkB located above the current block, respectively.

– With N replaced by A and B, in mbAddrN, blkN, and nN the following applies.

  – If any of the following conditions is true, nN is set equal to 0.

    – mbAddrN is not available.

    – The current macroblock is coded using an Intra prediction mode, constrained_intra_pred_flag is equal to 1 and mbAddrN is coded using Inter prediction and slice data partitioning is in use (nal_unit_type is in the range of 2 to 4, inclusive).

    – The macroblock mbAddrN has mb_type equal to P_Skip or B_Skip.

    – All AC residual transform coefficient levels of the neighbouring block blkN are equal to 0 due to the corresponding bit of CodedBlockPatternLuma or CodedBlockPatternChroma being equal to 0.

  – Otherwise, if mbAddrN is an I_PCM macroblock, nN is set equal to 16.

  – Otherwise, nN is set equal to the value TotalCoeff(coeff_token) of the neighbouring block blkN.

    NOTE 1 – The values nA and nB that are derived using TotalCoeff(coeff_token) do not include the DC transform coefficient levels in Intra_16x16 macroblocks or DC transform coefficient levels in chroma blocks, because these transform coefficient levels are decoded separately. When the block above or to the left belongs to an Intra_16x16 macroblock, or is a chroma block, nA and nB is the number of decoded non-zero AC transform coefficient levels.

    NOTE 2 – When parsing for Intra16x16DCLevel, CbIntra16x16DCLevel, or CrIntra16x16DCLevel, the values nA and nB are based on the number of non-zero transform coefficient levels in adjacent 4x4 blocks and not on the number of non-zero DC transform coefficient levels in adjacent 16x16 blocks.

– Given the values of nA and nB, the variable nC is derived as follows.

  – If both mbAddrA and mbAddrB are available, the variable nC is set equal to (nA + nB + 1) >> 1.

  – Otherwise (mbAddrA is not available or mbAddrB is not available), the variable nC is set equal to nA + nB.

The value of TotalCoeff(coeff_token) resulting from decoding coeff_token shall be in the range of 0 to maxNumCoeff, inclusive.

Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token)

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = -1	nC = = -2
0	0	1	11	1111	0000 11	01	1
0	1	0001 01	0010 11	0011 11	0000 00	0001 11	0001 111
1	1	01	10	1110	0000 01	1	01
0	2	0000 0111	0001 11	0010 11	0001 00	0001 00	0001 110
1	2	0001 00	0011 1	0111 1	0001 01	0001 10	0001 101
2	2	001	011	1101	0001 10	001	001
0	3	0000 0011 1	0000 111	0010 00	0010 00	0000 11	0000 0011 1
1	3	0000 0110	0010 10	0110 0	0010 01	0000 011	0001 100
2	3	0000 101	0010 01	0111 0	0010 10	0000 010	0001 011
3	3	0001 1	0101	1100	0010 11	0001 01	0000 1
0	4	0000 0001 11	0000 0111	0001 111	0011 00	0000 10	0000 0011 0
1	4	0000 0011 0	0001 10	0101 0	0011 01	0000 0011	0000 0010 1
2	4	0000 0101	0001 01	0101 1	0011 10	0000 0010	0001 010
3	4	0000 11	0100	1011	0011 11	0000 000	0000 01
0	5	0000 0000 111	0000 0100	0001 011	0100 00	-	0000 0001 11
1	5	0000 0001 10	0000 110	0100 0	0100 01	-	0000 0001 10
2	5	0000 0010 1	0000 101	0100 1	0100 10	-	0000 0010 0
3	5	0000 100	0011 0	1010	0100 11	-	0001 001
0	6	0000 0000 0111 1	0000 0011 1	0001 001	0101 00	-	0000 0000 111
1	6	0000 0000 110	0000 0110	0011 10	0101 01	-	0000 0000 110
2	6	0000 0001 01	0000 0101	0011 01	0101 10	-	0000 0001 01
3	6	0000 0100	0010 00	1001	0101 11	-	0001 000
0	7	0000 0000 0101 1	0000 0001 111	0001 000	0110 00	-	0000 0000 0111
1	7	0000 0000 0111 0	0000 0011 0	0010 10	0110 01	-	0000 0000 0110
2	7	0000 0000 101	0000 0010 1	0010 01	0110 10	-	0000 0000 101
3	7	0000 0010 0	0001 00	1000	0110 11	-	0000 0001 00
0	8	0000 0000 0100 0	0000 0001 011	0000 1111	0111 00	-	0000 0000 0011 1
1	8	0000 0000 0101 0	0000 0001 110	0001 110	0111 01	-	0000 0000 0101
2	8	0000 0000 0110 1	0000 0001 101	0001 101	0111 10	-	0000 0000 0100
3	8	0000 0001 00	0000 100	0110 1	0111 11	-	0000 0000 100
0	9	0000 0000 0011 11	0000 0000 1111	0000 1011	1000 00	-	
1	9	0000 0000 0011 10	0000 0001 010	0000 1110	1000 01	-	
2	9	0000 0000 0100 1	0000 0001 001	0001 010	1000 10	-	
3	9	0000 0000 100	0000 0010 0	0011 00	1000 11	-	

Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token)

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = -1	nC = = -2
0	10	0000 0000 0010 11	0000 0000 1011	0000 0111 1	1001 00	-	
1	10	0000 0000 0010 10	0000 0000 1110	0000 1010	1001 01	-	
2	10	0000 0000 0011 01	0000 0000 1101	0000 1101	1001 10	-	
3	10	0000 0000 0110 0	0000 0001 100	0001 100	1001 11	-	
0	11	0000 0000 0001 111	0000 0000 1000	0000 0101 1	1010 00	-	
1	11	0000 0000 0001 110	0000 0000 1010	0000 0111 0	1010 01	-	
2	11	0000 0000 0010 01	0000 0000 1001	0000 1001	1010 10	-	
3	11	0000 0000 0011 00	0000 0001 000	0000 1100	1010 11	-	
0	12	0000 0000 0001 011	0000 0000 0111 1	0000 0100 0	1011 00	-	
1	12	0000 0000 0001 010	0000 0000 0111 0	0000 0101 0	1011 01	-	
2	12	0000 0000 0001 101	0000 0000 0110 1	0000 0110 1	1011 10	-	
3	12	0000 0000 0010 00	0000 0000 1100	0000 1000	1011 11	-	
0	13	0000 0000 0000 1111	0000 0000 0101 1	0000 0011 01	1100 00	-	
1	13	0000 0000 0000 001	0000 0000 0101 0	0000 0011 1	1100 01	-	
2	13	0000 0000 0001 001	0000 0000 0100 1	0000 0100 1	1100 10	-	
3	13	0000 0000 0001 100	0000 0000 0110 0	0000 0110 0	1100 11	-	
0	14	0000 0000 0000 1011	0000 0000 0011 1	0000 0010 01	1101 00	-	
1	14	0000 0000 0000 1110	0000 0000 0010 11	0000 0011 00	1101 01	-	
2	14	0000 0000 0000 1101	0000 0000 0011 0	0000 0010 11	1101 10	-	
3	14	0000 0000 0001 000	0000 0000 0100 0	0000 0010 10	1101 11	-	
0	15	0000 0000 0000 0111	0000 0000 0010 01	0000 0001 01	1110 00	-	
1	15	0000 0000 0000 1010	0000 0000 0010 00	0000 0010 00	1110 01	-	
2	15	0000 0000 0000 1001	0000 0000 0010 10	0000 0001 11	1110 10	-	
3	15	0000 0000 0000 1100	0000 0000 0000 1	0000 0001 10	1110 11	-	
0	16	0000 0000 0000 0100	0000 0000 0001 11	0000 0000 01	1111 00	-	
1	16	0000 0000 0000 0110	0000 0000 0001 10	0000 0001 00	1111 01	-	
2	16	0000 0000 0000 0101	0000 0000 0001 01	0000 0000 11	1111 10	-	
3	16	0000 0000 0000 1000	0000 0000 0001 00	0000 0000 10	1111 11	-	

9.2.2 Parsing process for level information

Inputs to this process are bits from slice data, the number of non-zero transform coefficient levels TotalCoeff(coeff_token), and the number of trailing one transform coefficient levels TrailingOnes(coeff_token).

Output of this process is a list with name level containing transform coefficient levels.

Initially an index i is set equal to 0. Then the following procedure is iteratively applied TrailingOnes(coeff_token) times to decode the trailing one transform coefficient levels (if any):

– A 1-bit syntax element trailing_ones_sign_flag is decoded and evaluated as follows.
  – If trailing_ones_sign_flag is equal to 0, the value +1 is assigned to level[i].
  – Otherwise (trailing_ones_sign_flag is equal to 1), the value -1 is assigned to level[i].
– The index i is incremented by 1.

Following the decoding of the trailing one transform coefficient levels, a variable suffixLength is initialised as follows.

– If TotalCoeff(coeff_token) is greater than 10 and TrailingOnes(coeff_token) is less than 3, suffixLength is set equal to 1.
– Otherwise (TotalCoeff(coeff_token) is less than or equal to 10 or TrailingOnes(coeff_token) is equal to 3), suffixLength is set equal to 0.

The following procedure is then applied iteratively (TotalCoeff(coeff_token) – TrailingOnes(coeff_token)) times to decode the remaining levels (if any):

– The syntax element level_prefix is decoded as specified in subclause 9.2.2.1.
– The variable levelSuffixSize is set equal to the variable suffixLength with the exception of the following two cases.
– When level_prefix is equal to 14 and suffixLength is equal to 0, levelSuffixSize is set equal to 4.
– When level_prefix is greater than or equal to 15, levelSuffixSize is set equal to level_prefix - 3.
– The syntax element level_suffix is decoded as follows.
  – If levelSuffixSize is greater than 0, the syntax element level_suffix is decoded as unsigned integer representation u(v) with levelSuffixSize bits.
  – Otherwise (levelSuffixSize is equal to 0), the syntax element level_suffix is inferred to be equal to 0.
– A variable levelCode is set equal to (Min(15, level_prefix) << suffixLength) + level_suffix.
– When level_prefix is greater than or equal to 15 and suffixLength is equal to 0, levelCode is incremented by 15.
– When level_prefix is greater than or equal to 16, levelCode is incremented by (1<<(level_prefix – 3)) – 4096.
– When the index i is equal to TrailingOnes(coeff_token) and TrailingOnes(coeff_token) is less than 3, levelCode is incremented by 2.
– The variable level[i] is derived as follows.
  – If levelCode is an even number, the value (levelCode + 2) >> 1 is assigned to level[i].
  – Otherwise (levelCode is an odd number), the value (-levelCode – 1) >> 1 is assigned to level[i].
– When suffixLength is equal to 0, suffixLength is set equal to 1.
– When the absolute value of level[i] is greater than (3 << (suffixLength – 1)) and suffixLength is less than 6, suffixLength is incremented by 1.
– The index i is incremented by 1.

9.2.2.1 Parsing process for level_prefix

Inputs to this process are bits from slice data.

Output of this process is level_prefix.

The parsing process for this syntax element consists in reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows:

```
leadingZeroBits = -1
for( b = 0; !b; leadingZeroBits++ )
    b = read_bits( 1 )
level_prefix = leadingZeroBits
```

Table 9-6 illustrates the codeword table for level_prefix.

Table 9-6 – Codeword table for level_prefix (informative)

level_prefix	bit string
0	1
1	01
2	001
3	0001
4	0000 1
5	0000 01
6	0000 001
7	0000 0001
8	0000 0000 1
9	0000 0000 01
10	0000 0000 001
11	0000 0000 0001
12	0000 0000 0000 1
13	0000 0000 0000 01
14	0000 0000 0000 001
15	0000 0000 0000 0001
…	…

9.2.3 Parsing process for run information

Inputs to this process are bits from slice data, the number of non-zero transform coefficient levels TotalCoeff(coeff_token), and the maximum number of non-zero transform coefficient levels maxNumCoeff.

Output of this process is a list of runs of zero transform coefficient levels preceding non-zero transform coefficient levels called run.

Initially, an index i is set equal to 0.

The variable zerosLeft is derived as follows.

– If the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is equal to the maximum number of non-zero transform coefficient levels maxNumCoeff, a variable zerosLeft is set equal to 0.

– Otherwise (the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is less than the maximum number of non-zero transform coefficient levels maxNumCoeff), total_zeros is decoded and zerosLeft is set equal to its value.

Let the variable tzVlcIndex be equal to TotalCoeff(coeff_token).

The VLC used to decode total_zeros is derived as follows:

If maxNumCoeff is equal to 4, one of the VLCs specified in Table 9-9 (a) is used.

– Otherwise, if maxNumCoeff is equal to 8, one of the VLCs specified in Table 9-9 (b) is used.

– Otherwise (maxNumCoeff is not equal to 4 and not equal to 8), VLCs from Table 9-7 and 9-8 are used.

The following procedure is then applied iteratively (TotalCoeff(coeff_token) – 1) times:

– The variable run[i] is derived as follows.

  – If zerosLeft is greater than zero, a value run_before is decoded based on Table 9-10 and zerosLeft. run[i] is set equal to run_before.

– Otherwise (zerosLeft is equal to 0), run[i] is set equal to 0.

– The value of run[i] is subtracted from zerosLeft and the result assigned to zerosLeft. The result of the subtraction shall be greater than or equal to 0.

– The index i is incremented by 1.

Finally the value of zerosLeft is assigned to run[i].

Table 9-7 – total_zeros tables for 4x4 blocks with tzVlcIndex 1 to 7

total_zeros	tzVlcIndex						
	1	2	3	4	5	6	7
0	1	111	0101	0001 1	0101	0000 01	0000 01
1	011	110	111	111	0100	0000 1	0000 1
2	010	101	110	0101	0011	111	101
3	0011	100	101	0100	111	110	100
4	0010	011	0100	110	110	101	011
5	0001 1	0101	0011	101	101	100	11
6	0001 0	0100	100	100	100	011	010
7	0000 11	0011	011	0011	011	010	0001
8	0000 10	0010	0010	011	0010	0001	001
9	0000 011	0001 1	0001 1	0010	0000 1	001	0000 00
10	0000 010	0001 0	0001 0	0001 0	0001	0000 00	
11	0000 0011	0000 11	0000 01	0000 1	0000 0		
12	0000 0010	0000 10	0000 1	0000 0			
13	0000 0001 1	0000 01	0000 00				
14	0000 0001 0	0000 00					
15	0000 0000 1						

Table 9-8 – total_zeros tables for 4x4 blocks with tzVlcIndex 8 to 15

total_zeros	tzVlcIndex							
	8	9	10	11	12	13	14	15
0	0000 01	0000 01	0000 1	0000	0000	000	00	0
1	0001	0000 00	0000 0	0001	0001	001	01	1
2	0000 1	0001	001	001	01	1	1	
3	011	11	11	010	1	01		
4	11	10	10	1	001			
5	10	001	01	011				
6	010	01	0001					
7	001	0000 1						
8	0000 00							

Table 9-9 – total_zeros tables for chroma DC 2x2 and 2x4 blocks

(a) Chroma DC 2x2 block (4:2:0 chroma sampling)

total_zeros	tzVlcIndex		
	1	2	3
0	1	1	1
1	01	01	0
2	001	00	
3	000		

(b) Chroma DC 2x4 block (4:2:2 chroma sampling)

total_zeros	tzVlcIndex						
	1	2	3	4	5	6	7
0	1	000	000	110	00	00	0
1	010	01	001	00	01	01	1
2	011	001	01	01	10	1	
3	0010	100	10	10	11		
4	0011	101	110	111			
5	0001	110	111				
6	0000 1	111					
7	0000 0						

Table 9-10 – Tables for run_before

run_before	zerosLeft						
	1	2	3	4	5	6	>6
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	-	00	01	01	011	001	101
3	-	-	00	001	010	011	100
4	-	-	-	000	001	010	011
5	-	-	-	-	000	101	010
6	-	-	-	-	-	100	001
7	-	-	-	-	-	-	0001
8		-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001

9.2.4 Combining level and run information

Input to this process are a list of transform coefficient levels called level, a list of runs called run, and the number of non-zero transform coefficient levels TotalCoeff(coeff_token).

Output of this process is an list coeffLevel of transform coefficient levels.

A variable coeffNum is set equal to -1 and an index i is set equal to (TotalCoeff(coeff_token) – 1). The following procedure is iteratively applied TotalCoeff(coeff_token) times:

– coeffNum is incremented by run[i] + 1.

– coeffLevel[coeffNum] is set equal to level[i].

– The index i is decremented by 1.

9.3 CABAC parsing process for slice data

This process is invoked when parsing syntax elements with descriptor ae(v) in subclauses 7.3.4 and 7.3.5 when entropy_coding_mode_flag is equal to 1.

Inputs to this process are a request for a value of a syntax element and values of prior parsed syntax elements.

Output of this process is the value of the syntax element.

When starting the parsing of the slice data of a slice in subclause 7.3.4, the initialisation process of the CABAC parsing process is invoked as specified in subclause 9.3.1.

The parsing of syntax elements proceeds as follows:

For each requested value of a syntax element a binarization is derived as described in subclause 9.3.2.

The binarization for the syntax element and the sequence of parsed bins determines the decoding process flow as described in subclause 9.3.3.

For each bin of the binarization of the syntax element, which is indexed by the variable binIdx, a context index ctxIdx is derived as specified in subclause 9.3.3.1.

For each ctxIdx the arithmetic decoding process is invoked as specified in subclause 9.3.3.2.

The resulting sequence (b_0 .. b_{binIdx}) of parsed bins is compared to the set of bin strings given by the binarization process after decoding of each bin. When the sequence matches a bin string in the given set, the corresponding value is assigned to the syntax element.

In case the request for a value of a syntax element is processed for the syntax element mb_type and the decoded value of mb_type is equal to I_PCM, the decoding engine is initialised after the decoding of any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data as specified in subclause 9.3.1.2.

The whole CABAC parsing process is illustrated in the flowchart of Figure 9-1 with the abbreviation SE for syntax element.

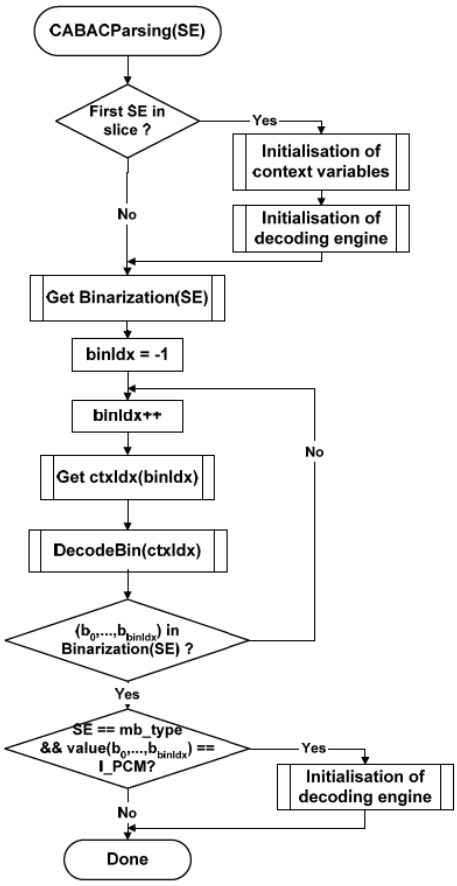


Figure 9-1 – Illustration of CABAC parsing process for a syntax element SE (informative)

9.3.1 Initialisation process

Outputs of this process are initialised CABAC internal variables.

The processes in subclauses 9.3.1.1 and 9.3.1.2 are invoked when starting the parsing of the slice data of a slice in subclause 7.3.4.

The process in subclause 9.3.1.2 is also invoked after decoding any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data for a macroblock of type I_PCM.

9.3.1.1 Initialisation process for context variables

Outputs of this process are the initialised CABAC context variables indexed by ctxIdx.

Table 9-12 to Table 9-33 contain the values of the variables n and m used in the initialisation of context variables that are assigned to all syntax elements in subclauses 7.3.4 and 7.3.5 except for the end-of-slice flag.

For each context variable, the two variables pStateIdx and valMPS are initialised.

NOTE 1 – The variable pStateIdx corresponds to a probability state index and the variable valMPS corresponds to the value of the most probable symbol as further described in subclause 9.3.3.2.

The two values assigned to pStateIdx and valMPS for the initialisation are derived from $SliceQP_Y$, which is derived in Equation 7-28. Given the two table entries (m, n),

1. preCtxState = Clip3(1, 126, ((m * Clip3(0, 51, $SliceQP_Y$)) >> 4) + n)
2. if(preCtxState <= 63) {

   pStateIdx = 63 - preCtxState

   valMPS = 0

   } else {

   pStateIdx = preCtxState - 64

   valMPS = 1

   }

In Table 9-11, the ctxIdx for which initialisation is needed for each of the slice types are listed. Also listed is the table number that includes the values of m and n needed for the initialisation. For P, SP and B slice type, the initialisation depends also on the value of the cabac_init_idc syntax element. Note that the syntax element names do not affect the initialisation process.

Table 9-11 – Association of ctxIdx and syntax elements for each slice type in the initialisation process

	Syntax element	Table	Slice type			
			SI	I	P, SP	B
slice_data( )	mb_skip_flag	Table 9-13 Table 9-14			11-13	24-26
	mb_field_decoding_flag	Table 9-18	70-72	70-72	70-72	70-72
macroblock_layer( )	mb_type	Table 9-12 Table 9-13 Table 9-14	0-10	3-10	14-20	27-35
	transform_size_8x8_flag	Table 9-16	na	399-401	399-401	399-401
	coded_block_pattern (luma)	Table 9-18	73-76	73-76	73-76	73-76
	coded_block_pattern (chroma)	Table 9-18	77-84	77-84	77-84	77-84
	mb_qp_delta	Table 9-17	60-63	60-63	60-63	60-63
mb_pred( )	prev_intra4x4_pred_mode_flag	Table 9-17	68	68	68	68
	rem_intra4x4_pred_mode	Table 9-17	69	69	69	69
	prev_intra8x8_pred_mode_flag	Table 9-17	na	68	68	68
	rem_intra8x8_pred_mode	Table 9-17	na	69	69	69
	intra_chroma_pred_mode	Table 9-17	64-67	64-67	64-67	64-67

Table 9-11 – Association of ctxIdx and syntax elements for each slice type in the initialisation process

	Syntax element	Table	Slice type			
			SI	I	P, SP	B
mb_pred( ) and sub_mb_pred( )	ref_idx_l0	Table 9-16			54-59	54-59
	ref_idx_l1	Table 9-16				54-59
	mvd_l0[ ][ ][0]	Table 9-15			40-46	40-46
	mvd_l1[ ][ ][0]	Table 9-15				40-46
	mvd_l0[ ][ ][1]	Table 9-15			47-53	47-53
	mvd_l1[ ][ ][1]	Table 9-15				47-53
sub_mb_pred( )	sub_mb_type	Table 9-13 Table 9-14			21-23	36-39
residual_block_cabac( )	coded_block_flag	Table 9-18 Table 9-25 Table 9-33	85-104 460-483	85-104 460-483 1012-1023	85-104 460-483 1012-1023	85-104 460-483 1012-1023
	significant_coeff_flag[ ]	Table 9-19 Table 9-22 Table 9-24 Table 9-24 Table 9-26 Table 9-30 Table 9-28 Table 9-29	105-165 277-337	105-165 277-337 402-416 436-450 484-571 776-863 660-689 718-747	105-165 277-337 402-416 436-450 484-571 776-863 660-689 718-747	105-165 277-337 402-416 436-450 484-571 776-863 660-689 718-747
	last_significant_coeff_flag[ ]	Table 9-20 Table 9-23 Table 9-24 Table 9-24 Table 9-27 Table 9-31 Table 9-28 Table 9-29	166-226 338-398	166-226 338-398 417-425 451-459 572-659 864-951 690-707 748-765	166-226 338-398 417-425 451-459 572-659 864-951 690-707 748-765	166-226 338-398 417-425 451-459 572-659 864-951 690-707 748-765
	coeff_abs_level_minus1[ ]	Table 9-21 Table 9-24 Table 9-32 Table 9-28 Table 9-29	227-275	227-275 426-435 952-1011 708-717 766-775	227-275 426-435 952-1011 708-717 766-775	227-275 426-435 952-1011 708-717 766-775

NOTE 2 – ctxIdx equal to 276 is associated with the end_of_slice_flag and the bin of mb_type, which specifies the I_PCM macroblock type. The decoding process specified in subclause 9.3.3.2.4 applies to ctxIdx equal to 276. This decoding process, however, may also be implemented by using the decoding process specified in subclause 9.3.3.2.1. In this case, the initial values associated with ctxIdx equal to 276 are specified to be pStateIdx = 63 and valMPS = 0, where pStateIdx = 63 represents a non-adapting probability state.

Table 9-12 – Values of variables m and n for ctxIdx from 0 to 10

Initialisation variables	ctxIdx										
	0	1	2	3	4	5	6	7	8	9	10
m	20	2	3	20	2	3	-28	-23	-6	-1	7
n	-15	54	74	-15	54	74	127	104	53	54	51

Table 9-13 – Values of variables m and n for ctxIdx from 11 to 23

Value of cabac_init_idc	Initialisation variables	ctxIdx												
		11	12	13	14	15	16	17	18	19	20	21	22	23
0	m	23	23	21	1	0	-37	5	-13	-11	1	12	-4	17
	n	33	2	0	9	49	118	57	78	65	62	49	73	50
1	m	22	34	16	-2	4	-29	2	-6	-13	5	9	-3	10
	n	25	0	0	9	41	118	65	71	79	52	50	70	54
2	m	29	25	14	-10	-3	-27	26	-4	-24	5	6	-17	14
	n	16	0	0	51	62	99	16	85	102	57	57	73	57

Table 9-14 – Values of variables m and n for ctxIdx from 24 to 39

Value of cabac_init_idc	Initialisation variables	ctxIdx															
		24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
0	m	18	9	29	26	16	9	-46	-20	1	-13	-11	1	-6	-17	-6	9
	n	64	43	0	67	90	104	127	104	67	78	65	62	86	95	61	45
1	m	26	19	40	57	41	26	-45	-15	-4	-6	-13	5	6	-13	0	8
	n	34	22	0	2	36	69	127	101	76	71	79	52	69	90	52	43
2	m	20	20	29	54	37	12	-32	-22	-2	-4	-24	5	-6	-14	-6	4
	n	40	10	0	0	42	97	127	117	74	85	102	57	93	88	44	55

Table 9-15 – Values of variables m and n for ctxIdx from 40 to 53

Value of cabac_init_idc	Initialisation variables	ctxIdx													
		40	41	42	43	44	45	46	47	48	49	50	51	52	53
0	m	-3	-6	-11	6	7	-5	2	0	-3	-10	5	4	-3	0
	n	69	81	96	55	67	86	88	58	76	94	54	69	81	88
1	m	-2	-5	-10	2	2	-3	-3	1	-3	-6	0	-3	-7	-5
	n	69	82	96	59	75	87	100	56	74	85	59	81	86	95
2	m	-11	-15	-21	19	20	4	6	1	-5	-13	5	6	-3	-1
	n	89	103	116	57	58	84	96	63	85	106	63	75	90	101

Table 9-16 – Values of variables m and n for ctxIdx from 54 to 59, and 399 to 401

Value of cabac_init_idc	Initialisation variables	ctxIdx								
		54	55	56	57	58	59	399	400	401
I slices	m	na	na	na	na	na	na	31	31	25
	n	na	na	na	na	na	na	21	31	50
0	m	-7	-5	-4	-5	-7	1	12	11	14
	n	67	74	74	80	72	58	40	51	59
1	m	-1	-1	1	-2	-5	0	25	21	21
	n	66	77	70	86	72	61	32	49	54
2	m	3	-4	-2	-12	-7	1	21	19	17
	n	55	79	75	97	50	60	33	50	61

Table 9-17 – Values of variables m and n for ctxIdx from 60 to 69

Initialisation variables	ctxIdx									
	60	61	62	63	64	65	66	67	68	69
m	0	0	0	0	-9	4	0	-7	13	3
n	41	63	63	63	83	86	97	72	41	62

Table 9-18 – Values of variables m and n for ctxIdx from 70 to 104

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
70	0	11	0	45	13	15	7	34	**88**	-11	115	-13	108	-4	92	5	78
71	1	55	-4	78	7	51	-9	88	**89**	-12	63	-3	46	0	39	-6	55
72	0	69	-3	96	2	80	-20	127	**90**	-2	68	-1	65	0	65	4	61
73	-17	127	-27	126	-39	127	-36	127	**91**	-15	84	-1	57	-15	84	-14	83
74	-13	102	-28	98	-18	91	-17	91	**92**	-13	104	-9	93	-35	127	-37	127
75	0	82	-25	101	-17	96	-14	95	**93**	-3	70	-3	74	-2	73	-5	79
76	-7	74	-23	67	-26	81	-25	84	**94**	-8	93	-9	92	-12	104	-11	104
77	-21	107	-28	82	-35	98	-25	86	**95**	-10	90	-8	87	-9	91	-11	91
78	-27	127	-20	94	-24	102	-12	89	**96**	-30	127	-23	126	-31	127	-30	127
79	-31	127	-16	83	-23	97	-17	91	**97**	-1	74	5	54	3	55	0	65
80	-24	127	-22	110	-27	119	-31	127	**98**	-6	97	6	60	7	56	-2	79
81	-18	95	-21	91	-24	99	-14	76	**99**	-7	91	6	59	7	55	0	72
82	-27	127	-18	102	-21	110	-18	103	**100**	-20	127	6	69	8	61	-4	92
83	-21	114	-13	93	-18	102	-13	90	**101**	-4	56	-1	48	-3	53	-6	56
84	-30	127	-29	127	-36	127	-37	127	**102**	-5	82	0	68	0	68	3	68
85	-17	123	-7	92	0	80	11	80	**103**	-7	76	-4	69	-7	74	-8	71
86	-12	115	-5	89	-5	89	5	76	**104**	-22	125	-8	88	-9	88	-13	98
87	-16	122	-7	96	-7	94	2	84									

Table 9-19 – Values of variables m and n for ctxIdx from 105 to 165

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
105	-7	93	-2	85	-13	103	-4	86	136	-13	101	5	53	0	58	-5	75
106	-11	87	-6	78	-13	91	-12	88	137	-13	91	-2	61	-1	60	-8	80
107	-3	77	-1	75	-9	89	-5	82	138	-12	94	0	56	-3	61	-21	83
108	-5	71	-7	77	-14	92	-3	72	139	-10	88	0	56	-8	67	-21	64
109	-4	63	2	54	-8	76	-4	67	140	-16	84	-13	63	-25	84	-13	31
110	-4	68	5	50	-12	87	-8	72	141	-10	86	-5	60	-14	74	-25	64
111	-12	84	-3	68	-23	110	-16	89	142	-7	83	-1	62	-5	65	-29	94
112	-7	62	1	50	-24	105	-9	69	143	-13	87	4	57	5	52	9	75
113	-7	65	6	42	-10	78	-1	59	144	-19	94	-6	69	2	57	17	63
114	8	61	-4	81	-20	112	5	66	145	1	70	4	57	0	61	-8	74
115	5	56	1	63	-17	99	4	57	146	0	72	14	39	-9	69	-5	35
116	-2	66	-4	70	-78	127	-4	71	147	-5	74	4	51	-11	70	-2	27
117	1	64	0	67	-70	127	-2	71	148	18	59	13	68	18	55	13	91
118	0	61	2	57	-50	127	2	58	149	-8	102	3	64	-4	71	3	65
119	-2	78	-2	76	-46	127	-1	74	150	-15	100	1	61	0	58	-7	69
120	1	50	11	35	-4	66	-4	44	151	0	95	9	63	7	61	8	77
121	7	52	4	64	-5	78	-1	69	152	-4	75	7	50	9	41	-10	66
122	10	35	1	61	-4	71	0	62	153	2	72	16	39	18	25	3	62
123	0	44	11	35	-8	72	-7	51	154	-11	75	5	44	9	32	-3	68
124	11	38	18	25	2	59	-4	47	155	-3	71	4	52	5	43	-20	81
125	1	45	12	24	-1	55	-6	42	156	15	46	11	48	9	47	0	30
126	0	46	13	29	-7	70	-3	41	157	-13	69	-5	60	0	44	1	7
127	5	44	13	36	-6	75	-6	53	158	0	62	-1	59	0	51	-3	23
128	31	17	-10	93	-8	89	8	76	159	0	65	0	59	2	46	-21	74
129	1	51	-7	73	-34	119	-9	78	160	21	37	22	33	19	38	16	66
130	7	50	-2	73	-3	75	-11	83	161	-15	72	5	44	-4	66	-23	124
131	28	19	13	46	32	20	9	52	162	9	57	14	43	15	38	17	37
132	16	33	9	49	30	22	0	67	163	16	54	-1	78	12	42	44	-18
133	14	62	-7	100	-44	127	-5	90	164	0	62	0	60	9	34	50	-34
134	-13	108	9	53	0	54	1	67	165	12	72	9	69	0	89	-22	127
135	-15	100	2	53	-5	61	-15	72									

Table 9-20 – Values of variables m and n for ctxIdx from 166 to 226

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
166	24	0	11	28	4	45	4	39	**197**	26	-17	28	3	36	-28	28	-3
167	15	9	2	40	10	28	0	42	**198**	30	-25	28	4	38	-28	24	10
168	8	25	3	44	10	31	7	34	**199**	28	-20	32	0	38	-27	27	0
169	13	18	0	49	33	-11	11	29	**200**	33	-23	34	-1	34	-18	34	-14
170	15	9	0	46	52	-43	8	31	**201**	37	-27	30	6	35	-16	52	-44
171	13	19	2	44	18	15	6	37	**202**	33	-23	30	6	34	-14	39	-24
172	10	37	2	51	28	0	7	42	**203**	40	-28	32	9	32	-8	19	17
173	12	18	0	47	35	-22	3	40	**204**	38	-17	31	19	37	-6	31	25
174	6	29	4	39	38	-25	8	33	**205**	33	-11	26	27	35	0	36	29
175	20	33	2	62	34	0	13	43	**206**	40	-15	26	30	30	10	24	33
176	15	30	6	46	39	-18	13	36	**207**	41	-6	37	20	28	18	34	15
177	4	45	0	54	32	-12	4	47	**208**	38	1	28	34	26	25	30	20
178	1	58	3	54	102	-94	3	55	**209**	41	17	17	70	29	41	22	73
179	0	62	2	58	0	0	2	58	**210**	30	-6	1	67	0	75	20	34
180	7	61	4	63	56	-15	6	60	**211**	27	3	5	59	2	72	19	31
181	12	38	6	51	33	-4	8	44	**212**	26	22	9	67	8	77	27	44
182	11	45	6	57	29	10	11	44	**213**	37	-16	16	30	14	35	19	16
183	15	39	7	53	37	-5	14	42	**214**	35	-4	18	32	18	31	15	36
184	11	42	6	52	51	-29	7	48	**215**	38	-8	18	35	17	35	15	36
185	13	44	6	55	39	-9	4	56	**216**	38	-3	22	29	21	30	21	28
186	16	45	11	45	52	-34	4	52	**217**	37	3	24	31	17	45	25	21
187	12	41	14	36	69	-58	13	37	**218**	38	5	23	38	20	42	30	20
188	10	49	8	53	67	-63	9	49	**219**	42	0	18	43	18	45	31	12
189	30	34	-1	82	44	-5	19	58	**220**	35	16	20	41	27	26	27	16
190	18	42	7	55	32	7	10	48	**221**	39	22	11	63	16	54	24	42
191	10	55	-3	78	55	-29	12	45	**222**	14	48	9	59	7	66	0	93
192	17	51	15	46	32	1	0	69	**223**	27	37	9	64	16	56	14	56
193	17	46	22	31	0	0	20	33	**224**	21	60	-1	94	11	73	15	57
194	0	89	-1	84	27	36	8	63	**225**	12	68	-2	89	10	67	26	38
195	26	-19	25	7	33	-25	35	-18	**226**	2	97	-9	108	-10	116	-24	127
196	22	-17	30	-7	34	-30	33	-25									

Table 9-21 – Values of variables m and n for ctxIdx from 227 to 275

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
227	-3	71	-6	76	-23	112	-24	115	252	-12	73	-6	55	-16	72	-14	75
228	-6	42	-2	44	-15	71	-22	82	253	-8	76	0	58	-7	69	-10	79
229	-5	50	0	45	-7	61	-9	62	254	-7	80	0	64	-4	69	-9	83
230	-3	54	0	52	0	53	0	53	255	-9	88	-3	74	-5	74	-12	92
231	-2	62	-3	64	-5	66	0	59	256	-17	110	-10	90	-9	86	-18	108
232	0	58	-2	59	-11	77	-14	85	257	-11	97	0	70	2	66	-4	79
233	1	63	-4	70	-9	80	-13	89	258	-20	84	-4	29	-9	34	-22	69
234	-2	72	-4	75	-9	84	-13	94	259	-11	79	5	31	1	32	-16	75
235	-1	74	-8	82	-10	87	-11	92	260	-6	73	7	42	11	31	-2	58
236	-9	91	-17	102	-34	127	-29	127	261	-4	74	1	59	5	52	1	58
237	-5	67	-9	77	-21	101	-21	100	262	-13	86	-2	58	-2	55	-13	78
238	-5	27	3	24	-3	39	-14	57	263	-13	96	-3	72	-2	67	-9	83
239	-3	39	0	42	-5	53	-12	67	264	-11	97	-3	81	0	73	-4	81
240	-2	44	0	48	-7	61	-11	71	265	-19	117	-11	97	-8	89	-13	99
241	0	46	0	55	-11	75	-10	77	266	-8	78	0	58	3	52	-13	81
242	-16	64	-6	59	-15	77	-21	85	267	-5	33	8	5	7	4	-6	38
243	-8	68	-7	71	-17	91	-16	88	268	-4	48	10	14	10	8	-13	62
244	-10	78	-12	83	-25	107	-23	104	269	-2	53	14	18	17	8	-6	58
245	-6	77	-11	87	-25	111	-15	98	270	-3	62	13	27	16	19	-2	59
246	-10	86	-30	119	-28	122	-37	127	271	-13	71	2	40	3	37	-16	73
247	-12	92	1	58	-11	76	-10	82	272	-10	79	0	58	-1	61	-10	76
248	-15	55	-3	29	-10	44	-8	48	273	-12	86	-3	70	-5	73	-13	86
249	-10	60	-1	36	-10	52	-8	61	274	-13	90	-6	79	-1	70	-9	83
250	-6	62	1	38	-10	57	-8	66	275	-14	97	-8	85	-4	78	-10	87
251	-4	65	2	43	-9	58	-7	70									

Table 9-22 – Values of variables m and n for ctxIdx from 277 to 337

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
277	-6	93	-13	106	-21	126	-22	127	308	-16	96	-1	51	-16	77	-10	67
278	-6	84	-16	106	-23	124	-25	127	309	-7	88	7	49	-2	64	1	68
279	-8	79	-10	87	-20	110	-25	120	310	-8	85	8	52	2	61	0	77
280	0	66	-21	114	-26	126	-27	127	311	-7	85	9	41	-6	67	2	64
281	-1	71	-18	110	-25	124	-19	114	312	-9	85	6	47	-3	64	0	68
282	0	62	-14	98	-17	105	-23	117	313	-13	88	2	55	2	57	-5	78
283	-2	60	-22	110	-27	121	-25	118	314	4	66	13	41	-3	65	7	55
284	-2	59	-21	106	-27	117	-26	117	315	-3	77	10	44	-3	66	5	59
285	-5	75	-18	103	-17	102	-24	113	316	-3	76	6	50	0	62	2	65
286	-3	62	-21	107	-26	117	-28	118	317	-6	76	5	53	9	51	14	54
287	-4	58	-23	108	-27	116	-31	120	318	10	58	13	49	-1	66	15	44
288	-9	66	-26	112	-33	122	-37	124	319	-1	76	4	63	-2	71	5	60
289	-1	79	-10	96	-10	95	-10	94	320	-1	83	6	64	-2	75	2	70
290	0	71	-12	95	-14	100	-15	102	321	-7	99	-2	69	-1	70	-2	76
291	3	68	-5	91	-8	95	-10	99	322	-14	95	-2	59	-9	72	-18	86
292	10	44	-9	93	-17	111	-13	106	323	2	95	6	70	14	60	12	70
293	-7	62	-22	94	-28	114	-50	127	324	0	76	10	44	16	37	5	64
294	15	36	-5	86	-6	89	-5	92	325	-5	74	9	31	0	47	-12	70
295	14	40	9	67	-2	80	17	57	326	0	70	12	43	18	35	11	55
296	16	27	-4	80	-4	82	-5	86	327	-11	75	3	53	11	37	5	56
297	12	29	-10	85	-9	85	-13	94	328	1	68	14	34	12	41	0	69
298	1	44	-1	70	-8	81	-12	91	329	0	65	10	38	10	41	2	65
299	20	36	7	60	-1	72	-2	77	330	-14	73	-3	52	2	48	-6	74
300	18	32	9	58	5	64	0	71	331	3	62	13	40	12	41	5	54
301	5	42	5	61	1	67	-1	73	332	4	62	17	32	13	41	7	54
302	1	48	12	50	9	56	4	64	333	-1	68	7	44	0	59	-6	76
303	10	62	15	50	0	69	-7	81	334	-13	75	7	38	3	50	-11	82
304	17	46	18	49	1	69	5	64	335	11	55	13	50	19	40	-2	77
305	9	64	17	54	7	69	15	57	336	5	64	10	57	3	66	-2	77
306	-12	104	10	41	-7	69	1	67	337	12	70	26	43	18	50	25	42
307	-11	97	7	46	-6	67	0	68									

Table 9-23 – Values of variables m and n for ctxIdx from 338 to 398

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
338	15	6	14	11	19	-6	17	-13	369	32	-26	31	-4	40	-37	37	-17
339	6	19	11	14	18	-6	16	-9	370	37	-30	27	6	38	-30	32	1
340	7	16	9	11	14	0	17	-12	371	44	-32	34	8	46	-33	34	15
341	12	14	18	11	26	-12	27	-21	372	34	-18	30	10	42	-30	29	15
342	18	13	21	9	31	-16	37	-30	373	34	-15	24	22	40	-24	24	25
343	13	11	23	-2	33	-25	41	-40	374	40	-15	33	19	49	-29	34	22
344	13	15	32	-15	33	-22	42	-41	375	33	-7	22	32	38	-12	31	16
345	15	16	32	-15	37	-28	48	-47	376	35	-5	26	31	40	-10	35	18
346	12	23	34	-21	39	-30	39	-32	377	33	0	21	41	38	-3	31	28
347	13	23	39	-23	42	-30	46	-40	378	38	2	26	44	46	-5	33	41
348	15	20	42	-33	47	-42	52	-51	379	33	13	23	47	31	20	36	28
349	14	26	41	-31	45	-36	46	-41	380	23	35	16	65	29	30	27	47
350	14	44	46	-28	49	-34	52	-39	381	13	58	14	71	25	44	21	62
351	17	40	38	-12	41	-17	43	-19	382	29	-3	8	60	12	48	18	31
352	17	47	21	29	32	9	32	11	383	26	0	6	63	11	49	19	26
353	24	17	45	-24	69	-71	61	-55	384	22	30	17	65	26	45	36	24
354	21	21	53	-45	63	-63	56	-46	385	31	-7	21	24	22	22	24	23
355	25	22	48	-26	66	-64	62	-50	386	35	-15	23	20	23	22	27	16
356	31	27	65	-43	77	-74	81	-67	387	34	-3	26	23	27	21	24	30
357	22	29	43	-19	54	-39	45	-20	388	34	3	27	32	33	20	31	29
358	19	35	39	-10	52	-35	35	-2	389	36	-1	28	23	26	28	22	41
359	14	50	30	9	41	-10	28	15	390	34	5	28	24	30	24	22	42
360	10	57	18	26	36	0	34	1	391	32	11	23	40	27	34	16	60
361	7	63	20	27	40	-1	39	1	392	35	5	24	32	18	42	15	52
362	-2	77	0	57	30	14	30	17	393	34	12	28	29	25	39	14	60
363	-4	82	-14	82	28	26	20	38	394	39	11	23	42	18	50	3	78
364	-3	94	-5	75	23	37	18	45	395	30	29	19	57	12	70	-16	123
365	9	69	-19	97	12	55	15	54	396	34	26	22	53	21	54	21	53
366	-12	109	-35	125	11	65	0	79	397	29	39	22	61	14	71	22	56
367	36	-35	27	0	37	-33	36	-16	398	19	66	11	86	11	83	25	61
368	36	-34	28	0	39	-36	37	-14									

Table 9-24 – Values of variables m and n for ctxIdx from 402 to 459

ctxIdx	I slices		Value of cabac_init_idc						ctxIdx	I slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
402	-17	120	-4	79	-5	85	-3	78	**431**	-2	55	-12	56	-9	57	-12	59
403	-20	112	-7	71	-6	81	-8	74	**432**	0	61	-6	60	-6	63	-8	63
404	-18	114	-5	69	-10	77	-9	72	**433**	1	64	-5	62	-4	65	-9	67
405	-11	85	-9	70	-7	81	-10	72	**434**	0	68	-8	66	-4	67	-6	68
406	-15	92	-8	66	-17	80	-18	75	**435**	-9	92	-8	76	-7	82	-10	79
407	-14	89	-10	68	-18	73	-12	71	**436**	-14	106	-5	85	-3	81	-3	78
408	-26	71	-19	73	-4	74	-11	63	**437**	-13	97	-6	81	-3	76	-8	74
409	-15	81	-12	69	-10	83	-5	70	**438**	-15	90	-10	77	-7	72	-9	72
410	-14	80	-16	70	-9	71	-17	75	**439**	-12	90	-7	81	-6	78	-10	72
411	0	68	-15	67	-9	67	-14	72	**440**	-18	88	-17	80	-12	72	-18	75
412	-14	70	-20	62	-1	61	-16	67	**441**	-10	73	-18	73	-14	68	-12	71
413	-24	56	-19	70	-8	66	-8	53	**442**	-9	79	-4	74	-3	70	-11	63
414	-23	68	-16	66	-14	66	-14	59	**443**	-14	86	-10	83	-6	76	-5	70
415	-24	50	-22	65	0	59	-9	52	**444**	-10	73	-9	71	-5	66	-17	75
416	-11	74	-20	63	2	59	-11	68	**445**	-10	70	-9	67	-5	62	-14	72
417	23	-13	9	-2	17	-10	9	-2	**446**	-10	69	-1	61	0	57	-16	67
418	26	-13	26	-9	32	-13	30	-10	**447**	-5	66	-8	66	-4	61	-8	53
419	40	-15	33	-9	42	-9	31	-4	**448**	-9	64	-14	66	-9	60	-14	59
420	49	-14	39	-7	49	-5	33	-1	**449**	-5	58	0	59	1	54	-9	52
421	44	3	41	-2	53	0	33	7	**450**	2	59	2	59	2	58	-11	68
422	45	6	45	3	64	3	31	12	**451**	21	-10	21	-13	17	-10	9	-2
423	44	34	49	9	68	10	37	23	**452**	24	-11	33	-14	32	-13	30	-10
424	33	54	45	27	66	27	31	38	**453**	28	-8	39	-7	42	-9	31	-4
425	19	82	36	59	47	57	20	64	**454**	28	-1	46	-2	49	-5	33	-1
426	-3	75	-6	66	-5	71	-9	71	**455**	29	3	51	2	53	0	33	7
427	-1	23	-7	35	0	24	-7	37	**456**	29	9	60	6	64	3	31	12
428	1	34	-7	42	-1	36	-8	44	**457**	35	20	61	17	68	10	37	23
429	1	43	-8	45	-2	42	-11	49	**458**	29	36	55	34	66	27	31	38
430	0	54	-5	48	-2	52	-10	56	**459**	14	67	42	62	47	57	20	64

Table 9-25 – Values of variables m and n for ctxIdx from 460 to 483

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
460	-17	123	-7	92	0	80	11	80	472	-17	123	-7	92	0	80	11	80
461	-12	115	-5	89	-5	89	5	76	473	-12	115	-5	89	-5	89	5	76
462	-16	122	-7	96	-7	94	2	84	474	-16	122	-7	96	-7	94	2	84
463	-11	115	-13	108	-4	92	5	78	475	-11	115	-13	108	-4	92	5	78
464	-12	63	-3	46	0	39	-6	55	476	-12	63	-3	46	0	39	-6	55
465	-2	68	-1	65	0	65	4	61	477	-2	68	-1	65	0	65	4	61
466	-15	84	-1	57	-15	84	-14	83	478	-15	84	-1	57	-15	84	-14	83
467	-13	104	-9	93	-35	127	-37	127	479	-13	104	-9	93	-35	127	-37	127
468	-3	70	-3	74	-2	73	-5	79	480	-3	70	-3	74	-2	73	-5	79
469	-8	93	-9	92	-12	104	-11	104	481	-8	93	-9	92	-12	104	-11	104
470	-10	90	-8	87	-9	91	-11	91	482	-10	90	-8	87	-9	91	-11	91
471	-30	127	-23	126	-31	127	-30	127	483	-30	127	-23	126	-31	127	-30	127

Table 9-26 – Values of variables m and n for ctxIdx from 484 to 571

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
484	-7	93	-2	85	-13	103	-4	86	528	-7	93	-2	85	-13	103	-4	86
485	-11	87	-6	78	-13	91	-12	88	529	-11	87	-6	78	-13	91	-12	88
486	-3	77	-1	75	-9	89	-5	82	530	-3	77	-1	75	-9	89	-5	82
487	-5	71	-7	77	-14	92	-3	72	531	-5	71	-7	77	-14	92	-3	72
488	-4	63	2	54	-8	76	-4	67	532	-4	63	2	54	-8	76	-4	67
489	-4	68	5	50	-12	87	-8	72	533	-4	68	5	50	-12	87	-8	72
490	-12	84	-3	68	-23	110	-16	89	534	-12	84	-3	68	-23	110	-16	89
491	-7	62	1	50	-24	105	-9	69	535	-7	62	1	50	-24	105	-9	69
492	-7	65	6	42	-10	78	-1	59	536	-7	65	6	42	-10	78	-1	59
493	8	61	-4	81	-20	112	5	66	537	8	61	-4	81	-20	112	5	66
494	5	56	1	63	-17	99	4	57	538	5	56	1	63	-17	99	4	57
495	-2	66	-4	70	-78	127	-4	71	539	-2	66	-4	70	-78	127	-4	71
496	1	64	0	67	-70	127	-2	71	540	1	64	0	67	-70	127	-2	71
497	0	61	2	57	-50	127	2	58	641	0	61	2	57	-50	127	2	58
498	-2	78	-2	76	-46	127	-1	74	542	-2	78	-2	76	-46	127	-1	74

Table 9-26 – Values of variables m and n for ctxIdx from 484 to 571

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
499	1	50	11	35	-4	66	-4	44	543	1	50	11	35	-4	66	-4	44
500	7	52	4	64	-5	78	-1	69	544	7	52	4	64	-5	78	-1	69
501	10	35	1	61	-4	71	0	62	545	10	35	1	61	-4	71	0	62
502	0	44	11	35	-8	72	-7	51	546	0	44	11	35	-8	72	-7	51
503	11	38	18	25	2	59	-4	47	547	11	38	18	25	2	59	-4	47
504	1	45	12	24	-1	55	-6	42	548	1	45	12	24	-1	55	-6	42
505	0	46	13	29	-7	70	-3	41	549	0	46	13	29	-7	70	-3	41
506	5	44	13	36	-6	75	-6	53	550	5	44	13	36	-6	75	-6	53
507	31	17	-10	93	-8	89	8	76	551	31	17	-10	93	-8	89	8	76
508	1	51	-7	73	-34	119	-9	78	552	1	51	-7	73	-34	119	-9	78
509	7	50	-2	73	-3	75	-11	83	553	7	50	-2	73	-3	75	-11	83
510	28	19	13	46	32	20	9	52	554	28	19	13	46	32	20	9	52
511	16	33	9	49	30	22	0	67	555	16	33	9	49	30	22	0	67
512	14	62	7	100	44	127	5	90	556	14	62	7	100	44	127	5	90
513	-13	108	9	53	0	54	1	67	557	-13	108	9	53	0	54	1	67
514	-15	100	2	53	-5	61	-15	72	558	-15	100	2	53	-5	61	-15	72
515	-13	101	5	53	0	58	-5	75	559	-13	101	5	53	0	58	-5	75
516	-13	91	-2	61	-1	60	-8	80	560	-13	91	-2	61	-1	60	-8	80
517	-12	94	0	56	-3	61	-21	83	561	-12	94	0	56	-3	61	-21	83
518	-10	88	0	56	-8	67	-21	64	562	-10	88	0	56	-8	67	-21	64
519	-16	84	-13	63	-25	84	-13	31	563	-16	84	-13	63	-25	84	-13	31
520	-10	86	-5	60	-14	74	-25	64	564	-10	86	-5	60	-14	74	-25	64
521	-7	83	-1	62	-5	65	-29	94	565	-7	83	-1	62	-5	65	-29	94
522	-13	87	4	57	5	52	9	75	566	-13	87	4	57	5	52	9	75
523	-19	94	-6	69	2	57	17	63	567	-19	94	-6	69	2	57	17	63
524	1	70	4	57	0	61	-8	74	568	1	70	4	57	0	61	-8	74
525	0	72	14	39	-9	69	-5	35	569	0	72	14	39	-9	69	-5	35
526	-5	74	4	51	-11	70	-2	27	570	-5	74	4	51	-11	70	-2	27
527	18	59	13	68	18	55	13	91	571	18	59	13	68	18	55	13	91

Table 9-27 – Values of variables m and n for ctxIdx from 572 to 659

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
572	24	0	11	28	4	45	4	39	616	24	0	11	28	4	45	4	39
573	15	9	2	40	10	28	0	42	617	15	9	2	40	10	28	0	42
574	8	25	3	44	10	31	7	34	618	8	25	3	44	10	31	7	34
575	13	18	0	49	33	-11	11	29	619	13	18	0	49	33	-11	11	29
576	15	9	0	46	52	-43	8	31	620	15	9	0	46	52	-43	8	31
577	13	19	2	44	18	15	6	37	621	13	19	2	44	18	15	6	37
578	10	37	2	51	28	0	7	42	622	10	37	2	51	28	0	7	42
579	12	18	0	47	35	-22	3	40	623	12	18	0	47	35	-22	3	40
580	6	29	4	39	38	-25	8	33	624	6	29	4	39	38	-25	8	33
581	20	33	2	62	34	0	13	43	625	20	33	2	62	34	0	13	43
582	15	30	6	46	39	-18	13	36	626	15	30	6	46	39	-18	13	36
583	4	45	0	54	32	-12	4	47	627	4	45	0	54	32	-12	4	47
584	1	58	3	54	102	-94	3	55	628	1	58	3	54	102	-94	3	55
585	0	62	2	58	0	0	2	58	629	0	62	2	58	0	0	2	58
586	7	61	4	63	56	-15	6	60	630	7	61	4	63	56	-15	6	60
587	12	38	6	51	33	-4	8	44	631	12	38	6	51	33	-4	8	44
588	11	45	6	57	29	10	11	44	632	11	45	6	57	29	10	11	44
589	15	39	7	53	37	-5	14	42	633	15	39	7	53	37	-5	14	42
590	11	42	6	52	51	-29	7	48	634	11	42	6	52	51	-29	7	48
591	13	44	6	55	39	-9	4	56	635	13	44	6	55	39	-9	4	56
592	16	45	11	45	52	-34	4	52	636	16	45	11	45	52	-34	4	52
593	12	41	14	36	69	-58	13	37	637	12	41	14	36	69	-58	13	37
594	10	49	8	53	67	-63	9	49	638	10	49	8	53	67	-63	9	49
595	30	34	-1	82	44	-5	19	58	639	30	34	-1	82	44	-5	19	58
596	18	42	7	55	32	7	10	48	640	18	42	7	55	32	7	10	48
597	10	55	-3	78	55	-29	12	45	641	10	55	-3	78	55	-29	12	45
598	17	51	15	46	32	1	0	69	642	17	51	15	46	32	1	0	69
599	17	46	22	31	0	0	20	33	643	17	46	22	31	0	0	20	33
600	0	89	-1	84	27	36	8	63	644	0	89	-1	84	27	36	8	63
601	26	-19	25	7	33	-25	35	-18	645	26	-19	25	7	33	-25	35	-18
602	22	-17	30	-7	34	-30	33	-25	646	22	-17	30	-7	34	-30	33	-25

Table 9-27 – Values of variables m and n for ctxIdx from 572 to 659

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
603	26	-17	28	3	36	-28	28	-3	**647**	26	-17	28	3	36	-28	28	-3
604	30	-25	28	4	38	-28	24	10	**648**	30	-25	28	4	38	-28	24	10
605	28	-20	32	0	38	-27	27	0	**649**	28	-20	32	0	38	-27	27	0
606	33	-23	34	-1	34	-18	34	-14	**650**	33	-23	34	-1	34	-18	34	-14
607	37	-27	30	6	35	-16	52	-44	**651**	37	-27	30	6	35	-16	52	-44
608	33	-23	30	6	34	-14	39	-24	**652**	33	-23	30	6	34	-14	39	-24
609	40	-28	32	9	32	-8	19	17	**653**	40	-28	32	9	32	-8	19	17
610	38	-17	31	19	37	-6	31	25	**654**	38	-17	31	19	37	-6	31	25
611	33	-11	26	27	35	0	36	29	**655**	33	-11	26	27	35	0	36	29
612	40	-15	26	30	30	10	24	33	**656**	40	-15	26	30	30	10	24	33
613	41	-6	37	20	28	18	34	15	**657**	41	-6	37	20	28	18	34	15
614	38	1	28	34	26	25	30	20	**658**	38	1	28	34	26	25	30	20
615	41	17	17	70	29	41	22	73	**659**	41	17	17	70	29	41	22	73

Table 9-28– Values of variables m and n for ctxIdx from 660 to 717

ctxIdx	I slices		Value of cabac_init_idc						ctxIdx	I slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
660	-17	120	-4	79	-5	85	-3	78	689	2	59	2	59	2	58	-11	68
661	-20	112	-7	71	-6	81	-8	74	690	23	-13	9	-2	17	-10	9	-2
662	-18	114	-5	69	-10	77	-9	72	691	26	-13	26	-9	32	-13	30	-10
663	-11	85	-9	70	-7	81	-10	72	692	40	-15	33	-9	42	-9	31	-4
664	-15	92	-8	66	-17	80	-18	75	693	49	-14	39	-7	49	-5	33	-1
665	-14	89	-10	68	-18	73	-12	71	694	44	3	41	-2	53	0	33	7
666	-26	71	-19	73	-4	74	-11	63	695	45	6	45	3	64	3	31	12
667	-15	81	-12	69	-10	83	-5	70	696	44	34	49	9	68	10	37	23
668	-14	80	-16	70	-9	71	-17	75	697	33	54	45	27	66	27	31	38
669	0	68	-15	67	-9	67	-14	72	698	19	82	36	59	47	57	20	64
670	-14	70	-20	62	-1	61	-16	67	699	21	-10	21	-13	17	-10	9	-2
671	-24	56	-19	70	-8	66	-8	53	700	24	-11	33	-14	32	-13	30	-10
672	-23	68	-16	66	-14	66	-14	59	701	28	-8	39	-7	42	-9	31	-4
673	24	50	22	65	0	59	9	52	702	28	1	46	2	49	5	33	1
674	-11	74	-20	63	2	59	-11	68	703	29	3	51	2	53	0	33	7
675	-14	106	-5	85	-3	81	-3	78	704	29	9	60	6	64	3	31	12
676	-13	97	-6	81	-3	76	-8	74	705	35	20	61	17	68	10	37	23
677	-15	90	-10	77	-7	72	-9	72	706	29	36	55	34	66	27	31	38
678	-12	90	-7	81	-6	78	-10	72	707	14	67	42	62	47	57	20	64
679	-18	88	-17	80	-12	72	-18	75	708	-3	75	-6	66	-5	71	-9	71
680	-10	73	-18	73	-14	68	-12	71	709	-1	23	-7	35	0	24	-7	37
681	-9	79	-4	74	-3	70	-11	63	710	1	34	-7	42	-1	36	-8	44
682	-14	86	-10	83	-6	76	-5	70	711	1	43	-8	45	-2	42	-11	49
683	-10	73	-9	71	-5	66	-17	75	712	0	54	-5	48	-2	52	-10	56
684	-10	70	-9	67	-5	62	-14	72	713	-2	55	-12	56	-9	57	-12	59
685	-10	69	-1	61	0	57	-16	67	714	0	61	-6	60	-6	63	-8	63
686	-5	66	-8	66	-4	61	-8	53	715	1	64	-5	62	-4	65	-9	67
687	-9	64	-14	66	-9	60	-14	59	716	0	68	-8	66	-4	67	-6	68
688	-5	58	0	59	1	54	-9	52	717	-9	92	-8	76	-7	82	-10	79

Table 9-29– Values of variables m and n for ctxIdx from 718 to 775

ctxIdx	I slices		Value of cabac_init_idc						ctxIdx	I slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
718	-17	120	-4	79	-5	85	-3	78	747	2	59	2	59	2	58	-11	68
719	-20	112	-7	71	-6	81	-8	74	748	23	-13	9	-2	17	-10	9	-2
720	-18	114	-5	69	-10	77	-9	72	749	26	-13	26	-9	32	-13	30	-10
721	-11	85	-9	70	-7	81	-10	72	750	40	-15	33	-9	42	-9	31	-4
722	-15	92	-8	66	-17	80	-18	75	751	49	-14	39	-7	49	-5	33	-1
723	-14	89	-10	68	-18	73	-12	71	752	44	3	41	-2	53	0	33	7
724	-26	71	-19	73	-4	74	-11	63	753	45	6	45	3	64	3	31	12
725	-15	81	-12	69	-10	83	-5	70	754	44	34	49	9	68	10	37	23
726	-14	80	-16	70	-9	71	-17	75	755	33	54	45	27	66	27	31	38
727	0	68	-15	67	-9	67	-14	72	756	19	82	36	59	47	57	20	64
728	-14	70	-20	62	-1	61	-16	67	757	21	-10	21	-13	17	-10	9	-2
729	-24	56	-19	70	-8	66	-8	53	758	24	-11	33	-14	32	-13	30	-10
730	-23	68	-16	66	-14	66	-14	59	759	28	-8	39	-7	42	-9	31	-4
731	-24	50	-22	65	0	59	-9	52	760	28	-1	46	-2	49	-5	33	-1
732	-11	74	-20	63	2	59	-11	68	761	29	3	51	2	53	0	33	7
733	-14	106	-5	85	-3	81	-3	78	762	29	9	60	6	64	3	31	12
734	-13	97	-6	81	-3	76	-8	74	763	35	20	61	17	68	10	37	23
735	-15	90	-10	77	-7	72	-9	72	764	29	36	55	34	66	27	31	38
736	-12	90	-7	81	-6	78	-10	72	765	14	67	42	62	47	57	20	64
737	-18	88	-17	80	-12	72	-18	75	766	-3	75	-6	66	-5	71	-9	71
738	-10	73	-18	73	-14	68	-12	71	767	-1	23	-7	35	0	24	-7	37
739	-9	79	-4	74	-3	70	-11	63	768	1	34	-7	42	-1	36	-8	44
740	-14	86	-10	83	-6	76	-5	70	769	1	43	-8	45	-2	42	-11	49
741	-10	73	-9	71	-5	66	-17	75	770	0	54	-5	48	-2	52	-10	56
742	-10	70	-9	67	-5	62	-14	72	771	-2	55	-12	56	-9	57	-12	59
743	-10	69	-1	61	0	57	-16	67	772	0	61	-6	60	-6	63	-8	63
744	-5	66	-8	66	-4	61	-8	53	773	1	64	-5	62	-4	65	-9	67
745	-9	64	-14	66	-9	60	-14	59	774	0	68	-8	66	-4	67	-6	68
746	-5	58	0	59	1	54	-9	52	775	-9	92	-8	76	-7	82	-10	79

Table 9-30 – Values of variables m and n for ctxIdx from 776 to 863

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
776	-6	93	-13	106	-21	126	-22	127	820	-6	93	-13	106	-21	126	-22	127
777	-6	84	-16	106	-23	124	-25	127	821	-6	84	-16	106	-23	124	-25	127
778	-8	79	-10	87	-20	110	-25	120	822	-8	79	-10	87	-20	110	-25	120
779	0	66	-21	114	-26	126	-27	127	823	0	66	-21	114	-26	126	-27	127
780	-1	71	-18	110	-25	124	-19	114	824	-1	71	-18	110	-25	124	-19	114
781	0	62	-14	98	-17	105	-23	117	825	0	62	-14	98	-17	105	-23	117
782	-2	60	-22	110	-27	121	-25	118	826	-2	60	-22	110	-27	121	-25	118
783	-2	59	-21	106	-27	117	-26	117	827	-2	59	-21	106	-27	117	-26	117
784	-5	75	-18	103	-17	102	-24	113	828	-5	75	-18	103	-17	102	-24	113
785	-3	62	-21	107	-26	117	-28	118	829	-3	62	-21	107	-26	117	-28	118
786	-4	58	-23	108	-27	116	-31	120	830	-4	58	-23	108	-27	116	-31	120
787	-9	66	-26	112	-33	122	-37	124	831	-9	66	-26	112	-33	122	-37	124
788	-1	79	-10	96	-10	95	-10	94	832	-1	79	-10	96	-10	95	-10	94
789	0	71	12	95	14	100	15	102	833	0	71	12	95	14	100	15	102
790	3	68	-5	91	-8	95	-10	99	834	3	68	-5	91	-8	95	-10	99
791	10	44	-9	93	-17	111	-13	106	835	10	44	-9	93	-17	111	-13	106
792	-7	62	-22	94	-28	114	-50	127	836	-7	62	-22	94	-28	114	-50	127
793	15	36	-5	86	-6	89	-5	92	837	15	36	-5	86	-6	89	-5	92
794	14	40	9	67	-2	80	17	57	838	14	40	9	67	-2	80	17	57
795	16	27	-4	80	-4	82	-5	86	839	16	27	-4	80	-4	82	-5	86
796	12	29	-10	85	-9	85	-13	94	840	12	29	-10	85	-9	85	-13	94
797	1	44	-1	70	-8	81	-12	91	841	1	44	-1	70	-8	81	-12	91
798	20	36	7	60	-1	72	-2	77	842	20	36	7	60	-1	72	-2	77
799	18	32	9	58	5	64	0	71	843	18	32	9	58	5	64	0	71
800	5	42	5	61	1	67	-1	73	844	5	42	5	61	1	67	-1	73
801	1	48	12	50	9	56	4	64	845	1	48	12	50	9	56	4	64
802	10	62	15	50	0	69	-7	81	846	10	62	15	50	0	69	-7	81
803	17	46	18	49	1	69	5	64	847	17	46	18	49	1	69	5	64
804	9	64	17	54	7	69	15	57	848	9	64	17	54	7	69	15	57
805	-12	104	10	41	-7	69	1	67	849	-12	104	10	41	-7	69	1	67
806	-11	97	7	46	-6	67	0	68	850	-11	97	7	46	-6	67	0	68

Table 9-30 – Values of variables m and n for ctxIdx from 776 to 863

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
807	-16	96	-1	51	-16	77	-10	67	**851**	-16	96	-1	51	-16	77	-10	67
808	-7	88	7	49	-2	64	1	68	**852**	-7	88	7	49	-2	64	1	68
809	-8	85	8	52	2	61	0	77	**853**	-8	85	8	52	2	61	0	77
810	-7	85	9	41	-6	67	2	64	**854**	-7	85	9	41	-6	67	2	64
811	-9	85	6	47	-3	64	0	68	**855**	-9	85	6	47	-3	64	0	68
812	-13	88	2	55	2	57	-5	78	**856**	-13	88	2	55	2	57	-5	78
813	4	66	13	41	-3	65	7	55	**857**	4	66	13	41	-3	65	7	55
814	-3	77	10	44	-3	66	5	59	**858**	-3	77	10	44	-3	66	5	59
815	-3	76	6	50	0	62	2	65	**859**	-3	76	6	50	0	62	2	65
816	-6	76	5	53	9	51	14	54	**860**	-6	76	5	53	9	51	14	54
817	10	58	13	49	-1	66	15	44	**861**	10	58	13	49	-1	66	15	44
818	-1	76	4	63	-2	71	5	60	**862**	-1	76	4	63	-2	71	5	60
819	-1	83	6	64	-2	75	2	70	**863**	-1	83	6	64	-2	75	2	70

Table 9-31 – Values of variables m and n for ctxIdx from 864 to 951

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
864	15	6	14	11	19	-6	17	-13	908	15	6	14	11	19	-6	17	-13
865	6	19	11	14	18	-6	16	-9	909	6	19	11	14	18	-6	16	-9
866	7	16	9	11	14	0	17	-12	910	7	16	9	11	14	0	17	-12
867	12	14	18	11	26	-12	27	-21	911	12	14	18	11	26	-12	27	-21
868	18	13	21	9	31	-16	37	-30	912	18	13	21	9	31	-16	37	-30
869	13	11	23	-2	33	-25	41	-40	913	13	11	23	-2	33	-25	41	-40
870	13	15	32	-15	33	-22	42	-41	914	13	15	32	-15	33	-22	42	-41
871	15	16	32	-15	37	-28	48	-47	915	15	16	32	-15	37	-28	48	-47
872	12	23	34	-21	39	-30	39	-32	916	12	23	34	-21	39	-30	39	-32
873	13	23	39	-23	42	-30	46	-40	917	13	23	39	-23	42	-30	46	-40
874	15	20	42	-33	47	-42	52	-51	918	15	20	42	-33	47	-42	52	-51
875	14	26	41	-31	45	-36	46	-41	919	14	26	41	-31	45	-36	46	-41
876	14	44	46	-28	49	-34	52	-39	920	14	44	46	-28	49	-34	52	-39
877	17	40	38	12	41	17	43	19	921	17	40	38	12	41	17	43	19
878	17	47	21	29	32	9	32	11	922	17	47	21	29	32	9	32	11
879	24	17	45	-24	69	-71	61	-55	923	24	17	45	-24	69	-71	61	-55
880	21	21	53	-45	63	-63	56	-46	924	21	21	53	-45	63	-63	56	-46
881	25	22	48	-26	66	-64	62	-50	925	25	22	48	-26	66	-64	62	-50
882	31	27	65	-43	77	-74	81	-67	926	31	27	65	-43	77	-74	81	-67
883	22	29	43	-19	54	-39	45	-20	927	22	29	43	-19	54	-39	45	-20
884	19	35	39	-10	52	-35	35	-2	928	19	35	39	-10	52	-35	35	-2
885	14	50	30	9	41	-10	28	15	929	14	50	30	9	41	-10	28	15
886	10	57	18	26	36	0	34	1	930	10	57	18	26	36	0	34	1
887	7	63	20	27	40	-1	39	1	931	7	63	20	27	40	-1	39	1
888	-2	77	0	57	30	14	30	17	932	-2	77	0	57	30	14	30	17
889	-4	82	-14	82	28	26	20	38	933	-4	82	-14	82	28	26	20	38
890	-3	94	-5	75	23	37	18	45	934	-3	94	-5	75	23	37	18	45
891	9	69	-19	97	12	55	15	54	935	9	69	-19	97	12	55	15	54
892	-12	109	-35	125	11	65	0	79	936	-12	109	-35	125	11	65	0	79
893	36	-35	27	0	37	-33	36	-16	937	36	-35	27	0	37	-33	36	-16
894	36	-34	28	0	39	-36	37	-14	938	36	-34	28	0	39	-36	37	-14

Table 9-31 – Values of variables m and n for ctxIdx from 864 to 951

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
895	32	-26	31	-4	40	-37	37	-17	**939**	32	-26	31	-4	40	-37	37	-17
896	37	-30	27	6	38	-30	32	1	**940**	37	-30	27	6	38	-30	32	1
897	44	-32	34	8	46	-33	34	15	**941**	44	-32	34	8	46	-33	34	15
898	34	-18	30	10	42	-30	29	15	**942**	34	-18	30	10	42	-30	29	15
899	34	-15	24	22	40	-24	24	25	**943**	34	-15	24	22	40	-24	24	25
900	40	-15	33	19	49	-29	34	22	**944**	40	-15	33	19	49	-29	34	22
901	33	-7	22	32	38	-12	31	16	**945**	33	-7	22	32	38	-12	31	16
902	35	-5	26	31	40	-10	35	18	**946**	35	-5	26	31	40	-10	35	18
903	33	0	21	41	38	-3	31	28	**947**	33	0	21	41	38	-3	31	28
904	38	2	26	44	46	-5	33	41	**948**	38	2	26	44	46	-5	33	41
905	33	13	23	47	31	20	36	28	**949**	33	13	23	47	31	20	36	28
906	23	35	16	65	29	30	27	47	**950**	23	35	16	65	29	30	27	47
907	13	58	14	71	25	44	21	62	**951**	13	58	14	71	25	44	21	62

Table 9-32 – Values of variables m and n for ctxIdx from 952 to 1011

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
952	-3	71	-6	76	-23	112	-24	115	982	-3	71	-6	76	-23	112	-24	115
953	-6	42	-2	44	-15	71	-22	82	983	-6	42	-2	44	-15	71	-22	82
954	-5	50	0	45	-7	61	-9	62	984	-5	50	0	45	-7	61	-9	62
955	-3	54	0	52	0	53	0	53	985	-3	54	0	52	0	53	0	53
956	-2	62	-3	64	-5	66	0	59	986	-2	62	-3	64	-5	66	0	59
957	0	58	-2	59	-11	77	-14	85	987	0	58	-2	59	-11	77	-14	85
958	1	63	-4	70	-9	80	-13	89	988	1	63	-4	70	-9	80	-13	89
959	-2	72	-4	75	-9	84	-13	94	989	-2	72	-4	75	-9	84	-13	94
960	-1	74	-8	82	-10	87	-11	92	990	-1	74	-8	82	-10	87	-11	92
961	-9	91	-17	102	-34	127	-29	127	991	-9	91	-17	102	-34	127	-29	127
962	-5	67	-9	77	-21	101	-21	100	992	-5	67	-9	77	-21	101	-21	100
963	-5	27	3	24	-3	39	-14	57	993	-5	27	3	24	-3	39	-14	57
964	-3	39	0	42	-5	53	-12	67	994	-3	39	0	42	-5	53	-12	67
965	-2	44	0	48	-7	61	-11	71	995	-2	44	0	48	-7	61	-11	71
966	0	46	0	55	-11	75	-10	77	996	0	46	0	55	-11	75	-10	77
967	-16	64	-6	59	-15	77	-21	85	997	-16	64	-6	59	-15	77	-21	85
968	-8	68	-7	71	-17	91	-16	88	998	-8	68	-7	71	-17	91	-16	88
969	-10	78	-12	83	-25	107	-23	104	999	-10	78	-12	83	-25	107	-23	104
970	-6	77	-11	87	-25	111	-15	98	1000	-6	77	-11	87	-25	111	-15	98
971	-10	86	-30	119	-28	122	-37	127	1001	-10	86	-30	119	-28	122	-37	127
972	-12	92	1	58	-11	76	-10	82	1002	-12	92	1	58	-11	76	-10	82
973	-15	55	-3	29	-10	44	-8	48	1003	-15	55	-3	29	-10	44	-8	48
974	-10	60	-1	36	-10	52	-8	61	1004	-10	60	-1	36	-10	52	-8	61
975	-6	62	1	38	-10	57	-8	66	1005	-6	62	1	38	-10	57	-8	66
976	-4	65	2	43	-9	58	-7	70	1006	-4	65	2	43	-9	58	-7	70
977	-12	73	-6	55	-16	72	-14	75	1007	-12	73	-6	55	-16	72	-14	75
978	-8	76	0	58	-7	69	-10	79	1008	-8	76	0	58	-7	69	-10	79
979	-7	80	0	64	-4	69	-9	83	1009	-7	80	0	64	-4	69	-9	83
980	-9	88	-3	74	-5	74	-12	92	1010	-9	88	-3	74	-5	74	-12	92
981	-17	110	-10	90	-9	86	-18	108	1011	-17	110	-10	90	-9	86	-18	108

Table 9-33 – Values of variables m and n for ctxIdx from 1012 to 1023

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	N	m	n
1012	-3	70	-3	74	-2	73	-5	79	1018	-10	90	-8	87	-9	91	-11	91
1013	-8	93	-9	92	-12	104	-11	104	1019	-30	127	-23	126	-31	127	-30	127
1014	-10	90	-8	87	-9	91	-11	91	1020	-3	70	-3	74	-2	73	-5	79
1015	-30	127	-23	126	-31	127	-30	127	1021	-8	93	-9	92	-12	104	-11	104
1016	-3	70	-3	74	-2	73	-5	79	1022	-10	90	-8	87	-9	91	-11	91
1017	-8	93	-9	92	-12	104	-11	104	1023	-30	127	-23	126	-31	127	-30	127

9.3.1.2 Initialisation process for the arithmetic decoding engine

This process is invoked before decoding the first macroblock of a slice or after the decoding of any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data for a macroblock of type I_PCM.

Outputs of this process are the initialised decoding engine registers codIRange and codIOffset both in 16 bit register precision.

The status of the arithmetic decoding engine is represented by the variables codIRange and codIOffset. In the initialisation procedure of the arithmetic decoding process, codIRange is set equal to 0x01FE and codIOffset is set equal to the value returned from read_bits(9) interpreted as a 9 bit binary representation of an unsigned integer with most significant bit written first.

The bitstream shall not contain data that results in a value of codIOffset being equal to 0x01FE or 0x01FF.

NOTE – The description of the arithmetic decoding engine in this Recommendation | International Standard utilizes 16 bit register precision. However, the minimum register precision for the variables codIRange and codIOffset is 9 bits.

9.3.2 Binarization process

Input to this process is a request for a syntax element.

Output of this process is the binarization of the syntax element, maxBinIdxCtx, ctxIdxOffset, and bypassFlag.

Table 9-34 specifies the type of binarization process, maxBinIdxCtx, and ctxIdxOffset associated with each syntax element.

The specification of the unary (U) binarization process, the truncated unary (TU) binarization process, the concatenated unary / k-th order Exp-Golomb (UEGk) binarization process, and the fixed-length (FL) binarization process are given in subclauses 9.3.2.1 to 9.3.2.4, respectively. Other binarizations are specified in subclauses 9.3.2.5 to 9.3.2.7.

Except for I slices, the binarizations for the syntax element mb_type as specified in subclause 9.3.2.5 consist of bin strings given by a concatenation of prefix and suffix bit strings. The UEGk binarization as specified in 9.3.2.3, which is used for the binarization of the syntax elements mvd_lX (X = 0, 1) and coeff_abs_level_minus1, and the binarization of the coded_block_pattern also consist of a concatenation of prefix and suffix bit strings. For these binarization processes, the prefix and the suffix bit string are separately indexed using the binIdx variable as specified further in subclause 9.3.3. The two sets of prefix bit strings and suffix bit strings are referred to as the binarization prefix part and the binarization suffix part, respectively.

Associated with each binarization or binarization part of a syntax element is a specific value of the context index offset (ctxIdxOffset) variable and a specific value of the maxBinIdxCtx variable as given in Table 9-34. When two values for each of these variables are specified for one syntax element in Table 9-34, the value in the upper row is related to the prefix part while the value in the lower row is related to the suffix part of the binarization of the corresponding syntax element.

The use of the DecodeBypass process and the variable bypassFlag is derived as follows.

– If no value is assigned to ctxIdxOffset for the corresponding binarization or binarization part in Table 9-34 labelled as "na", all bins of the bit strings of the corresponding binarization or of the binarization prefix/suffix part are decoded by invoking the DecodeBypass process as specified in subclause 9.3.3.2.3. In such a case, bypassFlag is

set equal to 1, where bypassFlag is used to indicate that for parsing the value of the bin from the bitstream the DecodeBypass process is applied.

– Otherwise, for each possible value of binIdx up to the specified value of maxBinIdxCtx given in Table 9-34, a specific value of the variable ctxIdx is further specified in subclause 9.3.3. bypassFlag is set equal to 0.

The possible values of the context index ctxIdx are in the range 0 to 1023, inclusive. The value assigned to ctxIdxOffset specifies the lower value of the range of ctxIdx assigned to the corresponding binarization or binarization part of a syntax element.

ctxIdx = ctxIdxOffset = 276 is assigned to the syntax element end_of_slice_flag and the bin of mb_type, which specifies the I_PCM macroblock type as further specified in subclause 9.3.3.1. For parsing the value of the corresponding bin from the bitstream, the arithmetic decoding process for decisions before termination (DecodeTerminate) as specified in subclause 9.3.3.2.4 is applied.

NOTE – The bins of mb_type in I slices and the bins of the suffix for mb_type in SI slices that correspond to the same value of binIdx share the same ctxIdx. The last bin of the prefix of mb_type and the first bin of the suffix of mb_type in P, SP, and B slices may share the same ctxIdx.

Table 9-34 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset

Syntax element	Type of binarization	maxBinIdxCtx	ctxIdxOffset
mb_type (SI slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 0 suffix: 6	prefix: 0 suffix: 3
mb_type (I slices only)	as specified in subclause 9.3.2.5	6	3
mb_skip_flag (P, SP slices only)	FL, cMax=1	0	11
mb_type (P, SP slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 2 suffix: 5	prefix: 14 suffix: 17
sub_mb_type (P, SP slices only)	as specified in subclause 9.3.2.5	2	21
mb_skip_flag (B slices only)	FL, cMax=1	0	24
mb_type (B slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 3 suffix: 5	prefix: 27 suffix: 32
sub_mb_type (B slices only)	as specified in subclause 9.3.2.5	3	36
mvd_l0[ ][ ][0], mvd_l1[ ][ ][0]	prefix and suffix as given by UEG3 with signedValFlag=1, uCoff=9	prefix: 4 suffix: na	prefix: 40 suffix: na (uses DecodeBypass)
mvd_l0[ ][ ][1], mvd_l1[ ][ ][1]		prefix: 4 suffix: na	prefix: 47 suffix: na (uses DecodeBypass)
ref_idx_l0, ref_idx_l1	U	2	54
mb_qp_delta	as specified in subclause 9.3.2.7	2	60
intra_chroma_pred_mode	TU, cMax=3	1	64
prev_intra4x4_pred_mode_flag, prev_intra8x8_pred_mode_flag	FL, cMax=1	0	68
rem_intra4x4_pred_mode, rem_intra8x8_pred_mode	FL, cMax=7	0	69
mb_field_decoding_flag	FL, cMax=1	0	70
coded_block_pattern	prefix and suffix as specified in subclause 9.3.2.6	prefix: 3 suffix: 1	prefix: 73 suffix: 77

Table 9-34 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset

Syntax element	Type of binarization	maxBinIdxCtx	ctxIdxOffset
coded_block_flag	FL, cMax=1	0	85
significant_coeff_flag (frame coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	105
last_significant_coeff_flag (frame coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	166
coeff_abs_level_minus1 (blocks with ctxBlockCat < 5)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 227 suffix: na, (uses DecodeBypass)
coeff_sign_flag	FL, cMax=1	0	na, (uses DecodeBypass)
end_of_slice_flag	FL, cMax=1	0	276
significant_coeff_flag (field coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	277
last_significant_coeff_flag (field coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	338
transform_size_8x8_flag	FL, cMax=1	0	399
significant_coeff_flag (frame coded blocks with ctxBlockCat == 5)	FL, cMax=1	0	402
last_significant_coeff_flag (frame coded blocks with ctxBlockCat == 5)	FL, cMax=1	0	417
coeff_abs_level_minus1 (blocks with ctxBlockCat == 5)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 426 suffix: na, (uses DecodeBypass)
significant_coeff_flag (field coded blocks with ctxBlockCat == 5)	FL, cMax=1	0	436
last_significant_coeff_flag (field coded blocks with ctxBlockCat == 5)	FL, cMax=1	0	451
coded_block_flag (5 < ctxBlockCat < 9)	FL, cMax=1	0	460
coded_block_flag (9 < ctxBlockCat < 13)	FL, cMax=1	0	472
coded_block_flag (ctxBlockCat == 5, 9, or 13)	FL, cMax=1	0	1012
significant_coeff_flag (frame coded blocks with 5 < ctxBlockCat < 9)	FL, cMax=1	0	484
significant_coeff_flag (frame coded blocks with 9 < ctxBlockCat < 13)	FL, cMax=1	0	528
last_significant_coeff_flag (frame coded blocks with 5 < ctxBlockCat < 9)	FL, cMax=1	0	572
last_significant_coeff_flag (frame coded blocks with 9 < ctxBlockCat < 13)	FL, cMax=1	0	616
coeff_abs_level_minus1 (blocks with 5 < ctxBlockCat < 9)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 952 suffix: na, (uses DecodeBypass)
coeff_abs_level_minus1 (blocks with 9 < ctxBlockCat < 13)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 982 suffix: na, (uses DecodeBypass)

Table 9-34 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset

Syntax element	Type of binarization	maxBinIdxCtx	ctxIdxOffset
significant_coeff_flag (field coded blocks with 5 < ctxBlockCat < 9)	FL, cMax=1	0	776
significant_coeff_flag (field coded blocks with 9 < ctxBlockCat < 13)	FL, cMax=1	0	820
last_significant_coeff_flag (field coded blocks with 5 < ctxBlockCat < 9)	FL, cMax=1	0	864
last_significant_coeff_flag (field coded blocks with 9 < ctxBlockCat < 13)	FL, cMax=1	0	908
significant_coeff_flag (frame coded blocks with ctxBlockCat == 9)	FL, cMax=1	0	660
significant_coeff_flag (frame coded blocks with ctxBlockCat == 13)	FL, cMax=1	0	718
last_significant_coeff_flag (frame coded blocks with ctxBlockCat == 9)	FL, cMax=1	0	690
last_significant_coeff_flag (frame coded blocks with ctxBlockCat == 13)	FL, cMax=1	0	748
coeff_abs_level_minus1 (blocks with ctxBlockCat == 9)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 708 suffix: na, (uses DecodeBypass)
coeff_abs_level_minus1 (blocks with ctxBlockCat == 13)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 766 suffix: na, (uses DecodeBypass)
significant_coeff_flag (field coded blocks with ctxBlockCat == 9)	FL, cMax=1	0	675
significant_coeff_flag (field coded blocks with ctxBlockCat == 13)	FL, cMax=1	0	733
last_significant_coeff_flag (field coded blocks with ctxBlockCat == 9)	FL, cMax=1	0	699
last_significant_coeff_flag (field coded blocks with ctxBlockCat == 13)	FL, cMax=1	0	757

9.3.2.1 Unary (U) binarization process

Input to this process is a request for a U binarization for a syntax element.

Output of this process is the U binarization of the syntax element.

The bin string of a syntax element having (unsigned integer) value synElVal is a bit string of length synElVal + 1 indexed by BinIdx. The bins for binIdx less than synElVal are equal to 1. The bin with binIdx equal to synElVal is equal to 0.

Table 9-35 illustrates the bin strings of the unary binarization for a syntax element.

Table 9-35 – Bin string of the unary binarization (informative)

Value of syntax element	Bin string					
0 (I_NxN)	0					
1	1	0				
2	1	1	0			
3	1	1	1	0		
4	1	1	1	1	0	
5	1	1	1	1	1	0
...						
binIdx	0	1	2	3	4	5

9.3.2.2 Truncated unary (TU) binarization process

Input to this process is a request for a TU binarization for a syntax element and cMax.

Output of this process is the TU binarization of the syntax element.

For syntax element (unsigned integer) values less than cMax, the U binarization process as specified in subclause 9.3.2.1 is invoked. For the syntax element value equal to cMax the bin string is a bit string of length cMax with all bins being equal to 1.

NOTE – TU binarization is always invoked with a cMax value equal to the largest possible value of the syntax element being decoded.

9.3.2.3 Concatenated unary/ k-th order Exp-Golomb (UEGk) binarization process

Input to this process is a request for a UEGk binarization for a syntax element, signedValFlag and uCoff.

Output of this process is the UEGk binarization of the syntax element.

A UEGk bin string is a concatenation of a prefix bit string and a suffix bit string. The prefix of the binarization is specified by invoking the TU binarization process for the prefix part Min(uCoff, Abs(synElVal)) of a syntax element value synElVal as specified in subclause 9.3.2.2 with cMax = uCoff, where uCoff > 0.

The UEGk bin string is derived as follows.

- If one of the following is true, the bin string of a syntax element having value synElVal consists only of a prefix bit string,
  - signedValFlag is equal to 0 and the prefix bit string is not equal to the bit string of length uCoff with all bits equal to 1.
  - signedValFlag is equal to 1 and the prefix bit string is equal to the bit string that consists of a single bit with value equal to 0.
- Otherwise, the bin string of the UEGk suffix part of a syntax element value synElVal is specified by a process equivalent to the following pseudo-code:

```
if( Abs( synElVal ) >= uCoff ) {
    sufS = Abs( synElVal ) − uCoff
    stopLoop = 0
    do {
        if( sufS >= ( 1 << k ) ) {
            put( 1 )
            sufS = sufS − ( 1<<k )
            k++
        } else {
            put( 0 )
            while( k− − )
                put( ( sufS >> k ) & 0x01 )
            stopLoop = 1
```

```
        }
    } while( !stopLoop )
}
if( signedValFlag && synElVal != 0)
    if( synElVal > 0 )
        put( 0 )
    else
        put( 1 )
```

NOTE – The specification for the k-th order Exp-Golomb (EGk) code uses 1's and 0's in reverse meaning for the unary part of the Exp-Golomb code of 0-th order as specified in subclause 9.1.

9.3.2.4 Fixed-length (FL) binarization process

Input to this process is a request for a FL binarization for a syntax element and cMax.

Output of this process is the FL binarization of the syntax element.

FL binarization is constructed by using a fixedLength-bit unsigned integer bin string of the syntax element value, where fixedLength = Ceil(Log2(cMax + 1)). The indexing of bins for the FL binarization is such that the binIdx = 0 relates to the least significant bit with increasing values of binIdx towards the most significant bit.

9.3.2.5 Binarization process for macroblock type and sub-macroblock type

Input to this process is a request for a binarization for syntax elements mb_type or sub_mb_type.

Output of this process is the binarization of the syntax element.

The binarization scheme for decoding of macroblock type in I slices is specified in Table 9-36.

For macroblock types in SI slices, the binarization consists of bin strings specified as a concatenation of a prefix and a suffix bit string as follows.

The prefix bit string consists of a single bit, which is specified by b_0 = ((mb_type == SI) ? 0 : 1). For the syntax element value for which b_0 is equal to 0, the bin string only consists of the prefix bit string. For the syntax element value for which b_0 is equal to 1, the binarization is given by concatenating the prefix b_0 and the suffix bit string as specified in Table 9-36 for macroblock type in I slices indexed by subtracting 1 from the value of mb_type in SI slices.

Table 9-36 – Binarization for macroblock types in I slices

Value (name) of mb_type	Bin string						
0 (I_4x4)	0						
1 (I_16x16_0_0_0)	1	0	0	0	0	0	
2 (I_16x16_1_0_0)	1	0	0	0	0	1	
3 (I_16x16_2_0_0)	1	0	0	0	1	0	
4 (I_16x16_3_0_0)	1	0	0	0	1	1	
5 (I_16x16_0_1_0)	1	0	0	1	0	0	0
6 (I_16x16_1_1_0)	1	0	0	1	0	0	1
7 (I_16x16_2_1_0)	1	0	0	1	0	1	0
8 (I_16x16_3_1_0)	1	0	0	1	0	1	1
9 (I_16x16_0_2_0)	1	0	0	1	1	0	0
10 (I_16x16_1_2_0)	1	0	0	1	1	0	1
11 (I_16x16_2_2_0)	1	0	0	1	1	1	0
12 (I_16x16_3_2_0)	1	0	0	1	1	1	1
13 (I_16x16_0_0_1)	1	0	1	0	0	0	
14 (I_16x16_1_0_1)	1	0	1	0	0	1	
15 (I_16x16_2_0_1)	1	0	1	0	1	0	
16 (I_16x16_3_0_1)	1	0	1	0	1	1	
17 (I_16x16_0_1_1)	1	0	1	1	0	0	0
18 (I_16x16_1_1_1)	1	0	1	1	0	0	1
19 (I_16x16_2_1_1)	1	0	1	1	0	1	0
20 (I_16x16_3_1_1)	1	0	1	1	0	1	1
21 (I_16x16_0_2_1)	1	0	1	1	1	0	0
22 (I_16x16_1_2_1)	1	0	1	1	1	0	1
23 (I_16x16_2_2_1)	1	0	1	1	1	1	0
24 (I_16x16_3_2_1)	1	0	1	1	1	1	1
25 (I_PCM)	1	1					
binIdx	0	1	2	3	4	5	6

The binarization schemes for P macroblock types in P and SP slices and for B macroblocks in B slices are specified in Table 9-37.

The bin string for I macroblock types in P and SP slices corresponding to mb_type values 5 to 30 consists of a concatenation of a prefix, which consists of a single bit with value equal to 1 as specified in Table 9-37 and a suffix as specified in Table 9-36, indexed by subtracting 5 from the value of mb_type.

mb_type equal to 4 (P_8x8ref0) is not allowed.

For I macroblock types in B slices (mb_type values 23 to 48) the binarization consists of bin strings specified as a concatenation of a prefix bit string as specified in Table 9-37 and suffix bit strings as specified in Table 9-36, indexed by subtracting 23 from the value of mb_type.

Table 9-37 – Binarization for macroblock types in P, SP, and B slices

Slice type	Value (name) of mb_type	Bin string						
P, SP slice	0 (P_L0_16x16)	0	0	0				
	1 (P_L0_L0_16x8)	0	1	1				
	2 (P_L0_L0_8x16)	0	1	0				
	3 (P_8x8)	0	0	1				
	4 (P_8x8ref0)	na						
	5 to 30 (Intra, prefix only)	1						
B slice	0 (B_Direct_16x16)	0						
	1 (B_L0_16x16)	1	0	0				
	2 (B_L1_16x16)	1	0	1				
	3 (B_Bi_16x16)	1	1	0	0	0	0	
	4 (B_L0_L0_16x8)	1	1	0	0	0	1	
	5 (B_L0_L0_8x16)	1	1	0	0	1	0	
	6 (B_L1_L1_16x8)	1	1	0	0	1	1	
	7 (B_L1_L1_8x16)	1	1	0	1	0	0	
	8 (B_L0_L1_16x8)	1	1	0	1	0	1	
	9 (B_L0_L1_8x16)	1	1	0	1	1	0	
	10 (B_L1_L0_16x8)	1	1	0	1	1	1	
	11 (B_L1_L0_8x16)	1	1	1	1	1	0	
	12 (B_L0_Bi_16x8)	1	1	1	0	0	0	0
	13 (B_L0_Bi_8x16)	1	1	1	0	0	0	1
	14 (B_L1_Bi_16x8)	1	1	1	0	0	1	0
	15 (B_L1_Bi_8x16)	1	1	1	0	0	1	1
	16 (B_Bi_L0_16x8)	1	1	1	0	1	0	0
	17 (B_Bi_L0_8x16)	1	1	1	0	1	0	1
	18 (B_Bi_L1_16x8)	1	1	1	0	1	1	0
	19 (B_Bi_L1_8x16)	1	1	1	0	1	1	1
	20 (B_Bi_Bi_16x8)	1	1	1	1	0	0	0
	21 (B_Bi_Bi_8x16)	1	1	1	1	0	0	1
	22 (B_8x8)	1	1	1	1	1	1	
	23 to 48 (Intra, prefix only)	1	1	1	1	0	1	
binIdx		0	1	2	3	4	5	6

For P, SP, and B slices the specification of the binarization for sub_mb_type is given in Table 9-38.

Table 9-38 – Binarization for sub-macroblock types in P, SP, and B slices

Slice type	Value (name) of sub_mb_type	Bin string					
P, SP slice	0 (P_L0_8x8)	1					
	1 (P_L0_8x4)	0	0				
	2 (P_L0_4x8)	0	1	1			
	3 (P_L0_4x4)	0	1	0			
B slice	0 (B_Direct_8x8)	0					
	1 (B_L0_8x8)	1	0	0			
	2 (B_L1_8x8)	1	0	1			
	3 (B_Bi_8x8)	1	1	0	0	0	
	4 (B_L0_8x4)	1	1	0	0	1	
	5 (B_L0_4x8)	1	1	0	1	0	
	6 (B_L1_8x4)	1	1	0	1	1	
	7 (B_L1_4x8)	1	1	1	0	0	0
	8 (B_Bi_8x4)	1	1	1	0	0	1
	9 (B_Bi_4x8)	1	1	1	0	1	0
	10 (B_L0_4x4)	1	1	1	0	1	1
	11 (B_L1_4x4)	1	1	1	1	0	
	12 (B_Bi_4x4)	1	1	1	1	1	
binIdx		0	1	2	3	4	5

9.3.2.6 Binarization process for coded block pattern

Input to this process is a request for a binarization for the syntax element coded_block_pattern.

Output of this process is the binarization of the syntax element.

The binarization of coded_block_pattern consists of a prefix part and (when present) a suffix part. The prefix part of the binarization is given by the FL binarization of CodedBlockPatternLuma with cMax = 15. When ChromaArrayType is not equal to 0 or 3, the suffix part is present and consists of the TU binarization of CodedBlockPatternChroma with cMax = 2. The relationship between the value of the syntax element coded_block_pattern and the values of CodedBlockPatternLuma and CodedBlockPatternChroma is given as specified in subclause 7.4.5.

9.3.2.7 Binarization process for mb_qp_delta

Input to this process is a request for a binarization for the syntax element mb_qp_delta.

Output of this process is the binarization of the syntax element.

The bin string of mb_qp_delta is derived by the U binarization of the mapped value of the syntax element mb_qp_delta, where the assignment rule between the signed value of mb_qp_delta and its mapped value is given as specified in Table 9-3.

9.3.3 Decoding process flow

Input to this process is a binarization of the requested syntax element, maxBinIdxCtx, bypassFlag and ctxIdxOffset as specified in subclause 9.3.2.

Output of this process is the value of the syntax element.

This process specifies how each bit of a bit string is parsed for each syntax element.

After parsing each bit, the resulting bit string is compared to all bin strings of the binarization of the syntax element and the following applies.

– If the bit string is equal to one of the bin strings, the corresponding value of the syntax element is the output.

– Otherwise (the bit string is not equal to one of the bin strings), the next bit is parsed.

While parsing each bin, the variable binIdx is incremented by 1 starting with binIdx being set equal to 0 for the first bin.

When the binarization of the corresponding syntax element consists of a prefix and a suffix binarization part,, the variable binIdx is set equal to 0 for the first bin of each part of the bin string (prefix part or suffix part). In this case, after parsing the prefix bit string, the parsing process of the suffix bit string related to the binarizations specified in subclauses 9.3.2.3 and 9.3.2.5 is invoked depending on the resulting prefix bit string as specified in subclauses 9.3.2.3 and 9.3.2.5. Note that for the binarization of the syntax element coded_block_pattern, the suffix bit string is present regardless of the prefix bit string of length 4 as specified in subclause 9.3.2.6.

Depending on the variable bypassFlag, the following applies.

– If bypassFlag is equal to 1, the bypass decoding process as specified in subclause 9.3.3.2.3 is applied for parsing the value of the bins from the bitstream.

– Otherwise (bypassFlag is equal to 0), the parsing of each bin is specified by the following two ordered steps:

1. Given binIdx, maxBinIdxCtx and ctxIdxOffset, ctxIdx is derived as specified in subclause 9.3.3.1.
2. Given ctxIdx, the value of the bin from the bitstream as specified in subclause 9.3.3.2 is decoded.

9.3.3.1 Derivation process for ctxIdx

Inputs to this process are binIdx, maxBinIdxCtx and ctxIdxOffset.

Output of this process is ctxIdx.

Table 9-39 shows the assignment of ctxIdx increments (ctxIdxInc) to binIdx for all ctxIdxOffset values except those related to the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1.

The ctxIdx to be used with a specific binIdx is specified by first determining the ctxIdxOffset associated with the given bin string or part thereof. The ctxIdx is determined as follows.

– If the ctxIdxOffset is listed in Table 9-39, the ctxIdx for a binIdx is the sum of ctxIdxOffset and ctxIdxInc, which is found in Table 9-39. When more than one value is listed in Table 9-39 for a binIdx, the assignment process for ctxIdxInc for that binIdx is further specified in the subclauses given in parenthesis of the corresponding table entry.

– Otherwise (ctxIdxOffset is not listed in Table 9-39), the ctxIdx is specified to be the sum of the following terms: ctxIdxOffset and ctxIdxBlockCatOffset(ctxBlockCat) as specified in Table 9-40 and ctxIdxInc(ctxBlockCat). Subclause 9.3.3.1.3 specifies which ctxBlockCat is used. Subclause 9.3.3.1.1.9 specifies the assignment of ctxIdxInc(ctxBlockCat) for coded_block_flag, and subclause 9.3.3.1.3 specifies the assignment of ctxIdxInc(ctxBlockCat) for significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1.

All bins with binIdx greater than maxBinIdxCtx are parsed using the value of ctxIdx being assigned to binIdx equal to maxBinIdxCtx.

All entries in Table 9-39 labelled with "na" correspond to values of binIdx that do not occur for the corresponding ctxIdxOffset.

ctxIdx = 276 is assigned to the binIdx of mb_type indicating the I_PCM mode. For parsing the value of the corresponding bins from the bitstream, the arithmetic decoding process for decisions before termination as specified in subclause 9.3.3.2.4 is applied.

Table 9-39 – Assignment of ctxIdxInc to binIdx for all ctxIdxOffset values except those related to the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

ctxIdxOffset	binIdx						
	0	1	2	3	4	5	>= 6
0	0,1,2 (subclause 9.3.3.1.1.3)	na	na	na	na	na	na
3	0,1,2 (subclause 9.3.3.1.1.3)	ctxIdx=276	3	4	5,6 (subclause 9.3.3.1.2)	6,7 (subclause 9.3.3.1.2)	7
11	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na	na	na
14	0	1	2,3 (subclause 9.3.3.1.2)	na	na	na	na
17	0	ctxIdx=276	1	2	2,3 (subclause 9.3.3.1.2)	3	3
21	0	1	2	na	na	na	na
24	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na	na	na
27	0,1,2 (subclause 9.3.3.1.1.3)	3	4,5 (subclause 9.3.3.1.2)	5	5	5	5
32	0	ctxIdx=276	1	2	2,3 (subclause 9.3.3.1.2)	3	3
36	0	1	2,3 (subclause 9.3.3.1.2)	3	3	3	na
40	0,1,2 (subclause 9.3.3.1.1.7)	3	4	5	6	6	6
47	0,1,2 (subclause 9.3.3.1.1.7)	3	4	5	6	6	6
54	0,1,2,3 (subclause 9.3.3.1.1.6)	4	5	5	5	5	5
60	0,1 (subclause 9.3.3.1.1.5)	2	3	3	3	3	3
64	0,1,2 (subclause 9.3.3.1.1.8)	3	3	na	na	na	na
68	0	na	na	na	na	na	na
69	0	0	0	na	na	na	na
70	0,1,2 (subclause 9.3.3.1.1.2)	na	na	na	na	na	na
73	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	na	na	na
77	0,1,2,3 (subclause 9.3.3.1.1.4)	4,5,6,7 (subclause 9.3.3.1.1.4)	na	na	na	na	na
276	0	na	na	na	na	na	na
399	0,1,2 (subclause 9.3.3.1.1.10)	na	na	na	na	na	na

Table 9-40 shows the values of ctxIdxBlockCatOffset depending on ctxBlockCat for the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1. The specification of ctxBlockCat is given in Table 9-42.

Table 9-40 – Assignment of ctxIdxBlockCatOffset to ctxBlockCat for syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

Syntax element	ctxBlockCat (as specified in Table 9-42)													
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
coded_block_flag	0	4	8	12	16	0	0	4	8	4	0	4	8	8
significant_coeff_flag	0	15	29	44	47	0	0	15	29	0	0	15	29	0
last_significant_coeff_flag	0	15	29	44	47	0	0	15	29	0	0	15	29	0
coeff_abs_level_minus1	0	10	20	30	39	0	0	10	20	0	0	10	20	0

9.3.3.1.1 Assignment process of ctxIdxInc using neighbouring syntax elements

Subclause 9.3.3.1.1.1 specifies the derivation process of ctxIdxInc for the syntax element mb_skip_flag.

Subclause 9.3.3.1.1.2 specifies the derivation process of ctxIdxInc for the syntax element mb_field_decoding_flag.

Subclause 9.3.3.1.1.3 specifies the derivation process of ctxIdxInc for the syntax element mb_type.

Subclause 9.3.3.1.1.4 specifies the derivation process of ctxIdxInc for the syntax element coded_block_pattern.

Subclause 9.3.3.1.1.5 specifies the derivation process of ctxIdxInc for the syntax element mb_qp_delta.

Subclause 9.3.3.1.1.6 specifies the derivation process of ctxIdxInc for the syntax elements ref_idx_l0 and ref_idx_l1.

Subclause 9.3.3.1.1.7 specifies the derivation process of ctxIdxInc for the syntax elements mvd_l0 and mvd_l1.

Subclause 9.3.3.1.1.8 specifies the derivation process of ctxIdxInc for the syntax element intra_chroma_pred_mode.

Subclause 9.3.3.1.1.9 specifies the derivation process of ctxIdxInc for the syntax element coded_block_flag.

Subclause 9.3.3.1.1.10 specifies the derivation process of ctxIdxInc for the syntax element transform_size_8x8_flag.

9.3.3.1.1.1 Derivation process of ctxIdxInc for the syntax element mb_skip_flag

Output of this process is ctxIdxInc.

When MbaffFrameFlag is equal to 1 and mb_field_decoding_flag has not been decoded (yet) for the current macroblock pair with top macroblock address 2 * (CurrMbAddr / 2), the inference rule for the syntax element mb_field_decoding_flag as specified in subclause 7.4.4 is applied.

The derivation process for neighbouring macroblocks specified in subclause 6.4.10.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If mbAddrN is not available or mb_skip_flag for the macroblock mbAddrN is equal to 1, condTermFlagN is set equal to 0.

– Otherwise (mbAddrN is available and mb_skip_flag for the macroblock mbAddrN is equal to 0), condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$ctxIdxInc = condTermFlagA + condTermFlagB \tag{9-1}$$

9.3.3.1.1.2 Derivation process of ctxIdxInc for the syntax element mb_field_decoding_flag

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblock addresses and their availability in MBAFF frames as specified in subclause 6.4.9 is invoked and the output is assigned to mbAddrA and mbAddrB.

When both macroblocks mbAddrN and mbAddrN + 1 have mb_type equal to P_Skip or B_Skip, the inference rule for the syntax element mb_field_decoding_flag as specified in subclause 7.4.4 is applied for the macroblock mbAddrN.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If any of the following conditions is true, condTermFlagN is set equal to 0,
  – mbAddrN is not available,
  – the macroblock mbAddrN is a frame macroblock.
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$ctxIdxInc = condTermFlagA + condTermFlagB \quad (9\text{-}2)$$

9.3.3.1.1.3 Derivation process of ctxIdxInc for the syntax element mb_type

Input to this process is ctxIdxOffset.

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.10.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If any of the following conditions is true, condTermFlagN is set equal to 0,
  – mbAddrN is not available,
  – ctxIdxOffset is equal to 0 and mb_type for the macroblock mbAddrN is equal to SI,
  – ctxIdxOffset is equal to 3 and mb_type for the macroblock mbAddrN is equal to I_NxN,
  – ctxIdxOffset is equal to 27 and mb_type for the macroblock mbAddrN is equal to P_Skip, B_Skip, or B_Direct_16x16.
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived as

$$ctxIdxInc = condTermFlagA + condTermFlagB \quad (9\text{-}3)$$

9.3.3.1.1.4 Derivation process of ctxIdxInc for the syntax element coded_block_pattern

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

Depending on the value of the variable ctxIdxOffset, the following applies.

– If ctxIdxOffset is equal to 73, the following applies,
  – The derivation process for neighbouring 8x8 luma blocks specified in subclause 6.4.10.2 is invoked with luma8x8BlkIdx = binIdx as input and the output is assigned to mbAddrA, mbAddrB, luma8x8BlkIdxA, and luma8x8BlkIdxB.
  – Let the variable condTermFlagN (with N being either A or B) be derived as follows.
    – If any of the following conditions are true, condTermFlagN is set equal to 0,
      – mbAddrN is not available,
      – mb_type for the macroblock mbAddrN is equal to I_PCM,
      – the macroblock mbAddrN is not the current macroblock CurrMbAddr and the macroblock mbAddrN does not have mb_type equal to P_Skip or B_Skip, and

((CodedBlockPatternLuma >> luma8x8BlkIdxN) & 1) is not equal to 0 for the value of CodedBlockPatternLuma for the macroblock mbAddrN.

– the macroblock mbAddrN is the current macroblock CurrMbAddr and the prior decoded bin value b_k of coded_block_pattern with k = luma8x8BlkIdxN is not equal to 0.

– Otherwise, condTermFlagN is set equal to 1.

– The variable ctxIdxInc is derived as

$$ctxIdxInc = condTermFlagA + 2 * condTermFlagB \quad (9\text{-}4)$$

– Otherwise (ctxIdxOffset is equal to 77), the following applies.

– The derivation process for neighbouring macroblocks specified in subclause 6.4.10.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

– Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If mbAddrN is available and mb_type for the macroblock mbAddrN is equal to I_PCM, condTermFlagN is set equal to 1,

– Otherwise, if any of the following conditions is true, condTermFlagN is set equal to 0,

– mbAddrN is not available or the macroblock mbAddrN has mb_type equal to P_Skip or B_Skip,

– binIdx is equal to 0 and CodedBlockPatternChroma for the macroblock mbAddrN is equal to 0,

– binIdx is equal to 1 and CodedBlockPatternChroma for the macroblock mbAddrN is not equal to 2.

– Otherwise, condTermFlagN is set equal to 1.

– The variable ctxIdxInc is derived as

$$ctxIdxInc = condTermFlagA + 2 * condTermFlagB + ((binIdx == 1) ? 4 : 0) \quad (9\text{-}5)$$

NOTE – When a macroblock uses an Intra_16x16 prediction mode, the values of CodedBlockPatternLuma and CodedBlockPatternChroma for the macroblock are derived from mb_type as specified in Table 7-11.

9.3.3.1.1.5 Derivation process of ctxIdxInc for the syntax element mb_qp_delta

Output of this process is ctxIdxInc.

Let prevMbAddr be the macroblock address of the macroblock that precedes the current macroblock in decoding order. When the current macroblock is the first macroblock of a slice, prevMbAddr is marked as not available.

Let the variable ctxIdxInc be derived as follows.

– If any of the following conditions is true, ctxIdxInc is set equal to 0,

– prevMbAddr is not available or the macroblock prevMbAddr has mb_type equal to P_Skip or B_Skip,

– mb_type of the macroblock prevMbAddr is equal to I_PCM,

– The macroblock prevMbAddr is not coded in Intra_16x16 prediction mode and both CodedBlockPatternLuma and CodedBlockPatternChroma for the macroblock prevMbAddr are equal to 0,

– mb_qp_delta for the macroblock prevMbAddr is equal to 0,

– Otherwise, ctxIdxInc is set equal to 1.

9.3.3.1.1.6 Derivation process of ctxIdxInc for the syntax elements ref_idx_l0 and ref_idx_l1

Input to this process is mbPartIdx.

Output of this process is ctxIdxInc.

The interpretation of ref_idx_lX and Pred_LX within this subclause is specified as follows.

– If this process is invoked for the derivation of ref_idx_l0, ref_idx_lX is interpreted as ref_idx_l0 and Pred_LX is interpreted as Pred_L0.

– Otherwise (this process is invoked for the derivation of ref_idx_l1), ref_idx_lX is interpreted as ref_idx_l1 and Pred_LX is interpreted as Pred_L1.

The derivation process for neighbouring partitions specified in subclause 6.4.10.7 is invoked with mbPartIdx, currSubMbType, and subMbPartIdx = 0 as input and the output is assigned to mbAddrA\mbPartIdxA and mbAddrB\mbPartIdxB.

With ref_idx_lX[mbPartIdxN] (with N being either A or B) specifying the syntax element for the macroblock mbAddrN, let the variable refIdxZeroFlagN be derived as follows.

– If MbaffFrameFlag is equal to 1, the current macroblock is a frame macroblock, and the macroblock mbAddrN is a field macroblock

$$refIdxZeroFlagN = ((ref\_idx\_lX[mbPartIdxN] > 1) ? 0 : 1) \quad (9\text{-}6)$$

– Otherwise,

$$refIdxZeroFlagN = ((ref\_idx\_lX[mbPartIdxN] > 0) ? 0 : 1) \quad (9\text{-}7)$$

Let the variable predModeEqualFlagN be specified as follows.

– If the macroblock mbAddrN has mb_type equal to P_8x8 or B_8x8, the following applies.
  – If SubMbPredMode(sub_mb_type[mbPartIdxN]) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where sub_mb_type specifies the syntax element for the macroblock mbAddrN.
  – Otherwise, predModeEqualFlagN is set equal to 1.
– Otherwise, the following applies.
  – If MbPartPredMode(mb_type, mbPartIdxN) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where mb_type specifies the syntax element for the macroblock mbAddrN.
  – Otherwise, predModeEqualFlagN is set equal to 1.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If any of the following conditions is true, condTermFlagN is set equal to 0,
  – mbAddrN is not available,
  – the macroblock mbAddrN has mb_type equal to P_Skip or B_Skip,
  – the macroblock mbAddrN is coded in Intra prediction mode,
  – predModeEqualFlagN is equal to 0,
  – refIdxZeroFlagN is equal to 1.
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived as

$$ctxIdxInc = condTermFlagA + 2 * condTermFlagB \quad (9\text{-}8)$$

9.3.3.1.1.7 Derivation process of ctxIdxInc for the syntax elements mvd_l0 and mvd_l1

Inputs to this process are mbPartIdx, subMbPartIdx, and ctxIdxOffset.

Output of this process is ctxIdxInc.

The interpretation of mvd_lX and Pred_LX within this subclause is specified as follows.

– If this process is invoked for the derivation of mvd_l0, mvd_lX is interpreted as mvd_l0 and Pred_LX is interpreted as Pred_L0.
– Otherwise (this process is invoked for the derivation of mvd_l1), mvd_lX is interpreted as mvd_l1 and Pred_LX is interpreted as Pred_L1.

Let currSubMbType be set equal to sub_mb_type[mbPartIdx].

The derivation process for neighbouring partitions specified in subclause 6.4.10.7 is invoked with mbPartIdx, currSubMbType, and subMbPartIdx as input and the output is assigned to mbAddrA\mbPartIdxA\subMbPartIdxA and mbAddrB\mbPartIdxB\subMbPartIdxB.

Let the variable compIdx be derived as follows.

– If ctxIdxOffset is equal to 40, compIdx is set equal to 0.

– Otherwise (ctxIdxOffset is equal to 47), compIdx is set equal to 1.

Let the variable predModeEqualFlagN be specified as follows.

– If the macroblock mbAddrN has mb_type equal to P_8x8 or B_8x8, the following applies.

  – If SubMbPredMode(sub_mb_type[mbPartIdxN]) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where sub_mb_type specifies the syntax element for the macroblock mbAddrN.

  – Otherwise, predModeEqualFlagN is set equal to 1.

– Otherwise, the following applies.

  – If MbPartPredMode(mb_type, mbPartIdxN) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where mb_type specifies the syntax element for the macroblock mbAddrN.

  – Otherwise, predModeEqualFlagN is set equal to 1.

Let the variable absMvdCompN (with N being either A or B) be derived as follows.

– If any of the following conditions is true, absMvdCompN is set equal to 0.

  – mbAddrN is not available,

  – the macroblock mbAddrN has mb_type equal to P_Skip or B_Skip,

  – the macroblock mbAddrN is coded in an Intra prediction mode,

  – predModeEqualFlagN is equal to 0.

– Otherwise, the following applies.

  – If compIdx is equal to 1, MbaffFrameFlag is equal to 1, the current macroblock is a frame macroblock, and the macroblock mbAddrN is a field macroblock

$$absMvdCompN = Abs(mvd\_lX[mbPartIdxN][subMbPartIdxN][compIdx]) * 2 \quad (9\text{-}9)$$

  – Otherwise, if compIdx is equal to 1, MbaffFrameFlag is equal to 1, the current macroblock is a field macroblock, and the macroblock mbAddrN is a frame macroblock

$$absMvdCompN = Abs(mvd\_lX[mbPartIdxN][subMbPartIdxN][compIdx]) / 2 \quad (9\text{-}10)$$

  – Otherwise,

$$absMvdCompN = Abs(mvd\_lX[mbPartIdxN][subMbPartIdxN][compIdx]) \quad (9\text{-}11)$$

The variable ctxIdxInc is derived as follows.

– If (absMvdCompA + absMvdCompB) is less than 3, ctxIdxInc is set equal to 0.

– Otherwise, if (absMvdCompA + absMvdCompB) is greater than 32, ctxIdxInc is set equal to 2.

– Otherwise ((absMvdCompA + absMvdCompB) is in the range of 3 to 32, inclusive), ctxIdxInc is set equal to 1.

9.3.3.1.1.8 Derivation process of ctxIdxInc for the syntax element intra_chroma_pred_mode

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.10.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being replaced by either A or B) be derived as follows.

– If any of the following conditions is true, condTermFlagN is set equal to 0.
  – mbAddrN is not available,
  – The macroblock mbAddrN is coded in Inter prediction mode,
  – mb_type for the macroblock mbAddrN is equal to I_PCM,
  – intra_chroma_pred_mode for the macroblock mbAddrN is equal to 0.
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$ctxIdxInc = condTermFlagA + condTermFlagB \quad (9\text{-}12)$$

9.3.3.1.1.9 Derivation process of ctxIdxInc for the syntax element coded_block_flag

Input to this process is ctxBlockCat and additional input is specified as follows.

– If ctxBlockCat is equal to 0, 6, or 10, no additional input.
– Otherwise, if ctxBlockCat is equal to 1 or 2, luma4x4BlkIdx.
– Otherwise, if ctxBlockCat is equal to 3, the chroma component index iCbCr.
– Otherwise, if ctxBlockCat is equal to 4, chroma4x4BlkIdx and the chroma component index iCbCr.
– Otherwise, if ctxBlockCat is equal to 5, luma8x8BlkIdx.
– Otherwise, if ctxBlockCat is equal to 7 or 8, cb4x4BlkIdx.
– Otherwise, if ctxBlockCat is equal to 9, cb8x8BlkIdx.
– Otherwise, if ctxBlockCat is equal to 11 or 12, cr4x4BlkIdx.
– Otherwise (ctxBlockCat is equal to 13), cr8x8BlkIdx.

Output of this process is ctxIdxInc(ctxBlockCat).

Let the variable transBlockN (with N being either A or B) be derived as follows.

– If ctxBlockCat is equal to 0, 6, or 10, the following applies.
  – The derivation process for neighbouring macroblocks specified in subclause 6.4.10.1 is invoked and the output is assigned to mbAddrN (with N being either A or B).
  – The variable transBlockN is derived as follows.
  – If mbAddrN is available and the macroblock mbAddrN is coded in Intra_16x16 prediction mode, the following applies.
    – If ctxBlockCat is equal to 0, the luma DC block of macroblock mbAddrN is assigned to transBlockN.
    – Otherwise, if ctxBlockCat is equal to 6, the Cb DC block of macroblock mbAddrN is assigned to transBlockN.
    – Otherwise (ctxBlockCat is equal to 10), the Cr DC block of macroblock mbAddrN is assigned to transBlockN.
  – Otherwise, transBlockN is marked as not available.
– Otherwise, if ctxBlockCat is equal to 1 or 2, the following applies.
  – The derivation process for neighbouring 4x4 luma blocks specified in subclause 6.4.10.4 is invoked with luma4x4BlkIdx as input and the output is assigned to mbAddrN, luma4x4BlkIdxN (with N being either A or B).
  – The variable transBlockN is derived as follows.
    – If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, ((CodedBlockPatternLuma >> (luma4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 0 for the macroblock mbAddrN, the 4x4 luma block with index luma4x4BlkIdxN of macroblock mbAddrN is assigned to transBlockN.

- Otherwise, if mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip or B_Skip, ((CodedBlockPatternLuma >> (luma4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 luma block with index (luma4x4BlkIdxN >> 2) of macroblock mbAddrN is assigned to transBlockN.
- Otherwise, transBlockN is marked as not available.

– Otherwise, if ctxBlockCat is equal to 3, the following applies.
- The derivation process for neighbouring macroblocks specified in subclause 6.4.10.1 is invoked and the output is assigned to mbAddrN (with N being either A or B).
- The variable transBlockN is derived as follows.
  - If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, and CodedBlockPatternChroma is not equal to 0 for the macroblock mbAddrN, the chroma DC block of chroma component iCbCr of macroblock mbAddrN is assigned to transBlockN.
  - Otherwise, transBlockN is marked as not available.

– Otherwise, if ctxBlockCat is equal to 4, the following applies.
- The derivation process for neighbouring 4x4 chroma blocks specified in subclause 6.4.10.5 is invoked with chroma4x4BlkIdx as input and the output is assigned to mbAddrN, chroma4x4BlkIdxN (with N being either A or B).
- The variable transBlockN is derived as follows.
  - If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, and CodedBlockPatternChroma is equal to 2 for the macroblock mbAddrN, the 4x4 chroma block with chroma4x4BlkIdxN of the chroma component iCbCr of macroblock mbAddrN is assigned to transBlockN.
  - Otherwise, transBlockN is marked as not available.

– Otherwise, if ctxBlockCat is equal to 5, the following applies.
- The derivation process for neighbouring 8x8 luma blocks specified in subclause 6.4.10.2 is invoked with luma8x8BlkIdx as input and the output is assigned to mbAddrN, luma8x8BlkIdxN (with N being either A or B).
- The variable transBlockN is derived as follows.
  - If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, ((CodedBlockPatternLuma >>luma8x8BlkIdx) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 luma block with index luma8x8BlkIdxN of macroblock mbAddrN is assigned to transBlockN.
  - Otherwise, transBlockN is marked as not available.

– Otherwise, if ctxBlockCat is equal to7 or 8, the following applies.
- The derivation process for neighbouring 4x4 Cb blocks specified in subclause 6.4.10.5 is invoked with cb4x4BlkIdx as input and the output is assigned to mbAddrN, cb4x4BlkIdxN (with N being either A or B).
- The variable transBlockN is derived as follows.
  - If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, ((CodedBlockPatternLuma >> (cb4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 0 for the macroblock mbAddrN, the 4x4 Cb block with index cb4x4BlkIdxN of macroblock mbAddrN is assigned to transBlockN.
  - Otherwise, if mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip or B_Skip, ((CodedBlockPatternLuma >> (cb4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 Cb block with index (cb4x4BlkIdxN >> 2) of macroblock mbAddrN is assigned to transBlockN.
  - Otherwise, transBlockN is marked as not available.

– Otherwise, if ctxBlockCat is equal to 9, the following applies.
- The derivation process for neighbouring 8x8 Cb blocks specified in subclause 6.4.10.3 is invoked with cb8x8BlkIdx as input and the output is assigned to mbAddrN, cb8x8BlkIdxN (with N being either A or B).

- The variable transBlockN is derived as follows.
  - If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, ((CodedBlockPatternLuma >>cb8x8BlkIdx) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 Cb block with index cb8x8BlkIdxN of macroblock mbAddrN is assigned to transBlockN.
  - Otherwise, transBlockN is marked as not available.
- Otherwise, if ctxBlockCat is equal to 11 or 12, the following applies.
  - The derivation process for neighbouring 4x4 Cr blocks specified in subclause 6.4.10.5 is invoked with cr4x4BlkIdx as input and the output is assigned to mbAddrN, cr4x4BlkIdxN (with N being either A or B).
  - The variable transBlockN is derived as follows.
    - If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, ((CodedBlockPatternLuma >> (cr4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 0 for the macroblock mbAddrN, the 4x4 Cr block with index cr4x4BlkIdxN of macroblock mbAddrN is assigned to transBlockN.
    - Otherwise, if mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip or B_Skip, ((CodedBlockPatternLuma >> (cr4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 Cr block with index (cr4x4BlkIdxN >> 2) of macroblock mbAddrN is assigned to transBlockN.
    - Otherwise, transBlockN is marked as not available.
- Otherwise (ctxBlockCat is equal to 13), the following applies.
  - The derivation process for neighbouring 8x8 Cr blocks specified in subclause 6.4.10.3 is invoked with cr8x8BlkIdx as input and the output is assigned to mbAddrN, cr8x8BlkIdxN (with N being either A or B).
  - The variable transBlockN is derived as follows.
    - If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM,, ((CodedBlockPatternLuma >>cr8x8BlkIdx) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 Cr block with index cr8x8BlkIdxN of macroblock mbAddrN is assigned to transBlockN.
    - Otherwise, transBlockN is marked as not available.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

- If any of the following conditions is true, condTermFlagN is set equal to 0.
  - mbAddrN is not available and the current macroblock is coded in Inter prediction mode.
  - mbAddrN is available and transBlockN is not available and mb_type for the macroblock mbAddrN is not equal to I_PCM.
  - The current macroblock is coded in Intra prediction mode, constrained_intra_pred_flag is equal to 1, the macroblock mbAddrN is available and coded in Inter prediction mode, and slice data partitioning is in use (nal_unit_type is in the range of 2 through 4, inclusive).
- Otherwise, if any of the following conditions is true, condTermFlagN is set equal to 1.
  - mbAddrN is not available and the current macroblock is coded in Intra prediction mode.
  - mb_type for the macroblock mbAddrN is equal to I_PCM.
- Otherwise, condTermFlagN is set equal to the value of the coded_block_flag of the transform block transBlockN that was decoded for the macroblock mbAddrN.

The variable ctxIdxInc(ctxBlockCat) is derived by

$$ctxIdxInc(ctxBlockCat) = condTermFlagA + 2 * condTermFlagB \quad (9\text{-}13)$$

9.3.3.1.1.10 Derivation process of ctxIdxInc for the syntax element transform_size_8x8_flag

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.10.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If any of the following conditions is true, condTermFlagN is set equal to 0.
  – mbAddrN is not available,
  – transform_size_8x8_flag for the macroblock mbAddrN is equal to 0.
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$ctxIdxInc = condTermFlagA + condTermFlagB \quad (9\text{-}14)$$

9.3.3.1.2 Assignment process of ctxIdxInc using prior decoded bin values

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

Table 9-41 contains the specification of ctxIdxInc for the given values of ctxIdxOffset and binIdx.

For each value of ctxIdxOffset and binIdx, ctxIdxInc is derived by using some of the values of prior decoded bin values (b_0, b_1, b_2,…, b_k), where the value of the index k is less than the value of binIdx.

Table 9-41 – Specification of ctxIdxInc for specific values of ctxIdxOffset and binIdx

Value (name) of ctxIdxOffset	binIdx	ctxIdxInc
3	4	(b_3 != 0) ? 5: 6
	5	(b_3 != 0) ? 6: 7
14	2	(b_1 != 1) ? 2: 3
17	4	(b_3 != 0) ? 2: 3
27	2	(b_1 != 0) ? 4: 5
32	4	(b_3 != 0) ? 2: 3
36	2	(b_1 != 0) ? 2: 3

9.3.3.1.3 Assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

The assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1 as well as for coded_block_flag depends on categories of different blocks denoted by the variable ctxBlockCat. The specification of these block categories is given in Table 9-42.