INTERNATIONAL STANDARD

ISO/IEC 23002-4

Second edition
2014-04-15

AMENDMENT 1
2014-11-15

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)

Reference number
ISO/IEC 23002-4:2014/Amd.1:2014(E)

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*.

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.

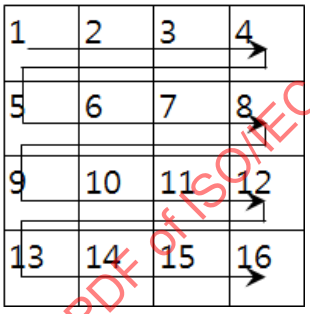
ID & Name	Description
55 BOOLEAN	Token which value is 0 or 1.
56 SIGN	Token which value is 0 or 1.
57 FLAG	Token which value is 0 or 1.
58 UINT_2	Unsigned integer on 2 bits.
59 UINT_4	Unsigned integer on 4 bits.
60 UINT_8	Unsigned integer on 8 bits.
61 UINT_16	Unsigned integer on 16 bits.
62 UINT_32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

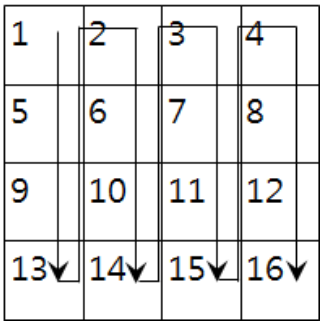
Add 4.4 "Array data order":

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

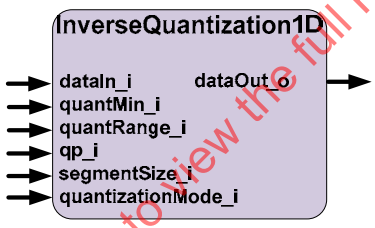
The significance of the “flag” values is:

Value	Significance
0	false
1	true

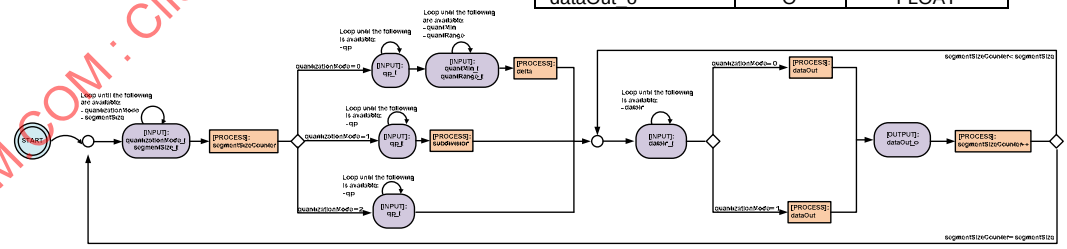
Add 5.2 "General Processing FUs":

5.2 General Processing FUs

5.2.1 Algo_InverseQuantization1D

FU Name	Algo_InverseQuantization1D
Description	

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

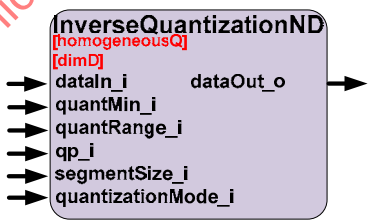
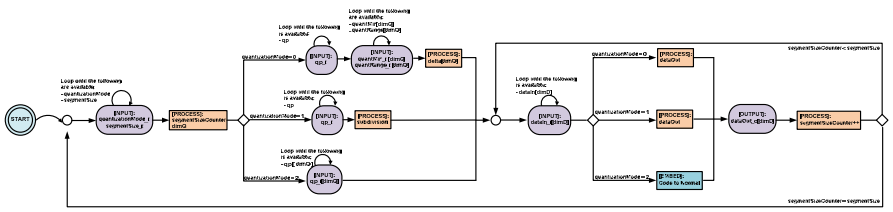


Inverse Quantization Process:

```
START:
INPUT_SEGMENT_PARAM:
INPUT:
        quantizationMode
        segmentSize
SegmentSizeCounter = 0
IF quantizatioMode = 0
    INPUT:
          qp
          quantMin
          quantRange
    IF ( quantRange > 0.0 )
          delta = ( ( 1 << qp ) - 1 ) / quantRange
```

	ELSE delta = 1.0 IF quantizationMode = 1 INPUT: qp INPUT_DATA_IN: INPUT: dataIn IF quantizatioMode = 0 PROCESS: dataOut = quantMin + (dataIn / delta) IF quantizatioMode = 1 PROCESS: dataOut = dataIn / qp OUTPUT dataOut SegmentSizeCounter ++ IF SegmentSizeCounter < segmentSize GOTO INPUT_DATA_IN ELSE GOTO INPUT_SEGMENT_PARAM The following table contains the quantization types index used in the Inverse Quantization 1D FU: **Name of quantization mode** / **Value** Uniform Quantization / **0** Uniform Texture Quantization / **1** 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" of input data (dataIn), it uses the same set of quantMin, quantRange and quantValue to produce a set of output data (dataOut) of size "segmentSize"
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

5.2.2 Algo_InverseQuantizationND

FU Name	Algo_InverseQuantizationND
Description	 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 

Inverse Quantization Process:

$$dimQ = \begin{cases} dimD, \ if \ homogeneousQ = 0 \\ 1, \ if \ homogeneousQ = 1 \end{cases}$$

```
START:
INPUT_SEGMENT_PARAM
INPUT:
        quantizationMode
        segmentSize
SegmentSizeCounter = 0
IF quantizatioMode = 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 quantizatioMode = 0
    PROCESS:
        dataOut = quantMin [ segmentSizeCounter%dimD ] + (dataIn / delta [segmentSizeCounter%dimD])
IF quantizatioMode = 1
    PROCESS: EMBED Code 2 Normal
IF quantizatioMode = 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 ( (factor^2) - 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
```

```
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
```

The following table contains the quantization types index used in the Inverse Quantization ND FU:

Name of quantization mode	Value
Uniform Quantization	**0**
Normal Quantization	**1**
Uniform Texture Quantization	**2**

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.

ISO Standards using the FU

ISO/IEC 14496-16:2011

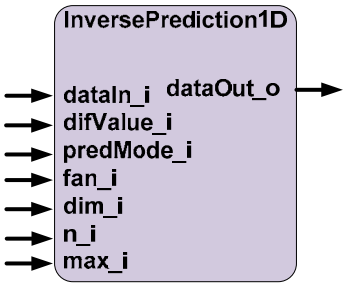
Profiles@levels supported

Parameter

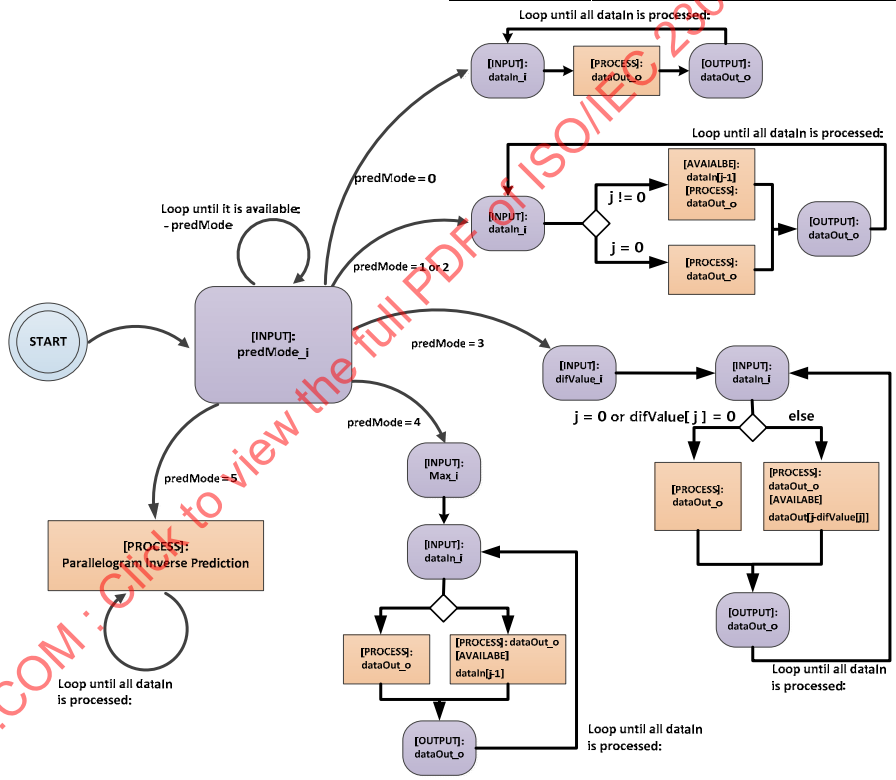
Name	Description	Type / Range
dimD	Describes the number of tokens of type dataIn_i that are consumed at each firing. This parameter is set at the network configuration level.	Type: Integer Range: [1 .. 2^5]
homogeneousQ	Describes the number of tokens of type quantRange_i, quantMin_i and quantValue_i 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

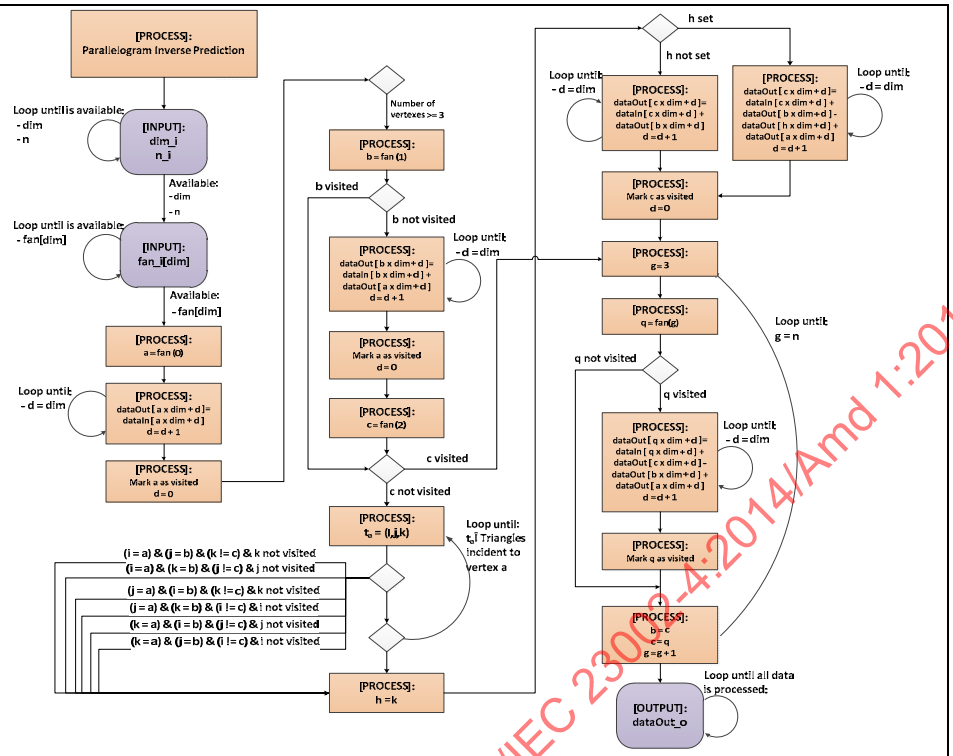
FU Name	Algo_InversePrediction1D
Description	This FU allows five modes of inverse prediction. Not all the ports are used for each prediction mode. This FU can handle one input token at a time.



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



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

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

Case 2: XOR – based prediction

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

Case 3: Adaptive Prediction

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

Case 4: Circular Differential Prediction

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

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

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 = (I,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
```

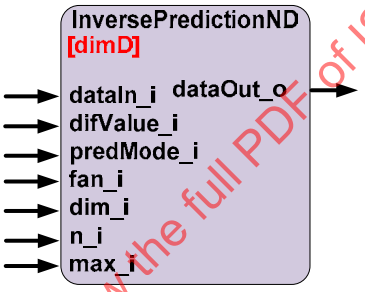
The following table contains the prediction types index used in the Inverse Prediction 1D FU:

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

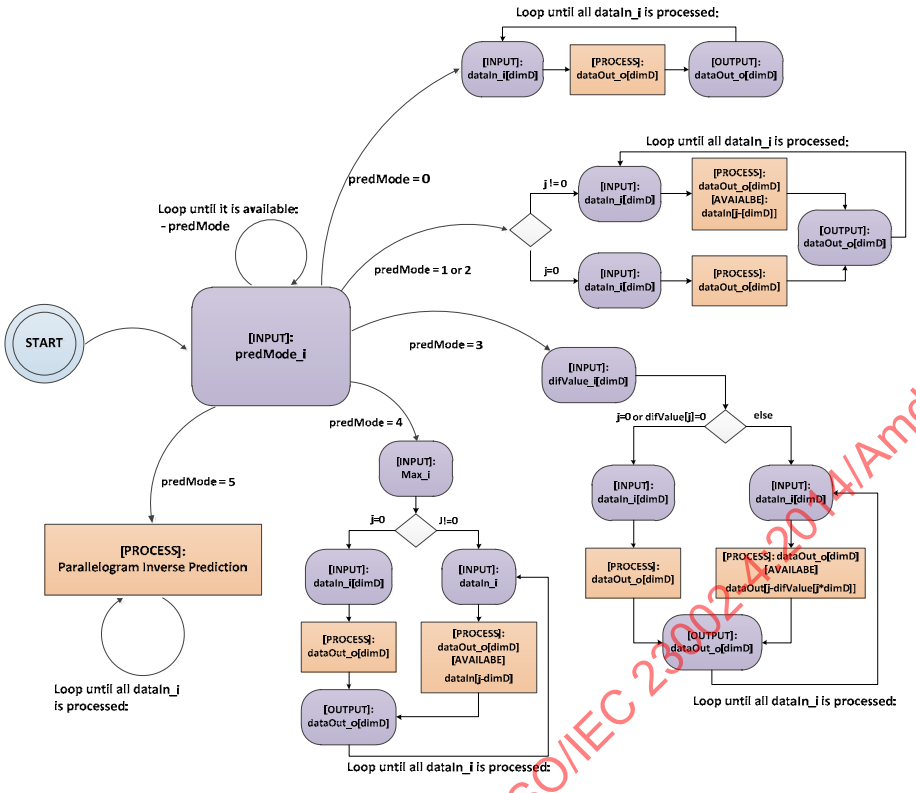
	Adaptive Differential Prediction **3** Circular Differential Prediction **4** Parallelogram Inverse Prediction **5** The detailed description of the inverse parallelogram prediction is described in Annex F.
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

Parameter		
Name	**Description**	**Range**

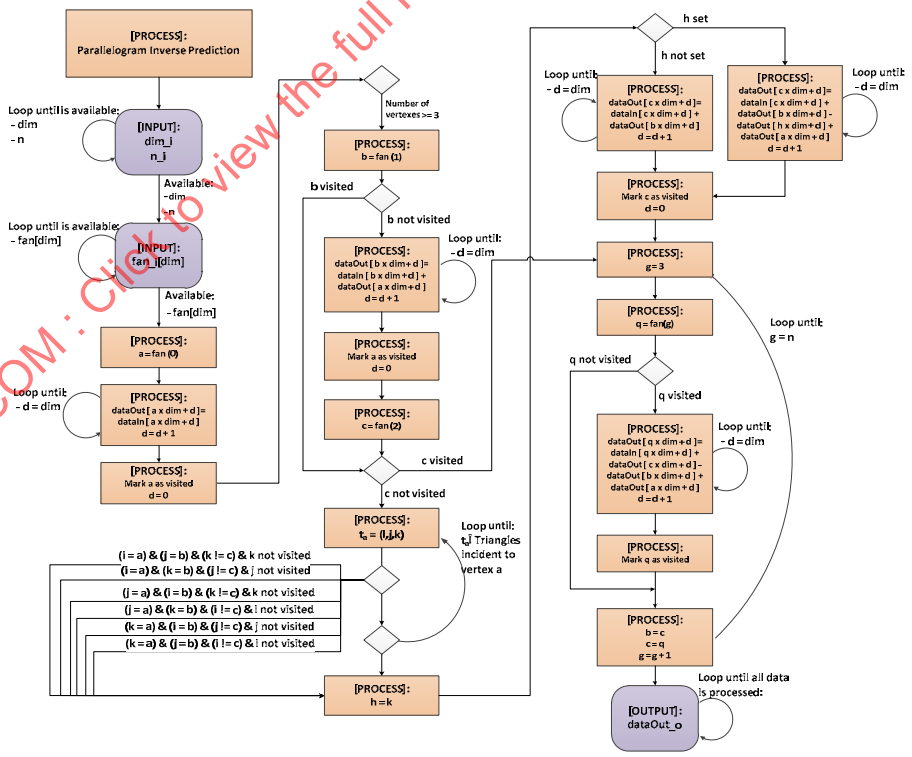
5.2.4 Algo_InversePredictionND

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.  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



Process Parallelogram Inverse Prediction Schematic (FSM):



Inverse Prediction ND Process:

Switch(predMode)

{

Case 0: NP – No Prediction

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

Case 1: Diff – Differential Prediction

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

Case 2: XOR – based prediction

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

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, t_a
            WHILE t_a = (I,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**

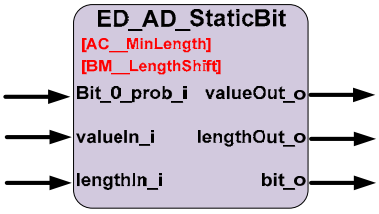
The detailed description of the inverse parallelogram prediction is described in Annex F.

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]

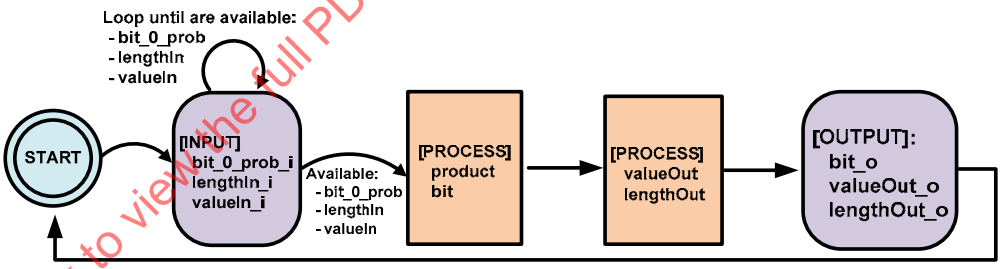
5.2.5 Algo_ED_AD_StaticBit

FU Name	Algo_ED_AD_StaticBit
Description	This FU describes the arithmetic decoding process based on a static bit model.



Port Name	Direction (I/O)	Token RANGE
Bit_0_prob_i	I	UINT8, UINT16, UINT32, UINT64
valueIn_i	I	UINT8, UINT16, UINT32, UINT64
length_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

Process Schematic (FSM):



ED_AD_StaticBit Process:

```
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
```

ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

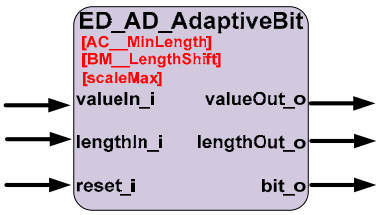
Parameter		
Name	**Description**	**Range**
AC__MinLength	Describes the threshold for renormalization. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: [1 .. 2^{32}]
BM__LengthShift	Describes the length bits discarded before multiplication. This parameter is set at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

5.2.6 Alog_ED_AD_AdaptiveBit

FU Name: Algo_ED_AD_AdaptiveBit

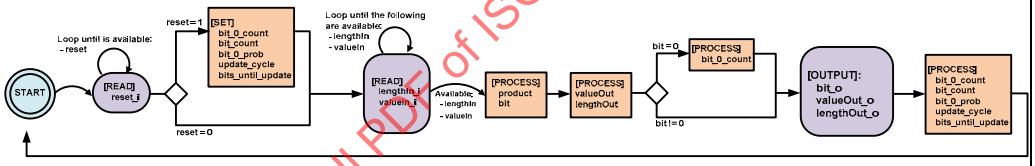
Description

This FU describes the arithmetic decoding process based on a adaptive bit model as presented in [3,4].



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:

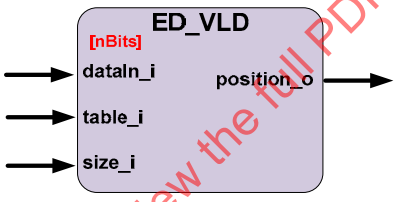
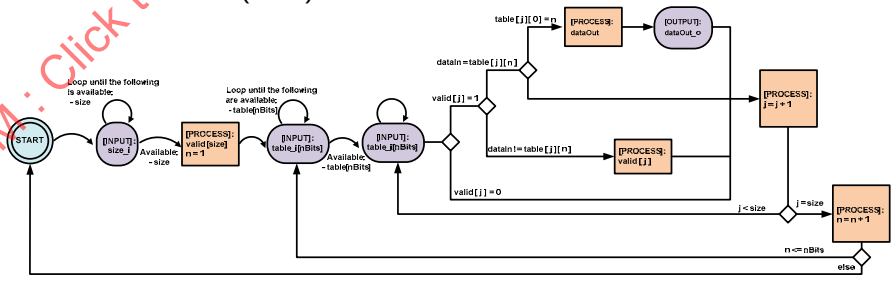
```
START
If reset = 1
    bit_0_count = 1;
    bit_count   = 2;
    bit_0_prob  = 1U << (BM__LengthShift - 1);
    update_cycle = bits_until_update = 4;

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;
```

	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.  Process Schematic (FSM):  ED_VLD Process: 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

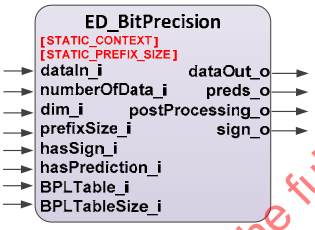
Port Name	Direction (I/O)	Token RANGE
dataIn_i	I	UINT8, UINT16, UINT32, UINT64
table_i	I	UINT8, UINT16, UINT32, UINT64
size_i	I	UINT8, UINT16, UINT32, UINT64
position_o	O	UINT8, UINT16, UINT32, UINT64

	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]

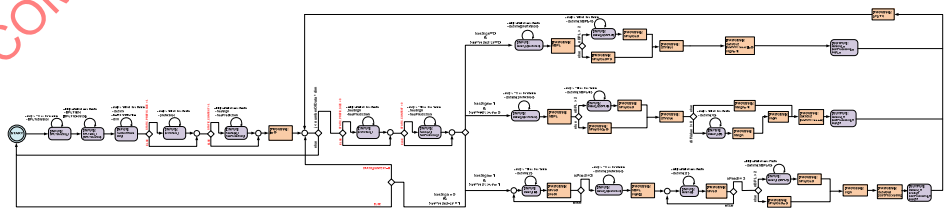
5.2.8 Algo_ED_BitPrecision

FU Name	Algo_ED_BitPrecision
Description	This FU describes the Entropy Decoding Bit Precision processes.



Port Name	Direction (I/O)	Token RANGE
dataIn_i	I	UINT_8
numerOfData_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

ED_BitPrecision Schematic (FSM):



ED_BitPrecision process:

```
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
```

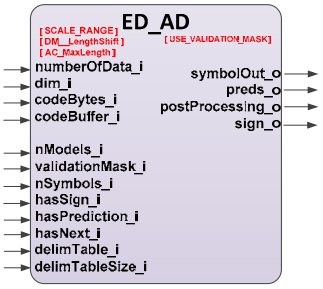
```
PROCESS_CHUNK:
WHILE j < numberOfData * dim
    IF STATIC_PREFIX_SIZE = 0
        INPUT: prefixSize_i
    IF STATIC_CONTEXT = 0
        INPUT: hasSign_i
        INPUT: hasPrediction_i
    IF hasSign = 0 & hasPrediction = 0
        INPUT: dataIn { prefixSize }
        nBPL = dataIn
        IF nBPL > 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 interation. 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 interation. This parameter is set at the network configuration level	Type:Boolean Range: {0,1}

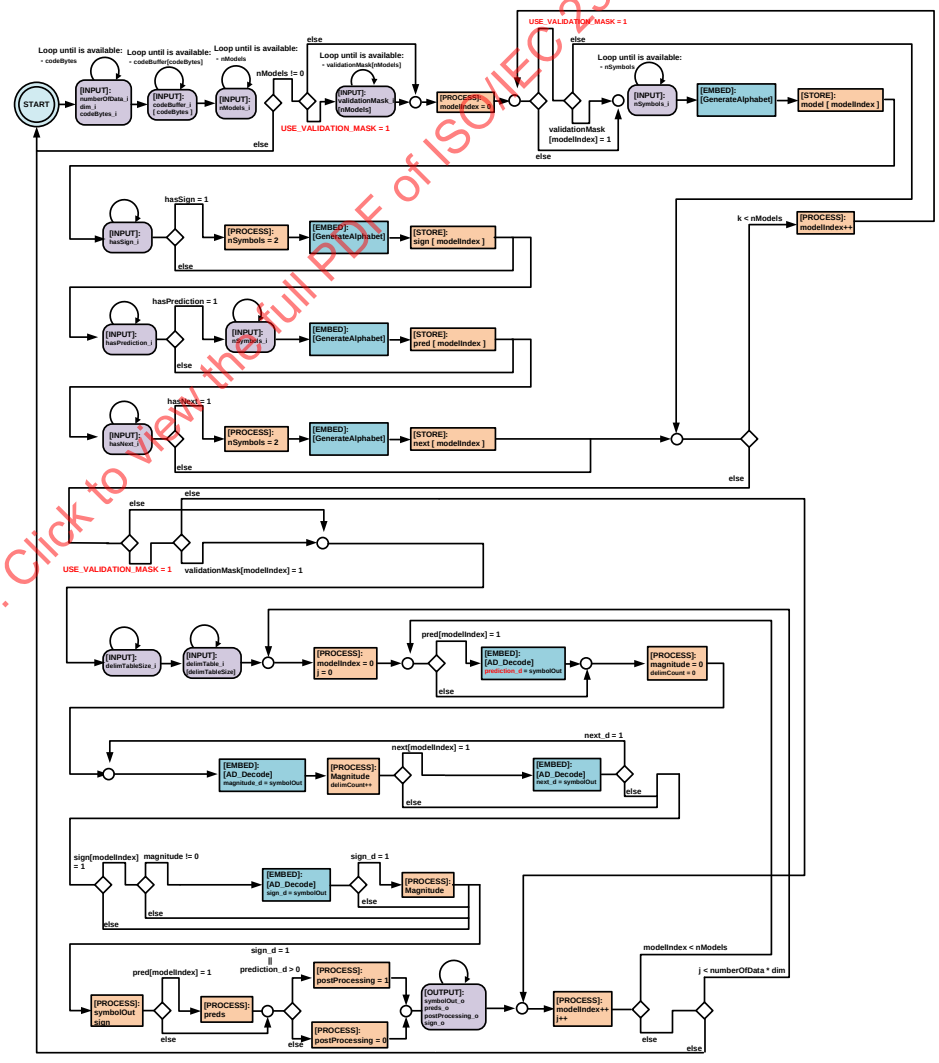
5.2.9 Algo_ED_AD

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

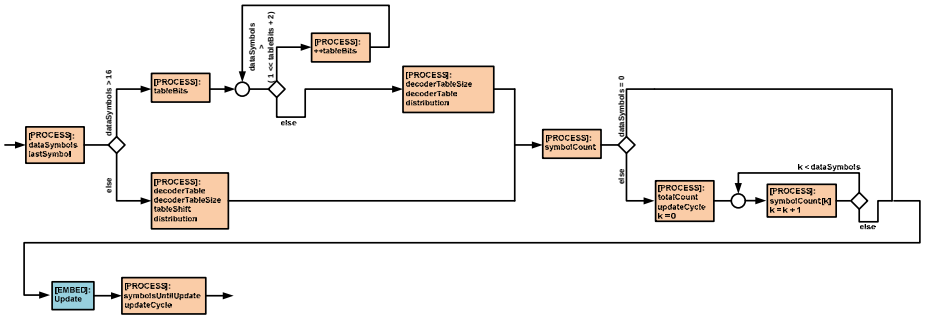


Port Name	Direction (I/O)	Token RANGE
numberOfData_i	I	UINT8, UINT16, UINT32, UINT64
dim_i	I	UINT8, UINT16, UINT32, UINT64
codeBytes_i	I	UINT8, UINT16, UINT32, UINT64
codeBuffer_i	I	UINT_8
nModels_i	I	UINT_8
validationMask_i	I	BOOLEAN
nSymbols_i	I	UINT8, UINT16, UINT32, UINT64
hasSign_i	I	BOOLEAN
hasPrediction_i	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

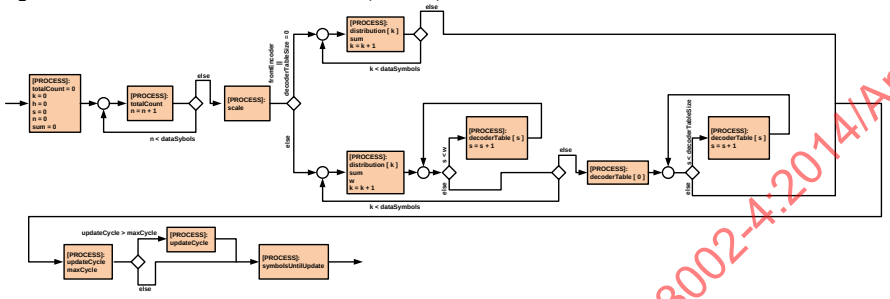
ED_AD Schematic (FSM):



Generate Alphabet Process Schematic (FSM):

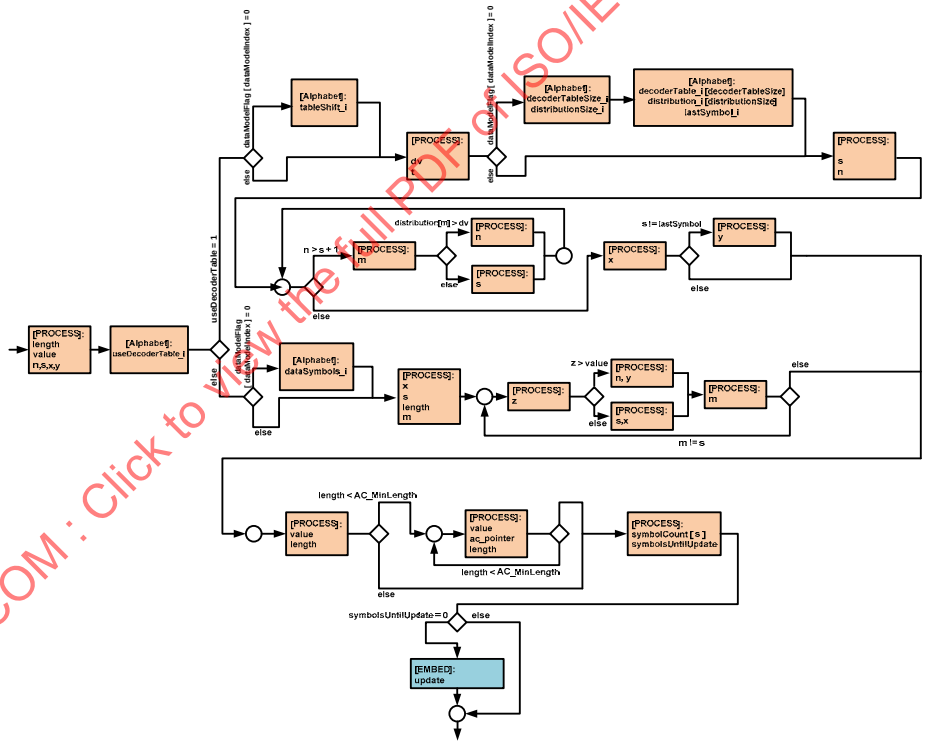


Update - Process Schematic (FSM):



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
```

```
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
```

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}

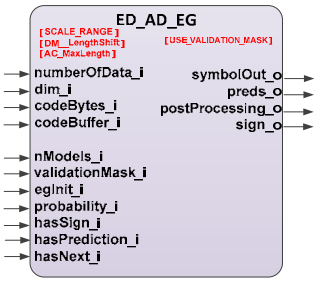
5.2.10 Algo_ED_AD_EG

FU Name

Algo_ED_AD_EG

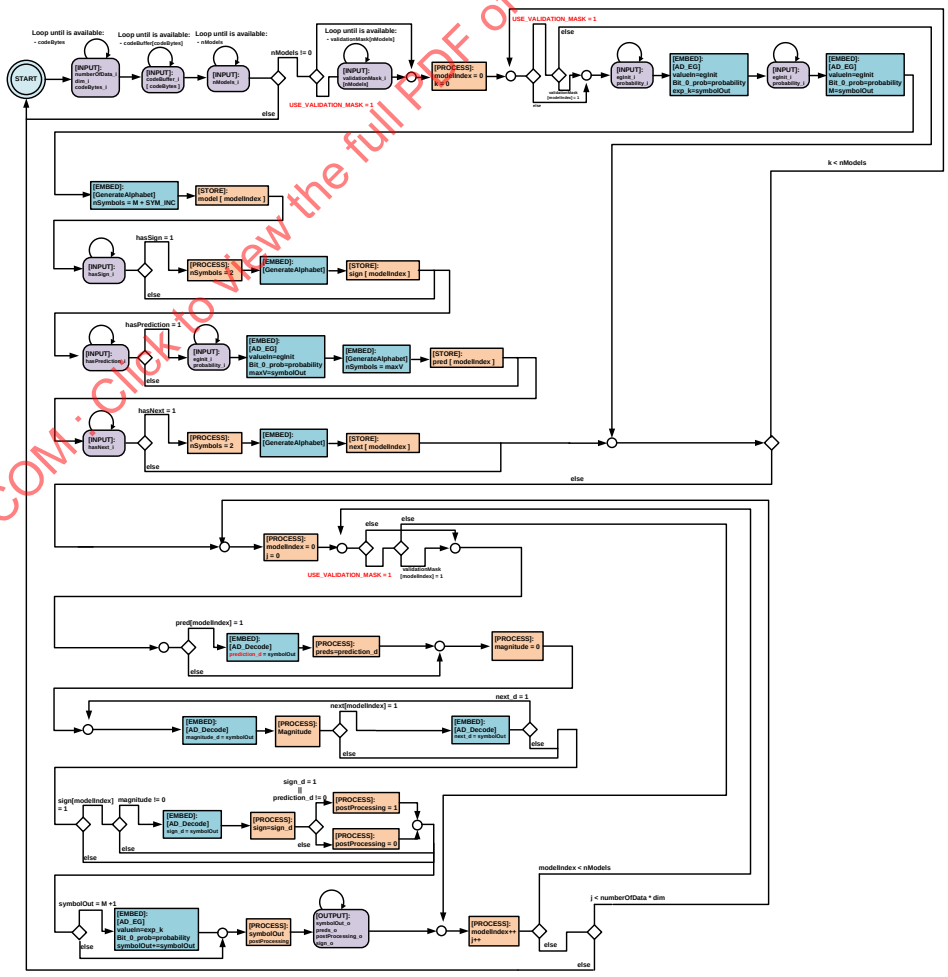
Description

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

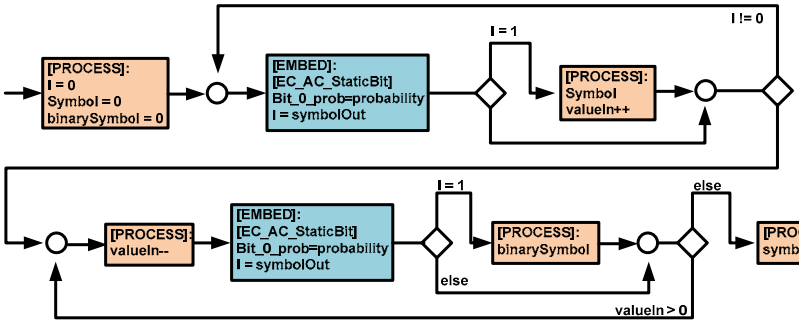


Port Name	Direction (I/O)	Token RANGE
numberOfData_i	I	UINT8, UINT16, UINT32, UINT64
dim_i	I	UINT8, UINT16, UINT32, UINT64
codeBytes_i	I	UINT8, UINT16, UINT32, UINT64
codeBuffer_i	I	UINT_8
nModels_i	I	UINT_8
validationMask_i	I	BOOLEAN
egInit_i	I	UINT8, UINT16, UINT32, UINT64
probability_i	I	UINT8, UINT16, 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

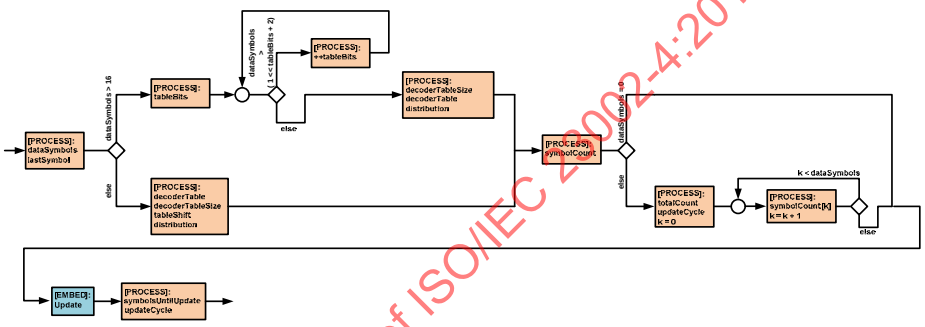
ED_AD_EG Schematic (FSM):



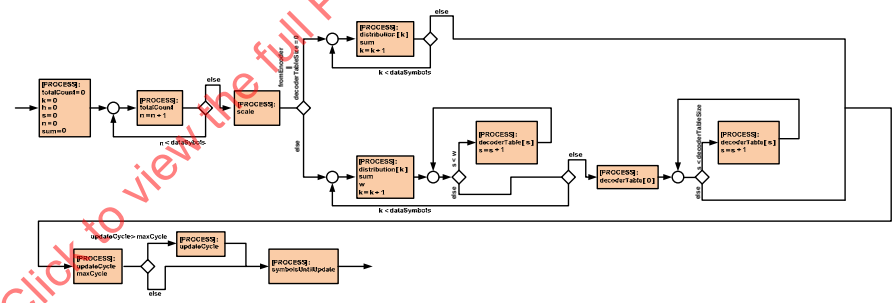
ED_AD_EG Process Schematic (FSM):



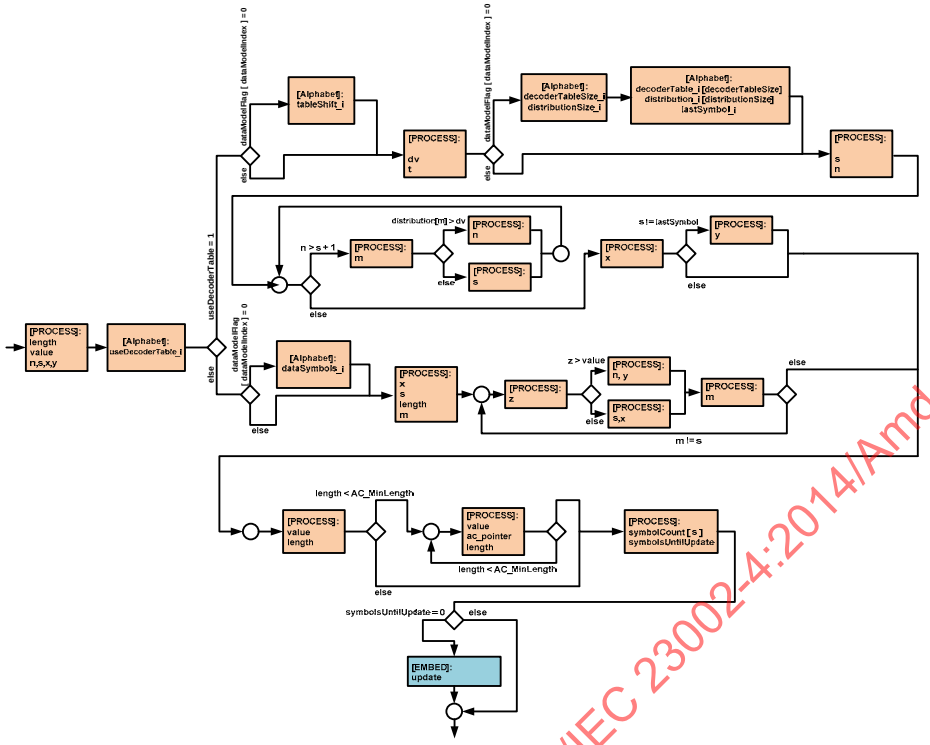
Generate Alphabet Process Schematic (FSM):



Update - Process Schematic (FSM):



Arithmetic Decode Adaptive Data - Process Schematic (FSM):



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: egInit_i
    INPUT: probability_i
    EMBED: AD_EG (valueIn=egInit, Bit_0_prob=probability,exp_k=symbolOut)
    INPUT: egInit_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: egInit_i
                    INPUT: probability_i
                    EMBED: AD_EG (valueIn=egInit,
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

```
                    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
```

```
        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
```

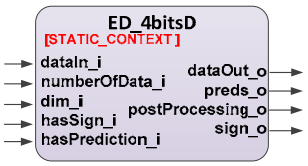
	m = (n = dataSymbols) >> 1
	DO
	z = length * distribution [m]
	if z > value
	n = m
	y = z
	else
	s = m
	x = z
	WHILE m = ((s+n) >> 1) != s
	value -= x
	length = y – x
	if length < AC__MinLength
	DO
	value = (value << 8) | ++ac_pointer
	WHILE (length <<= 8) < AC__MinLength
	++symbolCount [s]
	if -- symbolsUntilUpdate = 0
	EMBED: Update

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}

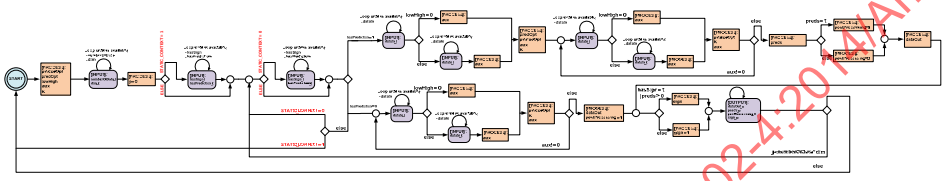
5.2.11 Algo_ED_4bitsD

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
numerOfData_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
```

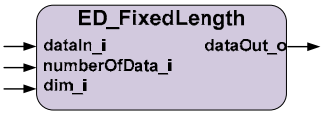
	pValueOpt += aux << k
	IF preds > 0
	postProcessing = 1
	ELSE
	postProcessing = 0
	preds = predOpt
	IF hasSign = 1 | preds > 0
	sign = signDef [pValueOpt % 2]
	ELSE
	sign = 1
	dataOut = pValueOpt
	OUTPUT: dataOut_o
	OUTPUT: preds_o
	OUTPUT: postProcessing_o
	OUTPUT: sign_o
	
	ELSE IF hasPrediction = 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
	
	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 interation. This parameter is set at the network configuration level.	Type:Boolean Range: {0,1}

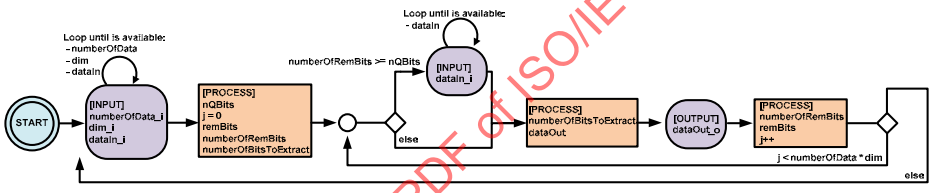
5.2.12 Algo_ED_ FixedLength

FU Name	Algo_ED_FixedLength
Description	This FU describes the Entropy Decoding Fixed Length processes.



Port Name	Direction (I/O)	Token RANGE
dataIn_i	I	UINT_8
numerOfData_i	I	UINT8, UINT16, UINT32, UINT64
dim_i	I	UINT8, UINT16, UINT32, UINT64
dataOut_o	O	INT8, INT16, INT32, INT64

ED Fixed Length Decoding Schematic (FSM):



ED Fixedlength Decoding process:

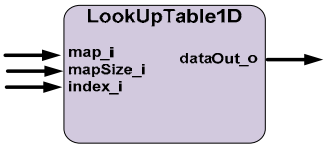
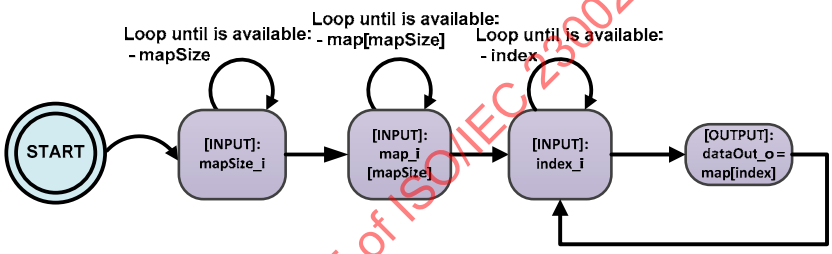
START
INPUT:
numberOfData_i
dim_i
predMode_i
dataIn_i
nQBits = dataIn
j = 0
remBits = 0
numberOfRemBits = 0
numberOfBitsToExtract = 0
DO
If numberOfRemBits >= nQBits
INPUT:
dataIn_i
numberOfBitsToExtract = nQBits - numberOfRemBits
dataOut = remBits | (dataIn[numberOfBitsToExtract])
numberOfRemBits = nQBits – numberOfBitsToExtract
remBits = numberOfRemBits from dataIn
j++
WHILE j < numberOfData * dim
GOTO START

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

ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

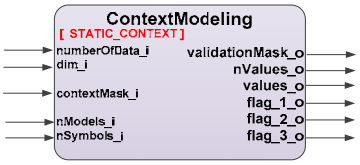
Parameter

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.  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):  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			

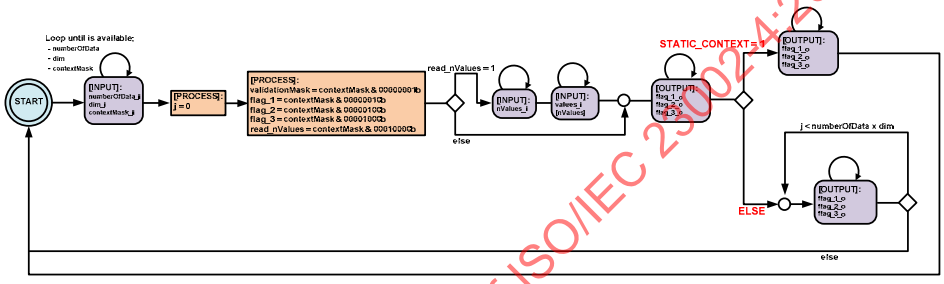
5.2.14 Algo_ContextModeling

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):



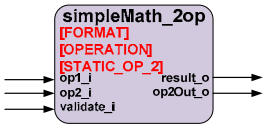
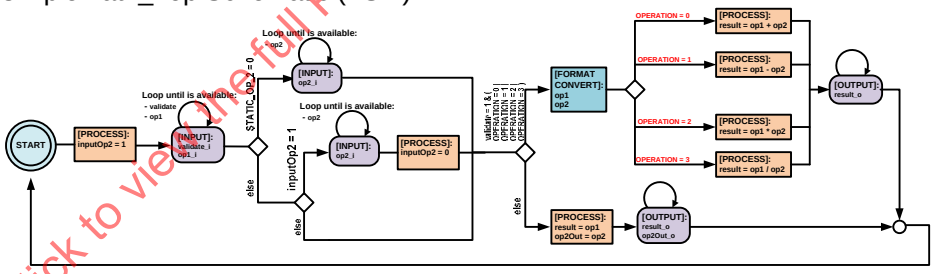
ContextModeling process:

```
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.  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 Schematic (FSM):  simpleMath_2op process: 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

```
                break
            CASE 3:
                result = op1 / op2
                break
        OUTPUT: result_o
    ELSE
        result = op1
        op2Out = op2
        OUTPUT: result_o
        OUTPUT: op2Out_o
    GOTO START
```

The following table contains the interpretation of the FORMAT codes:

FORMAT	Interpretation
0	Unsigned Integer
1	Signed Integer
2	Fixed Point
3	Floating Point (IEEE 754)

The following table contains the supported operations:

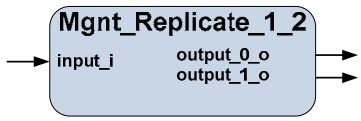
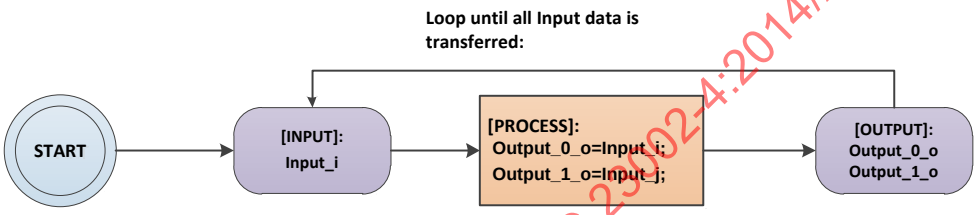
Index	Operation
0	Addition (+)
1	Substraction (-)
2	Multiplication (*)
3	Division (/)

Note:

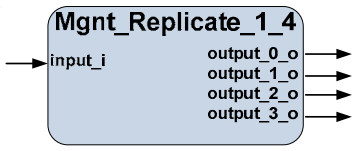
If the validate_i flag is true (value 1), one of the operations above are applied to the 2 operands. Else, the operands are repeated at the output ports (result_o = op1 and op2Out_o = op2).

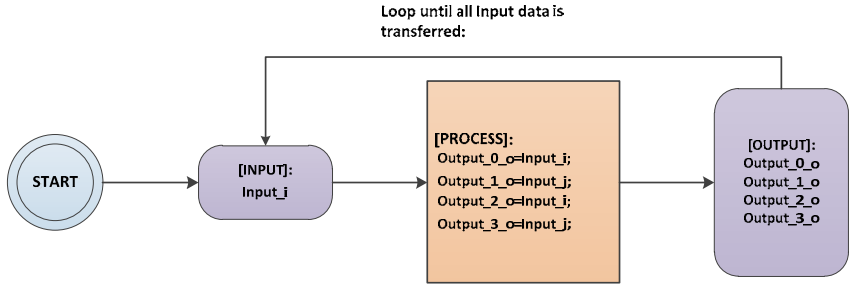
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
FORMAT	Describes the input operands number format representation. This parameter is set at the network configuration level.	Type:Unsigned Integer Range: {0,1,2,3}
OPERATION	Describes the index of the operation to 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

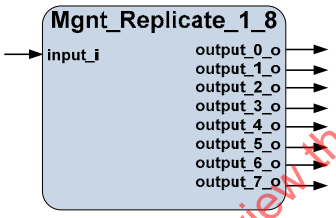
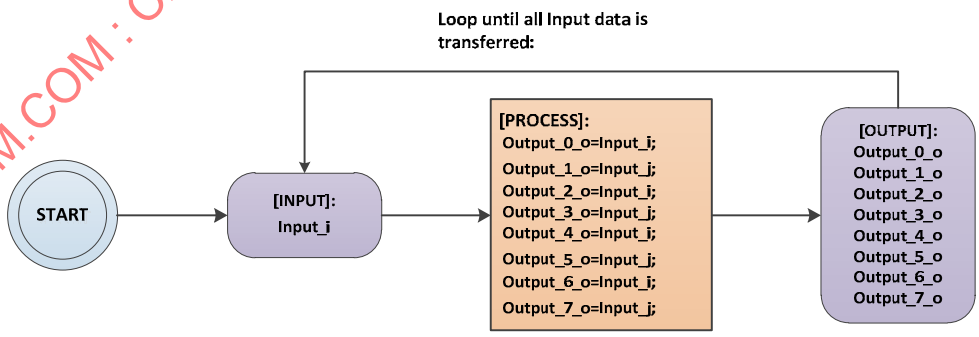
FU Name	Mgnt_Replicate_1_2	
Description	The output is generated by replicating the transferred input data. The detailed process is described in the above FSM. Block diagram:  Ports description Port Constraints: - the output ports have to be of the same data type as the input port Process Schematic (FSM):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

5.2.17 Mgnt_Replicate_1_4

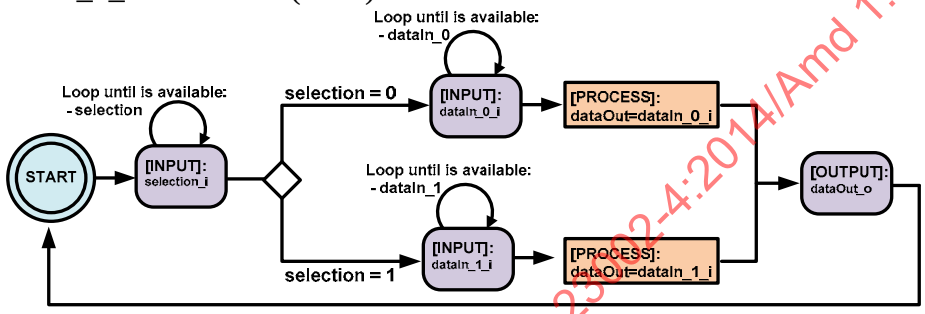
FU Name	Mgnt_Replicate_1_4
Description	The output is generated by replicating the transferred input data. The detailed process is described in the above FSM. Block diagram:  Ports description Port Constraints: - the output ports have to be of the same data type as the input port Process Schematic (FSM):

	
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	
Parameter	
Name	**Description** / **Range**

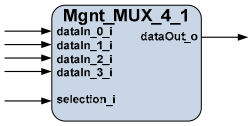
5.2.18 Mgnt_Replicate_1_8

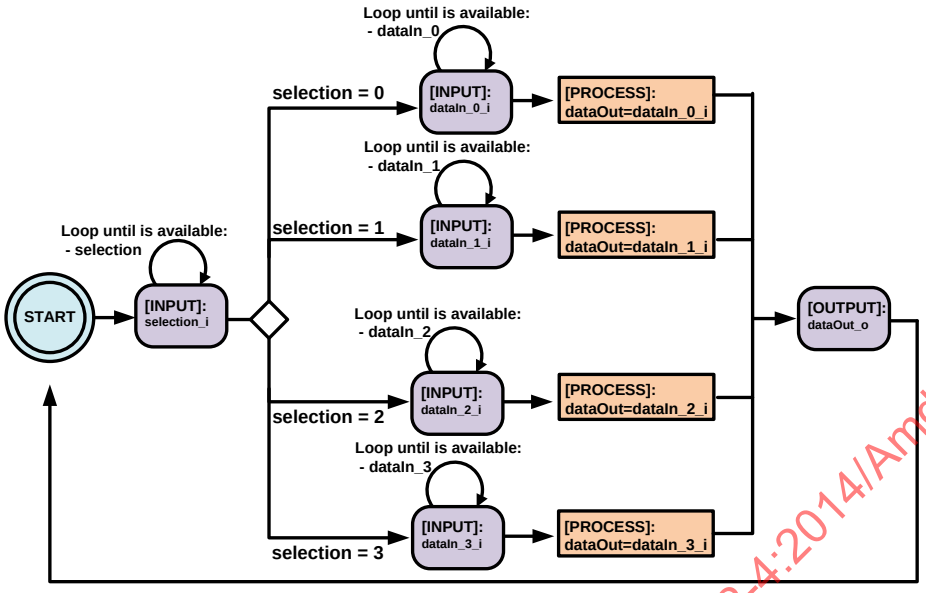
FU Name	Mgnt_Replicate_1_8
Description	The output is generated by replicating the transferred input data. The detailed process is described in the above FSM. Block diagram:  Ports description Port Constraints: - the output ports have to be of the same data type as the input port Process Schematic (FSM):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

5.2.19 Mgnt_MUX_2_1

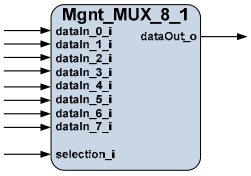
FU Name	Mgnt_MUX_2_1
Description	 Port Constraints: - the dataIn ports have to be of the same data type - the dataOut port has to be of the same data type as the dataIn input ports - the selection port is of type bit. MUX_2_1 Schematic (FSM):  MUX_2_1 Process: START INPUT: selection SWITCH selection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

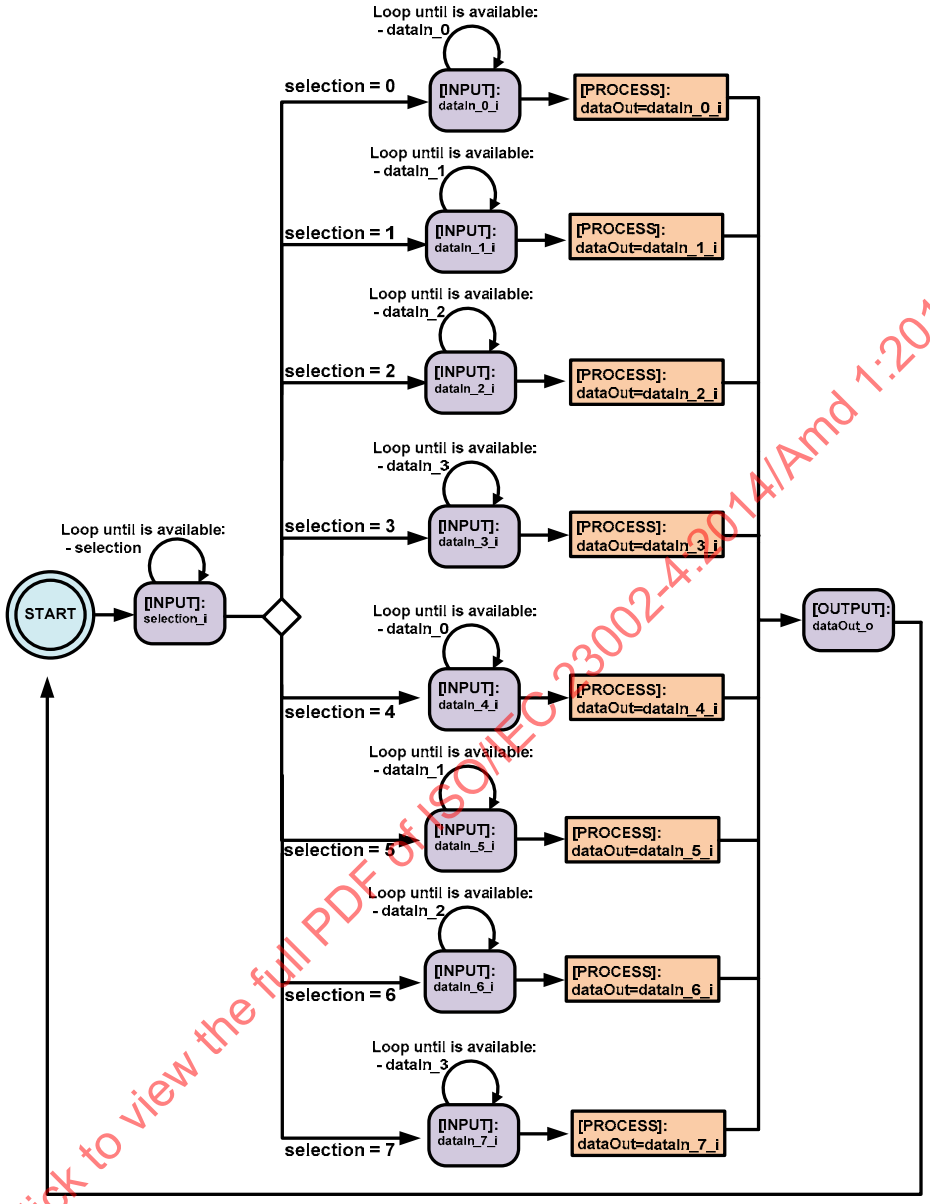
5.2.20 Mgnt_MUX_4_1

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):

	 MUX_4_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 OUTPUT: dataOut_o GOTO START
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	 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 3 bits. MUX_8_1 Schematic (FSM):

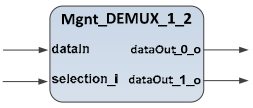
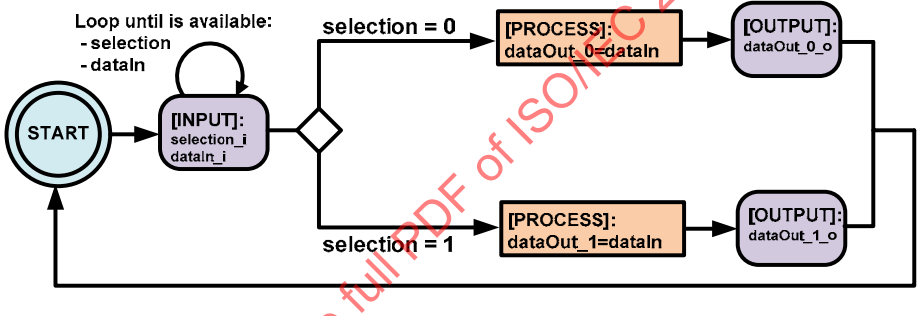


MUX_8_1 Process:

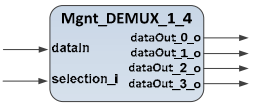
START
INPUT: selection
SWITCH selection
    CASE 0:
        INPUT: dataIn_0_i
        dataOut = dataIn_0
    CASE 1:
        INPUT: dataIn_1_i
        dataOut = dataIn_1
    CASE 2:
        INPUT: dataIn_2_i
        dataOut = dataIn_2
    CASE 3:
        INPUT: dataIn_3_i
        dataOut = dataIn_3
    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

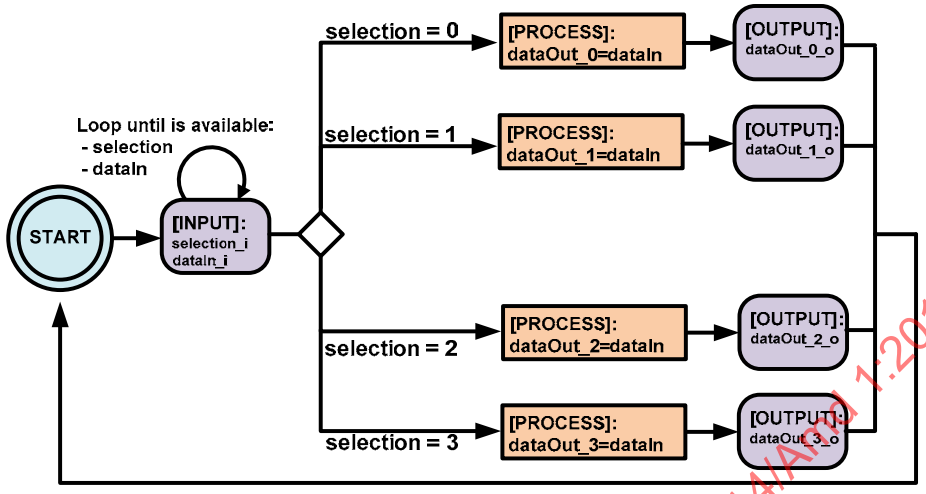
	CASE 7: INPUT: dataIn_7_i dataOut = dataIn_7 OUTPUT: dataOut_o GOTO START
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

5.2.22 Mgnt_DEMUX_1_2

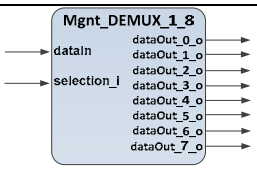
FU Name	Mgnt_DEMUX_1_2
Description	 Port Constraints: - the dataOut ports have to be of the same data type - the dataIn port has to be of the same data type as the dataOut ports - the selection port is of type bit. DEMUX_1_2 Schematic (FSM):  DEMUX_1_2 Process: START INPUT: selection INPUT: dataIn SWITCH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

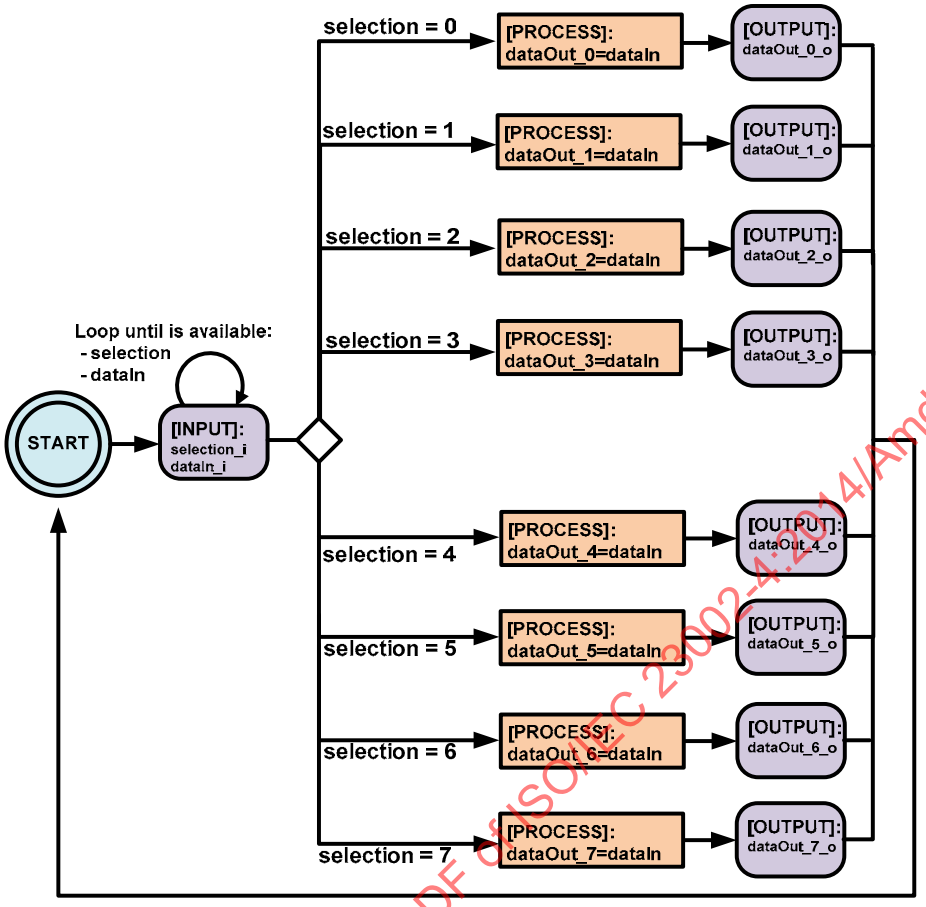
5.2.23 Mgnt_DEMUX_1_4

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: 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
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:

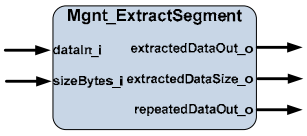
START
INPUT: selection
INPUT: dataIn
SWITCH selection
CASE 0:
dataOut_0 = dataIn
OUTPUT: dataOut_0
CASE 1:
dataOut_1 = dataIn
OUTPUT: dataOut_1
CASE 2:
dataOut_2 = dataIn
OUTPUT: dataOut_2
CASE 3:
dataOut_3 = dataIn
OUTPUT: dataOut_3
CASE 4:
dataOut_4 = dataIn
OUTPUT: dataOut_4
CASE 5:
dataOut_5 = dataIn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

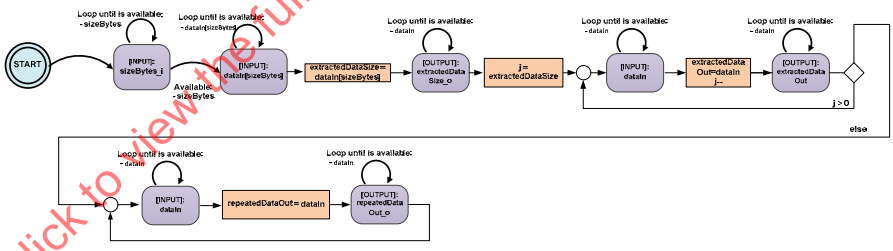
5.2.25 Mgnt_ExtractSegment

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:

extractedDataSize	extractedData	repeatedData
size = sizeBytes	size = extractedDataSize	size = rest of input



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

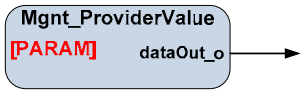


ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

Parameter		
Name	**Description**	**Range**

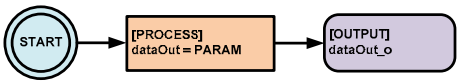
5.2.26 Mgnt_ProviderValue

FU Name	Mgnt_ProviderValue
Description	



Port Name	Direction (I/O)	Token RANGE
dataOut_o	O	UINT8, UINT16, UINT32, UINT64

ProviderValue Process Schematic



(FSM):

ProviderValue Process:
START:
dataOut = PARAM
OUTPUT:
dataOut_o

ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

Parameter

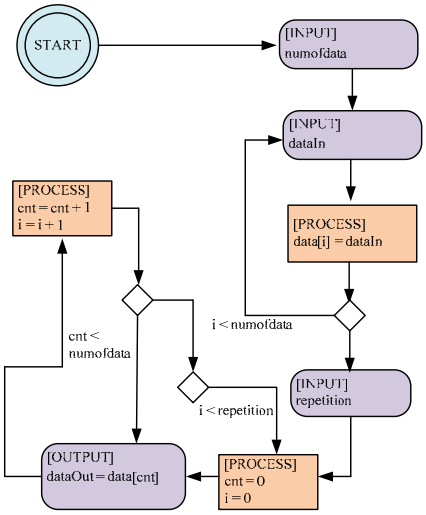
Name	Description	Range
PARAM	Describes the value of the parameter that is outputted one time as dataOut. It is set at the network configuration level.	

5.2.27 Mgnt_RepeatSegment

FU Name	Mgnt_RepeatSegment
Description	This FU describes how to repetitively transfer the given data.

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

RepeatSegment Process Schematic (FSM):

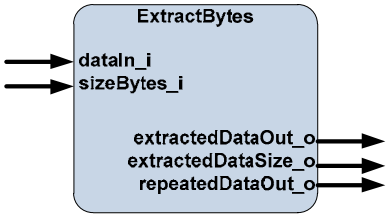


RepeatSegment Process:

START:
INPUT: numofdata
WHILE i < numofdata
INPUT: dataIn
data[i] = dataIn
INPUT: repetition
WHILE i < repetition
WHILE cnt < numofdata
dataOut = data[cnt]
OUTPUT: dataOut

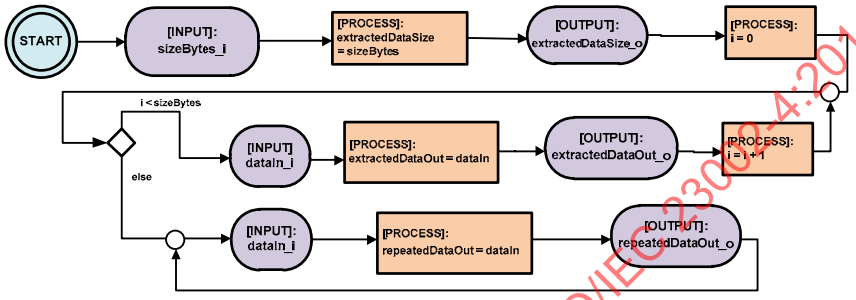
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	**Description**	**Range**

5.2.28 Mgnt_ExtractBytes

FU Name	Mgnt_ExtractBytes
Description	

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

Mgnt_ExtractBytes Schematic:



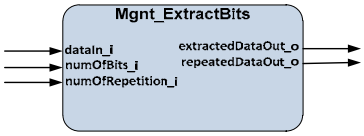
Mgnt_ExtractBytes Process:

```
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
```

ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

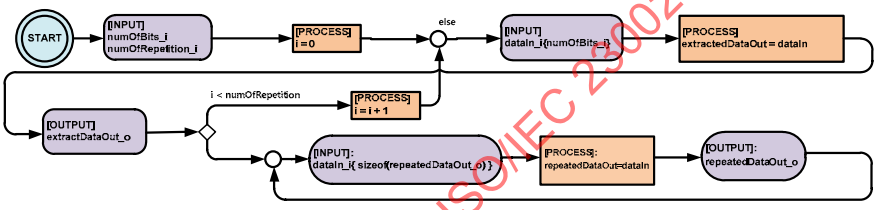
5.2.29 Mgnt_ExtractBits

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.



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
repatedDataOut_o	O	UINT8, UINT16, UINT32, UINT64

ExtractBits Process Schematic (FSM):



Ext

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

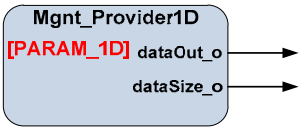
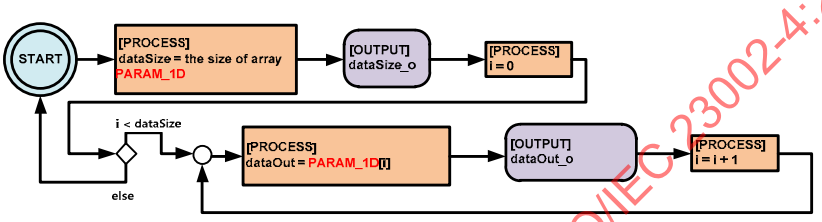
Note: The ‘sizeof’ operation is used to obtain the size of the token in bits (bus width). The way to use the ‘sizeof’ operation is as follows:

sizeof(name of port)

ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	
Parameter	

Name	Description	Range

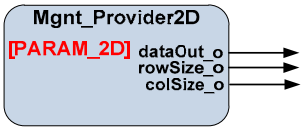
5.2.30 Mgnt_Provider1D

FU Name	Mgnt_Provider1D
Description	This FU describes how to provide 1D array values from FU network description.  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):  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]}**

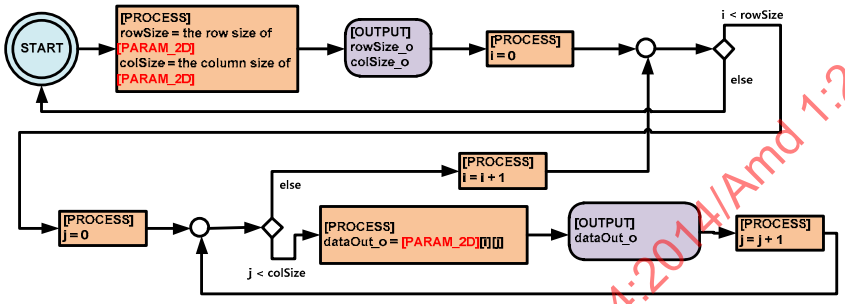
5.2.31 Mgnt_Provider2D

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



Port Name	Direction (I/O)	Token RANGE
dataOut_o	O	UINT8, UINT16, UINT32, UINT64
rowSize_o	O	UINT8, UINT16, UINT32, UINT64
colSize_o	O	UINT8, UINT16, UINT32, UINT64

Provider2D Process Schematic



(FSM):
Provider2D Process:
START:
rowSize = the row size of 2-dimensional array PARAM_2D
colSize = the column size of 2-dimensional array PARAM_2D
OUTPUT:
rowSize_o
colSize_o
i = 0
ROW_LOOP
IF i < rowSize
j = 0
COL_LOOP
IF i < colSize
dataOut = PARAM_2D[i][j]
OUTPUT:
dataOut_o
j = j + 1
GOTO COL_LOOP
ELSE
i = i + 1
GOTO ROW_LOOP
ELSE GOTO START

ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

Parameter

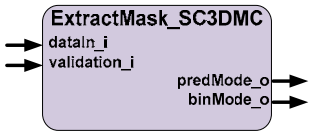
Name	**Description**	**Range**
PARAM_2D	It is stored internally as 2-dimensional array, loaded at the network configuration stage.	**{{[0, 2^{32}-1], ..., [0, 2^{32}-1]}, {[0, 2^{32}-1], ..., [0, 2^{32}-1]} ... {[0, 2^{32}-1], ..., [0, 2^{32}-1]}}**

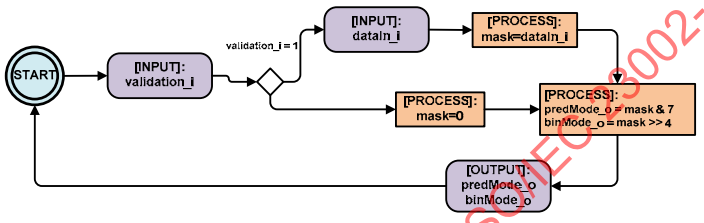
Add section 9. MPEG-4 Part 16 SC3DMC Decoder Specific FUs

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

<table>
<tr><td>FU Name</td><td>Algo_ExtractMask_SC3DMC</td></tr>
<tr><td>Description</td><td>


<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>validation_i</td><td>I</td><td>BOOLEAN</td></tr>
<tr><td>predMode_o</td><td>O</td><td>UINT_8</td></tr>
<tr><td>binMode_o</td><td>O</td><td>UINT_8</td></tr>
</table>
Algo_ExtractMask_SC3DMC Schematic


Algo_ExtractMask_SC3DMC Process:
START:
INPUT:
validation_i
IF validation_i = 1
INPUT:
dataIn_i
PROCESS:
mask = dataIn_i;
ELSE
PROCESS:
mask = 0
PROCESS:
predMode_o = mask & 7;
binMode_o = mask >> 4;
OUTPUT:
predMode_o
binMode_o
GOTO START

Note:
SC3DMC mask description:
<table>
<tr><td>Bit position</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td></td><td>bit</td><td>bit</td><td>bit</td><td>bit</td><td>bit</td><td>bit</td><td>bit</td><td>bit</td></tr>
<tr><td>Flag</td><td colspan="4">Binarization Mode</td><td>x</td><td colspan="3">Prediction Mode</td></tr>
</table>
</td></tr>
<tr><td>ISO Standards using the FU</td><td>ISO/IEC 14496-16:2011</td></tr>
<tr><td>Profiles@levels supported</td><td></td></tr>
</table>

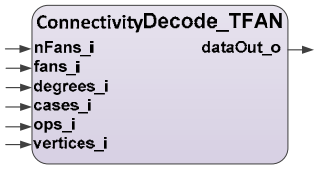
9.2 MPEG-4 SC3DMC TFAN Specific FUs

9.2.1 Algo_DecodeConnectivity_TFAN

FU Name

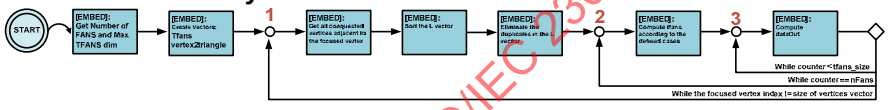
Algo_DecodeConnectivity_TFAN

Description

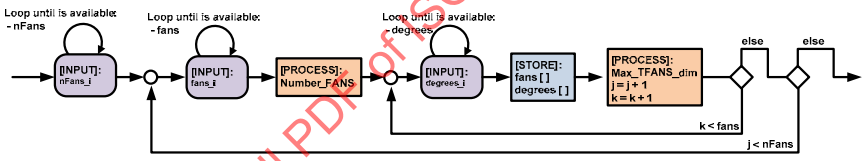


Port Name	Direction (I/O)	Token RANGE
nFans_i	I	UINT8, UINT16, UINT32, UINT64
fans_i	I	UINT8, UINT16, UINT32, UINT64
degrees_i	I	UINT8, UINT16, UINT32, UINT64
cases_i	I	UINT8, UINT16, UINT32, UINT64
ops_i	I	UINT8, UINT16, UINT32, UINT64
vertices_i	I	UINT8, UINT16, UINT32, UINT64
dataOut_o	O	UINT8, UINT16, UINT32, UINT64

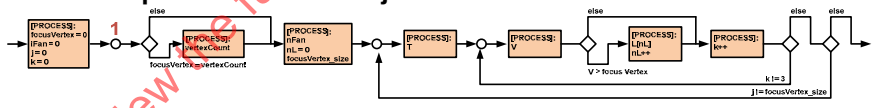
Decode Connectivity TFAN FSM:



Get Number of FANS and Max TFANS dim:



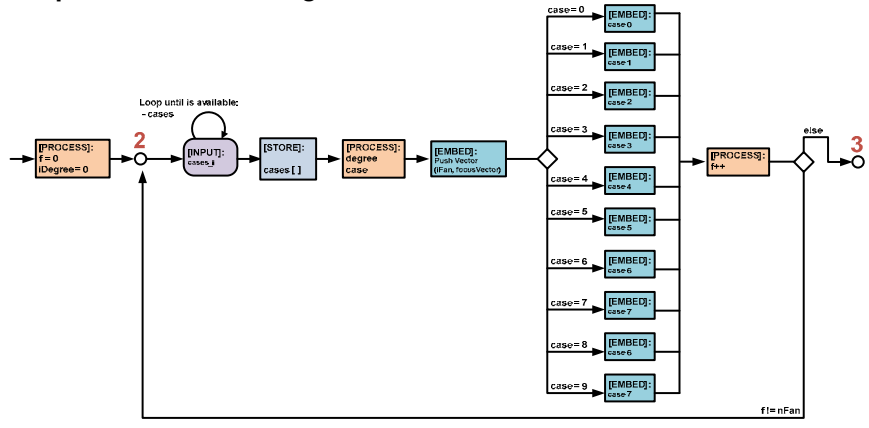
Get all conquested vertices adjacent to the focused vertex:



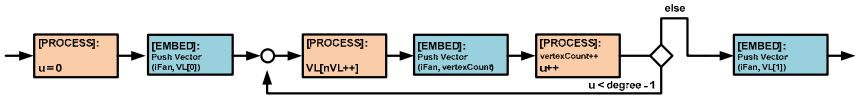
Sort the L vector: The implementation of any algorithm that is able to sort an array

Eliminate the duplicates in the L vector: The implementation of any algorithm that is able to eliminate the duplicates from a sorted array

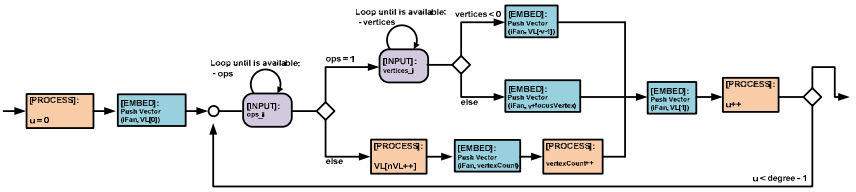
Compute tfans according to the defined cases:



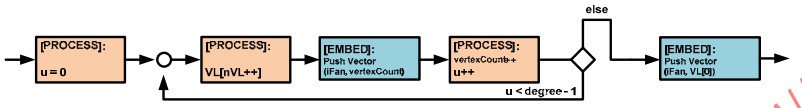
Cases:
CASE = 0



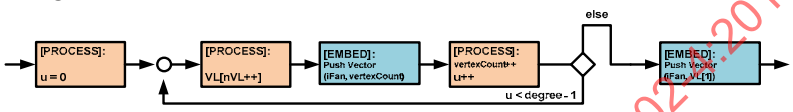
CASE = 1



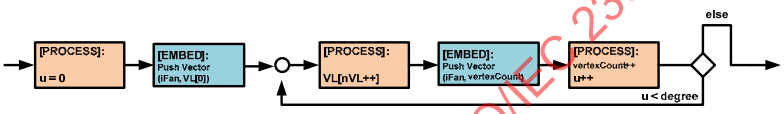
CASE = 2



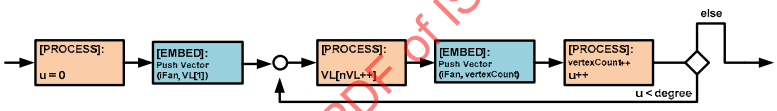
CASE = 3



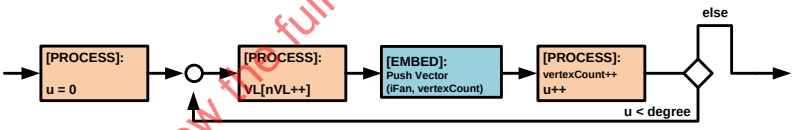
CASE = 4



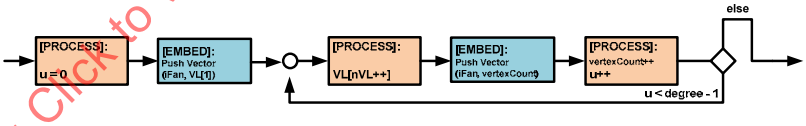
CASE = 5



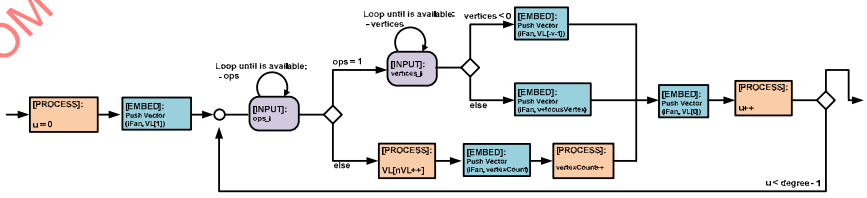
CASE = 6



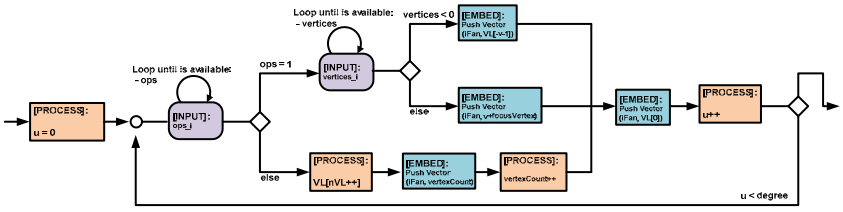
CASE = 7



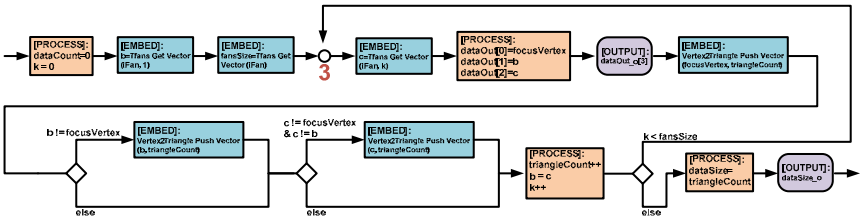
CASE = 8



CASE = 9



Compute dataOut:



Decode Connectivity TFAN Process:

EMBED: **Get Number of FANS and Max TFANS dim**
j = 0
INPUT: nFans
WHILE j < nFans
    INPUT: fans[]
    Number_FANS += fans[j]
    k = 0
    WHILE k < Number_FANS
        INPUT: degree[]
        IF Max_TFANS_dim < degree + 1
            Max_TFANS_dim = degree + 1
        k = k +1
    j = j + 1

EMBED: **Get all conquested vertices adjacent to the focused vertex**
focusVertex = 0
iFan = 0
j = 0
k = 0
if focusVertex = vertexCount
    vertexCount++
nFan = fans [
nL = 0
EMBED: vector size = Vertex2Triangle Get Vector (focusVertex)
WHILE j != vector size
    t = EMBED: vector size = Vertex2Triangle Get Vector (focusVertex,j)
    WHILE k != 3
        v = data[t*3+k]
        IF v > focusVertex
            L[nL]=v
            nL++
        k++
    j++

EMBED: **Compute tfans according to the defined cases**
f = 0
iDegree = 0
iCase = 0
iOps=0
iVertices=0
WHILE f != nFan
    INPUT: cases[]
    degree = degree[iDegree]
    case = cases[iCase]
    EMBED: tFan Push Vector (iFan,focusVector)
    SWITCH case
        case = 0: EMBED: CASE 0
        case = 1: EMBED: CASE 1
        case = 2: EMBED: CASE 2
        case = 3: EMBED: CASE 3
        case = 4: EMBED: CASE 4
        case = 5: EMBED: CASE 5
        case = 6: EMBED: CASE 6
        case = 7: EMBED: CASE 7
        case = 8: EMBED: CASE 8
        case = 9: EMBED: CASE 9
    f++

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

```
        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++
```

	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
ISO Standards using the FU	ISO/IEC 14496-16:2011

Profiles@levels supported	

9.3 MPEG-4 SC3DMC SVA Specific FUs

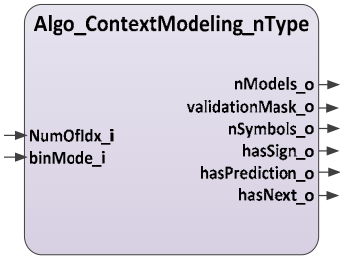
9.3.1 Algo_ContextModeling_SVA_nType

FU Name

Algo_ContextModeling_SVA_nType

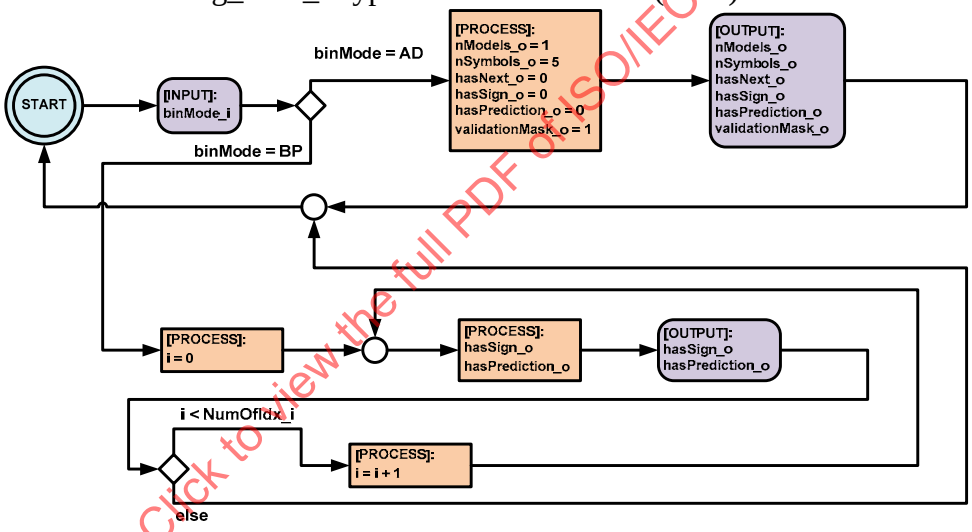
Description

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



Port Name	Direction (I/O)	Token RANGE
NumOfIdx_i	I	UINT8, UINT16, UINT32, UINT64
binMode_i	I	UINT_8
nModels_o	O	UINT8, UINT16, UINT32, UINT64
validationMask_o	O	BOOLEAN
nSymbols_o	O	UINT32, UINT64
hasSign_o	O	BOOLEAN
hasPrediction_o	O	BOOLEAN
hasNext_o	O	BOOLEAN

ContextModeling_SVA_nType Process Schematic (FSM):



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	

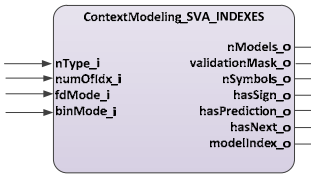
Parameter		
Name	**Description**	**Range**

9.3.2 Algo_ContextModeling_SVA_Indexes

FU Name	Algo_ContextModeling_SVA_Indexes

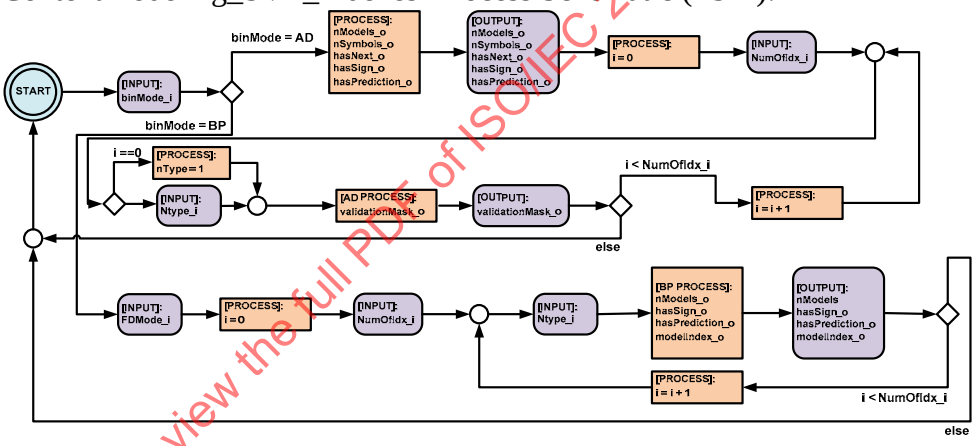
Description

This FU describes how to generate context models of 'connectivity' depending on the nType and the binarization mode.



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:

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, 2^{10}, 2^{20}, 2^{30}}
  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}

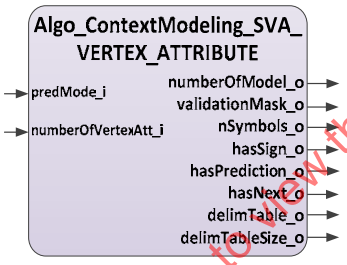
```
        ELSE IF nType = 1
            validationMask = {0,0,1,0,0,0,1,0,0,0,1,0}
        ELSE IF nType = 2
            validationMask = {1,0,1,0,0,0,1,1,0,0,0,0}
        ELSE IF nType = 3
            validationMask = {0,0,1,0,0,0,1,0,0,0,1,0}
          ELSE IF nType = 4
              validationMask = {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}
```

	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
ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

Parameter		
Name	**Description**	**Range**

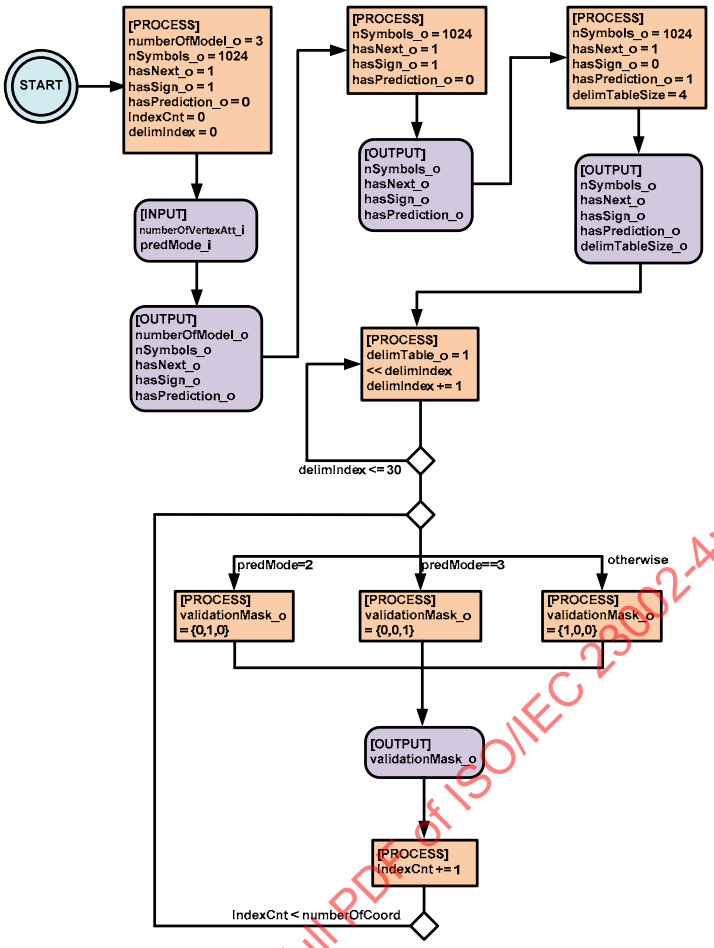
9.3.3 Algo_ContextModeling_SVA_Vertex_Attribute

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.



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 < numberOfVertxAtt; IndexCnt++) {
       if (predMode == 2)
           validationMask_o = {0, 1, 0}
      else if (predMode == 3)
           validationMask_o = {0, 0, 1}

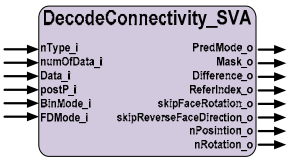
	else validationMask_o = {1, 0, 0}	
ISO Standards using the FU	ISO/IEC 14496-16:2011	
Profiles@levels supported		
Parameter		
Name	**Description**	**Range**

9.3.4 Algo_DecodeConnectivity_SVA

FU Name	Algo_DecodeConnectivity_SVA

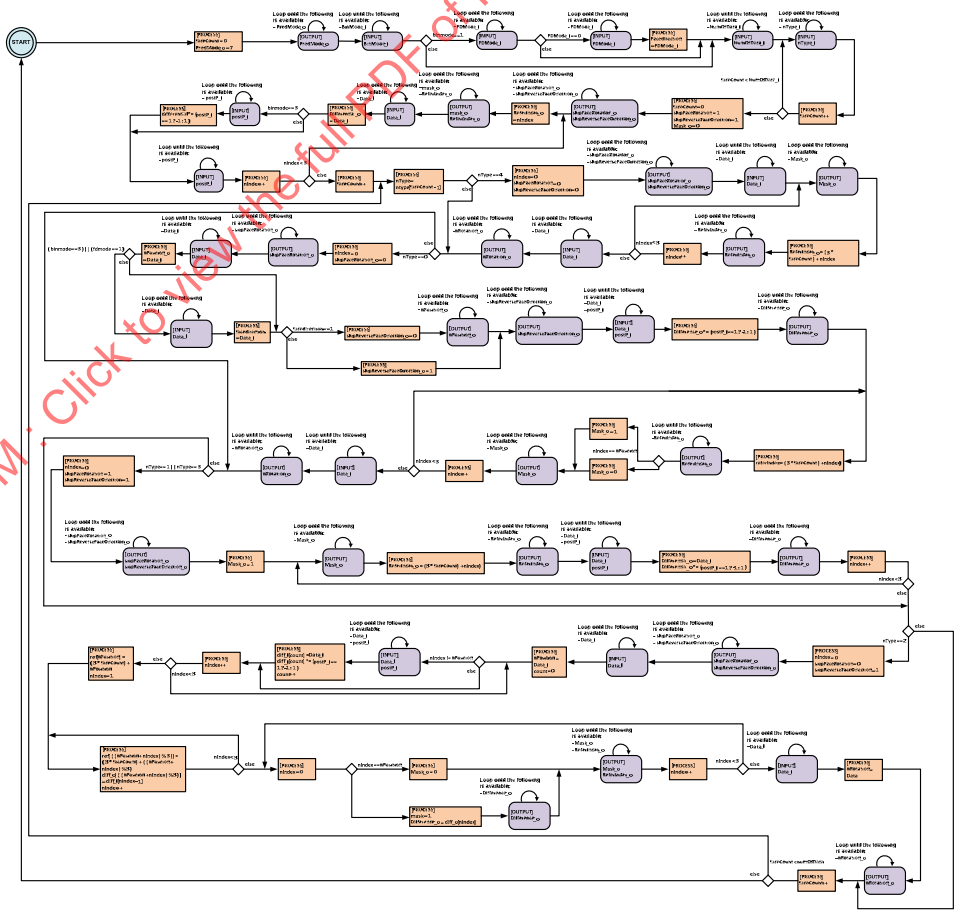
Description

This FU describes the connectivity decoding process for SVA.



Port Name	Direction (I/O)	Token RANGE
nType_i	I	UINT8, UINT16, UINT32, UINT64
NumOfData_i	I	UINT8, UINT16, UINT32, UINT64
Data_i	I	UINT8, UINT16, UINT32, UINT64
postP_i	I	BOOLEAN
BinMode_i	I	UINT_8
FDMode_i	I	UINT8, UINT16, UINT32, UINT64
Mask_o	O	UINT8, UINT16, UINT32, UINT64
Difference_o	O	UINT8, UINT16, UINT32, UINT64
ReferIndex_o	O	UINT8, UINT16, UINT32, UINT64
PredMode_o	O	UINT_8
skipFaceRotation_o	O	BOOLEAN
skipReverseFaceDirection_o	O	BOOLEAN
nPosition_o	O	UINT8, UINT16, UINT32, UINT64
nRotation_o	O	UINT8, UINT16, UINT32, UINT64

DecodeConnectivity_SVA Process Schematic (FSM):



ConnectivityDecoding SVA Process:

```
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
```

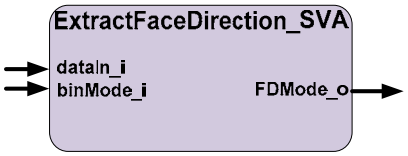
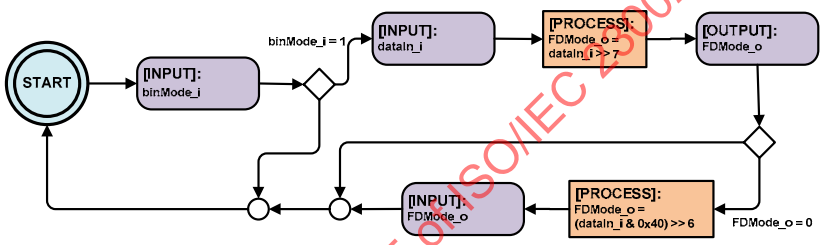
	nIndex++
	ENDWHILE
	INPUT:Data_i
	OUTPUT:nRotation_o
	ENDIF
	
	ELSEIF (nType == 0)
	nIndex = 0;
	skipFaceRotation = 0;
	OUTPUT:skipFaceRotation_o
	INPUT:Data_i
	nPosition_o = Data_i
	
	IF ((binmode == 3) || (fdmode == 1))
	INPUT:Data_i
	facedirection = Data_i
	ENDIF
	
	IF (facedirection == 1) {
	skipReverseFaceDirection = 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++

	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

ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	
Parameter	

Name	Description	Range

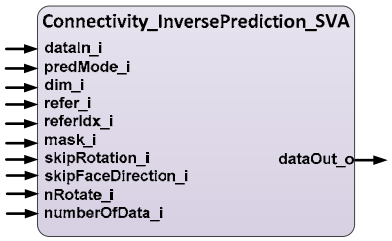
9.3.5 Algo_ExtractFaceDirection_SVA

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

9.3.6 Algo_Connectivity_InversePrediction_SVA

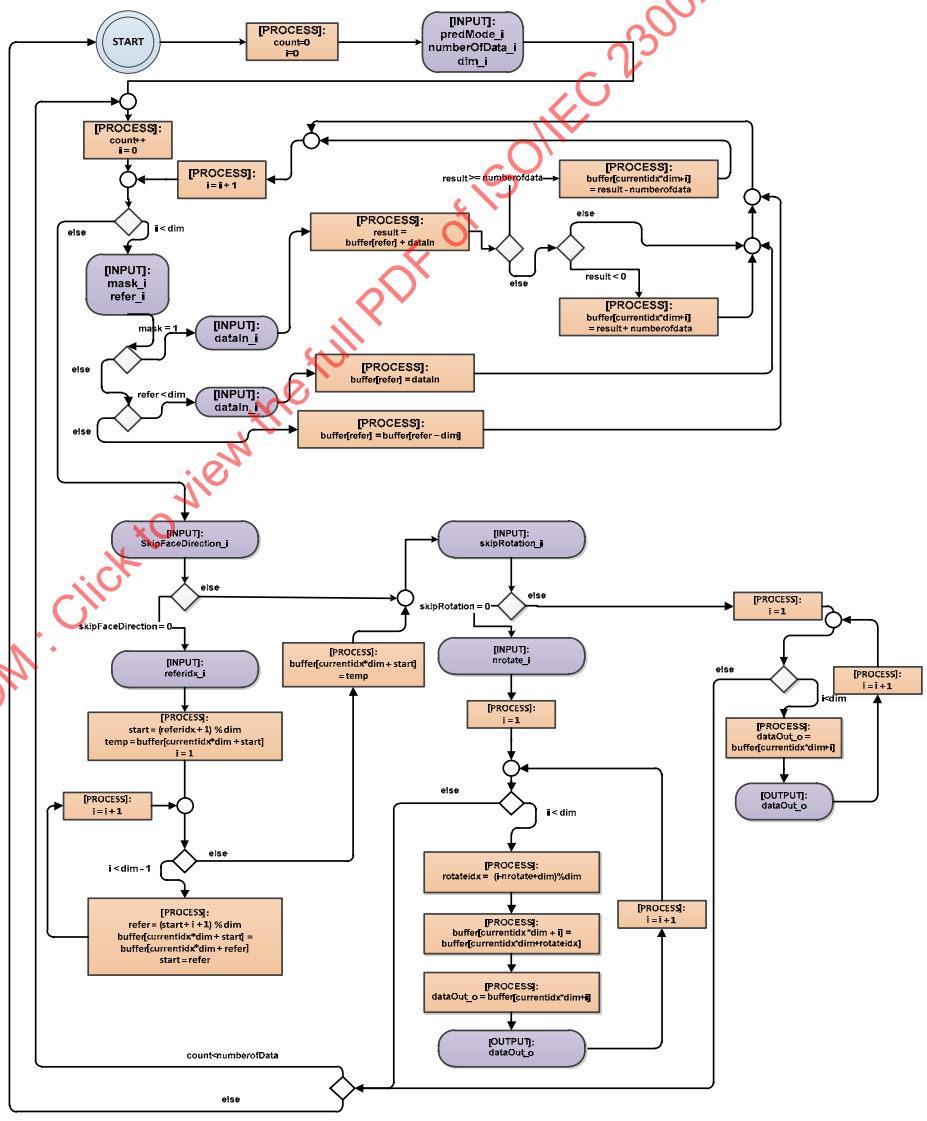
FU Name	Algo_Connectivity_InversePrediction_SVA

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):



Connectivity_Inverse Prediction_ SVA Process:
START:

	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

ISO Standards using the FU	ISO/IEC 14496-16:2011
Profiles@levels supported	

Parameter		
Name	**Description**	**Range**

After Annex C, add the following annexes:

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.

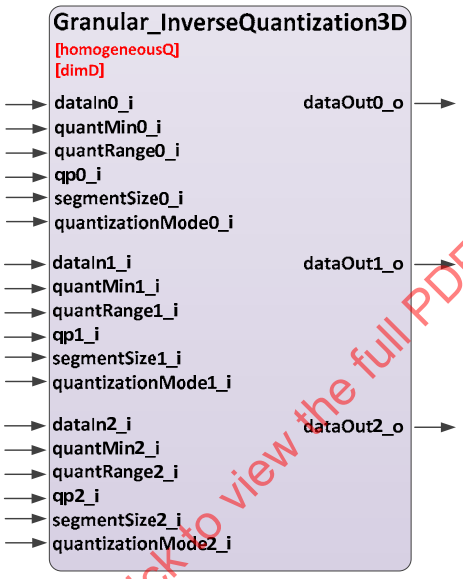
The Granular FU extends the usage of some FU for the broad use and reusability of FU. There are bound traffics between FUs, or sometimes some FUs need communal buffers. To reduce the data traffic or eliminate the dependence between FUs, those necessary existing FUs can be designed as one granular FU. 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.

Annex E
(informative)

Granular FUs

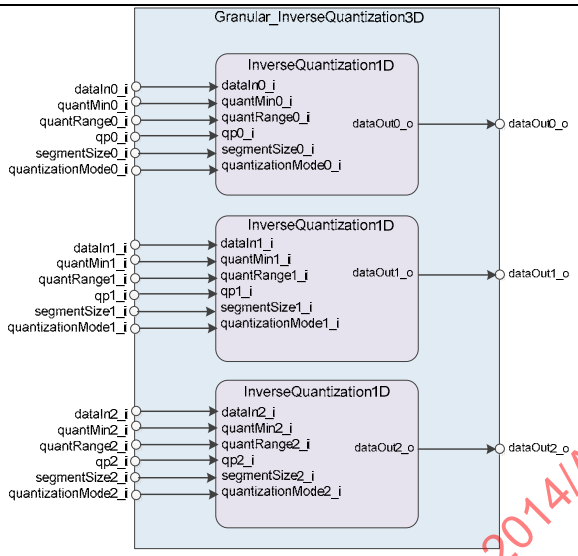
E.1 Granular_InverseQuantizaiton3D

FU Name	Granular_InverseQuantization3D
Description	Three ALGO_InverseQuantization1D FUs are used to make a 3-dimensional inverse quantization process.



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 schematic



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"
```

	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>
ISO Standards using the FU	
Profiles@levels supported	

Parameter

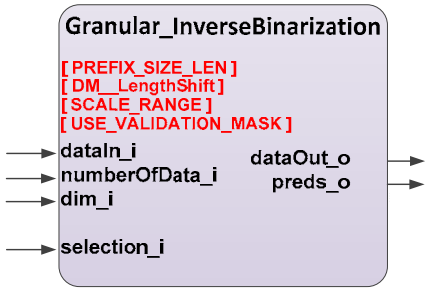
Name	Description	Type / Range
dimD	Describes the number of tokens of type dataIn_i that are consumed at each firing. This parameter is set at the network configuration level.	Type: Integer Range: [1 .. 2^5]
homogeneousQ	Describes the number of tokens of type quantRange_i, quantMin_i and quantValue_i 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

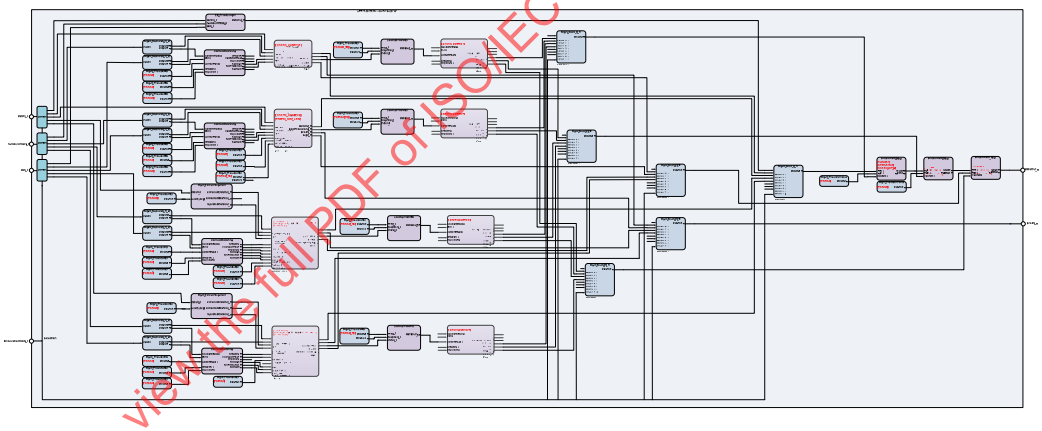
Description

This FU describes the inverse binarization process used in SC3DMC.



Port Name	Direction (I/O)	Token RANGE
dataIn_i	I	UINT8, UINT16, UINT32, UINT64
numerOfData_i	I	UINT8, UINT16, UINT32, UINT64
dim_i	I	UINT8
dataOut_o	O	UINT8, UINT16, UINT32, UINT64
preds_o	O	UINT8, UINT16, UINT32, UINT64

Granular ED_InverseBinarization network schematic



Granular ED_InverseBinarization SC3DMC network FNL code

```
<?xml version="1.0" encoding="UTF-8"?>
<XDF 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"/>
```