**INTERNATIONAL STANDARD ISO/IEC 14496-3:2005/Amd.2:2006**
TECHNICAL CORRIGENDUM 2

Published 2008-12-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

# Information technology — Coding of audio-visual objects —
Part 3:
# Audio

AMENDMENT 2: Audio Lossless Coding (ALS), new audio profiles and BSAC extensions

TECHNICAL CORRIGENDUM 2

*Technologies de l'information — Codage des objets audiovisuels —*

*Partie 3: Codage audio*

*AMENDEMENT 2: Codage audio sans perte (ALS), nouveaux profils audio et extensions BSAC*

*RECTIFICATIF TECHNIQUE 2*

Technical Corrigendum 2 to ISO/IEC 14496-3:2005/Amd.2:2006 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

---

*Throughout this Corrigendum, modifications or additions to existing text are highlighted in grey.*

*Page 4, after Table 1.11A, replace:*

A HE AAC v2 Profile decoder of a certain level shall operate the HQ SBR tool for streams containing Parametric Stereo data. For streams not containing Parametric Stereo data, the HE AAC v2 Profile decoder may operate the HQ SBR tool, or the LP SBR tool.

**ICS 35.040** **Ref. No. ISO/IEC 14496-3:2005/Amd.2:2006/Cor.2:2008(E)**

© ISO/IEC 2008 – All rights reserved

Published in Switzerland

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

*with:*

An HE AAC v2 Profile decoder shall operate the HQ SBR tool for bitstreams containing Parametric Stereo data. For bitstreams not containing Parametric Stereo data, the HE AAC v2 Profile decoder may operate the HQ SBR tool, or the LP SBR tool.
Only bitstreams consisting of exactly one AAC single channel element may contain Parametric Stereo data. Bitstreams containing more than one channel in the AAC part shall not contain Parametric Stereo data.

*Page 6, in Table 1.13 replace the rows:*

```
    case 36:
        ALSSpecificConfig();
        break;
```

*with:*

```
    case 36:
        fillBits;                                   5       bslbf
        ALSSpecificConfig();
        break;
```

*In ISO/IEC 14496-3:2005, after 1.6.3.14 add:*

#### 1.6.3.15 FillBits

Fill bits for byte alignment of ALSSpecificConfig() relative to the start of AudioSpecificConfig().

*Page 8, replace the final sentence:*

Similarly the HE AAC v2 decoder can handle all HE AAC Profile streams as well as all AAC Profile streams.

*with:*

Similarly an HE AAC v2 profile decoder of a certain level can handle all HE AAC Profile streams of the same or lower level as well as all AAC Profile streams of the same or lower level.

*In 11.2.1, replace:*

(see subclause 11.6.7)

*with:*

(see subclause 11.6.5.2.1 and 11.6.7)

*Page 18, at the end of 11.3.3 add:*

x.y Fixed-point signed fractional representation, where x is the number of bits to the left of the binary point, and y is the number of bits to the right of the binary point (two's complement sign representation). 64-bit signed integer (two's complement)

*Page 18, replace Table 11.1 with the following table:*


© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

**Table 11.1 — Syntax of ALSSpecificConfig**

| Syntax | No. of bits | Mnemonic |
|---|---|---|
| ALSSpecificConfig() | | |
| { | | |
| **als_id;** | **32** | **uimsbf** |
| **samp_freq;** | **32** | **uimsbf** |
| **samples;** | **32** | **uimsbf** |
| **channels;** | **16** | **uimsbf** |
| **file_type;** | **3** | **uimsbf** |
| **resolution;** | **3** | **uimsbf** |
| **floating;** | **1** | **uimsbf** |
| **msb_first;** | **1** | **uimsbf** |
| **frame_length;** | **16** | **uimsbf** |
| **random_access;** | **8** | **uimsbf** |
| **ra_flag;** | **2** | **uimsbf** |
| **adapt_order;** | **1** | **uimsbf** |
| **coef_table;** | **2** | **uimsbf** |
| **long_term_prediction;** | **1** | **uimsbf** |
| **max_order;** | **10** | **uimsbf** |
| **block_switching;** | **2** | **uimsbf** |
| **bgmc_mode;** | **1** | **uimsbf** |
| **sb_part;** | **1** | **uimsbf** |
| **joint_stereo;** | **1** | **uimsbf** |
| **mc_coding;** | **1** | **uimsbf** |
| **chan_config;** | **1** | **uimsbf** |
| **chan_sort;** | **1** | **uimsbf** |
| **crc_enabled;** | **1** | **uimsbf** |
| **RLSLMS** | **1** | **uimsbf** |
| **(reserved)** | **5** | |
| **aux_data_enabled;** | **1** | **uimsbf** |
| if (chan_config) { | | |
| **chan_config_info;** | **16** | **uimsbf** |
| } | | |
| if (chan_sort) { | | |
| for (c = 0; c < channels; c++) | | |
| **chan_pos[c];** | **1..16** | **uimsbf** |
| } | | |
| **byte_align;** | **0..7** | **bslbf** |
| **header_size;** | **32** | **uimsbf** |
| **trailer_size;** | **32** | **uimsbf** |
| **orig_header[];** | **header_size * 8** | **bslbf** |
| **orig_trailer[];** | **trailer_size * 8** | **bslbf** |
| if (crc_enabled) { | | |
| **crc;** | **32** | **uimsbf** |
| } | | |
| if ((ra_flag == 2) && (random_access > 0)) { | | |
| for (f = 0; f < ((samples-1) / (frame_length+1)) + 1; f++) { | | |
| **ra_unit_size[f]** | **32** | **uimsbf** |
| } | | |
| } | | |
| if (aux_data_enabled) { | | |
| **aux_size;** | **32** | **uimsbf** |
| **aux_data[];** | **aux_size * 8** | **bslbf** |
| } | | |
| } | | |

**Note**: “byte_align” denotes byte alignment of subsequent data relative to the start of ALSSpecificConfig().

© ISO/IEC 2008 - All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

*Page 21, in Table 11.3, replace:*

| | | |
|---|---|---|
| if (adapt_order == 1) {<br>**opt_order;**<br>}<br>for (p = 0; p < opt_order; p++) {<br>**quant_cof[p];**<br>} | **1..10**<br><br>**varies** | **uimsbf**<br><br>**Rice code** |

| | | |
|---|---|---|
| if (RLSLMS) {<br>RLSLMS_extension_data()<br>}<br>} | | |

*with:*

| | | |
|---|---|---|
| if (adapt_order == 1) {<br>**opt_order;**<br>}<br>else {<br>opt_order = max_order;<br>}<br>for (p = 0; p < opt_order; p++) {<br>**quant_cof[p];**<br>} | **1..10**<br><br><br><br><br>**varies** | **uimsbf**<br><br><br><br><br>**Rice code** |

| | | |
|---|---|---|
| if (RLSLMS) {<br>RLSLMS_extension_data()<br>}<br>**byte_align;**<br>} | <br><br><br>**0..7** | <br><br><br>**bslbf** |


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2008 – All rights reserved

*Page 23, replace Table 11.5 with the following table:*

**Table 11.5 – Syntax of RLSLMS_extension_data**

| Syntax | No. of bits | Mnemonic |
|---|---|---|
| RLSLMS_extension() | | |
| { | | |
| **mono_block** | **1** | **uimsbf** |
| **ext_mode** | **1** | **uimsbf** |
| if (ext_mode) { | | |
| **extension_bits** | **3** | **uimsbf** |
| if (extension_bits&0x01) { | | |
| **RLS_order** | **4** | **uimsbf** |
| **LMS_stage** | **3** | **uimsbf** |
| for(i=0; i<LMS_stage;i++){ | | |
| **LMS_order[i]** | **5** | **uimsbf** |
| } | | |
| } | | |
| if (extension_bits&0x02) { | | |
| if (RLS_order) { | | |
| **RLS_lambda** | **10** | **uimsbf** |
| if (RA) | | |
| **RLS_lambda_ra** | **10** | **uimsbf** |
| } | | |
| } | | |
| if (extension_bits&04) { | | |
| for(i=0; i<LMS_stage;i++) { | | |
| **LMS_mu[i]** | **5** | **uimsbf** |
| } | | |
| **LMS_stepsize** | **3** | **uimsbf** |
| } | | |
| } | | |
| } | | |

*Page 25, replace Table 11.9 with the following table:*

**Table 11.9 — Elements of ALSSpecificConfig**

| Field | #Bits | Description / Values |
|---|---|---|
| als_id | 32 | ALS identifier<br>fixed value = 1095521024 = 0x414C5300 (Hex) |
| samp_freq | 32 | Sampling frequency in Hz |
| samples | 32 | Number of samples (per channel)<br>If samples = 0xFFFFFFFF (Hex), the number of samples is not specified (see 11.6.1.3) |
| channels | 16 | Number of channels-1<br>(0 = mono, 1 = stereo, …) |
| file_type | 3 | 000 = unknown / raw file<br>001 = wave file<br>010 = aiff file<br>011 = bwf file<br>(other values are reserved) |

© ISO/IEC 2008 - All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

| | | |
|---|---|---|
| resolution | 3 | 000 = 8-bit<br>001 = 16-bit<br>010 = 24-bit<br>011 = 32-bit<br>(other values are reserved) |
| floating | 1 | 1 = IEEE 32-bit floating-point, 0 = integer |
| msb_first | 1 | Original byte order of the input audio data:<br>0 = least significant byte first (little-endian)<br>1 = most significant byte first (big-endian)<br>If resolution = 0 (8-bit data), msb_first = 0 indicates unsigned data (0…255), while msb_first = 1 indicates signed data (-128…127). |
| frame_length | 16 | Frame Length - 1 (e.g. frame_length = 0x1FFF signals a frame length of N = 8192) |
| random_access | 8 | Distance between RA frames (in frames, 0…255). If no RA is used, the value is zero. If each frame is an RA frame, the value is 1. |
| ra_flag | 2 | Indicates where the size of random access units (ra_unit_size) is stored:<br>00: not stored<br>01: stored at the beginning of frame_data()<br>10: stored at the end of ALSSpecificConfig() |
| adapt_order | 1 | Adaptive Order: 1 = on, 0 = off |
| coef_table | 2 | Table index (00, 01, or 10, see Table 11.20) of Rice code parameters for entropy coding of predictor coefficients, 11 = no entropy coding |
| long_term_prediction | 1 | Long term prediction (LTP): 1 = on, 0 = off |
| max_order | 10 | Maximum prediction order (0..1023) |
| block_switching | 2 | Number of block switching levels:<br>00 = no block switching<br>01 = up to 3 levels<br>10 = 4 levels<br>11 = 5 levels |
| bgmc_mode | 1 | BGMC Mode: 1 = on, 0 = off (Rice coding only) |
| sb_part | 1 | Sub-block partition for entropy coding of the residual.<br>if bgmc_mode = 0:<br>0 = no partition, no ec_sub bit in block_data<br>1 = 1:4 partition, one ec_sub bit in block_data<br>if bgmc_mode = 1:<br>0 = 1:4 partition, one ec_sub bit in block_data<br>1 = 1:2:4:8 partition, two ec_sub bits in block_data |
| joint_stereo | 1 | Joint Stereo: 1 = on, 0 = off<br>If channels = 0 (mono), joint_stereo = 0 |

--`,,```,,,,````-`-`,,`,,`,`,,`---


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2008 – All rights reserved
Not for Resale

| | | |
|---|---|---|
| mc_coding | 1 | Extended inter-channel coding: 1 = on, 0 = off<br>If channels = 0 (mono), mc_coding = 0 |
| chan_config | 1 | Indicates that a chan_config_info field is present |
| chan_sort | 1 | Channel rearrangement: 1 = on, 0 = off<br>If channels = 0 (mono), chan_sort = 0 |
| crc_enabled | 1 | Indicates that the crc field is present |
| RLSLMS | 1 | Use RLS-LMS predictor: 1 = on, 0 = off |
| (reserved) | 5 | |
| aux_data_enabled | 1 | Indicates that auxiliary data is present (fields aux_size and aux_data) |
| chan_config_info | 16 | Mapping of channels to loudspeaker locations. Each bit indicates whether a channel for a particular predefined location exists (see 11.6.1.5). |
| chan_pos[] | (channels+1)*ChBits | If channel rearrangement is on (chan_sort = 1), these are the original channel positions. The number of bits per channel is<br>ChBits = ceil[log2(channels+1)] = 1..16<br>where channels+1 is the number of channels. |
| header_size | 32 | Header size of original audio file in bytes<br>If header_size = 0xFFFFFFFF (Hex), there is no orig_header[] field, but the original header may be stored elsewhere, e.g. in the meta data of an MPEG-4 file. |
| trailer_size | 32 | Trailer size of original audio file in bytes<br>If trailer_size = 0xFFFFFFFF (Hex), there is no orig_trailer[] field, but the original trailer may be stored elsewhere, e.g. in the meta data of an MPEG-4 file. |
| orig_header[] | header_size*8 | Header of original audio file |
| orig_trailer[] | trailer_size*8 | Trailer of original audio file |
| crc | 32 | 32-bit CCITT-32 CRC checksum of the original audio data bytes (polynomial: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$). |
| ra_unit_size[] | #frames*32 | Distances (in bytes) between the random access frames, i.e. the sizes of the random access units, where the number of frames is<br>#frames = ((samples-1) / (frame_length+1)) +1<br>In ALSSpecificConfig(), this field appears only if ra_flag = 2. |
| aux_size | 32 | Size of the aux_data field in bytes<br>If aux_size = 0xFFFFFFFF (Hex), there is no aux_data[] field, but the auxiliary data may be stored elsewhere, e.g. in the meta data of an MPEG-4 file. |
| aux_data[] | aux_size*8 | Auxiliary data (not required for decoding) |

© ISO/IEC 2008 - All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

*Page 28, in Table 11.10 replace:*

ra_flag = 2

*with:*

ra_flag = 1

*Page 30, replace Table 11.13 with the following table:*

**Table 11.13 — Elements of RLSLMS_extension_data**

| Field | #Bits | Description / Values |
|---|---|---|
| mono_block | 1 | mono_frame == 0: CPE coded with joint-stereo RLS<br>mono_frame == 1: CPE coded with mono RLS |
| ext_mode | 1 | RLS-LMS predictor parameters are updated in extension block.<br>1 == extension block<br>0 == non-extension block |
| extension_bits | 3 | Type of RLS-LMS parameters carried in extension block<br>xtension&01 == RLS-LMS predictors orders<br>extension&02 == RLS_lambda and RLS_lambda_ra<br>extension&04 == LMS_mu and LMS_stepsize |
| RLS_order | 4 | RLS predictor order |
| LMS_stage | 3 | Number of LMS predictors in cascade |
| LMS_order[] | 5*LMS_stage | LMS predictor order |
| RLS_lambda | 10 | RLS predictor parameter lambda. |
| RLS_lambda_ra | 10 | RLS predictor parameter lambda for random access frame |
| LMS_mu[] | 5*LMS_stage | LMS predictor parameter – NLMS stepsize |
| LMS_stepsize | 3 | Linear combiner parameter – Sign-Sign LMS stepsize |

*Page 33, 11.6.1.2, add at the top of the list of bullet items:*

- ALS identifier: This field must contain the value 1095521024 = 0x414C5300 (Hex). Using byte-wise reading, the first three bytes are equivalent to the ASCII codes for 'ALS'.

--`,,```,,,,````-`-`,,`,,`,`,,`---

8

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

© ISO/IEC 2008 – All rights reserved

Not for Resale

*Page 34, 11.6.1.3, replace the pseudo code with the following pseudo code:*

```
N = frame_length + 1.
frames = samples / N;
remainder = samples % N;
if (remainder)
{
    frames++;
    N_last = remainder;
}
else
    N_last = N;
```

*and add the following paragraphs at the end of the subclause:*

If the value of samples is 0xFFFFFFFF (Hex), the number of samples is not specified. If the ALS payload is stored using the MPEG-4 file format, the number of samples can be obtained from the meta data of the file. In that case, the number of frames and the length of the last frame are determined as described above.

If the number of samples is not available, the number of frames is undefined, and all frames are assumed to have the same length N. In that case, the sizes of random access units shall not be stored in ALSSpecifcConfig (i.e. only ra_flag = 0 or ra_flag = 1 shall be used), since the number of random access units is undefined as well.

*Page 37, 11.6.2, immediately before Table 11.18, add:*

The frame length N must be divisible by $2^{levels}$ without remainder, in order to obtain integer block lengths $N_B$.

*Page 43, 11.6.3.2.1, replace:*

The following algorithm describes the calculation of the residual *d* for an input signal *x*, a predictor order *K* and LPC coefficients *cof*:

*with:*

The following algorithm describes the calculation of the residual *d* for an input signal *x*, a block length *N*, a predictor order *K* and LPC coefficients *cof*:

*Page 44, 11.6.3.2.1, second pseudo-code listing and Page 45, 11.6.3.2.2, second pseudo-code listing, replace in both cases the fourth line:*

```
for (n = 0; n < K; n++)
```

*with:*

```
for (n = 0; n < min(K, N); n++)
```

*Page 48, replace 11.6.5 entirely, with the following text:*

### 11.6.5 RLS-LMS Predictor

This subclause describes the backward-adaptive prediction scheme that uses adaptive RLS-LMS predictor. Block diagram of the corresponding encoder and decoder are shown in Figure 11.8 and Figure 11.9, respectively.

© ISO/IEC 2008 - All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

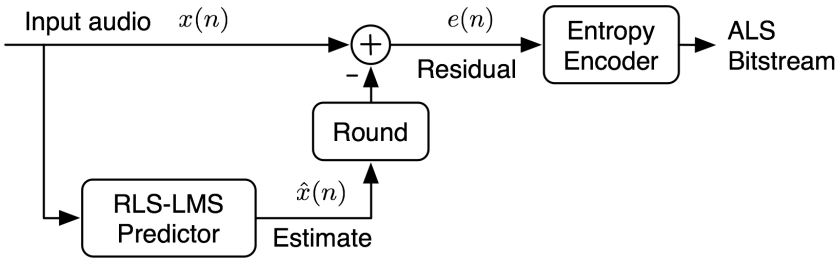


**Figure 11.8 — Encoder of the RLS-LMS backward-adaptive prediction scheme**

In the encoder, the RLS-LMS predictor generates an estimate of the current input audio sample by using the past samples. This estimate is subtracted from the current sample to generate the residual, which is subsequently coded by the entropy encoder (11.6.6) to form the ALS bit-stream.

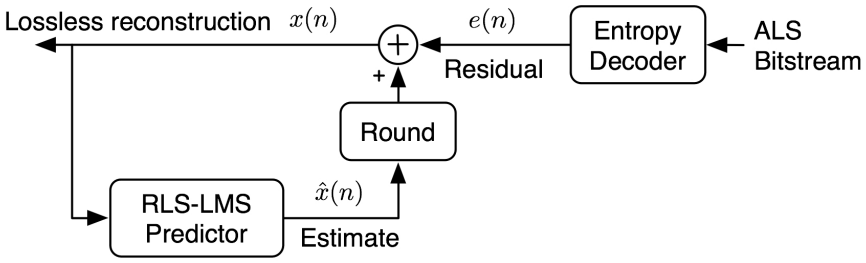


**Figure 11.9 — Decoder of the RLS-LMS backward-adaptive prediction scheme**

In the decoder, a reverse process is performed. The entropy decoder decodes the ALS bit-stream to the residual, which is then added to the estimate of the RLS-LMS predictor to re-generate the original audio sample.

As shown in Figure 11.10, the RLS-LMS predictor consists of a cascade of predictors in the order of a Differential Pulse Code Modulation (DPCM) predictor (11.6.5.1), a Recursive Least Square (RLS) predictor (11.6.5.2), and a series of Least Mean Square (LMS) predictors (11.6.5.3). The input samples sequentially pass through the cascade of predictors. The residual of one predictor serves as the input to the next predictor. The estimates from the predictors in the cascade are weighted and added together by a linear combiner (11.6.5.4) to generate the final estimate of the current input sample.

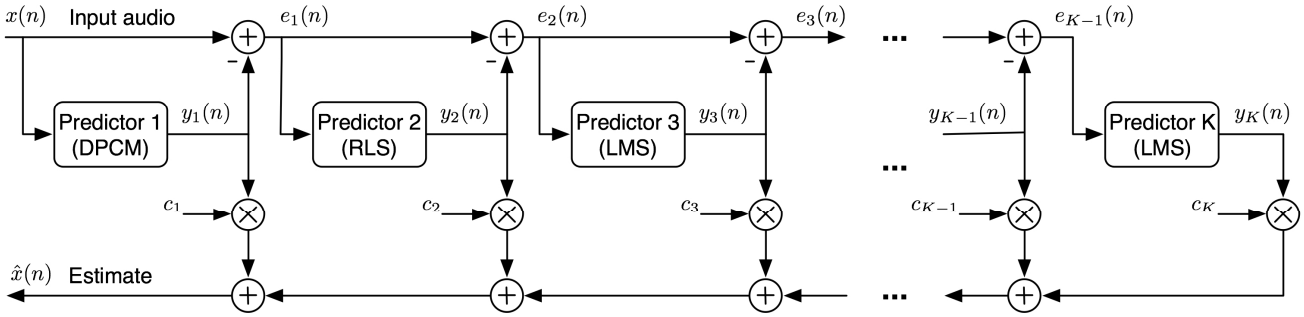


**Figure 11.1 — Block diagram of the RLS-LMS predictor**

The RLS-LMS predictor can be turned on/off by setting RLSLMS in ALSSpecificConfig() to 1/0, respectively.


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

#### 11.6.5.1 DPCM Predictor

**Input**: x(n) original audio sample
**Residual**: $e_1(n)$ as input to the RLS predictor
**Estimate**: $y_1(n)$ as input to the linear combiner
The DPCM predictor is the first predictor in the RLS-LMS predictor cascade. It is a simple first-order predictor with coefficient set to unity, i.e., the previous input sample is used as the estimate of the current input sample. This is illustrated as follows:

$$y_1(n) = x(n-1)$$

where $y_1(n)$ is the estimate by the DPCM predictor, and x(n – 1) is the previous input sample.

#### 11.6.5.2 RLS Predictor

**Input**: $e_1(n)$ residual of DPCM predictor
**Residual**: $e_2(n)$ as input to the LMS predictor
**Estimate**: $y_2(n)$ as input to the linear combiner
The RLS predictor is the second predictor in the RLS-LMS predictor cascade. The RLS algorithm is used to adapt the predictor weights. The algorithm is initialized by setting the M × M inverse auto-correlation matrix **P** to a predetermined value as follows:

$$\mathbf{P}(0) = \delta\mathbf{I}$$

where $\delta$ is a small positive number, **I** is an M × M identity matrix, and M is the RLS predictor order.
The weight vector of the RLS predictor, defined as

$$\mathbf{w}_{\mathrm{RLS}}(n) = [w_{\mathrm{RLS},1}(n), w_{\mathrm{RLS},2}(n), ..., w_{\mathrm{RLS,M}}(n)]^{\mathrm{T}}$$

is initialized by

$$\mathbf{w}_{\mathrm{RLS}}(0) = 0$$

For each time index n, n = 1, 2, … , the following calculations are made

$$\mathbf{v}(n) = \mathbf{P}(n-1)\mathbf{e}_1(n)$$

where $\mathbf{e}_1(n)$ is the RLS predictor input vector defined as

$$\mathbf{e}_1(n) = [e_1(n-1), e_1(n-2), ..., e_1(n-M)]^{\mathrm{T}}$$

and

$$m = \begin{cases} \dfrac{1}{\mathbf{e}_1^{\mathrm{T}}(n)\mathbf{v}(n)} & \text{if } \mathbf{e}_1^{\mathrm{T}}(n)\mathbf{v}(n) \neq 0 \\ 1 & \text{else} \end{cases}$$

$$\mathbf{k}(n) = m\,\mathbf{v}(n)$$

$$y_2(n) = \mathbf{w}_{\mathrm{RLS}}^{\mathrm{T}}(n-1)\mathbf{e}_1(n)$$

$$e_2(n) = e_1(n) - y_2(n)$$

$$\mathbf{w}_{\mathrm{RLS}}(n) = \mathbf{w}_{\mathrm{RLS}}(n-1) + \mathbf{k}(n)e_2(n)$$

$$\mathbf{P}(n) = \mathrm{Tri}\left\{\lambda^{-1}(\mathbf{P}(n-1) - \mathbf{k}(n)\mathbf{v}^{\mathrm{T}}(n))\right\}$$

where **k**(n) is the M × 1 gain vector, $\lambda$ is the forgetting factor that is a positive value slightly smaller than 1, T is the transpose symbol, and Tri{*} denotes the operation to compute the lower triangular part of **P**(n) and fill the upper triangular part of the matrix with the same values as in the lower triangular part. In other words, the value of **P**(n) at the i-th row and j-th column (i > j) is copied to the value of **P**(n) at the j-th row and i-th column.

© ISO/IEC 2008 - All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

#### 11.6.5.2.1 Joint-Stereo RLS Predictor

For a channel pair element (CPE), if joint_stereo in ALSSpecificConfig() is set to 1, the RLS predictor will work in the joint-stereo mode. In the joint-stereo mode, the RLS predictor employs both intra-channel prediction and inter-channel prediction. Figure 11.11 shows the joint-stereo RLS predictor for the left audio channel.

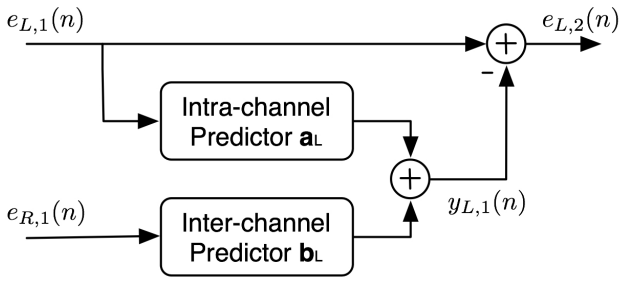


**Figure 11.11 — Joint-stereo RLS predictor for L channel**

As shown in the figure, the input signals to the predictor are L channel DPCM predictor residual $e_{L,1}(n)$ and R channel DPCM predictor residual $e_{R,1}(n)$. The joint-stereo predictor consists of an intra-channel predictor $\mathbf{a}_L$ and an inter-channel predictor $\mathbf{b}_L$. The intra-channel predictor $\mathbf{a}_L$ generates an estimate of the L channel current input sample $e_{L,1}(n)$ from past samples $e_{L,1}(n-1)$, $e_{L,1}(n-2)$, …, $e_{L,1}(n-M/2)$, where M is the order of the joint-stereo RLS predictor. At the same time, the inter-channel predictor $\mathbf{b}_L$ generates another estimate of $e_{L,1}(n)$ from R channel past input samples $e_{R,1}(n-1)$, $e_{R,1}(n-2)$, …, $e_{R,1}(n-M/2)$. These two estimates are added together. The result $y_{L,1}(n)$ is the estimate of the L channel joint-stereo RLS predictor. This process is represented as

$$y_{L,1}(n) = \sum_{m=1}^{M/2} a_{L,m} e_{L,1}(n-m) + \sum_{m=1}^{M/2} b_{L,m} e_{R,1}(n-m)$$

Where $a_{L,m}$ are coefficients of the intra-channel predictor $\mathbf{a}_L$, $b_{L,m}$ are coefficients of the inter-channel predictor $\mathbf{b}_L$. In the equation, the first summation term is the intra-channel predictor estimate, and the second summation term the inter-channel predictor estimate. The residual $e_{L,2(}n)$ is generated by subtracting estimate $y_{L,1}(n)$ from $e_{L,1}(n)$ as

$$e_{L,2}(n) = e_{L,1}(n) - y_{L,1}(n)$$

The L channel joint-stereo RLS predictor is updated by the RLS algorithm given in 11.6.5.2, where the weight vector, input vector, and residual in the RLS algorithm are redefined as follows:

$$\mathbf{w}_{\mathrm{RLS}}(n) = [b_{L,1}, a_{L,1}, b_{L,2}, a_{L,2}, ..., b_{L,M/2}, a_{L,M/2}]^{\mathrm{T}}$$

$$\mathbf{e}_1(n) = [e_{R,1}(n-1), e_{L,1}(n-1), e_{R,1}(n-2), e_{L,1}(n-2), ..., e_{R,1}(n-M/2), e_{L,1}(n-M/2)]^{\mathrm{T}}$$

$$e_2(n) = e_{L,2}(n)$$

The joint-stereo predictor for the right audio channel is shown in Figure 11.12.

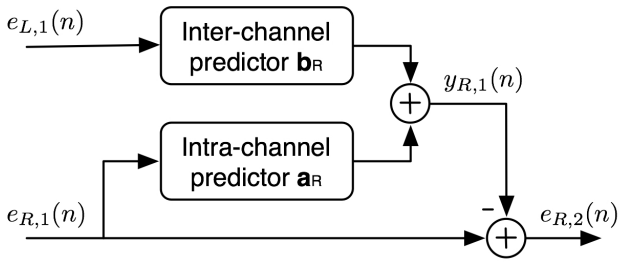


**Figure 11.12 — Joint-stereo RLS predictor for R channel**


© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

As shown in the figure, the input signals to the predictor are L channel DPCM predictor residual $e_{L,1}(n)$ and R channel DPCM predictor residual $e_{R,1}(n)$. The joint-stereo RLS predictor consists of an intra-channel predictor $\mathbf{a}_R$ and an inter-channel predictor $\mathbf{b}_R$. The intra-channel $\mathbf{a}_R$ predictor generates an estimate of the R channel current input sample $e_{R,1}(n)$ from past samples $e_{R,1}(n-1)$, $e_{R,1}(n-2)$, …, $e_{R,1}(n-M/2)$, where M is the order of the joint-stereo RLS predictor. At the same time, the inter-channel predictor $\mathbf{b}_R$ generates another estimate of $e_{R,1}(n)$ from L channel input samples $e_{L,1}(n)$, $e_{R,1}(n-1)$, …, $e_{R,1}(n-M/2+1)$. These two estimates are added together. The result $y_{R,1}(n)$ is the estimate of the R-channel joint-stereo RLS predictor. This process is represented as

$$y_{R,1}(n) = \sum_{m=1}^{M/2} a_{R,m} e_{R,1}(n-m) + \sum_{m=0}^{M/2-1} b_{R,m} e_{L,1}(n-m)$$

where $a_{R,m}$ are coefficients of the intra-channel predictor $\mathbf{a}_R$, $b_{R,m}$ are coefficients of the inter-channel predictor $\mathbf{b}_R$. In the equation, the first summation term is the intra-channel predictor estimate, and the second summation term the inter-channel predictor estimate. The residual $e_{R,2}(n)$ is generated by subtracting estimate $y_{R,1}(n)$ from $e_{R,1}(n)$, that is

$$e_{R,2}(n) = e_{R,1}(n) - y_{R,1}(n)$$

The R channel joint-stereo RLS predictor is updated by the RLS algorithm given in 11.6.5.2, where the weight vector, input vector, and residual in the RLS algorithm are redefined as follows:

$$\mathbf{w}_{\mathrm{RLS}}(n) = [b_{L,0}, a_{L,1}, b_{L,1}, a_{L,2}, ..., b_{L,M/2-1}, a_{L,M/2}]^{\mathrm{T}}$$
$$\mathbf{e}_1(n) = [e_{L,1}(n), e_{R,1}(n-1), e_{L,1}(n-1), e_{R,1}(n-2), ..., e_{L,1}(n-M/2+1), e_{R,1}(n-M/2)]^{\mathrm{T}}$$
$$e_2(n) = e_{R,2}(n)$$

The decoder processes a CPE in the order of LRLRLR.... The current L channel sample is always decoded before the current R channel sample.

#### 11.6.5.2.2 Mono RLS Predictor

For a single channel element (SCE), the RLS predictor operates in the mono mode. In the mono mode, the RLS predictor is updated by the RLS algorithm given in 11.6.5.2.
For a CPE, if joint_stereo in ALSSpecificConfig() is set to 0, mono RLS is used for each individual channel in the CPE. For a CPE, if mono_block in RLSLMS_extension() is set to 1, the CPE will be coded as two individual channels L and L-R. The L channel is treated as a SCE, whereas the difference channel L-R goes directly to the entropy encoder. For a SCE, if an input frame contains only constant values, the RLS-LMS predictor is bypassed for that frame. For a CPE, if both channels contain only constant values, the RLS-LMS predictor is bypassed for the frame.

#### 11.6.5.2.3 Filtering Operation in RLS predictor

The following pseudo code illustrates how an order-M RLS predictor generates the estimate signal. The order of the RLS predictor is defined in 11.6.5.6.1.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2008 - All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

| Pseudo code | Comments |
|---|---|
| INT32 *w, *buf;<br>INT64 temp = 0;<br><br>for (i=0; i<M; i++)<br>  temp += (INT64) w[i] * buf[i]; | w is RLS predictor weight vector (.16 format)<br>buf is RLS predictor input vector (.0 format)<br><br><br>temp = w * buf |
| temp >= 12;<br><br>if (temp>0x40000000) temp = 0x40000000;<br>if (temp<-0x40000000) temp =-0x40000000; | temp is .4 format<br><br>Limit the range of temp to<br>[-0x40000000, 0x40000000] |
| INT32 y;<br><br>y = (INT32) (temp); | y is RLS predictor estimate (.4 format) |

#### 11.6.5.2.4 Weight Adaptation in RLS predictor

The following pseudo code illustrates how the weight vector of an order-M RLS predictor is updated.

| Pseudo code | Comments |
|---|---|
| INT32 x, y, e;<br><br>e = x – y; | x is RLS predictor current input sample<br>x is .4 format<br>y is RLS predictor esitimate (.4 format)<br>e is RLS predictor residual (.4 format) |
| INT64 **P;<br>INT32 *buf, *v;<br>INT16 vs;<br><br>[v, vs] = MulMtxVec(P, buf); | P is RLS algorithm matrix P (.60 format)<br>buf is RLS predictor input vector (.0 format)<br>v is RLS algorithm vector v<br>vs is a scale factor<br><br>v = P*buf<br>v is .(28-vs) format |
| INT64 temp;<br>INT16 ds;<br><br>[temp, ds] = MulVecVec(buf, v); | <br><br>ds is a scale factor<br><br>temp = buf * v<br>temp is .(60-vs-ds) format |
| i = 0;<br>while(temp>0x20000000 && temp!=0)<br>  temp >>= 1; i++;<br>i += vs + ds; | <br><br><br><br>temp is .(60-i) format |
| if (i<=60)<br>  temp += (((INT64) 1) << (60-i));<br>else<br>  reinit_P(P); | If (i<=60)<br>  temp = temp + 1<br>else<br>  Re-initialize matrix P |


© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

| | |
|---|---|
| INT64 mm;<br>INT32 m;<br><br>if (temp==0)<br>  mm = 1L<<30;<br>else if (i<=28)<br>  shift = 28-i;<br>  mm = (((INT64)1)<<62) / temp;<br>  if (shift>32)<br>  mm = 1L<<30;<br>  else<br>  mm <<= shift;<br>else<br>  mm = (((INT64)1)<<(90-i)) / temp;<br><br>m = (INT32)mm; | m is the RLS algorithm variable m<br><br>If (temp==0)<br>  m = 1;<br>else<br>  m = 1/temp;<br><br><br><br><br><br><br>m is .30 format |
| INT32 *k;<br><br>for (i=0; i<M; i++)<br>  temp = (INT64) v[i] * m;<br>  if (vs>=12)<br>  k[i] = temp<<(vs-12);<br>  else<br>  k[i] = temp>>(11-vs);<br>  k[i] = ROUND2(k[i]); | k is the RLS algorithm gain vector k<br><br><br>k = m * v<br><br>Scale k<br><br><br>k is .46 format |
| temp = 0;<br>for (i=0; i<M; i++)<br>  temp \|= (k[i]>0 ? k[i]:-k[i]);<br>ds = fast_bitcount(temp);<br>if (ds>30)<br>  ds -= 30;<br>  for (i=0; i<M; i++) k[i] >>= ds;<br>else<br>  ds = 0; | <br><br><br>Obtain MSB in k<br>Get position of MSB bit<br><br>Scale k<br><br><br>k is .(46-ds) format |
| INT32 *w;<br><br>for (i=0; i<M; i++)<br>  temp = (((INT64) k[i] * (e>>3))>>(30-ds));<br>  temp = w[i] + ROUND2(temp);<br>  w[i] = (long) temp;<br>vs += ds; | w is RLS predictor weight vector<br><br>Update RLS weight vector<br>w = w + k * e<br>w is .16 format |

© ISO/IEC 2008 - All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

| | |
|---|---|
| INT16 lambda;<br><br>for (i=0; i<M; i++)<br>  for (j=0; j<=i; j++)<br>  temp = ((INT64) k[i] * v[j])>>(14-vs);<br>  P[i][j] -= temp;<br>  if (P[i][j]>=0x4000000000000000<br>  || P[i][j]<=-0x4000000000000000)<br>  reinit_P(P); break;<br>  temp = P[i][j] / lambda;<br>  P[i][j] += temp; | lambda is defined in 11.6.5.6.6<br><br>Update matrix P (lower triangular)<br><br>P = P – k * v<br>P = P * (1+1/lambda)<br>P is .60 format |
| for (i=1; i<M; i++)<br>  for (j=0; j<i; j++)<br>  P[j][i] = P[i][j]; | Update matrix P (upper triangular) |
| for (i=M-1; i>0; i--)<br>  buf[i] = buf[i-1];<br>buf[0] = x>>4; | Update RLS predictor input vector<br><br>buf is .0 format |

| Pseudo code | Comments |
|---|---|
| [INT32 *v, INT16 vs] = MulMtxVec(INT64 **P, INT32 *buf) | Multiply matrix P by buf,<br>Return the result in v<br>P is .60 format, buf is .0 format |
| INT64 temp=0;<br>INT16 ps;<br><br>for(i=0; i<M; i++)<br>  for(j=0; j<=i; j++)<br>  temp \|= (P[i][j]>0 ? P[i][j] : -P[i][j]);<br><br>ps = 63 - fast_bitcount(temp); | ps is a scale factor<br><br>Obtain MSB in matrix P<br><br>Calculate the shift needed<br>to maximize P |
| INT64 *u;<br><br>for (i=0; i<M; i++)<br>  u[i]=0;<br>  for (j=0; j<M; j++) u[i] +=<br>(INT64) (((P[i][j]<<ps)+(INT64)0x80000000)>>32) * buf[j]; | u = P * buf<br>u is .(28+ps) format. |
| INT16 ns;<br><br>temp =0;<br>for (i=0; i<M; i++)<br>  temp \|= (u[i]>0 ? u[i] : -u[i]);<br><br>ns = fast_bitcount(temp); | ns is a scale factor<br><br>Obtain MSB<br><br>Get the position of MSB bit |


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2008 – All rights reserved
Not for Resale

| | |
|---|---|
| if (ns>28)<br>  ns -= 28;<br>  for(i=0; i<M; i++)<br>  v[i] = (INT32) (u[i]>>ns);<br>  vs = ns - ps;<br>else<br>  for(i=0;i<M;i++)<br>  v[i] = (INT32) u[i];<br>  vs = -ps; | Scale v<br><br><br><br><br><br>v is .(28-vs) format |

| Pseudo code | Comments |
|---|---|
| [INT64 z, INT16 ds] = MulVecVec(INT32 *buf, INT32 *v) | Compute inner product of buf and v,<br>return the result in z<br>buf is .0 format, v is .(28-vs) format |
| INT 64 z = 0;<br><br>for (i=0; i<M; i++)<br>  z += (INT64) buf[i] * v[i]; | <br><br><br>z = buf * v |
| INT64 temp;<br><br>temp = (z>0 ? z : -z);<br>ds = fast_bitcount(temp); | <br><br><br><br>Get the position of MSB bit |
| if (ds>28)<br>  ds -= 28;<br>  z = (z<<(32-(ds-1)));<br>  z = ROUND2(z);<br>else<br>  ds = 0;<br>  z = (z<<32); | Scale z<br><br><br><br><br><br>z is .(60-vs-ds) format |

| Pseudo code | Comments |
|---|---|
| reinit_P(INT64 **P) | Re-initialize matrix P |
| for (i=0; i<M; i++)<br>  for (j=0; j<M; j++)<br>  P[i][j]= 0;<br><br>for (i=0; i<M; i++)<br>  P[i][i] = (INT64) JS_INIT_P; | <br><br><br>Clear P to zero<br><br><br>Initialize the diagonal components |

© ISO/IEC 2008 - All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

| Pseudo code | Comments |
|---|---|
| [INT16 count] = fast_bitcount (INT64 temp) | Return the MSB bit position |
| i = 56; j = 0;<br>while((temp>>i)==0 && i>0) i -= 8;<br>temp>>=i;<br>while(temp>0) temp>>=1; j++;<br>count = i + j; | |

The RLS predictor can be turned off by setting RLS_order to zero. In this case, the RLS predictor estimate $y_2(n)$ is set to zero.

#### 11.6.5.3 LMS Predictors

**Input**: $e_{k-1}(n)$ residual of the previous predictor (could be RLS predictor or LMS predictor)
**Residual**: $e_k(n)$ as input to the next LMS predictor
**Estimate**: $y_k(n)$ as input to the linear combiner
The RLS-LMS predictor contains a series of LMS predictors. The Normalized LMS (NLMS) algorithm is used to adapt the predictor weights. For an order-M LMS predictor, its weight vector

$$\mathbf{w}_{\mathrm{LMS}}(n) = [w_{\mathrm{LMS},1}(n), w_{\mathrm{LMS},2}(n), ..., w_{\mathrm{LMS,M}}(n)]^{\mathrm{T}}$$

is initialized by

$$\mathbf{w}_{\mathrm{LMS}}(n) = 0$$

For each time index n, n = 1, 2, … , the estimate is calculated as

$$y_k(n) = \mathbf{w}_{\mathrm{LMS}}^{\mathrm{T}}(n)\mathbf{e}_{k-1}(n)$$

where $\mathbf{e}_{k-1}(n)$ is the input vector defined as

$$\mathbf{e}_{k-1}(n) = [e_{k-1}(n-1), e_{k-1}(n-2), ..., e_{k-1}(n-M)]^{\mathrm{T}}$$

The LMS predictor weight vector is updated according to

$$e_k(n) = e_{k-1}(n) - y_k(n)$$

$$\mathbf{w}_{\mathrm{LMS}}(n) = \mathbf{w}_{\mathrm{LMS}}(n-1) + \frac{e_k(n)\mathbf{e}_{k-1}(n)}{2^7 + \mu_k \mathbf{e}_{k-1}^T(n)\mathbf{e}_{k-1}(n)}$$

where $\mu_k$ is the NLMS stepsize (Clause 0).

##### 11.6.5.3.1 Filtering operation in LMS Predictor

The following pseudo code illustrates how an order-M LMS predictor generates the estimate signal. The order of the LMS predictor is defined in Clause 0.


© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

| Pseudo code | Comments |
|---|---|
| INT32 *w, *buf;<br>INT64 temp = 0;<br><br>for (i=0; i<M; i++)<br>  temp += (INT64) w[i] * buf[i]; | w is LMS predictor weight vector (.24 format)<br>buf is LMS predictor input vector (.0 format)<br><br>temp = w * buf |
| temp >>= 20;<br><br>if (temp>0x7fffff) temp = 0x7fffff;<br>if (temp<-0x7fffff) temp = -0x7fffff; | temp is .4 format<br><br>Limit the range of temp to<br>[-0x7fffff, 0x7fffff] |
| INT32 y;<br><br>y = (INT32) (temp); | y is LMS predictor estimate (.4 format) |

#### 11.6.5.3.2 Weight adaptation in LMS Predictor

The following pseudo code illustrates how the weight vector of an order-M LMS predictor is updated.

| Pseudo code | Comments |
|---|---|
| INT32 x, y, e;<br><br>e = x – y; | x is LMS predictor current input sample (.4 format)<br>y is LMS predictor esitimate (.4 format)<br>e is LMS predictior residual (.4 format) |
| INT32 *buf;<br>INT64 pow = 0;<br><br>for (i=0; i<M; i++)<br>  pow += (INT64) buf[i] * buf[i];<br><br>if (pow>0x4000000000000000)<br>  pow = 0x4000000000000000; | buf is LMS predictor input vector (.0 format)<br><br>Compute total signal power in buf<br>pow = buf * buf<br><br>pow is .0 format |
| INT16 mu;<br>INT64 temp, temp1;<br><br>temp = (INT64) mu * (pow>>7);<br>temp1 = temp; | mu is NLMS algorithm stepsize (.0 format)<br>mu is defined in xxx.yyy<br><br>temp = mu * pow<br>temp is .(-7) format |
| i = 0;<br>while(temp>0x7fffffff) temp>>=1; i++;<br><br>temp = ((INT64) e<<(29-i)) / (INT64)((temp1+1)>>i); | Compute the number of shifts needed<br>to shift temp into the lower 32 bit<br><br>temp = e / (128 + mu*buf*buf)<br>temp is .40 format |

© ISO/IEC 2008 - All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

| | |
|---|---|
| INT32 *w;<br><br>for (i=0; i<M;i++) w[i] +=<br>(INT32) (((INT64) buf[i] * temp + 0x8000)>>16); | w is LMS predictor weight vector (.24 format)<br><br>Update weight vector<br>w = w + (buf*e) / (128 + mu*buf*buf) |
| for (i=M-1; i>0; i--)<br>buf[i] = buf[i-1];<br>buf[0] = x>>4; | Update the input vector<br><br>buf is .0 format |

#### 11.6.5.4 Linear Combiner

**Input**: $y_1(n)$, $y_2(n)$, …,$y_K(n)$ estimates from DPCM, RLS, and LMS predictors
**Output**: $\hat{x}(n)$ final estimate of the RLS-LMS predictor

The linear combiner multiplies the estimates from the DPCM, RLS, and LMS predictors by a set of weights. The results are summed together to provide the final estimate of the RLS-LMS predictor. The Sign-Sign LMS algorithm is used to update the linear combiner weights. If there are in total K predictors in the RLS-LMS predictor cascade, the linear combiner weight vector is given by

$$\mathbf{c}(n) = [c_1(n), c_2(n), ..., c_K(n)]^{\mathrm{T}}$$

The linear combiner input vector is given by

$$\mathbf{y}(n) = [y_1(n), y_2(n), ..., y_K(n)]^{\mathrm{T}}$$

The final estimate of the RLS-LMS predictor is given by

$$\hat{x}(n) = \mathbf{c}^{\mathrm{T}}(n)\mathbf{y}(n)$$

The linear combiner weight vector is updated by the Sign-Sign LMS algorithm

$$\mathbf{c}(n) = \mathbf{c}(n-1) + \alpha \operatorname{sgn}\{\mathbf{y}(n)\} \operatorname{sgn}\{x(n) - \hat{x}(n)\}$$

where x(n) is the RLS-LMS predictor current input sample, $\alpha$ is the Sign-Sign LMS stepsize (11.6.5.6.5). The sgn{*} function is defined as

$$\operatorname{sgn}\{r\} = \begin{cases} 1 & r > 0 \\ 0 & r = 0 \\ -1 & r < 0 \end{cases}$$

The following pseudo code illustrates how an order-K linear combiner generates the final estimate of the RLS-LMS predictor. The order of linear combiner is given by (LMS_stage+2). The code also shows how the linear combiner weight vector is updated. It shall be noted that the first two linear combiner weights are not updated.


© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

| Pseudo code | Comments |
|---|---|
| INT32 *c, *y;<br>INT64 xhat = 0;<br><br>for (i=0; i<K; i++)<br>  xhat += (INT64) c[i] * y[i];<br>xhat >>= 24 ; | c is linear combiner weight vector (.24 format)<br>y is linear combiner input vector (.4 format)<br>xhat is linear combiner output, i.e., final estimate of RLS-LMS predictor.<br><br>xhat is .4 format |
| INT32 x;<br>INT64 r;<br><br>r = (x<<4) – xhat; | x is RLS-LMS predictor current input sample<br>x is .0 format<br><br>Compute the difference between x and xhat<br>r is .4 format |
| INT32 LMS_stepsize;<br>INT64 temp;<br><br>for (i=2; i<K; i++)<br>  temp = (INT64) r * y[i];<br>  if (temp>0 && c[i]<0x40000000)<br>  c[i] += LMS_stepsize;<br>  if (temp<0 && c[i]>-0x40000000)<br>  c[i] -= LMS_stepsize; | Sign-Sign LMS algorithm stepsize (.24 format)<br>LMS_stepsize is defined in 11.6.5.6.5<br><br>Sign-Sign LMS algorithm<br><br>Only update linear combiner weights for the LMS predictors |
| INT32 e;<br><br>e = x - ROUND1(xhat) | e is RLS-LMS predictor residual (.0 format) |

#### 11.6.5.5 RLS-LMS Predictor Initialization

The RLS-LMS predictor is initialized at the following moments: start of encoding, start of decoding, start of each random access (RA) frame, and whenever filter order changes. The RLS-LMS predictor is initialized by zero-filling the following buffers: DPCM predictor previous input sample, RLS predictor input vector and weight vector, all LMS predictor input vectors and weight vectors, and linear combiner input vector. The RLS predictor P matrix is initialized by calling function reinit_P(P). The linear combiner weight vector is set to constant FRACTION, which represents 1.0 in 8.24 format.
Table 11.27 lists the constants and macros used by the RLS-LMS predictor.

**Table 11.27 — Constants and Macros**

| Constants & Macros | Value | Comments |
|---|---|---|
| JS_INIT_P | 115292150460684 | 0.0001 in 4.60 format |
| FRACTION | (1L<<24) | 1.0 in 8.24 format |
| ROUND1(x) | ((INT32) ((x+8)>>4)) | Rounding function |
| ROUND2(x) | ((INT64) ( (INT64) x + (INT64) 1 )>>1) | Rounding function |

© ISO/IEC 2008 - All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

#### 11.6.5.6 RLS-LMS Predictor Parameters

The parameters of the RLS-LMS predictor can be changed every frame. This is signalled in RLSLMS_extension() when ext_mode = 1. Sub-sections 11.6.5.6.1 to 11.6.5.6.6 describe the RLS-LMS predictor parameters that can be changed.

##### 11.6.5.6.1 RLS_order

This parameter specifies the order of the RLS predictor. Valid values and the corresponding 4-bit indices are listed in Table 11.28.

**Table 11.28 — RLS_order**

| index | RLS_order | index | RLS_order |
|---|---|---|---|
| 0 | 0 | 8 | 16 |
| 1 | 2 | 9 | 18 |
| 2 | 4 | 10 | 20 |
| 3 | 6 | 11 | 22 |
| 4 | 8 | 12 | 24 |
| 5 | 10 | 13 | 26 |
| 6 | 12 | 14 | 28 |
| 7 | 14 | 15 | 30 |

##### 11.6.5.6.2 LMS_stage

This parameter specifies the number of LMS predictors in the RLS-LMS predictor cascade. Valid values and the corresponding 3-bit indices are listed in Table 11.29.

**Table 11.29 — LMS_stage**

| index | LMS_stage |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |
| 3 | 4 |
| 4 | 5 |
| 5 | 6 |
| 6 | 7 |
| 7 | 8 |


© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

#### 11.6.5.6.3 LMS_order

This parameter specifies the order of the LMS predictor. Valid values and the corresponding 5-bit indices are listed in Table 11.30.

**Table 11.30 — LMS_order**

| index | LMS_order | index | LMS_order |
|---|---|---|---|
| 0 | 2 | 16 | 32 |
| 1 | 3 | 17 | 36 |
| 2 | 4 | 18 | 48 |
| 3 | 5 | 19 | 64 |
| 4 | 6 | 20 | 80 |
| 5 | 7 | 21 | 96 |
| 6 | 8 | 22 | 128 |
| 7 | 9 | 23 | 256 |
| 8 | 10 | 24 | 384 |
| 9 | 12 | 25 | 448 |
| 10 | 14 | 26 | 512 |
| 11 | 16 | 27 | 640 |
| 12 | 18 | 28 | 768 |
| 13 | 20 | 29 | 896 |
| 14 | 24 | 30 | 1024 |
| 15 | 28 | 31 | reserved |

#### 11.6.5.6.4 LMS_mu

This parameter specifies the stepsize of NLMS algorithm that is used to update the LMS predictor. Valid values and the corresponding 5-bit indices are listed in Table 11.31.

**Table 11.31 — LMS_mu**

| index | LMS_mu | index | LMS_mu |
|---|---|---|---|
| 0 | 1 | 16 | 18 |
| 1 | 2 | 17 | 20 |
| 2 | 3 | 18 | 22 |
| 3 | 4 | 19 | 24 |
| 4 | 5 | 20 | 26 |
| 5 | 6 | 21 | 28 |
| 6 | 7 | 22 | 30 |
| 7 | 8 | 23 | 35 |
| 8 | 9 | 24 | 40 |
| 9 | 10 | 25 | 45 |
| 10 | 11 | 26 | 50 |
| 11 | 12 | 27 | 55 |

© ISO/IEC 2008 - All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

| | | | |
|---|---|---|---|
| 12 | 13 | 28 | 60 |
| 13 | 14 | 29 | 70 |
| 14 | 15 | 30 | 80 |
| 15 | 16 | 31 | 100 |

#### 11.6.5.6.5 LMS_stepsize

This parameter specifies the stepsize of Sign-Sign LMS algorithm that is used to update the linear combiner. Valid values (7.24 format) and the corresponding 3-bit indices are listed in Table 11.32.

**Table 11.32 — LMS_stepsize**

| index | LMS_stepsize | Comments |
|---|---|---|
| 0 | 1677 | 0.0001 (.24 format) |
| 1 | 3355 | 0.0002 |
| 2 | 8388 | 0.0005 |
| 3 | 16777 | 0.001 |
| 4 | 33554 | 0.002 |
| 5 | 83886 | 0.005 |
| 6 | 167772 | 0.01 |
| 7 | 335544 | 0.02 |

#### 11.6.5.6.6 RLS_lambda and RLS_lambda_ra

RLS_lambda and RLS_lambda_ra are parameters that control the forgetting factor of the RLS predictor. RLS_lambda_ra is used for RA frame only, whereas RLS_lambda is used for both RA and non-RA frames. RLS_lambda and RLS_lambda_ra take integer values in the range of [1, 1023]. The RLS forgetting factor $\lambda$ is calculated as

$$\lambda = \frac{\text{RLS\_lambda}}{\text{RLS\_lambda+1}}$$

and

$$\lambda = \frac{\text{RLS\_lambda\_ra}}{\text{RLS\_lambda\_ra+1}}$$

#### 11.6.5.7 Random Access

The RLS-LMS predictor is re-initialized at the start of every random access frame. In an RA frame, the RLS-LMS predictor uses RLS_lambda_ra to determine the RLS forgetting factor for the first 300 samples. For the rest of the samples in the frame, RLS_lambda is used. In an RA frame, the LMS predictors in the RLS-LMS predictor cascade start to update their states only after the first N/32 samples in the frame, where N is the frame length.

*and increase the subsequent figure and table indices accordingly.*

*Page 72, replace 11.6.8 entirely, with the following text:*

--`,,```,,,,````-`-`,,`,,`,`,,`---


© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

### 11.6.8 Multi-Channel Coding (MCC)

Multi-channel coding uses adaptively weighted inter-channel subtraction in order to further reduce the amplitudes of the prediction residuals.

#### 11.6.8.1 Decoding and reconstruction of single channels

Sets of inter-channel dependency information, including master_channel_index, weighting_factor and time_difference_flag, are decoded for each channel. The numbers of iterations is controlled by stop_flag.
In addition, there is a js_switch bit which activates simple Joint Stereo coding of input signals (see 11.6.7) instead of multi-channel coding of LPC prediction residual signals, even when inter-channel correlation mode is used in general.
If time_difference_flag is zero, 3-tap in-phase weighted addition is carried out as shown in the following pseudo-code.

```
INT64 u;
for(smpl=1 ;smpl<N-1; smpl++)
{
    u = (INT64)MCCgain[0]*d_masterchannel[smpl-1]
      +(INT64)MCCgain[1]*d_masterchannel[smpl]
      +(INT64)MCCgain[2]*d_masterchannel[smpl+1];

    d[smpl] += (long) ((u+64)/128);
}
```

---

d*:* integer for residual signal

MCCgain: gain value $\gamma(i)$*128

If time_difference_flag is one, 6-tap weighted addition is carried out with the decoded time difference parameter TDL as shown in the following pseudo-code.

```
INT64 u;
if(TDL>0) {ss=1; se=N-TDL-1;}
else {ss=-TDL+1; se=N-1;}
for(smpl=ss ;smpl<se; smpl++)
{
    u = (INT64)MCCgain[0]*d_masterchannel[smpl-1]
      + (INT64)MCCgain[1]*d_masterchannel[smpl]
      + (INT64)MCCgain[2]*d_masterchannel[smpl+1]
      + (INT64)MCCgain[3]*d_masterchannel[smpl+TDL-1]
      + (INT64)MCCgain[4]*d_masterchannel[smpl+TDL]
      + (INT64)MCCgain[5]*d_masterchannel[smpl+TDL+1];

    d[smpl] += (long) ((u+64)/128);
}
```

---

d*:* integer for residual signal

MCCgain: gain value $\gamma(i)$*128

TDL: time difference lag value: $\tau$

The range of TDL depends on the sampling frequency as shown in Table 11.28. There is a sign bit for the TDL to support both positive and negative differences relative to the coding signal. To avoid conflicts with the in-phase weighting, TDL has to be larger than 3 or less than -3.

**Table 11.39 — Range of lag $\tau$**

| search range of $\tau(i)$ | start | end |
|---|---|---|
| Freq < 96 kHz | 3 | 31+3 |
| Freq >= 96 kHz | 3 | 63+3 |
| Freq >= 192 kHz | 3 | 127+3 |

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2008 - All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The gain value $\gamma(i)$*128 is quantized according to Table 11.40. The actual indexes of the gain values are coded using Rice codes with a parameter of 1 and an offset of 16, except for weighting_factor[1]. The index of weighting_factor[1] is decoded using Rice codes with a parameter of 2 with an offset of 14.

**Table 11.40 — Rice code of Inter-channel weighting factors $\gamma$ *(i)*128***

| index of weighting factor | for weighting factor [1] | for other factors | $\gamma$ (i) *128 | index of weighting factor | for weighting factor [1] | for other factors | $\gamma$ (i) *128 |
|---|---|---|---|---|---|---|---|
| 0 | 111111001 | 1111111111111100 | 204 | 16 | 1010 | 01 | 0 |
| 1 | 11111100 0 | 111111111111100 | 192 | 17 | 1011 | 101 | -12 |
| 2 | 11111001 | 111111111111100 | 179 | 18 | 11010 | 1101 | -25 |
| 3 | 11111000 | 11111111111100 | 166 | 19 | 11011 | 11101 | -38 |
| 4 | 1111001 | 1111111111100 | 153 | 20 | 111010 | 111101 | -51 |
| 5 | 1111000 | 111111111100 | 140 | 21 | 111011 | 1111101 | -64 |
| 6 | 111001 | 11111111100 | 128 | 22 | 1111010 | 11111101 | -76 |
| 7 | 111000 | 1111111100 | 115 | 23 | 1111011 | 111111101 | -89 |
| 8 | 11001 | 111111100 | 102 | 24 | 11111010 | 1111111101 | -102 |
| 9 | 11000 | 11111100 | 89 | 25 | 11111011 | 11111111101 | -115 |
| 10 | 1001 | 1111100 | 76 | 26 | 111111010 | 111111111101 | -128 |
| 11 | 1000 | 111100 | 64 | 27 | 111111011 | 1111111111101 | -140 |
| 12 | 001 | 11100 | 51 | 28 | 1111111010 | 11111111111101 | -153 |
| 13 | 000 | 1100 | 38 | 29 | 1111111011 | 111111111111101 | -166 |
| 14 | 010 | 100 | 25 | 30 | 11111111010 | 1111111111111101 | -179 |
| 15 | 011 | 00 | 12 | 31 | 11111111011 | 1111111111111110 1 | -192 |

### 11.6.8.2 Sequential decoding for multi-channel signals

A sequence of prediction residual samples in one channel and one block is treated as a vector, $\mathbf{e} = (e_o, e_1, \cdots, e_{n-1})^{\mathrm{T}}$, where *n* represents the number of samples in a frame. Let $\mathbf{e}^j$ be the prediction residual vector of the master channel, where *j* is specified by master_channel_index, and $\mathbf{e}^i$ is the prediction residual vector of the coded (slave) channel. The residual signal of the slave channel is reconstructed by adding the master channel signal with multiplication factors of $\gamma$ (weighting factor) ranging from -192/128 to 204/128.

$$\mathbf{e}^i = \tilde{\mathbf{e}}^i + \sum_{m=-1}^{1} \gamma_{m,k}^{i,j} \mathbf{e}^j$$

If stop_flag is on, no inter-channel information follows for this channel. If stop_flag is off, inter-channel information follows and this is allowed repeatedly. Thus the syntax is allowed to have multiple stages of inter-channel information. Note that the number of layers should be common to all channels. If the master_channel_index is identical to the channel number in all stages deeper than a certain stage, which means no more weighted subtraction is activated, the syntax elements for further stages are skipped for this channel. As a result, stop_flag is set at the end of the effective set of information.
For reconstructing residual signals from multiple stage inter-channel information, the process should start from the right most set (the deepest stage) of information. The intermediate residual vector at *i*-th channel and *(k-1)*-th layer, $\mathbf{e}_k{}^i$, is reconstructed as follows using the master channel vector $\mathbf{e}_{k-1}{}^j$ and $\gamma_{k-1}^{i,j}$ of the *(k-1)*-th layer.


© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$\mathbf{e}_{k-1}{}^{i} = \mathbf{e}_{k}{}^{i} + \sum_{m} \gamma_{m,k-1}^{i,j} \cdot \mathbf{e}_{k-1}{}^{j}$, where $\sum_{m} \gamma_{m,k-1}^{i,j}$ means either 3-tap or 6-tap filtering.

An example of bit stream for 5 channels and three iteration stages is shown in Table 11.41. Channel 0 has three syntax elements corresponding to the stages. Reconstruction should be carried out from the last stage, i.e. the third stage. The third stage has a stop_flag and operation moves to the second stage.

At the second stage, channel 0 has channel 4 as a master channel with weighting factors of $\gamma[14]$, $\gamma[12]$, and $\gamma[15]$. Thus, $\mathbf{e}_{2}{}^{0} = \mathbf{e}_{3}{}^{0} + \sum \gamma \cdot \mathbf{e}_{2}{}^{1}$. Other channels need no operations at the second iteration stage. Thus, $\mathbf{e}_{2}{}^{1} = \mathbf{e}_{3}{}^{1}$, $\mathbf{e}_{2}{}^{2} = \mathbf{e}_{3}{}^{2}$, $\mathbf{e}_{2}{}^{3} = \mathbf{e}_{3}{}^{3}$, $\mathbf{e}_{2}{}^{4} = \mathbf{e}_{3}{}^{4}$.

At the first stage, channel 0 has channel 2 as a master channel with weighting factors of $\gamma[13]$, $\gamma[10]$, and $\gamma[15]$. This means $\mathbf{e}_{0}{}^{0} = \mathbf{e}_{1}{}^{0} + \sum \gamma \cdot \mathbf{e}_{0}{}^{2}$. However, $\mathbf{e}_{0}{}^{2}$ is not ready. This process needs to be suspended until $\mathbf{e}_{0}{}^{2}$ is reconstructed. Channel 1 has channel 4 as a master channel with multiplication factors of $\gamma[13]$, $\gamma[11]$, and $\gamma[12]$. Thus, $\mathbf{e}_{0}{}^{1} = \mathbf{e}_{1}{}^{1} + \sum \gamma \cdot \mathbf{e}_{0}{}^{4}$. Channel 2 has channel 1 as a master channel with the time difference value "5" and multiplication factors of $\gamma[16]$, $\gamma[15]$, $\gamma[16]$, $\gamma[13]$, $\gamma[12]$, and $\gamma[14]$. Thus, $\mathbf{e}_{0}{}^{2} = \mathbf{e}_{1}{}^{2} + \sum \gamma \cdot \mathbf{e}_{0}{}^{1}$. Now that channel 2 has been reconstructed, channel 0 can be reconstructed. Other channels have no operation at the first stage. Thus, $\mathbf{e}_{0}{}^{3} = \mathbf{e}_{1}{}^{3}$, $\mathbf{e}_{0}{}^{4} = \mathbf{e}_{1}{}^{4}$.

**Table 11.41 — Example of a bit stream for inter-channel information for 5 channels and 3 stages**

| stage 1 | | | | | |
|---|---|---|---|---|---|
| channel no. | stop_flag (S) | master_channel_index (M) | time_difference_flag (T) | weighting factor (W) | time_difference_value(L) |
| 0 | 0 | 2 | 0 | 13-10-15 | |
| 1 | 0 | 4 | 0 | 13-11-12 | |
| 2 | 0 | 1 | 1 | 16-15-16-13-12-14 | 5 |
| 3 | 1 | | | | |
| 4 | 1 | | | | |
| | | | | | |
| stage 2 | | | | | |
| channel no. | S | M | T | W | L |
| 0 | 0 | 4 | 0 | 14-12-15 | |
| 1 | 1 | | | | |
| 2 | 1 | | | | |
| 3 | | | | | |
| 4 | | | | | |
| | | | | | |

© ISO/IEC 2008 - All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

| stage 3 | | | | | |
|---|---|---|---|---|---|
| channel no. | S | M | T | W | L |
| 0 | 1 | | | | |
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |

We can reconstruct a vector only when the master channel vector is already reconstructed at the stage. Figure 11.8 shows the iterative process that ensures perfect reconstruction of all channel vectors.

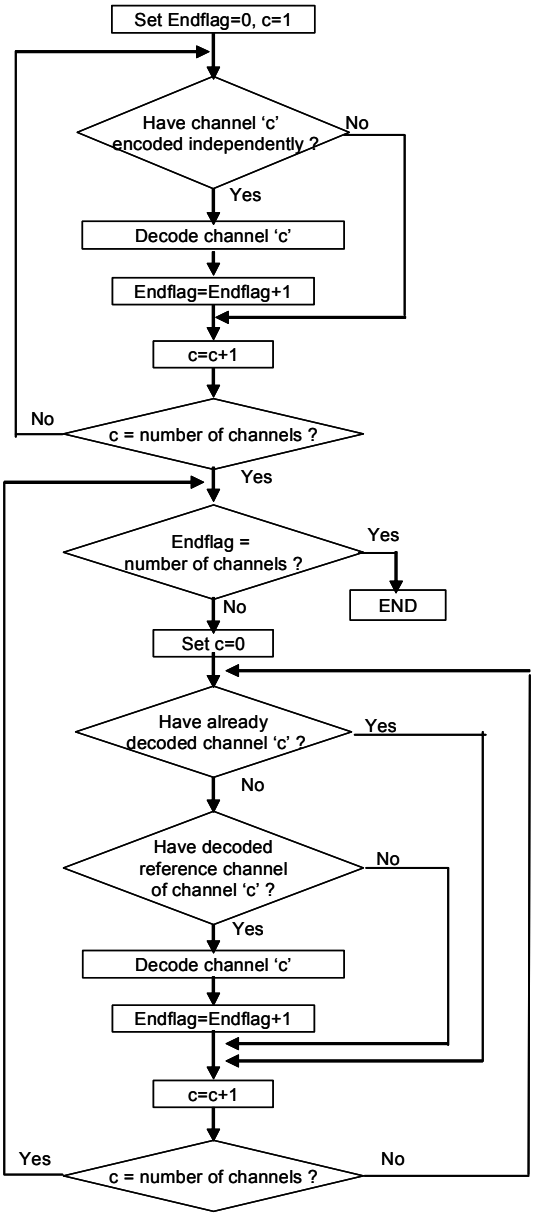


**Figure 11.8 – Process that ensures perfect reconstruction of all channel vectors**


© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

*Page 77,, replace Figure 11.10 with the following figure:*

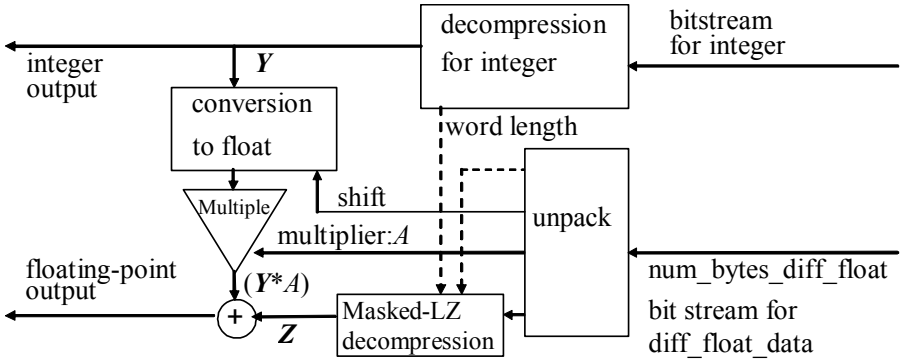


**Figure 11.10 — Integrated lossless decoder for integer and floating-point bitstream**

Copyright International Organization for Standardization - All rights reserved
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale