
**Information technology — MPEG video
technologies —**

**Part 4:
Video tool library**

**AMENDMENT 1: Graphics tool library (GTL)
for the reconfigurable multimedia coding
(RMC) framework**

Technologies de l'information — Technologies vidéo MPEG —

Partie 4: Bibliothèque d'outils vidéo

*AMENDEMENT 1: Bibliothèque d'outils graphiques (GTL) pour le cadre
de codage multimédia reconfigurable (RMC)*



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2014

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: [Foreword - Supplementary Information](#)

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23002-4:2014/Amd 1:2014

Information technology — MPEG video technologies —

Part 4: Video Tool Library

AMENDMENT 1: Graphics tool library (GTL) for the reconfigurable multimedia coding (RMC) framework

In 4.1 "FU Interfaces", before Table 1, add the following text:

- In several FU diagrams, the ports are named with a trailing "_i" for the input port type and with a trailing "_o" for the output port type.
- Some FU diagrams contains as well the Finite State Machine diagram. The following conventions apply: INPUT - the action of reading a token or a set of tokens from the input port, OUTPUT - the action of writing the token or a set of tokens to an output port.
- "Parameter" is set at network configuration stage (cannot be changed during the process) and it is characteristic for each FU
- Token RANGE: describes the mathematical interval for the token value

Examples:

Token RANGE: { 0, 1} – binary value

Token RANGE: [0 .. N], $value \in [0, N]$ real values, closed interval

- All the FUs require the data to be in little-endian format.

In 4.2 "FU IDs", complete Table 2 with the following lines:

Note: update the FU table..

ID	FU Name
107	Algo_Parser_SC3DMC
108	Algo_InverseQuantization1D
109	Algo_InverseQuantizationND
110	Algo_InversePrediction1D
111	Algo_InversePredictionND
112	Algo_ED_AD_StaticBit
113	Algo_ED_AD_AdaptiveBit
114	Algo_ED_VLD
115	Algo_ED_BitPrecision
116	Algo_ED_AD
117	Algo_ED_AD_EG
118	Algo_ContextModeling

119	Algo_ContextModeling_SVA_nType
120	Algo_ContextModeling_SVA_Indexes
121	Algo_ContextModeling_SVA_Vertex_Attribute
122	Algo_ED_4bitsD
123	Algo_ED_FixedLength
124	Algo_LookUpTable1D
125	Algo_DecodeConnectivity_SVA
126	Algo_DecodeConnectivity_TFAN
127	Algo_ExtractMask_SC3DMC
128	Algo_ExtractFaceDirection_SVA
129	Algo_simpleMath_2op
130	Algo_Connectivity_InversePrediction_SVA
131	Mgnt_Replicate_1_2
132	Mgnt_Replicate_1_4
133	Mgnt_Replicate_1_8
134	Mgnt_MUX_2_1
135	Mgnt_MUX_4_1
136	Mgnt_MUX_8_1
137	Mgnt_DEMUX_1_2
138	Mgnt_DEMUX_1_4
139	Mgnt_DEMUX_1_8
140	Mgnt_ExtractSegment
141	Mgnt_ProviderValue
142	Mgnt_RepeatSegment
143	Mgnt_ExtractBytes
144	Mgnt_ExtractBits
145	Mgnt_Provider1D
146	Mgnt_Provider2D

4.3 "Token Pool":

Add the following rows at the end of the table.

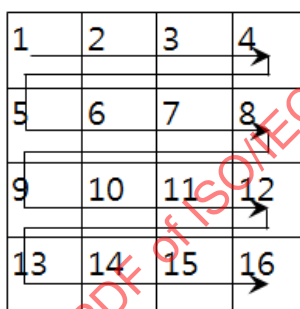
ID & Name	Description
55 BOOLEAN	Token which value is 0 or 1.
56 SIGN	Token which value is 0 or 1.
57 FLAG	Token which value is 0 or 1.
58 UINT_2	Unsigned integer on 2 bits.
59 UINT_4	Unsigned integer on 4 bits.
60 UINT_8	Unsigned integer on 8 bits.
61 UINT_16	Unsigned integer on 16 bits.
62 UINT_32	Unsigned integer on 32 bits.

63 UINT_64	Unsigned integer on 64 bits.
64 INT_8	Integer on 8 bits.
65 INT_16	Integer on 16 bits.
66 INT_32	Integer on 32 bits.
67 INT_64	Integer on 64 bits.
68 FLOAT_32	Float on 32 bits.

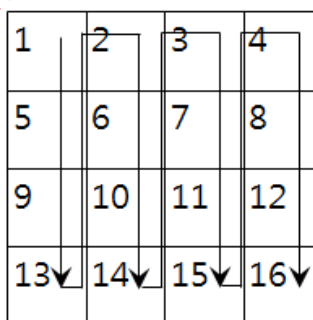
Add 4.4 "Array data order":

4.4 Array data order

- Row based:
 - The data is processed or sent in a sequential order, row by row
- Example:



- Column based
 - The data is processed or sent in a sequential order, column by column
- Example:



Add 4.5 "Input ports":

4.5 Input ports (reset_i, init_i, start_i)

An FU does not have an outside synchronization signal or synchronization mechanism. These ports are used for the purpose of changing the values of the local variables to default or initialization values.

Add 4.6 "FU block diagram notations":

4.6 FU block diagram notations

The notation [EMBED] defines a part of the main FSM schematic that is described as a separate schematic (for complexity reasons). The [EMBED] schematic is an integrated part of the main FSM schematic

The notation [MODULE] defines a part of the main FSM schematic that is defined as a separate FU. The module schematic is integrated in the main schematic with the entire FU logic, except the “START” FSM state. The INPUT/OUTPUT states do not read or write values from the ports, they refer to local variables relative to the FU that embeds the other schematic.

Add 4.7 "Conventions":

4.7 Conventions

The significance of the “sign” port values is:

Value	Significance
0	negative
1	positive

The significance of the “flag” values is:

Value	Significance
0	false
1	true

Add 5.2 "General Processing FUs":

5.2 General Processing FUs

5.2.1 Algo_InverseQuantization1D

FU Name

Algo_InverseQuantization1D

Port Name	Direction (I/O)	Token TYPE
dataIn_i	I	INT8, INT16, INT32, INT64
qp_i	I	UINT_32
quantMin_i	I	FLOAT
quantRange_i	I	FLOAT
segmentSize_i	I	UINT8, UINT16, UINT32, UINT64
quantizationMode_i	I	UINT_2
dataOut_o	O	FLOAT

Description

Inverse Quantization Process:

START:

INPUT_SEGMENT_PARAM:

INPUT:

quantizationMode

segmentSize

SegmentSizeCounter = 0

IF quantizationMode = 0

INPUT:

qp

quantMin

quantRange

IF (quantRange > 0.0)

delta = ((1 << qp) - 1) / quantRange

FU Name	Algo InverseQuantizationND
---------	----------------------------

[illegible]

Inverse Quantization Process:

$$\dim Q = \begin{cases} \dim D, & \text{if homogeneous } Q = 0 \\ 1, & \text{if homogeneous } Q = 1 \end{cases}$$

START:

INPUT_SEGMENT_PARAM

INPUT:

quantizationMode

segmentSize

SegmentSizeCounter = 0

IF quantizationMode = 0

INPUT:

qp

quantMin [dimQ]

quantRange [dimQ]

WHILE dimQ_counter < dimQ

IF (quantRange [dimQ_counter] > 0.0)

delta [dimQ_counter] = ((1 << qp) - 1) / quantRange [dimQ_counter]

ELSE

delta [dimQ_counter] = 1.0;

dimQ_counter++

IF quantizationMode = 1

INPUT:

Qp [dimQ]

IF quantizationMode = 2

INPUT:

qp

PROCESS:

Subdivision = (qp - 3) / 2

INPUT_DATA_IN:

INPUT:

dataIn [dimQ]

IF quantizationMode = 0

PROCESS:

dataOut = quantMin [segmentSizeCounter%dimD] + (dataIn / delta [segmentSizeCounter%dimD])

IF quantizationMode = 1

PROCESS: EMBED Code 2 Normal

IF quantizationMode = 2

PROCESS:

dataOut = dataIn / qp [segmentSizeCounter % dimD]

OUTPUT

dataOut

SegmentSizeCounter++

IF SegmentSizeCounter < segmentSize

GOTO INPUT_DATA_IN

ELSE

GOTO INPUT_SEGMENT_PARAM

EMBED: Code 2 Normal

Mask = (1 << (2 * subdivision)) - 1

tricode = data & mask;

// Find y coordinate by solving 2nd degree equation

factor = 1 << subdivision

y = factor - sqrt ((factor²) - tricode)

tricode = tricode + (y * (y - (2 * factor)))

x = tricode / 2

upsideDown = tricode % 2

// Calculate coordinates for all vertices in triangle

v1x = x + upsideDown

v1y = y + upsideDown

v2x = x + 1

v2y = y

v3x = x

v3y = y + 1

// Calculate coordinates of barycenter

	<pre>invMaxCoord = 1 / factor normal [0] = (v1x + v2x + v3x) * invMaxCoord normal [1] = (v1y + v2y + v3y) * invMaxCoord normal [2] = 3 - normal [0] - normal [1] // Flip component signs if necessary octantCode = (data >> 2 * subdivision) & 0x7 if (octantCode & 0x4) normal [0] = (-1) * normal [0] if (octantCode & 0x2) normal [1] = (-1) * normal [1] if (octantCode & 0x1) normal [2] = (-1) * normal [2] invNorm:= 1 / sqrt ((normal[0])² + (normal [1])² + (normal [2])²); //Write the 3 output values normal [0] = normal [0] * invNorm normal [1] = normal [1] * invNorm normal [2] = normal [2] * invNorm dataOut = normal</pre> The following table contains the quantization types index used in the Inverse Quantization ND FU: <table><tr><th>Name of quantization mode</th><th>Value</th></tr><tr><td>Uniform Quantization</td><td>0</td></tr><tr><td>Normal Quantization</td><td>1</td></tr><tr><td>Uniform Texture Quantization</td><td>2</td></tr></table> The "Inverse Quantization" is an algorithm (step by step procedures) that allows a set of data to be represented with a limited set of values that are associated with its nearest representative. For a number of "segmentSize" x "dimD" of input data (dataIn), it uses the same set of quantMin, quantRange and quantValue of size "dimD" to produce a set of output data (dataOut) of size segmentSize" x "dimD". For each set of size "dimD" of input data (dataIn) it uses the corresponding value of quantMin, quantRange and quantValue.	Name of quantization mode	Value	Uniform Quantization	0	Normal Quantization	1	Uniform Texture Quantization	2
Name of quantization mode	Value								
Uniform Quantization	0								
Normal Quantization	1								
Uniform Texture Quantization	2								
ISO Standards using the FU	ISO/IEC 14496-16:2011								
Profiles@levels supported									
Parameter									
Name	Description	Type / Range							
dimD	Describes the number of tokens of type dataIn_i that are consumed at each firing. This parameter is set at the network configuration level.	Type: Integer Range: [1 .. 2 ⁵]							
homogeneousQ	Describes the number of tokens of type quantRange_i, quantMin_i and quantValue_i that are necessary for the inverse quantization process. This parameter is set at the network configuration level. The number of tokens is equal to dimD if this parameter is 0 and the number of tokens is equal to 1 if this parameter is 1	Type: Boolean Range: {0,1}							

5.2.3 Algo_InversePrediction1D

FU Name Algo_InversePrediction1D

This FU allows five modes of inverse prediction. Not all the ports are used for each prediction mode. This FU can handle one input token at a time.

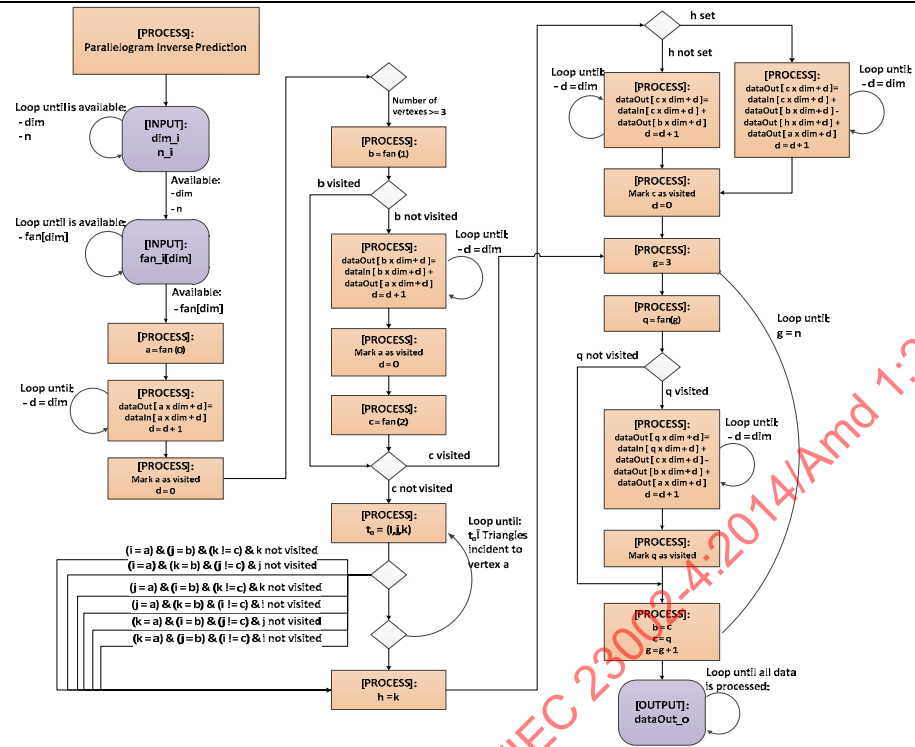
Port Name	Direction (I/O)	Token TYPE
dataIn_i	I	INT8, INT16, INT32, INT64
difValue_i	I	INT8, INT16, INT32, INT64
predMode_i	I	UINT 4
fan_i	I	INT8, INT16, INT32, INT64
dim_i	I	UINT8, UINT16, UINT32, UINT64
n_i	I	INT8, INT16, INT32, INT64
max_i	I	UINT8, UINT16, UINT32, UINT64
dataOut_o	O	INT8, INT16, INT32, INT64

Description

```

graph TD
    START((START)) --> State0([INPUT: predMode_j])
    State0 -- "predMode = 0" --> Loop0[Loop until all dataIn is processed:]
    Loop0 --> In0([INPUT: dataIn_i])
    In0 --> Proc0([PROCESS: dataOut_o])
    Proc0 --> State0
    
    State0 -- "predMode = 1 or 2" --> Loop1[Loop until it is available: - predMode]
    Loop1 --> State0
    State0 -- "predMode = 1 or 2" --> Dec1{ }
    Dec1 -- "j != 0" --> In1([AVAILABLE: dataIn[j-1]])
    In1 --> Proc1([PROCESS: dataOut_o])
    Proc1 --> State0
    Dec1 -- "j = 0" --> Proc2([PROCESS: dataOut_o])
    Proc2 --> State0
    
    State0 -- "predMode = 3" --> In2([INPUT: difValue_i])
    In2 --> Dec2{ }
    Dec2 -- "j = 0 or difValue[j] = 0" --> Proc3([PROCESS: dataOut_o])
    Proc3 --> State0
    Dec2 -- "else" --> Proc4([PROCESS: dataOut_o])
    Proc4 --> In3([AVAILABLE: dataOut[j-difValue[j]]])
    In3 --> State0
    
    State0 -- "predMode = 4" --> In4([INPUT: Max_i])
    In4 --> In5([INPUT: dataIn_i])
    In5 --> Dec3{ }
    Dec3 --> Proc5([PROCESS: dataOut_o])
    Proc5 --> State0
    Dec3 --> Proc6([PROCESS: dataOut_o])
    Proc6 --> In6([AVAILABLE: dataIn[j-1]])
    In6 --> State0
    
    State0 -- "predMode = 5" --> Proc7([PROCESS: Parallelogram Inverse Prediction])
    Proc7 --> Loop2[Loop until all dataIn is processed:]
    Loop2 --> State0
  
```

Process Parallelogram Inverse Prediction Schematic (FSM):



Inverse Prediction Process:

Switch (predMode)

{

Case 0: NP – No Prediction

$$dataOut[j] = dataIn[j], \forall j \in \{0 .. N-1\}.$$

Case 1: Diff – Differential Prediction

$$dataOut[j] = \begin{cases} DataIn[j], & \forall j = 0 \\ DataIn[j] + DataOut[j-1], & \forall j \in \{1 .. N-1\} \end{cases}$$

Case 2: XOR – based prediction

$$dataOut[j] = \begin{cases} dataIn[j], & \forall j = 0 \\ dataIn[j] \otimes dataOut[j-1], & \forall j \in \{1 .. N-1\} \end{cases}$$

Case 3: Adaptive Prediction

$$dataOut[j] = \begin{cases} dataIn[j], & \text{if } difValue[j] = 0 \text{ or } j = 0 \\ dataIn[j] + dataOut[j - difValue[j]], & \text{otherwise} \end{cases}$$

Case 4: Circular Differential Prediction

$$dataOut[j] = \begin{cases} dataIn[j], & \forall j = 0 \\ d, & \text{if } dataIn[j] < dataIn[j-1], \text{ where} \\ -d, & \text{otherwise} \end{cases}$$

$$d = \begin{cases} dataIn[j-1] + M_d - dataIn[j], & \text{if } dataIn[j] > dataIn[j-1] \\ dataIn[j] + M_d - dataIn[j-1], & \text{otherwise} \end{cases}$$

Case 5: Parallelogram Inverse Prediction

```

a = fan ( 0 )
if a not visited
  d = 0
  WHILE d < dim
    dataOut [ a x dim + d ] = dataIn [ a x dim + d ]
    d = d + 1
  Mark a as visited
If number of vertexes > 3
  b = fan ( 1 )
  if b not visited
    d = 0
    WHILE d < dim
      dataOut [ b x dim + d ] = dataIn [ b x dim + d ] + dataOut [ a x dim + d ]
      d = d + 1
    Mark b as visited
  c = fan ( 2 )
  if c not visited
    init h, ta
    WHILE ta = (l,j,k) ∈ Triangles incident to vertex a
      If ( i=a & j=b & k≠c and k not visited ) h = k, break
      If ( i=a & k=b & j≠c and j not visited ) h = k, break
      If ( j=a & i=b & k≠c and k not visited ) h = k, break
      If ( j=a & k=b & i≠c and i not visited ) h = k, break
      If ( k=a & i=b & j≠c and j not visited ) h = k, break
      If ( k=a & b=b & i≠c and i not visited ) h = k, break
    If h not set
      d = 0
      WHILE d < dim
        dataOut [ c x dim + d ] = dataIn [ c x dim + d ] + dataOut [ b x dim + d ]
        d = d + 1
    else
      d = 0
      WHILE d < dim
        dataOut [ c x dim + d ] =
          dataIn [ c x dim + d ] + dataOut [ b x dim + d ]
          - dataOut [ h x dim + d ] + dataOut [ a x dim + d ]
        d = d + 1
      Mark c as visited
  g = 3, d = 0
  WHILE g < fanSize
    q = fan ( g )
    if q not visited
      WHILE d < dim
        dataOut [ q x dim + d ] =
          dataIn [ q x dim + d ] + dataOut [ c x dim + d ]
          - dataOut [ b x dim + d ] + dataOut [ a x dim + d ]
        d = d + 1
      b = c
      c = q

```

The following table contains the prediction types index used in the Inverse Prediction 1D FU:

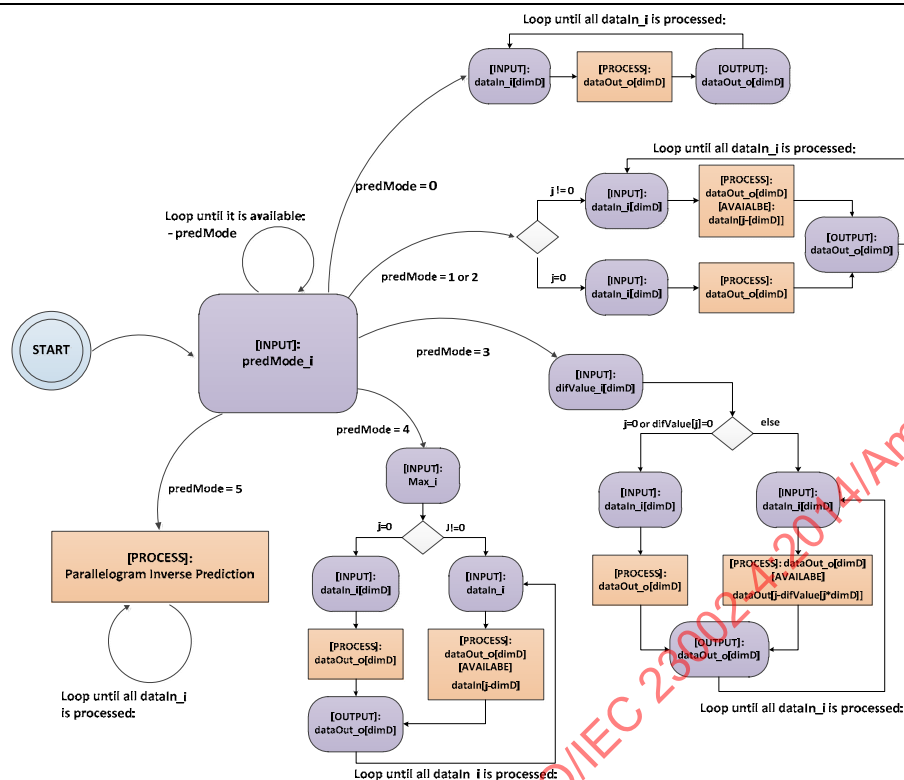
Name of prediction mode	Value
No Prediction	0
Differential Prediction	1
XOR based prediction	2

	Adaptive Differential Prediction	3
	Circular Differential Prediction	4
	Parallelogram Inverse Prediction	5
	The detailed description of the inverse parallelogram prediction is described in Annex F.	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	Description	Range

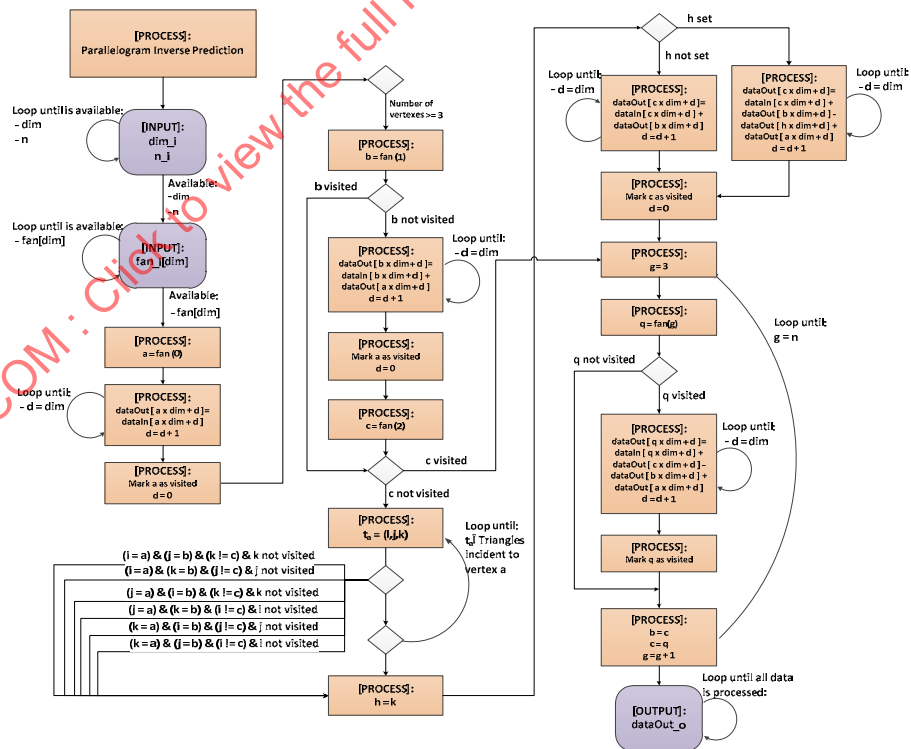
5.2.4 Algo_InversePredictionND

3.2.4 Algo_InversePredictionND

FU Name	Algo_InversePredictionND																											
Description	This FU allows five modes of inverse prediction. Not all the ports are used for each prediction mode. This FU can handle a number of dimD input tokens at a time. InversePredictionND [dimD] dataIn_i dataOut_o difValue_i predMode_i fan_i dim_i n_i max_i <table><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr><tr><td>dataIn_i</td><td>I</td><td>INT8, INT16, INT32, INT64</td></tr><tr><td>difValue_i</td><td>I</td><td>INT8, INT16, INT32, INT64</td></tr><tr><td>predMode_i</td><td>I</td><td>UINT_4</td></tr><tr><td>fan_i</td><td>I</td><td>INT8, INT16, INT32, INT64</td></tr><tr><td>dim_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>n_i</td><td>I</td><td>INT8, INT16, INT32, INT64</td></tr><tr><td>max_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>dataOut_o</td><td>O</td><td>INT8, INT16, INT32, INT64</td></tr></table> Process Schematic (FSM):	Port Name	Direction (I/O)	Token RANGE	dataIn_i	I	INT8, INT16, INT32, INT64	difValue_i	I	INT8, INT16, INT32, INT64	predMode_i	I	UINT_4	fan_i	I	INT8, INT16, INT32, INT64	dim_i	I	UINT8, UINT16, UINT32, UINT64	n_i	I	INT8, INT16, INT32, INT64	max_i	I	UINT8, UINT16, UINT32, UINT64	dataOut_o	O	INT8, INT16, INT32, INT64
Port Name	Direction (I/O)	Token RANGE																										
dataIn_i	I	INT8, INT16, INT32, INT64																										
difValue_i	I	INT8, INT16, INT32, INT64																										
predMode_i	I	UINT_4																										
fan_i	I	INT8, INT16, INT32, INT64																										
dim_i	I	UINT8, UINT16, UINT32, UINT64																										
n_i	I	INT8, INT16, INT32, INT64																										
max_i	I	UINT8, UINT16, UINT32, UINT64																										
dataOut_o	O	INT8, INT16, INT32, INT64																										



Process Parallelogram Inverse Prediction Schematic (FSM):



Inverse Prediction ND Process:

Switch(predMode)

```

{
Case 0: NP – No Prediction


$$dataOut[j] = dataIn[j], \forall j \in \{0 \dots (N \times dimD) - 1\}$$


Case 1: Diff – Differential Prediction


$$dataOut[j] = \begin{cases} dataIn[j], & \forall j = 0 \\ dataIn[j] + dataOut[j - dimD], & \forall j \in \{1 \dots (N \times dimD) - 1\} \end{cases}$$


Case 2: XOR – based prediction


$$dataOut[j] = \begin{cases} dataIn[j], & \forall j = 0 \\ dataIn[j] \otimes dataOut[j - dimD], & \forall j \in \{1 \dots (N \times dimD) - 1\} \end{cases}$$


Case 3: Adaptive Prediction


$$dataOut[j] = \begin{cases} dataIn[j], & \text{if } difValue[j] = 0 \text{ or } j = 0 \\ dataIn[j] + dataOut[j - difValue[j] \otimes dimD], & \text{otherwise} \end{cases}$$


Case 4: Circular Differential Prediction


$$dataOut[j] = \begin{cases} dataIn[j], & \forall j = 0 \\ d, & \text{if } dataIn[j] < dataIn[j - dimD], \text{ where} \\ -d, & \text{otherwise} \end{cases}$$



$$d = \begin{cases} dataIn[j - dimD] + M_d - dataIn[j], & \text{if } dataIn[j] > dataIn[j - dimD] \\ dataIn[j] + M_d - dataIn[j - dimD], & \text{otherwise} \end{cases}$$


}

Case 5: Parallelogram Inverse Prediction

a = fan ( 0 )
if a not visited
    d = 0
    WHILE d < dim
        dataOut [ a x dim + d ] = dataIn [ a x dim + d ]
        d = d + 1
    Mark a as visited
if number of vertexes > 3
    b = fan ( 1 )
    if b not visited
        d = 0
        WHILE d < dim
            dataOut [ b x dim + d ] = dataIn [ b x dim + d ] + dataOut [ a x dim + d ]

```

```

    d = d + 1
    Mark b as visited
    c = fan ( 2 )
    if c not visited
        init h, ta
        WHILE ta = (l,j,k) ∈ Triangles incident to vertex a
            If ( i=a & j=b & k≠c and k not visited ) h = k, break
            If ( i=a & k=b & j≠c and j not visited ) h = k, break
            If ( j=a & i=b & k≠c and k not visited ) h = k, break
            If ( j=a & k=b & i≠c and i not visited ) h = k, break
            If ( k=a & i=b & j≠c and j not visited ) h = k, break
            If ( k=a & b=b & i≠c and i not visited ) h = k, break
        If h not set
            d = 0
            WHILE d < dim
                dataOut [ c x dim + d ] = dataIn [ c x dim + d ] + dataOut [ b x dim + d ]
                d = d + 1
            else
                d = 0
                WHILE d < dim
                    dataOut [ c x dim + d ] =
                        dataIn [ c x dim + d ] + dataOut [ b x dim + d ]
                        - dataOut [ h x dim + d ] + dataOut [ a x dim + d ]

                d = d + 1
            Mark c as visited
            g = 3, d = 0
            WHILE g < fanSize
                q = fan ( g )
                if q not visited
                    WHILE d < dim
                        dataOut [ q x dim + d ] =
                            dataIn [ q x dim + d ] + dataOut [ c x dim + d ]
                            - dataOut [ b x dim + d ] + dataOut [ a x dim + d ]

                        d = d + 1
                    b = c
                    c = q

```

The following table contains the prediction types index used in the Inverse Prediction ND FU:

Name of prediction mode	Value
No Prediction	0
Differential Prediction	1
XOR based prediction	2
Adaptive Differential Prediction	3
Circular Differential Prediction	4
Parallelogram Inverse Prediction	5

The detailed description of the inverse parallelogram prediction is described in Annex F.

**ISO Standards
using the FU**

ISO/IEC 14496-16:2011

**Profiles@levels
supported**

Parameter		
Name	Description	Range
dimD	Describes the number of tokens of type dataIn_i that are consumed at each firing. This parameter is set at the network configuration level.	Type: Integer Range: [1 .. 2 ⁵]

5.2.5 Algo_ED_AD_StaticBit

FU Name	Algo_ED_AD_StaticBit																					
Description	This FU describes the arithmetic decoding process based on a static bit model. ED_AD_StaticBit [AC_MinLength] [BM_LengthShift] Bit_0_prob_i valueOut_o valueIn_i lengthOut_o lengthIn_i bit_o <table><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr><tr><td>Bit_0_prob_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>valueIn_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>lengthIn_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>valueOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>lengthOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>bit_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr></table> Process Schematic (FSM): Loop until are available: - bit_0_prob - lengthIn - valueIn START [INPUT] bit_0_prob_i lengthIn_i valueIn_i Available: - bit_0_prob - lengthIn - valueIn [PROCESS] product bit [PROCESS] valueOut lengthOut [OUTPUT]: bit_o valueOut_o lengthOut_o ED_AD_StaticBit Process: <pre>START INPUT: Bit_0_prob_i lengthIn_i valueIn_i product = bit_0_prob * (lengthIn >> BM_LengthShift) bit = (valueIn >= product) if (bit == 0) valueOut = valueIn lengthOut = product else valueOut = valueIn - product lengthOut = lengthIn - product OUTPUT: valueOut_o lengthOut_o GOTO START</pre>	Port Name	Direction (I/O)	Token RANGE	Bit_0_prob_i	I	UINT8, UINT16, UINT32, UINT64	valueIn_i	I	UINT8, UINT16, UINT32, UINT64	lengthIn_i	I	UINT8, UINT16, UINT32, UINT64	valueOut_o	O	UINT8, UINT16, UINT32, UINT64	lengthOut_o	O	UINT8, UINT16, UINT32, UINT64	bit_o	O	UINT8, UINT16, UINT32, UINT64
	Port Name	Direction (I/O)	Token RANGE																			
	Bit_0_prob_i	I	UINT8, UINT16, UINT32, UINT64																			
	valueIn_i	I	UINT8, UINT16, UINT32, UINT64																			
lengthIn_i	I	UINT8, UINT16, UINT32, UINT64																				
valueOut_o	O	UINT8, UINT16, UINT32, UINT64																				
lengthOut_o	O	UINT8, UINT16, UINT32, UINT64																				
bit_o	O	UINT8, UINT16, UINT32, UINT64																				
ISO Standards using the FU	ISO/IEC 14496-16:2011																					
Profiles@levels supported																						

Parameter		
Name	Description	Range
AC__MinLength	Describes the threshold for renormalization. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: $[1 .. 2^{32}]$
BM__LengthShift	Describes the length bits discarded before multiplication. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: $[1 .. 2^5]$

5.2.6 Alog_ED_AD AdaptiveBit

FU Name	Algo_ED_AD_AdaptiveBit																									
Description	This FU describes the arithmetic decoding process based on a adaptive bit model as presented in [3,4]. ED_AD_AdaptiveBit [AC_MinLength] [BM_LengthShift] [scaleMax] valueIn_i valueOut_o lengthIn_i lengthOut_o reset_i bit_o <table><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr><tr><td>valueIn_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>length_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>reset_i</td><td>I</td><td>BOOLEAN</td></tr><tr><td>valueOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>lengthOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>bit_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr></table> <tr><td></td><td> Process Schematic (FSM): </td></tr> <tr><td></td><td> ED_AD_AdaptiveBit Process: <pre>START If reset = 1 bit_0_count = 1; bit_count = 2; bit_0_prob = 1U << (BM_LengthShift - 1); update_cycle = bits_until_update = 4; product = bit_0_prob * (lengthIn >> BM_LengthShift) bit = (valueIn >= product) if (bit == 0) valueOut = valueIn lengthOut = product ++bit_0_count else valueOut = valueIn - product lengthOut = lengthIn - product if (bit_count += update_cycle) > BM_MaxCount bit_count = (bit_count + 1) >> 1 bit_0_count = (bit_0_count + 1) >> 1 if bit_0_count = bit_count ++bit_count scale = scaleMax / bit_count; bit_0_prob = (bit_0_count * scale) >> (31 - BM_LengthShift); update_cycle = (5 * update_cycle) >> 2;</pre></td></tr>	Port Name	Direction (I/O)	Token RANGE	valueIn_i	I	UINT8, UINT16, UINT32, UINT64	length_i	I	UINT8, UINT16, UINT32, UINT64	reset_i	I	BOOLEAN	valueOut_o	O	UINT8, UINT16, UINT32, UINT64	lengthOut_o	O	UINT8, UINT16, UINT32, UINT64	bit_o	O	UINT8, UINT16, UINT32, UINT64		Process Schematic (FSM):		ED_AD_AdaptiveBit Process: <pre>START If reset = 1 bit_0_count = 1; bit_count = 2; bit_0_prob = 1U << (BM_LengthShift - 1); update_cycle = bits_until_update = 4; product = bit_0_prob * (lengthIn >> BM_LengthShift) bit = (valueIn >= product) if (bit == 0) valueOut = valueIn lengthOut = product ++bit_0_count else valueOut = valueIn - product lengthOut = lengthIn - product if (bit_count += update_cycle) > BM_MaxCount bit_count = (bit_count + 1) >> 1 bit_0_count = (bit_0_count + 1) >> 1 if bit_0_count = bit_count ++bit_count scale = scaleMax / bit_count; bit_0_prob = (bit_0_count * scale) >> (31 - BM_LengthShift); update_cycle = (5 * update_cycle) >> 2;</pre>
	Port Name	Direction (I/O)	Token RANGE																							
	valueIn_i	I	UINT8, UINT16, UINT32, UINT64																							
length_i	I	UINT8, UINT16, UINT32, UINT64																								
reset_i	I	BOOLEAN																								
valueOut_o	O	UINT8, UINT16, UINT32, UINT64																								
lengthOut_o	O	UINT8, UINT16, UINT32, UINT64																								
bit_o	O	UINT8, UINT16, UINT32, UINT64																								
	Process Schematic (FSM):																									
	ED_AD_AdaptiveBit Process: <pre>START If reset = 1 bit_0_count = 1; bit_count = 2; bit_0_prob = 1U << (BM_LengthShift - 1); update_cycle = bits_until_update = 4; product = bit_0_prob * (lengthIn >> BM_LengthShift) bit = (valueIn >= product) if (bit == 0) valueOut = valueIn lengthOut = product ++bit_0_count else valueOut = valueIn - product lengthOut = lengthIn - product if (bit_count += update_cycle) > BM_MaxCount bit_count = (bit_count + 1) >> 1 bit_0_count = (bit_0_count + 1) >> 1 if bit_0_count = bit_count ++bit_count scale = scaleMax / bit_count; bit_0_prob = (bit_0_count * scale) >> (31 - BM_LengthShift); update_cycle = (5 * update_cycle) >> 2;</pre>																									

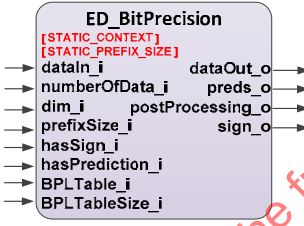
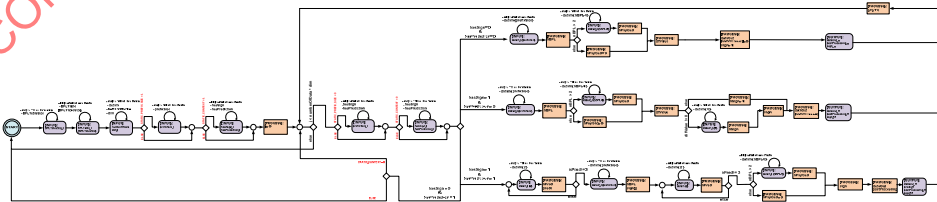
	if update_cycle > 64 update_cycle = 64; bits_until_update = update_cycle; GOTO START	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	Description	Range
AC__MinLength	Describes the threshold for renormalization. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: [1 .. 2 ³²]
BM__LengthShift	Describes the length bits discarded before multiplication. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: [1 .. 2 ⁵]
scaleMax	Describes the max value to compute the scaled bit 0 probability. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: [1 .. 2 ³²]

5.2.7 Algo_ED_VLD

FU Name	Algo_ED_VLD															
Description	This FU describes the Variable Length Decoding process. ED_VLD [nBits] dataIn_i table_i size_i position_o <table><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr><tr><td>dataIn_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>table_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>size_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>position_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr></table> Process Schematic (FSM): ED_VLD Process: <pre>START SET valid[size] = 1 SET n = 1 DO DO If valid[j] = 1 If dataIn=table[j][n] If table[j][0] = n dataOut = j else valid[j] = 0 j = j + 1 WHILE j < size </pre>	Port Name	Direction (I/O)	Token RANGE	dataIn_i	I	UINT8, UINT16, UINT32, UINT64	table_i	I	UINT8, UINT16, UINT32, UINT64	size_i	I	UINT8, UINT16, UINT32, UINT64	position_o	O	UINT8, UINT16, UINT32, UINT64
	Port Name	Direction (I/O)	Token RANGE													
	dataIn_i	I	UINT8, UINT16, UINT32, UINT64													
table_i	I	UINT8, UINT16, UINT32, UINT64														
size_i	I	UINT8, UINT16, UINT32, UINT64														
position_o	O	UINT8, UINT16, UINT32, UINT64														

	n = n + 1 WHILE n < nBits GOTO START The matrix tokens have to be send column based.	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	Description	Range
nBits	Describes the length of the bits used for the search algorithm. This parameter is set at the network configuration level.	Type: Integer Range: [1 .. 2 ⁵]

5.2.8 Algo_ED_BitPrecision

FU Name	Algo_ED_BitPrecision																																								
Description	This FU describes the Entropy Decoding Bit Precision processes.  <table border="1"> <thead> <tr> <th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr> </thead> <tbody> <tr> <td>dataIn_i</td><td>I</td><td>UINT_8</td></tr> <tr> <td>numberOfData_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr> <tr> <td>dim_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr> <tr> <td>prefixSize_i</td><td>I</td><td>UINT_8</td></tr> <tr> <td>hasSign_i</td><td>I</td><td>BOOLEAN</td></tr> <tr> <td>hasPrediction_i</td><td>I</td><td>BOOLEAN</td></tr> <tr> <td>BPLTable_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr> <tr> <td>BPLTableSize_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr> <tr> <td>dataOut_o</td><td>O</td><td>INT8, INT16, INT32, INT64</td></tr> <tr> <td>preds_o</td><td>O</td><td>INT8, INT16, INT32, INT64</td></tr> <tr> <td>postProcessing_o</td><td>O</td><td>BOOLEAN</td></tr> <tr> <td>sign_o</td><td>O</td><td>BOOLEAN</td></tr> </tbody> </table> ED_BitPrecision Schematic (FSM):  ED_BitPrecision process: <pre> START INPUT: BPLTableSize_i INPUT: BPLTable_i [BPLTableSize] INPUT: numberOfData_i INPUT: dim_i IF STATIC_CONTEXT = 1 INPUT: hasSign_i INPUT: hasPrediction_i IF STATIC_PREFIX_SIZE = 1 INPUT: prefixSize_i j = 0 </pre>		Port Name	Direction (I/O)	Token RANGE	dataIn_i	I	UINT_8	numberOfData_i	I	UINT8, UINT16, UINT32, UINT64	dim_i	I	UINT8, UINT16, UINT32, UINT64	prefixSize_i	I	UINT_8	hasSign_i	I	BOOLEAN	hasPrediction_i	I	BOOLEAN	BPLTable_i	I	UINT8, UINT16, UINT32, UINT64	BPLTableSize_i	I	UINT8, UINT16, UINT32, UINT64	dataOut_o	O	INT8, INT16, INT32, INT64	preds_o	O	INT8, INT16, INT32, INT64	postProcessing_o	O	BOOLEAN	sign_o	O	BOOLEAN
Port Name	Direction (I/O)	Token RANGE																																							
dataIn_i	I	UINT_8																																							
numberOfData_i	I	UINT8, UINT16, UINT32, UINT64																																							
dim_i	I	UINT8, UINT16, UINT32, UINT64																																							
prefixSize_i	I	UINT_8																																							
hasSign_i	I	BOOLEAN																																							
hasPrediction_i	I	BOOLEAN																																							
BPLTable_i	I	UINT8, UINT16, UINT32, UINT64																																							
BPLTableSize_i	I	UINT8, UINT16, UINT32, UINT64																																							
dataOut_o	O	INT8, INT16, INT32, INT64																																							
preds_o	O	INT8, INT16, INT32, INT64																																							
postProcessing_o	O	BOOLEAN																																							
sign_o	O	BOOLEAN																																							

```

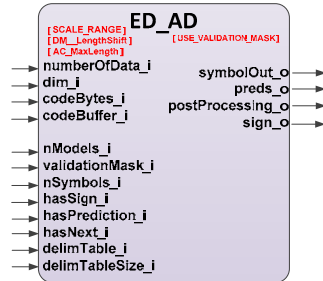
PROCESS_CHUNK:
WHILE j < numberOfData * dim
  IF STATIC_PREFIX_SIZE = 0
    INPUT: prefixSize_i
  IF STATIC_CONTEXT = 0
    INPUT: hasSign_i
    INPUT: hasPrediction_i
  IF hasSign = 0 & hasPrediction = 0
    INPUT: dataIn { prefixSize }
    nBPL = dataIn
    IF nBPL > 2
      INPUT: dataIn { nBPL - 1 }
      nPayload = dataIn
    ELSE
      nPayload = 0
    difValue = BPLTable [ nBPL ] + nPayload
    dataOut = difValue
    postProcessing = 0
    sign = 0
    OUTPUT: dataOut
    OUTPUT: postProcessing_o
    OUTPUT: sign_o
  ELSE IF hasSign = 1 & hasPrediction = 0
    INPUT: dataIn { prefixSize }
    nBPL = dataIn
    if nBPL > 2
      INPUT: dataIn { nBPL - 1 }
      nPayload = dataIn
    ELSE
      nPayload = 0
    difValue = BPLTable [ nBPL ] + nPayload
    IF difValue != 0
      INPUT: dataIn { 1 }
      nSign = dataIn
    ELSE
      nSign = 1
    sign = nSign
    postProcessing = 1
    dataOut = difValue
    OUTPUT: dataOut
    OUTPUT: postProcessing_o
    OUTPUT: sign_o
  ELSE IF hasSign = 1 & hasPrediction = 1
    signDef[2] = { 1, -1 }
    predsOut [ j ] = 0
    DO
      INPUT: dataIn { 2 }
      nPred = dataIn
      preds += nPred
      WHILE nPred != 3
      INPUT: dataIn { prefixSize }
      nBPL = dataIn
      IF ( nBPL > 2 )
        INPUT: dataIn { nBPL - 1 }
        nPayload = dataIn
      ELSE
        nPayload = 0
      difValue = BPLTable [ nBPL ] + nPayload
      dataOut = difValue
      IF preds != 0
        postProcessing = 1
      ELSE
        postProcessing = 0
      IF postProcessing != 0
        sign = signDef [ difValue % 2 ]
      ELSE
        sign = 1
      OUTPUT: dataOut_o
      OUTPUT: preds_o
      OUTPUT: postProcessing_o
      OUTPUT: sign_o
    ELSE IF hasSign = 1 & hasPrediction = 0
      IF STATIC_CONTEXT = 0
        PROCESS_CHUNK
      ELSE

```

	GOTO START	
	GOTO START	
	An input of size {N} has the meaning of a input of a N-size bit value.	
	Note: The case when both hasSign=1 and hasPrediction=0 is not considered. The behaviour in this case is to return to the START stage (if the parameter STATIC_CONTEXT=1) or to process the next data (if the parameter STATIC_CONTEXT=0).	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	Description	Range
STATIC_CONTEXT	Describes whether the context data (hasSign and hasPrediction) is read for every processing iteration. If set to 1, the context data is read only once and reused during the process, if set to 0, the context data is read for every processing iteration. This parameter is set at the network configuration level.	Type:Boolean Range: {0,1}
STATIC_PREFIX_SIZE	Describes whether the prefix size value is read for every processing iteration. If set to 1, the prefix value is read only once and reused during the process, if set to 0, the prefix value is read for every processing iteration. This parameter is set at the network configuration level	Type:Boolean Range: {0,1}

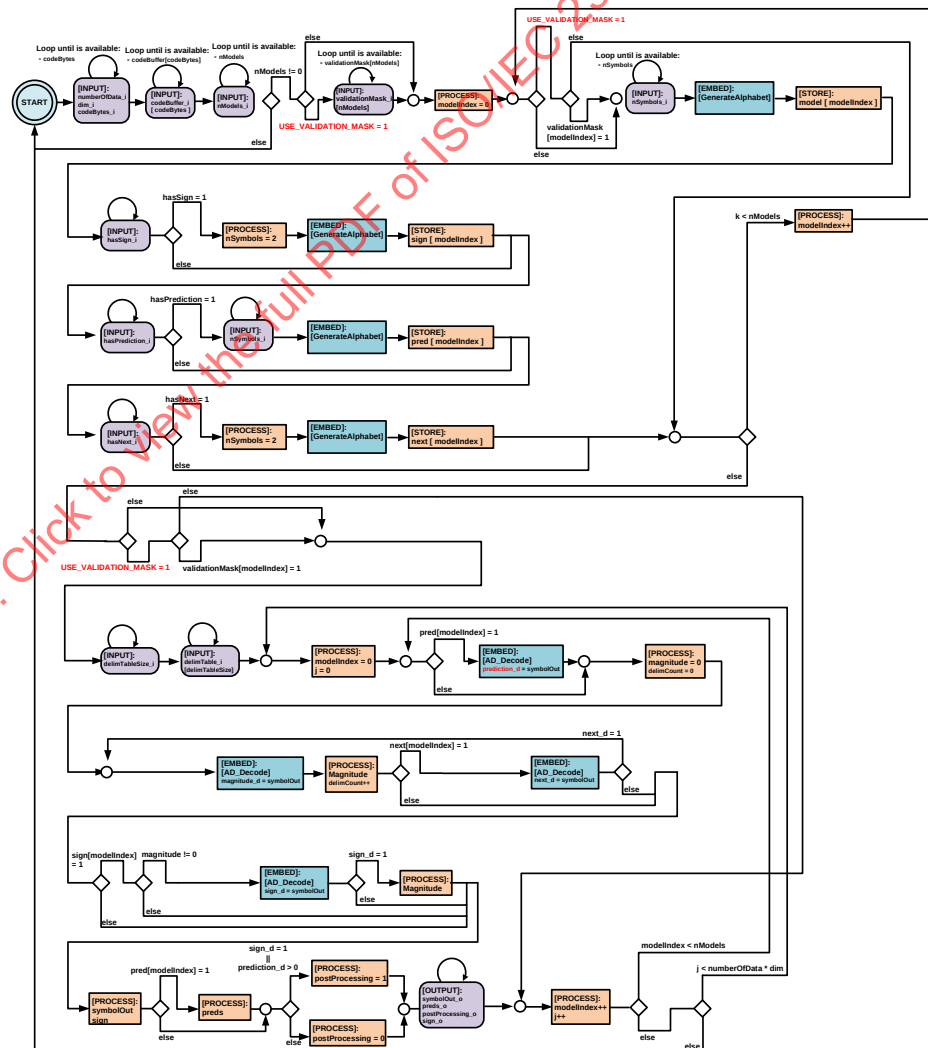
5.2.9 Algo_ED_AD

FU Name	Algo_ED_AD
Description	This FU describes the Entropy Decoding Arithmetic Decoding processes.

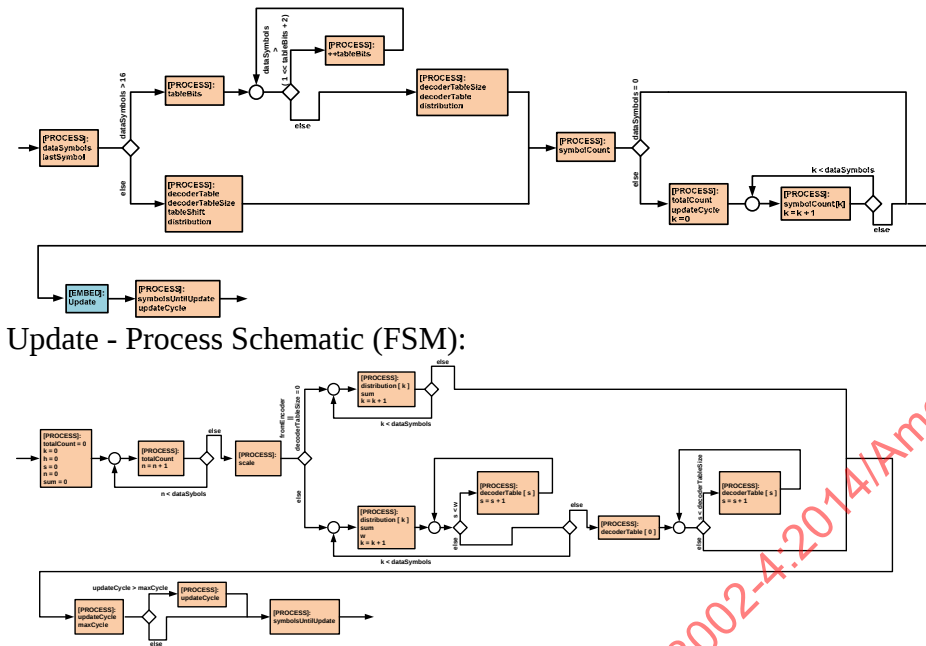


Port Name	Direction (I/O)	Token RANGE
numberOfData_i	I	UINT8, UINT16, UINT32, UINT64
dim_i	I	UINT8, UINT16, UINT32, UINT64
codeBytes_i	I	UINT8, UINT16, UINT32, UINT64
codeBuffer_i	I	UINT_8
nModels_i	I	UINT_8
validationMask_i	I	BOOLEAN
nSymbols_i	I	UINT8, UINT16, UINT32, UINT64
hasSign_i	I	BOOLEAN
hasPrediction_i	I	BOOLEAN
hasNext_i	I	BOOLEAN
delimTableSize_i	I	UINT_8
delimTable_i	I	UINT8, UINT16, UINT32, UINT64
symbolOut_o	O	INT8, INT16, INT32, INT64
preds_o	O	INT8, INT16, INT32, INT64
postProcessing_o	O	BOOLEAN
sign_o	O	BOOLEAN

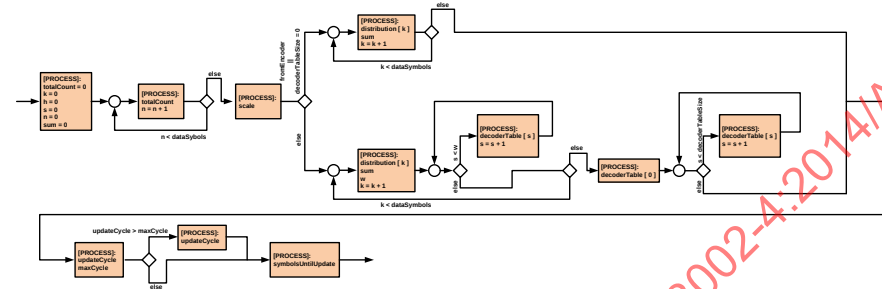
ED_AD Schematic (FSM):



Generate Alphabet Process Schematic (FSM):

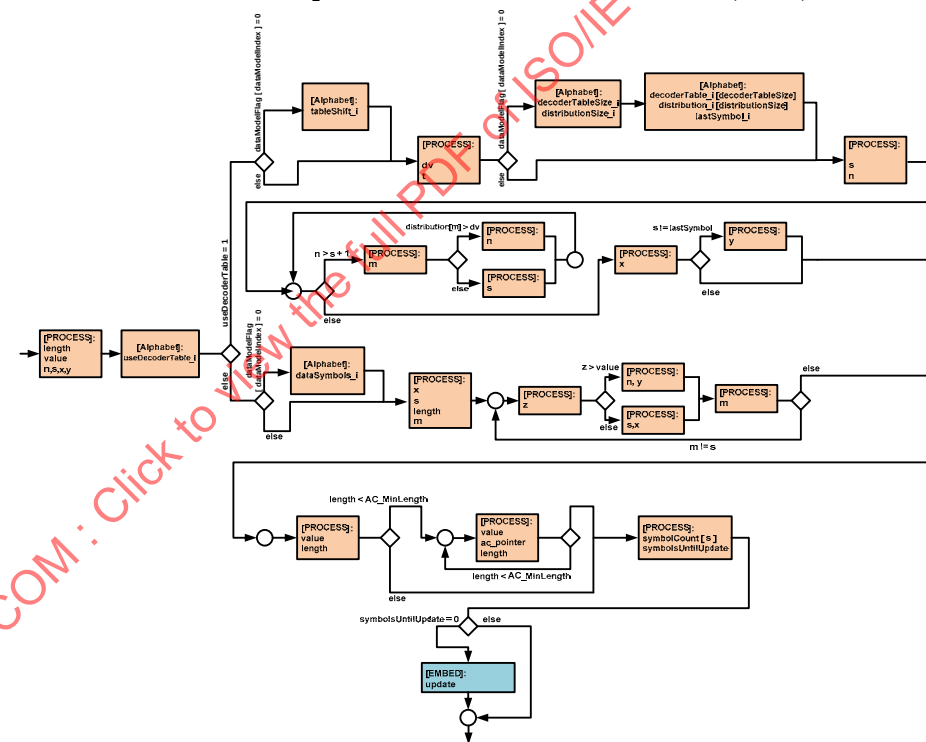


Update - Process Schematic (FSM):



Generate Alphabet Process:

Arithmetic Decode Adaptive Data - Process Schematic (FSM):

**ED_AD process:**

```

START
INPUT: numberOfData_i
INPUT: dim_i
INPUT: codeBytes_i
INPUT: codeBuffer_i[codeBytes]
INPUT: nModels_i
if nModels != 0
    if USE_VALIDATION_MASK = 1
        INPUT: validationMask_i [ nModels ]
    modelIndex = 0
    k = 0
  
```

```

DO
  if USE_VALIDATION_MASK = 1
    if validationMask [ modelIndex ] = 1
      GOTO MODEL_CONFIG
    else
      GOTO NEXT_MODEL_CONFIG
  else
    GOTO MODEL_CONFIG

MODEL_CONFIG:
  INPUT: nSymbols_i
  EMBED: Generate Alphabet
  STORE: model [ modelIndex ]
  INPUT: hasSign_i
  if hasSign = 1
    nSymbols = 2
    EMBED: Generate Alphabet
    STORE: sign [ modelIndex ]
  INPUT: hasPrediction_i
  if hasPrediction = 1
    INPUT: nSymbols_i
    EMBED: Generate Alphabet
    STORE: pred [ modelIndex ]
  INPUT: hasNext_i
  nSymbols = 2
  EMBED: Generate Alphabet
  STORE: next [ modelIndex ]
  k = k + 1
  WHILE k < nModels
    INPUT: delimTableSize_i
    INPUT: delimTable_i [ delimTableSize ]

INIT:
  modelIndex = 0
  j = 0
DECODE:
  if USE_VALIDATION_MASK = 1
    if validationMask [ modelIndex ] = 1
      GOTO MODEL_PROCESS
    else
      GOTO NEXT_MODEL
  else
    GOTO MODEL_PROCESS

MODEL_PROCESS:
  if pred[modelIndex] = 1
    EMBED: AD_Decode (prediction_d = symbolOut)
    magnitude = 0
    delimCount = 0
  HAS_NEXT:
    EMBED: AD_DECODE (magnitude_d = symbolOut)
    Magnitude
    delimCount++
    if next [ modelIndex ] = 1
      EMBED: AD_DECODE (next_d = symbolOut)
    if next_d = 1
      GOTO HAS_NEXT
    if sign[modelIndex] = 1
      if magnitude != 0
        EMBED: AD_DECODE (sign_d = symbolCount)
    symbolOut = magnitude
    sign = sign_d
    if pred[modelIndex] = 1
      preds = prediction_d
    if sign_d = 1 || prediction_d > 0
      postProcessing = 1
    else
      postProcessing = 0
    OUTPUT: symbolOut_o
    OUTPUT: preds_o
    OUTPUT: postProcessing_o
    OUTPUT: sign_o

NEXT_MODEL:
  modelIndex++

```

```

j++

if modelIndex < nModels
    GOTO DECODE
if j < numberOfData * dim
    GOTO INIT

GOTO START

```

Generate Alphabet Process:

```

dataSymbols = nSymbols
lastSymbol = nSymbols - 1
if dataSymbols > 16
    tableBits = 3
    WHILE dataSymbols > ( 1 << (tableBits + 2) )
        ++tableBits
    tableSize = 1 << tableBits
    tableShift = DM__LengthShift - tableBits
    distribution [ 2 * dataSymbols + tableSize + 2 ]
    decoderTable = distribution + 2 * dataSymbols
else
    decoderTable = 0
    tableSize = 0
    tableShift = 0
    distribution [ 2 * dataSymbols ]
    symbolCount = distribution + dataSymbols
    if dataSymbols != 0
        totalCount = 0
        updateCycle = dataSymbols
        k = 0
        WHILE k < dataSymbols
            symbolCount [ k ] = 1
            k = k + 1
    EMBED: Update
    symbolsUntilUpdate = ( dataSymbols + 6 ) >> 1
    updateCycle = ( dataSymbols + 6 ) >> 1

```

Update – Process:

```

totalCount = 0
k = 0
h = 0
s = 0
n = 0
sum = 0
WHILE n < dataSymbols
    totalCount += ( symbolCount [ n ] = (symbolCount [ n ] + 1) >> 1 )
scale = SCALE_RANGE / totalCount
if fromEncoder = true | tableSize = 0
    WHILE k < dataSymbols
        distribution [ k ] = ( scale * sum ) >> ( 31 - DM_LengthShift )
        sum += symbolCount [ k ]
        k = k + 1
else
    WHILE k < dataSymbols
        distribution [ k ] = ( scale * sum ) >> ( 31 - DM_LengthShift )
        sum += symbolCount [ k ]
        w = distribution [ k ] >> tableShift
        WHILE s < w
            decoderTable [ ++s ] = k - 1
        decoderTable [ 0 ] = 0
        WHILE s <= tableSize
            decoderTable [ ++s ] = dataSymbols - 1
    updateCycle = ( 5 * updateCycle ) >> 2
    maxCycle = ( dataSymbols + 6 ) << 3
    if ( updateCycle > maxCycle )
        updateCycle = maxCycle
    symbolsUntilUpdate = updateCycle

```

Arithmetic Decode Adaptive Data Process:

```

codeBytes = 0
WHILE codeBytes <= 0
    INPUT: codeBytes
    length = 0

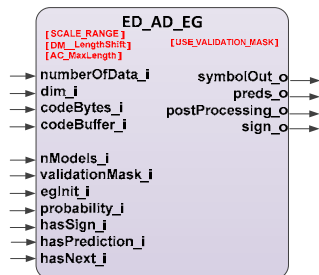
```

	<pre>value = 0 n = 0 s = 0 x = 0 y = 0 if useDecoderTable = 1 if dataModelFlag [dataModelIndex] = 0 INPUT: tableShift dv = value / (length >>= DM__LengthShift) t = dv >> tableShift s = decoderTable [t] n = decoderTable [t+1] + 1 WHILE n > s + 1 m = (s + n) >> 1 if distribution [m] > dv n = m else s = m x = distribution [s] * length if s != lastSymbol y = distribution [s + 1] * length else x = 0 s = 0 length >>= DM__LengthShift m = (n = dataSymbols) >> 1 DO z = length * distribution [m] if z > value n = m y = z else s = m x = z WHILE m = ((s+n) >> 1) != s value -= x length = y - x if length < AC__MinLength DO value = (value << 8) ++ac_pointer WHILE (length <= 8) < AC__MinLength ++symbolCount [s] if -- symbolsUntilUpdate = 0 EMBED: Update</pre>	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
AC__MaxLength	Describes the maximum AC interval. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: [1 .. 2 ³²]
DM__LengthShift	Describes the number of bits discarded before multiplication. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: [1 .. 2 ³²]
SCALE_RANGE	Describes the range for the scale value. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: [1 .. 2 ³²]
USE_VALIDATION_MASK	Indicates whether to use the validation mask input port or not.(0 – NO, 1 – YES)	Type:Unsigned Integer Range: {0,1}

5.2.10 Algo_ED_AD_EG

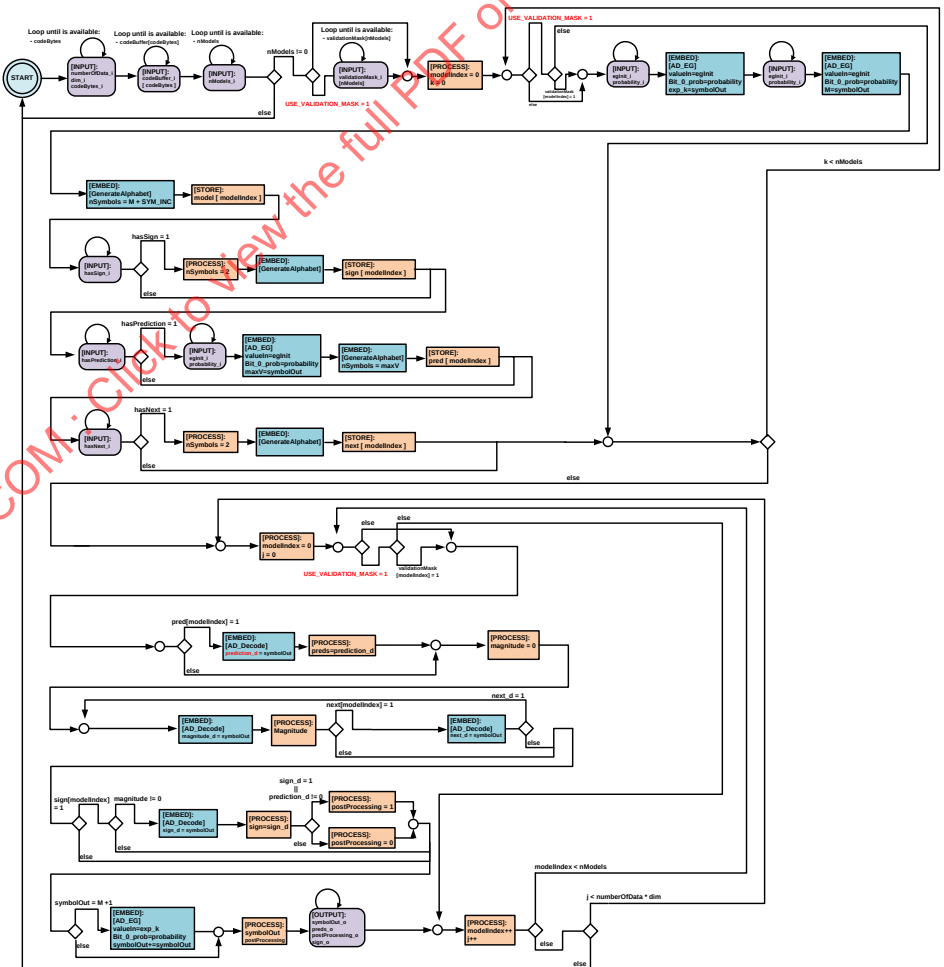
FU Name	Algo_ED_AD_EG
----------------	---------------

This FU describes the Entropy Decoding Arithmetic Decoding Exponential Golomb processes.

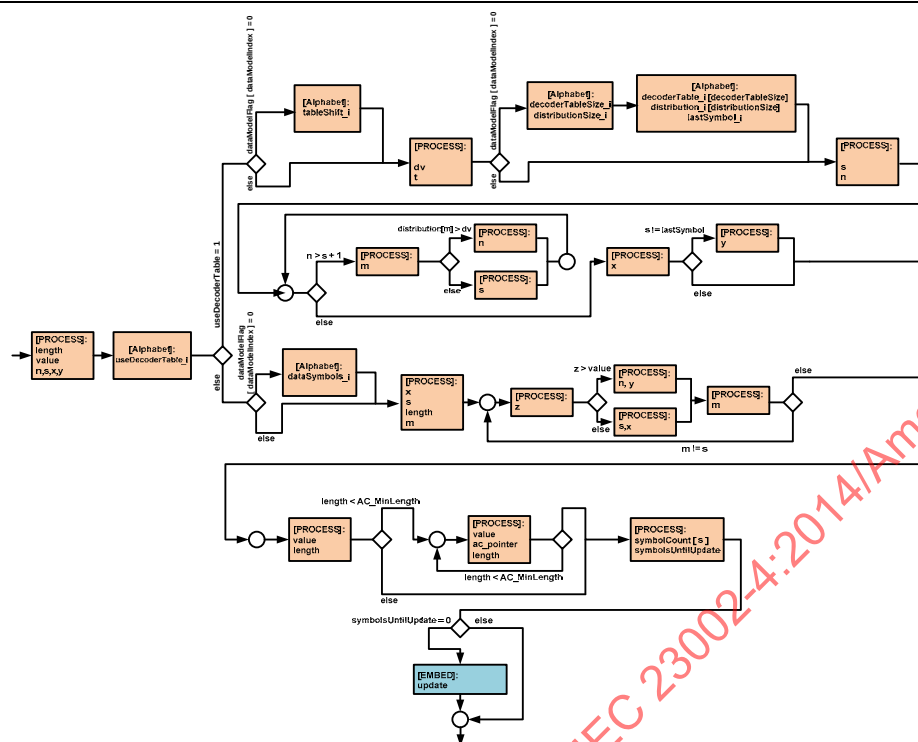


Port Name	Direction (I/O)	Token RANGE
numberOfData_i	I	UINT8, UINT16, UINT32, UINT64
dim_i	I	UINT8, UINT16, UINT32, UINT64
codeBytes_i	I	UINT8, UINT16, UINT32, UINT64
codeBuffer_i	I	UINT_8
nModels_i	I	UINT_8
validationMask_i	I	BOOLEAN
eglnit_i	I	UINT8, UINT16, UINT32, UINT64
probability_i	I	UINT8, UINT16, UINT32, UINT64
hasSign_i	I	BOOLEAN
hasPrediction_i	I	BOOLEAN
hasNext_i	I	BOOLEAN
symbolOut_o	O	INT8, INT16, INT32, INT64
preds_o	O	INT8, INT16, INT32, INT64
postProcessing_o	O	BOOLEAN
sign_o	O	BOOLEAN

ED_AD_EG Schematic (FSM):



Description



ED_AD_EG process:

START

INPUT: numberOfData i

INPUT: dim i

INPUT: codeBytes i

```
INPUT: delimTableSize i
```

```
INPUT: codeBuffer_i[codeBytes]
```

```
INPUT: delimTable_i[delimTableSize]
```

INPUT: nModels_i

```
if USE_VALIDATION_MASK=1
```

```
INPUT: validationMask[nModels]
```

```
if nModels != 0
```

modelIndex = 0

 $k = 0$

INPUT: eglnit i

INPUT: probability i

```
EMBED: AD_EG (valueIn=egInit, Bit_0_prob=probability,exp_k=symbolOut)
```

INPUT: eglNit i

INPUT: probability i

EMBED: AD_EG (valueIn=egInit, Bit_0_prob=probability,exp_k=symbolOut)

DO

if USE_VALIDATION_MASK = 1

```
if validationMask [nModels] = 1
```

EMBED: Generate Alphabet (nSymbols=M+SYM_INC)

```
STORE: model [ modelIndex ]
```

INPUT: hasSign_i

```
if hasSign = 1
```

nSymbols = 2

EMBED: Generate Alphabet

```
STORE: sign [ modelIndex ]
```

INPUT: hasPrediction_i

if hasPrediction = 1

INPUT: eglnit i

INPUT: probability_i

```
EMBED: AD_E
Bit_0_prob=probability,maxV=symbolOut)
```

EMBED: Generate Alphabet (nSymbols=maxV)

```
STORE: pred [ modelIndex ]
```

INPUT: hasNext i

nSymbols = 2

EMBED: Generate Alphabet

```
STORE: next [ modelIndex ]
```

$$k = k + 1$$

	<pre> WHILE k < nModels j = 0 DECODE: modelIndex = 0 NEXT_MODEL: if USE_VALIDATION_MASK = 1 if validationMask [nModel] = 1 if pred[modelIndex] = 1 EMBED: AD_Decode (prediction_d = symbolOut) magnitude = 0 HAS_NEXT: EMBED: AD_DECODE (magnitude_d = symbolOut) Magnitude if next [modelIndex] = 1 EMBED: AD_DECODE (next_d = symbolOut) if next_d = 1 GOTO HAS_NEXT if sign[modelIndex] = 1 if magnitude != 0 EMBED: AD_DECODE (sign_d = symbolCount) EMBED: AD_EG (valueIn=exp_k, bit_0_prob=probability, symbolOut+=symbolOut) symbolOut = magnitude sign = sign_d if pred[modelIndex] = 1 preds = prediction_d if sign_d = 1 prediction_d != 0 postProcessing = 1 else postProcessing = 0 OUTPUT: symbolOut_o OUTPUT: predst_o OUTPUT: postProcessing_o OUTPUT: sign_o modelIndex++ if modelIndex < nModels GOTO NEXT_MODEL j++ if j < numberOfData * dim GOTO DECODE GOTO START AD_EG Process: START l = 0 symbol = 0 binarySymbol = 0 DO EMBED: EC_AC_StaticBit (bit_0_prob=probability, l = symbolOut) if l = 1 symbol += (1<<valueIn) valueIn++ WHILE l != 0 WHILE valueIn— EMBED: EC_AC_StaticBit (bit_0_prob=probability, l = symbolOut) if l = 1 binarySymbol = (1<<valueIn) symbolOut = symbol + binarySymbol Generate Alphabet Process: dataSymbols = nSymbols lastSymbol = nSymbols – 1 if dataSymbols > 16 tableBits = 3 WHILE dataSymbols > (1 << (tableBits + 2)) ++tableBits tableSize = 1 << tableBits tableShift = DM__LengthShift – tableBits distribution [2 * dataSymbols + tableSize + 2] decoderTable = distribution + 2 * dataSymbols else decoderTable = 0 tableSize = 0 tableShift = 0 </pre>
--	---

```

distribution [ 2 * dataSymbols ]
symbolCount = distribution + dataSymbols
if dataSymbols != 0
totalCount = 0
updateCycle = dataSymbols
k = 0
WHILE k < dataSymbols
symbolCount [ k ] = 1
k = k + 1
EMBED: Update
symbolsUntilUpdate = ( dataSymbols + 6 ) >> 1
updateCycle = ( dataSymbols + 6 ) >> 1

```

Update – Process:

```

totalCount = 0
k = 0
h = 0
s = 0
n = 0
sum = 0
WHILE n < dataSymbols
totalCount += ( symbolCount [ n ] = (symbolCount [ n ] + 1 ) >> 1 )
scale = SCALE_RANGE / totalCount
if fromEncofer = true | tableSize = 0
WHILE k < dataSymbols
distribution [ k ] = ( scale * sum ) >> ( 31 – DM_LengthShift )
sum += symbolCount [ k ]
k = k + 1
else
WHILE k < dataSymbols
distribution [ k ] = ( scale * sum ) >> ( 31 – DM_LengthShift )
sum += symbolCount [ k ]
w = distribution [ k ] >> tableShift
WHILE s < w
decoderTable [ ++s ] = k – 1
decoderTable [ 0 ] = 0
WHILE s <= tableSize
decoderTable [ ++s ] = dataSymbols – 1
updateCycle = ( 5 * updateCycle ) >> 2
maxCycle = ( dataSymbols + 6 ) << 3
if ( updateCycle > maxCycle )
updateCycle = maxCycle
symbolsUntilUpdate = updateCycle

```

Arithmetic Decode Adaptive Data Process:

```

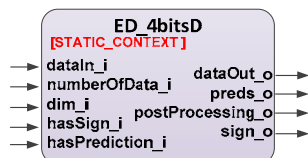
codeBytes = 0
WHILE codeBytes <= 0
INPUT: codeBytes
length = 0
value = 0
n = 0
s = 0
x = 0
y = 0
if useDecoderTable = 1
if dataModelFlag [ dataModelIndex ] = 0
INPUT: tableShift
dv = value / ( length >>= DM__LengthShift )
t = dv >> tableShift
s = decoderTable [ t ]
n = decoderTable [ t+1 ] + 1
WHILE n > s + 1
m = ( s + n ) >> 1
if distribution [ m ] > dv
n = m
else
s = m
x = distribution [ s ] * length
if s != lastSymbol
y = distribution [ s + 1 ] * length
else
x = 0
s = 0
length >>= DM_LengthShift

```

	<pre>m = (n = dataSymbols) >> 1 DO z = length * distribution [m] if z > value n = m y = z else s = m x = z WHILE m = ((s+n) >> 1) != s value -= x length = y - x if length < AC__MinLength DO value = (value << 8) ++ac_pointer WHILE (length <= 8) < AC__MinLength ++symbolCount [s] if -- symbolsUntilUpdate = 0 EMBED: Update</pre>	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
AC__MaxLength	Describes the maximum AC interval. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: [1 .. 2 ³²]
DM__LengthShift	Describes the number of bits discarded before multiplication. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: [1 .. 2 ³²]
SCALE_RANGE	Describes the range for the scale value. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: [1 .. 2 ³²]
USE_VALIDATION_MASK	Indicates whether to use the validation mask input port or not.(0 – NO, 1 – YES)	Type:Unsigned Integer Range: {0,1}

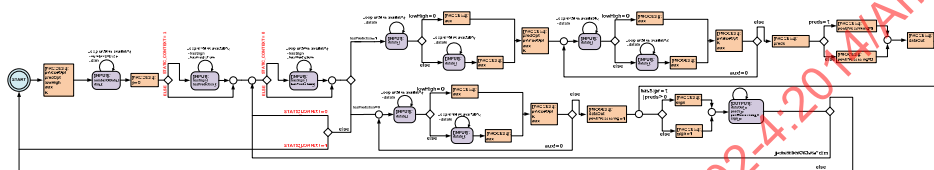
5.2.11 Algo_ED_4bitsD

FU Name	Algo_ED_4bitsD
Description	This FU describes the Entropy Decoding - 4 bits Decoding processes.



Port Name	Direction (I/O)	Token RANGE
dataIn_i	I	UINT_8
numberOfData_i	I	UINT8, UINT16, UINT32, UINT64
dim_i	I	UINT8, UINT16, UINT32, UINT64
hasSign_i	I	BOOLEAN
hasPrediction_i	I	BOOLEAN
dataOut_o	O	INT8, INT16, INT32, INT64
preds_o	O	INT8, INT16, INT32, INT64
postProcessing_o	O	BOOLEAN
sign_o	O	BOOLEAN

ED 4 bits Decoding Schematic (FSM):



ED 4 bits Decoding process:

```

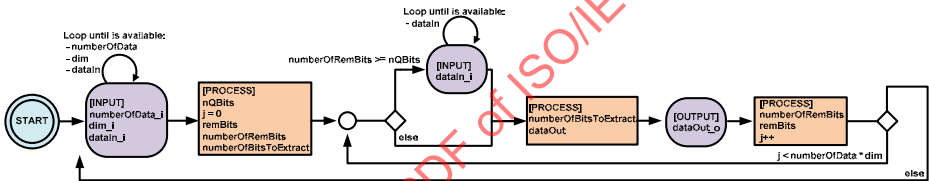
START
signDef = { -1, 1 }
pValueOpt = 0
predOpt = 0
lowHigh = 0
aux = 0
k = 0
INPUT:
  numberOfData_i
  dim_i
IF STATIC_CONTEXT = 1
  INPUT:
    hasSign_i
    hasPrediction_i
  j = 0
  DO
    IF STATIC_CONTEXT = 1
      INPUT:
        hasSign_i
        hasPrediction_i
      IF hasPrediction = 1
        INPUT:
          dataIn_i
          IF lowHigh = 0
            aux = dataIn & 15
          ELSE
            INPUT:
              dataIn_i
              aux = dataIn & 240
            lowHigh = (lowHigh+1)%2
            predOpt = aux & 7
            pValueOpt = 0
            aux >>= 3
            pValueOpt += aux
            WHILE aux
              INPUT:
                dataIn_i
              IF lowHigh = 0
                aux = dataIn & 15
              ELSE
                INPUT:
                  dataIn_i
                  aux = dataIn & 240
                lowHigh = (lowHigh+1)%2
                pValueOpt += (aux&7) << k
                k += 3
                aux >>= 3

```

	<pre> pValueOpt += aux << k IF preds > 0 postProcessing = 1 ELSE postProcessing = 0 preds = predOpt IF hasSign = 1 preds > 0 sign = signDef [pValueOpt % 2] ELSE sign = 1 dataOut = pValueOpt OUTPUT: dataOut_o OUTPUT: preds_o OUTPUT: postProcessing_o OUTPUT: sign_o ELSE IF hasPrediction = 0 k = 0 pValueOpt = 0 DO INPUT: dataIn_i if lowHigh = 0 aux = dataIn & 15 else INPUT: dataIn_i aux = dataIn & 240 lowHigh = (lowHigh+1)%2 pValueOpt += (aux&7) << k k += 3 aux >>= 3 pValueOpt += aux << k WHILE aux dataOut = pValueOpt IF hasSign = 1 sign = signDef [pValueOpt % 2] ELSE sign = 1 postProcessing = 1 OUTPUT: dataOut_o OUTPUT: postProcessing_o OUTPUT: sign_o j++ ELSE IF STATIC_CONTEXT = 1 GOTO START WHILE j < numberOfData * dim GOTO START </pre> Note: The following cases are not considered: - hasSign=0, hasPrediction=0 - hasSign=0, hasPrediction=1 The behaviour in this case is to return to the START stage (if the parameter STATIC_CONTEXT=1) or to process the next data (if the parameter STATIC_CONTEXT=0).
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	
Parameter	

STATIC_CONTEXT	Describes whether the context data (hasSign and hasPrediction) is read for every processing iteration. If set to 1, the context data is read only once and reused during the process, if set to 0, the context data is read for every processing iteration. This parameter is set at the network configuration level.	Type:Boolean Range: {0,1}
-----------------------	---	----------------------------------

5.2.12 Algo_ED_FixedLength

FU Name	Algo_ED_FixedLength																
Description	This FU describes the Entropy Decoding Fixed Length processes.  <table border="1"> <thead> <tr> <th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr> </thead> <tbody> <tr> <td>dataIn_i</td><td>I</td><td>UINT8</td></tr> <tr> <td>numberOfData_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr> <tr> <td>dim_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr> <tr> <td>dataOut_o</td><td>O</td><td>INT8, INT16, INT32, INT64</td></tr> </tbody> </table> ED Fixed Length Decoding Schematic (FSM):  ED Fixedlength Decoding process: <pre> START INPUT: numberOfData_i dim_i predMode_i dataIn_i nQBits = dataIn_i j = 0 remBits = 0 numberOfRemBits = 0 numberOfBitsToExtract = 0 DO If numberOfRemBits >= nQBits INPUT: dataIn_i numberOfBitsToExtract = nQBits - numberOfRemBits dataOut = remBits (dataIn[numberOfBitsToExtract]) numberOfRemBits = nQBits - numberOfBitsToExtract remBits = numberOfRemBits from dataIn j++ WHILE j < numberOfData * dim GOTO START </pre> Notes: - remBits – the remaining bits from the last iterative operation - numberOfRemBits – the number of remaining bits from the last iterative operation - numberOfBitsToExtract – the number of necessary bits to complete a number of nQBits		Port Name	Direction (I/O)	Token RANGE	dataIn_i	I	UINT8	numberOfData_i	I	UINT8, UINT16, UINT32, UINT64	dim_i	I	UINT8, UINT16, UINT32, UINT64	dataOut_o	O	INT8, INT16, INT32, INT64
Port Name	Direction (I/O)	Token RANGE															
dataIn_i	I	UINT8															
numberOfData_i	I	UINT8, UINT16, UINT32, UINT64															
dim_i	I	UINT8, UINT16, UINT32, UINT64															
dataOut_o	O	INT8, INT16, INT32, INT64															
ISO Standards using the FU	ISO/IEC 14496-16:2011																
Profiles@levels supported																	

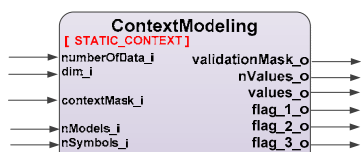
Parameter

5.2.13 Algo_LookUpTable1D

FU Name	Algo_LookUpTable1D																
Description	The FU implements the Look-Up-Table concept by using an 1-dimensional array as a table.																
	LookUpTable1D map_i mapSize_i index_i dataOut_o <table><thead><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr></thead><tbody><tr><td>map_i</td><td>I</td><td>INT8, INT16, INT32, INT64</td></tr><tr><td>mapSize_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>index_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>dataOut_o</td><td>O</td><td>INT8, INT16, INT32, INT64</td></tr></tbody></table>		Port Name	Direction (I/O)	Token RANGE	map_i	I	INT8, INT16, INT32, INT64	mapSize_i	I	UINT8, UINT16, UINT32, UINT64	index_i	I	UINT8, UINT16, UINT32, UINT64	dataOut_o	O	INT8, INT16, INT32, INT64
	Port Name	Direction (I/O)	Token RANGE														
	map_i	I	INT8, INT16, INT32, INT64														
	mapSize_i	I	UINT8, UINT16, UINT32, UINT64														
index_i	I	UINT8, UINT16, UINT32, UINT64															
dataOut_o	O	INT8, INT16, INT32, INT64															
LookUpTable1D Process Schematic (FSM):																	
START [INPUT]: mapSize_i Loop until is available: - mapSize [INPUT]: map_i [mapSize] Loop until is available: - map[mapSize] [INPUT]: index_i Loop until is available: - index [OUTPUT]: dataOut_o = map[index]																	
LookUpTable1D Process: START: INPUT: mapSize_i INPUT: map[mapSize] WHILE (true) INPUT: index_i dataOut=map[index_i] OUTPUT: dataOut_o																	
ISO Standards using the FU	ISO/IEC 14496-16:2011																
Profiles@levels supported																	
Parameter																	
Name	Description	Range															
NONE																	

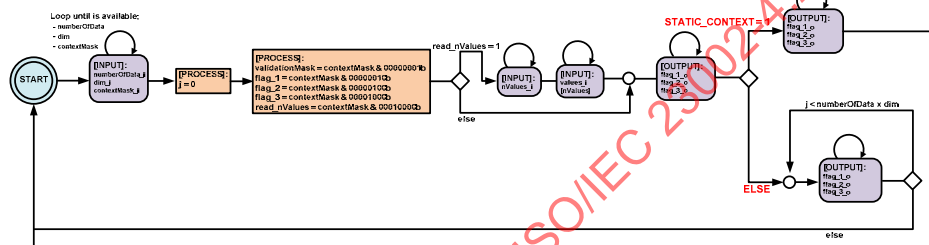
5.2.14 Algo_ContextModeling

FU Name	Algo_ContextModeling
Description	This FU describes context generation to be used in the entropy decoding units. It generates codec specific data sets based on the input values.



Port Name	Direction (I/O)	Token RANGE
numberOfData_i	I	UINT8, UINT16, UINT32, UINT64
dim_i	I	UINT8, UINT16, UINT32, UINT64
contextMask_i	I	UINT 8
nValues_i	I	UINT8, UINT16, UINT32, UINT64
values_i	I	INT8, INT16, INT32, INT64
validationMask_o	O	BOOLEAN
nValues_o	O	UINT8, UINT16, UINT32, UINT64
values_o	O	INT8, INT16, INT32, INT64
flag_1_o	O	BOOLEAN
flag_2_o	O	BOOLEAN
flag_3_o	O	BOOLEAN

ContextModeling Schematic (FSM):



ContextModeling process:

```

START
INPUT: numberOfData_i
INPUT: dim_i
INPUT: ccontextMask_i
j = 0
validationMask = contextMask & 00000001b
flag_1 = contextMask & 00000010b
flag_2 = contextMask & 00000100b
flag_3 = contextMask & 00001000b
read_nValues = contextMask & 00010000b
if read_nValues = 1
    INPUT: nValues_i
    INPUT: values_i [nValues]
    OUTPUT: validationMask_o
    OUTPUT: nValues_o
    OUTPUT: values_o [nValues]
    outputCounter = 0
    IF STATIC_CONTEXT = 1
        DO
            OUTPUT: flag_1_o
            OUTPUT: flag_2_o
            OUTPUT: flag_3_o
            outputCounter++
            WHILE outputCounter <= nValues
        ELSE
            DO
                outputCounter = 0
                DO
                    OUTPUT: flag_1_o
                    OUTPUT: flag_2_o
                    OUTPUT: flag_3_o
                    WHILE outputCounter <= nValues
                j = j + 1
                WHILE j < numberOfData * dim
            GOTO START

```

Bit position	7	6	5	4	3	2	1	0
	X	X	X	bit	bit	bit	bit	bit

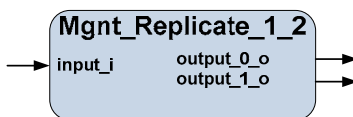
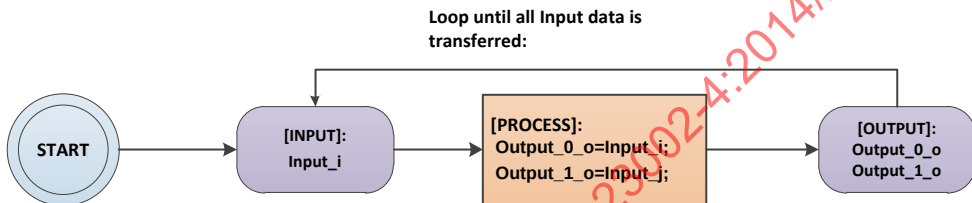
	Flag	X	X	X	Read N Values	flag_3	flag_3	flag_1	Validation Mask
ISO Standards using the FU	ISO/IEC 14496-16:2011								
Profiles@levels supported									
Parameter									

5.2.15 Algo_simpleMath_2op

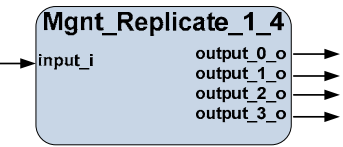
FU Name	Algo_simpleMath_2op																		
Description	This FU describes the simple mathematical operation on 2 operands. simpleMath_2op [FORMAT] [OPERATION] [STATIC_OP_2] op1_i → result_o op2_i → op2Out_o validate_i <table><thead><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr></thead><tbody><tr><td>op1_i</td><td>I</td><td>INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT</td></tr><tr><td>op2_i</td><td>I</td><td>INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT</td></tr><tr><td>validate_i</td><td>I</td><td>BOOLEAN</td></tr><tr><td>result_o</td><td>O</td><td>INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT</td></tr><tr><td>op2Out_o</td><td>O</td><td>INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT</td></tr></tbody></table> simpleMath_2op Schematic (FSM):	Port Name	Direction (I/O)	Token RANGE	op1_i	I	INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT	op2_i	I	INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT	validate_i	I	BOOLEAN	result_o	O	INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT	op2Out_o	O	INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT
	Port Name	Direction (I/O)	Token RANGE																
	op1_i	I	INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT																
op2_i	I	INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT																	
validate_i	I	BOOLEAN																	
result_o	O	INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT																	
op2Out_o	O	INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT																	
	simpleMath_2op process: <pre>START inputOp2 = 1 INPUT: validate_i op1_i IF STATIC_OP_2 = 0 INPUT: op2_i ELSE IF inputOp2 = 1 INPUT: op2_i inputOp2 = 0 IF validate = 1 & (OPERATION = 0 OPERATION = 1 OPERATION = 2 OPERATION = 3) FORMAT CONVERT: op1 FORMAT CONVERT: op2 SWITCH (OPERATION) CASE 0: result = op1 + op2 break CASE 1: result = op1 - op2 break CASE 2: result = op1 * op2 CASE 3: result = op1 / op2 ELSE result = op1 op2Out_o = op2</pre>																		

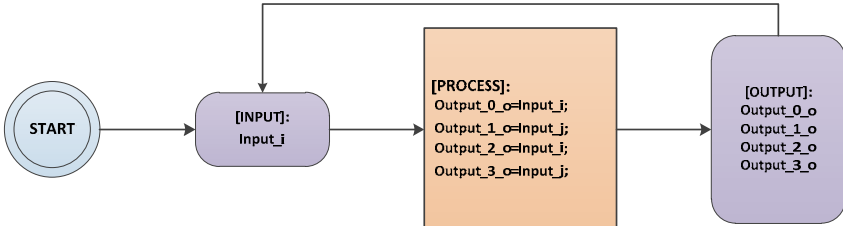
	<pre> break CASE 3: result = op1 / op2 break OUTPUT: result_o ELSE result = op1 op2Out = op2 OUTPUT: result_o OUTPUT: op2Out_o GOTO START </pre> The following table contains the interpretation of the FORMAT codes: <table border="1"> <thead> <tr> <th>FORMAT</th><th>Interpretation</th></tr> </thead> <tbody> <tr> <td>0</td><td>Unsigned Integer</td></tr> <tr> <td>1</td><td>Signed Integer</td></tr> <tr> <td>2</td><td>Fixed Point</td></tr> <tr> <td>3</td><td>Floating Point (IEEE 754)</td></tr> </tbody> </table> The following table contains the supported operations: <table border="1"> <thead> <tr> <th>Index</th><th>Operation</th></tr> </thead> <tbody> <tr> <td>0</td><td>Addition (+)</td></tr> <tr> <td>1</td><td>Substraction (-)</td></tr> <tr> <td>2</td><td>Multiplication (*)</td></tr> <tr> <td>3</td><td>Division (/)</td></tr> </tbody> </table> Note: If the validate_i flag is true (value 1), one of the operations above are applied to the 2 operands. Else, the operands are repeated at the output ports (result_o = op1 and op2Out_o = op2).	FORMAT	Interpretation	0	Unsigned Integer	1	Signed Integer	2	Fixed Point	3	Floating Point (IEEE 754)	Index	Operation	0	Addition (+)	1	Substraction (-)	2	Multiplication (*)	3	Division (/)
FORMAT	Interpretation																				
0	Unsigned Integer																				
1	Signed Integer																				
2	Fixed Point																				
3	Floating Point (IEEE 754)																				
Index	Operation																				
0	Addition (+)																				
1	Substraction (-)																				
2	Multiplication (*)																				
3	Division (/)																				
ISO Standards using the FU	ISO/IEC 14496-16:2011																				
Profiles@levels supported																					
Parameter																					
FORMAT	Describes the input operands number format representation. This parameter is set at the network configuration level. Type: Unsigned Integer Range: {0,1,2,3}																				
OPERATION	Describes the index of the operation to be applied. This parameter is set at the network configuration level. Type: Unsigned Integer Range: {0,1,2,3}																				
STATIC_OP_2	If it is set to 0, the FU expects that the input op2 will be read for each computation. If it is set to 1, the op 2 will be read just once and then used for any of the following operations. This parameter is set at the network configuration level. Type: Unsigned Integer Range: {0,1}																				

5.2.16 Mgnt_Replicate_1_2

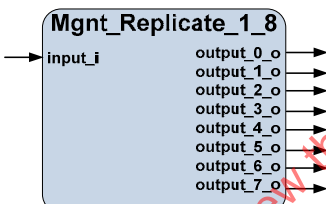
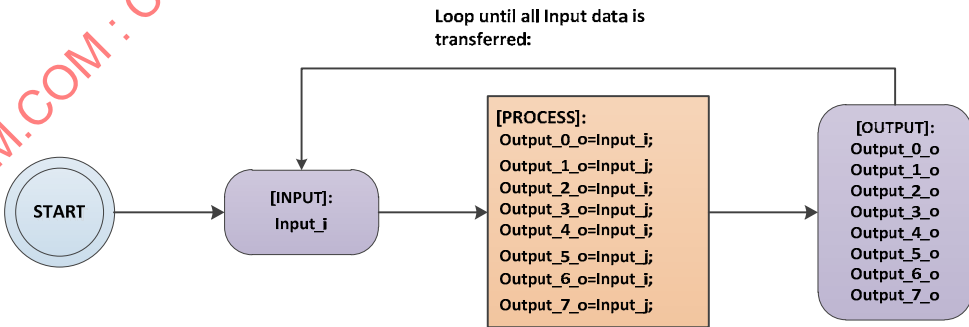
FU Name	Mgnt_Replicate_1_2	
Description	The output is generated by replicating the transferred input data. The detailed process is described in the above FSM. Block diagram:  Ports description Port Constraints: - the output ports have to be of the same data type as the input port	
	Process Schematic (FSM):  Loop until all Input data is transferred:	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	Description	Range

5.2.17 Mgnt_Replicate_1_4

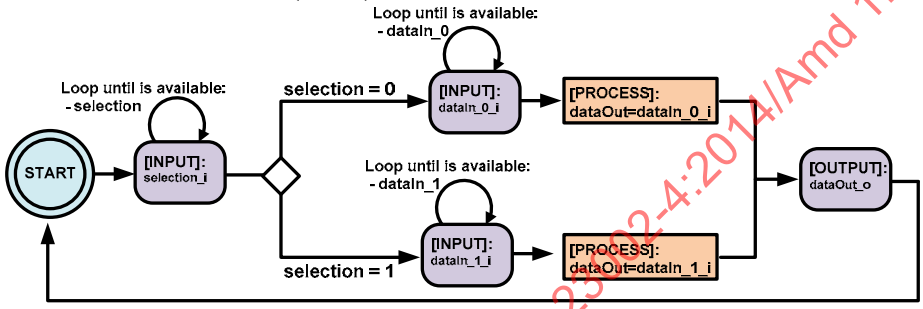
FU Name	Mgnt_Replicate_1_4	
Description	The output is generated by replicating the transferred input data. The detailed process is described in the above FSM. Block diagram:  Ports description Port Constraints: - the output ports have to be of the same data type as the input port Process Schematic (FSM):	

	 <pre>graph LR; START((START)) --> INPUT([INPUT: Input_i]); INPUT --> PROCESS[PROCESS: Output_0_o=Input_i; Output_1_o=Input_i; Output_2_o=Input_i; Output_3_o=Input_i;]; PROCESS --> OUTPUT([OUTPUT: Output_0_o Output_1_o Output_2_o Output_3_o]); OUTPUT -- "Loop until all Input data is transferred:" --> INPUT;</pre>	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	Description	Range

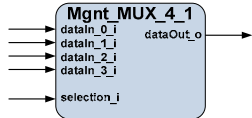
5.2.18 Mgnt_Replicate_1_8

FU Name	Mgnt_Replicate_1_8	
Description	The output is generated by replicating the transferred input data. The detailed process is described in the above FSM. Block diagram:  Ports description Port Constraints: - the output ports have to be of the same data type as the input port Process Schematic (FSM): 	
	ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported		
Parameter		
Name	Description	Range

5.2.19 Mgnt_MUX_2_1

FU Name	Mgnt_MUX_2_1
Description	 Port Constraints: - the dataIn ports have to be of the same data type - the dataOut port has to be of the same data type as the dataIn input ports - the selection port is of type bit. MUX_2_1 Schematic (FSM):  MUX_2_1 Process: <pre> START INPUT: selection SWITCH selection CASE 0: INPUT: dataIn_0_i dataOut = dataIn_0 CASE 1: INPUT: dataIn_1_i dataOut = dataIn_1 OUTPUT: dataOut_o GOTO START </pre>
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

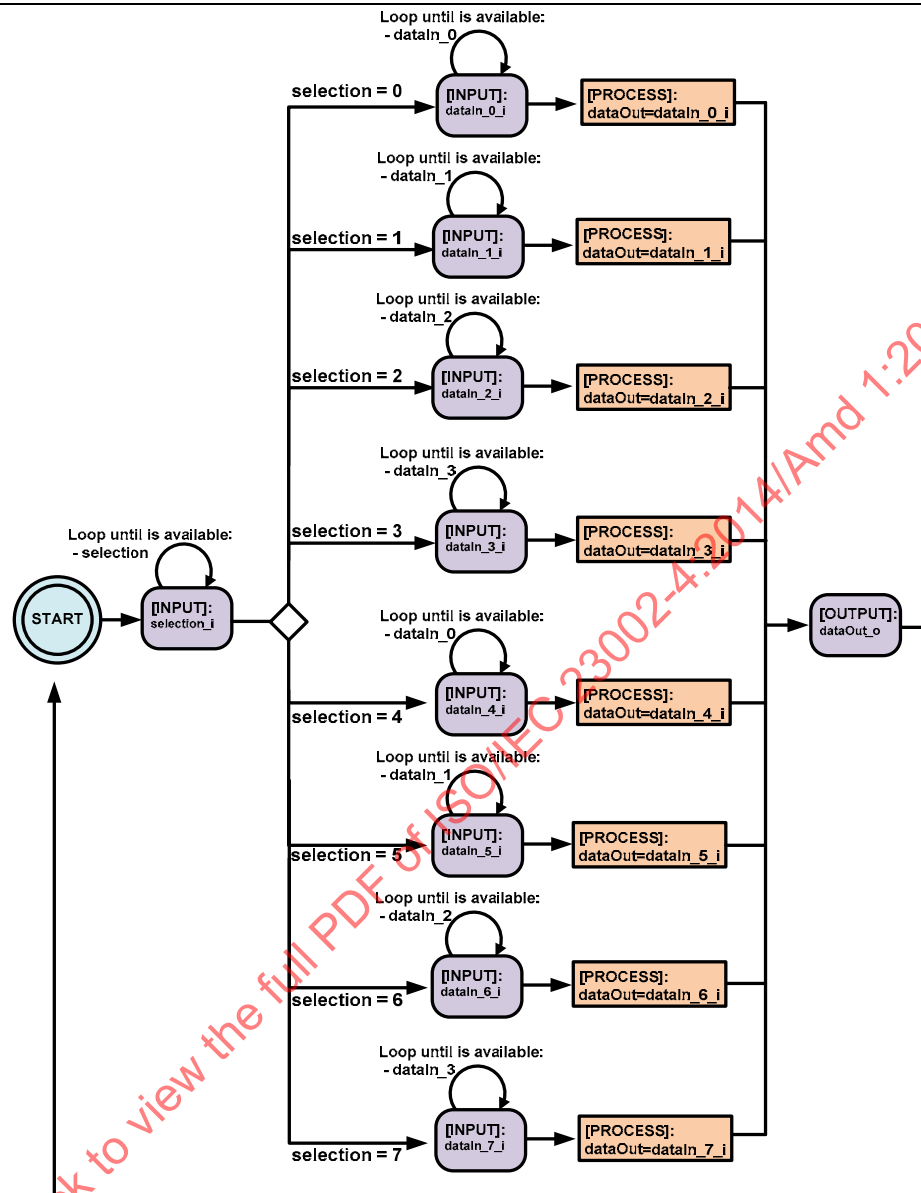
5.2.20 Mgnt_MUX_4_1

FU Name	Mgnt_MUX_4_1
Description	 Port Constraints: - the dataIn ports have to be of the same data type - the dataOut port has to be of the same data type as the dataIn input ports - the selection port is of type unsigned integer, represented on 2 bits. MUX_4_1 Schematic (FSM):

	The diagram shows a state machine for the MUX_4_1 process. It starts at a 'START' state, leading to an input state 'selection_i'. A loop 'Loop until is available: - selection' surrounds this input state. A decision diamond follows, branching into four paths based on 'selection' values: 0, 1, 2, and 3. Each path leads to an input state (dataIn_0_i to dataIn_3_i), each with its own 'Loop until is available' condition. These lead to process states (dataOut=dataIn_0_i to dataOut=dataIn_3_i), which then converge to an output state 'dataOut_o'. MUX_4_1 Process: <pre>START INPUT: selection SWITCH selection CASE 0: INPUT: dataIn_0_i dataOut = dataIn_0 CASE 1: INPUT: dataIn_1_i dataOut = dataIn_1 CASE 2: INPUT: dataIn_2_i dataOut = dataIn_2 CASE 3: INPUT: dataIn_3_i dataOut = dataIn_3 OUTPUT: dataOut_o GOTO START</pre>
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

5.2.21 Mgnt_MUX_8_1

FU Name	Mgnt_MUX_8_1
Description	The schematic shows a block 'Mgnt_MUX_8_1' with eight input ports labeled dataIn_0_i through dataIn_7_i and one output port dataOut_o. A selection port 'selection_i' is also shown. To the right, 'Port Constraints' are listed: dataIn ports must be the same data type; dataOut must be the same data type as the dataIn ports; selection_i must be an unsigned integer, 3 bits. MUX_8_1 Schematic (FSM):

**MUX_8_1 Process:**

START

INPUT: selection

SWITCH selection

CASE 0:

INPUT: dataIn_0_i
dataOut = dataIn_0

CASE 1:

INPUT: dataIn_1_i
dataOut = dataIn_1

CASE 2:

INPUT: dataIn_2_i
dataOut = dataIn_2

CASE 3:

INPUT: dataIn_3_i
dataOut = dataIn_3

CASE 4:

INPUT: dataIn_4_i
dataOut = dataIn_4

CASE 5:

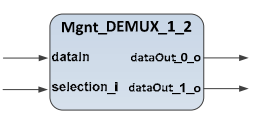
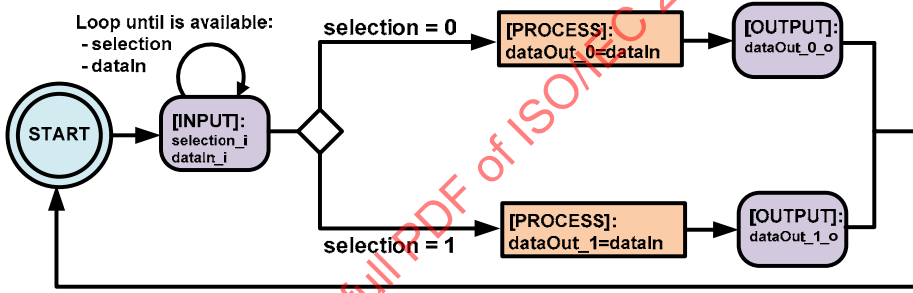
INPUT: dataIn_5_i
dataOut = dataIn_5

CASE 6:

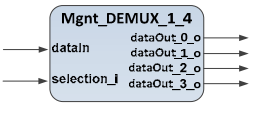
INPUT: dataIn_6_i
dataOut = dataIn_6

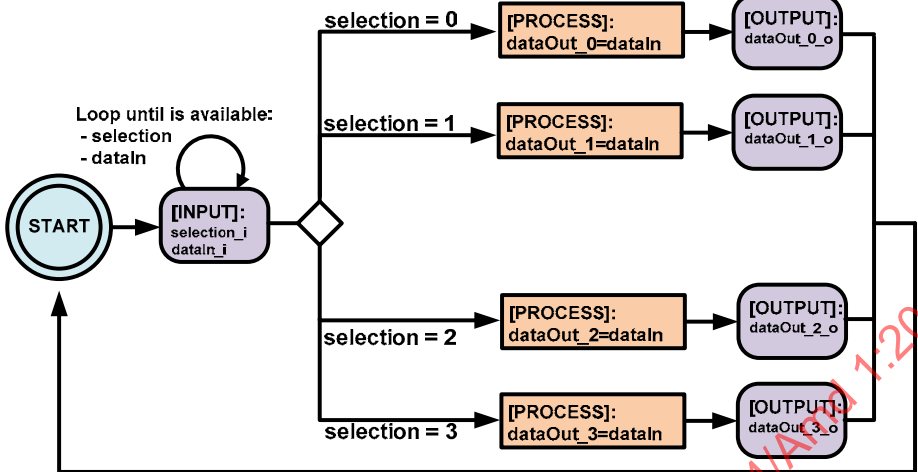
	CASE 7: INPUT: dataIn_7_i dataOut = dataIn_7 OUTPUT: dataOut_o GOTO START
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

5.2.22 Mgnt_DEMUX_1_2

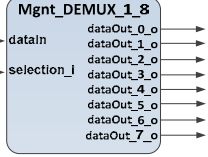
FU Name	Mgnt_DEMUX_1_2
Description	 Port Constraints: - the dataOut ports have to be of the same data type - the dataIn port has to be of the same data type as the dataOut ports - the selection port is of type bit. DEMUX_1_2 Schematic (FSM):  DEMUX_1_2 Process: <pre> START INPUT: selection INPUT: dataIn SWITCH selection CASE 0: dataOut_0 = dataIn OUTPUT: dataOut_0 CASE 1: dataOut_1 = dataIn OUTPUT: dataOut_1 GOTO START </pre>
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

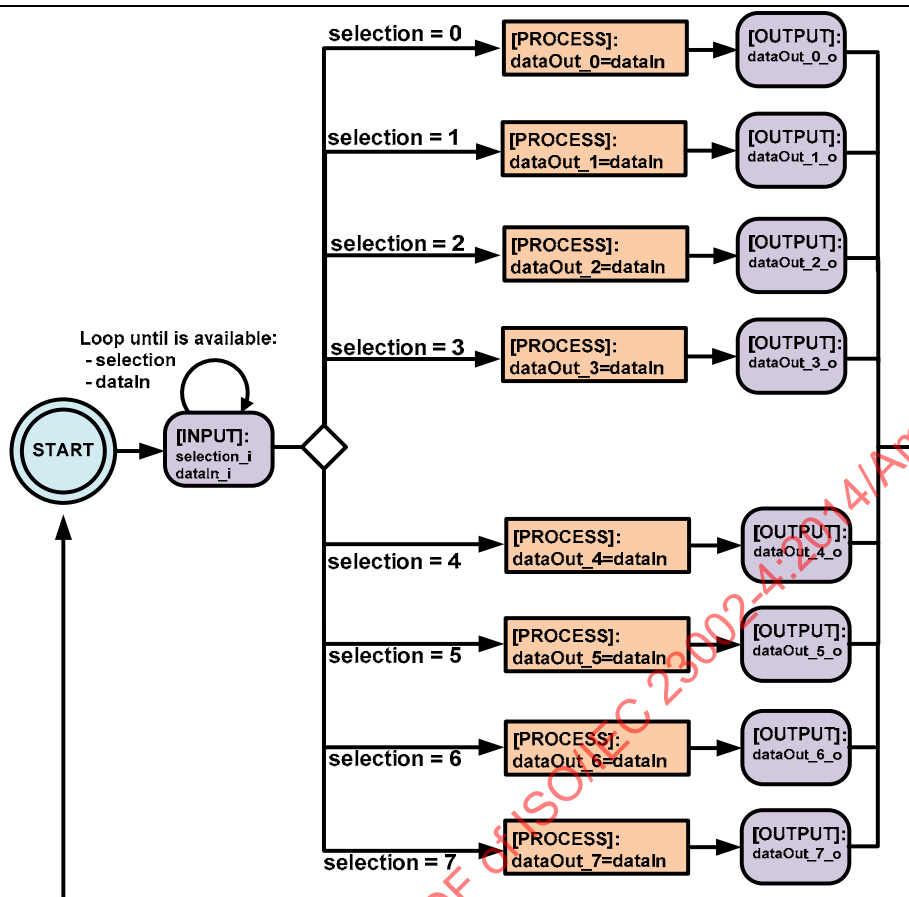
5.2.23 Mgnt_DEMUX_1_4

FU Name	Mgnt_DEMUX_1_4
Description	 Port Constraints: - the dataOut ports have to be of the same data type - the dataIn port has to be of the same data type as the dataOut ports - the selection port is of type unsigned integer represented on 2 bits. DEMUX_1_4 Schematic (FSM):

	 DEMUX_1_4 Process: <pre> START INPUT: selection INPUT: dataIn SWITCH selection CASE 0: dataOut_0 = dataIn OUTPUT: dataOut_0 CASE 1: dataOut_1 = dataIn OUTPUT: dataOut_1 CASE 2: dataOut_2 = dataIn OUTPUT: dataOut_2 CASE 3: dataOut_3 = dataIn OUTPUT: dataOut_3 GOTO START </pre>
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

5.2.24 Mgnt_DEMUX_1_8

FU Name	Mgnt_DEMUX_1_8
Description	 Port Constraints: - the dataOut ports have to be of the same data type - the dataIn port has to be of the same data type as the dataOut ports - the selection port is of type unsigned integer represented on 3 bits. DEMUX_1_8 Schematic (FSM):



DEMUX_1_8 Process:

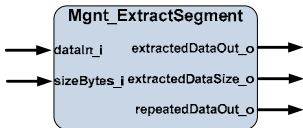
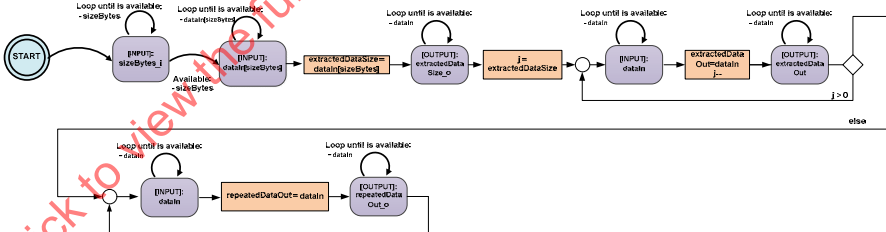
START
 INPUT: selection
 INPUT: dataIn
 SWITCH selection
 CASE 0:
 dataOut_0 = dataIn
 OUTPUT: dataOut_0
 CASE 1:
 dataOut_1 = dataIn
 OUTPUT: dataOut_1
 CASE 2:
 dataOut_2 = dataIn
 OUTPUT: dataOut_2
 CASE 3:
 dataOut_3 = dataIn
 OUTPUT: dataOut_3
 CASE 4:
 dataOut_4 = dataIn
 OUTPUT: dataOut_4
 CASE 5:
 dataOut_5 = dataIn
 OUTPUT: dataOut_5
 CASE 6:
 dataOut_6 = dataIn
 OUTPUT: dataOut_6
 CASE 7:
 dataOut_7 = dataIn
 OUTPUT: dataOut_7
 GOTO START

ISO Standards
using the FU

ISO/IEC 14496-16:2011

Profiles@levels
supported

5.2.25 Mgnt_ExtractSegment

FU Name	Mgnt_ExtractSegment																			
Description	This FU extracts a piece of data from the input of size embedded in the input data, outputs the size of the segment, the segment data and repeats the rest of the input on the output port. The size of the segment is embedded in the beginning of the input data and it is represented using a number of sizeBytes of type unsigned integer. The computed size is outputted as unsigned integer on the extractedDataSize_o output port, the extracted data segment is outputted on the extractedDataOut_o port and the rest of the input data is repeated on the repeatedDataOut_o port. Example: <table><tr><td>extractedDataSize</td><td>extractedData</td><td>repeatedData</td></tr><tr><td>size = sizeBytes</td><td>size = extractedDataSize</td><td>size = rest of input</td></tr></table>		extractedDataSize	extractedData	repeatedData	size = sizeBytes	size = extractedDataSize	size = rest of input												
	extractedDataSize	extractedData	repeatedData																	
	size = sizeBytes	size = extractedDataSize	size = rest of input																	
																				
	<table><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr><tr><td>dataIn_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>sizeBytes_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>extractedDataOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>extractedDataSize_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>repeatedDataOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr></table>			Port Name	Direction (I/O)	Token RANGE	dataIn_i	I	UINT8, UINT16, UINT32, UINT64	sizeBytes_i	I	UINT8, UINT16, UINT32, UINT64	extractedDataOut_o	O	UINT8, UINT16, UINT32, UINT64	extractedDataSize_o	O	UINT8, UINT16, UINT32, UINT64	repeatedDataOut_o	O
Port Name	Direction (I/O)	Token RANGE																		
dataIn_i	I	UINT8, UINT16, UINT32, UINT64																		
sizeBytes_i	I	UINT8, UINT16, UINT32, UINT64																		
extractedDataOut_o	O	UINT8, UINT16, UINT32, UINT64																		
extractedDataSize_o	O	UINT8, UINT16, UINT32, UINT64																		
repeatedDataOut_o	O	UINT8, UINT16, UINT32, UINT64																		
																				
ISO Standards using the FU	ISO/IEC 14496-16:2011																			
Profiles@levels supported																				
Parameter																				
Name	Description	Range																		

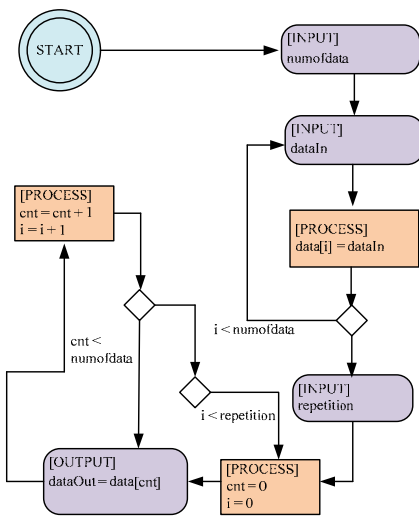
5.2.26 Mgnt_ProviderValue

FU Name	Mgnt_ProviderValue
Description	

	Mgnt_ProviderValue [PARAM] dataOut_o <table><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr><tr><td>dataOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr></table> ProviderValue Process Schematic START [PROCESS] dataOut = PARAM [OUTPUT] dataOut_o (FSM): ProviderValue Process: START: dataOut = PARAM OUTPUT: dataOut_o	Port Name	Direction (I/O)	Token RANGE	dataOut_o	O	UINT8, UINT16, UINT32, UINT64
Port Name	Direction (I/O)	Token RANGE					
dataOut_o	O	UINT8, UINT16, UINT32, UINT64					
ISO Standards using the FU	ISO/IEC 14496-16:2011						
Profiles@levels supported							
Parameter							
Name	Description	Range					
PARAM	Describes the value of the parameter that is outputted one time as dataOut. It is set at the network configuration level.						

5.2.27 Mgnt_RepeatSegment

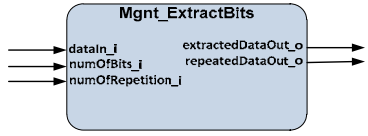
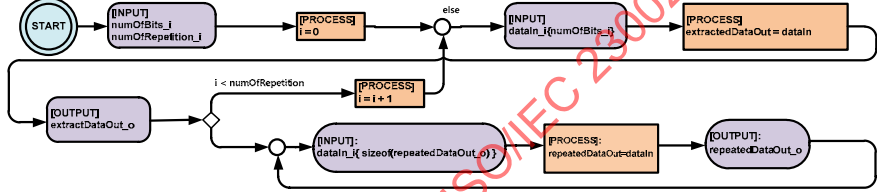
FU Name	Mgnt_RepeatSegment																
Description	This FU describes how to repetitively transfer the given data. <table border="1"> <thead> <tr> <th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr> </thead> <tbody> <tr> <td>dataIn_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr> <tr> <td>repetition_i</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr> <tr> <td>numofdata_i</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr> <tr> <td>dataOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr> </tbody> </table> RepeatSegment Process Schematic (FSM):		Port Name	Direction (I/O)	Token RANGE	dataIn_i	I	UINT8, UINT16, UINT32, UINT64	repetition_i	O	UINT8, UINT16, UINT32, UINT64	numofdata_i	O	UINT8, UINT16, UINT32, UINT64	dataOut_o	O	UINT8, UINT16, UINT32, UINT64
Port Name	Direction (I/O)	Token RANGE															
dataIn_i	I	UINT8, UINT16, UINT32, UINT64															
repetition_i	O	UINT8, UINT16, UINT32, UINT64															
numofdata_i	O	UINT8, UINT16, UINT32, UINT64															
dataOut_o	O	UINT8, UINT16, UINT32, UINT64															

	<pre>graph TD START((START)) --> INPUT1([INPUT numofdata]) INPUT1 --> INPUT2([INPUT dataIn]) INPUT2 --> PROCESS1([PROCESS data[i] = dataIn]) PROCESS1 --> COND1{ } COND1 -- "i < numofdata" --> INPUT2 COND1 -- "i < numofdata" --> COND2{ } COND2 -- "i < repetition" --> INPUT3([INPUT repetition]) INPUT3 --> PROCESS2([PROCESS cnt = 0 i = 0]) PROCESS2 --> OUTPUT([OUTPUT dataOut = data[cnt]]) OUTPUT --> COND3{ } COND3 -- "cnt < numofdata" --> PROCESS3([PROCESS cnt = cnt + 1 i = i + 1]) PROCESS3 --> COND3 COND3 -- "cnt < numofdata" --> OUTPUT</pre> RepeatSegment Process: START: INPUT: numofdata WHILE i < numofdata INPUT: dataIn data[i] = dataIn INPUT: repetition WHILE i < repetition WHILE cnt < numofdata dataOut = data[cnt] OUTPUT: dataOut	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	Description	Range

5.2.28 Mgnt_ExtractBytes

FU Name	Mgnt_ExtractBytes																		
Description	ExtractBytes dataIn_i sizeBytes_i extractedDataOut_o extractedDataSize_o repeatedDataOut_o <table><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr><tr><td>dataIn_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>sizeBytes_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>extractedDataOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>repeatedDataOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>extractedDataSize_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr></table> Mgnt_ExtractBytes Schematic: Mgnt_ExtractBytes Process: START: INPUT: sizeBytes_i extractedDataSize = sizeBytes OUTPUT: extractedDataSize_o i = 0 EXTRACT_DATA IF i < sizeBytes INPUT: dataIn_i extractedDataOut = dataIn; OUTPUT: extractedDataOut_o i = i + 1 GOTO EXTRACT_DATA REPEATED_DATA INPUT: dataIn_i repeatedDataOut = dataIn; OUTPUT: repeatedDataOut_o GOTO REPEATED_DATA	Port Name	Direction (I/O)	Token RANGE	dataIn_i	I	UINT8, UINT16, UINT32, UINT64	sizeBytes_i	I	UINT8, UINT16, UINT32, UINT64	extractedDataOut_o	O	UINT8, UINT16, UINT32, UINT64	repeatedDataOut_o	O	UINT8, UINT16, UINT32, UINT64	extractedDataSize_o	O	UINT8, UINT16, UINT32, UINT64
	Port Name	Direction (I/O)	Token RANGE																
	dataIn_i	I	UINT8, UINT16, UINT32, UINT64																
	sizeBytes_i	I	UINT8, UINT16, UINT32, UINT64																
extractedDataOut_o	O	UINT8, UINT16, UINT32, UINT64																	
repeatedDataOut_o	O	UINT8, UINT16, UINT32, UINT64																	
extractedDataSize_o	O	UINT8, UINT16, UINT32, UINT64																	
ISO Standards using the FU	ISO/IEC 14496-16:2011																		
Profiles@levels supported																			

5.2.29 Mgnt_ExtractBits

FU Name	Mgnt_ExtractBits																		
Description	This FU describes how to repetitively extract a given number of bits and output the values, sequentially. The rest of the input data is repeated at the output port. Internally, the bit alignment is done after finishing the bits extraction operation.  <table><thead><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr></thead><tbody><tr><td>dataIn_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>numOfBits_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>numOfRepetition_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>extractedValuesOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>repeatedDataOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr></tbody></table> ExtractBits Process Schematic (FSM):  Ext	Port Name	Direction (I/O)	Token RANGE	dataIn_i	I	UINT8, UINT16, UINT32, UINT64	numOfBits_i	I	UINT8, UINT16, UINT32, UINT64	numOfRepetition_i	I	UINT8, UINT16, UINT32, UINT64	extractedValuesOut_o	O	UINT8, UINT16, UINT32, UINT64	repeatedDataOut_o	O	UINT8, UINT16, UINT32, UINT64
	Port Name	Direction (I/O)	Token RANGE																
	dataIn_i	I	UINT8, UINT16, UINT32, UINT64																
	numOfBits_i	I	UINT8, UINT16, UINT32, UINT64																
	numOfRepetition_i	I	UINT8, UINT16, UINT32, UINT64																
extractedValuesOut_o	O	UINT8, UINT16, UINT32, UINT64																	
repeatedDataOut_o	O	UINT8, UINT16, UINT32, UINT64																	
	ExtractBits Process: START: INPUT: numOfBits_i numOfRepetition_i i = 0 EXTRACT BITS INPUT: dataIn_i{numOfBits} extractedDataOut = dataIn OUTPUT: extractedDataOut_o IF i < numOfRepetition i++ GOTO EXTRACT BITS REPEATED_DATA_OUT INPUT: dataIn_i{ sizeof (repeatedDataOut) } repeatedDataOut = dataIn OUTPUT: repeatedDataOut_o GOTO REPEATED_DATA_OUT Note: The ‘sizeof’ operation is used to obtain the size of the token in bits (bus width). The way to use the ‘sizeof’ operation is as follows: sizeof(name of port)																		
ISO Standards using the FU	ISO/IEC 14496-16:2011																		
Profiles@level supported																			
Parameter																			

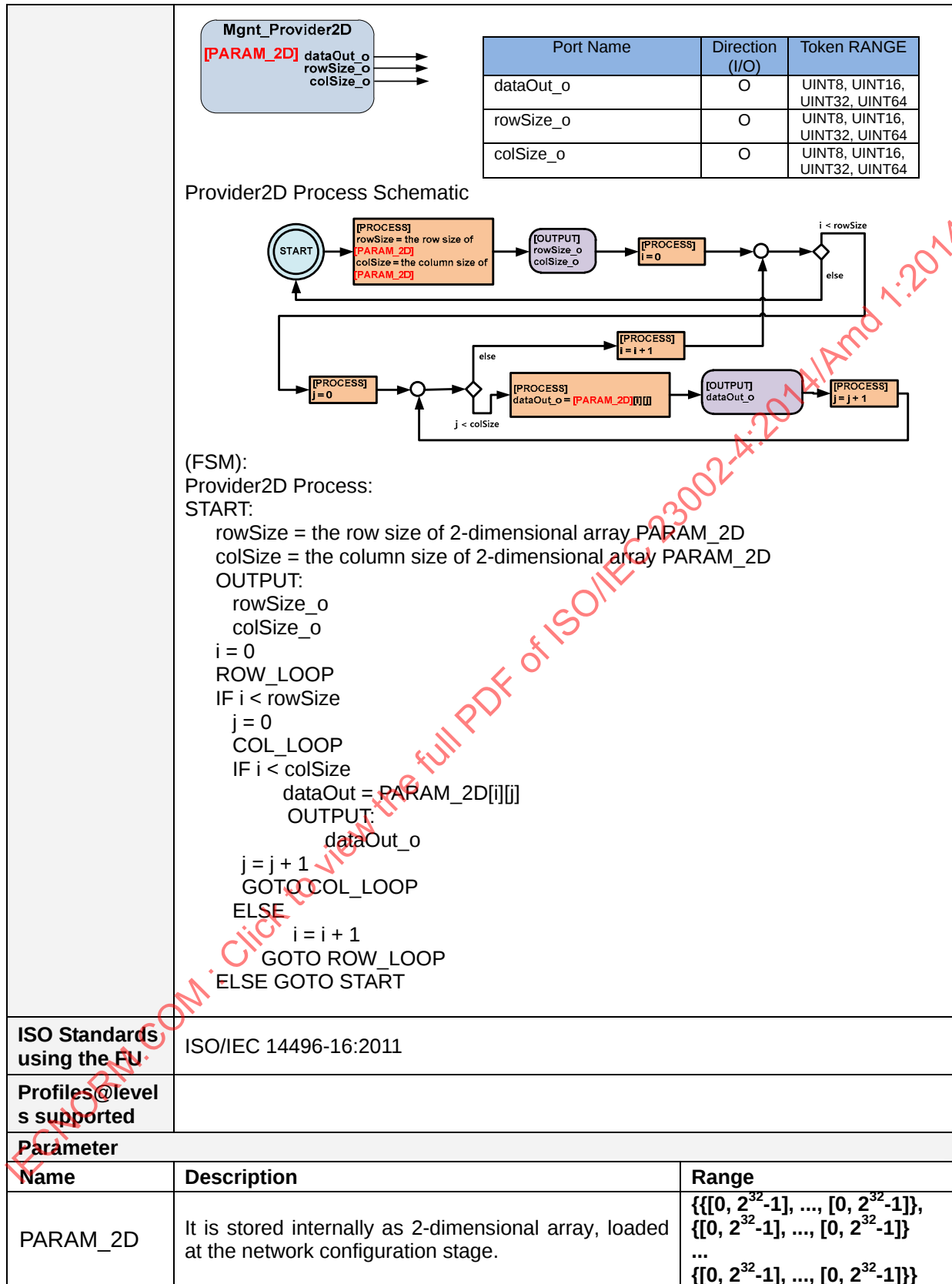
Name	Description	Range

5.2.30 Mgnt_Provider1D

FU Name	Mgnt_Provider1D		
Description	This FU describes how to provide 1D array values from FU network description.		
	Mgnt_Provider1D [PARAM_1D] dataOut_o dataSize_o Port Name Direction (I/O) Token RANGE dataOut_o O UINT8, UINT16, UINT32, UINT64 dataSize_o O UINT8, UINT16, UINT32, UINT64		
	Provider1D Process Schematic (FSM):		
	START [PROCESS] dataSize = the size of array PARAM_1D [OUTPUT] dataSize_o [PROCESS] i = 0 i < dataSize else [PROCESS] dataOut = PARAM_1D[i] [OUTPUT] dataOut_o [PROCESS] i = i + 1		
	Provider1D Process: START: dataSize = the size of array PARAM_1D OUTPUT: dataSize_o i = 0 // READ 1D TABLE IF i < dataSize dataOut = PARAM_1D[i] OUTPUT: dataOut_o i = i + 1 GOTO READ 1D TABLE		
ISO Standards using the FU	ISO/IEC 14496-16:2011		
Profiles@levels supported			
Parameter			
Name	Description	Range	
PARAM_1D	It is stored internally as an array, loaded at the network configuration stage.	{[0, 2 ³² -1], ..., [0, 2 ³² -1]}	

5.2.31 Mgnt_Provider2D

FU Name	Mgnt_Provider2D
Description	This FU describes how to provide 2D values in the row-base manner.



Add section 9. MPEG-4 Part 16 SC3DMC Decoder Specific FUs

9 MPEG-4 Part 16 SC3DMC Decoder Specific FUs

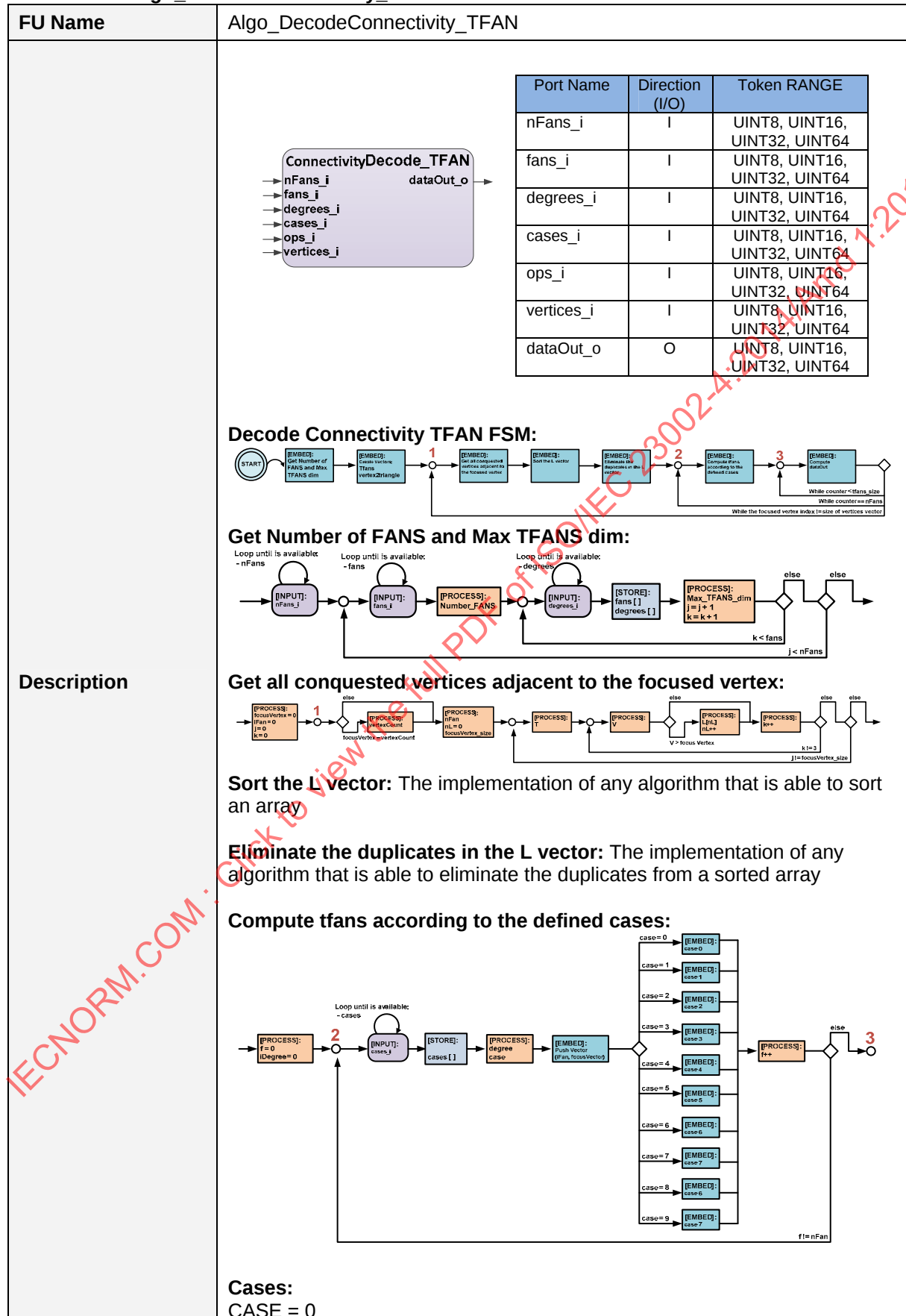
The specific FUs for building MPEG-4 Part 16 SC3DMC decoder are described in this sub-clause

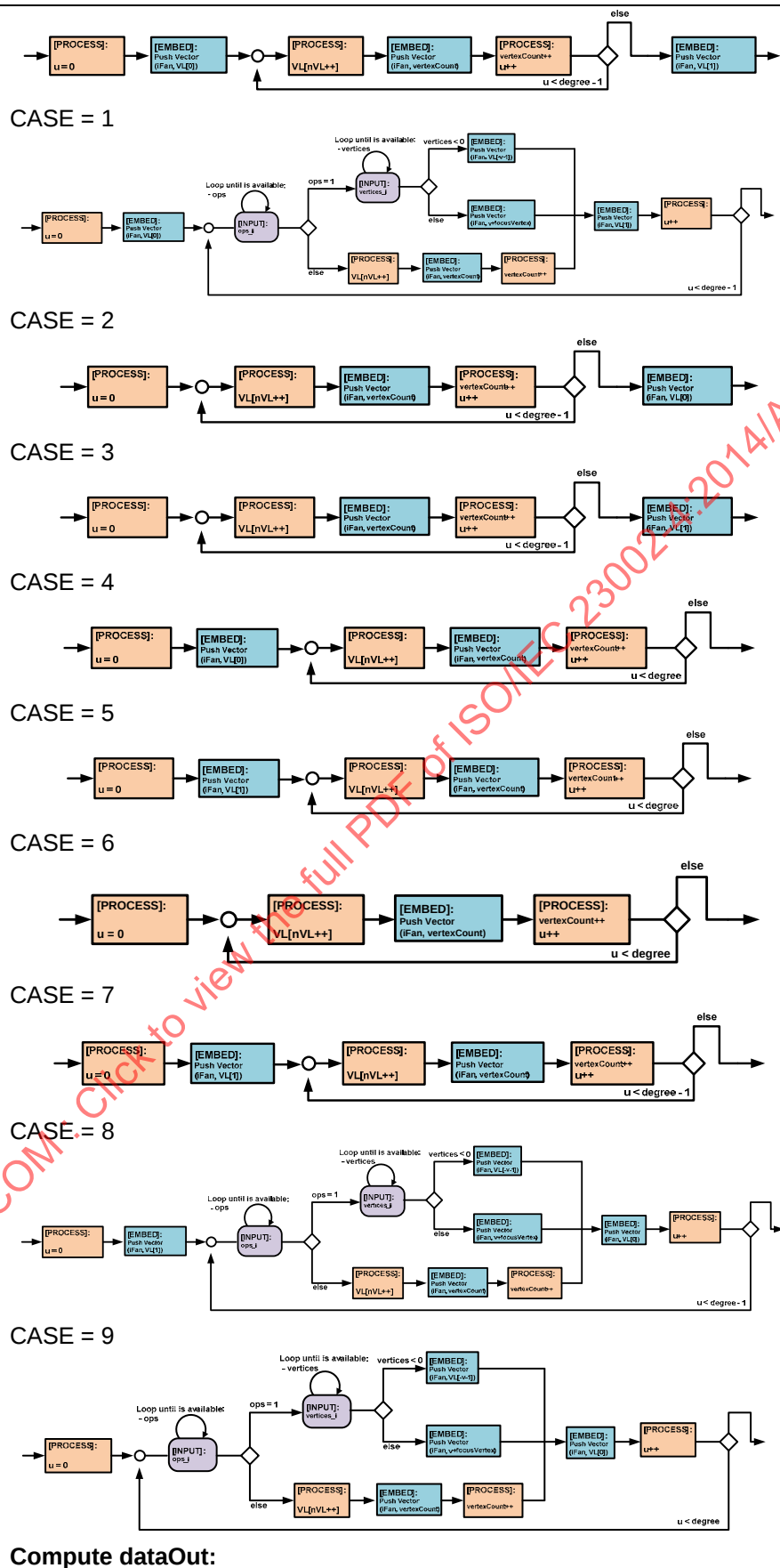
9.1 Algo_ExtractMask_SC3DMC

FU Name	Algo_ExtractMask_SC3DMC
Description	ExtractMask_SC3DMC dataIn_i validation_i <

9.2 MPEG-4 SC3DMC TFAN Specific FUS

9.2.1 Algo_DecodeConnectivity_TFAN






```

    EMBED: tFan Push Vector (iFan,vertexCount)
    u++
    EMBED: tFan Push Vector (iFan,VL[1])

    EMBED: CASE 1
    u=0
    iOps=0
    iVertices=0
    EMBED: tFan Push Vector (iFan,VL[0])
    WHILE u != degree-1
        INPUT: ops[]
        IF ops[iOps] = 1
            INPUT: vertices[]
            v = vertices[iVertices]
            IF v < 0
                EMBED: tFan Push Vector (iFan,VL[-v-1])
            ELSE
                EMBED: tFan Push Vector (iFan,v+focusVertex)
        ELSE
            VL[nVL++]=vertexCount
            EMBED: tFan Push Vector (iFan,vertexCount)
            nVL++
            vertexCount++
        u++
    EMBED: tFan Push Vector (iFan,VL[1])

    EMBED: CASE 2
    u=0
    WHILE u != degree-1
        VL[nVL]=vertexCount
        EMBED: tFan Push Vector (iFan,vertexCount)
        nVL++
        vertexCount++
        u++
    EMBED: tFan Push Vector (iFan,VL[0])

    EMBED: CASE 3
    u=0
    WHILE u != degree-1
        VL[nVL]=vertexCount
        EMBED: tFan Push Vector (iFan,vertexCount)
        nVL++
        vertexCount++
        u++
    EMBED: tFan Push Vector (iFan,VL[1])

    EMBED: CASE 4
    EMBED: tFan Push Vector (iFan,VL[0])
    u=0
    WHILE u != degree
        VL[nVL]=vertexCount
        EMBED: tFan Push Vector (iFan,vertexCount)
        nVL++
        vertexCount++
        u++

    EMBED: CASE 5
    EMBED: tFan Push Vector (iFan,VL[1])
    u=0
    WHILE u != degree
        VL[nVL]=vertexCount
        EMBED: tFan Push Vector (iFan,vertexCount)
        nVL++
        vertexCount++
        u++

    EMBED: CASE 6
    u=0
    WHILE u != degree
        VL[nVL]=vertexCount
        EMBED: tFan Push Vector (iFan,vertexCount)
        nVL++
        vertexCount++
        u++

```

	<pre> EMBED: CASE 7 EMBED: tFan Push Vector (iFan,VL[1]) u=0 WHILE u != degree-1 VL[nVL]=vertexCount EMBED: tFan Push Vector (iFan,vertexCount) nVL++ vertexCount++ u++ EMBED: tFan Push Vector (iFan,VL[0]) EMBED: CASE 8 u=0 iOps=0 iVertices=0 EMBED: tFan Push Vector (iFan,VL[1]) WHILE u != degree-1 INPUT: ops[] IF ops[iOps] = 1 INPUT: vertices[] v = vertices[iVertices] IF v < 0 EMBED: tFan Push Vector (iFan,VL[-v-1]) ELSE EMBED: tFan Push Vector (iFan,v+focusVertex) ELSE VL[nVL++]=vertexCount EMBED: tFan Push Vector (iFan,vertexCount) nVL++ vertexCount++ u++ EMBED: tFan Push Vector (iFan,VL[0]) EMBED: CASE 9 u=0 WHILE u != degree INPUT: ops[] IF ops[iOps] = 1 INPUT: vertices[] v = vertices[iVertices] IF v < 0 EMBED: tFan Push Vector (iFan,VL[-v-1]) ELSE EMBED: tFan Push Vector (iFan,v+focusVertex) ELSE VL[nVL++]=vertexCount EMBED: tFan Push Vector (iFan,vertexCount) nVL++ vertexCount++ u++ EMBED: Compute dataOut k=2 dataCount=0 b=EMBED: tFan Get Vector (iFan,1) fansSize=EMBED: tFan Get Vector (iFan) WHILE k < fansSize c=EMBED: tFan Get Vector (iFan,k) dataOut[0]=focusVertex dataOut[1]=b dataOut[2]=c EMBED:Vertex2Triangle Push Vector(focusVertex,triangleCount) IF b != focusVertex EMBED:Vertex2Triangle Push Vector(b,triangleCount) IF c != focusVertex & c!= b EMBED:Vertex2Triangle Push Vector(c,triangleCount) triangleCount++ b=c k++ OUTPUT: dataOut[] OUTPUT: dataSize </pre>
ISO Standards using the FU	ISO/IEC 14496-16:2011

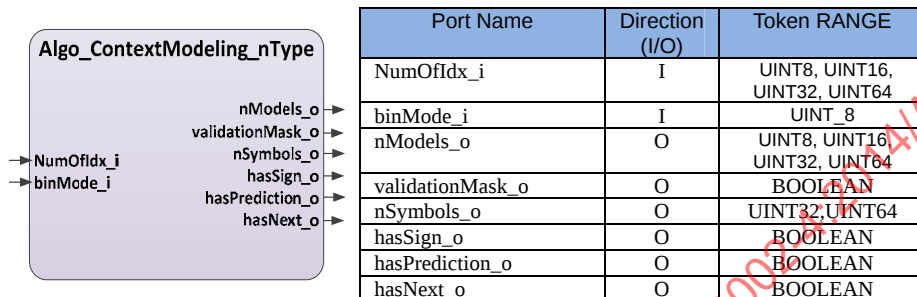
Profiles@levels
supported

9.3 MPEG-4 SC3DMC SVA Specific FUs

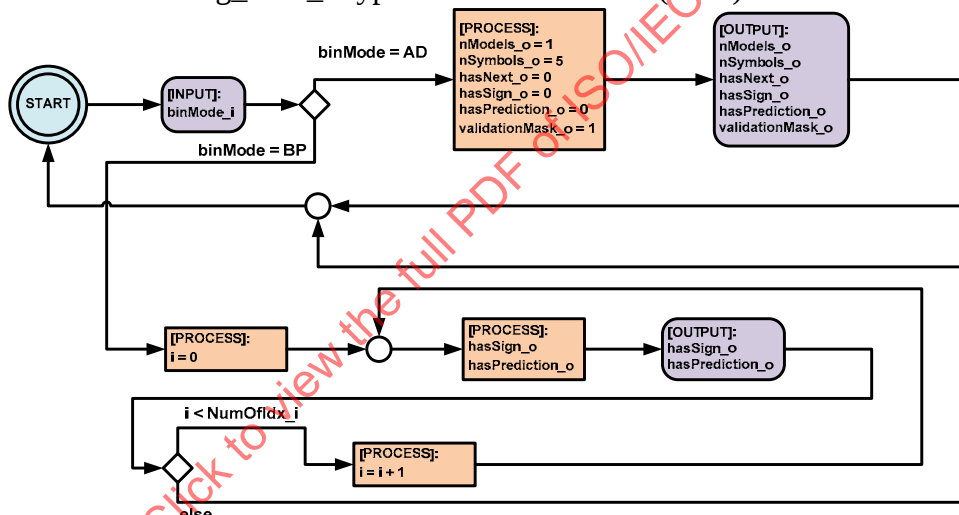
9.3.1 Algo_ContextModeling_SVA_nType

FU Name Algo_ContextModeling_SVA_nType

This FU describes how to generate the context model of 'nType' in the arithmetic decoding process.



ContextModeling_SVA_nType Process Schematic (FSM):



Description

ContextModeling_SVA_nType Process:

```

START
INPUT:
  numOfIdx_i
  binMode_i
IF binMode = AD
  nModels = 1
  nSymbols = 5
  hasNext = 0
  hasSign = 0
  hasPrediction = 0
  validationMask = 1
OUTPUT:
  nModels_o
  nSymbols_o
  hasNext_o
  hasSign_o
  hasPrediction_o
  validationMask_o
GOTO START

```

```

ELSE IF binMode = BP
  i = 0

```

	BP_PROCESS: hasSign = 0 hasPrediction = 0 OUTPUT: hasSign_o hasPrediction_o IF i < numOfIdx i++ GOTO BP_PROCESS ELSE GOTO START	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	Description	Range

9.3.2 Algo_ContextModeling_SVA_Indexes

FU Name	Algo_ContextModeling_SVA_Indexes																																				
Description	This FU describes how to generate context models of ‘connectivity’ depending on the nType and the binarization mode.																																				
	ContextModeling_SVA_INDEXES nType_i numOfIdx_i fdMode_i binMode_i nModels_o validationMask_o nSymbols_o hasSign_o hasPrediction_o hasNext_o modelIndex_o <table><thead><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr></thead><tbody><tr><td>nType_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>numOfIdx_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>fdMode_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>binMode_i</td><td>I</td><td>UINT_8</td></tr><tr><td>nModels_o</td><td>O</td><td>UINT_8</td></tr><tr><td>validationMask_o</td><td>O</td><td>BOOLEAN</td></tr><tr><td>nSymbols_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>hasSign_o</td><td>O</td><td>BOOLEAN</td></tr><tr><td>hasPrediction_o</td><td>O</td><td>BOOLEAN</td></tr><tr><td>hasNext_o</td><td>O</td><td>BOOLEAN</td></tr><tr><td>modelIndex_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr></tbody></table>	Port Name	Direction (I/O)	Token RANGE	nType_i	I	UINT8, UINT16, UINT32, UINT64	numOfIdx_i	I	UINT8, UINT16, UINT32, UINT64	fdMode_i	I	UINT8, UINT16, UINT32, UINT64	binMode_i	I	UINT_8	nModels_o	O	UINT_8	validationMask_o	O	BOOLEAN	nSymbols_o	O	UINT8, UINT16, UINT32, UINT64	hasSign_o	O	BOOLEAN	hasPrediction_o	O	BOOLEAN	hasNext_o	O	BOOLEAN	modelIndex_o	O	UINT8, UINT16, UINT32, UINT64
	Port Name	Direction (I/O)	Token RANGE																																		
nType_i	I	UINT8, UINT16, UINT32, UINT64																																			
numOfIdx_i	I	UINT8, UINT16, UINT32, UINT64																																			
fdMode_i	I	UINT8, UINT16, UINT32, UINT64																																			
binMode_i	I	UINT_8																																			
nModels_o	O	UINT_8																																			
validationMask_o	O	BOOLEAN																																			
nSymbols_o	O	UINT8, UINT16, UINT32, UINT64																																			
hasSign_o	O	BOOLEAN																																			
hasPrediction_o	O	BOOLEAN																																			
hasNext_o	O	BOOLEAN																																			
modelIndex_o	O	UINT8, UINT16, UINT32, UINT64																																			
	ContextModeling_SVA_Indexes Process Schematic (FSM): ContextModeling_SVA_Indexes Process: <pre>START INPUT: binMode_i IF binMode = AD nModels = 4 nSymbols = {3, 2, 1024, 3} hasNext = {0, 0, 1, 0} hasSign = {0, 0, 1, 0} hasPrediction = {0, 0, 0, 0} delimTableSize = 4 delimTable = {0, 2¹⁰, 2²⁰, 2³⁰} OUTPUT: nModels nSymbols hasNext hasSign hasPrediction delimTableSize delimTable i = 0 AD VALIDATION IF i==0 nType = 1 ELSE INPUT: nType_i IF nType = 0 validationMask = {1,1,1,1,0,0,0,0,0,0,0,0}</pre>																																				

```

ELSE IF nType = 1
    validationMask = {0,0,1,0,0,0,1,0,0,0,1,0}
ELSE IF nType = 2
    validationMask = {1,0,1,0,0,0,1,1,0,0,0,0}
ELSE IF nType = 3
    validationMask = {0,0,1,0,0,0,1,0,0,0,1,0}
ELSE IF nType = 4
    validationMask = {0,1,0,1,0,0,0,0,0,0,0,0}
OUTPUT:
    validationMask_o
    IF i < numOfIdx
        i = i + 1
        GOTO AD VALIDATION
    ELSE GOTO START

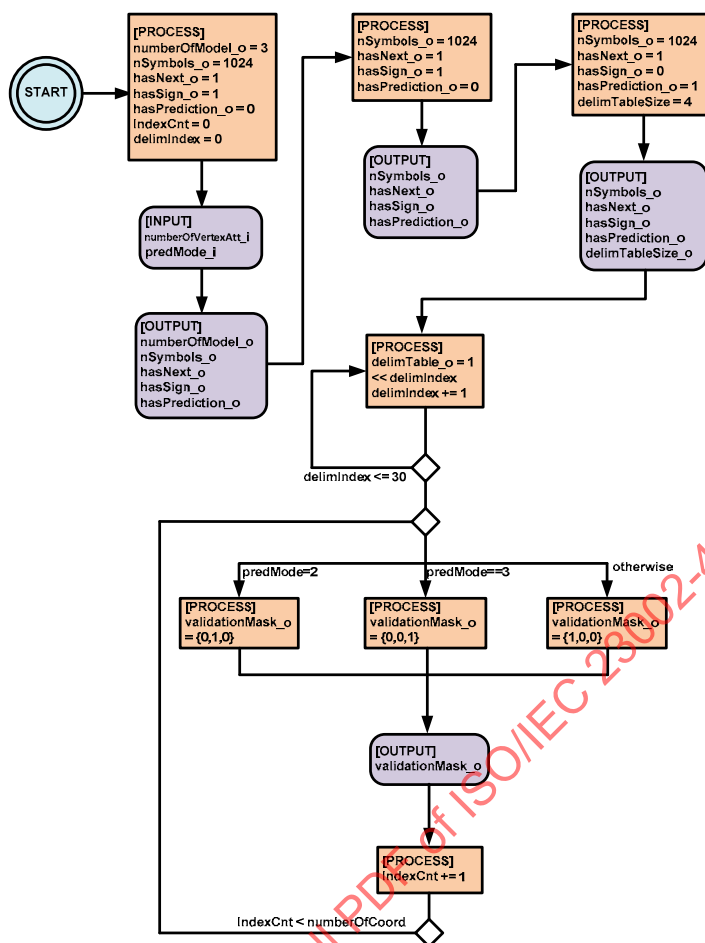
ELSE IF binMode = BP
    i = 0
    BP PROCESS
    INPUT:
        FDMode_i
        numOfIdx_i
    IF i = 0
        nType = 1
    ELSE
        INPUT: nType_i
    IF FDMode = 0
        IF nType = 0
            nModels = 3
            hasSign = {0,1,0}
            hasPrediction = {0,1,0}
            modelIndex = {0,2,3}
        ELSE IF nType = 1
            nModels = 3
            hasSign = {1,1,1}
            hasPrediction = {1,1,1}
            modelIndex = {2,2,2}
        ELSE IF nType = 2
            nModels = 4
            hasSign = {0,1,1,0}
            hasPrediction = {0,1,1,0}
            modelIndex = {0,2,2,3}
        ELSE IF nType = 3
            nModels = 3
            hasSign = {1,1,1}
            hasPrediction = {1,1,1}
            modelIndex = {2,2,2}
        ELSE IF nType = 4
            nModels = 1
            hasSign = {0}
            hasPrediction = {0}
            modelIndex = {3}
    ELSE IF FDMode = 1
        IF nType = 0
            nModels = 4
            hasSign = {0,0,1,0}
            hasPrediction = {0,0,1,0}
            modelIndex = {0,1,2,3}
        ELSE IF nType = 1
            nModels = 3
            hasSign = {1,1,1}
            hasPrediction = {1,1,1}
            modelIndex = {2,2,2}
        ELSE IF nType = 2
            nModels = 4
            hasSign = {0,1,1,0}
            hasPrediction = {0,1,1,0}
            modelIndex = {0,2,2,3}
        ELSE IF nType = 3
            nModels = 3
            hasSign = {1,1,1}
            hasPrediction = {1,1,1}
            modelIndex = {2,2,2}
        ELSE IF nType = 4
            nModels = 2
            hasSign = {0,0}

```

	<pre>hasPrediction = {0,0} modelIndex = {1, 3} OUTPUT: nModels_o hasSign_o hasPrediction_o modelIndex_o IF i < numOfIdx i = i + 1 GOTO BP PROCESS ELSE GOTO START</pre>	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	Description	Range

9.3.3 Algo_ContextModeling_SVA_Vertex_Attribute

FU Name	Algo_ContextModeling_SVA_Vertex_Attribute																																	
Description	This FU describes how to generate the context model of 'vertex attribute' such as coordinate, normal, color, etc, in the arithmetic decoding process. Algo_ContextModeling_SVA_VERTEX_ATTRIBUTE predMode_i numberOfModel_o validationMask_o numberOfVertexAtt_i nSymbols_o hasSign_o hasPrediction_o hasNext_o delimTable_o delimTableSize_o → → → → → → → → → → <table><thead><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr></thead><tbody><tr><td>predMode_i</td><td>I</td><td>UINT_4</td></tr><tr><td>numberOfCoord_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>numberOfModel_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>validationMask_o</td><td>O</td><td>BOOLEAN</td></tr><tr><td>nSymbols_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>hasSign_o</td><td>O</td><td>BOOLEAN</td></tr><tr><td>hasPrediction_o</td><td>O</td><td>BOOLEAN</td></tr><tr><td>hasNext_o</td><td>O</td><td>BOOLEAN</td></tr><tr><td>delimTable_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>delimTableSize_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr></tbody></table>	Port Name	Direction (I/O)	Token RANGE	predMode_i	I	UINT_4	numberOfCoord_i	I	UINT8, UINT16, UINT32, UINT64	numberOfModel_o	O	UINT8, UINT16, UINT32, UINT64	validationMask_o	O	BOOLEAN	nSymbols_o	O	UINT8, UINT16, UINT32, UINT64	hasSign_o	O	BOOLEAN	hasPrediction_o	O	BOOLEAN	hasNext_o	O	BOOLEAN	delimTable_o	O	UINT8, UINT16, UINT32, UINT64	delimTableSize_o	O	UINT8, UINT16, UINT32, UINT64
	Port Name	Direction (I/O)	Token RANGE																															
predMode_i	I	UINT_4																																
numberOfCoord_i	I	UINT8, UINT16, UINT32, UINT64																																
numberOfModel_o	O	UINT8, UINT16, UINT32, UINT64																																
validationMask_o	O	BOOLEAN																																
nSymbols_o	O	UINT8, UINT16, UINT32, UINT64																																
hasSign_o	O	BOOLEAN																																
hasPrediction_o	O	BOOLEAN																																
hasNext_o	O	BOOLEAN																																
delimTable_o	O	UINT8, UINT16, UINT32, UINT64																																
delimTableSize_o	O	UINT8, UINT16, UINT32, UINT64																																
ContextModeling_SVA_Vertex_Attribute Process Schematic (FSM):																																		



ContextModeling_SVA_Vertex_Attribute Process:

numberOfModel_o = 3
IndexCnt = 0

// Data type 1 symbol
nSymbols_o = 1024
hasNext_o = 1
hasSign_o = 1
hasPrediction_o = 0

// Data type 2 symbol
nSymbols_o = 1024
hasNext_o = 1
hasSign_o = 0
hasPrediction_o = 0

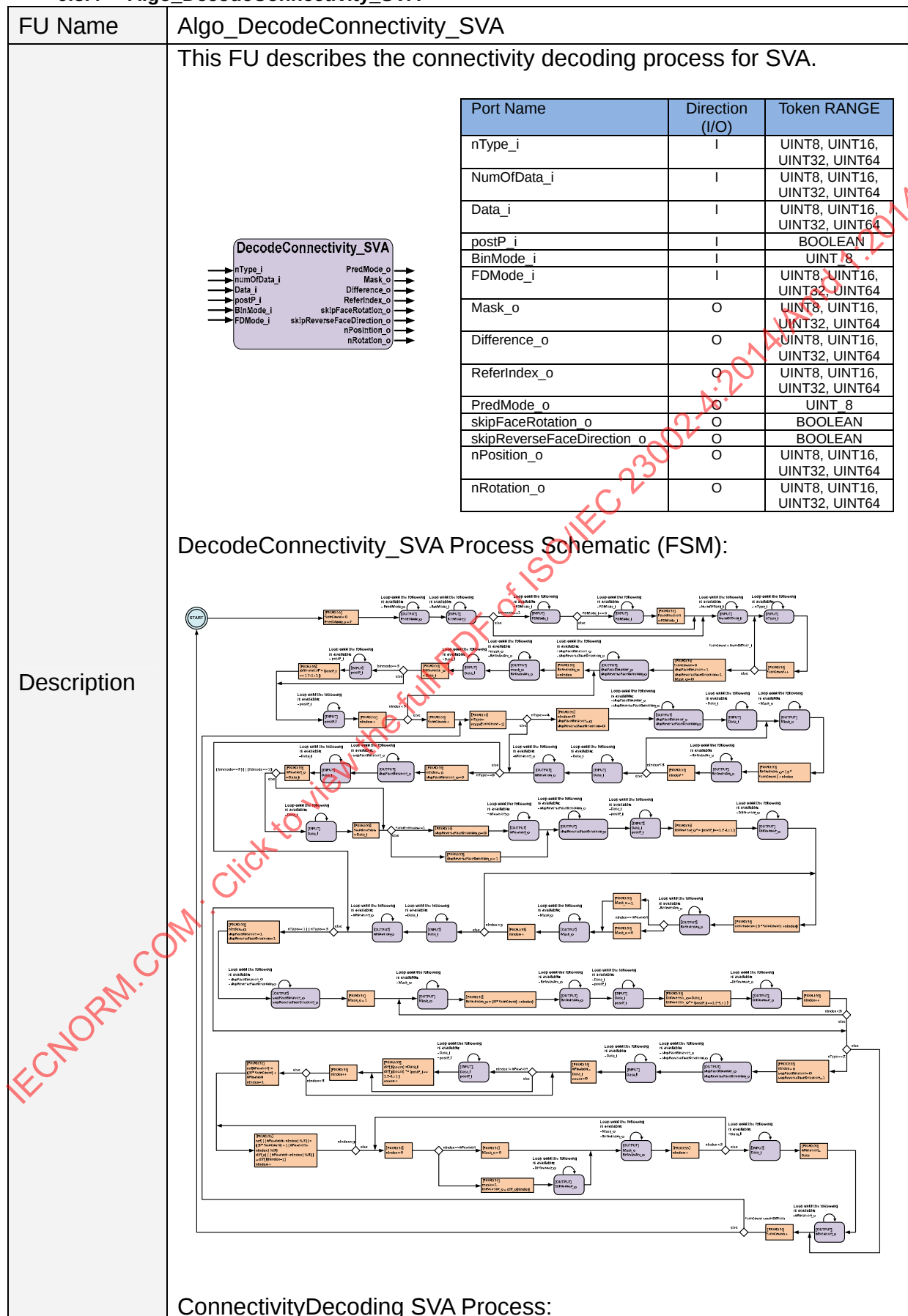
// Data type 3 symbol
nSymbols_o = 1024
hasNext_o = 1
hasSign_o = 0
hasPrediction_o = 1

delimTableSize_o = 4
for (delimIndex = 0; delimIndex <= 30; delimIndex += 10)
 delimTable_o = 1 << delimIndex
 for (; IndexCnt < numberOfVertexAtt; IndexCnt++) {
 if (predMode == 2)
 validationMask_o = {0, 1, 0}
 else if (predMode == 3)
 validationMask_o = {0, 0, 1}

	else validationMask_o = {1, 0, 0}	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	Description	Range

IECNORM.COM : Click to view the full PDF of ISO/IEC 23002-4:2014/Amd 1:2014

9.3.4 Algo DecodeConnectivity_SVA



```

START
faceCount = 0
PredMode_o = 7
OUTPUT:PredMode_o
INPUT:BinMode_i

IF ( binmode == 1 )
INPUT:FDMode_i
IF ( FDMode_i == 0 )
INPUT:FDMode_i
facedirection = FDMode_i;
ENDIF
ENDIF

INPUT:NumOfData_i
while ( faceCount < NumOfData_i )
INPUT:nType_i
faceCount++
ENDWHILE

faceCount = 0;
skipFaceRotation = 1;
skipReverseFaceDirection = 1;
mask_o = 0
OUTPUT:skipFaceRotation_o
OUTPUT:skipReverseFaceDirection_o
mask_o = 0
WHILE ( nIndex < 3 )
ReferIndex_o = nIndex
OUTPUT:Mask_o
OUTPUT:ReferIndex_o
INPUT:Data_i
Differential_o = Data_i

IF ( binmode == 3 )
INPUT: postP_i
Differential_o *= (postP_i == 1 ? -1 : 1 );
ENDIF

OUTPUT:Difference_o
nIndex++
ENDIF
faceCount++

WHILE ( faceCount < numOfData ) {
nType = ntype[faceCount - 1];

IF ( nType == 4 )
nIndex = 0;
skipFaceRotation = 0;
skipReverseFaceDirection = 0;
OUTPUT:skipFaceRotation_o
OUTPUT:skipReverseFaceDirection_o
INPUT:Data_i
WHILE ( nIndex < 3 ) {
OUTPUT:Mask_o
ReferIndex_o = ( 3 * faceCount ) + nIndex;
OUTPUT:ReferIndex_o

```

```

nIndex++
ENDWHILE
INPUT:Data_i
OUTPUT:nRotation_o
ENDIF

ELSEIF ( nType == 0 )
nIndex = 0;
skipFaceRotation = 0;
OUTPUT:skipFaceRotation_o
INPUT:Data_i
nPosition_o = Data_i

IF ( ( binmode == 3 ) || ( fdmode == 1 ) )
INPUT:Data_i
facedirection = Data_i
ENDIF

IF ( facedirection == 1 ) {
skipReverseFaceDirection = 0
OUTPUT:nPosition_o
ELSE
skipReverseFaceDirection_o = 1

OUTPUT:skipReverseFaceDirection_o
INPUT:Data_i
INPUT: postP_i
Differential_o *= (postP_i == 1 ? -1 : 1);
OUTPUT:Difference_o
WHILE ( nIndex < 3 ) {
referIndex = ( 3 * faceCount ) + nIndex;
OUTPUT:ReferIndex_o
IF ( nIndex == nPosition )
Mask_o = 1
ELSE
Mask_o = 0
OUTPUT:Mask_o
nIndex++
ENDWHILE
INPUT:Data_i
OUTPUT:nRotation_o
ENDIF

ELSE IF ( nType == 1 || nType == 3 ) {
nIndex = 0
skipFaceRotation = 1
skipReverseFaceDirection = 1
OUTPUT:skipFaceRotation_o
OUTPUT:skipReverseFaceDirection_o
Mask_o = 1
while ( nIndex < 3 ) {
OUTPUT:Mask_o
referIndex = ( faceCount * 3 ) + nIndex
OUTPUT:ReferIndex_o
INPUT:Data_i
INPUT: postP_i
Differential_o = Data_i
Differential_o *= (postP_i == 1 ? -1 : 1 )
OUTPUT:Difference_o
nIndex++

```

	<pre> ENDWHILE ENDIF ELSE IF (nType == 2) nIndex = 0 skipFaceRotation = 0 skipReverseFaceDirection = 1 OUTPUT:skipFaceRotation_o OUTPUT:skipReverseFaceDirection_o INPUT:Data_i nPosition = Data_i count = 0 WHILE (nIndex < 3) { IF (nIndex != nPosition) INPUT:Data_i INPUT: postP_i diff_i[count] = Data_i diff_i[count] *= (postP_i == 1 ? -1 : 1); count++ ENDIF nIndex++ ENDWHILE ref[nPosition] = (3 * faceCount) + nPosition; nIndex = 1 WHILE (nIndex < 3) ref[((nPosition + nIndex) % 3)] = (3 * faceCount) + ((nPosition + nIndex) % 3); diff_o[((nPosition + nIndex) % 3)] = diff_i[nIndex - 1] nIndex++ ENDWHILE nIndex = 0 WHILE (nIndex < 3) { IF (nIndex == nPosition) mask = 0 ELSE mask = 1 Difference_o = diff_o[nIndex] OUTPUT:Difference_o ENDELSE ReferIndex_o = ref[nIndex] OUTPUT:Mask_o OUTPUT:ReferIndex_o nIndex++ ENDWHILE INPUT:Data_i nRotation = Data OUTPUT:nRotation_o ENDIF faceCount++ ENDWHILE </pre>
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	
Parameter	

Name	Description	Range

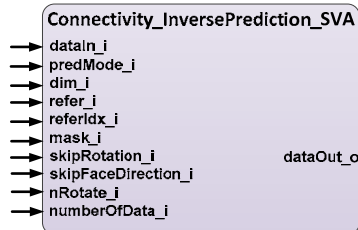
9.3.5 Algo_ExtractFaceDirection_SVA

FU Name	Algo_ExtractFaceDirection_SVA												
Description	ExtractFaceDirection_SVA dataIn_i binMode_i FDMo_o <table><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr><tr><td>dataIn_i</td><td>I</td><td>UINT_8</td></tr><tr><td>binMode_i</td><td>I</td><td>UINT_8</td></tr><tr><td>FDMo_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr></table>	Port Name	Direction (I/O)	Token RANGE	dataIn_i	I	UINT_8	binMode_i	I	UINT_8	FDMo_o	O	UINT8, UINT16, UINT32, UINT64
	Port Name	Direction (I/O)	Token RANGE										
	dataIn_i	I	UINT_8										
	binMode_i	I	UINT_8										
FDMo_o	O	UINT8, UINT16, UINT32, UINT64											
Algo_ExtractFaceDirection_SVA Schematic (FSM):													
Algo_ExtractFaceDirection_SVA Process:													
<pre>START: INPUT: binMode_i IF binMode_i = 1 INPUT: dataIn_i PROCESS: FDMo_o = dataIn_i >> 7; OUTPUT: FDMo_o IF FDMo_o = 0 PROCESS: FDMo_o = (dataIn_i & 0x40) >> 6 OUTPUT: FDMo_o GOTO START</pre>													
ISO Standards using the FU	ISO/IEC 14496-16:2011												
Profiles@levels supported													

9.3.6 Algo_Connectivity_InversePrediction_SVA

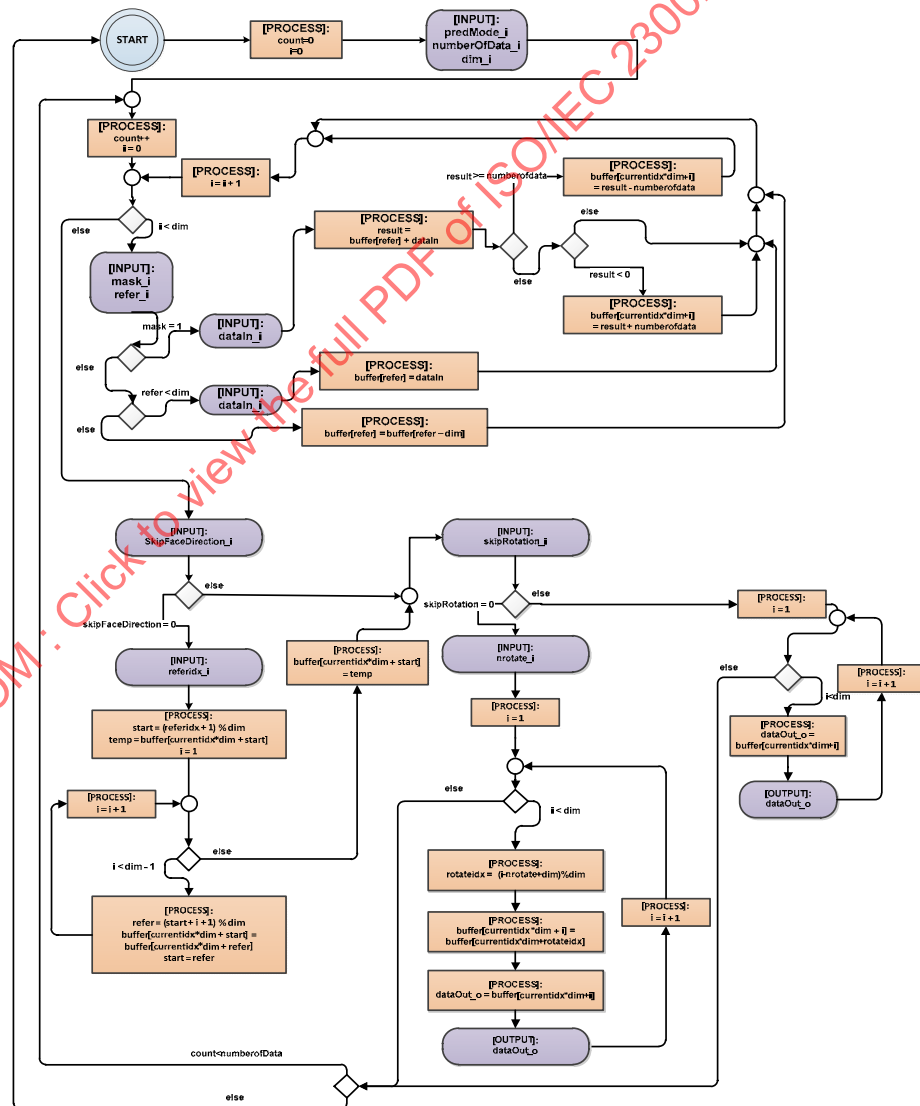
FU Name	Algo_Connectivity_InversePrediction_SVA
---------	---

Description



Port Name	Direction (I/O)	Token RANGE
dataIn_i	I	INT8, INT16, INT32, INT64
predMode_i	I	UINT_4
dim_i	I	UINT8, UINT16, UINT32, UINT64
refer_i	I	UINT8, UINT16, UINT32, UINT64
referIdx_i	I	UINT8, UINT16, UINT32, UINT64
mask_i	I	BOOLEAN
skipRotation_i	I	BOOLEAN
skipFaceDirection_i	I	BOOLEAN
nRotate_i	I	UINT8, UINT16, UINT32, UINT64
numberOfData_i	I	UINT8, UINT16, UINT32, UINT64
dataOut_o	O	INT8, INT16, INT32, INT64

Connectivity_InversePrediction_SVA Schematic (FSM):



	<pre>INPUT : PredictionMode_i numberOfData_i dim_i currentidx =0 i = 0 WHILE(currentidx < numberOfData) WHILE(i < dim) INPUT: refer_i INPUT: mask_i IF mask = 1 INPUT : dataIn_i result = buffer[refer] + dataIn IF (result >= numberOfData) buffer[currentidx*dim + i] = result – numberOfData IF (result < 0) buffer[currentidx*dim + i]= result + numberOfData ELSE IF (refer < 3) INPUT : dataIn_i buffer[refer] = dataIn ELSE buffer[refer] = buffer[refer - dim] i = i + 1 INPUT: skipFaceDirection_i IF skipFaceDirection = 0 INPUT : referIdx_i start = (referIdx + 1) % dim temp = buffer[currentidx*dim + start] i = 1 WHILE i < dim - 1 refer = (start + i + 1) % dim buffer[currentidx*dim + start] = buffer[currentidx*dim + refer] start = refer i = i + 1 buffer[currentidx*dim + start] = temp INPUT: skipRotation_i IF (skipRotation = 0) INPUT : nRotate_i i = 0 WHILE(i < dim) rotateidx = (i – nRotate + dim) % dim buffer[currentidx*dim + i] = buffer[currentidx*dim+rotateidx] dataOut_o = buffer[currentidx*dim + i] OUTPUT: dataOut_o i = i + 1 ELSE WHILE(i < dim) dataOut_o = buffer[currentidx*dim + i] OUTPUT: dataOut_o i = i + 1 currentidx ++ GOTO START</pre>	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	Description	Range

After Annex C, add the following annexes:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23002-4:2014/Amd 1:2014

Annex D (normative)

Granular FUs concept

The Granular FU is a special type of FU designed to address in a unitary manner the functionalities of several FUs. It can be represented and described as a group of individual FUs and some management units. By using granular FUs more efficient and platform dependent implementation can be achieved.

The Granular FU extends the usage of some FU for the broad use and reusability of FU. There are bound traffics between FUs, or sometimes some FUs need communal buffers. To reduce the data traffic or eliminate the dependence between FUs, those necessary existing FUs can be designed as one granular FU. The FU description of Granular FU is similar with other FUs except one additional part - FU network diagram to describe the internal network instead of FSM diagram. At the following section, an instance of Granular FU (Granular_InverseQuantization3D) is showing.

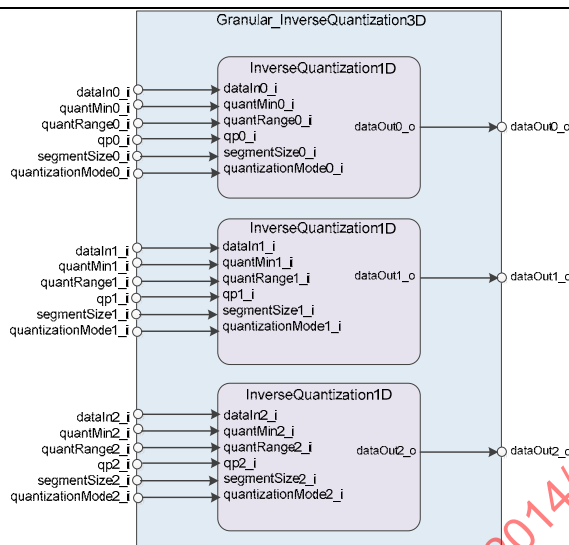
IECNORM.COM : Click to view the full PDF of ISO/IEC 23002-4:2014/Amd. 1:2014

Annex E (informative)

Granular FUs

E.1 Granular InverseQuantization3D

FU Name	Granular_InverseQuantization3D																																																																		
Description	Three ALGO_InverseQuantization1D FUs are used to make a 3-dimensional inverse quantization process. Granular_InverseQuantization3D [homogeneousQ] [dimD] dataIn0_i dataOut0_o quantMin0_i quantRange0_i qp0_i segmentSize0_i quantizationMode0_i dataIn1_i dataOut1_o quantMin1_i quantRange1_i qp1_i segmentSize1_i quantizationMode1_i dataIn2_i dataOut2_o quantMin2_i quantRange2_i qp2_i segmentSize2_i quantizationMode2_i <table><thead><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr></thead><tbody><tr><td>dataIn0_i</td><td>I</td><td>INT8, INT16, INT32, INT64</td></tr><tr><td>quantValue0_i</td><td>I</td><td>FLOAT</td></tr><tr><td>quantMin0_i</td><td>I</td><td>FLOAT</td></tr><tr><td>qp0_i</td><td>I</td><td>UINT_32</td></tr><tr><td>segmentSize0_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>quantizationMode0_i</td><td>I</td><td>UINT_2</td></tr><tr><td>dataIn1_i</td><td>I</td><td>INT8, INT16, INT32, INT64</td></tr><tr><td>quantValue1_i</td><td>I</td><td>FLOAT</td></tr><tr><td>quantMin1_i</td><td>I</td><td>FLOAT</td></tr><tr><td>qp1_i</td><td>I</td><td>UINT_32</td></tr><tr><td>segmentSize1_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>quantizationMode1_i</td><td>I</td><td>UINT_2</td></tr><tr><td>dataIn2_i</td><td>I</td><td>INT8, INT16, INT32, INT64</td></tr><tr><td>quantValue2_i</td><td>I</td><td>FLOAT</td></tr><tr><td>quantMin2_i</td><td>I</td><td>FLOAT</td></tr><tr><td>qp2_i</td><td>I</td><td>UINT_32</td></tr><tr><td>segmentSize2_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>quantizationMode2_i</td><td>I</td><td>UINT_2</td></tr><tr><td>dataOut0_o</td><td>O</td><td>FLOAT</td></tr><tr><td>dataOut1_o</td><td>O</td><td>FLOAT</td></tr><tr><td>dataOut2_o</td><td>O</td><td>FLOAT</td></tr></tbody></table> Granular InverseQuantization3D network schematic	Port Name	Direction (I/O)	Token RANGE	dataIn0_i	I	INT8, INT16, INT32, INT64	quantValue0_i	I	FLOAT	quantMin0_i	I	FLOAT	qp0_i	I	UINT_32	segmentSize0_i	I	UINT8, UINT16, UINT32, UINT64	quantizationMode0_i	I	UINT_2	dataIn1_i	I	INT8, INT16, INT32, INT64	quantValue1_i	I	FLOAT	quantMin1_i	I	FLOAT	qp1_i	I	UINT_32	segmentSize1_i	I	UINT8, UINT16, UINT32, UINT64	quantizationMode1_i	I	UINT_2	dataIn2_i	I	INT8, INT16, INT32, INT64	quantValue2_i	I	FLOAT	quantMin2_i	I	FLOAT	qp2_i	I	UINT_32	segmentSize2_i	I	UINT8, UINT16, UINT32, UINT64	quantizationMode2_i	I	UINT_2	dataOut0_o	O	FLOAT	dataOut1_o	O	FLOAT	dataOut2_o	O	FLOAT
Port Name	Direction (I/O)	Token RANGE																																																																	
dataIn0_i	I	INT8, INT16, INT32, INT64																																																																	
quantValue0_i	I	FLOAT																																																																	
quantMin0_i	I	FLOAT																																																																	
qp0_i	I	UINT_32																																																																	
segmentSize0_i	I	UINT8, UINT16, UINT32, UINT64																																																																	
quantizationMode0_i	I	UINT_2																																																																	
dataIn1_i	I	INT8, INT16, INT32, INT64																																																																	
quantValue1_i	I	FLOAT																																																																	
quantMin1_i	I	FLOAT																																																																	
qp1_i	I	UINT_32																																																																	
segmentSize1_i	I	UINT8, UINT16, UINT32, UINT64																																																																	
quantizationMode1_i	I	UINT_2																																																																	
dataIn2_i	I	INT8, INT16, INT32, INT64																																																																	
quantValue2_i	I	FLOAT																																																																	
quantMin2_i	I	FLOAT																																																																	
qp2_i	I	UINT_32																																																																	
segmentSize2_i	I	UINT8, UINT16, UINT32, UINT64																																																																	
quantizationMode2_i	I	UINT_2																																																																	
dataOut0_o	O	FLOAT																																																																	
dataOut1_o	O	FLOAT																																																																	
dataOut2_o	O	FLOAT																																																																	



Granular InverseQuantization3D network FNL code

```
<Network name = "Granular_InverseQuantization3D">
```

```
<!-- input tokens -->
```

```
<Port kind = "Input" name="dataIn0">
<Port kind = "Input" name="quantMin0">
<Port kind = "Input" name="quantRange0">
<Port kind = "Input" name="qp0">
<Port kind = "Input" name="segmentSize0">
<Port kind = "Input" name="quantizationMode0">
<Port kind = "Input" name="dataIn1">
<Port kind = "Input" name="quantMin1">
<Port kind = "Input" name="quantRange1">
<Port kind = "Input" name="qp1">
<Port kind = "Input" name="segmentSize1">
<Port kind = "Input" name="quantizationMode1">
<Port kind = "Input" name="dataIn2">
<Port kind = "Input" name="quantMin2">
<Port kind = "Input" name="quantRange2">
<Port kind = "Input" name="qp2">
<Port kind = "Input" name="segmentSize2">
<Port kind = "Input" name="quantizationMode2">
```

```
<!-- output tokens -->
```

```
<Port kind = "Output" name="dataOut0">
<Port kind = "Output" name="dataOut1">
<Port kind = "Output" name="dataOut2">
```

```
<!-- Instantiation -->
```

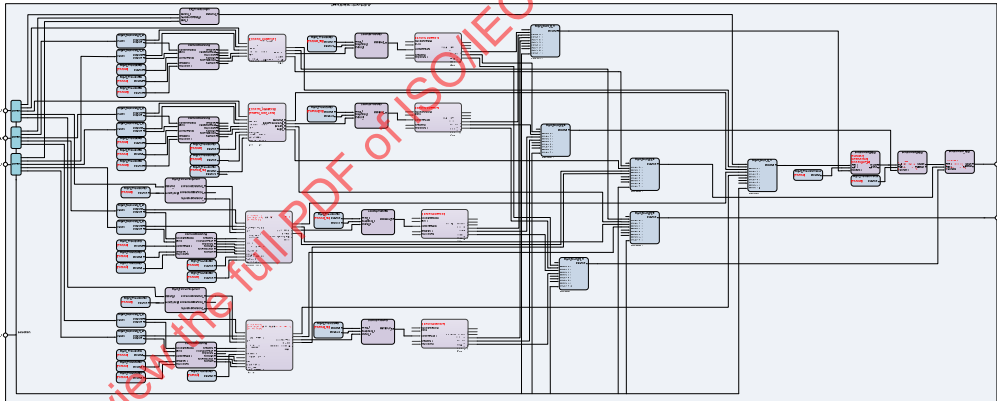
```
<Instance id="Inverse_Quantization_x">
  <Class name="InverseQuantization1D"/>
</Instance>
<Instance id="Inverse_Quantization_y">
  <Class name="InverseQuantization1D"/>
</Instance>
<Instance id="Inverse_Quantization_z">
  <Class name="InverseQuantization1D"/>
</Instance>
```

```
<!-- connections -->
```

```
<Connection dst = "Inverse_Quantization_x" dst-port="dataIn0_i"
src-port="dataIn0_i">
<Connection dst = "Inverse_Quantization_x" dst-port="quantMin0_i"
src-sort="quantMin0_i">
<Connection dst = "Inverse_Quantization_x" dst-port="quantRange0_i"
src-sort="quantRange0_i">
<Connection dst = "Inverse_Quantization_x" dst-port="qp0_i"
src-sort="qp0_i">
<Connection dst = "Inverse_Quantization_x" dst-port="segmentSize0_i"
src-sort="segmentSize0_i">
<Connection dst = "Inverse_Quantization_x" dst-port="quantizationMode0_i"
```

	<pre>src-sort=" quantizationMode0_i"> <Connection dst-port="dataOut0_o" src="Inverse_Quantization_x" src-port="dataOut0_o"> <Connection dst = "Inverse_Quantization_y" dst-port="dataIn1_i" src-port="dataIn1_i"> <Connection dst = "Inverse_Quantization_y" dst-port="quantMin1_i" src-sort="quantMin1_i"> <Connection dst = "Inverse_Quantization_y" dst-port="quantRange1_i" src-sort="quantRange1_i"> <Connection dst = "Inverse_Quantization_y" dst-port="qp1_i" src-sort="qp1_i"> <Connection dst = "Inverse_Quantization_y" dst-port="segmentSize1_i" src-sort=" segmentSize1_i"> <Connection dst = "Inverse_Quantization_y" dst-port="quantizationMode1_i" src-sort=" quantizationMode1_i"> <Connection dst-port="dataOut1_o" src="Inverse_Quantization_y" src-port="dataOut1_o"> <Connection dst = "Inverse_Quantization_z" dst-port="dataIn2_i" src-port="dataIn2_i"> <Connection dst = "Inverse_Quantization_z" dst-port="quantMin2_i" src-sort="quantMin2_i"> <Connection dst = "Inverse_Quantization_z" dst-port="quantRange2_i" src-sort="quantRange2_i"> <Connection dst = "Inverse_Quantization_z" dst-port="qp2_i" src-sort="qp2_i"> <Connection dst = "Inverse_Quantization_z" dst-port="segmentSize2_i" src-sort=" segmentSize2_i"> <Connection dst = "Inverse_Quantization_z" dst-port="quantizationMode2_i" src-sort=" quantizationMode2_i"> <Connection dst-port="dataOut2_o" src="Inverse_Quantization_z" src-port="dataOut2_o"> </Network></pre>	
ISO Standards using the FU		
Profiles@levels supported		
Parameter		
Name	Description	Type / Range
dimD	Describes the number of tokens of type dataIn_i that are consumed at each firing. This parameter is set at the network configuration level.	Type: Integer Range: [1 .. 2 ⁵]
homogeneousQ	Describes the number of tokens of type quantRange_i, quantMin_i and quantValue_i that are necessary for the inverse quantization process. This parameter is set at the network configuration level. The number of tokens is equal to dimD if this parameter is 0 and the number of tokens is equal to 1 if this parameter is 1	Type: Boolean Range: {0,1}

E.2 Granular_ED_InverseBinarization

FU Name	Algo_Granular_ED_InverseBinarization																		
Description	This FU describes the inverse binarization process used in SC3DMC. Granular_InverseBinarization [PREFIX_SIZE_LEN] [DM_LengthShift] [SCALE_RANGE] [USE_VALIDATION_MASK] dataIn_i numberOfData_i dim_i selection_i dataOut_o preds_o <table><thead><tr><th>Port Name</th><th>Direction (I/O)</th><th>Token RANGE</th></tr></thead><tbody><tr><td>dataIn_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>numberOfData_i</td><td>I</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>dim_i</td><td>I</td><td>UINT8</td></tr><tr><td>dataOut_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr><tr><td>preds_o</td><td>O</td><td>UINT8, UINT16, UINT32, UINT64</td></tr></tbody></table> Granular ED_InverseBinarization network schematic  Granular ED_InverseBinarization SC3DMC network FNL code <pre><?xml version="1.0" encoding="UTF-8"?> <XDE name="Granular_InverseBinarization"> <Port kind="Input" name="dataIn_i"> <Type name="int"> <Entry kind="Expr" name="size"> <Expr kind="Literal" literal-kind="Integer" value="32"/> </Entry> </Type> </Port> <Port kind="Input" name="numberOfData_i"> <Type name="int"> <Entry kind="Expr" name="size"> <Expr kind="Literal" literal-kind="Integer" value="32"/> </Entry> </Type> </Port> <Port kind="Input" name="dim_i"> <Type name="int"> <Entry kind="Expr" name="size"> <Expr kind="Literal" literal-kind="Integer" value="32"/> </Entry> </Type> </Port> <Port kind="Input" name="selection_i"> <Type name="int"> <Entry kind="Expr" name="size"> <Expr kind="Literal" literal-kind="Integer" value="32"/> </Entry> </Type> </Port></pre>	Port Name	Direction (I/O)	Token RANGE	dataIn_i	I	UINT8, UINT16, UINT32, UINT64	numberOfData_i	I	UINT8, UINT16, UINT32, UINT64	dim_i	I	UINT8	dataOut_o	O	UINT8, UINT16, UINT32, UINT64	preds_o	O	UINT8, UINT16, UINT32, UINT64
	Port Name	Direction (I/O)	Token RANGE																
	dataIn_i	I	UINT8, UINT16, UINT32, UINT64																
numberOfData_i	I	UINT8, UINT16, UINT32, UINT64																	
dim_i	I	UINT8																	
dataOut_o	O	UINT8, UINT16, UINT32, UINT64																	
preds_o	O	UINT8, UINT16, UINT32, UINT64																	